



32-Bit Microprogrammable Products
Am29C300/29300

1988 Data Book

Advanced
Micro
Devices





Advanced Micro Devices

**Am29C300/29300
Data Book**

© 1988 Advanced Micro Devices

Advanced Micro Devices reserves the right to make changes in its products without notice in order to improve design or performance characteristics.

This manual neither states nor implies any warranty of any kind, including but not limited to implied warranties of merchantability or fitness for a particular application. AMD assumes no responsibility for the use of any circuitry other than the circuitry embodied in an AMD product.

The information in this publication is believed to be accurate in all respects at the time of publication, but is subject to change without notice. AMD assumes no responsibility for any errors or omissions, and disclaims responsibility for any consequences resulting from the use of the information included herein. Additionally, AMD assumes no responsibility for the functioning of undescribed features or parameters.

901 Thompson Place, P.O. Box 3453, Sunnyvale, California 94088-3000
(408) 732-2400 TWX: 910-339-9280 TELEX: 34-6306

AMDASM, AmSYS, and IMOX are trademarks of Advanced Micro Devices, Incorporated.

UNIX is a trademark of Bell Laboratories.

VAX is a registered trademark of Digital Equipment Corporation.

Multibus is a registered trademark of Intel, Corporation.

IBM is a registered trademark of International Business Machines, Incorporated.

IBM-AT and IBM-PC are trademarks of International Business Machines, Incorporated.

SmartModel is a registered trademark of Logic Automation, Incorporated.

Symbolic Hardware Debugging is a trademark of Logic Automation, Incorporated.

QuickSim is a trademark of Mentor Graphics.

PAL is a registered trademark of Monolithic Memories, Incorporated.

AUTOSTEP, Meta-Disassembler, MetaStep, QuickLearn, STEP-40 SDT, and User-Defined Symbolics are trademarks of Step Engineering.

Thank you for your interest in the Am29C300/29300 family of microprogrammable products. This manual reflects our commitment to you to bring together all the essential ingredients for making your 32-bit system design as smooth and straightforward as possible.

This manual contains detailed product specifications, applications information, software support products and third-party vendors, instruction definitions, and the latest reliability information for both CMOS and bipolar technologies.

We have the quality, reliability and innovative products you need. Our worldwide hardware and software support teams of field applications engineers are ready to help you utilize our advanced microprogrammable products to complete your designs in a timely and cost-effective manner.

It is with sincere appreciation that we welcome you to the growing family of satisfied AMD customers. We look forward to serving your semiconductor needs and thank you for the opportunity to contribute to your success.

A handwritten signature in black ink, reading "George Rigg". The signature is written in a cursive, flowing style with a large initial "G".

George Rigg
Vice President
Processor Products Division
Advanced Micro Devices

Preface

Advanced Micro Devices is recognized as the pioneer and leader in microprogrammable "bit slice" integrated circuits. The Am29300 family sets the current standard in general purpose 32-bit building blocks. Designed for high performance and flexibility with a choice of elegant, easy to implement architectures, this chip set brings microprogrammable products into the next generation.

The Am29300 generation gives the system designer flexibility both in hardware architecture and at the microprogram level. This 32-bit product family achieves high performance and high integration, while avoiding architectural restrictions. The products are designed to meet the high computational requirements of advanced graphics systems, image processing, high-end controllers, fault-tolerant processors, work stations, and other 32-bit applications limited not by process technology, but only by the designer's imagination.

Chapters 2, 3, and 4 of this databook describe the current full range of the Am29300 product offerings in bipolar and CMOS technologies. Three different types of data sheets are presented: Advanced Information, Preliminary, and Final.

- Advanced Information data sheets are developed from simulation data after circuit design is completed. After a process change, advanced information is again provided for speed select data.
- Preliminary data sheets are based on actual measurements when silicon is available and units have been tested for AC characteristics. The preliminary test programs are in place, but the normal fabrication process variations have not allowed setting of final AC limits.
- Final data-sheet status is applied to products that are fully characterized over the operating range and are in volume production.

Over 75 application notes and technical articles have been written in 11 different languages describing the features and benefits of the Am29300/29C300 family. A few representative articles are reprinted in Chapter 6 to serve as a starting point for readers less familiar with the broad scope of this chip set. A full list of articles is offered in the bibliography of Chapter 6.

Technical information regarding product and process reliability, as well as the Advanced Micro Devices model for reliability studies is provided in Chapter 7. This chapter also outlines the basic thermal characteristic data for the bipolar Am29300 products and describes test philosophy and methods.

Chapter 8 gives general information regarding package outlines and ordering information.

Table of Contents

CHAPTER 1	Am29300/29C300 Family Overview	
	1.1 Am29300/29C300 GENERAL OVERVIEW	1-1
	1.2 Am29300/29C300 FAMILY DEVICE OVERVIEW	1-3
	1.3 A.C. AND D.C. PARAMETER DEFINITIONS	1-17
CHAPTER 2	CMOS Family	
	Am29C331 CMOS 16-Bit Microprogram Sequencer	2-1
	Am29C332 CMOS 32-Bit Arithmetic Logic Unit	2-38
	Am29C334 CMOS Four-Port Dual-Access Register File	2-76
	Am29C325 CMOS 32-Bit Floating-Point Processor*	2-94
	Am29C327 CMOS Double-Precision Floating-Point Processor*	2-95
CHAPTER 3	Bipolar Family	
	Am29331 16-Bit Microprogram Sequencer	3-1
	Am29332 32-Bit Arithmetic Logic Unit	3-36
	Am29334 Four-Port Dual-Access Register File	3-74
	Am29434 ECL Four-Port, Dual-Access Register File	3-89
	Am29325 32-Bit Floating-Point Processor*	3-103
	Am29337 16-Bit Bounds Checker	3-104
	Am29338 32-Bit Byte Queue	3-115
CHAPTER 4	Arithmetic Processors	
	Am29C323 CMOS 32-Bit Parallel Multiplier	4-1
	Am29325 32-Bit Floating-Point Processor	4-24
	Am29C325 CMOS 32-Bit Floating-Point Processor	4-78
	Am29C327 CMOS Double-Precision Floating-Point Processor	4-133
CHAPTER 5	Support Tools	
	5.1 Am29C300 EVALUATION BOARD	5-1
	5.2 Am29300 TEST BOARD	5-4
	5.3 Am29300 DEFINITION FILE	5-7
	5.4 MICROCODE DEVELOPMENT	5-23
	5.4.1 Step Engineering 32-Bit Development Tools	5-23
	5.4.2 Microtech mcASM Structured Microcode Assembler	5-31
	5.4.3 Hilevel Technology	5-37
	5.4.4 Hewlett-Packard Microprogram Development Support	5-43
	5.5 SIMULATION MODELS	5-48
	5.6 C COMPILER SUPPORT	5-54
	5.7 WRITABLE CONTROL STORE	5-59
	5.7.1 Agility AG-11B Microprogram Development	5-59

* Front page only of data sheet. See Chapter 4 for complete data sheet.

TABLE OF CONTENTS
Continued

CHAPTER 6	Articles/Application Notes	
	6.1 BIPOLAR BUILDING BLOCKS DELIVER SUPERMINI SPEED TO MICROCODED SYSTEMS	6-1
	6.2 Am29300 DEMONSTRATION SYSTEM APPLICATION NOTE	6-12
	6.3 THE FAST WAY TO BUILD A RISC PROCESSOR	6-92
	6.4 FAULT-TOLERANT CHIPS INCREASE SYSTEM RELIABILITY	6-97
	6.5 FLOATING-POINT MATH HANDLES ITERATIVE AND RECURSIVE ALGORITHMS	6-102
	6.6 FLOATING-POINT ARRAY PROCESSOR IMPROVES COMPUTATIONAL POWER	6-109
	6.7 FLOATING-POINT μ P IMPLEMENTS HIGH-SPEED MATH FUNCTIONS	6-116
	6.8 OPTIMIZE YOUR GRAPHICS SYSTEM FOR BOTH 2D AND 3D	6-123
	6.9 VARIABLE-WIDTH FIFO BUFFER SEQUENCES LARGE DATA WORDS	6-136
	6.10 DIGITAL SYSTEMS VME 29300-1	6-141
	6.11 BIBLIOGRAPHY	6-144
CHAPTER 7	Technical Information	
	7.1 THE Am29300/29C300 TIMING ANALYSIS	7-1
	7.2 THERMAL CHARACTERISTICS/AIR FLOW	7-6
	7.3 CMOS/BIPOLAR RELIABILITY	7-10
	7.4 CMOS LATCH-UP TEST METHODS AND RESULTS	7-17
	7.5 TEST PHILOSOPHY AND METHODS	7-18
CHAPTER 8	General Information	
	8.1 PHYSICAL DIMENSIONS	8-1
	8.2 ORDERING INFORMATION	8-8

NUMERICAL DEVICE LISTING

Am29C323	CMOS 32-Bit Parallel Multiplier	4-1
Am29C323-1	CMOS 32-Bit Parallel Multiplier – Speed Select	4-1
Am29C323-2	CMOS 32-Bit Parallel Multiplier – Speed Select	4-1
Am29325	32-Bit Floating-Point Processor	4-24
Am29325A	32-Bit Floating-Point Processor – Speed Enhancement	4-24
Am29C325	CMOS 32-Bit Floating-Point Processor	4-78
Am29C325-1	CMOS 32-Bit Floating-Point Processor – Speed Select	4-78
Am29C325-2	CMOS 32-Bit Floating-Point Processor – Speed Select	4-78
Am29C327	CMOS Double-Precision Floating-Point Processor	4-133
Am29331	16-Bit Microprogram Sequencer	3-1
Am29331A	16-Bit Microprogram Sequencer – Speed Enhancement	3-1
Am29C331	CMOS 16-Bit Microprogram Sequencer	2-1
Am29C331-1	CMOS 16-Bit Microprogram Sequencer – Speed Select	2-1
Am29C331-2	CMOS 16-Bit Microprogram Sequencer – Speed Select	2-1
Am29332	32-Bit Arithmetic Logic Unit	3-36
Am29332A	32-Bit Arithmetic Logic Unit – Speed Enhancement	3-36
Am29C332	CMOS 32-Bit Arithmetic Logic Unit	2-38
Am29C332-1	CMOS 32-Bit Arithmetic Logic Unit – Speed Select	2-38
Am29C332-2	CMOS 32-Bit Arithmetic Logic Unit – Speed Select	2-38
Am29334	Four-Port Dual-Access Register File	3-74
Am29C334	CMOS Four-Port Dual-Access Register File	2-76
Am29C334-1	CMOS Four-Port Dual-Access Register File – Speed Select	2-76
Am29C334-2	CMOS Four-Port Dual-Access Register File – Speed Select	2-76
Am29337	16-Bit Bounds Checker	3-104
Am29338	32-Bit Byte Queue	3-115
Am29434	ECL Four-Port, Dual-Access Register File	3-89

CHAPTER 1

Am29300/29C300 Family Overview

1.1 Am29300/29C300 GENERAL OVERVIEW	1-1
1.2 Am29300/29C300 FAMILY DEVICE OVERVIEW	1-3
1.3 A.C. AND D.C PARAMETER DEFINITIONS	1-17



Am29300/29C300 Family Overview

1.1 Am29300/29C300 GENERAL OVERVIEW

CMOS and Bipolar 32-Bit High Performance Building Blocks

AMD's Am29300/29C300 family has been developed to provide systems designers with flexible, off-the-shelf, high-performance, 32-bit microprogrammable building blocks. The Am29300/29C300 family is ideal for complex and calculation-intensive applications such as intelligent peripheral controllers including graphics, telecommunications, switching systems and laser printers; artificial intelligence and RISC CPUs; array and digital signal processing; and a multitude of military applications.

Am29300/29C300 Pushes the Limits of Your Imagination

Flexibility of Design

Success is driven by innovation and differentiation. While "me too" systems companies merely struggle to be the lowest cost manufacturers, innovative companies strive ahead toward the future. The designers of AMD's 32-bit family recognize the need for system innovation and differentiation. The Am29300/29C300 family provides powerful building blocks with unlimited architectural flexibility, thus returning design innovation and value-added back to the design engineer. With the flexibility of custom architectures and custom microcode, system performance is limited only by imagination.

Improve Your Time to Market

Because AMD's 32-bit family integrates high performance features such as master/slave, parity checking,

funnel shifters, priority encoders, and mask generators, the Am29300/29C300 family meets the complex functional requirements of sophisticated systems and can eliminate the need for custom ICs. With the Am29300/29C300 there are no engineering circuit turnaround delays, no hidden Non-Recurring-Engineering costs, no complex test engineering correlations, and no waiting. Off-the-shelf availability of a highly integrated, fully tested product of guaranteed quality can mean improved profits for the system application.

Specifications that Count

We provide you with the tools and data necessary to make your design right the first time. You can be assured that the specifications of the parts you order are guaranteed by AMD as printed in the data sheets. Designers require worst case guaranteed parameter values, and AMD provides them. AMD removes the uncertainty of customized design with fully guaranteed, standard, off-the-shelf, 32-bit products. These state-of-the-art bipolar and CMOS building blocks are the ideal solution for 32-bit applications.

Military Product Position

AMD is committed to support the industry with military qualified and specified Am29C300 family products. The entire family is being offered as 883C level B fully compliant APL products. In addition, we plan to release the family in DESC military drawings. This will provide the user with alternatives to source control drawings, thus saving cost and time.

Manufacturing – Processes and Planning

AMD's Commitment to Process Technology Improvements

The Am2901 industry standard bit-slice ALU is an ideal example of AMD's commitment to process improvements. Table 1-1 and Figure 1-1 demonstrate the per-

formance improvements of the Am2901. Since its introduction, the Am2901's performance has increased nearly three-fold while its price has dropped by a factor of ten. This represents **25 percent** annual price/performance improvement over **12 years**. The philosophy of performance improvements through process technologies applies to all members of AMD's microprogrammable products.

Table 1-1

Year	Device	Technology	Die Size	Speed	Power
				A,B → G,P	
1975	Am2901	Low-Power Schottky	33 K mil ²	80 ns	1.5 W
1977	Am2901A	Dual Layer Metal Ion Implantation	20 K mil ²	65 ns	1.5 W
1978	Am2901B	Projection Printing	15 K mil ²	50 ns	1.5 W
1981	Am2901C	ECL Internal TTL, I/O IMOX	15 K mil ²	37 ns	1.5 W
1986	Am29C01	1.6 μm CMOS	15 K mil ²	37 ns	0.5 W
1987	Am29C01-1	1.2 μm CMOS Speed Select	15 K mil ²	28 ns	0.5 W
1987	Am29C01-2	1.0 μm CMOS	15 K mil ²	19 ns <small>(est)</small>	0.5 W

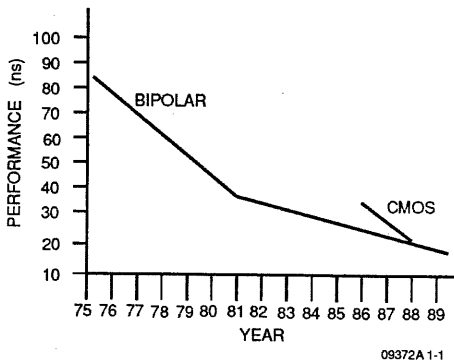


Figure 1-1. Am2901 Performance

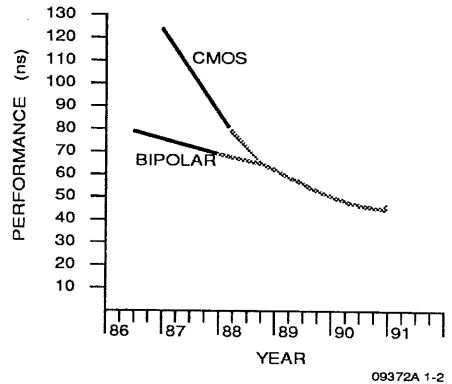


Figure 1-2. Am29300/29C300 Performance

Bipolar VLSI

The Am29300 family contains some of the largest bipolar ICs produced anywhere in the world. For example, the Am29332 has over 5,000 gates, 31,000 devices, and measures 142,000 mils². AMD's IMOX S-2 process allows for such integration and high performance. Future advances in AMD's bipolar process will include process "tweaks" as well as total changes in process approach. These advances will provide improved performance and yields, directly affecting the price/performance of the Am29300 family.

CMOS VLSI

The Am29C300 family, like its bipolar counterpart, also contains very large die. The Am29C325 encompasses nearly 11,000 gates and measures almost 130,000 mils².

AMD's CS-11 is the current CMOS workhorse process for the Am29C300 family. At an effective channel width of 1.6 microns, CS-11 is capable of approaching the bipolar speeds on all specifications.

There will be continued process improvements to the current CMOS technology. The first improvement, CS-11A, will be available on all Am29C300 products in Q4 1987. CS-11A has an effective channel width of 1.2 microns, resulting in a 25 percent performance improvement over CS-11.

Table 1-2 demonstrates the performance improvements expected on the Am29C300 family as these processes are incorporated into the family.

Table 1-2 CMOS Evolution

Year	Process	Effective Channel Length	Typical Gate Delay
1986	CS-11	1.6 micron	1.25 ns
1987	CS-11A	1.2 micron	0.90 ns
1988	CS-21	1.0 micron	0.65 ns

The Philosophy Behind the Functionality

When AMD introduced the 4-bit slice (memory plus ALU) Am2901 in 1975, semiconductor and packaging technologies prevented the integration of a 16- or 32-bit unit. The 4-bit slice with internal memory and external carry-

look-ahead and a 48-pin package were the right compromise then. Today, semiconductor and packaging technologies have advanced to a point where a full 32-bit ALU with many non-sliceable features, internal carry-look-ahead, and systems access to all buses can be put on one chip, with expandable memory on another. This results in higher versatility and higher performance.

There are several reasons for the choice of a wider data path. First, cycle time is improved significantly if carry lookahead is contained entirely on the chip. Second, certain powerful on-chip functions, such as the funnel shifter, priority encoder, and mask generator are extremely difficult to "slice." Third, a higher level of integration leads to a more cost-effective system solution. These and other advantages contributed to the decision to make the Am29332/29C332 a complete 32-bit function rather than a slice.

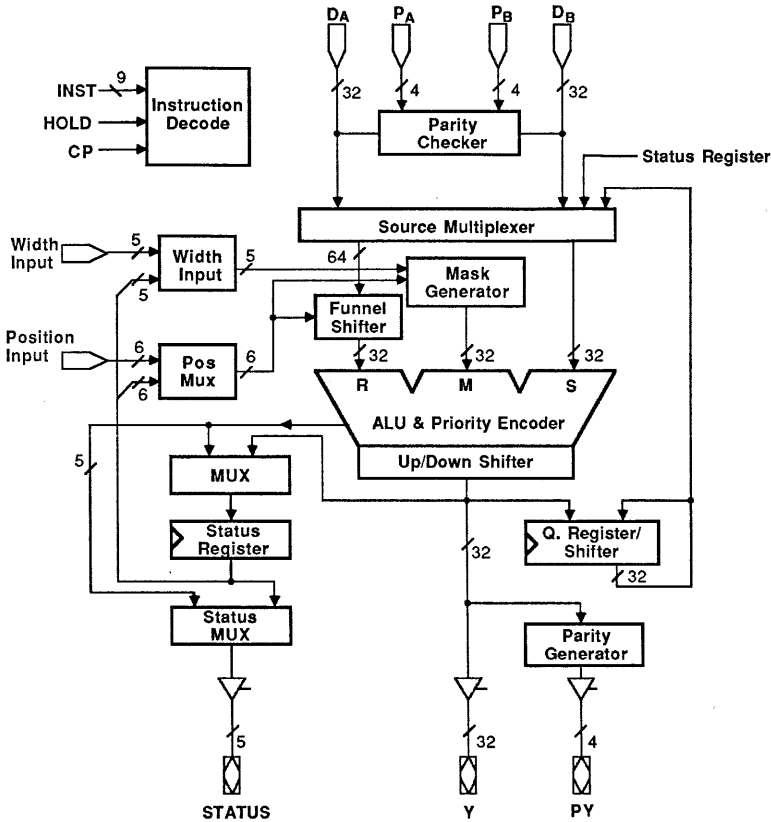
The Am29300/29C300 philosophy has also removed the register file from the ALU, providing the designer greater system flexibility and making expansion and regular addressing much easier. The new partitioning results in a number of benefits. The user gets a functionally more powerful processor with two uncommitted input buses and gains the flexibility of adding storage elements to those buses. The Am29300/29C300 family is designed to be the most functional and powerful family of microprogrammable building block products available on the market.

1.2 Am29300/29C300 FAMILY DEVICE OVERVIEW

The Am29332/29C332 32-Bit ALU – The Heart of a New Generation of Machines

The Am29332/29C332 is AMD's first 32 bit wide ALU. Parallel processing of 32 bits of data, coupled with very fast cycle time, provides throughput unprecedented in VLSI-based systems.

The 32-bit ALU combines maximum performance and integration by keeping all critical timing paths short and balanced. All ALU instructions have the same short cycle time. This includes barrel shifting, normalization, priority encoding and field logical operations.



09372A 1-3

Figure 1-3. Am29332/29C332 32-Bit ALU

Three Ports Facilitate High Throughput

The Am29332/29C332 has two input ports (A and B) and an output port (Y), all 32 bits wide. These three ports provide flexibility and accessibility for high-performance processor designs. Dedicated input and output ports provide a flow-through architecture and avoid the penalty associated with switching a bidirectional bus halfway through the cycle. In addition, the three-bus architecture allows easy parallel connection of other arithmetic units for even higher performance.

Arithmetic and Logic Unit

The 32-bit wide ALU in the Am29332/29C332 has full carry-lookahead to improve cycle time for all arithmetic operations. The ALU is a unique three-input structure with two data input ports and a mask input that is used on every cycle, thus providing very powerful instructions

that execute in a single cycle. The mask supports byte-aligned arithmetic operations and field logical operations on variable-position, variable-length fields. The byte-aligned arithmetic operations use 8-, 16-, 24-, and 32-bit LSB-aligned operands. Field-logical instructions operate on operands of arbitrary length and starting position.

Priority Encoder

The priority encoder generates a 5-bit vector indicating the highest order 'one' in the 32-bit operand. These 5 bits are then stored in the position field of the status register for use during the next cycle. The priority encoder supports all byte-aligned data types; the result is dependent upon the byte width specified. This function supports normalization necessary for floating point operations; it also enhances certain graphics primitives.

64-Bit Funnel Shifter

The on-board 64-bit input, 32-bit output funnel shifter is much more than a conventional barrel shifter. The shifter can extract any contiguous field of 32 bits from a 64-bit input. This input may consist of concatenated A and B input words or, for barrel shifting, duplicated A or B input words.

Residing in the ALU data path, the shifter can perform n-bit shift or rotate in conjunction with a logical ALU operation—all in the same cycle, without increasing the length of the cycle. This capability affords single-cycle execution of logical operations between unaligned fields — a function that would take multiple cycles in other architectures.

Mask Generator

The power and flexibility of the processor stems partly from its ability to generate a mask to control the width of an operation for each instruction without any cycle time penalty. The mask generator at the ALU input creates a contiguous field of ones and contains its own shifter to position this control field anywhere in the data path. The mask generator can also be used as a pattern generator, bypassing the mask through the ALU.

Status Register

The processor has a 32-bit wide status register that contains: information on position and width of the operand; the ALU status flags Carry, Negative, Overflow, and Zero; status bits for evaluation of inequalities; a link bit for multiprecision shifts; an M flag for high speed multiply and divide; and intermediate nibble carries for BCD arithmetic. An extract-status instruction is provided that allows any bit from the status register to be output at the Y-port. This is particularly useful in machines employing stack architectures. Instructions to save and restore the status register are also provided.

Multiply and Divide Support

The chip incorporates dedicated hardware to allow efficient implementation of multiply and divide algorithms for

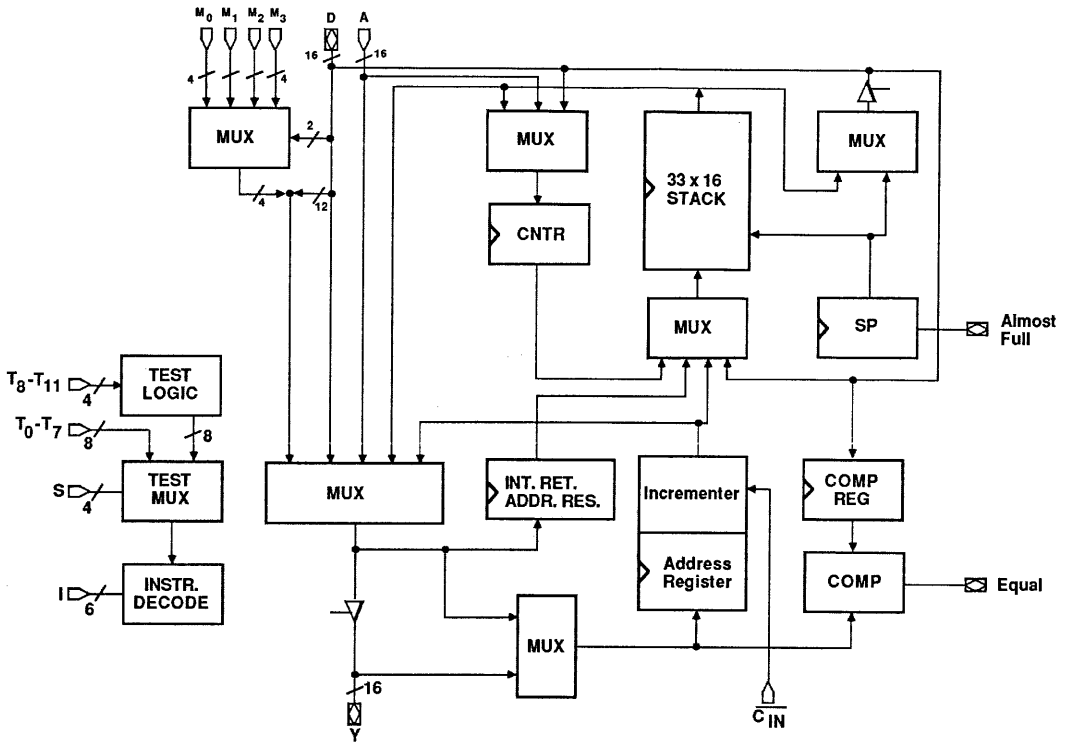
both unsigned and signed arithmetic data types. The modified Booth multiply algorithm processes **two bits per cycle**. The four-quadrant, non-restoring divide algorithm processes one bit per cycle. Since the data path width is fixed at 32 bits, the instructions can be simplified to provide “first step,” “iterate step” and “last step” commands for both multiply and divide. Programming slices is no longer necessary since all multiply and divide steps are provided in the instruction set. For business-oriented machines, the ALU is capable of performing BCD arithmetic on packed BCD numbers. In order to keep non-BCD operations fast, BCD arithmetic is executed by binary arithmetic followed by BCD correction.

The Instruction Set: Powerful and Flexible Yet Simple and Regular

The Am29332/29C332 instruction set complements the powerful hardware. To ease the task of code generation, the instruction set is symmetrical and regular. There are two large classes of instructions. The first class handles byte-aligned data (8-, 16-, 24-, or 32-bit LSB-aligned). It is comprised of: data movement instructions; arithmetic instructions, including multiply and divide steps and BCD instructions; logical instructions; and single-bit shift and prioritize operations. The second class of instructions operates on variable-length, variable-position fields. It includes N-bit shift and rotate, field extract, and field logical operations.

The Am29331/29C331 – 16-Bit Micro-Interruptible Sequencer

The Am29331/29C331 is a high speed sequencer controlling the sequence of microinstructions stored in microprogram memory. The instruction set aids structured microprogramming and handles sequential execution, branches, subroutines and loops. The sequencer instructions may be unconditional or conditional based on CPU status, an on-board 8-input test multiplexer, and a polarity control. The sequencer has a 16-bit wide address path and can thus access 64K words of microcode memory. It is transparently interruptible at any microinstruction boundary.



09372A 1-4

Figure 1-4. Am29331/29C331 16-Bit Microinterruptible Sequencer

Balanced Timing Means Greater Throughput

In previous generation microprogrammed systems, the control path containing the sequencer has often been the bottleneck, because the sequencers were slower than the associated data paths. Not so in the Am29300/29C300 family. The speed of the Am29331/29C331 sequencer has been designed such that the entire system timing is balanced between the control path and data path, leading to higher overall throughput.

Micro-Level Interruptible

Real time interrupt handling at the microinstruction level is made possible by the interrupt return address register and the bidirectional Y-port. While the interrupt address enters the part through the Y-port, the interrupt return address is saved on the stack. Nested interrupts are handled the same way.

Built-in Trap Handling

As an architectural alternative to the interrupt-driven approach, the Am29331/29C331 Sequencer also has provision for handling "traps" transparently at the microinstruction level, upon the occurrence of specified system events. In this mode, the current microinstruction is aborted. The specified trap routine is executed (like an interrupt). But, following the trap routine, the aborted microinstruction is re-executed (instead of proceeding on to the next microinstruction, as in an interrupt).

33-Level Stack

The 33-level stack provides sufficient depth to handle nested loops and subroutines; it is also used to save the status of the sequencer when handling interrupts. Since the stack is externally accessible, its contents may be

unloaded through the bidirectional D-port for diagnostic, debugging or fault recovery purposes. The stack may also be loaded from the outside through the D-port. This may be used for context switching, for example.

Multitasking Support

By providing a HOLD control pin, the designer may use multiple sequencers in a multitasking system, with only one sequencer active at any one time. The output Y-ports of the sequencers are tied together to address the same microcode memory. This is useful, for example, for rapid context switching at the microinstruction level.

Address Comparator Eases Debugging

The sequencer compares the address on the Y-port with the contents of an internal break-point register. Break-point detection is useful for debugging the system or gathering run-time statistics.

Two-Branch Address Inputs

Two separate branch address inputs, D and A, are provided to speed up source address selection. Both A and D ports can be used to load the counter. The D port can also be used to load or unload the stack while the A port may be used to input a branch or map address, eliminating the need to three-state selected sources.

Built-in Test Generation Logic

In the Am29331/29C331, unlike previous sequencers, test generation logic and one layer of condition test multiplexer logic are built-in. This not only reduces component count, but also improves cycle time by minimizing inter-chip delays and by moving the multiplexer into fast internal ECL gates.

Multiway

Four sets of four-bit multiway inputs are provided. Each such set of 4 bits can replace the four least significant bits of D input, allowing a direct branch to any of 16 consecutive locations in the microprogram memory. The multiway capability allows checking of up to four simultaneous test conditions in a single cycle. This is obviously an attractive alternative to checking each test condition serially, a much slower multicycle process.

The Most Versatile Sequencer Ever

The combination of 16 bits of address, real time interrupt capability, two address ports, a deep stack and other

capabilities make this device the most feature-loaded sequencer ever offered.

The Am29334/29C334 Register File

The Am29334/29C334 is a 64 word by 18 bit, dual-access, four-port register file. It is deliberately separate from the ALU to allow easy, regular expansion, both horizontally for wide data paths and vertically for large register file machines.

Four-Port Architecture

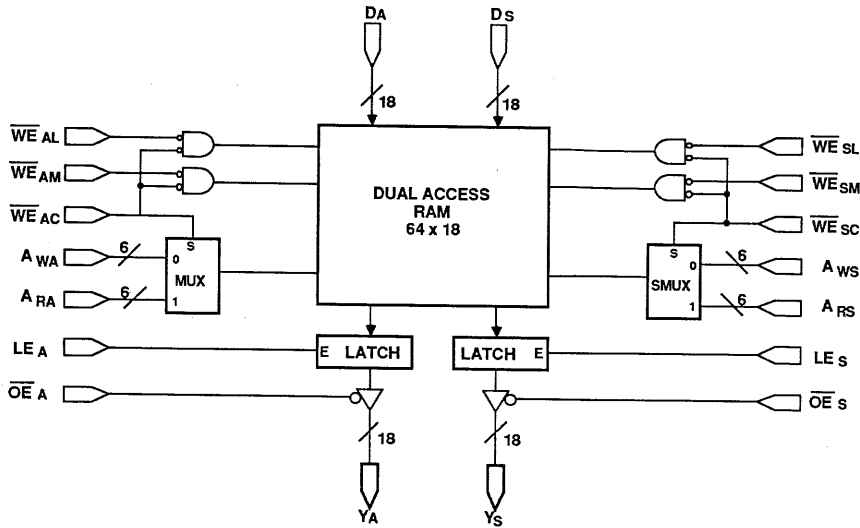
Two Read and two Write data ports allow independent and simultaneous access to two register file locations. The Read and Write ports are separated to eliminate the delay caused by turn-around of bidirectional buses. The dual-address, four-port architecture allows any combination of two reads, writes, or read-writes – no restrictions.

Organization Supports Parity

Since the Am29334/29C334 has a by-18 organization, it can store two bytes with parity in each of its 64 words. As a data path storage element, the register file neither generates nor checks parity. When used in conjunction with the Am29332/29C332 processor (which provides parity checking on its inputs and parity generation on its output), it provides a bus compatible register file, thus extending parity protection to the entire data path loop.

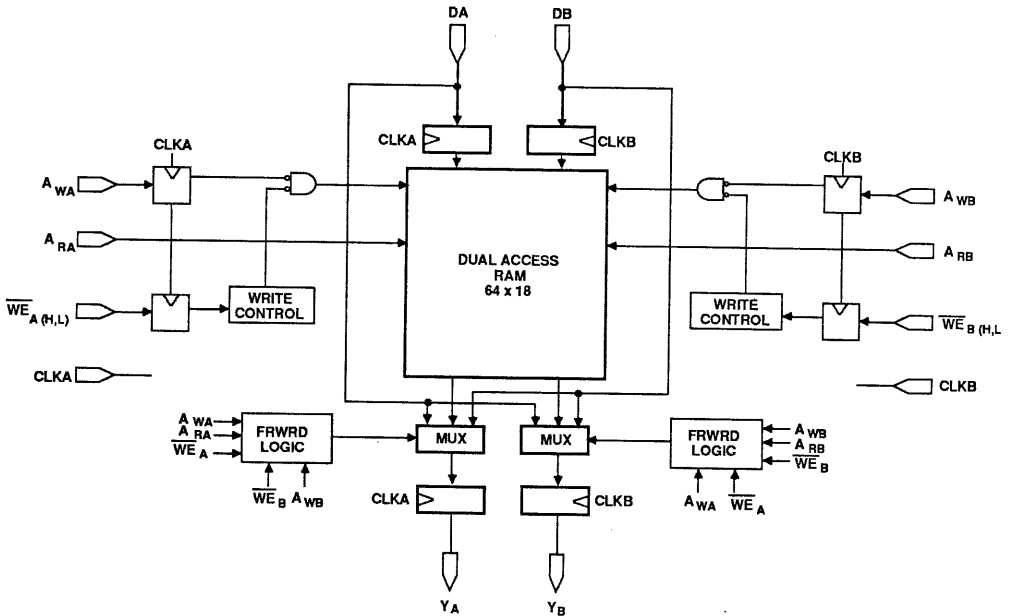
Array Processing Products/Arithmetic Accelerators

The Am29300/29C300 family is capable of very fast operation on 32-bit fixed-point numbers. When greater dynamic range is necessary, floating-point numbers are often chosen. Advanced Micro Devices offers high-speed VLSI integrated circuits designed to support the growing need for high-performance array and signal processing. Applications include graphics, image processing, communications, medical instrumentation, radar and other electronic warfare applications. Three AMD devices address these needs: Am29325/29C325 32-bit Floating-Point Processor, Am29C323 32x32-bit Multiprecision Multiplier, and Am29C327 64-bit Floating-Point Processor. These devices achieve very high speeds through a combination of innovative architecture and AMD's advanced bipolar IMOX process and CMOS process.



09372A 1-5

Figure 1-5. Am29334/29C334 Non-Pipelined Mode



09372A 1-6

Figure 1-6. Am29334/29C334 Pipelined Mode

Am29325/29C325

The Am29325/29C325 is a high-speed, single precision floating-point processor. It performs 32-bit floating-point addition, subtraction and multiplication operations in a single device, using either IEEE-P754, draft 10.0 or DEC VAX format.

Single-Cycle Execution

Since performance is the objective, all instructions—including multiply—require only one cycle to execute.

No Mandatory Pipelining

Although the Am29325/29C325 FPP has input and output registers to make it a general purpose accelerator, there are no pipeline registers internal to the floating point array. Even the I/O registers can be made transparent.

Three-Bus Architecture

The Am29325/29C325, like the Am29332/29C332, has a three-bus architecture, with two input buses and one output bus, thereby providing a bus compatible accelerator. This configuration provides high I/O bandwidth allowing the user to take full advantage of the single cycle, high-speed, floating-point ALU. Naturally, the input and output registers may be made transparent with individual clock enables. In addition, the input and output registers may be made transparent with independent feed-

through controls. The rules remain consistent – the system architecture achieves the highest performance when the component architectures do not interfere.

Powerful Instruction Set

The Am29325/29C325 executes the following instructions:

- Add (R plus S)
- Subtract (R minus S)
- Multiply (R times S)
- Constant Subtract (2 minus S)
- Integer to Floating Point Conversion
- Floating Point to Integer Conversion
- IEEE to DEC Format Conversion
- DEC to IEEE Format Conversion

The instruction (2 minus S) is provided to support the Newton-Raphson division algorithm.

Internal Data Paths Support Accumulation

The Am29325/29C325 has two internal feedback paths to facilitate two-cycle internal multiply-accumulate operation. The F1 bus can store the results of the multiply operation in an input register for subsequent accumulation. The F2 bus lets the output register function as an accumulator by making its output available as an operand for the next cycle.

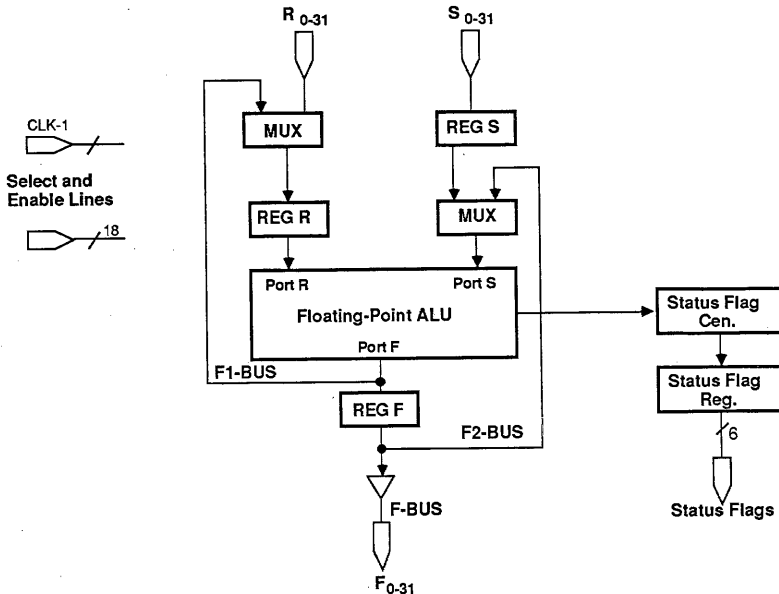


Figure 1-7. Am29325/29C325 32-Bit Floating Point Processor

09372A 1-7

Am29C325 Stand-Alone Performance

The Am29C325 is a stand-alone CMOS Floating Point Processor. When used with a simple sequencer such as the Am29C10A, it can be used as a low cost floating-point engine for applications requiring iterative algorithms such as Chebyshev and Newton-Raphson. These algorithms are used extensively in guidance, image and signal processing, and other DSP applications.

Programmable I/O Structure

To provide compatability with different system buses, controls are provided for the following options:

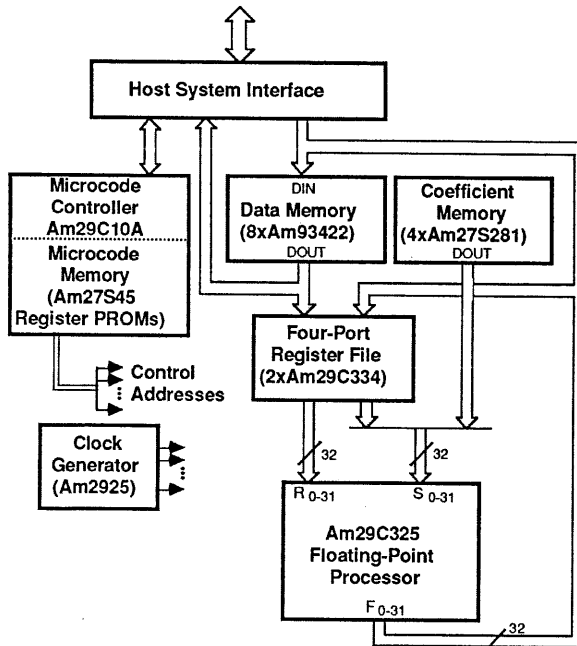
- Two 32-bit input buses and one 32-bit output bus
- One 32-bit input bus and one 32-bit output bus
- Two 16-bit input buses and one 16-bit output bus

The input modes affect only the manner in which operands are entered into the device. The operation

of the floating-point ALU is not altered. For example, in the 32-bit/one input-bus mode, the two 32-bit inputs are tied together and the two input operands are clocked into the input registers on alternate rising and falling edges of the clock. In the 16-bit, 3-bus mode, the 32-bit operands are delivered on two consecutive clock cycles in 16-bit increments.

Am29C327 Double-Precision Floating-Point Processor

The Am29C327 double-precision floating-point processor is a high performance, single VLSI device that implements an extensive floating-point and integer instruction set. It can perform operations on single-, double-, or mixed-precision operands. The three most popular floating-point formats – IEEE, DEC, and IBM – are supported. IEEE operations comply with the standard P754, with direct implementation of special features such as gradual underflow and trap handling.



09372A 1-8

Figure 1-8. Microcoded Floating Point Co-Processor

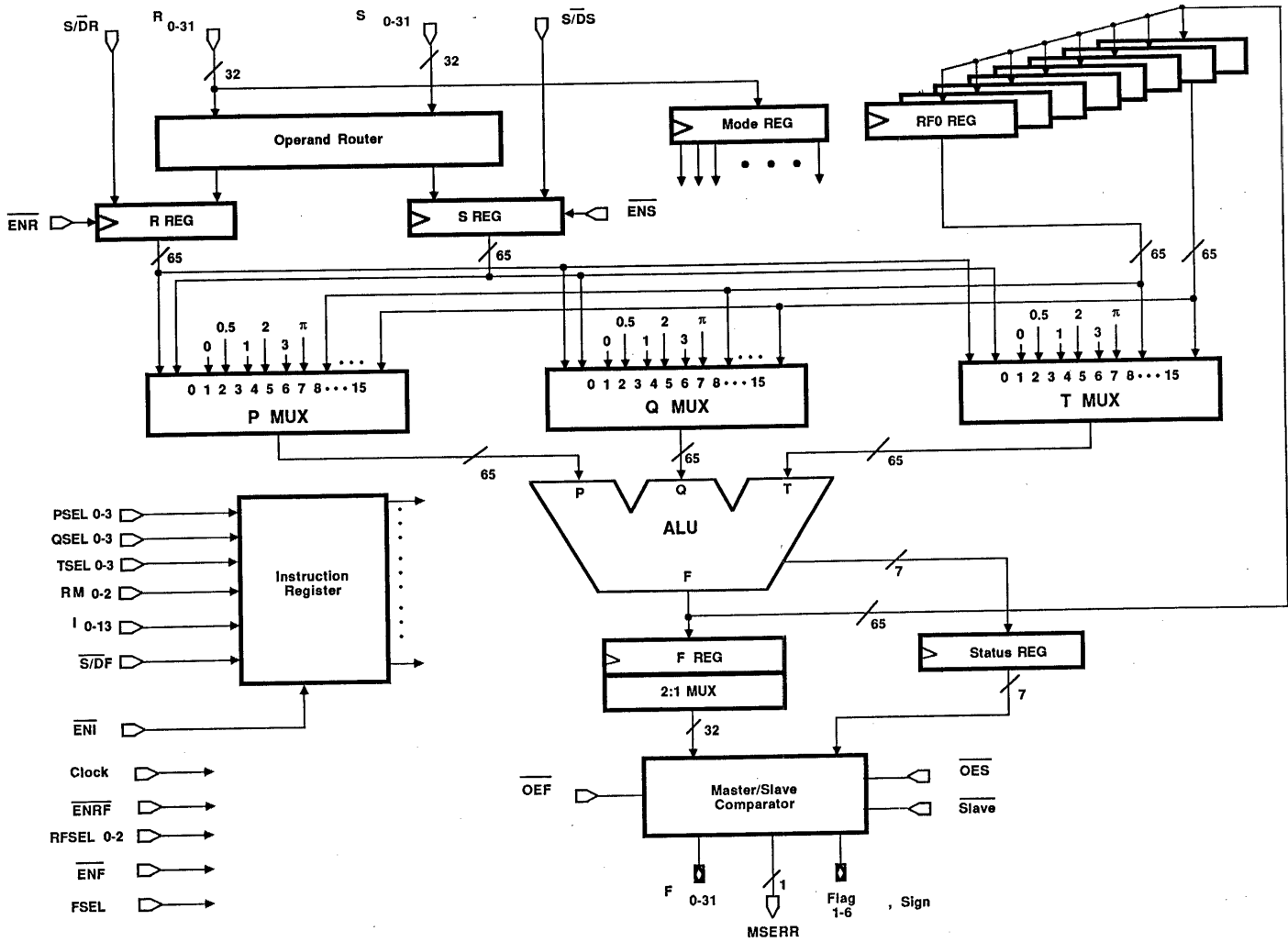


Figure 1-9. Am29C327 Floating-Point Processor

Flow-Through or Pipelined

Operations can be performed in either of two modes: flow-through or pipelined. In the flow-through mode, the ALU is completely combinatorial; this mode is best suited for scalar operations. Pipelined mode divides the ALU into one or two pipelined stages for use in vector operations, as is often found in graphics or signal processing.

Three-Bus Architecture

The Am29C327 has two input buses and one output bus – a three-bus architecture just like the Am29C325 floating-point processor. It provides flexibility and ease of interface, making it a very high performance accelerator.

Input/Output Modes

The Am29C327 supports eight I/O modes which provide a flexible interface to a variety of 32-bit and 64-bit systems. The input buses can be configured as separate 32-bit input buses or as a single 64-bit input bus. It is possible to load two 64-bit operands in a single clock cycle. The input modes are:

- 32-bit, double-cycle, LSWs first
- 32-bit, double-cycle, MSWs first
- 32-bit, single-cycle, LSWs first
- 32-bit, single-cycle, MSWs first
- 64-bit, double-cycle, R first
- 64-bit, double-cycle, S first
- 64-bit, single-cycle, R first
- 64-bit, single-cycle, S first

Integer or Floating-Point

In addition to supporting 32-bit and 64-bit integer operations, the Am29C327 supports the following floating-point formats in single- or double-precision:

- IEEE P754 version 10.1
- DEC F, DEC D, and DEC G formats
- IBM system 370 format.

Conversion between the floating-point formats and conversion between floating-point and integer formats are also provided. This is a very powerful feature not available in any other architecture.

Mixed-Precision Operations

All Am29C327 instructions, floating-point or integer, can be performed in either single- or double-precision operands. In addition, the user can elect to mix precisions within an operation. All operations are internally performed in double precision; the user specifies the desired precision of the input and output operands. The

necessary precision conversions are made in concert with the selected operation, with no additional cycle-time overhead.

Register File and Internal Datapath Support Compound Operations

The ALU of the Am29C327 has three data input ports and can perform operations of the form $(A*B)+C$. An eight-deep register file for storing immediate results used in recursive operations, and the on-chip 64-bit datapath, facilitates compound operations such as Newton-Raphson division, sum-of-products, and transcendentals.

Comprehensive Floating-Point and Integer Instruction Sets

The Am29C327 implements an extensive number of arithmetic and logical instructions. These instructions fall into the following categories:

- addition/subtraction
- multiplication
- multiplication/ accumulation
- comparison
- max/min
- saturation (clipping)
- rounding to integral value
- absolute value, negation
- reciprocal seed generation
- floating-point \leftarrow \rightarrow floating-point conversion
- floating-point \leftarrow \rightarrow integer conversion
- integer \leftarrow \rightarrow integer conversion
- pass operand
- logical operations; e.g. AND, OR, XOR, NOT
- move data

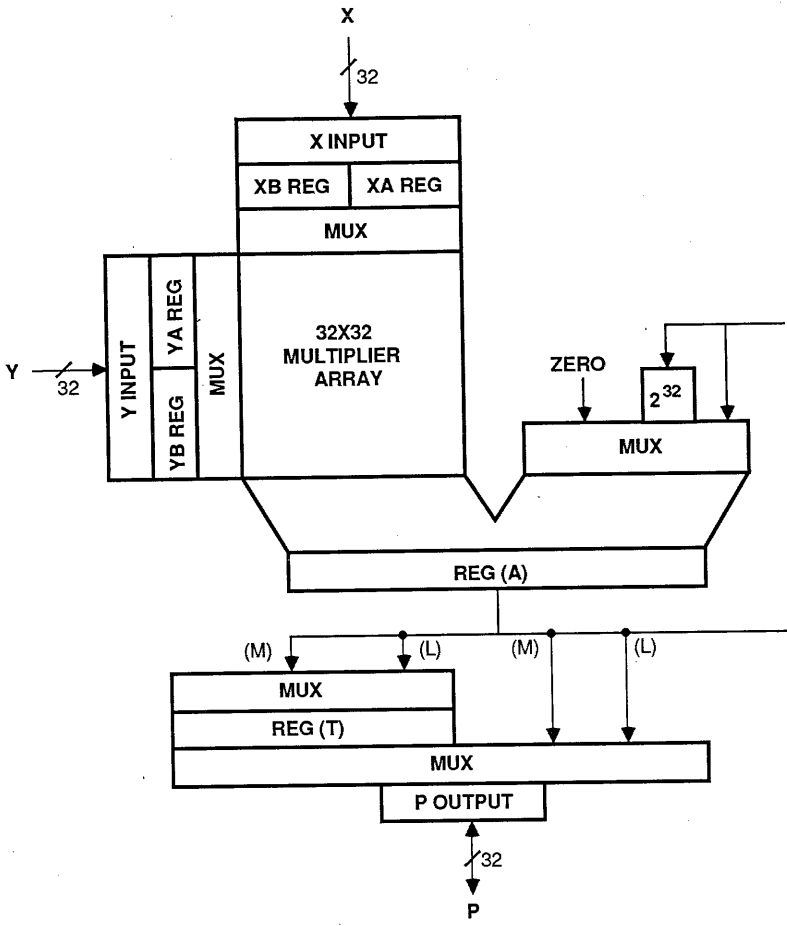
By concatenating these operations, the user can also perform division, square-root extraction, polynomial evaluation, and other functions not implemented directly.

Am29C323 Multiplier

The Am29C323 is a high-speed parallel 32x32-bit multiplier designed to speed up systems using fixed or floating-point notation.

Three-Bus Architecture

Just like other members of the family, the Am29C323 has two input buses and one output bus. This configuration provides high I/O bandwidth, allowing the user to take full advantage of the high-speed parallel multiplier core of the device.



09372A 1-10

Figure 1-10. Am29C323 32x32 Parallel Multiplier

Multiprecision Multiplication Made Easy

By including 32-bit shift and accumulate to generate partial products, the internal architecture of the Am29C323 supports fast multiprecision multiplication. Both input ports have dual 32-bit registers, and the output port can select from a 67-bit product register, a 32-bit temporary register, or directly from the 32x32-bit multiplier array. A complete 32x32-bit clocked multiplication takes a single cycle (naturally – and with no pipelining!). Multiprecision multiplication uses the shift and accumulate logic to collect partial products starting with the least significant product. The number of cycles depends upon the input data width, with three-cycle latency, as shown in the table below. By using the I/O registers for pipelin-

ing, much greater throughput can be achieved. For example, by overlapping 64x64-bit operations, a full 128-bit product is available every four cycles. Multiplying the mantissas of two double-precision 64-bit floating-point numbers, for example, is one possible application of this high speed multiprecision multiplication capacity.

Operands	Number of Cycles	
	Single Product	Overlapped Operations
32x32	1	1
64x64	7	4
96x96	12	9
128x128	19	16

Registered Buses

All buses in the device are registered, and each register has its own Clock Enable. The device operates from a single clock, ideal for microprogrammed systems. All ports – input, output, and instruction – can be made transparent independently.

Complete Interlocking Fault Detection

To enhance system reliability by ensuring data integrity and correct hardware operation, the family supports both master/slave fault detection and data path parity. The system features byte parity checking on the inputs and byte parity generation on the outputs of the Am29332/29C332 ALU and Am29C323 32x32-bit multiplier. Also, the organization of the Am29334/29C334 64x18 register file accommodates parity bits for each byte. The parity mechanism assures data path integrity. Major functional blocks—Am29332/29C332 ALU, Am29331/29C331 sequencer, Am29C323 32x32 bit multiplier, and

Am29C327 64-bit floating-point processor—have “master/slave fault detection” to ensure correct operation without having to carry parity through complex internal logic (shifters, mask generators, etc.) and without having to pay the resulting delay penalties. In master/slave mode, two functional units are connected in parallel with one unit doing the actual operation and the other checking the result, on a cycle-by-cycle, bit-by-bit basis. The master is used for the normal data path. In the slave, however, all outputs become inputs, and the slave compares the outputs of the master with its own internally generated result. If the two don’t match, an error signal is generated, triggering an interrupt at the microinstruction level. No specialized software is required for the master/slave scheme. Also, the designer can choose to impose redundancy at the component or board level. The parity mechanism and the master/slave concept, which use cost-effective hardware rather than expensive software, provide a comprehensive solution for fault tolerant systems.

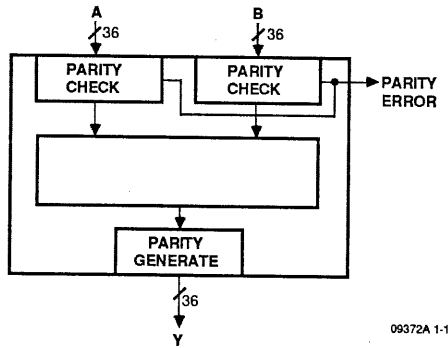


Figure 1-11. Input Parity Checking / Output Parity Checking

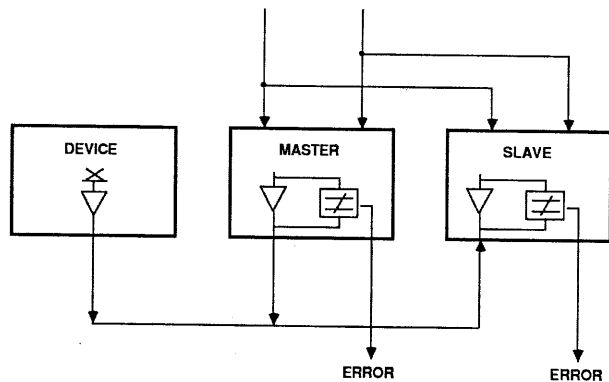


Figure 1-12. Master/Slave Error Checking

Am29337 16-Bit Bounds Checker

The need for simple yet sophisticated functionality and board space savings created the Am29337, a 16-bit bounds checker. This product provides inexpensive, easy-to-use solutions for the following applications:

- intelligent address decoder
- window clipping in graphics
- filter in DSP
- memory protection systems
- RISC processors
- multi/parallel processors
- logic analyzers
- tag/data buffers

The Am29337 compares incoming 16-bit data against both lower and upper bounds and reports whether the

data is inside or outside the bounds. It can be cascaded for 32-bit data and longer without sacrificing speed.

The Am29337 is housed in a 400 mil ceramic 28-pin DIP for board space savings.

User Benefits

- Replaces MSI devices, saves board space
- Low-cost solution compared to conventional alternatives

Distinctive Features

- Double Comparators compare a 16-bit input number against a lower and an upper limit
- 16-bit operation, cascadable to longer words
- Compares signed or unsigned numbers

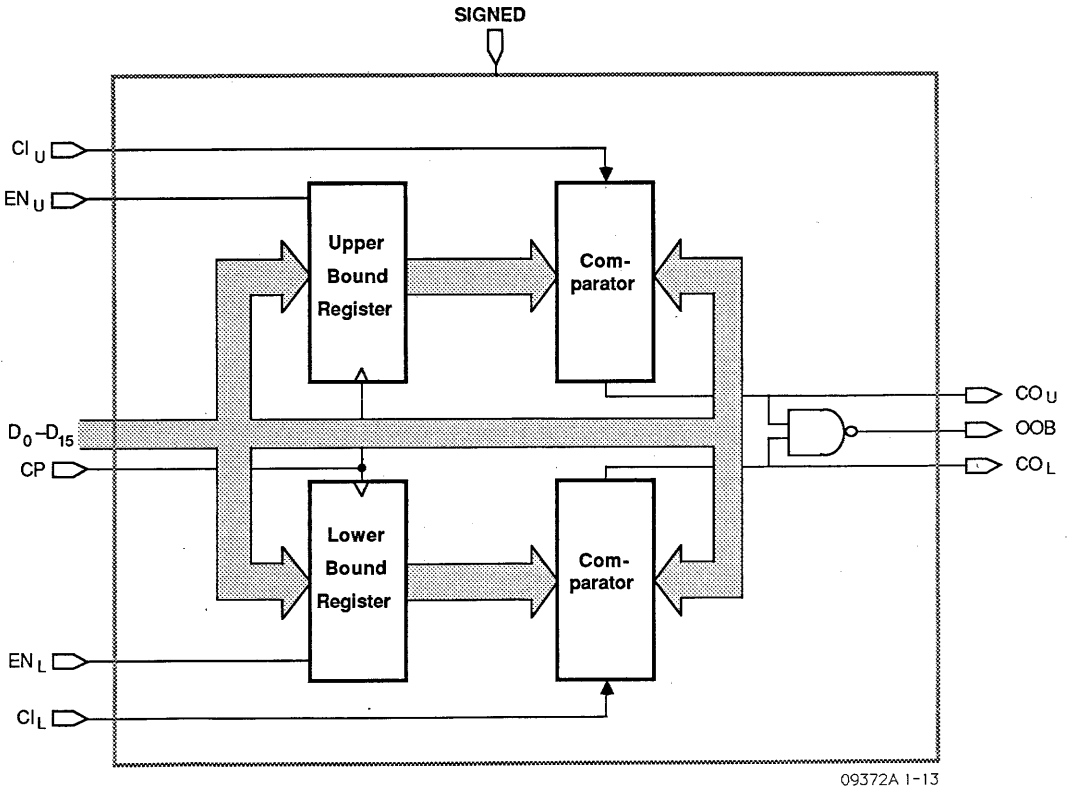


Figure 1-13. Am29337 Block Diagram

Am29338 32-Bit Byte Queue

The Am29338 is a general purpose 32-bit intelligent FIFO that allows up to four bytes to be queued or de-queued in a single cycle.

Fabricated with AMD's IMOX-S2 technology and housed in a 120-pin PGA, the Am29338 meets the requirements for a high-speed FIFO buffer with minimum real estate. The part will also be made available in high-speed, low-power 1.2 micron CMOS technology.

Features of the Am29338 include:

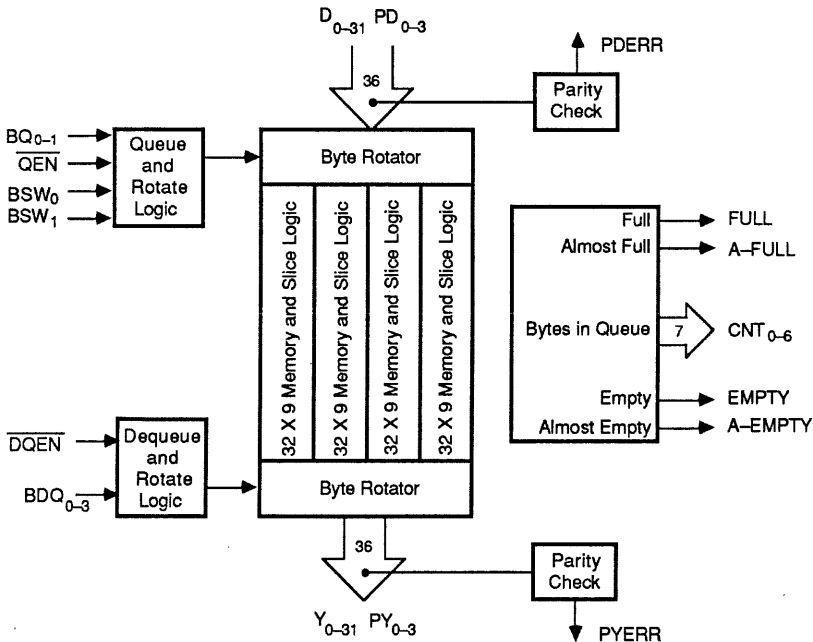
- Queuing of up to 128 bytes
- Queuing or de-queuing of up to 4 bytes at a time
- Byte rotation on the inputs and outputs
- Asynchronous/synchronous operations
- Accepts 8-, 16-, 24-, and 32-bit input data
- Repetitive queuing of block data
- Almost empty/full signal if less than 4 bytes available

Significant User Benefits

The Am29388 is an excellent choice for a wide variety of system design problems. Its benefits include: a shorter design cycle when compared with implementing the same functions with traditional FIFOs, higher performance, off-the-shelf functionality, less board space, and less power than the separate parts needed to combine this logic.

Applications

- Hardware mailbox between two heterogeneous processors
- I/O bus buffers between a processor and controller
- Instruction prefetch queue for byte addressable microprocessor systems
- Write buffer between CPU and main memory
- Bus conversions, 8-, 16-, 24-, and 32-bits.



09372A 1-14

Figure 1-14. Am29338 Block Diagram

1.3 A.C. AND D.C. PARAMETER DEFINITIONS

Definition of A.C. Switching Terms

f_{MAX}	The highest operating clock frequency.
t_{PLH}	The propagation delay time from an input change to an output LOW-to-HIGH transition.
t_{PHL}	The propagation delay time from an input change to an output HIGH-to-LOW transition.
t_{PW}	Pulse width. The time between the leading and trailing edges of a pulse.
t_r	Rise time. The time required for a signal to change from 10% to 90% of its measured values.
t_f	Fall time. The time required for a signal to change from 90% to 10% of its measured values.
t_s	Set-up time. The time interval for which a signal must be applied and maintained at one input terminal before an active transition occurs at another terminal.
t_h	Hold time. The time interval for which a signal must be retained at one input after an active transition occurs at another input terminal.
t_{HZ}	HIGH to disable. The delay time from a control input change to the output transition from the HIGH-level to high-impedance (measured at 0.5V change).
t_{LZ}	LOW to disable. The delay time from a control input change to the output transition from the LOW-level to high-impedance transition (measured at 0.5 V change).
t_{ZH}	Enable HIGH. The delay time from a control input change to the output transition from high-impedance to HIGH-level.
t_{ZL}	Enable LOW. The delay time from a control input change to the output transition from high-impedance to LOW-level.

Definition of D.C. Terms

C_{PD}	Power dissipation capacitance used to determine the no-load dynamic current consumption.
H	HIGH, applying to a HIGH voltage level.
L	LOW, applying to a LOW voltage level.
I	Input
O	Output
Negative Current	Current flowing out of the device.
Positive Current	Current flowing into the device.
I_{IL}	LOW-level input current with a specified LOW-level voltage applied.
I_{IH}	HIGH-level input current with a specified HIGH-level voltage applied.
I_{OL}	LOW-level output current.
I_{OH}	HIGH-level output current.
I_{SC}	Output short-circuit source current.
I_{CC}	Supply current drawn by the device from the V_{CC} power supply.
I_{OZH}	Three-state off-state output current, HIGH- level voltage applied.
I_{OZL}	Three-state off-state output current, LOW- level voltage applied.

CHAPTER 1

Am29300/29C300 Family Overview

V_{CC}	The range of supply voltage over which the device is guaranteed to operate.
V_{IL}	The highest input voltage that is guaranteed to be recognized by the device as a logic LOW.
V_{IH}	The lowest input voltage that is guaranteed to be recognized by the device as a logic HIGH.
V_{OL}	The highest logic LOW voltage guaranteed at the output terminal while sinking the specified load current I_{OL} .
V_{OH}	The lowest logic HIGH voltage guaranteed at the output terminal when sourcing the specified source current I_{OH} .
I_{EE}	The supply current drawn by the device from the V_{EE} power supply for an ECL circuit.
V_{EE}	Most negative power supply for an ECL circuit.

CHAPTER 2

CMOS Family

Am29C331 CMOS 16-Bit Microprogram Sequencer	2-1
Am29C332 CMOS 32-Bit Arithmetic Logic Unit	2-38
Am29C334 CMOS Four-Port Dual-Access Register File	2-76
Am29C325 CMOS 32-Bit Floating-Point Processor*	2-94
Am29C327 CMOS Double-Precision Floating-Point Processor*	2-95

* Front page only of data sheet. See Chapter 4 for complete data sheet.

Am29C331

CMOS 16-Bit Microprogram Sequencer



Am29C331

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- **16-Bits Address up to 64K Words**
Supports 110-ns microcycle time for a 32-bit high-performance system when used with the other members of the Am29C300 Family.
- **Speed Select**
Supports 80-ns system cycle time.
- **Real-Time Interrupt Support**
Micro-trap and interrupts are handled transparently at any microinstruction boundary.
- **Built-In Conditional Test Logic**
Has twelve external test inputs, four of which are used to internally generate an additional four test conditions. Test multiplexer selects one out of 16 test inputs.
- **Break-Point Logic**
Built-in address comparator allows break-points in the microcode for debugging and statistics collection.
- **Master/Slave Error Checking**
Two sequencers can operate in parallel as a master and a slave. The slave generates a fault flag for unequal results.
- **33-Level Stack**
Provides support for interrupts, loops, and subroutine nesting. It can be accessed through the D-bus to support diagnostics.

GENERAL DESCRIPTION

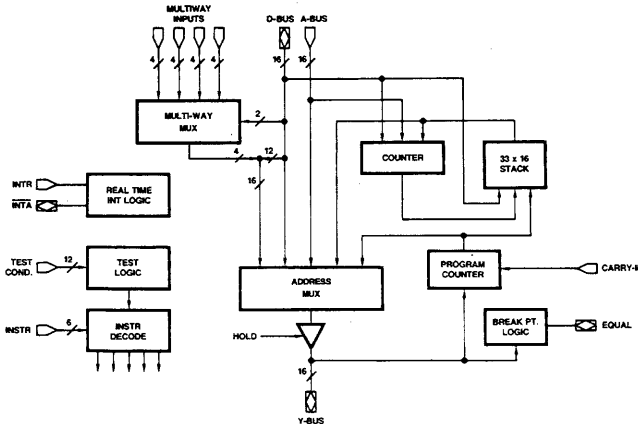
The Am29C331 is a 16-bit wide, high-speed single-chip sequencer designed to control the execution sequence of microinstructions stored in the microprogram memory. The instruction set is designed to resemble high-level language constructs, thereby bringing high-level language programming to the micro level.

The Am29C331 is interruptible at any microinstruction boundary to support real-time interrupts. Interrupts are handled transparently to the microprogrammer as an unexpected procedure call. Traps are also handled transparently at any microinstruction boundary. This feature allows re-execution of the prior microinstruction. Two separate buses are provided to bring a branch address directly into the chip from two sources to avoid slow turn-on and turn-off times for different sources connected to the data-input bus. Four

sets of multiway inputs are also provided to avoid slow turn-on and turn-off times for different branch-address sources. This feature allows implementation of table look-up or use of external conditions as part of a branch address. The 33-deep stack provides the ability to support interrupts, loops, and subroutine nesting. The stack can be read through the D-bus to support diagnostics or to implement multitasking at the micro-architecture level. The master/slave mode provides a complete function check capability for the device.

Fabricated using Advanced Micro Devices' 1.6 micron CMOS process, the Am29C331 is powered by a single 5-volt supply. The device is housed in a 120-terminal pin-grid array package.

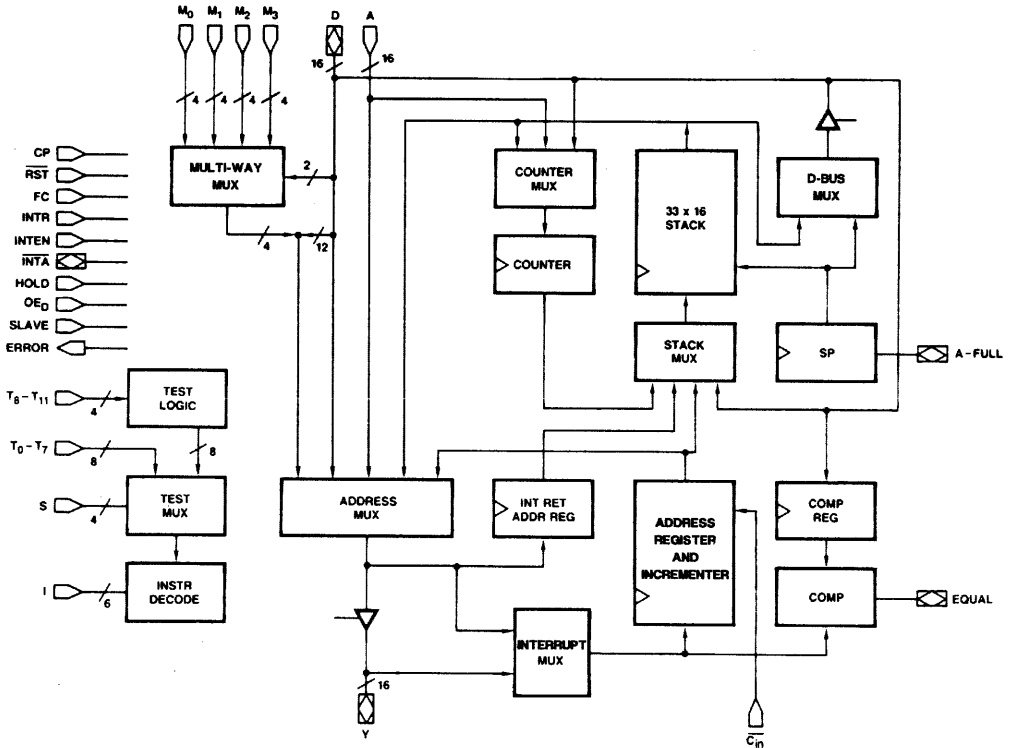
SIMPLIFIED BLOCK DIAGRAM



BD006091

RELATED AMD PRODUCTS

Part No.	Description
Am29114	Vectored Priority Interrupt Controller
Am29116	High-Performance Bipolar 16-Bit Microprocessor
Am29C116	High-Performance CMOS 16-Bit Microprocessor
Am29PL141	Field-Programmable Controller
Am29C323	CMOS 32-Bit Parallel Multiplier
Am29325	32-Bit Floating-Point Processor
Am29C325	CMOS 32-Bit Floating-Point Processor
Am29332	32-Bit Extended Function ALU
Am29C332	CMOS 32-Bit Extended Function ALU
Am29334	64 x 18 Four-Port, Dual-Access Register File
Am29C334	CMOS 64 x 18 Four-Port Dual-Access Register File
Am29337	16-Bit Bounds Checker
Am29338	Byte Queue



BD006102

Figure 1. Am29C331 Detailed Block Diagram

CONNECTION DIAGRAM

120-Lead PGA*

	A	B	C	D	E	F	G	H	J	K	L	M	N
1	M0,0	M1,0	M2,0	M2,1	\overline{CIN}	M1,2	M1,3	M2,3	GND	\overline{RST}	INTR	SLAVE	D15
2	D0	A0	M3,0	M1,1	M0,2	M2,2	M0,3	M3,3	EQUAL	OED	INTEN	HOLD	A15
3	VCC	Y0	D1	M0,1	M3,1	GND	M3,2	VCC	A-FULL	ERROR	\overline{INTA}	Y15	VCC
4	A1	Y1	D2								D14	A14	Y14
5	GND	A2	Y2								D13	A13	GND
6	A3	D3	GND								GND	D12	Y13
7	Y3	D4	A4								A12	Y12	D11
8	D5	Y4	VCC								VCC	Y11	A11
9	GND	A5	Y5								D10	A10	GND
10	D6	A6	Y6								Y10	D9	A9
11	VCC	D7	T3	T6	GND	T10	T11	I0	VCC	I3	Y9	D8	VCC
12	A7	T1	T2	T5	GND	T7	S0	S1	VCC	I2	I4	A8	Y8
13	Y7	T0	T9	T4	GND	T8	CP	S3	VCC	I1	S2	I5	FC

CD010380

*Pins facing up.

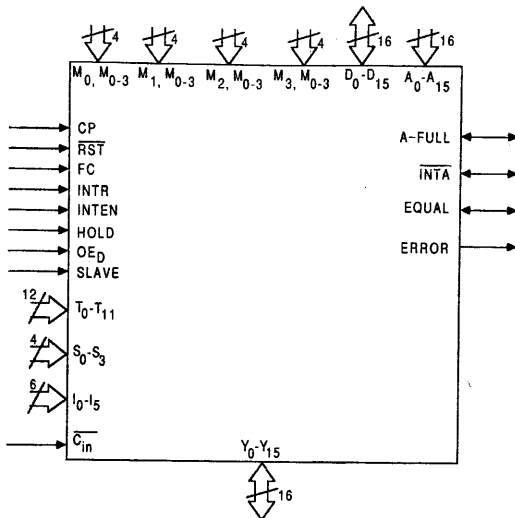
PIN DESIGNATIONS
(Sorted by Pin No.)

PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.
			C-5	Y ₂	115	H-2	M _{3, 3}	10	M-5	A ₁₃	80
			C-6	GND	113	H-3	VCC	68	M-6	D ₁₂	81
			C-7	A ₄	52	H-11	I ₀	34	M-7	Y ₁₂	82
			C-8	VCC	53	H-12	S ₁	95	M-8	Y ₁₁	25
A-1	M _{0, 0}	1	C-9	Y ₅	109	H-13	S ₃	94	M-9	A ₁₀	86
A-2	D ₀	120	C-10	Y ₆	48	J-1	GND	11	M-10	D ₉	87
A-3	VCC	59	C-11	T ₃	44	J-2	EQUAL	71	M-11	D ₈	89
A-4	A ₁	58	C-12	T ₂	104	J-3	A-FULL	70	M-12	A ₈	30
A-5	GND	56	C-13	T ₉	41	J-11	VCC	37	M-13	I ₅	91
A-6	A ₃	114	D-1	M _{2, 1}	4	J-12	VCC	38	N-1	D ₁₅	16
A-7	Y ₃	54	D-2	M _{1, 1}	63	J-13	VCC	39	N-2	A ₁₅	76
A-8	D ₅	51	D-3	M _{0, 1}	3	K-1	RST	13	N-3	VCC	17
A-9	GND	50	D-11	T ₆	102	K-2	OE _D	72	N-4	Y ₁₄	19
A-10	D ₆	49	D-12	T ₅	43	K-3	ERROR	12	N-5	GND	20
A-11	VCC	47	D-13	T ₄	103	K-11	I ₃	92	N-6	Y ₁₃	21
A-12	A ₇	106	E-1	C _{in}	5	K-12	I ₂	33	N-7	D ₁₁	24
A-13	Y ₇	46	E-2	M _{0, 2}	65	K-13	I ₁	93	N-8	A ₁₁	84
B-1	M _{1, 0}	61	E-3	M _{3, 1}	64	L-1	INTR	14	N-9	GND	26
B-2	A ₀	60	E-11	GND	97	L-2	INTEN	74	N-10	A ₉	28
B-3	Y ₀	119	E-12	GND	98	L-3	INTA	73	N-11	VCC	29
B-4	Y ₁	117	E-13	GND	99	L-4	D ₁₄	18	N-12	Y ₈	90
B-5	A ₂	116	F-1	M _{1, 2}	6	L-5	D ₁₃	79	N-13	FC	31
B-6	D ₃	55	F-2	M _{2, 2}	66	L-6	GND	23			
B-7	D ₄	112	F-3	GND	8	L-7	A ₁₂	22			
B-8	Y ₄	111	F-11	T ₁₀	100	L-8	VCC	83			
B-9	A ₅	110	F-12	T ₇	42	L-9	D ₁₀	85			
B-10	A ₆	108	F-13	T ₈	101	L-10	Y ₁₀	27			
B-11	D ₇	107	G-1	M _{1, 3}	9	L-11	Y ₉	88			
B-12	T ₁	45	G-2	M _{0, 3}	67	L-12	I ₄	32			
B-13	T ₀	105	G-3	M _{3, 2}	7	L-13	S ₂	35			
C-1	M _{2, 0}	2	G-11	T ₁₁	40	M-1	SLAVE	75			
C-2	M _{3, 0}	62	G-12	S ₀	36	M-2	HOLD	15			
C-3	D ₁	118	G-13	CP	96	M-3	Y ₁₅	77			
C-4	D ₂	57	H-1	M _{2, 3}	69	M-4	A ₁₄	78			

PIN DESIGNATIONS
(Sorted by Pin Name)

PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.
-	-	37	D ₈	M-11	89	INTEN	L-2	74	T ₆	D-11	102
-	-	39	D ₉	M-10	87	INTR	L-1	14	T ₇	F-12	42
-	-	97	D ₁₀	L-9	85	M _{0, 0}	A-1	1	T ₈	F-13	101
-	-	99	D ₁₁	N-7	24	M _{0, 1}	D-3	3	T ₉	C-13	41
A-FULL	J-3	70	D ₁₂	M-6	81	M _{0, 2}	E-2	65	T ₁₀	F-11	100
A ₀	B-2	60	D ₁₃	L-5	79	M _{0, 3}	G-2	67	T ₁₁	G-11	40
A ₁	A-4	58	D ₁₄	L-4	18	M _{1, 0}	B-1	61	GND	J-1	11
A ₂	B-5	116	D ₁₅	N-1	16	M _{1, 1}	D-2	63	GND	N-5	20
A ₃	A-6	114	GND	E-12	97	M _{1, 2}	F-1	6	GND	A-9	50
A ₄	C-7	52	GND	E-13	98	M _{1, 3}	G-1	9	GND	N-9	26
A ₅	B-9	110	GND	E-11	99	M _{2, 0}	C-1	2	GND	A-5	56
A ₆	B-10	108	GND	F-3	8	M _{2, 1}	D-1	4	VCC	N-3	17
A ₇	A-12	106	GND	L-6	23	M _{2, 2}	F-2	66	VCC	N-11	29
A ₈	M-12	30	GND	C-6	113	M _{2, 3}	H-1	69	VCC	A-3	59
A ₉	N-10	28	VCC	J-13	38	M _{3, 0}	C-2	62	VCC	A-11	47
A ₁₀	M-9	86	VCC	H-3	68	M _{3, 1}	E-3	64	Y ₀	B-3	119
A ₁₁	N-8	84	VCC	C-8	53	M _{3, 2}	G-3	7	Y ₁	B-4	117
A ₁₂	L-7	22	VCC	L-8	83	M _{3, 3}	H-2	10	Y ₂	C-5	115
A ₁₃	M-5	80	VCC	J-12	37	OE _D	K-2	72	Y ₃	A-7	54
A ₁₄	M-4	78	VCC	J-11	39	RST	K-1	13	Y ₄	B-8	111
A ₁₅	N-2	76	EQUAL	J-2	71	S ₀	G-12	36	Y ₅	C-9	109
C _{in}	E-1	5	ERROR	K-3	12	S ₁	H-12	95	Y ₆	C-10	48
CP	G-13	96	FC	N-13	31	S ₂	L-13	35	Y ₇	A-13	46
D ₀	A-2	120	HOLD	M-2	15	S ₃	H-13	94	Y ₈	N-12	90
D ₁	C-3	118	I ₀	H-11	34	SLAVE	M-1	75	Y ₉	L-11	88
D ₂	C-4	57	I ₁	K-13	93	T ₀	B-13	105	Y ₁₀	L-10	27
D ₃	B-6	55	I ₂	K-12	33	T ₁	B-12	45	Y ₁₁	M-8	25
D ₄	B-7	112	I ₃	K-11	92	T ₂	C-12	104	Y ₁₂	M-7	82
D ₅	A-8	51	I ₄	L-12	32	T ₃	C-11	44	Y ₁₃	N-6	21
D ₆	A-10	49	I ₅	M-13	91	T ₄	D-13	103	Y ₁₄	N-4	19
D ₇	B-11	107	INTA	L-3	73	T ₅	D-12	43	Y ₁₅	M-3	77

LOGIC SYMBOL



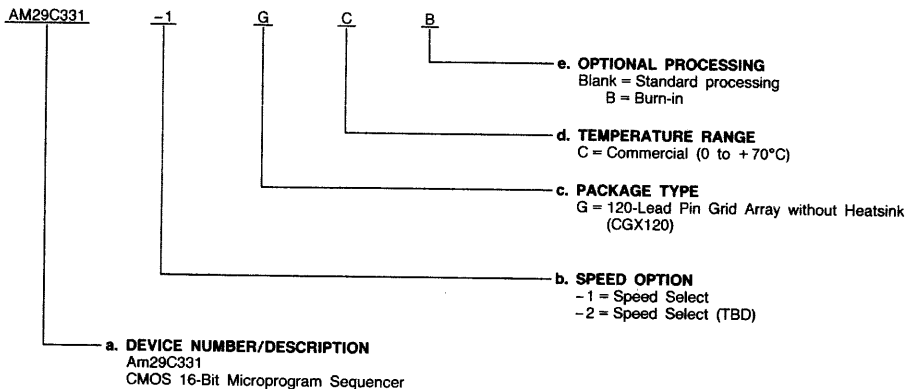
LS002872

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. Device Number
- b. Speed Option (if applicable)
- c. Package Type
- d. Temperature Range
- e. Optional Processing



Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

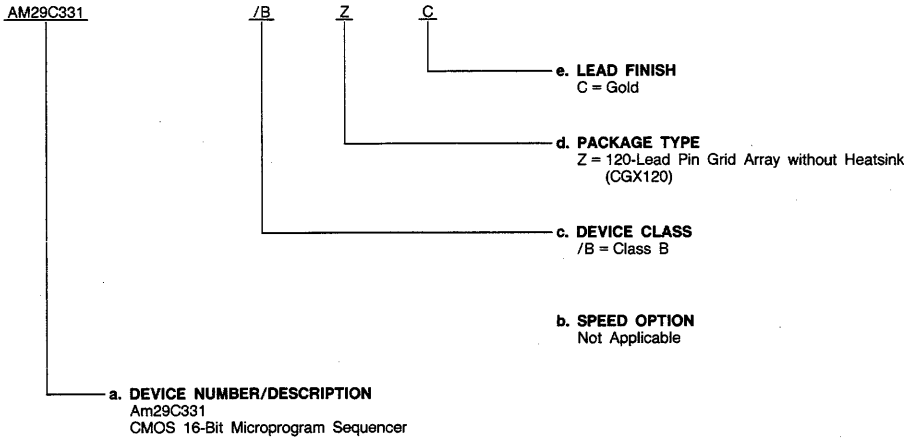
Valid Combinations	
AM29C331	GC, GCB
AM29C331-1	

MILITARY ORDERING INFORMATION

APL Products

AMD products for Aerospace and Defense applications are available in several packages and operating ranges. APL (Approved Products List) products are fully compliant with MIL-STD-883C requirements. The order number (Valid Combination) for APL products is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Device Class**
- d. **Package Type**
- e. **Lead Finish**



Valid Combinations	
AM29C331	/BZC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations or to check for newly released valid combinations.

Group A Tests

Group A tests consist of Subgroups 1, 2, 3, 7, 8, 9, 10, 11.

PIN DESCRIPTION

A₀ - A₁₅ Alternate Data (Input)

Input to address multiplexer and counter.

A-FULL Almost Full (Bidirectional; Three-State)

Indicates that $28 \leq SP \leq 63$ (meaning there are five or less empty locations left on stack). Also active during stack underflow.

C_{in} Carry In (Input, Active LOW)

Carry-in to the incrementer.

CP Clock Pulse (Input)

Clocks sequencer at the LOW-to-HIGH transition.

D₀ - D₁₅ Data (Bidirectional, Three-State)

Input to address multiplexer, counter, stack, and comparator register. Output for stack and stack pointer.

EQUAL Equal (Bidirectional, Three-State)

Indicates that the address comparator is enabled and has found a match.

ERROR Error (Output)

Indicates a master/slave error in the slave mode. Indicates a malfunctioning driver or contention of any output in the master mode.

FC Force Continue (Input)

Overrides instruction with CONTINUE.

HOLD Hold (Input)

Stops the sequencer and three-states the outputs.

I₀ - I₅ Instruction (Input)

Selects one of 64 instructions.

INTA Interrupt Acknowledge (Bidirectional; Three-State, Active LOW)

Indicates that an interrupt is accepted.

INTEN Interrupt Enable (Input)

Enables interrupts.

INTR Interrupt Request (Input)

Requests the sequencer to interrupt execution.

M₀₋₃, 0-3 Multiway (Input)

Four sets of multiway inputs providing 16-way branches. The first index refers to the set number.

OE_D Output Enable - D-Bus (Input)

Enables the D-bus driver, provided that the sequencer is not in the hold or slave mode.

RST Reset (Input; Active LOW)

Resets the sequencer.

S₀ - S₃ Select (Input)

Selects one of 16 test conditions.

SLAVE Slave (Input)

Makes the sequencer a slave.

T₀ - T₁₁ Test (Input)

Provides external test inputs.

Y₀ - Y₁₅ Address (Bidirectional; Three-State)

Output of microcode address. Input for interrupt address.

FUNCTIONAL DESCRIPTION

Architecture

The major blocks of the sequencer are the address multiplexer, the address register (AR), the stack (with the top of stack denoted TOS), the counter (C), the test multiplexer with logic, and the address comparison register (R) (Figure 1). The bidirectional D-bus provides branch addresses and iteration counts; it also allows access to the stack from the outside. The A-bus may be used for map addresses. There are four sets of four-bit multiway branch inputs (M). The bidirectional Y-bus either outputs microprogram addresses or inputs interrupt addresses. The buses are all 16 bits wide. Figure 1 shows a detailed block diagram of the sequencer.

Address Multiplexer

The address multiplexer can select an address from any of five sources:

- 1) A branch address supplied by the D-bus
- 2) A branch address supplied by the A-bus

3) A multiway-branch address

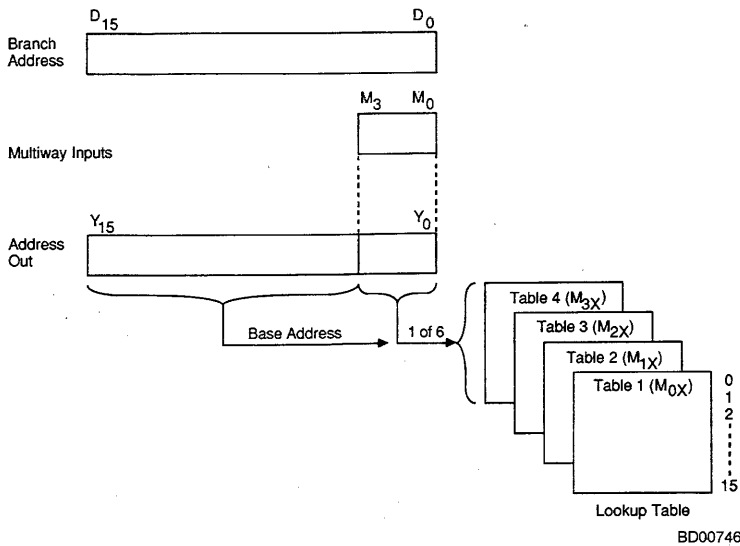
4) A return or loop address from the top of stack

5) The next sequential address from the incrementer

Multiway-Branch Address

A multiway-branch address is formed by substituting the lower four bits of the address on the D-bus (D₃, D₂, D₁, D₀) with one of the four sets (M_{0X}, M_{1X}, M_{2X}, or M_{3X}) of four-bit multiway-branch addresses. The multiway-branch set is selected by the number D₁D₀, while the bits D₃ and D₂ are "don't cares" (see Figure 2).

D ₁	D ₀	Multiway Set Selected
0	0	M _{0X}
0	1	M _{1X}
1	0	M _{2X}
1	1	M _{3X}



- Notes: 1. D_{15} and D_0 select one out of four multiway sets. D_3 and D_2 are "don't cares."
 2. Each set of $M_{3X} - M_{0X}$ can select one of sixteen locations. The multiway-branch address is the concatenation of $D_{15} - D_4$ (base address) and $M_{X3} - M_{X0}$.
 3. For a given base address, there can be four look-up tables, each sixteen deep.

Figure 2. Multiway Branch

Address Register and Incrementer

The address register contains the current address. It is loaded from the interrupt multiplexer and feeds the incrementer. The incrementer is inhibited if \overline{C}_{IN} is taken HIGH.

Stack

A 33-word-deep and 16 bit-wide stack provides first-in last-out storage for return addresses, loop addresses, and counter values. Items to be pushed come from the incrementer, the interrupt-return-address register, the counter, or the D-bus. Items popped go to the address multiplexer, the counter, or the D-bus.

The access to the stack via the D-bus may be used for context switching, stack extension, or diagnostics. As the stack is only accessible from the top, stack extension is done by temporarily storing the whole or some lower part of the stack outside the sequencer. The save and the later restore are done with pop and push operations, respectively, at balanced points in the microprogram; for example, points with the same stack depth. The internal D-bus driver must be turned on when popping an item to the D-bus; if the driver is off, the item will be unstacked instead. The driver is normally turned on when the Output Enable signal is asserted and the sequencer is not being reset ($OE_D = 1$, $RST = 1$).

The stack pointer is a modulo 64 counter, which is incremented on each push and decremented on each pop. The stack pointer is reset to zero when the sequencer is reset, but the pointer may also be reset by instruction. Thus, the stack pointer indicates the number of items on the stack as long as stack overflow or underflow has not occurred. Overflow happens when an item is pushed onto a full stack, whereby the item at the bottom of the stack is overwritten. Underflow

happens when an item is popped from an empty stack; in this case the item is undefined.

In the case of stack overflow, the SP is incremented for every push after overflow. Thus, immediately after the first occurrence of stack overflow, the SP will be equal to 34. Subsequent pushes will increment the SP to 35, 36 ... 61, 62, 63, 0, 1, etc. In the case of stack underflow, the SP is decremented for every pop after underflow. Thus, immediately after the first occurrence of stack underflow, the SP will be equal to 63. Subsequent pops will decrement the SP to 62, 61, ... 2, 1, 0, 63, etc.

The contents of the stack pointer are present on the D-bus for all instructions except POP D, provided the driver is turned on. The output signal, A-FULL, is active under the following condition: $28 \leq SP \leq 63$.

Counter

The counter may be used as a loop counter. It may be loaded from the D-bus, the A-bus, or via a pop from the stack. Its contents may also be pushed onto the stack.

A normal for-loop is set up by a FOR instruction, which loads the counter from the D- or A-bus with the desired number of iterations; the instruction also pushes onto the stack a loop address that points to the next sequential instruction. The end of the loop is given by an unconditional END FOR instruction, which tests the counter value against the value one and then decrements the counter. If the values differ, the loop is repeated by selecting the address at the stack as the next address. If the values are equal, the loop is terminated by popping the stack, thereby removing the loop address, and selecting the address from the incrementer as the next address. The number of iterations is a 16-bit unsigned number, except that the number zero corresponds to 65,536 iterations.

By pushing and popping counter values it is possible to handle nested loops.

Address Comparison

The sequencer is able to compare the address from the interrupt multiplexer with the contents of the comparator register. The instruction SET loads the comparator register with the address on the D-bus and enables the comparison, while CLEAR disables it. The comparison is disabled at reset. A HIGH is present at the output EQUAL if the comparison is enabled and the two addresses are equal. The comparison is useful for detection of a break point or counting the number of times a microinstruction at a specific address is executed.

Instruction Set

The sequencer has 64 instructions that are divided into four classes of 16 instructions each. The instruction lines $I_0 - I_5$ use I_5 and I_4 to select a class, and $I_0 - I_3$ to select an instruction within a class. The classes are:

I_5	I_4	Classes
0	0	Conditional sequence control,
0	1	Conditional sequence control with inverted polarity,
1	0	Unconditional sequence control, and
1	1	Special function with implicit continue.

Note that for the first three classes I_5 forces the condition to be true and I_4 inverts the condition. The basic instructions of the first three classes are shown in Table 1 and the instructions of the fourth class in Table 2.

Structured microprogramming is supported by sequencer instructions that singly or in pairs correspond to high-level language control constructs. Examples are FOR $I := D$ DOWN TO 1 DO . . . END FOR and CASE N OF . . . END CASE. The instructions have been given high-level language names where appropriate. Figure 2 shows how to microprogram important control constructs; the high-level language is on the left and the microcode on the right.

Test Conditions

The condition for a conditional instruction is supplied by a test multiplexer, which selects one out of sixteen tests with the select lines $S_0 - S_3$. Twelve of these are supplied directly by the inputs $T_0 - T_{11}$, while the remaining four tests are generated by the test logic from the inputs $T_8 - T_{11}$. The following table shows the assignments.

$(S_0 - S_3)_H$ Test	Intended Use
0 - 7	$T_0 - T_7$ General
8	T_8 C (Carry)
9	T_9 N (Negative)
A	T_{10} V (Overflow)
B	T_{11} Z (Zero or equal)
C	$T_8 + T_{11}$ C + Z (Unsigned less than or equal, borrow mode)
D	$\overline{T_8} + T_{11}$ $\overline{C} + Z$ (Unsigned less than or equal)
E	$T_9 \oplus T_{10}$ N \oplus V (Signed less than)
F	$(T_9 \oplus T_{10}) + T_{11}$ (N \oplus V) + Z (Signed less than or equal)

Force Continue

The sequencer has a force continue (FC) input, which overrides the instruction inputs $I_0 - I_5$ with a CONTINUE instruction. This makes it possible to share the microinstruction field for the sequencer instruction with some other control or to initialize a writable control store.

Reset

In order to start a microprogram properly, the sequencer must be reset. The reset works like an instruction overriding both the instruction input and the force continue input. The reset selects the address 0 at the address multiplexer, forces the EQUAL output to LOW, and disregards a potential interrupt request. It synchronously disables the address comparison and initializes the stack pointer to 0. The contents of the stack are invalid after a reset.

TABLE 1. INSTRUCTION SET for I₅I₄ = 00, 01, 10

I ₅ - I ₀	Instruction	Cond.: Fail		Cond.: Pass		Counter	Comp.	D-Mux
		Y	Stack	Y	Stack			
00, 10, 20	Goto D	INC	-	D	-	-	-	SP
01, 11, 21	Call D	INC	-	D	Push INC	-	-	SP
02, 12, 22	Exit D	INC	-	D	Pop	-	-	SP
03, 13, 23	End for D, C ≠ 1	INC	-	D	-	C←C-1	-	SP
	End for D, C = 1	INC	-	INC	-	C←C-1	-	SP
04, 14, 24	Goto A	INC	-	A	-	-	-	SP
05, 15, 25	Call A	INC	-	A	Push INC	-	-	SP
06, 16, 26	Exit A	INC	-	A	Pop	-	-	SP
07, 17, 27	End for A, C ≠ 1	INC	-	A	-	C←C-1	-	SP
	End for A, C = 1	INC	-	INC	-	C←C-1	-	SP
08, 18, 28	Goto M	INC	-	D:M	-	-	-	SP
09, 19, 29	Call M	INC	-	D:M	Push INC	-	-	SP
0A, 1A, 2A	Exit M	INC	-	D:M	Pop	-	-	SP
0B, 1B, 2B	End for M, C ≠ 1	INC	-	D:M	-	C←C-1	-	SP
	End for M, C = 1	INC	-	INC	-	C←C-1	-	SP
0C, 1C, 2C	End Loop	INC	Pop	TOS	-	-	-	SP
0D, 1D, 2D	Call Coroutine	INC	-	TOS	Pop & Push INC	-	-	SP
					Pop	-	-	SP
0E, 1E, 2E	Return	INC	-	TOS	Pop	-	-	SP
0F, 1F, 2F	End for, C ≠ 1	INC	Pop	TOS	-	C←C-1	-	SP
	End for, C = 1	INC	Pop	INC	Pop	C←C-1	-	SP

Cond. = (Test [S] OR I₅) XOR I₄

: = Concatination

C = Counter

INC = Output of Incrementer = AR + 1 (if $\overline{C_{in}}$ = LOW)

Note: For unconditional instructions, the action marked under "Cond: Pass" is taken.

TABLE 2. INSTRUCTION SET for I₅I₄ = 11

I ₅ - I ₀	Instruction	Y	Stack	Counter	Comp.	D-Mux
30	Continue	INC	-	-	-	SP
31	For D	INC	Push INC	C←D	-	SP
32	Decrement	INC	-	C←C-1	-	SP
33	Loop	INC	Push INC	-	-	SP
34	Pop D	INC	Pop	-	-	TOS
35	Push D	INC	Push D	-	-	SP
36	Reset SP	INC	SP←0	-	-	SP
37	For A	INC	Push INC	C←A	-	SP
38	Pop C	INC	Pop	C←TOS	-	SP
39	Push C	INC	Push C	-	-	SP
3A	Swap	INC	TOS←C	C←TOS	-	SP
3B	Push C Load D	INC	Push C	C←D	-	SP
3C	Load D	INC	-	C←D	-	SP
3D	Load A	INC	-	C←A	-	SP
3E	Set	INC	-	-	R←D, Enable	SP
3F	Clear	INC	-	-	Disable	SP

R = Comp. Register

Interrupts

The sequencer may be interrupted at the completion of the current microcycle by asserting the interrupt request input INTR. The return address of the interrupted routine is saved on the stack so that nested interrupts can be easily implemented. An interrupt is accepted if interrupts are enabled and the sequencer is not being reset or held (INTEN = HIGH, RST = HIGH, and HOLD = LOW). The interrupt-acknowledge output (INTA) goes LOW when an interrupt is accepted.

When there is no interrupt, addresses go from the address multiplexer to the Y-bus via the driver, and to the address register and the comparator via the interrupt multiplexer. When there is an interrupt, the driver of the sequencer is turned off, an external driver is turned on, and the interrupt multiplexer is switched. The interrupt address is supplied via the external driver to the Y-bus, the address register, and the comparator (Figure 4). In order to save the address from the address multiplexer, the address is stored in the interrupt return address register, which for simplicity is clocked every cycle. The next microinstruction is the first microinstruction of the interrupt routine (Figure 5).

In this cycle the address in the interrupt return address register is automatically pushed onto the stack. Therefore the microinstruction in this cycle must not use the stack; if a stack operation is programmed, the result is undefined. The instructions that do not use the stack are GOTO D, GOTO A, GOTO M, CONTINUE, DECREMENT, LOAD D, LOAD A, SET and CLEAR. A RETURN instruction terminates the interrupt routine and the interrupted routine is resumed. Interrupts only work with a single-level control path.

Traps

A trap is an unexpected situation linked to current microinstruction that must be handled before the microinstruction completes and changes the state of the system. An example of such a situation is an attempt to read a word from memory across a word boundary in a single cycle. When a trap occurs, the current microinstruction must be aborted and re-executed after the execution of a trap routine, which in the meantime will take corrective measures. An interrupt, on the other hand, is not linked directly to the current microinstruction that can complete safely before an interrupt routine is executed.

Execution of a trap requires that the sequencer ignore the current microinstruction, select the trap return address at the address multiplexer, and initiate an interrupt. This will save the trap return address on the stack and issue the trap address from an external source (Figure 6). The address register

contains the address of the microinstruction in the pipeline register, thus the address register already contains the trap return address when a trap occurs. This address can be selected by the address multiplexer by disabling the incrementer ($CIN = 1$), and using the force continue mode ($FC = 1$). In this mode the sequencer ignores the current microinstruction. The remaining part of the trap handling is done by the interrupt (Figure 7), thus the section on interrupts also applies to traps. There is one exception, however. The interrupt enable cannot be used as a trap enable as it does not control the force continue mode and the carry-in to the incrementer.

Hold Mode

The sequencer has a hold mode in which the operation is suspended.

The outputs (Y, INTA, A-FULL & EQUAL) are disabled and the sequencer enters the hold mode immediately after the HOLD signal goes active. While the sequencer is in this mode, the internal state is left unchanged and the D-bus is disabled. The outputs (Y, INTA, A-FULL & EQUAL) are enabled again and the sequencer leaves the hold mode after the cycle immediately after the HOLD signal goes inactive.

In a time-multiplexed multi-microprocess system there may be one sequencer for all processes with microprogrammed context save and restore, or there may be one sequencer per microprocess permitting fast process switch. In the latter case the Y-buses of the sequencers are tied together and connected to a single microprogram store. A control unit decides on a cycle-by-cycle basis what sequencer should be running, and activates the HOLD signal to the remaining sequencers. The hold mode has higher priority than interrupts, and works independently of the reset. The hold mode can only be used with a single-level control path.

Master/Slave Configuration

In some systems reliability is very important. The master/slave configuration that consists of two sequencers operated in parallel is able to detect faults in both the interconnect and the internal function of the sequencers. One sequencer is the master and operates normally. The other is the slave, i.e., all outputs except the signal ERROR are turned into inputs and connected to the outputs of the master. Since the slave is operated in parallel with the master, it can compare its result with the result of the master and signal an error if they differ. The error signal from the master indicates a malfunctioning driver or contention. Because a TTL output goes HIGH when power is missing, the ERROR signal also indicates power failure.

High-Level Language Constructs

An example of high-level language constructs using Am29C331 instructions is given in Figure 3 (3-1, 3-2, 3-3, and 3-4).

```

REPEAT          LOOP
-              -
-              -
UNTIL CC        END LOOP NOT CC

WHILE CC DO     LOOP
-              IF NOT CC THEN EXIT L
-              -
END WHILE       END LOOP
                L:

LOOP           LOOP
-             -
IF CC THEN EXIT IF CC THEN EXIT L
-             -
END LOOP      END LOOP
                L:
    
```

Figure 3-1. Loops with Unknown Number of Iterations

```

FOR CNT: = 10 DOWN TO 1 DO  FOR D 10
-                             -
-                             -
END FOR                      END FOR
    
```

Figure 3-2. Loop with Known Number of Iterations

```

CASE I OF      PUSH D B
0: -           GOTO M
-             A: -
-             -, RETURN (TO B)
1: -           A + 2: -
-             -, RETURN (TO B)
2: -           A + 4: -
-             -, RETURN (TO B)
3: -           A + 6: -
-             -, RETURN
END CASE B:
    
```

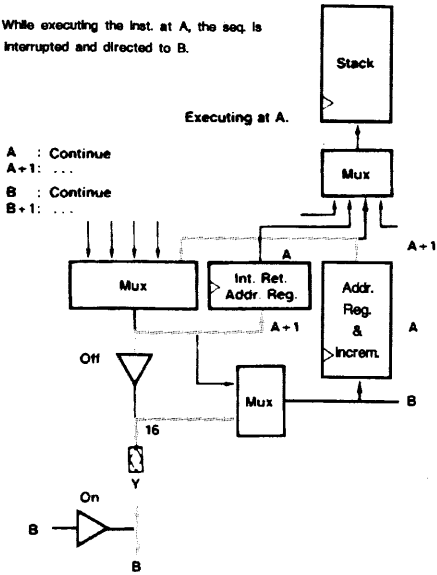
Figure 3-3. Case Statement
(with D = A₁₅ . . . A₄XX00 and M_{0, 0-3} = A₃₁l₀ during the GOTO M instruction. A₁A₀ must be 00, and X signifies a don't care.)

```

IF X THEN     PUSH D C
IF Y THEN     IF NOT X THEN GOTO A
-             IF NOT Y THEN GOTO B
-             -
-             -, RETURN (TO C)
ELSE          B:
-             -
-             -, RETURN (TO C)
END IF
ELSE          A:
IF Z THEN     IF NOT Z THEN GOTO D
-             -
-             -, RETURN (TO D)
ELSE          D:
-             -
-             -, RETURN (TO C)
END IF
END IF       C:
    
```

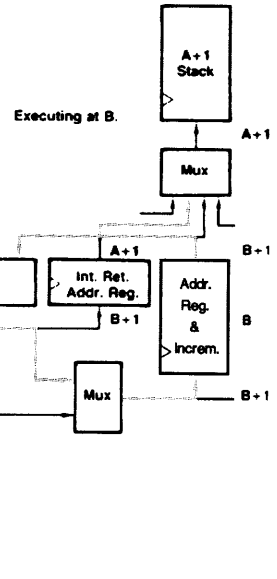
Figure 3-4. Double-Nested If Statement

While executing the inst. at A, the seq. is interrupted and directed to B.



AF004191

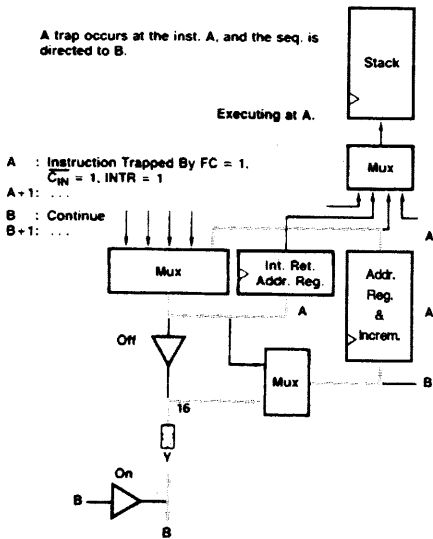
Figure 4. Am29C331 Interrupt Cycle 1



AF004211

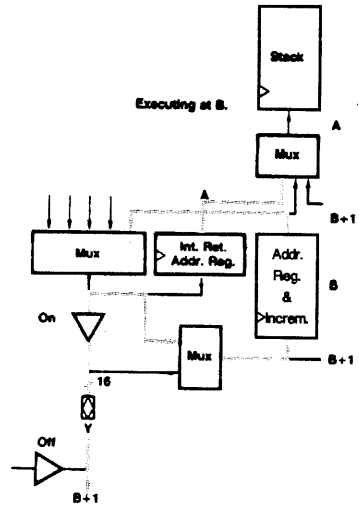
Figure 5. Am29C331 Interrupt Cycle 2

A trap occurs at the inst. A, and the seq. is directed to B.



AF004201

Figure 6. Am29C331 Traps Cycle 1



AF004181

Figure 7. Am29C331 Traps Cycle 2

Instruction Set Definition

Legend: ● = Other instruction

⊙ = Instruction being described

CC = (Test [S₃ - S₀])

P = Test pass

F = Test fail

○ = Register in part

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
20H	BRA_D	GOTO D Unconditional branch to the address specified by the D inputs. The D port must be disabled to avoid bus contention.	
24H	BRA_A	GOTO A Unconditional branch to the address specified by the A inputs.	
28H	BRA_M	GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) Unconditional branch to the address specified by the M inputs concatenated with the D input. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the four-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
2CH	BRA_S	GOTO TOS Unconditional branch to the address on the top of the stack.	
00H	BRCC_D	IF CC THEN GOTO D ELSE CONTINUE If CC is HIGH (pass), branch to the address specified by D. If CC is LOW (fail), continue. The D port must be disabled to avoid bus contention.	
04H	BRCC_A	IF CC THEN GOTO A ELSE CONTINUE If CC is HIGH (pass), branch to the address specified by A. If CC is LOW (fail), continue.	
08H	BRCC_M	IF CC THEN GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is HIGH (pass), branch to the address specified by D inputs concatenated with the M inputs. If CC is LOW (fail) continue. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
0CH	BRCC_S	IF CC THEN GOTO TOS ELSE POP STACK CONTINUE If CC is HIGH (pass), branch to the address on the top of the stack. If CC is LOW (fail), pop the stack and continue.	

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
10H	BRNC_D	IF NOT CC THEN GOTO D ELSE CONTINUE If CC is LOW (pass), branch to the address specified by D. If CC is HIGH (fail), continue. The D Port must be disabled to avoid Bus contention.	
14H	BRNC_A	IF NOT CC THEN GOTO A ELSE CONTINUE If CC is LOW (pass), branch to the address specified by A. If CC is HIGH (fail), continue.	
18H	BRNC_M	IF NOT CC THEN GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is LOW (pass), branch to the address specified by D inputs concatenated with the M inputs. If CC is HIGH (fail), continue. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
1CH	BRNC_S	IF NOT CC THEN GOTO TOS ELSE POP STACK CONTINUE If CC is LOW (pass), branch to the address on the top of the stack. If CC is HIGH (fail), pop the stack and continue.	

21H	CALL_D	CALL D Unconditional branch to the subroutine specified by the D inputs. Push the return address (address Reg. + 1) on the stack. The D port must be disabled to avoid bus contention.	
25H	CALL_A	CALL A Unconditional branch to the subroutine specified by the A inputs. Push the return address (Address Reg. + 1) on the stack.	
29H	CALL_M	CALL Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) Unconditional branch to the subroutine specified by the D inputs concatenated with the multiway inputs. Push the return address (Address Reg. + 1) on the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
2DH	CALL_S	CALL TOS Unconditional branch to the subroutine specified by the address on the top of the stack. The stack is popped and the return address (Address Reg. + 1) is then pushed onto the stack.	
			PF001760

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
01H	CCC_D	<p>IF CC, THEN CALL D ELSE CONTINUE</p> <p>If CC is HIGH (pass), call the subroutine specified by the D inputs. Push the return address (Address Reg. + 1) on the stack. If CC is LOW (fail), continue. The D port must be disabled to avoid bus contention.</p>	
05H	CCC_A	<p>IF CC, THEN CALL A ELSE CONTINUE</p> <p>If CC is HIGH (pass), call the subroutine specified by the A inputs. Push the return address (Address Reg. + 1) on the stack. If CC is LOW (fail), continue.</p>	
09H	CCC_M	<p>IF CC, THEN CALL Multiway (D₁₅ - D₄ M_{X3} - M_{X0}) ELSE CONTINUE</p> <p>If CC is HIGH (pass), call the subroutine specified by the D inputs concatenated with the M inputs. Push the return address (Address Reg. + 1) on the stack. The lower four bits on the D bus (D₃ - D₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D₁ and D₀ while bits D₃ and D₂ are "don't cares."</p>	
0DH	CCC_S	<p>IF CC, THEN CALL TOS ELSE CONTINUE</p> <p>If CC is HIGH (pass), call the subroutine specified by the address on the top of the stack. The stack is popped and the return address (Address Reg. + 1) is pushed onto the stack. If CC is LOW (fail), continue.</p>	PF001770
11H	CNC_D	<p>IF NOT CC, THEN CALL D ELSE CONTINUE</p> <p>If CC is LOW (pass), call the subroutine specified by the D inputs. Push the return address (Address Reg. + 1) on the stack. If CC is HIGH (fail), continue. The D port must be disabled to avoid bus contention.</p>	
15H	CNC_A	<p>IF NOT CC, THEN CALL A ELSE CONTINUE</p> <p>If CC is LOW (pass), call the subroutine specified by the A inputs. Push the return address (Address Reg. + 1) on the stack. If CC is HIGH (fail), continue.</p>	
19H	CNC_M	<p>IF NOT CC, THEN CALL Multiway (D₁₅ - D₄ M_{X3} - M_{X0}) ELSE CONTINUE</p> <p>If CC is LOW (pass), call the subroutine specified by the D inputs concatenated with the M inputs. Push the return address (Address Reg. + 1) on the stack. The lower four bits on the D bus (D₃ - D₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D₁ and D₀ while bits D₃ and D₂ are "don't cares."</p>	
1DH	CNC_S	<p>IF NOT CC, THEN CALL TOS ELSE CONTINUE</p> <p>If CC is LOW (pass), call the subroutine specified by the address on the top of the stack. The stack is popped and the return address (Address Reg. + 1) is pushed onto the stack.</p>	PF001780

Note: Opcode numbers are in hexadecimal notation.

Opcode (I5 - I0)	Mnemonics	Description	Execution Example
22H	EXIT_D	EXIT TO D Unconditional branch to the address specified by the D inputs and pop the stack. The D port must be disabled to avoid bus contention.	
26H	EXIT_A	EXIT TO A Unconditional branch to the address specified by the A inputs and pop the stack.	
2AH	EXIT_M	EXIT TO Multiway (D15 - D4 Mx3 - Mx0) Unconditional branch to the address specified by the D inputs concatenated with the M inputs and pop the stack. The lower four bits on the D bus (D3 - D0) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D1 and D0 while D3 and D2 are "don't cares."	
2EH	EXIT_S	EXIT TO TOS Unconditional branch to the address on the top of the stack and pop the stack. Also used for unconditional returns.	

PF001790

02H	XTCC_D	IF CC, THEN EXIT TO D ELSE CONTINUE If CC is HIGH (pass), exit to the address specified by the D inputs and pop the stack. If CC is LOW (fail), continue with no pop. The D port must be disabled to avoid bus contention.	
06H	XTCC_A	IF CC, THEN EXIT TO A ELSE CONTINUE If CC is HIGH (pass), exit to the address specified by the A inputs and pop the stack. If CC is LOW (fail), continue with no pop.	
0AH	XTCC_M	IF CC, THEN EXIT TO Multiway (D15 - D4 Mx3 - Mx0) ELSE CONTINUE If CC is HIGH (pass), exit to the address specified by the D inputs concatenated with the M inputs and pop the stack. The lower four bits on the D bus (D3 - D0) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D1 and D0 while bits D3 and D2 are "don't cares."	
0EH	XTCC_S	IF CC, THEN EXIT TO TOS ELSE CONTINUE If CC is HIGH (pass), exit to the address on the top of the stack and pop the stack. If CC is LOW (fail), continue with no pop. Also used for conditional returns.	

PF001800

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
12H	XTNC_D	IF NOT CC, THEN EXIT TO D ELSE CONTINUE If CC is LOW (pass), exit to the address specified by the D inputs and pop the stack. If CC is HIGH (fail), continue with no pop. The D port must be disabled to avoid bus contention.	
16H	XTNC_A	IF NOT CC, THEN EXIT TO A ELSE CONTINUE If CC is LOW (pass), exit to the address specified by the A inputs and pop the stack. If CC is HIGH (fail), continue with no pop.	
1AH	XTNC_M	IF NOT CC, THEN EXIT TO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is LOW (pass), exit to the address specified by the D inputs concatenated with the M inputs and pop the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiply branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
1EH	XTNC_S	IF NOT CC, THEN EXIT TO TOS ELSE CONTINUE If CC is LOW (pass), exit to the address on the top of the stack and pop the stack. If CC is HIGH (fail), continue with no pop. Also used for conditional returns.	
23H	DJMP_D	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO D ELSE CNT: = CNT - 1 CONTINUE If the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs. If the counter is equal to one, then decrement the counter and continue. The D port must be disabled to avoid bus contention.	
27H	DJMP_A	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO A ELSE CNT: = CNT - 1 CONTINUE If the counter is not equal to one, decrement the counter and branch to the address specified by the A inputs. If the counter is equal to one, then decrement the counter and continue.	
2BH	DJMP_M	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CNT: = CNT - 1 CONTINUE If the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs concatenated with the M inputs. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
2FH	DJMP_S	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO TOS ELSE CNT: = CNT - 1 POP STACK CONTINUE If the counter is not equal to one, decrement the counter and branch to the address on the top of the stack. If the counter is equal to one, then decrement the counter, pop the stack and continue.	

PF001810

PF001820

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example	
03H	DJCC_D	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO D ELSE CNT: = CNT - 1 CONTINUE</p> <p>If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs. If CC is LOW (fail) or the counter is equal to one, then decrement the counter and continue. The D port must be disabled to avoid bus contention.</p>		
07H	DJCC_A	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO A ELSE CNT: = CNT - 1 CONTINUE</p> <p>If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the A inputs. If CC is LOW (fail) or the counter is equal to one, then decrement the counter and continue.</p>		
0BH	DJCC_M	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO Multiway (D₁₅ - D₄ M_{X3} - M_{X0}) ELSE CNT: = CNT - 1 CONTINUE</p> <p>If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs concatenated with the M inputs. The lower four bits on the D bus (D₃ - D₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D₁ and D₀ while bits D₃ and D₂ are "don't cares."</p>		PF001830
0FH	DJCC_S	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO TOS ELSE CNT: = CNT - 1 POP STACK CONTINUE</p> <p>If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address on the top of the stack. If CC is LOW (fail) or the counter is equal to one, then decrement the counter, pop the stack and continue.</p>		

Note: Opcode numbers are in hexadecimal notation.

Opcode (15-10)	Mnemonics	Description	Execution Example	
13H	DJNCC_D	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO D ELSE CNT: = CNT - 1 CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs. If CC is HIGH (fail) or the counter is equal to one, then decrement the counter and continue. The D port must be disabled to avoid bus contention.</p>		
17H	DJNCC_A	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO A ELSE CNT: = CNT - 1 CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the A inputs. The content of the interrupt return address register and the address register is replaced by the A address in this case. If CC is HIGH (fail) or the counter is equal to one, the current address is incremented, appears on the bus for continue, and is stored into the above two registers.</p>		
1BH	DJNCC_M	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO Multiway (D₁₅ - D₄ M₃ - M₀) ELSE CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs concatenated with the M inputs. The lower four bits on the D bus (D₃ - D₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D₁ and D₀ while bits D₃ and D₂ are "don't cares."</p>		PF001840
1FH	DJNCC_S	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO TOS ELSE CNT: = CNT - 1 POP STACK CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address on the top of the stack. If CC is HIGH (fail) or the counter is equal to one, then decrement the counter, pop the stack and continue.</p>		
2EH	RET	<p>RETURN</p> <p>Unconditional return from subroutine. The return address is popped from the stack.</p>		
0EH	RETCC	<p>IF CC THEN RETURN ELSE CONTINUE</p> <p>If CC is HIGH (pass), return from subroutine. The return address is popped from the stack. If CC is LOW (fail), continue.</p>		
1EH	RETNC	<p>IF NOT CC THEN RETURN ELSE CONTINUE</p> <p>If CC is LOW (pass), return from subroutine. The return address is popped from the stack. If CC is HIGH (fail), continue.</p>		

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
31H	FOR_D	INITIALIZE LOOP Push the Address Reg. + 1 on the stack, load the counter from the D inputs and continue. Use with DJUMP_S for FOR...NEXT loops. The D port must be disabled to avoid bus contention.	
37H	FOR_A	INITIALIZE LOOP Push the Address Reg. + 1 on the stack, load the counter from the A inputs and continue. Use with DJUMP_S for FOR...NEXT loops.	
33H	LOOP	INITIALIZE LOOP Push the Address Reg. + 1 on the stack and continue. Use with BRCC_S for REPEAT...UNTIL loops, or with XTCC_D and BRA_S for WHILE...END WHILE loops.	

PF001860

34H	POP_D	Pop the stack and output the value on the D outputs and continue. The D port must be enabled.	
38H	POP_C	Pop the stack and store the value in the counter and continue.	
35H	PUSH_D	Push the D inputs on the stack and continue. The D port must be disabled to avoid bus contention.	
39H	PUSH_C	Push the counter on the stack and continue.	
3AH	SWAP	Exchange the counter and the top of stack and continue.	

PF001870

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
3BH	STACK_C	Push the counter on the stack and load the counter with the value of the D inputs and continue.	
3CH	LOAD_D	Load the counter with the value of the D inputs and continue. The D port must be disabled to avoid bus contention.	
3DH	LOAD_A	Load the counter with the value of the A inputs and continue.	PF001880
30H	CONT	Continue.	
32H	DECR	Decrement the counter and continue.	PF001890
36H	RESET_SP	Reset the stack pointer and continue.	
3EH	SET	Load the comparison register with the value of the D inputs, enable the comparator and continue.	PF001900
3FH	CLEAR	Disable the comparator and continue.	

Note: Opcode numbers are in hexadecimal notation.

APPLICATIONS

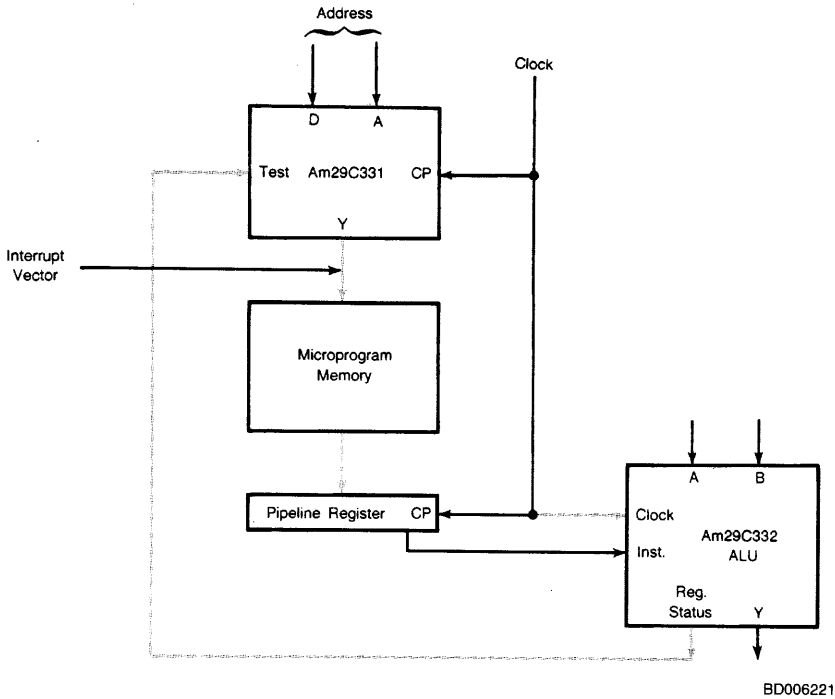
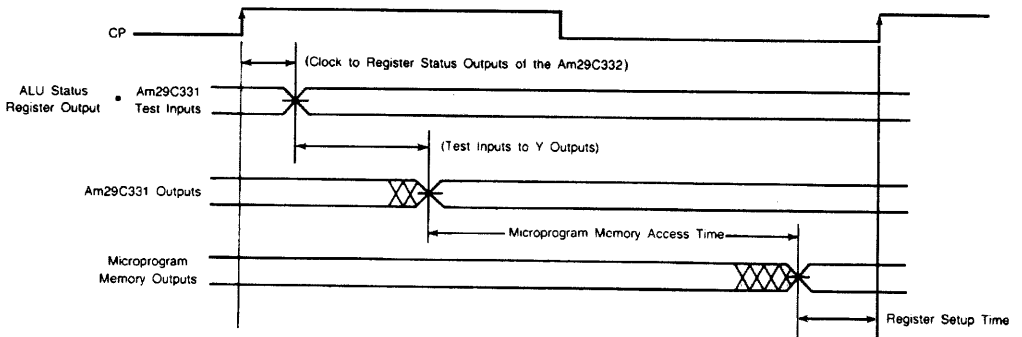


Figure 8. Typical Control-Path Architecture For Am29C300 Family



WF021093

Figure 9. Cycle Timing Waveform*

*This waveform shows the timing relationship for the configuration shown in Figure 8.

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 (Case) Temperature Under Bias -55 to +125°C
 Supply Voltage to
 Ground Potential Continuous -0.3 V to +7.0 V
 DC Voltage Applied to Outputs For
 High Output State -0.3 V to +V_{CC} +0.3 V
 DC Input Voltage -0.3 V to +V_{CC} +0.3 V
 DC Output Current, Into LOW Outputs 30 mA
 DC Input Current -10 mA to +10 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices
 Temperature (T_A) 0 to +70°C
 Supply Voltage (V_{CC}) +4.75 V to +5.25 V
 Military* (M) Devices
 Temperature (T_A) -55 to +125°C
 Supply Voltage (V_{CC}) +4.5 V to +5.5 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military Product 100% tested at T_A = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified (for APL Products, Group A, Subgroups 1, 2, 3 are tested unless otherwise noted)

Parameter Symbol	Parameter Description	Test Conditions (Note 1)		Min.	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min. V _{IN} = V _{IH} or V _{IL}	I _{OH} = 0.4 mA	2.4		Volts
V _{OL}	Output LOW Voltage	V _{CC} = Min. V _{IN} = V _{IH} or V _{IL}	I _{OL} = 8 mA for Y-BUS = 4 mA for All Other Pins		0.5	Volts
V _{IH}	Guaranteed Input Logical HIGH Voltage (Note 2)			2.0		Volts
V _{IL}	Guaranteed Input Logical LOW Voltage (Note 2)				0.8	Volts
I _{IL}	Input LOW Current	V _{CC} = Max. V _{IN} = 0.5 Volts			-10	μA
I _{IH}	Input HIGH Current	V _{CC} = Max. V _{IN} = V _{CC} - 0.5 V			10	μA
I _{OZH}	Off-State (HIGH Impedance) Output Current	V _{CC} = Max. V _O = 2.4 Volts			10	μA
I _{OZL}	Off-State (HIGH Impedance) Output Current	V _{CC} = Max. V _O = 0.5 Volts			-10	μA
I _{CC}	Static Power Supply Current (Note 3)	V _{CC} = Max., V _{IN} = V _{CC} or GND, I _O = 0 μA	COM'L	29C331	40	mA
				29C331-1/-2	50	
			MIL	29C331 only	50	
C _{PD}	Power Dissipation Capacitance (Note 4)	V _{CC} = 5.0 V T _A = 25°C No Load		pF Typical		

- Notes: 1. V_{CC} conditions shown as Min. or Max. refer to the commercial and military V_{CC} limits.
 2. These input levels provide zero-noise immunity and should only be statically tested in a noise-free environment (not functionally tested).
 3. Worst-case I_{CC} is measured at the lowest temperature in the specified operating range.
 4. C_{PD} determines the no-load dynamic current consumption.
 I_{CC} (Total) = I_{CC} (Static) + C_{PD} V_{CC} f, where f is the switching frequency of the majority of the internal nodes, normally one-half of the clock frequency. This specification is not tested.

SWITCHING CHARACTERISTICS over **COMMERCIAL** operating range

A. COMBINATIONAL PROPAGATION DELAYS

No.	From	To	29C331	29C331-1	29C331-2	Unit
			Max. Delay	Max. Delay	Max. Delay	
1	D15-0	Y15-0	22*	20*	18	ns
	D15-0	EQUAL	32	28	23	ns
	D15-0	ERROR	36	32	26	ns
2	A15-0	Y15-0	20	18	16	ns
	A15-0	EQUAL	31	27	22	ns
	A15-0	ERROR	33	29	24	ns
3	Mx3-X0	Y15-0	19	18	16	ns
	Mx3-X0	EQUAL	29	26	21	ns
	Mx3-X0	ERROR	33	29	24	ns
	Y15-0	EQUAL	31	28	23	ns
	Y15-0	ERROR	26	23	19	ns
4	I5-0	Y31-0	24	22	18	ns
	I5-0	D15-0	29	26	21	ns
	I5-0	EQUAL	36	33	27	ns
6	I5-0	ERROR	40	35	28	ns
	T11-0	Y15-0	24	22	18	ns
	T11-0	EQUAL	35	32	26	ns
	T11-0	ERROR	37	33	27	ns
	S3-0	Y15-0	24	22	18	ns
7	S3-0	EQUAL	35	32	26	ns
	S3-0	ERROR	37	33	27	ns
	CP	Y15-0	28	25	20	ns
	CP	D15-0	27/Z	25/Z	20/Z	ns
	CP	A-FULL	27	24	20	ns
9	CP	EQUAL	36	32	26	ns
	CP	ERROR	50	45	36	ns
	RST	Y15-0	26/Z	24/Z	20/Z	ns
11	RST	D15-0	Z	Z	Z	ns
	RST	INTA	22	19	17	ns
	RST	EQUAL	35	31	25	ns
	RST	ERROR	38	34	28	ns
12	FC	Y15-0	24	22	18	ns
	FC	D15-0	28	25	20	ns
	FC	EQUAL	33	30	24	ns
	FC	ERROR	35	31	25	ns
14	INTR	Y15-0	Z	Z	Z	ns
	INTR	INTA	17	16	9	ns
	INTR	EQUAL	(Note 1)	(Note 1)	(Note 1)	ns
	INTR	ERROR	46	21	18	ns
	INTEN	Y15-0	Z	Z	Z	ns
15	INTEN	INTA	16	15	9	ns
	INTEN	EQUAL	(Note 1)	(Note 1)	(Note 1)	ns
	INTEN	ERROR	46	21	18	ns
	HOLD	Y15-0	Z	Z	Z	ns
	HOLD	INTA	Z	Z	Z	ns
	HOLD	A-FULL	Z	Z	Z	ns
	HOLD	EQUAL	34/Z	31/Z	17/Z	ns
	HOLD	ERROR	46	18	17	ns
	OED	D15-0	Z	17	Z	ns
	OED	ERROR	19	Z	17	ns
	INTA	ERROR	19**	17**	17	ns
16	A-FULL	ERROR	21**	20**	17	ns
	EQUAL	ERROR	19**	17**	17	ns
	C _{in}	Y15-0	24	21	18	ns
	C _{in}	EQUAL	36	33	20	ns
	C _{in}	ERROR	37	33	21	ns
	SLAVE	Y15-0	Z	Z	Z	ns
	SLAVE	D15-0	Z	Z	Z	ns
	SLAVE	INTA	Z	Z	Z	ns
SLAVE	A-FULL	Z	Z	Z	ns	
SLAVE	EQUAL	Z	Z	Z	ns	

Notes: See notes following Table D.

*This includes using D as select lines for multiway sets.
 **In the slave mode.

SWITCHING CHARACTERISTICS over **COMMERCIAL** operating range (Cont'd.)

B. OUTPUT DISABLE TIME

No.	From	To	Description	29C331	29C331-1	29C331-2	Unit
				Max. Value	Max. Value	Max. Value	
43	RST	Y15-0	Reset-to-Address Enable	29	25	25	ns
	RST	Y15-0	Reset-to-Address Disable	29	25	25	ns
44	INTR	Y15-0	INTR-to-Address Enable	24	21	21	ns
	INTR	Y15-0	INTR-to-Address Disable	24	21	21	ns
	INTEN	Y15-0	INTEN-to-Address Enable	24	21	21	ns
	INTEN	Y15-0	INTEN-to-Address Disable	24	21	21	ns
	HOLD	Y15-0	HOLD-to-Address Enable	23	20	20	ns
	HOLD	Y15-0	HOLD-to-Address Disable	23	20	20	ns
	SLAVE	Y15-0	SLAVE-to-Address Enable	24	21	21	ns
	SLAVE	Y15-0	SLAVE-to-Address Disable	24	21	21	ns
	OED	Y15-0	OED-to-Data Enable	26	22	22	ns
	OED	D15-0	OED-to-Data Disable	26	22	22	ns
	RST	D15-0	Reset-to-Data Enable	27	23	23	ns
	RST	D15-0	Reset-to-Data Disable	27	23	23	ns
	SLAVE	D15-0	SLAVE-to-Data Enable	26	22	22	ns
	SLAVE	D15-0	SLAVE-to-Data Disable	26	22	22	ns
	CP	D15-0	Clock-to-Data Enable	35	24	24	ns
	CP	D15-0	Clock-to-Data Disable	35	24	24	ns
	HOLD	INTA	HOLD-to-INTA Enable	22	19	19	ns
	HOLD	INTA	HOLD-to-INTA Disable	22	19	19	ns
	HOLD	A-FULL	HOLD-to-A-FULL Enable	21	18	18	ns
	HOLD	A-FULL	HOLD-to-A-FULL Disable	21	18	18	ns
	HOLD	EQUAL	HOLD-to-EQUAL Enable	21	18	18	ns
	HOLD	EQUAL	HOLD-to-EQUAL Disable	21	18	18	ns
	SLAVE	INTA	SLAVE-to-INTA Enable	22	19	19	ns
	SLAVE	INTA	SLAVE-to-INTA Disable	22	19	19	ns
	SLAVE	A-FULL	SLAVE-to-A-FULL Enable	22	19	19	ns
	SLAVE	A-FULL	SLAVE-to-A-FULL Disable	22	19	19	ns
	SLAVE	EQUAL	SLAVE-to-EQUAL Enable	22	19	19	ns
	SLAVE	EQUAL	SLAVE-to-EQUAL Disable	22	19	19	ns

Notes: See notes following Table D.

SWITCHING CHARACTERISTICS over **COMMERCIAL** over operating range (Cont'd.)

C. SETUP AND HOLD TIMES

No.	Parameter	For	With Respect To	29C331	29C331-1	29C331-2	Unit
				Max. Value	Max. Value	Max. Value	
17	Data Setup	D ₁₅₋₀	CP ↑	21	19	19	ns
18	Data Hold	D ₁₅₋₀	CP ↑	0	0	0	ns
19	Alternate Data Setup	A ₁₅₋₀	CP ↑	23	21	21	ns
20	Alternate Data Hold	A ₁₅₋₀	CP ↑	0	0	0	ns
21	Multiway Setup	M _{X3-X0}	CP ↑	23	21	21	ns
22	Multiway Hold	M _{X3-X0}	CP ↑	0	0	0	ns
23	Address Setup	Y ₁₅₋₀	CP ↑	18	17	17	ns
24	Address Hold	Y ₁₅₋₀	CP ↑	0	0	0	ns
25	Instruction Setup	I ₅₋₀	CP ↑	24	21	21	ns
26	Instruction Hold	I ₅₋₀	CP ↑	0	0	0	ns
27	Forced Continue Setup	FC	CP ↑	21	19	19	ns
28	Forced Continue Hold	FC	CP ↑	0	0	0	ns
29	Test Setup	T ₁₁₋₀	CP ↑	24	20	20	ns
30	Test Hold	T ₁₁₋₀	CP ↑	0	0	0	ns
31	Select Setup	S ₃₋₀	CP ↑	22	20	20	ns
32	Select Hold	S ₃₋₀	CP ↑	0	0	0	ns
33	Reset Setup	RST	CP ↑	22	20	20	ns
34	Reset Hold	RST	CP ↑	0	0	0	ns
35	Interrupt Request Setup	INTR	CP ↑	20	18	18	ns
36	Interrupt Request Hold	INTR	CP ↑	0	0	0	ns
37	Interrupt Enable Setup	INTEN	CP ↑	16	16	16	ns
38	Interrupt Enable Hold	INTEN	CP ↑	0	0	0	ns
39	Hold Mode Setup	HOLD	CP ↑	21	16	16	ns
40	Hold Mode Hold	HOLD	CP ↑	0	0	0	ns
41	Carry-In Setup	C _{in}	CP ↑	22	20	20	ns
42	Carry-In Hold	C _{in}	CP ↑	0	0	0	ns

D. MINIMUM CLOCK REQUIREMENT

No.	Description	29C331	29C331-1	29C331-2	Unit
		Max. Value	Max. Value	Max. Value	
53	Minimum Clock LOW Time	23	22	22	ns
54	Minimum Clock HIGH Time	19	16	16	ns

Notes: 1. (INTR, INTEN)-to-EQUAL is the sum of (INTR, INTEN)-to-Y disable time and Y-to-EQUAL delay time.

2. C_L = 50 pF; C_L = 5 pF for Disable Time only.

SWITCHING CHARACTERISTICS over **MILITARY** operating range (for APL Products, Group A, Subgroups 9, 10, 11 are tested unless otherwise noted)

A. COMBINATIONAL PROPAGATION DELAYS

No.	From	To	29C331	Unit
			Max. Delay	
1	D ₁₅₋₀	Y ₁₅₋₀	30*	ns
	D ₁₅₋₀	EQUAL	48	ns
	D ₁₅₋₀	ERROR	29**	ns
2	A ₁₅₋₀	Y ₁₅₋₀	27	ns
	A ₁₅₋₀	EQUAL	44	ns
	A ₁₅₋₀	ERROR	50	ns
3	M _{x3-X0}	Y ₁₅₋₀	30	ns
	M _{x3-X0}	EQUAL	48	ns
	M _{x3-X0}	ERROR	55	ns
	Y ₁₅₋₀	EQUAL	41	ns
4	Y ₁₅₋₀	ERROR	29**	ns
	I ₅₋₀	Y ₃₁₋₀	32	ns
5	I ₅₋₀	D ₁₅₋₀	37	ns
	I ₅₋₀	EQUAL	48	ns
	I ₅₋₀	ERROR	55	ns
6	T ₁₁₋₀	Y ₁₅₋₀	32	ns
	T ₁₁₋₀	EQUAL	48	ns
	T ₁₁₋₀	ERROR	55	ns
	S ₃₋₀	Y ₁₅₋₀	32	ns
	S ₃₋₀	EQUAL	46	ns
	S ₃₋₀	ERROR	55	ns
7	CP	Y ₁₅₋₀	37	ns
	CP	D ₁₅₋₀	37/Z	ns
9	CP	A-FULL	32	ns
	CP	EQUAL	54	ns
	CP	ERROR	60	ns
10	RST	Y ₁₅₋₀	32/Z	ns
	RST	D ₁₅₋₀	Z	ns
11	RST	INTA	22	ns
	RST	EQUAL	48	ns
	RST	ERROR	55	ns
12	FC	Y ₁₅₋₀	32	ns
	FC	D ₁₅₋₀	37	ns
	FC	EQUAL	48	ns
13	FC	ERROR	55	ns
	INTR	Y ₁₅₋₀	Z	ns
	INTR	INTA	21	ns
14	INTR	EQUAL	(Note 1)	ns
	INTR	ERROR	49	ns
	INTEN	Y ₁₅₋₀	Z	ns
	INTEN	INTA	21	ns
	INTEN	EQUAL	(Note 1)	ns
	INTEN	ERROR	49	ns
15	HOLD	Y ₁₅₋₀	Z	ns
	HOLD	INTA	Z	ns
	HOLD	A-FULL	21/Z	ns
	HOLD	EQUAL	43/Z	ns
	HOLD	ERROR	49	ns
	OED	D ₁₅₋₀	26	ns
	OED	ERROR	Z	ns
	INTA	ERROR	29**	ns
	A-FULL	ERROR	29**	ns
	EQUAL	ERROR	29**	ns
	C _{in}	Y ₁₅₋₀	32	ns
	C _{in}	EQUAL	48	ns
	C _{in}	ERROR	55	ns
16	SLAVE	Y ₁₅₋₀	Z	ns
	SLAVE	D ₁₅₋₀	Z	ns
	SLAVE	INTA	Z	ns
	SLAVE	A-FULL	Z	ns
	SLAVE	EQUAL	Z	ns

Notes: See notes following Table D.

*This includes using D as select lines for multiway sets.

**In the slave mode.

SWITCHING CHARACTERISTICS over **MILITARY** operating range (Cont'd.)

B. OUTPUT DISABLE TIME

No.	From	To	Description	29C331	Unit
				Max. Value	
43 44	RST	Y15-0	Reset-to-Address Enable	26	ns
	RST	Y15-0	Reset-to-Address Disable	26	ns
	INTR	Y15-0	INTR-to-Address Enable	26	ns
	INTR	Y15-0	INTR-to-Address Disable	26	ns
	INTEN	Y15-0	INTEN-to-Address Enable	26	ns
	INTEN	Y15-0	INTEN-to-Address Disable	26	ns
	HOLD	Y15-0	HOLD-to-Address Enable	26	ns
	HOLD	Y15-0	HOLD-to-Address Disable	26	ns
	SLAVE	Y15-0	SLAVE-to-Address Enable	26	ns
	SLAVE	Y15-0	SLAVE-to-Address Disable	26	ns
	OED	Y15-0	OED-to-Data Enable	26	ns
	OED	D15-0	OED-to-Data Disable	26	ns
	RST	D15-0	Reset-to-Data Enable	26	ns
	RST	D15-0	Reset-to-Data Disable	26	ns
	SLAVE	D15-0	SLAVE-to-Data Enable	26	ns
	SLAVE	D15-0	SLAVE-to-Data Disable	26	ns
	CP	D15-0	Clock-to-Data Enable	23	ns
	CP	D15-0	Clock-to-Data Disable	23	ns
	HOLD	INTA	HOLD-to-INTA Enable	21	ns
	HOLD	INTA	HOLD-to-INTA Disable	21	ns
	HOLD	A-FULL	HOLD-to-A-FULL Enable	21	ns
	HOLD	A-FULL	HOLD-to-A-FULL Disable	21	ns
	HOLD	EQUAL	HOLD-to-EQUAL Enable	21	ns
	HOLD	EQUAL	HOLD-to-EQUAL Disable	21	ns
	SLAVE	INTA	SLAVE-to-INTA Enable	21	ns
	SLAVE	INTA	SLAVE-to-INTA Disable	21	ns
	SLAVE	A-FULL	SLAVE-to-A-FULL Enable	21	ns
	SLAVE	A-FULL	SLAVE-to-A-FULL Disable	21	ns
	SLAVE	EQUAL	SLAVE-to-EQUAL Enable	21	ns
	SLAVE	EQUAL	SLAVE-to-EQUAL Disable	21	ns

Notes: See notes following Table D.

SWITCHING CHARACTERISTICS over **MILITARY** operating range (Cont'd.)

C. SETUP AND HOLD TIMES

No.	Parameter	For	With Respect To	29C331	Unit
				Max. Value	
17	Data Setup	D ₁₅₋₀	CP ↑	32	ns
18	Data Hold	D ₁₅₋₀	CP ↑	1	ns
19	Alternate Data Setup	A ₁₅₋₀	CP ↑	32	ns
20	Alternate Data Hold	A ₁₅₋₀	CP ↑	1	ns
21	Multiway Setup	M _{X3-X0}	CP ↑	32	ns
22	Multiway Hold	M _{X3-X0}	CP ↑	1	ns
23	Address Setup	Y ₁₅₋₀	CP ↑	27	ns
24	Address Hold	Y ₁₅₋₀	CP ↑	2	ns
25	Instruction Setup	I ₅₋₀	CP ↑	32	ns
26	Instruction Hold	I ₅₋₀	CP ↑	0	ns
27	Forced Continue Setup	FC	CP ↑	32	ns
28	Forced Continue Hold	FC	CP ↑	1	ns
29	Test Setup	T ₁₁₋₀	CP ↑	32	ns
30	Test Hold	T ₁₁₋₀	CP ↑	0	ns
31	Select Setup	S ₃₋₀	CP ↑	32	ns
32	Select Hold	S ₃₋₀	CP ↑	0	ns
33	Reset Setup	RST	CP ↑	32	ns
34	Reset Hold	RST	CP ↑	1	ns
35	Interrupt Request Setup	INTR	CP ↑	27	ns
36	Interrupt Request Hold	INTR	CP ↑	1	ns
37	Interrupt Enable Setup	INTEN	CP ↑	27	ns
38	Interrupt Enable Hold	INTEN	CP ↑	1	ns
39	Hold Mode Setup	HOLD	CP ↑	27	ns
40	Hold Mode Hold	HOLD	CP ↑	1	ns
41	Carry-In Setup	C _{in}	CP ↑	30	ns
42	Carry-In Hold	C _{in}	CP ↑	1	ns

D. MINIMUM CLOCK REQUIREMENTS

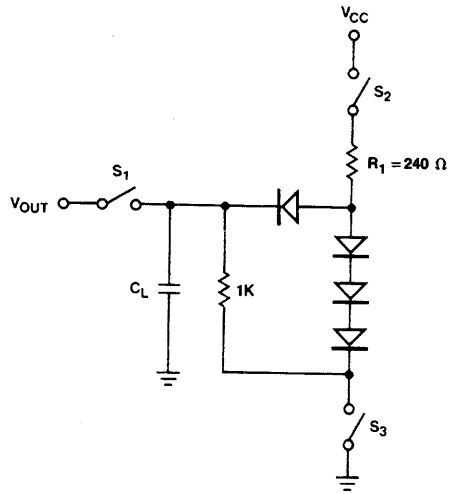
No.		29C331	Unit
		Max. Value	
53	Minimum Clock LOW Time	33	ns
54	Minimum Clock HIGH Time	28	ns

Notes: 1. (INTR, INTEN)-to-EQUAL is the sum of (INTR, INTEN)-to-Y disable time and Y-to-EQUAL delay time.

2. C_L = 50 pF; C_L = 5 pF for Disable Time only.

3. The status of I₅₋₀ and FC must not be changed during the clock LOW time.

SWITCHING TEST CIRCUIT

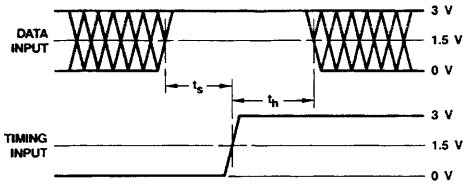


TC003420

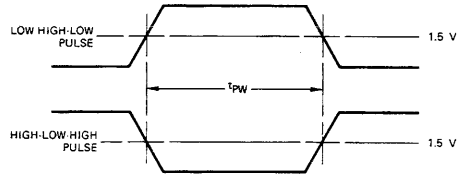
A. Three-State Outputs

- Notes:
1. $C_L = 50$ pF includes scope probe, wiring, and stray capacitances without device in test fixture.
 2. S_1 , S_2 , S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 S_1 and S_2 are closed while S_3 is open for t_{pZL} test.
 4. $C_L = 5.0$ pF for output disable tests.

SWITCHING TEST WAVEFORMS



WFR02970

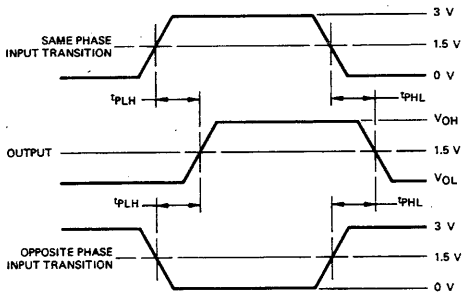


WFR02790

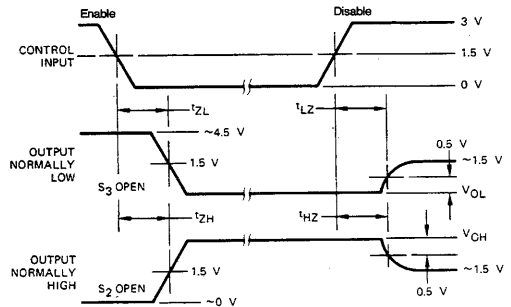
Pulse Width

- Notes: 1. Diagram shown for HIGH data only. Output transition may be opposite sense.
2. Cross hatched area is don't care condition.

Setup, Hold, and Release Times



WFR02980



WFR02663

Propagation Delay

- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
2. S_1 , S_2 , and S_3 of Load Circuit are closed except where shown.

Enable and Disable Times

Test Philosophy and Methods

The following points give the general philosophy that we apply to tests that must be properly engineered if they are to be implemented in an automatic environment. The specifics of what philosophies applied to which test are shown.

1. Ensure the part is adequately decoupled at the test head. Large changes in supply current when the device switches may cause function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an input transition, ground current may change by as much as 400 mA in 5–8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily. Current level may vary from product to product.
4. Use extreme care in defining input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins which may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \geq 3$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. Capacitive Loading for AC Testing

Automatic testers and their associated hardware have stray capacitance which varies from one type of tester to another, but is generally around 50 pF. This makes it impossible to make direct measurements of parameters which call for a smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays," which measure the propagation delays into and out of the high-impedance state, and are usually specified at a load capacitance of 5.0 pF. In these cases, the test is performed at the higher load capacitance (typically 50 pF), and engineering correlations based on data taken with a bench setup are used to predict the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impossible to make measurements at both capacitances even though they may both be greater than the stray capacitance. In

these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is predicted from engineering correlations based on data taken with a bench setup and the knowledge that certain DC measurements (I_{OH} , I_{OL} , for example) have already been taken and are within specification. In some cases, special DC tests are performed in order to facilitate this correlation.

7. Threshold Testing

The noise associated with automatic testing, the long inductive cables, and the high gain of bipolar devices when in the vicinity of the actual device threshold frequently give rise to oscillations when testing high-speed circuits. These oscillations are not indicative of a reject device, but instead, of an overtaxed test system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" high and low levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels rather than at V_{IL} max. and V_{IH} min.

8. AC Testing

Occasionally parameters are specified that cannot be measured directly on automatic testers because of tester limitations. Data input hold times often fall into this category. In these cases, the parameter in question is guaranteed by correlating these tests with other AC tests that have been performed. These correlations are arrived at by the cognizant engineer using data from precise bench measurements in conjunction with the knowledge that certain DC parameters have already been measured and are within specification.

In some cases, certain AC tests are redundant since they can be shown to be predicted by other tests that have already been performed. In these cases, the redundant tests are not performed.

9. Output Short-Circuit Current Testing

When performing I_{OS} tests on devices containing RAM or registers, great care must be taken that undershoot caused by grounding the high-state output does not trigger parasitic elements which in turn cause the device to change state. In order to avoid this effect, it is common to make the measurement at a voltage (V_{output}) that is slightly above ground. The V_{CC} is raised by the same amount so that the result (as confirmed by Ohm's law and precise bench testing) is identical to the $V_{OUT} = 0$, $V_{CC} = \text{Max.}$ case.

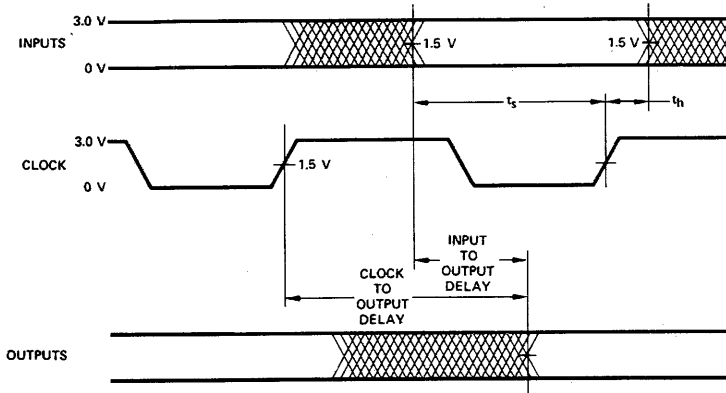
SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

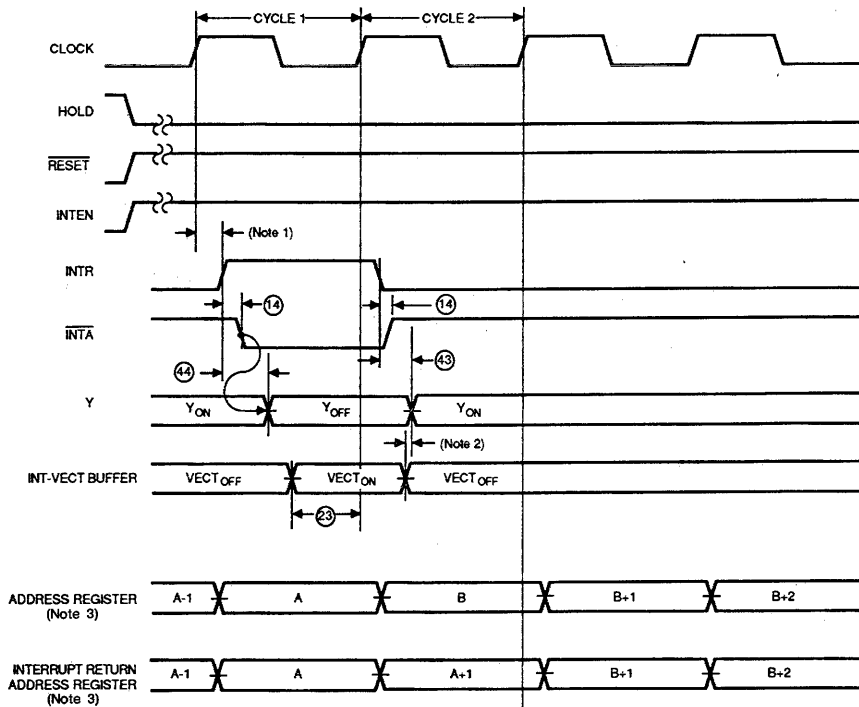
WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE, ANY CHANGE PERMITTED	CHANGING, STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE 'OFF' STATE

KS000010

SWITCHING WAVEFORMS (Cont'd.)



WFR02990

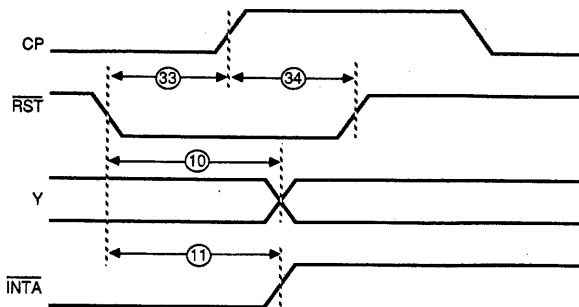


WF025100

Interrupt Timing

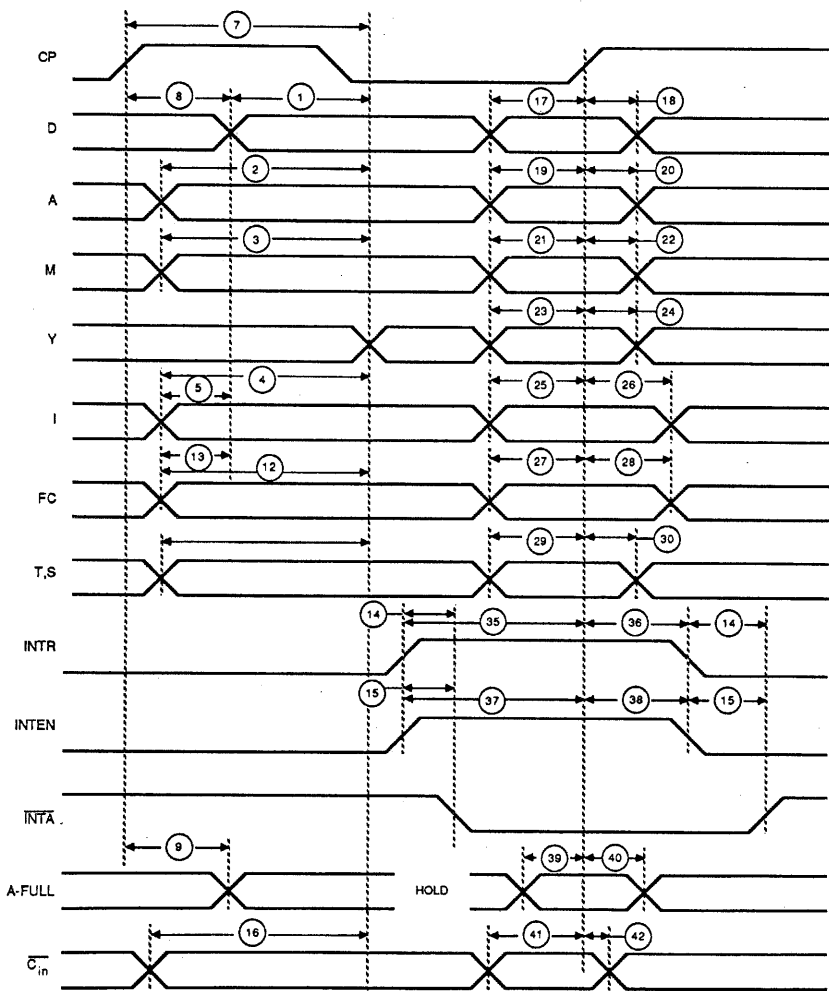
- Notes:
1. Interrupt Request comes from an interrupt-controller register. It reflects the CP \uparrow to INTR time of the interrupt controller.
 2. During Cycle 2, there may be contention on the Y-bus if the Y-bus is turned ON before the INT-VECT buffer is turned OFF.
 3. Refer to Figures 4 and 5 for definition of A and B.

SWITCHING WAVEFORMS (Cont'd.)



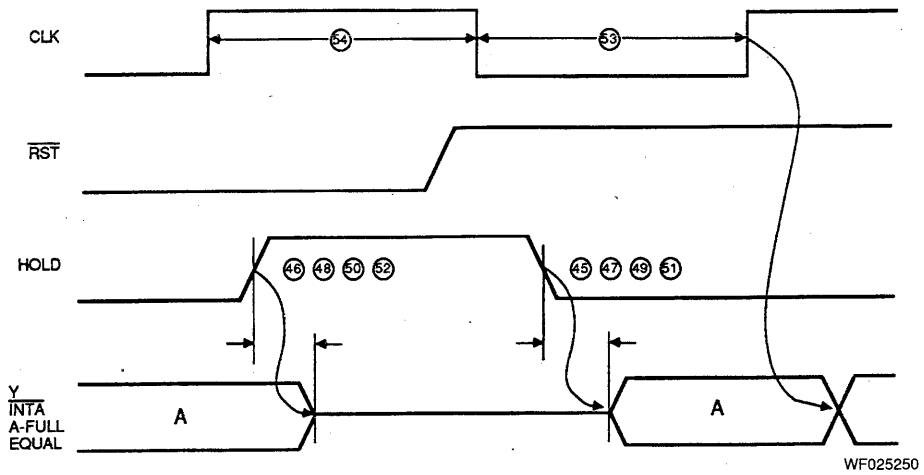
WF024770

Reset Timing



WF025320

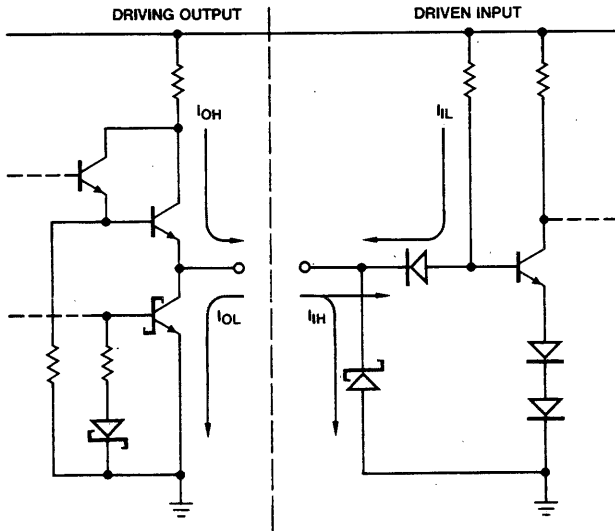
SWITCHING WAVEFORMS (Cont'd.)



Am29331 Hold Timing

INPUT/OUTPUT CIRCUIT DIAGRAM

(All Devices)



ICR00480

Am29C332

CMOS 32-Bit Arithmetic Logic Unit

ADVANCE INFORMATION

Am29C332

DISTINCTIVE CHARACTERISTICS

- **Single Chip, 32-Bit ALU**
Standard product supports 110 ns microcycle time for the 32-bit data path. It is a combinatorial ALU with equal cycle time for all instructions.
- **Speed Select supports 80-ns system cycle time**
- **Flow-through Architecture**
A combinatorial ALU with two input data ports and one output data port allows implementation of either parallel or pipelined architectures.
- **64-Bit In, 32-Bit Out Funnel Shifter**
This unique functional block allows n-bit shift-up, shift-down, 32-bit barrel shift or 32-bit field extract.
- **Supports All Data Types**
It supports one-, two-, three- and four-byte data for all operations and variable-length fields for logical operations.
- **Multiply and Divide Support**
Built-in hardware to support two-bit-at-a-time modified Booth's algorithm and one-bit-at-a-time division algorithm.
- **Extensive Error Checking**
Parity check and generate provides data transmission check and master/slave mode provides complete function checking.

GENERAL DESCRIPTION

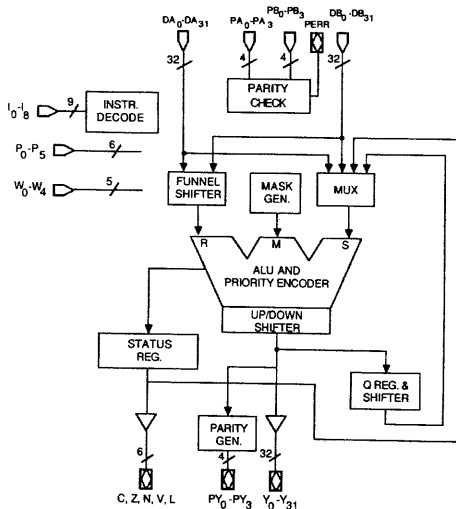
The Am29C332 is a 32-bit wide non-cascadable Arithmetic Logic Unit (ALU) with integration of functions that normally don't cascade, such as barrel shifters, priority encoders and mask generators. Two input data ports and one output data port provide flow-through architecture and allow the designer to implement his/her architecture with any degree of pipelining and no built-in penalties for branching. Also, the simplicity of a three-bus ALU allows easy implementation of parallel or reconfigurable architectures. The register file is off-chip to allow unlimited expansion and regular addressability.

multiprecision arithmetic and shift operations. For logical operations, it can support variable-length fields up to 32 bits. When fewer than four bytes are selected, unselected bits are passed to the destination without modification. The device also supports two-bit-at-a-time modified Booth's algorithm for high-speed multiplication and one-bit-at-a-time division. Both signed and unsigned integers for all byte aligned data types mentioned above are supported.

The Am29C332 supports one-, two-, three- and four-byte data for arithmetic and logic operations. It also supports

The Am29C332 is designed to support 110-ns microcycle time standard speed, and 80-ns microcycle time with speed select. The device is packaged in a 169-lead pin-grid-array package.

SIMPLIFIED BLOCK DIAGRAM



RELATED AMD PRODUCTS

Part No.	Description
Am29C01	CMOS 4-Bit Microprocessor Slice
Am29C10A	CMOS 12-Bit Sequencer
Am29C101	CMOS 16-Bit Microprocessor
Am29112	8-Bit Cascadable Microprogram Sequencer
Am29114	Real-Time Interrupt Controller
Am29C116	CMOS 16-Bit Microcontroller
Am29C323	CMOS 32 x 32 Parallel Multiplier
Am29325	32-Bit Floating Point Processor
Am29C325	CMOS 32-Bit Floating Point Processor
Am29331	16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29334	64 x 18 Four-Port, Dual-Access Register File
Am29C334	CMOS 64 x 18 Four-Port, Dual-Access Register File
Am29337	16-Bit Bounds Checker
Am29338	32-Bit Byte Queue
Am29C516	CMOS 16 x 16 Multiplier
Am29C517	CMOS 16 x 16 Multiplier with Separate I/O

CONNECTION DIAGRAM 169-Lead PGA Bottom View

	A	B	C	D	E	F	G	H	J	K	L	M	N	P	R	T	U
1	DB6	DA6	DB7	DB8	DB9	DB10	DA11	DB12	DA14	DB16	PA1	DB18	DB19	DB20	DB22	DA23	PA2
2	DA5	DA5	DA7	PB0	DA9	DB11	DA12	DA13	DB14	PA1	DA16	DA17	DA19	DA20	DA21	DB23	PB2
3	DB4	DA3	DA4	PA0	DA8	DA10	GND	DB13	DB15	DA15	VCC	DB17	DA18	DB21	DA22	DA24	DB24
4	DB2	DA2	DB3	*											DB25	DA25	DB26
5	DB1	DA1	DA0												DA26	DA27	DB27
6	DB0	P5	P4												DB28	DA28	DB29
7	P1	P3	VCC												VCC	DA30	DA29
8	P2	P0	W4												DB31	DA31	DB30
9	W2	W1	W3												MSERR	PA3	PB3
10	I2	W0	I0												Y31	Y30	Y28
11	I3	I1	GND												GND	Y27	Y29
12	I5	I4	I5												VCC	GND	Y26
13	I8	I7	CP												Y25	Y23	Y24
14	MLINK	RS	SLAVE												GND	VCC	Y22
16	M \bar{M}	MCh	N	C	PY3	PY2	GND	Y3	GND	Y7	VCC	Y8	Y12	CEY	Y19	Y21	Y20
18	BOROW	V	L	VCC	PY1	GNDT	GND	Y2	Y5	Y8	VCC	Y11	Y10	Y13	Y15	Y18	Y17
17	HOLD	Z	GND	PY0	Y0	PERR	GND	Y1	Y4	GND	VCC	Y9	VCC	GND	Y14	Y16	GND

CD010463

* This pin is not used

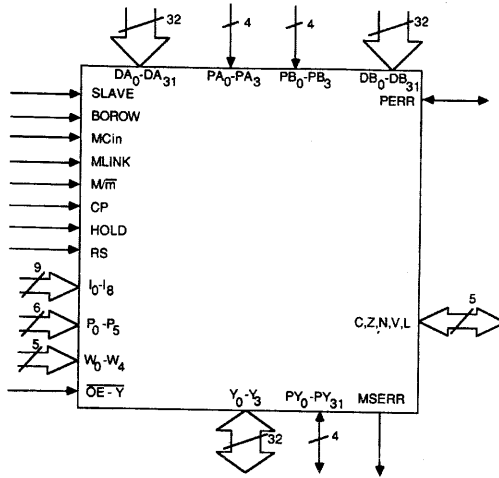
PIN DESIGNATIONS
(Sorted by Pin No.)

PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.
A-1	DB ₆	1	C-9	W ₃	145	J-15	GND	105	R-10	Y ₃₁	66
A-2	DA ₅	164	C-10	l ₀	139	J-16	Y ₅	101	R-11	GND	64
A-3	DB ₄	161	C-11	GND	143	J-17	Y ₄	102	R-12	V _{CC}	71
A-4	DB ₂	157	C-12	l ₅	134	K-1	DB ₁₆	27	R-13	Y ₂₅	74
A-5	DB ₁	155	C-13	CP	130	K-2	PA ₁	25	R-14	GND	79
A-6	DB ₀	153	C-14	SLAVE	127	K-3	DA ₁₅	24	R-15	Y ₁₉	82
A-7	P ₁	148	C-15	N	120	K-15	Y ₇	99	R-16	Y ₁₅	88
A-8	P ₂	149	C-16	L	118	K-16	Y ₆	100	R-17	Y ₁₄	89
A-9	W ₂	142	C-17	GND	117	K-17	GND	98	T-1	DA ₂₃	42
A-10	l ₂	137	D-1	DB ₈	7	L-1	PB ₁	26	T-2	DB ₂₃	41
A-11	l ₃	136	D-2	PB ₀	6	L-2	DA ₁₆	28	T-3	DA ₂₄	46
A-12	l ₆	133	D-3	PA ₀	5	L-3	V _{CC}	22	T-4	DA ₂₅	48
A-13	l ₈	131	D-15	C	119	L-15	V _{CC}	103	T-5	DA ₂₇	52
A-14	MLINK	129	D-16	V _{CC}	116	L-16	V _{CC}	103	T-6	DA ₂₈	54
A-15	M/ \bar{m}	125	D-17	PY ₀	115	L-17	V _{CC}	103	T-7	DA ₃₀	58
A-16	BOROW	124	E-1	DB ₉	9	M-1	DB ₁₈	31	T-8	DA ₃₁	60
A-17	HOLD	123	E-2	DA ₉	10	M-2	DA ₁₇	30	T-9	PA ₃	61
B-1	DA ₆	2	E-3	DA ₈	8	M-3	DB ₁₇	29	T-10	Y ₃₀	67
B-2	DB ₅	163	E-15	PY ₃	112	M-15	Y ₈	96	T-11	Y ₂₇	70
B-3	DA ₃	160	E-16	PY ₁	114	M-16	Y ₁₁	93	T-12	GND	72
B-4	DA ₂	158	E-17	Y ₀	109	M-17	Y ₉	95	T-13	Y ₂₃	76
B-5	DA ₁	156	F-1	DB ₁₀	11	N-1	DB ₁₉	33	T-14	V _{CC}	78
B-6	P ₅	152	F-2	DB ₁₁	13	N-2	DA ₁₉	34	T-15	Y ₂₁	80
B-7	P ₃	150	F-3	DA ₁₀	12	N-3	DA ₁₈	32	T-16	Y ₁₈	83
B-8	P ₀	147	F-15	PY ₂	113	N-15	Y ₁₂	92	T-17	Y ₁₆	86
B-9	W ₁	141	F-16	GND	110	N-16	Y ₁₀	94	U-1	PA ₂	43
B-10	W ₀	140	F-17	PERR	111	N-17	V _{CC}	97	U-2	PB ₂	44
B-11	l ₁	138	G-1	DA ₁₁	14	P-1	DB ₂₀	35	U-3	DB ₂₄	45
B-12	l ₄	135	G-2	DA ₁₂	16	P-2	DA ₂₀	36	U-4	DB ₂₆	49
B-13	l ₇	132	G-3	GND	21	P-3	DB ₂₁	37	U-5	DB ₂₇	51
B-14	RS	128	G-15	GND	104	P-15	OE- \bar{Y}	87	U-6	DB ₂₉	55
B-15	MCin	126	G-16	GND	104	P-16	Y ₁₃	90	U-7	DA ₂₉	56
B-16	V	121	G-17	GND	104	P-17	GND	91	U-8	DB ₃₀	57
B-17	Z	122	H-1	DB ₁₂	15	R-1	DB ₂₂	39	U-9	PB ₃	62
C-1	DB ₇	3	H-2	DA ₁₃	18	R-2	DA ₂₁	38	U-10	Y ₂₈	69
C-2	DA ₇	4	H-3	DB ₁₃	17	R-3	DA ₂₂	40	U-11	Y ₂₉	68
C-3	DA ₄	162	H-15	Y ₃	106	R-4	DB ₂₅	47	U-12	Y ₂₆	73
C-4	DB ₃	159	H-16	Y ₂	107	R-5	DA ₂₆	50	U-13	Y ₂₄	75
C-5	DA ₀	154	H-17	Y ₁	108	R-6	DB ₂₈	53	U-14	Y ₂₂	77
C-6	P ₄	151	J-1	DA ₁₄	20	R-7	V _{CC}	63	U-15	Y ₂₀	81
C-7	V _{CC}	144	J-2	DB ₁₄	19	R-8	DB ₃₁	59	U-16	Y ₁₇	84
C-8	W ₄	146	J-3	DB ₁₅	23	R-9	MSERR	65	U-17	GND	85

PIN DESIGNATIONS
(Sorted by Pin Names)

PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.
BOROW	A-16	124	DB ₇	C-1	3	I ₂	A-10	137	V _{CC}	T-14	78
C	D-15	119	DB ₈	D-1	7	I ₃	A-11	136	V _{CC}	N-17	97
CP	C-13	130	DB ₉	E-1	9	I ₄	B-12	135	V _{CC}	D-16	116
DA ₀	C-5	154	DB ₁₀	F-1	11	I ₅	C-12	134	V _{CC}	H-12	71
DA ₁	B-5	156	DB ₁₁	F-2	13	I ₆	A-12	133	W ₀	B-10	140
DA ₂	B-4	158	DB ₁₂	H-1	15	I ₇	B-13	132	W ₁	B-9	141
DA ₃	B-3	160	DB ₁₃	H-3	17	I ₈	A-13	131	W ₂	A-9	142
DA ₄	C-3	162	DB ₁₄	J-2	19	L	C-16	118	W ₃	C-9	145
DA ₅	A-2	164	DB ₁₅	J-3	23	MC _{in}	B-15	126	W ₄	C-8	146
DA ₆	B-1	2	DB ₁₆	K-1	27	MLINK	A-14	129	Y ₀	E-17	109
DA ₇	C-2	4	DB ₁₇	M-3	29	M/ \bar{m}	A-15	125	Y ₁	H-17	108
DA ₈	E-3	8	DB ₁₈	M-1	31	MSERR	R-9	65	Y ₂	H-16	107
DA ₉	E-2	10	DB ₁₉	N-1	33	N	C-15	120	Y ₃	H-15	106
DA ₁₀	F-3	12	DB ₂₀	P-1	35	OE- \bar{Y}	P-15	87	Y ₄	J-17	102
DA ₁₁	G-1	14	DB ₂₁	P-3	37	P ₀	B-8	147	Y ₅	J-16	101
DA ₁₂	G-2	16	DB ₂₂	R-1	39	P ₁	A-7	148	Y ₆	K-16	100
DA ₁₃	H-2	18	DB ₂₃	T-2	41	P ₂	A-8	149	Y ₇	K-15	99
DA ₁₄	J-1	20	DB ₂₄	U-3	45	P ₃	B-7	150	Y ₈	M-15	96
DA ₁₅	K-3	24	DB ₂₅	R-4	47	P ₄	C-6	151	Y ₉	M-17	95
DA ₁₆	L-2	28	DB ₂₆	U-4	49	P ₅	B-6	152	Y ₁₀	N-16	94
DA ₁₇	M-2	30	DB ₂₇	U-5	51	PA ₀	D-3	5	Y ₁₁	M-16	93
DA ₁₈	N-3	32	DB ₂₈	R-6	53	PA ₁	K-2	25	Y ₁₂	N-15	92
DA ₁₉	N-2	34	DB ₂₉	U-6	55	PA ₂	U-1	43	Y ₁₃	P-16	90
DA ₂₀	P-2	36	DB ₃₀	U-8	57	PA ₃	T-9	61	Y ₁₄	R-17	89
DA ₂₁	R-2	38	DB ₃₁	R-8	59	PB ₀	D-2	6	Y ₁₅	R-16	88
DA ₂₂	R-3	40	GND	G-3	21	PB ₁	L-1	26	Y ₁₆	T-17	86
DA ₂₃	T-1	42	GND	R-11	64	PB ₂	U-2	44	Y ₁₇	U-16	84
DA ₂₄	T-3	46	GND	G-17	104	PB ₃	U-9	62	Y ₁₈	T-16	83
DA ₂₅	T-4	48	GND	G-15	104	PERR	F-17	111	Y ₁₉	R-15	82
DA ₂₆	R-5	50	GND	G-16	104	PY ₀	D-17	115	Y ₂₀	U-15	81
DA ₂₇	T-5	52	GND	C-11	143	PY ₁	E-16	114	Y ₂₁	T-15	80
DA ₂₈	T-6	54	GND	T-12	72	PY ₂	F-15	113	Y ₂₂	U-14	77
DA ₂₉	U-7	56	GND	R-14	79	PY ₃	E-15	112	Y ₂₃	T-13	76
DA ₃₀	T-7	58	GND	U-17	85	RS	B-14	128	Y ₂₄	U-13	75
DA ₃₁	T-8	60	GND	P-17	91	SLAVE	C-14	127	Y ₂₅	R-13	74
DB ₀	A-6	153	GND	K-17	98	V	B-16	121	Y ₂₆	U-12	73
DB ₁	A-5	155	GND	J-15	105	V _{CC}	R-7	63	Y ₂₇	T-11	70
DB ₂	A-4	157	GND	F-16	110	V _{CC}	L-16	103	Y ₂₈	U-10	69
DB ₃	C-4	159	GND	C-17	117	V _{CC}	L-15	103	Y ₂₉	U-11	68
DB ₄	A-3	161	HOLD	A-17	123	V _{CC}	L-17	103	Y ₃₀	T-10	67
DB ₅	B-2	163	I ₀	C-10	139	V _{CC}	C-7	144	Y ₃₁	R-10	66
DB ₆	A-1	1	I ₁	B-11	138	V _{CC}	L-3	22	Z	B-17	122

LOGIC SYMBOL



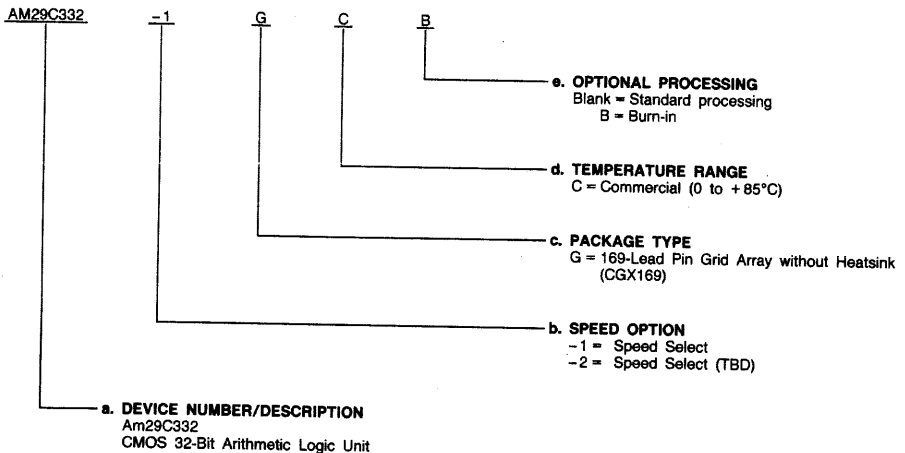
LS002911

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. Device Number
- b. Speed Option (if applicable)
- c. Package Type
- d. Temperature Range
- e. Optional Processing



Valid Combinations	
AM29C332	GC, GCB
AM29C332-1	

Valid Combinations

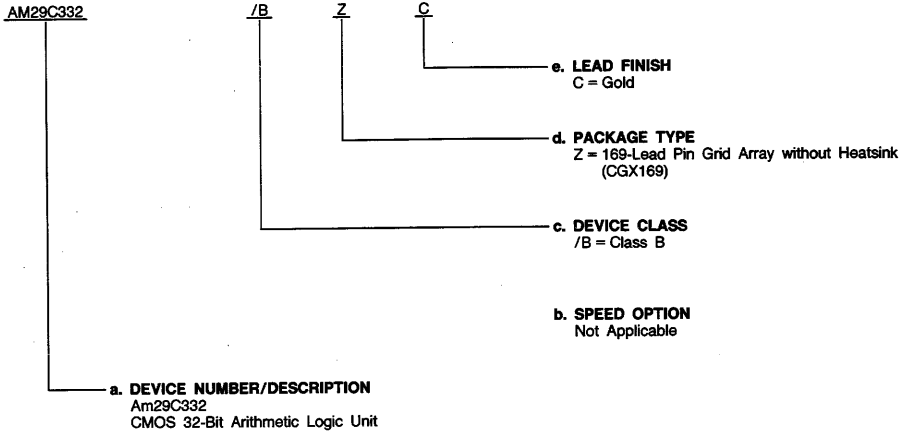
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL Products

AMD products for Aerospace and Defense applications are available in several packages and operating ranges. APL (Approved Products List) products are fully compliant with MIL-STD-883C requirements. The order number (Valid Combination) for APL products is formed by a combination of:

- a. Device Number
- b. Speed Option (if applicable)
- c. Device Class
- d. Package Type
- e. Lead Finish



Valid Combinations	
AM29C332	/BZC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations or to check for newly released valid combinations.

Group A Tests

Group A tests include Subgroups
1, 2, 3, 7, 8, 9, 10, 11.

PIN DESCRIPTION

BOROW Borrow (Input)

When HIGH, the Carry In and Carry Out are borrows for subtract operations.

C, Z, N, V, L Status (Input/Output)

When the Register Status pin is LOW, these pins give the Carry, Zero, Negative, Overflow and Link outputs of the ALU where applicable to the instruction being executed. When not applicable to the instruction being executed, or when the Register Status pin is HIGH, these pins give the outputs of the Carry, Zero, Negative, Overflow and Link bits of the internal Status Register. In Slave mode, C, Z, N, V and L become inputs.

CP Clock Input (Input)

Clocks internal registers (status, Q) at the LOW to HIGH transition, provided HOLD input is LOW.

DA₀ - DA₃₁ Data Input for DA-bus (Input)

Data input lines for operand A.

DB₀ - DB₃₁ Data Input for DB-bus (Input)

Data input lines for operand B.

HOLD Hold (Input, Active HIGH)

When HIGH, it inhibits the update of the status and Q registers.

I₀ - I₆ Instruction Inputs (Input)

Used to select the operation to be performed.

I₇ - I₈ Byte Width Inputs (Input)

Byte width inputs for byte boundary aligned operand instructions. Selects the sources for width and position inputs for variable field bit operands. If I₇ is LOW it selects the width input from pins W₄ - W₀. If I₇ is HIGH the width input is selected from the internal width register. Similarly if I₈ is LOW it selects the position inputs from pins P₅ - P₀ and if HIGH it selects input from the internal position register.

MCin Macro Status Carry (Input)

External Carry input.

MLINK Macro Status Link (Input)

External link input.

M/m Macro/Micro Select (Input)

When HIGH, selects macro carry and macro link pins as input instead of micro carry and micro link from the micro-status register.

MSERR Master-Slave Error (Output)

When HIGH, this signal indicates that the master's and slave's data were not identical.

OE-Y Output Enable (Input, Active LOW)

When OE-Y is HIGH the Y-bus is disabled (three-stated).

P₀ - P₅ Position Inputs (Input)

Position input to select the position of the least significant bit of a field. Also indicates the amount by which data is to be shifted up (P₅ = LOW) or down (P₅ = HIGH) or rotated.

PA₀ - PA₃ Parity Input for DA-bus (Input)

Parity input for operand A on DA-bus (one per byte). Even parity is used for the Am29C332.

PB₀ - PB₃ Parity Input for DB-bus (Input)

Parity input for operand B on DB-bus (one per byte).

PERR Parity Error (Input/Output)

When HIGH, indicates that a parity error was detected on the DA or DB inputs.

PY₀ - PY₃ Parity for Y-bus (Input/Output)

Parity output for data on Y-bus (one per byte). Even parity is used for the Am29C332. In slave mode, PY₀ - PY₃ become inputs.

RS Register Status Mode Pin (Input)

Selects between ALU status (Register Status = LOW) or register status (Register Status = HIGH) on the C, Z, N, V and L outputs.

SLAVE Slave (Input)

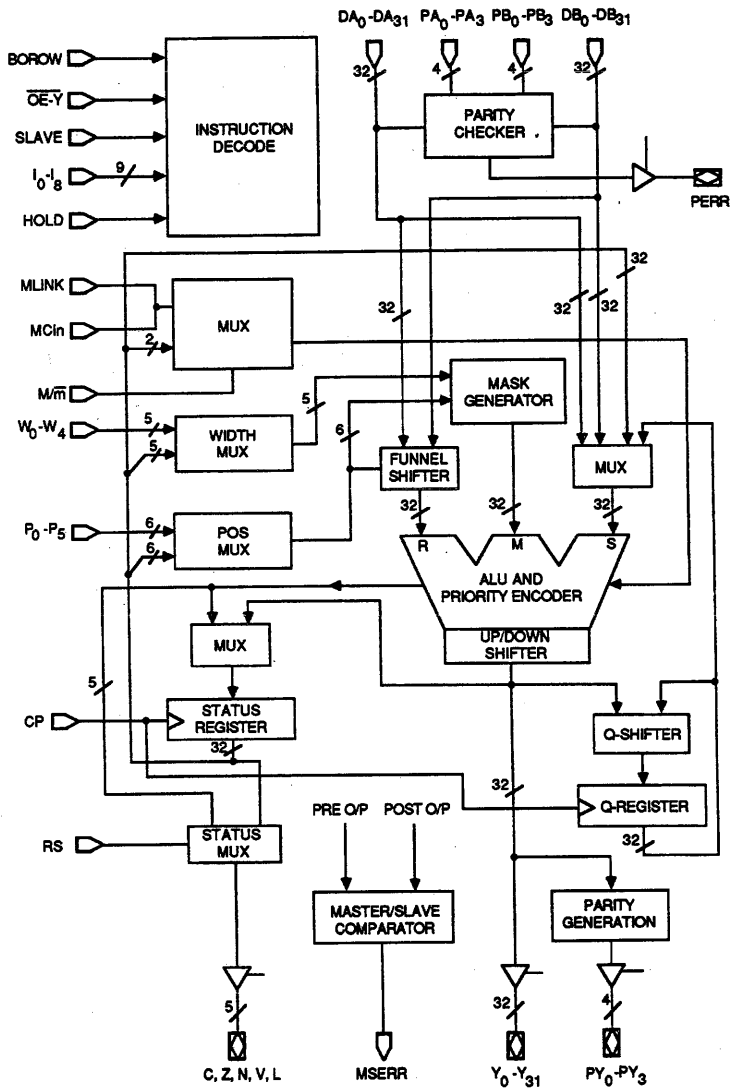
When HIGH, this pin puts the ALU in the slave mode. All output pins become input pins and signals on them are compared with the ALU's internally generated results. When OE-Y is HIGH, the Y₀ - Y₃₁ and PY₀ - PY₃ inputs are ignored. When the SLAVE pin is LOW, the ALU is put in master mode where outputs are generated as normal.

W₀ - W₄ Width Inputs (Input)

Width input to select the width of a contiguous bit field.

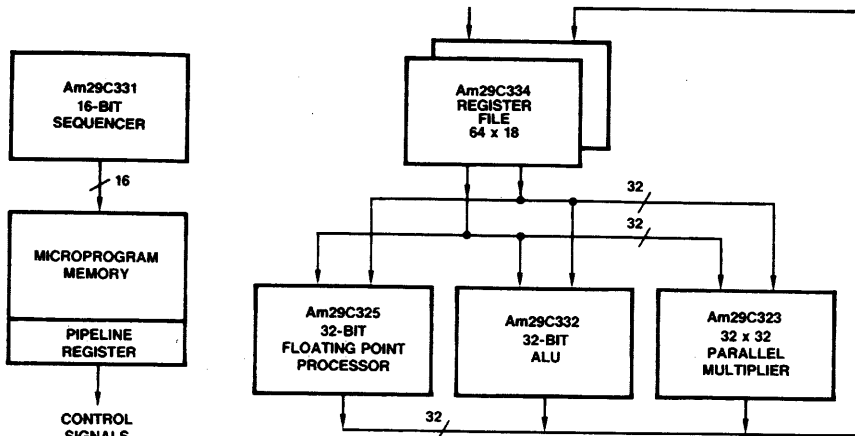
Y₀ - Y₃₁ Data Out/In Lines (Input/Output)

When OE-Y is LOW and the ALU is in the Master mode, the ALU result is enabled on the Y-bus. When OE-Y is HIGH, the Y-bus is three-stated. In Slave mode the Y-bus acts as external data input.



BD007031

Figure 1. Detailed Block Diagram



AF003484

Figure 2. Am29C332 Family High-Performance System Block Diagram

PRODUCT OVERVIEW

The Am29C332 is a 32-bit wide, high-performance, non-expandable Arithmetic Logic Unit (ALU). It has two 32-bit wide input ports (A and B) and one 32-bit wide output port (Y). These three ports provide flexibility and accessibility for high-performance processor designs. Dedicated input and output ports provide a flow-through architecture and avoid the penalty associated with switching the bus half-way through the cycle for input and output of data. The chip is designed for use with a dual-access RAM (Am29C334) as a register file. In addition, the three-bus architecture facilitates the connection of other arithmetic units in parallel with the Am29C332 for high-performance systems.

The Am29C332 supports one-, two-, three-, and four-byte arithmetic operations. It also supports multiprecision arithmetic and multiple bit shifts. For logical operations, it can handle variable-length fields of up to 32 bits. The chip incorporates dedicated hardware to allow efficient implementation of a two bit-at-a-time (modified Booth) multiply algorithm, supporting signed and unsigned arithmetic data types. Similarly, hardware is provided to support a bit-at-a-time divide algorithm, also supporting signed and unsigned arithmetic data types. An internal 32-bit register (Q) is used by the multiply and divide hardware for double precision operands. For business applications, the Am29C332 supports variable-length BCD arithmetic.

Field logical instructions operate on bit-fields taken from the A and B data inputs; they may be of variable width and starting position. A is normally the source input and B the destination input. In general, destination bits not falling within a specified field are passed by the ALU unchanged. Field width and position are specified either by direct inputs to the chip, or by entries in the status register. There are two kinds of field logical instructions - aligned and non-aligned. The first type of instruction assumes that source and destination fields are aligned and the operation is performed only for bits within the specified fields. In the second type of instruction, source and destination fields are normally non-aligned. However, it is always assumed that one field (either source or destination) is least-significant-bit (LSB) aligned.

If the destination field is LSB aligned then the source field is downshifted in order to make it LSB aligned as well. Down-

shifting is accomplished by making the 6-bit position input equal to the two's complement of the number of places the field is to be downshifted. If the source field is LSB aligned then it is upshifted in order to align it with the destination. Upshifting is accomplished by making the position inputs equal to the number of places the field is to be upshifted. Any other type of field operation is not allowed. Whenever the field crosses the word boundary, the portion not falling within the word boundary is ignored. This effect is useful when performing operations on fields that overlap two different words. Instructions to perform straightforward multiple-bit shifts (either up or down) are also provided. Additionally, it is possible to extract a bit-field from a word in one instruction, even if that field overlaps a word boundary.

The power and the flexibility of the processor comes partly from its ability to generate a mask to control the width of an operation for each instruction without any overhead. For all byte aligned instructions (three quarters of the instruction set), the mask is either 1, 2, 3 or 4 bytes wide and is generated from the byte width input ($I_8 - I_7$). For all field instructions the mask is of variable width and is generated from the position inputs ($P_0 - P_5$) and the width inputs ($W_0 - W_4$). Table 1 describes the position displacement from the position inputs and Table 2 the bit field from the width inputs.

TABLE 1. POSITION INPUTS AND BIT DISPLACEMENT

Inputs						Bit Displacement p
P ₅	P ₄	P ₃	P ₂	P ₁	P ₀	
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	2
:	:	:	:	:	:	:
0	1	1	1	1	1	31
1	0	0	0	0	0	-32
1	0	0	0	0	1	-31
:	:	:	:	:	:	:
1	1	1	1	1	1	-1

TABLE 2. WIDTH INPUTS AND BIT FIELD

Inputs					Bit Field w
W ₄	W ₃	W ₂	W ₁	W ₀	
0	0	0	0	0	32
0	0	0	0	1	1
0	0	0	1	0	2
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	31

Whenever the width of the operand is less than 32-bits, all unselected bits from the inputs of the ALU are passed to the output without any modification. Depending upon the instruction type, unselected bits are taken from different sources. For example in all single operand instructions, bits from the source operand (from either A or B input) are passed in unselected bit positions. For two operand instructions, bits from the B input are passed in unselected bit positions. There are some exceptions which are explained in the instruction set section.

The processor has a 32-bit status register to indicate the status of different operations performed. The status register is loaded at the rising edge of the clock with new status unless the HOLD signal is HIGH. The bit position for each status bit is given in the functional description. The least significant byte of the status register holds the six position bits (PR₀ – PR₅). The two most significant bits of this byte may be read or loaded but are otherwise unused by the ALU. The second byte (bits 8 to 15) consists of the five width bits (WR₀ – WR₄) and three read-only bits that are a combinational function of other status bits, and which indicate useful branch conditions. The third byte consists of ALU status bits plus bits for high-speed multiply and divide. The most significant byte holds intermediate nibble carries for BCD operations. An extract-status instruction is provided which allows a Boolean value to be formed from any selected bit. This is particularly useful in machines employing a stack architecture. Instructions to save and restore the status register are provided. As the entire status of each instruction is stored in the status register, interrupts at any microinstruction boundary are feasible.

The processor has a 32-bit wide priority encoder to support floating-point and graphics operations. The priority encoder supports all byte aligned data types – the result is dependent upon the byte width specified. The result of a priority encode is also loaded into the position bits of the status register. The result of the prioritize operation can then be used in the following clock cycle, e.g., to normalize a floating-point number or to help detect the edge of a polygon in graphics applications.

To support system diagnostics, the Am29C332 has a special "Master-Slave" mode. To use this mode, two chips are connected in parallel, and hence receive the same instructions and data. The master chip is used for the normal data path. However, in the slave chip, all outputs becomes inputs. The slave compares the outputs of the master with its own internally generated result. If the two do not match, the slave will activate an error signal.

As a further diagnostic aid, byte-wise parity checking is performed at both the A and B data inputs. The "parity" signal is activated if an error is detected. Parity bits (one per byte) are generated for the 32-bit output bus.

FUNCTIONAL DESCRIPTION

A detailed description of each functional block is given in the following paragraphs.

64-Bit Funnel Shifter

The 64-bit funnel shifter is a combinatorial network. The 64-bit input is formed from a combination of the A and B inputs. This may be left-shifted by up to 31 bits before being used by the ALU. The output of the shifter is the most significant 32 bits of the result. The 64-bit shifter can be used on either the A or B operands to perform barrel shifts (either up or down) or rotates. The operation is controlled by positioning operands properly at the input of the 64-bit up-shifter.

The number "n" by which the operand is shifted comes from two sources: the microprogram memory via the P₀ – P₅ pins or the internal register (byte 0 of the status register), PR₀ – PR₅, as selected by an instruction bit.

In general, the 6-bit position input, P₀ – P₅, takes a 6-bit two's complement number representing upshifts from 0 to 31 places (positive numbers) or downshifts from 1 to 32 places (negative numbers).

Mask Generator

The mask generator logic provides the ability to generate the appropriate mask for an operand of given width and position. The generation of the mask depends upon two types of instructions. The first type has byte boundary aligned operands (widths of either 1, 2, 3 or 4 bytes) with the least significant bit aligned to bit 0. The width of an operand is specified by the byte width inputs (I₈ and I₇) as shown in Table 3. The second type of instruction has operands of variable width (1 to 32 bits) and position. The operand is specified by the width inputs (W₀ – W₄) and the position inputs (P₀ – P₅) indicating the least significant bit position of the operand. Thus, in this type of instruction the operand may or may not be least significant bit aligned. Depending upon the type of instruction, the mask generator first generates a fence of all zeros starting from the least significant bit with the width specified either by the byte width or the width input fields. This fence can be upshifted by up to 31 bits by the 32-bit mask shifter. Whenever the mask is moved up over the 32-bit boundary, it does not wrap around. Instead, ONE's are inserted from the least significant end. This configuration provides the ability to operate on a contiguous field located anywhere in a word, or across a word boundary.

The mask generator can be used as a pattern generator by allowing the mask to pass through ALU (by using the PASS-MASK instruction). For example, a single-bit wide mask can be generated and by shifting it up by different amounts can give walking ONE or walking ZERO patterns for memory tests.

TABLE 3.

I ₈	I ₇	Width in Bytes
0	0	4
0	1	1
1	0	2
1	1	3

Arithmetic and Logical Unit

The ALU is a three input unit which uses the mask as a second or third operand in every instruction. The mask is used to merge two operands. For all selected bits (wherever the mask is 0), the desired operation specified by the instruction input is performed, and for all unselected bits either corresponding destination bits or zeros are passed through. The status of each operation (carry, negative, zero, overflow, link) applies to the result only over the specified width. For all byte aligned arithmetic and logical operations (first three quarters of the instruction set), the status is extracted from the appropriate

byte boundary. For all field operations (last quarter of the instruction set), the operand width is assumed to be 32 bits for status generation. The ZERO flag always indicates the status of all bits selected by the mask.

The actual width of the ALU is 34 bits. There are two extra bits used for the high speed signed and unsigned multiplication instructions. These two bits are automatically concatenated to the most-significant end of the ALU depending upon the width specified for the operation. Since the modified Booth algorithm requires a two-bit down-shift each cycle, these ALU bits generate the two most-significant bits of the partial product.

The ALU is capable of shifting data down by two bits for the multiplication algorithm, up by one bit for the divide algorithm and single-bit-up-shifts.

The processor is capable of performing BCD arithmetic on packed BCD numbers. The ALU has separate carry logic for BCD operations. This logic generates nibble carries (BCD digit carry) from propagate and generate signals formed from the A and B operands. In order to simplify the hardware while maintaining throughput, the BCD add and subtract operations are performed in two cycles. In the first cycle, ordinary binary addition or subtraction is performed and BCD nibble carries are generated. These are blocked from affecting the result at this stage, but are saved in the status register to be used later for BCD correction ($NC_0 - NC_7$). In the second cycle all BCD numbers are adjusted by examining the previously generated nibble carries. Since all the necessary information is stored in the status register, the processor can be interrupted after the first BCD cycle.

Priority Encoder

The priority encoder is provided to support floating-point arithmetic and some graphics primitives. The priority encoder takes up to 32 bits as input and generates a 5-bit wide binary code to indicate location of the most significant one in the operand. Input to the priority encoder comes from the input multiplexer, which masks all bits that the user does not want to participate in the prioritization. The priority encoder supports 8, 16, 24 and 32-bit operations depending upon the byte width specified. For each data type the priority encoder generates the appropriate binary weighted code. For example, when a byte width of two is specified ($I_7 - I_8 = 10$), the output of the encoder is zero when bit 15 is HIGH. However, if byte width of four is specified ($I_8 - I_7 = 00$), the output of encoder is 16 (decimal) if bit 15 is HIGH and bits 31 - 16 are LOW. Table 4 shows the output for each data type. If none of the inputs are HIGH or the most significant bit of the data type specified is HIGH, then the output is zero. The difference between these two cases is indicated by the Z-flag of the status register which is HIGH only if all inputs are zero.

Q-Register

The Q-register holds dividend and quotient bits for division, and multiplier and product bits for multiplication. During division, the contents of the Q-register are shifted left, a bit at a time, with quotient bits inserted into bit 0. During multiplication, the contents of the Q-register are shifted right, two bits at

a time, with product bits inserted into the most-significant two bits (according to the selected byte width). The Q-register may be loaded from the A or B inputs and read onto the Y bus.

Master-Slave Comparator

All ALU outputs (except MSERR) employ three-state buffers. The master-slave comparator compares the input and output of each buffer. Any difference causes the MSERR signal to be made true. In Slave mode, all output buffers are disabled. Outputs from a second ALU may then be connected to the equivalent pins of the first. The comparator in the slave will then detect any difference in the results generated by the two. When the Y bus is three-stated by making Output-Enable false, the Y bus master-slave comparators are disabled.

Parity Logic

For each byte of the DA and DB inputs there is an associated parity bit (8 in all). If a parity error is detected on any byte, the Parity-Error signal is made true. Four parity signals (one per byte) are also generated for the Y bus outputs. EVEN parity is employed for the Am29C332.

Status Register

All necessary information about operations performed in the ALU is stored in the 32-bit wide status register after every microcycle. Since the register can be saved, an interrupt can occur after any cycle. The status register can be loaded from either the A or B input of the chip and can be read out on the Y bus for saving in an external register file. For loading, the byte width indicates how many bytes are to be updated. The status register is only updated if the HOLD input is inactive.

Each byte of the status register holds different types of information (see Figure 3). The least significant byte (bits 0 to 7) holds eight position bits ($PR_0 - PR_7$) for the data shifter. The two most significant bits are not used. The next most significant byte (bits 8 to 15) holds the 5-bit width field ($WR_0 - WR_4$) for the mask generator. The three most-significant bits of that byte (bits 13 to 15) are read-only bits that represent three different conditions extracted from the other bits of the status register. They are $\bar{C} + Z$, $N \oplus V$, and $(N \oplus V) + Z$ for bits 13, 14 and 15 respectively. These bits can be read on the Y_0 pin by the extract-status instruction. The next byte contains all the necessary information generated by an ALU operation. The least-significant four bits (bits 16 to 19) hold carry, negative, overflow and zero flags. Bit 20 holds link information for single bit shifts and bits 21 and 22 are used by the multiply and divide instructions. The M flag holds the multiplier bit for the modified Booth algorithm or it holds the sign comparison result for the divide algorithm. The S flag holds the sign of the partial remainder for unsigned division. Both the flags (M and S) are provided as a part of the status register so that multiply and divide instructions can be interrupted at microinstruction boundaries. The most significant byte of the status register holds nibble carries for BCD arithmetic. Since BCD arithmetic is performed in two cycles, the nibble carries are saved in the first cycle and used in the second cycle. Since all the information is stored, BCD instructions are also interruptible at the microinstruction boundary.

TABLE 4.

Highest Priority Active Bit	Encoder Output
l₇ - l₈ = 00 (32-bit)	
None	0
31	0
30	1
29	2
28	3
.	.
.	.
1	30
0	31
l₇ - l₈ = 01 (8-bit)	
None	0
7	0
6	1
5	2
.	.
.	.
1	6
0	7
l₇ - l₈ = 10 (16-bit)	
None	0
15	0
14	1
13	2
12	3
.	.
.	.
1	14
0	15
l₇ - l₈ = 11 (24-bit)	
None	0
23	0
22	1
21	2
20	3
.	.
.	.
1	22
0	23

Status₀₋₇: Position Register

PR ₇	PR ₆	PR ₅	PR ₄	PR ₃	PR ₂	PR ₁	PR ₀
7	6	5	4	3	2	1	0

Status₈₋₁₂: Width Register
Status₁₃: C + Z
Status₁₄: N ⊕ V
Status₁₅: (N ⊕ V) + Z

} Read Only

SIGNED LE	SIGNED LT	UNSIGNED LE	WR ₄	WR ₃	WR ₂	WR ₁	WR ₀
15	14	13	12	11	10	9	8

Status₁₆: Carry
Status₁₇: Negative
Status₁₈: Overflow
Status₁₉: Zero
Status₂₀: Link
Status₂₁: Multiply (and divide) Bit
Status₂₂: Sign Flag
Status₂₃: 0

0	S	M	L	Z	V	N	C
23	22	21	20	19	18	17	16

Status₂₄₋₃₁: Nibble Carries

NC ₇	NC ₆	NC ₅	NC ₄	NC ₃	NC ₂	NC ₁	NC ₀
31	30	29	28	27	26	25	24

Note: Overflow is defined as follows:
 $V = (\text{carry in to MSB}) \oplus (\text{carry out of MSB})$

Figure 3. ALU Status Register Bit Assignment

Am29C332 INSTRUCTION SET

Data Types

The Am29C332 supports the following data types:

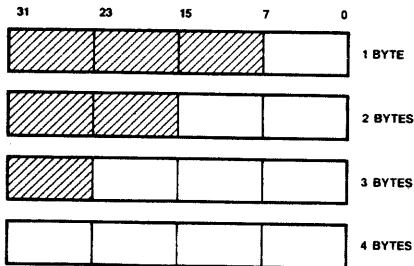
1. Integer
2. Binary-coded decimal
3. Variable-length bit field

The first two data types fall into the category of byte boundary aligned operands (Figure 4). The size of the operand could be 1 byte, 2 bytes, 3 bytes or 4 bytes. All operands are least significant bit (bit 0) aligned. The byte width is determined by bits I_8 and I_7 of the instruction as shown in Table 5.

TABLE 5.

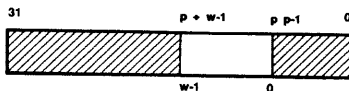
I_8	I_7	Width in Bytes
0	0	4
0	1	1
1	0	2
1	1	3

The third data type has operands of variable width (1 to 32 bits) as shown in Figure 4. The operand is specified by width inputs ($W_0 - W_4$) and position inputs ($P_0 - P_5$). The position inputs indicate the least significant bit position of the operand. Depending on bits I_8 and I_7 of the instruction, the width and position inputs can be selected from either the Status Register or the Width and Position Pins as shown in Table 6. A summary of the data types available is illustrated in Table 7.



TB000096

Byte Boundary Aligned Operands



TB000630

Variable-Length Bit Field

- p = Bit displacement of the least significant field with respect to bit 0.
 w = Width of bit field.

Figure 4. Data Types

TABLE 6.

I_8	I_7	Position		Width	
		Pins	Reg	Pins	Reg
0	0	X		X	
0	1	X			X
1	0		X	X	
1	1		X		X

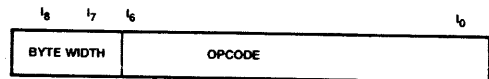
TABLE 7.

Data Type	Size	Range	
Integer		Signed	Unsigned
1 byte	8 bits	-128 to +127	0 to 255
2 bytes	16 bits	-2^{15} to $+2^{15}-1$	0 to $2^{16}-1$
3 bytes	24 bits	-2^{23} to $2^{23}-1$	0 to $2^{24}-1$
4 bytes	32 bits	-2^{31} to $2^{31}-1$	0 to $2^{32}-1$
BCD	1 to 4 bytes (8 digits)	Numeric, 2 digits per byte. Most-significant digit may be used for sign.	
Variable	1 to 32 bits	Dependent on position and width inputs.	

Instruction Format

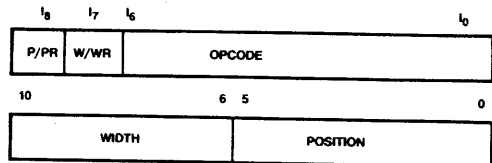
The Am29C332 has two types of Instruction Formats:

1. Byte Boundary Aligned Instructions (FORMAT 1):



TB000098

2. Variable-Length Field Bit Instructions (FORMAT 2):



TB000099

For instructions that allow a field to be shifted up or down, $P_0 - P_5$ is a two's-complement number in the range -32 to +31 representing the direction and magnitude of the shift. For instructions that assume a fixed field position, $P_0 - P_4$ represent the position of the least-significant bit of the field and P_5 is ignored.

Instruction Classification

ALU instructions can be classified as follows:

A. Byte Boundary Aligned Operand Instructions:

1. Arithmetic
 - Binary, BCD
 - Multiply steps
 - Division steps (single and multiple precision)
2. Prioritize
3. Logical
4. Single-bit shifts
5. Data movement

B. Variable-Length Bit Field Operand Instructions:

1. N-bit shifts and rotates
2. Bit manipulations
3. Field logical operations (aligned, non-aligned, extract)
4. Mask generation

Three-fourths of the ALU instructions apply to operands that are byte boundary aligned. For these instructions, two orthogonal issues are the width of the operand (in bytes) and the contents of the high order unselected bytes on the Y bus. As mentioned earlier, the width of the operand is specified by l_6 and l_7 . With the exception of a few instructions, the unselected bytes are assigned values as follows: for single operand instructions, unselected bytes are passed unchanged from the source (A or B). For two operand instructions, unselected bytes are passed unchanged from the destination (B input).

In the last quarter of the instruction set, the width of the operand is from 1 to 32 bits (based on the width input) for field operations, 32 bits for N-bit shift operations and 1-bit for bit-oriented operations. In the case of field-aligned and single-bit operands, the position bits ($P_0 - P_4$) determine the least significant bit of the operand. In the case of N-bit shifts and field non-aligned operands, the position bits $P_0 - P_5$ is a 6-bit signed integer determining the magnitude and direction of the shift.

Flags

Byte-Aligned Instructions

The zero flag always looks only at the selected bytes:

$$Z \leftarrow (Y \text{ and bytemask (byte width)} = 0)$$

Similarly, $N \leftarrow$ sign bit (Y, byte width), where the function "sign-bit" returns bit 7, 15, 23, or 31 of the first argument for byte widths 01, 10, 11, or 00 respectively.

Also, $C \leftarrow$ carry (byte width) returns the carry from the appropriate byte boundary, and:

$$V \leftarrow \text{overflow (byte width)} = (\text{carry into MSB}) \oplus (\text{carry out of MSB})$$

returns the overflow from the appropriate byte boundary.

The link (L) flag is generally loaded with the bit moved out of the highest selected byte in the case of upshifts, or the bit moved out of the least significant byte for downshifts. Figure 5 shows the shift operation using link bit. Other status flags have specialized uses, explained in the following sections.

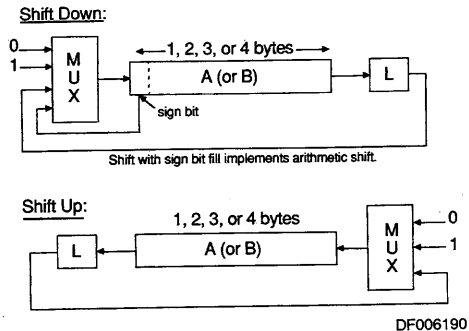


Figure 5. Upshift/Downshift Using Link Bit

Variable-Length Field Instruction:

Generally, only N and Z are affected. N takes the most-significant bit of the 32-bit result (i.e., $N \leftarrow Y_{31}$). Z detects zeros in the selected field of the result (i.e., $Z \leftarrow (Y \text{ and bitmask (position, width)} = 0)$).

Output Select

The Register Status pin, RS, may be used to switch the C, Z, N, V, and L output pins between the direct output of the ALU and the outputs of the corresponding bits in the status register. If the direct status output is selected, then for instructions that do not affect a particular flag (e.g., carry for logical arithmetic) that output will reflect the state of its corresponding bit in the status register. Similarly, when the HOLD signal is made HIGH, the C, Z, N, V and L pins will be made equal to the contents of the status register, regardless of the RS input.

INSTRUCTION SET SUMMARY

Operand Size: Variable Byte Width: 1, 2, 3, 4 Bytes

Type	Operation	Data Type
Arithmetic	<ul style="list-style-type: none"> ● Increment by one, two, four ● Decrement by one, two, four ● Add, addc (carry = macro/micro) ● Sub, subr ● Subc, subrc (carry/borrow) ● BCD sum and difference correct steps 	Binary Integer and BCD
	<ul style="list-style-type: none"> ● Negate (two's complement) ● Multiply steps (modified Booth) ● Divide steps (non-restoring) 	(Signed and unsigned)
Prioritize	<ul style="list-style-type: none"> ● Prioritize 	Binary
Logical	<ul style="list-style-type: none"> ● Not, OR, AND, XOR, XNOR, zero, sign 	Binary
Single-Bit Shifts	<ul style="list-style-type: none"> ● Upshift with 0, 1, link fill ● Downshift with 0, 1, link, sign fill 	(Single and double precision)
Data Movement	<ul style="list-style-type: none"> ● Zero extend ● Sign extend ● Pass-status, Q-Reg ● Load-status, Q-Reg ● Merge 	Binary

Operand Size: 32 Bits

Type	Operation	Data Type
N-Bit Shifts N-Bit Rotates	<ul style="list-style-type: none"> ● Upshift by 0 to 31 bits with 0 fill ● Downshift by 1 to 32 bits with 0, sign fill ● Rotate by 0 to 31 bits 	Binary

Operand Size: Single Bit

Type	Operation	Data Type
Bit Manipulation	<ul style="list-style-type: none"> ● Extract ● Set ● Reset 	Binary

Operand Size: Variable Length Bitfield: 1 to 32 Bits

Type	Operation	Data Type
Field Logical (aligned and non-aligned)	<ul style="list-style-type: none"> ● Not, OR, XOR, AND, extract, insert 	Binary
Mask	<ul style="list-style-type: none"> ● Pass-mask 	Binary

INSTRUCTION SET GLOSSARY
(Sorted by Opcode in Hex Notation)

Opcode	Name	Opcode	Name	Opcode	Name	Opcode	Name
00	ZERO-EXTA	20	DN1-0F-A	40	AND	60	NB-SN-SHA
01	ZERO-EXTB	21	DN1-0F-B	41	XNOR	61	NB-SN-SHB
02	SIGN-EXTA	22	DN1-0F-AQ	42	ADD	62	NB-0F-SHA
03	SIGN-EXTB	23	DN1-0F-BQ	43	ADDC	63	NB-0F-SHB
04	PASS-STAT	24	DN1-1F-A	44	SUB	64	NBROT-A
05	PASS-Q	25	DN1-1F-B	45	SUBC	65	NBROT-B
06	LOADQ-A	26	DN1-1F-AQ	46	SUBR	66	EXTBIT-A
07	LOADQ-B	27	DN1-1F-BQ	47	SUBRC	67	EXTBIT-B
08	NOT-A	28	DN1-LF-A	48	SUM-CORR-A	68	SETBIT-A
09	NOT-B	29	DN1-LF-B	49	SUM-CORR-B	69	SETBIT-B
0A	NEG-A	2A	DN1-LF-AQ	4A	DIFF-CORR-A	6A	RSTBIT-A
0B	NEG-B	2B	DN1-LF-BQ	4B	DIFF-CORR-B	6B	RSTBIT-B
0C	PRIOR-A	2C	DN1-AR-A	4C	-	6C	SETBIT-STAT
0D	PRIOR-B	2D	DN1-AR-B	4D	-	6D	RSTBIT-STAT
0E	MERGEA-B	2E	DN1-AR-AQ	4E	SDIVFIRST	6E	NOTF-AL-B
0F	MERGB-A	2F	DN1-AR-BQ	4F	UDIVFIRST	6F	PASSF-AL-B
10	DECR-A	30	UP1-0F-A	50	SDIVSTEP	70	NOTF-A
11	DECR-B	31	UP1-0F-B	51	SDIVLAST1	71	NOTF-AL-A
12	INCR-A	32	UP1-0F-AQ	52	MPDIVSTEP1	72	PASSF-A
13	INCR-B	33	UP1-0F-BQ	53	MPSDIVSTEP3	73	PASSF-AL-A
14	DECR2-A	34	UP1-1F-A	54	UDIVSTEP	74	ORF-A
15	DECR2-B	35	UP1-1F-B	55	UDIVLAST	75	ORF-AL-A
16	INCR2-A	36	UP1-1F-AQ	56	MPDIVSTEP2	76	XORF-A
17	INCR2-B	37	UP1-1F-BQ	57	MPDIVSTP3	77	XORF-AL-A
18	DECR4-A	38	UP1-LF-A	58	REMCORR	78	ANDF-A
19	DECR4-B	39	UP1-LF-B	59	QUOCORR	79	ANDF-AL-A
1A	INCR4-A	3A	UP1-LF-AQ	5A	SDIVLAST2	7A	EXTF-A
1B	INCR4-B	3B	UP1-LF-BQ	5B	UMULFIRST	7B	EXTF-B
1C	LDSTAT-A	3C	ZERO	5C	UMULSTEP	7C	EXTF-AB
1D	LDSTAT-B	3D	SIGN	5D	UMULLAST	7D	EXTF-BA
1E	-	3E	OR	5E	SMULSTEP	7E	EXTBIT-STAT
1F	-	3F	XOR	5F	SMULFIRST	7F	PASS-MASK

TABLE 6-1. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
ZERO-EXTA	00	Zero Extend	0	A				*		*	
ZERO-EXTB	01		0	B				*		*	
SIGN-EXTA	02	Sign Extend	Sign	A				*		*	
SIGN-EXTB	03		Sign	B				*		*	
MERGEA-B	0E	Merge A with B	B	A Merge B				*		*	
MERGEB-A	0F	Merge B with A	A	B Merge A				*		*	

TABLE 6-2. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status Register	Status						
			Unsel	Sel		S	M	L	Z	V	N	C
PASS-STAT	04	Pass Status Register	B	S								
LDSTAT-A	1C	Load Status Register	S	A	A	+	+	+	+	+	+	+
LDSTAT-B	1D		S	B	B	+	+	+	+	+	+	+

TABLE 6-3. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Q Register	Status						
			Unsel	Sel		S	M	L	Z	V	N	C
PASS-Q	05	Pass Q Register	B	Q								
LOADQ-A	06	Load Q	Q	A	A				*		*	
LOADQ-B	07		Q	B	B				*		*	

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 + = Updated only if byte width is 3 or 4
 * = Updated

Examples:

- 2, ZERO EXTB Pass lower two bytes of B to Y with zero fill on upper two bytes
- 0, LOADQ-A Load all four bytes of A into Q Register pass updated Q Register to Y

TABLE 7. LOGICAL INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status							
			Unsel	Sel	S	M	L	Z	V	N	C	
NOT-A	08	One's Complement	A	\bar{A}				*		*		
NOT-B	09		B	\bar{B}				*		*		
ZERO	3C	Pass Zero	B	0				1		0		
SIGN	3D	Pass Sign	B	$0(N=0); -1(N=1)$				N				
OR	3E	OR	B	A OR B				*		*		
XOR	3F	EXOR	B	A XOR B				*		*		
AND	40	AND	B	A AND B				*		*		
XNOR	41	XNOR	B	A XNOR B				*		*		

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Examples:

- 2, NOT-A Complement low order two bytes of A and output to Y with high order two bytes of A uncomplemented.
- 1, AND AND first byte of A and B. Output to Y with high three bytes of B.

TABLE 8-1. SINGLE-BIT SHIFT INSTRUCTIONS (SINGLE PRECISION)

Mnemonics	Code	Description	Y Output		Status							
			Unsel	Sel	S	M	L	Z	V	N	C	
DN1-0F-A	20	Downshift, Zero Fill	A	$Y_i = A_{i+1}, Y_{msb} = 0$			*	*		*		
			B	$Y_i = B_{i+1}, Y_{msb} = 0$			*	*		*		
DN1-1F-A	24	Downshift, One Fill	A	$Y_i = A_{i+1}, Y_{msb} = 1$			*	*		*		
			B	$Y_i = B_{i+1}, Y_{msb} = 1$			*	*		*		
DN1-LF-A	28	Downshift, Link Fill	A	$Y_i = A_{i+1}, Y_{msb} = L$			*	*		*		
			B	$Y_i = B_{i+1}, Y_{msb} = L$			*	*		*		
DN1-AR-A	2C	Downshift, Sign Fill	A	$Y_i = A_{i+1}, Y_{msb} = N$			*	*		*		
			B	$Y_i = B_{i+1}, Y_{msb} = N$			*	*		*		
UP1-0F-A	30	Upshift, Zero Fill	A	$Y_i = A_{i-1}, Y_0 = 0$			*	*	*	*		
			B	$Y_i = B_{i-1}, Y_0 = 0$			*	*	*	*		
UP1-1F-A	34	Upshift, One Fill	A	$Y_i = A_{i-1}, Y_0 = 1$			*	*	*	*		
			B	$Y_i = B_{i-1}, Y_0 = 1$			*	*	*	*		
UP1-LF-A	38	Upshift, Link Fill	A	$Y_i = A_{i-1}, Y_0 = L$			*	*	*	*		
			B	$Y_i = B_{i-1}, Y_0 = L$			*	*	*	*		

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Example:

- 2, UP1-1F-A Shift lower two bytes of A up one bit. Set LSB to 1. Fill unselected bytes to upper two bytes of A.

TABLE 8-2. SINGLE-BIT SHIFT INSTRUCTIONS (DOUBLE PRECISION)

Mnemonics	Code	Description	Y Output & Q Register	Status						
			Selected Bytes	S	M	L	Z	V	N	C
DN1-0F-AQ	22	Downshift, Zero Fill	0 → A → Q 2)			*	*		*	
DN1-0F-BQ	23		0 → B → Q 3)			*	*		*	
DN1-1F-AQ	26	Downshift, One Fill	1 → A → Q 2)			*	*		*	
DN1-1F-BQ	27		1 → B → Q 3)			*	*		*	
DN1-LF-AQ	2A	Downshift, Link Fill	L → A → Q 2)			*	*		*	
DN1-LF-BQ	2B		L → B → Q 3)			*	*		*	
DN1-AR-AQ	2E	Downshift, Sign Fill	N → A → Q 2)			*	*		*	
DN1-AR-BQ	2F		N → B → Q 3)			*	*		*	
UP1-0F-AQ	32	Upshift, Zero Fill	A ← Q ← 0 2)			*	*	*	*	
UP1-0F-BQ	33		B ← Q ← 0 3)			*	*	*	*	
UP1-1F-AQ	36	Upshift, One Fill	A ← Q ← 1 2)			*	*	*	*	
UP1-1F-BQ	37		B ← Q ← 1 3)			*	*	*	*	
UP1-LF-AQ	3A	Upshift, Link Fill	A ← Q ← L 2)			*	*	*	*	
UP1-LF-BQ	3B		B ← Q ← L 3)			*	*	*	*	

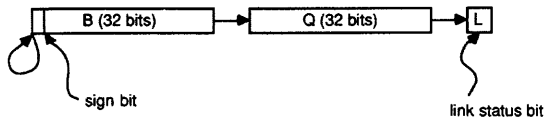
- Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. Y Unselected byte from A, Q Unselected byte unchanged.
 3. Y Unselected byte from B, Q Unselected byte unchanged.

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

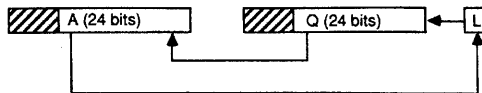
0, DN1-AR-BQ

Shift 64 bits (all 32 bits of both B and Q) down by one bit. LSB of B fills MSB of Q. MSB of B set to sign bit (bit N of status register).



3, UP1-LF-AQ

Shift 48 bits (24-bits of A and 24-bits of Q) up by one bit. MSB of 24-bit Q fills LSB of A. MSB of 24-bit A sets link status bit. LSB of Q is filled with original link value.



DF006200

TABLE 9. PRIORITIZE INSTRUCTIONS

Mnemonics	Code	Description	Y Output	Status						
				S	M	L	Z	V	N	C
PRIOR-A	0C	Prioritization	Location of Highest 1 Bit				*			
PRIOR-B	0D						*			

- Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. Priority also loaded into STATUS <7:0>
 3. Refer to Table 4.

Legend: A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

3, PRIOR-A Value placed on Y is 2

Assume A is

01001011	00100010	00000000	00000000
----------	----------	----------	----------

TABLE 10-1. ARITHMETIC INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
NEG-A	0A	Two's Complement	A	$\bar{A} + 1$				*	*	*	*
NEG-B	0B		B	$\bar{B} + 1$				*	*	*	*
INCR-A	12	Increment by One	A	A + 1				*	*	*	*
INCR-B	13		B	B + 1				*	*	*	*
INCR2-A	16	Increment by Two	A	A + 2				*	*	*	*
INCR2-B	17		B	B + 2				*	*	*	*
INCR4-A	1A	Increment by Four	A	A + 4				*	*	*	*
INCR4-B	1B		B	B + 4				*	*	*	*
DECR-A	10	Decrement by One	A	A - 1				*	*	*	*
DECR-B	11		B	B - 1				*	*	*	*
DECR2-A	14	Decrement by Two	A	A - 2				*	*	*	*
DECR2-B	15		B	B - 2				*	*	*	*
DECR4-A	18	Decrement by Four	A	A - 4				*	*	*	*
DECR4-B	19		B	B - 4				*	*	*	*

- Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. Borrow, rather than carry, is generated if BORROW is HIGH (borrow = carry).
 3. Nibble bits are set by these instructions. NEG-A (or NEG-B) and DIFF-CORR may be used to form 10's complement of a BCD number. Use SUM-CORR (for increment) or DIFF-CORR (for decrement) to increment or decrement a BCD number.

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

2, DECR4-A Decrement lower two bytes of A by 4

TABLE 10-2. ARITHMETIC INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
ADD	42	Add	B	A + B				*	*	*	*
ADDC	43	Add with Carry	B	A + B + C 6)				*	*	*	*
SUB	44	Subtract	B	A + \bar{B} + 1				*	*	*	*
SUBR	46		B	B + \bar{A} + 1				*	*	*	*
SUBC	45	Subtract with Carry	B	A + \bar{B} + 1 + C 2) 6)				*	*	*	*
SUBRC	47		B	B + \bar{A} + 1 + C 2) 6)				*	*	*	*
SUM-CORR-A	48	Correct BCD Nibbles for Addition	A	Corrected A 3)				*	*	*	*
SUM-CORR-B	49		B	Corrected B 3)				*	*	*	*
DIFF-CORR-A	4A	Correct BCD Nibbles for Subtraction	A	Corrected A 3)				*	*	*	*
DIFF-CORR-B	4B		B	Corrected B 3)				*	*	*	*

- Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. BOROW is LOW. For subtract operations, a borrow rather than a carry is stored in STATUS if BOROW is HIGH. Carry is always generated for ADD regardless of BOROW.
 3. First, the nibble carries $NC_0 - NC_7$ are tested. Any nibble carry/borrow that is set to 1 generates "6" internally as a correction word and then the correction word is added (SUM-CORR-) or subtracted (DIFF-CORR-) from the operand. $NC_0 - NC_7$ are not affected by this operation.
 4. Use SUM-CORR or DIFF-CORR to add or subtract a BCD number.
 5. Use ADDC, SUBC, or SUBRC to perform operations on integers longer than 32 bits.
 6. Carry bit is obtained from MCin if M/m is HIGH. Otherwise, carry is obtained from the C status bit.

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register

* = Updated only if byte width is 3 or 4

Example:

0, ADD Add two 32-bit two's-complement integers

TABLE 11-1. DIVIDE INSTRUCTIONS (Aligned Format)

Name	I ₆ - I ₀ Code	Description	Source for Unselected Bytes	Output	Status						
					S	M	L	Z	V	N	C
Signed Divide Steps											
SDIVFIRST	4 E	First Instruction for Signed Divide	B	Y, Q	*	*	*	*	*	*	*
SDIVSTEP	5 0	Iterate Step (#bits - 1 times)	B	Y, Q		*	*	*	*	*	*
SDIVLAST1	5 1	Last Divide Instruction Unless	B	Y, Q		*	*	*	*	*	*
SDIVLAST2	5 A	Dividend & Remainder Negative	B	Y				*	*	*	*
Unsigned Divide Steps											
UDIVFIRST	4 F	First Instruction for Unsigned Divide	B	Y, Q			*	*	*	*	*
UDIVSTEP	5 4	Iterate Step (#bits - 1 times)	B	Y, Q	*	*	*	*	*	*	*
UDIVLAST	5 5	Last Instruction	B	Y, Q	0	*	*	*	*	*	*
Multiprecision Divide Steps											
MPDIVSTEP1	5 2	First Instruction	B	Y, Q							
MPDIVSTEP2	5 6	Executed 0 Times for Double	B	Y, Q							
MPSDIVSTEP3	5 3	Last Instruction of Inner Loop	B	Y, Q							
MPUDIVSTP3	5 7	Used for Unsigned Divide	B	Y, Q							
Correction Steps											
REMCORR	5 8	Correct Remainder After Divide	B	Y							*
QUOCORR	5 9	Correct Quotient After Divide	B	Y						*	*

TABLE 11-2. EXAMPLE CODING FORM (Signed Division)

Am29C331				Am29C332				Am29C334			Am29C332 Y-Out
OP	Branch	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	
CONT				2	LOADQ-A			R2			1
CONT				0	SIGN					R3	0
FOR_D	15			2	SDIVFIRST			R4	R3	R3	0
DJMP_S				2	SDIVSTEP			R4	R3	R3	0
CONT				2	SDIVLAST1			R4	R3	R3	0
BRCC_D	DONE	Z									1
CONT				2	SDIVLAST2A			R4	R3	R3	0
CONT				2	PASS-Q					R1	0
CONT				2	QUOCORR				R1	R1	0
CONT				2	REMCORR			R4	R3	R3	0

Note: Divisor in A, Dividend in A
Quotient in Q, Remainder in B

Legend: A = A Input
B = B Input
S = Status Register
Q = Q Register
R1 = Quotient
R2 = Dividend
R3 = Remainder
R4 = Divisor

TABLE 12-1. MULTIPLY INSTRUCTIONS (Aligned Format)

Name	I ₆ - I ₀ Code	Description	Source for Unselected Bytes	Output	Status						
					S	M	L	Z	V	N	C
Signed Multiply Steps											
SMULFIRST	5 F	First multiply instruction	B	Y ⁽¹⁾							
SMULSTEP	5 E	Iterate step (#bits/2 - 1 steps)	B	Y ⁽¹⁾							
Unsigned Multiply Steps											
UMULFIRST	5 B	First multiply instruction	B	Y ⁽¹⁾		*					
UMULSTEP	5 C	Iterate step (#bits/2 - 1 steps)	B	Y ⁽¹⁾		*					
UMULLAST	5 D	Last multiply instruction	B	Y ⁽¹⁾			*				

TABLE 12-2. EXAMPLE CODING FORM (Unsigned Multiply)

Am29C331				Am29C332				Am29C334			Am29C332 Y-Out
OP	Branch	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	
CONT				3	ZERO				R3	R3	0
CONT				3	LOADQ-A			R1			1
FOR_D	11 ₁₀			3	ULMULFIRST			R2	R3	R3	0
DJMP_S				3	UMULSTEP			R2	R3	R3	0
CONT				3	UMULLAST			R2	R3	R3	0
CONT				3	PASS-Q					R4	0

Note: 1. Put ALU output in B.
 2. Multiplicand in A, Multiplier in Q
 Product (HIGH) in B, Product (LOW) in Q

Legend: A = A Input
 B = B Input
 S = Status Register
 Q = Q Register
 R1 = Multiplier
 R2 = Multiplicand
 R3 = Product (HIGH)
 R4 = Product (LOW)

TABLE 13. SHIFT/ROTATE INSTRUCTIONS

Mnemonics	Code	Description	Y Output	Status						
				S	M	L	Z	V	N	C
NB-OF-SHA	62	Field Shift, Zero Fill	$Y_{i+p} = A_i, 0$ 2)				*	*		
NB-OF-SHB	63		$Y_{i+p} = B_i, 0$ 2)				*	*		
NB-SN-SHA	60	Field Shift, Sign Fill	$Y_{i+p} = A_i, N$ 2)				*	*		
NB-SN-SHB	61		$Y_{i+p} = B_i, N$ 2)				*	*		
NBROT-A	64	Field Rotate	$Y_i = A_{(i-p) \bmod 32}$ 3)				*	*		
NBROT-B	65		$Y_i = B_{(i-p) \bmod 32}$ 3)				*	*		

- Notes: 1. These instructions use the field instruction format (FORMAT 2).
 2. "p" stands for bit displacement from P₀-P₅ or from PR₀-PR₅ (-32 ≤ p ≤ 31).
 If p is positive, Y_{p-1} to Y₀ are equal to the fill bit.
 If p is negative, Y₃₁ to Y_{31+p+1} are equal to the fill bit.
 3. The sign of the position input is ignored for this instruction and P₀-P₄ are treated as a positive magnitude for a circular upshift.

Legend: A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Examples: *
 NB-OF-SHA,,4 Shift A up 4 bits and zero fill
 NB-OF-SHB,-17 Shift B down 17 bits and sign fill

*Width field not used

TABLE 14-1. BIT-MANIPULATION INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
SETBIT-A	68	Bit Set	A	$Y_i = A_i, Y_p = 1$				*	*		
SETBIT-B	69		B	$Y_i = B_i, Y_p = 1$				*	*		
RSTBIT-A	6A	Bit Reset	A	$Y_i = A_i, Y_p = 0$				*	*		
RSTBIT-B	6B		B	$Y_i = B_i, Y_p = 0$				*	*		
EXTBIT-A	66	Bit Extract	0	if p > 0, Y ₀ = A _p 2) if p < 0, Y ₀ = A _p			*	*			
EXTBIT-B	67		0	if p > 0, Y ₀ = B _p 2) if p < 0, Y ₀ = B _p			*	*			
EXTBIT-STAT	7E		0	if p > 0, Y ₀ = S _p 2) if p < 0, Y ₀ = S _p			*				

- Notes: 1. These instructions use the field instruction format (FORMAT 2).
 2. Y₃₁ to Y₁ are set to zero. "p" stands for the bit displacement from P₀-P₄ or from PR₀-PR₅. The sign of the position input is ignored.

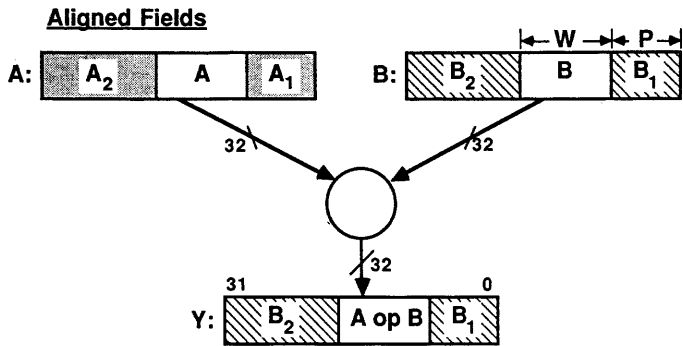
TABLE 14-2. BIT-MANIPULATION INSTRUCTIONS

Mnemonics	Code	Description	Status Register	Y Output	Status						
					S	M	L	Z	V	N	C
SETBIT-STAT	6C	Status Bit Set	S _p = 1	S	*	*	*	*	*	*	*
RSTBIT-STAT	6D		S _p = 0	S	*	*	*	*	*	*	*

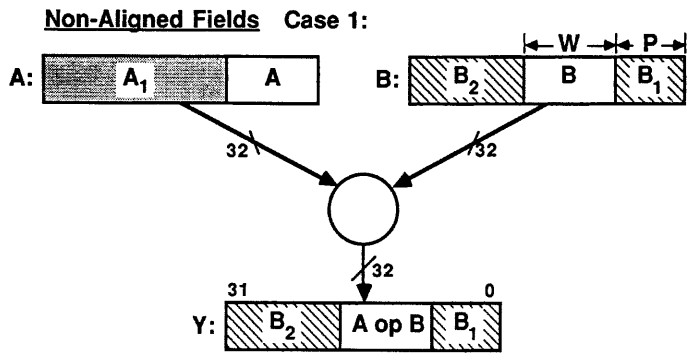
- Notes: 1. These instructions use the Field instruction format (FORMAT 2).
 2. "p" stands for the bit displacement from P₀-P₅ or from PR₀-PR₅.

Legend: Unsel = Unselected field
 Sel = Selected field
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Examples:
 RSTBIT-B,,3 3rd bit is set to 0 in B
 EXTBIT-STAT,-4 4th bit in status register is extracted and inverted.

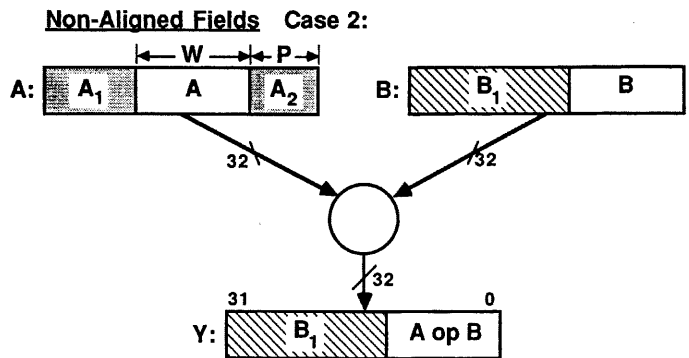


LD000140



If position (P_0-P_5) ≥ 0 , A is LSB aligned
Width (W_0-W_4) = 1 to 32

LD000151



If position (P_0-P_5) < 0 , B is LSB aligned
Width (W_0-W_5) = 1 to 32

LD000161

Figure 6. Field Logical Operations

TABLE 15. FIELD LOGICAL INSTRUCTIONS

Mnemonics	Code	Description		Y Output		Status					
				Unsel	Sel	S	M	L	Z	V	N
PASSF-AL-A	73	Field Pass	3)	B	$Y_i = A_i$				*	*	
PASSF-AL-B	6F		3)	B	$Y_i = B_i$				*	*	
PASSF-A	72		4)	B	if $p \geq 0$, $Y_i = A_{i-p}$ if $p < 0$, $Y_{i- p } = A_i$				*	*	
NOTF-AL-A	71	Field Complement	3)	B	$Y_i = \bar{A}_i$				*	*	
NOTF-AL-B	6E		3)	B	$Y_i = \bar{B}_i$				*	*	
NOTF-A	70		4)	B	if $p \geq 0$, $Y_i = \bar{A}_{i-p}$ if $p < 0$, $Y_{i- p } = \bar{A}_i$				*	*	
ORF-AL-A	75	Field OR	3)	B	$Y_i = A_i \text{ OR } B_i$				*	*	
ORF-A	74		4)	B	if $p \geq 0$, $Y_i = A_{i-p} \text{ OR } B_i$ if $p < 0$, $Y_{i- p } = A_i \text{ OR } B_{i- p }$				*	*	
XORF-AL-A	77	Field XOR	3)	B	$Y_i = A_i \text{ XOR } B_i$				*	*	
XORF-A	76		4)	B	if $p \geq 0$, $Y_i = A_{i-p} \text{ XOR } B_i$ if $p < 0$, $Y_{i- p } = A_i \text{ XOR } B_{i- p }$				*	*	
ANDF-AL-A	79	Field AND	3)	B	$Y_i = A_i \text{ AND } B_i$				*	*	
ANDF-A	78		4)	B	if $p \geq 0$, $Y_i = A_{i-p} \text{ AND } B_i$ if $p < 0$, $Y_{i- p } = A_i \text{ AND } B_{i- p }$				*	*	
EXTF-A	7A	Field Extract	4) 5)	0	if $p \geq 0$, $Y_i = A_{i-p}$ if $p < 0$, $Y_{i- p } = A_i$				*	*	
EXTF-B	7B			0	if $p \geq 0$, $Y_i = B_{i-p}$ if $p < 0$, $Y_{i- p } = B_i$				*	*	
EXTF-AB	7C		0	6)				*	*		
EXTF-BA	7D		0	7)				*	*		

- Notes: 1. These instructions use the field instruction format (FORMAT 2).
 2. $p \leq i \leq p + w - 1$. "p" stands for position displacement from $P_0 - P_5$ or from $PR_0 - PR_5$ and "w" for the width of the bit field from $W_0 - W_4$ or $WR_0 - WR_4$. Whenever $p + w > 32$, operation takes place only over the portion of the field up to the end of the word. No wraparound occurs.
 3. This instruction uses the aligned format (see Figure 6).
 4. This instruction uses the unaligned field format (see Figure 6).
 $p \geq 0$: Case 1
 $p < 0$: Case 2
 5. If p is positive, the input is LSB aligned and Y output aligned at position. If p is negative, the input is aligned at |p| and Y output at LSB.
 6. Firstly, the concatenation of A(High Word) and B(Low Word) is rotated by the amount specified by the position (p). If p is positive, left-rotate is performed. If p is negative, right-rotate is performed. Secondly, the least significant bits on the Y output specified by the width (w) are extracted.
 7. Same as 6) except that B input is taken as a high word and A input as a low word.

Legend: Unsel = Unselected Field
 Sel = Selected Field
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

For all examples, assume STATUS (7:0) is -7 and STATUS (12:8) is 3.

1. 0,PASSF-AL-B,11,20 Pass B to Y and test if B_{20} to B_{30} are all zero. Set Z status if so.

B: 10000000000000000101011100110100

Z set to 1 in this case

2. 3,XORF-A,, Exclusive-OR bits $A_7 - A_9$ with bits $B_0 - B_2$ and output to $Y_0 - Y_2$. Pass $B_3 - B_{31}$ to $Y_3 - Y_{31}$. Width and position values are obtained from STATUS(12:0).

A: 01101110001001000010111001101011

B: 00011100001010001100101001001001

$A_9 - 7 \oplus B_{2-0} = Y$: 00011100001010001100101001001101

TABLE 16. MASK INSTRUCTION

Mnemonics	Code	Description	Y Output		Status							
			Unsel	Sel	S	M	L	Z	V	N	C	
PASS-MASK	7F	Generate Mask	p ₅	$Y_i = \bar{p}_5$								

Notes: 1. This instruction uses the field instruction format (FORMAT 2).

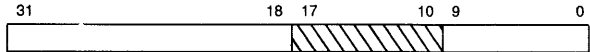
2. $p \leq i \leq p + w - 1$. "p" stands for the position displacement and "w" for the width of bit field.

Legend: Unsel = Unselected Field
 Sel = Selected Field
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

Generates an 8-bit field mask pattern starting from bit position 10.

0, PASS-MASK, 8, 10



ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Case Temperature Under Bias (T_C) -55 to +125°C
 Supply Voltage to Ground Potential
 Continuous -0.3 to +7.0 V
 DC Voltage Applied to Outputs
 for HIGH Output State -0.3 V to V_{CC} + 0.3 V
 DC Input Voltage -0.3 to V_{CC} + 0.3 V
 DC Output Current, Into LOW Outputs 30 mA
 DC Input Current -10 mA to +10 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Case Devices
 Temperature (T_A) 0 to +70°C
 Supply Voltage V_{CC} +4.75 V to +5.25 V
 Military* (M) Devices
 Temperature (T_A) -55 to +125°C
 Supply Voltage (V_{CC}) +4.5 V to +5.5 V

*Military product 100% tested at T_A = +25°C, +125°C, and -55°C.

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range unless otherwise specified (for APL Products, Group A, Subgroups 1, 2, 3 are tested unless otherwise noted)

Parameter Symbol	Parameter Description	Test Conditions (Note 1)		Min.	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OH} = 0.4 mA	2.4		Volts
V _{OL}	Output LOW Voltage	V _{CC} = Min., V _{IN} = V _{IH} or V _{IL}	I _{OL} = 8 mA for Y-Bus & 4 mA for All Other Pins		0.5	Volts
V _{IH}	Guaranteed Input Logical HIGH Voltage (Note 2)			2.0		Volts
V _{IL}	Guaranteed Input Logical LOW Voltage (Note 2)				0.8	Volts
I _{IL}	Input LOW Current	V _{CC} = Max., V _{IN} = 0.5 V			-10	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = V _{CC} - 0.5 V			10	μA
I _{OZH}	Off State (High Impedance) Output Current	V _{CC} = Max., V _O = 2.4 V			10	μA
I _{OZL}		V _{CC} = Max., V _O = 0.5 V			-10	
I _{CC}	Static Power Supply Current (Note 3)	V _{CC} = Max., V _{IN} = V _{CC} or GND, I _O = 0 μA	COM'L		70	mA
			MIL		70	
C _{PD} *	Power Dissipation Capacitance (Note 4)	V _{CC} = 5.0 V, T _A = 25°C No Load				pF Typical

- Notes:**
1. V_{CC} conditions shown as Min. or Max. refer to the Commercial or Military V_{CC} limits.
 2. These input levels provide zero-noise immunity and should only be statically tested in a noise-free environment (not functionally tested).
 3. Worst-case I_{CC} is measured at the lowest temperature in the specified operating range.
 4. C_{PD} determines the no-load dynamic current consumption:
 $I_{CC}(\text{Total}) = I_{CC}(\text{Static}) + C_{PD} V_{CC} f$, where f is the switching frequency of the majority of the internal nodes, normally one-half of the clock frequency.

*This parameter is not tested.

SWITCHING CHARACTERISTICS over **COMMERCIAL** operating range

A. COMBINATIONAL PROPAGATION DELAYS

No.	From	To	29C332	29C332-1	29C332-2	Unit
			Max. Delay	Max. Delay	Max. Delay	
1	PA ₀ - PA ₃ , PB ₀ - PB ₃	PERR	25	20	18	ns
2	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	PERR	32	28	23	ns
3	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	PY ₀ - PY ₃	59	42	34	ns
4	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	Y ₀ - Y ₃₁	49	35	28	ns
5	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	C, Z, V, N, L	60	43	34	ns
6	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	MSERR	68	49	40	ns
7	I ₀ - I ₈	PY ₀ - PY ₃	74	53	43	ns
8	I ₀ - I ₈	Y ₀ - Y ₃₁	66	47	38	ns
9	I ₀ - I ₈	C, Z, V, N, L	67	48	39	ns
10	I ₀ - I ₈	MSERR	77	55	44	ns
11	W ₀ - W ₄	PY ₀ - PY ₃	58	40	32	ns
12	W ₀ - W ₄	Y ₀ - Y ₃₁	52	34	28	ns
13	W ₀ - W ₄	C, Z, V, N, L	57	35	28	ns
14	W ₀ - W ₄	MSERR	62	41	33	ns
15	P ₀ - P ₅	PY ₀ - PY ₃	67	48	39	ns
16	P ₀ - P ₅	Y ₀ - Y ₃₁	59	42	34	ns
17	P ₀ - P ₅	C, Z, V, N, L	60	43	35	ns
18	P ₀ - P ₅	MSERR	68	45	36	ns
19	CP	PY ₀ - PY ₃	74	55	44	ns
20	CP	Y ₀ - Y ₃₁	68	52	42	ns
21	CP	C, Z, V, N, L	74	55	44	ns
22	CP	STATUS REG	28	25	20	ns
23	RS	C, Z, V, N, L	23	21	17	ns
24	MC _{in}	Y ₀ - Y ₃₁	43	31	25	ns
25	MC _{in}	C, Z, V, N, L	48	34	28	ns
26	MC _{in}	MSERR	52	37	30	ns
27	MLINK	Y ₀ - Y ₃₁	46	33	27	ns
28	MLINK	C, Z, V, N, L	52	37	30	ns
29	MLINK	MSERR	53	38	31	ns
30	M/ \bar{m}	Y ₀ - Y ₃₁	46	33	27	ns
31	M/ \bar{m}	C, Z, V, N, L	52	37	30	ns
32	M/ \bar{m}	MSERR	53	38	31	ns
33	BOROW	Y ₀ - Y ₃₁	46	33	27	ns
34	BOROW	C, Z, V, N, L	52	37	30	ns
35	BOROW	MSERR	53	38	31	ns
36	HOLD	C, Z, V, N, L	31	22	18	ns
37	HOLD	MSERR	35	29	24	ns
38	PY ₀ - PY ₃	MSERR	24	22	18	ns
39	Y ₀ - Y ₃₁	MSERR	24	22	18	ns
40	C, Z, V, N, L	MSERR	24	22	18	ns
41	PERR	MSERR	24	22	18	ns

SWITCHING CHARACTERISTICS over **COMMERCIAL** operating range (Cont'd.)

B. SETUP AND HOLD TIMES

No.	Parameter (Note 1)	For	With Respect To	29C332	29C332-1	29C332-2	Unit
				Max. Value	Max. Value	Max. Value	
42	Input Data Setup	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	CP ↑	56	31	31	ns
43	Input Data Hold	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	CP ↑	0	0	0	ns
44	Byte Width Setup	I ₇ - I ₈	CP ↑	66	30	30	ns
45	Byte Width Hold	I ₇ - I ₈	CP ↑	0	0	0	ns
46	Instruction Setup	I ₀ - I ₆	CP ↑	71	37	37	ns
47	Instruction Hold	I ₀ - I ₆	CP ↑	0	0	0	ns
48	Width Setup	W ₀ - W ₄	CP ↑	64	28	28	ns
49	Width Hold	W ₀ - W ₄	CP ↑	0	0	0	ns
50	Position Setup	P ₀ - P ₅	CP ↑	66	28	28	ns
51	Position Hold	P ₀ - P ₅	CP ↑	0	0	0	ns
52	Borrow Setup	BOROW	CP ↑	51	22	22	ns
53	Borrow Hold	BOROW	CP ↑	0	0	0	ns
54	Macro Carry Setup	MCin	CP ↑	50	21	21	ns
55	Macro Carry Hold	MCin	CP ↑	0	0	0	ns
56	Macro Link Setup	MLINK	CP ↑	43	22	22	ns
57	Macro Link Hold	MLINK	CP ↑	0	0	0	ns
58	Macro/Micro Setup	M/ \bar{m}	CP ↑	50	22	22	ns
59	Macro/Micro Hold	M/ \bar{m}	CP ↑	0	0	0	ns
60	Hold Mode Setup	HOLD	CP ↑	28	11	11	ns
61	Hold Mode Hold	HOLD	CP ↑	0	0	0	ns

C. MINIMUM CLOCK REQUIREMENTS

No.	Description	29C332	29C332-1	29C332-2	Unit
		Max. Value	Max. Value	Max. Value	
62	Minimum Clock LOW Time	20	20	20	ns
63	Minimum Clock HIGH Time	20	20	20	ns

D. ENABLE AND DISABLE TIMES

No.	From	To	Description	29C332	29C332-1	29C332-2	Unit
				Max. Value	Max. Value	Max. Value	
64	$\bar{O}E - \bar{Y}$	Y ₀ - Y ₃₁ , PY ₀ - PY ₃	Output Enable Time				ns
65	$\bar{O}E - \bar{Y}$	Y ₀ - Y ₃₁ , PY ₀ - PY ₃	Output Disable Time				ns
66	SLAVE	C, Z, V, N, L PERR	Slave Mode Enable Time				ns
67	SLAVE	Y ₀ - Y ₃₁ , PY ₀ - PY ₃ C, Z, V, N, L PERR	Slave Mode Disable Time				ns

Notes: 1. See timing diagram for desired mode of operation to determine clock edge to which these setup and hold times apply.

SWITCHING CHARACTERISTICS over **MILITARY** operating range

A. COMBINATIONAL PROPAGATION DELAYS

No.	From	To	29C332	
			Max. Delay	Unit
1	PA ₀ - PA ₃ , PB ₀ - PB ₃	PERR	28	ns
2	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	PERR	35	ns
3	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	PY ₀ - PY ₃	65	ns
4	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	Y ₀ - Y ₃₁	54	ns
5	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	C, Z, V, N, L	66	ns
6	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	MSERR	75	ns
7	I ₀ - I ₈	PY ₀ - PY ₃	82	ns
8	I ₀ - I ₈	Y ₀ - Y ₃₁	73	ns
9	I ₀ - I ₈	C, Z, V, N, L	74	ns
10	I ₀ - I ₈	MSERR	85	ns
11	W ₀ - W ₄	PY ₀ - PY ₃	64	ns
12	W ₀ - W ₄	Y ₀ - Y ₃₁	57	ns
13	W ₀ - W ₄	C, Z, V, N, L	63	ns
14	W ₀ - W ₄	MSERR	68	ns
15	P ₀ - P ₅	PY ₀ - PY ₃	74	ns
16	P ₀ - P ₅	Y ₀ - Y ₃₁	65	ns
17	P ₀ - P ₅	C, Z, V, N, L	66	ns
18	P ₀ - P ₅	MSERR	69	ns
19	CP	PY ₀ - PY ₃	82	ns
20	CP	Y ₀ - Y ₃₁	75	ns
21	CP	C, Z, V, N, L	82	ns
22	CP	STATUS REG.	31	ns
23	RS	C, Z, V, N, L	25	ns
24	MC _{in}	Y ₀ - Y ₃₁	47	ns
25	MC _{in}	C, Z, V, N, L	53	ns
26	MC _{in}	MSERR	57	ns
27	MLINK	Y ₀ - Y ₃₁	51	ns
28	MLINK	C, Z, V, N, L	57	ns
29	MLINK	MSERR	58	ns
30	M/m	Y ₀ - Y ₃₁	51	ns
31	M/m	C, Z, V, N, L	57	ns
32	M/m	MSERR	58	ns
33	BOROW	Y ₀ - Y ₃₁	51	ns
34	BOROW	C, Z, V, N, L	57	ns
35	BOROW	MSERR	58	ns
36	HOLD	C, Z, V, N, L	34	ns
37	HOLD	MSERR	39	ns
38	PY ₀ - PY ₃	MSERR	26	ns
39	Y ₀ - Y ₃₁	MSERR	26	ns
40	C, Z, V, N, L	MSERR	26	ns
41	PERR	MSERR	26	ns

SWITCHING CHARACTERISTICS over **MILITARY** operating range (Cont'd.)

B. SETUP AND HOLD TIMES

No.	Parameter (Note 1)	For	With Respect To	29C332	
				Max. Value	Unit
42	Input Data Setup	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	CP ↑	62	ns
43	Input Data Hold	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	CP ↑	0	ns
44	Byte Width Setup	I ₇ - I ₈	CP ↑	73	ns
45	Byte Width Hold	I ₇ - I ₈	CP ↑	0	ns
46	Instruction Setup	I ₀ - I ₆	CP ↑	78	ns
47	Instruction Hold	I ₀ - I ₆	CP ↑	0	ns
48	Width Setup	W ₀ - W ₄	CP ↑	70	ns
49	Width Hold	W ₀ - W ₄	CP ↑	0	ns
50	Position Setup	P ₀ - P ₅	CP ↑	73	ns
51	Position Hold	P ₀ - P ₅	CP ↑	0	ns
52	Borrow Setup	BOROW	CP ↑	56	ns
53	Borrow Hold	BOROW	CP ↑	0	ns
54	Macro Carry Setup	MCin	CP ↑	55	ns
55	Macro Carry Hold	MCin	CP ↑	0	ns
56	Macro Link Setup	MLINK	CP ↑	47	ns
57	Macro Link Hold	MLINK	CP ↑	0	ns
58	Macro/Micro Setup	M/ \bar{m}	CP ↑	55	ns
59	Macro/Micro Hold	M/ \bar{m}	CP ↑	0	ns
60	Hold Mode Setup	HOLD	CP ↑	31	ns
61	Hold Mode Hold	HOLD	CP ↑	0	ns

C. MINIMUM CLOCK REQUIREMENTS

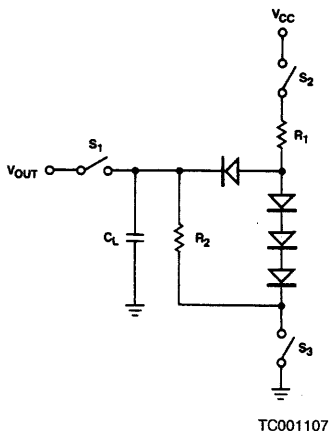
No.	Description	29C332	
		Max. Value	Unit
62	Minimum Clock LOW Time	22	ns
63	Minimum Clock HIGH Time	22	ns

D. ENABLE AND DISABLE TIMES

No.	From	To	Description	29C332	
				Max. Value	Unit
64	OE - Y	Y ₀ - Y ₃₁ , PY ₀ - PY ₃	Output Enable Time		ns
65	OE - Y	Y ₀ - Y ₃₁ , PY ₀ - PY ₃	Output Disable Time		ns
66	SLAVE	C, Z, V, N, L PERR	Slave Mode Enable Time		ns
67	SLAVE	Y ₀ - Y ₃₁ , PY ₀ - PY ₃ C, Z, V, N, L PERR	Slave Mode Disable Time		ns

Notes: 1. See timing diagram for desired mode of operation to determine clock edge to which these setup and hold times apply.

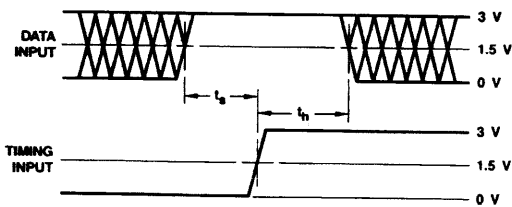
SWITCHING TEST CIRCUIT



A. Three-State Outputs

- Notes:
1. $C_L = 50$ pF includes scope probe, wiring and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 4. S_1 and S_2 are closed while S_3 is open for t_{pZL} test.
 5. $C_L =$ TBD for output disable tests.

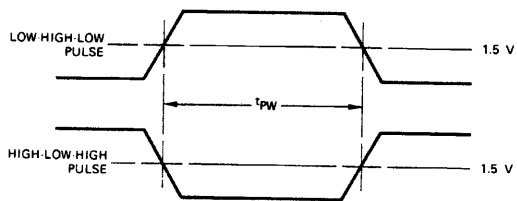
SWITCHING TEST WAVEFORMS



WFR02970

Setup, Hold, and Release Times

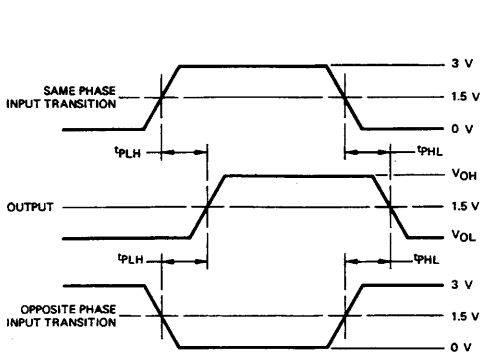
- Notes:
1. Diagram shown for HIGH data only. Output transition may be opposite sense.
 2. Cross hatched area is don't care condition.



WFR02790

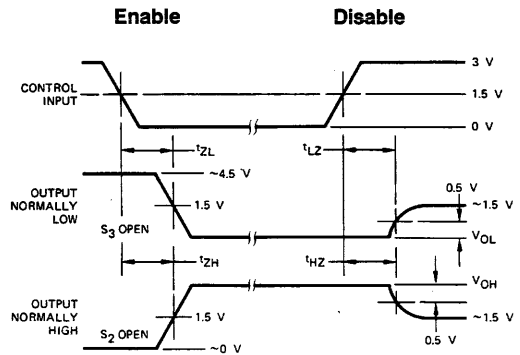
Pulse Width

SWITCHING TEST WAVEFORMS (Cont'd.)



Propagation Delay

WFR02980



Enable and Disable Times

WFR02660

- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
 2. S_1 , S_2 and S_3 of Load Circuit are closed except where shown.

Test Philosophy and Methods

The following points give the general philosophy that we apply to tests that must be properly engineered if they are to be implemented in an automatic environment. The specifics of what philosophies applied to which test are shown.

1. Ensure the part is adequately decoupled at the test head. Large changes in supply current when the device switches may cause function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an input transition, ground current may change by as much as 400 mA in 5 - 8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily.
4. Use extreme care in defining input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins that may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \geq 3$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. Capacitive Loading for AC Testing

Automatic testers and their associated hardware have stray capacitance that varies from one type of tester to another, but is generally around 50 pF. This, of course, makes it impossible to make direct measurements of parameters that call for a smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays" which measure the propagation delays into and out of the high impedance state and are usually specified at a load capacitance of 5.0 pF. In these cases, the test is performed at the higher load capacitance (typically 50 pF) and engineering correlations based on data taken with a bench set up are used to predict the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impossible to make measurements at both capacitances even though they may both be greater than the stray capacitance. In these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is predicted from engineering correlations based on data taken with a bench set up and the knowledge that certain DC measurements (I_{OH} , I_{OL} , for example) have already been taken and are within specification. In some cases, special DC tests are performed in order to facilitate this correlation.

7. Threshold Testing

The noise associated with automatic testing, the long, inductive cables, and the high gain of bipolar devices when in the vicinity of the actual device threshold, frequently give rise to oscillations when testing high-speed speed circuits. These oscillations are not indicative of a reject device, but instead, of an overtaxed test system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" HIGH and LOW levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels rather than at V_{IL} Max. and V_{IH} Min.






8. AC Testing

Occasionally, parameters are specified that cannot be measured directly on automatic testers because of tester limitations. Data input hold times often fall into this category. In these cases, the parameter in question is guaranteed by correlating these tests with other AC tests that have been performed. These correlations are arrived at by the cognizant engineer by using data from precise bench measurements in conjunction with the knowledge that certain DC parameters have already been measured and are within specification.

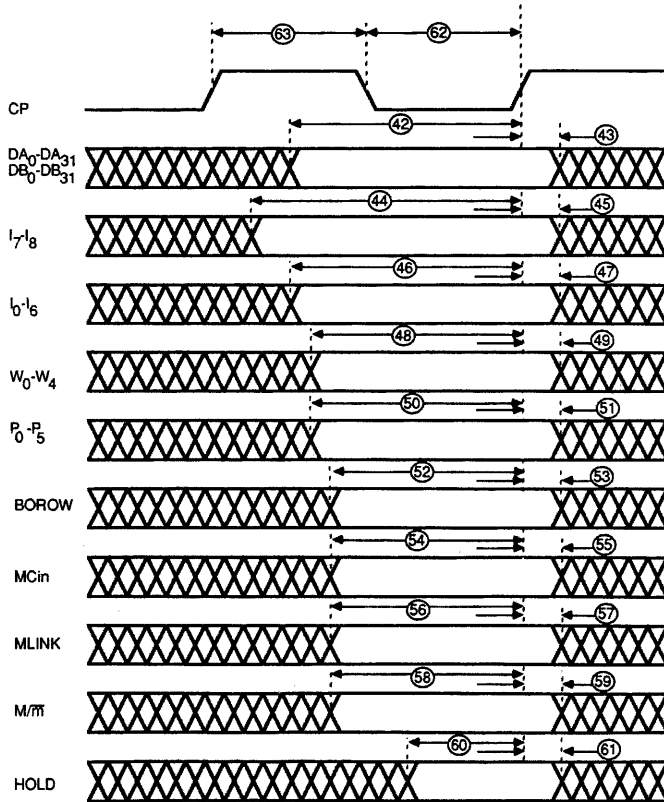
In some cases, certain AC tests are redundant since they can be shown to be predicted by other tests that have already been performed. In these cases, the redundant tests are not performed.

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

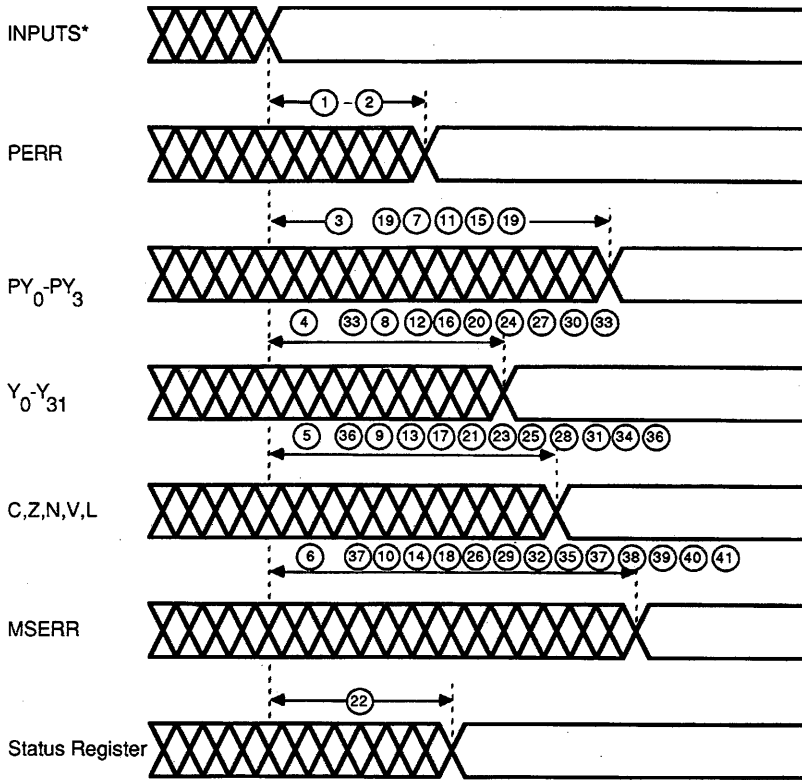
KS000010



WF023680

Setup and Hold Timing

SWITCHING WAVEFORMS (Cont'd.)

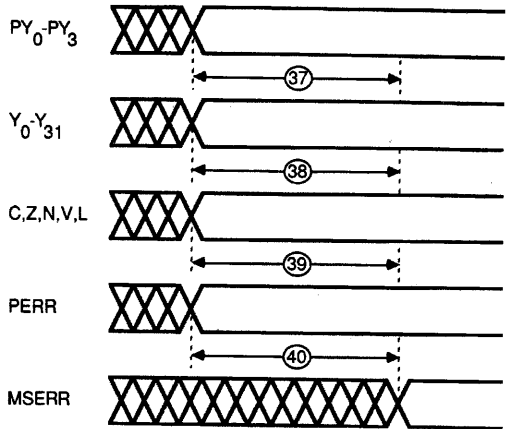


WF023691

Propagation Delays (SLAVE = LOW)

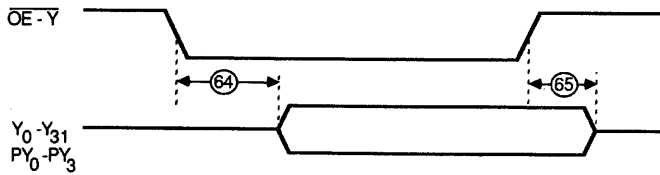
Inputs: PA₀-PA₃, PB₀-PB₃, DA₀-DA₃₁, DB₀-DB₃₁, I₀-I₈, W₀-W₄, P₀-P₅, CP, RS, MCin, MLINK, M/ \bar{m} , BOROW, HOLD

SWITCHING WAVEFORMS (Cont'd.)



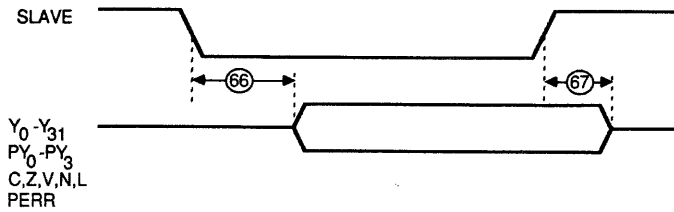
WF023700

Propagation Delay (SLAVE = HIGH)



WF023710

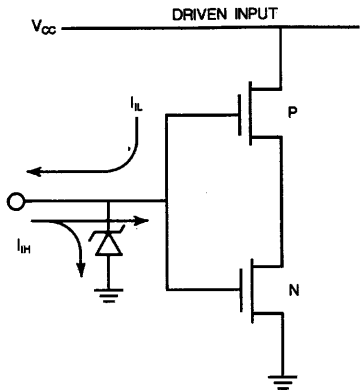
Enable/Disable I (SLAVE = HIGH)



WF023720

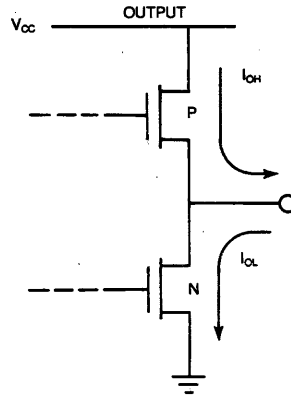
Enable/Disable II ($\overline{OE-Y} = \text{LOW}$)

INPUT/OUTPUT CIRCUIT DIAGRAMS



IC000861

$C_i \approx 5.0$ pF, all inputs



IC000871

$C_o \approx 5.0$ pF, all outputs

Am29C334

CMOS Four-Port Dual-Access Register File



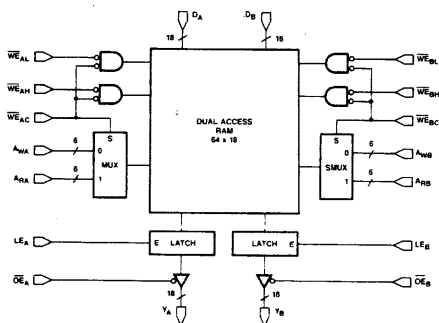
Am29C334

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

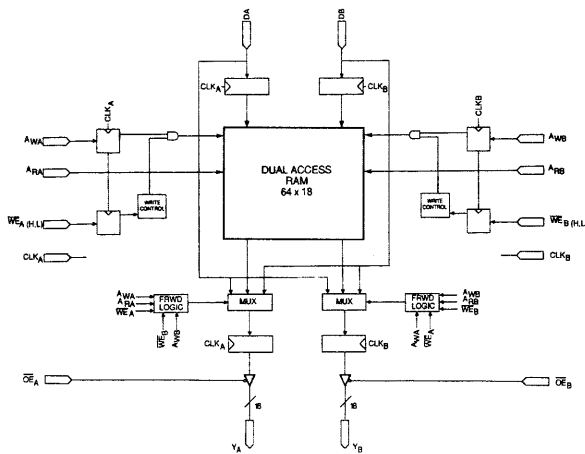
- 64 x 18 Bit Wide Register File**
 The Am29C334 is a 64 x 18-bit, dual-access RAM with two read ports and two write ports.
- Pipelined Data Path**
 The Am29C334 can be configured to support either a non-pipelined data path (similar to the Am29334) or a pipelined data path.
- Cascadable**
 The Am29C334 is cascadable to support either wider word widths, deeper register files, or both.
- Built in Forwarding Logic**
 The Am29C334 provides simultaneous read/write access to the same address for double pipelined systems.
- Byte Parity Storage**
 Width of 18 bits facilitates byte parity storage for each port and provides consistency with the Am29C332 32-bit ALU.
- Byte Write Capability**
 Individual byte-write enables allow byte or full word write.

BLOCK DIAGRAMS



BD003022

Non-Pipelined Mode



BD007021

Pipelined Mode

GENERAL DESCRIPTION

The Am29C334 is a 64-word by 18-bit dual-access RAM with two read ports and two write ports. Two independent, simultaneous accesses are possible and each access can be either a read or a write. It is designed to be used in a system that requires as many as two reads and two writes in a single cycle. The device can be configured to support either a non-pipelined data path or a pipelined data path.

The Am29C334 is also fully compatible with the bipolar Am29334. When the device is connected to the pinout specified for the Am29334, it will appear as a 64-word by 18-bit array without support for pipelined operation. The pipelined operation of the Am29C334 is made possible because of the availability of unused power pins not required by the CMOS part. The pipelined operation is disabled by attaching the PIPE pin to V_{CC} .

RELATED AMD PRODUCTS

Part No.	Description
Am29C323	CMOS 32-Bit Parallel Multiplier
Am29325	32-Bit Floating Point Processor
Am29C325	CMOS 32-Bit Floating Point Processor
Am29331	16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29332	32-Bit Extended Function ALU
Am29C332	CMOS 32-Bit Extended Function ALU
Am29334	64 x 18 Four-Port Dual-Access Register File
Am29337	16-Bit Bounds Checker
Am29338	128 x 9 Byte Queue

CONNECTION DIAGRAM

120 Lead PGA*

	A	B	C	D	E	F	G	H	J	K	L	M	N
1	AWA2	ARA2	AWA1	DA00	DA02	DA04	DA08	DA09	DA12	DA16	LEA	WEAC	WEAL
2	ARA3	AWA3	ARA1	ARA0	DA03	DA05	DA07	DA10	DA13	DA15	ARA5	AWA5	WEAH
3	AWA4	ARA4	YB00	AWA0	DA01	GND	DA06	PIPE	DA11	DA14	DA17	ARB4	AWB4
4	YB01	YB02	YB03								YA00	YA01	YA02
5	GND	YB04	YB05								YA03	YA04	GND
6	YB07	YB06	VCCA								OE \bar{A}	YA06	YA05
7	YB08	YB09	YB10								YA07	YA08	YA09
8	YB12	YB11	OE \bar{B}								VCCA	YA11	YA10
9	GND	YB13	YB14								YA12	YA13	GND
10	YB15	YB16	YB17								YA14	YA15	YA16
11	WE \bar{B} L	WE \bar{B} H	DB01	DB04	VCC	DB08	DB09	DB15	GND	ARB0	YA17	ARB3	AWB3
12	WE \bar{B} C	LEB	DB00	DB03	VCC	DB05	DB11	DB12	GND	DB17	AWB0	AWB2	ARB2
13	AWB5	ARB5	DB07	DB02	VCC	DB06	DB10	DB14	GND	DB16	DB13	ARB1	AWB1

CD010320

*Pins facing up.

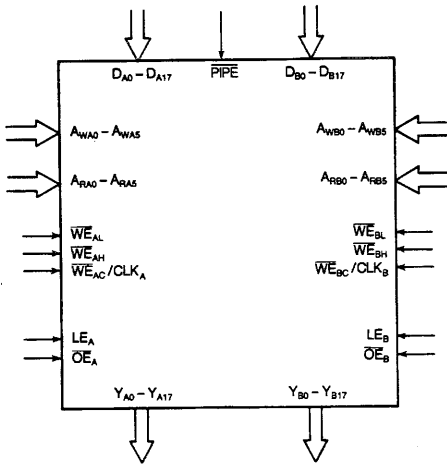
TABLE OF INTERCONNECTIONS
(Sorted by Pin Name)

PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.
			DA03	E02	65	DB16	K13	93	YA05	N06	21
			DA04	F01	6	DB17	K12	33	YA06	M06	81
			DA05	F02	66	GND	F03	8	YA07	L07	22
			DA06	G03	7	GND	J11	37	YA08	M07	82
			DA07	G02	67	GND	J12	38	YA09	N07	24
			DA08	G01	9	GND	J13	39	YA10	N08	84
ARA0	D02	63	DA09	H01	69	GND	N05	20	YA11	M08	25
ARA1	C02	62	DA10	H02	10	GND	N09	26	YA12	L09	85
ARA2	B01	61	DA11	J03	70	GND	A09	50	YA13	M09	86
ARA3	A02	120	DA12	J01	11	GND	A05	56	YA14	L10	27
ARA4	B03	119	DA13	J02	71	LEA	L01	14	YA15	M10	87
ARA5	L02	74	DA14	K03	12	LEB	B12	45	YA16	N10	28
ARB0	K11	92	DA15	K02	72	OE \bar{A}	L06	23	YA17	L11	88
ARB1	M13	91	DA16	K01	13	OE \bar{B}	C08	53	YB00	C03	118
ARB2	N12	90	DA17	L03	73	PIPE	H03	68	YB01	A04	58
ARB3	M11	89	DB00	C12	104	VCC	E11	97	YB02	B04	117
ARB4	N12	90	DB01	C11	44	VCC	E12	98	YB03	C04	57
ARB5	M03	77	DB02	D13	103	VCC	E13	99	YB04	B05	116
AWA0	D03	3	DB03	D12	43	VCCA	L08	83	YB05	C05	115
AWA1	C01	2	DB04	D11	102	VCCA	C06	113	YB06	B06	55
AWA2	A01	1	DB05	F12	42	WEAC/CLKA	M01	75	YB07	A06	114
AWA3	B02	60	DB06	F13	101	WEAH	N02	76	YB08	A07	54
AWA4	A03	59	DB07	G13	96	WEAL	N01	16	YB09	B07	112
AWA5	M02	15	DB08	G11	40	WEBC/CLKB	A12	106	YB10	C07	52
AWB0	L12	32	DB09	G11	40	WE \bar{B} H	B11	107	YB11	B08	111
AWB1	N13	31	DB10	G13	96	WE \bar{B} L	A11	47	YB12	A08	51
AWB2	M12	30	DB11	G12	36	YA00	L04	18	YB13	B09	110
AWB3	N11	29	DB12	H12	95	YA01	M04	78	YB14	C09	109
AWB5	N03	17	DB13	H13	35	YA02	N04	19	YB15	A10	49
DA00	D01	4	DB14	H13	94	YA03	L05	79	YB16	B10	108
DA01	E03	64	DB15	H11	34	YA04	M05	80	YB17	C10	48
DA02	E01	5									

TABLE OF INTERCONNECTIONS (Cont'd.)
(Sorted by Pin No.)

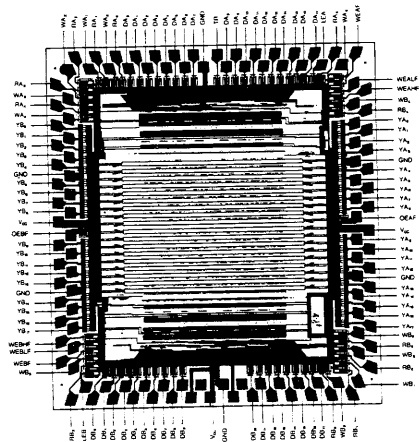
PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.
			C05	YB05	115	H02	DA10	10	M05	YA04	80
			C06	VCCA	113	H03	PIPE	68	M06	YA06	81
			C07	YB10	52	H11	DB15	34	M07	YA08	82
			C08	OE _B	53	H12	DB12	95	M08	YA11	25
A01	AWA2	1	C09	YB14	109	H13	DB14	94	M09	YA13	86
A02	ARA3	120	C10	YB17	48	J01	DA12	11	M10	YA15	87
A03	AWA4	59	C11	DB01	44	J02	DA13	71	M11	ARB3	89
A04	YB01	58	C12	DB00	104	J03	DA11	70	M12	AWB2	30
A05	GNDA	56	C13	DB07	41	J11	GND	37	M13	ARB1	91
A06	YB07	114	D01	DA00	4	J12	GND	38	N01	WE _{AL}	16
A07	YB08	54	D02	ARA0	63	J13	GND	39	N02	WE _{AH}	76
A08	YB12	51	D03	AWA0	3	K01	DA16	13	N03	AWB4	17
A09	GNDA	50	D11	DB04	102	K02	DA15	72	N04	YA02	19
A10	YB15	49	D12	DB03	43	K03	DA14	12	N05	GNDA	20
A11	WE _{BL}	47	D13	DB02	103	K11	ARB0	92	N06	YA05	21
A12	WE _{BC} /CLK _B	106	E01	DA02	5	K12	DB17	33	N07	YA09	24
A13	AWB5	46	E02	DA03	65	K13	DB16	93	N08	YA10	84
B01	ARA2	61	E03	DA01	64	L01	LE _A	14	N09	GNDA	26
B02	AWA3	60	E11	VCC	97	L02	ARA5	74	N10	YA16	28
B03	ARA4	119	E12	VCC	98	L03	DA17	73	N11	AWB3	29
B04	YB02	117	E13	VCC	99	L04	YA00	18	N12	ARB2	90
B05	YB04	116	F01	DA04	6	L05	YA03	79	N13	AWB1	31
B06	YB06	55	F02	DA05	66	L06	OE _A	23			
B07	YB09	112	F03	GND	8	L07	YA07	22			
B08	YB11	111	F11	DB08	100	L08	VCCA	83			
B09	YB13	110	F12	DB05	42	L09	YA12	85			
B10	YB16	108	F13	DB06	101	L10	YA14	27			
B11	WE _{BH}	107	G01	DA08	9	L11	YA17	88			
B12	LE _B	45	G02	DA07	67	L12	AW _{B0}	32			
B13	ARB5	105	G03	DA06	7	L13	D _{B13}	35			
C01	AWA1	2	G11	DB09	40	M01	WE _{AC} /CLK _A	75			
C02	ARA1	62	G12	DB11	36	M02	AW _{A5}	15			
C03	YB00	118	G13	DB10	96	M03	ARB4	77			
C04	YB03	57	H01	DA09	69	M04	YA01	78			

LOGIC SYMBOL



LS00222

METALLIZATION AND PAD LAYOUT

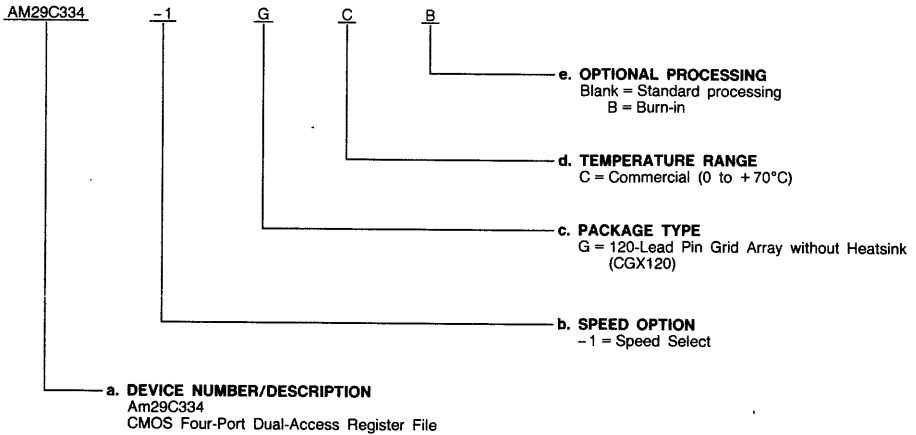


ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Package Type**
- d. **Temperature Range**
- e. **Optional Processing**



Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

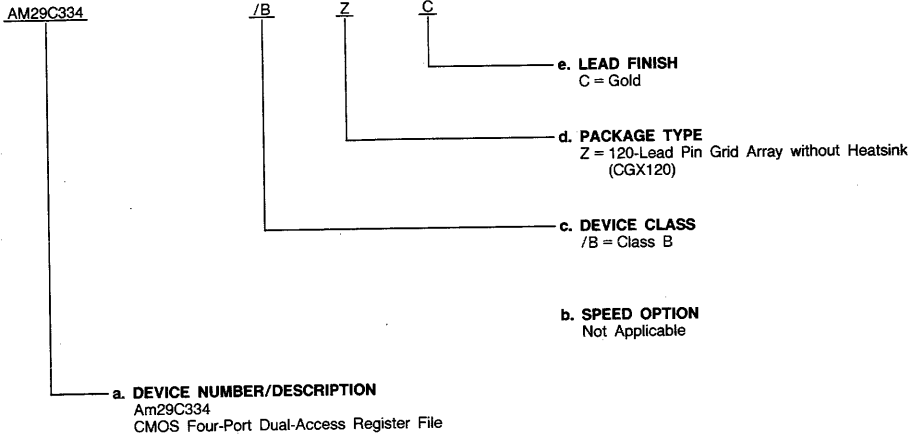
Valid Combinations	
AM29C334	GC, GCB
AM29C334-1	

ORDERING INFORMATION (Cont'd.)

APL Products

AMD products for Aerospace and Defense applications are available in several packages and operating ranges. APL (Approved Products List) products are fully compliant with MIL-STD-883C requirements. The order number (Valid Combination) for APL products is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Device Class**
- d. **Package Type**
- e. **Lead Finish**



Valid Combinations	
AM29C334	/BZC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations or to check for newly released valid combinations.

Group A Tests

Group A tests consist of Subgroups
1, 2, 3, 7, 8, 9, 10, 11.

PIN DESCRIPTION

ARA0 - ARA5 Read Address A-Side (Input)

The 6-bit read address input selects one of the 64 memory locations for output to the Y_A Data Latch.

ARB0 - ARB5 Read Address B-Side (Input)

The 6-bit read address input selects one of the 64 memory locations for output to the Y_B Data Latch.

AWA0 - AWA5 Write Address A-Side (Input)

The 6-bit write address input selects one of the 64 memory locations for writing new data from the D_A input.

AWB0 - AWB5 Write Address B-Side (Input)

The 6-bit write address input selects one of the 64 memory locations for writing new data from the D_B input.

DA0 - DA17 Data A-Side (Input)

New data is written into memory from this input, as selected by the A_{WA} address input.

DB0 - DB17 Data B-Side (Input)

New data is written into memory from this input, as selected by the A_{WB} address input.

GND, VCC Power

Power supply for the internal logic (0, 5 V).

GND_A, VCC_A Power

Power supply for the output drivers (0, 5 V).

LE_A Y_A Data Latch Enable (Input, Active HIGH)

The LE_A input controls the latch for the Y_A output port. When LE_A is HIGH, the latch is open (transparent) and data from the RAM, as selected by the A_{RA} address inputs, is passed to the Y_A output. When LE_A is LOW, the latch is closed and it retains the last data read from the RAM. LE_A is disabled in the pipelined mode.

LE_B Y_B Data Latch Enable (Input, Active HIGH)

The LE_B input controls the latch for the Y_B output port. When LE_B is HIGH, the latch is open (transparent), and data from the RAM, as selected by the A_{RB} address inputs, is passed to the Y_B output. When LE_B is LOW, the latch is closed and it retains the last data read from the RAM. LE_B is disabled in the pipelined mode.

OE_A Y_A Output Enable (Input, Active LOW)

When OE_A is LOW, data in the Y_A Data Latch is driven on the Y_A output. When OE_A is HIGH, Y_A output is in the high-impedance (off) state.

OE_B Y_B Output Enable (Input, Active LOW)

When OE_B is LOW, data in the Y_B Data Latch is driven on the Y_B outputs. When OE_B is HIGH, Y_B output is in the high-impedance (off) state.

PIPE Pipeline Enable (Input, Active LOW)

When $PIPE$ is LOW, the input and output registers are enabled, allowing for pipelined operation. When HIGH, these registers are made transparent.

WE_{AC}/CLK_A Write Enable A-Side Common (Input, Active LOW)

When WE_{AC} is LOW together with WE_{AH} or WE_{AL} , new data is written into the location selected by the A_{WA} address. When WE_{AC} is HIGH, no data is written into the RAM through the A port. WE_{AC} acts as a clock input in the pipeline mode for the A side.

WE_{BC}/CLK_B Write Enable B-Side Common (Input, Active LOW)

When WE_{BC} is LOW together with WE_{BH} or WE_{BL} , new data is written into the location selected by the A_{WB} address. When WE_{BC} is HIGH, no data is written into the RAM through the B port. WE_{BC} acts as a clock input in the pipeline mode for the B side.

WE_{AH} High-Byte Write Enable A-Side (Input, Active LOW)

When WE_{AH} is LOW together with WE_{AC} , new data is written into the high byte of the location selected by the A_{WA} address input. When WE_{AH} is HIGH, no data is written into the high byte.

WE_{BH} High-Byte Write Enable B-Side (Input, Active LOW)

When WE_{BH} is LOW together with WE_{BC} , new data is written into the high byte of the location selected by the A_{WB} address input. When WE_{BH} is HIGH, no data is written into the high byte.

WE_{AL} Low-Byte Write Enable A-Side (Input, Active LOW)

When WE_{AL} is LOW together with WE_{AC} , new data is written into the low byte of the location selected by the A_{WA} address input. When WE_{AL} is HIGH, no data is written into the low byte.

WE_{BL} Low-Byte Write Enable B-Side (Input, Active LOW)

When WE_{BL} is LOW together with WE_{BC} , new data is written into the low byte of the location selected by the A_{WB} address input. When WE_{BL} is HIGH, no data is written into the low byte.

YA0 - YA17 Data Latch (Outputs, Three-State)

The 18-bit Y_A Data Latch outputs.

YB0 - YB17 Data Latch (Outputs, Three-State)

The 18-bit Y_B Data Latch outputs.

FUNCTIONAL DESCRIPTION

The heart of the Am29C334 is a high-speed 64-word by 18-bit dual RAM cell array. Six write enables permit the RAM word to be written in one or both of its 9-bit bytes. Data to be written is presented to each side of the RAM array through the two data ports (D_A and D_B).

The remainder of the logic surrounding the RAM array supports pipelining the RAM access and providing a forwarding path for data around the RAM. This forwarding path is needed to eliminate the latency cycle associated with consecutive write/read accesses to the same memory location in a pipelined system.

Pipelining of the RAM is controlled by the \overline{PIPE} pin. When not asserted (i.e., in non-pipelined mode) the registers on the inputs (write ports $D_{A/B}$, write addresses $A_{WA/B}$, and write enables $\overline{WE}_{AC/BC}$) are made fully transparent, while the registers at the outputs (the read ports $Y_{A/B}$) are turned into latches, controlled by the latch enables $LE_{A/B}$.

In either mode of operation, each side of the RAM is controlled by its individual control signals. This means that the two sides of the RAM can operate at different clock rates to one

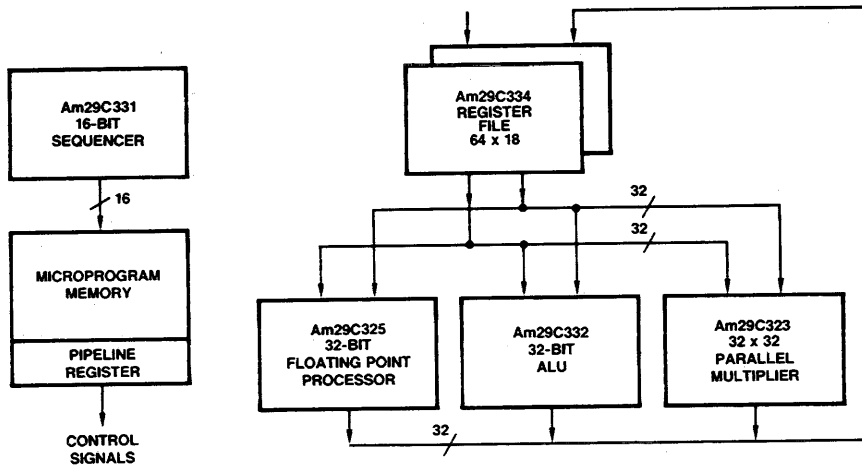
another. In the pipelined mode, these clock rates must have a known relationship between each other.

In the non-pipelined mode, there is no need for a relationship between the clock rates. Two special cases of operation arise because of this. The first is where the location written to by one side is being read from the other side. In this case, known as A-to-B transparency, the value read is the value being written. The second occurs when two writes to the same location occur at the same time. In this case the value written can not be defined, but the operation is not harmful to the device.

The transparency mode (A-A or B-B) during a write ($\overline{WE}_A = \text{LOW}$) allows the data in (D_A) to not only be written into memory, but also to appear at the output (Y_A) when the output latch (LE_A) is HIGH and the output enable control (\overline{OE}_A) is LOW.

Extensions to Four Read Ports and Two Write Ports

A RAM with four read ports and two write ports can be made by using two dual-access RAMs and connecting each of the write ports, write addresses, and write enables in parallel for the two devices. Figure 2 details this in a non-pipelined mode.



AF003482

Figure 1. Am29C300 CMOS Family High-Performance System Block Diagram

32 Word x 36 Bit Single-Access RAM

It is possible to convert the 64 word x 18 bit dual-access RAM into a 32 word x 36 bit single-access RAM. This is performed by storing the upper half of the 36 bits in the upper half of the 64 words and addressing these from the A side, and storing the lower half of the 36 bits in the lower half of the 64 words and addressing these from the B side. This arrangement does not change the capacity of the RAM, but the dual access is lost (see Figure 4).

Operational Modes

The Am29C334 may be configured in a non-pipelined mode or in a pipelined mode by controlling the \overline{PIPE} pin. This mode is selected via hardwiring the pin to either LOW or HIGH. This option should not be changed during operation.

Non-Pipelined Data Path

In non-pipelined mode ($\overline{PIPE} = 1$), the Am29C334 is a flow-through device; data is read out, used, and written back all in the same cycle. In this mode all the registers are made transparent except the registers at the two read ports that are configured as latches. The read port latches are controlled individually by the LE_A and LE_B , so that they are transparent when the latch enables are HIGH and retain the data when the latch enables are LOW. The "forwarding logic" incorporated to support the pipelined mode of operation is also disabled in this mode of operation (specifically, the address comparators are disabled).

In the non-pipelined mode of operation it is possible to simultaneously read two ports, read one port and write to the other, or write to two ports, concurrently. The read and write

addresses are internally multiplexed on each side. The selection of the read and write addresses is controlled by the exclusive-OR of the PIPE pin and $\overline{WE}_{AC/BC}$. Normally, the $\overline{WE}_{AC/BC}$ are connected to the system clock. With PIPE deasserted, the read address will be selected in the high part of the clock cycle ($\overline{WE}_{AC/BC} = 1$) and the write address selected only in the low part. Byte selection for writing on either ports is controlled by the $\overline{WE}_{H/L}$ pins.

Two interesting cases arise as a result of the dual access capability. The first occurs if a location is written into by one side while it is being read out by the other side. In this case, known as A-to-B transparency, the data being written will appear on the read port after the Transparency_{AB} time (if other read access time parameters are met). The second case of interest occurs if both sides write to the same location at the same time. The value written as a result of this operation cannot be defined.

Pipelined Data Path

The Am29C334 can be configured in a pipelined system by asserting the PIPE signal (PIPE = 0) and adding an additional external register in the write address and the write control path on both A and B ports as shown in Figure 3. The registers on each side are controlled by separate clocks that are supplied over the \overline{WE}_{AC} and \overline{WE}_{BC} pins.

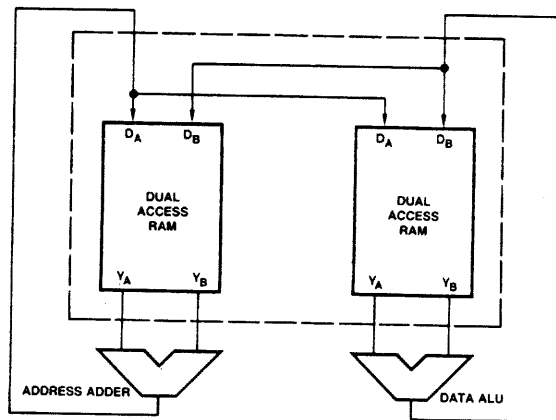
Typically, in a pipelined system a read - modify - write would span three cycles. In the second half of the first cycle, a read of the operand(s) is performed and the data is clocked into the output registers at the end of the cycle. In the second cycle, the operation is performed on the operands and the result is clocked into the data register on the write port at the end of the second cycle. In the first half of the third cycle, the data is written to the register file. Therefore, in any cycle, a pipelined system is writing the result of instruction n (in the first half),

executing instruction n + 1, and reading the operands needed in instruction n + 2. In any case, a write operation followed by a read operation is performed in the RAM in a cycle.

A special case arises if the data to be written by the previous instruction is needed in the next instruction as an operand. Due to the pipeline register being at its write port, the location is not written into until the next cycle, and hence only the previous value is available in the current cycle. To overcome this problem, "forwarding logic" is included as shown in the block diagram. This logic consists of three elements: an address comparator, an AND gate, and a three-to-one multiplexer, as shown. If the read address of the current instruction is the same as the write address of the previous instruction, and if the result is to be written, then the data to be written is forwarded by the forwarding multiplexer to the output registers. Since there are two write ports, forwarding paths on both ports are provided. As each write port has byte write capability, the forwarding is further broken into the upper and lower bytes.

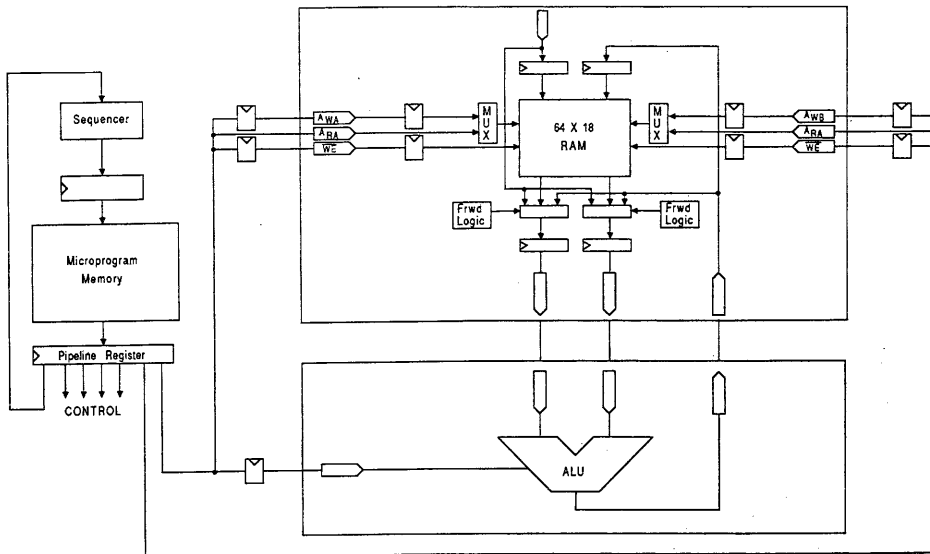
Since each side has its own \overline{WE}_{C}/CLK control, it is possible to clock each side of the chip differently. However, if the part is used at different frequencies, the forwarding cannot be guaranteed unless the addresses compared are held valid long enough to allow for a comparison to be made and the results of the forwarding setup on the output register.

As mentioned earlier, it is necessary to use an external write address and write control registers in a pipelined system. These registers have not been included for two reasons. First, it is possible for the user to abort the writing before it fills the internal pipe. This situation may arise in cases such as in "traps." Second, by providing an external write address register it provides the flexibility of obtaining the write address from several sources by using an external multiplexer.



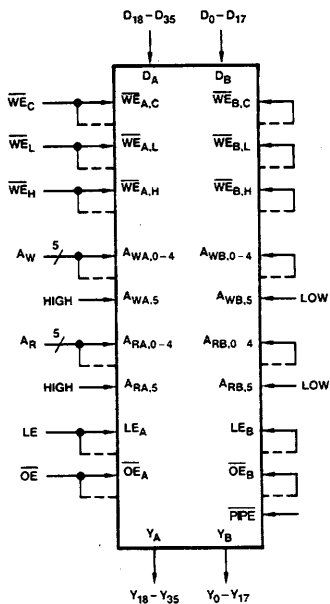
AF003490

Figure 2. RAM with Four Read Ports and Two Write Ports for Non-pipelined Mode



BD006880

Figure 3. System Diagram With the Am29C334 in a Double Pipelined Data Path



LS001791

Figure 4. 32 x 36 RAM (Single Access) Using 64 x 18 Dual-Access RAM

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Temperature Under Bias - T _C	-55 to +125°C
Supply Voltage to Ground Potential	
Continuous	-0.3 to +7.0 V
DC Voltage Applied to Outputs	
for HIGH Output State	-0.3 V to +V _{CC} + 0.3 V
DC Input Voltage	-0.3 V to +V _{CC} + 0.3 V
DC Output Current, Into LOW Outputs	30 mA
DC Input Current	-10 mA to +10 mA

Stresses above those listed under **ABSOLUTE MAXIMUM RATINGS** may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature (T _A)	0 to +70°C
Supply Voltage	+4.75 to +5.25 V
Military* (M) Devices	
Temperature (T _A)	-55 to +125°C
Supply Voltage (V _{CC})	+4.5 to +5.5 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

* Military product 100% tested at T_A = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified (for APL Products, Group A, Subgroups 1, 2, 3 are tested unless otherwise noted)

Parameter Symbol	Parameter Description	Test Conditions (Note 1)	Min.	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH} I _{OH} = -4 mA	2.4		Volts
V _{OL}	Output LOW Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH} I _{OL} = 8 mA		0.5	Volts
V _{IH}	Input HIGH Level	Guaranteed Input Logical HIGH Voltage (Note 2)	2.0		Volts
V _{IL}	Input LOW Level	Guaranteed Input Logical LOW Voltage (Note 2)		0.8	Volts
I _{IL}	Input LOW Current	V _{CC} = Max. V _{IN} = 0.5 V		-10	μA
I _{IH}	Input HIGH Current	V _{CC} = Max. V _{IN} = V _{CC} - 0.5 V		10	μA
I _{OZH}	Off State (High-Impedance) Output Current:	V _{CC} = Max.	V _O = 2.4 V	10	μA
I _{OZL}			V _O = 0.5 V	-10	
I _{CC}	Static Power Supply Current	V _{IN} = V _{CC} or GND V _{CC} = Max I _O = 0 μA	T _A = -55 to 125°C	80	mA
			T _A = 0 to +70°C	70	mA
C _{PD}	Power Dissipation Capacitance (Note 3)	V _{CC} = 5.0 V T _A = 25°C No Load	900 pF Typical		

- Notes: 1. V_{CC} conditions shown as Min. or Max. refer to the commercial (±5%) V_{CC} limits.
 2. These input levels provide zero-noise immunity and should only be statically tested in a noise-free environment (not functionally tested).
 3. C_{PD} determines the no-load dynamic current consumption:
 I_{CC (Total)} = I_{CC (Static)} + C_{PD} V_{CC} f, where f is the switching frequency of the majority of the internal nodes, normally one-half of the clock frequency. This specification is not tested.

SWITCHING CHARACTERISTICS over **COMMERCIAL** operating range unless otherwise specified

NON-PIPELINED MODE (Note 1)

No.	Parameter	Description	Test Conditions	29C334		29C334-1		29C334-2		Unit
				Min.	Max.	Min.	Max.	Min.	Max.	
1	Access Time	A_{RA} or A_{RB} to Y_A or Y_B	LE_A or $LE_B = H$		32		26		21	ns
2	Access Time	\overline{WE}_{AC} or \overline{WE}_{BC} to Y_A or Y_B	LE_A or $LE_B = H$		30		25		20	ns
3	Turn-On Time	\overline{OE}_A or $\overline{OE}_B \downarrow$ to Y_A or Y_B Active		0	20	0	16	0	16	ns
4	Turn-Off Time (Note 2)	\overline{OE}_A or $\overline{OE}_B \uparrow$ to Y_A or $Y_B = \text{High Impedance}$		0	20	0	16	0	16	ns
5	Enable Time	LE_A or $LE_B \uparrow$ to Y_A or Y_B		0	16	0	14	0	14	ns
6	Transparency	\overline{WE}_A or $\overline{WE}_B \uparrow$ to Y_A or Y_B	LE_A or $LE_B = H$		39		33		27	ns
7	Transparency	D_A or D_B to Y_A or Y_B	LE_A or $LE_B = H$, \overline{WE}_A or $\overline{WE}_B = L$		39		33		27	ns
8	Write Recovery Time	A_{RA} or A_{RB} to \overline{WE}_{AC} or \overline{WE}_{BC}			(2)-1		(2)-1		(2)-1	ns
9	Data Setup Time	D_A or D_B to \overline{WE}_A or $\overline{WE}_B \uparrow$		15		13		13		ns
10	Data Hold Time	D_A or D_B to \overline{WE}_A or $\overline{WE}_B \downarrow$		0		0		0		ns
11	Address Setup Time	A_{WA} or A_{WB} to \overline{WE}_A or $\overline{WE}_B \uparrow$		2		2		2		ns
12	Address Hold Time	A_{WA} or A_{WB} to \overline{WE}_A or $\overline{WE}_B \downarrow$		1		1		1		ns
13	Address Setup Time	A_{RA} or A_{RB} to LE_A or $LE_B \downarrow$		20		17		17		ns
14	Address Hold Time	A_{RA} or A_{RB} to LE_A or $LE_B \downarrow$		1		1		1		ns
15	Latch Close Before Write	LE_A or LE_B to \overline{WE}_A or $\overline{WE}_B \downarrow$		0		0		0		ns
16	Read Before Latch Close	\overline{WE}_{AC} or \overline{WE}_{BC} to LE_A or $LE_B \downarrow$		20		16		16		ns
17	Write Pulse Width	\overline{WE}_A or \overline{WE}_B (LOW)		20		16		16		ns
18	Latch Data Capture Pulse Width	LE_A or LE_B (HIGH)		14		12		12		ns

Notes: See notes following Military table.

SWITCHING CHARACTERISTICS over **MILITARY** operating range unless otherwise specified (for APL Products, Group A, Subgroups 9, 10, 11 are tested unless otherwise noted)

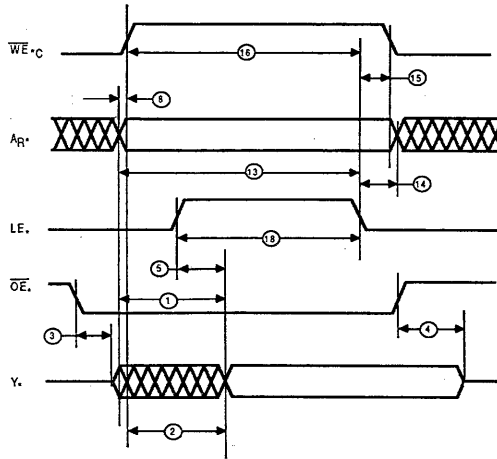
NON-PIPELINED MODE (Note 1)

No.	Parameter	Description	Test Conditions	29C334		Unit
				Min.	Max.	
1	Access Time	A_{RA} or A_{RB} to Y_A or Y_B	LE_A or $LE_B = H$		40	ns
2	Access Time	\overline{WE}_{AC} or \overline{WE}_{BC} to Y_A or Y_B	LE_A or $LE_B = H$	0	37	ns
3	Turn-On Time	\overline{OE}_A or $\overline{OE}_B \downarrow$ to Y_A or Y_B Active		0	16	ns
4	Turn-Off Time (Note 2)	\overline{OE}_A or $\overline{OE}_B \uparrow$ to Y_A or $Y_B = \text{High Impedance}$		0	25	ns
5	Enable Time	LE_A or $LE_B \uparrow$ to Y_A or Y_B		0	21	ns
6	Transparency	\overline{WE}_A or $\overline{WE}_B \downarrow$ to Y_A or Y_B	LE_A or $LE_B = H$	0	47	ns
7	Transparency	D_A or D_B to Y_A or Y_B	LE_A or $LE_B = H$, \overline{WE}_A or $\overline{WE}_B = L$		47	ns
8	Write Recovery Time	A_{RA} or A_{RB} to \overline{WE}_{AC} or \overline{WE}_{BC}			(2)-(1)	ns
9	Data Setup Time	D_A or D_B to \overline{WE}_A or $\overline{WE}_B \uparrow$		19		ns
10	Data Hold Time	D_A or D_B to \overline{WE}_A or $\overline{WE}_B \uparrow$		2		ns
11	Address Setup Time	A_{WA} or A_{WB} to \overline{WE}_A or $\overline{WE}_B \downarrow$		4		ns
12	Address Hold Time	A_{WA} or A_{WB} to \overline{WE}_A or $\overline{WE}_B \uparrow$		2		ns
13	Address Setup Time	A_{RA} or A_{RB} to LE_A or $LE_B \downarrow$		23		ns
14	Address Hold Time	A_{RA} or A_{RB} to LE_A or $LE_B \downarrow$		1		ns
15	Latch Close Before Write	LE_A or LE_B to \overline{WE}_A or $\overline{WE}_B \downarrow$		0		ns
16	Read Before Latch Close	\overline{WE}_{AC} or \overline{WE}_{BC} to LE_A or $LE_B \downarrow$		24		ns
17	Write Pulse Width	\overline{WE}_A or \overline{WE}_B (LOW)		23		
18	Latch Data Capture Pulse Width	LE_A or LE_B (HIGH)		17		ns

- Notes: 1. $\overline{WE}_A = \overline{WE}_{AC} + \overline{WE}_{AL}/H$
 $\overline{WE}_B = \overline{WE}_{BC} + \overline{WE}_{BL}/H$
 2. Y_A and Y_B are tested independently.
 3. Minimum delays are not tested.

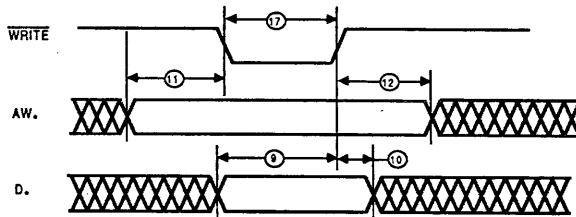
SWITCHING WAVEFORMS

NON-PIPELINED MODE



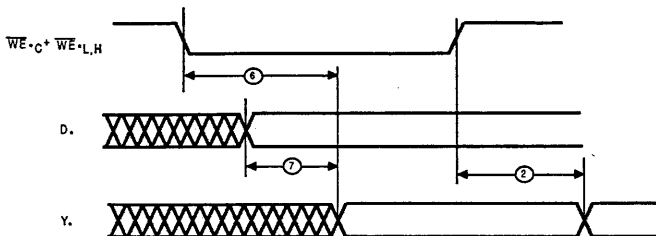
WF023330

Read Function (* means A or B)



WF023340

Write Function (* means A or B)



WF023320

Transparency

NOTE: \overline{LE}_A = HIGH
 \overline{OE}_A = LOW
 * means A or B

SWITCHING CHARACTERISTICS over **COMMERCIAL** operating range (Cont'd.)

PIPELINED MODE

No.	Parameter	Description	29C334		29C334-1		29C334-2		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	
19	Write Data Setup Time	D_A or D_B to CLK_A or CLK_B †	15		13		13		ns
20	Write Data Hold Time	D_A or D_B to CLK_A or CLK_B †	1		1		1		ns
21	Write Address Setup Time	A_{WA} or A_{WB} to CLK_A or CLK_B †	23		20		20		ns
22	Write Address Hold Time	A_{WA} or A_{WB} to CLK_A or CLK_B †	0		0		0		ns
23	Write Enable Setup Time	\overline{WE}_H or \overline{WE}_L to CLK_A or CLK_B †	20		16		16		ns
24	Write Enable Hold Time	\overline{WE}_H or \overline{WE}_L to CLK_A or CLK_B †	0		0		0		ns
25	Read Address Setup Time	A_{RA} or A_{RB} to CLK_A or CLK_B †	24		20		20		ns
26	Read Address Hold Time	A_{RA} or A_{RB} to CLK_A or CLK_B †	0		0		0		ns
27	Minimum Clock Cycle	CLK_A or CLK_B (LOW)	50		40		40		ns
28	Minimum Clock Pulse	CLK_A or CLK_B (HIGH)	17		14		14		ns
29	Minimum Clock Pulse	CLK_A or CLK_B (LOW)	17		14		14		ns
30	Clock to Y	Y_A or Y_B to CLK_A or CLK_B	14		12		10		ns

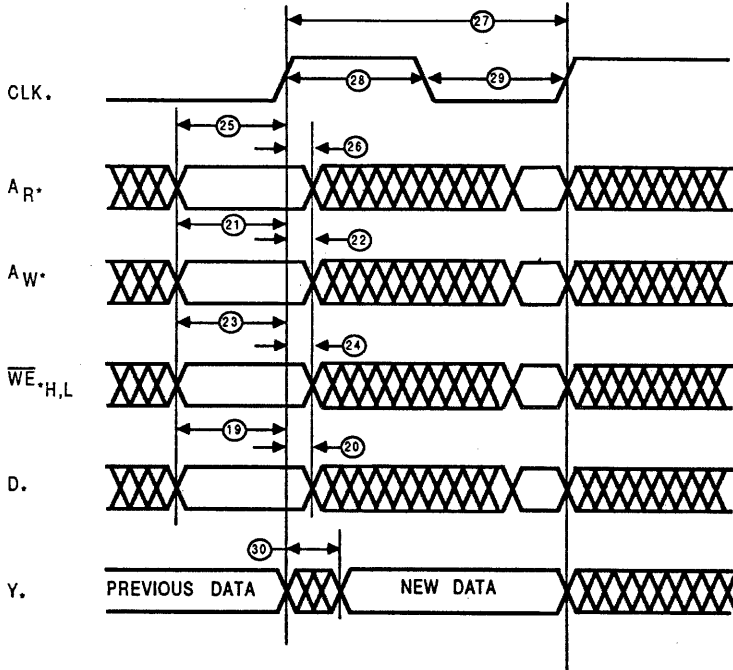
SWITCHING CHARACTERISTICS over **MILITARY** operating range (Cont'd.)

PIPELINED MODE

No.	Parameter	Description	29C334		Unit
			Min.	Max.	
19	Write Data Setup Time	D_A or D_B to CLK_A or CLK_B ↑	19		ns
20	Write Data Hold Time	D_A or D_B to CLK_A or CLK_B ↑	2		ns
21	Write Address Setup Time	A_{WA} or A_{WB} to CLK_A or CLK_B ↑	27		ns
22	Write Address Hold Time	A_{WA} or A_{WB} to CLK_A or CLK_B ↑	2		ns
23	Write Enable Setup Time	\overline{WE}_H or \overline{WE}_L to CLK_A or CLK_B ↑	23		ns
24	Write Enable Hold Time	\overline{WE}_H or \overline{WE}_L to CLK_A or CLK_B ↑	2		ns
25	Read Address Setup Time	A_{RA} or A_{RB} to CLK_A or CLK_B ↑	28		ns
26	Read Address Hold Time	A_{RA} or A_{RB} to CLK_A or CLK_B ↑	0		ns
27	Minimum Clock Cycle	CLK_A or CLK_B (LOW)	55		ns
28	Minimum Clock Pulse	CLK_A or CLK_B (HIGH)	20		ns
29	Minimum Clock Pulse	CLK_A or CLK_B (LOW)	20		ns
30	Clock to Y	Y_A or Y_B to CLK_A or CLK_B	18		ns

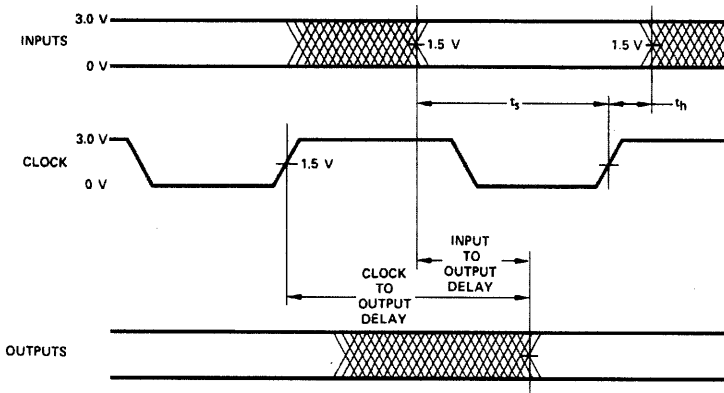
SWITCHING WAVEFORMS (Cont'd.)

PIPELINED MODE



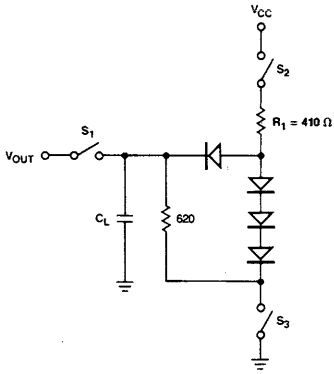
WF023310

* means A or B



WFR02990

SWITCHING TEST CIRCUIT



TC003424

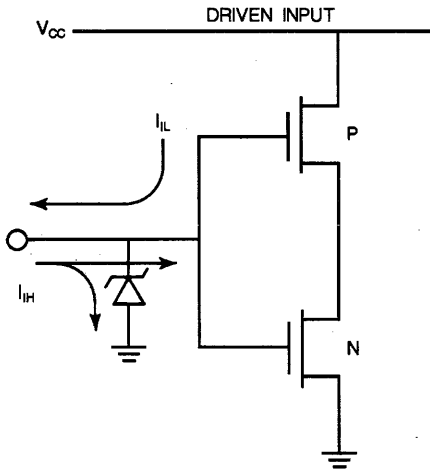
- Notes:**
1. $C_L = 50\text{pF}$ includes scope probe, wiring and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during functions tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test. S_1 and S_2 are closed while S_3 is open for t_{pZL} test.
 4. $C_L = \text{TBD}$ for output disable tests.

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

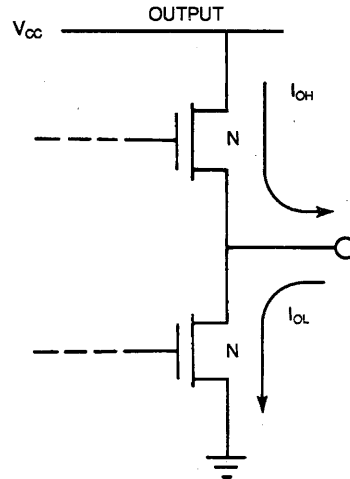
KS000010

INPUT/OUTPUT CIRCUIT DIAGRAMS



IC000861

$C_i \approx 5.0 \text{ pF}$, all inputs



IC000870

$C_o \approx 5.0 \text{ pF}$, all outputs

Am29C325

CMOS 32-Bit Floating-Point Processor



Am29C325

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- Single VLSI device performs high-speed floating-point arithmetic
 - Floating-point addition, subtraction, and multiplication in a single clock cycle
 - Internal architecture supports sum-of-products, Newton-Raphson division
- 32-bit, three-bus flow-through architecture
 - Programmable I/O allows interface to 32- and 16-bit systems
- IEEE and DEC formats
 - Performs conversions between formats
 - Performs integer ↔ floating-point conversions
- Input and output registers can be made transparent independently
- Pin and functionally compatible with the Bipolar Am29325
- The Am29C325 uses less than one-quarter the power of the Am29325
- 145 PGA requires no heatsink

GENERAL DESCRIPTION

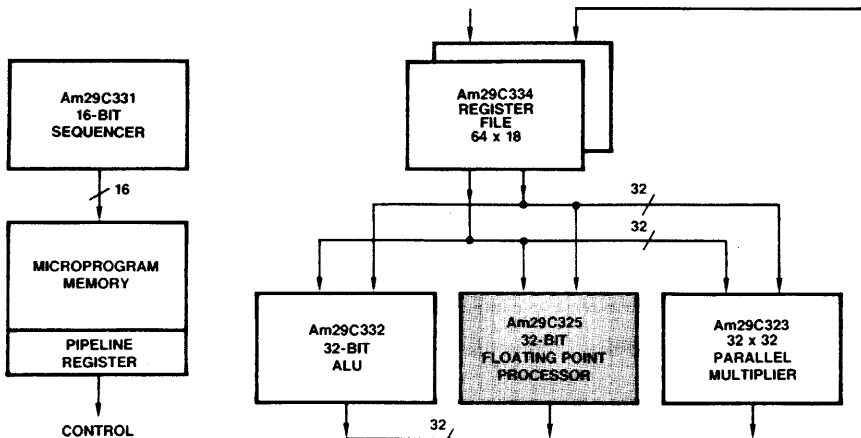
The Am29C325 is a high-speed floating-point processor unit. It performs 32-bit single-precision floating-point addition, subtraction, and multiplication operations in a single VLSI circuit, using the format specified by the proposed IEEE floating-point standard, 754. The DEC single-precision floating-point format is also supported. Operations for conversion between 32-bit integer format and floating-point format are available, as are operations for converting between the IEEE and DEC floating-point formats. Any operation can be performed in a single clock cycle. Six flags — invalid operation, inexact result, zero, not-a-number, overflow, and underflow — monitor the status of operations.

The Am29C325 has a three-bus, 32-bit architecture, with two input buses and one output bus. This configuration

provides high I/O bandwidth, allows access to all buses, and affords a high degree of flexibility when connecting this device in a system. All buses are registered, with each register having a clock enable. Input and output registers may be made transparent independently. Two other I/O configurations, a 32-bit, two-bus architecture and a 16-bit, three-bus architecture, are user-selectable, easing interface with a wide variety of systems. Thirty-two-bit internal feedforward datapaths support accumulation operations, including sum-of-products and Newton-Raphson division.

Fabricated using Advanced Micro Devices' 1.2 micron CMOS process, the Am29C325 is powered by a single 5-volt supply. The device is housed in a 145-lead pin-grid-array package.

Am29C300 FAMILY HIGH-PERFORMANCE SYSTEM BLOCK DIAGRAM



AF004651

Am29C327

CMOS Double-Precision Floating-Point Processor



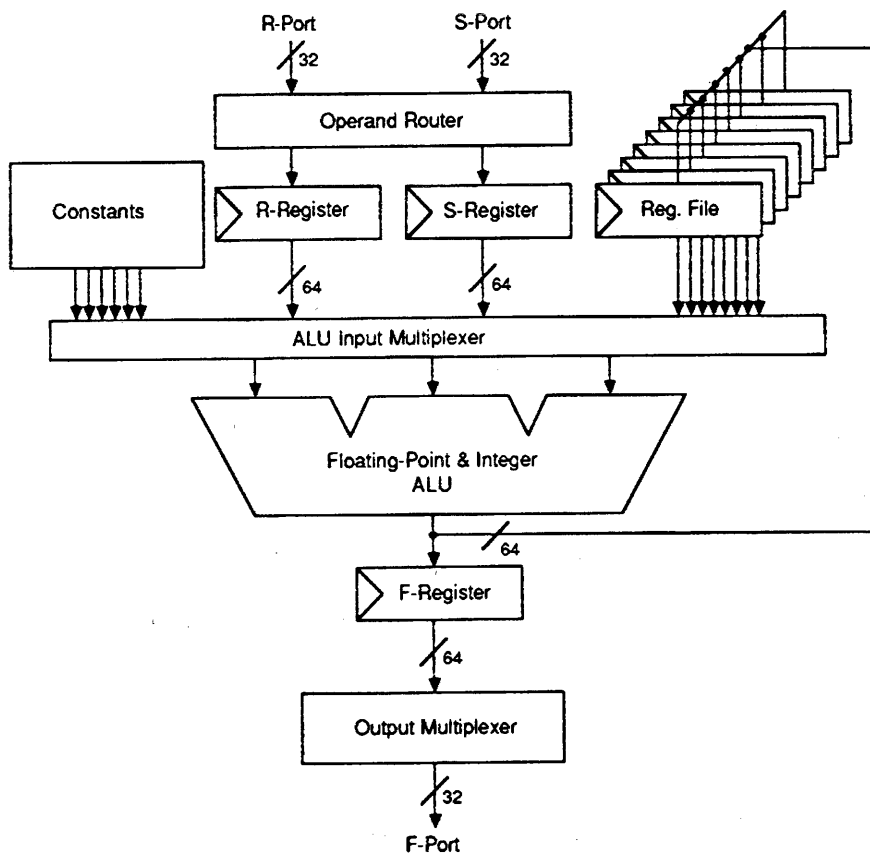
ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- High-performance double-precision floating-point processor
- Comprehensive floating-point and integer instruction sets
- Single VLSI device performs single-, double-, and mixed-precision operations
- Performs conversions between precisions and between data formats
- Compatible with industry-standard floating-point formats
 - IEEE 754 format
 - DEC F, DEC D, and DEC G formats
 - IBM system/370 format
- Exact IEEE compliance for denormalized numbers with no speed penalty
- Eight-deep register file for intermediate results and on-chip 64-bit data path facilitates compound operations; e.g., Newton-Raphson division, sum-of-products, and transcendentals
- Supports pipelined or flow-through operation
- Fabricated with Advanced Micro Devices' 1.2 micron CMOS process

AM29C327

SIMPLIFIED SYSTEM DIAGRAM



BD007470

CHAPTER 3

Bipolar Family

Am29331 16-Bit Microprogram Sequencer	3-1
Am29332 32-Bit Arithmetic Logic Unit	3-36
Am29334 Four-Port Dual-Access Register File	3-74
Am29434 ECL Four-Port, Dual-Access Register File	3-89
Am29325 32-Bit Floating-Point Processor*	3-103
Am29337 16-Bit Bounds Checker	3-104
Am29338 32-Bit Byte Queue	3-115

* Front page only of data sheet. See Chapter 4 for complete data sheet.

Am29331

16-Bit Microprogram Sequencer



Am29331

DISTINCTIVE CHARACTERISTICS

- **16-Bits Address up to 64K Words**
Supports 80-90 ns microcycle time for a 32-bit high-performance system when used with the other members of the Am29300 Family.
- **Real-Time Interrupt Support**
Micro-trap and interrupts are handled transparently at any microinstruction boundary.
- **Built-In Conditional Test Logic**
Has twelve external test inputs, four of which are used to internally generate four additional test conditions.
- **Break-Point Logic**
Built-in address comparator allows break-points in the microcode for debugging and statistics collection.
- **Master/Slave Error Checking**
Two sequencers can operate in parallel as a master and a slave. The slave generates a fault flag for unequal results.
- **33-Level Stack**
Provides support for interrupts, loops, and subroutine nesting. It can be accessed through the D-bus to support diagnostics.
- **Speed improvement with Am29331A (15% faster than Am29331)**

GENERAL DESCRIPTION

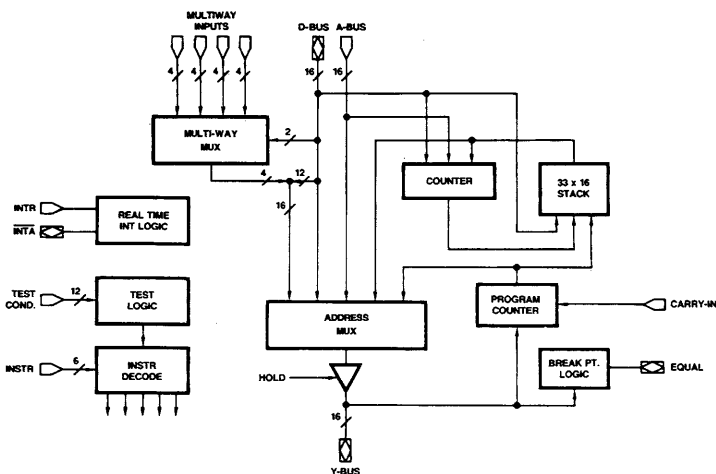
The Am29331 is a 16-bit wide, high-speed single-chip sequencer designed to control the execution sequence of microinstructions stored in the microprogram memory. The instruction set is designed to resemble high-level language constructs, thereby bringing high-level language programming to the micro level.

The Am29331 is interruptible at any microinstruction boundary to support real-time interrupts. Interrupts are handled transparently to the microprogrammer as an unexpected procedure call. Traps are also handled transparently at any microinstruction boundary. This feature allows re-execution of the prior microinstruction. Two separate buses are provided to bring a branch address directly into the chip from two sources to avoid slow turn-on and turn-off times

for different sources connected to the data-input bus. Four sets of multiway inputs are also provided to avoid slow turn-on and turn-off times for different branch-address sources. This feature allows implementation of table look-up or use of external conditions as part of a branch address. The 33-deep stack provides the ability to support interrupts, loops, and subroutine nesting. The stack can be read through the D-bus to support diagnostics or to implement multitasking at the micro-architecture level. The master/slave mode provides a complete function check capability for the device.

The Am29331 is designed with the IMOX™ process which allows internal ECL circuits with TTL-compatible I/O. It is housed in a 120-lead pin-grid-array package.

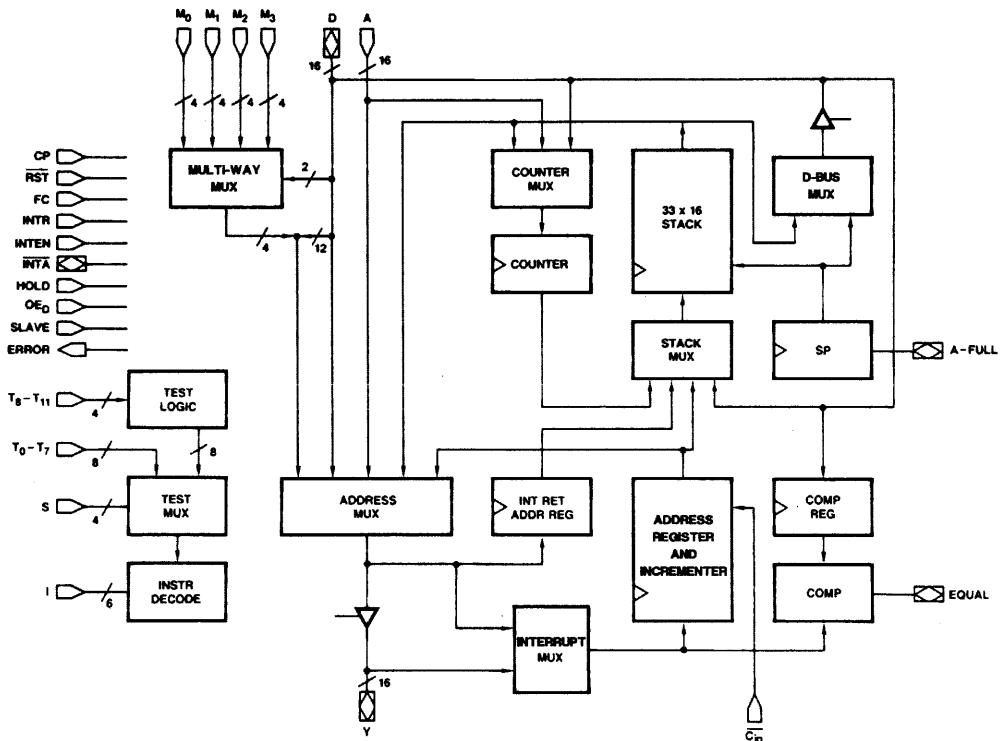
SIMPLIFIED BLOCK DIAGRAM



BD006091

RELATED AMD PRODUCTS

Part No.	Description
Am29114	Vectored Priority Interrupt Controller
Am29116	High-Performance Bipolar 16-Bit Microprocessor
Am29C116	High-Performance CMOS 16-Bit Microprocessor
Am29PL141	Field-Programmable Controller
Am29C323	CMOS 32-Bit Parallel Multiplier
Am29325	32-Bit Floating-Point Processor
Am29C325	CMOS 32-Bit Floating-Point Processor
Am29332	32-Bit Extended Function ALU
Am29C332	CMOS 32-Bit Extended Function ALU
Am29334	64 x 18 Four-Port, Dual-Access Register File
Am29C334	CMOS 64 x 18 Four-Port, Dual-Access Register File
Am29337	16-Bit Bounds Checker
Am29338	Byte Queue



BD006102

Figure 1. Am29331 Detailed Block Diagram

CONNECTION DIAGRAM

(Bottom View)

PGA*

	A	B	C	D	E	F	G	H	J	K	L	M	N
1	M0,0	M1,0	M2,0	M2,1	CTN	M1,2	M1,3	M2,3	GNDT	RST	INTR	SLAVE	D15
2	D0	A0	M3,0	M1,1	M0,2	M2,2	M0,3	M3,3	EQUAL	OED	INTEN	HOLD	A15
3	VCCT	Y0	D1	M0,1	M3,1	GNDE	M3,2	VCCE	A-FULL	ERROR	INTA	Y15	VCCT
4	A1	Y1	D2								D14	A14	Y14
5	GNDT	A2	Y2								D13	A13	GNDT
6	A3	D3	GNDE								GNDE	D12	Y13
7	Y3	D4	A4								A12	Y12	D11
8	D5	Y4	VCCE								VCCE	Y11	A11
9	GNDT	A5	Y5								D10	A10	GNDT
10	D6	A6	Y6								Y10	D9	A9
11	VCCT	D7	T3	T6	GNDE	T10	T11	I0	VCCE	I3	Y9	D8	VCCT
12	A7	T1	T2	T5	GNDE	T7	S0	S1	VCCE	I2	I4	A8	Y8
13	Y7	T0	T9	T4	GNDE	T8	CP	S3	VCCE	I1	S2	I5	FC

CD010382

*Pinout observed from pin side of package.

Key: VCCE = V_{CC}, ECL
 VCCT = V_{CC}, TTL
 GNDE = GND, ECL
 GNDT = GND, TTL

PIN DESIGNATIONS

(Sorted by Pin No.)

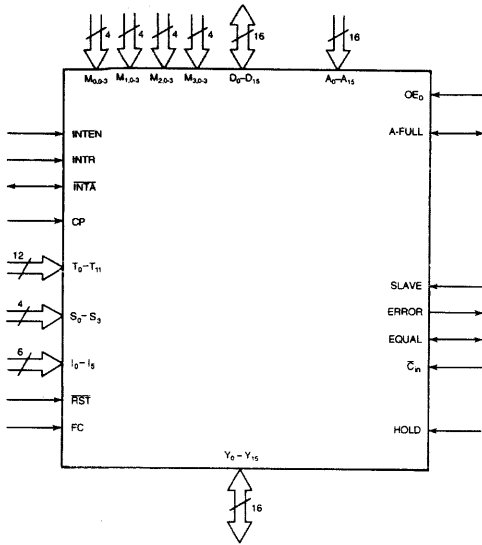
PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.
-	-	99	C-5	Y ₂	115	H-2	M _{3,3}	10	M-5	A ₁₃	80
-	-	97	C-6	GNDE	113	H-3	VCCE	68	M-6	D ₁₂	81
-	-	39	C-7	A ₄	52	H-11	I ₀	34	M-7	Y ₁₂	82
-	-	37	C-8	VCCE	53	H-12	S ₁	95	M-8	Y ₁₁	25
A-1	M _{0,0}	1	C-9	Y ₅	109	H-13	S ₃	94	M-9	A ₁₀	86
A-2	D ₀	120	C-10	Y ₆	48	J-1	GNDD	11	M-10	D ₉	87
A-3	VCCT	59	C-11	T ₃	44	J-2	EQUAL	71	M-11	D ₈	89
A-4	A ₁	58	C-12	T ₂	104	J-3	A-FULL	70	M-12	A ₈	30
A-5	GNDD	56	C-13	T ₉	41	J-11	VCCE	38	M-13	I ₅	91
A-6	A ₃	114	D-1	M _{2,1}	4	J-12	VCCE	38	N-1	D ₁₅	16
A-7	Y ₃	54	D-2	M _{1,1}	63	J-13	VCCE	38	N-2	A ₁₅	76
A-8	D ₅	51	D-3	M _{0,1}	3	K-1	RST	13	N-3	VCCT	17
A-9	GNDD	50	D-11	T ₆	102	K-2	OED	72	N-4	Y ₁₄	19
A-10	D ₆	49	D-12	T ₅	43	K-3	ERROR	12	N-5	GNDD	20
A-11	VCCT	47	D-13	T ₄	103	K-11	I ₃	92	N-6	Y ₁₃	21
A-12	A ₇	106	E-1	C _{in}	5	K-12	I ₂	33	N-7	D ₁₁	24
A-13	Y ₇	46	E-2	M _{0,2}	65	K-13	I ₁	93	N-8	A ₁₁	84
B-1	M _{1,0}	61	E-3	M _{3,1}	64	L-1	INTR	14	N-9	GNDD	26
B-2	A ₀	60	E-11	GNDE	98	L-2	INTEN	74	N-10	A ₉	28
B-3	Y ₀	119	E-12	GNDE	98	L-3	INTA	73	N-11	VCCT	29
B-4	Y ₁	117	E-13	GNDE	98	L-4	D ₁₄	18	N-12	Y ₈	90
B-5	A ₂	116	F-1	M _{1,2}	6	L-5	D ₁₃	79	N-13	FC	31
B-6	D ₃	55	F-2	M _{2,2}	66	L-6	GNDE	23			
B-7	D ₄	112	F-3	GNDE	8	L-7	A ₁₂	22			
B-8	Y ₄	111	F-11	T ₁₀	100	L-8	VCCE	83			
B-9	A ₅	110	F-12	T ₇	42	L-9	D ₁₀	85			
B-10	A ₆	108	F-13	T ₈	101	L-10	Y ₁₀	27			
B-11	D ₇	107	G-1	M _{1,3}	9	L-11	Y ₉	88			
B-12	T ₁	45	G-2	M _{0,3}	67	L-12	I ₄	32			
B-13	T ₀	105	G-3	M _{3,2}	7	L-13	S ₂	35			
C-1	M _{2,0}	2	G-11	T ₁₁	40	M-1	SLAVE	75			
C-2	M _{3,0}	62	G-12	S ₀	36	M-2	HOLD	15			
C-3	D ₁	118	G-13	CP	96	M-3	Y ₁₅	77			
C-4	D ₂	57	H-1	M _{2,3}	69	M-4	A ₁₄	78			

PIN DESIGNATIONS
(Sorted by Pin Name)

PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.
-	-	37	D ₈	M-11	89	INTR	L-1	14	T ₇	F-12	42
-	-	39	D ₉	M-10	87	M _{0, 0}	A-1	1	T ₈	F-13	101
-	-	97	D ₁₀	L-9	85	M _{0, 1}	D-3	3	T ₉	C-13	41
-	-	99	D ₁₁	N-7	24	M _{0, 2}	E-2	65	T ₁₀	F-11	100
A-FULL	J-3	70	D ₁₂	M-6	81	M _{0, 3}	G-2	67	T ₁₁	G-11	40
A ₀	B-2	60	D ₁₃	L-5	79	M _{1, 0}	B-1	61	V _{CC} E*	C-8	53
A ₁	A-4	58	D ₁₄	L-4	18	M _{1, 1}	D-2	63	V _{CC} E*	H-3	68
A ₂	B-5	116	D ₁₅	N-1	16	M _{1, 2}	F-1	6	V _{CC} E*	J-11	38
A ₃	A-6	114	EQUAL	J-2	71	M _{1, 3}	G-1	9	V _{CC} E*	J-12	38
A ₄	C-7	52	ERROR	K-3	12	M _{2, 0}	C-1	2	V _{CC} E*	J-13	38
A ₅	B-9	110	FC	N-13	31	M _{2, 1}	D-1	4	V _{CC} E*	L-8	83
A ₆	B-10	108	GNDE	C-6	113	M _{2, 2}	F-2	66	V _{CC} T	A-3	59
A ₇	A-12	106	GNDE	E-11	98	M _{2, 3}	H-1	69	V _{CC} T	A-11	47
A ₈	M-12	30	GNDE	E-12	98	M _{3, 0}	C-2	62	V _{CC} T	N-3	17
A ₉	N-10	28	GNDE	E-13	98	M _{3, 1}	E-3	64	V _{CC} T	N-11	29
A ₁₀	M-9	86	GNDE	F-3	8	M _{3, 2}	G-3	7	Y ₀	B-3	119
A ₁₁	N-8	84	GNDE	L-6	23	M _{3, 3}	H-2	10	Y ₁	B-4	117
A ₁₂	L-7	22	GNDT*	A-5	56	OE _D	K-2	72	Y ₂	C-5	115
A ₁₃	M-5	80	GNDT*	A-9	50	R _{ST}	K-1	13	Y ₃	A-7	54
A ₁₄	M-4	78	GNDT*	J-1	11	S ₀	G-12	36	Y ₄	B-8	111
A ₁₅	N-2	76	GNDT*	N-5	20	S ₁	H-12	95	Y ₅	C-9	109
C _{in}	E-1	5	GNDT*	N-9	26	S ₂	L-13	35	Y ₆	C-10	48
CP	G-13	96	HOLD	M-2	15	S ₃	H-13	94	Y ₇	A-13	46
D ₀	A-2	120	I ₀	H-11	34	SLAVE	M-1	75	Y ₈	N-12	90
D ₁	C-3	118	I ₁	K-13	93	T ₀	B-13	105	Y ₉	L-11	88
D ₂	C-4	57	I ₂	K-12	33	T ₁	B-12	45	Y ₁₀	L-10	27
D ₃	B-6	55	I ₃	K-11	92	T ₂	C-12	104	Y ₁₁	M-8	25
D ₄	B-7	112	I ₄	L-12	32	T ₃	C-11	44	Y ₁₂	M-7	82
D ₅	A-8	51	I ₅	M-13	91	T ₄	D-13	103	Y ₁₃	N-6	21
D ₆	A-10	49	INT _A	L-3	73	T ₅	D-12	43	Y ₁₄	N-4	19
D ₇	B-11	107	INTEN	L-2	74	T ₆	D-11	102	Y ₁₅	M-3	77

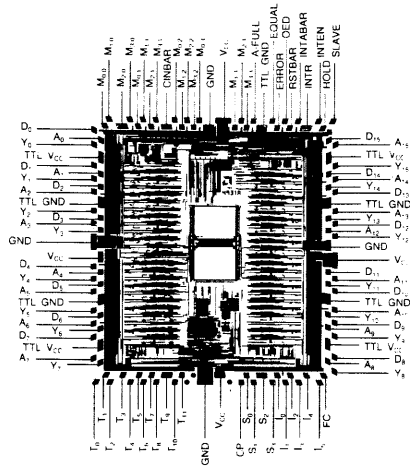
*Single +5-Volt supply.

LOGIC SYMBOL



LS002352

METALLIZATION AND PAD LAYOUT



Die Size: 260 x 245 mil
Equivalent Gate Count: 2500

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. Device Number
- b. Speed Option (if applicable)
- c. Package Type
- d. Temperature Range
- e. Optional Processing

AM29331

G

C

B

e. OPTIONAL PROCESSING

Blank = Standard processing
B = Burn-in

d. TEMPERATURE RANGE

C = Commercial (0 to +85°C)

c. PACKAGE TYPE

G = 120-Lead Pin Grid Array with Heatsink
(CG 120)

b. SPEED OPTION

Not Applicable

a. DEVICE NUMBER/DESCRIPTION

Am29331/Am29331A
16-Bit Microprogram Sequencer

Valid Combinations

Valid Combinations	
AM29331	GC, GCB
AM29331A	

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

A₀ - A₁₅ Alternate Data (Input)

Input to address multiplexer and counter.

A-FULL Almost Full (Bidirectional, Three-State)

Indicates that $28 \leq SP \leq 63$ (meaning there are five or less empty locations left on stack). Also active during stack-under flow.

C_{in} Carry In (Input, Active LOW)

Carry-in to the incrementer.

CP Clock Pulse (Input)

Clocks sequencer at the LOW-to-HIGH transition.

D₀ - D₁₅ Data (Bidirectional, Three-State)

Input to address multiplexer, counter, stack, and comparator register. Output for stack and stack pointer.

EQUAL Equal (Bidirectional, Three-State)

Indicates that the address comparator is enabled and has found a match.

ERROR Error (Output, Active HIGH)

Indicates a master/slave error in the slave mode. Indicates a malfunctioning driver or contention of any output in the master mode.

FC Force Continue (Input, Active HIGH)

Overrides instruction with CONTINUE.

HOLD Hold (Input, Active HIGH)

Stops the sequencer and three-states the outputs.

I₀ - I₅ Instruction (Input)

Selects one of 64 instructions.

INTA Interrupt Acknowledge (Bidirectional, Three-State, Active LOW)

Indicates that an interrupt is accepted.

INTEN Interrupt Enable (Input, Active HIGH)

Enables interrupts.

INTR Interrupt Request (Input, Active HIGH)

Requests the sequencer to interrupt execution.

M₀₋₃, O-3 Multiway (Input)

Four sets of multiway inputs providing 16-way branches. The first index refers to the set number.

OE_D Output Enable — D-Bus (Input, Active HIGH)

Enables the D-bus driver, provided that the sequencer is not in the hold or slave mode.

RST Reset (Input, Active LOW)

Resets the sequencer.

S₀ - S₃ Select (Input)

Selects one of 16 test conditions.

SLAVE Slave (Input, Active HIGH)

Makes the sequencer a slave.

T₀ - T₁₁ Test (Input)

Provides external test inputs.

Y₀ - Y₁₅ Address (Bidirectional, Three-State)

Output of microcode address. Input for interrupt address.

FUNCTIONAL DESCRIPTION

Architecture

The major blocks of the sequencer are the address multiplexer, the address register (AR), the stack (with the top of stack denoted TOS), the counter (C), the test multiplexer with logic, and the address comparison register (R) (Figure 1). The bidirectional D-bus provides branch addresses and iteration counts; it also allows access to the stack from the outside. The A-bus may be used for map addresses. There are four sets of 4-bit multiway branch inputs (M). The bidirectional Y-bus either outputs microprogram addresses or inputs interrupt addresses. The buses are all 16 bits wide. Figure 1 shows a detailed block diagram of the sequencer.

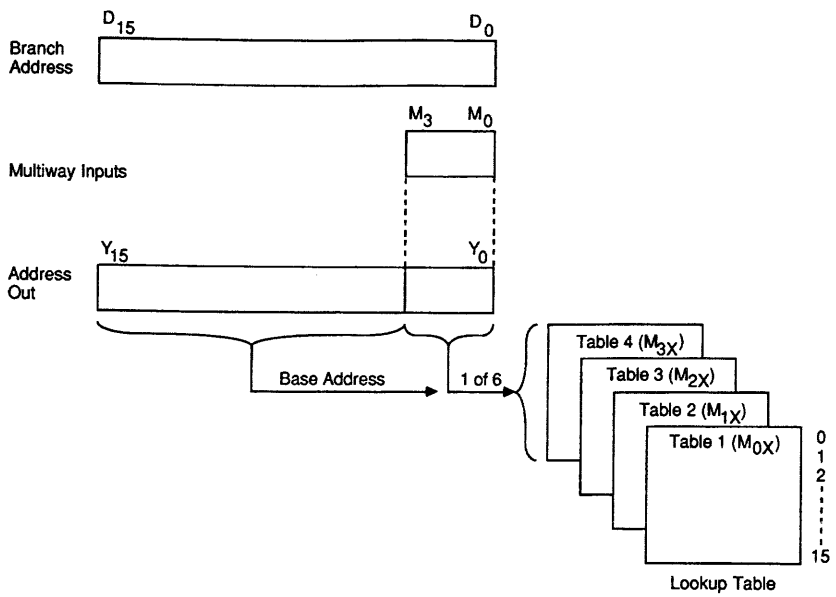
Address Multiplexer

The address multiplexer can select an address from any of five sources:

- 1) A branch address supplied by the D-bus
- 2) A branch address supplied by the A-bus
- 3) A multiway-branch address
- 4) A return or loop address from the top of stack
- 5) The next sequential address from the incrementer

Multiway-Branch Address

A multiway-branch address is formed by substituting the lower four bits of the address on the D-bus (D₃, D₂, D₁, D₀) with one of the four sets (M_{0X}, M_{1X}, M_{2X}, or M_{3X}) of 4-bit multiway-branch addresses. The multiway-branch set is selected by the number D₁D₀, while the bits D₃ and D₂ are "don't cares."



BD007460

- Notes:
1. D_1 and D_0 select one out of four multiway sets. D_3 and D_2 are "don't cares."
 2. Each set of $M_{3X} - M_{0X}$ can select one of sixteen locations. The multiway-branch address is the concatenation of $D_{15} - D_4$ (base address) and $M_{X3} - M_{X0}$.
 3. For a given base address, there can be four look-up tables, each sixteen deep.

Figure 2. Multiway Branch

Address Register

The address register contains the current address. It is loaded from the interrupt multiplexer and feeds the incrementer. The incrementer is inhibited if \overline{CIN} is taken HIGH.

Stack

A 33-word-deep and 16-bit-wide stack provides first-in last-out storage for return addresses, loop addresses, and counter values. Items to be pushed come from the incrementer, the interrupt-return-address register, the counter, or the D-bus. Items popped go to the address multiplexer, the counter, or the D-bus.

The access to the stack via the D-bus may be used for context switching, stack extension, or diagnostics. As the stack is only accessible from the top, stack extension is done by temporarily storing the whole or some lower part of the stack outside the sequencer. The save and the later restore are done with pop and push operations, respectively, at balanced points in the microprogram; for example, points with the same stack depth. The internal D-bus driver must be turned on when popping an item to the D-bus; if the driver is off, the item will be unstacked instead. The driver is normally turned on when the Output Enable signal is asserted and the sequencer is not being reset ($OE_D = 1$, $RST = 1$).

The stack pointer is a modulo 64 counter, which is incremented on each push and decremented on each pop. The stack pointer is reset to zero when the sequencer is reset, but the pointer may also be reset by instruction. Thus, the stack pointer indicates the number of items on the stack as long as stack overflow or underflow has not occurred. Overflow happens when an item is pushed onto a full stack, whereby the item at the bottom of the stack is overwritten. Underflow happens when an item is popped from an empty stack; in this case the item is undefined.

The contents of the stack pointer are present on the D-bus for all instructions except POP D, provided the driver is turned on. The output signal, A-FULL, is active under the following conditions: $28 \leq SP \leq 63$.

Counter

The counter may be used as a loop counter. It may be loaded from the D-bus, the A-bus, or via a pop from the stack. Its contents may also be pushed onto the stack.

A normal for-loop is set up by a FOR instruction, which loads the counter from the D- or A-bus with the desired number of iterations; the instruction also pushes onto the stack a loop address that points to the next sequential instruction. The end of the loop is given by an unconditional END FOR instruction, which tests the counter value against the value one and then decrements the counter. If the values differ, the loop is repeated by selecting the address at the stack as the next address. If the values are equal, the loop is terminated by popping the stack, thereby removing the loop address, and selecting the address from the incrementer as the next address. The number of iterations is a 16-bit unsigned number, except that the number zero corresponds to 65,536 iterations. By pushing and popping counter values it is possible to handle nested loops.

Address Comparison

The sequencer is able to compare the address from the interrupt multiplexer with the contents of the comparator register. The instruction SET loads the comparator register with the address on the D-bus and enables the comparison, while CLEAR disables it. The comparison is disabled at reset. A HIGH is present at the output EQUAL if the comparison is enabled and the two addresses are equal. The comparison is

useful for detection of a break point or counting the number of times a microinstruction at a specific address is executed.

Instruction Set

The sequencer has 64 instructions that are divided into four classes of 16 instructions each. The instruction lines $I_0 - I_5$ use I_5 and I_4 to select a class, and $I_0 - I_3$ to select an instruction within a class. The classes are:

I_5	I_4	Classes
0	0	Conditional sequence control,
0	1	Conditional sequence control with inverted polarity,
1	0	Unconditional sequence control, and
1	1	Special function with implicit continue.

Note that for the first three classes I_5 forces the condition to be true and I_4 inverts the condition. The basic instructions of the first three classes are shown in Table 1 and the instructions of the fourth class in Table 2.

Structured microprogramming is supported by sequencer instructions that singly or in pairs correspond to high-level language control constructs. Examples are FOR I: = D DOWN TO 1 DO ... END FOR and CASE N OF ... END CASE. The instructions have been given high-level language names where appropriate. Figure 3 shows how to microprogram important control constructs; the high-level language is on the left and the microcode on the right.

Test Conditions

The condition for a conditional instruction is supplied by a test multiplexer, which selects one out of sixteen tests with the select lines $S_0 - S_3$. Twelve of these are supplied directly by the inputs $T_0 - T_{11}$, while the remaining four tests are generated by the test logic from the inputs $T_8 - T_{11}$. The following table shows the assignments.

$(S_0 - S_3)_H$	Test	Intended Use
0-7	$T_0 - T_7$	General
8	T_8	C (Carry)
9	T_9	N (Negative)
A	T_{10}	V (Overflow)
B	T_{11}	Z (Zero or equal)
C	$T_8 + T_{11}$	C + Z (Unsigned less than or equal, borrow mode)
D	$\overline{T_8} + T_{11}$	$\overline{C} + Z$ (Unsigned less than or equal)
E	$T_9 \oplus T_{10}$	N \oplus V (Signed less than)
F	$(T_9 \oplus T_{10}) + T_{11}$	(N \oplus V) + Z (Signed less than or equal)

Force Continue

The sequencer has a force continue (FC) input, which overrides the instruction inputs $I_0 - I_5$ with a CONTINUE instruction. This makes it possible to share the microinstruction field for the sequencer instruction with some other control or to initialize a writable control store.

Reset

In order to start a microprogram properly, the sequencer must be reset. The reset works like an instruction overriding both the instruction input and the force continue input. The reset selects the address 0 at the address multiplexer, forces the EQUAL output to LOW, and disregards a potential interrupt request. It synchronously disables the address comparison and initializes the stack pointer to 0. The contents of the stack are invalid after a reset.

TABLE 1. INSTRUCTION SET for I₅I₄ = 00, 01, 10

I ₅ -I ₀	Instruction	Cond.: Fail		Cond.: Pass		Counter	Comp.	D-Mux
		Y	Stack	Y	Stack			
00, 10, 20	Goto D	INC	-	D	-	-	-	SP
01, 11, 21	Call D	INC	-	D	Push INC	-	-	SP
02, 12, 22	Exit D	INC	-	D	Pop	-	-	SP
03, 13, 23	End for D, C ≠ 1	INC	-	D	-	C←C-1	-	SP
	End for D, C = 1	INC	-	INC	-	C←C-1	-	SP
04, 14, 24	Goto A	INC	-	A	-	-	-	SP
05, 15, 25	Call A	INC	-	A	Push INC	-	-	SP
06, 16, 26	Exit A	INC	-	A	Pop	-	-	SP
07, 17, 27	End for A, C ≠ 1	INC	-	A	-	C←C-1	-	SP
	End for A, C = 1	INC	-	INC	-	C←C-1	-	SP
08, 18, 28	Goto M	INC	-	D:M	-	-	-	SP
09, 19, 29	Call M	INC	-	D:M	Push INC	-	-	SP
0A, 1A, 2A	Exit M	INC	-	D:M	Pop	-	-	SP
0B, 1B, 2B	End for M, C ≠ 1	INC	-	D:M	-	C←C-1	-	SP
	End for M, C = 1	INC	-	INC	-	C←C-1	-	SP
0C, 1C, 2C	End Loop	INC	Pop	TOS	-	-	-	SP
0D, 1D, 2D	Call Coroutine	INC	-	TOS	Pop & Push INC	-	-	SP
0E, 1E, 2E	Return	INC	-	TOS	Pop	-	-	SP
0F, 1F, 2F	End for, C ≠ 1	INC	Pop	TOS	-	C←C-1	-	SP
	End for, C = 1	INC	Pop	INC	Pop	C←C-1	-	SP

Cond. = (Test [S] OR I₅) XOR I₄

: = Concatination

C = Counter

INC = Output of Incrementer = AR + 1 (if $\overline{C_{in}}$ = LOW)

Note: For unconditional instructions, the action marked under Cond.:Pass is taken.

TABLE 2. INSTRUCTION SET for I₅I₄ = 11

I ₅ -I ₀	Instruction	Y	Stack	Counter	Comp.	D-Mux
30	Continue	INC	-	-	-	SP
31	For D	INC	Push INC	C←D	-	SP
32	Decrement	INC	-	C←C-1	-	SP
33	Loop	INC	Push INC	-	-	SP
34	Pop D	INC	Pop	-	-	TOS
35	Push D	INC	Push D	-	-	SP
36	Reset SP	INC	SP←0	-	-	SP
37	For A	INC	Push INC	C←A	-	SP
38	Pop C	INC	Pop	C←TOS	-	SP
39	Push C	INC	Push C	-	-	SP
3A	Swap	INC	TOS←C	C←TOS	-	SP
3B	Push C Load D	INC	Push C	C←D	-	SP
3C	Load D	INC	-	C←D	-	SP
3D	Load A	INC	-	C←A	-	SP
3E	Set	INC	-	-	R←D, Enable	SP
3F	Clear	INC	-	-	Disable	SP

R = Comp. Register

Interrupts

The sequencer may be interrupted at the completion of the current microcycle by asserting the interrupt request input INTR. The return address of the interrupted routine is saved on the stack so that nested interrupts can be easily implemented. An interrupt is accepted if interrupts are enabled and the sequencer is not being reset or held (INTEN = HIGH, RST = HIGH, and HOLD = LOW). The interrupt-acknowledge output (INTA) goes LOW when an interrupt is accepted.

When there is no interrupt, addresses go from the address multiplexer to the Y-bus via the driver, and to the address register and the comparator via the interrupt multiplexer. When there is an interrupt, the driver of the sequencer is turned off, an external driver is turned on, and the interrupt multiplexer is switched. The interrupt address is supplied via the external driver to the Y-bus, the address register, and the comparator (Figure 4). In order to save the address from the address multiplexer, the address is stored in the interrupt return address register, which for simplicity is clocked every cycle. The next microinstruction is the first microinstruction of the interrupt routine (Figure 5).

In this cycle the address in the interrupt return address register is automatically pushed onto the stack. Therefore the microinstruction in this cycle must not use the stack; if a stack operation is programmed, the result is undefined. The instructions that do not use the stack are GOTO D, GOTO A, GOTO M, CONTINUE, DECREMENT, LOAD D, LOAD A, SET and CLEAR. A RETURN instruction terminates the interrupt routine and the interrupted routine is resumed. Interrupts only work with a single-level control path.

Traps

A trap is an unexpected situation linked to current microinstruction that must be handled before the microinstruction completes and changes the state of the system. An example of such a situation is an attempt to read a word from memory across a word boundary in a single cycle. When a trap occurs, the current microinstruction must be aborted and re-executed after the execution of a trap routine, which in the meantime will take corrective measures. An interrupt, on the other hand, is not linked directly to the current microinstruction that can complete safely before an interrupt routine is executed.

Execution of a trap requires that the sequencer ignore the current microinstruction, select the trap return address at the address multiplexer, and initiate an interrupt. This will save the trap return address on the stack and issue the trap address from an external source (Figure 6). The address register

contains the address of the microinstruction in the pipeline register, thus the address register already contains the trap return address when a trap occurs. This address can be selected by the address multiplexer by disabling the incrementer ($\overline{CIN} = 1$), and using the force continue mode (FC = 1). In this mode the sequencer ignores the current microinstruction. The remaining part of the trap handling is done by the interrupt (Figure 7), thus the section on interrupts also applies to traps. There is one exception, however. The interrupt enable cannot be used as a trap enable as it does not control the force continue mode and the carry-in to the incrementer.

Hold Mode

The sequencer has a hold mode in which the operation is suspended.

When the HOLD signal goes active, the outputs (Y, INTA, A-FULL & EQUAL) are disabled and the sequencer enters the hold mode after the current cycle. While the sequencer is in this mode, the internal state is left unchanged and the D-bus is disabled. When the HOLD signal goes inactive, the outputs (Y, INTA, A-FULL & EQUAL) are enabled again and the sequencer leaves the hold mode after the cycle.

In a time-multiplexed multimicroprocess system there may be one sequencer for all processes with microprogrammed context save and restore, or there may be one sequencer per microprocess permitting fast process switch. In the latter case the Y-buses of the sequencers are tied together and connected to a single microprogram store. A control unit decides on a cycle-by-cycle basis what sequencer should be running, and activates the HOLD signal to the remaining sequencers. The hold mode has higher priority than interrupts, and works independently of the reset. The hold mode can only be used with a single-level control path.

Master/Slave Configuration

In some systems reliability is very important. The master/slave configuration that consists of two sequencers operated in parallel is able to detect faults in both the interconnect and the internal function of the sequencers. One sequencer is the master and operates normally. The other is the slave, i.e., all outputs except the signal ERROR are turned into inputs and connected to the outputs of the master. Since the slave is operated in parallel with the master, it can compare its result with the result of the master and signal an error if they differ. The error signal from the master indicates a malfunctioning driver or contention. Because a TTL output goes HIGH when power is missing, the ERROR signal also indicates power failure.

High-Level Language Constructs

An example of high-level language constructs using Am29331 instructions is given in Figure 3 (3-1, 3-2, 3-3, and 3-4).

```

REPEAT          LOOP
-              -
-              -
UNTIL CC        END LOOP NOT CC

WHILE CC DO     LOOP
-              IF NOT CC THEN EXIT L
-              -
-              -
END WHILE       END LOOP
                L:

LOOP           LOOP
-             -
IF CC THEN EXIT IF CC THEN EXIT L
-             -
END LOOP      END LOOP
                L:
    
```

Figure 3-1. Loops with Unknown Number of Iterations

```

FOR CNT: = 10 DOWN TO 1 DO  FOR D 10
-                             -
-                             -
END FOR                       END FOR
    
```

Figure 3-2. Loop with Known Number of Iterations

```

CASE I OF      PUSH D B
0: -           GOTO M
-             A: -
-             -, RETURN (TO B)
1: -           A + 2: -
-             -, RETURN (TO B)
2: -           A + 4: -
-             -, RETURN (TO B)
3: -           A + 6: -
-             -, RETURN
END CASE      B:
    
```

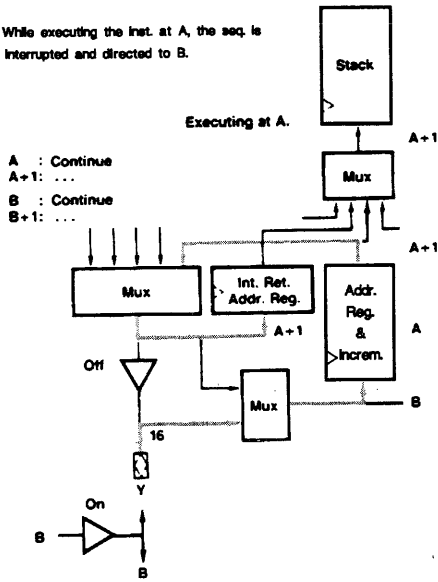
Figure 3-3. Case Statement
 (with $D = A_{15} \dots A_4XX00$ and $M_{0,0-3} = A_3I_1I_00$ during the GOTO M instruction. A_1A_0 must be 00, and X signifies a don't care.)

```

IF X THEN     PUSH D C
IF Y THEN     IF NOT X THEN GOTO A
-             IF NOT Y THEN GOTO B
-             -
-             -, RETURN (TO C)
ELSE          B:
-             -
-             -, RETURN (TO C)
END IF
ELSE          A:
IF Z THEN     IF NOT Z THEN GOTO D
-             -
-             -, RETURN (TO D)
ELSE          D:
-             -
-             -, RETURN (TO C)
END IF
END IF        C:
    
```

Figure 3-4. Double-Nested If Statement

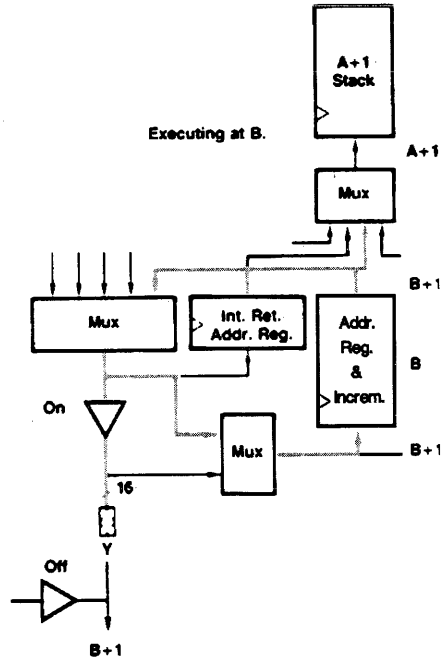
While executing the inst. at A, the seq. is interrupted and directed to B.



AF004192

Figure 4. Am29331 Interrupt Cycle 1

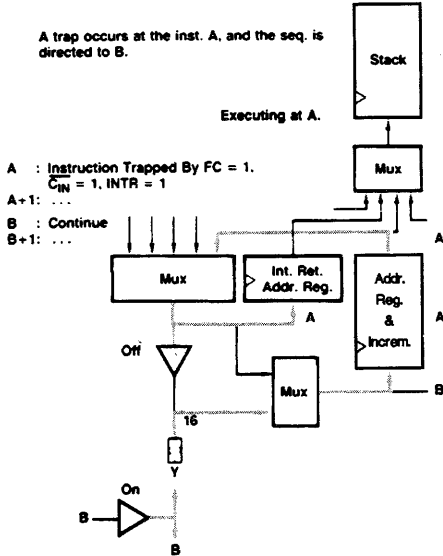
Executing at B.



AF004212

Figure 5. Am29331 Interrupt Cycle 2

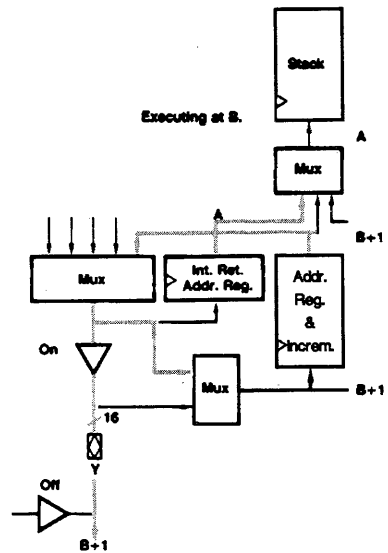
A trap occurs at the inst. A, and the seq. is directed to B.



AF004201

Figure 6. Am29331 Traps Cycle 1

Executing at B.



AF004182

Figure 7. Am29331 Traps Cycle 2

Instruction Set Definition

Legend: ● = Other instruction

⊙ = Instruction being described

CC = (Test [S₃ - S₀])

P = Test pass

F = Test fail

○ = Register in part

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
20H	BRA_D	GOTO D Unconditional branch to the address specified by the D inputs. The D port must be disabled to avoid bus contention.	
24H	BRA_A	GOTO A Unconditional branch to the address specified by the A inputs.	
28H	BRA_M	GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) Unconditional branch to the address specified by the M inputs concatenated with the D input. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the four-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
2CH	BRA_S	GOTO TOS Unconditional branch to the address on the top of the stack.	

PF001730

00H	BRCC_D	IF CC THEN GOTO D ELSE CONTINUE If CC is HIGH (pass), branch to the address specified by D. If CC is LOW (fail), continue. The D port must be disabled to avoid bus contention.	
04H	BRCC_A	IF CC THEN GOTO A ELSE CONTINUE If CC is HIGH (pass), branch to the address specified by A. If CC is LOW (fail), continue.	
08H	BRCC_M	IF CC THEN GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is HIGH (pass), branch to the address specified by D inputs concatenated with the M inputs. If CC is LOW (fail) continue. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
0CH	BRCC_S	IF CC THEN GOTO TOS ELSE POP STACK CONTINUE If CC is HIGH (pass), branch to the address on the top of the stack. If CC is LOW (fail), pop the stack and continue.	

PF001740

Note: Opcode numbers are in hexadecimal notation.

Opcode (15 - 10)	Mnemonics	Description	Execution Example
10H	BRNC_D	IF NOT CC THEN GOTO D ELSE CONTINUE If CC is LOW (pass), branch to the address specified by D. If CC is HIGH (fail), continue. The D Port must be disabled to avoid Bus contention.	
14H	BRNC_A	IF NOT CC THEN GOTO A ELSE CONTINUE If CC is LOW (pass), branch to the address specified by A. If CC is HIGH (fail), continue.	
18H	BRNC_M	IF NOT CC THEN GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is LOW (pass), branch to the address specified by D inputs concatenated with the M inputs. If CC is HIGH (fail), continue. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
1CH	BRNC_S	IF NOT CC THEN GOTO TOS ELSE POP STACK CONTINUE If CC is LOW (pass), branch to the address on the top of the stack. If CC is HIGH (fail), pop the stack and continue.	PF001750
21H	CALL_D	CALL D Unconditional branch to the subroutine specified by the D inputs. Push the return address (address Reg. + 1) on the stack. The D port must be disabled to avoid bus contention.	
25H	CALL_A	CALL A Unconditional branch to the subroutine specified by the A inputs. Push the return address (Address Reg. + 1) on the stack.	
29H	CALL_M	CALL Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) Unconditional branch to the subroutine specified by the D inputs concatenated with the multiway inputs. Push the return address (Address Reg. + 1) on the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
2DH	CALL_S	CALL TOS Unconditional branch to the subroutine specified by the address on the top of the stack. The stack is popped and the return address (Address Reg. + 1) is then pushed onto the stack.	PF001760

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
01H	CCC_D	IF CC, THEN CALL D ELSE CONTINUE If CC is HIGH (pass), call the subroutine specified by the D inputs. Push the return address (Address Reg. + 1) on the stack. If CC is LOW (fail), continue. The D port must be disabled to avoid bus contention.	
05H	CCC_A	IF CC, THEN CALL A ELSE CONTINUE If CC is HIGH (pass), call the subroutine specified by the A inputs. Push the return address (Address Reg. + 1) on the stack. If CC is LOW (fail), continue.	
09H	CCC_M	IF CC, THEN CALL Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is HIGH (pass), call the subroutine specified by the D inputs concatenated with the M inputs. Push the return address (Address Reg. + 1) on the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
0DH	CCC_S	IF CC, THEN CALL TOS ELSE CONTINUE If CC is HIGH (pass), call the subroutine specified by the address on the top of the stack. The stack is popped and the return address (Address Reg. + 1) is pushed onto the stack. If CC is LOW (fail), continue.	
11H	CNC_D	IF NOT CC, THEN CALL D ELSE CONTINUE If CC is LOW (pass), call the subroutine specified by the D inputs. Push the return address (Address Reg. + 1) on the stack. If CC is HIGH (fail), continue. The D port must be disabled to avoid bus contention.	
15H	CNC_A	IF NOT CC, THEN CALL A ELSE CONTINUE If CC is LOW (pass), call the subroutine specified by the A inputs. Push the return address (Address Reg. + 1) on the stack. If CC is HIGH (fail), continue.	
19H	CNC_M	IF NOT CC, THEN CALL Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is LOW (pass), call the subroutine specified by the D inputs concatenated with the M inputs. Push the return address (Address Reg. + 1) on the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
1DH	CNC_S	IF NOT CC, THEN CALL TOS ELSE CONTINUE If CC is LOW (pass), call the subroutine specified by the address on the top of the stack. The stack is popped and the return address (Address Reg. + 1) is pushed onto the stack.	

Note: Opcode numbers are in hexadecimal notation.

Opcode (i ₅ - i ₀)	Mnemonics	Description	Execution Example
22H	EXIT_D	EXIT TO D Unconditional branch to the address specified by the D inputs and pop the stack. The D port must be disabled to avoid bus contention.	
26H	EXIT_A	EXIT TO A Unconditional branch to the address specified by the A inputs and pop the stack.	
2AH	EXIT_M	EXIT TO Multiway (D ₁₅ - D ₄ M _{x3} - M _{x0}) Unconditional branch to the address specified by the D inputs concatenated with the M inputs and pop the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while D ₃ and D ₂ are "don't cares."	
2EH	EXIT_S	EXIT TO TOS Unconditional branch to the address on the top of the stack and pop the stack. Also used for unconditional returns.	PF001790
02H	XTCC_D	IF CC, THEN EXIT TO D ELSE CONTINUE If CC is HIGH (pass), exit to the address specified by the D inputs and pop the stack. If CC is LOW (fail), continue with no pop. The D port must be disabled to avoid bus contention.	
06H	XTCC_A	IF CC, THEN EXIT TO A ELSE CONTINUE If CC is HIGH (pass), exit to the address specified by the A inputs and pop the stack. If CC is LOW (fail), continue with no pop.	
0AH	XTCC_M	IF CC, THEN EXIT TO Multiway (D ₁₅ - D ₄ M _{x3} - M _{x0}) ELSE CONTINUE If CC is HIGH (pass), exit to the address specified by the D inputs concatenated with the M inputs and pop the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
0EH	XTCC_S	IF CC, THEN EXIT TO TOS ELSE CONTINUE If CC is HIGH (pass), exit to the address on the top of the stack and pop the stack. If CC is LOW (fail), continue with no pop. Also used for conditional returns.	PF001800

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₅ - I ₀)	Mnemonics	Description	Execution Example
12H	XTNC_D	IF NOT CC, THEN EXIT TO D ELSE CONTINUE If CC is LOW (pass), exit to the address specified by the D inputs and pop the stack. If CC is HIGH (fail), continue with no pop. The D port must be disabled to avoid bus contention.	
16H	XTNC_A	IF NOT CC, THEN EXIT TO A ELSE CONTINUE If CC is LOW (pass), exit to the address specified by the A inputs and pop the stack. If CC is HIGH (fail), continue with no pop.	
1AH	XTNC_M	IF NOT CC, THEN EXIT TO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CONTINUE If CC is LOW (pass), exit to the address specified by the D inputs concatenated with the M inputs and pop the stack. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiply branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
1EH	XTNC_S	IF NOT CC, THEN EXIT TO TOS ELSE CONTINUE If CC is LOW (pass), exit to the address on the top of the stack and pop the stack. If CC is HIGH (fail), continue with no pop. Also used for conditional returns.	
23H	DJMP_D	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO D ELSE CNT: = CNT - 1 CONTINUE If the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs. If the counter is equal to one, then decrement the counter and continue. The D port must be disabled to avoid bus contention.	
27H	DJMP_A	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO A ELSE CNT: = CNT - 1 CONTINUE If the counter is not equal to one, decrement the counter and branch to the address specified by the A inputs. If the counter is equal to one, then decrement the counter and continue.	
2BH	DJMP_M	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO Multiway (D ₁₅ - D ₄ M _{X3} - M _{X0}) ELSE CNT: = CNT - 1 CONTINUE If the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs concatenated with the M inputs. The lower four bits on the D bus (D ₃ - D ₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D ₁ and D ₀ while bits D ₃ and D ₂ are "don't cares."	
2FH	DJMP_S	IF CNT ≠ 1 THEN CNT: = CNT - 1 GOTO TOS ELSE CNT: = CNT - 1 POP STACK CONTINUE If the counter is not equal to one, decrement the counter and branch to the address on the top of the stack. If the counter is equal to one, then decrement the counter, pop the stack and continue.	

Note: Opcode numbers are in hexadecimal notation.

Opcode (I ₆ - I ₀)	Mnemonics	Description	Execution Example
03H	DJCC_D	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO D ELSE CNT: = CNT - 1 CONTINUE If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs. If CC is LOW (fail) or the counter is equal to one, then decrement the counter and continue. The D port must be disabled to avoid bus contention.</p>	
07H	DJCC_A	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO A ELSE CNT: = CNT - 1 CONTINUE If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the A inputs. If CC is LOW (fail) or the counter is equal to one, then decrement the counter and continue.</p>	PF001830
0BH	DJCC_M	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO Multiway (D₁₅ - D₄ M_{X3} - M_{X0}) ELSE CNT: = CNT - 1 CONTINUE If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs concatenated with the M inputs. The lower four bits on the D bus (D₃ - D₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D₁ and D₀ while bits D₃ and D₂ are "don't cares."</p>	
0FH	DJCC_S	<p>IF CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO TOS ELSE CNT: = CNT - 1 POP STACK CONTINUE If CC is HIGH (pass) and the counter is not equal to one, decrement the counter and branch to the address on the top of the stack. If CC is LOW (fail) or the counter is equal to one, then decrement the counter, pop the stack and continue.</p>	

Note: Opcode numbers are in hexadecimal notation.

Opcode (15-10)	Mnemonics	Description	Execution Example	
13H	DJNCC_D	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO D ELSE CNT: = CNT - 1 CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs. If CC is HIGH (fail) or the counter is equal to one, then decrement the counter and continue. The D port must be disabled to avoid bus contention.</p>		
17H	DJNCC_A	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO A ELSE CNT: = CNT - 1 CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the A inputs. The content of the interrupt return address register and the address register is replaced by the A address in this case. If CC is HIGH (fail) or the counter is equal to one, the current address is incremented, and is stored into the above two registers.</p>		
1BH	DJNCC_M	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO Multiway (D₁₅ - D₄ M₃ - M₀) ELSE CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address specified by the D inputs concatenated with the M inputs. The lower four bits on the D bus (D₃ - D₀) are replaced by one of the four sets of the 4-bit multiway branch addresses. The multiway branch set is selected by bits D₁ and D₀ while bits D₃ and D₂ are "don't cares."</p>		PF001840
1FH	DJNCC_S	<p>IF NOT CC AND CNT ≠ 1 THEN CNT: = CNT - 1 GOTO TOS ELSE CNT: = CNT - 1 POP STACK CONTINUE</p> <p>If CC is LOW (pass) and the counter is not equal to one, decrement the counter and branch to the address on the top of the stack. If CC is HIGH (fail) or the counter is equal to one, then decrement the counter, pop the stack and continue.</p>		

2EH	RET	<p>RETURN</p> <p>Unconditional return from subroutine. The return address is popped from the stack.</p>	
0EH	RETCC	<p>IF CC THEN RETURN ELSE CONTINUE</p> <p>If CC is HIGH (pass), return from subroutine. The return address is popped from the stack. If CC is LOW (fail), continue.</p>	
1EH	RETNC	<p>IF NOT CC THEN RETURN ELSE CONTINUE</p> <p>If CC is LOW (pass), return from subroutine. The return address is popped from the stack. If CC is HIGH (fail), continue.</p>	

PF001850

Note: Opcode numbers are in hexadecimal notation.

Opcode (!5 - !0)	Mnemonics	Description	Execution Example
31H	FOR_D	INITIALIZE LOOP Push the Address Reg. + 1 on the stack, load the counter from the D inputs and continue. Use with DJUMP_S for FOR...NEXT loops. The D port must be disabled to avoid bus contention.	
37H	FOR_A	INITIALIZE LOOP Push the Address Reg. + 1 on the stack, load the counter from the A inputs and continue. Use with DJUMP_S for FOR...NEXT loops.	
33H	LOOP	INITIALIZE LOOP Push the Address Reg. + 1 on the stack and continue. Use with BRCC_S for REPEAT...UNTIL loops, or with XTCC_D and BRA_S for WHILE...END WHILE loops.	

PF001860

34H	POP_D	Pop the stack and output the value on the D outputs and continue. The D port must be enabled.	
38H	POP_C	Pop the stack and store the value in the counter and continue.	
35H	PUSH_D	Push the D inputs on the stack and continue. The D port must be disabled to avoid bus contention.	
39H	PUSH_C	Push the counter on the stack and continue.	
3AH	SWAP	Exchange the counter and the top of stack and continue.	

PF001870

Note: Opcode numbers are in hexadecimal notation.

Opcode (15 - 10)	Mnemonics	Description	Execution Example
3BH	STACK_C	Push the counter on the stack and load the counter with the value of the D inputs and continue.	
3CH	LOAD_D	Load the counter with the value of the D inputs and continue. The D port must be disabled to avoid bus contention.	
3DH	LOAD_A	Load the counter with the value of the A inputs and continue.	
			PF001880
30H	CONT	Continue.	
32H	DECR	Decrement the counter and continue.	PF001890
36H	RESET_SP	Reset the stack pointer and continue.	
3EH	SET	Load the comparison register with the value of the D inputs, enable the comparator and continue.	PF001900
3FH	CLEAR	Disable the comparator and continue.	

Note: Opcode numbers are in hexadecimal notation.

APPLICATIONS

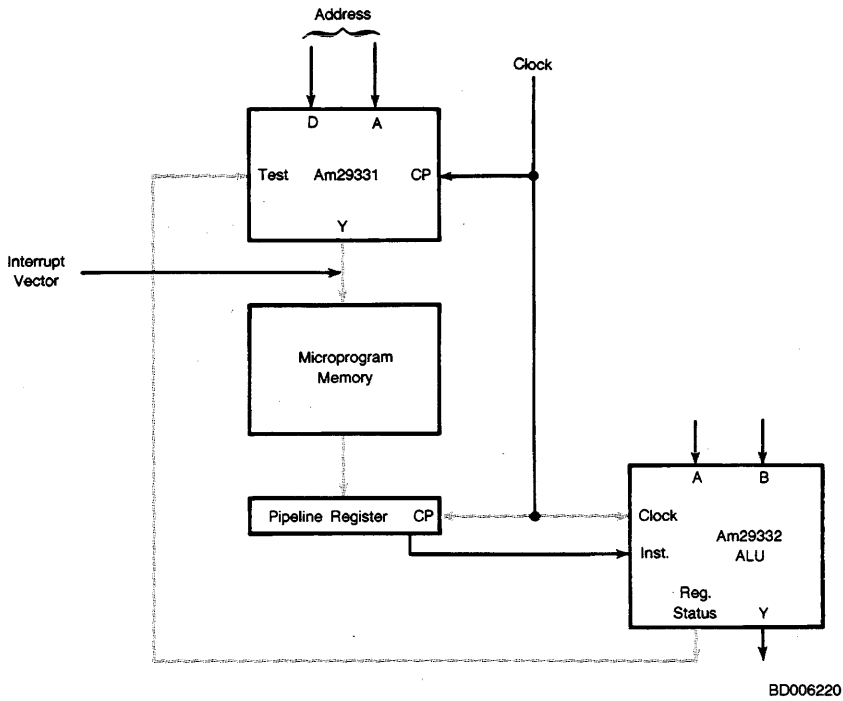


Figure 8. Typical Control-Path Architecture For Am29300 Family

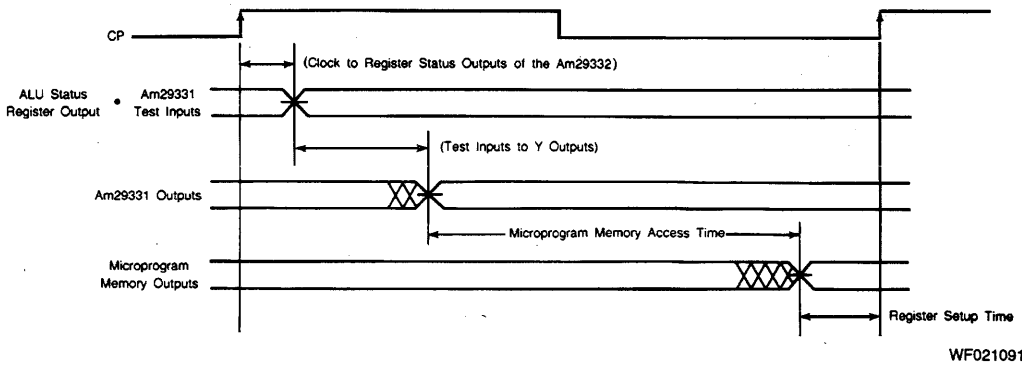


Figure 9. Cycle Timing Waveform*

* This waveform shows the timing relationship for the configuration shown in Figure 8.

Suggestions for Power and Ground Pin Connections

The Am29331 operates in an environment of fast signal rise times and substantial switching currents. Therefore, care must be exercised during circuit board design and layout, as with any high-performance component. The following is a suggested layout, but since systems vary widely in electrical configuration, an empirical evaluation of the intended layout is recommended.

The V_{CCT} and $GNDT$ pins, which carry output driver switching currents, tend to be electrically noisy. The V_{CCE} and $GNDE$ pins, which supply the ECL core of the device, tend to produce less noise, and the circuits they supply may be adversely affected by noise spikes on the V_{CCE} plane. For this reason, it is best to provide isolation between the V_{CCE} and V_{CCT} pins, as well as independent decoupling for each. Isolating the $GNDE$ and $GNDT$ pins is not required.

Printed Circuit-Board Layout Suggestions

1. Use of a multi-layer PC board with separate power, ground, and signal planes is highly recommended.
2. All V_{CCE} and V_{CCT} pins should be connected to the V_{CC} plane. V_{CCT} pins should be isolated from V_{CCE} pins by means of a slot cut in the V_{CCE} plane; see Figure 10. By physically separating the V_{CCE} and V_{CCT} pins, coupled noise will be reduced.
3. All $GNDE$ and $GNDT$ pins should be connected directly to the ground plane.
4. The V_{CCT} pins should be decoupled to ground with a $0.1\text{-}\mu\text{F}$ ceramic capacitor and a $10\text{-}\mu\text{F}$ electrolytic capacitor, placed as closely to the Am29331 as is practical. V_{CCE} pins should be decoupled to ground in a similar manner.

A suggested layout is shown in Figure 10.

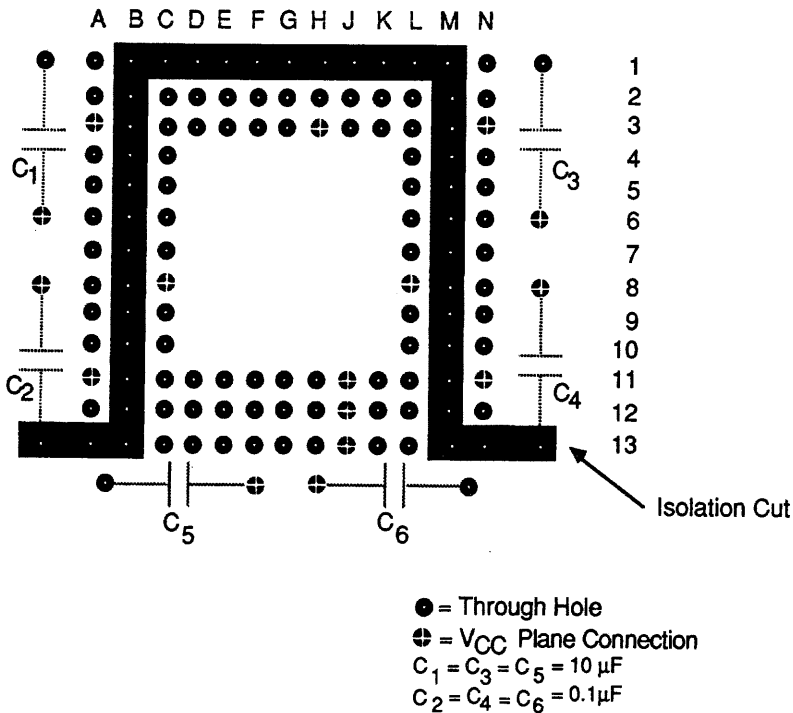
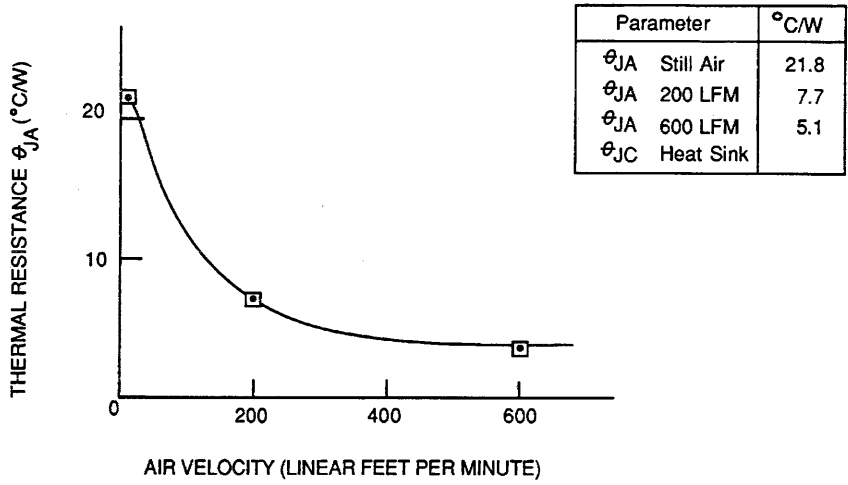


Figure 10. Suggested Printed Circuit-Board Layout

CD010890



OP002612

Figure 11. Am29331 Thermal Characteristics (Typical)

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Temperature Under Bias - T _C	-55 to +125°C
Supply Voltage to Ground Potential Continuous	-0.5 to +7.0 V
DC Voltage Applied to Outputs for High State	-0.5 V to +V _{CC} Max
DC Input Voltage	-0.5 to +5.5 V

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices Temperature (T _C)	0 to +85°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V
Air Velocity	200 linear feet per minute

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range

Parameters	Description	Test Conditions (Note 1)		Min.	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH}	I _{OH} = -1.6 mA for Y ₀ -Y ₁₅ , INTA I _{OH} = -1.2 mA for All Others	2.4		Volts
V _{OL}	Output LOW Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH}	I _{OL} = 16 mA for Y ₀ -Y ₁₅ , INTA I _{OL} = 12 mA for All Others		0.5	Volts
V _{IH}	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs		2.0		Volts
V _{IL}	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs			0.8	Volts
V _I	Input Clamp Voltage	V _{CC} = Min., I _{IN} = -18 mA			-1.5	Volts
I _{IL}	Input LOW Current	V _{CC} = Max., V _{IN} = 0.5 V	Y ₀ -Y ₁₅ , D ₀ -D ₁₅ , INTA, A-FULL, EQUAL A ₀ -A ₁₅ , M ₀ -3, 0-3, I ₀ -I ₅ , T ₀ -T ₁₁ , S ₀ -S ₃ , FC, C _{in} OED SLAVE, HOLD CP, INTR, INTEN RST		-0.55 -0.50 -1.0 -1.5 -2.5 -3.0	mA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = 2.4 V	Y ₀ -Y ₁₅ , D ₀ -D ₁₅ , INTA, A-FULL, EQUAL A ₀ -A ₁₅ , M ₀ -3, 0-3, I ₀ -I ₅ , T ₀ -T ₁₁ , S ₀ -S ₃ , FC, C _{in} OED SLAVE, HOLD CP, INTR, INTEN RST		100 50 100 150 250 300	μA
I _I	Input HIGH Current	V _{CC} = Max., V _{IN} = 5.5 V			1.0	mA
I _{OZH} I _{OZL}	Off State (High-Impedance) Output Current	V _{CC} = Max.	V _O = 2.4 V V _O = 0.5 V		100 -550	μA
I _{SC}	Output Short Circuit Current (Note 2)	V _{CC} = Max. +0.5 V V _{OUT} = +0.5 V		-15	-65	mA
I _{CC}	Power Supply Current (Note 3)	V _{CC} = Max.	COM'L Only T _C = 0 to +85°C T _C = +85°C		1,300 1,200	mA

- Notes: 1. For conditions shown as Min. or Max., use the appropriate value specified under Operating Ranges for the applicable device type.
 2. Not more than one output should be shorted at a time. Duration of the short-circuit test should not exceed one second.
 3. Measured with all inputs LOW and outputs disabled.
 4. It is the responsibility of the user to maintain a case temperature of +85°C or less. AMD recommends an air velocity of at least 200 linear feet per minute over the heatsink.

SWITCHING CHARACTERISTICS over operating range (Note 1)

A. COMBINATIONAL PROPAGATION DELAYS

No.	From	To	29331	29331A	Unit
			Max. Delay	Max. Delay	
1	D15-0	Y15-0	19	17	ns
	D15-0	EQUAL	23	20	ns
	D15-0	ERROR	25	22	ns
2	A15-0	Y15-0	19	17	ns
	A15-0	EQUAL	23	20	ns
	A15-0	ERROR	25	22	ns
3	Mx3-X0	Y15-0	19	17	ns
	Mx3-X0	EQUAL	23	20	ns
	Mx3-X0	ERROR	25	22	ns
	Y15-0	EQUAL	20	17	ns
	Y15-0	ERROR	21	18	ns
4	I5-0	Y31-0	25	22	ns
	I5-0	D15-0	31	27	ns
	I5-0	EQUAL	29	25	ns
	I5-0	ERROR	29	25	ns
6	T11-0	Y15-0	25	22	ns
	T11-0	EQUAL	30	26	ns
	T11-0	ERROR	30	26	ns
	S3-0	Y15-0	25	22	ns
	S3-0	EQUAL	30	26	ns
	S3-0	ERROR	30	26	ns
7	CP	Y15-0	20	17	ns
	CP	D15-0	20/Z	20/Z	ns
	CP	A-FULL	18	16	ns
9	CP	EQUAL	25	22	ns
	CP	ERROR	30	26	ns
	CP	Y15-0	26/Z	28/Z	ns
10	RST	D15-0	Z	Z	ns
	RST	INTA	12	12	ns
	RST	EQUAL	27	23	ns
11	RST	ERROR	29	25	ns
	FC	Y15-0	21	18	ns
	FC	D15-0	23	20	ns
12	FC	EQUAL	26	23	ns
	FC	ERROR	26	23	ns
	INTR	Y15-0	Z	Z	ns
14	INTR	INTA	11	11	ns
	INTR	EQUAL	(Note 2)	(Note 2)	ns
	INTR	ERROR	22	19	ns
	INTEN	Y15-0	Z	Z	ns
15	INTEN	INTA	11	11	ns
	INTEN	EQUAL	(Note 2)	(Note 2)	ns
	INTEN	ERROR	22	19	ns
	HOLD	Y15-0	Z	Z	ns
	HOLD	INTA	Z	Z	ns
	HOLD	A-FULL	Z	Z	ns
	HOLD	EQUAL	21/Z	21/Z	ns
	HOLD	ERROR	19	17	ns
16	OED	D15-0	Z	Z	ns
	OED	ERROR	19	17	ns
	INTA	ERROR	19	17	ns
	A-FULL	ERROR	19	17	ns
	EQUAL	ERROR	19	17	ns
	C _{in}	Y15-0	20	17	ns
	C _{in}	EQUAL	25	22	ns
	C _{in}	ERROR	26	23	ns
	SLAVE	Y15-0	Z	Z	ns
	SLAVE	D15-0	Z	Z	ns
	SLAVE	INTA	Z	Z	ns
SLAVE	A-FULL	Z	Z	ns	
SLAVE	EQUAL	Z	Z	ns	

Notes: See notes following Table C.

SWITCHING CHARACTERISTICS (Cont'd.)

B. OUTPUT DISABLE TIME

No.	From	To	Description	29331	29331A	Unit
				Max. Value	Max. Value	
43	RST	Y ₁₅₋₀	Reset-to-Address Enable	25	25	ns
	RST	Y ₁₅₋₀	Reset-to-Address Disable	25	25	ns
44	INTR	Y ₁₅₋₀	INTR-to-Address Enable	25	25	ns
	INTR	Y ₁₅₋₀	INTR-to-Address Disable	25	25	ns
	INTEN	Y ₁₅₋₀	INTEN-to-Address Enable	25	25	ns
	INTEN	Y ₁₅₋₀	INTEN-to-Address Disable	25	25	ns
	HOLD	Y ₁₅₋₀	HOLD-to-Address Enable	25	25	ns
	HOLD	Y ₁₅₋₀	HOLD-to-Address Disable	25	25	ns
	SLAVE	Y ₁₅₋₀	SLAVE-to-Address Enable	25	25	ns
	SLAVE	Y ₁₅₋₀	SLAVE-to-Address Disable	25	25	ns
	OED	Y ₁₅₋₀	OED-to-Data Enable	25	25	ns
	OED	D ₁₅₋₀	OED-to-Data Disable	25	25	ns
	RST	D ₁₅₋₀	Reset-to-Data Enable	25	25	ns
	RST	D ₁₅₋₀	Reset-to-Data Disable	25	25	ns
	SLAVE	D ₁₅₋₀	SLAVE-to-Data Enable	25	25	ns
	SLAVE	D ₁₅₋₀	SLAVE-to-Data Disable	25	25	ns
	CP	D ₁₅₋₀	Clock-to-Data Enable	30	30	ns
	CP	D ₁₅₋₀	Clock-to-Data Disable	30	30	ns
	HOLD	INTA	HOLD-to-INTA Enable	25	25	ns
	HOLD	INTA	HOLD-to-INTA Disable	25	25	ns
	HOLD	A-FULL	HOLD-to-A-FULL Enable	25	25	ns
	HOLD	A-FULL	HOLD-to-A-FULL Disable	25	25	ns
	HOLD	EQUAL	HOLD-to-EQUAL Enable	25	25	ns
	HOLD	EQUAL	HOLD-to-EQUAL Disable	25	25	ns
	SLAVE	INTA	SLAVE-to-INTA Enable	25	25	ns
	SLAVE	INTA	SLAVE-to-INTA Disable	25	25	ns
	SLAVE	A-FULL	SLAVE-to-A-FULL Enable	25	25	ns
	SLAVE	A-FULL	SLAVE-to-A-FULL Disable	25	25	ns
	SLAVE	EQUAL	SLAVE-to-EQUAL Enable	25	25	ns
	SLAVE	EQUAL	SLAVE-to-EQUAL Disable	25	25	ns

Notes: See notes following Table C.

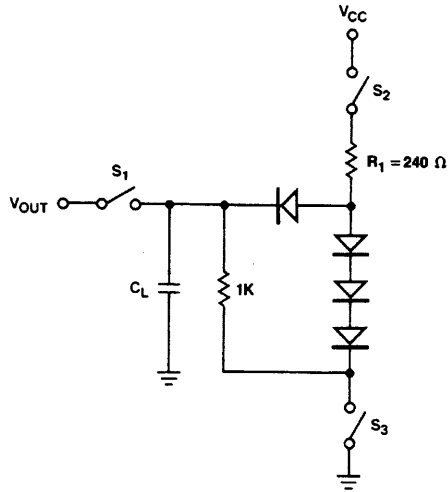
SWITCHING CHARACTERISTICS (Cont'd.)

C. SETUP AND HOLD TIMES

No.	Parameter	For	With Respect To	29331	29331A	Unit
				Max. Value	Max. Value	
17	Data Setup	D ₁₅₋₀	CP ↑	8	8	ns
18	Data Hold	D ₁₅₋₀	CP ↑	4	4	ns
19	Alternate Data Setup	A ₁₅₋₀	CP ↑	8	8	ns
20	Alternate Data Hold	A ₁₅₋₀	CP ↑	3	3	ns
21	Multiway Setup	M _{X3-X0}	CP ↑	8	8	ns
22	Multiway Hold	M _{X3-X0}	CP ↑	2	2	ns
23	Address Setup	Y ₁₅₋₀	CP ↑	5	5	ns
24	Address Hold	Y ₁₅₋₀	CP ↑	3	3	ns
25	Instruction Setup	I ₅₋₀	CP ↓	11	11	ns
26	Instruction Hold	I ₅₋₀	CP ↑	1	1	ns
27	Forced Continue Setup	FC	CP ↓	11	11	ns
28	Forced Continue Hold	FC	CP ↑	0	0	ns
29	Test Setup	T ₁₁₋₀	CP ↑	16	16	ns
30	Test Hold	T ₁₁₋₀	CP ↑	0	0	ns
31	Select Setup	S ₃₋₀	CP ↑	16	16	ns
32	Select Hold	S ₃₋₀	CP ↑	0	0	ns
33	Reset Setup	RST	CP ↑	15	15	ns
34	Reset Hold	RST	CP ↑	2	2	ns
35	Interrupt Request Setup	INTR	CP ↑	8	8	ns
36	Interrupt Request Hold	INTR	CP ↑	2	2	ns
37	Interrupt Enable Setup	INTEN	CP ↑	8	8	ns
38	Interrupt Enable Hold	INTEN	CP ↑	2	2	ns
39	Hold Mode Setup	HOLD	CP ↑	5	5	ns
40	Hold Mode Hold	HOLD	CP ↑	3	3	ns
41	Carry-In Setup	C _{in}	CP ↑	10	10	ns
42	Carry-In Hold	C _{in}	CP ↑	0	0	ns

- Notes: 1. It is the responsibility of the user to maintain a case temperature of +85°C or less. AMD recommends an air velocity of at least 200 linear feet per minute over the heatsink.
2. (INTR, INTEN)-to-EQUAL is the sum of (INTR, INTEN)-to-Y disable time and Y-to-EQUAL delay time. This is not tested due to bus turnaround in Master/Slave mode.
3. The status of I₅₋₀ and FC must not be changed during the Clock LOW time.
4. C_L = 50 pF; C_L = 5 pF for Disable Time only.
5. Z = Three-state output path; use Table B.

SWITCHING TEST CIRCUIT

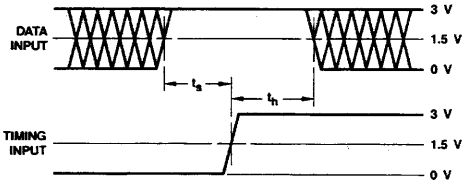


TC003420

A. Three-State Outputs

- Notes:
1. $C_L = 50$ pF includes scope probe, wiring, and stray capacitances without device in test fixture.
 2. S_1 , S_2 , S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for tp_{ZH} test.
 S_1 and S_2 are closed while S_3 is open for tp_{ZL} test.
 4. $C_L = 5.0$ pF for output disable tests.

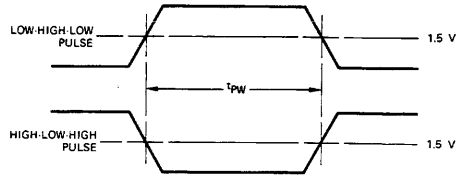
SWITCHING TEST WAVEFORMS



WFR02970

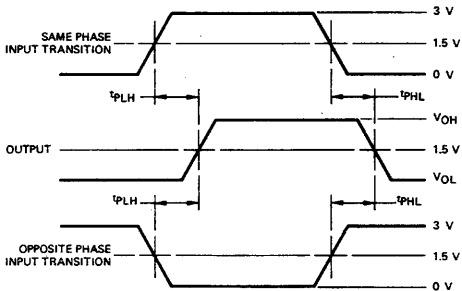
- Notes: 1. Diagram shown for HIGH data only. Output transition may be opposite sense.
2. Cross hatched area is don't care condition.

Setup, Hold, and Release Times



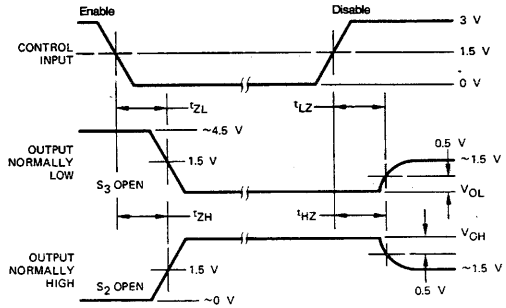
WFR02790

Pulse Width



WFR02980

Propagation Delay



WFR02663

- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
2. S_1 , S_2 , and S_3 of Load Circuit are closed except where shown.

Enable and Disable Times

Notes on Test Methods

The following points give the general philosophy which we apply to tests which must be properly engineered if they are to be implemented in an automatic environment. The specifics of what philosophies applied to which test are shown.

1. Ensure the part is adequately decoupled at the test head. Large changes in supply current when the device switches may cause function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an input transition, ground current may change by as much as 400 mA in 5 - 8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily.
4. Use extreme care in defining input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins which may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \geq 3$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. Capacitive Loading for AC Testing

Automatic testers and their associated hardware have stray capacitance which varies from one type of tester to another, but is generally around 50 pF. This makes it impossible to make direct measurements of parameters which call for a smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays" which measure the propagation delays into and out of the high-impedance state, and are usually specified at a load capacitance of 5.0 pF. In these cases, the test is performed at the higher load capacitance (typically 50 pF), and engineering correlations based on data taken with a bench setup are used to predict the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impossible to make measurements at both capacitances even though they may both be greater than the stray capacitance. In these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is predicted from engineering correlations based on data taken with a bench setup and the knowledge that certain DC measurements (I_{OH} , I_{OL} , for example) have already been taken and are within specification. In some cases, special DC tests are performed in order to facilitate this correlation.

7. Threshold Testing

The noise associated with automatic testing, the long inductive cables, and the high gain of bipolar devices when in the vicinity of the actual device threshold frequently give rise to oscillations when testing high-speed circuits. These oscillations are not indicative of a reject device, but instead, of an overtaxed test system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" high and low levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels rather than at V_{IL} max. and V_{IH} min.






8. AC Testing

Occasionally parameters are specified which cannot be measured directly on automatic testers because of tester limitations. Data input hold times often fall into this category. In these cases, the parameter in question is guaranteed by correlating these tests with other AC tests which have been performed. These correlations are arrived at by the cognizant engineer by using data from precise bench measurements in conjunction with the knowledge that certain DC parameters have already been measured and are within specification.

In some cases, certain AC tests are redundant since they can be shown to be predicted by other tests which have already been performed. In these cases, the redundant tests are not performed.

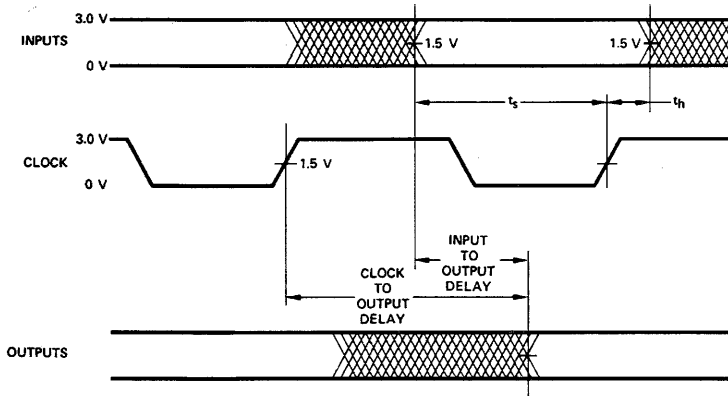
SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

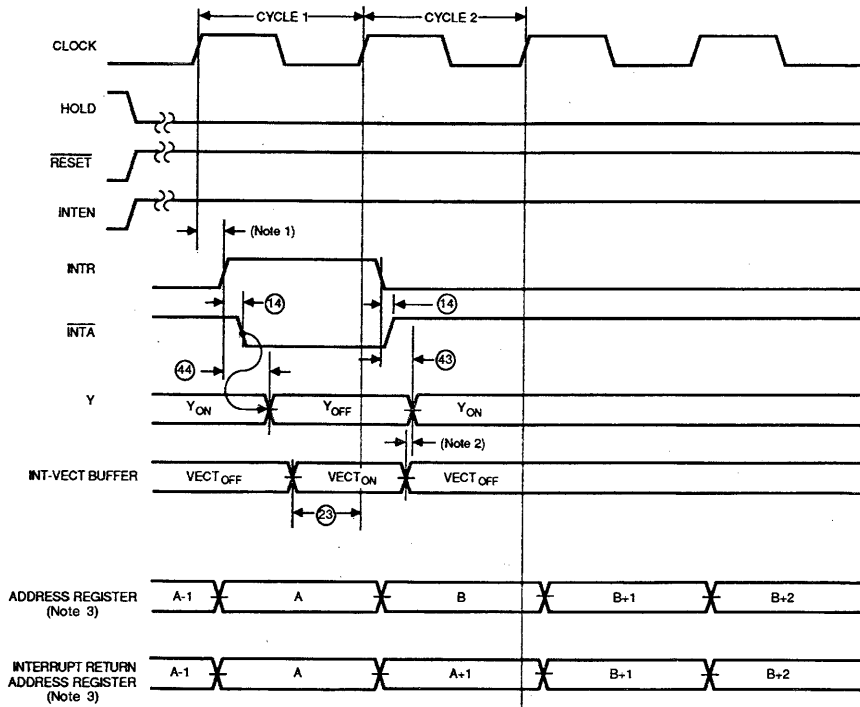
WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING WAVEFORMS (Cont'd.)



WFR02990

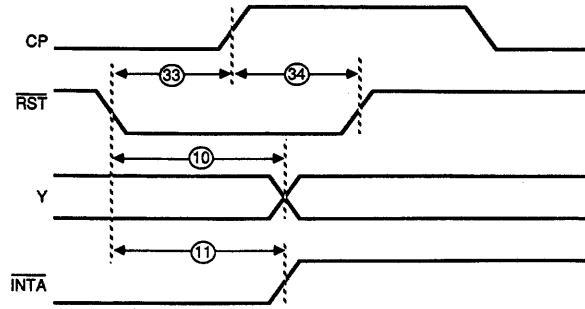


WF025100

Interrupt Timing

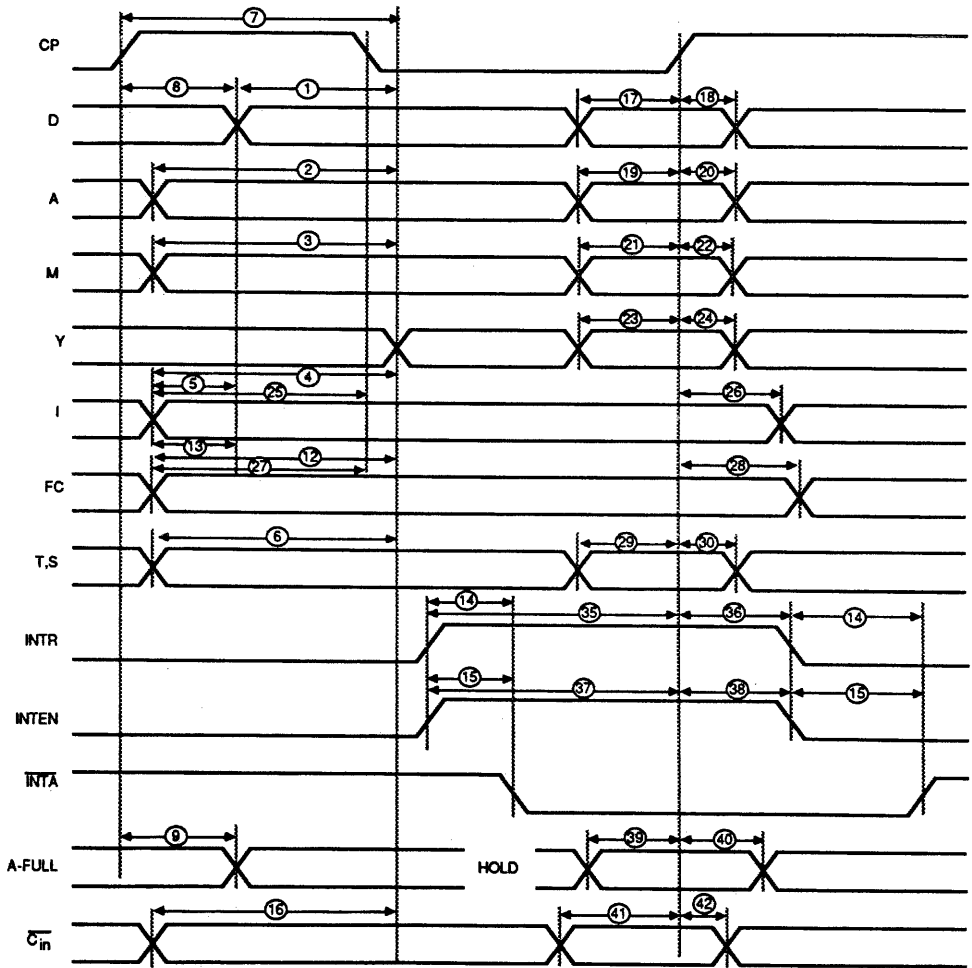
- Notes:
1. Interrupt Request comes from an interrupt-controller register. It reflects the CP \uparrow to INTR time of the interrupt controller.
 2. During Cycle 2, there may be contention on the Y-bus if the Y-bus is turned ON before the INT-VECT buffer is turned OFF.
 3. Refer to Figures 4 and 5 for definition of A and B.

SWITCHING WAVEFORMS (Cont'd.)



WF024770

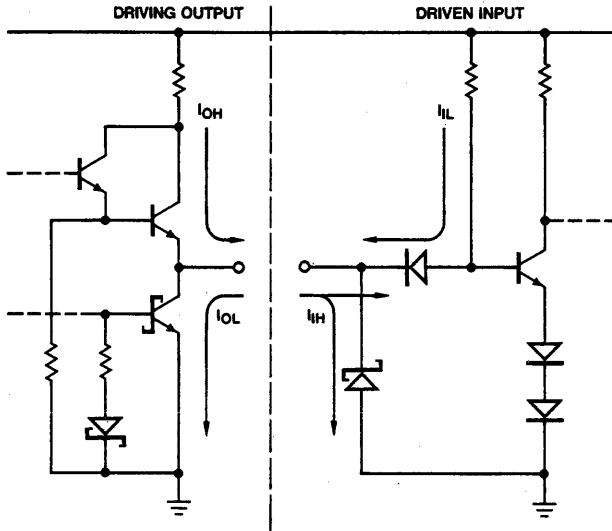
Reset Timing



WF024780

INPUT/OUTPUT INTERFACE CONDITIONS

(All Devices)



ICR00480

Am29332

32-Bit Arithmetic Logic Unit

Am29332

DISTINCTIVE CHARACTERISTICS

- Single Chip, 32-Bit ALU**
 Supports 80–90 ns microcycle time for the 32-bit data path. It is a combinatorial ALU with equal cycle time for all instructions.
- Flow-through Architecture**
 A combinatorial ALU with two input data ports and one output data port allows implementation of either parallel or pipelined architectures.
- 64-Bit In, 32-Bit Out Funnel Shifter**
 This unique functional block allows n-bit shift-up, shift-down, 32-bit barrel shift or 32-bit field extract.
- Supports All Data Types**
 It supports one-, two-, three- and four-byte data for all operations and variable-length fields for logical operations.
- Multiply and Divide Support**
 Built-in hardware to support two-bit-at-a-time modified Booth's algorithm and one-bit-at-a-time division algorithm.
- Extensive Error Checking**
 Parity check and generate provides data transmission check and master/slave mode provides complete function checking.

GENERAL DESCRIPTION

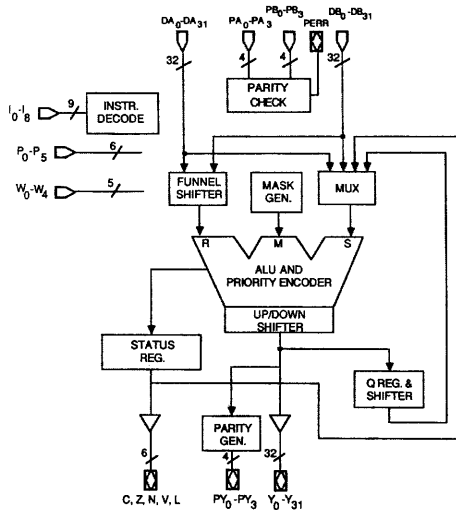
The Am29332 is a 32-bit wide non-cascadable Arithmetic Logic Unit (ALU) with integration of functions that normally don't cascade, such as barrel shifters, priority encoders and mask generators. Two input data ports and one output data port provide flow-through architecture and allow the designer to implement his/her architecture with any degree of pipelining and no built-in penalties for branching. Also, the simplicity of a three-bus ALU allows easy implementation of parallel or reconfigurable architectures. The register file is off-chip to allow unlimited expansion and regular addressability.

The Am29332 supports one-, two-, three- and four-byte data for arithmetic and logic operations. It also supports

multiprecision arithmetic and shift operations. For logical operations, it can support variable-length fields up to 32 bits. When fewer than four bytes are selected, unselected bits are passed to the destination without modification. The device also supports two-bit-at-a-time modified Booth's algorithm for high-speed multiplication and one-bit-at-a-time division. Both signed and unsigned integers for all byte aligned data types mentioned above are supported.

The Am29332 is designed to support 80–90 ns microcycle time. The device is packaged in a 169-lead pin-grid-array package.

SIMPLIFIED BLOCK DIAGRAM



BD007040

RELATED AMD PRODUCTS

Part No.	Description
Am29C01	CMOS 4-Bit Microprocessor Slice
Am29C10A	CMOS 12-Bit Sequencer
Am29C101	CMOS 16-Bit Microprocessor
Am29112	8-Bit Cascadable Microprogram Sequencer
Am29114	Real-Time Interrupt Controller
Am29C116	CMOS 16-Bit Microcontroller
Am29C323	CMOS 32 x 32 Parallel Multiplier
Am29325	32-Bit Floating Point Processor
Am29C325	CMOS 32-Bit Floating Point Processor
Am29331	16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29334	64 x 18 Four-Port, Dual-Access Register File
Am29C334	CMOS 64 x 18 Four-Port, Dual-Access Register File
Am29337	16-Bit Bounds Checker
Am29338	32-Bit Byte Queue
Am29C516	CMOS 16 x 16 Multiplier
Am29C517	CMOS 16 x 16 Multiplier with Separate I/O

CONNECTION DIAGRAM 169-Lead PGA Bottom View

	A	B	C	D	E	F	G	H	J	K	L	M	N	P	R	T	U
1	DB6	DA6	DB7	DB8	DB9	DB10	DA11	DB12	DA14	DB16	PB1	DB18	DB19	DB20	DA23	PA2	
2	DA5	DA5	DA7	PB0	DA9	DB11	DA12	DA13	DB14	PA1	DA16	DA17	DA19	DA20	DA21	DB23	PB2
3	DB4	DA3	DA4	PA0	DA8	DA10	GND	DB13	DB15	DA15	VCC	DB17	DA18	DB21	DA22	DA24	DB24
4	DB2	DA2	DB3	*											DB25	DA25	DB28
5	DB1	DA1	DA0												DB26	DA27	DB27
6	DB0	P5	P4												DB28	DA28	DB29
7	P1	P3	VCC												VCC	DA30	DA29
8	P2	P0	W4												DB31	DA31	DB30
9	W2	W1	W3												MSERR	PA3	PB3
10	I2	W0	I0												Y31	Y30	Y28
11	I3	I1	GND												GND	Y27	Y29
12	I6	I4	I5												VCC	GND	Y26
13	I8	I7	CP												Y25	Y23	Y24
14	MLINK	RS	SLAVE												GND	VCC	Y22
15	M/̄	MCn	N	C	PY3	PY2	GND	Y3	GND	Y7	VCC	Y8	Y12	OEY	Y19	Y21	Y20
16	BORROW	V	L	VCC	PY1	GNDT	GND	Y2	Y5	Y6	VCC	Y11	Y10	Y13	Y15	Y18	Y17
17	HOLD	Z	GND	PY0	Y0	PERR	GND	Y1	Y4	GNDT	VCC	Y9	VCC	GND	Y14	Y16	GND

CD010462

* This pin is not used

Key: VCCE = VCC, ECL
 VCCT = VCC, TTL
 GNDE = GND, ECL
 GNDD = GND, TTL

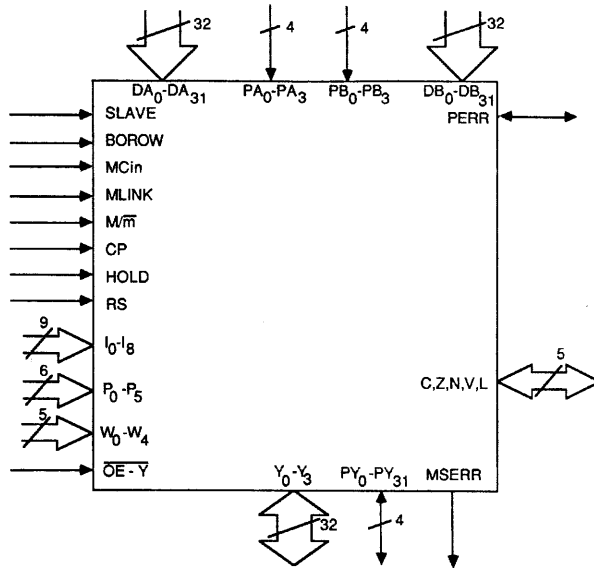
PIN DESIGNATIONS
(Sorted by Pin No.)

PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.
A-1	DB ₆	1	C-9	W ₃	145	J-15	GND, TTL	105	R-10	Y ₃₁	66
A-2	DA ₅	164	C-10	I ₀	139	J-16	Y ₅	101	R-11	GND, ECL	64
A-3	DB ₄	161	C-11	GND, ECL	143	J-17	Y ₄	102	R-12	V _{CC} , TTL	71
A-4	DB ₂	157	C-12	I ₅	134	K-1	DB ₁₆	27	R-13	Y ₂₅	74
A-5	DB ₁	155	C-13	CP	130	K-2	PA ₁	25	R-14	GND, TTL	79
A-6	DB ₀	153	C-14	SLAVE	127	K-3	DA ₁₅	24	R-15	Y ₁₉	82
A-7	P ₁	148	C-15	N	120	K-15	Y ₇	99	R-16	Y ₁₅	88
A-8	P ₂	149	C-16	L	118	K-16	Y ₆	100	R-17	Y ₁₄	89
A-9	W ₂	142	C-17	GND, TTL	117	K-17	GND, TTL	98	T-1	DA ₂₃	42
A-10	I ₂	137	D-1	DB ₈	7	L-1	PB ₁	26	T-2	DB ₂₃	41
A-11	I ₃	136	D-2	PB ₀	6	L-2	DA ₁₆	28	T-3	DA ₂₄	46
A-12	I ₆	133	D-3	PA ₀	5	L-3	V _{CC} , ECL	22	T-4	DA ₂₅	48
A-13	I ₈	131	D-15	C	119	L-15	V _{CC} , ECL	103	T-5	DA ₂₇	52
A-14	MLINK	129	D-16	V _{CC} , TTL	116	L-16	V _{CC} , ECL	103	T-6	DA ₂₈	54
A-15	M/ \bar{m}	125	D-17	PY ₀	115	L-17	V _{CC} , ECL	103	T-7	DA ₃₀	58
A-16	BOROW	124	E-1	DB ₉	9	M-1	DB ₁₈	31	T-8	DA ₃₁	60
A-17	HOLD	123	E-2	DA ₉	10	M-2	DA ₁₇	30	T-9	PA ₃	61
B-1	DA ₆	2	E-3	DA ₈	8	M-3	DB ₁₇	29	T-10	Y ₃₀	67
B-2	DB ₅	163	E-15	PY ₃	112	M-15	Y ₈	96	T-11	Y ₂₇	70
B-3	DA ₃	160	E-16	PY ₁	114	M-16	Y ₁₁	93	T-12	GND, TTL	72
B-4	DA ₂	158	E-17	Y ₀	109	M-17	Y ₉	95	T-13	Y ₂₃	76
B-5	DA ₁	156	F-1	DB ₁₀	11	N-1	DB ₁₉	33	T-14	V _{CC} , TTL	78
B-6	P ₅	152	F-2	DB ₁₁	13	N-2	DA ₁₉	34	T-15	Y ₂₁	80
B-7	P ₃	150	F-3	DA ₁₀	12	N-3	DA ₁₈	32	T-16	Y ₁₈	83
B-8	P ₀	147	F-15	PY ₂	113	N-15	Y ₁₂	92	T-17	Y ₁₆	86
B-9	W ₁	141	F-16	GND, TTL	110	N-16	Y ₁₀	94	U-1	PA ₂	43
B-10	W ₀	140	F-17	PERR	111	N-17	V _{CC} , TTL	97	U-2	PB ₂	44
B-11	I ₁	138	G-1	DA ₁₁	14	P-1	DB ₂₀	35	U-3	DB ₂₄	45
B-12	I ₄	135	G-2	DA ₁₂	16	P-2	DA ₂₀	36	U-4	DB ₂₆	49
B-13	I ₇	132	G-3	GND, ECL	21	P-3	DB ₂₁	37	U-5	DB ₂₇	51
B-14	RS	128	G-15	GND, ECL	104	P-15	$\overline{OE-Y}$	87	U-6	DB ₂₉	55
B-15	MCin	126	G-16	GND, ECL	104	P-16	Y ₁₃	90	U-7	DA ₂₉	56
B-16	V	121	G-17	GND, ECL	104	P-17	GND, TTL	91	U-8	DB ₃₀	57
B-17	Z	122	H-1	DB ₁₂	15	R-1	DB ₂₂	39	U-9	PB ₃	62
C-1	DB ₇	3	H-2	DA ₁₃	18	R-2	DA ₂₁	38	U-10	Y ₂₈	69
C-2	DA ₇	4	H-3	DB ₁₃	17	R-3	DA ₂₂	40	U-11	Y ₂₉	68
C-3	DA ₄	162	H-15	Y ₃	106	R-4	DB ₂₅	47	U-12	Y ₂₆	73
C-4	DB ₃	159	H-16	Y ₂	107	R-5	DA ₂₆	50	U-13	Y ₂₄	75
C-5	DA ₀	154	H-17	Y ₁	108	R-6	DB ₂₈	53	U-14	Y ₂₂	77
C-6	P ₄	151	J-1	DA ₁₄	20	R-7	V _{CC} , ECL	63	U-15	Y ₂₀	81
C-7	V _{CC} , ECL	144	J-2	DB ₁₄	19	R-8	DB ₃₁	59	U-16	Y ₁₇	84
C-8	W ₄	146	J-3	DB ₁₅	23	R-9	MSERR	65	U-17	GND, TTL	85

PIN DESIGNATIONS
(Sorted by Pin Names)

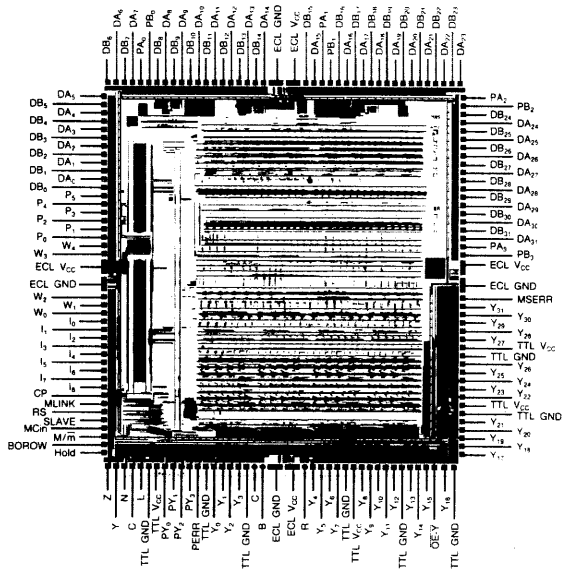
PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.
BOROW	A-16	124	DB ₇	C-1	3	I ₂	A-10	137	V _{CC} , TTL	T-14	78
C	D-15	119	DB ₈	D-1	7	I ₃	A-11	136	V _{CC} , TTL	N-17	97
CP	C-13	130	DB ₉	E-1	9	I ₄	B-12	135	V _{CC} , TTL	D-16	116
DA ₀	C-5	154	DB ₁₀	F-1	11	I ₅	C-12	134	V _{CC} , TTL	H-12	71
DA ₁	B-5	156	DB ₁₁	F-2	13	I ₆	A-12	133	W ₀	B-10	140
DA ₂	B-4	158	DB ₁₂	H-1	15	I ₇	B-13	132	W ₁	B-9	141
DA ₃	B-3	160	DB ₁₃	H-3	17	I ₈	A-13	131	W ₂	A-9	142
DA ₄	C-3	162	DB ₁₄	J-2	19	L	C-16	118	W ₃	C-9	145
DA ₅	A-2	164	DB ₁₅	J-3	23	MC _{in}	B-15	126	W ₄	C-8	146
DA ₆	B-1	2	DB ₁₆	K-1	27	MLINK	A-14	129	Y ₀	E-17	109
DA ₇	C-2	4	DB ₁₇	M-3	29	M/ \bar{m}	A-15	125	Y ₁	H-17	108
DA ₈	E-3	8	DB ₁₈	M-1	31	MSERR	R-9	65	Y ₂	H-16	107
DA ₉	E-2	10	DB ₁₉	N-1	33	N	C-15	120	Y ₃	H-15	106
DA ₁₀	F-3	12	DB ₂₀	P-1	35	OE- \bar{Y}	P-15	87	Y ₄	J-17	102
DA ₁₁	G-1	14	DB ₂₁	P-3	37	P ₀	B-8	147	Y ₅	J-16	101
DA ₁₂	G-2	16	DB ₂₂	R-1	39	P ₁	A-7	148	Y ₆	K-16	100
DA ₁₃	H-2	18	DB ₂₃	T-2	41	P ₂	A-8	149	Y ₇	K-15	99
DA ₁₄	J-1	20	DB ₂₄	U-3	45	P ₃	B-7	150	Y ₈	M-15	96
DA ₁₅	K-3	24	DB ₂₅	R-4	47	P ₄	C-6	151	Y ₉	M-17	95
DA ₁₆	L-2	28	DB ₂₆	U-4	49	P ₅	B-6	152	Y ₁₀	N-16	94
DA ₁₇	M-2	30	DB ₂₇	U-5	51	PA ₀	D-3	5	Y ₁₁	M-16	93
DA ₁₈	N-3	32	DB ₂₈	R-6	53	PA ₁	K-2	25	Y ₁₂	N-15	92
DA ₁₉	N-2	34	DB ₂₉	U-6	55	PA ₂	U-1	43	Y ₁₃	P-16	90
DA ₂₀	P-2	36	DB ₃₀	U-8	57	PA ₃	T-9	61	Y ₁₄	R-17	89
DA ₂₁	R-2	38	DB ₃₁	R-8	59	PB ₀	D-2	6	Y ₁₅	R-16	88
DA ₂₂	R-3	40	GND, ECL	G-3	21	PB ₁	L-1	26	Y ₁₆	T-17	86
DA ₂₃	T-1	42	GND, ECL	R-11	64	PB ₂	U-2	44	Y ₁₇	U-16	84
DA ₂₄	T-3	46	GND, ECL	G-17	104	PB ₃	U-9	62	Y ₁₈	T-16	83
DA ₂₅	T-4	48	GND, ECL	G-15	104	PERR	F-17	111	Y ₁₉	R-15	82
DA ₂₆	R-5	50	GND, ECL	G-16	104	PY ₀	D-17	115	Y ₂₀	U-15	81
DA ₂₇	T-5	52	GND, ECL	C-11	143	PY ₁	E-16	114	Y ₂₁	T-15	80
DA ₂₈	T-6	54	GND, TTL	T-12	72	PY ₂	F-15	113	Y ₂₂	U-14	77
DA ₂₉	U-7	56	GND, TTL	R-14	79	PY ₃	E-15	112	Y ₂₃	T-13	76
DA ₃₀	T-7	58	GND, TTL	U-17	85	RS	B-14	128	Y ₂₄	U-13	75
DA ₃₁	T-8	60	GND, TTL	P-17	91	SLAVE	C-14	127	Y ₂₅	R-13	74
DB ₀	A-6	153	GND, TTL	K-17	98	V	B-16	121	Y ₂₆	U-12	73
DB ₁	A-5	155	GND, TTL	J-15	105	V _{CC} , ECL	R-7	63	Y ₂₇	T-11	70
DB ₂	A-4	157	GND, TTL	F-16	110	V _{CC} , ECL	L-16	103	Y ₂₈	U-10	69
DB ₃	C-4	159	GND, TTL	C-17	117	V _{CC} , ECL	L-15	103	Y ₂₉	U-11	68
DB ₄	A-3	161	HOLD	A-17	123	V _{CC} , ECL	L-17	103	Y ₃₀	T-10	67
DB ₅	B-2	163	I ₀	C-10	139	V _{CC} , ECL	C-7	144	Y ₃₁	R-10	66
DB ₆	A-1	1	I ₁	B-11	138	V _{CC} , ECL	L-3	22	Z	B-17	122

LOGIC SYMBOL



LS002911

METALLIZATION AND PAD LAYOUT



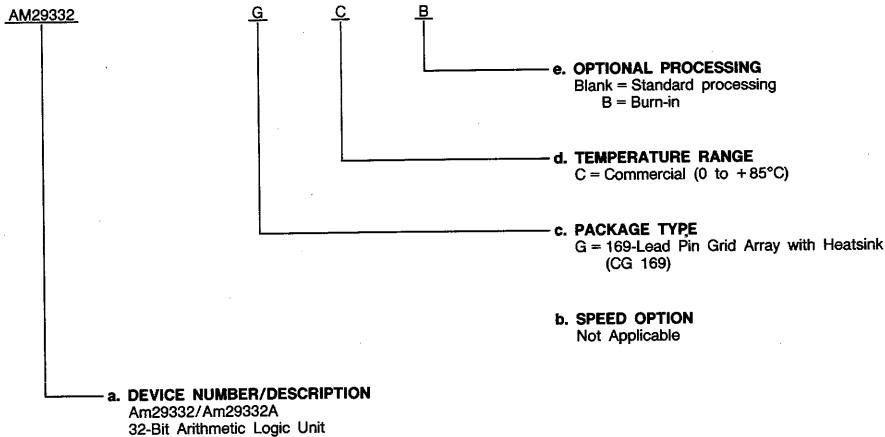
Die size: 367 x 387 mils
Gate Count: 5200

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Package Type**
- d. **Temperature Range**
- e. **Optional Processing**



Valid Combinations

Valid Combinations	
AM29332	GC, GCB
AM29332A	

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

BOROW Borrow (Input)

When HIGH, the Carry In and Carry Out are borrows for subtract operations.

C, Z, N, V, L Status (Input/Output)

When the Register Status pin is LOW, these pins give the Carry, Zero, Negative, Overflow and Link outputs of the ALU where applicable to the instruction being executed. When not applicable to the instruction being executed, or when the Register Status pin is HIGH, these pins give the outputs of the Carry, Zero, Negative, Overflow and Link bits of the internal Status Register. In Slave mode, C, Z, N, V and L become inputs.

CP Clock Input (Input)

Clocks internal registers (status, Q) at the LOW to HIGH transition, provided HOLD input is LOW.

DA₀ - DA₃₁ Data Input for DA-bus (Input)

Data input lines for operand A.

DB₀ - DB₃₁ Data Input for DB-bus (Input)

Data input lines for operand B.

HOLD Hold (Input, Active HIGH)

When HIGH, it inhibits the update of the status and Q registers.

I₀ - I₆ Instruction Inputs (Input)

Used to select the operation to be performed.

I₇ - I₈ Byte Width Inputs (Input)

Byte width inputs for byte boundary aligned operand instructions. Selects the sources for width and position inputs for variable field bit operands. If I₇ is LOW it selects the width input from pins W₄ - W₀. If I₇ is HIGH the width input is selected from the internal width register. Similarly if I₈ is LOW it selects the position inputs from pins P₅ - P₀ and if HIGH it selects input from the internal position register.

MCIn Macro Status Carry (Input)

External Carry input.

MLINK Macro Status Link (Input)

External link input.

M/ \bar{m} Macro/Micro Select (Input)

When HIGH, selects macro carry and macro link pins as input instead of micro carry and micro link from the micro-status register.

MSERR Master-Slave Error (Output)

When HIGH, this signal indicates that the master's and slave's data were not identical.

$\overline{OE-Y}$ Output Enable (Input, Active LOW)

When $\overline{OE-Y}$ is HIGH the Y-bus is disabled (three-stated).

P₀ - P₅ Position Inputs (Input)

Position input to select the position of the least significant bit of a field. Also indicates the amount by which data is to be shifted up (P₅ = LOW) or down (P₅ = HIGH) or rotated.

PA₀ - PA₃ Parity Input for DA-bus (Input)

Parity input for operand A on DA-bus (one per byte). Even parity is used for the Am29332.

PB₀ - PB₃ Parity Input for DB-bus (Input)

Parity input for operand B on DB-bus (one per byte).

PERR Parity Error (Input/Output)

When HIGH, indicates that a parity error was detected on the DA or DB inputs.

PY₀ - PY₃ Parity for Y-bus (Input/Output)

Parity output for data on Y-bus (one per byte). Even parity is used for the Am29332. In slave mode, PY₀ - PY₃ become inputs.

RS Register Status Mode Pin (Input)

Selects between ALU status (Register Status = LOW) or register status (Register Status = HIGH) on the C, Z, N, V and L outputs.

SLAVE Slave (Input)

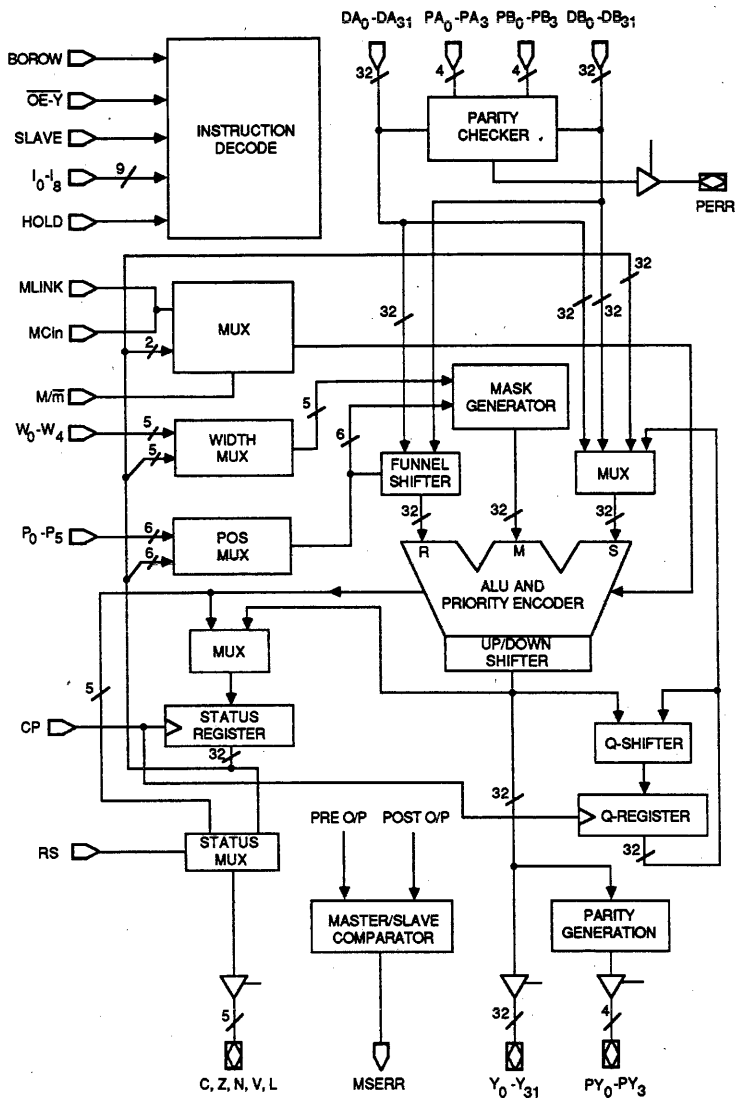
When HIGH, this pin puts the ALU in the slave mode. All output pins become input pins and signals on them are compared with the ALU's internally generated results. When $\overline{OE-Y}$ is HIGH, the Y₀ - Y₃₁ and PY₀ - PY₃ inputs are ignored. When the SLAVE pin is LOW, the ALU is put in master mode where outputs are generated as normal.

W₀ - W₄ Width Inputs (Input)

Width input to select the width of a contiguous bit field.

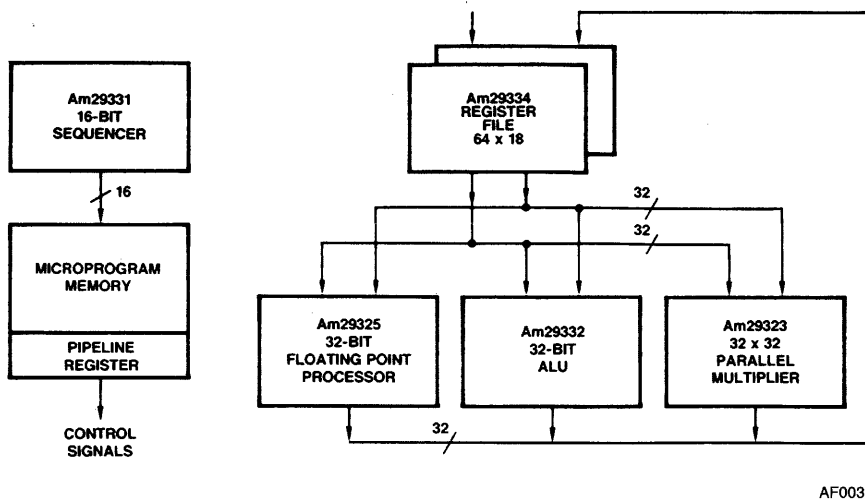
Y₀ - Y₃₁ Data Out/In Lines (Input/Output)

When $\overline{OE-Y}$ is LOW and the ALU is in the Master mode, the ALU result is enabled on the Y-bus. When $\overline{OE-Y}$ is HIGH, the Y-bus is three-stated. In Slave mode the Y-bus acts as external data input.



BD007031

Figure 1. Detailed Block Diagram



AF003480

Figure 2. Am29332 Family High-Performance System Block Diagram

PRODUCT OVERVIEW

The Am29332 is a 32-bit wide, high-performance, non-expandable Arithmetic Logic Unit (ALU). It has two 32-bit wide input ports (A and B) and one 32-bit wide output port (Y). These three ports provide flexibility and accessibility for high-performance processor designs. Dedicated input and output ports provide a flow-through architecture and avoid the penalty associated with switching the bus half-way through the cycle for input and output of data. The chip is designed for use with a dual-access RAM (Am29334) as a register file. In addition, the three-bus architecture facilitates the connection of other arithmetic units in parallel with the Am29332 for high-performance systems.

The Am29332 supports one-, two-, three-, and four-byte arithmetic operations. It also supports multiprecision arithmetic and multiple bit shifts. For logical operations, it can handle variable-length fields of up to 32 bits. The chip incorporates dedicated hardware to allow efficient implementation of a two bit-at-a-time (modified Booth) multiply algorithm, supporting signed and unsigned arithmetic data types. Similarly, hardware is provided to support a bit-at-a-time divide algorithm, also supporting signed and unsigned arithmetic data types. An internal 32-bit register (Q) is used by the multiply and divide hardware for double precision operands. For business applications, the Am29332 supports variable-length BCD arithmetic.

Field logical instructions operate on bit-fields taken from the A and B data inputs; they may be of variable width and starting position. A is normally the source input and B the destination input. In general, destination bits not falling within a specified field are passed by the ALU unchanged. Field width and position are specified either by direct inputs to the chip, or by entries in the status register. There are two kinds of field logical instructions - aligned and non-aligned. The first type of instruction assumes that source and destination fields are aligned and the operation is performed only for bits within the specified fields. In the second type of instruction, source and destination fields are normally non-aligned. However, it is always assumed that one field (either source or destination) is least-significant-bit (LSB) aligned.

If the destination field is LSB aligned then the source field is downshifted in order to make it LSB aligned as well. Down-

shifting is accomplished by making the 6-bit position input equal to the two's complement of the number of places the field is to be downshifted. If the source field is LSB aligned then it is upshifted in order to align it with the destination. Upshifting is accomplished by making the position inputs equal to the number of places the field is to be upshifted. Any other type of field operation is not allowed. Whenever the field crosses the word boundary, the portion not falling within the word boundary is ignored. This effect is useful when performing operations on fields that overlap two different words. Instructions to perform straightforward multiple-bit shifts (either up or down) are also provided. Additionally, it is possible to extract a bit-field from a word in one instruction, even if that field overlaps a word boundary.

The power and the flexibility of the processor comes partly from its ability to generate a mask to control the width of an operation for each instruction without any overhead. For all byte aligned instructions (three quarters of the instruction set), the mask is either 1, 2, 3 or 4 bytes wide and is generated from the byte width input ($i_8 - i_7$). For all field instructions the mask is of variable width and is generated from the position inputs ($P_0 - P_5$) and the width inputs ($W_0 - W_4$). Table 1 describes the position displacement from the position inputs and Table 2 the bit field from the width inputs.

TABLE 1. POSITION INPUTS AND BIT DISPLACEMENT

Inputs						Bit Displacement P
P ₅	P ₄	P ₃	P ₂	P ₁	P ₀	
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	2
:	:	:	:	:	:	:
0	1	1	1	1	1	31
1	0	0	0	0	0	-32
1	0	0	0	0	1	-31
:	:	:	:	:	:	:
1	1	1	1	1	1	-1

TABLE 2. WIDTH INPUTS AND BIT FIELD

Inputs					Bit Field w
W ₄	W ₃	W ₂	W ₁	W ₀	
0	0	0	0	0	32
0	0	0	0	1	1
0	0	0	1	0	2
⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	31

Whenever the width of the operand is less than 32-bits, all unselected bits from the inputs of the ALU are passed to the output without any modification. Depending upon the instruction type, unselected bits are taken from different sources. For example in all single operand instructions, bits from the source operand (from either A or B input) are passed in unselected bit positions. For two operand instructions, bits from the B input are passed in unselected bit positions. There are some exceptions which are explained in the instruction set section.

The processor has a 32-bit status register to indicate the status of different operations performed. The status register is loaded at the rising edge of the clock with new status unless the HOLD signal is HIGH. The bit position for each status bit is given in the functional description. The least significant byte of the status register holds the six position bits (PR₀ – PR₅). The two most significant bits of this byte may be read or loaded but are otherwise unused by the ALU. The second byte (bits 8 to 15) consists of the five width bits (WR₀ – WR₄) and three read-only bits that are a combinational function of other status bits, and which indicate useful branch conditions. The third byte consists of ALU status bits plus bits for high-speed multiply and divide. The most significant byte holds intermediate nibble carries for BCD operations. An extract-status instruction is provided which allows a Boolean value to be formed from any selected bit. This is particularly useful in machines employing a stack architecture. Instructions to save and restore the status register are provided. As the entire status of each instruction is stored in the status register, interrupts at any microinstruction boundary are feasible.

The processor has a 32-bit wide priority encoder to support floating-point and graphics operations. The priority encoder supports all byte aligned data types – the result is dependent upon the byte width specified. The result of a priority encode is also loaded into the position bits of the status register. The result of the prioritize operation can then be used in the following clock cycle, e.g., to normalize a floating-point number or to help detect the edge of a polygon in graphics applications.

To support system diagnostics, the Am29332 has a special "Master-Slave" mode. To use this mode, two chips are connected in parallel, and hence receive the same instructions and data. The master chip is used for the normal data path. However, in the slave chip, all outputs becomes inputs. The slave compares the outputs of the master with its own internally generated result. If the two do not match, the slave will activate an error signal.

As a further diagnostic aid, byte-wise parity checking is performed at both the A and B data inputs. The "parity" signal is activated if an error is detected. Parity bits (one per byte) are generated for the 32-bit output bus.

FUNCTIONAL DESCRIPTION

A detailed description of each functional block is given in the following paragraphs.

64-Bit Funnel Shifter

The 64-bit funnel shifter is a combinatorial network. The 64-bit input is formed from a combination of the A and B inputs. This may be left-shifted by up to 31 bits before being used by the ALU. The output of the shifter is the most significant 32 bits of the result. The 64-bit shifter can be used on either the A or B operands to perform barrel shifts (either up or down) or rotates. The operation is controlled by positioning operands properly at the input of the 64-bit up-shifter.

The number "n" by which the operand is shifted comes from two sources: the microprogram memory via the P₀ – P₅ pins or the internal register (byte 0 of the status register), PR₀ – PR₅, as selected by an instruction bit.

In general, the 6-bit position input, P₀ – P₅, takes a 6-bit two's complement number representing upshifts from 0 to 31 places (positive numbers) or downshifts from 1 to 32 places (negative numbers).

Mask Generator

The mask generator logic provides the ability to generate the appropriate mask for an operand of given width and position. The generation of the mask depends upon two types of instructions. The first type has byte boundary aligned operands (widths of either 1, 2, 3 or 4 bytes) with the least significant bit aligned to bit 0. The width of an operand is specified by the byte width inputs (I₈ and I₇) as shown in Table 3. The second type of instruction has operands of variable width (1 to 32 bits) and position. The operand is specified by the width inputs (W₀ – W₄) and the position inputs (P₀ – P₅) indicating the least significant bit position of the operand. Thus, in this type of instruction the operand may or may not be least significant bit aligned. Depending upon the type of instruction, the mask generator first generates a fence of all zeros starting from the least significant bit with the width specified either by the byte width or the width input fields. This fence can be upshifted by up to 31 bits by the 32-bit mask shifter. Whenever the mask is moved up over the 32-bit boundary, it does not wrap around. Instead, ONE's are inserted from the least significant end. This configuration provides the ability to operate on a contiguous field located anywhere in a word, or across a word boundary.

The mask generator can be used as a pattern generator by allowing the mask to pass through ALU (by using the PASS-MASK instruction). For example, a single-bit wide mask can be generated and by shifting it up by different amounts can give walking ONE or walking ZERO patterns for memory tests.

TABLE 3.

I ₈	I ₇	Width In Bytes
0	0	4
0	1	1
1	0	2
1	1	3

Arithmetic and Logical Unit

The ALU is a three input unit which uses the mask as a second or third operand in every instruction. The mask is used to merge two operands. For all selected bits (wherever the mask is 0), the desired operation specified by the instruction input is performed, and for all unselected bits either corresponding destination bits or zeros are passed through. The status of each operation (carry, negative, zero, overflow, link) applies to the result only over the specified width. For all byte aligned arithmetic and logical operations (first three quarters of the instruction set), the status is extracted from the appropriate

byte boundary. For all field operations (last quarter of the instruction set), the operand width is assumed to be 32 bits for status generation. The ZERO flag always indicates the status of all bits selected by the mask.

The actual width of the ALU is 34 bits. There are two extra bits used for the high speed signed and unsigned multiplication instructions. These two bits are automatically concatenated to the most-significant end of the ALU depending upon the width specified for the operation. Since the modified Booth algorithm requires a two-bit down-shift each cycle, these ALU bits generate the two most-significant bits of the partial product.

The ALU is capable of shifting data down by two bits for the multiplication algorithm, up by one bit for the divide algorithm and single-bit-up-shifts.

The processor is capable of performing BCD arithmetic on packed BCD numbers. The ALU has separate carry logic for BCD operations. This logic generates nibble carries (BCD digit carry) from propagate and generate signals formed from the A and B operands. In order to simplify the hardware while maintaining throughput, the BCD add and subtract operations are performed in two cycles. In the first cycle, ordinary binary addition or subtraction is performed and BCD nibble carries are generated. These are blocked from affecting the result at this stage, but are saved in the status register to be used later for BCD correction (NC₀ – NC₇). In the second cycle all BCD numbers are adjusted by examining the previously generated nibble carries. Since all the necessary information is stored in the status register, the processor can be interrupted after the first BCD cycle.

Priority Encoder

The priority encoder is provided to support floating-point arithmetic and some graphics primitives. The priority encoder takes up to 32 bits as input and generates a 5-bit wide binary code to indicate location of the most significant one in the operand. Input to the priority encoder comes from the input multiplexer, which masks all bits that the user does not want to participate in the prioritization. The priority encoder supports 8, 16, 24 and 32-bit operations depending upon the byte width specified. For each data type the priority encoder generates the appropriate binary weighted code. For example, when a byte width of two is specified ($l_7 - l_8 = 10$), the output of the encoder is zero when bit 15 is HIGH. However, if byte width of four is specified ($l_8 - l_7 = 00$), the output of encoder is 16 (decimal) if bit 15 is HIGH and bits 31 – 16 are LOW. Table 4 shows the output for each data type. If none of the inputs are HIGH or the most significant bit of the data type specified is HIGH, then the output is zero. The difference between these two cases is indicated by the Z-flag of the status register which is HIGH only if all inputs are zero.

Q-Register

The Q-register holds dividend and quotient bits for division, and multiplier and product bits for multiplication. During division, the contents of the Q-register are shifted left, a bit at a time, with quotient bits inserted into bit 0. During multiplication, the contents of the Q-register are shifted right, two bits at

a time, with product bits inserted into the most-significant two bits (according to the selected byte width). The Q-register may be loaded from the A or B inputs and read onto the Y bus.

Master-Slave Comparator

All ALU outputs (except MSERR) employ three-state buffers. The master-slave comparator compares the input and output of each buffer. Any difference causes the MSERR signal to be made true. In Slave mode, all output buffers are disabled. Outputs from a second ALU may then be connected to the equivalent pins of the first. The comparator in the slave will then detect any difference in the results generated by the two. When the Y bus is three-stated by making Output-Enable false, the Y bus master-slave comparators are disabled.

Parity Logic

For each byte of the DA and DB inputs there is an associated parity bit (8 in all). If a parity error is detected on any byte, the Parity-Error signal is made true. Four parity signals (one per byte) are also generated for the Y bus outputs. EVEN parity is employed for the Am29332.

Status Register

All necessary information about operations performed in the ALU is stored in the 32-bit wide status register after every microcycle. Since the register can be saved, an interrupt can occur after any cycle. The status register can be loaded from either the A or B input of the chip and can be read out on the Y bus for saving in an external register file. For loading, the byte width indicates how many bytes are to be updated. The status register is only updated if the HOLD input is inactive.

Each byte of the status register holds different types of information (see Figure 3). The least significant byte (bits 0 to 7) holds eight position bits (PR₀ – PR₇) for the data shifter. The two most significant bits are not used. The next most significant byte (bits 8 to 15) holds the 5-bit width field (WR₀ – WR₄) for the mask generator. The three most-significant bits of that byte (bits 13 to 15) are read-only bits that represent three different conditions extracted from the other bits of the status register. They are $\bar{C} + Z$, $N \oplus V$, and $(N \oplus V) + Z$ for bits 13, 14 and 15 respectively. These bits can be read on the Y₀ pin by the extract-status instruction. The next byte contains all the necessary information generated by an ALU operation. The least-significant four bits (bits 16 to 19) hold carry, negative, overflow and zero flags. Bit 20 holds link information for single bit shifts and bits 21 and 22 are used by the multiply and divide instructions. The M flag holds the multiplier bit for the modified Booth algorithm or it holds the sign comparison result for the divide algorithm. The S flag holds the sign of the partial remainder for unsigned division. Both the flags (M and S) are provided as a part of the status register so that multiply and divide instructions can be interrupted at microinstruction boundaries. The most significant byte of the status register holds nibble carries for BCD arithmetic. Since BCD arithmetic is performed in two cycles, the nibble carries are saved in the first cycle and used in the second cycle. Since all the information is stored, BCD instructions are also interruptible at the microinstruction boundary.

TABLE 4.

Highest Priority Active Bit	Encoder Output
$l_7 - l_8 = 00$ (32-bit)	
None	0
31	0
30	1
29	2
28	3
.	.
.	.
1	30
0	31
$l_7 - l_8 = 01$ (8-bit)	
None	0
7	0
6	1
5	2
.	.
.	.
1	6
0	7
$l_7 - l_8 = 10$ (16-bit)	
None	0
15	0
14	1
13	2
12	3
.	.
.	.
1	14
0	15
$l_7 - l_8 = 11$ (24-bit)	
None	0
23	0
22	1
21	2
20	3
.	.
.	.
1	22
0	23

Status₀₋₇: Position Register

PR ₇	PR ₆	PR ₅	PR ₄	PR ₃	PR ₂	PR ₁	PR ₀
7	6	5	4	3	2	1	0

Status₈₋₁₂: Width Register
Status₁₃: $\bar{C} + Z$
Status₁₄: $N \oplus V$
Status₁₅: $(N \oplus V) + Z$

} Read Only

SIGNED LE	SIGNED LT	UNSIGNED LE	WR ₄	WR ₃	WR ₂	WR ₁	WR ₀
15	14	13	12	11	10	9	8

Status₁₆: Carry
Status₁₇: Negative
Status₁₈: Overflow
Status₁₉: Zero
Status₂₀: Link
Status₂₁: Multiply (and divide) Bit
Status₂₂: Sign Flag
Status₂₃: 0

0	S	M	L	Z	V	N	C
23	22	21	20	19	18	17	16

Status₂₄₋₃₁: Nibble Carries

NC ₇	NC ₆	NC ₅	NC ₄	NC ₃	NC ₂	NC ₁	NC ₀
31	30	29	28	27	26	25	24

Note: Overflow is defined as follows:
 $V = (\text{carry in to MSB}) \oplus (\text{carry out of MSB})$

Figure 3. ALU Status Register Bit Assignment

Am29332 INSTRUCTION SET

Data Types

The Am29332 supports the following data types:

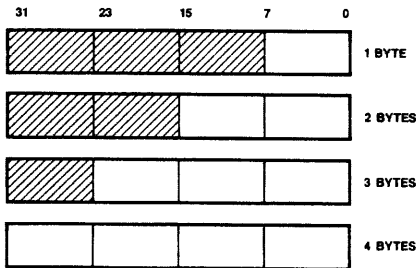
1. Integer
2. Binary-coded decimal
3. Variable-length bit field

The first two data types fall into the category of byte boundary aligned operands (Figure 4). The size of the operand could be 1 byte, 2 bytes, 3 bytes or 4 bytes. All operands are least significant bit (bit 0) aligned. The byte width is determined by bits I_8 and I_7 of the instruction as shown in Table 5.

TABLE 5.

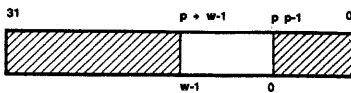
I_8	I_7	Width in Bytes
0	0	4
0	1	1
1	0	2
1	1	3

The third data type has operands of variable width (1 to 32 bits) as shown in Figure 4. The operand is specified by width inputs ($W_0 - W_4$) and position inputs ($P_0 - P_5$). The position inputs indicate the least significant bit position of the operand. Depending on bits I_8 and I_7 of the instruction, the width and position inputs can be selected from either the Status Register or the Width and Position Pins as shown in Table 6. A summary of the data types available is illustrated in Table 7.



TB000096

Byte Boundary Aligned Operands



TB000630

Variable-Length Bit Field

- p = Bit displacement of the least significant field with respect to bit 0.
 w = Width of bit field.

Figure 4. Data Types

TABLE 6.

I_8	I_7	Position		Width	
		Pins	Reg	Pins	Reg
0	0	X		X	
0	1	X			X
1	0		X	X	
1	1		X		X

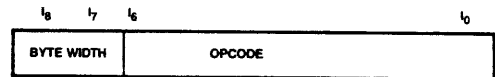
TABLE 7.

Data Type	Size	Range	
Integer		Signed	Unsigned
1 byte	8 bits	-128 to +127	0 to 255
2 bytes	16 bits	-2 ¹⁵ to +2 ¹⁵ - 1	0 to 2 ¹⁶ - 1
3 bytes	24 bits	-2 ²³ to 2 ²³ - 1	0 to 2 ²⁴ - 1
4 bytes	32 bits	-2 ³¹ to 2 ³¹ - 1	0 to 2 ³² - 1
BCD	1 to 4 bytes (8 digits)	Numeric, 2 digits per byte. Most-significant digit may be used for sign.	
Variable	1 to 32 bits	Dependent on position and width inputs.	

Instruction Format

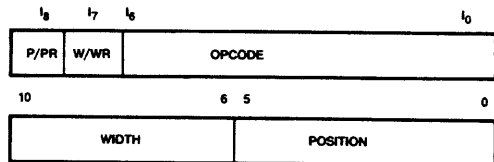
The Am29332 has two types of Instruction Formats:

1. Byte Boundary Aligned Instructions (FORMAT 1):



TB000098

2. Variable-Length Field Bit Instructions (FORMAT 2):



TB000099

For instructions that allow a field to be shifted up or down, $P_0 - P_5$ is a two's-complement number in the range -32 to +31 representing the direction and magnitude of the shift. For instructions that assume a fixed field position, $P_0 - P_4$ represent the position of the least-significant bit of the field and P_5 is ignored.

Instruction Classification

ALU instructions can be classified as follows:

A. Byte Boundary Aligned Operand Instructions:

1. Arithmetic
 - Binary, BCD
 - Multiply steps
 - Division steps (single and multiple precision)
2. Prioritize
3. Logical
4. Single-bit shifts
5. Data movement

B. Variable-Length Bit Field Operand Instructions:

1. N-bit shifts and rotates
2. Bit manipulations
3. Field logical operations (aligned, non-aligned, extract)
4. Mask generation

Three-fourths of the ALU instructions apply to operands that are byte boundary aligned. For these instructions, two orthogonal issues are the width of the operand (in bytes) and the contents of the high order unselected bytes on the Y bus. As mentioned earlier, the width of the operand is specified by I_8 and I_7 . With the exception of a few instructions, the unselected bytes are assigned values as follows: for single operand instructions, unselected bytes are passed unchanged from the source (A or B). For two operand instructions, unselected bytes are passed unchanged from the destination (B input).

In the last quarter of the instruction set, the width of the operand is from 1 to 32 bits (based on the width input) for field operations, 32 bits for N-bit shift operations and 1-bit for bit-oriented operations. In the case of field-aligned and single-bit operands, the position bits ($P_0 - P_4$) determine the least significant bit of the operand. In the case of N-bit shifts and field non-aligned operands, the position bits $P_0 - P_5$ is a 6-bit signed integer determining the magnitude and direction of the shift.

Flags

Byte-Aligned Instructions

The zero flag always looks only at the selected bytes:

$$Z \leftarrow (Y \text{ and bytemask (byte width)} = 0)$$

Similarly, $N \leftarrow$ sign bit (Y, byte width), where the function "sign-bit" returns bit 7, 15, 23, or 31 of the first argument for byte widths 01, 10, 11, or 00 respectively.

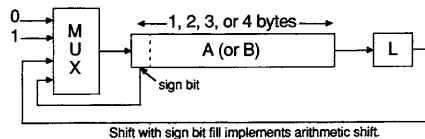
Also, $C \leftarrow$ carry (byte width) returns the carry from the appropriate byte boundary, and:

$$V \leftarrow \text{overflow (byte width)} = (\text{carry into MSB}) \oplus (\text{carry out of MSB})$$

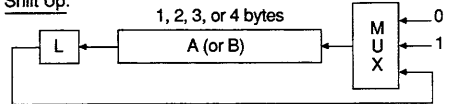
returns the overflow from the appropriate byte boundary.

The link (L) flag is generally loaded with the bit moved out of the highest selected byte in the case of upshifts, or the bit moved out of the least significant byte for downshifts. Figure 5 shows the shift operation using link bit. Other status flags have specialized uses, explained in the following sections.

Shift Down:



Shift Up:



DF006190

Figure 5. Upshift/Downshift Using Link Bit

Variable-Length Field Instruction:

Generally, only N and Z are affected. N takes the most-significant bit of the 32-bit result (i.e., $N \leftarrow Y_{31}$). Z detects zeros in the selected field of the result (i.e., $Z \leftarrow (Y \text{ and bitmask (position, width)} = 0)$).

Output Select

The Register Status pin, RS, may be used to switch the C, Z, N, V, and L output pins between the direct output of the ALU and the outputs of the corresponding bits in the status register. If the direct status output is selected, then for instructions that do not affect a particular flag (e.g., carry for logical arithmetic) that output will reflect the state of its corresponding bit in the status register. Similarly, when the HOLD signal is made HIGH, the C, Z, N, V and L pins will be made equal to the contents of the status register, regardless of the RS input.

INSTRUCTION SET SUMMARY

Operand Size: Variable Byte Width: 1, 2, 3, 4 Bytes

Type	Operation	Data Type
Arithmetic	<ul style="list-style-type: none"> ● Increment by one, two, four ● Decrement by one, two, four ● Add, addc (carry = macro/micro) ● Sub, subr ● Subc, subrc (carry/borrow) ● BCD sum and difference correct steps 	Binary Integer and BCD
	<ul style="list-style-type: none"> ● Negate (two's complement) ● Multiply steps (modified Booth) ● Divide steps (non-restoring) <div style="text-align: right; margin-right: 20px;">} (Signed and unsigned)</div>	Binary Integer
Prioritize	<ul style="list-style-type: none"> ● Prioritize 	Binary
Logical	<ul style="list-style-type: none"> ● Not, OR, AND, XOR, XNOR, zero, sign 	Binary
Single-Bit Shifts	<ul style="list-style-type: none"> ● Upshift with 0, 1, link fill ● Downshift with 0, 1, link, sign fill <div style="text-align: right; margin-right: 20px;">} (Single and double precision)</div>	Binary
Data Movement	<ul style="list-style-type: none"> ● Zero extend ● Sign extend ● Pass-status, Q-Reg ● Load-status, Q-Reg ● Merge 	Binary

Operand Size: 32 Bits

Type	Operation	Data Type
N-Bit Shifts N-Bit Rotates	<ul style="list-style-type: none"> ● Upshift by 0 to 31 bits with 0 fill ● Downshift by 1 to 32 bits with 0, sign fill ● Rotate by 0 to 31 bits 	Binary

Operand Size: Single Bit

Type	Operation	Data Type
Bit Manipulation	<ul style="list-style-type: none"> ● Extract ● Set ● Reset 	Binary

Operand Size: Variable Length Bitfield: 1 to 32 Bits

Type	Operation	Data Type
Field Logical (aligned and non-aligned)	<ul style="list-style-type: none"> ● Not, OR, XOR, AND, extract, insert 	Binary
Mask	<ul style="list-style-type: none"> ● Pass-mask 	Binary

INSTRUCTION SET GLOSSARY
(Sorted by Opcode in Hex Notation)

Opcode	Name	Opcode	Name	Opcode	Name	Opcode	Name
00	ZERO-EXTA	20	DN1-0F-A	40	AND	60	NB-SN-SHA
01	ZERO-EXTB	21	DN1-0F-B	41	XNOR	61	NB-SN-SHB
02	SIGN-EXTA	22	DN1-0F-AQ	42	ADD	62	NB-0F-SHA
03	SIGN-EXTB	23	DN1-0F-BQ	43	ADDC	63	NB-0F-SHB
04	PASS-STAT	24	DN1-1F-A	44	SUB	64	NBROT-A
05	PASS-Q	25	DN1-1F-B	45	SUBC	65	NBROT-B
06	LOADQ-A	26	DN1-1F-AQ	46	SUBR	66	EXTBIT-A
07	LOADQ-B	27	DN1-1F-BQ	47	SUBRC	67	EXTBIT-B
08	NOT-A	28	DN1-LF-A	48	SUM-CORR-A	68	SETBIT-A
09	NOT-B	29	DN1-LF-B	49	SUM-CORR-B	69	SETBIT-B
0A	NEG-A	2A	DN1-LF-AQ	4A	DIFF-CORR-A	6A	RSTBIT-A
0B	NEG-B	2B	DN1-LF-BQ	4B	DIFF-CORR-B	6B	RSTBIT-B
0C	PRIOR-A	2C	DN1-AR-A	4C	-	6C	SETBIT-STAT
0D	PRIOR-B	2D	DN1-AR-B	4D	-	6D	RSTBIT-STAT
0E	MERGEA-B	2E	DN1-AR-AQ	4E	SDIVFIRST	6E	NOTF-AL-B
0F	MERGB-A	2F	DN1-AR-BQ	4F	UDIVFIRST	6F	PASSF-AL-B
10	DECR-A	30	UP1-0F-A	50	SDIVSTEP	70	NOTF-A
11	DECR-B	31	UP1-0F-B	51	SDIVLAST1	71	NOTF-AL-A
12	INCR-A	32	UP1-0F-AQ	52	MPDIVSTEP1	72	PASSF-A
13	INCR-B	33	UP1-0F-BQ	53	MPSDIVSTEP3	73	PASSF-AL-A
14	DECR2-A	34	UP1-1F-A	54	UDIVSTEP	74	ORF-A
15	DECR2-B	35	UP1-1F-B	55	UDIVLAST	75	ORF-AL-A
16	INCR2-A	36	UP1-1F-AQ	56	MPDIVSTEP2	76	XORF-A
17	INCR2-B	37	UP1-1F-BQ	57	MPUDIVSTP3	77	XORF-AL-A
18	DECR4-A	38	UP1-LF-A	58	REMCORR	78	ANDF-A
19	DECR4-B	39	UP1-LF-B	59	QUOCORR	79	ANDF-AL-A
1A	INCR4-A	3A	UP1-LF-AQ	5A	SDIVLAST2	7A	EXTF-A
1B	INCR4-B	3B	UP1-LF-BQ	5B	UMULFIRST	7B	EXTF-B
1C	LDSTAT-A	3C	ZERO	5C	UMULSTEP	7C	EXTF-AB
1D	LDSTAT-B	3D	SIGN	5D	UMULLAST	7D	EXTF-BA
1E	-	3E	OR	5E	SMULSTEP	7E	EXTBIT-STAT
1F	-	3F	XOR	5F	SMULFIRST	7F	PASS-MASK

TABLE 6-1. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status							
			Unsel	Sel	S	M	L	Z	V	N	C	
ZERO-EXTA	00	Zero Extend	0	A				*		*		
ZERO-EXTB	01		0	B				*		*		
SIGN-EXTA	02	Sign Extend	Sign	A				*		*		
SIGN-EXTB	03		Sign	B				*		*		
MERGEA-B	0E	Merge A with B	B	A Merge B				*		*		
MERGEB-A	0F	Merge B with A	A	B Merge A				*		*		

TABLE 6-2. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status Register	Status						
			Unsel	Sel		S	M	L	Z	V	N	C
PASS-STAT	04	Pass Status Register	B	S								
LDSTAT-A	1C	Load Status Register	S	A	A	+	+	+	+	+	+	+
LDSTAT-B	1D		S	B	B	+	+	+	+	+	+	+

TABLE 6-3. DATA MOVEMENT INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Q Register	Status						
			Unsel	Sel		S	M	L	Z	V	N	C
PASS-Q	05	Pass Q Register	B	Q								
LOADQ-A	06	Load Q	Q	A	A				*		*	
LOADQ-B	07		Q	B	B				*		*	

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

- Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 + = Updated only if byte width is 3 or 4
 * = Updated

Examples:

- 2, ZERO EXTB Pass lower two bytes of B to Y with zero fill on upper two bytes
 0, LOADQ-A Load all four bytes of A into Q Register pass updated Q Register to Y

TABLE 7. LOGICAL INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status								
			Unsel	Sel	S	M	L	Z	V	N	C		
NOT-A	08	One's Complement	A	\bar{A}				*	*				
NOT-B	09		B	\bar{B}				*	*				
ZERO	3C	Pass Zero	B	0				1		0			
SIGN	3D	Pass Sign	B	$0(N=0); -1(N=1)$				N					
OR	3E	OR	B	A OR B				*	*	*	*		
XOR	3F	EXOR	B	A XOR B				*	*	*	*		
AND	40	AND	B	A AND B				*	*	*	*		
XNOR	41	XNOR	B	A XNOR B				*	*	*	*		

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Examples:

- 2, NOT-A Complement low order two bytes of A and output to Y with high order two bytes of A uncomplemented.
- 1, AND AND first byte of A and B. Output to Y with high three bytes of B.

TABLE 8-1. SINGLE-BIT SHIFT INSTRUCTIONS (SINGLE PRECISION)

Mnemonics	Code	Description	Y Output		Status							
			Unsel	Sel	S	M	L	Z	V	N	C	
DN1-0F-A	20	Downshift, Zero Fill	A	$Y_i = A_{i+1}, Y_{msb} = 0$			*	*		*		
DN1-0F-B	21		B	$Y_i = B_{i+1}, Y_{msb} = 0$			*	*		*		
DN1-1F-A	24	Downshift, One Fill	A	$Y_i = A_{i+1}, Y_{msb} = 1$			*	*		*		
DN1-1F-B	25		B	$Y_i = B_{i+1}, Y_{msb} = 1$			*	*		*		
DN1-LF-A	28	Downshift, Link Fill	A	$Y_i = A_{i+1}, Y_{msb} = L$			*	*		*		
DN1-LF-B	29		B	$Y_i = B_{i+1}, Y_{msb} = L$			*	*		*		
DN1-AR-A	2C	Downshift, Sign Fill	A	$Y_i = A_{i+1}, Y_{msb} = N$			*	*		*		
DN1-AR-B	2D		B	$Y_i = B_{i+1}, Y_{msb} = N$			*	*		*		
UP1-0F-A	30	Upshift, Zero Fill	A	$Y_i = A_{i-1}, Y_0 = 0$			*	*	*	*		
UP1-0F-B	31		B	$Y_i = B_{i-1}, Y_0 = 0$			*	*	*	*		
UP1-1F-A	34	Upshift, One Fill	A	$Y_i = A_{i-1}, Y_0 = 1$			*	*	*	*		
UP1-1F-B	35		B	$Y_i = B_{i-1}, Y_0 = 1$			*	*	*	*		
UP1-LF-A	38	Upshift, Link Fill	A	$Y_i = A_{i-1}, Y_0 = L$			*	*	*	*		
UP1-LF-B	39		B	$Y_i = B_{i-1}, Y_0 = L$			*	*	*	*		

Note: 1. These instructions use the byte aligned instruction format (FORMAT 1).

Example:

- 2, UP1-1F-A Shift lower two bytes of A up one bit. Set LSB to 1. Fill unselected bytes to upper two bytes of A.

TABLE 8-2. SINGLE-BIT SHIFT INSTRUCTIONS (DOUBLE PRECISION)

Mnemonics	Code	Description	Y Output & Q Register		Status						
			Selected Bytes		S	M	L	Z	V	N	C
DN1-0F-AQ	22	Downshift, Zero Fill	0 → A → Q 2)				*	*		*	
DN1-0F-BQ	23		0 → B → Q 3)				*	*		*	
DN1-1F-AQ	26	Downshift, One Fill	1 → A → Q 2)				*	*		*	
DN1-1F-BQ	27		1 → B → Q 3)				*	*		*	
DN1-LF-AQ	2A	Downshift, Link Fill	L → A → Q 2)				*	*		*	
DN1-LF-BQ	2B		L → B → Q 3)				*	*		*	
DN1-AR-AQ	2E	Downshift, Sign Fill	N → A → Q 2)				*	*		*	
DN1-AR-BQ	2F		N → B → Q 3)				*	*		*	
UP1-0F-AQ	32	Upshift, Zero Fill	A ← Q ← 0 2)				*	*	*	*	
UP1-0F-BQ	33		B ← Q ← 0 3)				*	*	*	*	
UP1-1F-AQ	36	Upshift, One Fill	A ← Q ← 1 2)				*	*	*	*	
UP1-1F-BQ	37		B ← Q ← 1 3)				*	*	*	*	
UP1-LF-AQ	3A	Upshift, Link Fill	A ← Q ← L 2)				*	*	*	*	
UP1-LF-BQ	3B		B ← Q ← L 3)				*	*	*	*	

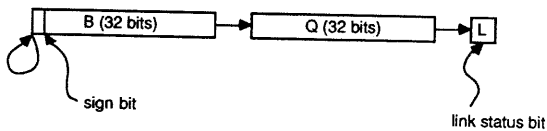
Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. Y Unselected byte from A, Q Unselected byte unchanged.
 3. Y Unselected byte from B, Q Unselected byte unchanged.

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

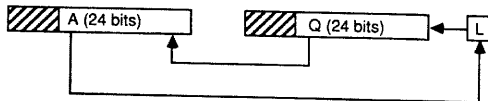
0, DN1-AR-BQ

Shift 64 bits (all 32 bits of both B and Q) down by one bit. LSB of B fills MSB of Q. MSB of B set to sign bit (bit N of status register).



3, UP1-LF-AQ

Shift 48 bits (24-bits of A and 24-bits of Q) up by one bit. MSB of 24-bit Q fills LSB of A. MSB of 24-bit A sets link status bit. LSB of Q is filled with original link value.



DF006200

TABLE 9. PRIORITIZE INSTRUCTIONS

Mnemonics	Code	Description	Y Output	Status						
				S	M	L	Z	V	N	C
PRIOR-A	0C	Prioritization	Location of Highest 1 Bit				*			
PRIOR-B	0D						*			

- Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. Priority also loaded into STATUS <7:0>
 3. Refer to Table 4.

Legend: A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

3, PRIOR-A Value placed on Y is 2



Assume A is

01001011	00100010	00000000	00000000
----------	----------	----------	----------

TABLE 10-1. ARITHMETIC INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
NEG-A	0A	Two's Complement	A	$\bar{A} + 1$				*	*	*	*
NEG-B	0B		B	$\bar{B} + 1$				*	*	*	*
INCR-A	12	Increment by One	A	A + 1				*	*	*	*
INCR-B	13		B	B + 1				*	*	*	*
INCR2-A	16	Increment by Two	A	A + 2				*	*	*	*
INCR2-B	17		B	B + 2				*	*	*	*
INCR4-A	1A	Increment by Four	A	A + 4				*	*	*	*
INCR4-B	1B		B	B + 4				*	*	*	*
DECR-A	10	Decrement by One	A	A - 1				*	*	*	*
DECR-B	11		B	B - 1				*	*	*	*
DECR2-A	14	Decrement by Two	A	A - 2				*	*	*	*
DECR2-B	15		B	B - 2				*	*	*	*
DECR4-A	18	Decrement by Four	A	A - 4				*	*	*	*
DECR4-B	19		B	B - 4				*	*	*	*

- Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. Borrow, rather than carry, is generated if BOROW is HIGH (borrow = carry).
 3. Nibble bits are set by these instructions. NEG-A (or NEG-B) and DIFF-CORR may be used to form 10's complement of a BCD number. Use SUM-CORR (for increment) or DIFF-CORR (for decrement) to increment or decrement a BCD number.

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

2, DECR4-A Decrement lower two bytes of A by 4

TABLE 10-2. ARITHMETIC INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
ADD	42	Add	B	A + B				*	*	*	*
ADDC	43	Add with Carry	B	A + B + C 6)				*	*	*	*
SUB	44	Subtract	B	A + \overline{B} + 1				*	*	*	*
SUBR	46		B	B + \overline{A} + 1				*	*	*	*
SUBC	45	Subtract with Carry	B	A + \overline{B} + 1 + C 2) 6)				*	*	*	*
SUBRC	47		B	B + \overline{A} + 1 + C 2) 6)				*	*	*	*
SUM-CORR-A	48	Correct BCD Nibbles for Addition	A	Corrected A 3)				*	*	*	*
SUM-CORR-B	49		B	Corrected B 3)				*	*	*	*
DIFF-CORR-A	4A	Correct BCD Nibbles for Subtraction	A	Corrected A 3)				*	*	*	*
DIFF-CORR-B	4B		B	Corrected B 3)				*	*	*	*

- Notes: 1. These instructions use the byte aligned instruction format (FORMAT 1).
 2. BOROW is LOW. For subtract operations, a borrow rather than a carry is stored in STATUS if BOROW is HIGH. Carry is always generated for ADD regardless of BOROW.
 3. First, the nibble carries NC₀ - NC₇ are tested. Any nibble carry/borrow that is set to 1 generates "6" internally as a correction word and then the correction word is added (SUM-CORR-) or subtracted (DIFF-CORR-) from the operand. NC₀ - NC₇ are not affected by this operation.
 4. Use SUM-CORR or DIFF-CORR to add or subtract a BCD number.
 5. Use ADDC, SUBC, or SUBRC to perform operations on integers longer than 32 bits.
 6. Carry bit is obtained from MCin if M/m is HIGH. Otherwise, carry is obtained from the C status bit.

Legend: Unsel = Unselected Byte(s)
 Sel = Selected Byte(s)
 A = A Input
 B = B Input
 Q = Q Register

* = Updated only if byte width is 3 or 4

Example:

0, ADD Add two 32-bit two's-complement integers

TABLE 11-1. DIVIDE INSTRUCTIONS (Aligned Format)

Name	I ₆ - I ₀ Code	Description	Source for Unselected Bytes	Output	Status						
					S	M	L	Z	V	N	C
Signed Divide Steps											
SDIVFIRST	4 E	First Instruction for Signed Divide	B	Y, Q	*	*	*	*		*	
SDIVSTEP	5 0	Iterate Step (#bits - 1 times)	B	Y, Q		*	*	*		*	*
SDIVLAST1	5 1	Last Divide Instruction Unless	B	Y, Q		*		*		*	*
SDIVLAST2	5 A	Dividend & Remainder Negative	B	Y				*			
Unsigned Divide Steps											
UDIVFIRST	4 F	First Instruction for Unsigned Divide	B	Y, Q			*	*		*	
UDIVSTEP	5 4	Iterate Step (#bits - 1 times)	B	Y, Q	*	*	*	*			*
UDIVLAST	5 5	Last Instruction	B	Y, Q	0	*		*		*	*
Multiprecision Divide Steps											
MPDIVSTEP1	5 2	First Instruction	B	Y, Q							
MPDIVSTEP2	5 6	Executed 0 Times for Double	B	Y, Q							
MPSDIVSTEP3	5 3	Last Instruction of Inner Loop	B	Y, Q							
MPUDIVSTEP3	5 7	Used for Unsigned Divide	B	Y, Q							
Correction Steps											
REMCORR	5 8	Correct Remainder After Divide	B	Y							*
QUOCORR	5 9	Correct Quotient After Divide	B	Y						*	*

TABLE 11-2. EXAMPLE CODING FORM (Signed Division)

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	
CONT				2	LOADQ-A			R2			1
CONT				0	SIGN					R3	0
FOR_D	15			2	SDIVFIRST			R4	R3	R3	0
DJMP_S				2	SDIVSTEP			R4	R3	R3	0
CONT				2	SDIVLAST1			R4	R3	R3	0
BRCC_D	DONE	Z									1
CONT				2	SDIVLAST2A			R4	R3	R3	0
CONT				2	PASS-Q					R1	0
CONT				2	QUOCORR				R1	R1	0
CONT				2	REMCORR			R4	R3	R3	0

Note: Divisor in A, Dividend in A
Quotient in Q, Remainder in B

Legend: A = A Input
B = B Input
S = Status Register
Q = Q Register
R1 = Quotient
R2 = Dividend
R3 = Remainder
R4 = Divisor

TABLE 12-1. MULTIPLY INSTRUCTIONS (Aligned Format)

Name	I ₆ - I ₀ Code	Description	Source for Unselected Bytes	Output	Status						
					S	M	L	Z	V	N	C
Signed Multiply Steps											
SMULFIRST	5 F	First multiply instruction	B	Y ⁽¹⁾							
SMULSTEP	5 E	Iterate step (#bits/2 - 1 steps)	B	Y ⁽¹⁾							
Unsigned Multiply Steps											
UMULFIRST	5 B	First multiply instruction	B	Y ⁽¹⁾		*					
UMULSTEP	5 C	Iterate step (#bits/2 - 1 steps)	B	Y ⁽¹⁾		*					
UMULLAST	5 D	Last multiply instruction	B	Y ⁽¹⁾				*			

TABLE 12-2. EXAMPLE CODING FORM (Unsigned Multiply)

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	
CONT				3	ZERO				R3	R3	0
CONT				3	LOADQ-A			R1			1
FOR_D	11 ₁₀			3	ULMULFIRST			R2	R3	R3	0
DJMP_S				3	UMULSTEP			R2	R3	R3	0
CONT				3	UMULLAST			R2	R3	R3	0
CONT				3	PASS-Q					R4	0

Note: 1. Put ALU output in B.
 2. Multiplicand in A, Multiplier in Q
 Product (HIGH) in B, Product (LOW) in Q

Legend: A = A Input
 B = B Input
 S = Status Register
 Q = Q Register
 R1 = Multiplier
 R2 = Multiplicand
 R3 = Product (HIGH)
 R4 = Product (LOW)

TABLE 13. SHIFT/ROTATE INSTRUCTIONS

Mnemonics	Code	Description	Y Output	Status						
				S	M	L	Z	V	N	C
NB-0F-SHA	62	Field Shift, Zero Fill	$Y_{i+p} = A_i, 0$ 2)				*	*		
NB-0F-SHB	63		$Y_{i+p} = B_i, 0$ 2)				*	*		
NB-SN-SHA	60	Field Shift, Sign Fill	$Y_{i+p} = A_i, N$ 2)				*	*		
NB-SN-SHB	61		$Y_{i+p} = B_i, N$ 2)				*	*		
NBROT-A	64	Field Rotate	$Y_i = A_{(i-p) \bmod 32}$ 3)				*	*		
NBROT-B	65		$Y_i = B_{(i-p) \bmod 32}$ 3)				*	*		

- Notes: 1. These instructions use the field instruction format (FORMAT 2).
 2. "p" stands for bit displacement from P₀-P₅ or from PR₀-PR₅ (-32 ≤ p ≤ 31).
 If p is positive, Y_{p-1} to Y₀ are equal to the fill bit.
 If p is negative, Y₃₁ to Y_{31+p+1} are equal to the fill bit.
 3. The sign of the position input is ignored for this instruction and P₀-P₄ are treated as a positive magnitude for a circular upshift.

Legend: A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Examples: *

NB-0F-SHA,,4 Shift A up 4 bits and zero fill

NB-0F-SHB,,-17 Shift B down 17 bits and sign fill

*Width field not used

TABLE 14-1. BIT-MANIPULATION INSTRUCTIONS

Mnemonics	Code	Description	Y Output		Status						
			Unsel	Sel	S	M	L	Z	V	N	C
SETBIT-A	68	Bit Set	A	$Y_i = A_i, Y_p = 1$				*	*		
SETBIT-B	69		B	$Y_i = B_i, Y_p = 1$				*	*		
RSTBIT-A	6A	Bit Reset	A	$Y_i = A_i, Y_p = 0$				*	*		
RSTBIT-B	6B		B	$Y_i = B_i, Y_p = 0$				*	*		
EXTBIT-A	66	Bit Extract	0	if p > 0, Y ₀ = A _p 2) if p < 0, Y ₀ = A _p				*	*		
EXTBIT-B	67		0	if p > 0, Y ₀ = B _p 2) if p < 0, Y ₀ = B _p				*	*		
EXTBIT-STAT	7E		0	if p > 0, Y ₀ = S _p 2) if p < 0, Y ₀ = S _p				*			

- Notes: 1. These instructions use the field instruction format (FORMAT 2).
 2. Y₃₁ to Y₁ are set to zero. "p" stands for the bit displacement from P₀-P₄ or from PR₀-PR₅. The sign of the position input is ignored.

TABLE 14-2. BIT-MANIPULATION INSTRUCTIONS

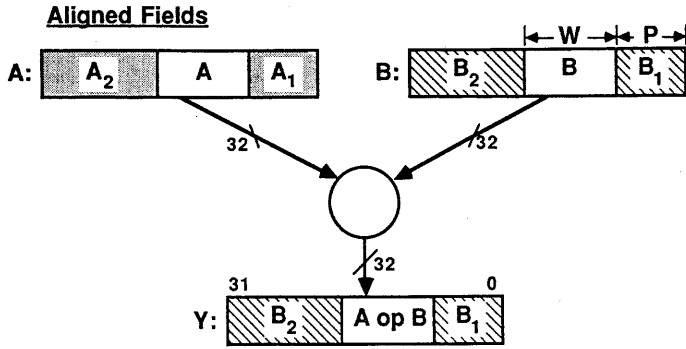
Mnemonics	Code	Description	Status Register	Y Output	Status						
					S	M	L	Z	V	N	C
SETBIT-STAT	6C	Status Bit Set	S _p = 1	S	*	*	*	*	*	*	*
RSTBIT-STAT	6D		S _p = 0	S	*	*	*	*	*	*	*

- Notes: 1. These instructions use the Field instruction format (FORMAT 2).
 2. "p" stands for the bit displacement from P₀-P₅ or from PR₀-PR₅.

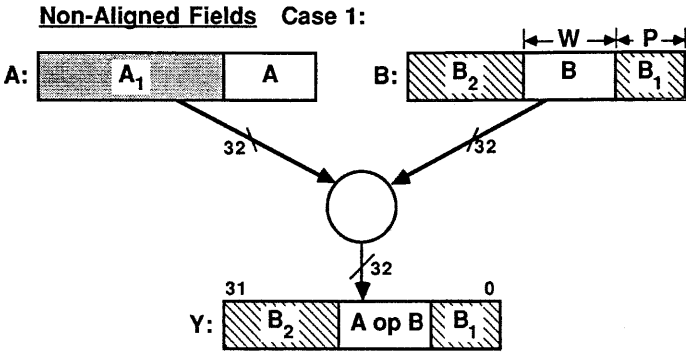
Legend: Unsel = Unselected field
 Sel = Selected field
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Examples:

RSTBIT-B,,3 3rd bit is set to 0 in B
 EXTBIT-STAT,,-4 4th bit in status register is extracted and inverted.

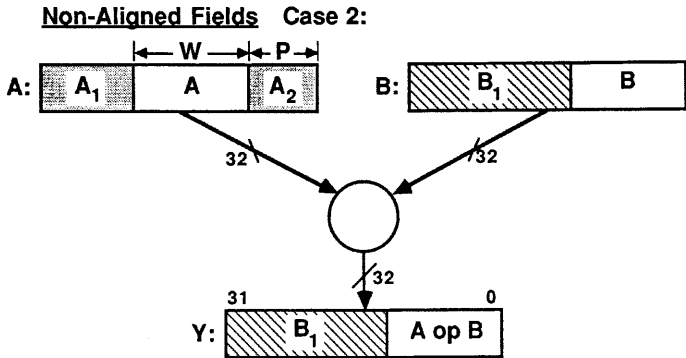


LD000140



If position $(P_0 - P_5) \geq 0$, A is LSB aligned
Width $(W_0 - W_4) = 1$ to 32

LD000151



If position $(P_0 - P_5) < 0$, B is LSB aligned
Width $(W_0 - W_5) = 1$ to 32

LD000161

Figure 6. Field Logical Operations

TABLE 15. FIELD LOGICAL INSTRUCTIONS

Mnemonics	Code	Description		Y Output		Status					
				Unsel	Sel	S	M	L	Z	V	N
PASSF-AL-A	73	Field Pass	3)	B	$Y_i = A_i$				*	*	
PASSF-AL-B	6F		3)	B	$Y_i = B_i$				*	*	
PASSF-A	72		4)	B	if $p \geq 0$, $Y_i = A_{i-p}$ if $p < 0$, $Y_{i- p } = \bar{A}_i$				*	*	
NOTF-AL-A	71	Field Complement	3)	B	$Y_i = \bar{A}_i$				*	*	
NOTF-AL-B	6E		3)	B	$Y_i = \bar{B}_i$				*	*	
NOTF-A	70		4)	B	if $p \geq 0$, $Y_i = \bar{A}_{i-p}$ if $p < 0$, $Y_{i- p } = \bar{A}_i$				*	*	
ORF-AL-A	75	Field OR	3)	B	$Y_i = A_i \text{ OR } B_i$				*	*	
ORF-A	74		4)	B	if $p \geq 0$, $Y_i = A_{i-p} \text{ OR } B_i$ if $p < 0$, $Y_{i- p } = \bar{A}_i \text{ OR } B_{i- p }$				*	*	
XORF-AL-A	77	Field XOR	3)	B	$Y_i = A_i \text{ XOR } B_i$				*	*	
XORF-A	76		4)	B	if $p \geq 0$, $Y_i = A_{i-p} \text{ XOR } B_i$ if $p < 0$, $Y_{i- p } = \bar{A}_i \text{ XOR } B_{i- p }$				*	*	
ANDF-AL-A	79	Field AND	3)	B	$Y_i = A_i \text{ AND } B_i$				*	*	
ANDF-A	78		4)	B	if $p \geq 0$, $Y_i = A_{i-p} \text{ AND } B_i$ if $p < 0$, $Y_{i- p } = \bar{A}_i \text{ AND } B_{i- p }$				*	*	
EXTF-A	7A	Field Extract	4) 5)	0	if $p \geq 0$, $Y_i = A_{i-p}$ if $p < 0$, $Y_{i- p } = \bar{A}_i$				*	*	
EXTF-B	7B			0	if $p \geq 0$, $Y_i = B_{i-p}$ if $p < 0$, $Y_{i- p } = \bar{B}_i$				*	*	
EXTF-AB	7C		0	6)				*	*		
EXTF-BA	7D		0	7)				*	*		

- Notes: 1. These instructions use the field instruction format (FORMAT 2).
 2. $p \leq i \leq p + w - 1$. "p" stands for position displacement from $P_0 - P_5$ or from $PR_0 - PR_5$ and "w" for the width of the bit field from $W_0 - W_4$ or $WR_0 - WR_4$. Whenever $p + w > 32$, operation takes place only over the portion of the field up to the end of the word. No wraparound occurs.
 3. This instruction uses the aligned format (see Figure 6).
 4. This instruction uses the unaligned field format (see Figure 6).
 $p \geq 0$: Case 1
 $p < 0$: Case 2
 5. If p is positive, the input is LSB aligned and Y output aligned at position.
 If p is negative, the input is aligned at |p| and Y output at LSB.
 6. Firstly, the concatenation of A(High Word) and B(Low Word) is rotated by the amount specified by the position (p). If p is positive, left-rotate is performed. If p is negative, right-rotate is performed. Secondly, the least significant bits on the Y output specified by the width (w) are extracted.
 7. Same as 6) except that B input is taken as a high word and A input as a low word.

Legend: Unsel = Unselected Field
 Sel = Selected Field
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

For all examples, assume STATUS (7:0) is -7 and STATUS (12:8) is 3.

1. 0,PASSF-AL-B,11,20 Pass B to Y and test if $B_{20} - B_{30}$ are all zero. Set Z status if so.

B: 10000000000000000101011100110100

Z set to 1 in this case

2. 3,XORF-A,, Exclusive-OR bits $A_7 - A_9$ with bits $B_0 - B_2$ and output to $Y_0 - Y_2$. Pass $B_3 - B_{31}$ to $Y_3 - Y_{31}$. Width and position values are obtained from STATUS(12:0).

A: 01101110001001000010111001101011

B: 00011100001010001100101001001001

$A_9 - 7 \oplus B_{2-0} = Y$: 00011100001010001100101001001101

TABLE 16. MASK INSTRUCTION

Mnemonics	Code	Description	Y Output		Status							
			Unsel	Sel	S	M	L	Z	V	N	C	
PASS-MASK	7F	Generate Mask	p_5	$Y_i = \bar{p}_5$								

Notes: 1. This instruction uses the field instruction format (FORMAT 2).

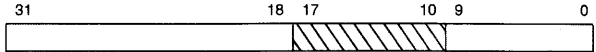
2. $p \leq i \leq p + w - 1$. "p" stands for the position displacement and "w" for the width of bit field.

- Legend: Unsel = Unselected Field
 Sel = Selected Field
 A = A Input
 B = B Input
 Q = Q Register
 * = Updated

Example:

Generates an 8-bit field mask pattern starting from bit position 10.

0, PASS-MASK, 8, 10



APPLICATIONS

Suggestions for Power and Ground Pin Connections

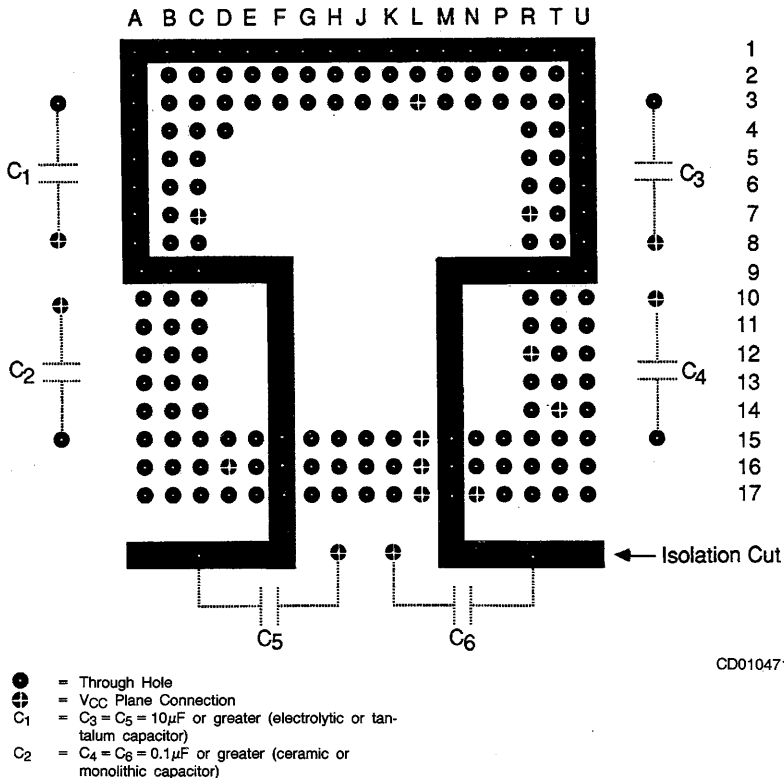
The Am29332 operates in an environment of fast signal rise times and substantial switching currents. Therefore, care must be exercised during circuit board design and layout, as with any high-performance component. The following is a suggested layout, but since systems vary widely in electrical configuration, an empirical evaluation of the intended layout is recommended.

The V_{CC} and $GNDT$ pins, which carry output driver switching currents, tend to be electrically noisy. The V_{CC} and $GNDE$ pins, which supply the ECL core of the device, tend to produce less noise, and the circuits they supply may be adversely affected by noise spikes on the V_{CC} plane. For this reason, it is best to provide isolation between the V_{CC} and V_{CC} pins, as well as independent decoupling for each. Isolating the $GNDE$ and $GNDT$ pins is not required.

Printed Circuit-Board Layout Suggestions

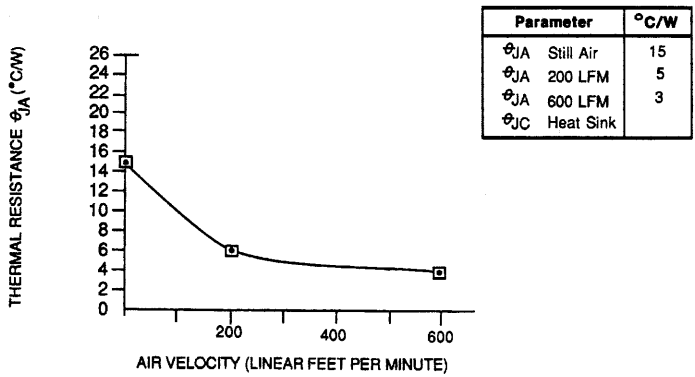
1. Use of a multi-layer PC board with separate power, ground, and signal planes is highly recommended.
2. All V_{CC} and V_{CC} pins should be connected to the V_{CC} plane. V_{CC} pins should be isolated from V_{CC} pins by means of a slot cut in the V_{CC} plane; see Figure 7. By physically separating the V_{CC} and V_{CC} pins, coupled noise will be reduced.
3. All $GNDE$ and $GNDT$ pins should be connected directly to the ground plane.
4. The V_{CC} pins should be decoupled to ground with a $0.1\text{-}\mu\text{F}$ ceramic capacitor and a $10\text{-}\mu\text{F}$ electrolytic capacitor, placed as closely to the Am29332 as is practical. V_{CC} pins should be decoupled to ground in a similar manner.

A suggested layout is shown in Figure 7.



CD010471

Figure 7. Suggested Printed Circuit-Board Layout



OP002241

Figure 8. Am29332 Thermal Characteristics (Typical)

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Temperature Under Bias - T_C -55 to +125°C
 Supply Voltage to Ground Potential
 Continuous -0.5 to +7.0 V
 DC Voltage Applied to Outputs
 for HIGH State -0.5 V to + V_{CC} Max.
 DC Input Voltage -0.5 to +5.5 V

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Case Devices
 Temperature (T_O) 0 to +85°C
 Supply Voltage V_{CC} +4.75 V to +5.25 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range

Parameter Symbol	Parameter Description	Test Conditions (Note 1)		Min.	Max.	Units
V_{OH}	Output HIGH Voltage	$V_{CC} = 4.75$ V, $V_{IN} = V_{IH}$ or V_{IL} , $I_{OH} = -1.2$ mA	All Outputs	-2.4		Volts
V_{OL}	Output LOW Voltage	$V_{CC} = 4.75$ V, $V_{IN} = V_{IH}$ or V_{IL} , $I_{OL} = 8$ mA	All Outputs		0.5	Volts
V_{IH}	Input HIGH Level (Guaranteed Logic HIGH Voltage)		All Inputs	2.0		Volts
V_{IL}	Input LOW Level (Guaranteed Logic LOW Voltage)		All Inputs		0.8	Volts
V_I	Input Clamp Voltage	$V_{CC} = 4.75$ V, $I_{IN} = -18$ mA	All Inputs		-1.5	Volts
I_{IL}	Input LOW Current	$V_{CC} = 5.25$ V, $V_{IN} = 0.5$ V	PY0-3, Y0-31		-0.55	mA
			I4-6		-1.50	
			I7-8		-1.00	
			SLAVE		-3.00	
			OE- \bar{Y}		-2.50	
			CLK		-2.00	
			C, Z, V, N, L; PERR		-0.55	
			Other		-0.50	
I_{IH}	Input HIGH Current	$V_{CC} = 5.25$ V, $V_{IN} = 2.4$ V	PY0-3, Y0-31		100	μ A
			I4-6		150	
			I7-8		100	
			SLAVE		300	
			OE- \bar{Y}		250	
			CLK		200	
			C, Z, V, N, L; PERR		100	
			Other		50	
I_I	Input HIGH Current	$V_{CC} = 5.25$ V, $V_{IN} = 5.5$ V	All Inputs		1.0	mA
I_{OZH}	Off State Output Current	$V_{CC} = 5.25$ V, $V_O = 2.4$ V	All Outputs Except MSERR		100	μ A
I_{OZL}		$V_{CC} = 5.25$ V, $V_O = 0.5$ V			-550	
I_{OS}	Output Short-Circuit Current (Note 2)	$V_{CC} = 5.75$ V, $V_O = 0.5$ V		-15	-50	mA
I_{CC}	Power Supply Current (Note 3)	$V_{CC} = 5.25$ V	$T_C = 0$ to 85°C		1800	mA
			$T_C = 85^\circ\text{C}$		1690	mA

Notes: 1. For conditions shown as Min. or Max., use the appropriate value specified under Operating Ranges for the applicable device type.
 2. Not more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.
 3. Measured with all inputs HIGH and outputs disabled.

SWITCHING CHARACTERISTICS over operating range

A. COMBINATIONAL PROPAGATION DELAYS

No.	From	To	Am29332	Am29332A	Unit
			Max. Delay	Max. Delay	
1	PA ₀ - PA ₃ , PB ₀ - PB ₃	PERR	19	16	ns
2	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	PERR	28	24	ns
3	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	PY ₀ - PY ₃	42	36	ns
4	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	Y ₀ - Y ₃₁	35	30	ns
5	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	C, Z, V, N, L	43	37	ns
6	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	MSERR	49	42	ns
7	I ₀ - I ₈	PY ₀ - PY ₃	53	45	ns
8	I ₀ - I ₈	Y ₀ - Y ₃₁	47	40	ns
9	I ₀ - I ₈	C, Z, V, N, L	48	41	ns
10	I ₀ - I ₈	MSERR	55	47	ns
11	W ₀ - W ₄	PY ₀ - PY ₃	40	34	ns
12	W ₀ - W ₄	Y ₀ - Y ₃₁	34	29	ns
13	W ₀ - W ₄	C, Z, V, N, L	35	30	ns
14	W ₀ - W ₄	MSERR	41	35	ns
15	P ₀ - P ₅	PY ₀ - PY ₃	48	41	ns
16	P ₀ - P ₅	Y ₀ - Y ₃₁	42	36	ns
17	P ₀ - P ₅	C, Z, V, N, L	43	37	ns
18	P ₀ - P ₅	MSERR	45	39	ns
19	CP	PY ₀ - PY ₃	47	40	ns
20	CP	Y ₀ - Y ₃₁	41	35	ns
21	CP	C, Z, V, N, L	42	36	ns
22	CP	STATUS REG.	20	17	ns
23	RS	C, Z, V, N, L	16	14	ns
24	MC _{in}	Y ₀ - Y ₃₁	31	27	ns
25	MC _{in}	C, Z, V, N, L	34	29	ns
26	MC _{in}	MSERR	37	32	ns
27	MLINK	Y ₀ - Y ₃₁	33	28	ns
28	MLINK	C, Z, V, N, L	37	32	ns
29	MLINK	MSERR	38	33	ns
30	M/ \bar{m}	Y ₀ - Y ₃₁	33	28	ns
31	M/ \bar{m}	C, Z, V, N, L	37	32	ns
32	M/ \bar{m}	MSERR	38	33	ns
33	BOROW	Y ₀ - Y ₃₁	33	28	ns
34	BOROW	C, Z, V, N, L	37	32	ns
35	BOROW	MSERR	38	33	ns
36	HOLD	C, Z, V, N, L	22	19	ns
37	HOLD	MSERR	29	25	ns
38	PY ₀ - PY ₃	MSERR	20	17	ns
39	Y ₀ - Y ₃₁	MSERR	19	16	ns
40	C, Z, V, N, L	MSERR	21	18	ns
41	PERR	MSERR	20	17	ns

SWITCHING CHARACTERISTICS (Cont'd.)

B. SETUP AND HOLD TIMES

No.	Parameter (Note 2)	For	With Respect To	Am29332	Am29332A	Unit
				Max. Value	Max. Value	
42	Input Data Setup	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	CP ↑	31	31	ns
43	Input Data Hold	DA ₀ - DA ₃₁ , DB ₀ - DB ₃₁	CP ↑	1	1	ns
44	Byte Width Setup	I ₇ - I ₈	CP ↑	30	30	ns
45	Byte Width Hold	I ₇ - I ₈	CP ↑	1	1	ns
46	Instruction Setup	I ₀ - I ₆	CP ↑	37	37	ns
47	Instruction Hold	I ₀ - I ₆	CP ↑	2	2	ns
48	Width Setup	W ₀ - W ₄	CP ↑	28	28	ns
49	Width Hold	W ₀ - W ₄	CP ↑	0	0	ns
50	Position Setup	P ₀ - P ₅	CP ↑	28	28	ns
51	Position Hold	P ₀ - P ₅	CP ↑	0	0	ns
52	Borrow Setup	BOROW	CP ↑	22	22	ns
53	Borrow Hold	BOROW	CP ↑	1	1	ns
54	Macro Carry Setup	MCin	CP ↑	21	21	ns
55	Macro Carry Hold	MCin	CP ↑	0	0	ns
56	Macro Link Setup	MLINK	CP ↑	22	22	ns
57	Macro Link Hold	MLINK	CP ↑	1	1	ns
58	Macro/Micro Setup	M/m	CP ↑	22	22	ns
59	Macro/Micro Hold	M/m	CP ↑	1	1	ns
60	Hold Mode Setup	HOLD	CP ↑	11	11	ns
61	Hold Mode Hold	HOLD	CP ↑	1	1	ns

C. MINIMUM CLOCK REQUIREMENTS

No.	Description	Am29332	Am29332A	Unit
		Max. Value	Max. Value	
62	Minimum Clock LOW Time	20	20	ns
63	Minimum Clock HIGH Time	20	20	ns

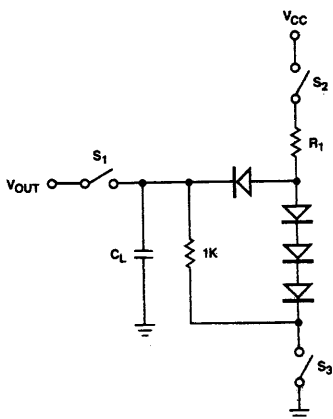
D. ENABLE AND DISABLE TIMES

No.	From	To	Description	Am29332	Am29332A	Unit
				Max. Delay	Max. Delay	
64	OE - Y	Y ₀ - Y ₃₁ , PY ₀ - PY ₃	Output Enable Time	25	25	ns
65	OE - Y	Y ₀ - Y ₃₁ , PY ₀ - PY ₃	Output Disable Time	25	25	ns
66	SLAVE	C, Z, V, N, L PERR	Slave Mode Enable Time	25	25	ns
67	SLAVE	Y ₀ - Y ₃₁ , PY ₀ - PY ₃ C, Z, V, N, L PERR	Slave Mode Disable Time	25	25	ns

Notes: 1. It is the responsibility of the user to maintain a case temperature of 85°C or less. AMD recommends an air velocity of at least 200 linear feet per minute over the heatsink.

2. See timing diagram for desired mode of operation to determine clock edge to which these setup and hold times apply.

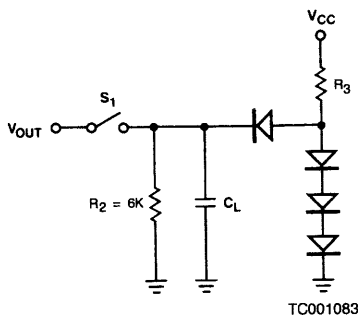
SWITCHING TEST CIRCUITS



TC001102

$$R_1 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OL} + \frac{V_{OL}}{1K}}$$

A. Three-State Outputs



TC001083

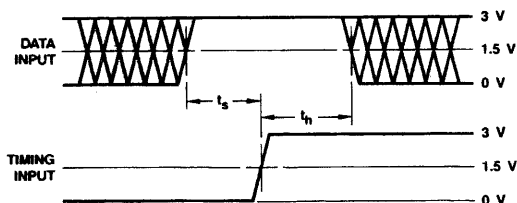
$$R_2 = \frac{2.4 \text{ V}}{I_{OH}}$$

$$R_1 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OL} + \frac{V_{OL}}{R_2}}$$

B. Normal Outputs

- Notes:
1. $C_L = 50 \text{ pF}$ includes scope probe, wiring and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for $tpZH$ test.
 4. S_1 and S_2 are closed while S_3 is open for $tpZL$ test.
 5. $C_L = 5.0 \text{ pF}$ for output disable tests.

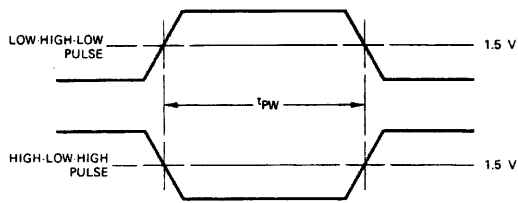
SWITCHING TEST WAVEFORMS



WFR02970

Setup, Hold, and Release Times

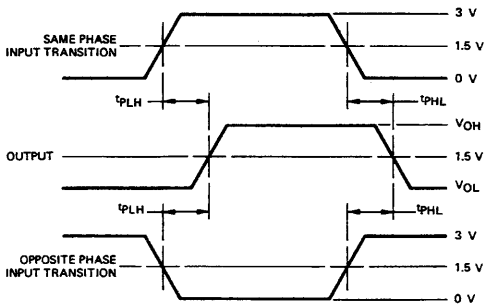
- Notes:
1. Diagram shown for HIGH data only. Output transition may be opposite sense.
 2. Cross hatched area is don't care condition.



WFR02790

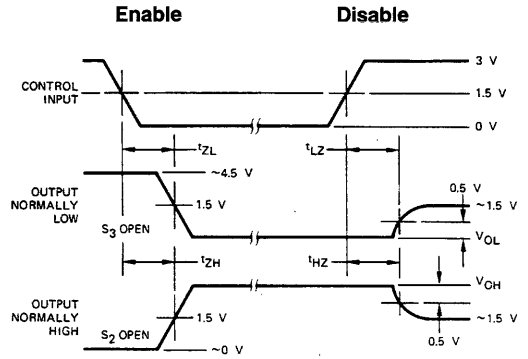
Pulse Width

SWITCHING TEST WAVEFORMS (Cont'd.)



WFR02980

Propagation Delay



WFR02660

Enable and Disable Times

- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
 2. S_1 , S_2 and S_3 of Load Circuit are closed except where shown.

Test Philosophy and Methods

The following points give the general philosophy that we apply to tests that must be properly engineered if they are to be implemented in an automatic environment. The specifics of what philosophies applied to which test are shown.

1. Ensure the part is adequately decoupled at the test head. Large changes in supply current when the device switches may cause function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an input transition, ground current may change by as much as 400 mA in 5 - 8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily.
4. Use extreme care in defining input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins that may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \geq 3$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. Capacitive Loading for AC Testing

Automatic testers and their associated hardware have stray capacitance that varies from one type of tester to another, but is generally around 50 pF. This, of course, makes it impossible to make direct measurements of parameters that call for a smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays" which measure the propagation delays into and out of the high impedance state and are usually specified at a load capacitance of 5.0 pF. In these cases, the test is performed at the higher load capacitance (typically 50 pF) and engineering correlations based on data taken with a bench set up are used to predict the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impossible to make measurements at both capacitances even though they may both be greater than the stray capacitance. In these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is predicted from engineering correlations based on data taken with a bench set up and the knowledge that certain DC measurements (I_{OH} , I_{OL} for example) have already been taken and are within specification. In some cases, special DC tests are performed in order to facilitate this correlation.

7. Threshold Testing

The noise associated with automatic testing, the long, inductive cables, and the high gain of bipolar devices when in the vicinity of the actual device threshold, frequently give rise to oscillations when testing high-speed speed circuits. These oscillations are not indicative of a reject device, but instead, of an overtaxed test system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" HIGH and LOW levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels rather than at V_{IL} Max. and V_{IH} Min.






8. AC Testing

Occasionally, parameters are specified that cannot be measured directly on automatic testers because of tester limitations. Data input hold times often fall into this category. In these cases, the parameter in question is guaranteed by correlating these tests with other AC tests that have been performed. These correlations are arrived at by the cognizant engineer by using data from precise bench measurements in conjunction with the knowledge that certain DC parameters have already been measured and are within specification.

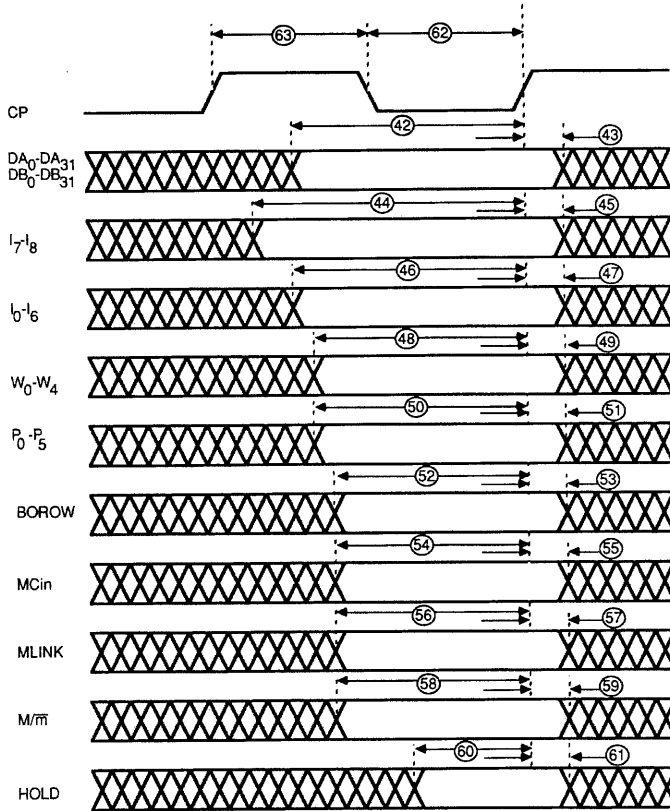
In some cases, certain AC tests are redundant since they can be shown to be predicted by other tests that have already been performed. In these cases, the redundant tests are not performed.

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

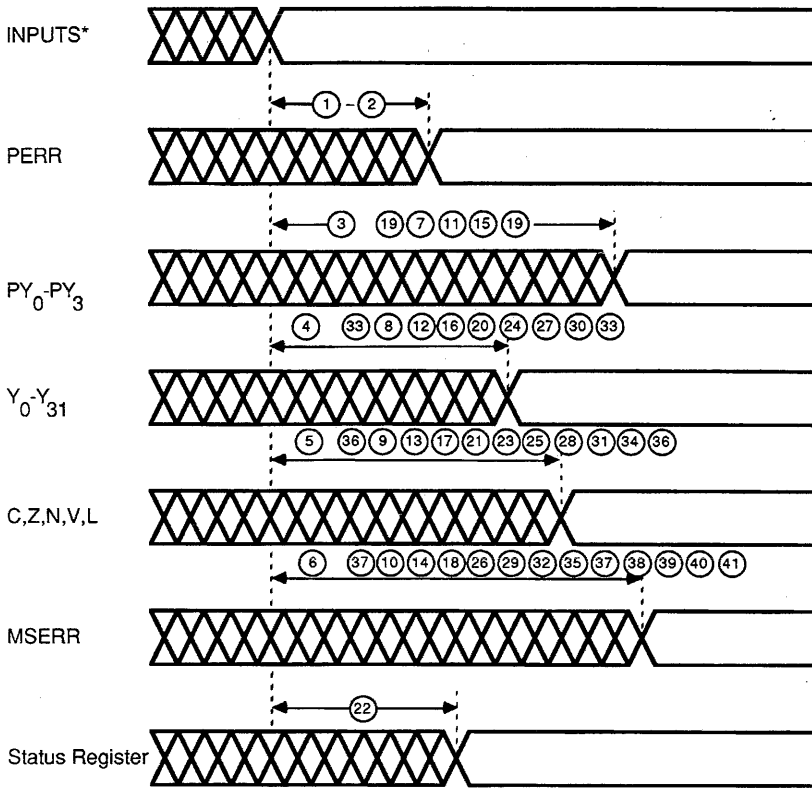
KS000010



WF023680

Setup and Hold Timing

SWITCHING WAVEFORMS (Cont'd.)

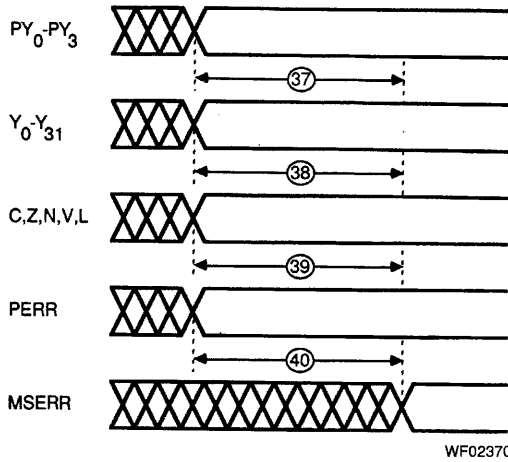


WF023691

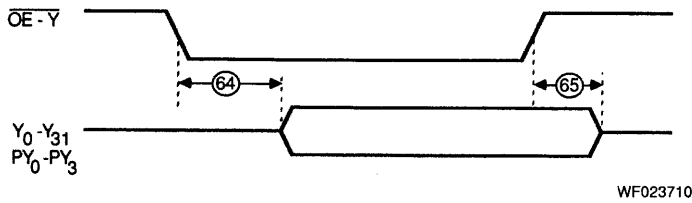
Propagation Delays (SLAVE = LOW)

Inputs: PA₀-PA₃, PB₀-PB₃, DA₀-DA₃₁, DB₀-DB₃₁, I₀-I₈, W₀-W₄, P₀-P₅, CP, RS, MCin, MLINK, M/m, BOROW, HOLD

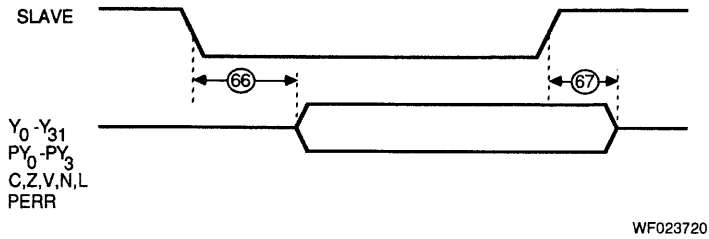
SWITCHING WAVEFORMS (Cont'd.)



Propagation Delay (SLAVE = HIGH)



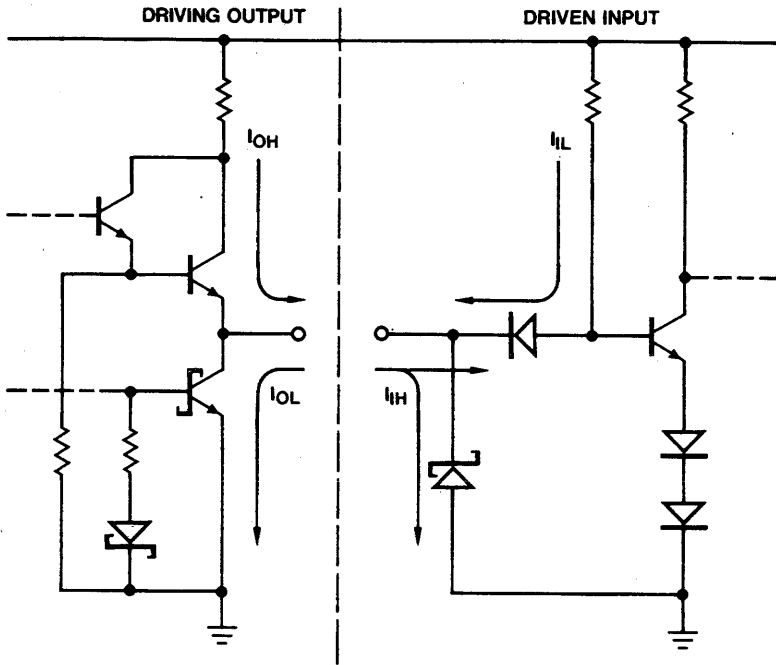
Enable/Disable I (SLAVE = HIGH)



Enable/Disable II ($\overline{OE-Y}$ = LOW)

INPUT/OUTPUT CIRCUIT DIAGRAM

(All Devices)



ICR00480

Am29334

Four-Port Dual-Access Register File



Am29334

DISTINCTIVE CHARACTERISTICS

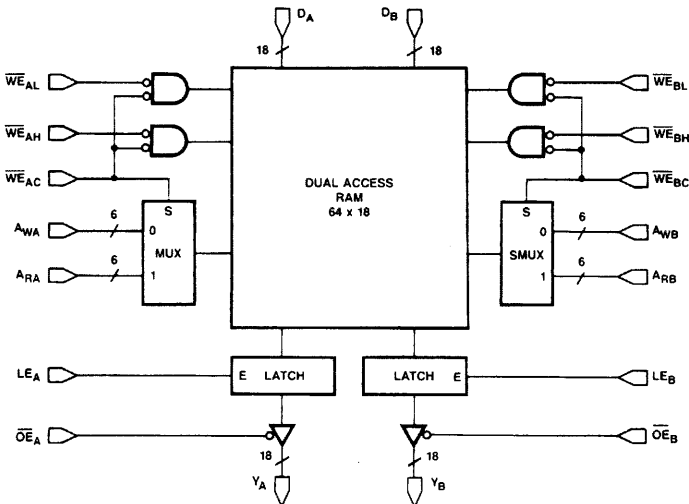
- Fast**
 With an access time of 24 ns, the Am29334 supports 80-90 ns microcycle time when used with the Am29300 Family for 32-bit systems.
- 64 x 18 Bits Wide Register File**
 The Am29334 is a high-performance, high-speed, dual-access RAM with two READ ports and two WRITE ports.
- Cascadable**
 The Am29334 is cascadable to support either wider word widths, deeper register files, or both.
- Simplified Timing Control**
 Control for write enable timing and for on-chip read/write address multiplexer are derived from a single-phase clock input.
- Byte Parity Storage**
 Width of 18 bits facilitates byte parity storage for each port and provides consistency with the Am29332 32-bit ALU.
- Byte Write Capability**
 Individual byte write enables allow byte or full word write.

GENERAL DESCRIPTION

The Am29334 is a 64-word deep and 18-bit wide dual-access register file designed to support other members of the Am29300 Family by providing high-speed storage. It has two write and two read ports for data and four 6-bit address ports. Two address ports are associated with each pair of read and write data ports, one to read data and the other to write. The device is capable of performing two reads and two writes in one cycle. The 18-bit wide register

file allows storage of byte parity to support parity check and generate in the Am29332 32-bit ALU. Independent control for each read and write data port allows the Am29334 to be used as a high-speed shared memory or as a mailbox for a multiprocessor system. The device is designed with an access time of 24 ns. It is housed in a 120-lead pin-grid-array package.

BLOCK DIAGRAM



BD003022

RELATED AMD PRODUCTS

Part No.	Description
Am29325	32-Bit Floating Point Processor
Am29331	16-Bit Microprogram Sequencer
Am29332	32-Bit Extended Function ALU

CONNECTION DIAGRAM

	A	B	C	D	E	F	G	H	J	K	L	M	N
1	AWA2	ARA2	AWA1	DA00	DA02	DA04	DA08	DA09	DA12	DA16	LEA	WEAC	WEAL
2	ARA3	AWA3	ARA1	ARA0	DA03	DA05	DA07	DA10	DA13	DA15	ARA5	AWA5	WEAH
3	AWA4	ARA4	YB00	AWA0	DA01	GND E	DA06	VCCE	DA11	DA14	DA17	ARB4	AWB4
4	YB01	YB02	YB03								YA00	YA01	YA02
5	GNDT	YB04	YB05								YA03	YA04	GNDT
6	YB07	YB06	VCCT								OE \bar{A}	YA06	YA05
7	YB08	YB09	YB10								YA07	YA08	YA09
8	YB12	YB11	OE \bar{B}								VCCT	YA11	YA10
9	GNDT	YB13	YB14								YA12	YA13	GNDT
10	YB15	YB16	YB17								YA14	YA15	YA16
11	WE \bar{B} L	WE \bar{B} H	DB01	DB04	VCCE	DB08	DB09	DB15	GND E	ARB0	YA17	ARB3	AWB3
12	WE \bar{B} C	LEB	DB00	DB03	VCCE	DB05	DB11	DB12	GND E	DB17	AWB0	AWB2	ARB2
13	AWB5	ARB5	DB07	DB02	VCCE	DB06	DB10	DB14	GND E	DB16	DB13	ARB1	AWB1

CD010391

Note: GNDT = TTL GND
 GNDE = ECL GND
 VCCT = TTL VCC
 VCCE = ECL VCC

TABLE OF INTERCONNECTIONS

(Sorted by Pin No.)

PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.
-	-	39	C-5	YB5	115	H-2	DA10	10	M-5	YA4	80
-	-	37	C-6	TTL VCC	113	H-3	ECL VCC	68	M-6	YA6	81
-	-	99	C-7	YB10	52	H-11	DB15	34	M-7	YAB	82
-	-	97	C-8	OE _B	53	H-12	DB12	95	M-8	YA11	25
A-1	AWA2	1	C-9	YB14	109	H-13	DB14	94	M-9	YA13	86
A-2	ARA3	120	C-10	YB17	48	J-1	DA12	11	M-10	YA15	87
A-3	AWA4	59	C-11	DB1	44	J-2	DA13	71	M-11	ARB3	89
A-4	YB1	58	C-12	DB0	104	J-3	DA11	70	M-12	AWB2	30
A-5	TTL GND	56	C-13	DB7	41	J-11	ECL GND	38	M-13	ARB1	91
A-6	YB7	114	D-1	DA0	4	J-12	ECL GND	38	N-1	WE _{AL}	16
A-7	YB8	54	D-2	ARA0	63	J-13	ECL GND	38	N-2	WE _{AH}	76
A-8	YB12	51	D-3	AWA0	3	K-1	DA16	13	N-3	AWB4	17
A-9	TTL GND	50	D-11	DB4	102	K-2	DA15	72	N-4	YA2	19
A-10	YB15	49	D-12	DB3	43	K-3	DA14	12	N-5	TTL GND	20
A-11	WE _{BL}	47	D-13	DB2	103	K-11	ARB0	92	N-6	YA5	21
A-12	WE _{BC}	106	E-1	DA2	5	K-12	DB17	33	N-7	YA9	24
A-13	AWB5	46	E-2	DA3	65	K-13	DB16	93	N-8	YA10	84
B-1	ARA2	61	E-3	DA1	64	L-1	LEA	14	N-9	TTL GND	26
B-2	AWA3	60	E-11	ECL VCC	98	L-2	ARA5	74	N-10	YA16	28
B-3	ARA4	119	E-12	ECL VCC	98	L-3	DA17	73	N-11	AWB3	29
B-4	YB2	117	E-13	ECL VCC	98	L-4	YA0	18	N-12	ARB2	90
B-5	YB4	116	F-1	DA4	6	L-5	YA3	79	N-13	AWB1	31
B-6	YB6	55	F-2	DA5	66	L-6	OE _A	23			
B-7	YB9	112	F-3	ECL GND	8	L-7	YA7	22			
B-8	YB11	111	F-11	DB8	100	L-8	TTL VCC	83			
B-9	YB13	110	F-12	DB5	42	L-9	YA12	85			
B-10	YB16	108	F-13	DB6	101	L-10	YA14	27			
B-11	WE _{BH}	107	G-1	DA8	9	L-11	YA17	88			
B-12	LE _B	45	G-2	DA7	67	L-12	AWB0	32			
B-13	ARB5	105	G-3	DA6	7	L-13	DB13	35			
C-1	AWA1	2	G-11	DB9	40	M-1	WE _{AC}	75			
C-2	ARA1	62	G-12	DB11	36	M-2	AWA5	15			
C-3	YB0	118	G-13	DB10	96	M-3	ARB4	77			
C-4	YB3	57	H-1	DA9	69	M-4	YA1	78			

Notes:

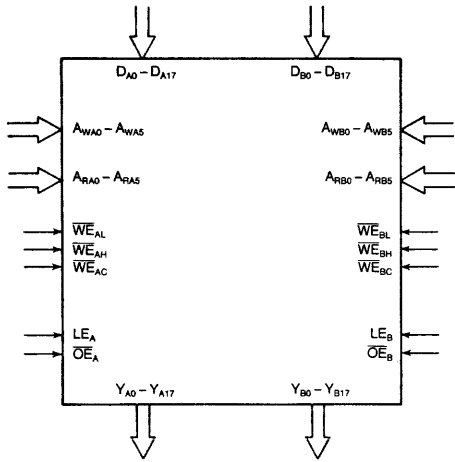
1. Pins E-1, E-12 and E-13 are physically shorted together in the package.
2. Pins J-11, J-12 and J-13 are physically shorted together in the package.

TABLE OF INTERCONNECTIONS (Cont'd.)

(Sorted by Pin Name)

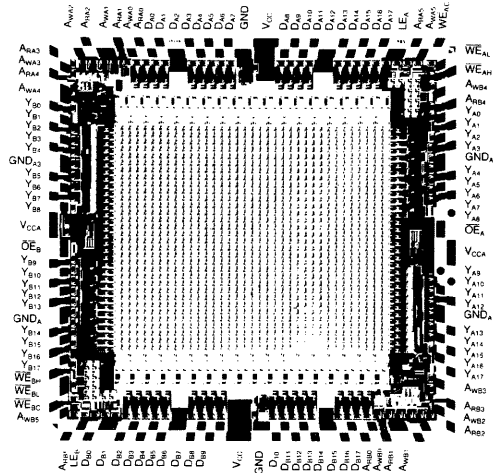
PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.
-	-	97	DA3	E-2	65	DB16	K-13	93	YA4	M-5	80
-	-	99	DA4	F-1	6	DB17	K-12	33	YA5	N-6	21
-	-	39	DA5	F-2	66	ECL GND	J-12	38	YA6	M-6	81
-	-	37	DA6	G-3	7	ECL GND	F-3	8	YA7	L-7	22
ARA0	D-2	63	DA7	G-2	67	ECL GND	J-11	38	YA8	M-7	82
ARA1	C-2	62	DA8	G-1	9	ECL GND	J-13	38	YA9	N-7	24
ARA2	B-1	61	DA9	H-1	69	ECL VCC	H-3	68	YA10	N-8	84
ARA3	A-2	120	DA10	H-2	10	ECL VCC	E-13	98	YA12	L-9	85
ARA4	B-3	119	DA11	J-3	70	ECL VCC	E-11	98	YA13	M-9	86
ARA5	L-2	74	DA12	J-1	11	ECL VCC	E-12	98	YA14	L-10	27
ARB0	K-11	92	DA13	J-2	71	LEA	L-1	14	YA15	M-10	87
ARB1	M-13	91	DA14	K-3	12	LEB	B-12	45	YA16	N-10	28
ARB2	N-12	90	DA15	K-2	72	OEA	L-6	23	YA17	L-11	88
ARB3	M-11	89	DA16	K-1	13	OEB	C-8	53	YB0	C-3	118
ARB4	M-3	77	DA17	L-3	73	TTL GND	A-5	56	YB1	A-4	58
ARB5	B-13	105	DB0	C-12	104	TTL GND	A-9	50	YB2	B-4	117
AWA0	D-3	3	DB1	C-11	44	TTL GND	N-5	20	YB3	C-4	57
AWA1	C-1	2	DB2	D-13	103	TTL GND	N-9	26	YB4	B-5	116
AWA2	A-1	1	DB3	D-12	43	TTL VCC	C-6	113	YB5	C-5	115
AWA3	B-2	60	DB4	D-11	102	TTL VCC	L-8	83	YB6	B-6	55
AWA4	A-3	59	DB5	F-12	42	WEAC	M-1	75	YB7	A-6	114
AWA5	M-2	15	DB6	F-13	101	WEAH	N-2	76	YB8	A-7	54
AWB0	L-12	32	DB7	C-13	41	WEAL	N-1	16	YB9	B-7	112
AWB1	N-13	31	DB8	F-11	100	WEBC	A-12	106	YB10	C-7	52
AWB2	M-12	30	DB9	G-11	40	WEBH	B-11	107	YB11	B-8	111
AWB3	N-11	29	DB10	G-13	96	WEBL	A-11	47	YB12	A-8	51
AWB4	N-3	17	DB11	G-12	36	YA0	L-4	18	YB13	B-9	110
AWB5	A-13	46	DB12	H-12	95	YA1	M-4	78	YB14	C-9	109
DA0	D-1	4	DB13	L-13	35	YA11	M-8	25	YB15	A-10	49
DA1	E-3	64	DB14	H-13	94	YA2	N-4	19	YB16	B-10	108
DA2	E-1	5	DB15	H-11	34	YA3	L-5	79	YB17	C-10	48

LOGIC SYMBOL



LS002220

METALLIZATION AND PAD LAYOUT



Die Size: 258 x 251 mils
Equivalent Gate Count: 3500

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. Device Number
- b. Speed Option (if applicable)
- c. Package Type
- d. Temperature Range
- e. Optional Processing

AM29334

G

C

B

e. OPTIONAL PROCESSING

Blank = Standard processing
B = Burn-in

d. TEMPERATURE RANGE

C = Commercial ($T_C = 0$ to $+85^\circ\text{C}$)

c. PACKAGE TYPE

G = 120-Lead Pin Grid Array with Heatsink (CG 120)

b. SPEED OPTION

Not Applicable

a. DEVICE NUMBER/DESCRIPTION

Am29334
Four-Port Dual-Access Register File

Valid Combinations

Valid Combinations	
AM29334	GC, GCB

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

ARA0 - ARA5 Addresses (Inputs, Active HIGH)

The 6-bit field presented at the AR_A inputs, selects one of 64 memory words for presentation to the Y_A Data Latch.

ARB0 - ARB5 Addresses (Inputs, Active HIGH)

The 6-bit field presented at the AR_B inputs, selects one of 64 memory words for presentation to the Y_B Data Latch.

YA0 - YA17 Data Latch (Outputs, Three-State)

The 18-bit Y_A Data Latch outputs.

YB0 - YB17 Data Latch (Outputs, Three-State)

The 18-bit Y_B Data Latch outputs.

AWA0 - AWA5 Addresses (Inputs, Active HIGH)

The 6-bit field presented at the AW_A inputs, selects one of 64 words for writing new data from the D_A inputs.

AWB0 - AWB5 Addresses (Inputs, Active HIGH)

The 6-bit field presented at the AW_B inputs, selects one of 64 words for writing new data from the D_B inputs.

DA0 - DA17 Data (Inputs, Active HIGH)

New data is written into the word, selected by the AW_A address inputs, through these inputs.

DB0 - DB17 Data (Inputs, Active HIGH)

New data is written into the word, selected by the AW_B address inputs, through these inputs.

LEA YA Data Latch Enable (Input)

The LE_A input controls the latch for the Y_A output port. When LE_A is HIGH, the latch is open (transparent), and data from the RAM, as selected by the AR_A address inputs, is present at the Y_A outputs. When LE_A is LOW, the latch is closed and it retains the last data read from the RAM selected by the AR_A address inputs.

LEB YB Data Latch Enable (Input)

The LE_B input controls the latch for the Y_B output port. When LE_B is HIGH, the latch is open (transparent), and data from the RAM, as selected by the AR_B address inputs, is present at the Y_B outputs. When LE_B is LOW, the latch is closed and it retains the last data read from the RAM selected by the AR_B address inputs.

OE \bar{A} YA Output Enable (Input, Active LOW)

When $OE_{\bar{A}}$ is LOW, data in the Y_A Data Latch is present at the Y_A outputs. If $OE_{\bar{A}}$ is HIGH, Y_A outputs are in the high-impedance (off) state.

OE \bar{B} YB Output Enable (Input, Active LOW)

When $OE_{\bar{B}}$ is LOW, data in the Y_B Data Latch is present at the Y_B outputs. If $OE_{\bar{B}}$ is HIGH, Y_B outputs are in the high-impedance (off) state.

WE $\bar{A}C$ Write Enable (Input, Active LOW)

When $WE_{\bar{A}C}$ is LOW together with $WE_{\bar{A}H}$ and $WE_{\bar{A}L}$, new data is written into the word selected by the AW_A address inputs. When $WE_{\bar{A}C}$ is HIGH, no data is written into the RAM through the A port.

WE $\bar{B}C$ Write Enable (Input, Active LOW)

When $WE_{\bar{B}C}$ is LOW together with $WE_{\bar{B}H}$ and $WE_{\bar{B}L}$, new data is written into the word selected by the AW_B address inputs. When $WE_{\bar{B}C}$ is HIGH, no data is written into the RAM through the B port.

WE $\bar{A}H$ High-Byte Write Enable (Input, Active LOW)

When $WE_{\bar{A}H}$ is LOW together with $WE_{\bar{A}C}$, new data is written into the high byte of the word selected by the AW_A address inputs. When $WE_{\bar{A}H}$ is HIGH, no data is written into the high byte of the word selected by the AW_A address inputs.

WE $\bar{B}H$ High-Byte Write Enable (Input, Active LOW)

When $WE_{\bar{B}H}$ is LOW together with $WE_{\bar{B}C}$, new data is written into the high byte of the word selected by the AW_B address inputs. When $WE_{\bar{B}H}$ is HIGH, no data is written into the high byte of the word selected by the AW_B address inputs.

WE $\bar{A}L$ Low-Byte Write Enable (Input, Active LOW)

When $WE_{\bar{A}L}$ is LOW together with $WE_{\bar{A}C}$, new data is written into the low byte of the word selected by the AW_A address inputs. When $WE_{\bar{A}L}$ is HIGH, no data is written into the low byte of the word selected by the AW_A address inputs.

WE $\bar{B}L$ Low-Byte Write Enable (Input, Active LOW)

When $WE_{\bar{B}L}$ is LOW together with $WE_{\bar{B}C}$, new data is written into the low byte of the word selected by the AW_B address inputs. When $WE_{\bar{B}L}$ is HIGH, no data is written into the low byte of the word selected by the AW_B address inputs.

FUNCTIONAL DESCRIPTION

The part has two read ports ($Y_{A0}-Y_{A17}$, $Y_{B0}-Y_{B17}$), two write ports ($D_{A0}-D_{A17}$, $D_{B0}-D_{B17}$), four addresses ($A_{RA0}-A_{RA5}$, $A_{WA0}-A_{WA5}$, $A_{RB0}-A_{RB5}$, $A_{WB0}-A_{WB5}$), two latch enables (LE_A , LE_B), two output enables (\overline{OE}_A , \overline{OE}_B), and six write enables (\overline{WE}_{AC} , \overline{WE}_{AL} , \overline{WE}_{AH} , \overline{WE}_{BC} , \overline{WE}_{BL} , \overline{WE}_{BH}) that allow writing of data into one or both bytes of a word. The separate read and write addresses facilitate creation of three- and four-address architectures and allow address set-up and RAM access to overlap.

Since the A and B sides are identical, only operation of the A side is described. The address multiplexer provides the RAM with the address A_{RA} when $\overline{WE}_{AC} = \text{HIGH}$ and with the address A_{WA} when $\overline{WE}_{AC} = \text{LOW}$. Internally the part is designed so that there is no race condition between the write address and the write enable. In most cases \overline{WE}_{AC} and LE_A will be connected to the clock as shown in Figure 2 so that reading will take place in the first part of a clock cycle and writing in the last part. The latch at the output of the RAM is transparent when $LE_A = \text{HIGH}$ and retains the data when $LE_A = \text{LOW}$. The latch has a three-state output Y_A controlled by \overline{OE}_A . Each word is split into two bytes of 9 bits that can be individually written. The low byte covers bits 0 through 8 and the high byte covers bits 9 through 17. One or both bytes of the data at D_A are written into the location given by A_{WA} when the common write enable (\overline{WE}_{AC}) and the appropriate byte write enables (\overline{WE}_{AL} and \overline{WE}_{AH}) are active. Two special cases then arise. First, if a location is written into and read at

the same time, the value read is the value being written. Second, if a location is written into from both the A side and the B side, the value written is undefined, but the operation is not harmful.

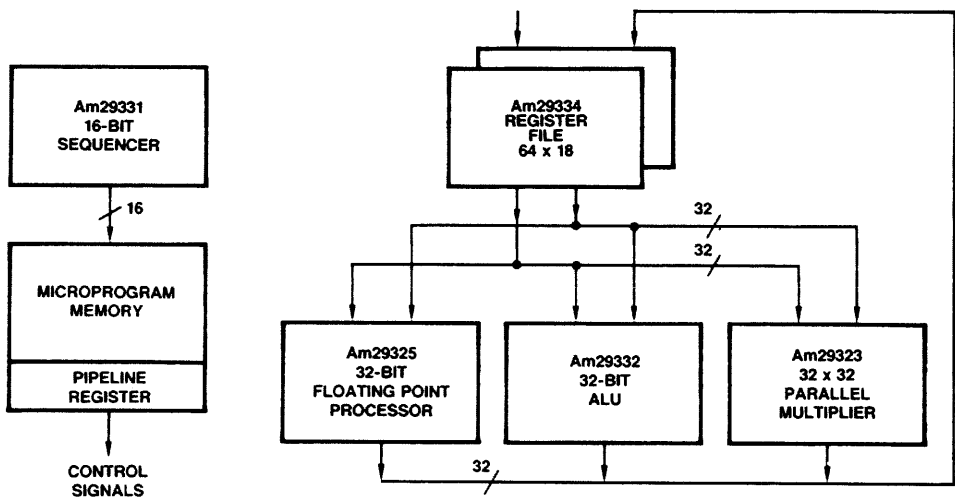
The transparency mode during a write ($\overline{WE}_A = \text{LOW}$) allows the data-in (D_A) to not only be written into memory but also to appear at the output (Y_A) when the output latch (LE_A) is HIGH and the output enable control (\overline{OE}_A) is LOW.

Extension To Four Read Ports and Two Write Ports

A RAM with four read ports and two write ports can be made by using two dual access RAMs and connecting each of the write ports, write addresses, and write enables in parallel for the two devices. As an example, this RAM may provide data storage for a data ALU and an address adder as shown in Figure 3. A location should not be read before it has been written into for the first time as the contents of the two dual access RAMs are likely to be different upon power-up.

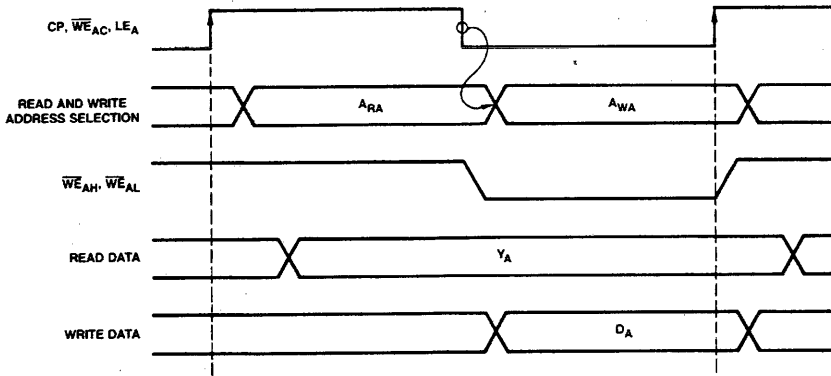
32 Words x 36 Bits Single-Access RAM

It is possible to convert the 64 words x 18 bits dual-access RAM into a 32 word x 36 bit single-access RAM. This is done by storing the upper half of the 36 bits in the upper half of the 64 words and addressing these from the A side. Then store the lower half of the 36 bits in the lower half of the 64 words and address these from the B side. This arrangement, which is shown in Figure 4, does not change the capacity of the RAM, but the dual access is lost.



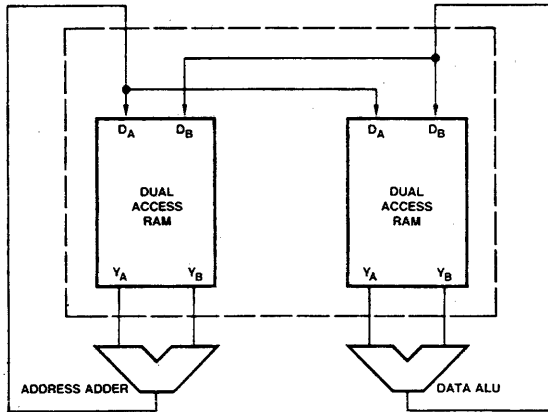
AF003480

Figure 1. Am29300 Family High-Performance System Block Diagram



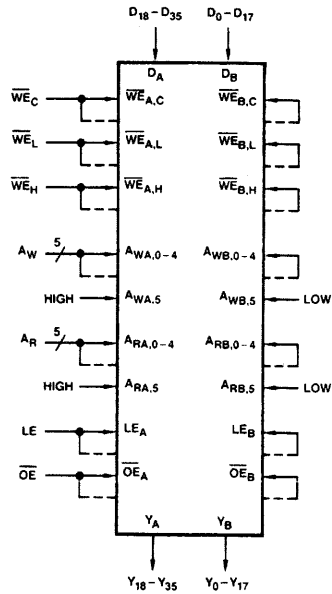
WF009520

Figure 2. Read through Y_A and Write through D_A in a Single Cycle (Two Bytes)



AF003490

Figure 3. RAM with Four Read Ports and Two Write Ports



LS001790

Figure 4. 32 x 36 RAM (Single Access) Using 64 x 18 Dual-Access RAM

APPLICATIONS

Suggestions for Power and Ground Pin Connections

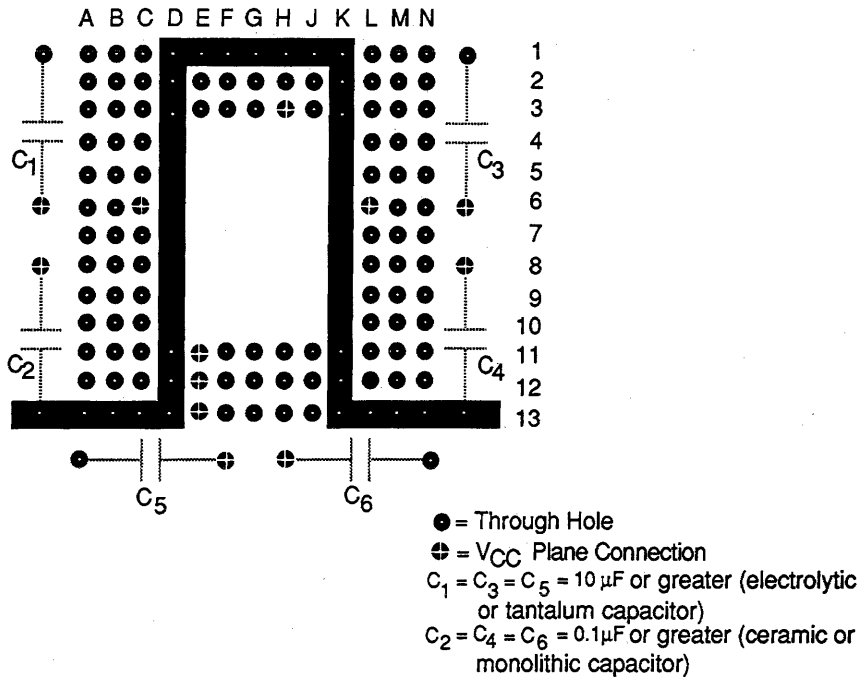
The Am29334 operates in an environment of fast signal rise times and substantial switching currents. Therefore, care must be exercised during circuit board design and layout, as with any high-performance component. The following is a suggested layout, but since systems vary widely in electrical configuration, an empirical evaluation of the intended layout is recommended.

The V_{CCT} and $GNDT$ pins, which carry output driver switching currents, tend to be electrically noisy. The V_{CCE} and $GNDE$ pins, which supply the ECL core of the device, tend to produce less noise, and the circuits they supply may be adversely affected by noise spikes on the V_{CCE} plane. For this reason, it is best to provide isolation between the V_{CCE} and V_{CCT} pins, as well as independent decoupling for each. Isolating the $GNDE$ and $GNDT$ pins is not required.

Printed Circuit Board Layout Suggestions

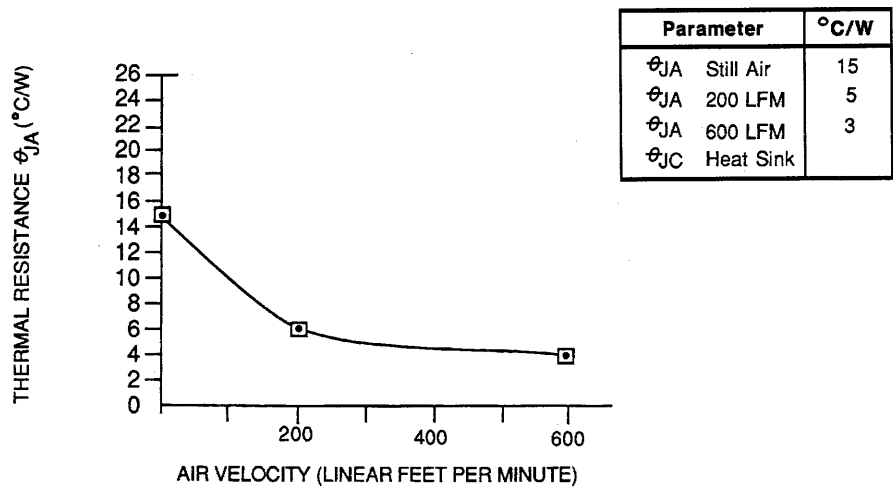
1. Use of a multi-layer PC board with separate power, ground, and signal planes is highly recommended.
2. All V_{CCE} and V_{CCT} pins should be connected to the V_{CC} plane. V_{CCT} pins should be isolated from V_{CCE} pins by means of a slot cut in the V_{CCE} plane; see Figure 5. By physically separating the V_{CCE} and V_{CCT} pins, coupled noise will be reduced.
3. All $GNDE$ and $GNDT$ pins should be connected directly to the ground plane.
4. The V_{CCT} pins should be decoupled to ground with a 0.1- μ F ceramic capacitor and a 10- μ F electrolytic capacitor, placed as closely to the Am29334 as is practical. V_{CCE} pins should be decoupled to ground in a similar manner.

A suggested layout is shown in Figure 5.



CD010900

Figure 5. Suggested Printed Circuit Board Layout



OP002241

Figure 6. Am29334 Thermal Characteristics (Typical)

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Temperature Under Bias - T_C -55 to +125°C
 Supply Voltage to Ground Potential
 Continuous -0.5 to +7.0 V
 DC Voltage Applied to Outputs
 for High State -0.5 V to + V_{CC} Max
 DC Input Voltage -0.5 to +5.5 V

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices
 Temperature (T_C) 0 to +85°C
 Supply Voltage +4.75 to +5.25 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range

Parameter Symbol	Parameter Description	Test Conditions (Note 1)			Min.	Max.	Unit
V_{OH}	Output HIGH Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IL}$ or V_{IH} $I_{OH} = -3 \text{ mA}$			2.4		Volts
V_{OL}	Output LOW Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IL}$ or V_{IH} $I_{OL} = 16 \text{ mA}$				0.5	Volts
V_{IH}	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs			2.0		Volts
V_{IL}	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs				0.8	Volts
V_I	Input Clamp Voltage	$V_{CC} = \text{Min.}$ $I_{IN} = -18 \text{ mA}$				-1.2	Volts
I_{IL}	Input LOW Current	$V_{CC} = \text{Max.}$ $V_{IN} = 0.5 \text{ V}$				-0.5	mA
I_{IH}	Input HIGH Current	$V_{CC} = \text{Max.}$ $V_{IN} = 2.4 \text{ V}$				50	μA
I_I	Input HIGH Current	$V_{CC} = \text{Max.}$ $V_{IN} = 5.5 \text{ V}$				1.0	mA
I_{OZH} I_{OZL}	Off-State (High-Impedance) Output Current	$V_{CC} = \text{Max.}$	$V_O = 2.4 \text{ V}$			50	μA
			$V_O = 0.5 \text{ V}$			-50	
I_{SC}	Output Short-Circuit Current (Note 2)	$V_{CC} = \text{Max. to } +0.5 \text{ V}$ $V_O = 0.5 \text{ V}$			-15	-50	mA
I_{CC}	Power Supply Current (Note 3)	$V_{CC} = \text{Max}$	COM'L Only	$T_C = 0 \text{ to } +85^\circ\text{C}$		950	mA
				$T_C = +85^\circ\text{C}$		820	
			MIL Only	$T_C = -55 \text{ to } +125^\circ\text{C}$			
				$T_C = +125^\circ\text{C}$			

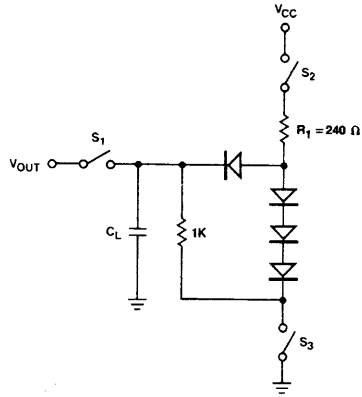
- Notes: 1. For conditions shown as Min. or Max., use the appropriate value specified under Operating Ranges for the applicable device type.
 2. Not more than one output should be shorted at a time. Duration of the short-circuit test should not exceed one second.
 3. Measured with all inputs HIGH.
 4. Recommended air velocity is 200 linear feet per minute.

SWITCHING CHARACTERISTICS over operating range (Note 1)

No.	Parameter	Description	Test Conditions	Max. Delay	Unit
1	Access Time	A_{RA} or A_{RB} to Y_A or Y_B	LE_A or $LE_B = H$	24	ns
2	Turn-On Time	\overline{OE}_A or $\overline{OE}_B \downarrow$ to Y_A or Y_B Active		20	ns
3	Turn-Off Time (Note 2)	\overline{OE}_A or $\overline{OE}_B \uparrow$ to Y_A or $Y_B =$ High Impedance	$C_L = 5$ pF load	16	ns
4	Enable Time	LE_A or $LE_B \uparrow$ to Y_A or Y_B		16	ns
5	Transparency	\overline{WE}_A or $\overline{WE}_B \downarrow$ to Y_A or Y_B	LE_A or $LE_B = H$	32	ns
6	Transparency	D_A or D_B to Y_A or Y_B	LE_A or $LE_B = H$, \overline{WE}_A or $\overline{WE}_B = L$	33	ns
7	Data Setup Time	D_A or D_B to \overline{WE}_A or $\overline{WE}_B \uparrow$		9	ns
8	Data Hold Time	D_A or D_B to \overline{WE}_A or $\overline{WE}_B \uparrow$		2	ns
9	Address Setup Time	A_{WA} or A_{WB} to \overline{WE}_A or $\overline{WE}_B \downarrow$		0	ns
10	Address Hold Time	A_{WA} or A_{WB} to \overline{WE}_A or $\overline{WE}_B \uparrow$		3	ns
11	Address Setup Time	A_{RA} or A_{RB} to LE_A or $LE_B \downarrow$		7	ns
12	Address Hold Time	A_{RA} or A_{RB} to LE_A or $LE_B \downarrow$		4	ns
13	Latch Close Before Write	LE_A or $LE_B \downarrow$ to \overline{WE}_A or $\overline{WE}_B \downarrow$		0	ns
14	Write Pulse Width	\overline{WE}_A or \overline{WE}_B (LOW)		18	ns
15	Latch Data Capture Pulse Width	LE_A or LE_B (HIGH)		10	ns

- Notes: 1. $\overline{WE}_A = \overline{WE}_{AC} + \overline{WE}_{AL}/H$
 $\overline{WE}_B = \overline{WE}_{BC} + \overline{WE}_{BL}/H$
 2. Y_A and Y_B are tested independently.

SWITCHING TEST CIRCUIT

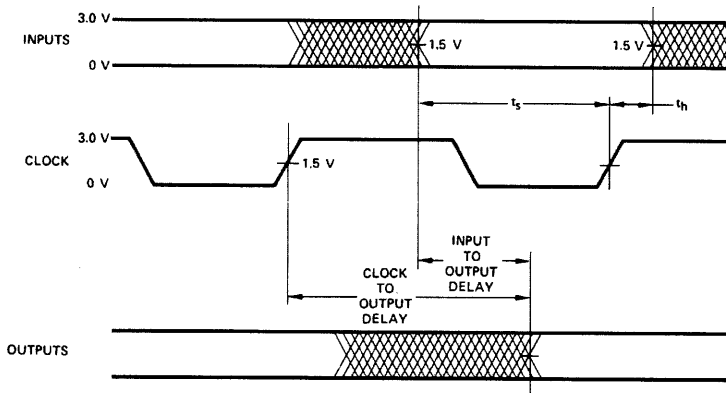


TC003420

Three-State Outputs

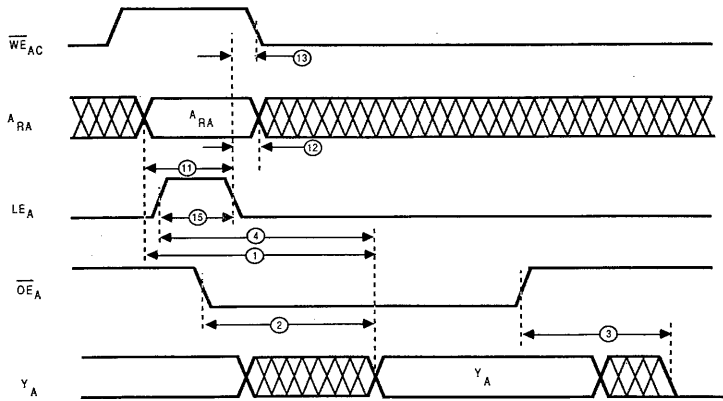
- Notes:
1. $C_L = 50$ pF includes scope probe, wiring and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during functions tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for tp_{ZH} test.
 S_1 and S_2 are closed while S_3 is open for tp_{ZL} test.
 4. $C_L = 5.0$ pF for output disable tests.

SWITCHING WAVEFORMS



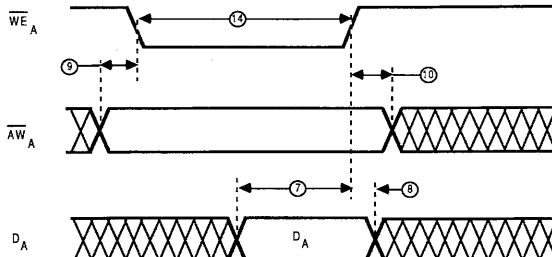
WFR02990

SWITCHING WAVEFORMS (Cont'd.)



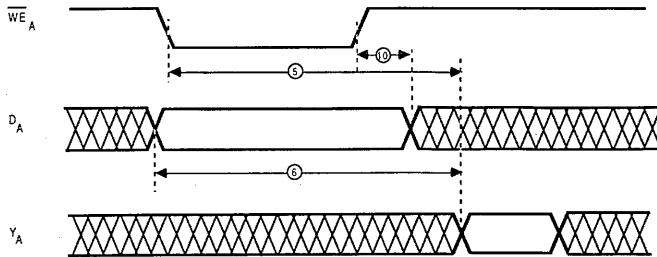
WF023530

Read Function (same for B Port)



WF023520

Write Function (same for B Port)

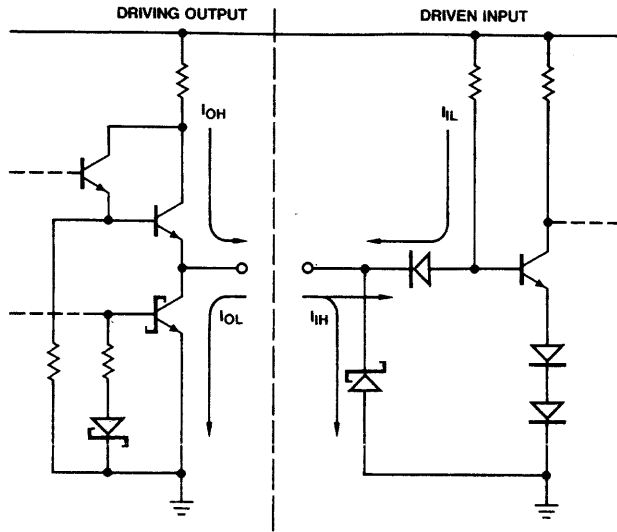


WF023510

Note: $LE_A = \text{HIGH}$
 $\overline{OE}_A = \text{LOW}$

Transparency Function (same for B Port)

INPUT/OUTPUT CIRCUIT DIAGRAM



ICR00480

Am29434

ECL Four-Port, Dual-Access Register File



Am29434

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

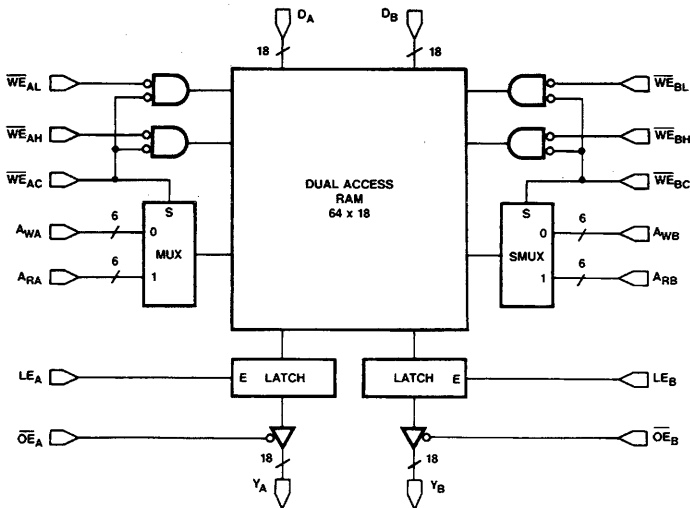
- **Fast**
With an access time of 20 ns, the Am29434 supports 50-60 ns microcycle time when used with the Am29400 Family for 32-bit systems.
- **64 x 18 Bits Wide Register File**
The Am29434 is a high-performance, high-speed, dual-access RAM with two READ ports and two WRITE ports.
- **Cascadable**
The Am29434 is cascadable to support either wider word widths, deeper register files, or both.
- **Simplified Timing Control**
Control for write enable timing and for on-chip read/write address multiplexer are derived from a single-phase clock input.
- **Byte Parity Storage**
Width of 18 bits facilitates byte parity storage for each port and provides consistency with the Am29432 32-bit ALU.
- **Byte Write Capability**
Individual byte-write enables allows byte or full word write.

GENERAL DESCRIPTION

The Am29434 is a 64-word deep and 18-bit wide dual-access register file designed to support other members of the Am29400 Family by providing high-speed storage. It has two write and two read ports for data and four 6-bit address ports. Two address ports are associated with each pair of read and write data ports, one to read data and the other to write. The device is capable of performing two reads and two writes in one cycle. The 18-bit wide register

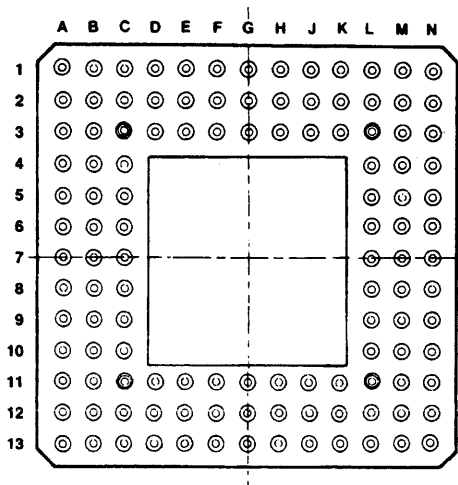
file allows storage of byte parity to support parity check and generate in the Am29432 32-bit ALU. Independent control for each read and write data port allows the Am29434 to be used as a high-speed shared memory or as a mailbox for a multiprocessor system. The device is designed with an access time of 20 ns. It is housed in a 120-lead pin grid array package.

BLOCK DIAGRAM



BD003022

CONNECTION DIAGRAM 120-Lead PGA*



CD009170

*Pinout observed from pin side of package.

TABLE OF INTERCONNECTIONS (Sorted by Pin No.)

PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.
-	-	99	C-5	YB5	115	H-2	DA10	10	M-5	YA4	80
-	-	97	C-6	VCCO	113	H-3	VCC	68	M-6	YA6	81
-	-	39	C-7	YB10	52	H-11	DB15	34	M-7	YA8	82
-	-	37	C-8	OE _B	53	H-12	DB12	95	M-8	YA11	25
A-1	AWA2	1	C-9	YB14	109	H-13	DB14	94	M-9	YA13	86
A-2	ARA3	120	C-10	YB17	48	J-1	DA12	11	M-10	YA15	87
A-3	AWA4	59	C-11	DB1	44	J-2	DA13	71	M-11	ARB3	89
A-4	YB1	58	C-12	DB0	104	J-3	DA11	70	M-12	AWB2	30
A-5	VCCO	56	C-13	DB7	41	J-11	VEE	38	M-13	ARB1	91
A-6	YB7	114	D-1	DA0	4	J-12	VEE	38	N-1	WEAL	16
A-7	YB8	54	D-2	ARA0	63	J-13	VEE	38	N-2	WEAH	76
A-8	YB12	51	D-3	AWA0	3	K-1	DA16	13	N-3	AWB4	17
A-9	VCCO	50	D-11	DB4	102	K-2	DA15	72	N-4	YA2	19
A-10	YB15	49	D-12	DB3	43	K-3	DA14	12	N-5	VCCO	20
A-11	WEBL	47	D-13	DB2	103	K-11	ARB0	92	N-6	YA5	21
A-12	WECB	106	E-1	DA2	5	K-12	DB17	33	N-7	YA9	24
A-13	AWB5	46	E-2	DA3	65	K-13	DB16	93	N-8	YA10	84
B-1	ARA2	61	E-3	DA1	64	L-1	LEA	14	N-9	VCCO	26
B-2	AWA3	60	E-11	VCC	98	L-2	ARA5	74	N-10	YA16	28
B-3	ARA4	119	E-12	VCC	98	L-3	DA17	73	N-11	AWB3	29
B-4	YB2	117	E-13	VCC	98	L-4	YA0	18	N-12	ARB2	90
B-5	YB4	116	F-1	DA4	6	L-5	YA3	79	N-13	AWB1	31
B-6	YB6	55	F-2	DA5	66	L-6	OEA	23			
B-7	YB9	112	F-3	VEE	8	L-7	YA7	22			
B-8	YB11	111	F-11	DB8	100	L-8	VCCO	83			
B-9	YB13	110	F-12	DB5	42	L-9	YA12	85			
B-10	YB16	108	F-13	DB6	101	L-10	YA14	27			
B-11	WEBH	107	G-1	DA8	9	L-11	YA17	88			
B-12	LEB	45	G-2	DA7	67	L-12	AWB0	32			
B-13	ARB5	105	G-3	DA6	7	L-13	DB13	35			
C-1	AWA1	2	G-11	DB9	40	M-1	WEAC	75			
C-2	ARA1	62	G-12	DB11	36	M-2	AWA5	15			
C-3	YB0	118	G-13	DB10	96	M-3	ARB4	77			
C-4	YB3	57	H-1	DA9	69	M-4	YA1	78			

TABLE OF INTERCONNECTIONS

(Sorted By Pin Name)

PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.	PIN NAME	PIN NO.	PAD NO.
-	-	99	DA4	F-1	6	LEA	L-1	14	YA8	M-7	82
-	-	97	DA5	F-2	66	LEB	B-12	45	YA9	N-7	24
-	-	39	DA6	G-3	7	OEa	L-6	23	YA10	N-8	84
-	-	37	DA7	G-2	67	OEb	C-8	53	YA11	M-8	25
ARA0	D-2	63	DA8	G-1	9	VCC	H-3	68	YA12	L-9	85
ARA1	C-2	62	DA9	H-1	69	VCC	E-11,	98	YA13	M-9	86
ARA2	N-13	61	DA10	H-2	10		E-12,		YA14	L-10	27
ARA3	A-2	120	DA11	J-3	70		E-13		YA15	M-10	87
ARA4	B-3	119	DA12	J-1	11	VCCO	N-5	20	YA16	N-10	28
ARA5	L-2	74	DA13	J-2	71	VCCO	N-9	26	YA17	L-11	88
ARB0	K-11	92	DA14	K-3	12	VCCO	A-9	50	YB0	C-3	118
ARB1	M-13	91	DA15	K-2	72	VCCO	A-5	56	YB1	A-4	58
ARB2	N-12	90	DA16	K-1	13	VCCO	L-8	83	YB2	B-4	117
ARB3	M-11	89	DA17	L-2	73	VCCO	C-6	113	YB3	C-4	57
ARB4	M-3	77	DB0	C-12	104	VEE	F-3	8	YB4	B-5	116
ARB5	B-13	105	DB1	C-11	44	VEE	J-11,	38	YB5	C-5	115
AWA0	D-3	3	DB2	D-13	103		J-12,		YB6	B-6	55
AWA1	C-1	2	DB3	D-12	43		J-13		YB7	A-6	114
AWA2	A-1	1	DB4	D-11	102	WEAC	M-1	75	YB8	A-7	54
AWA3	B-2	60	DB5	F-12	42	WEAH	N-2	76	YB9	B-7	112
AWA4	A-3	59	DB6	F-13	101	WEAL	N-1	16	YB10	C-7	52
AWA5	M-2	15	DB7	C-13	41	WEBC	A-12	106	YB11	B-8	111
AWB0	L-12	32	DB8	F-11	100	WEBH	B-11	107	YB12	A-8	51
AWB1	N-13	31	DB9	G-11	40	WEBL	A-11	47	YB13	B-9	110
AWB2	M-12	30	DB10	G-13	96	YA0	L-4	18	YB14	C-9	109
AWB3	N-11	29	DB11	G-12	36	YA1	M-4	78	YB15	A-10	49
AWB4	N-3	17	DB12	H-12	95	YA2	N-4	19	YB16	B-10	108
AWB5	A-13	46	DB13	L-13	35	YA3	L-5	79	YB17	C-10	48
DA0	D-1	4	DB14	H-13	94	YA4	M-5	80			
DA1	E-3	64	DB15	H-11	34	YA5	N-6	21			
DA2	E-1	5	DB16	K-13	93	YA6	M-6	81			
DA3	E-2	65	DB17	K-12	33	YA7	L-7	22			

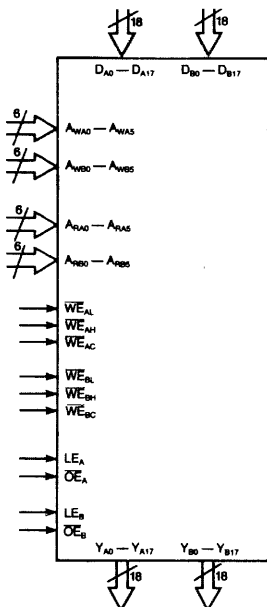
TABLE OF INTERCONNECTIONS

(Sorted by Pad No.)

PIN NAME	PAD NUMBER	PIN NUMBER	PIN NAME	PAD NUMBER	PIN NUMBER	PIN NAME	PAD NUMBER	PIN NUMBER	PIN NAME	PAD NUMBER	PIN NUMBER
AWA2	1	A-1	AWB1	31	N-13	ARA2	61	B-1	ARB1	91	M-13
AWA1	2	C-1	AWB0	32	L-12	ARA1	62	C-2	ARB0	92	K-11
AWA0	3	D-3	DB17	33	K-12	ARA0	63	D-2	DB16	93	K-13
DA0	4	D-1	DB15	34	H-11	DA1	64	E-3	DB14	94	H-13
DA2	5	E-1	DB13	35	L-13	DA3	65	E-2	DB12	95	H-12
DA4	6	F-1	DB11	36	G-12	DA5	66	F-2	DB10	96	G-13
DA6	7	G-3	-	37	-	DA7	67	G-2	-	97	-
VEE	8	F-3	VEE	38	J-11, J-12, J-13	VCC	68	H-3	VCC	98	E-11, E-12, E-13
DA8	9	G-1	-	39	-	DA9	69	H-1	-	99	-
DA10	10	H-2	DB9	40	G-11	DA11	70	J-3	DB8	100	F-11
DA12	11	J-1	DB7	41	C-13	DA13	71	J-2	DB6	101	F-13
DA14	12	K-3	DB5	42	F-12	DA15	72	K-2	DB4	102	D-11
DA16	13	K-1	DB3	43	D-12	DA17	73	L-3	DB2	103	D-13
LEA	14	L-1	DB1	44	C-11	ARA5	74	L-2	DB0	104	C-12
AWA5	15	M-2	LEB	45	B-12	WEAC	75	M-1	ARB5	105	B-13
WEAL	16	N-1	AWB5	46	A-13	WEAH	76	N-2	WEBC	106	A-12
AWB4	17	N-3	WEBL	47	A-11	ARB4	77	M-3	WEBH	107	B-11
YA0	18	L-4	YB17	48	C-10	YA1	78	M-4	YB16	108	B-10
YA2	19	N-4	YB15	49	A-10	YA3	79	L-5	YB14	109	C-9
VCC0	20	N-5	VCC0	50	A-9	YA4	80	M-5	YB13	110	B-9
YA5	21	N-6	YB12	51	A-8	YA6	81	M-6	YB11	111	B-8
YA7	22	L-7	YB10	52	C-7	YA8	82	M-7	YB9	112	B-7
OEa	23	L-6	OEb	53	C-8	VCC0	83	L-8	VCC0	113	C-6
YA9	24	N-7	YB8	54	A-7	YA10	84	N-8	YB7	114	A-6
YA11	25	M-8	YB6	55	B-6	YA12	85	L-9	YB5	115	C-5
VCC0	26	N-9	VCC0	56	A-5	YA13	86	M-9	YB4	116	B-5
YA14	27	L-10	YB3	57	C-4	YA15	87	M-10	YB2	117	B-4
YA16	28	N-10	YB1	58	A-4	YA17	88	L-11	YB0	118	C-3
AWB3	29	N-11	AWA4	59	A-3	ARB3	89	M-11	ARA4	119	B-3
AWB2	30	M-12	AWA3	60	B-2	ARB2	90	N-12	ARA3	120	A-2

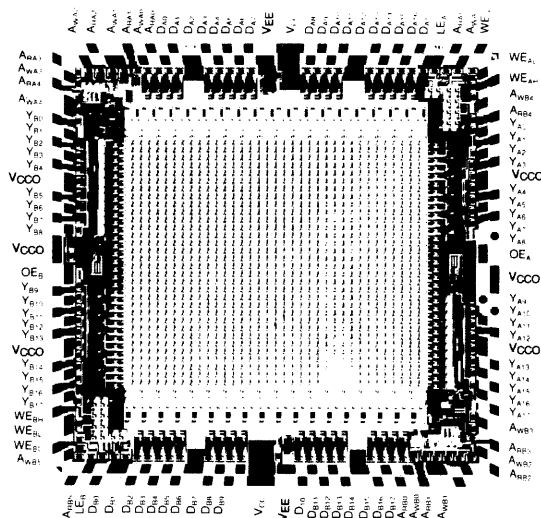
- Notes:
1. VCC is the most positive power supply voltage for internal chip logic.
 2. VCC0 is the most positive power supply for output buffers.
 3. VEE is the most negative power supply for all logic.
 4. Pins E-11, E-12, and E-13 are physically shorted together in the package.
 5. Pins J-11, J-12, and J-13 are physically shorted together in the package.

LOGIC SYMBOL



LS002821

METALLIZATION AND PAD LAYOUT



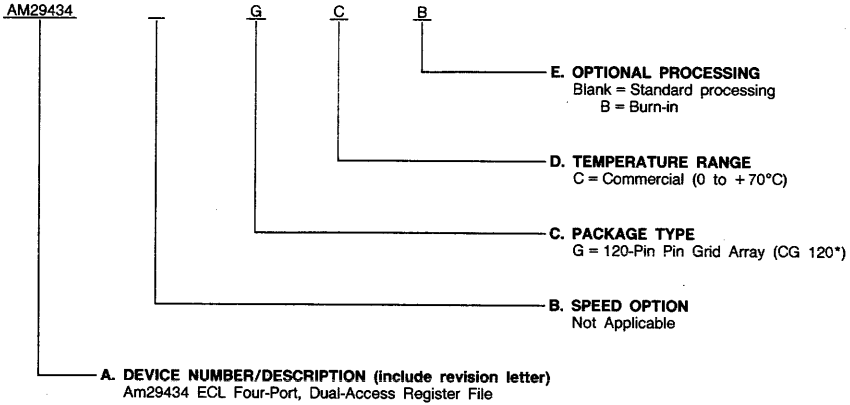
Die Size: 251 x 258 mils
Equivalent gate count - 2700 gates

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- A. Device Number**
- B. Speed Option** (if applicable)
- C. Package Type**
- D. Temperature Range**
- E. Optional Processing**



* Preliminary. Subject to Change.

Valid Combinations	
AM29434	GC, GCB

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released valid combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

AR_{A0} - AR_{A5} Addresses (Inputs, Active HIGH)

The 6-bit field presented at the AR_A inputs selects one of 64 memory words for presentation to the Y_A Data Latch.

AR_{B0} - AR_{B5} Addresses (Inputs, Active HIGH)

The six-bit field presented at the AR_B inputs selects one of 64 memory words for presentation to the Y_B Data Latch.

Y_{A0} - Y_{A17} Data Latch (Outputs)

The 18-bit Y_A Data Latch Outputs.

Y_{B0} - Y_{B17} Data Latch (Outputs)

The 18-bit Y_B Data Latch Outputs.

AW_{A0} - AW_{A5} Addresses (Inputs, Active HIGH)

The six-bit field presented at the AW_A inputs selects one of 64 words for writing new data from the D_A inputs.

AW_{B0} - AW_{B5} Addresses (Inputs, Active HIGH)

The six-bit field presented at the AW_B inputs selects one of 64 words for writing new data from the D_B inputs.

DA₀ - DA₁₇ Data (Inputs, Active HIGH)

New data is written into the word, selected by the AW_A address inputs, through these inputs.

DB₀ - DB₁₇ Data (Inputs, Active HIGH)

New data is written into the word, selected by the AW_B address inputs, through these inputs.

LE_A Y_A Data Latch Enable (Input)

The LE_A input controls the Latch for the Y_A output port. When LE_A is HIGH, the latch is open (transparent) and data from the RAM, as selected by the AR_A address inputs, is present at the Y_A outputs. When LE_A is LOW, the Latch is closed and it retains the last data read from the RAM selected by the AR_A address inputs.

LE_B Y_B Data Latch Enable (Input)

The LE_B input controls the Latch for the Y_B output port. When LE_B is HIGH, the Latch is open (transparent) and data from the RAM, as selected by the AR_B address inputs, is present at the Y_B outputs. When LE_B is LOW, the Latch is closed and it retains the last data read from the RAM selected by the AR_B address inputs.

OE_A Y_A Output Enable (Input, Active LOW)

When OE_A is LOW, data in the Y_A Data Latch is present at the Y_A outputs. If OE_A is HIGH, Y_A outputs are in the LOW logic (off) state.

OE_B Y_B Output Enable (Input, Active LOW)

When OE_B is LOW, data in the Y_B Data Latch is present at the Y_B outputs. If OE_B is HIGH, Y_B outputs are in the LOW logic (off) state.

WE_{AC} Write Enable (Input, Active LOW)

When WE_{AC} is LOW together with WE_{AH} and WE_{AL}, new data is written into the word selected by the AW_A address inputs. When WE_{AC} is HIGH, no data is written into the RAM through the A port.

WE_{BC} Write Enable (Input, Active LOW)

When WE_{BC} is LOW together with WE_{BH} and WE_{BL}, new data is written into the word selected by the AW_B address inputs. When WE_{BC} is HIGH, no data is written into the RAM through the B port.

WE_{AH} High-Byte Write Enable (Input, Active LOW)

When WE_{AH} is LOW together with WE_{AC}, new data is written into the high byte of the word selected by the AW_A address inputs. When WE_{AH} is HIGH, no data is written into the high byte of the word selected by the AW_A address inputs.

WE_{BH} High-Byte Write Enable (Input, Active LOW)

When WE_{BH} is LOW together with WE_{BC}, new data is written into the high byte of the word selected by the AW_B address inputs. When WE_{BH} is HIGH, no data is written into the high byte of the word selected by the AW_B address inputs.

WE_{AL} Low-Byte Write Enable (Input, Active LOW)

When WE_{AL} is LOW together with WE_{AC}, new data is written into the low byte of the word selected by the AW_A address inputs. When WE_{AL} is HIGH, no data is written into the low byte of the word selected by the AW_A address inputs.

WE_{BL} Low-Byte Write Enable (Input, Active LOW)

When WE_{BL} is LOW together with WE_{BC}, new data is written into the low byte of the word selected by the AW_B address inputs. When WE_{BL} is HIGH, no data is written into the low byte of the word selected by the AW_B address inputs.

VCC Internal Logic Ground

This is the most positive voltage in the internal logic. It is used as the reference level for internal logic.

VCCO Out Drive Ground

This is the most positive voltage in the output buffer logic. It is used as the reference level for the buffer logic.

VEE Power Supply Volatge

This is the most negative voltage. It provides power for internal and buffer logic.

FUNCTIONAL DESCRIPTION

The part has two read ports ($Y_{A0}-Y_{A17}$, $Y_{B0}-Y_{B17}$), two write ports (DA_0-DA_{17} , DB_0-DB_{17}), four addresses (ARA_0-ARA_5 , AWA_0-AWA_5 , ARB_0-ARB_5 , AWB_0-AWB_5), two latch enables (LE_A , LE_B), two output enables (OE_A , OE_B), and six write enables (WE_{AC} , WE_{AL} , WE_{AH} , WE_{BC} , WE_{BL} , WE_{BH}) that allow writing of data into one or both bytes of a word. The separate read and write addresses facilitate creation of three- and four-address architectures and allow address set-up and RAM access to overlap.

Since the A and B sides are identical, only operation of the A side is described. The address multiplexer provides the RAM with the address A_{RA} when $WE_{AC} = HIGH$ and with the address A_{WA} when $WE_{AC} = LOW$. Internally the part is designed so that there is no race condition between the write address and the write enable. In most cases WE_{AC} and LE_A will be connected to the clock as shown in Figure 2 so that reading will take place in the first part of a clock cycle and writing in the last part. The latch at the output of the RAM is transparent when $LE_A = HIGH$ and retains the data when $LE_A = LOW$. The latch has an output Y_A controlled by OE_A . Each word is split into two bytes of nine bits that can be individually written. The low byte covers bits 0 through 8 and the high byte covers bits 9 through 17. One or both bytes of the data at D_A are written into the location given by A_{WA} when the common write enable (WE_{AC}) and the appropriate byte write enables (WE_{AL} and WE_{AH}) are active. Two special cases arise. First, if a location is written into and read at the

same time, the value read is the value being written. Second, if a location is written into from both the A side and the B side, the value written is undefined, but the operation is not harmful.

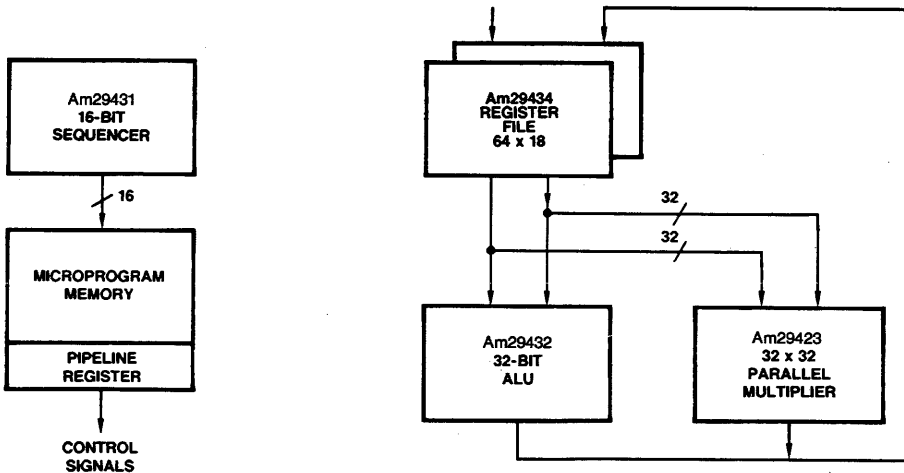
The transparency mode during a write ($WE_A = LOW$) allows the data-in (D_A) to not only be written into memory but also to appear at the output (Y_A) when the output latch (LE_A) is HIGH and the output enable control (OE_A) is LOW.

Extension To Four Read Ports and Two Write Ports

A RAM with four read ports and two write ports can be made by using two dual access RAMs and connecting each of the write ports, write addresses, and write enables in parallel for the two devices. As an example, this RAM may provide data storage for a data ALU and an address adder as shown in Figure 3. A location should not be read before it has been written into for the first time as the contents of the two dual access RAMs are likely to be different upon power-up.

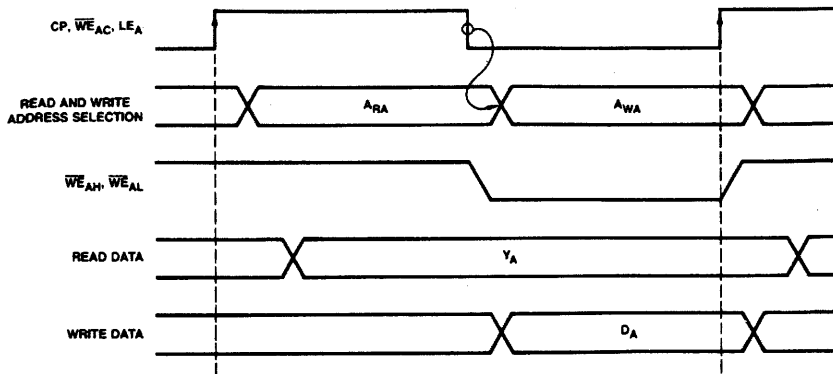
32 Words x 36 Bits Single Access Ram

It is possible to convert the 64 word x 18-bit dual-access RAM into a 32 word x 36-bit single-access RAM. This is done by storing the upper half of the 36 bits in the upper half of the 64 words and addressing them from the A side. The lower half of the 36 bits should then be stored in the lower half of the 64 words and addressed from the B side. This arrangement, which is shown in Figure 4, does not change the capacity of the RAM, but the dual access is lost.



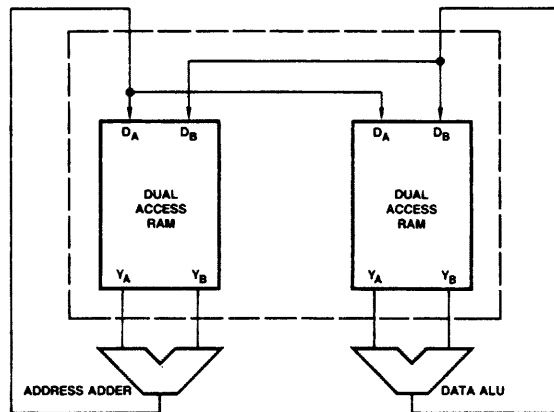
AF003483

Figure 1. Am29400 Family High-Performance System Block Diagram



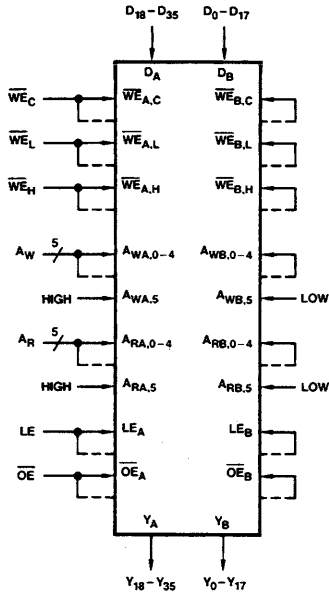
WF009520

Figure 2. Read through Y_A and Write through D_A in a Single Cycle (Two Bytes)



AF003490

Figure 3. RAM with 4 Read Ports and 2 Write Ports



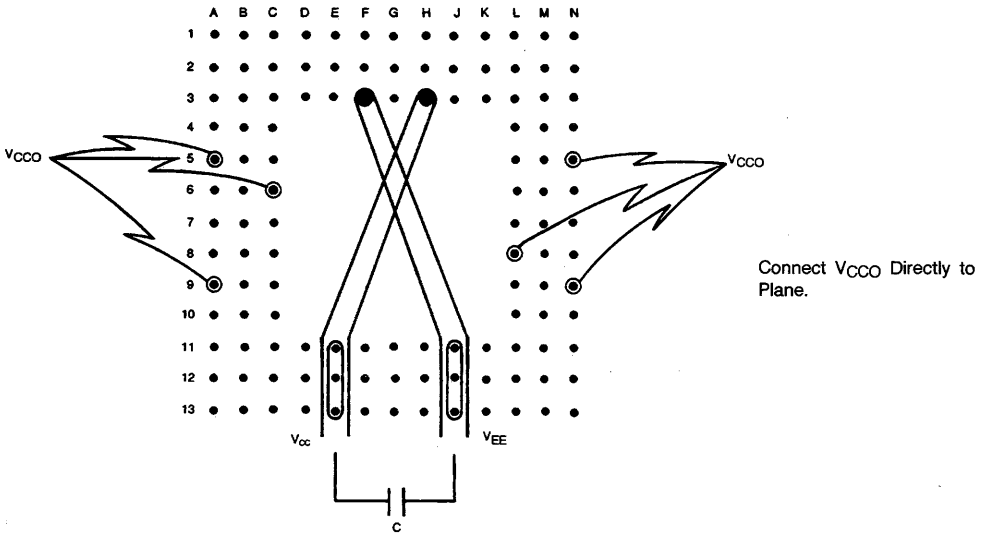
LS001790

Figure 4. 32 x 36 RAM (Single Access) Using 64 x 18 Dual Access RAM

APPLICATIONS

Suggested Printed Circuit Board Layout

Bottom View



AF004151

Connect VCC & VEE Directly to Plane from E-13 and J-13.

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Ambient Temperature with
 Power Applied -55 to +125°C
 V_{EE} Pin Potential to GND Pin -7.0 V to +0.5 V
 Input Voltage (DC) V_{EE} to +0.5 V
 Output Current (DC Output HIGH) -30 mA to +0.1 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices
 Temperature 0 to +75°C
 Supply Voltage -5.46 V to -4.94 V
 Air Velocity 200 linear feet per minute

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS (Commercial) (Notes 1 and 2)

Parameter Symbol	Parameter Description	Test Conditions (Note 5)	Operating Ranges			Units	
			T_A	Min. (Note 3)	Typ. (Note 1)		Max. (Note 3)
V_{OH}	Output Voltage HIGH	$V_{IN} = V_{IH}$ Max. or V_{IL} Min.	0°C	-1000		-840	mV
			+25°C	-960		-810	
			+75°C	-900		-720	
V_{OL}	Output Voltage LOW		0°C	-1870		-1665	mV
			+25°C	-1850		-1650	
			+75°C	-1830		-1625	
V_{OHC}	Output Voltage HIGH	$V_{IN} = V_{IH}$ Min. or V_{IL} Max.	0°C	-1020			mV
			+25°C	-980			
			+75°C	-920			
V_{OLC}	Output Voltage LOW		0°C			-1645	mV
			+25°C			-1630	
			+75°C			-1605	
V_{IH}	Input Voltage HIGH	Guaranteed Input Voltage HIGH for All Inputs	0°C	-1145		-840	mV
			+25°C	-1105		-810	
			+75°C	-1045		-720	
V_{IL}	Input Voltage LOW	Guaranteed Input Voltage LOW for All Inputs	0°C	-1870		-1490	mV
			+25°C	-1850		-1475	
			+75°C	-1830		-1450	
I_{IH}	Input Current HIGH	$V_{IN} = V_{IH}$ Max.	0 to +75°C			220	μA
			+25°C			140	
I_{IL}	Input Current LOW	$V_{IN} = V_{IL}$ Min.	0 to +75°C			950	mA
			+75°C			850	

- Notes: 1. Typical values are:
 $V_{EE} = -5.2$ V, $V_{CC} = \text{GND}$, $V_{CCO} = \text{GND}$
 Output Load = 50 Ω and 30 pF to -2.0 V.
2. Guaranteed with transverse air flow exceeding 200 linear F.P.M. and 2-minute warm-up period. Typical thermal resistance values of the package are:
 θ_{JA} (Junction-to-Ambient) = 22°C/Watt (still air)
 θ_{JA} (Junction-to-Ambient) = 7.5°C/Watt (at 200 F.P.M. air flow)
 θ_{JC} (Junction-to-Case) = 5°C/Watt
3. These are absolute voltages with respect to device ground pin and include all overshoots due to system and/or tester noise. Do not attempt to test these values without suitable equipment.

SWITCHING CHARACTERISTICS (Commercial Only)

No.	Parameters	From	To	Test Conditions	Time (ns)
1	Access Time	A_{RA} or A_{RB}	Y_A or Y_B	LE_A or $LE_B = H$	20
2	Turn-On Time	\overline{OE}_A or $\overline{OE}_B = L$	Y_A or Y_B		10
3	Turn-Off Time	\overline{OE}_A or $\overline{OE}_B = H$	Y_A or $Y_B = L$		10
4	Enable Time	LE_A or $LE_B = H$	Y_A or Y_B		13
5	Transparency	\overline{WE}_A or $\overline{WE}_B = L$	Y_A or Y_B	LE_A or $LE_B = H$	28
6	Transparency	D_A or D_B	Y_A or Y_B	LE_A or $LE_B = H$ \overline{WE}_A or $\overline{WE}_B = L$	29

Minimum Setup and Hold Time

No.	Parameters	For	WRT	Time (ns)
7	Data Setup	D_A or D_B	\overline{WE}_A or \overline{WE}_B (L TO H)	9
8	Data Hold	D_A OR D_B	\overline{WE}_A or \overline{WE}_B (L TO H)	2
9	Address Setup	A_{WA} or A_{WB}	\overline{WE}_A or \overline{WE}_B (H TO L)	0
10	Address Hold	A_{WA} or A_{WB}	\overline{WE}_A or \overline{WE}_B (L TO H)	3
11	Address Setup	A_{RA} or A_{RB}	LE_A or LE_B (H TO L)	7
12	Address Hold	A_{RA} or A_{RB}	LE_A or LE_B (H TO L)	4
13	Latch close before Write	LE_A or LE_B (H TO L)	\overline{WE}_A or \overline{WE}_B (H TO L)	0

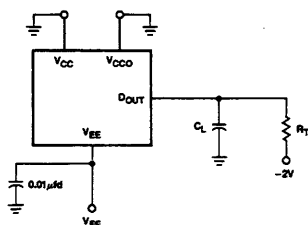
Minimum Pulse Widths

No.	Parameters	Input	Pulse	Time (ns)
14	Write Pulse	\overline{WE}_A or \overline{WE}_B	HIGH - LOW - HIGH	18
15	Latch Data Capture	LE_A or LE_B	LOW - HIGH - LOW	10

$WE_A = WE_{AC} \bullet (WE_{AL} + WE_{AH})$
 $WE_B = WE_{BC} \bullet (WE_{BL} + WE_{BH})$

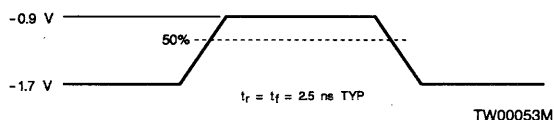
** Y_A and Y_B Are Tested Independently

SWITCHING TEST CIRCUIT



TC000232

SWITCHING TEST WAVEFORM



TW00053M

$R_T = 50 \Omega$ termination of measurement system

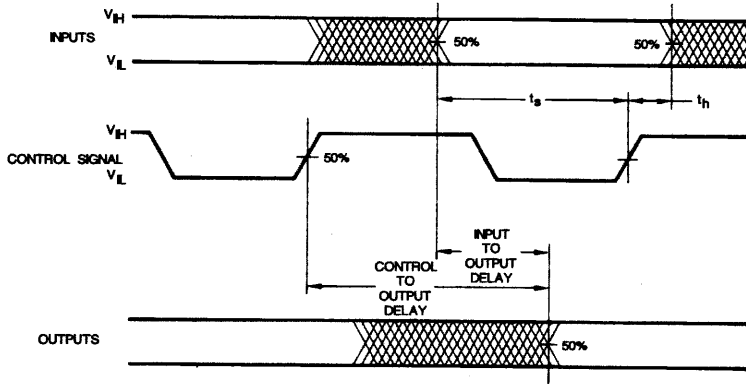
$C_L = 30$ pF (including stray jig capacitance)

KEY TO SWITCHING WAVEFORMS

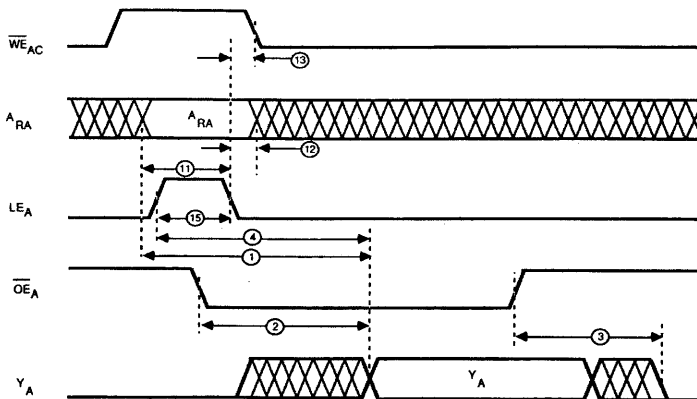
WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE: ANY CHANGE PERMITTED	CHANGING: STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING WAVEFORMS



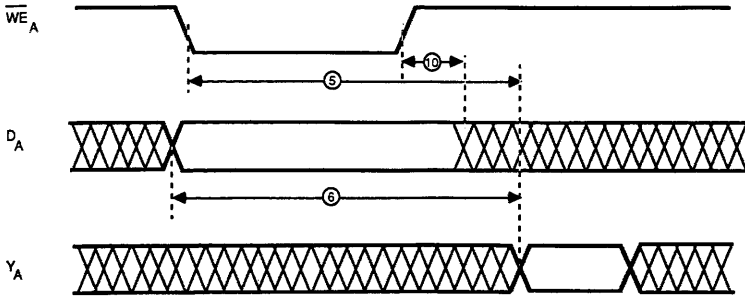
WFR02991



WF023070

Read Function (same for B Port)

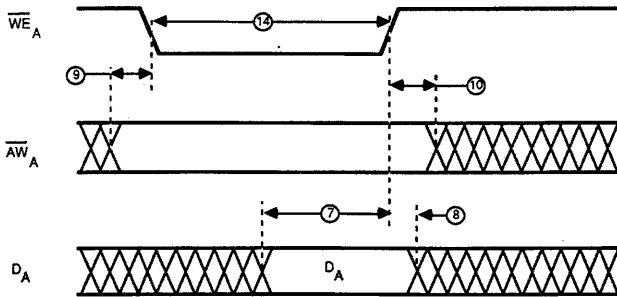
SWITCHING WAVEFORMS (Cont'd.)



WF023050

Note: $\overline{LE}_A = \text{HIGH}$
 $\overline{OE}_A = \text{LOW}$

Transparency Function (same for B Port)

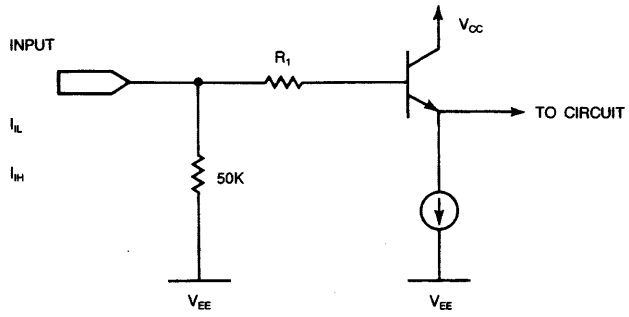


WF023060

Write Function (same for B Port)

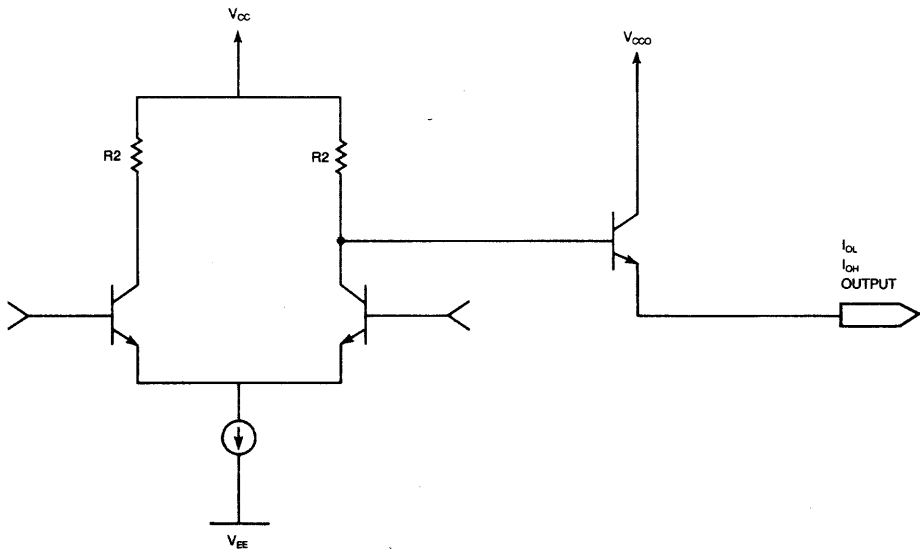
I/O CURRENT INTERFACE DIAGRAM

INPUT CIRCUIT



IC000920

OUTPUT CIRCUIT



IC000930

Am29325

32-Bit Floating-Point Processor



Am29325

DISTINCTIVE CHARACTERISTICS

- Single VLSI device performs high-speed floating-point arithmetic
 - Floating-point addition, subtraction, and multiplication in a single clock cycle
 - Internal architecture supports sum-of-products, Newton-Raphson division
- 32-bit, three-bus flow-through architecture
 - Programmable I/O allows interface to 32- and 16-bit systems
- IEEE and DEC formats
 - Performs conversions between formats
 - Performs integer ↔ floating-point conversions
- Six flags indicate operation status
- Register enables eliminate clock skew
- Input and output registers can be made transparent independently

GENERAL DESCRIPTION

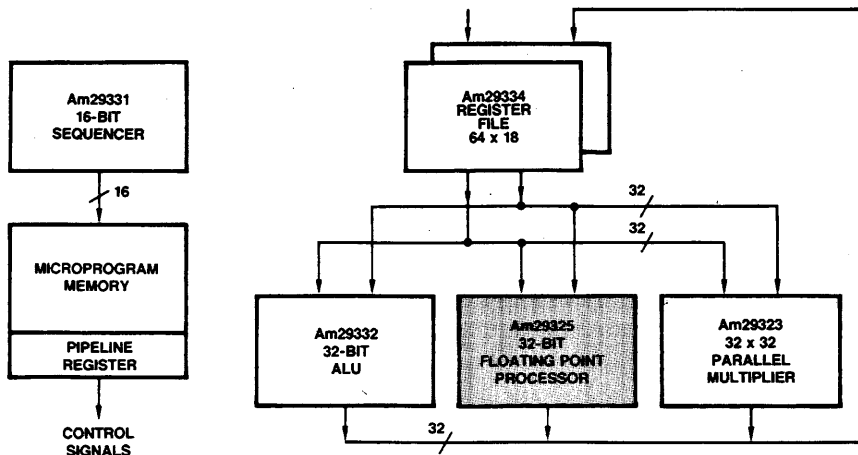
The Am29325 is a high-speed floating-point processor unit. It performs 32-bit single-precision floating-point addition, subtraction, and multiplication operations in a single VLSI circuit, using the format specified by the proposed IEEE floating-point standard, P754. The DEC single-precision floating-point format is also supported. Operations for conversion between 32-bit integer format and floating-point format are available, as are operations for converting between the IEEE and DEC floating-point formats. Any operation can be performed in a single clock cycle. Six flags — invalid operation, inexact result, zero, not-a-number, overflow, and underflow — monitor the status of operations.

The Am29325 has a three-bus, 32-bit architecture, with two input buses and one output bus. This configuration provides

high I/O bandwidth, allows access to all buses and affords a high degree of flexibility when connecting this device in a system. All buses are registered with each register having a clock enable. Input and output registers may be made transparent independently. Two other I/O configurations, a 32-bit, two-bus architecture and a 16-bit, three-bus architecture, are user-selectable, easing interface with a wide variety of systems. Thirty-two-bit internal feedforward datapaths support accumulation operations, including sum-of-products and Newton-Raphson division.

Fabricated with the high-speed IMOX™ bipolar process, the Am29325 is powered by a single 5-volt supply. The device is housed in a 145-terminal pin-grid-array package.

Am29300 FAMILY HIGH-PERFORMANCE SYSTEM BLOCK DIAGRAM



AF004650

Am29337

16-Bit Bounds Checker



Am29337

DISTINCTIVE CHARACTERISTICS

- **Double Comparator**
 - Compares a 16-bit input number with a lower limit and an upper limit
- **Cascadable**
 - 16-bit cascadable to longer words
- **Out-of-Bounds Flag**
 - Flags values that are outside the bounds of a lower and an upper limit
- **Compares Signed or Unsigned Numbers**
- **28-Pin Packages**

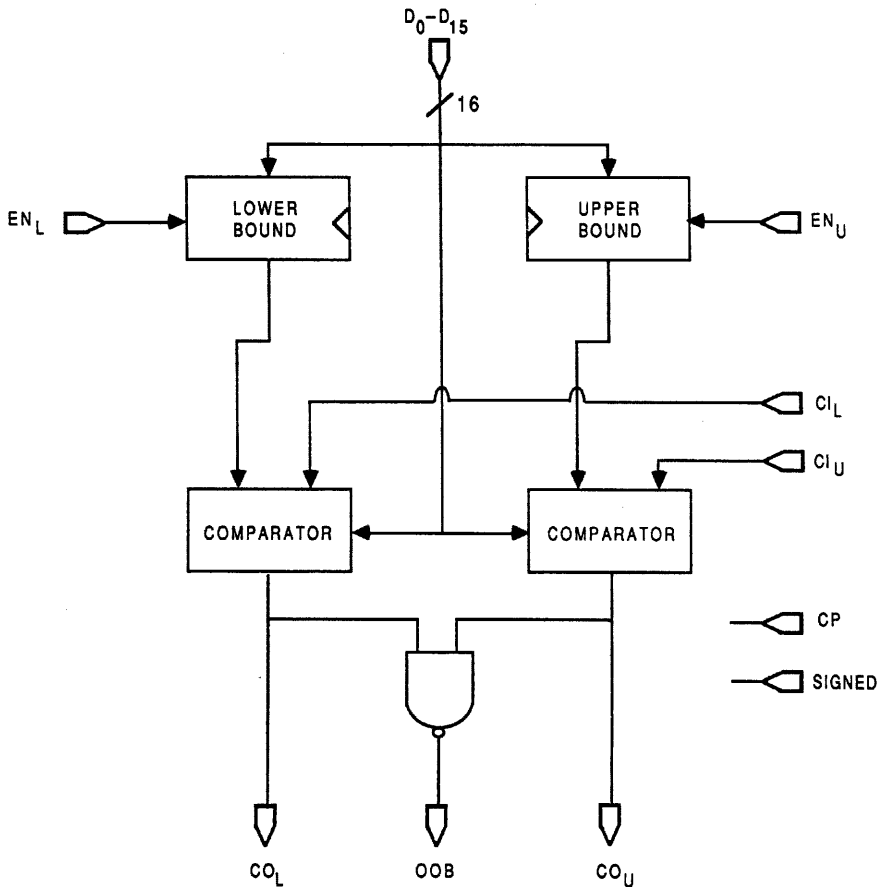
GENERAL DESCRIPTION

The Am29337 is the 16-bit bounds checker that compares a 16-bit signed or unsigned number with a lower and an upper limit stored in the registers. The part flags values that

are out of bounds, or triggers a counter used to count the number of values that lie within the given range.

The Am29337 is cascadable up to 32 bits or greater.

BLOCK DIAGRAM



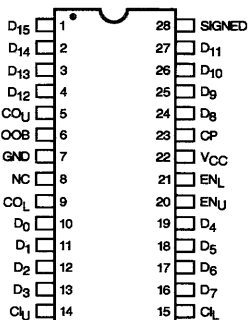
BD006640

RELATED AMD PRODUCTS

Part No.	Description
Am2900	Bipolar Bit-Slice Family
Am29C00	CMOS Bit-Slice Family
Am29112	Bipolar 8-Bit Cascadable Microprogram Sequencer
Am29114	Bipolar Interrupt Controller
Am29116	Bipolar 16-Bit Microprogrammable Controller
Am29C116	CMOS 16-Bit Microprogrammable Controller
Am29117	Bipolar 16-Bit Two-Port Microprogrammable Controller
Am29C117	CMOS 16-Bit Two-Port Microprogrammable Controller
Am29C323	CMOS 32 x 32 Multiplier
Am29325	Bipolar 32-Bit Floating Point Processor
Am29C325	CMOS 32-Bit Floating Point Processor
Am29331	Bipolar 16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29332	Bipolar 32-Bit Non-Cascadable ALU
Am29C332	CMOS 32-Bit Non-Cascadable ALU
Am29334	Bipolar 64 x 18 Four-Port Dual-Access Register File
Am29C334	CMOS 64 x 18 Four-Port Dual-Access Register File

CONNECTION DIAGRAM

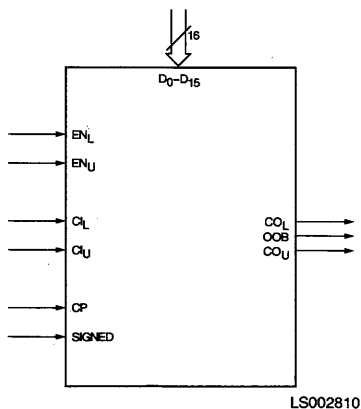
Top View



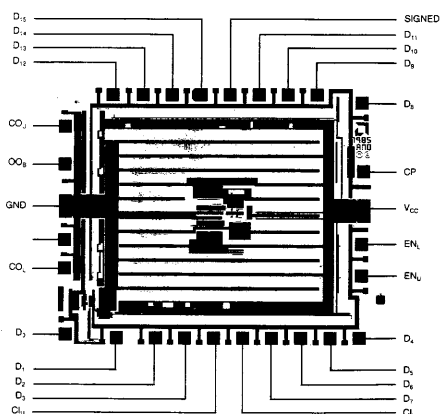
CD010100

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



METALLIZATION AND PAD LAYOUT



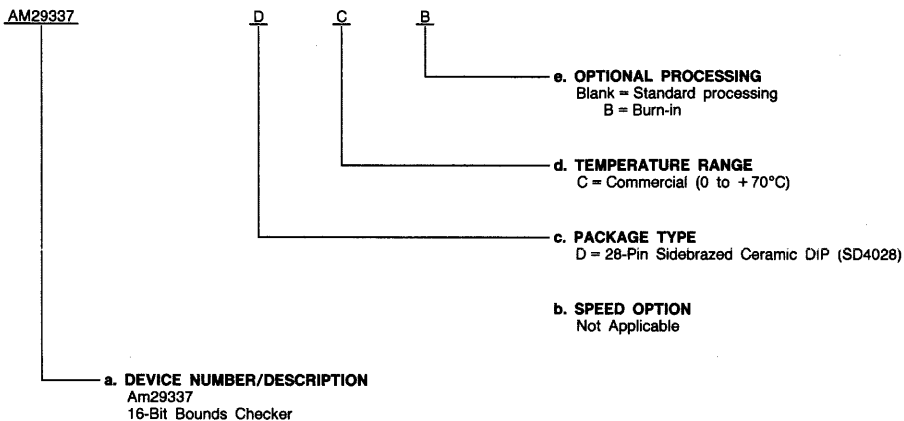
Die Size: 117 x 143
Gate Count: 250

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Package Type**
- d. **Temperature Range**
- e. **Optional Processing**



Valid Combinations	
AM29337	DC, DCB,

Valid Combinations

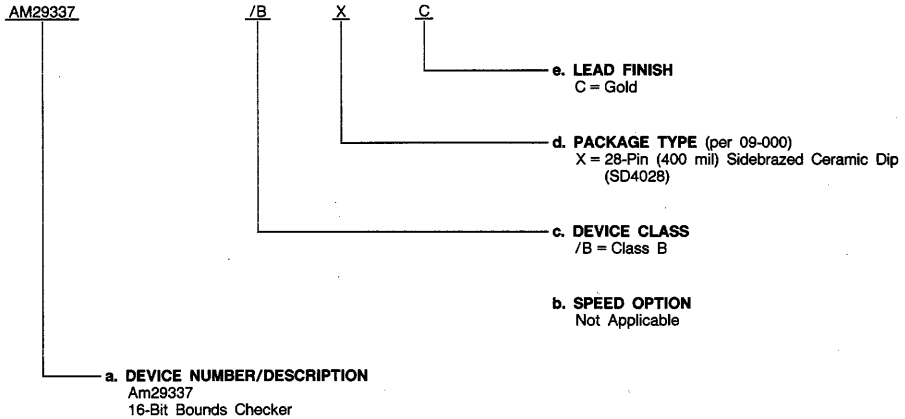
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION (Cont'd.)

APL Products

AMD products for Aerospace and Defense applications are available in several packages and operating ranges. APL (Approved Products List) products are fully compliant with MIL-STD-883C requirements. The order number (Valid Combination) for APL products is formed by a combination of:

- a. Device Number
- b. Speed Option (if applicable)
- c. Device Class
- d. Package Type
- e. Lead Finish



Valid Combinations	
AM29337	/BXC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations or to check for newly released valid combinations.

Group A Tests

Group A tests consist of Subgroups
1, 2, 3, 7, 8, 9, 10, 11.

PIN DESCRIPTION

Cl_L, Cl_U Carry-In (Inputs)

Carry input for cascading.

CO_L, CO_U Carry Out (Outputs)

Carry outputs for the result of comparison.

CP System Clock (Input)

Clocks limit registers at the LOW-to-HIGH transition.

D₀-D₁₅ Data Input (Input)

Input to the comparators and limit registers.

EN_L, EN_U Load Enable (Inputs)

Loads enables for the limit registers.

OOB Out-of-Bounds Flag (Output)

Flags values that are out of bounds. Defined as $\overline{CO_L} \cdot CO_U$.

SIGNED Sign Input (Input)

Selects signed comparisons when HIGH and unsigned comparisons when LOW.

FUNCTIONAL DESCRIPTION

The Am29337 is a high-speed bounds checker that determines if a 16-bit number lies within a lower and an upper limit. It consists of two comparators and two limit registers, as shown in the Block Diagram.

Limit Registers, Double Comparator

The Am29337 has a lower limit register and an upper limit register. The values of these two registers are loaded from the D-bus with the load enable inputs EN_L and EN_U on the clock's rising edge. The values of the data present on the D-bus are compared with the values stored in the limit registers through the two comparators. The comparators operate on signed numbers when SIGNED is HIGH and on unsigned numbers when it is LOW. The results of the comparisons are given by the outputs CO_L, CO_U, and OOB. The definitions of carry inputs Cl_L and Cl_U are given in Table 1, and the combination of the different regions in Table 2. If the data being compared is out of the region, the out-of-bounds flag, OOB, which is defined as $\overline{CO_L} \cdot CO_U$, is set.

Cascading

Comparison of numbers longer than 16 bits requires cascading of two or more bounds-checker slices. Figure 1 shows an example of this for a 32-bit bounds checker. The comparison starts from the least significant slice. CO_L, CO_U, and OOB of the most significant slice act as outputs of the overall bounds checker, while CO_L and CO_U of the least significant slice are connected to Cl_L and Cl_U of the most significant slice. Cl_L and Cl_U of the least significant slice act as inputs to the overall bounds checker. The SIGNED input of the most significant slice identifies the value when being compared with either signed or unsigned number when the SIGNED input of the least significant slice is tied LOW.

The comparison can start from the most significant slice. In this case, CO_L, CO_U, OOB of the least significant slice act as outputs of the overall bounds checker, while CO_L and CO_U of the most significant slice are connected to Cl_L and Cl_U of the least significant slice.

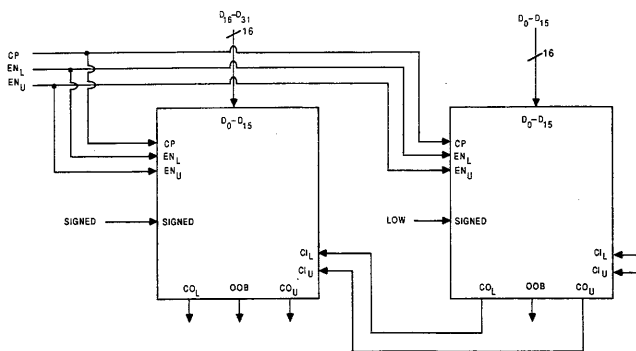
TABLE 1. DEFINITION OF CO_L AND CO_U

Inputs		Outputs	
Cl _L	Cl _U	CO _L	CO _U
0	0	L < D	D < U
0	1	L < D	D ≤ U
1	0	L ≤ D	D < U
1	1	L ≤ D	D ≤ U

Note:
 D = Data Input
 L = Lower Unit
 U = Upper Unit

TABLE 2. DIFFERENT COMBINATIONS OF REGIONS

Inputs		Outputs			Description
Cl _L	Cl _U	CO _L	CO _U	OOB	
0	0	0	0	1	Impossible Combination
		0	1	1	D ≤ L
		1	0	1	U ≤ D
		1	1	0	L < D < U
0	1	0	0	1	Impossible Combination
		0	1	1	D ≤ L
		1	0	1	U < D
		1	1	0	L < D ≤ U
1	0	0	0	1	Impossible Combination
		0	1	1	D < L
		1	0	1	U ≤ D
		1	1	0	L ≤ D < U
1	1	0	0	1	Impossible Combination
		0	1	1	D < L
		1	0	1	U < D
		1	1	0	L ≤ D ≤ U



AF004531

Figure 1. 32-Bit Bounds Checker

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Temperature Under Bias — T_C	-55 to +125°C
Supply Voltage to Ground	
Potential Continuous	-0.5 to +7.0 V
DC Voltage Applied to Outputs	
for HIGH State	-0.5 V to V_{CC} Max.
DC Input Voltage	-0.5 to +5.5 V
DC Output Current, into Outputs	30 mA
DC Input Current	-30 to +5.0 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature (T_A)	0 to +70°C
Supply Voltage (V_{CC})	+4.75 to +5.25 V
Military (M) Devices	
Temperature (T_C)	-55 to +125°C
Supply Voltage (V_{CC})	+4.5 to +5.5 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

Thermal Resistance (Preliminary) – SD4028
 $\theta_{JA} = 40^\circ\text{C/W}$
 $\theta_{JC} = 15^\circ\text{C/W}$

DC CHARACTERISTICS over operating range unless otherwise specified (for APL Products, Group A, Subgroups 1, 2, 3 are tested unless otherwise noted)

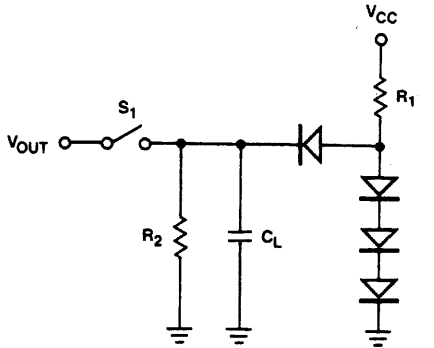
Parameter Symbol	Parameter Description	Test Conditions (Note 1)	Min.	Max.	Units
V_{OH}	Output HIGH Voltage	$V_{CC} = \text{Min.}, V_{IN} = V_{IL}$ or V_{IH} $I_{OH} = -1.0$ mA	2.4		V
V_{OL}	Output LOW Voltage	$V_{CC} = \text{Min.}, V_{IN} = V_{IL}$ or V_{IH} $I_{OL} = 8.0$ mA		0.5	V
V_{IH}	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs	2.0		V
V_{IL}	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs		0.8	V
V_I	Input Clamp Voltage	$V_{CC} = \text{Min.}, I_{IN} = -18$ mA		-1.2	V
I_{IL}	Input LOW Current	$V_{CC} = \text{Max.}, V_{IN} = 0.5$ V		-0.5	mA
I_{IH}	Input HIGH Current	$V_{CC} = \text{Max.}, V_{IN} = 2.4$ V		50	μA
I_I	Input HIGH Current	$V_{CC} = \text{Max.}, V_{IN} = 5.5$ V		1	mA
I_{OZH} I_{OZL}	$F_0 - F_{31}$ Off State (High Impedance) Output Current	$V_{CC} = \text{Max.}$	$V_O = 2.4$ V	25	μA
			$V_O = 0.4$ V	-25	
I_{SC}	Output Short-Circuit Current (Note 2)	$V_{CC} = \text{Max.}, V_O = 0$ V	-15	-50	mA
I_{CC}	Power Supply Current	$V_{CC} = \text{Max.}$	$T_A = +25^\circ\text{C}$	180	mA
			$T_A = 0$ to $+70^\circ\text{C}$	230	
			$T_A = +70^\circ\text{C}$	220	
			$T_C = -55$ to 125°C	235	
			$T_C = 125^\circ\text{C}$	215	

Notes: 1. For conditions as Min. or Max., use the appropriate value specified under Operating Ranges for the applicable device type.
 2. Not more than one output should be shorted at a time. Duration of the short-circuit test should not exceed one second.

SWITCHING CHARACTERISTICS over operating range unless otherwise specified (for APL Products, Subgroups 9, 10, 11 are tested unless otherwise noted)

No.	Parameter Symbol		COM'L	MIL	Units
			Max. Delay	Max. Delay	
1	t _{PD}	D ₀ -D ₁₅ to CO _L , CO _U , OOB	21	23	ns
2	t _{PC}	CI _L , CI _U to CO _L , CO _U , OOB	13	14	ns
3	t _{PS}	SIGNED to CO _L , CO _U , OOB	18	18	ns
4	t _{CP0}	CP to CO _L , CO _U , OOB	22	24	ns
5	t _{SD}	D ₀ -D ₁₅ Setup Time With Regard to CP ↑	12	13	ns
6	t _{SL}	EN _L , EN _U Setup Time With Regard to CP ↑	12	13	ns
7	t _{HD}	D ₀ -D ₁₅ Hold Time	2	2	ns
8	t _{HL}	EN _L , EN _U Hold Time	0	0	ns
9	t _{PWL}	Clock Pulse Width LOW	12	12	ns
10	t _{PWH}	Clock Pulse Width HIGH	12	12	ns

SWITCHING TEST CIRCUIT



TCR01240

$$R_2 = \frac{2.4 \text{ V}}{I_{OH}}$$

$$R_1 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OL} + V_{OL}} \cdot R_2$$

Normal Outputs

- Notes: 1. $C_L = 50 \text{ pF}$ includes scope probe, wiring, and stray capacitances without device in test fixture.
 2. S_1 is closed during function tests and all AC tests except output enable tests.
 3. $C_L = 5.0 \text{ pF}$ for output disable tests.

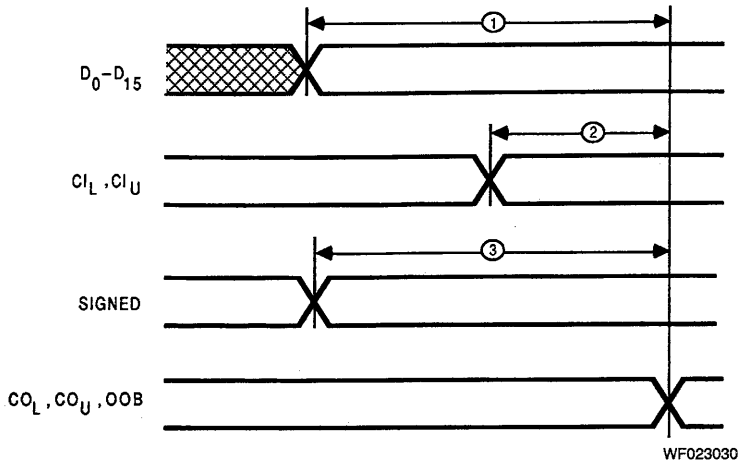
SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

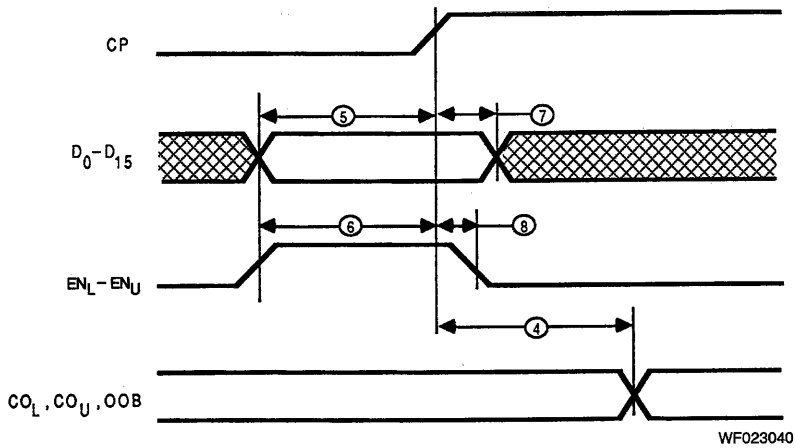
WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

SWITCHING WAVEFORMS (Cont'd.)

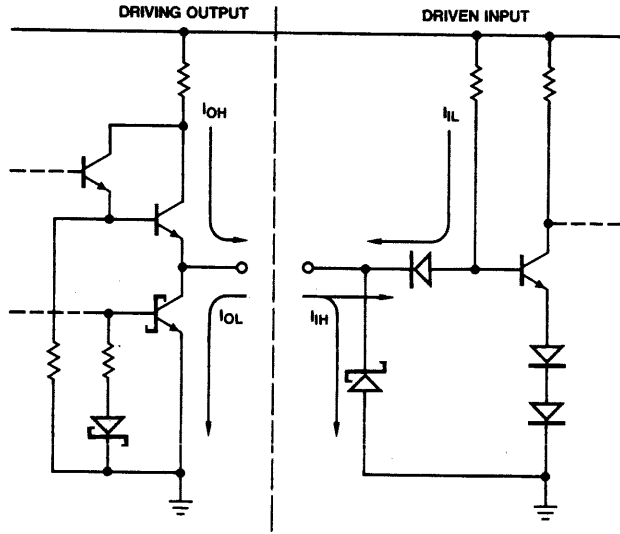


Propagation Delays from Data Input to Output



Loading the Limit Registers

INPUT/OUTPUT CIRCUIT DIAGRAM



ICR00480

$C_I \approx 5.0$ pF, All inputs

$C_O \approx 5.0$ pF, all outputs

Am29338

32-Bit Byte Queue

ADVANCE INFORMATION

Am29338

DISTINCTIVE CHARACTERISTICS

- **Intelligent FIFO Array**
 - Array of four intelligent FIFO buffers, each 9 bits wide, 32 bits deep (RAM-based)
- **Queuing/Dequeuing**
 - Allows variable width queuing/dequeuing in one cycle
- **Byte Rotation**
 - Four bytes can be rotated at the input as well as at the output of the Byte Queue. This allows interfacing between incompatible byte assignments.
- **Asynchronous and Synchronous Operation**
 - Supports communication between systems with different clocks and different bus widths
- **Retransmit**
 - Data can be read out repeatedly
- **Horizontal Cascading**
 - Up to four devices allow simultaneous input or output up to 16 bytes
- **Parity Check**
 - Protects data at the input and the output

GENERAL DESCRIPTION

The Am29338 is an intelligent FIFO that allows up to four bytes to be queued and up to four bytes to be dequeued in a single cycle. When four devices are cascaded horizontally, up to sixteen bytes can be dequeued in a single cycle.

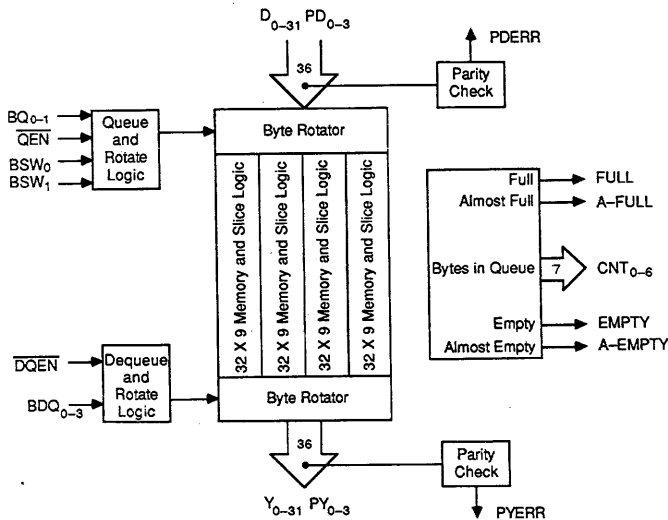
The Am29338 queues variable-length data by disassembling the input data, which is aligned on the least-significant byte of the input bus (D), into individual bytes. These bytes are packed internally in FIFO (first-in, first-out) order. The data to be dequeued is unpacked and realigned to the least-significant byte of the output bus (Y). Queuing and dequeuing can be performed simultaneously. With the

retransmit capability, the part can repeatedly send the block of data stored in the queue without having to requeue it. This is useful for retransmitting a block of data upon receipt of an error in I/O applications or for loop-locking in instruction-prefetch applications.

The queue operates in synchronous or asynchronous mode, and is useful as an instruction-prefetch queue or as a general-purpose FIFO buffer.

The device is manufactured in AMD's bipolar IMOX* technology and comes in a 120-lead pin-grid-array package.

BLOCK DIAGRAM



BD007490

RELATED AMD PRODUCTS

Part No.	Description
Am2900 Family	4-Bit Microprocessor Slice Family
Am29C00 Family	CMOS 4-Bit Microprocessor Slice Family
Am29C101	CMOS 16-Bit Microprocessor Slice
Am29114	Real-Time Interrupt Controller
Am29116	16-Bit Bipolar Microprocessor
Am29116A	High-Speed 16-Bit Bipolar Microprocessor
Am29L116A	Low-Power 16-Bit Bipolar Microprocessor
Am29C116	CMOS 16-Bit Microprocessor
Am29C116-1	CMOS 16-Bit Microprocessor
Am29325	32-Bit Floating Point Processor
Am29C325	CMOS 32-Bit Floating Point Processor
Am29331	16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29332	32-Bit Extended Function ALU
Am29C332	CMOS 32-Bit Extended Function ALU
Am29334	Four-Port, Dual-Access Register File
Am29C334	CMOS Four-Port, Dual-Access Register File
Am29337	16-Bit Cascadable Bounds Checker

CONNECTION DIAGRAM Bottom View

	A	B	C	D	E	F	G	H	J	K	L	M	N
1	Y16	Y17	\overline{OE}	Y21	GNDT	PY3	Y27	Y28	VCCT	CNT2	GNDT	CNT6	BDQ3
2	PY2	Y15	Y18	Y20	Y23	Y24	Y26	Y29	Y31	CNT1	CNT4	CNT5	BDQ2
3	GNDT	Y14	Y13	Y19	Y22	VCCE	Y25	GNDE	Y30	CNT0	CNT3	BDQ0	BDQ1
4	Y12	Y11	Y10								\overline{OEN}	RESET	FXMIT
5	VCCT	Y9	Y8								\overline{OEN}	BSW1	DOCLK
6	Y7	PY1	GNDT								QCLK	BQ1	BSW0
7	Y6	Y5	Y4								BQ0	NC	D30
8	Y2	Y3	VCCT								D31	D28	D29
9	GNDT	Y1	Y0								D27	D25	D26
10	PY0	PYERR	PDERR								D24	PD3	D23
11	VCCT	A-FULL	PD0	D2	VCCE	D6	D7	D12	GNDE	D15	D22	D20	D21
12	FULL	POS1	POS0	D1	VCCE	D3	D8	D9	GNDE	D14	PD2	D19	D18
13	A-EMPTY	EMPTY	D5	D0	VCCE	D4	PD1	D11	GNDE	D13	D10	D16	D17

CD011040

Legend: GNDE: GND, ECL
 GNDT: GND, TTL
 VCCE: VCC, ECL
 VCCT: VCC, TTL

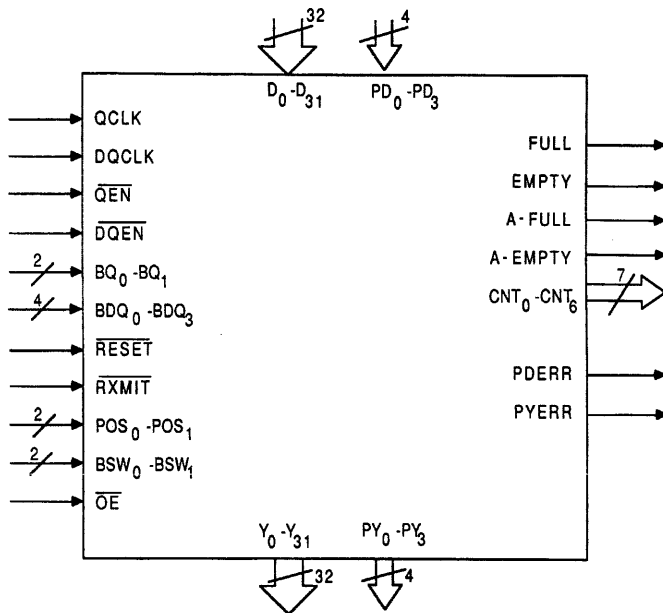
PIN DESIGNATIONS
(Sorted by Pin Number)

PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME
1	A1	Y ₁₆	115	C5	Y ₈	40	G11	D ₇	27	L10	D ₂₄
120	A2	PY ₂	113	C6	GND, TTL	36	G12	D ₈	88	L11	D ₂₂
59	A3	GND, TTL	52	C7	Y ₄	96	G13	PD ₁	32	L12	PD ₂
58	A4	Y ₁₂	53	C8	V _{CC} , TTL	69	H1	Y ₂₈	35	L13	D ₁₀
56	A5	V _{CC} , TTL	109	C9	Y ₀	10	H2	Y ₂₉	75	M1	CNT ₆
114	A6	Y ₇	48	C10	PDERR	68	H3	GND, ECL	15	M2	CNT ₅
54	A7	Y ₆	44	C11	PD ₀	34	H11	D ₁₂	77	M3	BDQ ₀
51	A8	Y ₂	104	C12	POS ₀	95	H12	D ₉	78	M4	RESET
50	A9	GND, TTL	41	C13	D ₅	94	H13	D ₁₁	80	M5	BSW ₁
49	A10	PY ₀	4	D1	Y ₂₁	11	J1	V _{CC} , TTL	81	M6	BQ ₁
47	A11	V _{CC} , TTL	63	D2	Y ₂₀	71	J2	Y ₃₁	82	M7	NC
106	A12	FULL	3	D3	Y ₁₉	70	J3	Y ₃₀	25	M8	D ₂₈
46	A13	A-EMPTY	102	D11	D ₂	38	J11	GND, ECL	86	M9	D ₂₅
61	B1	Y ₁₇	43	D12	D ₁	38	J12	GND, ECL	87	M10	PD ₃
60	B2	Y ₁₅	103	D13	D ₀	38	J13	GND, ECL	89	M11	D ₂₀
119	B3	Y ₁₄	5	E1	GND, TTL	13	K1	CNT ₂	30	M12	D ₁₉
117	B4	Y ₁₁	65	E2	Y ₂₃	72	K2	CNT ₁	91	M13	D ₁₆
116	B5	Y ₉	64	E3	Y ₂₂	12	K3	CNT ₀	16	N1	BDQ ₃
55	B6	PY ₁	98	E11	V _{CC} , ECL	92	K11	D ₁₅	76	N2	BDQ ₂
112	B7	Y ₅	98	E12	V _{CC} , ECL	33	K12	D ₁₄	17	N3	BDQ ₁
111	B8	Y ₃	98	E13	V _{CC} , ECL	93	K13	D ₁₃	19	N4	RXMIT
110	B9	Y ₁	6	F1	PY ₃	14	L1	GND, TTL	20	N5	DQCLK
108	B10	PYERR	66	F2	Y ₂₄	74	L2	CNT ₄	21	N6	BSW ₀
107	B11	A-FULL	8	F3	V _{CC} , ECL	73	L3	CNT ₃	24	N7	D ₃₀
45	B12	POS ₁	100	F11	D ₆	18	L4	QEN	84	N8	D ₂₉
105	B13	EMPTY	42	F12	D ₃	79	L5	QEN	26	N9	D ₂₆
2	C1	OE	101	F13	D ₄	23	L6	QCLK	28	N10	D ₂₃
62	C2	Y ₁₈	9	G1	Y ₂₇	22	L7	BQ ₀	29	N11	D ₂₁
118	C3	Y ₁₃	67	G2	Y ₂₆	83	L8	D ₃₁	90	N12	D ₁₈
57	C4	Y ₁₀	7	G3	Y ₂₅	85	L9	D ₂₇	31	N13	D ₁₇

PIN DESIGNATIONS
(Sorted by Pin Name)

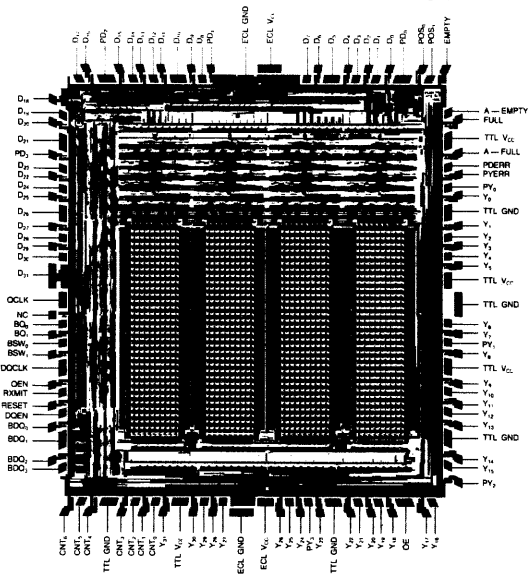
PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME
82	M7	NC	34	H11	D ₁₂	59	A3	GND, TTL	51	A8	Y ₂
46	A13	A-EMPTY	93	K13	D ₁₃	5	E1	GND, TTL	111	B8	Y ₃
107	B11	A-FULL	33	K12	D ₁₄	50	A9	GND, TTL	52	C7	Y ₄
77	M3	BDQ ₀	92	K11	D ₁₅	2	C1	OE	112	B7	Y ₅
17	N3	BDQ ₁	91	M13	D ₁₆	44	C11	PD ₀	54	A7	Y ₆
76	N2	BDQ ₂	31	N13	D ₁₇	96	G13	PD ₁	114	A6	Y ₇
16	N1	BDQ ₃	90	N12	D ₁₈	32	L12	PD ₂	115	C5	Y ₈
22	L7	BQ ₀	30	M12	D ₁₉	87	M10	PD ₃	116	B5	Y ₉
81	M6	BQ ₁	89	M11	D ₂₀	48	C10	PDERR	57	C4	Y ₁₀
21	N6	BSW ₀	29	N11	D ₂₁	104	C12	POS ₀	117	B4	Y ₁₁
80	M5	BSW ₁	88	L11	D ₂₂	45	B12	POS ₁	58	A4	Y ₁₂
12	K3	CNT ₀	28	N10	D ₂₃	49	A10	PY ₀	118	C3	Y ₁₃
72	K2	CNT ₁	27	L10	D ₂₄	55	B6	PY ₁	119	B3	Y ₁₄
13	K1	CNT ₂	86	M9	D ₂₅	120	A2	PY ₂	60	B2	Y ₁₅
73	L3	CNT ₃	26	N9	D ₂₆	6	F1	PY ₃	1	A1	Y ₁₆
74	L2	CNT ₄	85	L9	D ₂₇	108	B10	PYERR	61	B1	Y ₁₇
15	M2	CNT ₅	25	M8	D ₂₈	23	L6	QCLK	62	C2	Y ₁₈
75	M1	CNT ₆	84	N8	D ₂₉	79	L5	QEN	3	D3	Y ₁₉
103	D13	D ₀	24	N7	D ₃₀	78	M4	RESET	63	D2	Y ₂₀
43	D12	D ₁	83	L8	D ₃₁	19	N4	RXMIT	4	D1	Y ₂₁
102	D11	D ₂	20	N5	DQCLK	98	E11	V _{CC} , ECL	64	E3	Y ₂₂
42	F12	D ₃	18	L4	DQEN	98	E12	V _{CC} , ECL	65	E2	Y ₂₃
101	F13	D ₄	105	B13	EMPTY	98	E13	V _{CC} , ECL	66	F2	Y ₂₄
41	C13	D ₅	106	A12	FULL	8	F3	V _{CC} , ECL	7	G3	Y ₂₅
100	F11	D ₆	38	J11	GND, ECL	56	A5	V _{CC} , TTL	67	G2	Y ₂₆
40	G11	D ₇	38	J12	GND, ECL	53	C8	V _{CC} , TTL	9	G1	Y ₂₇
36	G12	D ₈	38	J13	GND, ECL	47	A11	V _{CC} , TTL	69	H1	Y ₂₈
95	H12	D ₉	68	H3	GND, ECL	11	J1	V _{CC} , TTL	10	H2	Y ₂₉
35	L13	D ₁₀	113	C6	GND, TTL	109	C9	Y ₀	70	J3	Y ₃₀
94	H13	D ₁₁	14	L1	GND, TTL	110	B9	Y ₁	71	J2	Y ₃₁

LOGIC SYMBOL



LS002851

METALLIZATION AND PAD LAYOUT



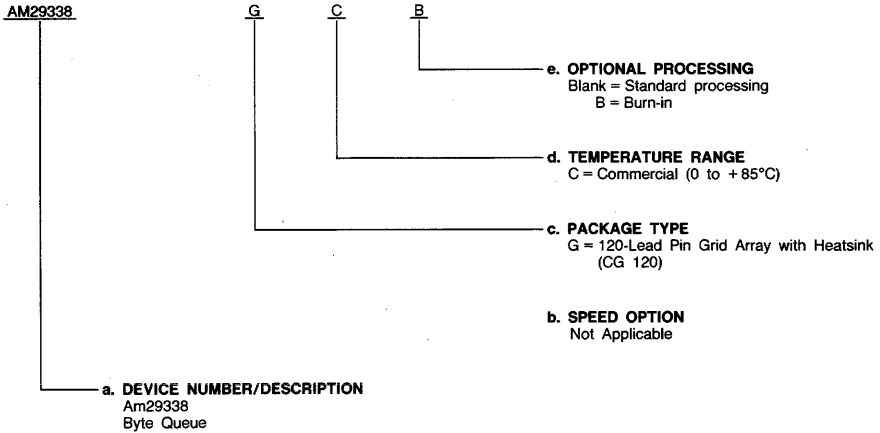
Die Size: 270 x 290 mils²
 Gate Count: 9000

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Package Type**
- d. **Temperature Range**
- e. **Optional Processing**



Valid Combinations	
AM29338	GC, GCB

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

A-EMPTY Almost Empty (Output; Active HIGH)

Indicates that there are less than four bytes of data in the queue. It is used in either synchronous or asynchronous operation.

A-FULL Almost Full (Output; Active HIGH)

Indicates that there are less than four bytes of space remaining. It is used in either synchronous or asynchronous operation.

BDQ₀ - BDQ₃ Bytes Dequeued (Input)

Selects the number of bytes to be dequeued (see Table 2). The byte queue must operate synchronously to be able to dequeue more than four bytes in a single cycle.

BQ₀ - BQ₁ Bytes Queued (Input)

Selects the number of bytes to be queued (see Table 1).

BSW₀ - BSW₁ Byte Swap (Input)

Allows the bytes on the input to be reordered (see Table 3).

CNT₀ - CNT₆ Byte Count (Output)

Gives the current number of bytes in the queue. These are used only in synchronous operation.

D₀ - D₃₁ Data Input (Input)

Data inputs to be queued.

DQCLK Dequeue Clock (Input)

Dequeues the number of bytes set up on the Y bus. A LOW-to-HIGH transition on this input adjusts the internal dequeue pointers by the number set up on the BDQ lines.

DQEN Dequeue Enable (Input; Active LOW)

While \overline{DQEN} is LOW, dequeuing is performed normally. When \overline{DQEN} is HIGH, DQCLK is disabled.

EMPTY Empty (Output; Active HIGH)

Indicates that the queue is empty. It is used in either synchronous or asynchronous operation.

FULL Full (Output; Active HIGH)

Indicates that the queue is full. It is used in either synchronous or asynchronous operation.

OE Output Enable (Input; Active LOW)

When \overline{OE} is LOW, the four bytes following the current dequeue pointer and the corresponding parity bits are on Y and PY outputs. When \overline{OE} is HIGH, Y and PY outputs are three stated.

PD₀ - PD₃ Data Input Parity (Input)

The input parity bits for the corresponding byte on the D inputs. Only the bytes to be queued and the corresponding

PD lines are checked for possible parity error. The byte queue has the even parity.

PDERR Data Input Parity Error (Output; Active HIGH)

If any of the bytes to be queued have a parity error, PDERR is asserted.

POS₀ - POS₁ Position (Input)

These inputs are used to program the location of each byte queue in horizontally cascaded system upon \overline{RESET} (see Table 4).

PY₀ - PY₃ Output Data Parity (Output; Three State)

The output parity bits for Y outputs. When \overline{OE} is HIGH, the parity bits of the four bytes following the dequeue pointer appear on these outputs. The byte queue has the even parity.

PYERR Y Output Parity Error (Output; Active HIGH)

If any of the bytes on the output has a parity error, PYERR is asserted.

QCLK Queue Clock (Input)

When QCLK is LOW, the number of bytes set up on the BQ lines are written into the next free space in the queue from the data set up on the D inputs. On a LOW-to-HIGH transition of this input, the internal queue pointers are updated. If \overline{QEN} is HIGH, QCLK has no effect.

QEN Queue Enable (Input; Active LOW)

When \overline{QEN} is LOW, queuing is performed normally. When \overline{QEN} is HIGH, QCLK is disabled.

RESET Reset (Input; Active LOW)

When \overline{RESET} is LOW, both the internal queue pointer and the internal dequeue pointer are reset to the first RAM location and both EMPTY and A_EMPTY are asserted.

RXMIT Retransmit (Input; Active LOW)

When \overline{RXMIT} is LOW, the internal dequeue pointers are reset to the first RAM location while the internal queue pointers remain unchanged. This allows the data contained between the current queue pointer and the first RAM location to become available for dequeuing again. The effect of asserting \overline{RXMIT} is defined only if 128 bytes or less have been queued since the last assertion of \overline{RESET} (see Figure 5).

Y₀ - Y₃₁ Data Output (Output; Three State)

The four bytes following the current dequeue pointer appear on these outputs when \overline{OE} is LOW. When \overline{OE} is HIGH, they are three stated.

FUNCTIONAL DESCRIPTION

Architecture

The Am29338 is a 32-bit high-performance general-purpose intelligent FIFO that stores up to 128 bytes in the internal RAM slices and queues or dequeues up to four bytes in a single cycle. The byte queue is divided into five functional blocks: 1) four memory-slice logics, 2) byte rotators for input and output buses, 3) rotate-enable logic, 4) byte-count logic, and 5) full/empty-generate logic. The byte-oriented parity checking is provided on both the D-input bus and the Y-output bus. Figure 1 shows a detailed block diagram of the byte queue.

Memory-Slice Logic

Figure 2 shows a detail of the memory-slice logic. It consists of a 32 x 9 RAM, queue and dequeue pointers, adders for the pointers, and a full/empty detector. The RAM has independent 9-bit read and write ports. Both ports are accessible simultaneously if different RAM locations are operated on. A parity bit is stored along with its corresponding byte into the RAM.

The queue and dequeue pointers point to the next location available for dequeuing. The next locations are produced by the internal adders with BQ_{0-1} or BDQ_{0-3} and the current pointer values. When \overline{RESET} is asserted, both pointers are set to zero and the RAM is flushed. These pointers are also used to indicate that the RAM is either empty or full for each memory slice. The slice-empty or slice-full signal is used to

combinationally form FULL, A-FULL, EMPTY, and A-EMPTY signals.

Byte Rotator

There are two byte rotators in the byte queue. Each accepts 36-bit wide data and performs rotation of bytes according to the 2-bit rotate values fed from the rotate-enable logic. The input byte rotator realigns and stores the bytes to be queued into the next free slice location. The output byte rotator realigns the bytes to be dequeued to the least significant byte of the Y-output bus.

Rotate-Enable Logic

The queue and dequeue rotate-enable logic keeps track of which slice holds the first byte of the next queue/dequeue operation. A modulo-4 counter is used to rotate the data in operation and enables the correct slices by the number of bytes specified by either BQ_{0-1} or BDQ_{0-3} .

The queue rotate-enable logic also performs byte and/or word swaps on the incoming data. The input bytes are swapped in one of four ways, according to Table 3, with BSW_{0-1} and the current modulo-4 byte count through the input byte rotator.

Byte-Count Logic

This logic consists of a queue count register and a dequeue count register. The registers are incremented during a queue/dequeue operation by the number of bytes in the operation. The combinational subtract logic outside of these registers determines the number of bytes stored in the byte queue.

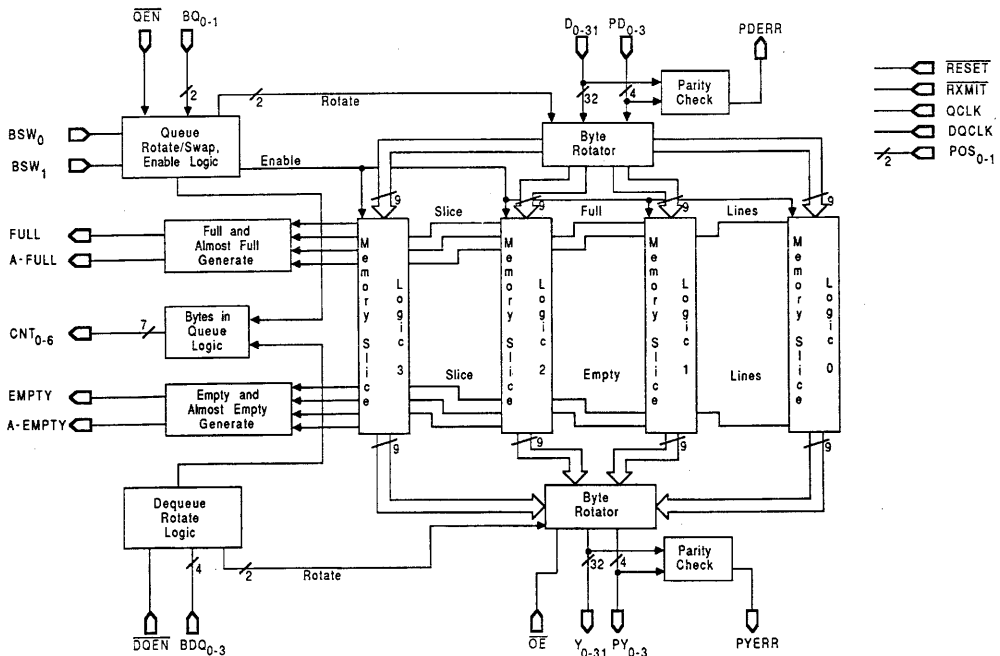
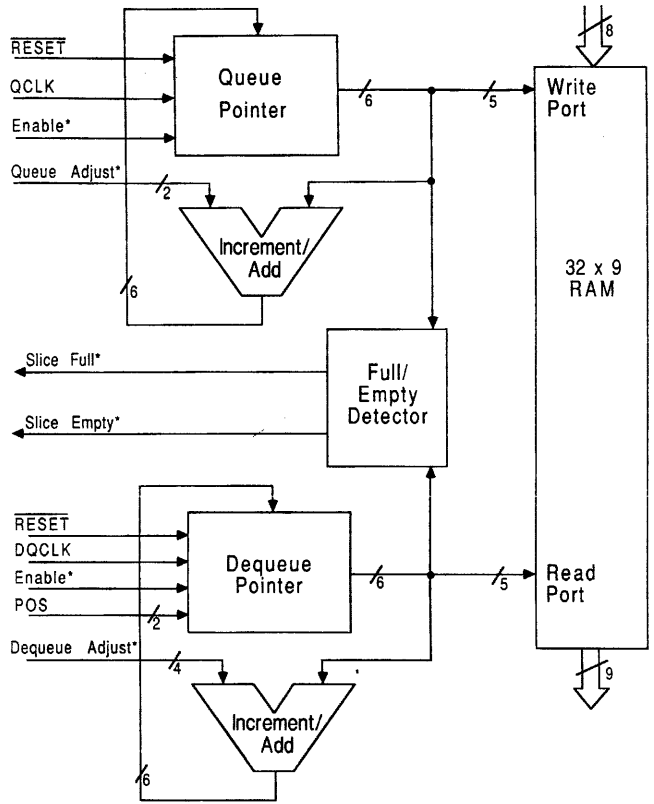


Figure 1. Am29338 Byte Queue Detailed Block Diagram

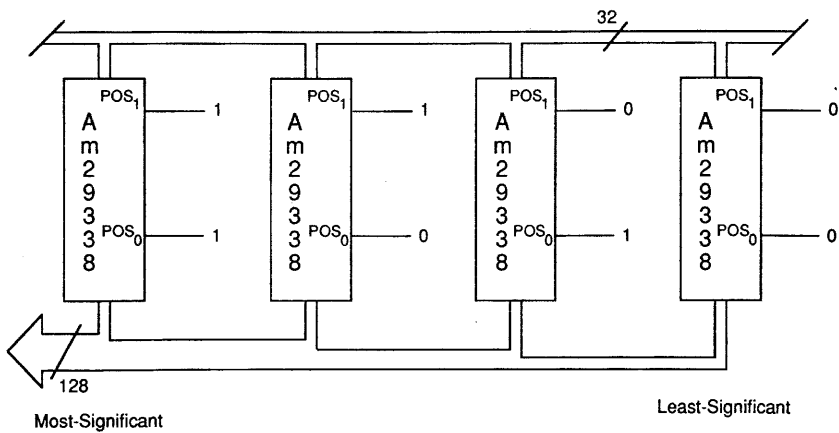
BD006902



BD006911

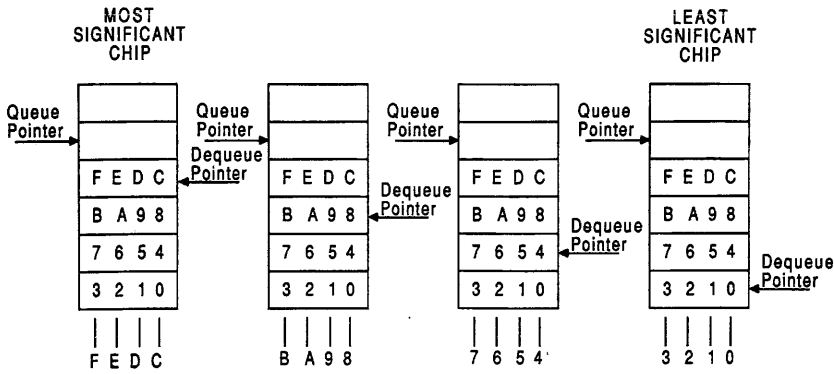
*Internally generated inputs.

Figure 2. Memory and Slice Logic



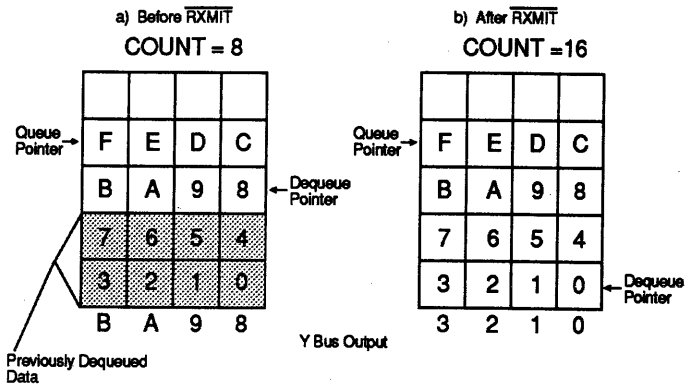
BD007010

Figure 3. Position Line Values in Horizontally Cascaded System



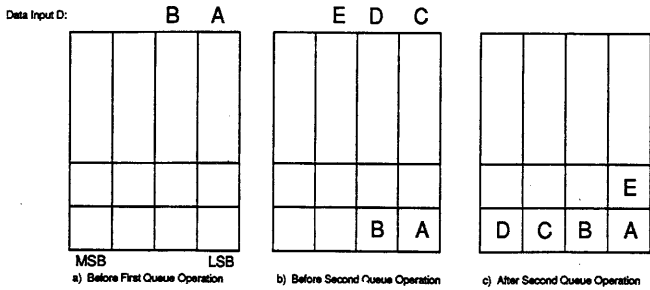
BD006930

Figure 4. An Example of Horizontal Cascading



TB001131

Figure 5. Retransmit Function with the Am29338



TB001141

Figure 6. Queuing with the Am29338

Notes: 1. Each of the four segments stands for a memory size; MSB = Most-Significant Byte, and LSB = Least-Significant Byte.

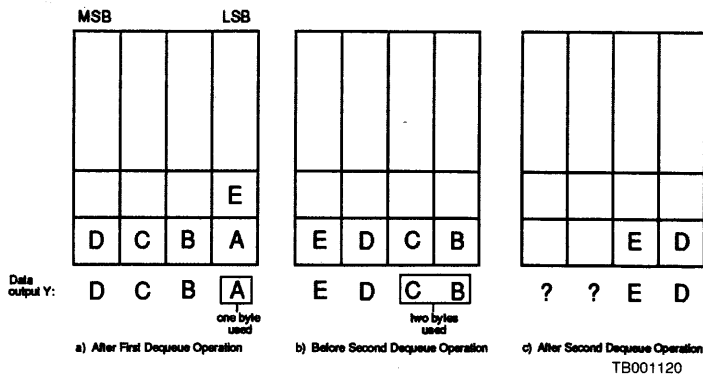


Figure 7. Dequeuing with the Am29338

- Notes: 1. Each of the four segments stands for a memory size; MSB = Most-Significant Byte, and LSB = Least-Significant Byte.
 2. First, one byte is dequeued ('A'), followed by a dequeue of two bytes ('CB').

TABLE 1. SELECTING THE NUMBER OF BYTES TO BE QUEUED

BQ ₁	BQ ₀	Bytes To Be Queued
L	H	1
H	L	2
H	H	3
L	L	4

Key: L = LOW
H = HIGH

TABLE 2. SELECTING THE NUMBER OF BYTES TO BE DEQUEUED

BDQ ₃	BDQ ₂	BDQ ₁	BDQ ₀	Bytes To Be Dequeued
L	L	L	H	1
L	L	H	L	2
L	L	H	H	3
L	H	L	L	4
L	H	L	H	5*
L	H	H	L	6*
L	H	H	H	7*
H	L	L	L	8*
H	L	L	H	9*
H	L	H	L	10*
H	L	H	H	11*
H	H	L	L	12*
H	H	L	H	13*
H	H	H	L	14*
H	H	H	H	15*
L	L	L	L	16*

Key: L = LOW
H = HIGH

* This is possible when four of the byte queues are cascaded together. The byte queue must be operated synchronously to select more than four bytes for dequeuing.

TABLE 3. ENCODING OF BSW INPUTS

Inputs		Outputs			
BSW ₁	BSW ₀	A	B	C	D
L	L	A	B	C	D
L	H	B	A	D	C
H	L	C	D	A	B
H	H	D	C	B	A

Key: L = LOW
H = HIGH

Note: The assumption is made that the 32-bit data "A B C D" appears on the input bus.

TABLE 4. LOCATION IDENTIFICATION FOR HORIZONTAL CASCADING

POS ₁	POS ₀	Location
L	L	0
L	H	1
H	L	2
H	H	3

Key: L = LOW
H = HIGH

Note: "0" stands for the least significant chip and "3" the most significant chip.

Operational Modes

General Operation

To enter data into the Am29338, the number of bytes to be queued is set up on the Bytes Queued (BQ) pins; the corresponding data to be queued is set up on the Data Input (D) and Data Input Parity (PD) pins, aligned to the least-significant byte. If Queue Enable (\overline{QEN}) is asserted, the data is entered into the Am29338 while the Queue Clock (QCLK) is LOW, and the internal queue pointers are updated on the LOW-to-HIGH transition of QCLK.

Figure 6 shows an example of two bytes being queued, followed by three bytes being queued. Data is packed in the Am29338 so that no holes exist.

If Output Enable (\overline{OE}) is asserted, the first four bytes available for dequeuing and their corresponding parity appear on the Data Output (Y) and Data Parity (PY) pins. The number of these bytes to be dequeued is set up on the Bytes Dequeued (BDQ) pins. If Dequeue Enable (\overline{DQEN}) is asserted, the LOW-to-HIGH transition of Dequeue Clock (DQCLK) updates the internal dequeue pointers, removing the dequeued bytes.

Figure 7 shows an example of one byte dequeued, followed by a dequeue of two bytes. The data to be dequeued next is least-significant-byte aligned on the output bus.

Synchronous Mode

Both synchronous and asynchronous operations are available for the byte queue. During synchronous operation, both QCLK and DQCLK must be asserted on the edge of a common clock within certain skew limits. The following signals can be used as valid status outputs for this mode: FULL, A-FULL, EMPTY, A-EMPTY, and CNT₀₋₆. Refer to the applications section for an example.

Asynchronous Mode

During asynchronous operation, QCLK and DQCLK clocks may be different. It is possible to execute queue and dequeue operations simultaneously if different locations are accessed. In this mode, CNT outputs are not guaranteed as valid and horizontal cascading is not possible. Refer to the applications section for an example.

Horizontal Cascading

In synchronous operation, four byte queues can be horizontally cascaded together. In this case, each of the four byte queues hold the same data and up to sixteen bytes may be dequeued in a single cycle, as shown in Table 2, and Figures 3 and 4. Each part has to be programmed with its position by the POS inputs, as shown in Table 4. In a normal operation, the internal dequeue pointer of each part is displaced according to the POS inputs. When \overline{RESET} or \overline{RXMIT} is asserted, the dequeue pointers are offset by the value programmed on the POS inputs.

Horizontal cascading is useful in instruction buffers designed for systems with large, variable instructions that can span many bytes.

APPLICATIONS

Using Am29338 as an Instruction-Prefetch Queue

Figure 8 shows the Am29338 used as an instruction-prefetch queue. Sequential 32-bit memory locations are fetched by the Instruction Fetch Unit (IFU) and are queued up in the byte queue. When the central processor needs the next instruction, it looks at the next four bytes from the byte queue. The central processor then determines the instruction length from the opcode and updates the dequeue pointer in the byte queue by setting up the instruction length on the BDQ lines and asserting DQCLK. When a jump occurs, the IFU flushes the

queue by asserting the $\overline{\text{RESET}}$ input and begins from the new address. For this application, the byte queue must be in synchronous mode.

Using the $\overline{\text{RXMIT}}$ input, the byte queue can resend the block data through dequeuing rather than having to requeue it. This is useful for locking the loops into the byte queue and allows the processor to run faster than if it had to refetch instructions from memory or cache. Figure 9 illustrates how a loop can execute directly out of the byte queue.

Using Am29338 as a Hardware Mailbox in Multiprocessing System

A mailbox is a communication device between loosely coupled processes in a multi-programming system. Messages from one process to another are queued in the mailbox on a first-in, first-out (FIFO) basis. In a multiprocessing system, hardware mailboxes are required. This can be implemented using the Am29338 as shown in Figure 10.

When a process wishes to send a message to the mailbox, it calls a special operating-system routine. This routine first

reads the status of the mailbox; if it is not FULL, the routine first writes the message to the mailbox and returns to the calling process. If the mailbox is FULL, the operating system blocks the calling process on a special queue and enables interrupts from the mailbox. When a slot becomes available in the mailbox, the sending processor is interrupted. The interrupt routine sends the message to the mailbox, disables interrupts from the mailbox, and unblocks the blocked process. On the receiving side, the EMPTY status of the mailbox must be available to the receiving processor in order to allow the receiving process to be blocked if the mailbox is empty. When a mailbox slot becomes filled, a blocked process must be awakened by interrupting the receiving processor.

The mailbox can be extended to operate in a heterogeneous multiprocessing system. In this type of system, processors with varying data-path widths and clock frequencies are interconnected. For example, a 32-bit main processor may control 8- to 16-bit coprocessors. The ability of the Am29338 to match data-path widths and to queue and dequeue asynchronously allows processors of different widths and clock rates to communicate.

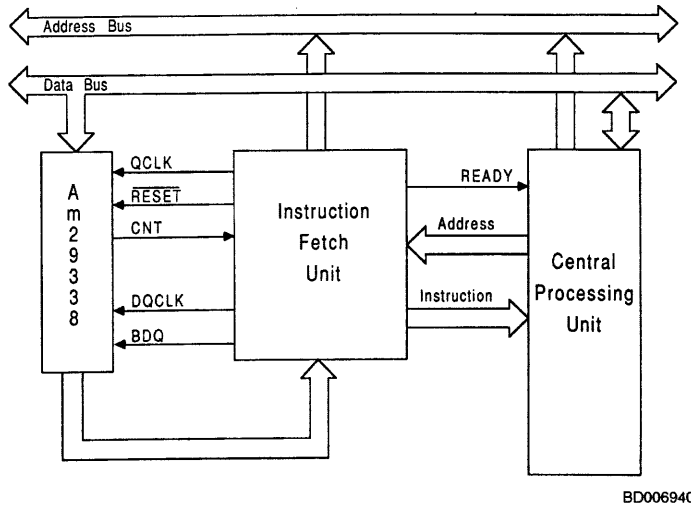
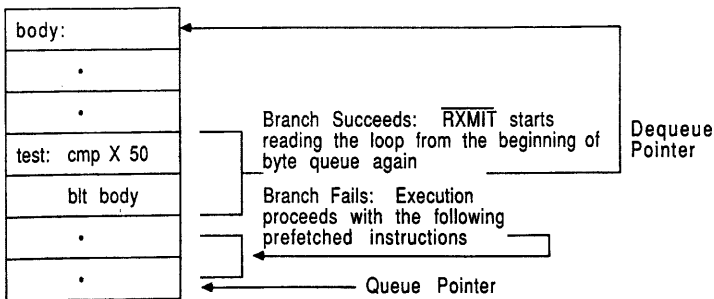
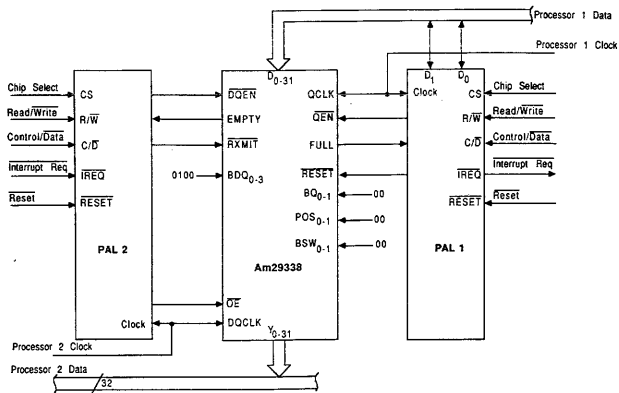


Figure 8. Instruction-Prefetch Queue



Note: This describes a block of macro instructions.

Figure 9. Loop Locking Using Am29338



BD006950

Figure 10. Implementation of a Hardware Mailbox

Suggestions for Power and Ground Pin Connections

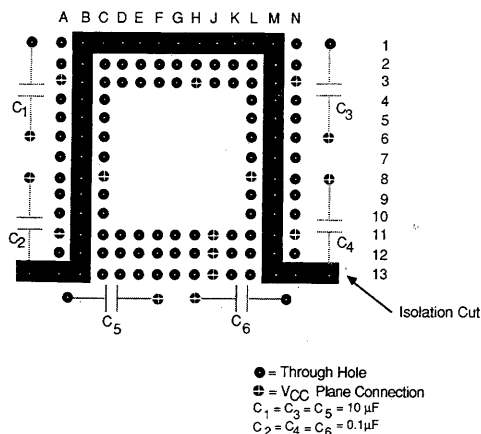
The Am29338 operates in an environment of fast signal rise times and substantial switching currents. Therefore, care must be exercised during circuit board design and layout, as with any high-performance component. The following is a suggested layout, but since systems vary widely in electrical configuration, an empirical evaluation of the intended layout is recommended.

The V_{CC} and GNDT pins, which carry output driver switching currents, tend to be electrically noisy. The V_{CC} and GNDE pins, which supply the ECL core of the device, tend to produce less noise, and the circuits they supply may be adversely affected by noise spikes on the V_{CC} plane. For this reason, it is best to provide isolation between the V_{CC} and V_{CC} pins, as well as independent decoupling for each. Isolating the GNDE and GNDT pins is not required.

Printed Circuit-Board Layout Suggestions

1. Use of a multi-layer PC board with separate power, ground, and signal planes is highly recommended.
2. All V_{CC} and V_{CC} pins should be connected to the V_{CC} plane. V_{CC} pins should be isolated from V_{CC} pins by means of a slot cut in the V_{CC} plane; see Figure 11. By physically separating the V_{CC} and V_{CC} pins, coupled noise will be reduced.
3. All GNDE and GNDT pins should be connected directly to the ground plane.
4. The V_{CC} pins should be decoupled to ground with a 0.1- μ F ceramic capacitor and a 10- μ F electrolytic capacitor, placed as closely to the Am29338 as is practical. V_{CC} pins should be decoupled to ground in a similar manner.

A suggested layout is shown in Figure 11.



CD010890

Figure 11. Suggested Printed Circuit-Board Layout

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Case Temperature	
with Power Applied	-55 to +125°C
Supply Voltage	
with Respect to Ground	-0.5 to +7.0 V
DC Voltage Applied to Outputs	
for HIGH State	-0.5 V to +V _{CC} Max.
DC Input Voltage	-0.5 V to +5.5 V

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Case Temperature (T _C)	0 to +85°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V
θ _{JA}	(under 200 lfm)

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating ranges unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions (Note 1)	Min.	Typ. (Note 2)	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH} I _{OH} = -3 mA	2.4			V
V _{OL}	Output LOW Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH} I _{OL} = 16 mA			0.4	V
V _{IH}	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs	2.0			V
V _{IL}	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs			0.8	V
V _I	Input Clamp Voltage	V _{CC} = Min. I _{IN} = -18 mA			-1.2	V
I _{IL}	Input LOW Current	V _{CC} = Max. V _{IN} = 0.5 V	QCLK, DQCLK		-1.0	mA
			Others		-0.5	
I _{IH}	Input HIGH Current	V _{CC} = Max. V _{IN} = 2.4 V			50	μA
I _I	Input HIGH Current	V _{CC} = Max. V _{IN} = 5.5 V			1.0	mA
I _{OZH} I _{OZL}	Off State (High Impedance) Output Current	V _{CC} = Max.	V _O = 2.4 V		50	μA
			V _O = 0.5 V		-50	
I _{SC}	Output Short-Circuit Current (Note 3)	V _{CC} = Max. to +0.5 V V _O = 0.5 V	-20		-80	mA
I _{CC}	Power Supply Current	V _{CC} = Max. All Inputs HIGH	T _C = 0 to +85°C	800	900	mA
			T _C = +85°C		800	

- Notes: 1. For conditions shown as Min. or Max., use the appropriate value specified under Operating Ranges for the applicable device type.
 2. Typical values are for V_{CC} = +25°C ambient and maximum loading.
 3. Not more than one output should be shorted at a time. Duration of the short-circuit test should not exceed one second.

SWITCHING CHARACTERISTICS over operating range (Note 1)

A. Combinational Propagation Delays

No.	From	To	Delay	Unit
1	D	PDERR	50	ns
2	PD	PDERR	50	ns
3	DQCLK ↑	A-EMPTY or A-FULL	44	ns
4	DQCLK ↑	CNT	46	ns
5	DQCLK ↑	EMPTY or FULL	44	ns
6	DQCLK ↑	PYERR	60	ns
7	DQCLK ↑	Y	52	ns
8	\overline{OE}	PYERR	25	ns
9	\overline{OE}	Y	25	ns
10	QCLK ↑	A-EMPTY or EMPTY	44	ns
11	QCLK ↑	CNT	46	ns
12	QCLK ↑	A-FULL or FULL	44	ns
13	RESET ↓	A-FULL or FULL	44	ns
14	RESET ↓	CNT	46	ns
15	RESET ↓	EMPTY or	44	ns
16	RESET ↓	PYERR	60	ns
17	RESET ↓	Y	52	ns
18	RXMIT ↓	A-FULL or FULL	44	ns
19	RXMIT ↓	CNT	46	ns
20	RXMIT ↓	A-EMPTY or EMPTY	44	ns
21	RXMIT ↓	PYERR	60	ns
22	RXMIT ↓	Y	52	ns

B. Setup and Hold Times

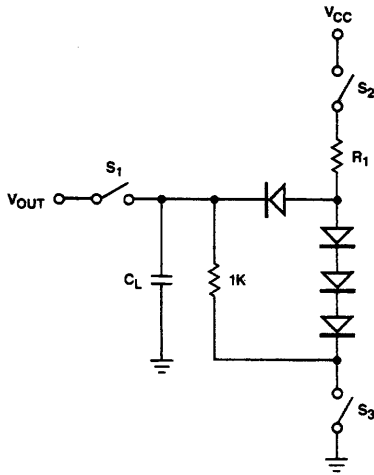
No.	Parameter	For	With Respect To	Delay	Unit
23	Bytes Dequeued Setup	BDQ	DQCLK ↑	20	ns
24	Bytes Dequeued Hold	BDQ	DQCLK ↑	0	ns
25	Bytes Queued Setup	BQ	QCLK ↓	12	ns
26	Bytes Queued Hold	BQ	QCLK ↑		ns
27	Byte Swap Setup	BSW	QCLK ↑	20	ns
28	Byte Swap Hold	BSW	QCLK ↓		ns
29	Data Setup	D	QCLK ↑	8	ns
30	Data Hold	D	QCLK ↑		ns
31	Data Parity Setup	PD	QCLK ↑	8	ns
32	Data Parity Hold	PD	QCLK ↑		ns
33	Dequeue Enable Setup	\overline{DQEN}	DQCLK ↑	8	ns
34	Dequeue Enable Hold	\overline{DQEN}	DQCLK ↑	0	ns
35	Queue Enable Setup	\overline{QEN}	QCLK ↓		ns
36	Queue Enable Hold	\overline{QEN}	QCLK ↑		ns

C. Minimum Clock Requirements

No.	Input	Description	Delay	Unit
37	DQCLK	Dequeue Min. Pulse Width LOW	10	ns
38		Dequeue Min. Pulse Width HIGH	10	
39		Dequeue Min. Cycle Time	80	
40	QCLK	Queue Min. Pulse Width LOW	10	ns
41		Queue Min. Pulse Width HIGH	10	
42		Queue Min. Cycle Time	80	

Notes: 1. Case temperature (T_C) = 0 to +85°C, supply voltage (V_{CC}) = 5 V ±5%. It is the responsibility of the user to maintain a case temperature of +85°C or less. AMD recommends an air velocity of at least 200 linear feet per minute over the heatsink.

SWITCHING TEST CIRCUITS

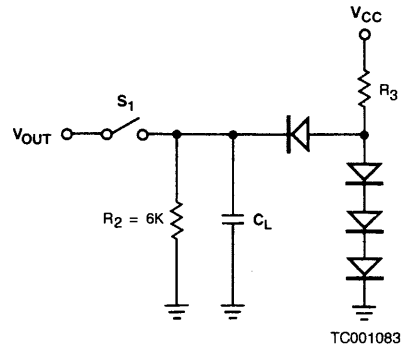


TC001102

$$R_1 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OL} + \frac{V_{OL}}{1K}}$$

A. Three-State Outputs

- Notes:
1. $C_L = 50$ pF includes scope probe, wiring and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 S_1 and S_2 are closed while S_3 is open for t_{pZL} test.
 4. $C_L = 5.0$ pF for output disable tests.



TC001083

$$R_2 = \frac{2.4 \text{ V}}{I_{OH}}$$

$$R_1 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OL} + \frac{V_{OL}}{R_2}}$$

B. Normal Outputs

Test Philosophy and Methods

The following points give the general philosophy that we apply to tests that must be properly engineered if they are to be implemented in an automatic environment. The specifics of what philosophies applied to which test are shown in the data sheet.

1. Ensure the part is adequately decoupled at the test head. Large changes in V_{CC} current as the device switches may cause erroneous function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may start to oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an output transition, ground current may change by as much as 400 mA in 5-8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily. Current level may vary from product to product.
4. Use extreme care in defining input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins and they may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \geq 3.0$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. Capacitive Loading for AC Testing

Automatic testers and their associated hardware have stray capacitance that varies from one type of tester to another but is generally around 50 pF. This, of course, makes it impossible to make direct measurements of parameters that call for a smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays," which measure the propagation delays into the high-impedance state and are usually specified at a load capacitance of 5.0 pF. In these cases, the test is performed at the higher load capacitance (typically 50 pF) and engineering correlations based on data taken with a bench set up are used to predict the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impossible to make measurements at both capacitances even though they may both be greater than the stray capacitance. In

these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is predicted from engineering correlations based on data taken with a bench setup and the knowledge that certain DC measurements (I_{OH} , I_{OL} , for example) have already been taken and are within spec. In some cases, special DC tests are performed in order to facilitate this correlation.

7. Threshold Testing

The noise associated with automatic testing (due to the long, inductive cables), and the high gain of the tested device when in the vicinity of the actual device threshold, frequently give rise to oscillations when testing high-speed circuits. These oscillations are not indicative of a reject device, but instead, of an overtaxed test system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" high and low levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels rather than at V_{IL} Max. and V_{IH} Min.

8. AC Testing

Occasionally parameters are specified that cannot be measured directly on automatic testers because of tester limitations. Data input hold times often fall into this category. In these cases, the parameter in question is guaranteed by correlating these tests with other AC tests that have been performed. These correlations are arrived at by the cognizant engineer using data from precise bench measurements in conjunction with the knowledge that certain DC parameters have already been measured and are within spec.







In some cases, certain AC tests are redundant since they can be shown to be predicted by other tests that have already been performed. In these cases, the redundant tests are not performed.

9. Output Short-Circuit Current Testing

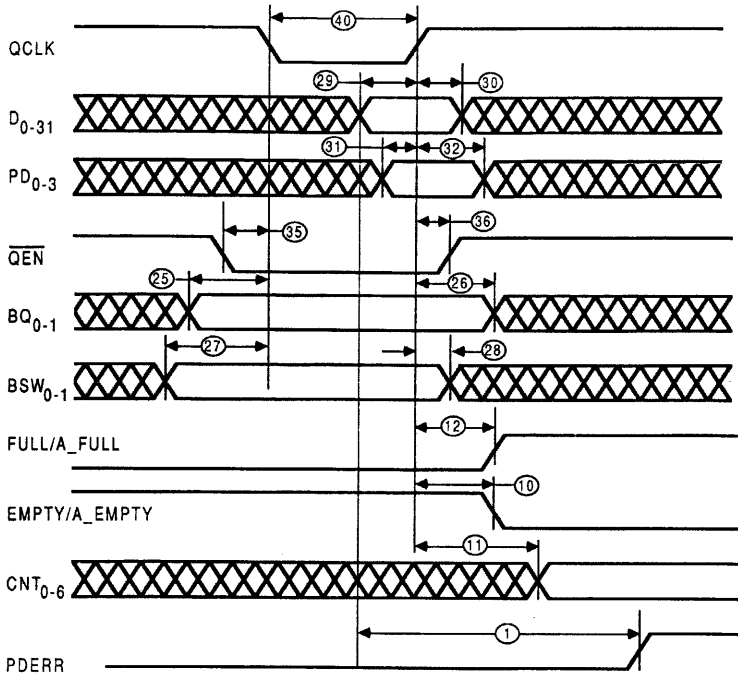
When performing I_{OS} tests on devices containing RAM or registers, great care must be taken that undershoot caused by grounding the high-state output does not trigger parasitic elements which in turn cause the device to change state. In order to avoid this effect, it is common to make the measurement at a voltage (V_{output}) that is slightly above ground. The V_{CC} is raised by the same amount so that the result (as confirmed by Ohm's law and precise bench testing) is identical to the $V_{OUT} = 0$, $V_{CC} = \text{Max. case}$.

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
 	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

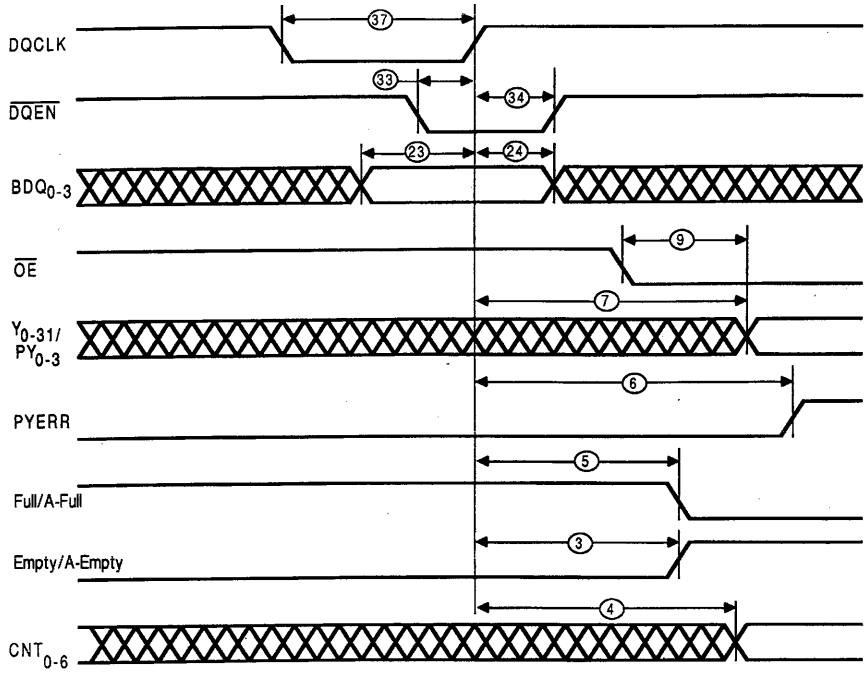
KS000010



WF023460

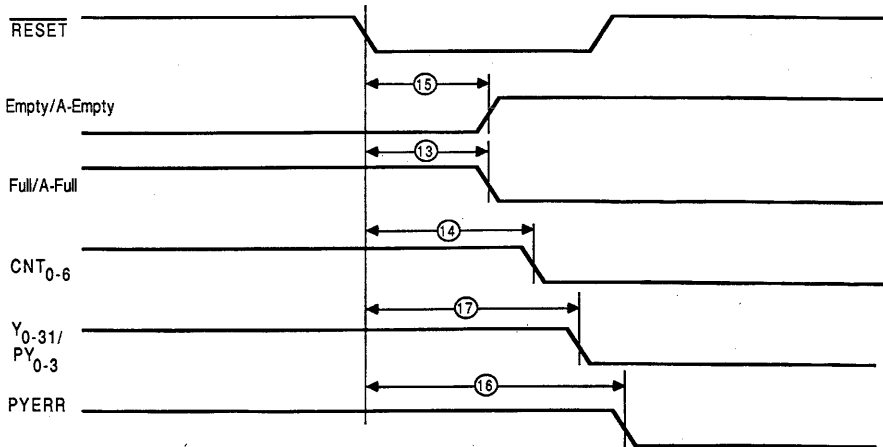
Queue Cycle

SWITCHING WAVEFORMS (Cont'd.)



WF023471

Dequeue Cycle

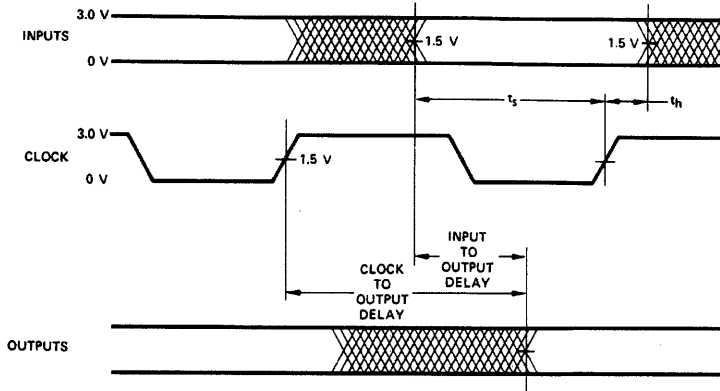


WF023481

RESET Timing Diagram

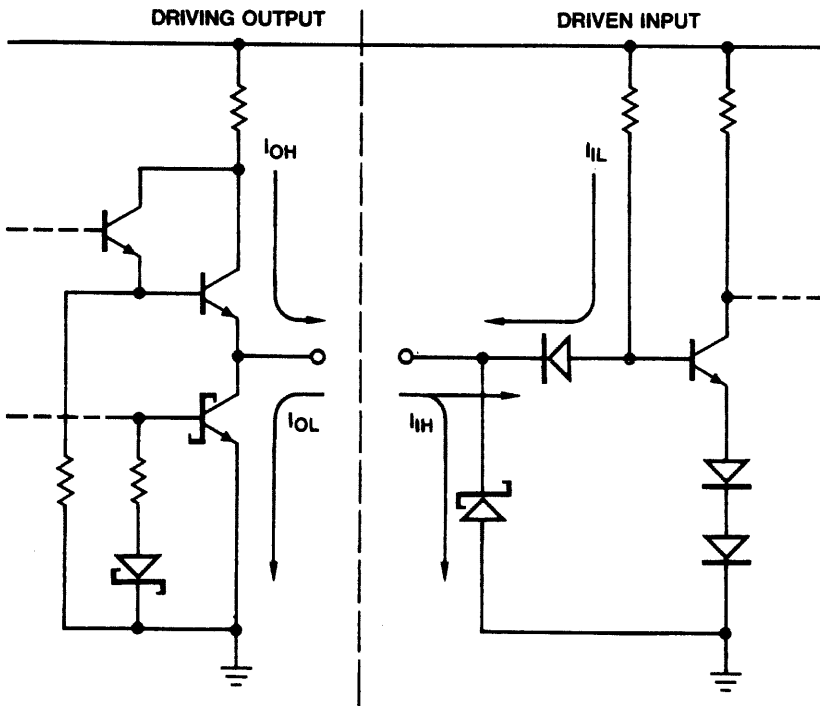
- Notes: 1. Minimum time $\overline{\text{RESET}}$ must be asserted.
 2. This timing diagram is applicable to $\overline{\text{RXMIT}}$.

SWITCHING WAVEFORMS (Cont'd.)



WFR02990

INPUT/OUTPUT CIRCUIT DIAGRAM



ICR00480

CHAPTER 4

Arithmetic Processors

Am29C323 CMOS 32-Bit Parallel Multiplier	4-1
Am29325 32-Bit Floating-Point Processor	4-24
Am29C325 CMOS 32-Bit Floating-Point Processor	4-78
Am29C327 CMOS Double-Precision Floating-Point Processor	4-133

Am29C323

CMOS 32-Bit Parallel Multiplier



Am29C323

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- **32-Bit Three-Bus Architecture**
 - The device has two 32-bit input ports and one 32-bit output port with clocked multiply time of 100 ns
- **Speed Selects**
 - 80- and 55-ns speed-select parts
- **Single Clock with Register Enables**
 - The Am29C323 is controlled by one clock with individual register enables
- **Supports Multiprecision Multiplication**
 - The device has dual 32-bit registers on each data input port to perform multiprecision multiplication
- **Registers can be made transparent**
 - Input and output registers can be made transparent independently to eliminate unwanted pipeline delay
- **Supports Two's Complement, Unsigned or Mixed Numbers**
- **Data Integrity Through Master-Slave Mode and Parity Check/Generate**
 - Parity check/generate catches inter-device connection errors and master/slave mode provides complete function check

GENERAL DESCRIPTION

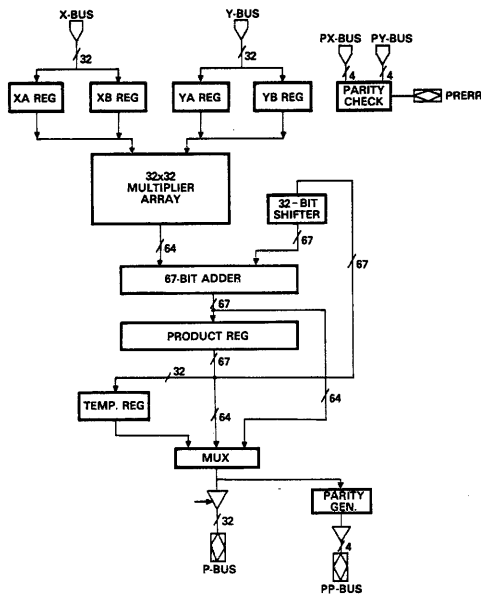
The Am29C323 is a high-speed 32 x 32-Bit CMOS Parallel Multiplier with 67-Bit Accumulator. The part is designed to maximize system level performance by providing a 32-bit three bus architecture and a single clock with register enables.

The Am29C323 further enhances system throughput by providing individual register feedthrough controls, byte parity checking on both input ports and generation on the output port, and dual input registers on each data input bus to support multiprecision multiplication. The Am29C323 can manage a wide variety of data types, including two's

complement, unsigned, or mixed mode input formats. A 64 x 64-bit multiplication can be performed in seven clock cycles, including input and output. Additional features provided are a format adjust control allowing for standard output or left shifted output suitable for fractional two's complement arithmetic, rounding, and master/slave operation.

The Am29C323 is designed in low-power, high-speed CMOS with TTL-compatible I/O. The device is housed in a 169-lead pin-grid-array package.

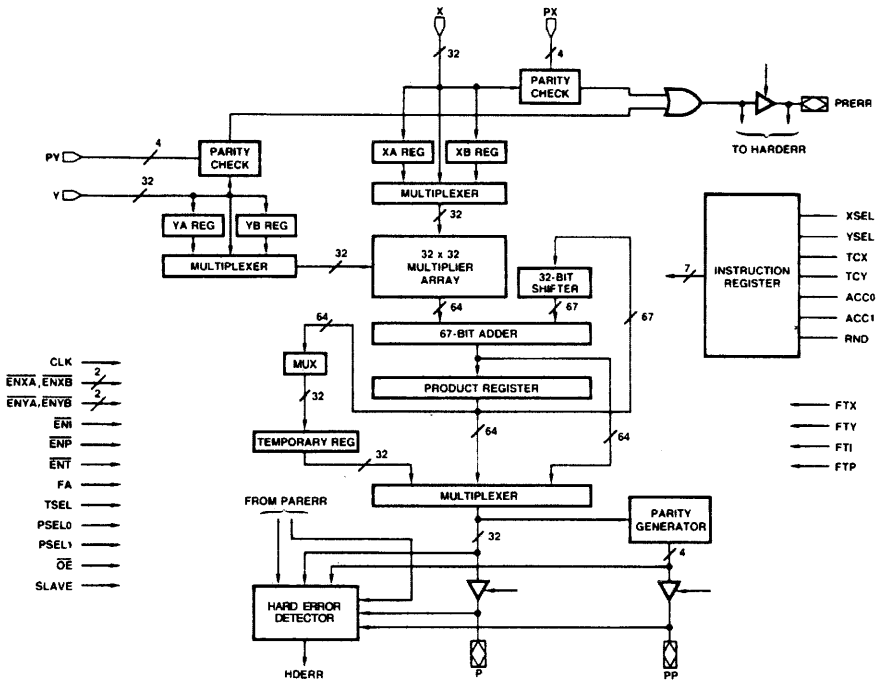
SIMPLIFIED BLOCK DIAGRAM



RELATED AMD PRODUCTS

Part No.	Description
Am29C01	CMOS 4-Bit Microprocessor Slice
Am29C10A	CMOS 12-Bit Sequencer
Am29C101	CMOS 16-Bit Microprocessor
Am29112	8-Bit Cascadable Microprogram Sequencer
Am29114	Real-Time Interrupt Controller
Am29C116	CMOS 16-Bit Microcontroller
Am29325	32-Bit Floating Point Processor
Am29C325	CMOS 32-Bit Floating Point Processor
Am29331	16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29332	32-Bit Extended Function ALU
Am29C332	CMOS 32-Bit Extended Function ALU
Am29334	64 x 18 Four-Port Dual Access Register File
Am29C334	CMOS 64 x 18 Four-Port Dual Access Register File
Am29337	16-Bit Bounds Checker
Am29338	32-Bit Byte Queue
Am29C516	CMOS 16 x 16 Multiplier
Am29C517	CMOS 16 x 16 Multiplier with Separate I/O

DETAILED BLOCK DIAGRAM



BD003049

CONNECTION DIAGRAM
169-Lead PGA
Bottom View

	A	B	C	D	E	F	G	H	J	K	L	M	N	P	R	T	U	
1	PY ₃	Y ₃₁	Y ₃₀	Y ₂₆	Y ₂₄	Y ₂₃	Y ₂₀	Y ₁₉	Y ₁₅	Y ₁₀	Y ₁₁	Y ₇	Y ₅	Y ₃	GND	ENYA	PX ₃	
2	GND	NC	Y ₂₉	Y ₂₇	PY ₂	Y ₂₁	Y ₁₈	Y ₁₆	PY ₁	Y ₁₂	Y ₀₉	PY ₀	Y ₄	Y ₂	Y ₀	ENVB	X ₃₁	
3	NC	PRERR	NC	Y ₂₈	Y ₂₅	Y ₂₂	V _{CC}	Y ₁₇	Y ₁₄	Y ₁₃	GND	Y ₈	Y ₆	Y ₁	FTY	X ₃₀	GND	
4	V _{CC}	PP ₃	NC	*												X ₂₉	X ₂₈	X ₂₇
5	P ₃₀	P ₃₁	P ₂₉													X ₂₆	X ₂₄	X ₂₅
6	GND	P ₂₈	P ₂₇													PX ₂	X ₂₃	X ₂₂
7	P ₂₅	P ₂₆	GND													V _{CC}	X ₁₉	X ₂₁
8	V _{CC}	P ₂₄	PP ₂													X ₁₈	X ₁₇	X ₂₀
9	NC	NC	P ₂₃													X ₁₅	X ₁₆	PX ₁
10	GND	P ₂₂	P ₂₁													X ₁₄	X ₁₃	X ₁₁
11	P ₁₉	P ₂₀	V _{CC}													GND	X ₁₀	X ₁₂
12	P ₁₆	P ₁₈	P ₁₇													X ₉	X ₈	PX ₀
13	V _{CC}	HDERR	FTP													X ₇	X ₅	X ₆
14	NC	ENP	NC													X ₂	X ₃	X ₄
15	ENT	OE	SLAVE	P ₁₅	P ₁₁	P ₁₂	GND	P ₇	P ₆	P ₃	V _{CC}	P ₁	FTI	TCY	FTX	X ₁	X ₀	
16	PSEL1	FA	PP ₁	P ₁₄	P ₁₃	P ₉	GND	PP ₀	P ₅	P ₄	V _{CC}	ENI	CLK	ACC1	TCX	ENXB	ENXA	
17	PSEL0	TSEL	GND	NC	V _{CC}	P ₁₀	GND	P ₈	GND	P ₂	V _{CC}	P ₀	V _{CC}	RND	ACCO	YSEL	XSEL	

CD011030

*Pinout observed from pin side of package.

**Pin 169 for reference only.

PIN DESIGNATIONS

(Sorted by Pin Number)

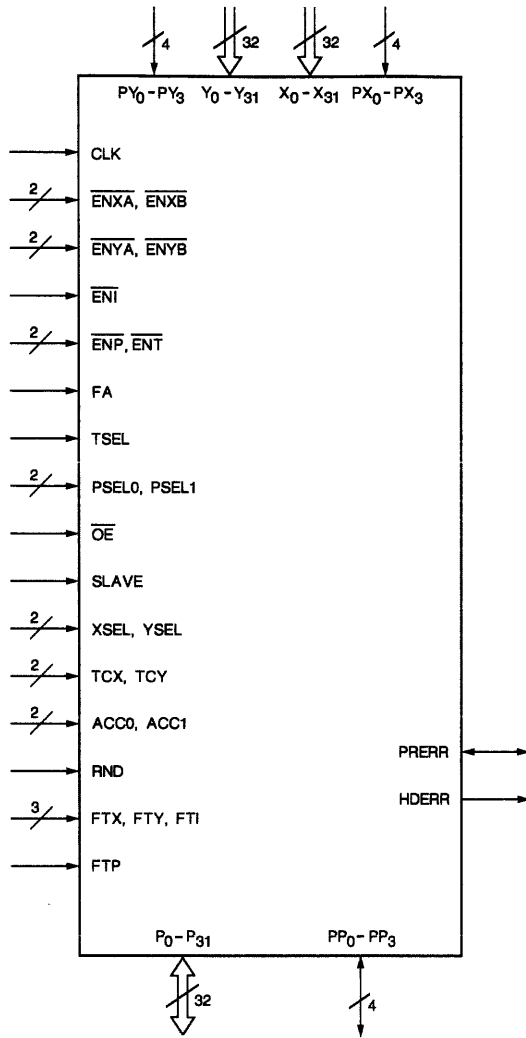
PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME
1	A1	PY ₃	75	C9	P ₂₃	55	J15	P ₆	117	R10	X ₁₄
168	A2	GND	72	C10	P ₂₁	51	J16	P ₅	116	R11	GND
83	A3	NC	74	C11	V _{CC}	135	J17	GND	36	R12	X ₉
81	A4	V _{CC}	153	C12	P ₁₇	14	K1	Y ₁₀	121	R13	X ₇
80	A5	P ₃₀	151	C13	FTP	13	K2	Y ₁₂	40	R14	X ₂
79	A6	GND	66	C14	NC	96	K3	Y ₁₃	125	R15	FTX
160	A7	P ₂₅	146	C15	SLAVE	50	K15	P ₃	128	R16	TCX
77	A8	V _{CC}	145	C16	PP ₁	134	K16	P ₄	45	R17	ACCO
157	A9	NC	61	C17	GND	133	K17	P ₂	105	T1	ENYA
71	A10	GND	4	D1	Y ₂₆	97	L1	Y ₁₁	21	T2	ENYB
154	A11	P ₁₉	87	D2	Y ₂₇	98	L2	Y ₉	107	T3	X ₃₀
69	A12	P ₁₆	3	D3	Y ₂₈	95	L3	GND	108	T4	X ₂₈
68	A13	V _{CC}	62	D15	P ₁₅	53	L15	V _{CC}	110	T5	X ₂₄
67	A14	NC	144	D16	P ₁₄	53	L16	V _{CC}	111	T6	X ₂₃
65	A15	ENT	60	D17	NC	53	L17	V _{CC}	113	T7	X ₁₉
148	A16	PSEL1	5	E1	Y ₂₄	16	M1	Y ₇	114	T8	X ₁₇
64	A17	PSEL0	89	E2	PY ₂	99	M2	PY ₀	31	T9	X ₁₆
85	B1	Y ₃₁	88	E3	Y ₂₅	15	M3	Y ₈	34	T10	X ₁₃
84	B2	NC	142	E15	P ₁₁	132	M15	P ₁	119	T11	X ₁₀
166	B3	PRERR	143	E16	P ₁₃	47	M16	ENI	120	T12	X ₈
165	B4	PP ₃	57	E17	V _{CC}	48	M17	P ₀	122	T13	X ₅
164	B5	P ₃₁	6	F1	Y ₂₃	17	N1	Y ₅	123	T14	X ₃
162	B6	P ₂₈	7	F2	Y ₂₁	101	N2	Y ₄	124	T15	X ₁
161	B7	P ₂₆	90	F3	Y ₂₂	100	N3	Y ₆	42	T16	ENXB
76	B8	P ₂₄	59	F15	P ₁₂	130	N15	FTI	127	T17	YSEL
73	B9	NC	141	F16	P ₉	131	N16	CLK	22	U1	PX ₃
156	B10	P ₂₂	58	F17	P ₁₀	49	N17	V _{CC}	106	U2	X ₃₁
155	B11	P ₂₀	91	G1	Y ₂₀	18	P1	Y ₃	23	U3	GND
70	B12	P ₁₈	92	G2	Y ₁₈	102	P2	Y ₂	25	U4	X ₂₇
152	B13	HDERR	11	G3	V _{CC}	19	P3	Y ₁	26	U5	X ₂₅
150	B14	ENP	137	G15	GND	44	P15	TCY	28	U6	X ₂₂
149	B15	OE	137	G16	GND	129	P16	ACC1	112	U7	X ₂₁
63	B16	FA	137	G17	GND	46	P17	RND	29	U8	X ₂₀
147	B17	TSEL	8	H1	Y ₁₉	20	R1	GND	115	U9	PX ₁
2	C1	Y ₃₀	93	H2	Y ₁₆	103	R2	Y ₀	35	U10	X ₁₁
86	C2	Y ₂₉	9	H3	Y ₁₇	104	R3	FTY	118	U11	X ₁₂
167	C3	NC	139	H15	P ₇	24	R4	X ₂₉	37	U12	PX ₀
82	C4	NC	56	H16	PP ₀	109	R5	X ₂₆	38	U13	X ₆
163	C5	P ₂₉	140	H17	P ₈	27	R6	PX ₂	39	U14	X ₄
78	C6	P ₂₇	94	J1	Y ₁₅	32	R7	V _{CC}	41	U15	X ₀
158	C7	GND	10	J2	PY ₁	30	R8	X ₁₈	126	U16	ENXA
159	C8	PP ₂	12	J3	Y ₁₄	33	R9	X ₁₅	43	U17	XSEL

PIN DESIGNATIONS

(Sorted by Pin Name)

PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME	PAD NO.	PIN NO.	PIN NAME
45	R17	ACC0	50	K15	P ₃	89	E2	PY ₂	110	T5	X ₂₄
129	P16	ACC1	134	K16	P ₄	1	A1	PY ₃	26	U5	X ₂₅
131	N16	CLK	51	J16	P ₅	46	P17	RND	109	R5	X ₂₆
47	M16	ENI	55	J15	P ₆	146	C15	SLAVE	25	U4	X ₂₇
150	B14	ENP	139	H15	P ₇	128	R16	TCX	108	T4	X ₂₈
65	A15	ENT	140	H17	P ₈	44	P15	TCY	24	R4	X ₂₉
126	U16	ENXA	141	F16	P ₉	147	B17	TSEL	107	T3	X ₃₀
42	T16	ENXB	58	F17	P ₁₀	68	A13	VCC	106	U2	X ₃₁
105	T1	ENYA	142	E15	P ₁₁	81	A4	VCC	43	U17	XSEL
21	T2	ENYB	59	F15	P ₁₂	77	A8	VCC	103	R2	Y ₀
63	B16	FA	143	E16	P ₁₃	74	C11	VCC	19	P3	Y ₁
130	N15	FTI	144	D16	P ₁₄	57	E17	VCC	102	P2	Y ₂
151	C13	FTP	62	D15	P ₁₅	11	G3	VCC	18	P1	Y ₃
125	R15	FTX	69	A12	P ₁₆	53	L15	VCC	101	N2	Y ₄
104	R3	FTY	153	C12	P ₁₇	53	L16	VCC	17	N1	Y ₅
71	A10	GND	70	B12	P ₁₈	53	L17	VCC	100	N3	Y ₆
168	A2	GND	154	A11	P ₁₉	49	N17	VCC	16	M1	Y ₇
79	A6	GND	155	B11	P ₂₀	32	R7	VCC	15	M3	Y ₈
61	C17	GND	72	C10	P ₂₁	41	U15	X ₀	98	L2	Y ₉
158	C7	GND	156	B10	P ₂₂	124	T15	X ₁	14	K1	Y ₁₀
137	G15	GND	75	C9	P ₂₃	40	R14	X ₂	97	L1	Y ₁₁
137	G16	GND	76	B8	P ₂₄	123	T14	X ₃	13	K2	Y ₁₂
137	G17	GND	160	A7	P ₂₅	39	U14	X ₄	96	K3	Y ₁₃
135	J17	GND	161	B7	P ₂₆	122	T13	X ₅	12	J3	Y ₁₄
95	L3	GND	78	C6	P ₂₇	38	U13	X ₆	94	J1	Y ₁₅
20	R1	GND	162	B6	P ₂₈	121	R13	X ₇	93	H2	Y ₁₆
116	R11	GND	163	C5	P ₂₉	120	T12	X ₈	9	H3	Y ₁₇
23	U3	GND	80	A5	P ₃₀	36	R12	X ₉	92	G2	Y ₁₈
152	B13	HDERR	164	B5	P ₃₁	119	T11	X ₁₀	8	H1	Y ₁₉
157	A9	NC	166	B3	PRERR	35	U10	X ₁₁	91	G1	Y ₂₀
60	D17	NC	56	H16	PP ₀	118	U11	X ₁₂	7	F2	Y ₂₁
73	B9	NC	145	C16	PP ₁	34	T10	X ₁₃	90	F3	Y ₂₂
82	C4	NC	159	C8	PP ₂	117	R10	X ₁₄	6	F1	Y ₂₃
83	A3	NC	165	B4	PP ₃	33	R9	X ₁₅	5	E1	Y ₂₄
84	B2	NC	64	A17	PSEL0	31	T9	X ₁₆	88	E3	Y ₂₅
66	C14	NC	148	A16	PSEL1	114	T8	X ₁₇	4	D1	Y ₂₆
167	C3	NC	37	U12	PX ₀	30	R8	X ₁₈	87	D2	Y ₂₇
67	A14	NC	115	U9	PX ₁	113	T7	X ₁₉	3	D3	Y ₂₈
149	B15	OE	27	R6	PX ₂	29	U8	X ₂₀	86	C2	Y ₂₉
48	M17	P ₀	22	U1	PX ₃	112	U7	X ₂₁	2	C1	Y ₃₀
132	M15	P ₁	99	M2	PY ₀	28	U6	X ₂₂	85	B1	Y ₃₁
133	K17	P ₂	10	J2	PY ₁	111	T6	X ₂₃	127	T17	YSEL

LOGIC SYMBOL



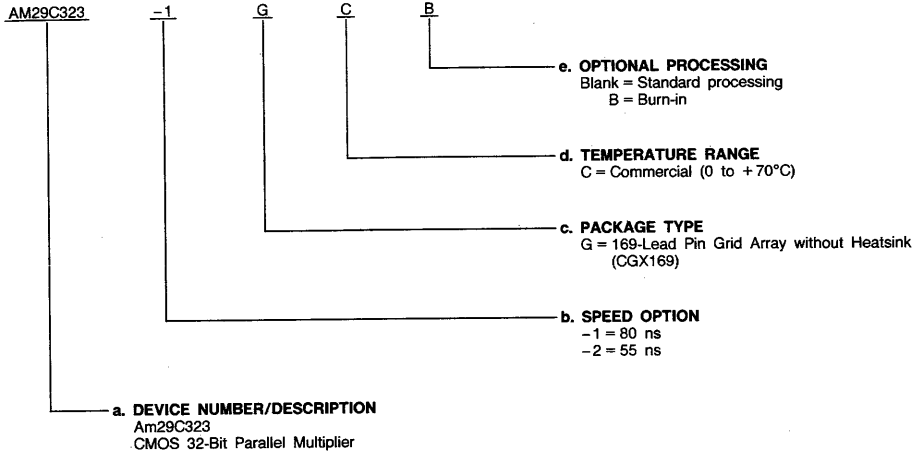
LS002803

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. Device Number
- b. Speed Option (if applicable)
- c. Package Type
- d. Temperature Range
- e. Optional Processing



Valid Combinations

Valid Combinations	
AM29C323	GC, GCB
AM29C323-1	
AM29C323-2	

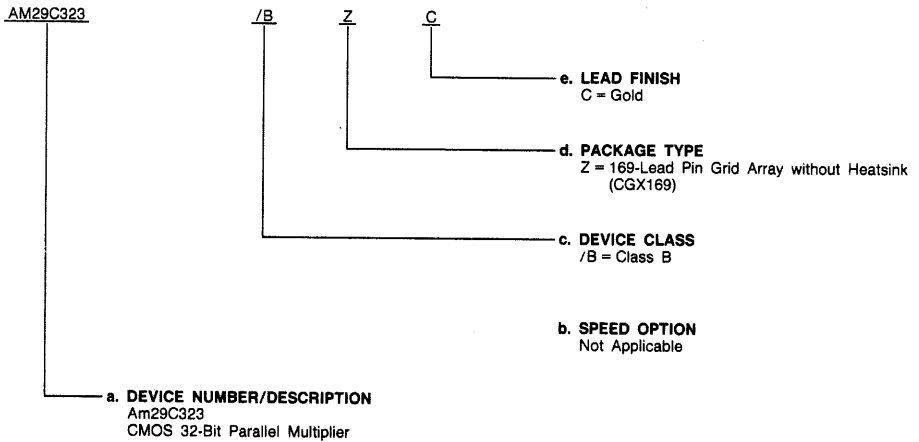
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

ORDERING INFORMATION

APL Products

AMD products for Aerospace and Defense applications are available in several packages and operating ranges. APL (Approved Products List) products are fully compliant with MIL-STD-883C requirements. The order number (Valid Combination) for APL products is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Device Class**
- d. **Package Type**
- e. **Lead Finish**



Valid Combinations	
AM29C323	/BZC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations or to check for newly released valid combinations.

Group A Tests

Group A tests consist of Subgroups
1, 2, 3, 7, 8, 9, 10, 11.

PIN DESCRIPTION

ACC0, ACC1 Accumulator Control (Input)

Accumulator control lines used to determine accumulator function; PASS, ACCUMULATE, and SHIFT and ACCUMULATE.

CLK Clock (Input)

Clock input for all registers.

ENI Instruction Register Enable (Input; Active LOW)

Register enable for instruction register I.

ENP Accumulator Register Enable (Input; Active LOW)

Register enable for product register P.

ENT Temporary Register Enable (Input; Active LOW)

Register enable for temporary register T.

ENXA, ENXB Multiplicand Register Enable (Input; Active LOW)

Register enables for multiplicand data input registers XA and XB.

ENYA, ENYB Multiplier Register Enable (Input; Active LOW)

Register enables for multiplier data input registers YA and YB.

FA Format Adjust (Input)

Format adjust selects either a full 64-bit product (HIGH) or a left shifted 63-bit product suitable for fractional two's complement arithmetic (LOW).

FTP Feedthrough Control (Input; Active HIGH)

Feedthrough control for product register.

FTX, FTY, FTI Feedthrough Control (Input; Active HIGH)

Feedthrough control lines for X, Y, and I registers.

HDERR Hard Error Flag (Output)

Used when two Am29C323s are configured as master and slave to indicate hardware errors.

OE Output Enable Control (Input; Active LOW)

Used to enable (LOW) or disable (HIGH) the P output port.

P0 - P31 Product Output (Input/Output; Three State)

Product output for P port.

PRRER Parity Error Flag (Input/Output; Three State)

Indicates a parity error on the input buses.

PP0 - PP3 Byte Parity (Input/Output; Three State)

Byte parity generated on P output port (even parity).

PSEL0, PSEL1 Product Control (Input)

Used to select desired output including disabling P and PP output ports.

PX0 - PX3 Byte Parity (Input)

Byte parity inputs on X input port (even parity).

PY0 - PY3 Byte Parity (Input)

Byte parity inputs on Y input port (even parity).

RND Round Control (Input; Active HIGH)

Round control for rounding the most significant product.

SLAVE Master/Slave Control (Input)

Used to determine mode of operation.

TCX, TCY Mode Control (Input)

Mode control inputs for each input data word; LOW for unsigned data and HIGH for two's complement format.

TSEL Select Control (Input)

Used to route the most significant product register (HIGH) or the least significant product register (LOW) into the temporary register.

X0 - X31 Multiplicand Data (Input)

Multiplicand data input for X port.

XSEL X Register Select (Input)

Control line used to route the contents of either the XA register (HIGH) or XB register (LOW) into the multiplier array.

Y0 - Y31 Multiplier Data (Input)

Multiplier data input for Y port.

YSEL Y Register Select (Input)

Control line used to route the contents of either the YA register (HIGH) or YB register (LOW) into the multiplier array.

FUNCTIONAL DESCRIPTION

Architecture

The Am29C323 comprises a high speed 32 by 32-bit multiplier array, a 67-bit accumulator, and a 32-bit data path.

Multiplier Array

The multiplier is a 32 by 32-bit array that produces a 64-bit product. This product is then fed to the accumulator section.

Accumulator

The accumulator is 67 bits wide. It performs accumulation for sum of product operations and multiprecision multiplication operations. The accumulator can perform three operations: store product without accumulation, accumulate product, and shift accumulator value and accumulate with product.

The shift and accumulate shifts the value in the product register 32 bits to the right (effectively moving the most significant 32 bits to the least significant 32 bits) and sign extends to a full 64 bits. This shifted value is then accumulated with the output of the multiplier array.

The 67-bit width is necessary to contain overflows in internal accumulations. These overflows are maintained and used when the product register is right shifted in the multiprecision multiplies. The lower 64 bits contain the 64-bit output while the upper 3 bits contain the overflow.

Data Path

The 32-bit data path consists of X and Y input buses; the P output bus; data registers XA, XB, YA, YB, and the product accumulator; two multiplier input multiplexers; byte parity input checkers; byte parity output generators; and master/slave comparators. Input operands enter the device through the two 32-bit input buses, X₀-X₃₁ and Y₀-Y₃₁. These operands may then be stored in one of the two registers for each bus (XA or XB for X, YA or YB for Y) or they may be fed directly through to the multiplier array. Input parity checking is performed as soon as the operands are put on the input buses. The signals used for output parity generation are taken from the input side of the output translator. In case of parity error, PRRER is enabled HIGH.

Operational Modes

The Am29C323 can perform signed, unsigned, or mixed mode multiplication. These different numerical representations are controlled by TCX and TCY. A HIGH input on one of these lines indicates to the device that the respective input should be treated as a two's complement number; a LOW, an unsigned number. The output format is unsigned when both inputs are unsigned. The output format is two's complement when either or both inputs are two's complement.

Slave Mode

Each output has an associated comparator which compares the signal on the output pin with the signal provided to the output driver. If any of these outputs do not agree, the HDERR is asserted. When not in slave mode, this enables the multiplier to check for contention and bus shorts. However, when in slave mode, one multiplier can be used to detect faults in both internal functions and interconnections of the other multiplier. This is accomplished through the master/slave configuration, where the two multipliers operate in parallel. One multiplier is the master and operates normally; the other operates in slave mode.

In slave mode all outputs are turned into inputs from the master, except for the HDERR signal. Since the slave is operated in parallel with the master, it can compare the results it generates to those of the master and signal an error if they differ.

Command Description and Formats

The accumulator is controlled by ACC0 and ACC1. These lines are used to select any of the three operations that the accumulator can perform. This instruction set is described in Table 1.

The temporary output register is controlled by TSEL and FA. These lines are used to select any of the four different sets of data that can be stored in the temporary register. This instruction set is described in Table 2.

The output multiplexer is controlled by PSEL0, PSEL1, and FA. These lines are used to select any of the five different sets of data that can be output through the P port. PSEL0 and PSEL1 can also be used to disable the outputs. (This instruction is independent of OE.) This instruction set is described in Table 3.

Format Adjust (FA) is used to select either a full 64-bit product or a left-shifted 63-bit product suitable for fractional two's complement arithmetic. This shifting increases the precision of the upper half of the product word by eliminating the redundant sign bit. Output Data Formats show the effect of FA.

Round (RND) is used to round the upper 32 bits of the 64-bit product. If only the upper 32 bits of the product are being used, then the lower 32 bits are truncated when rounding is not used (RND = 0). If rounding is used (RND = 1), then a "1" is added to the most significant of the lower 32 bits. This

results in a smaller possible error. This should only be used when the lower 32 bits are to be truncated.

User Visible Register Descriptions

The Am29C323 contains seven different register sets, each with its own clock enable. Two 32-bit registers are attached to each of the input data buses. These registers are differentiated by the suffix A or B. For example, the X bus has registers XA and XB. The 67-bit accumulator register can be used as a regular product register when the part is used as a multiplier only or as the register part of the accumulator section. The 32-bit temporary output register is included to aid in the pipelining of multiprecision operations. An instruction register is also provided.

All of these registers can be made transparent with the exception of the accumulator register and the temporary register. The product from the multiplier can be fed directly to the output by using the FTP control line.

TABLE 1. ACCUMULATOR OPERATION INSTRUCTIONS

ACC1	ACC0	Accumulator Operation
0	0	PASS
0	1	ACCUMULATE
1	0	INVALID
1	1	SHIFT AND ACCUMULATE

TABLE 2. INPUT SELECT INSTRUCTIONS FOR TEMPORARY (T) REGISTER

TSEL	FA	Temp Reg Input
0	0	P_{i-1}
0	1	P_i
1	0	P_{i+31}
1	1	P_{i+32}

TABLE 3. OUTPUT SELECT INSTRUCTIONS FOR PRODUCT (P) PORT

PSEL1	PSEL0	FA	P Port Output
0	0	X	TEMP REGISTER
0	1	0	P_{i-1}
0	1	1	P_i
1	0	0	P_{i+31}
1	0	1	P_{i+32}
1	1	X	DISABLE

Am29C323 X AND Y INPUT DATA FORMATS

Fractional Two's Complement

TCX, TCY = 1

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}						2^{-28}	2^{-29}	2^{-30}	2^{-31}

Integer Two's Complement

TCX, TCY = 1

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
-2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^3	2^2	2^1	2^0

Unsigned Fractional

TCX, TCY = 0

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}						2^{-29}	2^{-30}	2^{-31}	2^{-32}

Unsigned Integer

TCX, TCY = 0

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^3	2^2	2^1	2^0

Am29C323 P-PORT OUTPUT DATA FORMATS

Fractional Two's Complement (Shifted)*

FA = 0, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}						2^{-28}	2^{-29}	2^{-30}	2^{-31}

FA = 0, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}	2^{-37}						2^{-60}	2^{-61}	2^{-62}	2^{-63}^{**}

Fractional Two's Complement

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
-2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}						2^{-27}	2^{-28}	2^{-29}	2^{-30}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{-31}	2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}						2^{-59}	2^{-60}	2^{-61}	2^{-62}

Integer Two's Complement

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
-2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}						2^{35}	2^{34}	2^{33}	2^{32}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^3	2^2	2^1	2^0

Unsigned Fractional

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}						2^{-29}	2^{-30}	2^{-31}	2^{-32}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{-33}	2^{-34}	2^{-35}	2^{-36}	2^{-37}	2^{-38}						2^{-61}	2^{-62}	2^{-63}	2^{-64}

Unsigned Integer

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}						2^{35}	2^{34}	2^{33}	2^{32}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	-	-	-	-	-	3	2	1	0
2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^3	2^2	2^1	2^0

*In this format, an overflow occurs in the attempted multiplication of the two's complement number -1.000 with itself, yielding a product of $+1.000$ which cannot be represented in this format. **This bit position (2^{-63}) equals zero in this format.

64 x 64 Multiplication

To perform a 64 x 64-bit multiplication using the Am29C323, each 64-bit input must be split into two 32-bit inputs; a most significant half and a least significant half (XW1 and XW0 or YW1 and YW0, respectively). These 32-bit inputs are then used to perform the four multiplications needed to obtain the 128-bit product. This product is represented in four 32-bit words, PW3 - PW0, the least significant word being PW0. The product is output 32 bits at a time through the product (P) port. The following equation shows the required multiplications:

$$X * Y = ((XW1 * YW1) * 2^{64}) + ((XW0 * YW1) * 2^{32}) + ((XW1 * YW0) * 2^{32}) + ((XW0 * YW0) * 2^0)$$

$$P = (PW3 * 2^{96}) + (PW2 * 2^{64}) + (PW1 * 2^{32}) + (PW0 * 2^0)$$

The Am29C323 uses an internal accumulator to sum these intermediate products. The previous equation, in a slightly

different form, is shown with the necessary instructions below:

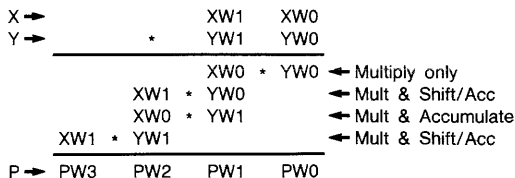


Table 4 details the movement of the input operands through the Am29C323. Table 5 defines the microcode required to perform a signed 64 x 64-bit multiplication. For an unsigned multiplication, TCX and TCY are LOW for all cycles. The operations and data movement are scheduled to produce a single product in seven clock cycles or a new pipelined product every four clock cycles.

TABLE 4. BUS AND REGISTER CONTENTS FOR A 64 x 64-BIT SIGNED MULTIPLICATION WITH ONE COMPLETE EXTENDED MULTIPLICATION SHOWN IN THE UNSHADED CYCLES

Cycle	0	1	2	3	4	5	6
X BUS	XW0	XW1			XW0	XW1	
XA REG	XW0	XW0	XW0	XW0	XW0	XW0	XW0
XB REG	XW1	XW1	XW1	XW1	XW1	XW1	XW1
Y BUS	YW0	YW1			YW0	YW1	
YA REG	YW0	YW0	YW0	YW0	YW0	YW0	YW0
YB REG	YW1	YW1	YW1	YW1	YW1	YW1	YW1
MPY OP	XW1·YW1	XW0·YW0	XW1·YW0	XW0·YW1	XW1·YW1	XW0·YW0	XW1·YW0
ACC OP	S/A	PASS	S/A	ACC	S/A	PASS	S/A
T REG		PW3	PW0			PW3	
P BUS	PW1	PW2	PW3	PW0	PW1	PW2	PW3

Note: MPY OP = Operation of multiplier array (X·Y)
 ACC OP = Operation of internal accumulator
 PASS = Pass through multiplier product
 ACC = Add previous result to current product
 S/A = Shift previous result then add to current product

TABLE 5. INSTRUCTION MICROCODE FOR 64 x 64-BIT SIGNED MULTIPLICATION WITH ONE COMPLETE EXTENDED MULTIPLICATION SHOWN IN THE UNSHADED CYCLES

Cycle	0	1	2	3	4	5	6	7	8	9	A	B	C	D
ENXA	0	1	1	1	0	1	1	1	0	1	1	1	0	1
ENXB	1	0	1	1	1	0	1	1	1	0	1	1	1	0
TCX	0	1	0	1	0	1	0	1	0	1	0	1	0	1
XSEL	1	0	1	0	1	0	1	0	1	0	1	0	1	0
ENYA	0	1	1	1	0	1	1	1	0	1	1	1	0	1
ENYB	1	0	1	1	1	0	1	1	1	0	1	1	1	0
TCY	0	0	1	1	0	0	1	1	0	0	1	1	0	0
YSEL	1	1	0	0	1	1	0	0	1	1	0	0	1	1
ENI	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ENT	1	0	0	1	1	0	0	1	1	0	0	1	1	0
TSEL	X	1	0	X	X	1	0	X	X	1	0	X	X	1
ACC0	0	1	1	1	0	1	1	1	0	1	1	1	0	1
ACC1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
ENP	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PSEL0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
PSEL1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Ambient Temperature Under Bias	-55 to +125°C
Supply Voltage to Ground Potential	
Continuous	-0.3 to +7.0 V
DC Voltage Applied to Outputs For	
High Output State	-0.3 to +V _{CC} + 0.3 V
DC Input Voltage	-0.3 to +V _{CC} + 0.3 V
DC Output Current, Into LOW Outputs	30 mA
DC Input Current	-10 to +10 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature (T _A)	0 to +70°C
Supply Voltage (V _{CC})	+4.75 to +5.25 V
Military* (M) Devices	
Temperature (T _A)	-55 to +125°C
Supply Voltage (V _{CC})	+4.5 to +5.5 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military Product 100% tested at T_A = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified (for APL Products, Group A, Subgroups 1, 2, 3 are tested unless otherwise noted)

Parameter Symbol	Parameter Description	Test Conditions (Note 1)	Min.	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min. V _{IN} = V _{IH} or V _{IL} I _{OH} = -0.4 mA	2.4		V
V _{OL}	Output LOW Voltage	V _{CC} = Min. V _{IN} = V _{IH} or V _{IL} I _{OL} = 4 mA		0.5	V
V _{IH}	Input HIGH Level	Guaranteed input logical HIGH voltage for all inputs (Note 2)	2.0		V
V _{IL}	Input LOW Level	Guaranteed input logical LOW voltage for all inputs (Note 2)		0.8	V
I _{IL}	Input LOW Current	V _{CC} = Max., V _{IN} = 0.4 V		-10	μA
I _{IH}	Input HIGH Current	V _{CC} = Max., V _{IN} = V _{CC} - 0.5 V		10	μA
I _{OZH} I _{OZL}	Off State (High Impedance) Output Current	V _{CC} = Max.		10 -10	μA
I _{CC}	Static Power Supply Current	V _{CC} = Max., V _{IN} = V _{CC} or GND, I _O = 0 μA		25 25	mA
C _{PD}	Power Dissipation Capacitance (Note 3)	V _{CC} = 5.0 V, T _A = 25°C, No Load		3000 pF Typical	

- Notes: 1. V_{CC} conditions shown as Min. or Max., refer to the military or commercial V_{CC} limits.
 2. These input levels provide zero noise immunity and should only be statically tested in a noise-free environment (not functionally tested).
 3. C_{PD} determines the no-load dynamic current consumption:
 I_{CC} (Total) = I_{CC} (Static) + C_{PD} V_{CC} f, where f is the switching frequency of the majority of the internal nodes, normally one-half of the clock frequency. This specification is not tested.

SWITCHING CHARACTERISTICS over **COMMERCIAL** operating range

No.	Parameter Symbol	Parameter Description	Test Conditions	29C323		29C323-1		29C323-2		Unit
				Min.	Max.	Min.	Max.	Min.	Max.	
UNLOCKED MODE										
1	t _{MUC}	Unlocked Multiply Time X ₀ - X ₃₁ , Y ₀ - Y ₃₁ to P ₀ - P ₃₁	FTX/Y/P = HIGH		120		100		70	ns
2	t _{MUCPP}	Unlocked Multiply Time X ₀ - X ₃₁ , Y ₀ - Y ₃₁ to PP ₀ - PP ₃	FTX/Y/P = HIGH		125		105		75	ns
3	t _{IP}	Instruction to P ₀ - P ₃₁ (Note 1)	Output Taken From Adder FTI = HIGH		120		100		70	ns
4	t _{IPP}	Instruction to PP ₀ - PP ₃	Output Taken From Adder FTI = HIGH		125		105		75	ns
CLOCKED MODE										
5	t _{MC}	Clocked Multiply Time	FTX/Y/P = LOW		100		80		55	ns
6	t _{DPP}	Clock to P ₀ - P ₃₁	Output Taken from Temp or Product Reg.		38		30		25	ns
7	t _{DPPPP}	Clock to PP ₀ - PP ₃	Output Taken from Temp or Product Reg.		43		35		30	ns
8	t _{PAP}	Clock to P ₀ - P ₃₁	Output Taken from Adder, FTX/Y/I = LOW		135		115		80	ns
9	t _{PAPP}	Clock to PP ₀ - PP ₃	Output Taken from Adder, FTX/Y/I = LOW		140		120		85	ns
10	t _{SP}	Data to Product Register Setup Time	FTX/Y = HIGH	110		90		65		ns
11	t _{HP}	Data to Product Register Hold Time	FTX/Y = HIGH	0		0		0		ns
12	t _{SIPT}	Instruction to Product Register Setup Time	FTI = HIGH	110		90		65		ns
13	t _{HIPT}	Instruction to Product Register Hold Time	FTI = HIGH	0		0		0		ns
14	t _{PWH}	Clock Pulse Width HIGH		20		20		15		ns
15	t _{PWL}	Clock Pulse Width LOW		20		20		15		ns
SETUP AND HOLD TIMES										
16	t _{SXY}	Register XA, XB, YA, YB Setup Time		21		18		15		ns
17	t _{HXY}	Register XA, XB, YA, YB Hold Time		0		0		0		ns
18	t _{SI}	Instruction Register Setup Time		18		15		10		ns
19	t _{HI}	Instruction Register Hold Time		0		0		0		ns
20	t _{SEN}	Register Enable Setup Time		18		15		10		ns
21	t _{HEN}	Register Enable Hold Time		0		0		0		ns
22	t _{STS}	TSEL Setup Time		18		15		10		ns
23	t _{HTS}	TSEL Hold Time		0		0		0		ns
COMMON PARAMETERS										
24	t _{PP}	PSEL ₀ - PSEL ₁ to P ₀ - P ₃₁	To Active State Only		35		30		25	ns
25	t _{PPP}	PSEL ₀ - PSEL ₁ to PP ₀ - PP ₃	To Active State Only		35		30		25	ns
26	t _{OE_{P1}}	\overline{OE} to P ₀ - P ₃₁ , PP ₀ - PP ₃ Output Enable			35		30		25	ns
27	t _{OD}	\overline{OE} or PSEL ₀ - PSEL ₁ to P ₀ - P ₃₁ , PP ₀ - PP ₃ Output Disable			35		30		25	ns
28	t _{DPE}	Data to PRERR			35		35		30	ns
29	t _{DHE}	Data to HDERR	Slave = HIGH		40		40		35	ns

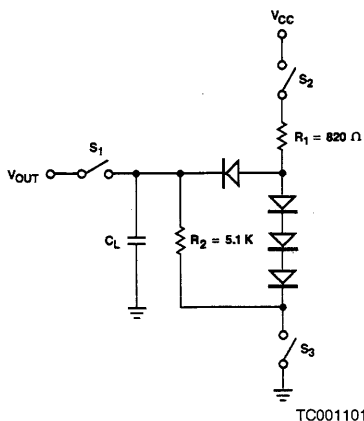
Notes: 1. Instruction signals are XSEL, YSEL, TCX, TCY, ACC0, ACC1, and RND.

SWITCHING CHARACTERISTICS over **MILITARY** operating range (for APL Products, Group A, Subgroups 9, 10, 11 are tested unless otherwise noted)

No.	Parameter Symbol	Parameter Description	Test Conditions	29C323		Unit
				Min.	Max.	
UNCLOCKED MODE						
1	tMUC	Unclocked Multiply Time X ₀ - X ₃₁ , Y ₀ - Y ₃₁ to P ₀ - P ₃₁	FTX/Y/P = HIGH		140	ns
2	tMUCPP	Unclocked Multiply Time X ₀ - X ₃₁ , Y ₀ - Y ₃₁ to PP ₀ - PP ₃	FTX/Y/P = HIGH		145	ns
3	t _I P	Instruction to P ₀ - P ₃₁ (Note 1)	Output Taken From Adder FTI = HIGH		140	ns
4	t _I PP	Instruction to PP ₀ - PP ₃	Output Taken From Adder FTI = HIGH		145	ns
CLOCKED MODE						
5	tMC	Clocked Multiply Time	FTX/Y/P = LOW		120	ns
6	tPDP	Clock to P ₀ - P ₃₁	Output Taken from Temp or Product Reg.		45	ns
7	tPDPp	Clock to PP ₀ - PP ₃	Output Taken from Temp or Product Reg.		50	ns
8	tPAP	Clock to P ₀ - P ₃₁	Output Taken from Adder, FTX/Y/I = LOW		150	ns
9	tPAPP	Clock to PP ₀ - PP ₃	Output Taken from Adder, FTX/Y/I = LOW		155	ns
10	tSP	Data to Product Register Setup Time	FTX/Y = HIGH	135		ns
11	tHP	Data to Product Register Hold Time	FTX/Y = HIGH	0		ns
12	tSIPT	Instruction to Product Reg. Setup Time	FTI = HIGH	135		ns
13	tHIPT	Instruction to Product Reg. Hold Time	FTI = HIGH	0		ns
14	tPWH	Clock Pulse Width HIGH		20		ns
15	tPWL	Clock Pulse Width LOW		20		ns
SETUP AND HOLD TIMES						
16	tSXY	Register XA, XB, YA, YB Setup Time		24		ns
17	tHXY	Register XA, XB, YA, YB Hold Time		0		ns
18	tSI	Instruction Register Setup Time		20		ns
19	tHI	Instruction Register Hold Time		0		ns
20	tSEN	Register Enable Setup Time		20		ns
21	tHEN	Register Enable Hold Time		0		ns
22	tSTS	TSEL Setup Time		20		ns
23	tHTS	TSEL Hold Time		0		ns
COMMON PARAMETERS						
24	t _{pp}	PSEL0 - PSEL1 to P ₀ - P ₃₁	To Active State Only		40	ns
25	t _{ppp}	PSEL0 - PSEL1 to PP ₀ - PP ₃	To Active State Only		40	ns
26	tOEP1	OE to P ₀ - P ₃₁ , PP ₀ - PP ₃ Output Enable			40	ns
27	tOD	OE or PSEL0 - PSEL1 to P ₀ - P ₃₁ , PP ₀ - PP ₃ Output Disable			40	ns
28	tDPE	Data to PRERR			40	ns
29	tDHE	Data to HDERR	Slave = HIGH		45	ns

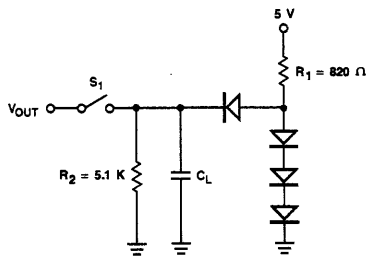
Notes: 1. Instruction signals are XSEL, YSEL, TCX, TCY, ACC0, ACC1, and RND.

SWITCHING TEST CIRCUITS



TC001101

A. Three-State Outputs



TC001082

B. Normal Outputs

- Notes:
1. $C_L = 50$ PF includes scope probe, wiring and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 4. S_1 and S_2 are closed while S_3 is open for t_{pZL} test.
 5. $C_L = \text{TBD}$ for output disable tests.

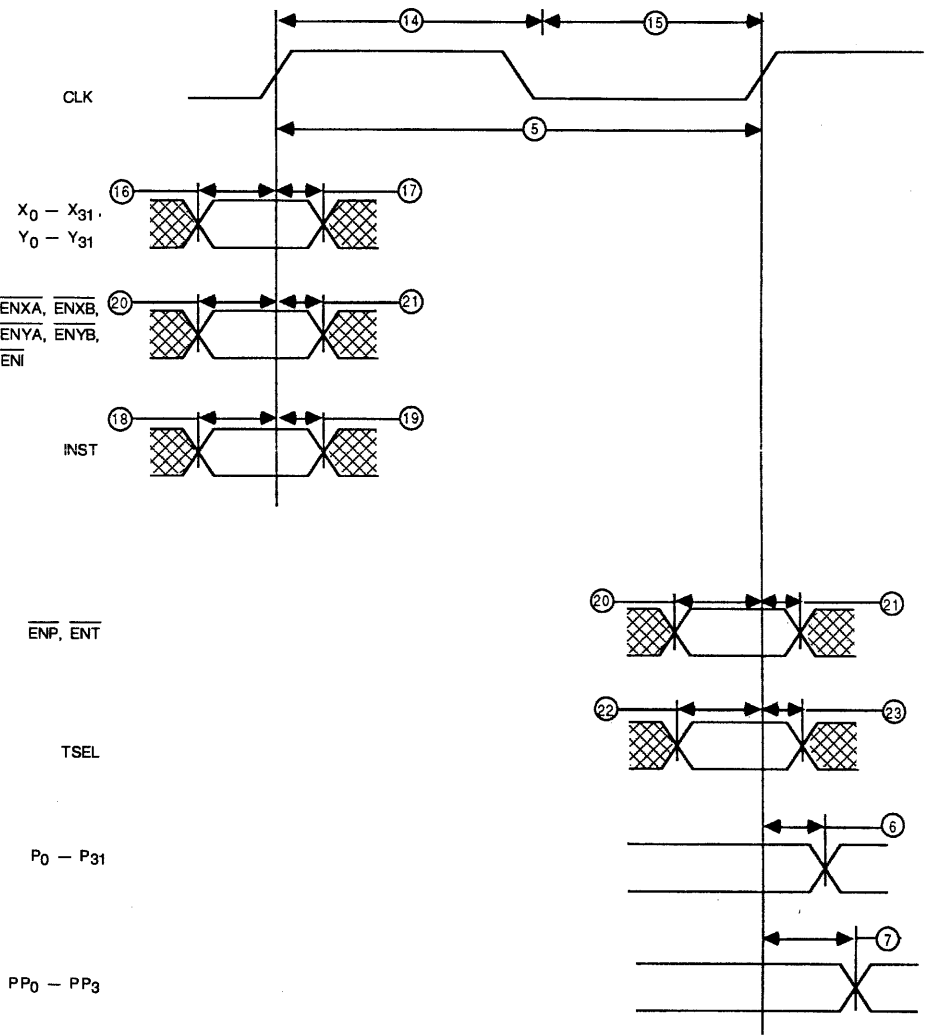
SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

KS000010

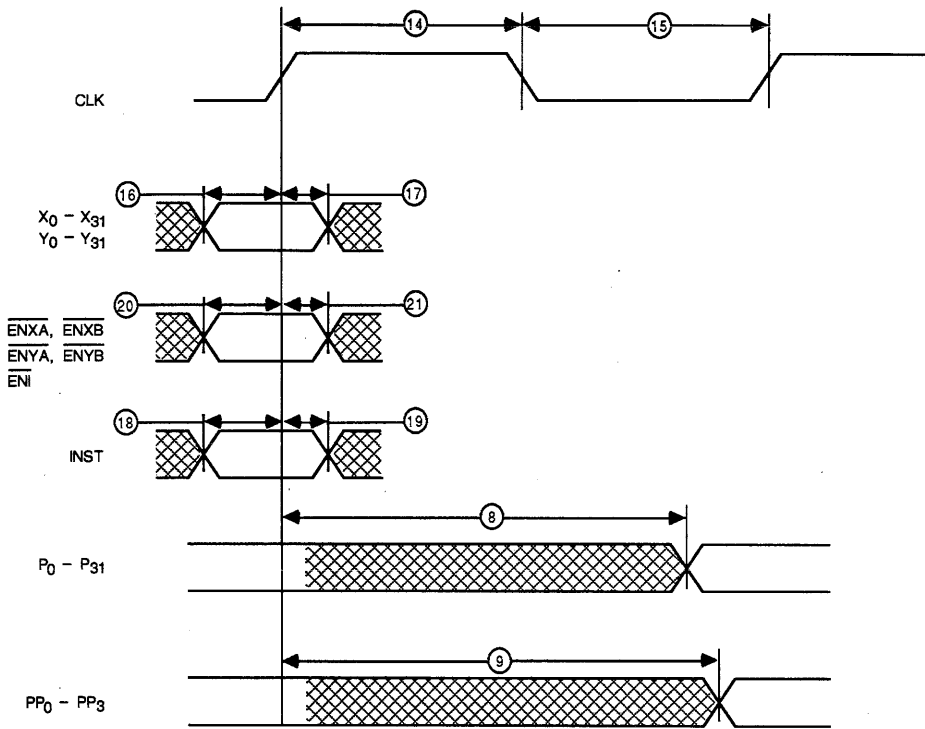
SWITCHING WAVEFORMS (Cont'd.)



WF022971

Clocked Operation: FTX, Y, P, I = LOW

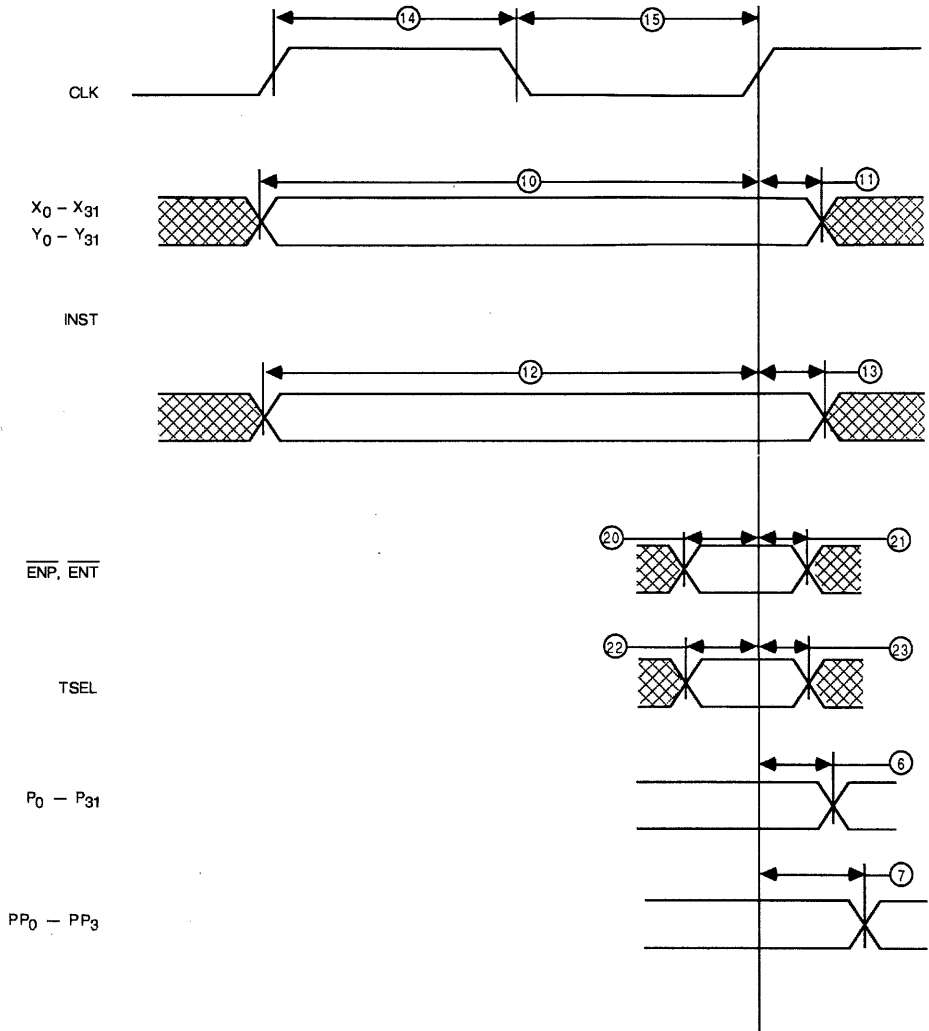
SWITCHING WAVEFORMS (Cont'd.)



WF022960

**Clocked Operation: Output Taken from Adder
(FTX, Y, I = LOW; FTP = HIGH; PSEL1 ≠ PSEL0)**

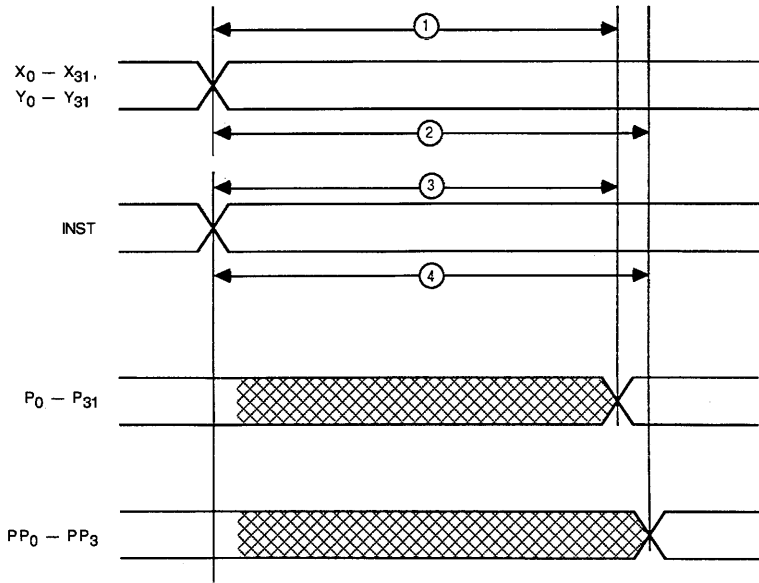
SWITCHING WAVEFORMS (Cont'd.)



WF022983

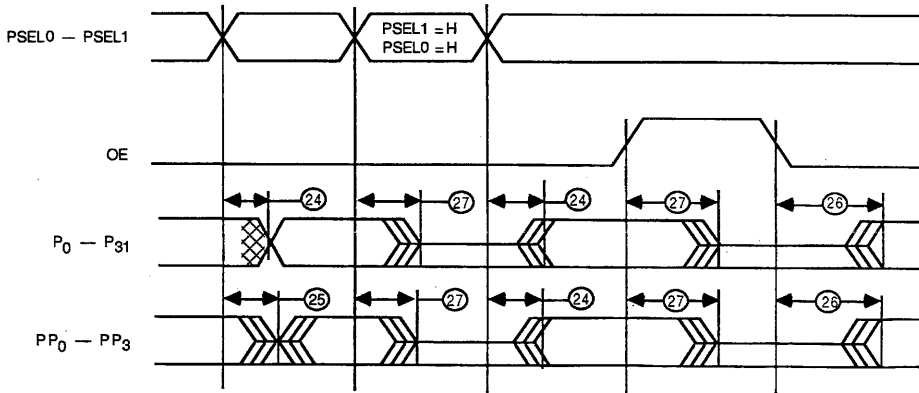
**Clocked Operation: Input Registers Bypassed
(FTX, Y, I = HIGH; FTP = LOW)**

SWITCHING WAVEFORMS (Cont'd.)



WF022990

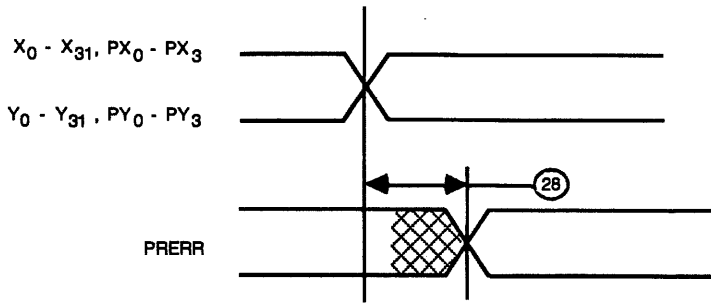
Unlocked Mode: FTX, Y, I, P = HIGH



WF023001

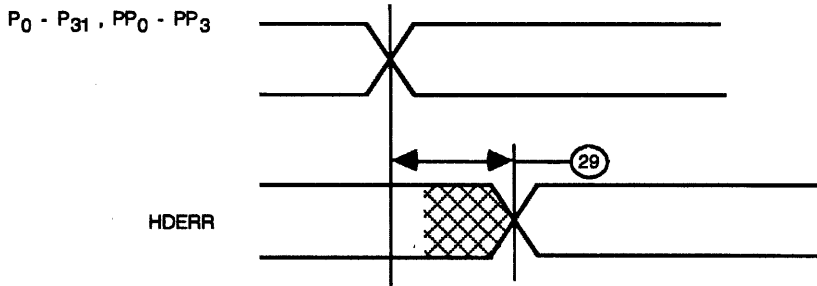
Output Select Timing

SWITCHING WAVEFORMS (Cont'd.)



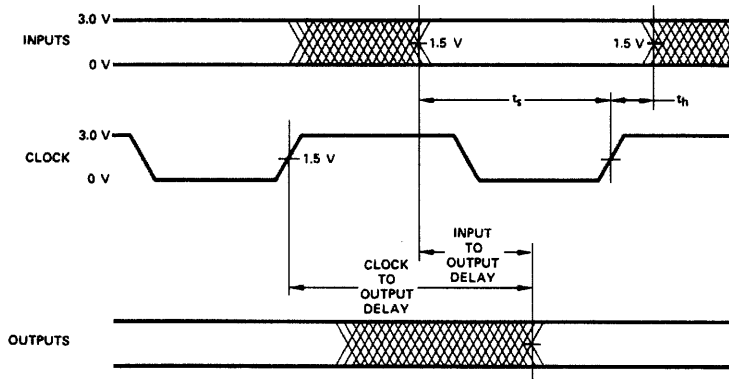
WF023013

PRERR Timing



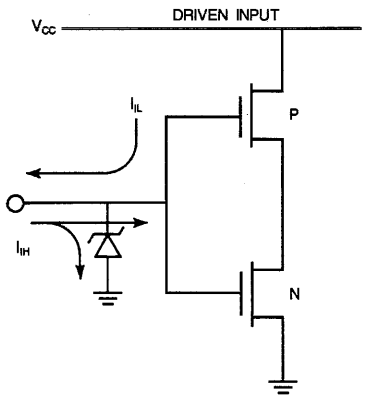
WF023024

Slave Mode Timing



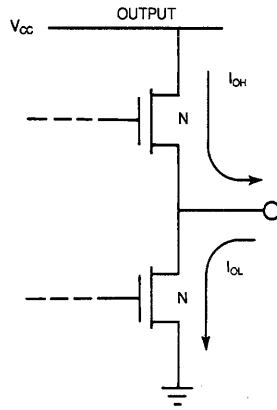
WFR02990

INPUT/OUTPUT CURRENT INTERFACE DIAGRAMS



IC000861

$C_i \approx 5.0$ pF, all inputs



IC000870

$C_o \approx 5.0$ pF, all outputs

Am29325

32-Bit Floating-Point Processor

Am29325

DISTINCTIVE CHARACTERISTICS

- Single VLSI device performs high-speed floating-point arithmetic
 - Floating-point addition, subtraction, and multiplication in a single clock cycle
 - Internal architecture supports sum-of-products, Newton-Raphson division
- 32-bit, three-bus flow-through architecture
 - Programmable I/O allows interface to 32- and 16-bit systems
- IEEE and DEC formats
 - Performs conversions between formats
 - Performs integer ↔ floating-point conversions
- Six flags indicate operation status
- Register enables eliminate clock skew
- Input and output registers can be made transparent independently

GENERAL DESCRIPTION

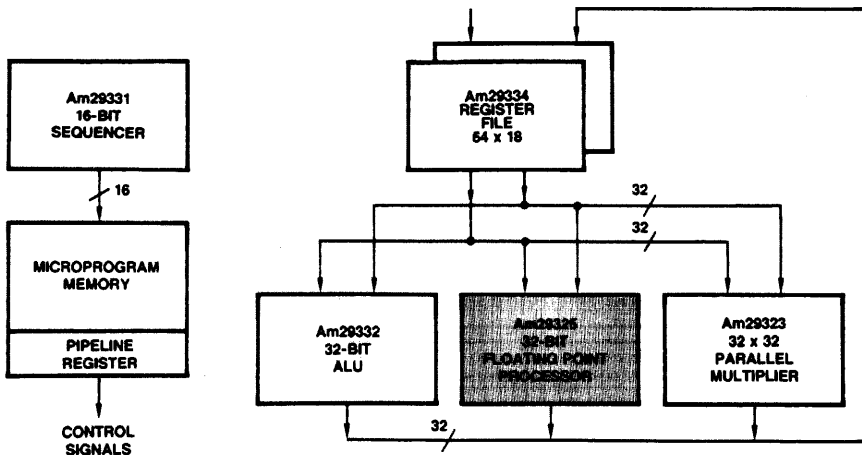
The Am29325 is a high-speed floating-point processor unit. It performs 32-bit single-precision floating-point addition, subtraction, and multiplication operations in a single VLSI circuit, using the format specified by the proposed IEEE floating-point standard, P754. The DEC single-precision floating-point format is also supported. Operations for conversion between 32-bit integer format and floating-point format are available, as are operations for converting between the IEEE and DEC floating-point formats. Any operation can be performed in a single clock cycle. Six flags — invalid operation, inexact result, zero, not-a-number, overflow, and underflow — monitor the status of operations.

The Am29325 has a three-bus, 32-bit architecture, with two input buses and one output bus. This configuration provides

high I/O bandwidth, allows access to all buses and affords a high degree of flexibility when connecting this device in a system. All buses are registered with each register having a clock enable. Input and output registers may be made transparent independently. Two other I/O configurations, a 32-bit, two-bus architecture and a 16-bit, three-bus architecture, are user-selectable, easing interface with a wide variety of systems. Thirty-two-bit internal feedforward datapaths support accumulation operations, including sum-of-products and Newton-Raphson division.

Fabricated with the high-speed IMOX™ bipolar process, the Am29325 is powered by a single 5-volt supply. The device is housed in a 145-terminal pin-grid-array package.

Am29300 FAMILY HIGH-PERFORMANCE SYSTEM BLOCK DIAGRAM

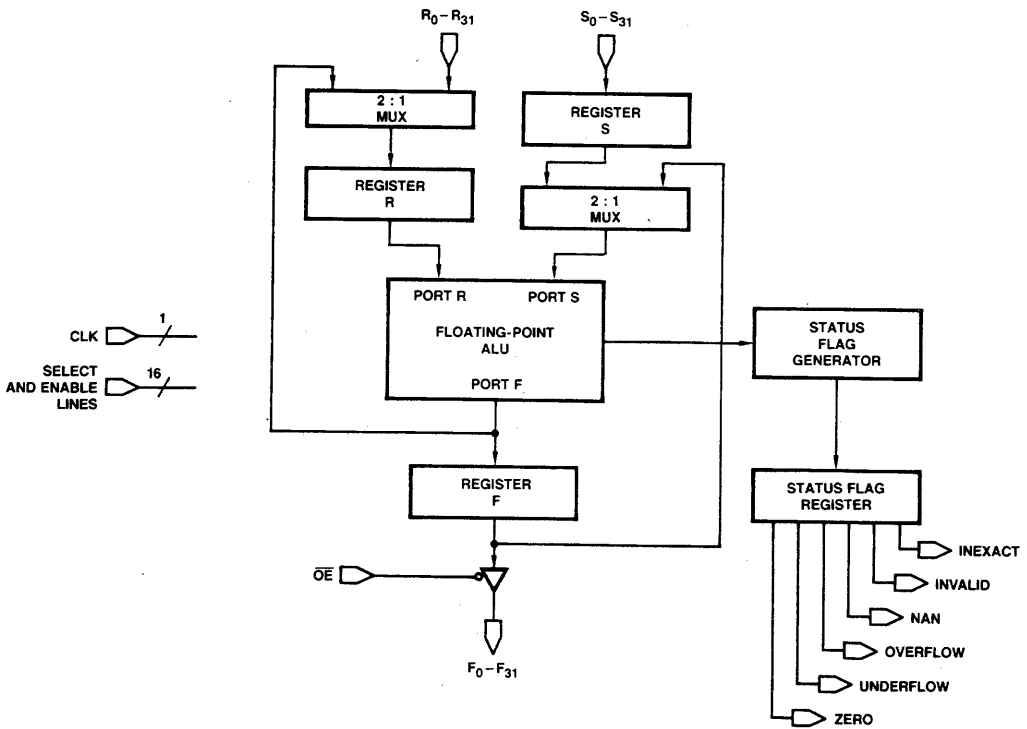


AF004650

RELATED AMD PRODUCTS

Part No.	Description
Am29114	Vectored Priority Interrupt Controller
Am29116	High-Performance Bipolar 16-Bit Microprocessor
Am29C116	High-Performance CMOS 16-Bit Microprocessor
Am29PL141	Fuse Programmable Controller
Am29C323	CMOS 32-Bit Parallel Multiplier
Am29331	16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29332	32-Bit Extended Function ALU
Am29C332	CMOS 32-Bit Extended Function ALU
Am29334	64 x 18 Four-Port, Dual-Access Register File
Am29C334	CMOS 64 x 18 Four-Port, Dual-Access Register File
Am29337	16-Bit Bounds Checker
Am29338	Byte Queue

BLOCK DIAGRAM



BD007080

CONNECTION DIAGRAM Top View

PGA

	A	B	C	D	E	F	G	H	J	K	L	M	N	P	R
1	INEX	I2	I1	$\overline{\text{ENR}}$	I4	OBUS	$\overline{\text{OE}}$	VCCE	CLK	R31	R30	R25	R24	R21	R20
2	INVA	NAN	I0	$\overline{\text{I/D}}$	FT0	FT1	VCCE	VCCE	RND0	RND1	R27	R28	R23	R22	R17
3	F29	ZERO	GNDT	$\overline{\text{ENR}}$	$\overline{\text{ENS}}$	16/32	VCCE	VCCE	VCCE	R29	R26	GNDE	GNDE	R19	R18
4	F30	F31	GNDT	*									R15	R16	R13
5	F23	OVFL	UNFL										R14	R11	R12
6	F26	F27	F28										R9	R10	R7
7	F21	F24	F25										R8	R5	R6
8	F22	F19	VCCT										R3	R4	R1
9	F17	F20	VCCT										R0	I3	R2
10	F18	F15	F16										S28	S31	S30
11	F13	F14	F11										S27	S26	S29
12	F12	F9	F10										VCCE	S25	S24
13	F7	F6	GNDT	GNDT	GNDT	GNDT	GNDE	GNDE	GNDE	S8	S13	S14	VCCE	S22	S23
14	F8	F3	F2	GNDT	F0	S1	S2	GNDE	S4	S9	S10	S15	S18	S21	S20
15	F5	F4	F1	GNDT	$\overline{\text{P/AFF}}$	S0	S3	S5	S7	S6	S11	S12	S17	S16	S19

CD010490

Key:

- 16/32 = S16/32
- GNDE = Ground, ECL
- GNDT = Ground, TTL
- I/D = IEEE/DEC
- INEX = INEXACT
- INVA = INVALID
- OBUS = ONEBUS
- OVFL = OVERFLOW
- P/AFF = PROJ/AFF
- UNFL = UNDERFLOW
- VCCE = V_{CC}, ECL
- VCCT = V_{CC}, TTL

*D4 is an alignment pin (not connected internally).

PIN DESIGNATIONS

(Sorted by Pin No.)

PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A-1	Inexact	C-7	F ₂₅	H-13	GNDE	N-10	S ₂₈
A-2	Invalid	C-8	V _{CC} T	H-14	GNDE	N-11	S ₂₇
A-3	F ₂₉	C-9	V _{CC} T	H-15	S ₅	N-12	V _{CC} E
A-4	F ₃₀	C-10	F ₁₆	J-1	CLK	N-13	V _{CC} E
A-5	F ₂₃	C-11	F ₁₁	J-2	RND ₀	N-14	S ₁₈
A-6	F ₂₆	C-12	F ₁₀	J-3	V _{CC} E	N-15	S ₁₇
A-7	F ₂₁	C-13	GNDT	J-13	GNDE	P-1	R ₂₁
A-8	F ₂₂	C-14	F ₂	J-14	S ₄	P-2	R ₂₂
A-9	F ₁₇	C-15	F ₁	J-15	S ₇	P-3	R ₁₉
A-10	F ₁₈	D-1	$\overline{\text{ENF}}$	K-1	R ₃₁	P-4	R ₁₆
A-11	F ₁₃	D-2	IEEE/ $\overline{\text{DEC}}$	K-2	RND ₁	P-5	R ₁₁
A-12	F ₁₂	D-3	$\overline{\text{ENR}}$	K-3	R ₂₉	P-6	R ₁₀
A-13	F ₇	D-13	GNDT	K-13	S ₈	P-7	R ₅
A-14	F ₈	D-14	GNDT	K-14	S ₉	P-8	R ₄
A-15	F ₅	D-15	GNDT	K-15	S ₆	P-9	I ₃
B-1	I ₂	E-1	I ₄	L-1	R ₃₀	P-10	S ₃₁
B-2	NAN	E-2	FT ₀	L-2	R ₂₇	P-11	S ₂₆
B-3	ZERO	E-3	$\overline{\text{ENS}}$	L-3	R ₂₆	P-12	S ₂₅
B-4	F ₃₁	E-13	GNDT	L-13	S ₁₃	P-13	S ₂₂
B-5	OVERFLOW	E-14	F ₀	L-14	S ₁₀	P-14	S ₂₁
B-6	F ₂₇	E-15	PROJ/ $\overline{\text{AFF}}$	L-15	S ₁₁	P-15	S ₁₆
B-7	F ₂₄	F-1	ONEBUS	M-1	R ₂₅	R-1	R ₂₀
B-8	F ₁₉	F-2	FT ₁	M-2	R ₂₈	R-2	R ₁₇
B-9	F ₂₀	F-3	S _{16/32}	M-3	GNDE	R-3	R ₁₈
B-10	F ₁₅	F-13	GNDT	M-13	S ₁₄	R-4	R ₁₃
B-11	F ₁₄	F-14	S ₁	M-14	S ₁₅	R-5	R ₁₂
B-12	F ₉	F-15	S ₀	M-15	S ₁₂	R-6	R ₇
B-13	F ₆	G-1	$\overline{\text{OE}}$	N-1	R ₂₄	R-7	R ₆
B-14	F ₃	G-2	V _{CC} E*	N-2	R ₂₃	R-8	R ₁
B-15	F ₄	G-3	V _{CC} E	N-3	GNDE	R-9	R ₂
C-1	I ₁	G-13	GNDE	N-4	R ₁₅	R-10	S ₃₀
C-2	I ₀	G-14	S ₂	N-5	R ₁₄	R-11	S ₂₉
C-3	GNDT*	G-15	S ₃	N-6	R ₉	R-12	S ₂₄
C-4	GNDT	H-1	V _{CC} E	N-7	R ₈	R-13	S ₂₃
C-5	UNDERFLOW	H-2	V _{CC} E	N-8	R ₃	R-14	S ₂₀
C-6	F ₂₈	H-3	V _{CC} E	N-9	R ₀	R-15	S ₁₉

*T and E represent TTL and ECL, respectively.

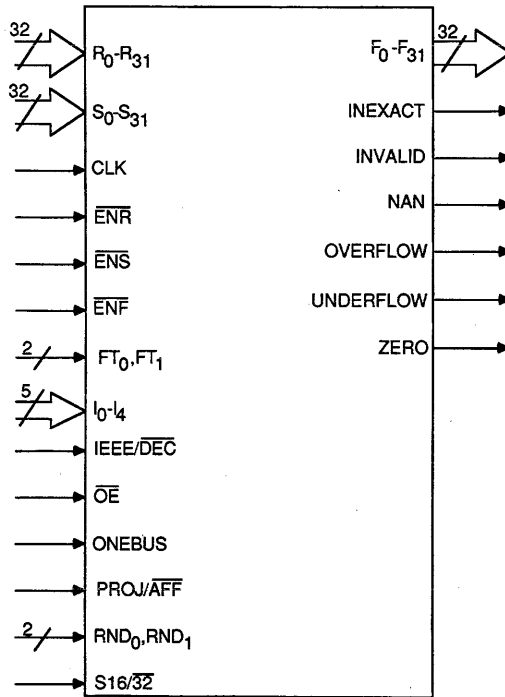
PIN DESIGNATIONS (Cont'd.)

(Sorted by Pin Name)

PIN NO.	PIN NAME	PIN NO.	PIN NAME.	PIN NO.	PIN NAME	PIN NO.	PIN NAME.
J-1	CLK	E-2	FT ₀	R-6	R ₇	K-14	S ₉
D-1	ENF	F-2	FT ₁	N-7	R ₈	L-14	S ₁₀
D-3	ENR	N-3	GNDE*	N-6	R ₉	L-15	S ₁₁
E-3	ENS	H-14	GNDE	P-6	R ₁₀	M-15	S ₁₂
E-14	F ₀	G-13	GNDE	P-5	R ₁₁	L-13	S ₁₃
C-15	F ₁	M-3	GNDE	R-5	R ₁₂	M-13	S ₁₄
C-14	F ₂	H-13	GNDE	R-4	R ₁₃	M-14	S ₁₅
B-14	F ₃	J-13	GNDE	N-5	R ₁₄	P-15	S ₁₆
B-15	F ₄	D-15	GNDT	N-4	R ₁₅	F-3	S _{16/32}
A-15	F ₅	D-14	GNDT	P-4	R ₁₆	N-15	S ₁₇
B-13	F ₆	E-13	GNDT	R-2	R ₁₇	N-14	S ₁₈
A-13	F ₇	F-13	GNDT	R-3	R ₁₈	R-15	S ₁₉
A-14	F ₈	C-4	GNDT	P-3	R ₁₉	R-14	S ₂₀
B-12	F ₉	C-3	GNDT	R-1	R ₂₀	P-14	S ₂₁
C-12	F ₁₀	D-13	GNDT	P-1	R ₂₁	P-13	S ₂₂
C-11	F ₁₁	C-13	GNDT	P-2	R ₂₂	R-13	S ₂₃
A-12	F ₁₂	C-2	I ₀	N-2	R ₂₃	R-12	S ₂₄
A-11	F ₁₃	C-1	I ₁	N-1	R ₂₄	P-12	S ₂₅
B-11	F ₁₄	B-1	I ₂	M-1	R ₂₅	P-11	S ₂₆
B-10	F ₁₅	P-9	I ₃	L-3	R ₂₆	N-11	S ₂₇
C-10	F ₁₆	E-1	I ₄	L-2	R ₂₇	N-10	S ₂₈
A-9	F ₁₇	D-2	IEEE/DEC	M-2	R ₂₈	R-11	S ₂₉
A-10	F ₁₈	A-1	INEXACT	K-3	R ₂₉	R-10	S ₃₀
B-8	F ₁₉	A-2	INVALID	L-1	R ₃₀	P-10	S ₃₁
B-9	F ₂₀	B-2	NAN	K-1	R ₃₁	C-5	UNDERFLOW
A-7	F ₂₁	G-1	OE	J-2	RND ₀	J-3	V _{CC} E
A-8	F ₂₂	F-1	ONEBUS	K-2	RND ₁	G-2	V _{CC} E
A-5	F ₂₃	B-5	OVERFLOW	F-15	S ₀	G-3	V _{CC} E
B-7	F ₂₄	E-15	PROJ/AF	F-14	S ₁	H-2	V _{CC} E
C-7	F ₂₅	N-9	R ₀	G-14	S ₂	N-13	V _{CC} E
A-6	F ₂₆	R-8	R ₁	G-15	S ₃	N-12	V _{CC} E
B-6	F ₂₇	R-9	R ₂	J-14	S ₄	H-3	V _{CC} E
C-6	F ₂₈	N-8	R ₃	H-15	S ₅	H-1	V _{CC} E
A-3	F ₂₉	P-8	R ₄	K-15	S ₆	C-8	V _{CC} T
A-4	F ₃₀	P-7	R ₅	J-15	S ₇	C-9	V _{CC} T
B-4	F ₃₁	R-7	R ₆	K-13	S ₈	B-3	ZERO

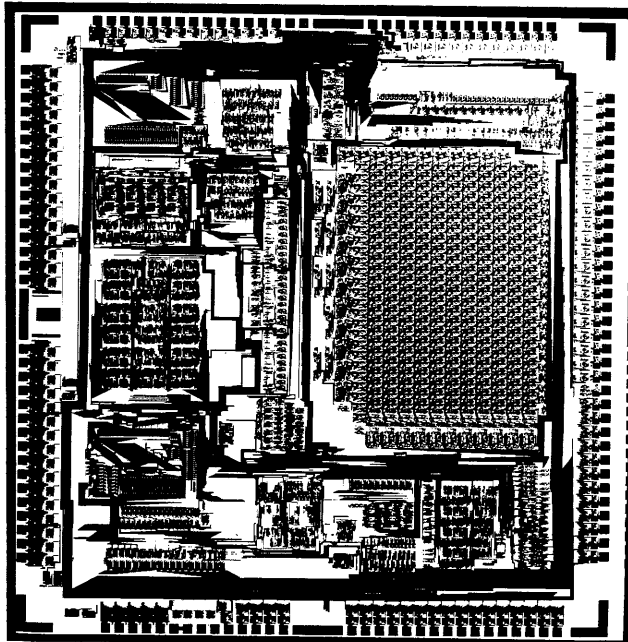
*E and T represent ECL and TTL, respectively.

LOGIC SYMBOL



LS002920

METALLIZATION AND PAD LAYOUT

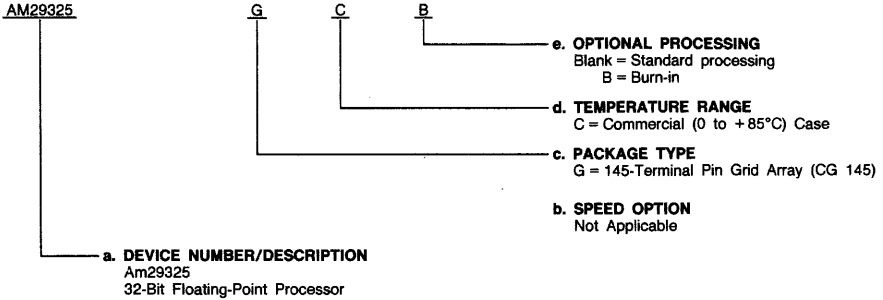


ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. Device Number**
- b. Speed Option** (if applicable)
- c. Package Type**
- d. Temperature Range**
- e. Optional Processing**



Valid Combinations	
Am29325	GC, GCB

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

R₀–R₃₁ R Operand Bus (Input)

R₀ is the least-significant bit.

S₀–S₃₁ S Operand Bus (Input)

S₀ is the least-significant bit.

F₀–F₃₁ F Operand Bus (Output)

F₀ is the least-significant bit.

CLK Clock (Input)

For the internal registers.

ENR Register R Clock Enable (Input; Active LOW)

When ENR is LOW, register R is clocked on the LOW-to-HIGH transition of CLK. When ENR is HIGH, register R retains the previous contents.

ENS Register S Clock Enable (Input; Active LOW)

When ENS is LOW, register S is clocked on the LOW-to-HIGH transition of CLK. When ENS is HIGH, register S retains the previous contents.

ENF Register F Clock Enable (Input; Active LOW)

When ENF is LOW, register F is clocked on the LOW-to-HIGH transition of CLK. When ENF is HIGH, register F retains the previous contents.

OE Output Enable (Input; Active LOW)

When OE is LOW, the contents of register F are placed on F₀–F₃₁. When OE is HIGH, F₀–F₃₁ assume a high-impedance state.

ONEBUS Input Bus Configuration Control (Input)

A LOW on ONEBUS configures the input bus circuitry for two-input bus operation. A HIGH on ONEBUS configures the input bus circuitry for single-input bus operation.

FT₀ Input Register Feedthrough Control (Input; Active HIGH)

When FT₀ is HIGH, registers R and S are transparent.

FT₁ Output Register Feedthrough Control (Input; Active HIGH)

When FT₁ is HIGH, register F and the status flag register are transparent.

I₀–I₂ Operation Select Lines (Input)

Used to select the operation to be performed by the ALU. See Table 1 for a list of operations and the corresponding codes.

I₃ ALU S Port Input Select (Input)

A LOW on I₃ selects register S as the input to the ALU S port. A HIGH on I₃ selects register F as the input to the ALU S port.

I₄ Register R Input Select (Input)

A LOW on I₄ selects R₀–R₃₁ as the input to register R. A HIGH selects the ALU F port as the input to register R.

IEEE/DEC IEEE/DEC Mode Select (Input)

When IEEE/DEC is HIGH, IEEE mode is selected. When IEEE/DEC is LOW, DEC mode is selected.

S16/32 16- or 32-Bit I/O Mode Select (Input)

A LOW on S16/32 selects the 32-bit I/O mode; a HIGH selects the 16-bit I/O mode. In 32-bit mode, input and output buses are 32 bits wide. In 16-bit mode, input and output buses are 16 bits wide, with the least- and most-significant portions of the 32-bit input and output words being placed on the buses during the HIGH and LOW portions of CLK, respectively.

RND₀, RND₁ Rounding Mode Selects (Input)

RND₀ and RND₁ select one of four rounding modes. See Table 5 for a list of rounding modes and the corresponding control codes.

PROJ/AFF Projective/Affine Mode Select (Input)

Choice of projective or affine mode determines the way in which infinities are handled in IEEE mode. A LOW on PROJ/AFF selects affine mode; a HIGH selects projective mode.

OVERFLOW Overflow Flag (Output; Active HIGH)

A HIGH indicates that the last operation produced a final result that overflowed the floating-point format.

UNDERFLOW Underflow Flag (Output; Active HIGH)

A HIGH indicates that the last operation produced a rounded result that underflowed the floating-point format.

ZERO Zero Flag (Output; Active HIGH)

A HIGH indicates that the last operation produced a final result of zero.

NAN Not-a-Number Flag (Output; Active HIGH)

A HIGH indicates that the final result produced by the last operation is not to be interpreted as a number. The output in such cases is either an IEEE Not-a-Number (NaN) or a DEC-reserved operand.

INVALID Invalid Operation Flag (Output; Active HIGH)

A HIGH indicates that the last operation performed was invalid; e.g., ∞ times 0.

INEXACT Inexact Result Flag (Output; Active HIGH)

A HIGH indicates that the final result of the last operation was not infinitely precise, due to rounding.

Definition of Terms

Affine Mode

One of two modes affecting the handling of operations on infinities — see the **Operations with Infinities** section under **Operations in IEEE Mode**.

Biased Exponent

The true exponent of a floating-point number, plus a constant. For IEEE floating-point numbers, the constant is 127; for DEC floating-point numbers, the constant is 128. See also **True Exponent**.

Bus

Data input or output channel for the floating-point processor.

DEC-Reserved Operand

A DEC floating-point number that is interpreted as a symbol and has no numeric value. A DEC-reserved operand has a sign of 1 and a biased exponent of 0.

Destination Format

The format of the final result produced by the floating-point ALU. The destination format can be IEEE floating point, DEC floating point, or integer.

Final Result

The result produced by the floating-point ALU.

Fraction

The 23 least-significant bits of the mantissa.

Infinitely Precise Result

The result that would be obtained from an operation if both exponent range and precision were unbounded.

Input Operands

The value or values on which an operation is performed. For example, the addition $2 + 3 = 5$ has input operands 2 and 3.

Mantissa

The portion of a floating-point number containing the number's significant bits. For the floating-point number 1.101×2^{-3} , the mantissa is 1.101.

NAN (Not-a-Number)

An IEEE floating-point number that is interpreted as a symbol, and has no numeric value. A NAN has a biased exponent of 255_{10} and a non-zero fraction.

Port

Data input or output channel for the floating-point ALU.

Projective Mode

One of two modes affecting the handling of operations on infinities — see the **Operations with Infinities** section under **Operation in IEEE Mode**.

Rounded Result

The result produced by rounding the infinitely precise result to fit the destination format.

True Exponent (or Exponent)

Number representing the power of two by which a floating-point number's mantissa is to be multiplied. For the floating-point number 1.101×2^{-3} , the true exponent is -3 .

FUNCTIONAL DESCRIPTION

Architecture

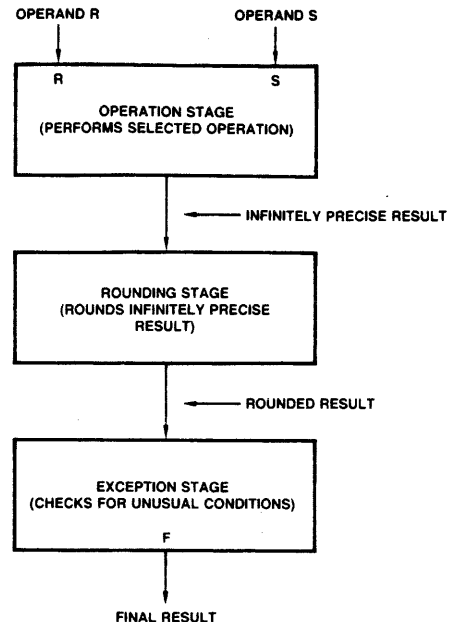
The Am29325 comprises a high-speed, floating-point ALU, a status flag generator, and a 32-bit data path.

Floating-Point ALU

The floating-point ALU performs 32-bit floating-point operations. It also performs floating-point-to-integer conversions, integer-to-floating-point floating-point conversions, and conversions between the IEEE and DEC formats. The ALU has two 32-bit input ports, R and S, and a 32-bit output port, F.

Conceptually, the process performed by the ALU can be divided into three stages (see Figure 1). The operation stage performs the arithmetic operation selected by the user; the output of this section is referred to as the infinitely precise result of the operation. The rounding stage rounds the infinitely precise result to fit in the destination format; the output of this stage is called the rounded result. The last stage checks for exceptional conditions. If no exceptional condition is found, the rounded result is passed through this stage. If some exceptional condition is found (e.g., overflow, underflow, or an invalid operation), this section may replace the rounded result with another output, such as $+\infty$, $-\infty$, a NAN, or a DEC-

reserved operand. The output of this last stage appears on port F, and is called the final result.



AF004540

Figure 1. Conceptual Model of the Process Performed by the Floating-Point ALU

The ALU performs one of eight operations; the operation to be performed is selected by placing the appropriate control code on lines $l_0 - l_2$. Table 1 gives the control codes corresponding to each of the eight operations.

The floating-point addition operation (R PLUS S) adds the floating-point numbers on ports R and S, and places the floating-point result on port F. In IEEE mode (IEEE/DEC = HIGH) the addition is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the addition is performed in DEC format.

The floating-point subtraction operation (R MINUS S) subtracts the floating-point number on port S from the floating-point number on port R and places the floating-point result on port F. In IEEE mode (IEEE/DEC = HIGH) the subtraction is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the subtraction is performed in DEC format.

The floating-point multiplication operation (R TIMES S) multiplies the floating-point numbers on ports R and S, and places the floating-point result on port F. In IEEE mode (IEEE/DEC = HIGH) the multiplication is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the multiplication is performed in DEC format.

The floating-point constant subtraction (2 MINUS S) operation subtracts the floating-point value on port S from 2, and places the result on port F. The operand on port R is not used in this operation; its value will not affect the operation in any way. In IEEE mode (IEEE/DEC = HIGH) the operation is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the operation is performed in DEC format. This operation is

used to support Newton-Raphson floating-point division; a description of its use appears in **Appendix C**.

The integer-to-floating-point conversion (INT-TO-FP) operation takes a 32-bit, two's-complement integer on port R and places the equivalent floating-point value on port F. The

operand on port S is not used in this operation; its value will not affect the operation in any way. In IEEE mode (IEEE/DEC = HIGH) the result is delivered in IEEE format; in DEC mode (IEEE/DEC = LOW) the result is delivered in DEC format.

TABLE 1. ALU OPERATION SELECT

I_2	I_1	I_0	Operation	Output Equation
0	0	0	Floating-point addition (R PLUS S)	$F = R + S$
0	0	1	Floating-point subtraction (R MINUS S)	$F = R - S$
0	1	0	Floating-point multiplication (R TIMES S)	$F = R * S$
0	1	1	Floating-point constant subtraction (2 MINUS S)	$F = 2 - S$
1	0	0	Integer-to-floating-point conversion (INT-TO-FP)	F (floating-point) = R (integer)
1	0	1	Floating-point-to-integer conversion (FP-TO-INT)	F (integer) = R (floating-point)
1	1	0	IEEE-TO-DEC format conversion (IEEE-TO-DEC)	F (DEC format) = R (IEEE format)
1	1	1	DEC-TO-IEEE format conversion (DEC-TO-IEEE)	F (IEEE format) = R (DEC format)

The floating-point-to-integer conversion (FP-TO-INT) operation takes a floating-point number on port R and places the equivalent 32-bit, two's-complement integer value on port F. The operand on port S is not used in this operation; its value will not affect the operation in any way. In IEEE mode (IEEE/DEC = HIGH) the operand on port R is interpreted using the IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) it is interpreted using the DEC floating-point format.

The IEEE-to-DEC conversion operation (IEEE-TO-DEC) takes an IEEE-format floating-point number on port R and places the equivalent DEC-format floating-point number on port F. The operand on port S is not used in this operation; its value will not affect the operation in any way. The operation can be performed in either IEEE mode (IEEE/DEC = HIGH) or DEC mode (IEEE/DEC = LOW).

The DEC-to-IEEE conversion operation (DEC-TO-IEEE) takes a DEC-format floating-point number on port R and places the equivalent IEEE-floating-point number on port F. The operand on port S is not used in this operation; its value will not affect the operation in any way. The operation can be performed in either IEEE mode (IEEE/DEC = HIGH) or DEC mode (IEEE/DEC = LOW).

Status Flag Generator

The status flag generator controls the state of six flags that report the status of floating-point ALU operations. The flags indicate when an operation is invalid (e.g., ∞ times 0) or when an operation has produced an overflow, an underflow, a non-numerical result (e.g., a NAN- or DEC-reserved operand), an inexact result, or a result of zero. The flags represent the status of the most recently performed operation. Flag status is stored in the flag status register on the LOW-to-HIGH transition of CLK. When the output register feedthrough control FT₁ is HIGH, the flag status register is made transparent.

Data Path

The 32-bit data path consists of the R and S input buses; the F output bus; data registers R, S, and F; the register R input multiplexer; and the ALU port S input multiplexer.

Input operands enter the floating-point processor through the 32-bit R and S input buses, R₀ - R₃₁ and S₀ - S₃₁. Results of operations appear on the 32-bit F bus, F₀ - F₃₁. The F bus assumes a high-impedance state when output enable OE is HIGH.

The R and S registers store input operands; the F register stores the final result of the floating-point ALU operation. Each register has an independent clock enable (ENR, ENS, and ENF). When a register's clock enable is LOW, the register stores the data on its input at the LOW-to-HIGH transition of CLK; when the clock enable is HIGH, the register retains its current data. All data registers are fully edge-triggered — both the input data and the register enable need only meet modest setup and hold time requirements. Registers R and S can be made transparent by setting FT₀, the input register feedthrough control, HIGH. Register F can be made transparent by setting FT₁, the output register feedthrough control, HIGH.

The register R input multiplexer selects either the R input bus or the floating-point ALU's F port as the input to register R. Selection is controlled by I₄ — a LOW selects the R input bus; a HIGH selects the ALU F port. The ALU port S input multiplexer selects either register S or register F as the input to the floating-point ALU's S port. Selection is controlled by I₃ — a LOW selects register S; a HIGH selects register F.

Data selected by I₃ and I₄ is described in Table 2. When registers R and S are transparent (FT₀ = HIGH), multiplexer select I₄ must be kept LOW, so that the register R input multiplexer selects R₀ - R₃₁. When register F is transparent (FT₁ = HIGH), multiplexer select I₃ must be kept LOW, so that the ALU port S input multiplexer selects register S.

TABLE 2. MUX SELECT

I_3	Data selected for floating-point ALU S port
0	Register S
1	Register F
I_4	Data selected for register R input
0	R bus
1	Floating-point ALU port F

TABLE 3. I/O MODE SELECTION

$S16/\overline{32}$	ONEBUS	I/O Mode
0	0	32-bit, two-input-bus mode
0	1	32-bit, single-input-bus mode(*)
1	0	16-bit, two-input-bus mode(*)
1	1	Illegal I/O mode selection value

* FT_0 must be held LOW in this mode (see text).

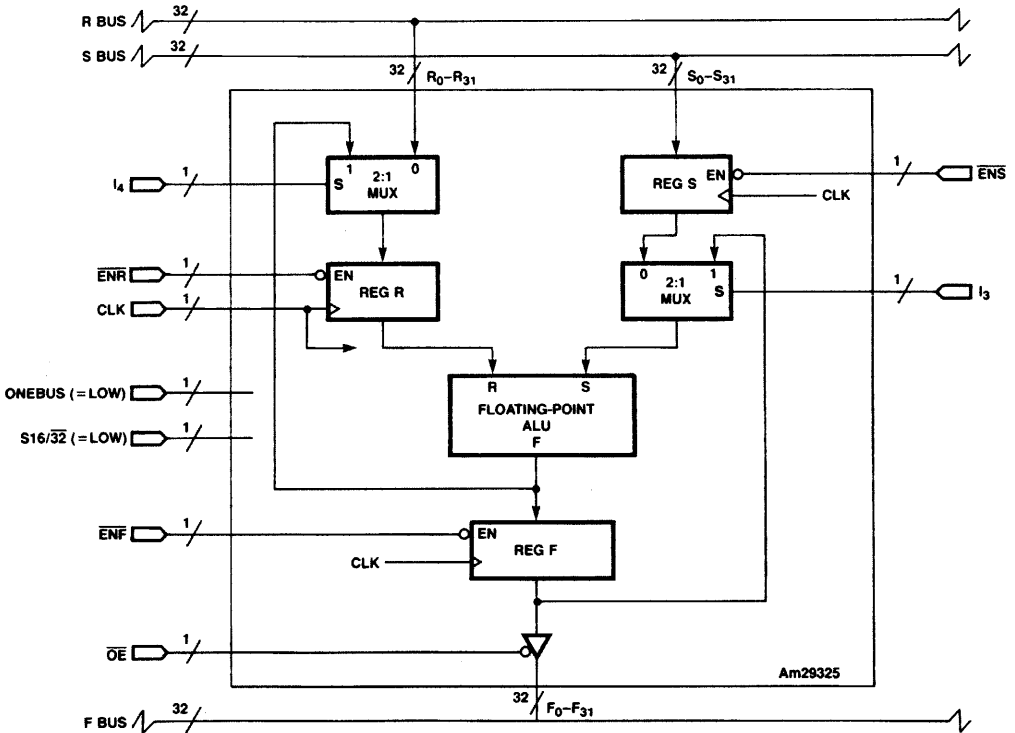
I/O Modes

The Am29325 datapath can be configured in one of three I/O modes: a 32-bit, two-input bus mode; a 32-bit, single-input bus mode; and a 16-bit, two-input bus mode. These modes affect only the manner in which data is delivered to and taken from the Am29325; operation of the floating-point ALU is not altered. The I/O mode is selected with the ONEBUS and $S16/\overline{32}$ controls. Table 3 lists the control codes needed to invoke each I/O mode.

32-Bit, Two-Input Bus Mode

In this I/O mode, the R and S buses are configured as independent 32-bit input buses, and the F bus is configured as a 32-bit output bus. Figure 2 is a functional block diagram of the Am29325 in this I/O mode.

R and S operands are taken from their respective input buses and clocked into the R and S registers on the LOW-to-HIGH transition of CLK. Register F is also clocked on the LOW-to-HIGH transition of CLK. Figure 5(a) depicts typical I/O timing in this mode.



BD007050

Figure 2. Functional Block Diagram for the 32-Bit, Two-Input Bus Mode

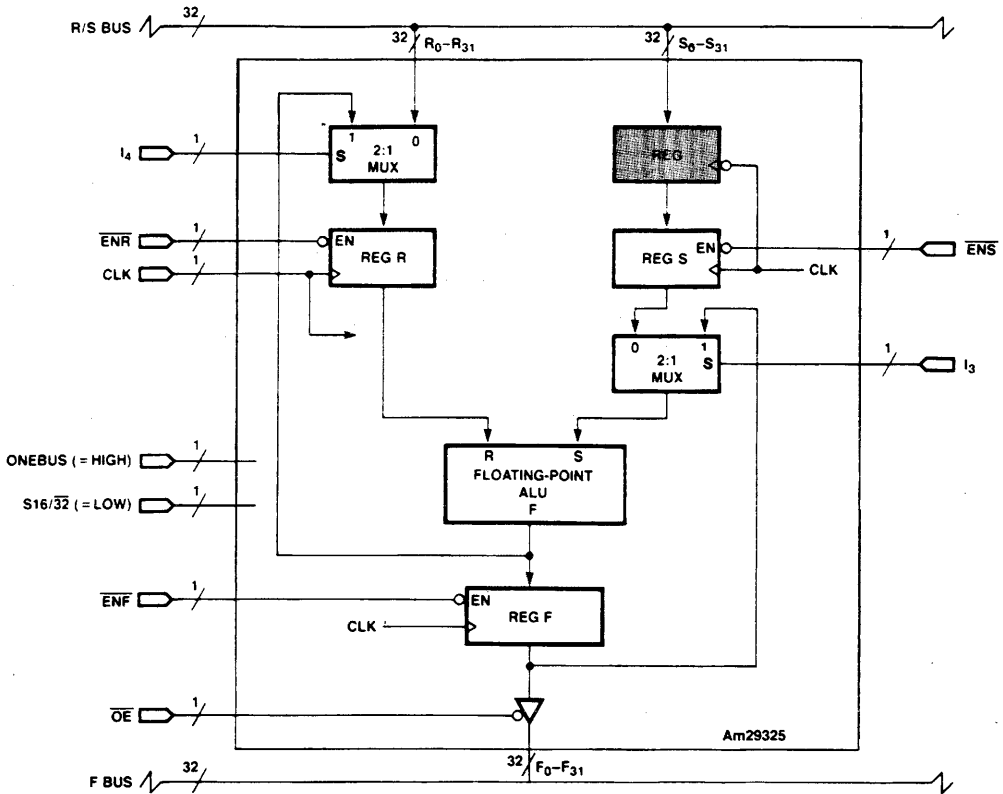
32-Bit, Single-Input Bus Mode

In this I/O mode, the R and S buses are connected to a single 32-bit multiplexed input data bus; the F bus is configured as an independent 32-bit output bus. Figure 3 is a functional block diagram of the Am29325 in this I/O mode. Note that both the R and S bus lines must be wired to the input bus.

R and S operands are multiplexed onto the input bus by the host system. The S operand is clocked from the input bus into a temporary holding register on the HIGH-to-LOW transition of CLK and is transferred to register S on the LOW-to-HIGH

transition of CLK. The R operand is clocked from the input bus into register R on the LOW-to-HIGH transition of CLK. Register F is clocked on the LOW-to-HIGH transition of CLK. Figure 5(b) depicts typical I/O timing in this mode.

When placed in this I/O mode, the data path will not function properly if the R and S registers are made transparent. Therefore, input register feedthrough control FT_0 must be held LOW in this mode.



BD007060

Figure 3. Functional Block Diagram for the 32-Bit, Single-Input Bus Mode

16-Bit, Two-Input Bus Mode

In this I/O mode, the R and S buses are configured as independent 16-bit input buses, and the F bus is configured as a 16-bit output bus. Figure 4 is a functional block diagram of the Am29325 in this I/O mode. Note that the 16 least-significant bits (LSBs) and 16 most-significant bits (MSBs) of the R, S, and F buses must be wired to their respective system buses in parallel.

Thirty-two-bit operands are passed along the 16-bit data buses by time-multiplexing the 16 LSBs and 16 MSBs of each 32-bit word. For the R input bus, the host system multiplexes the 16 LSBs and 16 MSBs of the R operand onto the 16-bit R bus. The 16 LSBs of the R operand are stored in a temporary holding register on the HIGH-to-LOW transition of CLK. The 16 MSBs are clocked into register R on the LOW-to-HIGH transition of CLK; at the same time, the 16 LSBs are transferred from the temporary holding register to register R. Transfer of data from the S input bus to the S register takes place in a similar fashion. Register F is clocked on the LOW-to-HIGH transition of CLK. Circuitry internal to the Am29325 multiplexes data from register F onto the 16-bit output bus by enabling the 16 LSBs of the F output bus when CLK is HIGH, and enabling the 16 MSBs of the F output bus when CLK is LOW. Figure 5(c) depicts typical I/O timing in this mode.

When placed in this I/O mode, the data path will not function properly if the R and S registers are made transparent. Therefore, input register feedthrough control FT₀ must be held LOW in this mode. Caution must also be taken in controlling the register R input multiplexer control line, I₄, in this I/O mode. I₄ should be changed only when CLK is HIGH, in

addition to meeting the setup and hold time requirements given in the **Switching Characteristics** section.

Operation in IEEE Mode

When input signal IEEE/ \overline{DEC} is HIGH, the IEEE mode of operation is selected. In this mode the Am29325 uses the floating-point format set forth in the IEEE Proposed Standard for Binary Floating-Point Arithmetic, P754. In addition, the IEEE mode complies with most other aspects of single-precision floating-point operation outlined in the proposed standard — differences are discussed in **Appendix A**.

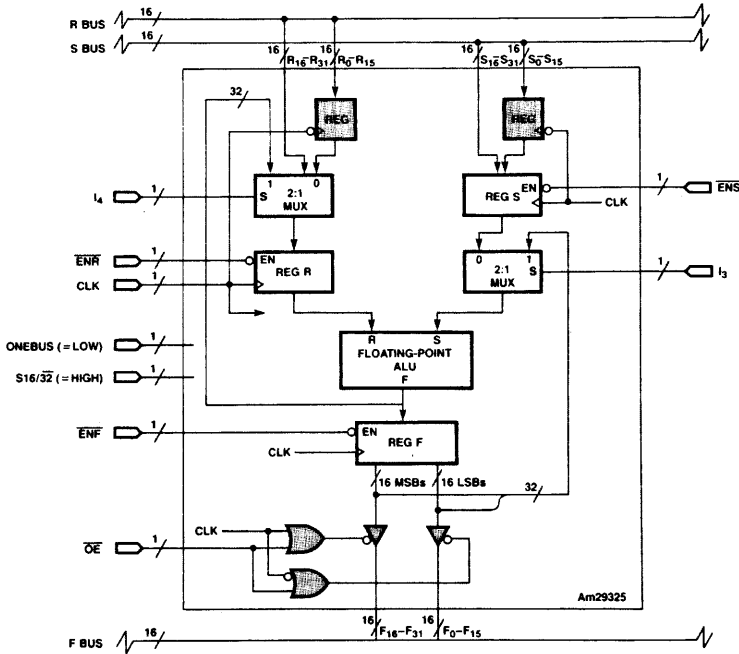
IEEE Floating-Point Format

The IEEE single-precision floating-point word is 32 bits wide, and is arranged in the format shown in Figure 6. The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 23-bit fraction.

The sign bit indicates the sign of the floating-point number's value. Non-negative values have a sign of 0; negative values, a sign of 1. The value zero may have either sign.

The biased exponent is an 8-bit unsigned integer field representing a multiplicative factor of some power of two. The bias value is 127. If, for example, the multiplicative factor for a floating-point number is to be 2^a , the value of the biased exponent would be a + 127; "a" is called the true exponent.

The fraction is a 23-bit unsigned fraction field containing the 23 LSBs of the floating-point number's 24-bit mantissa. The weight of fraction's MSB is 2^{-1} ; the weight of the LSB is 2^{-23} .



BD007070

Figure 4. Functional Block Diagram for the 16-Bit, Two-Input Bus Mode

A floating-point number is evaluated or interpreted per the following conventions:

let s = sign bit
 e = biased exponent
 f = fraction

if $e = 0$ and $f = 0$...value = $(-1)^s \cdot 0$ (+ 0, -0)

if $e = 0$ and $f \neq 0$...value = denormalized number

if $0 < e < 255$...value = $(-1)^s \cdot (2^{e-127}) \cdot (1.f)$
(normalized number)

if $e = 255$ and $f = 0$...value = $(-1)^s \cdot (\infty)$ (+ ∞ , - ∞)

if $e = 255$ and $f \neq 0$...value = not-a-number (NaN)

Zero: The value zero can have either a positive or negative sign. Rules for determining the sign of a zero produced by an operation are given in the **Sign Bit** section.

Denormalized Number: A denormalized number represents a quantity with magnitude less than 2^{-126} but greater than zero.

Normalized Number: A normalized number represents a quantity with magnitude greater than or equal to 2^{-126} but less than 2^{128} .

Example 1:

The number +3.5 can be represented in floating-point format as follows:

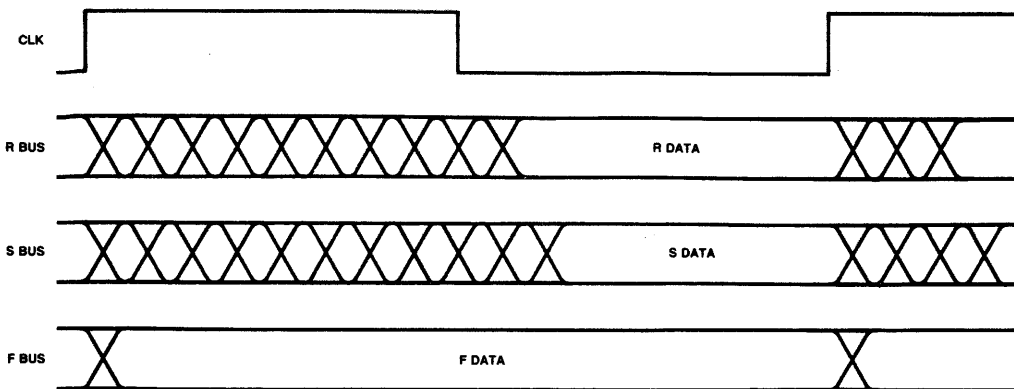
$$\begin{aligned} +3.5 &= 11.1_2 \times 2^0 \\ &= 1.11_2 \times 2^1 \end{aligned}$$

sign = 0

$$\begin{aligned} \text{biased exponent} &= 1_{10} + 127_{10} = 128_{10} \\ &= 10000000_2 \end{aligned}$$

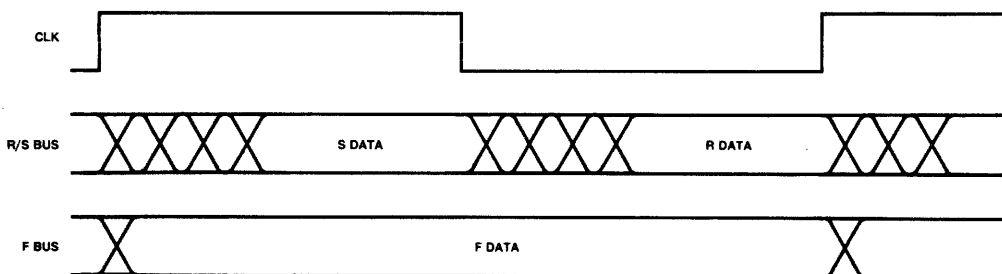
$$\begin{aligned} \text{fraction} &= 11000000000000000000000_2 \\ &\text{(the leading 1 is implied in the format)} \end{aligned}$$

Concatenating these fields produces the floating-point word 40600000₁₆.



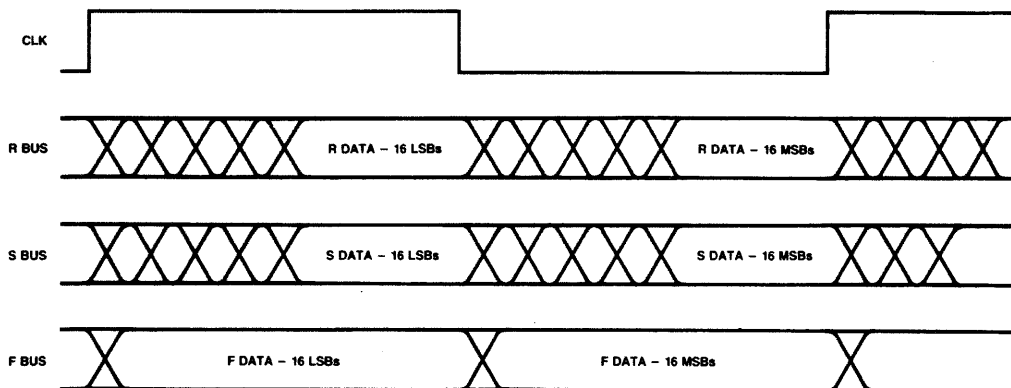
WF023730

a) 32-Bit, Two-Input-Bus Mode



WF023740

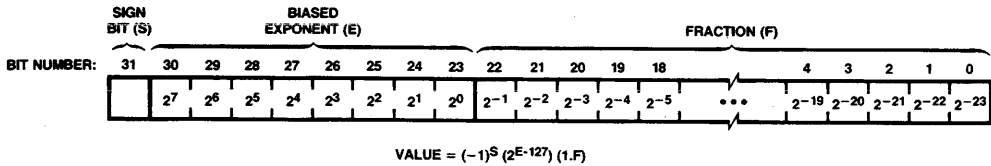
b) 32-Bit, Single-Input-Bus Mode



WF023750

c) 16-Bit, Two-Input-Bus Mode

Figure 5. Typical Bus Timing for the I/O Modes with $FT_0 = \text{LOW}$, $FT_1 = \text{LOW}$



TB000640

Figure 6. IEEE Mode Single-Precision Floating-Point Format

Example 2:

The number -11.375 can be represented in floating-point format as follows:

$$\begin{aligned}
 -11.375 &= -1011.011_2 \times 2^0 \\
 &= -1.011011_2 \times 2^3
 \end{aligned}$$

sign = 1

$$\begin{aligned}
 \text{biased exponent} &= 3_{10} + 127_{10} = 130_{10} \\
 &= 10000010_2
 \end{aligned}$$

fraction = 011011000000000000000000₂
 (the leading 1 is implied in the format)

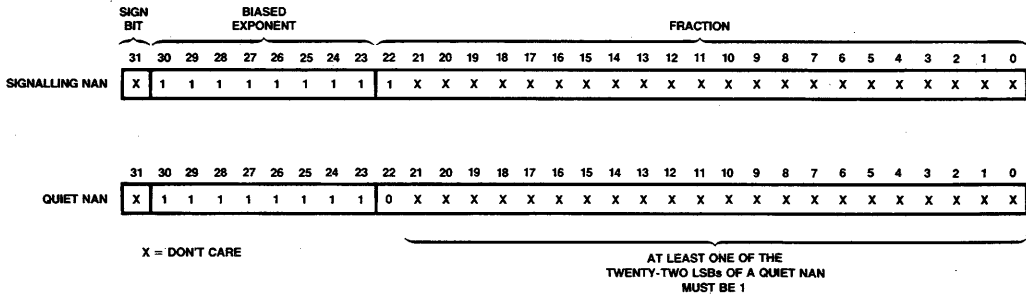
Concatenating these fields produces the floating-point word C1360000₁₆.

Infinity: Infinity can have either a positive or negative sign. The way in which infinities are interpreted is determined by the state of the projective/affine mode select, PROJ/AFF.

Not-a-Number: A not-a-number, or NAN, does not represent a numeric value, but is interpreted as a signal or symbol. NANs are used to indicate invalid operations, and as a means of passing process status information through a series of calculations. NANs arise in two ways: 1) they can be generated by the Am29325 to indicate that an invalid operation has taken place (e.g., $\infty \times 0$), or 2) be provided by the user as an input operand. There are two types of NANs, signalling and quiet (see Figure 7 for formats).

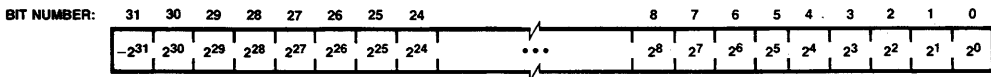
IEEE Mode Integer Format

Integer numbers are represented as 32-bit, two's-complement words (Figure 8 depicts the integer format). The integer word can represent a range of integer values from -2^{31} to $2^{31} - 1$.



TB000650

Figure 7. Signalling and Quiet NAN Formats



TB000660

Figure 8. 32-Bit Integer Format

Operations

All eight floating-point ALU operations discussed in the **Functional Description** section can be performed in IEEE mode. Various exceptional aspects of the R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, and FP-TO-INT operations for this mode are described below. The IEEE-TO-DEC and DEC-TO-IEEE operations are discussed separately in the **IEEE-TO-DEC AND DEC-TO-IEEE Operations** section.

Operations with NANs: NANs arise in two ways: 1) they can be generated by the Am29325 to indicate that an invalid operation has taken place (e.g., $\infty \times 0$), or 2) be provided by the user as an input operand. There are two types of NANs, signalling and quiet (see Figure 7 for formats).

Signalling NANs set the invalid operation flag when they appear as an input operand to an operation. They are useful for indicating uninitialized variables, or for implementing user-

designed extensions to the operations provided. The ALU never produces a signalling NAN as the final result of an operation.

Quiet NANs are generated for invalid operations. When they appear as an input operand, they are passed through most operations without setting the invalid flag, the floating-point-to-integer conversion operation being the exception.

The sign of any input operand NAN is ignored. All quiet NANs produced as the final result of an operation have a sign of 0.

When a NAN appears as an input operand, the final result of the operation is a quiet NAN that is created by taking the input NAN and forcing bit 22 LOW and bit 21 HIGH. If an operation has two NANs as input operands, the resulting quiet NAN is created using the NAN on the R port.

When a quiet NAN is produced as the final result of an invalid operation whose input operand or operands are not NANs, the resulting NAN will always have the value 7FA00000₁₆.

The NAN flag will be HIGH whenever an operation produces a NAN as a final result.

Example 1:

Suppose the floating-point addition operation is performed with the following input operands:

R port: 3F800000₁₆ (1.0*2⁰)
S port: 7FC12345₁₆ (signalling NAN)

Result: The signalling NAN on the S port is converted to a quiet NAN by forcing bit 22 LOW and bit 21 HIGH. The operation's final result will be 7FA12345₁₆. Since one of the two input operands is a signalling NAN, the invalid flag will be HIGH; the NAN flag will also be HIGH.

Example 2:

Suppose the floating-point multiplication operation is performed with the following input operands:

R port: FFF11111₁₆ (signalling NAN)
S port: 7FC22222₁₆ (quiet NAN)

Result: Since both input operands are NANs, the NAN on the R port is chosen for output. In addition to forcing bit 22 LOW, the sign bit (bit 31) is set LOW (bit 21 is already HIGH, and need not be changed). The operation's final result will be 7FB11111₁₆. Since one of the two input operands is a signalling NAN, the invalid flag is HIGH; the NAN flag will also be HIGH.

Example 3:

Suppose the floating-point subtraction operation is performed with the following input operands:

R port: FF800001₁₆ (quiet NAN)
S port: 7F800000₁₆ (+∞)

Result: To create the final result, the quiet NANs sign bit (bit 31) is forced LOW and bit 21 is forced HIGH (bit 22 is already LOW, and need not be changed). The final result will be 7FA00001₁₆. The NAN flag will be HIGH.

Operations with Denormalized Numbers: The proposed IEEE standard incorporates denormalized numbers to allow a means of gradual underflow for operations that produce non-zero results too small to be expressed as a normalized floating-point number. The Am29325 does not support gradual underflow. If a floating-point operation produces a non-zero rounded result that is not large enough to be expressed as a normalized floating-point number, the final result will be a zero

of the same sign; the inexact, underflow, and zero flags will be HIGH. If an input operand is a denormalized number, the floating-point ALU will assume that operand to be a zero of the same sign.

Operations Producing Overflows: If an operation has a finite input operand or operands, and if the operation produces a rounded result that is too large to fit in the destination format, the operation is said to have overflowed.

A floating-point overflow occurs if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation with finite input operand(s) produces a result which, after rounding, has a magnitude greater than or equal to 2¹²⁸. Positive or negative infinity will appear as the final result if the rounded result is positive or negative, respectively, and the overflow and inexact flags will be HIGH.

Integer overflow occurs when the floating-point-to-integer conversion operation attempts to convert a number which, after rounding, is greater than 2³¹ - 1 or less than -2³¹. The final result will be quiet NAN 7FA00000₁₆, and the invalid operation and NAN flags will be HIGH. Note that the overflow and inexact flags remain LOW for integer overflow.

Operations Producing Underflows: If an operation produces a floating-point rounded result having a magnitude too small to be expressed as a normalized floating-point number, but greater than zero, that operation is said to have underflowed. Underflow occurs when an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has a magnitude in the range:

$$0 < \text{magnitude} < 2^{-126}$$

In such cases, the final result will be +0 (00000000₁₆) if the rounded result is non-negative, and -0 (80000000₁₆) if the rounded result is negative. The underflow, inexact, and zero flags will be HIGH.

Underflow does not occur if the destination format is integer. If the infinitely precise result of a floating-point-to-integer conversion has a magnitude greater than 0 and less than 1, but the rounded result is 0, the underflow flag remains LOW.

Operations with Infinities: In most cases, positive and negative infinity are valid inputs for the R PLUS S, R MINUS S, R TIMES S, and 2 MINUS S operations. Those cases for which infinities are not valid inputs for these operations are listed in Table 4.

Infinities in IEEE mode can be handled either as projective or affine. The projective mode is selected when PROJ/∞FF is HIGH; the affine mode is selected when PROJ/∞FF is LOW. The only differences between the modes that are relevant to Am29325 operation occur during the addition and subtraction of infinities:

Operation	Affine Mode	Projective Mode
(+∞) + (+∞)	Output +∞	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags
(-∞) + (-∞)	Output -∞	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags
(+∞) - (-∞)	Output +∞	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags
(-∞) - (+∞)	Output -∞	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags

If an R PLUS S, R MINUS S, or 2 MINUS S operation has infinity as an input operand or operands, the final result, if valid, is presumed to be exact. For example, adding $+\infty$ and 2.0 will produce a final result of $+\infty$; since the result is considered exact, the inexact flag remains LOW.

Invalid Operations: If an input operand is invalid for the operation to be performed, that operation is considered invalid. When an invalid operation is performed, the floating-point ALU produces a quiet NAN as the final result, and the invalid operation flag goes HIGH. Table 4 lists the cases for which the invalid flag is HIGH in IEEE mode, and the final results produced for these operations.

TABLE 4. IEEE MODE INVALID OPERATIONS

Operation	Input Operand	Final Result
R PLUS S	$(+\infty) + (-\infty)$ or $(-\infty) + (+\infty)$	7FA00000 ₁₆ (quiet NAN)
R PLUS S	$(+\infty) + (+\infty)$ or $(-\infty) + (-\infty)$ (Note 1)	7FA00000 ₁₆ (quiet NAN)
R MINUS S	$(+\infty) - (+\infty)$ or $(-\infty) - (-\infty)$	7FA00000 ₁₆ (quiet NAN)
R MINUS S	$(+\infty) - (-\infty)$ or $(-\infty) - (+\infty)$ (Note 1)	7FA00000 ₁₆ (quiet NAN)
R TIMES S	$(+0) * (+\infty)$ or $(+0) * (-\infty)$ or $(-0) * (+\infty)$ or $(-0) * (-\infty)$	7FA00000 ₁₆ (quiet NAN)
R PLUS S R MINUS S R TIMES S	R or S is a signalling NAN	(Note 2)
2 MINUS S	S is a signalling NAN	(Note 2)
FP-TO-INT	R is a signalling or quiet NAN	(Note 2)
FP-TO-INT	$R > 2^{31} - 1$ or $R < -(2^{31})$	7FA00000 ₁₆ (quiet NAN)

Notes: 1. These cases are invalid in projective mode only.
2. Results for these operations are described in the **Operations with NANs** section.

The Sign Bit

For most floating-point operations, the sign bit of the final result is unambiguous; i.e., there is only one sign bit value that yields a numerically correct result. Operations that produce an infinitely precise result of zero, however, present a problem, as the IEEE floating-point format allows for representation of both +0 and -0. The following rules can be used to determine the signs of zero produced in such cases.

R PLUS S: The operations $+x + (-x)$ and $-x + (+x)$ produce a final result of zero; the sign of the zero is dependent on the rounding mode:

Operations $+0 + (-0)$ and $-0 + (+0)$ produce a result of 0, with the sign of the result determined by the table above.

The operation $+0 + (+0)$ produces a final result of +0; the operation $-0 + (-0)$ produces a final result of -0.

R MINUS S: The operations $+x - (+x)$ and $-x - (-x)$ produce a final result of zero; the sign of the zero is dependent on the rounding mode:

Rounding Mode	Sign of Result
Round to nearest	0
Round toward $-\infty$	1
Round toward $+\infty$	0
Round toward 0	0

Operations $+0 - (+0)$ and $-0 - (-0)$ produce a result of 0, with the sign of the result determined by the table above.

The operation $+0 - (-0)$ produces a final result of +0; the operation $-0 - (+0)$ produces a final result of -0.

R TIMES S: The sign of any multiplication result other than a NAN is the exclusive OR of the signs of the input operands. Therefore, if x is non-negative, +0 times +x produces a final result of +0, +0 times -x produces a final result of -0, -0 times +x produces a final result of -0, -0 times -x produces a final result of +0.

2 MINUS S: If S equals 2, the final result is -0 for the round toward $-\infty$ mode, and +0 for all other rounding modes.

Rounding

Rounding is performed whenever an operation produces an infinitely precise result that cannot be represented exactly in the destination format. For example, suppose a floating-point operation produces the infinitely precise result:

$$1.101010101010101010101010101010101 \setminus 01 \times 2^3.$$

In this example, the fraction portion of the mantissa has 25 bits; the IEEE floating-point format can accommodate only 23. The backslash (\) in the mantissa represents the boundary between the first 23 bits of the fraction and any remaining bits. Rounding is the process by which this result is approximated by a representation that fits the destination format.

There are four rounding modes in IEEE mode: 1) round to nearest, 2) round toward $+\infty$, 3) round toward $-\infty$, and 4) round toward 0. The rounding mode is chosen using the rounding mode select lines, RND₀ and RND₁. Table 5 lists the select states needed to obtain the desired rounding mode.

TABLE 5. ROUNDING MODE SELECT

Rounding Mode	RND ₁	RND ₀	Rounding Mode
Round to nearest	0	0	Round to nearest
Round toward $-\infty$	0	1	Round toward $-\infty$
Round toward $+\infty$	1	0	Round toward $+\infty$
Round toward 0	1	1	Round toward 0

Round to Nearest: In this rounding mode the infinitely precise result of an operation is rounded to the closest representation that fits in the destination format. If the infinitely precise result is exactly halfway between two representations, it is rounded to the representation having an LSB of zero. Rounding is performed both for floating-point and integer destination formats.

Figure 9 illustrates four examples of the round-to-nearest process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 9(a), the infinitely precise result of an operation is:
 $2^{20} + 2^{-4} + 2^{-5} = 1.00000000000000000000000000011 \times 2^{20}$

The result is rounded to the closest representable floating-point value,
 $2^{20} + 2^{-3} = 1.0000000000000000000000000001 \times 2^{20}$

Example 2:

In Figure 9(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.11111111111111111111111111\0001 \times 2^{19}$$

This result is rounded to the closest representable floating-point value,

$$2^{20} - 2^{-4} = 1.11111111111111111111111111 \times 2^{19}$$

Example 3:

In Figure 9(c), the infinitely precise result of an operation is:

$$-(2^{20} + 2^{-3} + 2^{-4}) = -1.00000000000000000000000000001 \times 2^{20}$$

This result is exactly halfway between two representable floating-point values. Accordingly, it is rounded to the closest representation with an LSB of zero, or

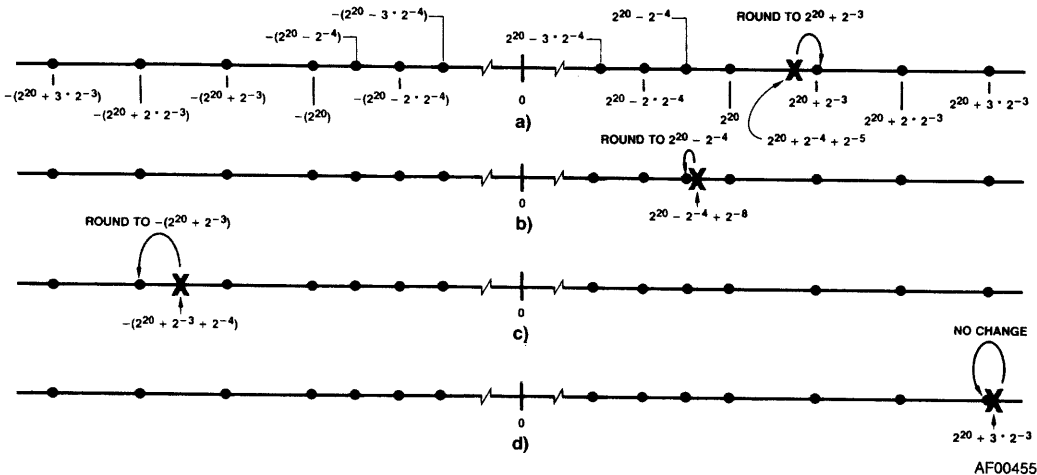
$$-(2^{20} + 2^{-3}) = -1.0000000000000000000000000001 \times 2^{20}$$

Example 4:

In Figure 9(d), the infinitely precise result of an operation is:

$$2^{20} + 3 \cdot 2^{-3} = 1.00000000000000000000000000011 \times 2^{20}$$

This result can be represented exactly in the floating-point format, and is left unaltered by the rounding process.



AF004550

Figure 9. Floating-Point Rounding Examples for Round-to-Nearest Mode

Figure 10 illustrates four examples of the round-to-nearest process for operations having an integer destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the integer format.

Example 1:

In Figure 10(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to the closest representable integer value,

$$2^{10} = 00...010000000000$$

Example 2:

In Figure 10(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...010000000001.001$$

This result is rounded to the closest representable integer value,

$$2^{10} + 2^0 = 00...010000000001$$

Example 3:

In Figure 10(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = -11...101111111110.1$$

This result is exactly halfway between two representable integer values. Accordingly, it is rounded to the closest representation with an LSB of zero, or

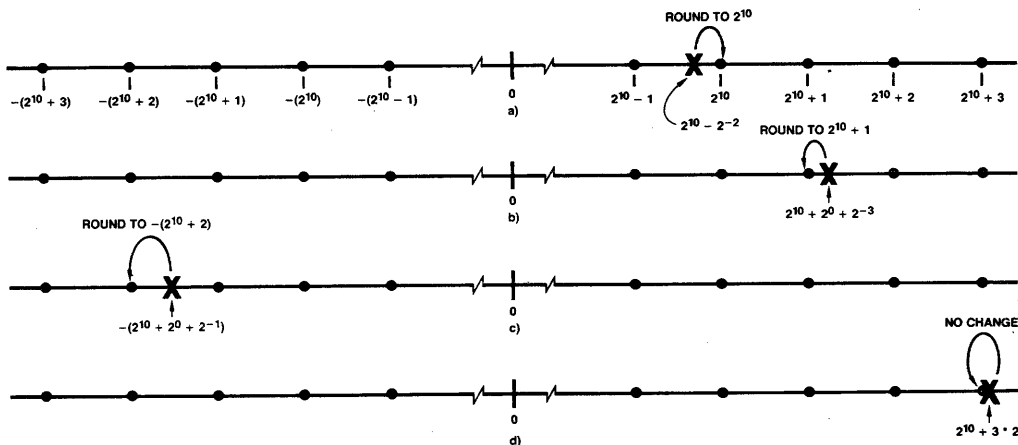
$$-(2^{10} + 2^*2^0) = 11...1011111111110$$

Example 4:

In Figure 10(d), the infinitely precise result of an operation is:

$$2^{10} + 3*2^0 = 00...010000000011$$

This result can be represented exactly in the integer format, and is left unaltered by the rounding process.



AF004560

Figure 10. Integer Rounding Examples for Round-to-Nearest Mode

Round Toward $-\infty$: In this rounding mode the result of an operation is rounded to the closest representation that is less than or equal to the infinitely precise result, and which fits the destination format. Rounding is performed both for floating-point and integer destination formats.

Figure 11 illustrates four examples of the round toward $-\infty$ process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 11(a), the infinitely precise result of an operation is:
 $2^{20} + 2^{-4} + 2^{-5} = 1.00000000000000000000000000000000111 \times 2^{20}$

This result cannot be represented exactly in floating-point format, and is rounded to the next-smaller floating-point representation:

$$2^{20} = 1.00000000000000000000000000000000 \times 2^{20}$$

Example 2:

In Figure 11(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.11111111111111111111111111111111 \backslash 0001 \times 2^{19}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-smaller floating point representation:

$$2^{20} - 2^{-4} = 1.1111111111111111111111111111111 \times 2^{19}$$

Example 3:

In Figure 11(c), the infinitely precise result of an operation is:

$$-(2^{20} + 2^{-3} + 2^{-4}) = -1.000000000000000000000000000000001 \backslash 1 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-smaller floating-point representation.

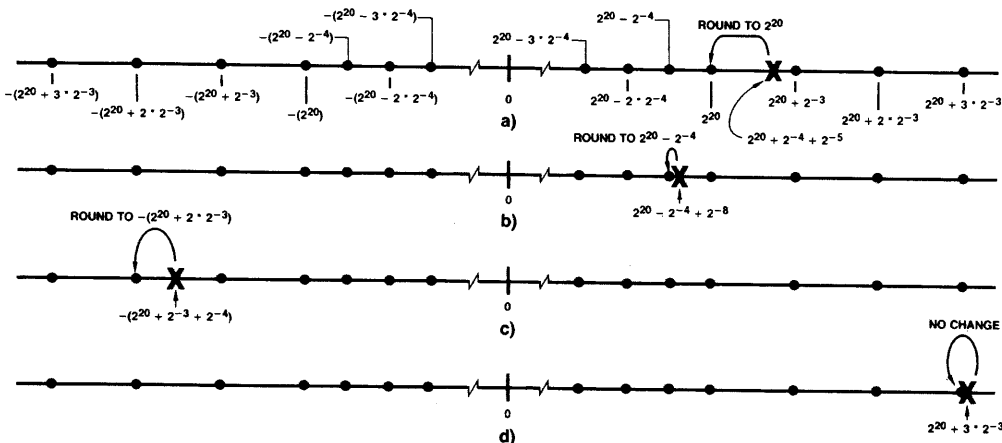
$$-(2^{20} + 2^{-3}) = -1.00000000000000000000000000000010 \times 2^{20}$$

Example 4:

In Figure 11(d), the infinitely precise result of an operation is:

$$2^{20} + 3 \cdot 2^{-3} = 1.00000000000000000000000000000011 \times 2^{20}$$

This result can be represented exactly in the floating-point format, and is left unaltered by the rounding process.



AF004510

Figure 11. Floating-Point Rounding Examples for Round Toward $-\infty$ Mode

Figure 12 illustrates four examples of the round toward $-\infty$ process for operations having an integer destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be exactly represented in the integer format.

Example 1:

In Figure 12(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to the next-smaller representable integer value,

$$2^{10} - 2^0 = 00...001111111111$$

Example 2:

In Figure 12(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...010000000001.001$$

This result is rounded to the next-smaller representable integer value,

$$2^{10} + 2^0 = 00...010000000001$$

Example 3:

In Figure 12(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = 11...1011111111110.1$$

This result is rounded to the next-smaller representable integer value:

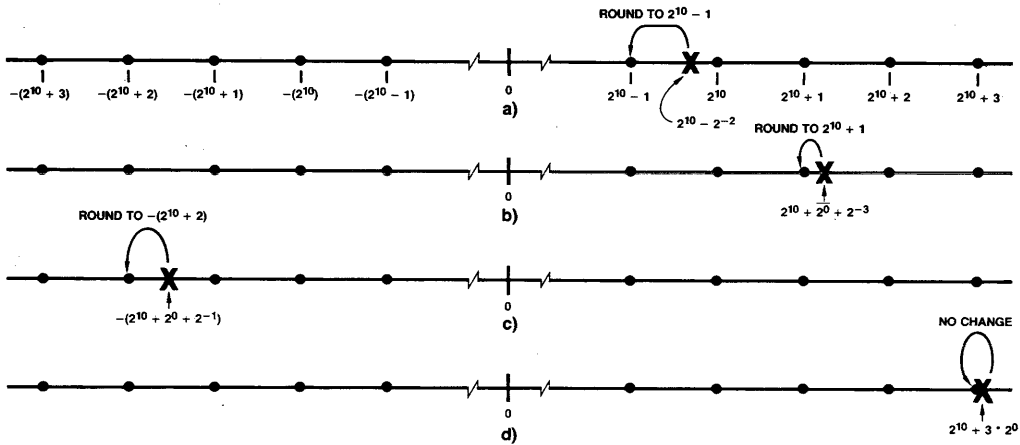
$$-(2^{10} + 2 \cdot 2^0) = 11...1011111111110$$

Example 4:

In Figure 12(d), the infinitely precise result of an operation is:

$$2^{10} + 3 \cdot 2^0 = 00...010000000011$$

This result can be represented exactly in the integer format, and is unaltered by the rounding process.



AF004580

Figure 12. Integer Rounding Examples for Round Toward $-\infty$ Mode

Round Toward +∞: In this rounding mode the result of an operation is rounded to the closest representation that is greater than or equal to the infinitely precise result, and which fits the destination format. Rounding is performed both for floating-point and integer destination formats.

Figure 13 illustrates four examples of the round toward +∞ process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 13(a), the infinitely precise result of an operation is:

$$2^{20} + 2^{-4} + 2^{-5} = 1.00000000000000000000000000011 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-larger floating-point representation:

$$2^{20} + 2^{-3} = 1.0000000000000000000000000001 \times 2^{20}$$

Example 2:

In Figure 13(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.11111111111111111111111111111 \times 2^{19}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-larger floating point representation:

$$2^{20} = 1.0000000000000000000000000000 \times 2^{20}$$

Example 3:

In Figure 13(c), the infinitely precise result of an operation is:

$$-(2^{20} + 2^{-3} + 2^{-4}) = -1.00000000000000000000000000011 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-larger floating-point representation.

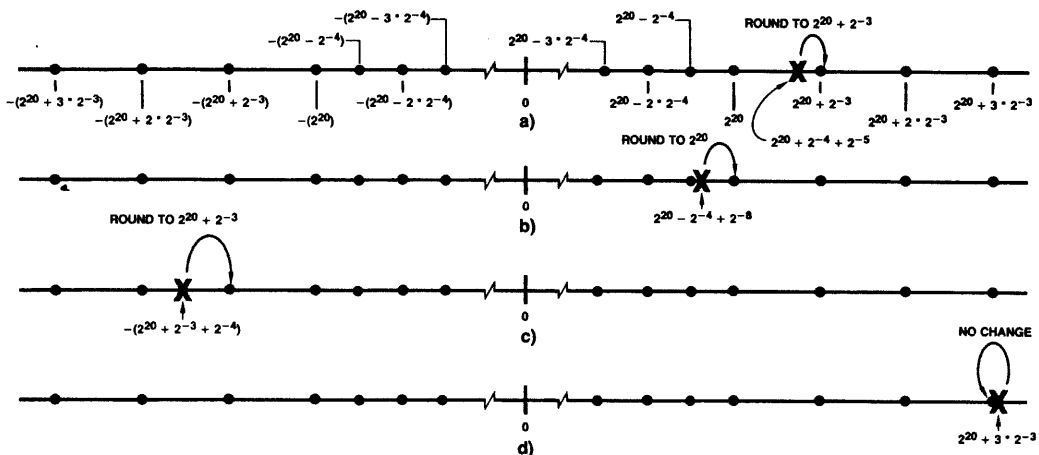
$$-(2^{20} + 2^{-3}) = -1.0000000000000000000000000001 \times 2^{20}$$

Example 4:

In Figure 13(d), the infinitely precise result of an operation is:

$$2^{20} + 3 \cdot 2^{-3} = 1.00000000000000000000000000011 \times 2^{20}$$

This result can be represented exactly in the floating-point format—no rounding takes place



AF004590

Figure 13. Floating-Point Rounding Examples for Round Toward +∞ Mode

Figure 14 illustrates four examples of the round toward $+\infty$ process for having an integer destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be exactly represented in the integer format.

Example 1:

In Figure 14(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to the next-larger representable integer value,

$$2^{10} = 00...010000000000$$

Example 2:

In Figure 14(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...010000000001.001$$

This result is rounded to the next-larger representable integer value,

$$2^{10} + 2^0 = 00...010000000010$$

Example 3:

In Figure 14(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = 11.1011111111110.1$$

This result is rounded to the next-larger representable integer value:

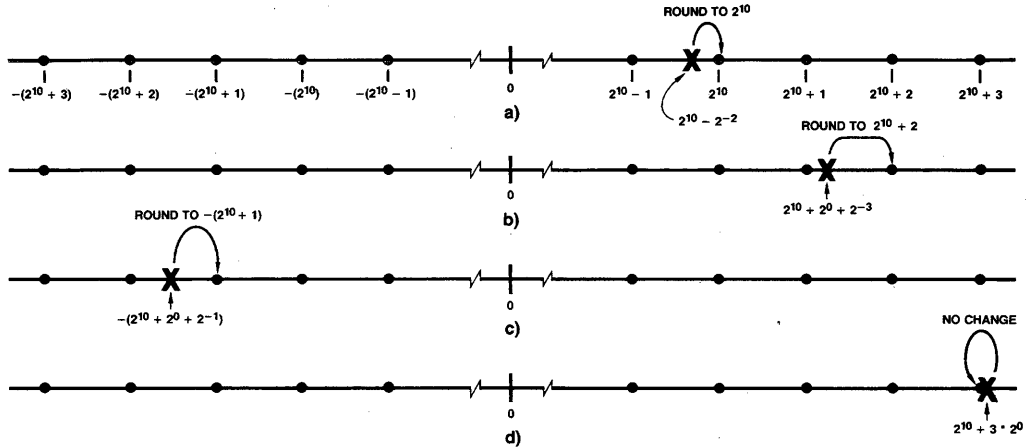
$$-(2^{10} + 2^0) = 11...1011111111110$$

Example 4:

In Figure 14(d), the infinitely precise result of an operation is:

$$2^{10} + 3 \cdot 2^0 = 00...010000000011$$

This result can be represented exactly in the integer format — no rounding takes place.



AF004600

Figure 14. Integer Rounding Examples for Round Toward $+\infty$ Mode

Round Toward 0: In this rounding mode the result of an operation is rounded to the closest representation whose magnitude is less than or equal to the infinitely precise result, and which fits the destination format. Rounding is performed both for floating-point and integer destination formats.

Figure 15 illustrates four examples of the round toward 0 process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 15(a), the infinitely precise result of an operation is:

$$2^{20} + 2^{-4} + 2^{-5} = 1.0000000000000000000000000000000111 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to:

$$2^{20} = 1.00000000000000000000000000000000 \times 2^{20}$$

Example 2:

In Figure 15(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.1111111111111111111111111111110001 \times 2^{19}$$

This result cannot be represented exactly in floating-point format, and is rounded to:

$$2^{20} - 2^{-4} = 1.111111111111111111111111111111 \times 2^{19}$$

Example 3:

In Figure 15(c), the infinitely precise result of an operation is:

$$-(2^{20} + 2^{-3} + 2^{-4}) = -1.000000000000000000000000000000011 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to:

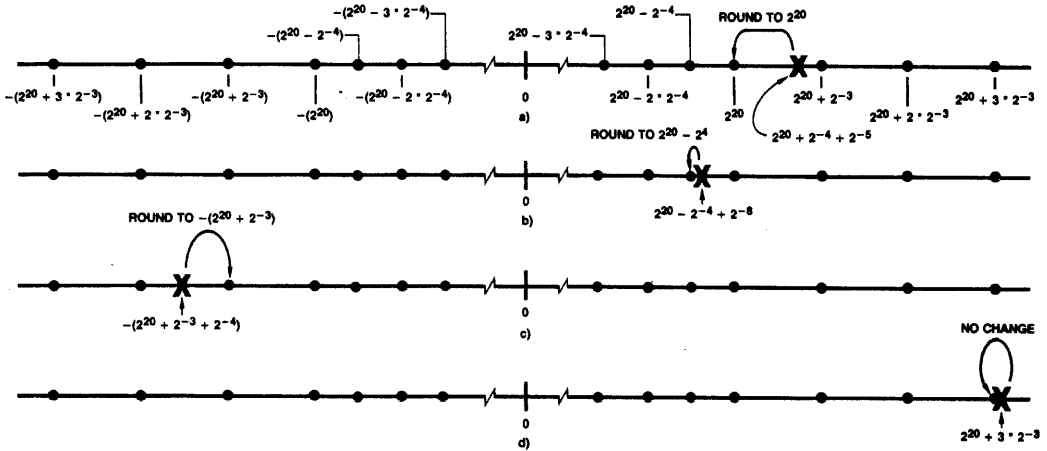
$$-(2^{20} + 2^{-3}) = -1.00000000000000000000000000000001 \times 2^{20}$$

Example 4:

In Figure 15(d), the infinitely precise result of an operation is:

$$2^{20} + 3 \cdot 2^{-3} = 1.00000000000000000000000000000011 \times 2^{20}$$

This result can be represented exactly in the floating-point format, and is unaffected by the rounding process.



AF004610

Figure 15. Floating-Point Rounding Examples for Round Toward 0 Mode

Figure 16 illustrates four examples of the round toward 0 process for operations having an integer destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be exactly represented in the integer format.

Example 1:

In Figure 16(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to:

$$2^{10} - 2^0 = 00...001111111111$$

Example 2:

In Figure 16(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...010000000001.001$$

The result is rounded to:

$$2^{10} + 2^0 = 00...010000000001$$

Example 3:

In Figure 16(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = 11...101111111110.1$$

The result is rounded to:

$$-(2^{10} + 2^0) = 11...101111111110$$

Example 4:

In Figure 16(d), the infinitely precise result of an operation is:

$$2^{10} + 3 \cdot 2^0 = 00...010000000011$$

This result can be represented exactly in the integer format, and is unaffected by the rounding process.

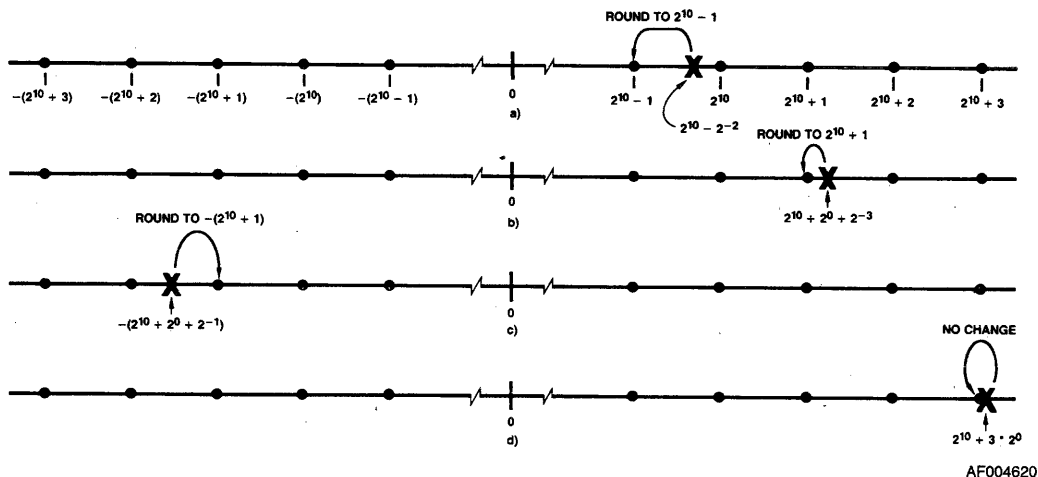


Figure 16. Integer Rounding Examples for Round Toward 0 Mode

Flag Operation

The Am29325 generates six status flags to monitor floating-point processor operation. The following is a summary of flag conventions in IEEE mode:

Invalid Operation Flag: The invalid operation flag is HIGH when an input operand is invalid for the operation to be performed. Table 4 lists the cases for which the invalid operation flag is HIGH in IEEE mode, and the corresponding final result. In cases where the invalid operation flag is HIGH, the overflow, underflow, zero, and inexact flags are LOW; the NAN flag will be HIGH.

Overflow Flag: The overflow flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation with finite input operand(s) produces a result which, after rounding, has a magnitude greater than or equal to 2^{126} . The final result will be $+\infty$ or $-\infty$.

Underflow Flag: The underflow flag is HIGH if an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has a magnitude in the range: $0 < \text{magnitude} < 2^{-126}$.

The final result will be $+0$ (00000000_{16}) if the rounded result is non-negative, and -0 (80000000_{16}) if the rounded result is negative.

Inexact Flag: The inexact flag is HIGH if the final result of an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, or FP-TO-INT operation is not equal to the infinitely precise result. Note that if the underflow or overflow flag is HIGH, the inexact flag will also be HIGH.

Zero Flag: The zero flag is HIGH if the final result of an operation is zero. For operations producing an IEEE floating-point number, the flag accompanies outputs $+0$ (00000000_{16}) and -0 (80000000_{16}). For operations producing an integer, the flag accompanies the output 0 (00000000_{16}).

NAN Flag: The NAN flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, or FP-TO-INT operation produces a NAN as a final result.

Operation in DEC Mode

When input signal $\overline{\text{IEEE/DEC}}$ is LOW, the DEC mode of operation is selected. In this mode the Am29325 uses the single-precision floating-point format (floating F) set forth in

Digital Equipment Corporation's VAX Architecture Manual. In addition, the DEC mode complies with most other aspects of single-precision floating-point operation outlined in the manual — differences are discussed in **Appendix B**.

DEC Floating-Point Format

The DEC single-precision floating-point word is 32 bits wide, and is arranged in the format shown in Figure 17. The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 23-bit fraction.

The sign bit indicates the sign of the floating-point number's value. Non-negative values have a sign of 0, negative values a sign of 1.

The biased exponent is an 8-bit unsigned integer field representing a multiplicative factor of some power of two. The bias value is 128. If, for example, the multiplicative factor for a floating-point number is to be 2^a , the value of the biased exponent would be a +128; "a" is called the true exponent.

The fraction is a 23-bit unsigned fractional field containing the 23 LSBs of the floating-point number's 24-bit mantissa. The weight of this field's MSB is 2^{-2} ; the weight of the LSB is 2^{-24} .

A floating-point number is evaluated or interpreted per the following conventions:

- let s = sign bit
- e = biased exponent
- f = fraction

- if e = 0 and s = 0...value = 0
- if e = 0 and s = 1...value = DEC-reserved operand
- if $0 < e \leq 255$...value = $(-1)^s \cdot (2^e - 128) \cdot (.1f)$
(normalized number)

Zero: The value zero always has a sign of zero.

DEC-Reserved Operand: A DEC-reserved operand does not represent a numeric value, but is interpreted as a signal or symbol. DEC-reserved operands are used to indicate invalid operations and operations whose results have overflowed the destination format. They may also be used to pass symbolic information from one calculation to another.

Normalized Number: A normalized number represents a quantity with magnitude greater than or equal to 2^{-128} but less than 2^{127} .

Example 1:

The number +3.5 can be represented in floating-point format as follows:

$$+3.5 = 11.12 \times 2^0 = .1112 \times 2^2$$

sign = 0

$$\text{biased exponent} = 2_{10} + 128_{10} = 130_{10} = 1000010_2$$

$$\text{fraction} = 110000000000000000000_2 \text{ (the leading 1 is implied in the format)}$$

Concatenating these fields produces the floating-point word 41600000₁₆.

Example 2:

The number -11.375 can be represented in floating-point format as follows:

$$-11.375 = -1011.0112 \times 2^0 = -.10110112 \times 2^4$$

sign = 1

$$\text{biased exponent} = 4_{10} + 128_{10} = 132_{10} = 1000100_2$$

$$\text{fraction} = 0110110000000000000000_2 \text{ (the leading 1 is implied in the format)}$$

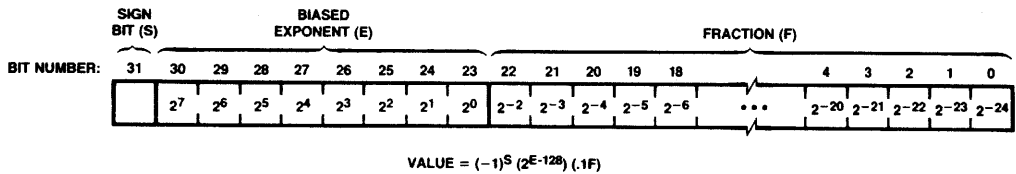
Concatenating these fields produces the floating-point word C2360000₁₆.

DEC Mode Integer Format

DEC mode integer format is identical to that of the IEEE mode. Integer numbers are represented as 32-bit, two's-complement words (Figure 8 depicts the integer format). The integer word can represent a range of integer values from -2^{31} to $2^{31} - 1$.

Operations

All eight floating-point ALU operations discussed in the **General Description** section can be performed in DEC mode.



TB000671

Figure 17. DEC-Mode Floating-Point Format

Various exceptional aspects of the R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, and FP-TO-INT operations for this mode are described below. The IEEE-TO-DEC and DEC-TO-IEEE operations are discussed separately in the **IEEE-TO-DEC and DEC-TO-IEEE Operations** section.

Operations with DEC-Reserved Operands: DEC-reserved operands arise in two ways: 1) they can be generated by the Am29325 to indicate that an invalid operation or floating-point

overflow has taken place, or 2) be provided by the user as an input operand.

When a DEC-reserved operand appears as an input operand, the final result of the operation is the same DEC-reserved operand. If an operation has two DEC-reserved operands as inputs, the DEC-reserved operand on the R port becomes the final result.

The NAN flag will be HIGH whenever an operation produces a DEC-reserved operand as a final result.

Example 1:

Suppose the floating-point addition operation is performed with the following input operands:

R port: 40800000₁₆ (0.1*2¹)
S port: 80012345₁₆ (DEC-reserved operand)

Result: This operation produces the DEC-reserved operand on the S port, 80012345₁₆, as the final result. The NAN flag will be HIGH.

Example 2:

Suppose the floating-point multiplication operation is performed with the following input operands:

R port: 80765432₁₆ (DEC-reserved operand)
S port: 80000001₁₆ (DEC-reserved operand)

Result: Since both input operands are DEC-reserved operands, the operand on the R port, 80765432₁₆, is the final result of the operation. The NAN flag will be HIGH.

Operations Producing Overflows: If an operation produces a rounded result that is too large to fit in the destination format, that operation is said to have overflowed.

A floating-point overflow occurs if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation with finite input operand(s) produces a result which, after rounding, has a magnitude greater than or equal to 2¹²⁷. The final result in such cases will be DEC-reserved operand 80000000₁₆; the overflow, inexact, and NAN flags will be HIGH.

Integer overflow occurs when the "floating-point-to-integer" conversion operation attempts to convert to integer a floating-point number which, after rounding, is greater than 2³¹ - 1 or less than -2³¹. The final result in such cases will be DEC-reserved operand 80000000₁₆; the invalid operation flag will be HIGH. Note that the overflow and inexact flags remain LOW for integer overflow.

Operations Producing Underflows: If an operation produces a floating-point result which, after rounding, has a magnitude too small to be expressed as a normalized floating-point number, but greater than 0, that operation is said to have underflowed. Underflow occurs when an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has the magnitude:

$$0 < \text{magnitude} < 2^{-128}$$

The final result in such cases will be 0 (00000000₁₆). The underflow, inexact, and zero flags will be HIGH.

Underflow does not occur if the destination format is integer. If the infinitely precise result of a floating-point-to-integer conversion has a magnitude greater than 0 and less than 1, but the rounded result is 0, the underflow flag remains LOW.

Invalid Operations: If an input operand is invalid for the operation to be performed, that operation is considered invalid. There is only one invalid operation in DEC mode: performing a floating-point-to-integer conversion on a value too large to be converted to an integer. In this case, the final result will be DEC-reserved operand 80000000₁₆, and the invalid operation and NAN flags will be HIGH.

Sign Bit

For all operations producing a DEC floating-point result, the sign bit of the final result is unambiguous; i.e., there is only one sign bit value that yields a numerically correct result.

Rounding

There are four rounding modes for DEC operation: 1) round to nearest, 2) round toward +∞, 3) round toward -∞, and 4) round toward 0. The round toward +∞, round toward -∞, and round toward 0 modes are performed in a manner identical to that for IEEE operation; refer to the **Rounding** section under **Operation in IEEE Mode**. The round to nearest mode is similar to that for IEEE operation, but differs in one respect: for the case in which the infinitely precise result of an operation is exactly halfway between two representable values, DEC round to nearest mode rounds to the value with the larger magnitude, rather than to the value whose LSB is 0.

Flag Operation

The Am29325 generates six status flags to monitor floating-point processor operation. The following is a summary of flag operation in DEC mode:

Invalid Operation Flag: The invalid operation flag is HIGH if the FP-TO-INT operation is performed on a floating-point number too large to be converted to an integer. The final result for such an operation will be the DEC-reserved operand 80000000₁₆.

Overflow Flag: The overflow flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation produces a result which, after rounding, has a magnitude greater than or equal to 2¹²⁷. The final result will be the DEC-reserved operand 80000000₁₆.

Underflow Flag: The underflow flag is HIGH if an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has a magnitude in the range:

$$0 < \text{magnitude} < 2^{-128}$$

The final result will be 0 (00000000₁₆) in such cases.

Inexact Flag: The inexact flag is HIGH if the final result of an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, or FP-TO-INT operation is not equal to the infinitely precise result. Note that if the underflow or overflow flag is HIGH, the inexact flag will also be HIGH.

Zero Flag: The zero flag is HIGH if the final result of an operation is 0. For operations producing an integer or a DEC floating-point number, the flag accompanies the output 0 (00000000₁₆). (It should be noted that any operation producing a floating-point 0 in DEC mode will output 00000000₁₆.)

NAN Flag: The NAN flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, or FP-TO-INT operation produces a DEC-reserved operand as the final result.

IEEE-TO-DEC and DEC-TO-IEEE Operations

The IEEE-TO-DEC and DEC-TO-IEEE operations are used to convert floating-point numbers between the IEEE and DEC formats. Both operations work in a manner independent of the IEEE/DEC mode control.

IEEE-TO-DEC Conversion

The operation converts an IEEE floating-point number to DEC floating-point format. Most conversions are exact; in no case does the round mode have any effect on the final result. There are, however, a few exceptional cases:

- a) If the IEEE floating-point input has a magnitude greater than or equal to 2¹²⁷, it is too large to be represented by a DEC floating-point number. The final result will be the DEC-reserved operand 80000000₁₆; the overflow, inexact, and NAN flags will be HIGH.

- b) If the IEEE floating-point input is a NAN, the final result will be the DEC-reserved operand 80000000₁₆; the invalid and NAN flags will be HIGH.
- c) If the IEEE floating-point input is a denormalized number, the final result will be a DEC 0 (00000000₁₆); the zero flag will be HIGH.
- d) If the IEEE floating-point input is +0 or -0, the final result will be a DEC 0 (00000000₁₆); the zero flag will be HIGH.

DEC-TO-IEEE Conversion

This operation converts a DEC floating-point number to IEEE floating-point format. Most conversions are exact; in no case does the round mode have any effect on the final result. There are, however, a few exceptional cases:

- a) If the DEC floating-point input is not 0, but has a magnitude less than 2^{-126} , it is too small to be expressed as a normalized IEEE floating-point number. The final result will be an IEEE floating-point 0 having the same sign as the input (00000000₁₆ for positive inputs and 80000000₁₆ for negative inputs); the underflow, inexact, and zero flags will be HIGH.
- b) If the DEC floating-point input is a DEC-reserved operand, the result will be quiet NAN 7FA00000₁₆; the invalid operation and NAN flags will be HIGH.
- c) If the DEC floating-point input is 0, the final result will be IEEE floating-point +0 (00000000₁₆); the zero flag will be HIGH.

APPLICATIONS

Suggestions for Power and Ground Pin Connections

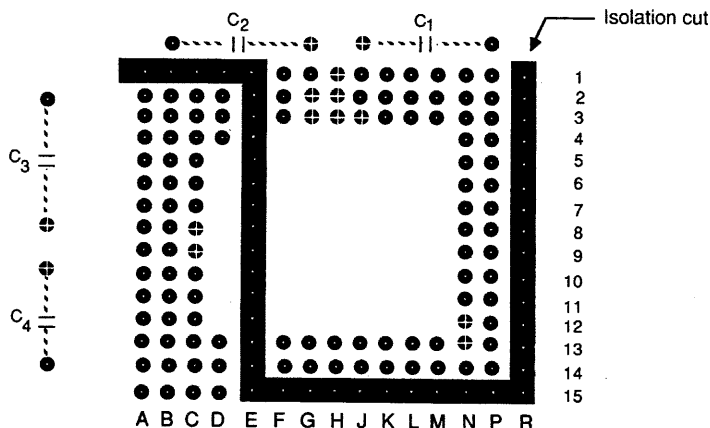
The Am29325 operates in an environment of fast signal rise times and substantial switching currents. Therefore, care must

be exercised during circuit board design and layout, as with any high-performance component. The following is a suggested layout, but since systems vary widely in electrical configuration, an empirical evaluation of the intended layout is recommended.

The V_{CC}T and GNDT pins, which carry output driver switching currents, tend to be electrically noisy. The V_{CC}E and GNDE pins, which supply the ECL core of the device, tend to produce less noise, and the circuits they supply may be adversely affected by noise spikes on the V_{CC}E plane. For this reason, it is best to provide isolation between the V_{CC}E and V_{CC}T pins, as well as independent decoupling for each. Isolating the GNDE and GNDT pins is not required.

Printed Circuit-Board Layout Suggestions

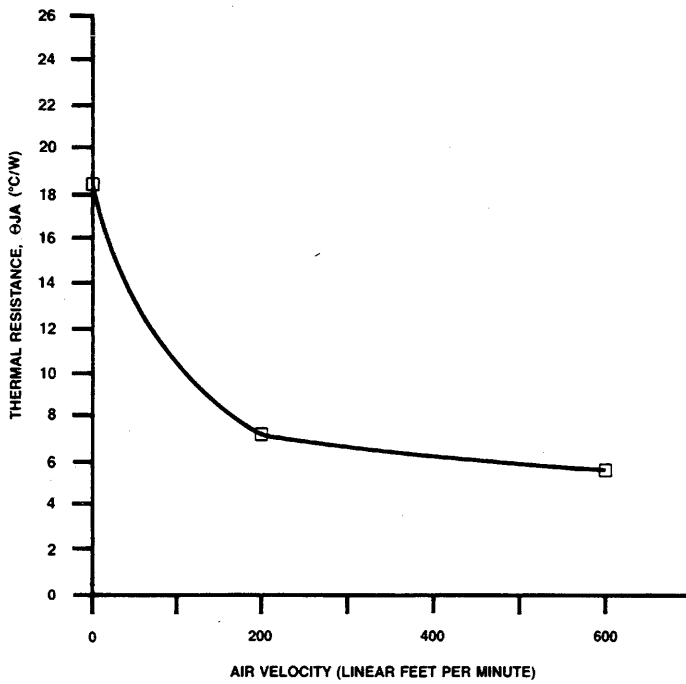
- 1) Use of a multilayer PC board with separate power ground and signal planes is highly recommended.
- 2) All V_{CC}E and V_{CC}T pins should be connected to the V_{CC} plane. V_{CC}T pins should be isolated from V_{CC}E pins by means of a slot cut in the V_{CC}E plane (see Figure 18). By physically separating the V_{CC}E and V_{CC}T pins, coupled noise will be reduced.
- 3) All GNDE and GNDT pins should be connected directly to the ground plane.
- 4) The V_{CC}T pins should be decoupled to ground with a 0.1- μ F ceramic capacitor and a 10- μ F electrolytic capacitor, placed as closely to the Am29325 as is practical. V_{CC}E pins should be decoupled to ground in a similar manner. A suggested layout is shown in Figure 18.



CD010480

- = Through Hole
- ⊕ = V_{CC} Plane Connection
- C₁ = C₃ = 0.1 μ F
- C₂ = C₄ = 10 μ F

Figure 18. Suggested Printed-Circuit Board Layout (Power and Ground Connections)



OP002251

Parameter	°C/W
θ_{JA} Still Air	19
θ_{JA} 200 L.F.M.	7
θ_{JA} 600 L.F.M.	5.5
θ_{JA} Heat Sink	2

Figure 19. Am29325 Thermal Characteristics (Typical)

APPENDIX A

DIFFERENCES BETWEEN THE IEEE PROPOSED STANDARD FOR BINARY FLOATING-POINT ARITHMETIC AND THE Am29325'S IEEE MODE

When operated in IEEE mode, the Am29325 High-Speed Floating-Point Processor complies with the single-precision portion of the IEEE Proposed Standard for Binary Floating-Point Arithmetic (P754, draft 10.0) in most respects. There are, however, several differences:

Denormalized Numbers

The Am29325 does not handle denormalized numbers. A denormalized input will be converted to zero of the same sign before the specified operation takes place. The operation proceeds in exactly the same manner as if the input were +0 or -0, producing the same numerical result and flags.

If the result of an operation, after rounding, has a magnitude smaller than 2^{-126} , the result is replaced by a zero of the same sign.

Representation of Overflows

In some rounding modes the proposed IEEE standard requires that overflows be represented as the format's most-positive or most-negative finite number. In particular:

- When rounding toward 0, all overflows should produce a result of the largest representable finite number with the sign of the intermediate result.
- When rounding toward $-\infty$, all positive overflows should produce a result of the largest representable positive finite number.
- When rounding toward $+\infty$, all negative overflows should produce a result of the largest representable negative finite number.

The Am29325, however, always represents positive overflows as $+\infty$ and negative overflows as $-\infty$, regardless of rounding mode.

Projective Mode

The proposed IEEE standard provides only for an affine mode to control the handling of infinities. The Am29325 provides

both affine and projective modes; the desired mode can be selected by the user.

Traps

The proposed IEEE standard stipulates that the user be able to request a trap on any exception. The Am29325 does not support trapped operation, and behaves as if traps are disabled.

Resetting of Flags

The proposed IEEE standard states that once an exception flag has been set, it is reset only at the user's request. The Am29325's flags, however, reflect the status of the most recent operation.

Generation of the Underflow Flag

The proposed IEEE standard suggests several possible criteria for determining if underflow occurs. These criteria generate underflow flags that differ in subtle ways. The underflow criteria chosen for the Am29325 stipulate that underflow occurs if:

- a) the rounded result of an operation has a magnitude in the range:

$$0 < \text{magnitude} < 2^{-126},$$

and

- b) the final result is not equal to the infinitely precise result.

Since the Am29325 never produces a denormalized number as the final result of a calculation, condition (b) is true whenever (a) is true. Note then that the operation of the Am29325's underflow flag is somewhat different than that of an "IEEE standard" system using the same underflow criteria. For example, if an operation should produce an infinitely precise result that is exactly 2^{-127} , an "IEEE standard" system would produce that value as the final result, expressed as a denormalized number. Since that system's final result is exact, the underflow flag would remain LOW. The Am29325, on the other hand, would output zero; since its final result is not exact, the underflow flag would be HIGH.

APPENDIX B

DIFFERENCES BETWEEN DEC VAX AND Am29325 DEC MODE

Operation in DEC mode complies with most aspects of single-precision floating-point operation outlined in the Digital Equipment Corporation's VAX Architecture Manual. However, there are some differences that should be noted:

Format

The Am29325's DEC format is:

sign	- bit 31
exponent	- bits 30 - 23
mantissa	- 22 - 0

The VAX format is:

sign	- bit 15
exponent	- 14 - 7
mantissa	- bits 6 - 0, bits 31 - 16

In both cases, fields are listed from MSB to LSB, with bit 31 the MSB of the 32-bit word. The Am29325's DEC format can be converted to VAX format by swapping the 16 LSBs and 16 MSBs of the 32-bit word.

Flags vs. Exceptions

In DEC VAX operation, certain unusual conditions arising during system operation may incur an exception, or an indication to the operating system that special handling is needed.

The VAX recognizes a number of arithmetic exceptions. The following exceptions are relevant to the operations supported by the Am29325:

Integer Overflow Trap: indicates that the last operation produced an integer overflow. The LSBs of the correct result are stored in the destination operand.

Floating-Point Overflow Trap/Fault: indicates that the last operation produced, after normalization and rounding, a floating-point number with magnitude greater than or equal to 2^{127} . A trap replaces the destination operand with the DEC-reserved operand 80000000_{16} ; a fault leaves the destination operand unchanged.

Floating-Point Underflow Trap/Fault: indicates that the last operation produced, after normalization and rounding, a floating-point number with magnitude less than 2^{-128} . A trap replaces the destination operand with zero; a fault leaves the destination operand unchanged.

Reserved Operand Fault: indicates that the last operation had a reserved operand as an input. The destination operand is unchanged.

The Am29325 does not directly support DEC traps and faults. Rather, it indicates unusual conditions by setting one or more of the six status flags HIGH. Table D2 describes flag operation in DEC mode.

Integer Overflow

In cases of integer overflow, the VAX signals the integer overflow trap and stores the LSBs of the correct result. The Am29325 sets the invalid operation flag and outputs the DEC-reserved operand 80000000_{16} .

Floating-Point Underflow/Overflow Operation

The VAX Architecture Manual specifies the action to be taken on the destination operand when floating-point underflow or overflow is encountered. The Am29325 has no immediate control over this destination operand, as it resides somewhere off-chip, either in a register or memory location. This isn't so much a difference between the VAX specification and Am29325 operation as it is a difference in scope.

The Am29325 responds to floating-point underflow by producing a final result of 0 (00000000_{16}); the underflow, inexact, and zero flags will be HIGH. It responds to floating-point overflow by producing the DEC-reserved operand 80000000_{16} as the final result; the overflow, inexact, and NAN flags will be HIGH.

Handling of DEC-Reserved Operands

If an operation has a DEC-reserved operand as an input, the Am29325 will produce that operand as the final result. If an operation has two input arguments and both are DEC-reserved operands, the operand on port R becomes the final result. For the VAX, operations with a DEC-reserved operand input or inputs do not modify the destination operand. As mentioned above, control of the destination operand is beyond the scope of the Am29325's operation.

Inexact Flag

The Am29325 provides an inexact flag to indicate that the final result produced by an operation is not equal to the infinitely precise result. The VAX does not provide this flag.

APPENDIX C

PERFORMING FLOATING-POINT DIVISION ON THE Am29325

While the Am29325 does not have a floating-point division instruction, it can be used to evaluate reciprocals. The division:

$$C = A/B$$

can then be performed by evaluating:

$$C = A*(1/B)$$

Only a modest amount of external hardware is needed to implement the reciprocal function.

The technique for calculating reciprocals is based on the Newton-Raphson method for obtaining the roots of an equation. The roots of equation:

$$F(x) = 0$$

can be found by iteratively evaluating the equation:

$$x_{i+1} = x_i - F(x_i)/F'(x_i)$$

The process begins by making a guess as to the value of x_i , and using this guess or "seed" value to perform the first iteration. Iterations are continued until the root is evaluated to the desired accuracy. The number of iterations needed to achieve a given accuracy depends both on the accuracy of the seed value and the nature of $F(x)$.

Now consider the equation:

$$F(x) = (1/x) - B$$

The root of $F(x)$ is $1/B$. The reciprocal of B , then, can be found by using the Newton-Raphson method to find the root of $F(x)$. The iterative equation for finding the root is:

$$\begin{aligned} x_{i+1} &= x_i - F(x_i)/F'(x_i) \\ &= x_i - (1/x_i - B) / - (x_i)^{-2} \\ &= x_i (2 - B*x_i) \end{aligned}$$

It can be shown that, in order for this iterative equation to converge, the seed value x_0 must fall in the range:

$$\begin{aligned} 0 < x_0 < 2/B & \quad \text{if } B > 0 \\ \text{or } 2/B < x_0 < 0 & \quad \text{if } B < 0 \end{aligned}$$

For example, if the reciprocal of 3 is to be evaluated, the seed value must be between 0 and 2/3.

The error of x_i reduces quadratically; that is, if the error of x_i is e , the error is reduced to order e^2 by the next iteration. The number of bits of accuracy in the result, then, roughly doubles after every iteration. While this is only an approximation of the actual error produced, it is a handy rule of thumb for determining the number of iterations needed to produce a result of a certain accuracy, given the accuracy of the seed.

Example 1:

Find the reciprocal of 7.25.

Solution:

The seed value must fall in the range:

$$\begin{aligned} 0 < x_0 < 2/7.25 \\ \text{or } 0 < x_0 < .275862 \end{aligned}$$

Suppose x_0 is chosen to be .1:

$$\begin{aligned} \text{Iteration 1: } x_1 &= x_0 (2 - B^*x_0) \\ &= .1(2 - (7.25) (.1)) \\ &= .1275 \end{aligned}$$

$$\begin{aligned} \text{Iteration 2: } x_2 &= x_1 (2 - B^*x_1) \\ &= .1275(2 - (7.25) (.1275)) \\ &= .1371421875 \end{aligned}$$

$$\begin{aligned} \text{Iteration 3: } x_3 &= x_2 (2 - B^*x_2) \\ &= .1371421875^* \\ &\quad (2 - (7.25) (.1371421875)) \\ &= .1379265230 \end{aligned}$$

The actual value of $1/7.25$, to ten decimal places, is .1379310345.

The error after each iteration is:

Iteration	x_i	Error to Ten Places
0	.1	-0.0379310345
1	.1275	-0.0104310345
2	.1371421875	-0.0007888470
3	.1379265230	-0.0000045115

Example 2:

Find the reciprocal of $-.3$.

Solution:

The seed value must fall in the range:

$$\begin{aligned} 2/(-.3) < x_0 < 0 \\ \text{or } -6.66 < x_0 < 0 \end{aligned}$$

Suppose x_0 is chosen to be -2.0 :

$$\begin{aligned} \text{Iteration 1: } x_1 &= x_0 (2 - B^*x_0) \\ &= -2.0(2 - (-.3) (-2.0)) \\ &= -2.8 \end{aligned}$$

$$\begin{aligned} \text{Iteration 2: } x_2 &= x_1 (2 - B^*x_1) \\ &= -2.8(2 - (-.3) (-2.8)) \\ &= -3.248 \end{aligned}$$

$$\begin{aligned} \text{Iteration 3: } x_3 &= x_2 (2 - B^*x_2) \\ &= -3.248(2 - (-.3) (-3.248)) \\ &= -3.3311488 \end{aligned}$$

$$\begin{aligned} \text{Iteration 4: } x_4 &= x_3 (2 - B^*x_3) \\ &= -3.3311488^* \\ &\quad (2 - (-.3) (-3.3311488)) \\ &= -3.333331902 \end{aligned}$$

The actual value of $1/(-.3)$, to ten decimal places, is -3.333333333 .

The error after each iteration is:

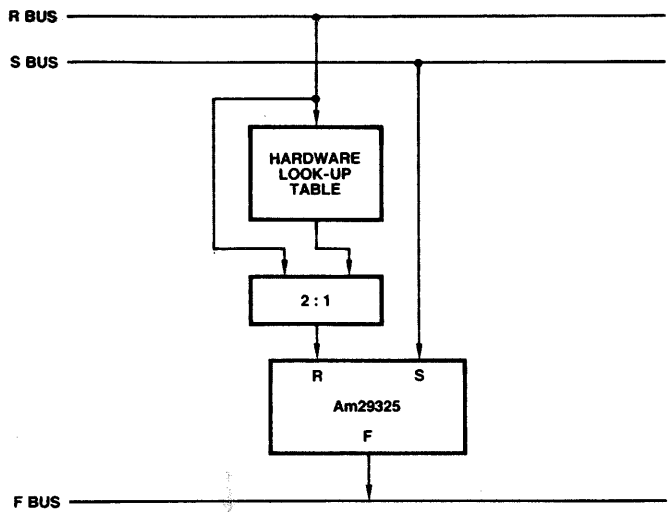
i	x_i	Error to Ten Places
0	-2.0	1.333333333
1	-2.8	0.533333333
2	-3.248	0.085333333
3	-3.3311488	0.002184533
4	-3.333331902	0.000001431

In order to implement the Newton-Raphson method on the Am29325, some means is needed to generate the seed used in the first iteration. One approach is to place a hardware seed look-up table between the R bus and the Am29325; see Table C1. A more detailed diagram of the look-up table appears in Figure C2.

TABLE C1. CONTENTS OF THE SEED EXPONENT PROM

DEC		IEEE	
Address (16)	Data (16)	Address (16)	Data (16)
000	(Note 1)	100	(Note 1)
001	(Note 1)	101	FC
002	FF	102	FB
003	FE	103	FA
004	FD	104	F9
005	FC	105	F8
006	FB	106	F7
007	FA	107	F6
008	F9	108	F5
009	F8	109	F4
00A	F7	10A	F3
00B	F6	10B	F2
00C	F5	10C	F1
00D	F4	10D	F0
00E	F3	10E	EF
00F	F2	10F	EE
010	F1	110	ED
011	F0	111	EC
012	EF	112	EB
.	.	.	.
.	.	.	.
.	.	.	.
0EE	13	1EE	0F
0EF	12	1EF	0E
0F0	11	1F0	0D
0F1	10	1F1	0C
0F2	0F	1F2	0B
0F3	0E	1F3	0A
0F4	0D	1F4	09
0F5	0C	1F5	08
0F6	0B	1F6	07
0F7	0A	1F7	06
0F8	09	1F8	05
0F9	08	1F9	04
0FA	07	1FA	03
0FB	06	1FB	02
0FC	05	1FC	01
0FD	04	1FD	(Note 2)
0FE	03	1FE	(Note 2)
0FF	02	1FF	(Note 2)

- Notes: 1. The reciprocals of these numbers are too large to be represented in the selected format.
 2. The reciprocals of these numbers are too small to be represented in normalized IEEE format.



AF004640

Figure C1. Adding a Hardware Look-Up Table to the Am29325

The look-up table has two sections: a biased exponent look-up PROM, and a fraction look-up PROM. The seed-biased exponent look-up table is stored in a 512-by-8-bit PROM. This table consists of two sections: the DEC format section (which occupies addresses 000-0FF₁₆), and the IEEE section (which occupies addresses 100-1FF₁₆). The appropriate table will be selected automatically if address line A₈ is wired to the Am29325's IEEE/DEC pin. The equations implemented by these table sections are:

DEC table: seed biased exponent
 = 257₁₀ - input biased exponent

IEEE table: seed biased exponent
 = 253₁₀ - input biased exponent

Table C1 lists the contents of this PROM.

The seed fraction look-up table is stored in one or more PROMs, the number of PROMs depending on the desired accuracy of the seed value. The hardware depicted in Figure

C2 uses two 4K-by-8-bit PROMs to implement a fraction look-up table whose inputs are the 12 MSBs of the input argument's fraction. These PROMs output the 16 MSBs of the seed's fraction field — the remaining 7 bits of fraction are set to 0. The equation implemented in this table is:

$$\text{seed fraction} = \frac{2}{1 + \text{input fraction}} - 1$$

where the value of the input fraction falls in the range

$$0 \leq \text{input fraction} < 1$$

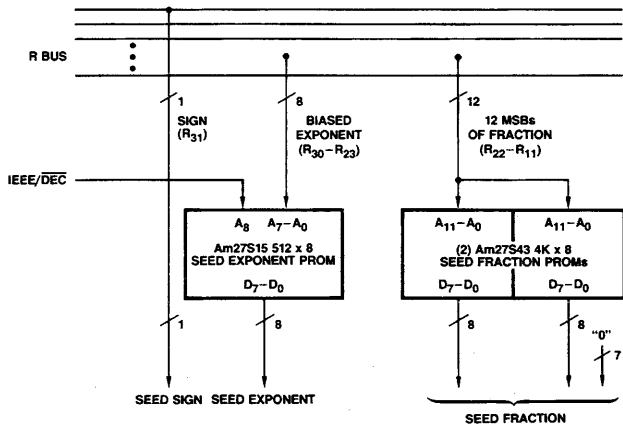
Note that the seed fraction must also be constrained to fall in the range

$$0 \leq \text{seed fraction} < 1$$

Therefore, if the input fraction is 0, the corresponding seed fraction stored in the table must be .111...111₂, not 1.0₂. The same seed fraction look-up table may be used for both IEEE and DEC formats. Table C2 contains a partial listing for the seed fraction look-up table shown in Figure C2.

TABLE C2. CONTENTS OF THE SEED FRACTION PROMS

Address (16)	Value of Input Fraction (10)	Value of Seed Fraction (10)	PROM Outputs (16)	
			R ₂₂ - R ₁₅	R ₁₄ - R ₇
000	0.0	0.9999999999 (see text)	FF	FF
001	0.0002441406	0.9995118370	FF	E0
002	0.0004882812	0.9990239150	FF	C0
003	0.0007324219	0.9985362280	FF	A0
004	0.0009765625	0.9980487790	FF	80
005	0.0012207031	0.9975615710	FF	60
006	0.0014648438	0.9970745970	FF	40
007	0.0017089844	0.9965878630	FF	20
008	0.0019531250	0.9961013650	FF	00
009	0.0021972656	0.9956151030	FE	E1
00A	0.0024414063	0.9951290800	FE	C0
00B	0.0026855469	0.9946432920	FE	A1
00C	0.0029296875	0.9941577400	FE	81
.
.
.
FF6	0.9975585938	0.0012221950	00	50
FF7	0.9978027344	0.0010998410	00	48
FF8	0.9980486750	0.0009775170	00	40
FF9	0.9982910156	0.0008552230	00	38
FFA	0.9985351563	0.0007329590	00	30
FFB	0.9987792969	0.0006107240	00	28
FFC	0.9990234375	0.0004885200	00	20
FFD	0.9992675781	0.0003663450	00	18
FFE	0.9995117188	0.0002442000	00	10
FFF	0.9997558594	0.0001220850	00	08



AF004631

Figure C2. The Hardware Look-Up Table

With the hardware look-up table in place, the reciprocal of value B can be calculated with the following series of operations:

- 1) Place B on both the R and S buses. The 2 : 1 multiplexer at the output of the hardware look-up table should select the output of the look-up table (see Figure C3-A).
- 2) Load the seed value x_0 into register R and load B into register S. Select the R TIMES S operation (see Figure C3-B).
- 3) Load product $B \cdot x_0$ into register F. Select the 2 MINUS S operation, and select register F as the input to the ALU S port (see Figure C3-C).
- 4) Load $2 - B \cdot x_0$ into register F. Select the R TIMES S operation and select register F as the input to the ALU S port (see Figure C3-D).
- 5) Load the value x_1 ($x_1 = x_0(2 - B \cdot x_0)$) into registers R and F. Select the R TIMES S operation (see Figure C3-E).
- 6) Repeat steps 3 through 5 until the result has the accuracy desired.

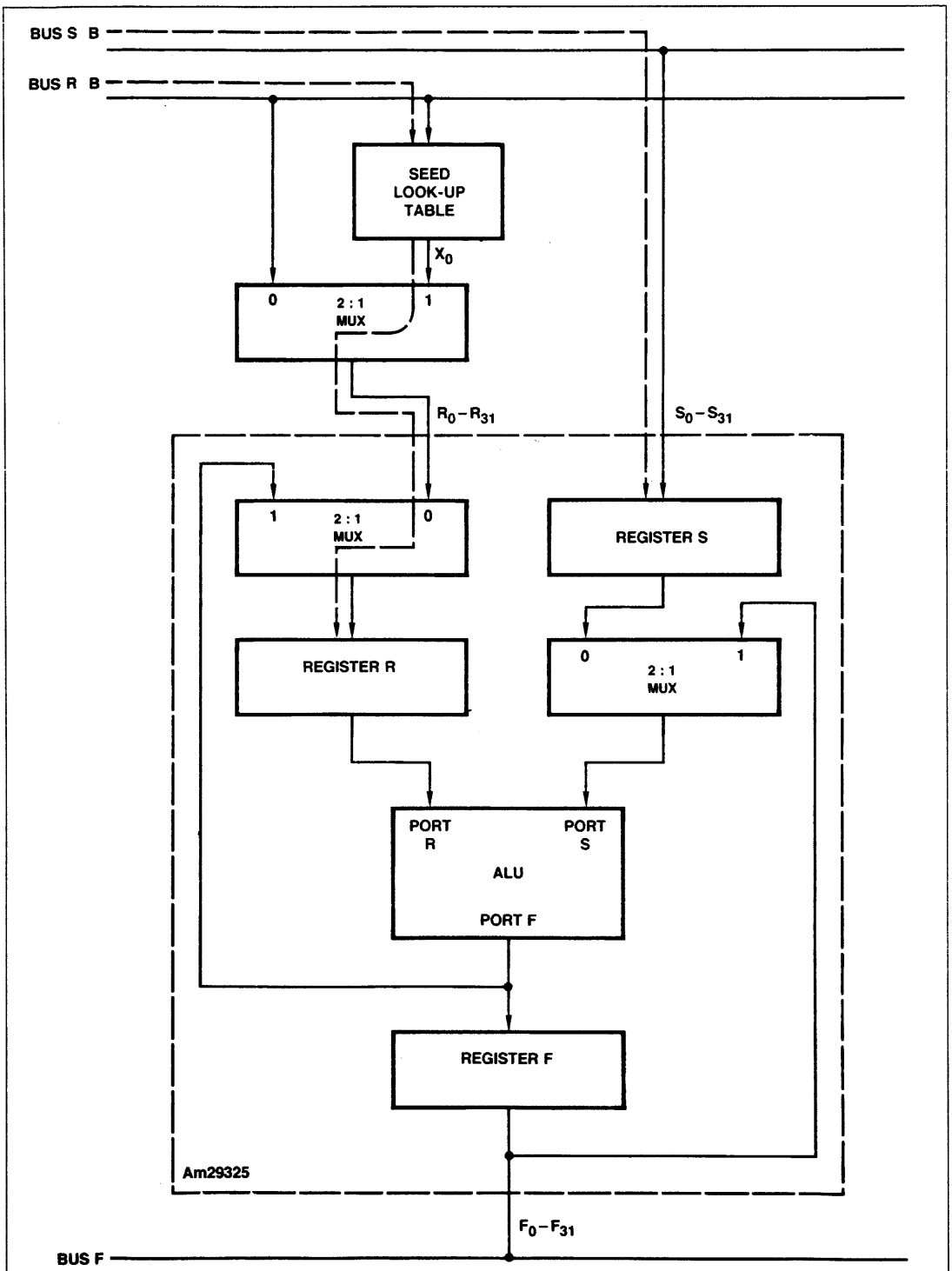


Figure C3-A. Data Flow for Step 1 of the Reciprocal Procedure

DF006210

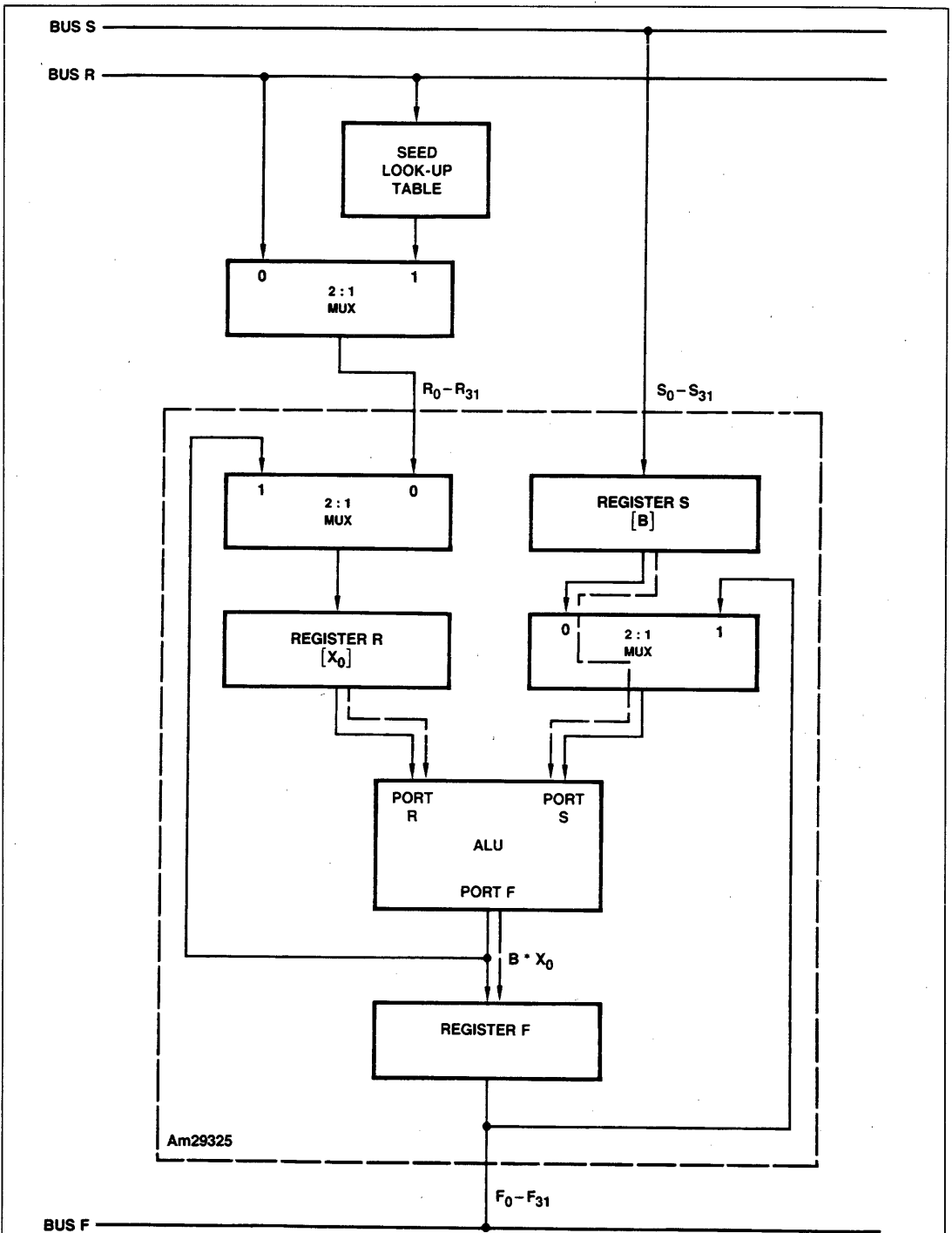


Figure C3-B. Data Flow for Step 2 of the Reciprocal Procedure

DF006220

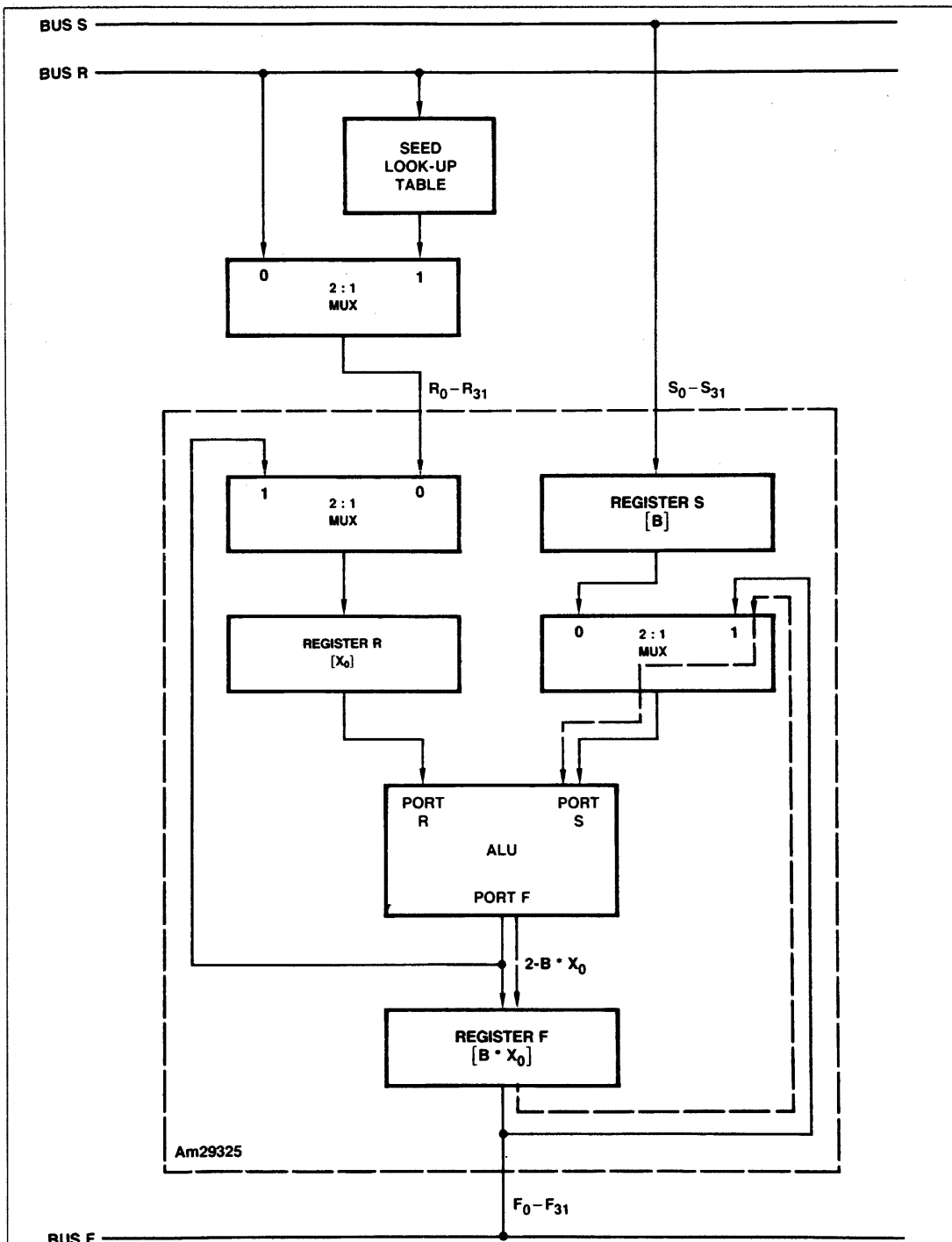


Figure C3-C. Data Flow for Step 3 of the Reciprocal Procedure

DF006230

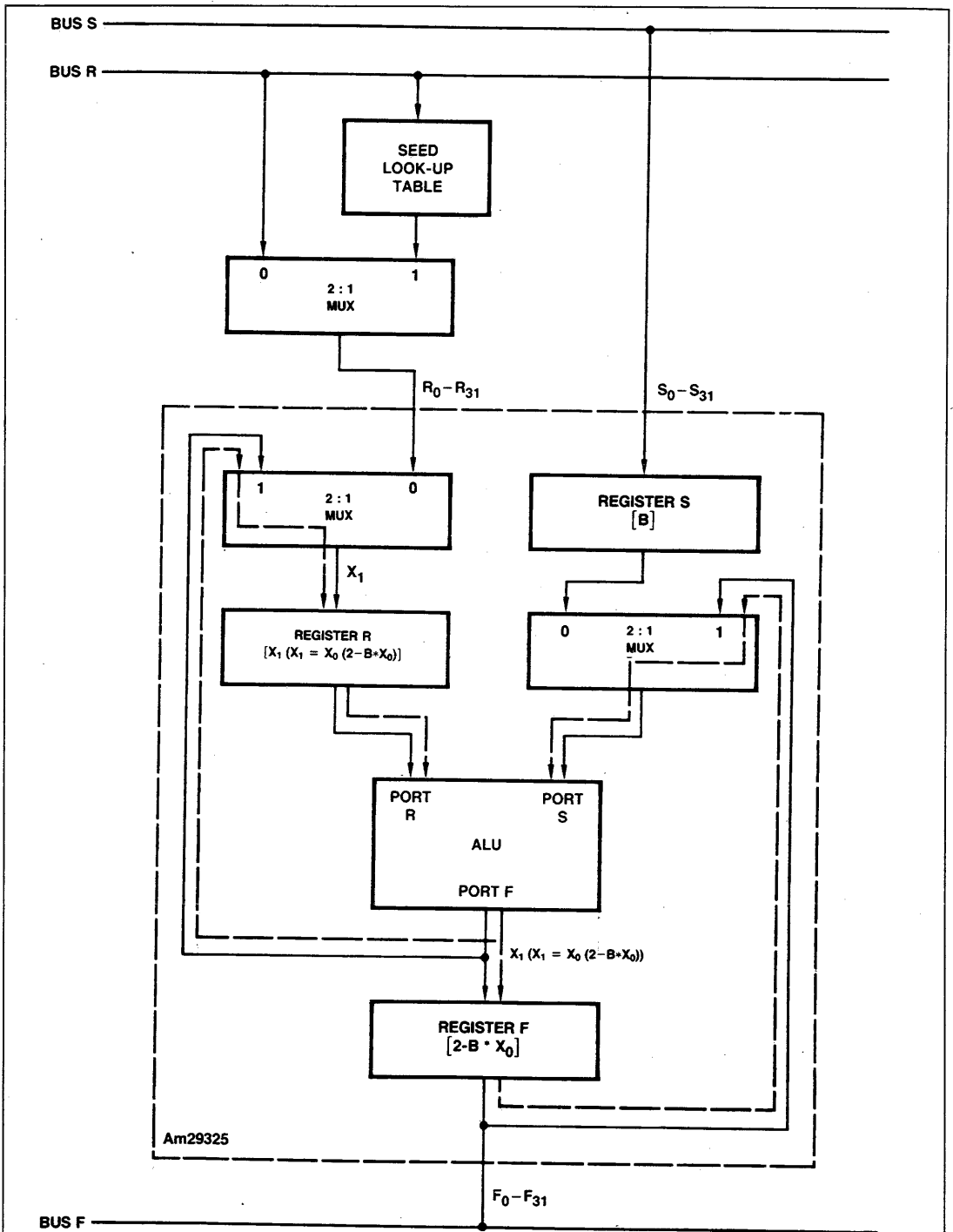


Figure C3-D. Data Flow for Step 4 of the Reciprocal Procedure

DF006240

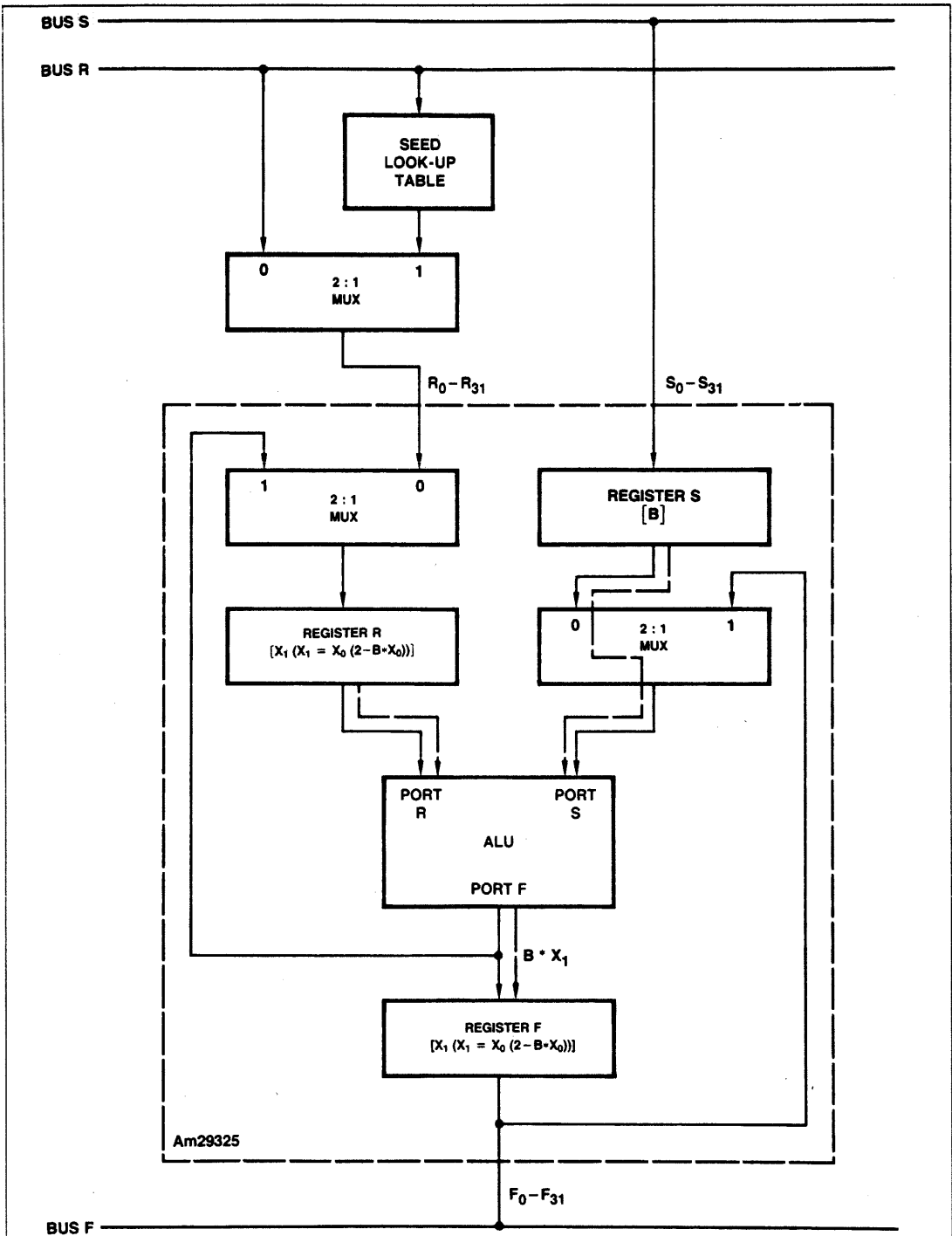


Figure C3-E. Data Flow for Step 5 of the Reciprocal Procedure

DF006250

A tabular description of the operations above is given in Table C3. The following examples, performed in IEEE format, illustrate the process.

Example 1:

Find the reciprocal of 25.3.

Solution: The IEEE floating-point representation for 25.3 is 41CA6666₁₆. The reciprocal process is begun by feeding this value to both the seed look-up table

and port S. The look-up table produces the value .03952789₁₀ (3D21E800₁₆). The reciprocal is evaluated using the procedure described above; register values for each step are given in Table C4. The expected result, to the precision of the floating-point word, is .03952569₁₀ (3D21E5B1₁₆). In this case the expected result is produced after the first iteration. All subsequent iterations produce the same result, and are therefore unnecessary.

TABLE C3. SEQUENCE OF EVENTS FOR EVALUATING RECIPROCALLS

Clock Cycle	I ₀ -I ₂	I ₃	I ₄	ENR	ENS	ENF	Register R	Register S	Register F
1	Y	X	0	0	0	X	-	-	-
2	R TIMES S	0	X	1	1	0	X ₀	B	-
3	2 MINUS S	1	X	1	1	0	X ₀	B	B*X ₀
4	R TIMES S	1	1	0	1	0	X ₀	B	2 - B*X ₀
5	R TIMES S	0	X	1	1	0	X ₁ (= X ₀ (2 - B*X ₀))	B	X ₁ (= X ₀ (2 - B*X ₀))
6	2 MINUS S	1	X	1	1	0	X ₁	B	B*X ₁
7	R TIMES S	1	1	0	1	0	X ₁	B	2 - B*X ₁
8	R TIMES S	0	X	1	1	0	X ₂ (= X ₁ (2 - B*X ₁))	B	X ₂ (= X ₁ (2 - B*X ₁))

First iteration

Second iteration

X = DON'T CARE

TABLE C4. INPUT BUS AND REGISTER VALUES FOR EXAMPLE 1

Clock Cycle	R Input	S Input	Register R	Register S	Register F
1	3D21E800 (.03952789)	41CA6666 ₁₆ (25.3)	-	-	-
2	-	-	3D21E800 ₁₆ (.03952789)	41CA6666 ₁₆ (25.3)	-
3	-	-	3D21E800 ₁₆ (.03952789)	41CA6666 ₁₆ (25.3)	3F8001D3 ₁₆ (1.0000556)
4	-	-	3D21E800 ₁₆ (.03952789)	41CA6666 ₁₆ (25.3)	3F7FFC5A ₁₆ (.99984419)
5	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3D21E5B1 ₁₆ (.03952569)
6	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3F7FFFFF ₁₆ (.99999994)
7	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3F800000 ₁₆ (1.0)
8	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3D21E5B1 ₁₆ (.03952569)

← Result of first iteration

← Result of second iteration

Example 2:

Find the reciprocal of -0.4725 .

Solution: The IEEE floating-point representation for -0.4725 is $BEF1EB85_{16}$. The reciprocal process is begun by feeding this value to both the seed look-up table and port S. The look-up table produces the value -2.11621094_{10} ($C0077000_{16}$). The reciprocal is

evaluated using the procedure described above; register values for each step are given in Table C5. The expected result, to the precision of the floating-point word, is -2.116402_{10} ($C0077322_{16}$). In this case the expected result is produced after the first iteration. All subsequent iterations produce the same result, and are therefore unnecessary.

TABLE C5. INPUT BUS AND REGISTER VALUES FOR EXAMPLE 2

Clock Cycle	R Input	S Input	Register R	Register S	Register F
1	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	-	-	-
2	-	-	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	-
3	-	-	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	$3F7FFA14_{16}$ (0.99990963)
4	-	-	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	$3F8002F6_{16}$ (1.0000904)
5	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$C0077322_{16}$ (-2.116402)
6	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$3F800000_{16}$ (1.0)
7	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$3F800000_{16}$ (1.0)
8	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$C0077322_{16}$ (-2.116402)

← Result of first iteration

← Result of second iteration

APPENDIX D

SUMMARY OF FLAG OPERATION

Tables D1, D2, and D3 summarize flag operation for the IEEE mode, the DEC mode, and for the IEEE-TO-DEC and DEC-TO-IEEE operations.

TABLE D1. FLAG SUMMARY FOR IEEE MODE

Operation	Condition(s)	INV	OVF	UNF	INE	ZER	NAN
Any operation listed in the IEEE Invalid Operations Table		H	L	L	L	L	H
R PLUS S R MINUS S R TIMES S 2 MINUS S	Input operands are finite $ \text{rounded result} \geq 2^{128}$	L	H	L	H	L	L
R PLUS S R MINUS S R TIMES S	$0 < \text{rounded result} < 2^{-126}$	L	L	H	H	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result does not equal infinitely precise result	L	*	*	H	*	L
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result is zero	L	L	*	*	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S FP-TO-INT	Final result is a NAN	*	L	L	L	L	H

Notes: INV = Invalid operation flag
 OVF = Overflow flag
 UNF = Underflow flag
 INE = Inexact flag
 ZER = Zero flag
 NAN = NAN flag
 L = LOW
 H = HIGH
 * = State of flag depends on the input operands and the operation performed

TABLE D2. FLAG SUMMARY FOR DEC MODE

Operation	Condition(s)	INV	OVF	UNF	INE	ZER	NAN
FP-TO-INT	Rounded result $> 2^{31}-1$ or rounded result $< -2^{31}$	H	L	L	L	L	H
FP-TO-INT	Input is a DEC-reserved operand	L	L	L	L	L	H
R PLUS S R MINUS S R TIMES S 2 MINUS S	$ \text{Rounded result} \geq 2^{127}$	L	H	L	H	L	H
R PLUS S R MINUS S R TIMES S	$0 < \text{rounded result} < 2^{-128}$	L	L	H	H	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result does not equal infinitely precise result	L	*	*	H	*	*
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result is zero	L	L	*	*	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S FP-TO-INT	Final result is a DEC-reserved operand	*	*	L	L	L	H

Notes: INV = Invalid operation flag H = HIGH
 OVF = Overflow flag * = State of flag
 UNF = Underflow flag depends on the
 INE = Inexact flag input operands
 ZER = Zero flag and the operation
 NAN = NAN flag performed
 L = LOW

TABLE D3. FLAG SUMMARY FOR IEEE-TO-DEC AND DEC-TO-IEEE CONVERSIONS

Operation	Condition(s)	INV	OVF	UNF	INE	ZER	NAN
IEEE-TO-DEC	Input is a NAN	H	L	L	L	L	H
IEEE-TO-DEC	$ \text{Input} \geq 2^{127}$	L	H	L	H	L	H
DEC-TO-IEEE	Input is a DEC-reserved operand	H	L	L	L	L	H
DEC-TO-IEEE	$0 < \text{rounded result} < 2^{-126}$	L	L	H	H	H	L
DEC-TO-IEEE IEEE-TO-DEC	Final result is zero	L	L	*	*	H	L

Notes: INV = Invalid operation flag H = HIGH
 OVF = Overflow flag * = State of flag
 UNF = Underflow flag depends on the
 INE = Inexact flag input operands
 ZER = Zero flag and the operation
 NAN = NAN flag performed
 L = LOW

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Temperature Under Bias — T_C	-55 to +125°C
Supply Voltage to Ground Potential	
Continuous	-0.5 to +7.0 V
DC Voltage Applied to Outputs	
for HIGH State	-0.5 V to + V_{CC} Max.
DC Input Voltage	-0.5 to +5.5 V
DC Output Current, into Outputs	30 mA
DC Input Current	-30 to +5.0 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature, Case (T_C)	0 to +85°C
Supply Voltage (V_{CC})	+4.75 to +5.25 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating ranges unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions (Note 1)		Min.	Max.	Units
V_{OH}	Output HIGH Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IL}$ or V_{IH} $I_{OH} = -1.0$ mA		2.4		Volts
V_{OL}	Output LOW Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IL}$ or V_{IH} $I_{OL} = 4.0$ mA			0.5	Volts
V_{IH}	Input HIGH Level	Guaranteed Input Logical HIGH Voltage for All Inputs		2.0		Volts
V_{IL}	Input LOW Level	Guaranteed Input Logical LOW Voltage for All Inputs			0.8	Volts
V_I	Input Clamp Voltage	$V_{CC} = \text{Min.}$ $I_{IN} = -18$ mA			-1.5	Volts
I_{IL}	Input LOW Current	$V_{CC} = \text{Max.}$ $V_{IN} = 0.5$ V	CLK, S16/32, OE Others		-1.0 -0.5	mA
I_{IH}	Input HIGH Current	$V_{CC} = \text{Max.}$ $V_{IN} = 2.4$ V	CLK, S16/32, OE Others		100 50	μ A
I_I	Input HIGH Current	$V_{CC} = \text{Max.}$ $V_{IN} = 5.5$ V			1	mA
I_{OZH} I_{OZL}	$F_0 - F_{31}$ Off State (High-Impedance) Output Current	$V_{CC} = \text{Max.}$	$V_O = 2.4$ V $V_O = 0.5$ V		50 -50	μ A
I_{SC}	Output Short-Circuit Current (Note 2)	$V_{CC} = \text{Max.}$ +0.5 V $V_O = 0.5$ V	$F_0 - F_{31}$ Outputs Flag Outputs	-15 -15	-50 -50	mA
I_{CC}	Power Supply Current (Notes 3, 4)	$V_{CC} = \text{Max.}$	COM'L, $T_C = +25^\circ\text{C}$	1800 pF Typical		
			COM'L Only	$T_C = 0$ to +85°C Case Temp.	2114	mA
				$T_C = +85^\circ\text{C}$ Case Temp.	1950	

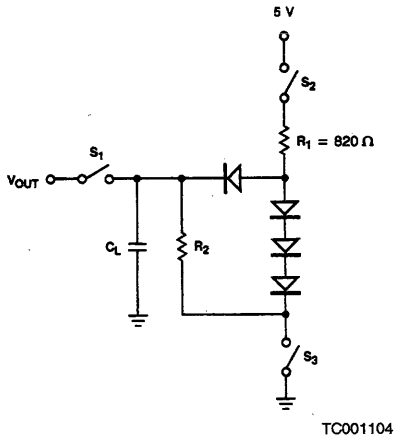
- Notes: 1. For conditions shown as Min. or Max., use the appropriate value specified under Operating Ranges for the applicable device type.
 2. Not more than one output should be shorted at a time. Duration of the short-circuit test should not exceed one second.
 3. Measured with OE LOW, and with all output bits ($F_0 - F_{31}$ and flag outputs) LOW.
 4. Worst-case I_{CC} applies to cold start at lowest operating temperature.

SWITCHING CHARACTERISTICS over operating ranges unless otherwise specified

No.	Parameter Symbol	Parameter Description	Test Conditions	COM'L (Note 2)				Units
				T _C = 0 to +85°C Case Temp.				
				Am29325		Am29325A		
				Min.	Max.	Min.	Max.	
1	t _{ASC}	Clocked Add, Subtract Time (R PLUS S, R MINUS S, 2 MINUS S)		93		83	ns	
2	t _{MC}	Clocked Multiply Time (R TIMES S)		93		83	ns	
3	t _{CC}	Clocked Conversion Time (INT-TO-FP, FP-TO-INT, IEEE-TO-DEC, DEC-TO-IEEE)		100		90	ns	
4	t _{ASUC}	Unclocked Add, Subtract Time (R, S to F, Flags) for R PLUS S, R MINUS S, and 2 MINUS S Instructions	FT ₀ = HIGH FT ₁ = HIGH			125	110	ns
5	t _{MUC}	Unclocked Multiply Time (R, S to F, Flags) for R TIMES S Instruction				125	110	ns
6	t _{CUC}	Unclocked Conversion Time (R, S to F, Flags) for INT-TO-FP, FP-TO-INT, IEEE-TO-DEC and DEC-TO-IEEE Instructions				125	110	ns
7	t _{PWH}	Clock Pulse Width HIGH		15		15	(Note 3) ns	
8	t _{PWL}	Clock Pulse Width LOW		15		15	(Note 3) ns	
9	t _{PDF1}	Clock to F ₀ - F ₃₁ and Flag Outputs	FT ₀ = LOW FT ₁ = HIGH			125	110	ns
10	t _{PDF2}		FT ₁ = LOW			34	30	ns
11	t _{PZL}	OE Enable Time	Z to LOW			31	29	ns
12	t _{PZH}		Z to HIGH			26	24	ns
13	t _{PLZ}	OE Disable Time	LOW to Z			31	31	ns
14	t _{PHZ}		HIGH to Z			26	26	ns
15	t _{PZL16}	Clock ↑ to F ₀ - F ₁₅ Enable, 16-Bit I/O Mode	Z to LOW	S16/32 = HIGH ONEBUS = LOW		41	39	ns
16	t _{PZH16}		Z to HIGH				33	33
17	t _{PLZ16}	Clock ↓ to F ₀ - F ₁₅ Disable, 16-Bit I/O Mode	LOW to Z			26	26	ns
18	t _{PHZ16}		HIGH TO Z			38	38	ns
19	t _{PZL16}	Clock ↓ to F ₁₆ - F ₃₁ Enable, 16-Bit I/O Mode	Z to LOW	S16/32 = HIGH ONEBUS = LOW		30	29	ns
20	t _{PZH16}		Z to HIGH				26	26
21	t _{PLZ16}	Clock ↑ to F ₁₆ - F ₃₁ Disable, 16-Bit I/O Mode	LOW to Z			34	34	ns
22	t _{PHZ16}		HIGH to Z			36	36	ns
23	t _{SCE}	Register Clock Enable Setup Time	FT ₀ = LOW FT ₁ = LOW	6		6		ns
24	t _{HCE}	Register Clock Enable Hold Time	FT ₀ = LOW FT ₁ = LOW	1		1		ns
25	t _{SD1}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Setup Time (Note 1)	FT ₀ = LOW	13		13		ns
26	t _{HD1}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Hold Time (Note 1)		6		6		ns
27	t _{SD2}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Setup Time (Note 1)	FT ₀ = HIGH FT ₁ = LOW	104		104		ns
28	t _{HD2}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Hold Time (Note 1)		-5		-5		ns
29	t _{SI02}	I ₀ - I ₂ Instruction Select Setup Time	FT for Destination Register = LOW	100		100		ns
30	t _{HI02}	I ₀ - I ₂ Instruction Select Hold Time		-5		-5		ns
31	t _{PD102}	I ₀ - I ₂ Instruction Select to F ₀ - F ₃₁ , Flags	FT ₁ = HIGH			129		ns
32	t _{SI3}	I ₃ Port S Input Select Setup Time	FT ₁ = LOW	93		93		ns
33	t _{HI3}	I ₃ Port S Input Select Hold Time		-5		-5		ns
34	t _{SI4}	I ₄ Register R Input Select Setup Time (Note 1)	FT ₀ = LOW	15		15		ns
35	t _{HI4}	I ₄ Register R Input Select Hold Time (Note 1)		0		0		ns
36	t _{SRM}	Round Mode Select Setup Time	FT for Destination Register = LOW	45		45		ns
37	t _{HRM}	Round Mode Select Hold Time		0		0		ns
38	t _{PRF}	Round Mode Select to F ₀ - F ₃₁ , Flags	FT ₁ = HIGH			76		ns

Notes: 1. See timing diagram for desired mode of operation to determine clock edge to which these setup and hold times apply.
 2. It is the responsibility of the user to maintain a case temperature of 85°C or less. AMD recommends an air velocity of at least 200 linear feet per minute over the heat sink.
 3. Tester limitations necessitate this spec limit. Typical value shown is actual worst-case value.

SWITCHING TEST CIRCUITS

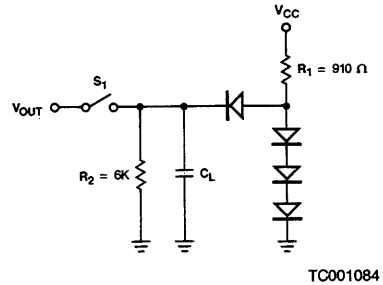


$$R_1 = I_{OL} + \frac{V_{OL}}{1K}$$

$$R_2 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OH}}$$

A. Three-State Outputs

- Notes:
1. $C_L = 50$ pF includes scope probe, wiring, and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 S_1 and S_2 are closed while S_3 is open for t_{pZL} test.
 4. $C_L = 5.0$ pF for output disable tests.

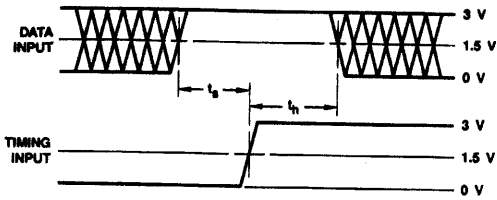


$$R_1 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OL} + \frac{V_{OL}}{R_2}}$$

$$R_2 = \frac{2.4 V}{I_{OH}}$$

B. Normal Outputs

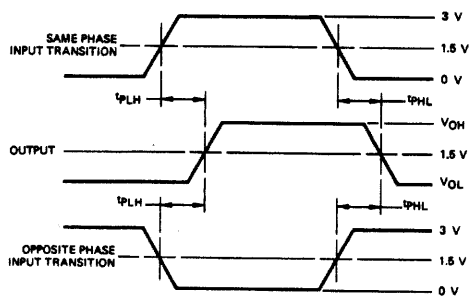
SWITCHING TEST WAVEFORMS



WFR02970

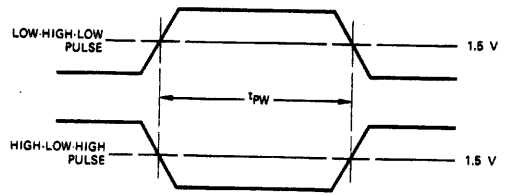
- Notes: 1. Diagram shown for HIGH data only.
Output transition may be opposite sense.
2. Cross hatched area is don't care condition.

Set-Up, Hold, and Release Times



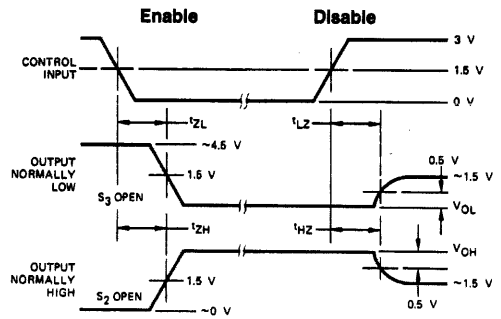
WFR02980

Propagation Delay



WFR02790

Pulse Width



WFR02660

- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
2. S_1 , S_2 and S_3 of Load Circuit are closed except where shown.

Enable and Disable Times

Notes on Test Methods

The following points give the general philosophy which we apply to tests which must be properly engineered if they are to be implemented in an automatic environment. The specifics of what philosophies applied to which test are shown.

1. Ensure that the part is adequately decoupled at the test head. Large changes in supply current when the device switches may cause function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an input transition, ground current may change by as much as 400 mA in 5 to 8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily.
4. Use extreme care in defining input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins which may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \leq 3$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. Capacitive Loading for AC Testing: Automatic testers and their associated hardware have stray capacitance which varies from one type of tester to another, but generally around 50 pF. This, of course, makes it impossible to make direct measurements of parameters which call for a smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays," which measure the propagation delays in to and out of the high-impedance state, and are usually specified at a load capacitance of 5.0 pF. In these cases the test is performed at the higher load capacitance (typically 50 pF), and engineering correlations based on data taken with a bench set up are used to predict the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impossible to make measurements at both capacitances even though they may both be greater than the stray capacitance. In these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is predicted from engineering correlations based on data taken with a bench set up and the knowledge that certain DC measurements (e.g., I_{OH} , I_{OL}) have already been taken and are within specification. In some cases, special DC tests are performed in order to facilitate this correlation.

7. Threshold Testing: The noise associated with automatic testing, the long, inductive cables, and the high gain of bipolar devices when in the vicinity of the actual device threshold, frequently give rise to oscillations when testing high-speed circuits. These oscillations are not indicative of a reject device, but instead, of an overtaxed test system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" high and low levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels rather than at V_{IL} Max. and V_{IH} Min.

8. AC Testing: Occasionally, parameters are specified which cannot be measured directly on automatic testers because

of tester limitations. Data input hold times often fall into this category. In these cases, the parameter in question is guaranteed by correlating tests with other AC tests which have been performed. These correlations are arrived at by the cognizant engineer by using data from precise bench measurements in conjunction with the knowledge that certain DC parameters have already been measured and are within specification.

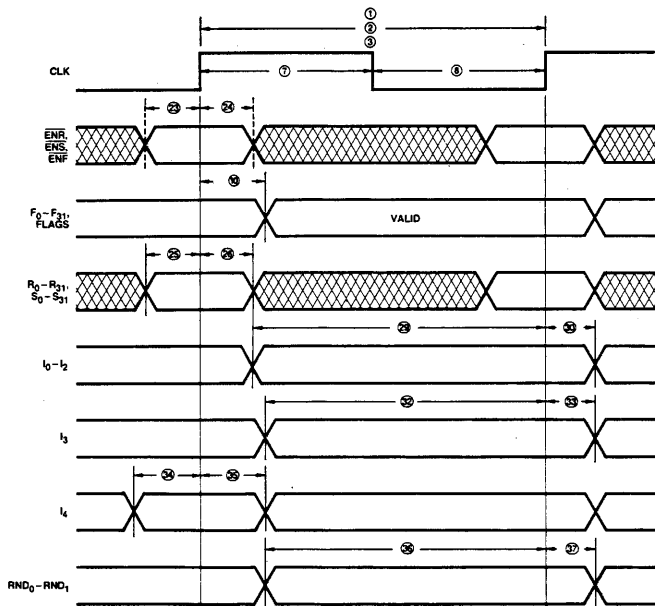
In some cases, certain AC tests are redundant since they can be shown to be predicted by other tests which have already been performed. In these cases, the redundant tests are not performed.

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

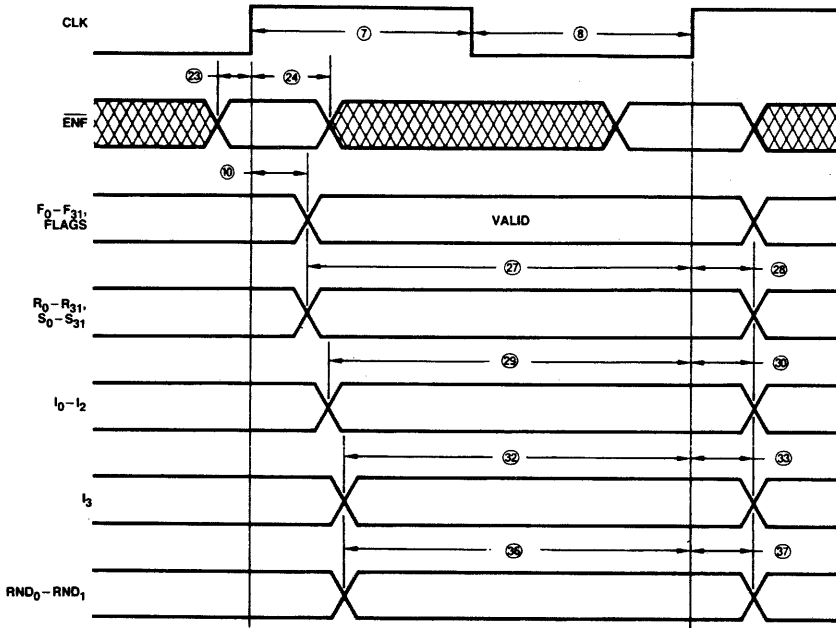
KS000010



WF023760

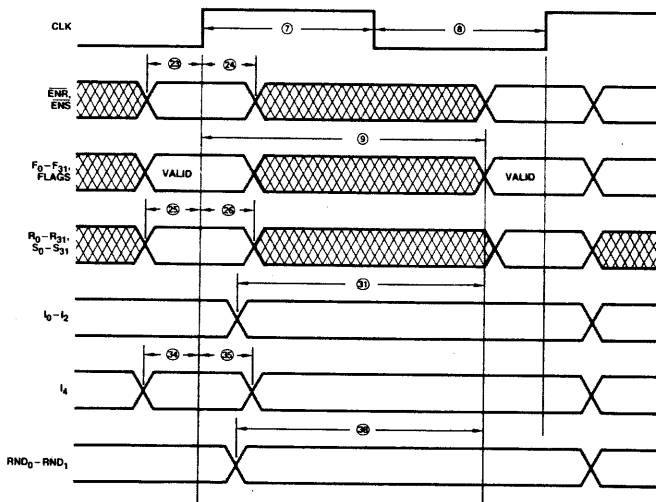
**Clocked Operation: $FT_0 = \text{LOW}$
 $FT_1 = \text{LOW}$**

SWITCHING WAVEFORMS (Cont'd.)



WF023770

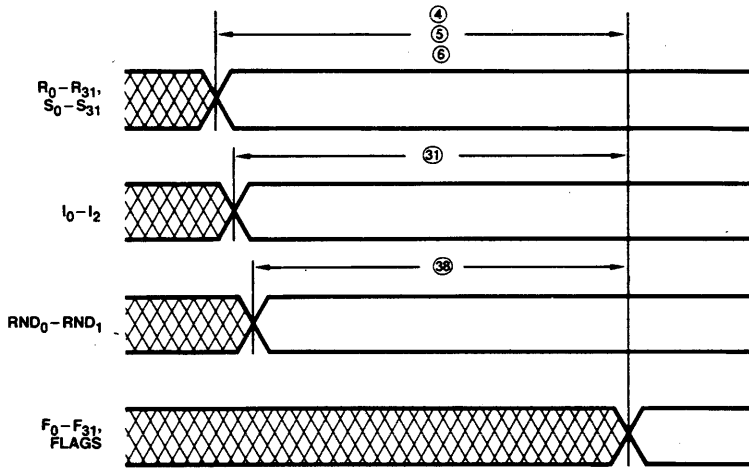
**Clocked Operation: $FT_0 = \text{HIGH}$
 $FT_1 = \text{LOW}$**



WF023780

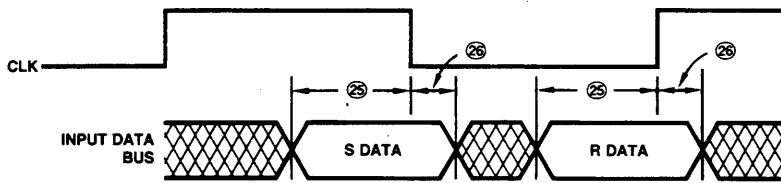
**Clocked Operation: $FT_0 = \text{LOW}$
 $FT_1 = \text{HIGH}$**

SWITCHING WAVEFORMS (Cont'd.)



WF023790

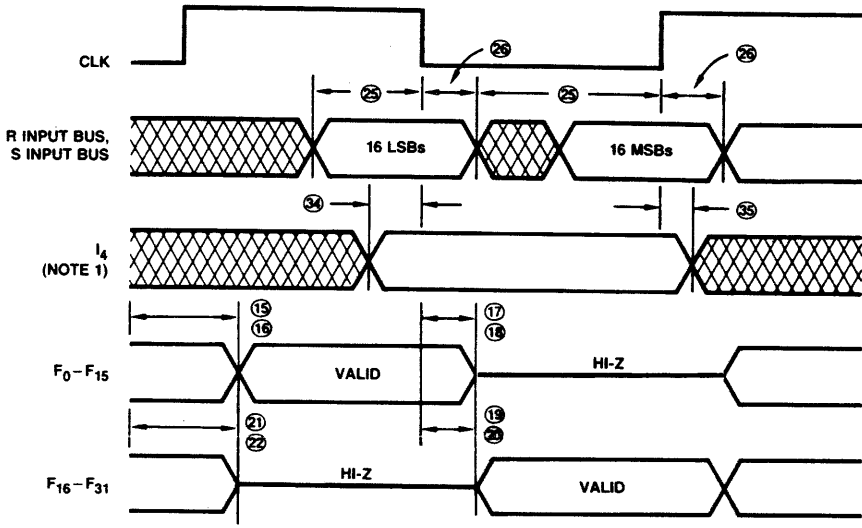
Flow-Through Operation ($FT_0 = \text{HIGH}$, $FT_1 = \text{HIGH}$)



WF023800

32-Bit, Single-Input Bus Mode

SWITCHING WAVEFORMS (Cont'd.)

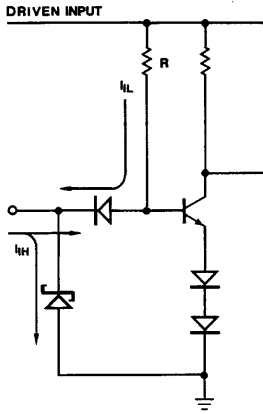


WF023810

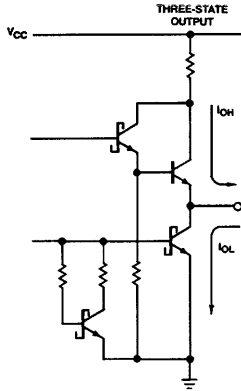
Note 1. I₄ has special setup and hold time requirements in this mode. All other control signals have timing requirements as shown in the diagram "Clocked operation, FT₀ = LOW, FT₁ = LOW."

16-Bit, Two-Input Bus Mode

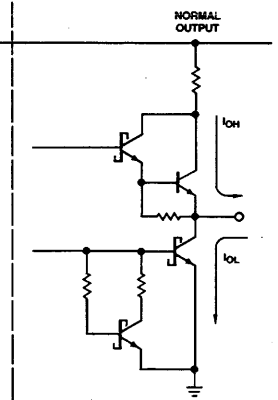
OUTPUT ENABLE/DISABLE TIMING



IC000960



IC000970



CLK, 16/32, OE
 R = 8KΩ
 ALL OTHER INPUTS
 R = 16KΩ

$C_1 \cong 5.0 \text{ pF}$, all inputs

$C_0 \cong 5.0 \text{ pF}$, all outputs
 Note: Actual current flow direction shown.

Am29C325

CMOS 32-Bit Floating-Point Processor



Am29C325

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- Single VLSI device performs high-speed floating-point arithmetic
 - Floating-point addition, subtraction, and multiplication in a single clock cycle
 - Internal architecture supports sum-of-products, Newton-Raphson division
- 32-bit, three-bus flow-through architecture
 - Programmable I/O allows interface to 32- and 16-bit systems
- IEEE and DEC formats
 - Performs conversions between formats
 - Performs integer ↔ floating-point conversions
- Input and output registers can be made transparent independently
- Pin and functionally compatible with the Bipolar Am29325
- The Am29C325 uses less than one-quarter the power of the Am29325
- 145 PGA requires no heatsink

GENERAL DESCRIPTION

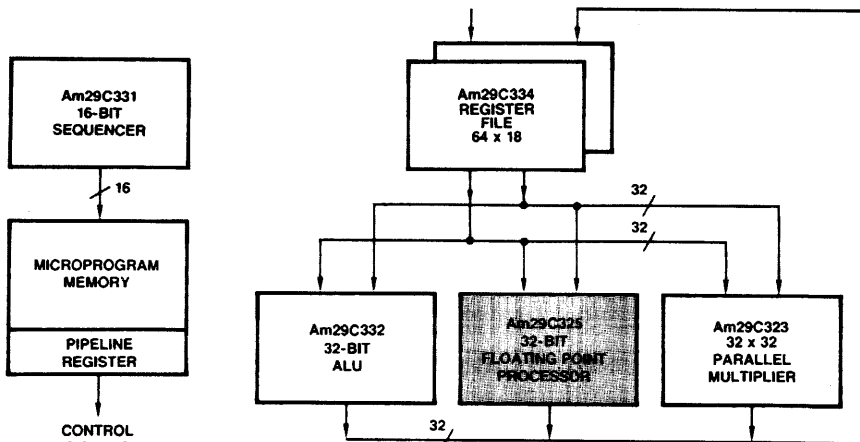
The Am29C325 is a high-speed floating-point processor unit. It performs 32-bit single-precision floating-point addition, subtraction, and multiplication operations in a single VLSI circuit, using the format specified by the proposed IEEE floating-point standard, 754. The DEC single-precision floating-point format is also supported. Operations for conversion between 32-bit integer format and floating-point format are available, as are operations for converting between the IEEE and DEC floating-point formats. Any operation can be performed in a single clock cycle. Six flags — invalid operation, inexact result, zero, not-a-number, overflow, and underflow — monitor the status of operations.

The Am29C325 has a three-bus, 32-bit architecture, with two input buses and one output bus. This configuration

provides high I/O bandwidth, allows access to all buses, and affords a high degree of flexibility when connecting this device in a system. All buses are registered, with each register having a clock enable. Input and output registers may be made transparent independently. Two other I/O configurations, a 32-bit, two-bus architecture and a 16-bit, three-bus architecture, are user-selectable, easing interface with a wide variety of systems. Thirty-two-bit internal feedforward datapaths support accumulation operations, including sum-of-products and Newton-Raphson division.

Fabricated using Advanced Micro Devices' 1.2 micron CMOS process, the Am29C325 is powered by a single 5-volt supply. The device is housed in a 145-lead pin-grid-array package.

Am29C300 FAMILY HIGH-PERFORMANCE SYSTEM BLOCK DIAGRAM



AF004651

This document contains information on a product under development at Advanced Micro Devices, Inc. The information is intended to help you to evaluate this product. AMD reserves the right to change or discontinue work on this product without notice.

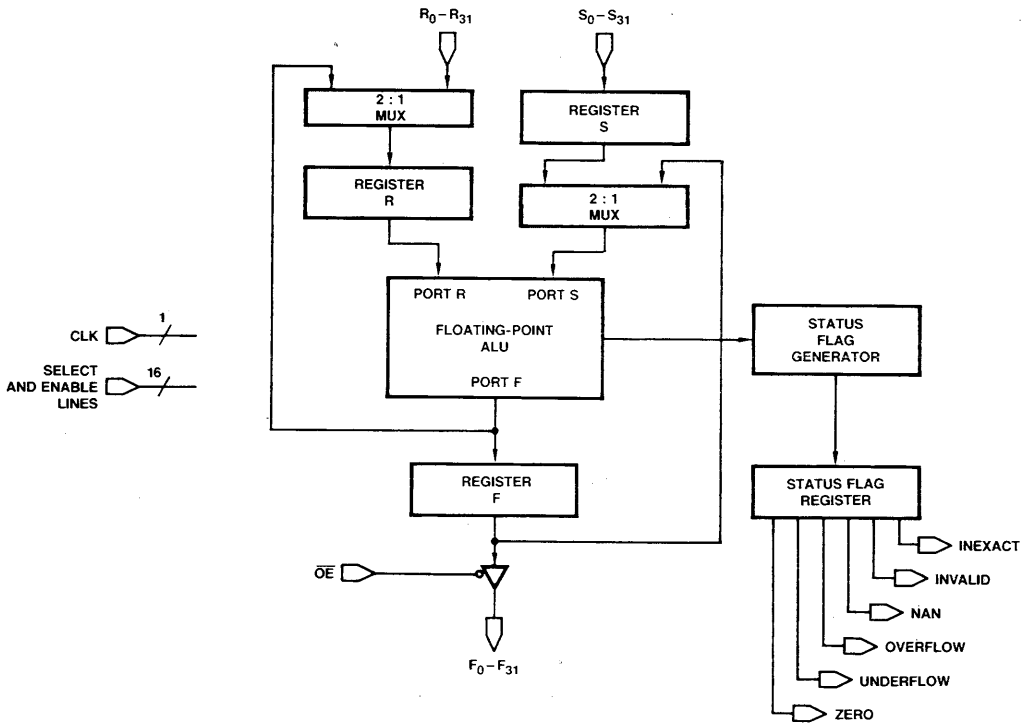
4-78

Publication #	Rev.	Amendment
07783	B	/0
Issue Date: September 1987		

RELATED AMD PRODUCTS

Part No.	Description
Am29114	Vectored Priority Interrupt Controller
Am29116	High-Performance Bipolar 16-Bit Microprocessor
Am29C116	High-Performance CMOS 16-Bit Microprocessor
Am29PL141	Fuse Programmable Controller
Am29C323	CMOS 32-Bit Parallel Multiplier
Am29331	16-Bit Microprogram Sequencer
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29332	32-Bit Extended Function ALU
Am29C332	CMOS 32-Bit Extended Function ALU
Am29334	64 x 18 Four-Port, Dual-Access Register File
Am29C334	CMOS 64 x 18 Four-Port, Dual-Access Register File
Am29337	16-Bit Bounds Checker
Am29338	Byte Queue

BLOCK DIAGRAM



BD007080

CONNECTION DIAGRAM Bottom View

PGA

	A	B	C	D	E	F	G	H	J	K	L	M	N	P	R
1	INEX	I2	I1	EN \bar{F}	I4	OBUS	O \bar{E}	VCC	CLK	R31	R30	R25	R24	R21	R20
2	INVA	NAN	I0	I \bar{D}	FT0	FT1	VCC	VCC	RND0	RND1	R27	R28	R23	R22	R17
3	F29	ZERO	GND	EN \bar{R}	EN \bar{S}	16/ $\bar{32}$	VCC	VCC	VCC	R29	R26	GND	GND	R19	R18
4	F30	F31	GND	*									R15	R16	R13
5	F23	OVFL	UNFL										R14	R11	R12
6	F26	F27	F28										R9	R10	R7
7	F21	F24	F25										R8	R5	R6
8	F22	F19	VCC										R3	R4	R1
9	F17	F20	VCC										R0	I3	R2
10	F18	F15	F16										S28	S31	S30
11	F13	F14	F11										S27	S26	S29
12	F12	F9	F10										VCC	S25	S24
13	F7	F6	GND	GND	GND	GND	GND	GND	GND	S8	S13	S14	VCC	S22	S23
14	F8	F3	F2	GND	F0	S1	S2	GND	S4	S9	S10	S15	S18	S21	S20
15	F5	F4	F1	GND	P/ $\bar{A}\bar{F}\bar{F}$	S0	S3	S5	S7	S6	S11	S12	S17	S16	S19

CD010491

Key:

- 16/ $\bar{32}$ = S16/ $\bar{32}$
- I \bar{D} = I $\bar{E}\bar{E}\bar{E}$ /DEC
- INEX = INEXACT
- INVA = INVALID
- OBUS = ONEBUS
- OVFL = OVERFLOW
- P/ $\bar{A}\bar{F}\bar{F}$ = PROJ/ $\bar{A}\bar{F}\bar{F}$
- UNFL = UNDERFLOW

*D4 is an alignment pin (not connected internally).

PIN DESIGNATIONS

(Sorted by Pin No.)

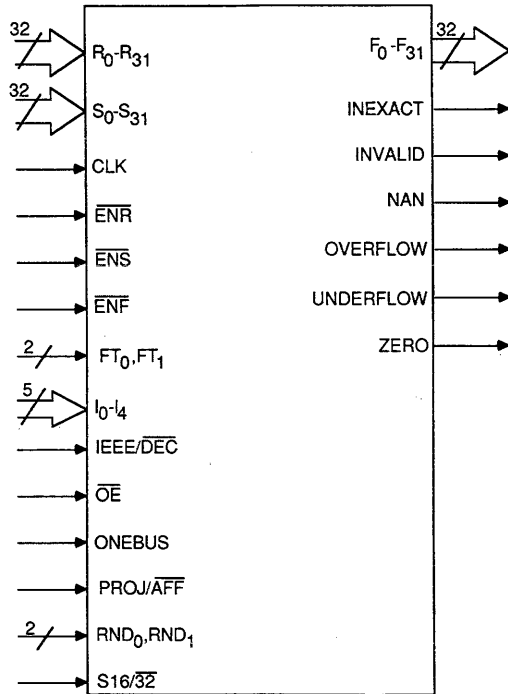
PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A-1	Inexact	C-7	F ₂₅	H-13	GND	N-10	S ₂₈
A-2	Invalid	C-8	V _{CC}	H-14	GND	N-11	S ₂₇
A-3	F ₂₉	C-9	V _{CC}	H-15	S ₅	N-12	V _{CC}
A-4	F ₃₀	C-10	F ₁₆	J-1	CLK	N-13	V _{CC}
A-5	F ₂₃	C-11	F ₁₁	J-2	RND ₀	N-14	S ₁₈
A-6	F ₂₆	C-12	F ₁₀	J-3	V _{CC}	N-15	S ₁₇
A-7	F ₂₁	C-13	GND	J-13	GND	P-1	R ₂₁
A-8	F ₂₂	C-14	F ₂	J-14	S ₄	P-2	R ₂₂
A-9	F ₁₇	C-15	F ₁	J-15	S ₇	P-3	R ₁₉
A-10	F ₁₈	D-1	$\overline{\text{ENF}}$	K-1	R ₃₁	P-4	R ₁₆
A-11	F ₁₃	D-2	$\overline{\text{IEEE/DEC}}$	K-2	RND ₁	P-5	R ₁₁
A-12	F ₁₂	D-3	$\overline{\text{ENR}}$	K-3	R ₂₉	P-6	R ₁₀
A-13	F ₇	D-13	GND	K-13	S ₈	P-7	R ₅
A-14	F ₈	D-14	GND	K-14	S ₉	P-8	R ₄
A-15	F ₅	D-15	GND	K-15	S ₆	P-9	I ₃
B-1	I ₂	E-1	I ₄	L-1	R ₃₀	P-10	S ₃₁
B-2	NAN	E-2	FT ₀	L-2	R ₂₇	P-11	S ₂₆
B-3	ZERO	E-3	$\overline{\text{ENS}}$	L-3	R ₂₆	P-12	S ₂₅
B-4	F ₃₁	E-13	GND	L-13	S ₁₃	P-13	S ₂₂
B-5	OVERFLOW	E-14	F ₀	L-14	S ₁₀	P-14	S ₂₁
B-6	F ₂₇	E-15	PROJ/AFF	L-15	S ₁₁	P-15	S ₁₆
B-7	F ₂₄	F-1	ONEBUS	M-1	R ₂₅	R-1	R ₂₀
B-8	F ₁₉	F-2	FT ₁	M-2	R ₂₈	R-2	R ₁₇
B-9	F ₂₀	F-3	S _{16/32}	M-3	GND	R-3	R ₁₈
B-10	F ₁₅	F-13	GND	M-13	S ₁₄	R-4	R ₁₃
B-11	F ₁₄	F-14	S ₁	M-14	S ₁₅	R-5	R ₁₂
B-12	F ₉	F-15	S ₀	M-15	S ₁₂	R-6	R ₇
B-13	F ₆	G-1	$\overline{\text{OE}}$	N-1	R ₂₄	R-7	R ₆
B-14	F ₃	G-2	V _{CC}	N-2	R ₂₃	R-8	R ₁
B-15	F ₄	G-3	V _{CC}	N-3	GND	R-9	R ₂
C-1	I ₁	G-13	GND	N-4	R ₁₅	R-10	S ₃₀
C-2	I ₀	G-14	S ₂	N-5	R ₁₄	R-11	S ₂₉
C-3	GND	G-15	S ₃	N-6	R ₉	R-12	S ₂₄
C-4	GND	H-1	V _{CC}	N-7	R ₈	R-13	S ₂₃
C-5	UNDERFLOW	H-2	V _{CC}	N-8	R ₃	R-14	S ₂₀
C-6	F ₂₈	H-3	V _{CC}	N-9	R ₀	R-15	S ₁₉

PIN DESIGNATIONS (Cont'd.)

(Sorted by Pin Name)

PIN NO.	PIN NAME	PIN NO.	PIN NAME.	PIN NO.	PIN NAME	PIN NO.	PIN NAME.
J-1	CLK	E-2	FT ₀	R-6	R ₇	K-14	S ₉
D-1	ENF	F-2	FT ₁	N-7	R ₈	L-14	S ₁₀
D-3	ENR	N-3	GND	N-6	R ₉	L-15	S ₁₁
E-3	ENS	H-14	GND	P-6	R ₁₀	M-15	S ₁₂
E-14	F ₀	G-13	GND	P-5	R ₁₁	L-13	S ₁₃
C-15	F ₁	M-3	GND	R-5	R ₁₂	M-13	S ₁₄
C-14	F ₂	H-13	GND	R-4	R ₁₃	M-14	S ₁₅
B-14	F ₃	J-13	GND	N-5	R ₁₄	P-15	S ₁₆
B-15	F ₄	D-15	GND	N-4	R ₁₅	F-3	S _{16/32}
A-15	F ₅	D-14	GND	P-4	R ₁₆	N-15	S ₁₇
B-13	F ₆	E-13	GND	R-2	R ₁₇	N-14	S ₁₈
A-13	F ₇	F-13	GND	R-3	R ₁₈	R-15	S ₁₉
A-14	F ₈	C-4	GND	P-3	R ₁₉	R-14	S ₂₀
B-12	F ₉	C-3	GND	R-1	R ₂₀	P-14	S ₂₁
C-12	F ₁₀	D-13	GND	P-1	R ₂₁	P-13	S ₂₂
C-11	F ₁₁	C-13	GND	P-2	R ₂₂	R-13	S ₂₃
A-12	F ₁₂	C-2	I ₀	N-2	R ₂₃	R-12	S ₂₄
A-11	F ₁₃	C-1	I ₁	N-1	R ₂₄	P-12	S ₂₅
B-11	F ₁₄	B-1	I ₂	M-1	R ₂₅	P-11	S ₂₆
B-10	F ₁₅	P-9	I ₃	L-3	R ₂₆	N-11	S ₂₇
C-10	F ₁₆	E-1	I ₄	L-2	R ₂₇	N-10	S ₂₈
A-9	F ₁₇	D-2	IEEE/DEC	M-2	R ₂₈	R-11	S ₂₉
A-10	F ₁₈	A-1	INEXACT	K-3	R ₂₉	R-10	S ₃₀
B-8	F ₁₉	A-2	INVALID	L-1	R ₃₀	P-10	S ₃₁
B-9	F ₂₀	B-2	NAN	K-1	R ₃₁	C-5	UNDERFLOW
A-7	F ₂₁	G-1	OE	J-2	RND ₀	J-3	V _{CC}
A-8	F ₂₂	F-1	ONEBUS	K-2	RND ₁	G-2	V _{CC}
A-5	F ₂₃	B-5	OVERFLOW	F-15	S ₀	G-3	V _{CC}
B-7	F ₂₄	E-15	PROJ/AFF	F-14	S ₁	H-2	V _{CC}
C-7	F ₂₅	N-9	R ₀	G-14	S ₂	N-13	V _{CC}
A-6	F ₂₆	R-8	R ₁	G-15	S ₃	N-12	V _{CC}
B-6	F ₂₇	R-9	R ₂	J-14	S ₄	H-3	V _{CC}
C-6	F ₂₈	N-8	R ₃	H-15	S ₅	H-1	V _{CC}
A-3	F ₂₉	P-8	R ₄	K-15	S ₆	C-8	V _{CC}
A-4	F ₃₀	P-7	R ₅	J-15	S ₇	C-9	V _{CC}
B-4	F ₃₁	R-7	R ₆	K-13	S ₈	B-3	ZERO

LOGIC SYMBOL



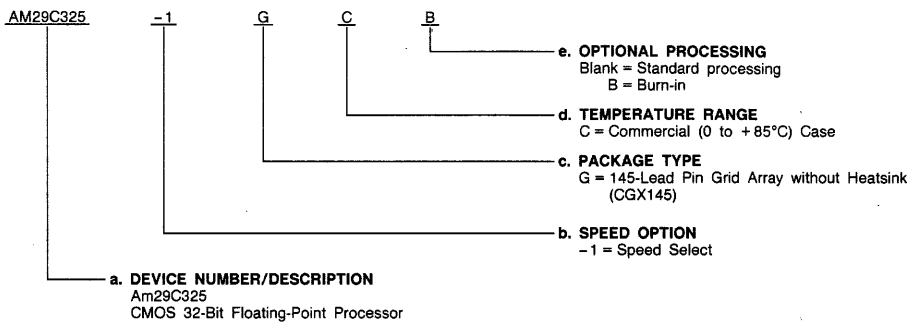
LS002920

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Package Type**
- d. **Temperature Range**
- e. **Optional Processing**



Valid Combinations	
Am29C325	GC, GCB
AM29C325-1	

Valid Combinations

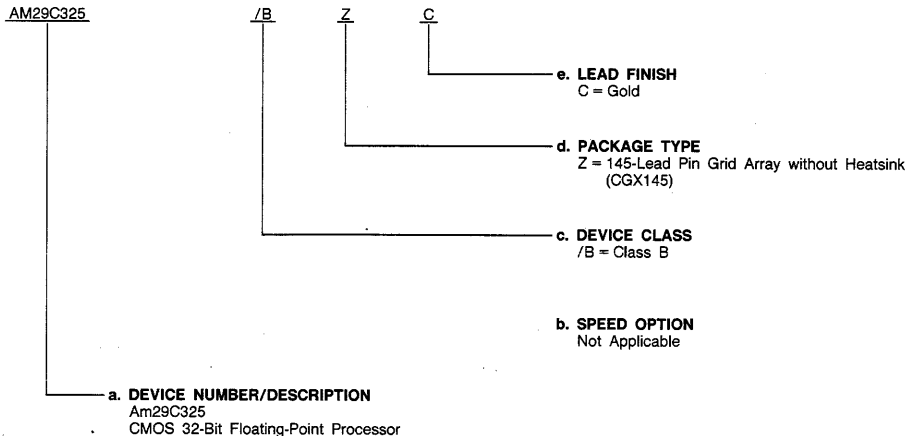
Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

MILITARY ORDERING INFORMATION

APL Products

AMD products for Aerospace and Defense applications are available in several packages and operating ranges. APL (Approved Products List) products are fully compliant with MIL-STD-883C requirements. The order number (Valid Combination) for APL products is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Device Class**
- d. **Package Type**
- e. **Lead Finish**



Valid Combinations	
AM29C325	/BZC

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations or to check for newly released valid combinations.

Group A Tests

Group A tests consist of Subgroups
1, 2, 3, 7, 8, 9, 10, 11.

PIN DESCRIPTION

CLK Clock (Input)

For the internal registers.

ENF Register F Clock Enable (Input; Active LOW)

When $\overline{\text{ENF}}$ is LOW, register F is clocked on the LOW-to-HIGH transition of CLK. When $\overline{\text{ENF}}$ is HIGH, register F retains the previous contents.

ENR Register R Clock Enable (Input; Active LOW)

When $\overline{\text{ENR}}$ is LOW, register R is clocked on the LOW-to-HIGH transition of CLK. When $\overline{\text{ENR}}$ is HIGH, register R retains the previous contents.

ENS Register S Clock Enable (Input; Active LOW)

When $\overline{\text{ENS}}$ is LOW, register S is clocked on the LOW-to-HIGH transition of CLK. When $\overline{\text{ENS}}$ is HIGH, register S retains the previous contents.

F₀-F₃₁ F Operand Bus (Output)

F₀ is the least-significant bit.

FT₀ Input Register Feedthrough Control (Input; Active HIGH)

When FT₀ is HIGH, registers R and S are transparent.

FT₁ Output Register Feedthrough Control (Input; Active HIGH)

When FT₁ is HIGH, register F and the status flag register are transparent.

I₀-I₂ Operation Select Lines (Input)

Used to select the operation to be performed by the ALU. See Table 1 for a list of operations and the corresponding codes.

I₃ ALU S Port Input Select (Input)

A LOW on I₃ selects register S as the input to the ALU S port. A HIGH on I₃ selects register F as the input to the ALU S port.

I₄ Register R Input Select (Input)

A LOW on I₄ selects R₀-R₃₁ as the input to register R. A HIGH selects the ALU F port as the input to register R.

IEEE/DEC IEEE/DEC Mode Select (Input)

When IEEE/DEC is HIGH, IEEE mode is selected. When IEEE/DEC is LOW, DEC mode is selected.

INEXACT Inexact Result Flag (Output; Active HIGH)

A HIGH indicates that the final result of the last operation was not infinitely precise, due to rounding.

INVALID Invalid Operation Flag (Output; Active HIGH)

A HIGH indicates that the last operation performed was invalid; e.g., ∞ times 0.

NAN Not-a-Number Flag (Output; Active HIGH)

A HIGH indicates that the final result produced by the last operation is not to be interpreted as a number. The output in such cases is either an IEEE Not-a-Number (NAN) or a DEC-reserved operand.

OE Output Enable (Input; Active LOW)

When $\overline{\text{OE}}$ is LOW, the contents of register F are placed on F₀-F₃₁. When $\overline{\text{OE}}$ is HIGH, F₀-F₃₁ assume a high-impedance state.

ONEBUS Input Bus Configuration Control (Input)

A LOW on ONEBUS configures the input bus circuitry for two-input bus operation. A HIGH on ONEBUS configures the input bus circuitry for single-input bus operation.

OVERFLOW Overflow Flag (Output; Active HIGH)

A HIGH indicates that the last operation produced a final result that overflowed the floating-point format.

PROJ/AFF Projective/Affine Mode Select (Input)

Choice of projective or affine mode determines the way in which infinities are handled in IEEE mode. A LOW on PROJ/AFF selects affine mode; a HIGH selects projective mode.

R₀-R₃₁ R Operand Bus (Input)

R₀ is the least-significant bit.

RND₀, RND₁ Rounding Mode Selects (Input)

RND₀ and RND₁ select one of four rounding modes. See Table 5 for a list of rounding modes and the corresponding control codes.

S₀-S₃₁ S Operand Bus (Input)

S₀ is the least-significant bit.

S16/32 16- or 32-Bit I/O Mode Select (Input)

A LOW on S16/32 selects the 32-bit I/O mode; a HIGH selects the 16-bit I/O mode. In 32-bit mode, input and output buses are 32 bits wide. In 16-bit mode, input and output buses are 16 bits wide, with the least- and most-significant portions of the 32-bit input and output words being placed on the buses during the HIGH and LOW portions of CLK, respectively.

UNDERFLOW Underflow Flag (Output; Active HIGH)

A HIGH indicates that the last operation produced a rounded result that underflowed the floating-point format.

ZERO Zero Flag (Output; Active HIGH)

A HIGH indicates that the last operation produced a final result of zero.

Definition of Terms

Affine Mode

One of two modes affecting the handling of operations on infinities — see the **Operations with Infinities** section under **Operations in IEEE Mode**.

Biased Exponent

The true exponent of a floating-point number, plus a constant. For IEEE floating-point numbers, the constant is 127; for DEC floating-point numbers, the constant is 128. See also **True Exponent**.

Bus

Data input or output channel for the floating-point processor.

DEC-Reserved Operand

A DEC floating-point number that is interpreted as a symbol and has no numeric value. A DEC-reserved operand has a sign of 1 and a biased exponent of 0.

Destination Format

The format of the final result produced by the floating-point ALU. The destination format can be IEEE floating point, DEC floating point, or integer.

Final Result

The result produced by the floating-point ALU.

Fraction

The 23 least-significant bits of the mantissa.

Infinitely Precise Result

The result that would be obtained from an operation if both exponent range and precision were unbounded.

Input Operands

The value or values on which an operation is performed. For example, the addition $2 + 3 = 5$ has input operands 2 and 3.

Mantissa

The portion of a floating-point number containing the number's significant bits. For the floating-point number 1.101×2^{-3} , the mantissa is 1.101.

NAN (Not-a-Number)

An IEEE floating-point number that is interpreted as a symbol, and has no numeric value. A NAN has a biased exponent of 255_{10} and a non-zero fraction.

Port

Data input or output channel for the floating-point ALU.

Projective Mode

One of two modes affecting the handling of operations on infinities — see the **Operations with Infinities** section under **Operation in IEEE Mode**.

Rounded Result

The result produced by rounding the infinitely precise result to fit the destination format.

True Exponent (or Exponent)

Number representing the power of two by which a floating-point number's mantissa is to be multiplied. For the floating-point number 1.101×2^{-3} , the true exponent is -3 .

FUNCTIONAL DESCRIPTION

Architecture

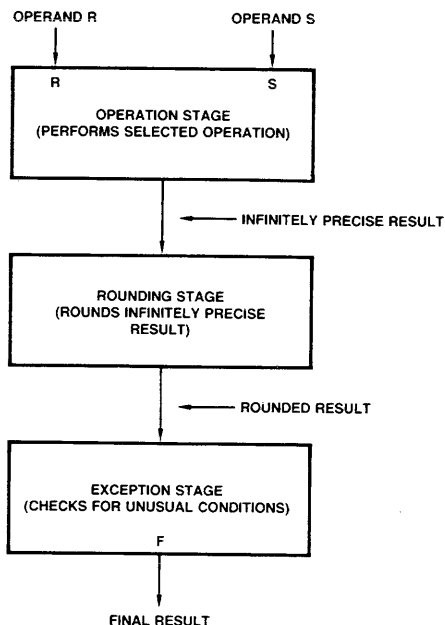
The Am29C325 comprises a high-speed, floating-point ALU, a status flag generator, and a 32-bit data path.

Floating-Point ALU

The floating-point ALU performs 32-bit floating-point operations. It also performs floating-point-to-integer conversions, integer-to-floating-point floating-point conversions, and conversions between the IEEE and DEC formats. The ALU has two 32-bit input ports, R and S, and a 32-bit output port, F.

Conceptually, the process performed by the ALU can be divided into three stages (see Figure 1). The operation stage performs the arithmetic operation selected by the user; the output of this section is referred to as the infinitely precise result of the operation. The rounding stage rounds the infinitely precise result to fit in the destination format; the output of this stage is called the rounded result. The last stage checks for exceptional conditions. If no exceptional condition is found, the rounded result is passed through this stage. If some exceptional condition is found (e.g., overflow, underflow, or an invalid operation), this section may replace the rounded result with another output, such as $+\infty$, $-\infty$, a NAN, or a DEC-

reserved operand. The output of this last stage appears on port F, and is called the final result.



AF004540

Figure 1. Conceptual Model of the Process Performed by the Floating-Point ALU

The ALU performs one of eight operations; the operation to be performed is selected by placing the appropriate control code on lines $I_0 - I_2$. Table 1 gives the control codes corresponding to each of the eight operations.

The floating-point addition operation (R PLUS S) adds the floating-point numbers on ports R and S, and places the floating-point result on port F. In IEEE mode (IEEE/DEC = HIGH) the addition is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the addition is performed in DEC format.

The floating-point subtraction operation (R MINUS S) subtracts the floating-point number on port S from the floating-point number on port R and places the floating-point result on port F. In IEEE mode (IEEE/DEC = HIGH) the subtraction is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the subtraction is performed in DEC format.

The floating-point multiplication operation (R TIMES S) multiplies the floating-point numbers on ports R and S, and places the floating-point result on port F. In IEEE mode (IEEE/DEC = HIGH) the multiplication is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the multiplication is performed in DEC format.

The floating-point constant subtraction (2 MINUS S) operation subtracts the floating-point value on port S from 2, and places the result on port F. The operand on port R is not used in this operation; its value will not affect the operation in any way. In IEEE mode (IEEE/DEC = HIGH) the operation is performed in IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) the operation is performed in DEC format. This operation is

used to support Newton-Raphson floating-point division; a description of its use appears in **Appendix C**.

The integer-to-floating-point conversion (INT-TO-FP) operation takes a 32-bit, two's-complement integer on port R and places the equivalent floating-point value on port F. The

operand on port S is not used in this operation; its value will not affect the operation in any way. In IEEE mode (IEEE/DEC = HIGH) the result is delivered in IEEE format; in DEC mode (IEEE/DEC = LOW) the result is delivered in DEC format.

TABLE 1. ALU OPERATION SELECT

I_2	I_1	I_0	Operation	Output Equation
0	0	0	Floating-point addition (R PLUS S)	$F = R + S$
0	0	1	Floating-point subtraction (R MINUS S)	$F = R - S$
0	1	0	Floating-point multiplication (R TIMES S)	$F = R * S$
0	1	1	Floating-point constant subtraction (2 MINUS S)	$F = 2 - S$
1	0	0	Integer-to-floating-point conversion (INT-TO-FP)	F (floating-point) = R (integer)
1	0	1	Floating-point-to-integer conversion (FP-TO-INT)	F (integer) = R (floating-point)
1	1	0	IEEE-TO-DEC format conversion (IEEE-TO-DEC)	F (DEC format) = R (IEEE format)
1	1	1	DEC-TO-IEEE format conversion (DEC-TO-IEEE)	F (IEEE format) = R (DEC format)

The floating-point-to-integer conversion (FP-TO-INT) operation takes a floating-point number on port R and places the equivalent 32-bit, two's-complement integer value on port F. The operand on port S is not used in this operation; its value will not affect the operation in any way. In IEEE mode (IEEE/DEC = HIGH) the operand on port R is interpreted using the IEEE floating-point format; in DEC mode (IEEE/DEC = LOW) it is interpreted using the DEC floating-point format.

The IEEE-to-DEC conversion operation (IEEE-TO-DEC) takes an IEEE-format floating-point number on port R and places the equivalent DEC-format floating-point number on port F. The operand on port S is not used in this operation; its value will not affect the operation in any way. The operation can be performed in either IEEE mode (IEEE/DEC = HIGH) or DEC mode (IEEE/DEC = LOW).

The DEC-to-IEEE conversion operation (DEC-TO-IEEE) takes a DEC-format floating-point number on port R and places the equivalent IEEE-floating-point number on port F. The operand on port S is not used in this operation; its value will not affect the operation in any way. The operation can be performed in either IEEE mode (IEEE/DEC = HIGH) or DEC mode (IEEE/DEC = LOW).

Status Flag Generator

The status flag generator controls the state of six flags that report the status of floating-point ALU operations. The flags indicate when an operation is invalid (e.g., ∞ times 0) or when an operation has produced an overflow, an underflow, a non-numerical result (e.g., a NAN- or DEC-reserved operand), an inexact result, or a result of zero. The flags represent the status of the most recently performed operation. Flag status is stored in the flag status register on the LOW-to-HIGH transition of CLK. When the output register feedthrough control FT₁ is HIGH, the flag status register is made transparent.

Data Path

The 32-bit data path consists of the R and S input buses; the F output bus; data registers R, S, and F; the register R input multiplexer; and the ALU port S input multiplexer.

Input operands enter the floating-point processor through the 32-bit R and S input buses, R₀ - R₃₁ and S₀ - S₃₁. Results of operations appear on the 32-bit F bus, F₀ - F₃₁. The F bus assumes a high-impedance state when output enable OE is HIGH.

The R and S registers store input operands; the F register stores the final result of the floating-point ALU operation. Each register has an independent clock enable (ENR, ENS, and ENF). When a register's clock enable is LOW, the register stores the data on its input at the LOW-to-HIGH transition of CLK; when the clock enable is HIGH, the register retains its current data. All data registers are fully edge-triggered — both the input data and the register enable need only meet modest setup and hold time requirements. Registers R and S can be made transparent by setting FT₀, the input register feedthrough control, HIGH. Register F can be made transparent by setting FT₁, the output register feedthrough control, HIGH.

The register R input multiplexer selects either the R input bus or the floating-point ALU's F port as the input to register R. Selection is controlled by I₄ — a LOW selects the R input bus; a HIGH selects the ALU F port. The ALU port S input multiplexer selects either register S or register F as the input to the floating-point ALU's S port. Selection is controlled by I₃ — a LOW selects register S; a HIGH selects register F.

Data selected by I₃ and I₄ is described in Table 2. When registers R and S are transparent (FT₀ = HIGH), multiplexer select I₄ must be kept LOW, so that the register R input multiplexer selects R₀ - R₃₁. When register F is transparent (FT₁ = HIGH), multiplexer select I₃ must be kept LOW, so that the ALU port S input multiplexer selects register S.

TABLE 2. MUX SELECT

I_3	Data selected for floating-point ALU S port
0	Register S
1	Register F
I_4	Data selected for register R input
0	R bus
1	Floating-point ALU port F

TABLE 3. I/O MODE SELECTION

S16/32	ONEBUS	I/O Mode
0	0	32-bit, two-input-bus mode
0	1	32-bit, single-input-bus mode(*)
1	0	16-bit, two-input-bus mode(*)
1	1	Illegal I/O mode selection value

*FT₀ must be held LOW in this mode (see text).

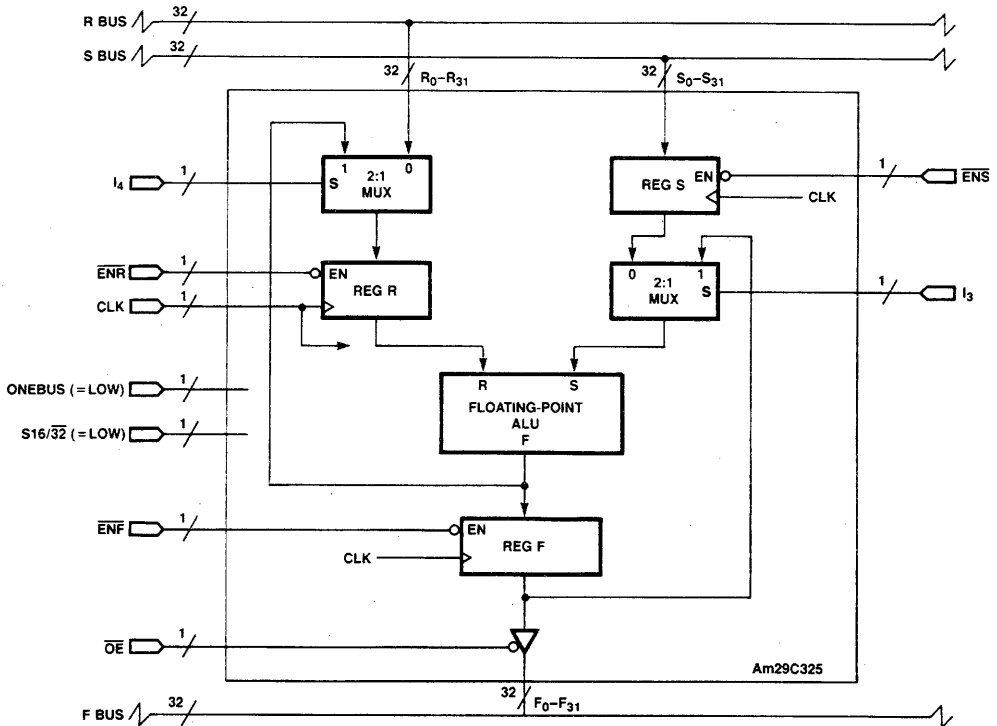
I/O Modes

The Am29C325 datapath can be configured in one of three I/O modes: a 32-bit, two-input bus mode; a 32-bit, single-input bus mode; and a 16-bit, two-input bus mode. These modes affect only the manner in which data is delivered to and taken from the Am29C325; operation of the floating-point ALU is not altered. The I/O mode is selected with the ONEBUS and S16/32 controls. Table 3 lists the control codes needed to invoke each I/O mode.

32-Bit, Two-Input Bus Mode

In this I/O mode, the R and S buses are configured as independent 32-bit input buses, and the F bus is configured as a 32-bit output bus. Figure 2 is a functional block diagram of the Am29C325 in this I/O mode.

R and S operands are taken from their respective input buses and clocked into the R and S registers on the LOW-to-HIGH transition of CLK. Register F is also clocked on the LOW-to-HIGH transition of CLK. Figure 5(a) depicts typical I/O timing in this mode.



BD007051

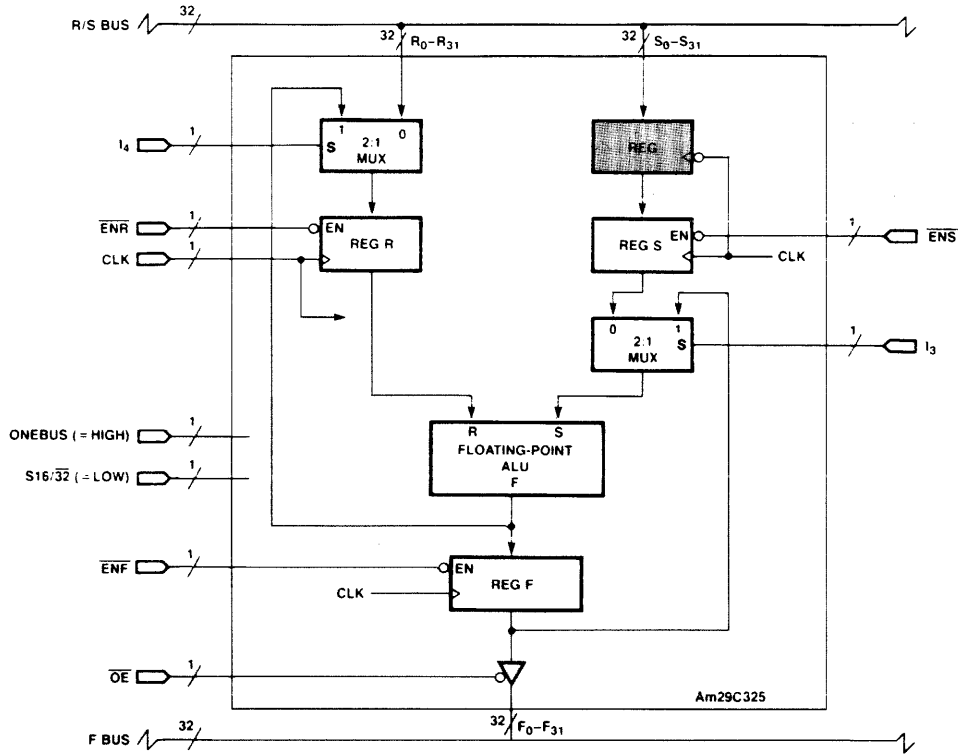
Figure 2. Functional Block Diagram for the 32-Bit, Two-Input Bus Mode

32-Bit, Single-Input Bus Mode

In this I/O mode, the R and S buses are connected to a single 32-bit multiplexed input data bus; the F bus is configured as an independent 32-bit output bus. Figure 3 is a functional block diagram of the Am29C325 in this I/O mode. Note that both the R and S bus lines must be wired to the input bus.

R and S operands are multiplexed onto the input bus by the host system. The S operand is clocked from the input bus into a temporary holding register in this I/O mode. The R operand is clocked on the LOW-to-HIGH transition of CLK. Register F is clocked on the LOW-to-HIGH transition of CLK. Figure 5(b) depicts typical I/O timing in this mode.

When placed in this I/O mode, the data path will not function properly if the R and S registers are made transparent. Therefore, input register feedthrough control FT₀ must be held LOW in this mode.



BD007061

Figure 3. Functional Block Diagram for the 32-Bit, Single-Input Bus Mode

16-Bit, Two-Input Bus Mode

In this I/O mode, the R and S buses are configured as independent 16-bit input buses, and the F bus is configured as a 16-bit output bus. Figure 4 is a functional block diagram of the Am29C325 in this I/O mode. Note that the 16 least-significant bits (LSBs) and 16 most-significant bits (MSBs) of the R, S, and F buses must be wired to their respective system buses in parallel.

Thirty-two-bit operands are passed along the 16-bit data buses by time-multiplexing the 16 LSBs and 16 MSBs of each 32-bit word. For the R input bus, the host system multiplexes the 16 LSBs and 16 MSBs of the R operand onto the 16-bit R bus. The 16 LSBs of the R operand are stored in a temporary holding register on the HIGH-to-LOW transition of CLK. The 16 MSBs are clocked into register R on the LOW-to-HIGH transition of CLK; at the same time, the 16 LSBs are transferred from the temporary holding register to register R. Transfer of data from the S input bus to the S register takes place in a similar fashion. Register F is clocked on the LOW-to-HIGH transition of CLK. Circuitry internal to the Am29C325 multiplexes data from register F onto the 16-bit output bus by enabling the 16 LSBs of the F output bus when CLK is HIGH, and enabling the 16 MSBs of the F output bus when CLK is LOW. Figure 5(c) depicts typical I/O timing in this mode.

When placed in this I/O mode, the data path will not function properly if the R and S registers are made transparent. Therefore, input register feedthrough control FT₀ must be held LOW in this mode. Caution must also be taken in controlling the register R input multiplexer control line, I₄, in this I/O mode. I₄ should be changed only when CLK is HIGH, in

addition to meeting the setup and hold time requirements given in the **Switching Characteristics** section.

Operation in IEEE Mode

When input signal IEEE/ \overline{DEC} is HIGH, the IEEE mode of operation is selected. In this mode the Am29C325 uses the floating-point format set forth in the IEEE Proposed Standard for Binary Floating-Point Arithmetic, P754. In addition, the IEEE mode complies with most other aspects of single-precision floating-point operation outlined in the proposed standard — differences are discussed in **Appendix A**.

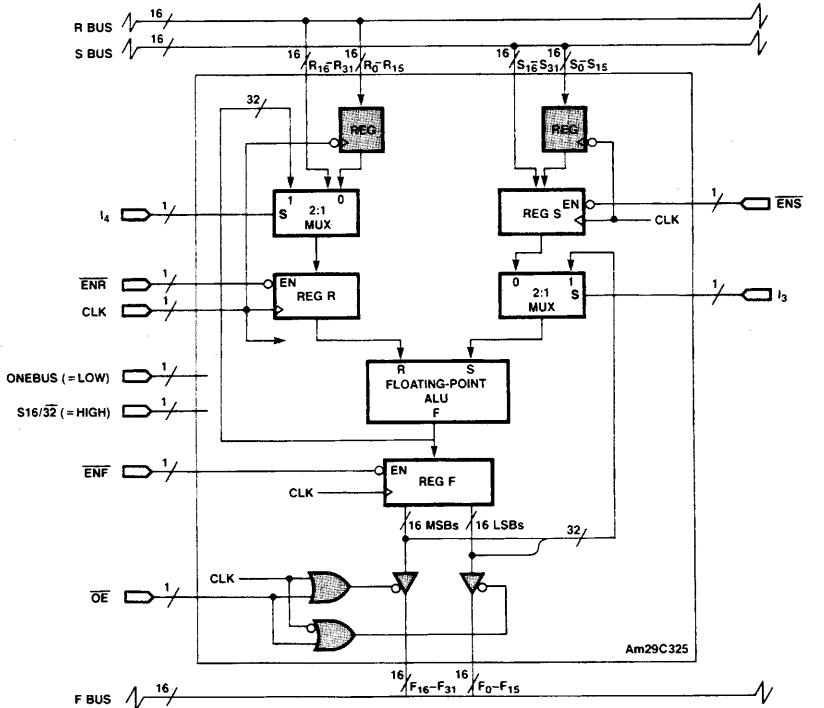
IEEE Floating-Point Format

The IEEE single-precision floating-point word is 32 bits wide, and is arranged in the format shown in Figure 6. The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 23-bit fraction.

The sign bit indicates the sign of the floating-point number's value. Non-negative values have a sign of 0; negative values, a sign of 1. The value zero may have either sign.

The biased exponent is an 8-bit unsigned integer field representing a multiplicative factor of some power of two. The bias value is 127. If, for example, the multiplicative factor for a floating-point number is to be 2^a , the value of the biased exponent would be $a + 127$; "a" is called the true exponent.

The fraction is a 23-bit unsigned fraction field containing the 23 LSBs of the floating-point number's 24-bit mantissa. The weight of fraction's MSB is 2^{-1} ; the weight of the LSB is 2^{-23} .



BD007071

Figure 4. Functional Block Diagram for the 16-Bit, Two-Input Bus Mode

A floating-point number is evaluated or interpreted per the following conventions:

let s = sign bit
 e = biased exponent
 f = fraction

if $e = 0$ and $f = 0$...value = $(-1)^s * 0$ (+ 0, -0)

if $e = 0$ and $f \neq 0$...value = denormalized number

if $0 < e < 255$...value = $(-1)^s * (2^e - 127) * (1.f)$
(normalized number)

if $e = 255$ and $f = 0$...value = $(-1)^s * (\infty)$ (+ ∞ , - ∞)

if $e = 255$ and $f \neq 0$...value = not-a-number (NaN)

Zero: The value zero can have either a positive or negative sign. Rules for determining the sign of a zero produced by an operation are given in the **Sign Bit** section.

Denormalized Number: A denormalized number represents a quantity with magnitude less than 2^{-126} but greater than zero.

Normalized Number: A normalized number represents a quantity with magnitude greater than or equal to 2^{-126} but less than 2^{128} .

Example 1:

The number +3.5 can be represented in floating-point format as follows:

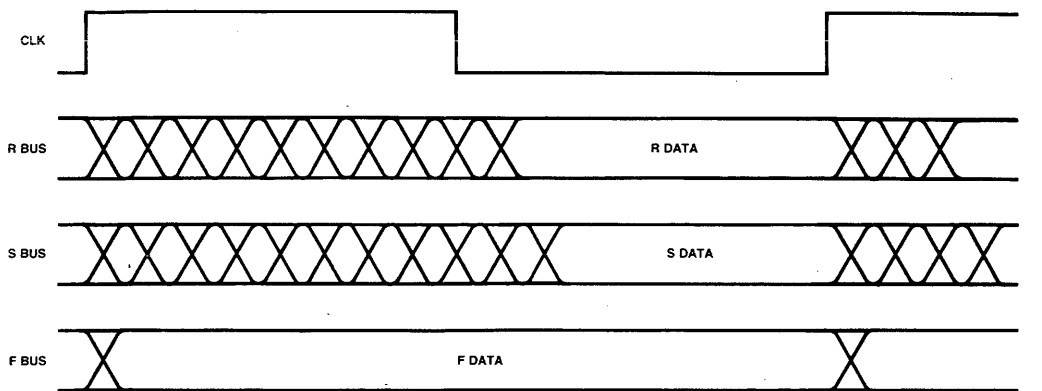
$$+3.5 = 11.1_2 \times 2^0 \\ = 1.11_2 \times 2^1$$

sign = 0

$$\text{biased exponent} = 1_{10} + 127_{10} = 128_{10} \\ = 10000000_2$$

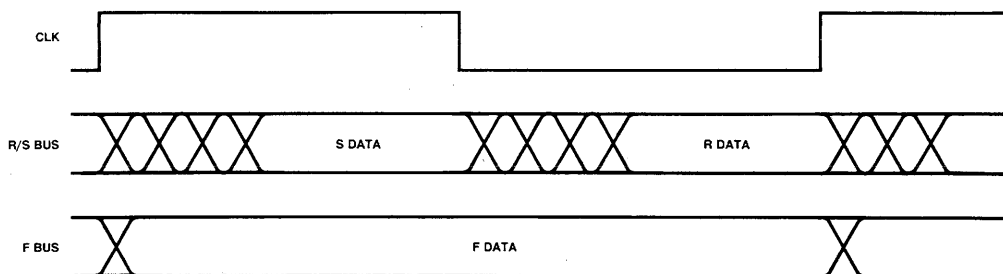
fraction = 11000000000000000000000000000000_2
(the leading 1 is implied in the format)

Concatenating these fields produces the floating-point word 40600000₁₆.



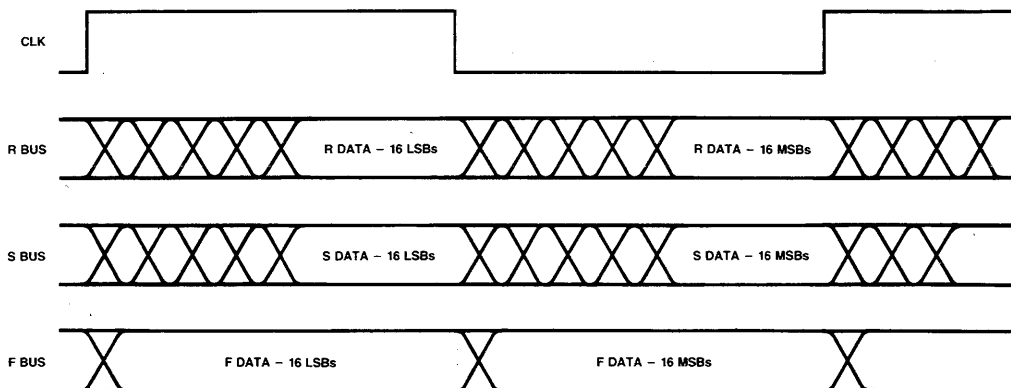
WF023730

a) 32-Bit, Two-Input-Bus Mode



WF023740

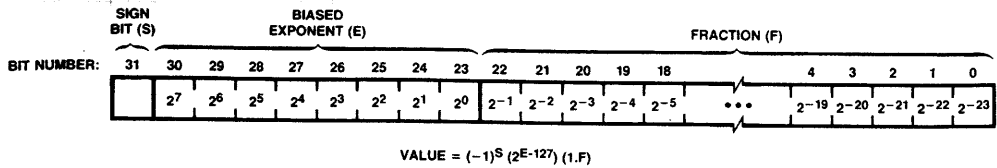
b) 32-Bit, Single-Input-Bus Mode



WF023750

c) 16-Bit, Two-Input-Bus Mode

Figure 5. Typical Bus Timing for the I/O Modes with $FT_0 = \text{LOW}$, $FT_1 = \text{LOW}$



TB000640

Figure 6. IEEE Mode Single-Precision Floating-Point Format

Example 2:

The number -11.375 can be represented in floating-point format as follows:

$$-11.375 = -1011.011_2 \times 2^0$$

$$= -1.011011_2 \times 2^3$$

sign = 1

$$\text{biased exponent} = 3_{10} + 127_{10} = 130_{10}$$

$$= 10000010_2$$

fraction = 011011000000000000000000
(the leading 1 is implied in the format)

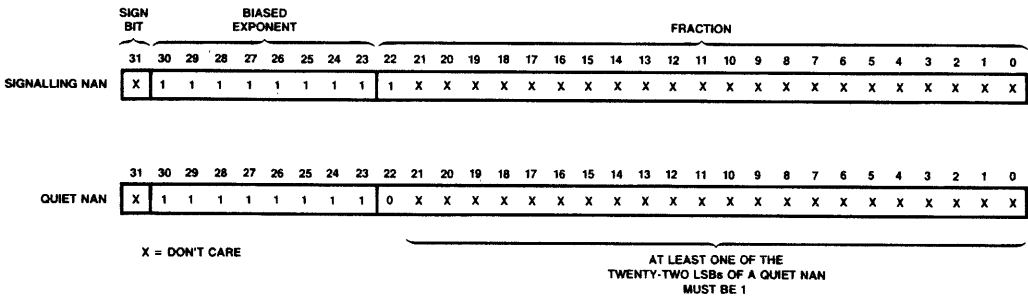
Concatenating these fields produces the floating-point word C1360000₁₆.

Infinity: Infinity can have either a positive or negative sign. The way in which infinities are interpreted is determined by the state of the projective/affine mode select, PROJ/AFF.

Not-a-Number: A not-a-number, or NAN, does not represent a numeric value, but is interpreted as a signal or symbol. NANs are used to indicate invalid operations, and as a means of passing process status information through a series of calculations. NANs arise in two ways: 1) they can be generated by the Am29C325 to indicate that an invalid operation has taken place (e.g., $\infty \times 0$), or 2) be provided by the user as an input operand. There are two types of NANs, signalling and quiet (see Figure 7 for formats).

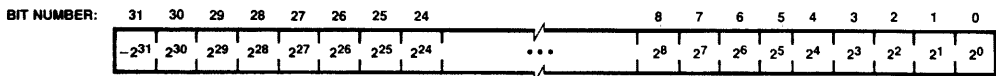
IEEE Mode Integer Format

Integer numbers are represented as 32-bit, two's-complement words (Figure 8 depicts the integer format). The integer word can represent a range of integer values from -2^{31} to $2^{31} - 1$.



TB000650

Figure 7. Signalling and Quiet NAN Formats



TB000660

Figure 8. 32-Bit Integer Format

Operations

All eight floating-point ALU operations discussed in the **Functional Description** section can be performed in IEEE mode. Various exceptional aspects of the R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, and FP-TO-INT operations for this mode are described below. The IEEE-TO-DEC and DEC-TO-IEEE operations are discussed separately in the **IEEE-TO-DEC AND DEC-TO-IEEE Operations** section.

Operations with NANs: NANs arise in two ways: 1) they can be generated by the Am29C325 to indicate that an invalid operation has taken place (e.g., $\infty \times 0$), or 2) be provided by the user as an input operand. There are two types of NANs, signalling and quiet (see Figure 7 for formats).

Signalling NANs set the invalid operation flag when they appear as an input operand to an operation. They are useful for indicating uninitialized variables, or for implementing user-

designed extensions to the operations provided. The ALU never produces a signalling NAN as the final result of an operation.

Quiet NANs are generated for invalid operations. When they appear as an input operand, they are passed through most operations without setting the invalid flag, the floating-point-to-integer conversion operation being the exception.

The sign of any input operand NAN is ignored. All quiet NANs produced as the final result of an operation have a sign of 0.

When a NAN appears as an input operand, the final result of the operation is a quiet NAN that is created by taking the input NAN and forcing bit 22 LOW and bit 21 HIGH. If an operation has two NANs as input operands, the resulting quiet NAN is created using the NAN on the R port.

When a quiet NAN is produced as the final result of an invalid operation whose input operand or operands are not NANs, the resulting NAN will always have the value 7FA00000₁₆.

The NAN flag will be HIGH whenever an operation produces a NAN as a final result.

Example 1:

Suppose the floating-point addition operation is performed with the following input operands:

R port: 3F800000₁₆ (1.0×2^0)
S port: 7FC12345₁₆ (signalling NAN)

Result: The signalling NAN on the S port is converted to a quiet NAN by forcing bit 22 LOW and bit 21 HIGH. The operation's final result will be 7FA12345₁₆. Since one of the two input operands is a signalling NAN, the invalid flag will be HIGH; the NAN flag will also be HIGH.

Example 2:

Suppose the floating-point multiplication operation is performed with the following input operands:

R port: FFF11111₁₆ (signalling NAN)
S port: 7FC22222₁₆ (quiet NAN)

Result: Since both input operands are NANs, the NAN on the R port is chosen for output. In addition to forcing bit 22 LOW, the sign bit (bit 31) is set LOW (bit 21 is already HIGH, and need not be changed). The operation's final result will be 7FB11111₁₆. Since one of the two input operands is a signalling NAN, the invalid flag is HIGH; the NAN flag will also be HIGH.

Example 3:

Suppose the floating-point subtraction operation is performed with the following input operands:

R port: FF800001₁₆ (quiet NAN)
S port: 7F800000₁₆ (+∞)

Result: To create the final result, the quiet NANs sign bit (bit 31) is forced LOW and bit 21 is forced HIGH (bit 22 is already LOW, and need not be changed). The final result will be 7FA00001₁₆. The NAN flag will be HIGH.

Operations with Denormalized Numbers: The proposed IEEE standard incorporates denormalized numbers to allow a means of gradual underflow for operations that produce non-zero results too small to be expressed as a normalized floating-point number. The Am29C325 does not support gradual underflow. If a floating-point operation produces a non-zero rounded result that is not large enough to be expressed as a normalized floating-point number, the final

result will be a zero of the same sign; the inexact, underflow, and zero flags will be HIGH. If an input operand is a denormalized number, the floating-point ALU will assume that operand to be a zero of the same sign.

Operations Producing Overflows: If an operation has a finite input operand or operands, and if the operation produces a rounded result that is too large to fit in the destination format, the operation is said to have overflowed.

A floating-point overflow occurs if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation with finite input operand(s) produces a result which, after rounding, has a magnitude greater than or equal to 2^{128} . Positive or negative infinity will appear as the final result if the rounded result is positive or negative, respectively, and the overflow and inexact flags will be HIGH.

Integer overflow occurs when the floating-point-to-integer conversion operation attempts to convert a number which, after rounding, is greater than $2^{31} - 1$ or less than -2^{31} . The final result will be quiet NAN 7FA00000₁₆, and the invalid operation and NAN flags will be HIGH. Note that the overflow and inexact flags remain LOW for integer overflow.

Operations Producing Underflows: If an operation produces a floating-point rounded result having a magnitude too small to be expressed as a normalized floating-point number, but greater than zero, that operation is said to have underflowed. Underflow occurs when an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has a magnitude in the range:

$$0 < \text{magnitude} < 2^{-126}$$

In such cases, the final result will be +0 (00000000₁₆) if the rounded result is non-negative, and -0 (80000000₁₆) if the rounded result is negative. The underflow, inexact, and zero flags will be HIGH.

Underflow does not occur if the destination format is integer. If the infinitely precise result of a floating-point-to-integer conversion has a magnitude greater than 0 and less than 1, but the rounded result is 0, the underflow flag remains LOW.

Operations with Infinities: In most cases, positive and negative infinity are valid inputs for the R PLUS S, R MINUS S, R TIMES S, and 2 MINUS S operations. Those cases for which infinities are not valid inputs for these operations are listed in Table 4.

Infinities in IEEE mode can be handled either as projective or affine. The projective mode is selected when PROJ/AFF is HIGH; the affine mode is selected when PROJ/AFF is LOW. The only differences between the modes that are relevant to Am29C325 operation occur during the addition and subtraction of infinities:

Operation	Affine Mode	Projective Mode
$(+\infty) + (+\infty)$	Output $+\infty$	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags
$(-\infty) + (-\infty)$	Output $-\infty$	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags
$(+\infty) - (-\infty)$	Output $+\infty$	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags
$(-\infty) - (+\infty)$	Output $-\infty$	Output 7FA00000 ₁₆ (quiet NAN), set invalid and NAN flags

If an R PLUS S, R MINUS S, or 2 MINUS S operation has infinity as an input operand or operands, the final result, if valid, is presumed to be exact. For example, adding $+\infty$ and 2.0 will produce a final result of $+\infty$; since the result is considered exact, the inexact flag remains LOW.

Invalid Operations: If an input operand is invalid for the operation to be performed, that operation is considered invalid. When an invalid operation is performed, the floating-point ALU produces a quiet NAN as the final result, and the invalid operation flag goes HIGH. Table 4 lists the cases for which the invalid flag is HIGH in IEEE mode, and the final results produced for these operations.

TABLE 4. IEEE MODE INVALID OPERATIONS

Operation	Input Operand	Final Result
R PLUS S	$(+\infty) + (-\infty)$ or $(-\infty) + (+\infty)$	7FA00000 ₁₆ (quiet NAN)
R PLUS S	$(+\infty) + (+\infty)$ or $(-\infty) + (-\infty)$ (Note 1)	7FA00000 ₁₆ (quiet NAN)
R MINUS S	$(+\infty) - (+\infty)$ or $(-\infty) - (-\infty)$	7FA00000 ₁₆ (quiet NAN)
R MINUS S	$(+\infty) - (-\infty)$ or $(-\infty) - (+\infty)$ (Note 1)	7FA00000 ₁₆ (quiet NAN)
R TIMES S	$(+0) * (+\infty)$ or $(+0) * (-\infty)$ or $(-0) * (+\infty)$ or $(-0) * (-\infty)$	7FA00000 ₁₆ (quiet NAN)
R PLUS S R MINUS S R TIMES S	R or S is a signalling NAN	(Note 2)
2 MINUS S	S is a signalling NAN	(Note 2)
FP-TO-INT	R is a signalling or quiet NAN	(Note 2)
FP-TO-INT	$R > 2^{31} - 1$ or $R < -(2^{31})$	7FA00000 ₁₆ (quiet NAN)

Notes: 1. These cases are invalid in projective mode only.

2. Results for these operations are described in the **Operations with NANs** section.

The Sign Bit

For most floating-point operations, the sign bit of the final result is unambiguous; i.e., there is only one sign bit value that yields a numerically correct result. Operations that produce an infinitely precise result of zero, however, present a problem, as the IEEE floating-point format allows for representation of both +0 and -0. The following rules can be used to determine the signs of zero produced in such cases.

R PLUS S: The operations $+x + (-x)$ and $-x + (+x)$ produce a final result of zero; the sign of the zero is dependent on the rounding mode:

Operations $+0 + (-0)$ and $-0 + (+0)$ produce a result of 0, with the sign of the result determined by the table above.

The operation $+0 + (+0)$ produces a final result of +0; the operation $-0 + (-0)$ produces a final result of -0.

R MINUS S: The operations $+x - (+x)$ and $-x - (-x)$ produce a final result of zero; the sign of the zero is dependent on the rounding mode:

Rounding Mode	Sign of Result
Round to nearest	0
Round toward $-\infty$	1
Round toward $+\infty$	0
Round toward 0	0

Operations $+0 - (+0)$ and $-0 - (-0)$ produce a result of 0, with the sign of the result determined by the table above.

The operation $+0 - (-0)$ produces a final result of +0; the operation $-0 - (+0)$ produces a final result of -0.

R TIMES S: The sign of any multiplication result other than a NAN is the exclusive OR of the signs of the input operands. Therefore, if x is non-negative, +0 times +x produces a final result of +0, +0 times -x produces a final result of -0, -0 times +x produces a final result of -0, -0 times -x produces a final result of +0.

2 MINUS S: If S equals 2, the final result is -0 for the round toward $-\infty$ mode, and +0 for all other rounding modes.

Rounding

Rounding is performed whenever an operation produces an infinitely precise result that cannot be represented exactly in the destination format. For example, suppose a floating-point operation produces the infinitely precise result:

$$1.101010101010101010101010101010101 \times 2^3.$$

In this example, the fraction portion of the mantissa has 25 bits; the IEEE floating-point format can accommodate only 23. The backslash (\) in the mantissa represents the boundary between the first 23 bits of the fraction and any remaining bits. Rounding is the process by which this result is approximated by a representation that fits the destination format.

There are four rounding modes in IEEE mode: 1) round to nearest, 2) round toward $+\infty$, 3) round toward $-\infty$, and 4) round toward 0. The rounding mode is chosen using the rounding mode select lines, RND₀ and RND₁. Table 5 lists the select states needed to obtain the desired rounding mode.

TABLE 5. ROUNDING MODE SELECT

Rounding Mode	Sign of Final Result	RND ₁	RND ₀	Rounding Mode
Round to nearest	0	0	0	Round to nearest
Round toward $-\infty$	1	0	1	Round toward $-\infty$
Round toward $+\infty$	0	1	0	Round toward $+\infty$
Round toward 0	0	1	1	Round toward 0

Round to Nearest: In this rounding mode the infinitely precise result of an operation is rounded to the closest representation that fits in the destination format. If the infinitely precise result is exactly halfway between two representations, it is rounded to the representation having an LSB of zero. Rounding is performed both for floating-point and integer destination formats.

Figure 9 illustrates four examples of the round-to-nearest process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 9(a), the infinitely precise result of an operation is:
 $2^{20} + 2^{-4} + 2^{-5} = 1.000000000000000000000000000000011 \times 2^{20}$

The result is rounded to the closest representable floating-point value,
 $2^{20} + 2^{-3} = 1.00000000000000000000000000000001 \times 2^{20}$

Example 2:

In Figure 9(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.111111111111111111111111111111110001 \times 2^{19}$$

This result is rounded to the closest representable floating-point value,

$$2^{20} - 2^{-4} = 1.11111111111111111111111111111111 \times 2^{19}$$

Example 3:

In Figure 9(c), the infinitely precise result of an operation is:

$$-(2^{20} + 2^{-3} + 2^{-4}) = -1.000000000000000000000000000000011 \times 2^{20}$$

This result is exactly halfway between two representable floating-point values. Accordingly, it is rounded to the closest representation with an LSB of zero, or

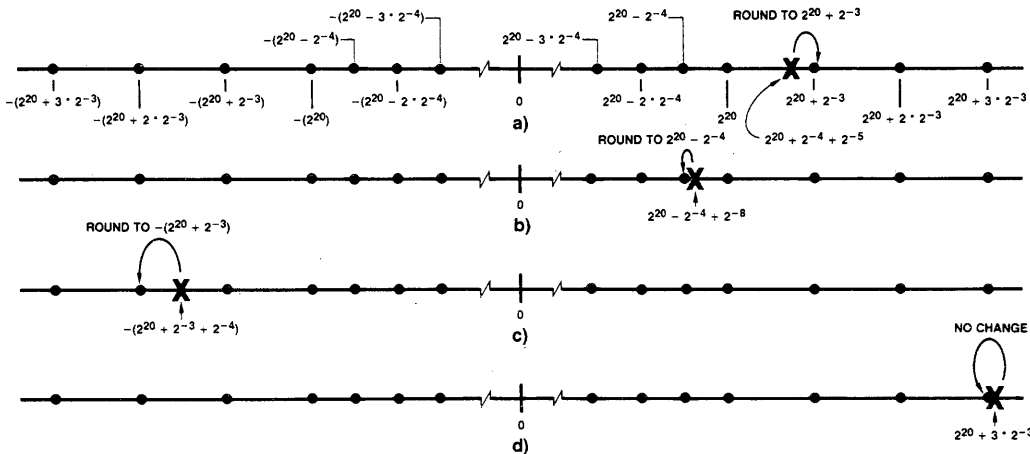
$$-(2^{20} + 2^{-3}) = -1.00000000000000000000000000000010 \times 2^{20}$$

Example 4:

In Figure 9(d), the infinitely precise result of an operation is:

$$2^{20} + 3 \cdot 2^{-3} = 1.000000000000000000000000000000011 \times 2^{20}$$

This result can be represented exactly in the floating-point format, and is left unaltered by the rounding process.



AF004550

Figure 9. Floating-Point Rounding Examples for Round-to-Nearest Mode

Figure 10 illustrates four examples of the round-to-nearest process for operations having an integer destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the integer format.

Example 1:

In Figure 10(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to the closest representable integer value,

$$2^{10} = 00...010000000000$$

Example 2:

In Figure 10(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...010000000001.001$$

This result is rounded to the closest representable integer value,

$$2^{10} + 2^0 = 00...010000000001$$

Example 3:

In Figure 10(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = -11...1011111111110.1$$

This result is exactly halfway between two representable integer values. Accordingly, it is rounded to the closest representation with an LSB of zero, or

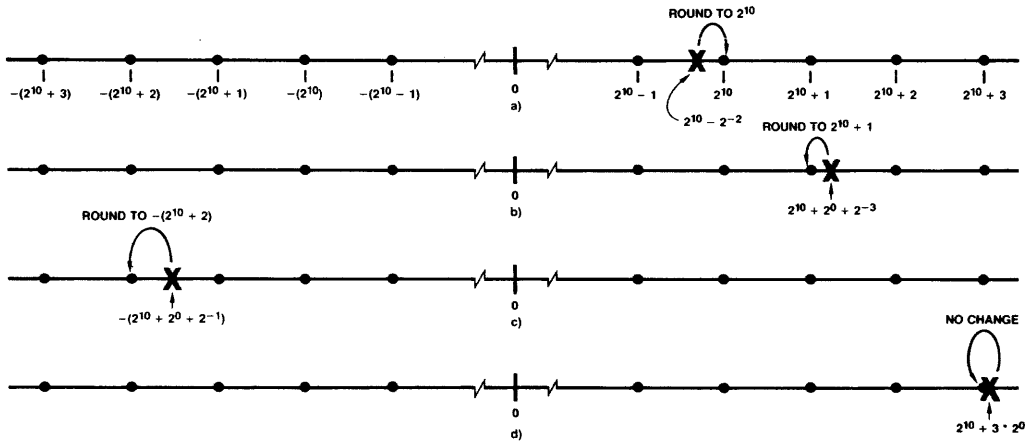
$$-(2^{10} + 2^*2^0) = 11...1011111111110$$

Example 4:

In Figure 10(d), the infinitely precise result of an operation is:

$$2^{10} + 3*2^0 = 00...0100000000011$$

This result can be represented exactly in the integer format, and is left unaltered by the rounding process.



AF004560

Figure 10. Integer Rounding Examples for Round-to-Nearest Mode

Round Toward $-\infty$: In this rounding mode the result of an operation is rounded to the closest representation that is less than or equal to the infinitely precise result, and which fits the destination format. Rounding is performed both for floating-point and integer destination formats.

Figure 11 illustrates four examples of the round toward $-\infty$ process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 11(a), the infinitely precise result of an operation is:
 $2^{20} + 2^{-4} + 2^{-5} = 1.00000000000000000000000000000011 \times 2^{20}$

This result cannot be represented exactly in floating-point format, and is rounded to the next-smaller floating-point representation:
 $2^{20} = 1.0000000000000000000000000000000 \times 2^{20}$

Example 2:

In Figure 11(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.1111111111111111111111111111111 \setminus 0001 \times 2^{19}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-smaller floating point representation:

$$2^{20} - 2^{-4} = 1.1111111111111111111111111111111 \times 2^{19}$$

Example 3:

In Figure 11(c), the infinitely precise result of an operation is:
 $-(2^{20} + 2^{-3} + 2^{-4}) = -1.0000000000000000000000000000001 \setminus 1 \times 2^{20}$

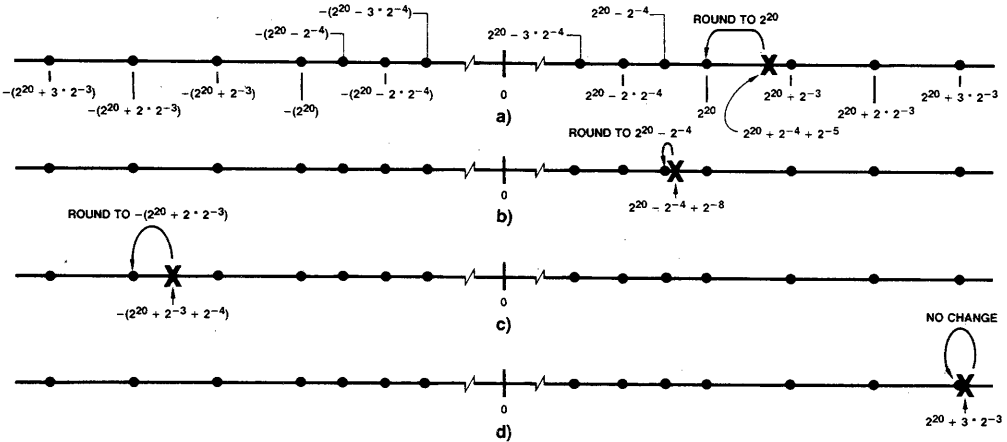
This result cannot be represented exactly in floating-point format, and is rounded to the next-smaller floating-point representation.

$$-(2^{20} + 2^{-3}) = -1.0000000000000000000000000000010 \times 2^{20}$$

Example 4:

In Figure 11(d), the infinitely precise result of an operation is:
 $2^{20} + 3 \cdot 2^{-3} = 1.0000000000000000000000000000011 \times 2^{20}$

This result can be represented exactly in the floating-point format, and is left unaltered by the rounding process.



AF004510

Figure 11. Floating-Point Rounding Examples for Round Toward $-\infty$ Mode

Figure 12 illustrates four examples of the round toward $-\infty$ process for operations having an integer destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be exactly represented in the integer format.

Example 1:

In Figure 12(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to the next-smaller representable integer value,

$$2^{10} - 2^0 = 00...001111111111$$

Example 2:

In Figure 12(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...010000000001.001$$

This result is rounded to the next-smaller representable integer value,

$$2^{10} + 2^0 = 00...010000000001$$

Example 3:

In Figure 12(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = 11...101111111110.1$$

This result is rounded to the next-smaller representable integer value:

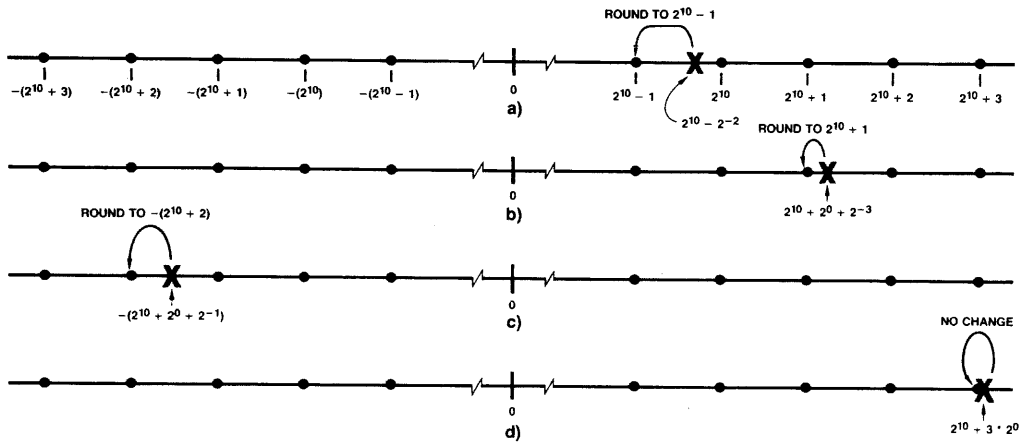
$$-(2^{10} + 2^*2^0) = 11...101111111110$$

Example 4:

In Figure 12(d), the infinitely precise result of an operation is:

$$2^{10} + 3*2^0 = 00...010000000011$$

This result can be represented exactly in the integer format, and is unaltered by the rounding process.



AF004580

Figure 12. Integer Rounding Examples for Round Toward $-\infty$ Mode

Round Toward $+\infty$: In this rounding mode the result of an operation is rounded to the closest representation that is greater than or equal to the infinitely precise result, and which fits the destination format. Rounding is performed both for floating-point and integer destination formats.

Figure 13 illustrates four examples of the round toward $+\infty$ process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 13(a), the infinitely precise result of an operation is:
 $2^{20} + 2^{-4} + 2^{-5} = 1.0000000000000000000000000011 \times 2^{20}$

This result cannot be represented exactly in floating-point format, and is rounded to the next-larger floating-point representation:

$$2^{20} + 2^{-3} = 1.00000000000000000000000001 \times 2^{20}$$

Example 2:

In Figure 13(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.111111111111111111111111110001 \times 2^{19}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-larger floating point representation:

$$2^{20} = 1.000000000000000000000000 \times 2^{20}$$

Example 3:

In Figure 13(c), the infinitely precise result of an operation is:

$$-(2^{20} + 2^{-3} + 2^{-4}) = -1.000000000000000000000000011 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to the next-larger floating-point representation.

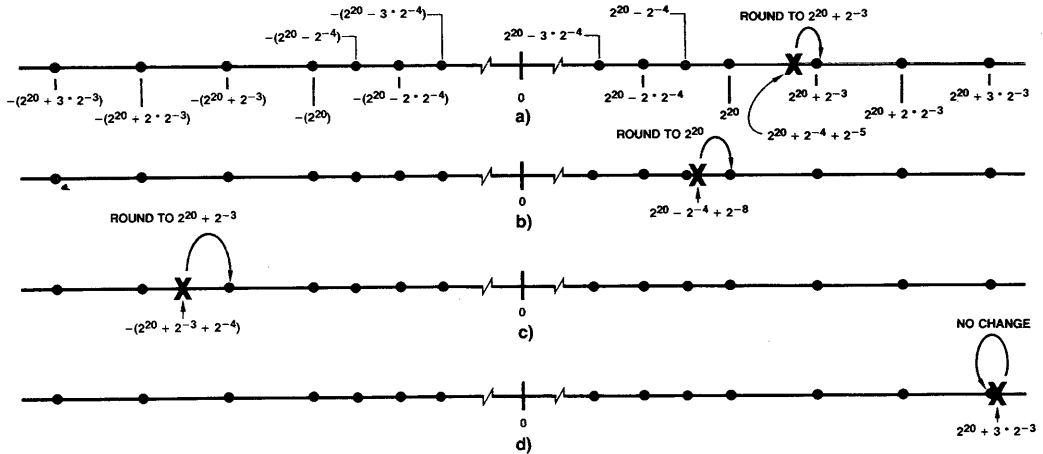
$$-(2^{20} + 2^{-3}) = -1.00000000000000000000000001 \times 2^{20}$$

Example 4:

In Figure 13(d), the infinitely precise result of an operation is:

$$2^{20} + 3 \cdot 2^{-3} = 1.00000000000000000000000011 \times 2^{20}$$

This result can be represented exactly in the floating-point format — no rounding takes place.



AF004590

Figure 13. Floating-Point Rounding Examples for Round Toward $+\infty$ Mode

Figure 14 illustrates four examples of the round toward $+\infty$ process for having an integer destination format. The infinitely precise result of an operation is represented by an 'X' on the number line; the black dots on the number line indicate those values that can be exactly represented in the integer format.

Example 1:

In Figure 14(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to the next-larger representable integer value,

$$2^{10} = 00...010000000000$$

Example 2:

In Figure 14(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...010000000001.001$$

This result is rounded to the next-larger representable integer value,

$$2^{10} + 2^0 = 00...010000000010$$

Example 3:

In Figure 14(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = 11.1011111111110.1$$

This result is rounded to the next-larger representable integer value:

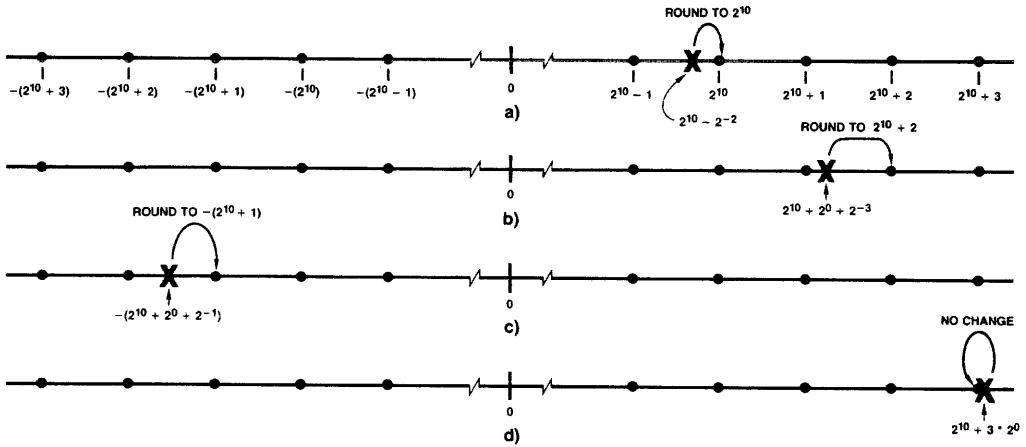
$$-(2^{10} + 2^0) = 11...1011111111110$$

Example 4:

In Figure 14(d), the infinitely precise result of an operation is:

$$2^{10} + 3 \cdot 2^0 = 00...010000000011$$

This result can be represented exactly in the integer format — no rounding takes place.



AF004600

Figure 14. Integer Rounding Examples for Round Toward $+\infty$ Mode

Round Toward 0: In this rounding mode the result of an operation is rounded to the closest representation whose magnitude is less than or equal to the infinitely precise result, and which fits the destination format. Rounding is performed both for floating-point and integer destination formats.

Figure 15 illustrates four examples of the round toward 0 process for operations having a floating-point destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be represented exactly in the floating-point format.

Example 1:

In Figure 15(a), the infinitely precise result of an operation is:

$$2^{20} + 2^{-4} + 2^{-5} = 1.000000000000000000000000000011 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to:

$$2^{20} = 1.000000000000000000000000000000 \times 2^{20}$$

Example 2:

In Figure 15(b), the infinitely precise result of an operation is:

$$2^{20} - 2^{-4} + 2^{-8} = 1.1111111111111111111111111111001 \times 2^{19}$$

This result cannot be represented exactly in floating-point format, and is rounded to:

$$2^{20} - 2^{-4} = 1.1111111111111111111111111111 \times 2^{19}$$

Example 3:

In Figure 15(c), the infinitely precise result of an operation is:

$$-(2^{20} + 2^{-3} + 2^{-4}) = -1.00000000000000000000000000001 \times 2^{20}$$

This result cannot be represented exactly in floating-point format, and is rounded to:

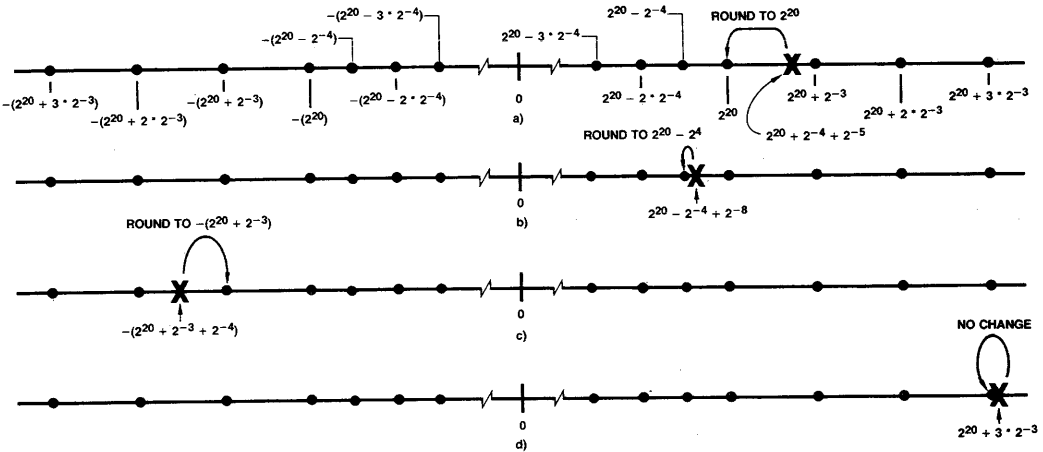
$$-(2^{20} + 2^{-3}) = -1.00000000000000000000000000001 \times 2^{20}$$

Example 4:

In Figure 15(d), the infinitely precise result of an operation is:

$$2^{20} + 3 \cdot 2^{-3} = 1.000000000000000000000000000011 \times 2^{20}$$

This result can be represented exactly in the floating-point format, and is unaffected by the rounding process.



AF004610

Figure 15. Floating-Point Rounding Examples for Round Toward 0 Mode

Figure 16 illustrates four examples of the round toward 0 process for operations having an integer destination format. The infinitely precise result of an operation is represented by an "X" on the number line; the black dots on the number line indicate those values that can be exactly represented in the integer format.

Example 1:

In Figure 16(a), the infinitely precise result of an operation is:

$$2^{10} - 2^{-2} = 00...001111111111.11$$

The result is rounded to:

$$2^{10} - 2^0 = 00...001111111111$$

Example 2:

In Figure 16(b), the infinitely precise result of an operation is:

$$2^{10} + 2^0 + 2^{-3} = 00...01000000001.001$$

The result is rounded to:

$$2^{10} + 2^0 = 00...01000000001$$

Example 3:

In Figure 16(c), the infinitely precise result of an operation is:

$$-(2^{10} + 2^0 + 2^{-1}) = 11...101111111111.1$$

The result is rounded to:

$$-(2^{10} + 2^0) = 11...101111111111$$

Example 4:

In Figure 16(d), the infinitely precise result of an operation is:

$$2^{10} + 3 \cdot 2^0 = 00...010000000011$$

This result can be represented exactly in the integer format, and is unaffected by the rounding process.

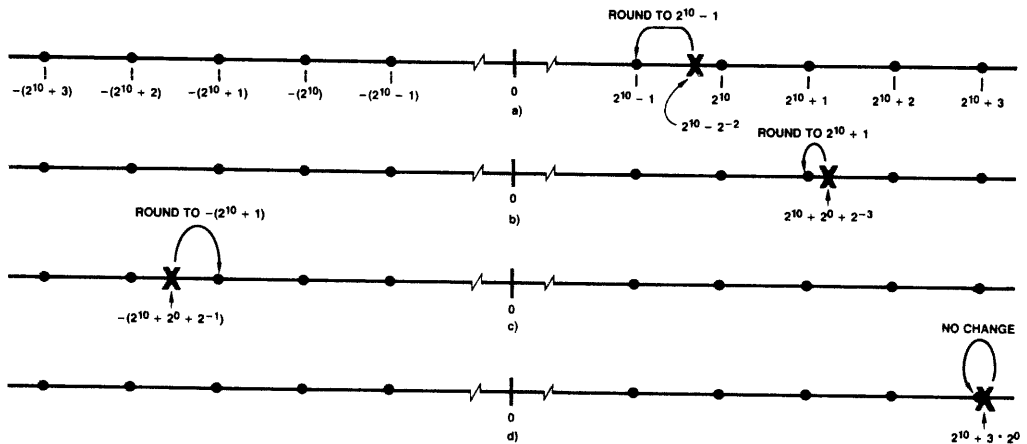


Figure 16. Integer Rounding Examples for Round Toward 0 Mode

Flag Operation

The Am29C325 generates six status flags to monitor floating-point processor operation. The following is a summary of flag conventions in IEEE mode:

Invalid Operation Flag: The invalid operation flag is HIGH when an input operand is invalid for the operation to be performed. Table 4 lists the cases for which the invalid operation flag is HIGH in IEEE mode, and the corresponding final result. In cases where the invalid operation flag is HIGH, the overflow, underflow, zero, and inexact flags are LOW; the NAN flag will be HIGH.

Overflow Flag: The overflow flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation with finite input operand(s) produces a result which, after rounding, has a magnitude greater than or equal to 2^{128} . The final result will be $+\infty$ or $-\infty$.

Underflow Flag: The underflow flag is HIGH if an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has a magnitude in the range: $0 < \text{magnitude} < 2^{-126}$.

The final result will be $+0$ (00000000_{16}) if the rounded result is non-negative, and -0 (80000000_{16}) if the rounded result is negative.

Inexact Flag: The inexact flag is HIGH if the final result of an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, or FP-TO-INT operation is not equal to the infinitely precise result. Note that if the underflow or overflow flag is HIGH, the inexact flag will also be HIGH.

Zero Flag: The zero flag is HIGH if the final result of an operation is zero. For operations producing an IEEE floating-point number, the flag accompanies outputs $+0$ (00000000_{16}) and -0 (80000000_{16}). For operations producing an integer, the flag accompanies the output 0 (00000000_{16}).

NAN Flag: The NAN flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, or FP-TO-INT operation produces a NAN as a final result.

Operation in DEC Mode

When input signal IEEE/DEC is LOW, the DEC mode of operation is selected. In this mode the Am29C325 uses the single-precision floating-point format (floating F) set forth in

Digital Equipment Corporation's VAX Architecture Manual. In addition, the DEC mode complies with most other aspects of single-precision floating-point operation outlined in the manual — differences are discussed in **Appendix B**.

DEC Floating-Point Format

The DEC single-precision floating-point word is 32 bits wide, and is arranged in the format shown in Figure 17. The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 23-bit fraction.

The sign bit indicates the sign of the floating-point number's value. Non-negative values have a sign of 0, negative values a sign of 1.

The biased exponent is an 8-bit unsigned integer field representing a multiplicative factor of some power of two. The bias value is 128. If, for example, the multiplicative factor for a floating-point number is to be 2^3 , the value of the biased exponent would be $a + 128$; "a" is called the true exponent.

The fraction is a 23-bit unsigned fractional field containing the 23 LSBs of the floating-point number's 24-bit mantissa. The weight of this field's MSB is 2^{-2} ; the weight of the LSB is 2^{-24} .

A floating-point number is evaluated or interpreted per the following conventions:

let s = sign bit
 e = biased exponent
 f = fraction

if $e = 0$ and $s = 0$...value = 0
 if $e = 0$ and $s = 1$...value = DEC-reserved operand
 if $0 < e \leq 255$...value = $(-1)^s \times (2^{e-128}) \times (.1f)$
 (normalized number)

Zero: The value zero always has a sign of zero.

DEC-Reserved Operand: A DEC-reserved operand does not represent a numeric value, but is interpreted as a signal or symbol. DEC-reserved operands are used to indicate invalid operations and operations whose results have overflowed the destination format. They may also be used to pass symbolic information from one calculation to another.

Normalized Number: A normalized number represents a quantity with magnitude greater than or equal to 2^{-128} but less than 2^{127} .

Example 1:

The number +3.5 can be represented in floating-point format as follows:

$$+3.5 = 11.1_2 \times 2^0$$

$$= .111_2 \times 2^2$$

sign = 0

$$\text{biased exponent} = 2_{10} + 128_{10} = 130_{10}$$

$$= 1000010_2$$

$$\text{fraction} = 110000000000000000000_2$$

(the leading 1 is implied in the format)

Concatenating these fields produces the floating-point word 41600000₁₆.

Example 2:

The number -11.375 can be represented in floating-point format as follows:

$$-11.375 = -1011.011_2 \times 2^0$$

$$= -.1011011_2 \times 2^4$$

sign = 1

$$\text{biased exponent} = 4_{10} + 128_{10} = 132_{10}$$

$$= 10000100_2$$

$$\text{fraction} = 011011000000000000000_2$$

(the leading 1 is implied in the format)

Concatenating these fields produces the floating-point word C2360000₁₆.

DEC Mode Integer Format

DEC mode integer format is identical to that of the IEEE mode. Integer numbers are represented as 32-bit, two's-complement words (Figure 8 depicts the integer format). The integer word can represent a range of integer values from -2^{31} to $2^{31} - 1$.

Operations

All eight floating-point ALU operations discussed in the **General Description** section can be performed in DEC mode.

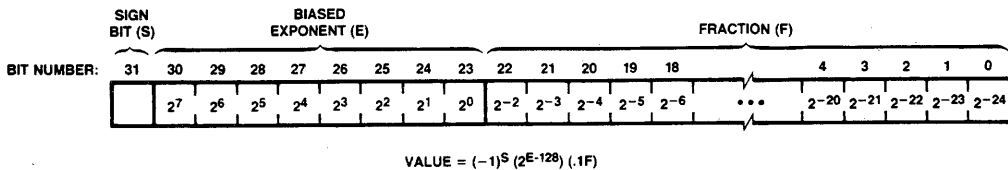


Figure 17. DEC-Mode Floating-Point Format

TB000671

Various exceptional aspects of the R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, and FP-TO-INT operations for this mode are described below. The IEEE-TO-DEC and DEC-TO-IEEE operations are discussed separately in the **IEEE-TO-DEC and DEC-TO-IEEE Operations** section.

Operations with DEC-Reserved Operands: DEC-reserved operands arise in two ways: 1) they can be generated by the Am29325 to indicate that an invalid operation or floating-point

overflow has taken place, or 2) be provided by the user as an input operand.

When a DEC-reserved operand appears as an input operand, the final result of the operation is the same DEC-reserved operand. If an operation has two DEC-reserved operands as inputs, the DEC-reserved operand on the R port becomes the final result.

The NAN flag will be HIGH whenever an operation produces a DEC-reserved operand as a final result.

Example 1:

Suppose the floating-point addition operation is performed with the following input operands:

R port: 40800000_{16} ($0.1 \cdot 2^1$)
S port: 80012345_{16} (DEC-reserved operand)

Result: This operation produces the DEC-reserved operand on the S port, 80012345_{16} , as the final result. The NAN flag will be HIGH.

Example 2:

Suppose the floating-point multiplication operation is performed with the following input operands:

R port: 80765432_{16} (DEC-reserved operand)
S port: 80000001_{16} (DEC-reserved operand)

Result: Since both input operands are DEC-reserved operands, the operand on the R port, 80765432_{16} , is the final result of the operation. The NAN flag will be HIGH.

Operations Producing Overflows: If an operation produces a rounded result that is too large to fit in the destination format, that operation is said to have overflowed.

A floating-point overflow occurs if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation with finite input operand(s) produces a result which, after rounding, has a magnitude greater than or equal to 2^{127} . The final result in such cases will be DEC-reserved operand 80000000_{16} ; the overflow, inexact, and NAN flags will be HIGH.

Integer overflow occurs when the "floating-point-to-integer" conversion operation attempts to convert to integer a floating-point number which, after rounding, is greater than $2^{31} - 1$ or less than -2^{31} . The final result in such cases will be DEC-reserved operand 80000000_{16} ; the invalid operation flag will be HIGH. Note that the overflow and inexact flags remain LOW for integer overflow.

Operations Producing Underflows: If an operation produces a floating-point result which, after rounding, has a magnitude too small to be expressed as a normalized floating-point number, but greater than 0, that operation is said to have underflowed. Underflow occurs when an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has the magnitude:

$$0 < \text{magnitude} < 2^{-128}$$

The final result in such cases will be 0 (00000000_{16}). The underflow, inexact, and zero flags will be HIGH.

Underflow does not occur if the destination format is integer. If the infinitely precise result of a floating-point-to-integer conversion has a magnitude greater than 0 and less than 1, but the rounded result is 0, the underflow flag remains LOW.

Invalid Operations: If an input operand is invalid for the operation to be performed, that operation is considered invalid. There is only one invalid operation in DEC mode: performing a floating-point-to-integer conversion on a value too large to be converted to an integer. In this case, the final result will be DEC-reserved operand 80000000_{16} , and the invalid operation and NAN flags will be HIGH.

Sign Bit

For all operations producing a DEC floating-point result, the sign bit of the final result is unambiguous; i.e., there is only one sign bit value that yields a numerically correct result.

Rounding

There are four rounding modes for DEC operation: 1) round to nearest, 2) round toward $+\infty$, 3) round toward $-\infty$, and 4) round toward 0. The round toward $+\infty$, round toward $-\infty$, and round toward 0 modes are performed in a manner identical to that for IEEE operation; refer to the **Rounding** section under **Operation in IEEE Mode**. The round to nearest mode is similar to that for IEEE operation, but differs in one respect: for the case in which the infinitely precise result of an operation is exactly halfway between two representable values, DEC round to nearest mode rounds to the value with the larger magnitude, rather than to the value whose LSB is 0.

Flag Operation

The Am29C325 generates six status flags to monitor floating-point processor operation. The following is a summary of flag operation in DEC mode:

Invalid Operation Flag: The invalid operation flag is HIGH if the FP-TO-INT operation is performed on a floating-point number too large to be converted to an integer. The final result for such an operation will be the DEC-reserved operand 80000000_{16} .

Overflow Flag: The overflow flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, or 2 MINUS S operation produces a result which, after rounding, has a magnitude greater than or equal to 2^{127} . The final result will be the DEC-reserved operand 80000000_{16} .

Underflow Flag: The underflow flag is HIGH if an R PLUS S, R MINUS S, or R TIMES S operation produces a result which, after rounding, has a magnitude in the range:

$$0 < \text{magnitude} < 2^{-128}$$

The final result will be 0 (00000000_{16}) in such cases.

Inexact Flag: The inexact flag is HIGH if the final result of an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, INT-TO-FP, or FP-TO-INT operation is not equal to the infinitely precise result. Note that if the underflow or overflow flag is HIGH, the inexact flag will also be HIGH.

Zero Flag: The zero flag is HIGH if the final result of an operation is 0. For operations producing an integer or a DEC floating-point number, the flag accompanies the output 0 (00000000_{16}). (It should be noted that any operation producing a floating-point 0 in DEC mode will output 00000000_{16} .)

NAN Flag: The NAN flag is HIGH if an R PLUS S, R MINUS S, R TIMES S, 2 MINUS S, or FP-TO-INT operation produces a DEC-reserved operand as the final result.

IEEE-TO-DEC and DEC-TO-IEEE Operations

The IEEE-TO-DEC and DEC-TO-IEEE operations are used to convert floating-point numbers between the IEEE and DEC formats. Both operations work in a manner independent of the IEEE/DEC mode control.

IEEE-TO-DEC Conversion

The operation converts an IEEE floating-point number to DEC floating-point format. Most conversions are exact; in no case does the round mode have any effect on the final result. There are, however, a few exceptional cases:

- a) If the IEEE floating-point input has a magnitude greater than or equal to 2^{127} , it is too large to be represented by a DEC floating-point number. The final result will be the DEC-reserved operand 80000000_{16} ; the overflow, inexact, and NAN flags will be HIGH.

- b) If the IEEE floating-point input is a NAN, the final result will be the DEC-reserved operand 80000000_{16} ; the invalid and NAN flags will be HIGH.
- c) If the IEEE floating-point input is a denormalized number, the final result will be a DEC 0 (0000000_{16}); the zero flag will be HIGH.
- d) If the IEEE floating-point input is +0 or -0, the final result will be a DEC 0 (0000000_{16}); the zero flag will be HIGH.

DEC-TO-IEEE Conversion

This operation converts a DEC floating-point number to IEEE floating-point format. Most conversions are exact; in no case does the round mode have any effect on the final result. There are, however, a few exceptional cases:

- a) If the DEC floating-point input is not 0, but has a magnitude less than 2^{-126} , it is too small to be expressed as a normalized IEEE floating-point number. The final result will be an IEEE floating-point 0 having the same sign as the input (0000000_{16} for positive inputs and 80000000_{16} for negative inputs); the underflow, inexact, and zero flags will be HIGH.
- b) If the DEC floating-point input is a DEC-reserved operand, the result will be quiet NAN $7FA0000_{16}$; the invalid operation and NAN flags will be HIGH.
- c) If the DEC floating-point input is 0, the final result will be IEEE floating-point +0 (0000000_{16}); the zero flag will be HIGH.

APPENDICES

APPENDIX A

DIFFERENCES BETWEEN THE IEEE PROPOSED STANDARD FOR BINARY FLOATING-POINT ARITHMETIC AND THE Am29C325'S IEEE MODE

When operated in IEEE mode, the Am29C325 High-Speed Floating-Point Processor complies with the single-precision portion of the IEEE Proposed Standard for Binary Floating-Point Arithmetic (P754, draft 10.0) in most respects. There are, however, several differences:

Denormalized Numbers

The Am29C325 does not handle denormalized numbers. A denormalized input will be converted to zero of the same sign before the specified operation takes place. The operation proceeds in exactly the same manner as if the input were +0 or -0, producing the same numerical result and flags.

If the result of an operation, after rounding, has a magnitude smaller than 2^{-126} , the result is replaced by a zero of the same sign.

Representation of Overflows

In some rounding modes the proposed IEEE standard requires that overflows be represented as the format's most-positive or most-negative finite number. In particular:

- When rounding toward 0, all overflows should produce a result of the largest representable finite number with the sign of the intermediate result.
- When rounding toward $-\infty$, all positive overflows should produce a result of the largest representable positive finite number.
- When rounding toward $+\infty$, all negative overflows should produce a result of the largest representable negative finite number.

The Am29C325, however, always represents positive overflows as $+\infty$ and negative overflows as $-\infty$, regardless of rounding mode.

Projective Mode

The proposed IEEE standard provides only for an affine mode to control the handling of infinities. The Am29C325 provides

both affine and projective modes; the desired mode can be selected by the user.

Traps

The proposed IEEE standard stipulates that the user be able to request a trap on any exception. The Am29C325 does not support trapped operation, and behaves as if traps are disabled.

Resetting of Flags

The proposed IEEE standard states that once an exception flag has been set, it is reset only at the user's request. The Am29C325's flags, however, reflect the status of the most recent operation.

Generation of the Underflow Flag

The proposed IEEE standard suggests several possible criteria for determining if underflow occurs. These criteria generate underflow flags that differ in subtle ways. The underflow criteria chosen for the Am29C325 stipulate that underflow occurs if:

- a) the rounded result of an operation has a magnitude in the range:

$$0 < \text{magnitude} < 2^{-126},$$

and

- b) the final result is not equal to the infinitely precise result.

Since the Am29C325 never produces a denormalized number as the final result of a calculation, condition (b) is true whenever (a) is true. Note then that the operation of the Am29C325's underflow flag is somewhat different than that of an "IEEE standard" system using the same underflow criteria. For example, if an operation should produce an infinitely precise result that is exactly 2^{-127} , an "IEEE standard" system would produce that value as the final result, expressed as a denormalized number. Since that system's final result is exact, the underflow flag would remain LOW. The Am29C325, on the other hand, would output zero; since its final result is not exact, the underflow flag would be HIGH.

APPENDIX B

DIFFERENCES BETWEEN DEC VAX AND Am29C325 DEC MODE

Operation in DEC mode complies with most aspects of single-precision floating-point operation outlined in the Digital Equipment Corporation's VAX Architecture Manual. However, there are some differences that should be noted:

Format

The Am29C325's DEC format is:

sign	- bit 31
exponent	- bits 30 - 23
mantissa	- 22 - 0

The VAX format is:

sign	- bit 15
exponent	- 14 - 7
mantissa	- bits 6 - 0, bits 31 - 16

In both cases, fields are listed from MSB to LSB, with bit 31 the MSB of the 32-bit word. The Am29C325's DEC format can be converted to VAX format by swapping the 16 LSBs and 16 MSBs of the 32-bit word.

Flags vs. Exceptions

In DEC VAX operation, certain unusual conditions arising during system operation may incur an exception, or an indication to the operating system that special handling is needed.

The VAX recognizes a number of arithmetic exceptions. The following exceptions are relevant to the operations supported by the Am29C325:

Integer Overflow Trap: indicates that the last operation produced an integer overflow. The LSBs of the correct result are stored in the destination operand.

Floating-Point Overflow Trap/Fault: indicates that the last operation produced, after normalization and rounding, a floating-point number with magnitude greater than or equal to 2^{127} . A trap replaces the destination operand with the DEC-reserved operand 80000000_{16} ; a fault leaves the destination operand unchanged.

Floating-Point Underflow Trap/Fault: indicates that the last operation produced, after normalization and rounding, a floating-point number with magnitude less than 2^{-128} . A trap

replaces the destination operand with zero; a fault leaves the destination operand unchanged.

Reserved Operand Fault: indicates that the last operation had a reserved operand as an input. The destination operand is unchanged.

The Am29C325 does not directly support DEC traps and faults. Rather, it indicates unusual conditions by setting one or more of the six status flags HIGH. Table D2 describes flag operation in DEC mode.

Integer Overflow

In cases of integer overflow, the VAX signals the integer overflow trap and stores the LSBs of the correct result. The Am29C325 sets the invalid operation flag and outputs the DEC-reserved operand 80000000_{16} .

Floating-Point Underflow/Overflow Operation

The VAX Architecture Manual specifies the action to be taken on the destination operand when floating-point underflow or overflow is encountered. The Am29C325 has no immediate control over this destination operand, as it resides somewhere off-chip, either in a register or memory location. This isn't so much a difference between the VAX specification and Am29C325 operation as it is a difference in scope.

The Am29C325 responds to floating-point underflow by producing a final result of 0 (00000000_{16}); the underflow, inexact, and zero flags will be HIGH. It responds to floating-point overflow by producing the DEC-reserved operand 80000000_{16} as the final result; the overflow, inexact, and NAN flags will be HIGH.

Handling of DEC-Reserved Operands

If an operation has a DEC-reserved operand as an input, the Am29C325 will produce that operand as the final result. If an operation has two input arguments and both are DEC-reserved operands, the operand on port R becomes the final result. For the VAX, operations with a DEC-reserved operand input or inputs do not modify the destination operand. As mentioned above, control of the destination operand is beyond the scope of the Am29C325's operation.

Inexact Flag

The Am29C325 provides an inexact flag to indicate that the final result produced by an operation is not equal to the infinitely precise result. The VAX does not provide this flag.

APPENDIX C

PERFORMING FLOATING-POINT DIVISION ON THE Am29C325

While the Am29C325 does not have a floating-point division instruction, it can be used to evaluate reciprocals. The division:

$$C = A/B$$

can then be performed by evaluating:

$$C = A \cdot (1/B)$$

Only a modest amount of external hardware is needed to implement the reciprocal function.

The technique for calculating reciprocals is based on the Newton-Raphson method for obtaining the roots of an equation. The roots of equation:

$$F(x) = 0$$

can be found by iteratively evaluating the equation:

$$x_{i+1} = x_i - F(x_i)/F'(x_i)$$

The process begins by making a guess as to the value of x_i , and using this guess or "seed" value to perform the first iteration. Iterations are continued until the root is evaluated to the desired accuracy. The number of iterations needed to achieve a given accuracy depends both on the accuracy of the seed value and the nature of $F(x)$.

Now consider the equation:

$$F(x) = (1/x) - B$$

The root of $F(x)$ is $1/B$. The reciprocal of B , then, can be found by using the Newton-Raphson method to find the root of $F(x)$. The iterative equation for finding the root is:

$$\begin{aligned} x_{i+1} &= x_i - F(x_i)/F'(x_i) \\ &= x_i - (1/x_i - B) / - (x_i)^{-2} \\ &= x_i (2 - B \cdot x_i) \end{aligned}$$

It can be shown that, in order for this iterative equation to converge, the seed value x_0 must fall in the range:

$$\begin{aligned} 0 < x_0 < 2/B & \quad \text{if } B > 0 \\ \text{or } 2/B < x_0 < 0 & \quad \text{if } B < 0 \end{aligned}$$

For example, if the reciprocal of 3 is to be evaluated, the seed value must be between 0 and $2/3$.

The error of x_i reduces quadratically; that is, if the error of x_i is e , the error is reduced to order e^2 by the next iteration. The number of bits of accuracy in the result, then, roughly doubles after every iteration. While this is only an approximation of the actual error produced, it is a handy rule of thumb for determining the number of iterations needed to produce a result of a certain accuracy, given the accuracy of the seed.

Example 1:

Find the reciprocal of 7.25.

Solution:

The seed value must fall in the range:

$$\begin{aligned} 0 < x_0 < 2/7.25 \\ \text{or } 0 < x_0 < .275862 \end{aligned}$$

Suppose x_0 is chosen to be .1:

$$\begin{aligned} \text{Iteration 1: } x_1 &= x_0 (2 - B \cdot x_0) \\ &= .1(2 - (7.25) (.1)) \\ &= .1275 \end{aligned}$$

$$\begin{aligned} \text{Iteration 2: } x_2 &= x_1 (2 - B \cdot x_1) \\ &= .1275(2 - (7.25) (.1275)) \\ &= .1371421875 \end{aligned}$$

$$\begin{aligned} \text{Iteration 3: } x_3 &= x_2 (2 - B \cdot x_2) \\ &= .1371421875^* \\ &\quad (2 - (7.25) (.1371421875)) \\ &= .1379265230 \end{aligned}$$

The actual value of $1/7.25$, to ten decimal places, is .1379310345.

The error after each iteration is:

Iteration	x_i	Error to Ten Places
0	0.1	-0.0379310345
1	0.1275	-0.0104310345
2	0.1371421875	-0.0007888470
3	0.1379265230	-0.000045115

Example 2:

Find the reciprocal of -0.3 .

Solution:

The seed value must fall in the range:

$$\begin{aligned} 2/(-0.3) < x_0 < 0 \\ \text{or } -6.66 < x_0 < 0 \end{aligned}$$

Suppose x_0 is chosen to be -2.0 :

$$\begin{aligned} \text{Iteration 1: } x_1 &= x_0 (2 - B \cdot x_0) \\ &= -2.0(2 - (-0.3) (-2.0)) \\ &= -2.8 \end{aligned}$$

$$\begin{aligned} \text{Iteration 2: } x_2 &= x_1 (2 - B \cdot x_1) \\ &= -2.8(2 - (-0.3) (-2.8)) \\ &= -3.248 \end{aligned}$$

$$\begin{aligned} \text{Iteration 3: } x_3 &= x_2 (2 - B \cdot x_2) \\ &= -3.248(2 - (-0.3) (-3.248)) \\ &= -3.3311488 \end{aligned}$$

$$\begin{aligned} \text{Iteration 4: } x_4 &= x_3 (2 - B \cdot x_3) \\ &= -3.3311488^* \\ &\quad (2 - (-0.3) (-3.3311488)) \\ &= -3.333331902 \end{aligned}$$

The actual value of $1/(-0.3)$, to ten decimal places, is -3.333333333 .

The error after each iteration is:

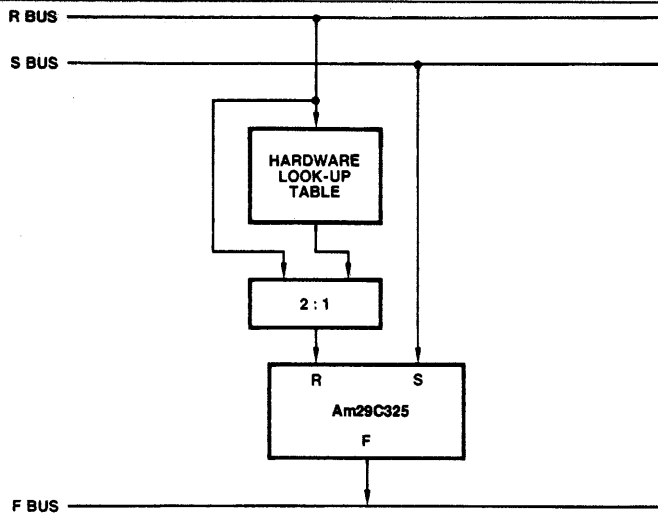
i	x_i	Error to Ten Places
0	-2.0	1.333333333
1	-2.8	0.533333333
2	-3.248	0.085333333
3	-3.3311488	0.002184533
4	-3.333331902	0.000001431

In order to implement the Newton-Raphson method on the Am29C325, some means is needed to generate the seed used in the first iteration. One approach is to place a hardware seed look-up table between the R bus and the Am29C325; see Table C1. A more detailed diagram of the look-up table appears in Figure C2.

TABLE C1. CONTENTS OF THE SEED EXPONENT PROM

DEC		IEEE	
Address (16)	Data (16)	Address (16)	Data (16)
000	(Note 1)	100	(Note 1)
001	(Note 1)	101	FC
002	FF	102	FB
003	FE	103	FA
004	FD	104	F9
005	FC	105	F8
006	FB	106	F7
007	FA	107	F6
008	F9	108	F5
009	F8	109	F4
00A	F7	10A	F3
00B	F6	10B	F2
00C	F5	10C	F1
00D	F4	10D	F0
00E	F3	10E	EF
00F	F2	10F	EE
010	F1	110	ED
011	F0	111	EC
012	EF	112	EB
.	.	.	.
.	.	.	.
0EE	13	1EE	0F
0EF	12	1EF	0E
0F0	11	1F0	0D
0F1	10	1F1	0C
0F2	0F	1F2	0B
0F3	0E	1F3	0A
0F4	0D	1F4	09
0F5	0C	1F5	08
0F6	0B	1F6	07
0F7	0A	1F7	06
0F8	09	1F8	05
0F9	08	1F9	04
0FA	07	1FA	03
0FB	06	1FB	02
0FC	05	1FC	01
0FD	04	1FD	(Note 2)
0FE	03	1FE	(Note 2)
0FF	02	1FF	(Note 2)

- Notes: 1. The reciprocals of these numbers are too large to be represented in the selected format.
 2. The reciprocals of these numbers are too small to be represented in normalized IEEE format.



AF004641

Figure C1. Adding a Hardware Look-Up Table to the Am29C325

The look-up table has two sections: a biased exponent look-up PROM, and a fraction look-up PROM. The seed-biased exponent look-up table is stored in a 512-by-8-bit PROM. This table consists of two sections: the DEC format section (which occupies addresses 000–0FF₁₆), and the IEEE section (which occupies addresses 100–1FF₁₆). The appropriate table will be selected automatically if address line A₈ is wired to the Am29C325's IEEE/DEC pin. The equations implemented by these table sections are:

DEC table: seed biased exponent
 $= 257_{10} - \text{input biased exponent}$

IEEE table: seed biased exponent
 $= 253_{10} - \text{input biased exponent}$

Table C1 lists the contents of this PROM.

The seed fraction look-up table is stored in one or more PROMs, the number of PROMs depending on the desired accuracy of the seed value. The hardware depicted in Figure

C2 uses two 4K-by-8-bit PROMs to implement a fraction look-up table whose inputs are the 12 MSBs of the input argument's fraction. These PROMs output the 16 MSBs of the seed's fraction field — the remaining 7 bits of fraction are set to 0. The equation implemented in this table is:

$$\text{seed fraction} = \frac{2}{1 + \text{input fraction}} - 1$$

where the value of the input fraction falls in the range

$$0 \leq \text{input fraction} < 1$$

Note that the seed fraction must also be constrained to fall in the range

$$0 \leq \text{seed fraction} < 1$$

Therefore, if the input fraction is 0, the corresponding seed fraction stored in the table must be .111...111₂, not 1.0₂. The same seed fraction look-up table may be used for both IEEE and DEC formats. Table C2 contains a partial listing for the seed fraction look-up table shown in Figure C2.

TABLE C2. CONTENTS OF THE SEED FRACTION PROMS

Address (16)	Value of Input Fraction (10)	Value of Seed Fraction (10)	PROM Outputs (16)	
			R22 - R15	R14 - R7
000	0.0	0.9999999999 (see text)	FF	FF
001	0.0002441406	0.9995118370	FF	E0
002	0.0004882812	0.9990239150	FF	C0
003	0.0007324219	0.9985362280	FF	A0
004	0.0009765625	0.9980487790	FF	80
005	0.0012207031	0.9975615710	FF	60
006	0.0014648438	0.9970745970	FF	40
007	0.0017089844	0.9965878630	FF	20
008	0.0019531250	0.9961013650	FF	00
009	0.0021972656	0.9956151030	FE	E1
00A	0.0024414063	0.9951290800	FE	C0
00B	0.0026855469	0.9946432920	FE	A1
00C	0.0029296875	0.9941577400	FE	81
.
.
FF6	0.9975585938	0.0012221950	00	50
FF7	0.9978027344	0.0010998410	00	48
FF8	0.9980486750	0.0009775170	00	40
FF9	0.9982910156	0.0008552230	00	38
FFA	0.9985351563	0.0007329590	00	30
FFB	0.9987792969	0.0006107240	00	28
FFC	0.9990234375	0.0004885200	00	20
FFD	0.9992675781	0.0003663450	00	18
FFE	0.9995117188	0.0002442000	00	10
FFF	0.9997558594	0.0001220850	00	08

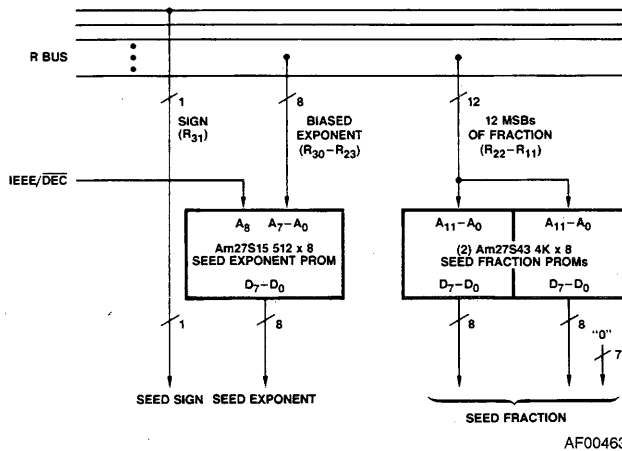


Figure C2. The Hardware Look-Up Table

With the hardware look-up table in place, the reciprocal of value B can be calculated with the following series of operations:

- 1) Place B on both the R and S buses. The 2 : 1 multiplexer at the output of the hardware look-up table should select the output of the look-up table (see Figure C3-A).
- 2) Load the seed value x_0 into register R and load B into register S. Select the R TIMES S operation (see Figure C3-B).
- 3) Load product $B \cdot x_0$ into register F. Select the 2 MINUS S operation, and select register F as the input to the ALU S port (see Figure C3-C).
- 4) Load $2 - B \cdot x_0$ into register F. Select the R TIMES S operation and select register F as the input to the ALU S port (see Figure C3-D).
- 5) Load the value x_1 ($x_1 = x_0(2 - B \cdot x_0)$) into registers R and F. Select the R TIMES S operation (see Figure C3-E).
- 6) Repeat steps 3 through 5 until the result has the accuracy desired.

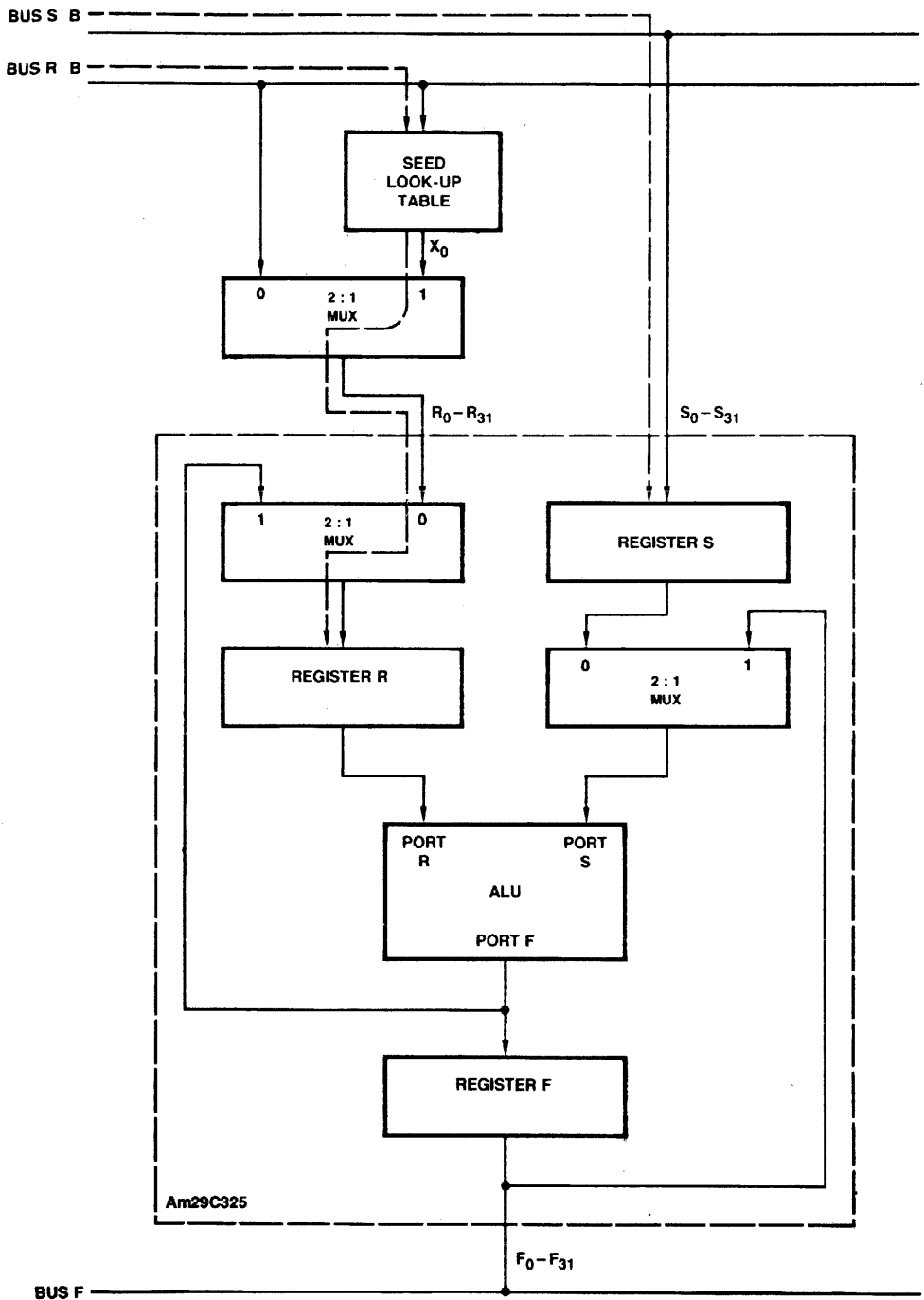


Figure C3-A. Data Flow for Step 1 of the Reciprocal Procedure

DF006211

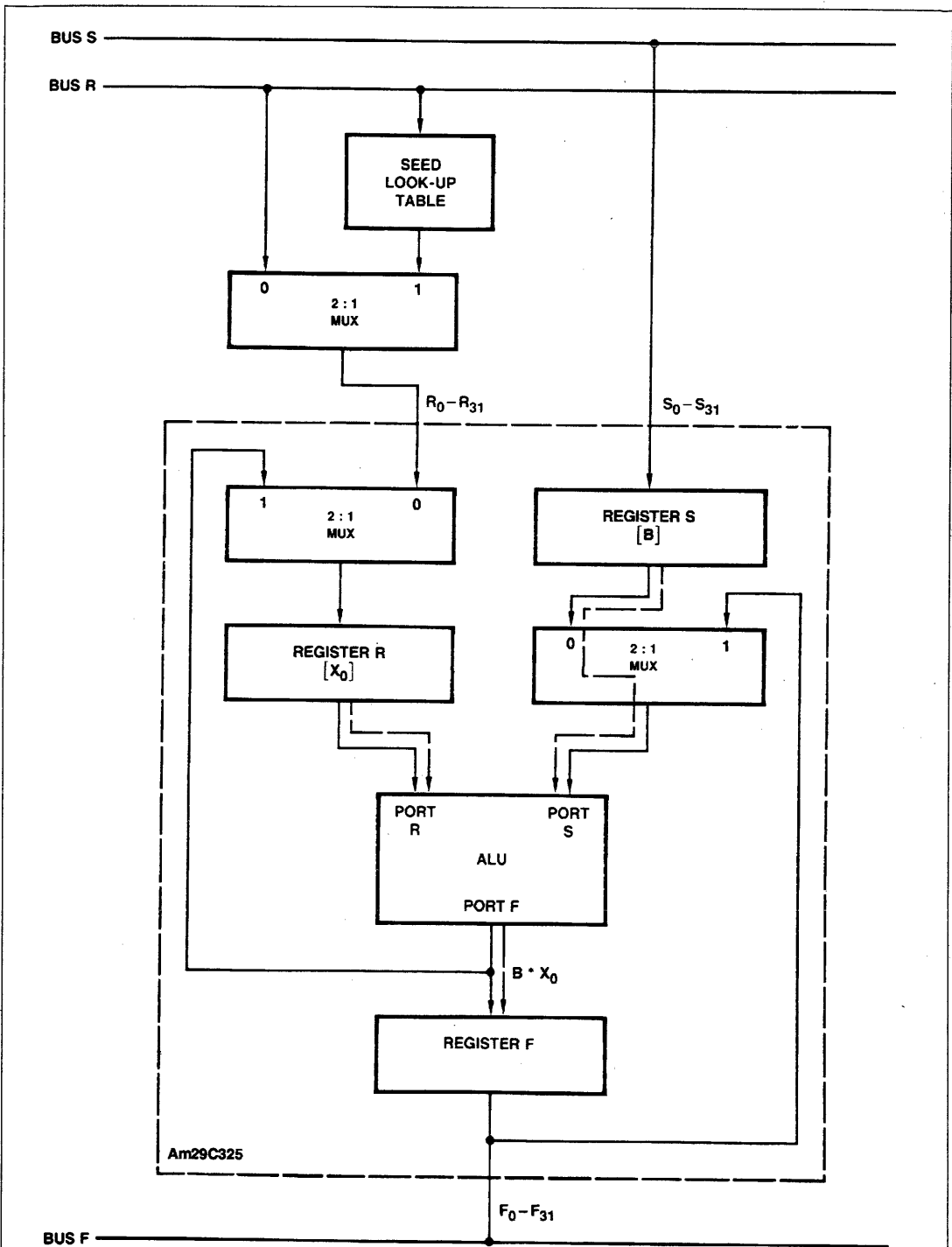


Figure C3-B. Data Flow for Step 2 of the Reciprocal Procedure

DF006221

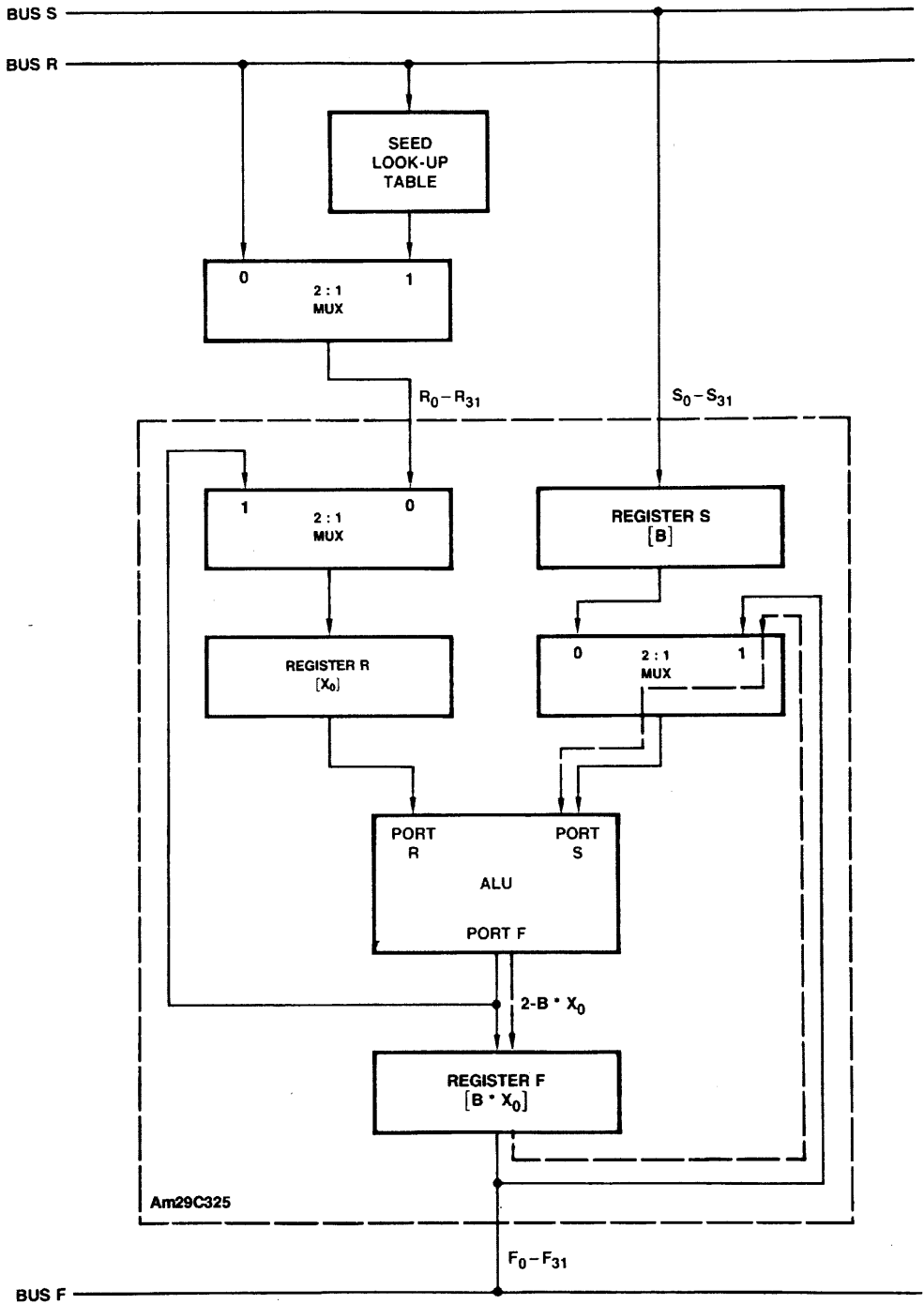
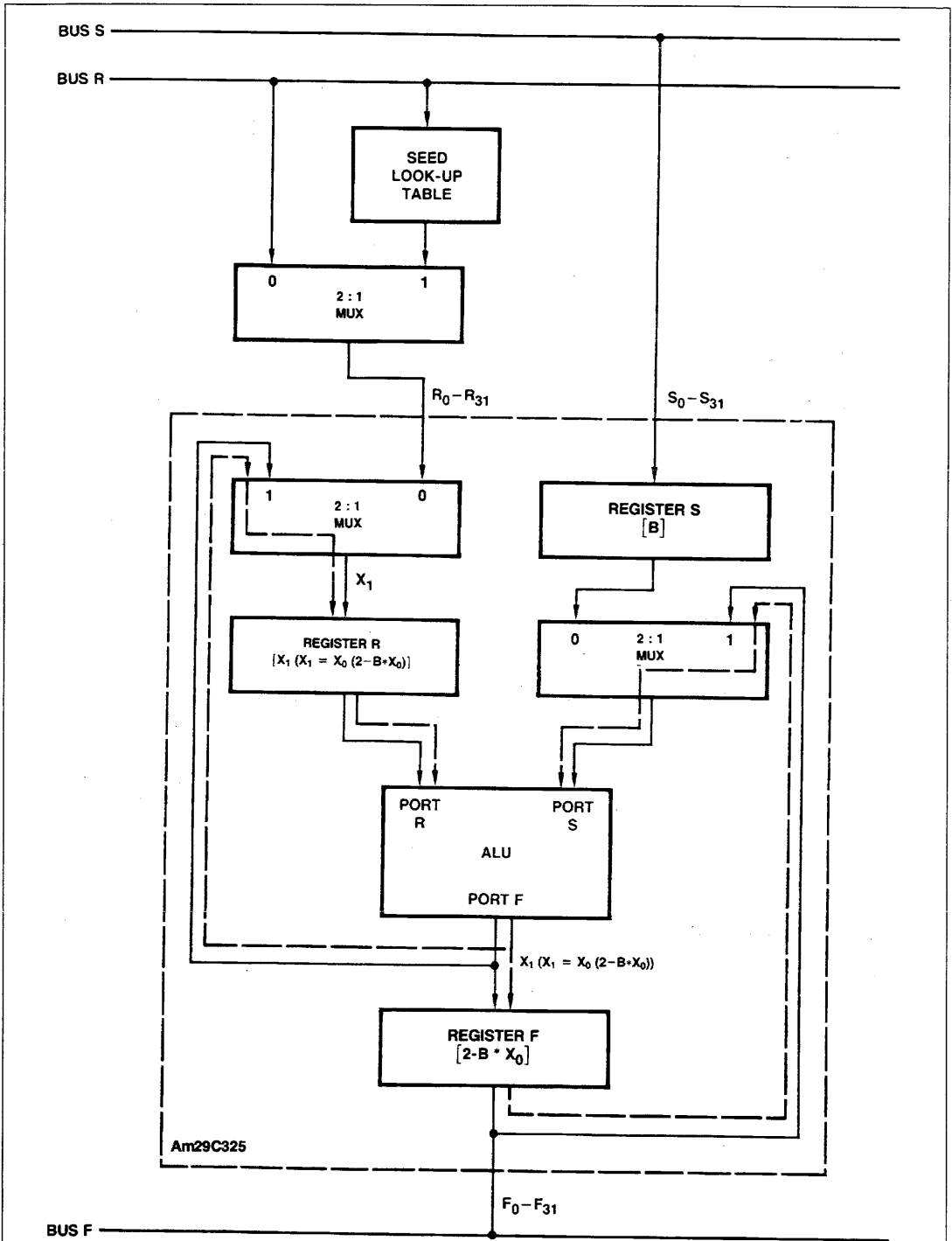


Figure C3-C. Data Flow for Step 3 of the Reciprocal Procedure

DF006231



DF006241

Figure C3-D. Data Flow for Step 4 of the Reciprocal Procedure

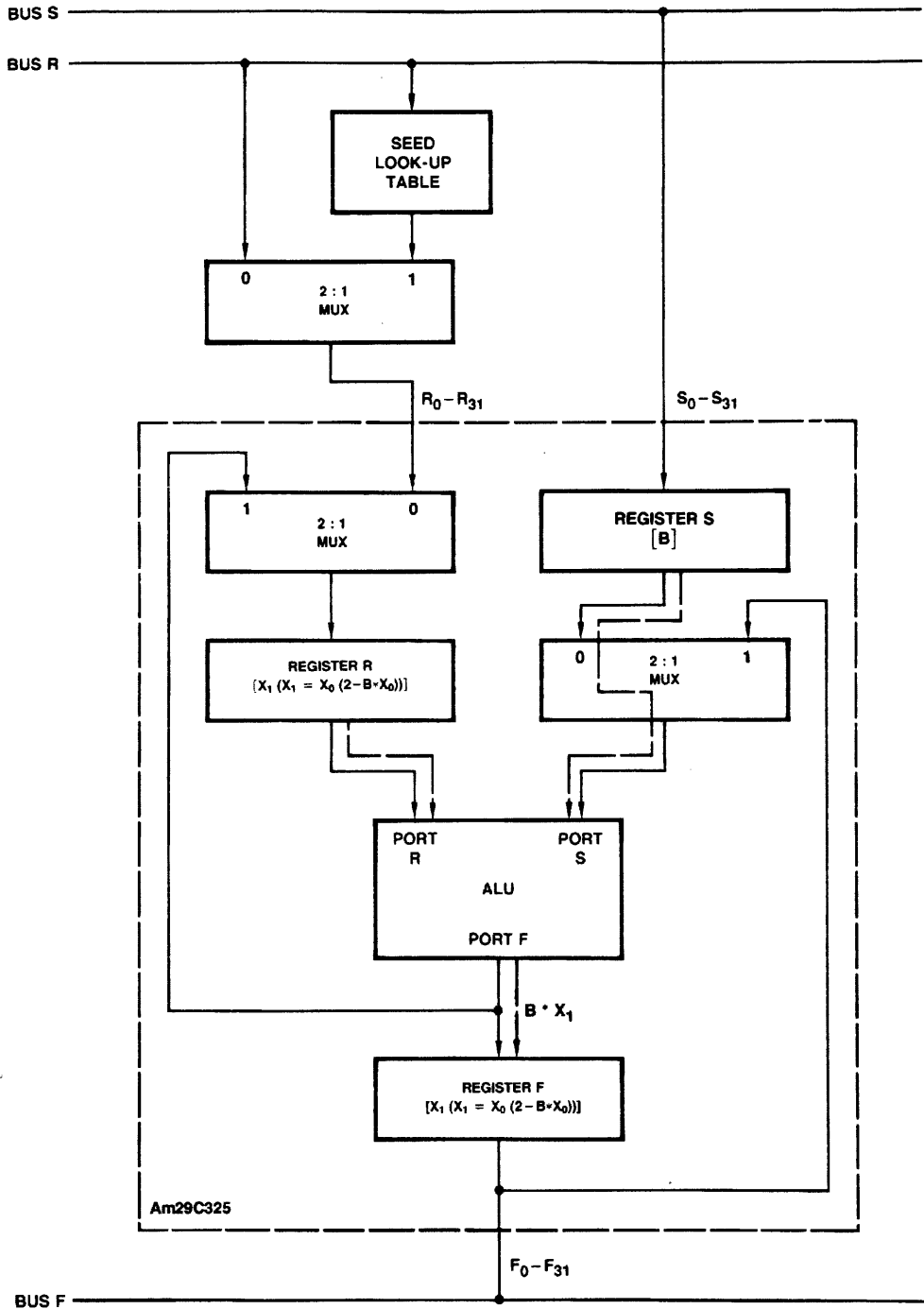


Figure C3-E. Data Flow for Step 5 of the Reciprocal Procedure

DF006251

A tabular description of the operations above is given in Table C3. The following examples, performed in IEEE format, illustrate the process.

Example 1:

Find the reciprocal of 25.3.

Solution: The IEEE floating-point representation for 25.3 is 41CA6666₁₆. The reciprocal process is begun by feeding this value to both the seed look-up table

and port S. The look-up table produces the value .03952789₁₀ (3D21E800₁₆). The reciprocal is evaluated using the procedure described above; register values for each step are given in Table C4. The expected result, to the precision of the floating-point word, is .03952569₁₀ (3D21E5B1₁₆). In this case the expected result is produced after the first iteration. All subsequent iterations produce the same result, and are therefore unnecessary.

TABLE C3. SEQUENCE OF EVENTS FOR EVALUATING RECIPROCALLS

Clock Cycle	I ₀ -I ₂	I ₃	I ₄	ENR	ENS	ENF	Register R	Register S	Register F
1	Y	X	0	0	0	X	-	-	-
2	R TIMES S	0	X	1	1	0	X ₀	B	-
3	2 MINUS S	1	X	1	1	0	X ₀	B	B*X ₀
4	R TIMES S	1	1	0	1	0	X ₀	B	2 - B*X ₀
5	R TIMES S	0	X	1	1	0	X ₁ (= X ₀ (2 - B*X ₀))	B	X ₁ (= X ₀ (2 - B*X ₀))
6	2 MINUS S	1	X	1	1	0	X ₁	B	B*X ₁
7	R TIMES S	1	1	0	1	0	X ₁	B	2 - B*X ₁
8	R TIMES S	0	X	1	1	0	X ₂ (= X ₁ (2 - B*X ₁))	B	X ₂ (= X ₁ (2 - B*X ₁))

} First iteration
} Second iteration

X = DON'T CARE

TABLE C4. INPUT BUS AND REGISTER VALUES FOR EXAMPLE 1

Clock Cycle	R Input	S Input	Register R	Register S	Register F
1	3D21E800 (.03952789)	41CA6666 ₁₆ (25.3)	-	-	-
2	-	-	3D21E800 ₁₆ (.03952789)	41CA6666 ₁₆ (25.3)	-
3	-	-	3D21E800 ₁₆ (.03952789)	41CA6666 ₁₆ (25.3)	3F8001D3 ₁₆ (1.0000556)
4	-	-	3D21E800 ₁₆ (.03952789)	41CA6666 ₁₆ (25.3)	3F7FFC5A ₁₆ (.99984419)
5	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3D21E5B1 ₁₆ (.03952569)
6	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3F7FFFFF ₁₆ (.99999994)
7	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3F800000 ₁₆ (1.0)
8	-	-	3D21E5B1 ₁₆ (.03952569)	41CA6666 ₁₆ (25.3)	3D21E5B1 ₁₆ (.03952569)

← Result of first iteration

← Result of second iteration

Example 2:

Find the reciprocal of -0.4725 .

Solution: The IEEE floating-point representation for -0.4725 is $BEF1EB85_{16}$. The reciprocal process is begun by feeding this value to both the seed look-up table and port S. The look-up table produces the value -2.11621094_{10} ($C0077000_{16}$). The reciprocal is

evaluated using the procedure described above; register values for each step are given in Table C5. The expected result, to the precision of the floating-point word, is -2.116402_{10} ($C0077322_{16}$). In this case the expected result is produced after the first iteration. All subsequent iterations produce the same result, and are therefore unnecessary.

TABLE C5. INPUT BUS AND REGISTER VALUES FOR EXAMPLE 2

Clock Cycle	R Input	S Input	Register R	Register S	Register F
1	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	-	-	-
2	-	-	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	-
3	-	-	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	$3F7FFA14_{16}$ (0.99990963)
4	-	-	$C0077000_{16}$ (-2.1162109)	$BEF1EB85_{16}$ (-0.4725)	$3F8002F6_{16}$ (1.0000904)
5	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$C0077322_{16}$ (-2.116402)
6	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$3F800000_{16}$ (1.0)
7	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$3F800000_{16}$ (1.0)
8	-	-	$C0077322_{16}$ (-2.116402)	$BEF1EB85_{16}$ (-0.4725)	$C0077322_{16}$ (-2.116402)

← Result of first iteration

← Result of second iteration

APPENDIX D

SUMMARY OF FLAG OPERATION

Tables D1, D2, and D3 summarize flag operation for the IEEE mode, the DEC mode, and for the IEEE-TO-DEC and DEC-TO-IEEE operations.

TABLE D1. FLAG SUMMARY FOR IEEE MODE

Operation	Condition(s)	INV	OVF	UNF	INE	ZER	NAN
Any operation listed in the IEEE Invalid Operations Table		H	L	L	L	L	H
R PLUS S R MINUS S R TIMES S 2 MINUS S	Input operands are finite $ \text{rounded result} \geq 2^{126}$	L	H	L	H	L	L
R PLUS S R MINUS S R TIMES S	$0 < \text{rounded result} < 2^{-126}$	L	L	H	H	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result does not equal infinitely precise result	L	*	*	H	*	L
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result is zero	L	L	*	*	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S FP-TO-INT	Final result is a NAN	*	L	L	L	L	H

Notes: INV = Invalid operation flag
 OVF = Overflow flag
 UNF = Underflow flag
 INE = Inexact flag
 ZER = Zero flag
 NAN = NAN flag
 L = LOW
 H = HIGH
 * = State of flag depends on the input operands and the operation performed

TABLE D2. FLAG SUMMARY FOR DEC MODE

Operation	Condition(s)	INV	OVF	UNF	INE	ZER	NAN
FP-TO-INT	Rounded result $> 2^{31}-1$ or rounded result $< -2^{31}$	H	L	L	L	L	H
FP-TO-INT	Input is a DEC-reserved operand	L	L	L	L	L	H
R PLUS S R MINUS S R TIMES S 2 MINUS S	$ \text{Rounded result} \geq 2^{127}$	L	H	L	H	L	H
R PLUS S R MINUS S R TIMES S	$0 < \text{rounded result} < 2^{-128}$	L	L	H	H	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result does not equal infinitely precise result	L	*	*	H	*	*
R PLUS S R MINUS S R TIMES S 2 MINUS S INT-TO-FP FP-TO-INT	Final result is zero	L	L	*	*	H	L
R PLUS S R MINUS S R TIMES S 2 MINUS S FP-TO-INT	Final result is a DEC-reserved operand	*	*	L	L	L	H

Notes: INV = Invalid operation flag
 OVF = Overflow flag
 UNF = Underflow flag
 INE = Inexact flag
 ZER = Zero flag
 NAN = NAN flag
 L = LOW

H = HIGH
 * = State of flag depends on the input operands and the operation performed

TABLE D3. FLAG SUMMARY FOR IEEE-TO-DEC AND DEC-TO-IEEE CONVERSIONS

Operation	Condition(s)	INV	OVF	UNF	INE	ZER	NAN
IEEE-TO-DEC	Input is a NAN	H	L	L	L	L	H
IEEE-TO-DEC	$ \text{Input} \geq 2^{127}$	L	H	L	H	L	H
DEC-TO-IEEE	Input is a DEC-reserved operand	H	L	L	L	L	H
DEC-TO-IEEE	$0 < \text{rounded result} < 2^{-126}$	L	L	H	H	H	L
DEC-TO-IEEE IEEE-TO-DEC	Final result is zero	L	L	*	*	H	L

Notes: INV = Invalid operation flag
 OVF = Overflow flag
 UNF = Underflow flag
 INE = Inexact flag
 ZER = Zero flag
 NAN = NAN flag
 L = LOW

H = HIGH
 * = State of flag depends on the input operands and the operation performed

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Case Temperature Under Bias -55 to +125°C
 Supply Voltage to Ground Potential
 Continuous -0.3 to +7.0 V
 DC Voltage Applied to Outputs
 for HIGH Output State -0.3 V to +V_{CC} + 0.3 V
 DC Input Voltage -0.3 to V_{CC} + 0.3 V
 DC Output Current, into LOW Outputs 30 mA
 DC Input Current -10 to +10 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices

Temperature, Case (T_A) 0 to +70°C
 Supply Voltage (V_{CC}) +4.75 to +5.25 V

Military* (M) Devices

Temperature (T_A) -55 to +125°C
 Supply Voltage (V_{CC}) +4.5 V to +5.5 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

*Military product 100% tested at T_A = +25°C, +125°C, and -55°C.

DC CHARACTERISTICS over operating range unless otherwise specified (for APL Products, Group A, Subgroups 1, 2, 3 are tested unless otherwise noted)

Parameter Symbol	Parameter Description	Test Conditions (Note 1)		Min.	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH}	I _{OH} = 0.4 mA	2.4		V
V _{OL}	Output LOW Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH}	I _{OL} = 8 mA for Y-BUS, 4 mA for All Other Pins		0.5	V
V _{IH}	Guaranteed Input Logical HIGH Voltage (Note 2)			2.0		V
V _{IL}	Guaranteed Input Logical LOW Voltage (Note 2)				0.8	V
I _{IL}	Input LOW Current	V _{CC} = Max. V _{IN} = 0.5 V			-10	μA
I _{IH}	Input HIGH Current	V _{CC} = Max. V _{IN} = V _{CC} - 0.5 V			10	μA
I _{OZH}	Off-State (HIGH Impedance) Output Current	V _{CC} = Max., V _O = 2.4 V			10	μA
I _{OZL}	Off-State (HIGH Impedance) Output Current	V _{CC} = Max., V _O = 0.5 V			-10	μA
I _{CC}	Static Power Supply Current	V _{CC} = Max., V _{IN} = V _{CC} or GND, I _O = 0 μA			I _{CC} = 30 mA (COM and MIL)	
C _{PD}	Power Dissipation Capacitance (Note 3)	V _{CC} = 5.0 V, T _A = 25°C, No Load			pF Typical	

Notes: 1. V_{CC} conditions shown as Min. or Max. refer to the commercial and military V_{CC} limits.
 2. These input levels provide zero-noise immunity and should only be statically tested in a noise-free environment (not functionally tested).
 3. C_{PD} determines the no-load dynamic current consumption:
 I_{CC} (Total) = I_{CC} (Static) + C_{PD} V_{CC} f, where f is the switching frequency of the majority of the internal nodes, normally one-half of the clock frequency.

SWITCHING CHARACTERISTICS over COMMERCIAL operating range

No.	Parameter Symbol	Parameter Description		Test Conditions	29C325		29C325-1		29C325-2		Unit	
					Min.	Max.	Min.	Max.	Min.	Max.		
1	t _{ASC}	Clocked Add, Subtract Time (R PLUS S, R MINUS S, 2 MINUS S)				130		98		78	ns	
2	t _{MC}	Clocked Multiply Time (R TIMES S)				130		98		78	ns	
3	t _{CC}	Clocked Conversion Time (INT-TO-FP, FP-TO-INT, IEEE-TO-DEC, DEC-TO-IEEE)				130		98		78	ns	
4	t _{ASUC}	Unclocked Add, Subtract Time (R, S to F, Flags) for R PLUS S, R MINUS S, and 2 MINUS S Instructions		FT ₀ = HIGH FT ₁ = HIGH		145		125		100	ns	
5	t _{MUC}	Unclocked Multiply Time (R, S to F, Flags) for R TIMES S Instruction				145		125		100	ns	
6	t _{CUC}	Unclocked Conversion Time (R, S to F, Flags) for INT-TO-FP, FP-TO-INT, IEEE-TO-DEC and DEC-TO-IEEE Instructions				145		125		100	ns	
7	t _{PWH}	Clock Pulse Width HIGH			20		15		15		ns	
8	t _{PWL}	Clock Pulse Width LOW			20		15		15		ns	
9	t _{PDOF1}	Clock to F ₀ - F ₃₁ and Flag Outputs			FT ₀ = LOW FT ₁ = HIGH		136		118		94	ns
10	t _{PDOF2}			FT ₁ = LOW		25		20		16	ns	
11	t _{PZL}	OE Enable Time	Z to LOW	S16/32 = HIGH ONEBUS = LOW		23		20		16	ns	
12	t _{PZH}		Z to HIGH			23		20		16	ns	
13	t _{PLZ}	OE Disable Time	LOW to Z			23		20		16	ns	
14	t _{PHZ}		HIGH to Z			23		20		16	ns	
15	t _{PZL16}	Clock ↑ to F ₀ - F ₁₅ Enable, 16-Bit I/O Mode	Z to LOW			27		22		18	ns	
16	t _{PZH16}		Z to HIGH			27		22		18	ns	
17	t _{PLZ16}	Clock ↓ to F ₀ - F ₁₅ Disable, 16-Bit I/O Mode	LOW to Z			29		22		18	ns	
18	t _{PHZ16}		HIGH to Z			29		22		18	ns	
19	t _{PZL16}	Clock ↓ to F ₁₆ - F ₃₁ Enable, 16-Bit I/O Mode	Z to LOW			30		22		18	ns	
20	t _{PZH16}		Z to HIGH			30		22		18	ns	
21	t _{PLZ16}	Clock ↑ to F ₁₆ - F ₃₁ Disable, 16-Bit I/O Mode	LOW to Z			25		21		17	ns	
22	t _{PHZ16}		HIGH to Z			25		21		17	ns	
23	t _{SCE}	Register Clock Enable Setup Time			FT ₀ = LOW FT ₁ = LOW	15		15		15		ns
24	t _{HCE}	Register Clock Enable Hold Time			FT ₀ = LOW FT ₁ = LOW	0		0		0		ns
25	t _{SD1}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Setup Time (Note 1)		FT ₀ = LOW	15		15		15		ns	
26	t _{HD1}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Hold Time (Note 1)			0		0		0		ns	
27	t _{SD2}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Setup Time (Note 1)		FT ₀ = HIGH FT ₁ = LOW	136		118		118		ns	
28	t _{HD2}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Hold Time (Note 1)			0		0		0		ns	
29	t _{SI02}	I ₀ - I ₂ Instruction Select Setup Time		FT for Destination Register = LOW	136		118		118		ns	
30	t _{HI02}	I ₀ - I ₂ Instruction Select Hold Time			0		0		0		ns	
31	t _{PI02}	I ₀ - I ₂ Instruction Select to F ₀ - F ₃₁ , Flags		FT ₁ = HIGH		136		118		118	ns	
32	t _{SI3}	I ₃ Port S Input Select Setup Time		FT ₁ = LOW	136		118		118		ns	
33	t _{HI3}	I ₃ Port S Input Select Hold Time			0		0		0		ns	
34	t _{SI4}	I ₄ Register R Input Select Setup Time (Note 1)		FT ₀ = LOW	15		15		15		ns	
35	t _{HI4}	I ₄ Register R Input Select Hold Time (Note 1)				0	0		0		ns	
36	t _{SRM}	Round Mode Select Setup Time		FT for Destination Register = LOW	50		46		46		ns	
37	t _{HRM}	Round Mode Select Hold Time				0	0		0		ns	
38	t _{PRF}	Round Mode Select to F ₀ - F ₃₁ , Flags		FT ₁ = HIGH		64		58		58	ns	

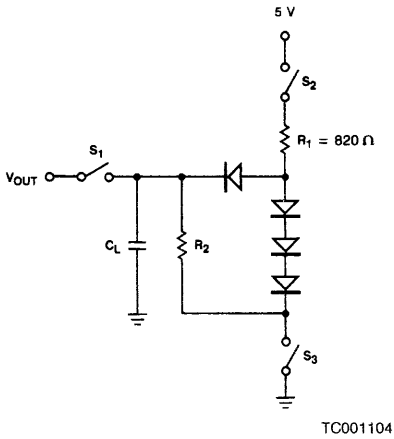
Notes: 1. See timing diagram for desired mode of operation to determine clock edge to which these setup and hold times apply.

SWITCHING CHARACTERISTICS over **MILITARY** operating range (for APL Products, Group A, Subgroups 9, 10, 11 are tested unless otherwise noted)

No.	Parameter Symbol	Parameter Description		Test Conditions	29C325		Unit	
					Min.	Max.		
1	t _{ASC}	Clocked Add, Subtract Time (R PLUS S, R MINUS S, 2 MINUS S)				145	ns	
2	t _{MC}	Clocked Multiply Time (R TIMES S)				145	ns	
3	t _{CC}	Clocked Conversion Time (INT-TO-FP, FP-TO-INT, IEEE-TO-DEC, DEC-TO-IEEE)				145	ns	
4	t _{ASUC}	Unlocked Add, Subtract Time (R, S to F, Flags) for R PLUS S, R MINUS S, and 2 MINUS S Instructions		FT ₀ = HIGH FT ₁ = HIGH		160	ns	
5	t _{MUC}	Unlocked Multiply Time (R, S to F, Flags) for R TIMES S Instruction				160	ns	
6	t _{CUC}	Unlocked Conversion Time (R, S to F, Flags) for INT-TO-FP, FP-TO-INT, IEEE-TO-DEC and DEC-TO-IEEE Instructions				160	ns	
7	t _{PWH}	Clock Pulse Width HIGH			20		ns	
8	t _{PWL}	Clock Pulse Width LOW			20		ns	
9	t _{PDOF1}	Clock to F ₀ - F ₃₁ and Flag Outputs		FT ₀ = LOW FT ₁ = HIGH		152	ns	
10	t _{PDOF2}				FT ₁ = LOW		30	ns
11	t _{PZL}	OE Enable Time	Z to LOW	S16/32 = HIGH ONEBUS = LOW		26	ns	
12	t _{PZH}		Z to HIGH			26	ns	
13	t _{PLZ}	OE Disable Time	LOW to Z			26	ns	
14	t _{PHZ}		HIGH to Z			26	ns	
15	t _{PZL16}	Clock ↑ to F ₀ - F ₁₅ Enable, 16-Bit I/O Mode	Z to LOW		S16/32 = HIGH ONEBUS = LOW		30	ns
16	t _{PZH16}		Z to HIGH				30	ns
17	t _{PLZ16}	Clock ↓ to F ₀ - F ₁₅ Disable, 16-Bit I/O Mode	LOW to Z			33	ns	
18	t _{PHZ16}		HIGH to Z			33	ns	
19	t _{PZL16}	Clock ↓ to F ₁₆ - F ₃₁ Enable, 16-Bit I/O Mode	Z to LOW	S16/32 = HIGH ONEBUS = LOW			34	ns
20	t _{PZH16}		Z to HIGH				34	ns
21	t _{PLZ16}	Clock ↑ to F ₁₆ - F ₃₁ Disable, 16-Bit I/O Mode	LOW to Z			28	ns	
22	t _{PHZ16}		HIGH to Z			28	ns	
23	t _{SCE}	Register Clock Enable Setup Time			FT ₀ = LOW FT ₁ = LOW	15		ns
24	t _{HCE}	Register Clock Enable Hold Time			FT ₀ = LOW FT ₁ = LOW	0		ns
25	t _{SD1}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Setup Time (Note 1)		FT ₀ = LOW	15		ns	
26	t _{HD1}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Hold Time (Note 1)			0		ns	
27	t _{SD2}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Setup Time (Note 1)		FT ₀ = HIGH FT ₁ = LOW	152		ns	
28	t _{HD2}	R ₀ - R ₃₁ , S ₀ - S ₃₁ Hold Time (Note 1)			-30		ns	
29	t _{SI02}	I ₀ - I ₂ Instruction Select Setup Time		FT for Destination Register = LOW	152		ns	
30	t _{HI02}	I ₀ - I ₂ Instruction Select Hold Time			0		ns	
31	t _{PDI02}	I ₀ - I ₂ Instruction Select to F ₀ - F ₃₁ , Flags		FT ₁ = HIGH		152	ns	
32	t _{SI3}	I ₃ Port S Input Select Setup Time		FT ₁ = LOW	152		ns	
33	t _{HI3}	I ₃ Port S Input Select Hold Time			0		ns	
34	t _{SI4}	I ₄ Register R Input Select Setup Time (Note 1)		FT ₀ = LOW	15		ns	
35	t _{HI4}	I ₄ Register R Input Select Hold Time (Note 1)			0		ns	
36	t _{SRM}	Round Mode Select Setup Time		FT for Destination Register = LOW	65		ns	
37	t _{HRM}	Round Mode Select Hold Time			0		ns	
38	t _{PRF}	Round Mode Select to F ₀ - F ₃₁ , Flags		FT ₁ = HIGH		80	ns	

Notes: 1. See timing diagram for desired mode of operation to determine clock edge to which these setup and hold times apply.

SWITCHING TEST CIRCUITS

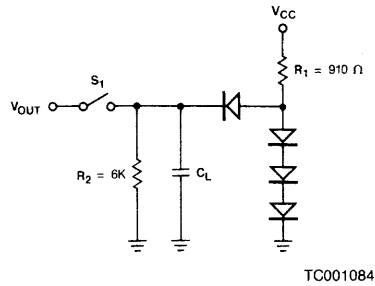


$$R_1 = I_{OL} + \frac{V_{OL}}{1K}$$

$$R_2 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OH}}$$

A. Three-State Outputs

- Notes:
1. $C_L = 50$ pF includes scope probe, wiring, and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 S_1 and S_2 are closed while S_3 is open for t_{pZL} test.
 4. $C_L = 5.0$ pF for output disable tests.

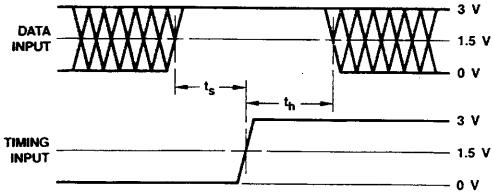


$$R_1 = \frac{2.4 V}{I_{OH}}$$

$$R_2 = \frac{5.0 - V_{BE} - V_{OL}}{I_{OL} + \frac{V_{OL}}{R_2}}$$

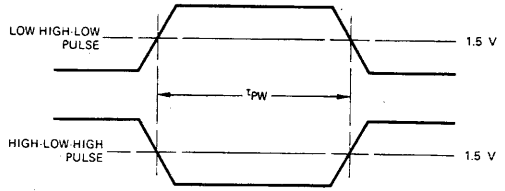
B. Normal Outputs

SWITCHING TEST WAVEFORMS



WFR02970

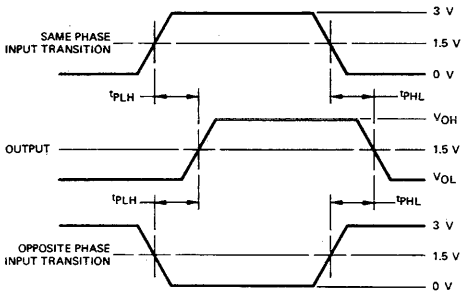
- Notes: 1. Diagram shown for HIGH data only.
Output transition may be opposite sense.
2. Cross hatched area is don't care condition.



WFR02790

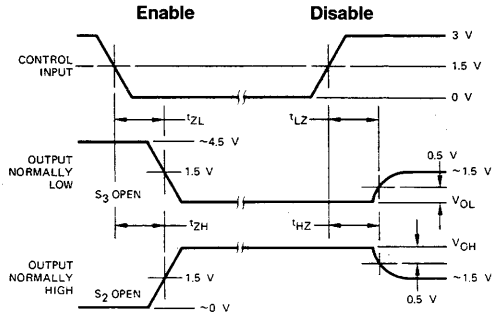
Pulse Width

Set-Up, Hold, and Release Times



WFR02980

Propagation Delay



WFR02660

- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
2. S₁, S₂ and S₃ of Load Circuit are closed except where shown.

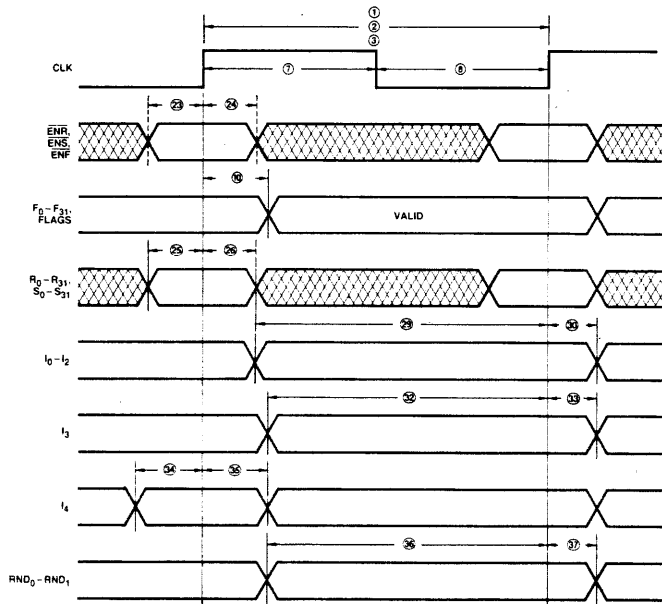
Enable and Disable Times

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
—————	MUST BE STEADY	WILL BE STEADY
//	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
\\	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE, ANY CHANGE PERMITTED	CHANGING, STATE UNKNOWN
// \\	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

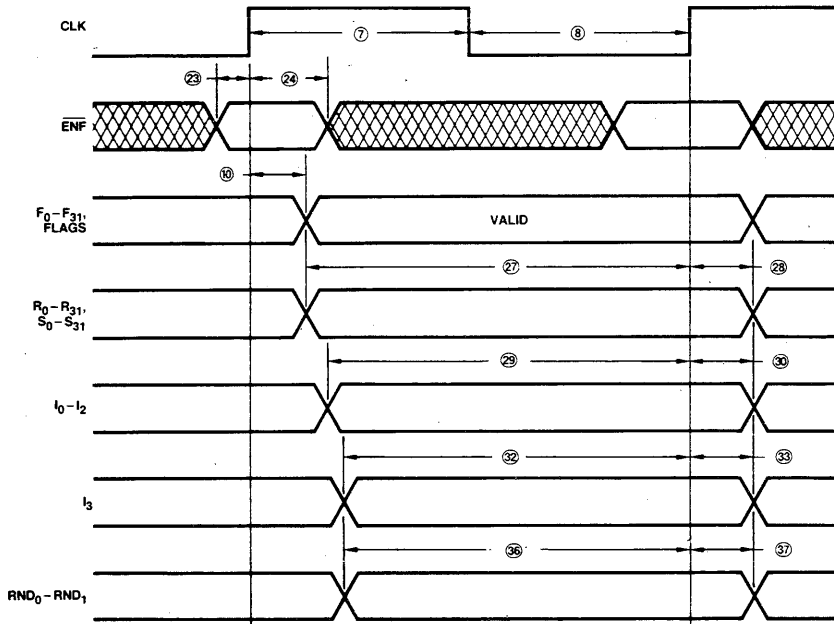
KS000010



WF023760

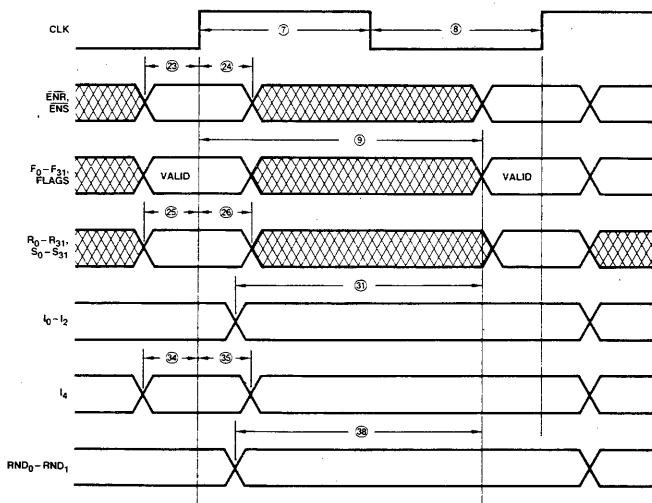
**Clocked Operation: FT₀ = LOW
FT₁ = LOW**

SWITCHING WAVEFORMS (Cont'd.)



WF023770

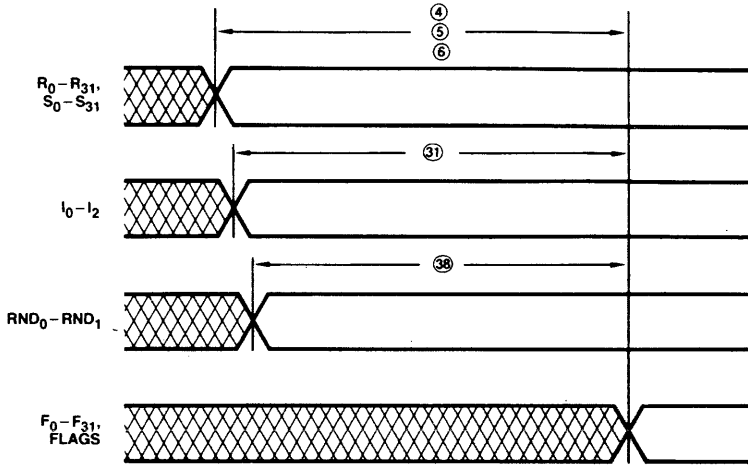
**Clocked Operation: $FT_0 = \text{HIGH}$
 $FT_1 = \text{LOW}$**



WF023780

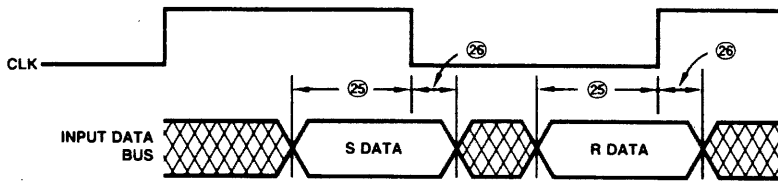
**Clocked Operation: $FT_0 = \text{LOW}$
 $FT_1 = \text{HIGH}$**

SWITCHING WAVEFORMS (Cont'd.)



WF023790

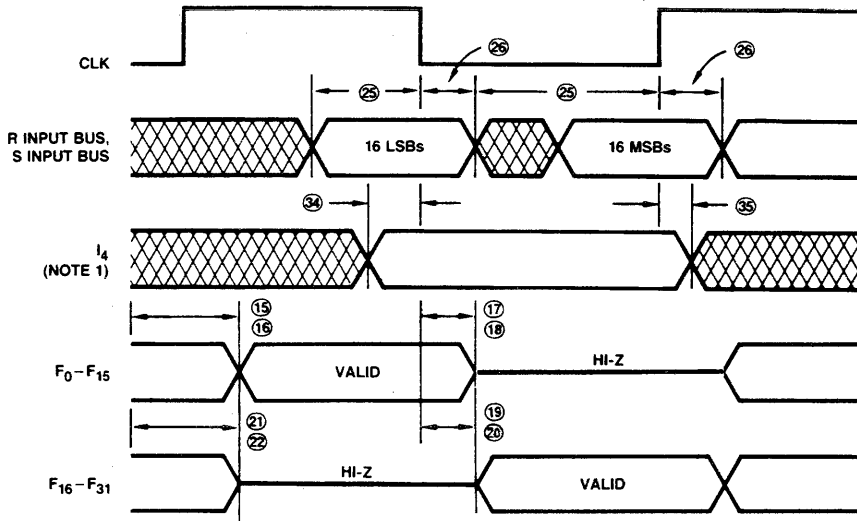
Flow-Through Operation ($FT_0 = \text{HIGH}, FT_1 = \text{HIGH}$)



WF023800

32-Bit, Single-Input Bus Mode

SWITCHING WAVEFORMS (Cont'd.)

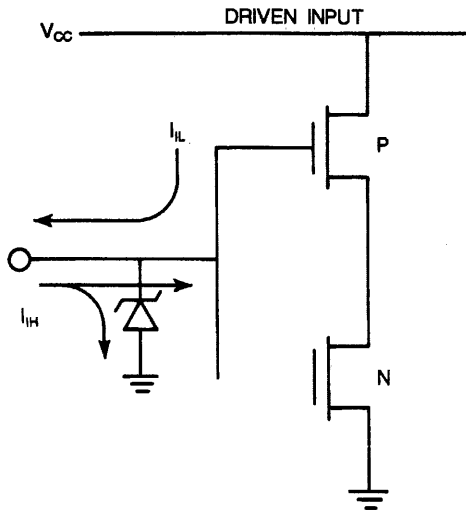


WF023810

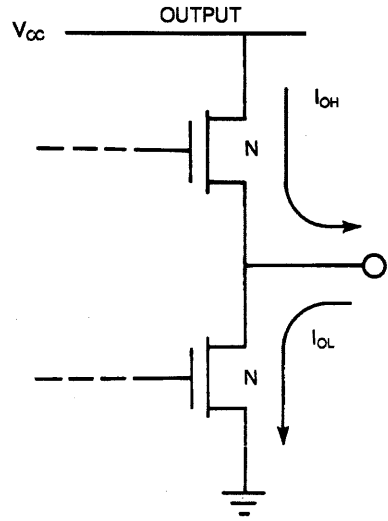
Note 1. I₄ has special setup and hold time requirements in this mode. All other control signals have timing requirements as shown in the diagram "Clocked operation, FT₀ = LOW, FT₁ = LOW."

16-Bit, Two-Input Bus Mode

INPUT/OUTPUT CIRCUIT DIAGRAMS



IC000860



IC000870

Am29C327

CMOS Double-Precision Floating-Point Processor



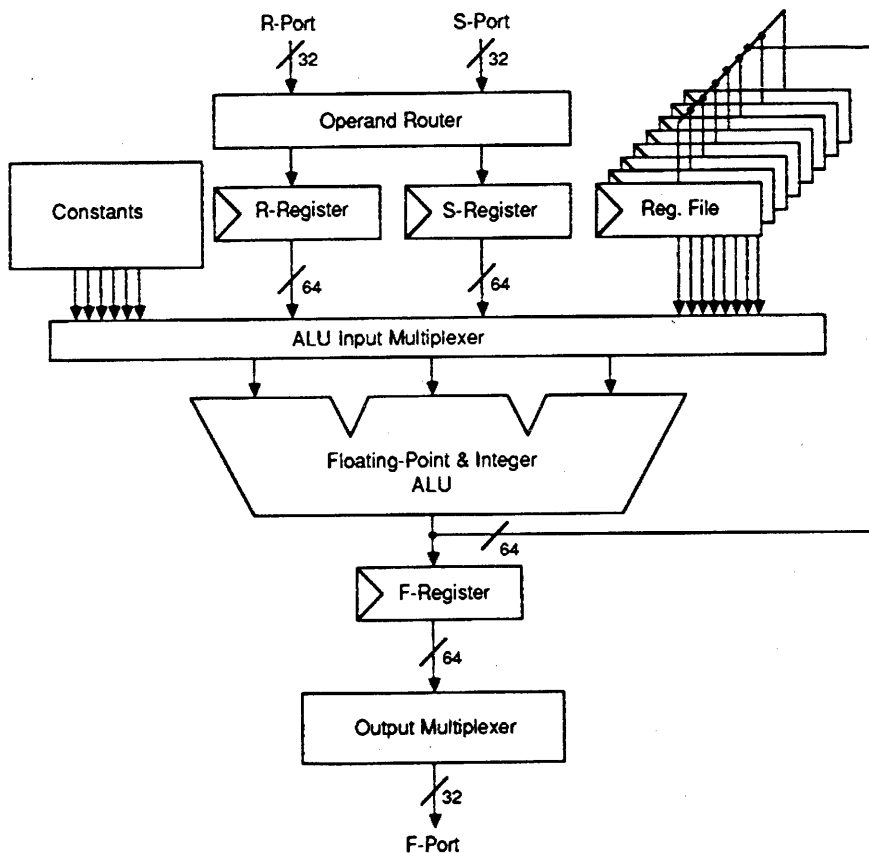
Am29C327

ADVANCE INFORMATION

DISTINCTIVE CHARACTERISTICS

- High-performance double-precision floating-point processor
- Comprehensive floating-point and integer instruction sets
- Single VLSI device performs single-, double-, and mixed-precision operations
- Performs conversions between precisions and between data formats
- Compatible with industry-standard floating-point formats
 - IEEE 754 format
 - DEC F, DEC D, and DEC G formats
 - IBM system/370 format
- Exact IEEE compliance for denormalized numbers with no speed penalty
- Eight-deep register file for intermediate results and on-chip 64-bit data path facilitates compound operations; e.g., Newton-Raphson division, sum-of-products, and transcendentals
- Supports pipelined or flow-through operation
- Fabricated with Advanced Micro Devices' 1.2 micron CMOS process

SIMPLIFIED SYSTEM DIAGRAM



BD007470

GENERAL DESCRIPTION

The Am29C327 double-precision floating-point processor is a single VLSI device that implements an extensive floating-point and integer instruction set, and can perform single-, double- or mixed-precision operations. The three most popular floating-point formats - IEEE, DEC, and IBM - are supported. IEEE operations comply with Standard 754, with direct implementation of special features such as gradual underflow and trap handling.

The Am29C327 consists of a 64-bit ALU, a 64-bit datapath, and a control unit. The ALU has three data input ports, and can perform compound operations of the form $(A * B) + C$. The data path comprises two 64-bit input operand registers, an 8-by-64-bit register file for storage of intermediate results, three operand-selection multiplexers that provide for orthogo-

nal selection of input operands, a 64-bit output register, and an output multiplexer that allows access to the 32 MSBs or 32 LSBs of the result data. Control signals determine the operation to be performed, the source of operands, operand precision, rounding mode, and other aspects of device operation.

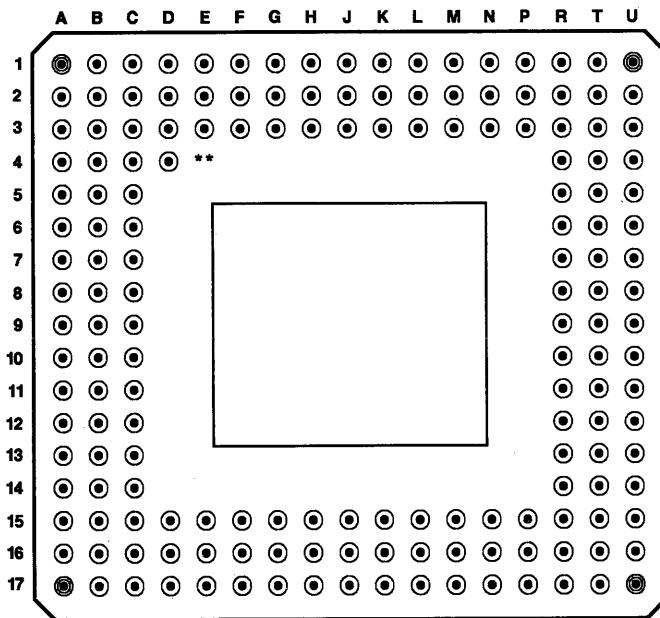
Operations can be performed in either of two modes: flow-through or pipelined. In the flow-through mode, the ALU is completely combinatorial; this mode is best suited for scalar operations. Pipelined mode divides the ALU into one or two pipelined stages, for use in vector operations, as often found in graphics or signal processing.

Fabricated with AMD's 1.2 micron technology, the Am29C327 is housed in a 169-lead pin-grid-array (PGA) package.

RELATED AMD PRODUCTS

Part No.	Description
Am29C10A	CMOS Microprogram Controller
Am29C116	CMOS Minimum Power 16-Bit Microprocessor
Am29C117	CMOS Two-Port 16-Bit Microprocessor
Am29PL141	Field-Programmable Controller (FPC)
Am29C323	CMOS 32-Bit Parallel Multiplier
Am29C325	CMOS 32-Bit Floating-Point Processor
Am29C331	CMOS 16-Bit Microprogram Sequencer
Am29C332	CMOS 32-Bit Arithmetic Logic Unit
Am29C334	CMOS Four-Port Dual-Access Register File

CONNECTION DIAGRAM 169-Lead PGA* Bottom View



CD009761

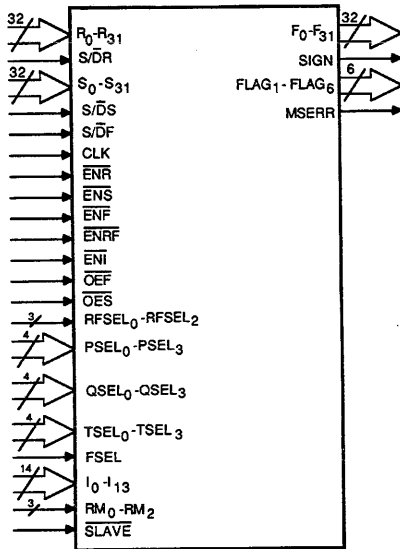
*Pinout observed from pin side of package.

**Alignment pin (not connected internally).

PIN DESIGNATIONS
(Sorted by Pin No.)

PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME	PIN NO.	PIN NAME
A-1		C-9		J-15		R-10	
A-2		C-10		J-16		R-11	
A-3		C-11		J-17		R-12	
A-4		C-12		K-1		R-13	
A-5		C-13		K-2		R-14	
A-6		C-14		K-3		R-15	
A-7		C-15		K-15		R-16	
A-8		C-16		K-16		R-17	
A-9		C-17		K-17		T-1	
A-10		D-1		L-1		T-2	
A-11		D-2		L-2		T-3	
A-12		D-3		L-3		T-4	
A-13		D-15		L-15		T-5	
A-14		D-16		L-16		T-6	
A-15		D-17		L-17		T-7	
A-16		E-1		M-1		T-8	
A-17		E-2		M-2		T-9	
B-1		E-3		M-3		T-10	
B-2		E-15		M-15		T-11	
B-3		E-16		M-16		T-12	
B-4		E-17		M-17		T-13	
B-5		F-1		N-1		T-14	
B-6		F-2		N-2		T-15	
B-7		F-3		N-3		T-16	
B-8		F-15		N-15		T-17	
B-9		F-16		N-16		U-1	
B-10		F-17		N-17		U-2	
B-11		G-1		P-1		U-3	
B-12		G-2		P-2		U-4	
B-13		G-3		P-3		U-5	
B-14		G-15		P-15		U-6	
B-15		G-16		P-16		U-7	
B-16		G-17		P-17		U-8	
B-17		H-1		R-1		U-9	
C-1		H-2		R-2		U-10	
C-2		H-3		R-3		U-11	
C-3		H-15		R-4		U-12	
C-4		H-16		R-5		U-13	
C-5		H-17		R-6		U-14	
C-6		J-1		R-7		U-15	
C-7		J-2		R-8		U-16	
C-8		J-3		R-9		U-17	

LOGIC SYMBOL



LS003081

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of:

- a. **Device Number**
- b. **Speed Option** (if applicable)
- c. **Package Type**
- d. **Temperature Range**
- e. **Optional Processing**

AM29C327

G

C

B

e. OPTIONAL PROCESSING

Blank = Standard processing
B = Burn-in

d. TEMPERATURE RANGE

C = Commercial (0 to +70°C)

c. PACKAGE TYPE

G = 169-Lead Pin Grid Array without Heatsink (CGX169)

b. SPEED OPTION

Not Applicable

a. DEVICE NUMBER/DESCRIPTION

Am29C327
Double-Precision Floating-Point Processor

Valid Combinations	
AM29C327	GC, GCB

Valid Combinations

Valid Combinations list configurations planned to be supported in volume for this device. Consult the local AMD sales office to confirm availability of specific valid combinations, to check on newly released combinations, and to obtain additional data on AMD's standard military grade products.

PIN DESCRIPTION

CLK Clock (Input)

Clock input to all registers.

ENF F Register Enable (Input; Active LOW)

When $\overline{\text{ENF}}$ is HIGH, the contents of the F register are static. When $\overline{\text{ENF}}$ is LOW, the ALU output data is clocked into the F register on the next LOW-to-HIGH transition of CLK. Note that the F register can be made transparent by setting the mode register bit M17 HIGH (as described in the Mode Register Description section); when the F register is transparent, $\overline{\text{ENF}}$ has no effect.

ENI Instruction Register Enable (Input; Active LOW)

When $\overline{\text{ENI}}$ is LOW, an instruction word is clocked into the instruction register on the next LOW-to-HIGH transition of CLK. The instruction word comprises the following fields: P, Q, and T-multiplexer control inputs, rounding modes, ALU instruction inputs, and the precision of the output operand.

ENR R Register Enable (Input; Active LOW)

When $\overline{\text{ENR}}$ is HIGH, the contents of the R register are static. When $\overline{\text{ENR}}$ is LOW, new data is loaded into the R register on the next LOW-to-HIGH transition of CLK.

ENRF Register File Enable (Input; Active LOW)

When $\overline{\text{ENRF}}$ is HIGH, the contents of the register file are static. When $\overline{\text{ENRF}}$ is LOW, the ALU output operand is clocked into the register file on the next LOW-to-HIGH transition of CLK.

ENS S Register Enable (Input; Active LOW)

When $\overline{\text{ENS}}$ is HIGH, the contents of the S register are static. When $\overline{\text{ENS}}$ is LOW, new data is loaded into the S register on the next LOW-to-HIGH transition of CLK.

F₀ - F₃₁ F Output Bus (Output)

FLAG₁ - FLAG₆ Flag Outputs (Output)

The six flag outputs report the status of the last operation executed.

FSEL Output Multiplexer Control (Input)

When FSEL is HIGH, the most significant 32 bits of the output register are connected to the output driver. When FSEL is LOW, the least significant 32 bits of the output register are connected to the output driver.

I₀ - I₁₃ ALU Instruction Inputs (Input)

I₀ - I₁₃ select the operation to be performed by the ALU.

MSERR Master/Slave Error Flag (Output)

A HIGH level indicates a master/slave error on the current output.

OE $\overline{\text{F}}$ F Output Bus Enable (Input; Active LOW)

When $\overline{\text{OEF}}$ is HIGH, signals F₀ - F₃₁ assume a high-impedance state. When $\overline{\text{OEF}}$ is LOW (and $\overline{\text{SLAVE}}$ is HIGH), the output of the F multiplexer is placed on F₀ - F₃₁.

OES Flag Output Enable (Input)

When $\overline{\text{OES}}$ is HIGH, outputs SIGN and FLAG₁ through FLAG₆ assume a high-impedance state. When $\overline{\text{OES}}$ is LOW (and $\overline{\text{SLAVE}}$ is HIGH), these signals are enabled.

PSEL₀ - PSEL₃ P-Multiplexer Control Inputs (Input)

PSEL₀ - PSEL₃ select the data input to the ALU P-port.

QSEL₀ - QSEL₃ Q-Multiplexer Control Inputs (Input)

QSEL₀ - QSEL₃ select the data input to the ALU Q-port.

R₀ - R₃₁ R Input Bus (Input)

RFSEL₀ - RFSEL₂ Register File Select (Input)

RFSEL₀ - RFSEL₂ select the register file location (RF₀ - RF₇) to which the ALU result is to be written. Data is written to the register file if $\overline{\text{ENRF}}$ is LOW.

RM₀ - RM₂ Round Mode Control Inputs (Input)

The Am29C327 supports six rounding modes. RM₀ - RM₂ select the rounding mode to be applied to the current operation.

S₀ - S₃₁ S Input Bus (Input)

S/DF F Output Single/Double Control (Input)

When S/DF is HIGH, the ALU generates a single-precision result. When S/DF is LOW, the ALU generates a double-precision result.

S/DR R Input Single/Double Control (Input)

When S/DR is HIGH, the data loaded into the R-port is treated as single precision. When S/DR is LOW, the data loaded into the R register is treated as double precision.

S/DS S Input Single/Double Control (Input)

When S/DS is HIGH, the data loaded into the S-port is treated as single precision. When S/DS is LOW, the data loaded into the S register is treated as double precision.

SIGN Sign Flag (Output)

If the final result of the last operation was negative, SIGN is HIGH. If the final result of the last operation was not negative, SIGN is LOW.

SLAVE Master/Slave Mode Select (Input)

When $\overline{\text{SLAVE}}$ is LOW, SLAVE mode is selected. In this mode, all outputs except MSERR are disabled. When $\overline{\text{SLAVE}}$ is HIGH, MASTER mode is selected.

TSEL₀ - TSEL₃ T-Multiplexer Control Inputs (Input)

TSEL₀ - TSEL₃ select the data input to the ALU T-port.

FUNCTIONAL DESCRIPTION

Overview

The Am29C327 is a high-performance, single-chip, double-precision floating-point processor.

Architecture

The Am29C327 comprises a high-speed ALU, a 64-bit data path, and control circuitry.

The core of the Am29C327 is a 64-bit floating-point/integer ALU. This ALU takes operands from three 64-bit input ports and performs the selected operation, placing the result on a 64-bit output port. Thirteen ALU flags report operation status via the 7-bit Flag port. The ALU is completely combinatorial for

reduced latency; optional pipelining is available to boost throughput for array operations.

The data path consists of the 32-bit input buses R and S; two 64-bit input operand registers; an 8-by-64-bit register file for storage of intermediate results; three operand-selection multiplexers that provide for orthogonal selection of input operands; a 64-bit output register; and an output multiplexer that permits the selection of 32 MSBs, or 32 LSBs of data. Input operands enter the processor through the R and S buses, and are then demultiplexed and buffered for subsequent storage in registers R and S. The operand selection multiplexers route the operands to the ALU. Operation results are stored in register F, and leave the device on the 32-bit output bus F. The results can also be stored in the register file for use in subsequent operations.

Instruction Set

The Am29C327 implements 58 arithmetic and logical instructions. Thirty-five instructions operate on floating-point numbers; these instructions fall into the following categories:

- Addition/subtraction
- Multiplication
- Multiplication-accumulation
- Comparison
- Selecting the larger or smaller of two numbers
- Rounding to integral value
- Absolute value, negation
- Reciprocal seed generation
- Conversion between any of the supported floating-point formats
- Conversion of a floating-point number to an integer format, with or without a scale factor
- Pass operand

By concatenating these operations, the user can also perform division, square-root extraction, polynomial evaluation, and other functions not implemented directly.

Twenty-two instructions operate on integers, and belong to the following general categories:

- Addition/subtraction
- Multiplication
- Comparison
- Selecting the smaller or larger of two numbers
- Absolute value, negation, pass operand

- Logical operations; e.g., AND, OR, XOR, NOT
- Arithmetic, logical, and funnel shifts
- Conversion between single- and double-precision integer formats
- Conversion of an integer number to a floating-point format, with or without a scale factor

One special instruction is provided to move data.

Mixed-Precision Operations

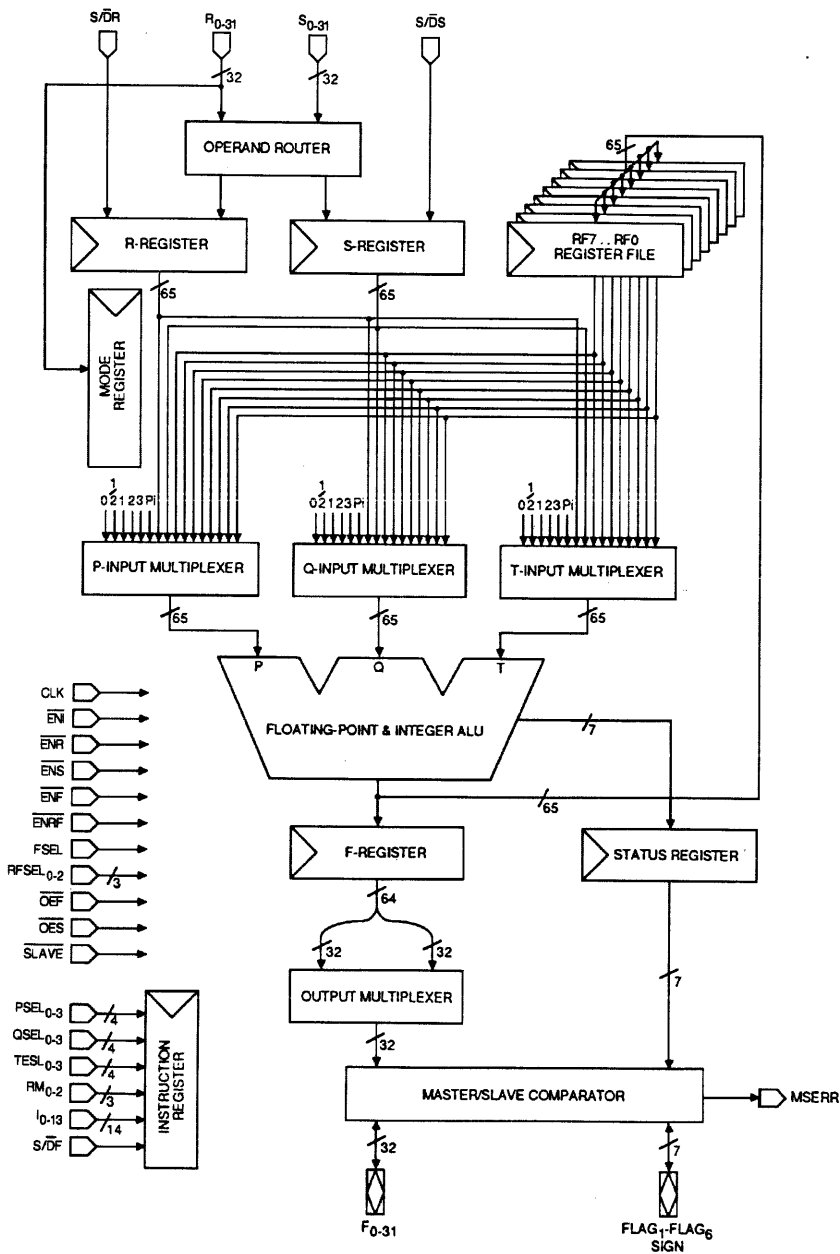
All Am29C327 instructions, floating-point or integer, can be performed with either single- or double-precision operands. In addition, the user can elect to mix precisions within an operation. All operations are performed in double-precision internally; the user specifies the precisions of the input operands and the required precision for the output operand. The necessary precision conversions are made in concert with the selected operation, with no additional cycle-time overhead.

I/O Modes

The Am29C327 supports eight I/O modes that afford flexible interface to a variety of 32- and 64-bit systems.

Fault Detection Features

The Am29C327 contains special comparison hardware to allow the operation of two processors in parallel, with one processor (the slave) checking the results produced by the other (the master). This feature is of particular importance in the design of high-reliability systems.



BD007610

Figure 1. Am29C327 Double-Precision Floating-Point Processor

Block Diagram Description

A block diagram of the Am29C327 is shown in Figure 1. The Am29C327 comprises input registers, operand selection multiplexers, instruction register, ALU, output register/register file, status register, output selection multiplexer, mode register, and the master/slave comparator.

Input Registers/Input Modes

Operands enter the processor through the R and S buses, and are then demultiplexed and buffered for subsequent storage in the 65-bit registers R and S. Input operands may be either single-precision (32-bit) or double-precision (64-bit) as specified by S/DR and S/DS. Accompanying the input registers are two 32-bit temporary registers, R-Temp and S-Temp, that allow for the overlapping of operand transfers and ALU operations. This arrangement of temporary registers and demultiplexers permits data and corresponding precision bit S/DR or S/DS to be loaded into the 65-bit R register and 65-bit S register via one of the eight input modes:

1. 32-bit-bus, double-cycle, LSWs first
2. 32-bit-bus, double-cycle, MSWs first
3. 32-bit-bus, single-cycle, LSWs first
4. 32-bit-bus, single-cycle, MSWs first
5. 64-bit-bus, double-cycle, R first
6. 64-bit-bus, double-cycle, S first
7. 64-bit-bus, single-cycle, R first
8. 64-bit-bus, single-cycle, S first

These modes are described in detail in the Input Modes Description section.

Operand Selection Multiplexers

The operand selection multiplexers route operands to the ALU. These multiplexers, as well as selecting operands from input registers R and S and register file locations RF0 – RF7, also have access to a set of constants (0, 0.5, 1, 2, 3, Pi). These constants are double-precision preprogrammed numbers for use in ALU operations, and are automatically provided in the appropriate floating-point or integer format.

Instruction Register

The instruction register stores a 32-bit word specifying the current processor operation. Included in the instruction word are fields that specify the P, Q, and T multiplexer selects, the rounding modes; the core operation to be performed by the ALU; sign-change controls for ALU input and result operands; and the single/double-precision control for the output operand. The multiplexer selects and the instruction word are described in detail in the Instruction Set section; Rounding modes are described in Appendix B.

ALU

The ALU is a combinatorial arithmetic/logic unit that performs a large repertoire of floating-point and integer operations. The

ALU has three operand inputs, and performs operations of the form $(P*Q) + T$. Most ALU operations require only one or two input operands; for example, addition requires only operands P and T, multiplication only operands P and Q, and precision conversion only operand P. Many ALU arithmetic operations allow for the independent control of operand signs, thus greatly increasing the number of arithmetic expressions that can be evaluated in a single ALU pass.

The ALU can be configured in either a flow-through mode, for which the ALU is completely combinatorial, or a pipelined mode, for which ALU operations incur one or two pipeline delays, but which results in a higher throughput than flow-through mode.

A detailed description of ALU operations appears in the Instruction Set section.

Output Register/Register File

The results of the operations performed by the ALU are stored in the 64-bit output register F. Results can also be stored in the 8-by-64-bit register file for use in subsequent operations. Each register file location contains a 65th bit indicating the precision of the operand stored in that location, thus permitting the ALU to correctly process the operand in subsequent operations.

Status Register

The status register is a 7-bit register that stores flags pertaining to the most recently performed operation. A detailed description is provided in the Instruction Set section.

Output Multiplexer

The output multiplexer routes operation results to the F bus. This multiplexer selects the 32 MSBs of the output register or the 32 LSBs.

Master/Slave Comparator

Each Am29C327 output signal has associated logic that compares that signal with the signal that the processor is providing internally to the output driver; any discrepancies are indicated by assertion of signal MSERR.

For a single processor, this output comparison detects short circuits in output signals or defective output drivers, but does not detect open circuits. It is possible to connect a second processor in parallel with the first, with the second processor's outputs disabled by assertion of signal SLAVE. The second processor detects open-circuit signals, as well as providing a check of the outputs of the first.

Mode Register

The mode register contains processor parameters that are changed infrequently. The 32-bit mode word is loaded into the register via the R bus. A detailed description of the mode register is provided in the Mode Register Description section.

Mode Register Description

The "Load Mode Register" instruction loads a 32-bit word appearing on the R port into the mode register. Data is clocked into the register on the LOW-to-HIGH transition of CLK. The register is organized as described below:

M0 – M3 — Floating-Point Format Select:

M1	M0	Primary Format
0	0	IEEE
0	1	DEC F (SINGLE), DEC D (DOUBLE)
1	0	DEC F (SINGLE), DEC G (DOUBLE)
1	1	IBM

M3	M2	Alternate Format
0	0	IEEE
0	1	DEC F (SINGLE), DEC D (DOUBLE)
1	0	DEC F (SINGLE), DEC G (DOUBLE)
1	1	IBM

Primary and Alternate Floating-Point Formats

All floating-point operations with the appropriate precisions are performed in the primary format selected by mode register bits M0 and M1 except for the two following operations:

- "Convert T to Alternate Floating-Point Format" in which the T operand is in the Primary Floating-Point Format selected by mode register bits M0 and M1, and the result generated is in the Alternate Floating-Point Format specified by mode register bits M2 and M3.
- "Convert T from Alternate Floating-Point Format" in which the T operand is in the Alternate Floating-Point Format specified by mode register bits M2 and M3, and the result is in the Primary Floating-Point Format specified by mode register bits M0 and M1.

Conversion or Scaling from Integer to Floating-Point generates a floating-point result in the Primary Floating-Point Format selected by mode register bits M0 and M1.

When mode register bits M2 and M3 are not used to specify an Alternate Floating-Point Format, they are "don't cares".

Floating-point formats are discussed in further detail in Appendix A.

M4 — Saturate Enable: If M4 is HIGH, overflowed results are replaced by the largest representable value in the selected format of the same sign as the overflowed result. If M4 is LOW, the result is not changed. If M6 is HIGH and the result format is IEEE, saturation is disabled.

M5 — IEEE Affine/Projective Select: If M5 is HIGH, affine mode is selected. If M5 is LOW, projective mode is selected. The interpretation of infinities is determined by M5. The only differences between the modes occur during the addition and subtraction of infinities.

Operation	Affine Mode	Projective Mode
$(+\infty) + (+\infty)$	Output $+\infty$	Output Quiet NAN, set invalid and reserved operand flags
$(-\infty) + (-\infty)$	Output $-\infty$	Output Quiet NAN, set invalid and reserved operand flags
$(+\infty) - (-\infty)$	Output $+\infty$	Output Quiet NAN, set invalid and reserved operand flags
$(-\infty) - (+\infty)$	Output $-\infty$	Output Quiet NAN, set invalid and reserved operand flags

If the current floating-point format is not IEEE, this bit has no effect.

M6 — IEEE Trap Enable: If M6 is HIGH and the result format is IEEE, IEEE trapped operation is enabled; the saturate (M4) and sudden underflow (M7) bits are ignored. For an underflowed result, the exponent is replaced by $e = e + 192$ (SP), or $e = e + 1536$ (DP), with the significand unchanged. For an overflowed result, the exponent is replaced by $e = e - 192$ (SP), or $e = e - 1536$ (DP), with the significand unchanged. If M6 is LOW and the result format is not IEEE, IEEE trapped operation is disabled.

M7 — IEEE Sudden Underflow Enable: If M7 is HIGH and IEEE traps are disabled (M6 LOW), all IEEE denormalized results are replaced by a zero of the same sign. If M7 is LOW, a valid denormalized number will be produced. This bit has no effect for result formats other than IEEE.

M8 — IBM Significance Mask Enable: If M8 is HIGH, certain IBM operations having intermediate results of 0 will produce a final result of 0 with the biased exponent unchanged. If M8 is LOW, these operations will produce a final result of true-zero. This bit has no effect for result formats other than IBM.

M9 — IBM Underflow Mask Enable: If M9 is HIGH, certain underflowed IBM operations will produce a normalized result with the exponent replaced by $e + 128$. If M9 is LOW, these operations will produce a final result of true-zero. This bit has no effect for result formats other than IBM.

M10: Reserved for future use (must be set to Logic 0)

M11 — Integer Multiplication Signed/Unsigned Select: If M11 is HIGH, the input operands are treated as two's-complement numbers. If M11 is LOW, the input operands are treated as unsigned numbers. This bit has no effect for operations other than integer multiplication.

M12, M13 — Integer Multiplication Format Adjust: Selects the output format for integer multiplications. The user may select either the MSBs or the LSBs of the result of an integer multiplication:

M13	M12	Output Format
0	0	LSBs
0	1	LSBs, format-adjusted
1	0	MSBs
1	1	MSBs, format adjusted

"Format-adjusted" indicates that the product is shifted left one place before the MSBs or LSBs are selected.

M14 – M16 — Input Mode: Selects the input bus mode:

M16	M15	M14	Input Mode
0	0	0	32-bit-bus, single-cycle, LSW first
0	0	1	32-bit-bus, single-cycle, MSW first
0	1	0	32-bit-bus, double-cycle, LSW first
0	1	1	32-bit-bus, double-cycle, MSW first
1	0	0	64-bit-bus, single-cycle, R first
1	0	1	64-bit-bus, single-cycle, S first
1	1	0	64-bit-bus, double-cycle, R first
1	1	1	64-bit-bus, double-cycle, S first

Additional information on input modes can be found in the Input Modes section.

M17 – F Register Feedthrough Enable: When M17 is HIGH, register F is made transparent. When M17 is LOW, the ALU output data is clocked into the F register on the next LOW-to-HIGH transition of CLK.

M18 – Status Register Feedthrough Enable: When M18 is HIGH, the status register is made transparent. When M18 is LOW, the output flags are clocked into the status register on the next LOW-to-HIGH transition on CLK.

M19, M20 – Pipeline Mode Select:

M20	M19	Pipeline Mode
0	X	Flow-through mode
1	0	Single-pipeline mode for all operations
1	1	Double-pipeline mode for multiply/accumulate Single-pipeline mode for other operations

M21 – M31 – Reserved for factory test (must be set to Logic 0)

Input Modes

The Am29C327 supports a total of eight input modes for loading data into the R and S registers.

The 32-bit bus modes allow the user to connect each input port ($R_0 - R_{31}$ and $S_0 - S_{31}$) to separate 32-bit buses. 64-bit operands can then be loaded by placing the MSBs and LSBs alternately on the appropriate ports. In the 64-bit bus modes, the two input ports are configured internally as a single 64-bit port. The Am29C327 may then be connected directly to a 64-bit bus, and 64-bit operands may be loaded in single operation. Either the 32-bit bus modes or the 64-bit bus modes may be used regardless of the precision of the operands being transferred — the choice of input modes will in practice be determined by the system into which the Am29C327 is to be integrated.

Single-cycle input modes allow two 64-bit operands to be loaded in a single clock cycle. This necessitates driving the input buses at twice the speed of the Am29C327. For systems when this is not practical, the double-cycle modes allow the loading of one 64-bit operand (or two 32-bit operands) per clock cycle.

Data may be loaded from the input buses to the R register and S register using one of the eight input modes:

1. 32-Bit Bus, Single-Cycle, LSWs First
2. 32-Bit Bus, Single-Cycle, MSWs First
3. 32-Bit Bus, Double-Cycle, LSWs First
4. 32-Bit Bus, Double-Cycle, MSWs First
5. 64-Bit Bus, Single-Cycle, R First
6. 64-Bit Bus, Single-Cycle, S First
7. 64-Bit Bus, Double-Cycle, R First
8. 64-Bit Bus, Double-Cycle, S First

The choice of the input modes is determined by mode register bits M14 – M16.

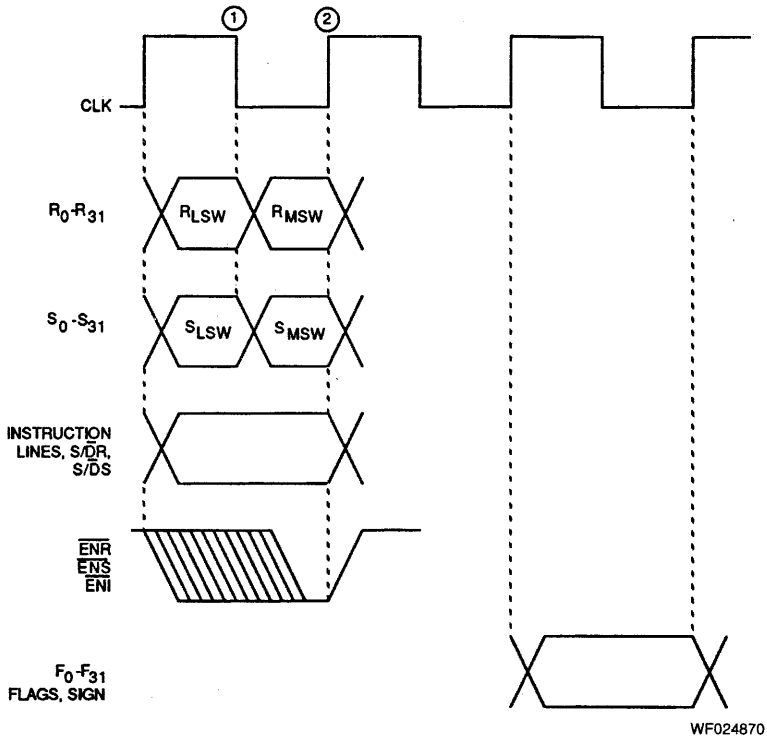
In order to permit the loading of new operands to be overlapped with the execution of a current operation, temporary registers are provided within the "operand router" block (shown in Figure 1). The operation of these temporary registers is transparent to the user. The conditions under which they are loaded depends on the input mode selected.

The eight input modes are described on the following pages.

32-Bit Bus, Single-Cycle, LSW First (M16 = 0, M15 = 0, M14 = 0)

In this mode, the two halves of the 64-bit R operand are placed on the R-input bus in successive half-cycles, with the S

operand similarly placed on the S-input port. After one complete cycle, the R and S registers contain the R and S operands, respectively.



**Timing of Operations with Input Mode 1
(32-Bit Bus, Single-Cycle, LSW First)***

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every HIGH-to-LOW clock transition.

At 1, the least-significant 32 bits of the R operand are loaded from the R-input port into the R-temp register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the S-temp register. Both words are loaded on the HIGH-to-LOW transition of the clock.

At 2, the most-significant 32 bits of the R operand are loaded from the R-input port into the most-significant half of the R

register, and the most-significant 32 bits of the S operand are loaded from the S-input port into the most-significant half of the S register.

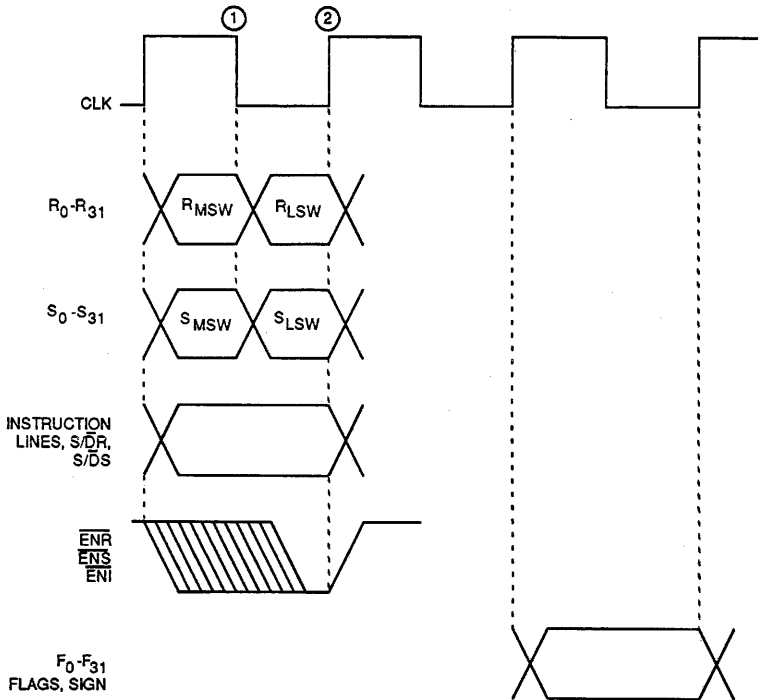
At the same time, at 2, the output of the R-temp register is loaded into the least-significant half of the R register, and the output of the S-temp register is loaded into the least-significant half of the S register.

If an input operand is single-precision, the 32-bit data is kept on the input bus for the full cycle.

32-Bit Bus, Single-Cycle, MSW First (M16 = 0, M15 = 0, M14 = 1)

In this mode, the two halves of the 64-bit R operand are placed on the R-input bus in successive half-cycles, with the S

operand similarly placed on the S-input port. After one complete cycle, the R and S registers contain the R and S operands, respectively.



WF024890

**Timing of Operations with Input Mode 2
(32-Bit Bus, Single-Cycle, MSW First)***

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every HIGH-to-LOW clock transition.

At 1, the most-significant 32 bits of the R operand are loaded from the R-input port into the R-temp register, and the most-significant 32 bits of the S operand are loaded from the S-input port into the S-temp register. Both words are loaded on the HIGH-to-LOW transition of the clock.

At 2, the least-significant 32 bits of the R operand are loaded from the R-input port into the least-significant half of the R

register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the least-significant half of the S register.

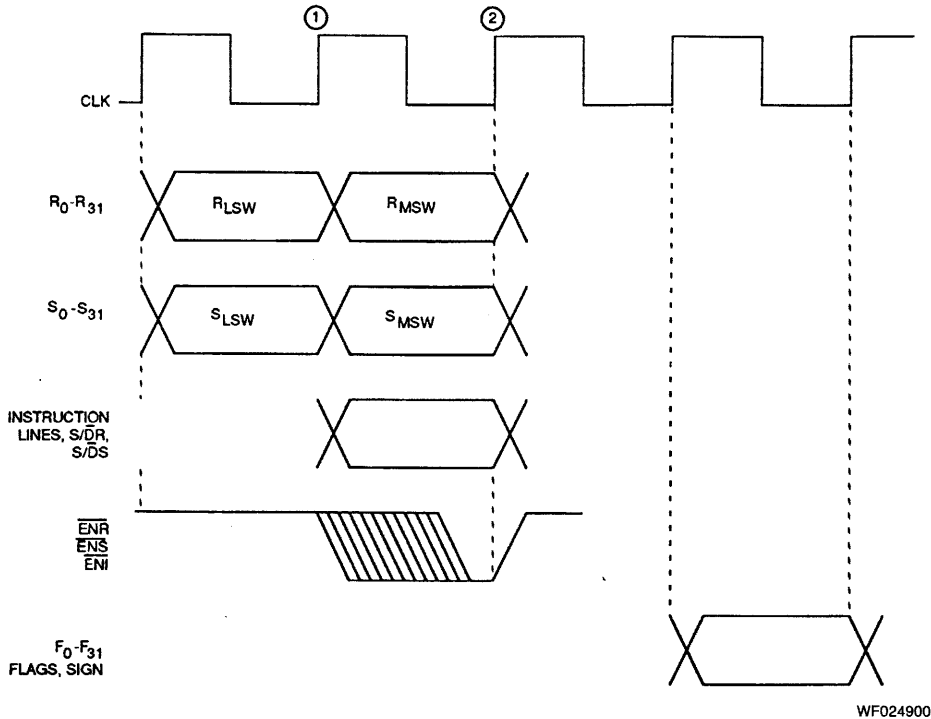
At the same time, at 2, the output of the R-temp register is loaded into the most-significant half of the R register, and the output of the S-temp register is loaded into the most-significant half of the S register.

If an input operand is single-precision, the 32-bit data is kept on the input bus for the full cycle.

32-Bit Bus, Double-Cycle, LSW First (M16 = 0, M15 = 1, M14 = 0)

In this mode, the two halves of the 64-bit R operand are placed on the R-input bus in successive cycles, with the S

operand similarly placed on the S-input port. After two cycles, the R and S registers contain the R and S operands, respectively.



**Timing of Operations with Input Mode 3
(32-Bit Bus, Double-Cycle, LSW First)***

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every LOW-to-HIGH clock transition.

At 1, the least-significant 32 bits of the R operand are loaded from the R-input port into the R-temp register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the S-temp register.

At 2, the most-significant 32 bits of the R operand are loaded from the R-input port into the most-significant half of the R

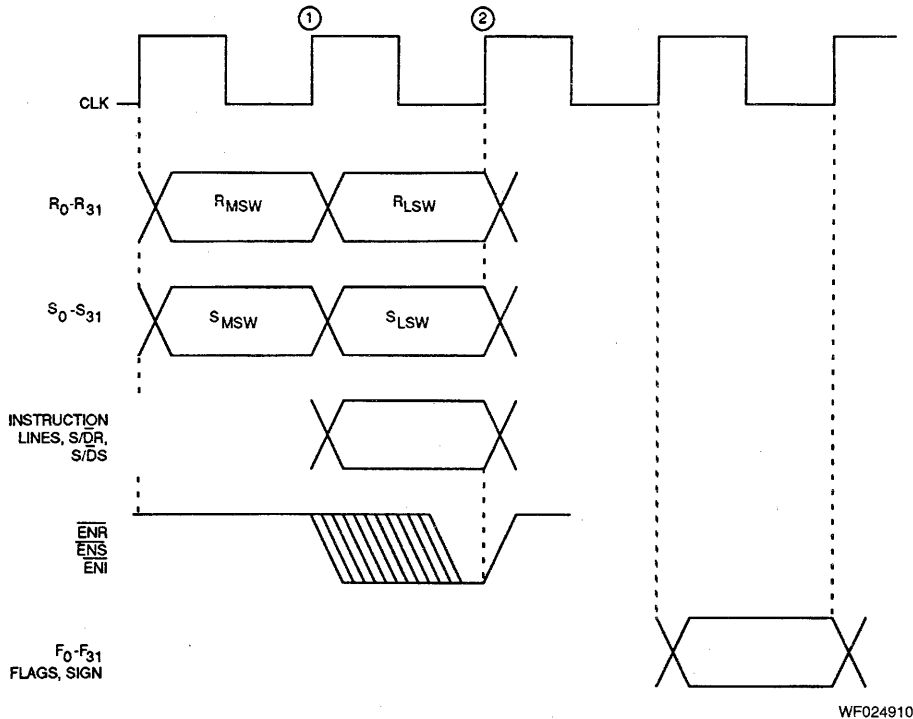
register, and the most-significant 32 bits of the S operand are loaded from the S-input port into the most-significant half of the S register.

At the same time, at 2, the output of the R-temp register is loaded into the least-significant half of the R register, and the output of the S-temp register is loaded into the least-significant half of the S register.

32-Bit Bus, Double-Cycle, MSW First (M16 = 0, M15 = 1, M14 = 1)

In this mode, the two halves of the 64-bit R operand are placed on the R-input bus in successive cycles, with the S

operand similarly placed on the S-input port. After two cycles, the R and S registers contain the R and S operands, respectively.



WF024910

**Timing of Operations with Input Mode 4
(32-Bit Bus, Double-Cycle, MSW First)***

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every LOW-to-HIGH clock transition.

At 1, the most-significant 32 bits of the R operand are loaded from the R-input port into the R-temp register, and the most-significant 32 bits of the S operand are loaded from the S-input port into the S-temp register.

At 2, the least-significant 32 bits of the R operand are loaded from the R-input port into the least-significant half of the R

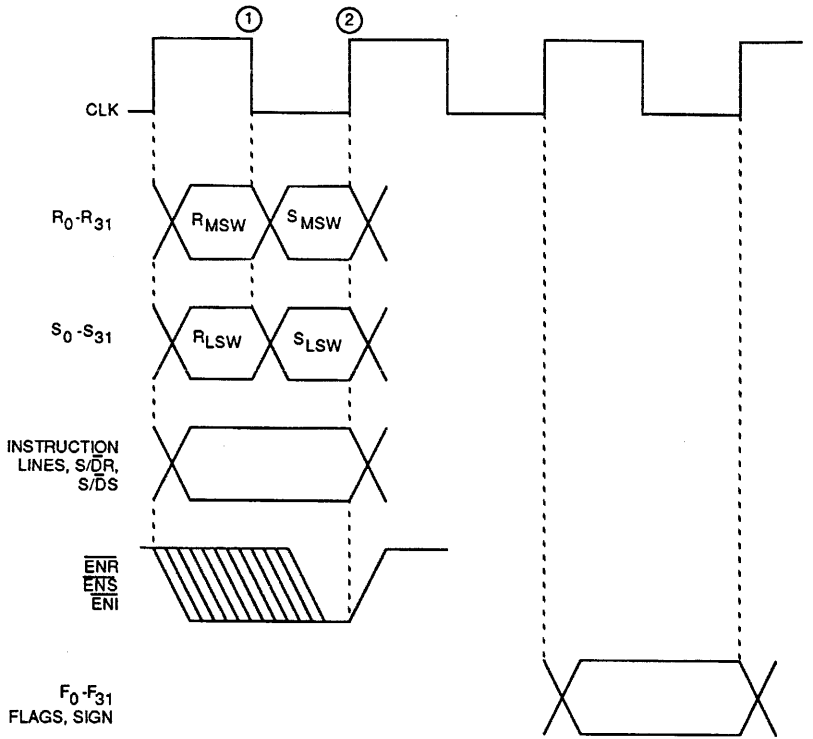
register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the least-significant half of the S register.

At the same time, at 2, the output of the R-temp register is loaded into the most-significant half of the R register, and the output of the S-temp register is loaded into the most-significant half of the S register.

64-Bit Bus, Single-Cycle, R First (M16 = 1, M15 = 0, M14 = 0)

In this mode, the MSW of the 64-bit R operand is placed on the R-input bus and the LSW of the S-input bus. Both

halfwords are loaded in the first half cycle. Similarly, the two halves of the S operand are loaded in the second half cycle. After one full cycle, the R and S registers contain the R and S operands, respectively.



**Timing of Operations with Input Mode 5
(64-Bit Bus, Single-Cycle, R First)***

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every HIGH-to-LOW clock transition.

At 1, the most-significant 32 bits of the R operand are loaded from the R-input port into the R-temp register, and the least-significant 32 bits of the R operand are loaded from the S-input port into the S-temp register.

At 2, the most-significant 32 bits of the S operand are loaded from the R-input port into the most-significant half of the S

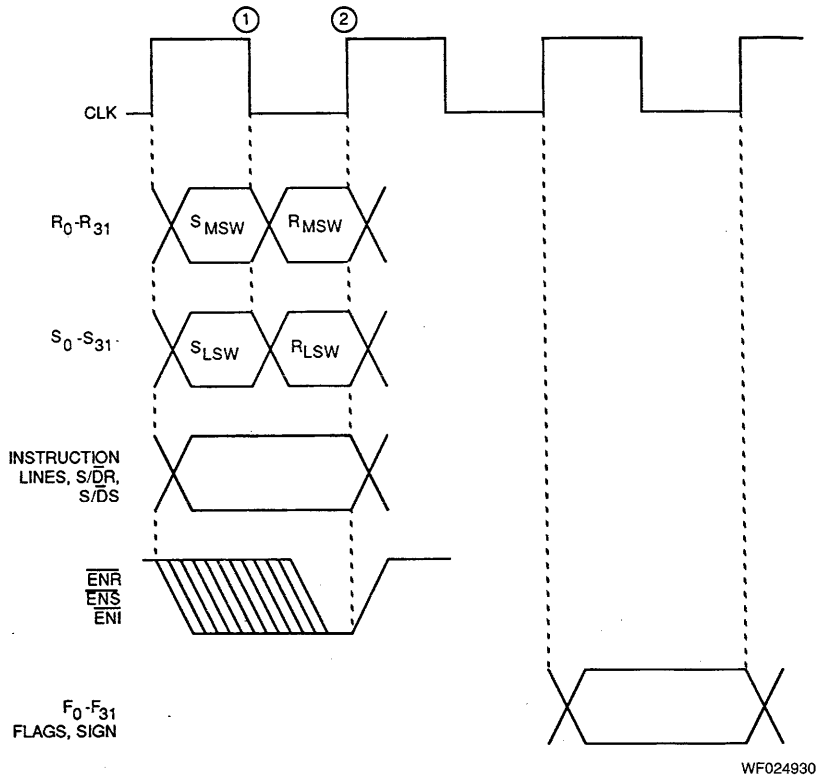
register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the least-significant half of the S register.

At the same time, at 2, the output of the R-temp register is loaded into the most-significant half of the R register, and the output of the S-temp register is loaded into the least-significant half of the R register.

64-Bit Bus, Single-Cycle, S First (M16 = 1, M15 = 0, M14 = 1)

In this mode, the MSW of the 64-bit S operand is placed on the R-input bus and the LSW on the S-input bus. Both halfwords

are loaded in the first half cycle. Similarly, the two halves of the R operand are loaded in the second half cycle. After one full cycle, the R and S registers contain the R and S operands, respectively.



Timing of Operations with Input Mode 6 (64-Bit Bus, Single-Cycle, S First)*

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every HIGH-to-LOW clock transition.

At 1, the most-significant 32 bits of the S operand are loaded from the R-input port into the R-temp register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the S-temp register.

At 2, the most-significant 32 bits of the R operand are loaded from the R-input port into the most-significant half of the R

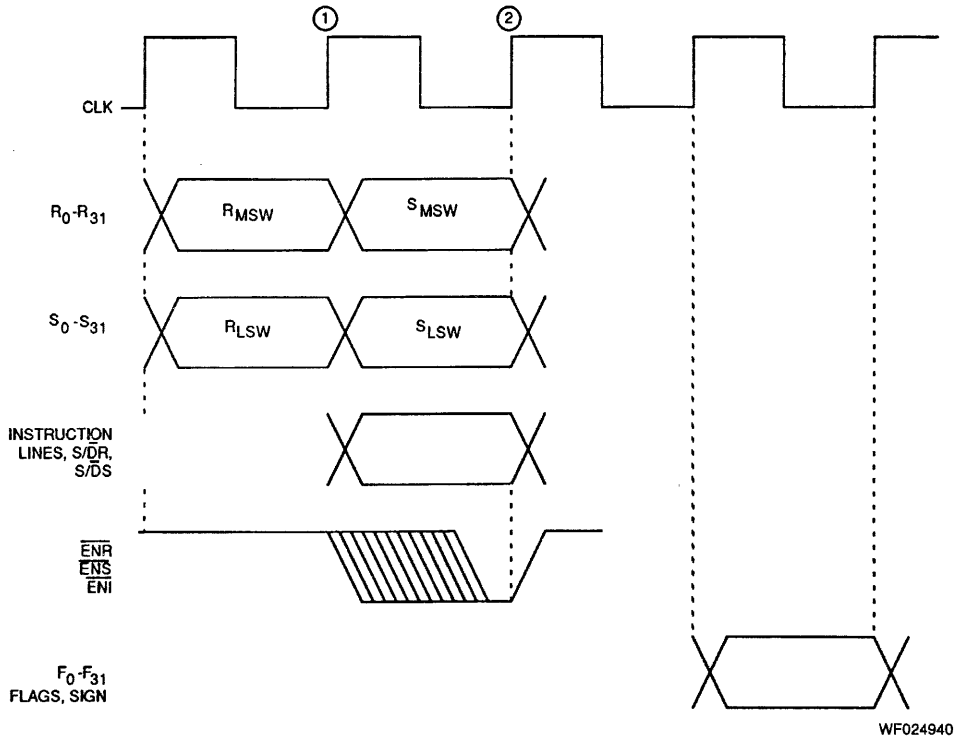
register, and the least-significant 32 bits of the R operand are loaded from the S-input port into the least-significant half of the R register.

At the same time, at 2, the output of the R-temp register is loaded into the most-significant half of the S register, and the output of the S-temp register is loaded into the least-significant half of the S register.

64-Bit Bus, Double-Cycle, R First (M16 = 1, M15 = 1, M14 = 0)

In this mode, the MSW of the 64-bit R operand is placed on the R-input bus and the LSW of the S-input bus. Both

halfwords are loaded in the first cycle. Similarly, the two halves of the S operand are loaded in the second cycle. After the two cycles, the R and S registers contain the R and S operands, respectively.



**Timing of Operations with Input Mode 7
(64-Bit Bus, Double-Cycle, R First)***

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every LOW-to-HIGH clock transition.

At 1, the most-significant 32 bits of the R operand are loaded from the R-input port into the R-temp register, and the least-significant 32 bits of the R operand are loaded from the S-input port into the S-temp register.

At 2, the most-significant 32 bits of the S operand are loaded from the R-input port into the most-significant half of the S

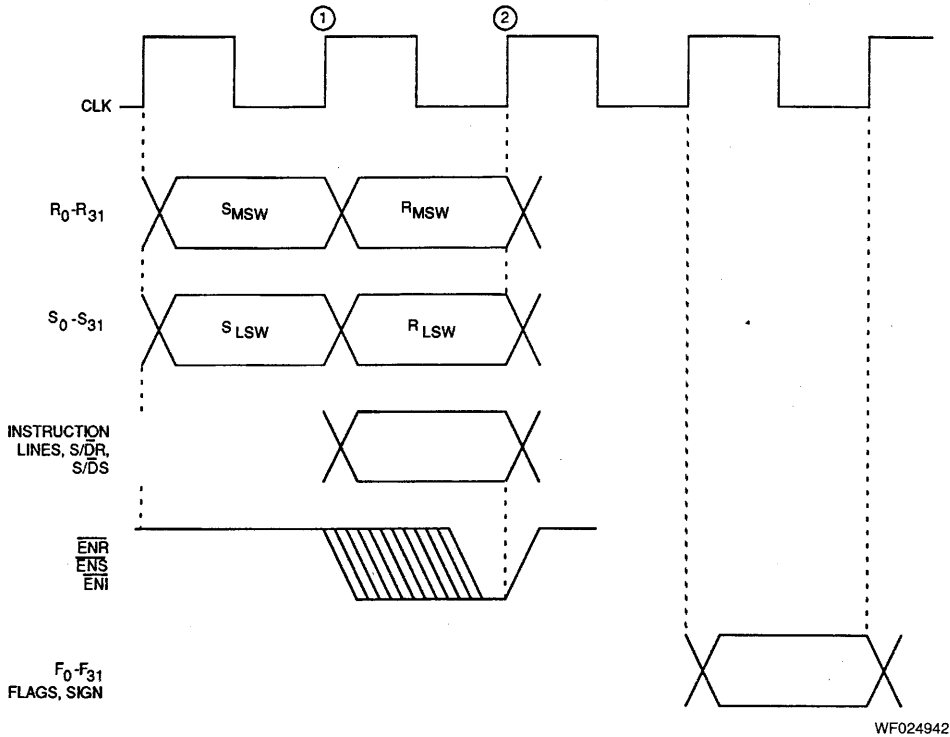
register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the least-significant half of the S register.

At the same time, at 2, the output of the R-temp register is loaded into the most-significant half of the R register, and the output of the S-temp register is loaded into the least-significant half of the R register.

64-Bit Bus, Double-Cycle, S First (M16 = 1, M15 = 1, M14 = 1)

In this mode, the MSW of the 64-bit S operand is placed on the R-input bus and the LSW of the S-input bus. Both halfwords

are loaded in the first cycle. Similarly, the two halves of the R operand are loaded in the second cycle. After the two cycles, the R and S registers contain the R and S operands, respectively.



WF024942

**Timing of Operations with Input Mode 8
(64-Bit Bus, Double-Cycle, S First)***

*Assumes flow-through operation, F register, and S register clocked.

In this mode, the temporary registers are clocked on every LOW-to-HIGH clock transition.

At 1, the most-significant 32 bits of the S operand are loaded from the R-input port into the R-temp register, and the least-significant 32 bits of the S operand are loaded from the S-input port into the S-temp register.

At 2, the most-significant 32 bits of the R operand are loaded from the R-input port into the most-significant half of the R

register, and the least-significant 32 bits of the R operand are loaded from the S-input port into the least-significant half of the R register.

At the same time, at 2, the output of the R-temp register is loaded into the most-significant half of the S register, and the output of the S-temp register is loaded into the least-significant half of the S register.

Pipelining of Operations

The floating-point ALU of the Am29C327 may be operated in one of three pipeline modes:

1. Flow-Through Mode
2. Single-Pipelined Mode
3. Double-Pipelined Mode

Flow-Through Mode

In this mode the floating-point ALU acts as a purely combinatorial device.

Single-Pipelined Mode

In this mode the floating-point ALU contains a single pipeline delay for all operations; throughput is roughly double that for unpipelined mode. Simplified diagrams for the ALU configuration for single-pipelined mode are shown in Figure 2.

Double-Pipelined Mode

In this mode, which applies only to the multiplication-accumulation operation, the ALU contains two pipeline delays; throughput is roughly triple that for the unpipelined multiplication-accumulation operation. Simplified block diagrams are shown in Figure 3.

Figures 4 and 5 provide timing diagrams for all operations except multiply-accumulate, illustrating flow-through mode and pipelined mode, respectively. Figures 6, 7, and 8 provide timing diagrams for multiply-accumulate, illustrating flow-through mode, single-pipelined mode, and double-pipelined mode, respectively.

The choice of pipelining mode affects only the floating-point ALU. Operations of other parts of the Am29C327, such as the input registers, the output register, the mode register, and the instruction register are not affected by the choice of pipelining mode. However, the instruction bits are pipelined as they pass through the ALU. This permits instructions to be interleaved in pipelined mode.

The desired pipeline mode or modes can be invoked by setting mode register bits M19 and M20 to the appropriate values.

When using the Am29C327 in either single-pipelined or double-pipelined mode, two conditions must be observed:

1. The "load mode register" instruction is not pipelined, nor are any of the mode register bits. When the mode register is loaded, any differences between the current mode and the previous mode take effect immediately. In single-pipelined mode, the user should separate the last valid ALU instruction and the "load mode register" instruction with one "NO-OP" instruction. In double-pipelined mode, the user should separate them with two "NO-OP" instructions. A NO-OP instruction is any instruction whose result is not stored in register F, or the register file.
2. A multiplication-accumulation instruction cannot be immediately followed by any other type of instruction. This problem can be avoided by inserting a "dummy" multiplication-accumulation instruction at the end of a multiplication-accumulation instruction. This "dummy" is any instruction whose results are not stored in register F or the register file.

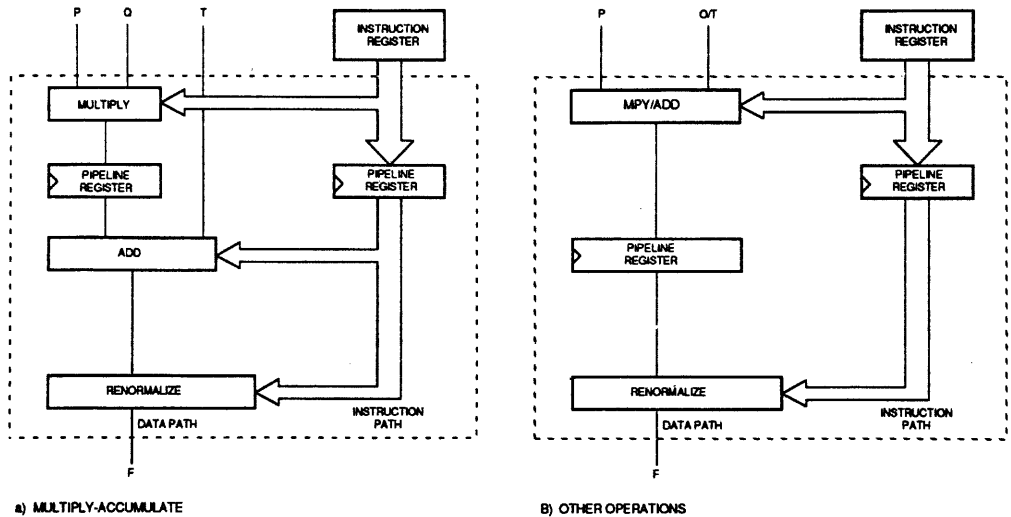
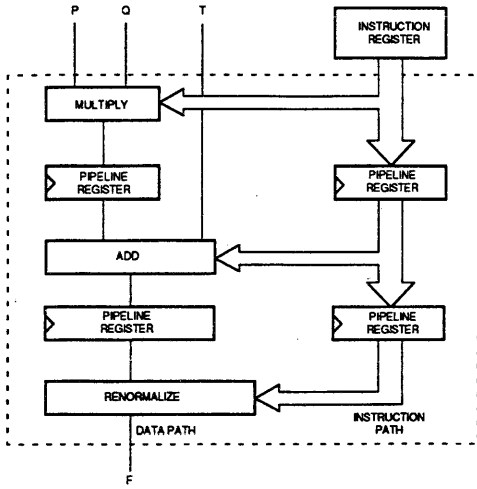
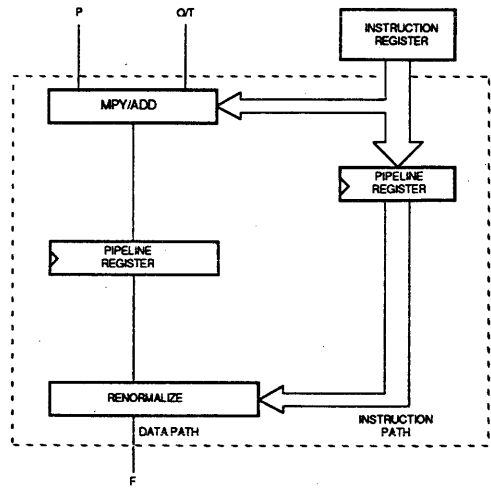


Figure 2. ALU Configuration for Single-Pipelined Mode

DF006260



a) MULTIPLY-ACCUMULATE



b) OTHER OPERATIONS

DF006270

Figure 3. ALU Configuration for Double-Pipelined Mode

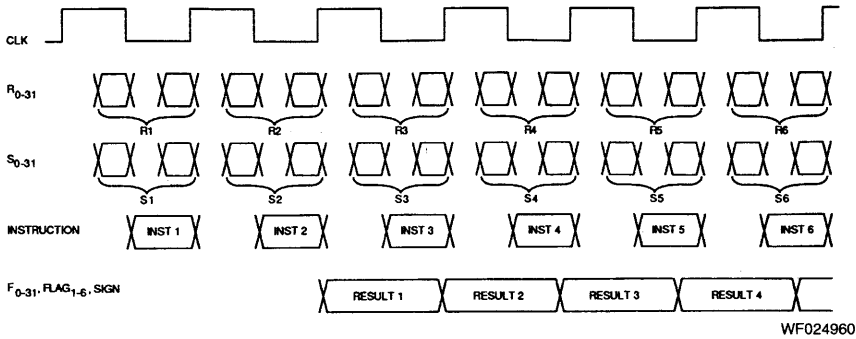


Figure 4. Timing for All Operations EXCEPT Multiply-Accumulate, Flow-Through Mode

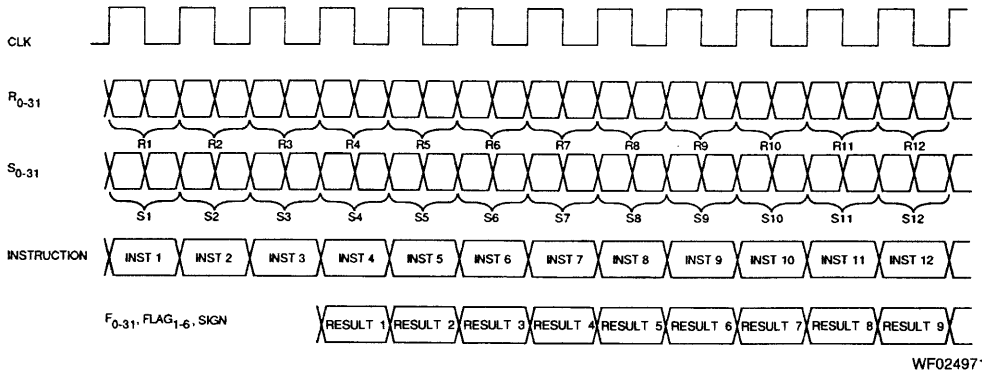


Figure 5. Timing for All Operations EXCEPT Multiply-Accumulate, Pipelined Mode

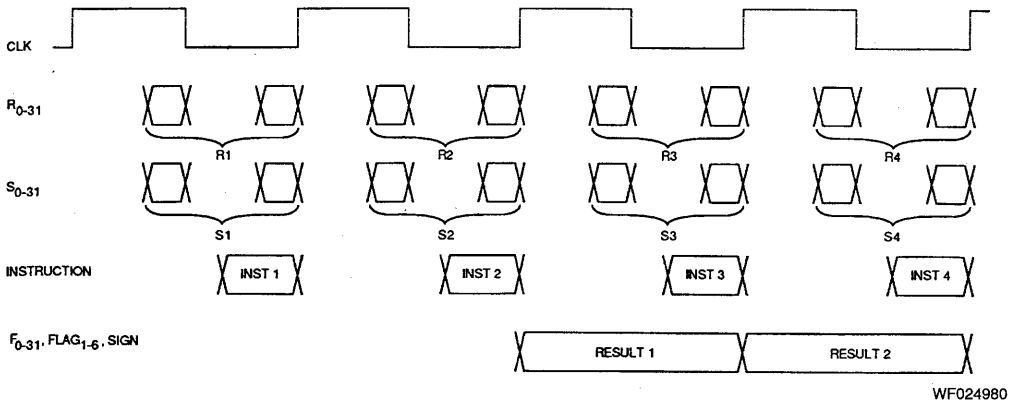


Figure 6. Timing for Multiply-Accumulate, Flow-Through Mode

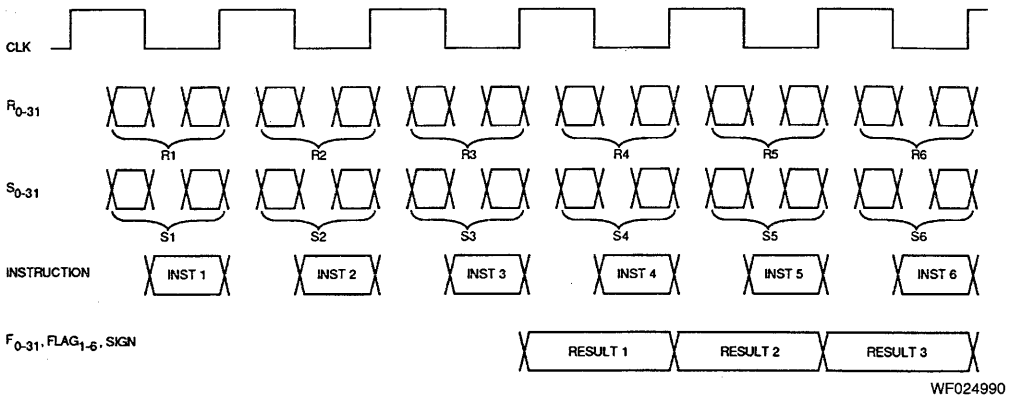


Figure 7. Timing for Multiply-Accumulate, Single-Pipelined Mode

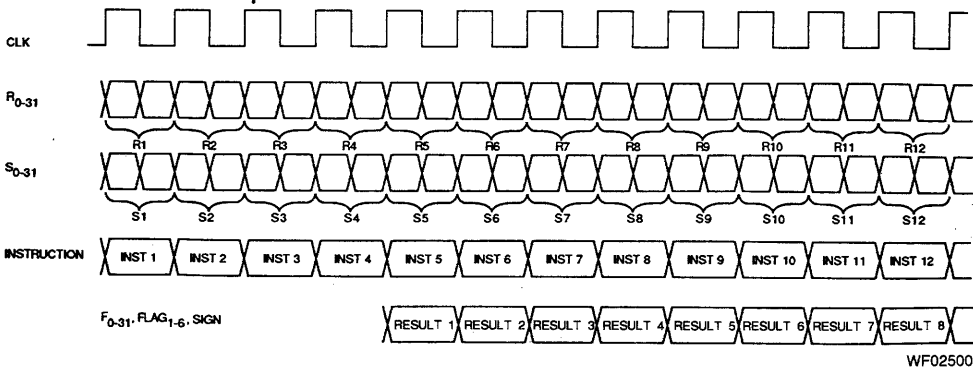


Figure 8. Timing for Multiply-Accumulate, Double-Pipelined Mode

Instruction Set

Instruction Register Format

The 14-bit instruction word I_0-I_{13} comprises sign-change controls, integer/floating-point select bit, and the opcode.

I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
SIGN (P)		SIGN (Q)		SIGN (T)		SIGN (F)		INT/FP	OPCODE				

The opcode field, I_4-I_0 , specifies the core operation to be performed by the ALU; instruction bit I_5 selects between floating-point and integer formats. The core operations and their corresponding opcodes are listed in Table 1.

TABLE 1. CORE OPERATIONS/OPCODES

I_5	I_4	I_3	I_2	I_1	I_0	Operation (Floating-Point)
0	0	0	0	0	0	P
0	0	0	0	0	1	P + T
0	0	0	0	1	0	P * Q
0	0	0	0	1	1	COMPARE P, T
0	0	0	1	0	0	MAX P, T
0	0	0	1	0	1	MIN P, T
0	0	0	1	1	0	CONVERT T TO INTEGER
0	0	0	1	1	1	SCALE T TO INTEGER BY Q
0	0	1	0	0	0	(P * Q) + T
0	0	1	0	0	1	ROUND T TO INTEGRAL VALUE
0	0	1	0	1	0	RECIPROCAL SEED OF P
0	0	1	0	1	1	CONVERT T TO ALTERNATE F.P. FORMAT
0	0	1	1	0	0	CONVERT T FROM ALTERNATE F.P. FORMAT
I_5	I_4	I_3	I_2	I_1	I_0	Operation (Integer)
1	0	0	0	0	0	P
1	0	0	0	0	1	P + T
1	0	0	0	1	0	P * Q
1	0	0	0	1	1	COMPARE P, T
1	0	0	1	0	0	MAX P, T
1	0	0	1	0	1	MIN P, T
1	0	0	1	1	0	CONVERT T TO FLOATING-POINT
1	0	0	1	1	1	SCALE T TO FLOATING-POINT BY Q
1	1	0	0	0	0	P OR T
1	1	0	0	0	1	P AND T
1	1	0	0	1	0	P XOR T
1	1	0	0	1	1	SHIFT P LOGICAL Q PLACES
1	1	0	1	0	0	SHIFT P ARITHMETIC Q PLACES
1	1	0	1	0	1	FUNNEL SHIFT PT LOGICAL Q PLACES

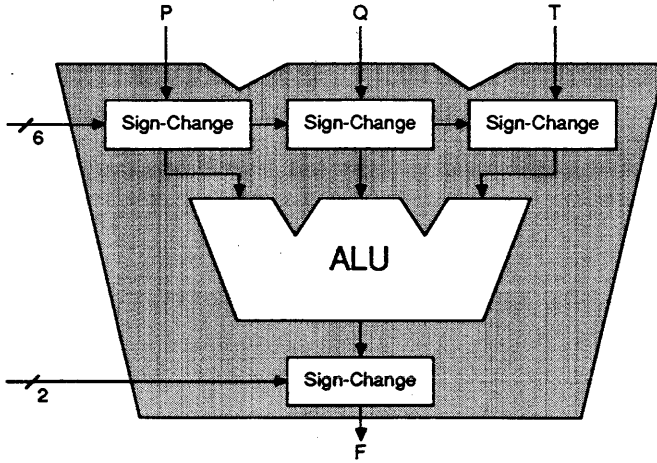
Core operations MOVE P and LOAD MODE REGISTER can both be performed in either floating-point or integer format:

I_5	I_4	I_3	I_2	I_1	I_0	Operation
X	1	1	0	0	0	MOVE P
X	1	1	1	1	1	LOAD MODE REGISTER

Sign-Change Selects

Each ALU input and output operand has associated hardware that can be used to modify operand signs (see Figure 9). These sign-change blocks, when applied to core operations, greatly increase the number of available operations. A core operation of $P + T$, for example, can be used to perform operations such as $P - T$, $ABS(P + T)$, $ABS(P) + ABS(T)$, and others, simply by modifying the signs of the input and output operands.

Using the sign-change blocks, the sign of an input operand may be left unchanged, inverted, set to zero, or set to one; the sign of the output operand may be left unchanged, set to zero, set to one, set to the sign of the P input operand, or set to the sign of the T input operand. Select decodes for the P, Q, T, and F operand sign-change blocks are shown in Table 2-1, 2-2, 2-3, and 2-4, respectively.



BD007600

Figure 9. ALU Sign-Change Blocks

TABLE 2-1. SELECT DECODE FOR P OPERAND SIGN-CHANGE BLOCK

I ₁₃	I ₁₂	Sign (P')
0	0	SIGN (P)
0	1	SIGN (P)
1	0	0
1	1	1

TABLE 2-2. SELECT DECODE FOR Q OPERAND SIGN-CHANGE BLOCK

I ₁₁	I ₁₀	Sign (Q')
0	0	SIGN (Q)
0	1	SIGN (Q)
1	0	0
1	1	1

TABLE 2-3. SELECT DECODE FOR T OPERAND SIGN-CHANGE BLOCK

I ₉	I ₈	Sign (T')
0	0	SIGN T
0	1	SIGN T
1	0	0
1	1	1

TABLE 2-4. SELECT DECODE FOR F OPERAND SIGN-CHANGE BLOCK

Core Operation	I ₁₁	I ₁₀	I ₇	I ₆	Sign (F)
P,	0	x	0	0	SIGN (F')
Max P, T	0	x	0	1	SIGN (F')
or	0	x	1	0	0
Min P, T	0	x	1	1	1
	1	0	x	x	SIGN (P)
	1	1	x	x	SIGN (T)
Other	x	x	0	0	SIGN (F')
	x	x	0	1	SIGN (F')
	x	x	1	0	0
	x	x	-1	1	1

Operand Multiplexer Selects

Instruction fields PSEL₀–PSEL₃, QSEL₀–QSEL₃, and TSEL₀–TSEL₃ specify the select codes for the P, Q, and T

operand multiplexers, respectively; the codes are summarized in Table 3.

TABLE 3. OPERAND MULTIPLEXER SELECT CODES

PSEL ₃ QSEL ₃ TSEL ₃	PSEL ₂ QSEL ₂ TSEL ₂	PSEL ₁ QSEL ₁ TSEL ₁	PSEL ₀ QSEL ₀ TSEL ₀	P Q T
0	0	0	0	R
0	0	0	1	S
0	0	1	0	O
0	0	1	1	0.5 (Floating Point) -1 (Integer)
0	1	0	0	1
0	1	0	1	2
0	1	1	0	3
0	1	1	1	Pi (Floating Point) Max Neg. Two's-Comp. Value (Integer)
1	0	0	0	Register File Location 0 (RF0)
1	0	0	1	Register File Location 1 (RF1)
1	0	1	0	Register File Location 2 (RF2)
1	0	1	1	Register File Location 3 (RF3)
1	1	0	0	Register File Location 4 (RF4)
1	1	0	1	Register File Location 5 (RF5)
1	1	1	0	Register File Location 6 (RF6)
1	1	1	1	Register File Location 7 (RF7)

Operand Precisions

The Am29C327 supports mixed-precision operations, so that it is possible, for example, for an operation to have single-precision inputs and a double-precision output, or one single- and one double-precision input, or any other combination.

Precision of the operands in registers R and S is specified by signals S/DR and S/DS. A logic HIGH indicates a single-precision operand or operands; a LOW, double precision.

Precision of an operation result is specified by signal S/DF. A logic HIGH indicates a single-precision operand; a logic LOW, double-precision.

Operands stored in the register file are each accompanied by a bit indicating that operand's precision; this precision informa-

tion is automatically supplied to the ALU when a register file location is used as an input operand to an operation.

Processor Operations

Table 4 illustrates a number of possible ALU instructions comprising the opcode, integer/floating-point select, and sign-change fields. Note that the remaining instruction bits — P, Q, and T operand multiplexer selects; the rounding modes; and the output operand precision — can be specified independently.

The user may create instructions using instruction words other than those listed in Table 4. For some core operations, sign-change control settings are completely arbitrary; for others, only the sign-change field values shown in Table 4 are valid. Table 5 summarizes permissible sign-change field values for each core operation.

TABLE 4. INSTRUCTION WORDS

Operation	Sign				I/ \bar{F}	Opcode
	P	Q	T	F		
FP P	00	00	xx	00	0	00000
FP -P	00	00	xx	01	0	00000
FP ABS (P)	00	00	xx	10	0	00000
FP Sign (T)*ABS (P)	00	11	xx	xx	0	00000
FP P + T	00	xx	00	00	0	00001
FP P - T	00	xx	01	00	0	00001
FP T - P	01	xx	00	00	0	00001
FP -P - T	01	xx	01	00	0	00001
FP ABS (P + T)	00	xx	00	10	0	00001
FP ABS (P - T)	00	xx	01	10	0	00001
FP ABS (P) + ABS (T)	10	xx	10	00	0	00001
FP ABS (P) - ABS (T)	10	xx	11	00	0	00001
FP ABS (ABS (P) - ABS (T))	10	xx	11	10	0	00001
FP P * Q	00	00	xx	00	0	00010
FP (-P) * Q	01	00	xx	00	0	00010
FP ABS (P * Q)	00	00	xx	10	0	00010
FP Compare P, T	00	xx	01	00	0	00011
FP Max P, T	00	00	01	00	0	00100
FP Max ABS (P), ABS (T)	10	00	11	00	0	00100
FP Min P, T	01	00	00	00	0	00101
FP Min ABS (P), ABS (T)	11	00	10	00	0	00101
FP Limit P to Magnitude T	11	10	10	xx	0	00101
FP Convert T to Integer	xx	xx	00	00	0	00110
FP Scale T to Integer by Q	xx	00	00	00	0	00111
FP T + P*Q	00	00	00	00	0	01000
FP T - P*Q	01	00	00	00	0	01000
FP -T + P*Q	00	00	01	00	0	01000
FP -T - P*Q	01	00	01	00	0	01000
FP ABS (T) + ABS (P*Q)	10	10	10	00	0	01000
FP ABS (T) - ABS (P*Q)	11	10	10	00	0	01000
FP ABS (P*Q) - ABS (T)	10	10	11	00	0	01000
FP Round T to Integral Value	xx	xx	00	00	0	01001
FP Reciprocal Seed (P)	00	xx	xx	00	0	01010
FP Convert T to Alternate Floating-point Format	xx	xx	00	00	0	01011
FP Convert T from Alternate Floating-point Format	xx	xx	00	00	0	01100

TABLE 4. INSTRUCTION WORDS (Cont'd.)

Operation	Sign				I/F	Opcode
	P	Q	T	F		
Int P	00	00	00	00	1	00000
Int -P	00	00	00	01	1	00000
Int ABS (P)	00	00	00	10	1	00000
Int sign (T)*ABS (P)	00	11	00	xx	1	00000
Int P + T	00	xx	00	00	1	00001
Int P - T	00	xx	01	00	1	00001
Int T - P	01	xx	00	00	1	00001
Int ABS (P + T)	00	xx	00	10	1	00001
Int ABS (P - T)	00	xx	01	10	1	00001
Int P * Q	00	00	xx	00	1	00010
Int Compare P, T	00	xx	01	00	1	00011
Int Max P, T	00	00	01	00	1	00100
Int Min P, T	01	00	00	00	1	00101
Int Convert T to Float	xx	xx	00	00	1	00110
Int Scale T to Float by Q	xx	00	00	00	1	00111
Int P OR T	xx	xx	xx	xx	1	10000
Int P AND T	xx	xx	xx	xx	1	10001
Int P XOR T	xx	xx	xx	xx	1	10010
Int NOT T (see Note 1)	xx	xx	xx	xx	1	10010
Int Shift P Logical Q Places	00	00	xx	00	1	10011
Int Shift P Arithmetic Q Places	00	00	xx	00	1	10100
Int Funnel Shift PT Q Places	00	00	00	00	1	10101
Int Move P	xx	xx	xx	xx	x	11000
Int Load Mode Register	xx	xx	xx	xx	x	11111

Notes: 1. NOT T is performed by XORing T with a word containing all 1's (integer - 1). When invoking NOT T the user must set PSEL₃-PSEL₀ to 0011₂, thus selecting integer constant - 1.

TABLE 5. ALLOWABLE SIGN-CHANGE/CORE-OPERATION COMBINATIONS

I 11111 5 43210	Core Operation	Sign-Change Fields			
		Sign (P)	Sign (Q)	Sign (T)	Sign (F)
0 00000	FP P	V	V	x	V
0 00001	FP P + T	V	x	V	V
0 00010	FP P*Q	V	V	x	V
0 00011	FP Compare P, T	F	x	F	F
0 00100	FP Max P, T	F	F	F	F
0 00101	FP Min P, T	F	F	F	F
0 00110	FP Cvt T to Int	x	x	F	F
0 00111	FP Scale T to Int	x	F	F	F
0 01000	FP P*Q + T	V	V	V	V
0 01001	FP Round T	x	x	F	F
0 01010	FP Recip Seed P	F	x	x	F
0 01011	FP Cvt T to Alt Fmt	x	x	F	F
0 01100	FP Cvt T fm Alt Fmt	x	x	F	F
1 00000	Int P	F	F	F	F
1 00001	Int P + T	F	x	F	F
1 00010	Int P*Q	F	F	x	F
1 00011	Int Compare P, T	F	x	F	F
1 00100	Int Max P, T	F	F	F	F
1 00101	Int Min P, T	F	F	F	F
1 00110	Int Cvt T to f.p.	x	x	F	F
1 00111	Int Scale T to f.p.	x	F	F	F
1 10000	Int P OR T	x	x	x	x
1 10001	Int P AND T	x	x	x	x
1 10010	Int P XOR T	x	x	x	x
1 10011	Int Shift P Logical	F	F	x	F
1 10100	Int Shift P Arith	F	F	x	F
1 10101	Int Funnel Shift PT	F	F	F	F
x 11000	Move P	x	x	x	x
x 11111	Load Mode Reg	x	x	x	x

Key: V = Variable; user can specify arbitrary sign change.

F = Fixed; user is restricted to sign change combinations shown in Table 4.

x = Don't care; this field does not affect the operation or its result.

Descriptions of Operations

P (Floating-Point or Integer): The operand on port P is passed through the ALU to port F. This operation may be used to change the precision of an operand, negate an operand, extract the absolute value of an operand, or transfer the sign of operand T to operand P.

P + T (Floating-Point or Integer): The addition operation (P + T) adds the operands on ports P and T, and places the result on port F.

P*Q (Floating-Point or Integer): The multiplication operation (P*Q) multiplies the operands on ports P and Q, and places the result on port F.

COMPARE P, T (Floating-Point or Integer): This operation compares the operands on ports P and T, and places (P - T) on port F. One of four comparison flags (=, >, <, #) is set according to the result of the comparison. Note that the unordered flag (#) can be set only when the format selected is IEEE or DEC.

MAX P, T (Floating-Point or Integer): This operation selects the most positive of the two operands on ports P and T, and places the result on port F.

MIN P, T (Floating-Point or Integer): This operation selects the most negative of the two operands on ports P and T, and places the result on port F.

LIMIT P TO MAGNITUDE T (Floating-Point): This operation imposes a clipping or saturation level on operand P by

comparing the magnitudes of the operands on ports P and T. If operand P has the smaller magnitude, it is placed on port F; if operand T has the smaller magnitude, it is placed on port F, but with its sign modified to agree with that of operand P. This operation is equivalent to operation SIGN(P) * MIN(ABS(P), ABS(T)).

CONVERT T TO INTEGER (Floating-Point): The floating-point-to-integer conversion operation takes a floating-point operand on port T and places the equivalent two's-complement integer value on port F.

CONVERT T TO FLOATING-POINT (Integer): The integer-to-floating-point conversion operation takes a two's-complement integer operand on port T and places the equivalent floating-point value on port F.

SCALE T TO INTEGER BY Q (Floating-Point): This operation converts the floating-point operand T to integer format using the floating-point operand Q as a scale factor. The true exponent of Q is added to the true exponent of T before the new value T is converted to integer format. The operation therefore permits T to be multiplied by any power of two when the source format is IEEE or DEC, and by any power of 16 when the source format is IBM.

SCALE T TO FLOATING-POINT BY Q (Integer): This operation converts the integer operand T to floating-point format using the operand Q as a scale factor, where Q is a floating-point operand in the destination format. The true exponent of Q is added to the true exponent of T after T has been converted from integer to floating-point. The operation

therefore permits T to be scaled by any multiple of two when the destination format is IEEE or DEC, and by any multiple of 16 when the destination format is IBM.

(P*Q) + T (Floating-Point): This operation multiplies the operands on port P and Q, adds the product to the operand on port T, and places the result on port F.

ROUND T TO INTEGRAL VALUE (Floating-Point): This operation rounds a floating-point operand to an integer-valued floating-point operand of the same format. A value of 3.5, for example, would be rounded to either 3.0 or 4.0, the choice depending on the rounding mode.

RECIPROCAL SEED OF P (Floating-Point): The reciprocal seed of the floating-point operand on port P is placed on port F; the result obtained is a crude estimate of the input operand's reciprocal. This operation can be used as the initial step in performing Newton-Raphson division. A single-precision result is obtained after five iterations, and a double-precision result after six iterations. Alternately, an external seed look-up table can be used for faster convergence. The result obtained through iteration is approximate.

CONVERT T TO ALTERNATE FLOATING-POINT FORMAT (Floating-Point): This operation converts operand T from the primary floating-point format to the alternate floating-point format, thus allowing conversions among the IEEE, DEC, and IBM floating-point formats.

CONVERT T FROM ALTERNATE FLOATING-POINT FORMAT (Floating-Point): This operation converts operand T from the alternate floating-point format to the primary floating-point format, in a manner similar to that of CONVERT T TO ALTERNATE FLOATING-POINT FORMAT above.

P OR T, P AND T, P XOR T, NOT T (Integer): The logical operations (OR, AND, EXCLUSIVE OR) are performed on the operands on ports P and T, and the result is placed on port F. NOT T is performed by XORing T with a word containing all ones (integer -1). When invoking NOT T, instruction bits PSEL₃ - PSEL₀ must be set to 0011, thus selecting integer constant -1.

SHIFT P LOGICAL Q PLACES (Integer): This operation logically shifts operand P by Q places. If the shift is Q places to the right, Q zeros are filled from the left. If the shift is Q places to the left, Q zeros are filled from the right.

SHIFT P ARITHMETIC Q PLACES (Integer): This operation arithmetically shifts operand P by Q places. With a right shift, the result is sign extended Q places. With a left shift, Q zeros are filled from the right.

FUNNEL SHIFT PT LOGICAL Q PLACES (Integer): The operands on ports P and T are concatenated to form a double-width operand PT, which is then shifted to the right or left by Q places; the 32- or 64-bit result is placed on port F.

MOVE P (Floating-Point or Integer): The operand on port P is moved to port F. The operand is left unchanged, and only the sign flag is set.

Operation Flags

For each operation, the ALU produces thirteen flags that indicate operation status. Of the flags produced, a maximum of seven are relevant to any given operation. The relevant flags are placed in the status register, and the other flags are discarded.

The ALU flags are:

C — CARRY: Carry-out bit produced by integer addition, subtraction, or comparison.

I — INVALID OPERATION: Input operands are unsuitable for the operation specified (e.g., $\infty * 0$).

R — RESERVED OPERAND: Reserved operand detected/generated.

S — SIGN: Result sign.

U — UNDERFLOW: Result underflowed the destination format.

V — OVERFLOW: Result overflowed the destination format.

W — WINNER: Indicates which of the two operands selected when performing Max/Min operations.

X — INEXACT RESULT: Result had to be rounded to fit the destination format.

Z — ZERO: Zero result.

>, =, <, # — GREATER THAN, EQUAL, LESS THAN, UNORDERED: Used to report the result of a comparison operation.

Table 6 lists the flags reported for each operation.

TABLE 6. ORGANIZATION OF FLAGS

Operations		Opcode I ₄ -I ₀	Flag Register							
			MSB							LSB
			7	6	5	4	3	2	1	
IEEE	Non-arithmetic single-operand	00000	S	Z	X	U	V	R	I	
IEEE	Operations using add	00001	S	Z	X	U	V	R	I	
IEEE	Operations using multiply	00010	S	Z	X	U	V	R	I	
IEEE	Compare	00011	S	=	>	<	#	R	I	
IEEE	Maximum, minimum, limit	0010x	S	Z		W		R	I	
IEEE	Convert/scale to integer	0011x	S	Z	X		V	R	I	
IEEE	Multiply/accumulate	01000	S	Z		U	V	R	I	
IEEE	Round to integral value	01001	S	Z	X		V	R	I	
IEEE	Reciprocal seed	01010	S	Z		U	V	R	I	
IEEE	Convert to alt. f.p. format	01011	S	Z	X	U	V	R	I	
IEEE	Convert from alt. f.p. format	01100	S	Z	X	U	V	R	I	
DEC D	Non-arithmetic single-operand	00000	S	Z	X		V	R		
DEC D	Operations using add	00001	S	Z	X	U	V	R		
DEC D	Operations using multiply	00010	S	Z	X	U	V	R		
DEC D	Compare	00011	S	=	>	<	#	R		
DEC D	Maximum, minimum, limit	0010x	S	Z		W		R		
DEC D	Convert/scale to integer	0011x	S	Z	X		V	R	I	
DEC D	Multiply/accumulate	01000	S	Z		U	V	R		
DEC D	Round to integral value	01001	S	Z	X		V	R		
DEC D	Reciprocal seed	01010	S	Z		U	V	R	I	
DEC D	Convert to alt. f.p. format	01011	S	Z	X	U	V	R	I	
DEC D	Convert from alt. f.p. format	01100	S	Z	X	U	V	R	I	
DEC G	Non-arithmetic single-operand	00000	S	Z	X	U	V	R		
DEC G	Operations using add	00001	S	Z	X	U	V	R		
DEC G	Operations using multiply	00010	S	Z	X	U	V	R		
DEC G	Compare	00011	S	=	>	<	#	R		
DEC G	Maximum, minimum, limit	0010x	S	Z		W		R		
DEC G	Convert/scale to integer	0011x	S	Z	X		V	R	I	
DEC G	Multiply/accumulate	01000	S	Z		U	V	R		
DEC G	Round to integral value	01001	S	Z	X		V	R		
DEC G	Reciprocal seed	01010	S	Z		U	V	R	I	
DEC G	Convert to alt. f.p. format	01011	S	Z	X	U	V	R	I	
DEC G	Convert from alt. f.p. format	01100	S	Z	X	U	V	R	I	
IBM	Non-arithmetic single-operand	00000	S	Z	X		V			
IBM	Operations using add	00001	S	Z	X	U	V			
IBM	Operations using multiply	00010	S	Z	X	U	V			
IBM	Compare	00011	S	=	>	<	#			
IBM	Maximum, minimum, limit	0010x	S	Z		W				
IBM	Convert/scale to integer	0011x	S	Z	X		V			
IBM	Multiply/accumulate	01000	S	Z		U	V			
IBM	Round to integral value	01001	S	Z	X		V			
IBM	Reciprocal seed	01010	S	Z			V		I	
IBM	Convert to alt. f.p. format	01011	S	Z	X	U	V	R		
IBM	Convert from alt. f.p. format	01100	S	Z	X	U	V	R	I	
Integer	Non-arithmetic single-operand	00000	S	Z			V			
Integer	Sign transfer	00000	S	Z			V			
Integer	Operations using add	00001	S	Z			V		C	
Integer	Operations using multiply	00010	S	Z			V			
Integer	Compare operations	00011	S	=	>	<	V		C	
Integer	Maximum, minimum, limit	0010x	S	Z		W				
Integer	Convert to float	00110	S	Z	X					
Integer	Scale to float	00111	S	Z	X	U	V	R		
Integer	Logical operations	100xx	S	Z						
Integer	Arithmetic shift	10100	S	Z			V			
Integer	Funnel shift	10101	S	Z						
	Move operand	11000	S							
	Load mode register	11111								

Note: Unused flags assume the LOW state.

Master/Slave Operation

Two Am29C327 processors can be tied together in master/slave configuration, with the slave checking the results produced by the master. All input and output signals of the slave, with the exception of $\overline{\text{SLAVE}}$ and $\overline{\text{MSERR}}$, are tied to the corresponding signals of the master. The master is selected by asserting signal $\overline{\text{SLAVE LOW}}$; the slave, by asserting signal $\overline{\text{SLAVE HIGH}}$.

The slave processor, by comparing its outputs to the outputs of the master processor, performs a comprehensive check of the operation of the master processor. In addition, the slave processor may detect open circuits and other faults in the electrical path between the master processor and the system. Note that the master processor still performs the comparison between its outputs and its own internally generated results, and is therefore able to detect faults in its output drivers.

APPENDICES

APPENDIX A — DATA FORMATS

The following data formats are supported: 32-bit integer, 64-bit integer, IEEE single-precision, IEEE double-precision, DEC F, DEC D, DEC G, IBM single-precision, and IBM double-precision.

The primary and alternate floating-point formats are selected by mode register bits M0 to M3. The user may select between floating-point operations and integer operations by means of instruction bit I5.

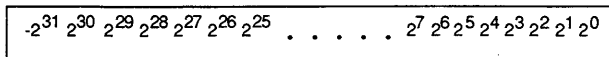
The nine supported formats are described below:

Integer Formats

32-Bit Integer

The 32-bit integer word is arranged as follows:

Bit 31 30 29 28 27 26 25 7 6 5 4 3 2 1 0



TB001030

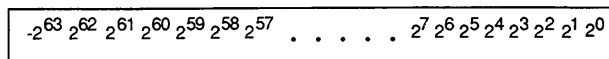
The 32-bit word is interpreted as a two's-complement integer. For integer multiplications, the user has the option of interpreting integers as unsigned. An unsigned single-precision integer

has a format similar to that of the two's-complement integer, but with an MSB weight of 2^{31} .

64-Bit Integer

The 64-bit integer word is arranged as follows:

Bit 63 62 61 60 59 58 57 7 6 5 4 3 2 1 0



TB001040

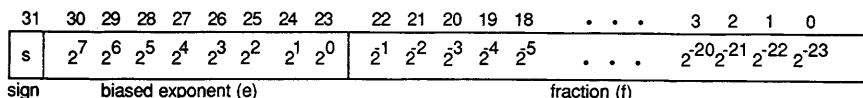
The 64-bit word is interpreted as a two's-complement integer. For integer multiplications, the user has the option of interpreting integers as unsigned. An unsigned double-precision integer

has a format similar to that of the two's-complement integer, but with an MSB weight of 2^{63} .

IEEE Formats

IEEE Single-Precision

The IEEE single-precision word is 32 bits wide and is arranged in the format as follows:



TB001050

The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 23-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers. Zero may have either sign.

The biased exponent is an 8-bit unsigned integer representing a multiplicative factor of some power of two. The bias value is 127. If, for example, the multiplicative value for a floating-point

number is to be 2^a , the value of the biased exponent is $a + 127$, where "a" is the true exponent.

The fraction is a 23-bit unsigned fractional field containing the 23 least-significant bits of the floating-point number's 24-bit mantissa. The weight of the fraction's most-significant bit is 2^{-1} . The weight of the least-significant bit is 2^{-23} .

An IEEE floating-point number is evaluated or interpreted as follows:

- | | | |
|-----------------------------------|----------------------------------|---------------------|
| If $e = 255$ and $f \neq 0$ | value = NaN | Not-a-Number |
| If $e = 255$ and $f = 0$ | value = $(-1)^s \infty$ | Infinity |
| If $0 < e < 255$ | value = $(-1)^s 2^{e-127} (1.f)$ | Normalized number |
| If $e = 0$ and $f \neq 0$ | value = $(-1)^s 2^{-126} (0.f)$ | Denormalized number |
| If $e = 0$ and $f = 0$ | value = $(-1)^s 0$ | Zero |

Infinity: Infinity can have either a positive or negative sign. The interpretation of infinities is determined by the Affine/Projective select input AFF/PROJ.

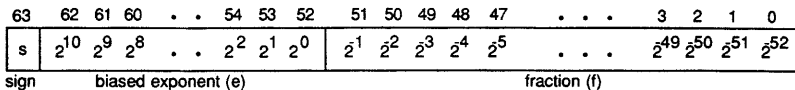
NaN: A NaN is interpreted as a signal or symbol. NaNs are used to indicate invalid operations, and as a means of passing process status through a series of calculations. They arise in

two ways: either generated by the Am29C327 to indicate an invalid operation, or provided by the user as an input. A signaling NaN has the MSB of its fraction set to 0 and at least one of the remaining fraction bits set to 1. A quiet NaN has the MSB of its fraction set to 1.

The IEEE format is fully described in IEEE Standard 754.

IEEE Double-Precision

The IEEE double-precision word is 64 bits wide and is arranged in the format shown below:



TB001060

The floating-point word is divided into three fields: a single-bit sign, an 11-bit biased exponent, and a 52-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; zero may have either sign.

The biased exponent is an 11-bit unsigned integer representing a multiplicative factor of some power of two. The bias value is 1023. If, for example, the multiplicative value for a

floating-point number is to be 2^a , the value of the biased exponent is $a + 1023$, where "a" is the true exponent.

The fraction is a 52-bit unsigned fractional field containing the 52 least-significant bits of the floating-point number's 53-bit mantissa. The weight of the fraction's most-significant bit is 2^{-1} . The weight of the least-significant bit is 2^{-52} .

An IEEE floating-point number is evaluated or interpreted as follows:

If $e = 2047$ and $f \neq 0$	value = Reserved operand Not-a-Number
If $e = 2047$ and $f = 0$	value = $(-1)^s \infty$ Infinity
If $0 < e < 2047$	value = $(-1)^s 2^{e-1023} (1.f)$ Normalized number
If $e = 0$ and $f \neq 0$	value = $(-1)^s 2^{-1022} (0.f)$ Denormalized number
If $e = 0$ and $f = 0$	value = $(-1)^s 0$ Zero

Infinity: Infinity can have either a positive or negative sign. The interpretation of infinities is determined by the Affine/Projective select input AFF/PROJ.

NaN: A NaN is interpreted as a signal or symbol. NaNs are used to indicate invalid operations, and as a means of passing process status through a series of calculations. They arise in

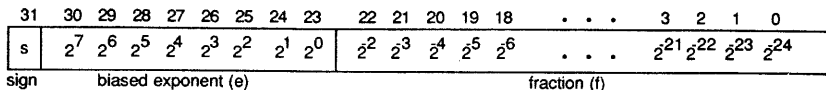
two ways: either generated by the Am29C327 to indicate an invalid operation, or provided by the user as an input. A signaling NaN has the MSB of its fraction set to 0 and at least one of the remaining fraction bits set to 1. A quiet NaN has the MSB of its fraction set to 1.

The IEEE format is fully described in IEEE Standard 754.

DEC Formats

DEC F

The DEC F word is 32 bits wide and is arranged in the format shown below:



TB001070

The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 23-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; zero has a positive sign.

The biased exponent is an 8-bit unsigned integer representing a multiplicative factor of some power of two. The bias value is 128. If, for example, the multiplicative value for a floating-point number is to be 2^a , the value of the biased exponent is $a + 128$, where "a" is the true exponent.

The fraction is a 23-bit unsigned fractional field containing the 23 least-significant bits of the floating-point number's 24-bit mantissa. The weight of the fraction's most-significant bit is 2^{-2} . The weight of the least-significant bit is 2^{-24} .

A DEC F floating-point number is evaluated or interpreted as follows:

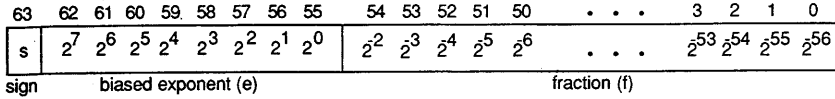
If $e \neq 0$	value $\neq (-1)^s 2^{e-128} (0.1f)$
If $s = 0$ and $e = 0$	value = 0
If $s = 1$ and $e = 0$	value = DEC-Reserved Operand

DEC-Reserved Operand: A DEC-Reserved Operand is interpreted as a signal or symbol. DEC-Reserved Operands are used to indicate invalid operations and operations whose results have overflowed the destination format. They may also be used to pass symbolic information from one calculation to another.

The DEC formats are fully described in the VAX Architecture Manual.

DEC D

The DEC D word is 64 bits wide and is arranged in the format shown below:



TB001080

The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 55-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; zero has a positive sign.

The biased exponent is an 8-bit unsigned integer representing a multiplicative factor of some power of two. The bias value is 128. If, for example, the multiplicative value for a floating-point number is to be 2^a , the value of the biased exponent is $a + 128$, where "a" is the true exponent.

The fraction is a 55-bit unsigned fractional field containing the 55 least-significant bits of the floating-point number's 56-bit mantissa. The weight of the fraction's most-significant bit is 2^{-2} . The weight of the least-significant bit is 2^{-56} .

A DEC D floating-point number is evaluated or interpreted as follows:

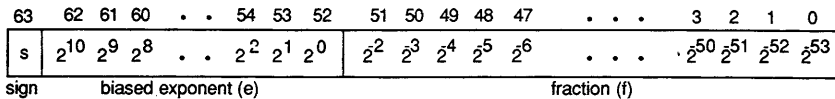
- If $e \neq 0$ value = $(-1)^s 2^{e-128} (0.1f)$
- If $s = 0$ and $e = 0$ value = 0
- If $s = 1$ and $e = 0$ value = DEC-Reserved Operand

DEC-Reserved Operand: A DEC-Reserved Operand is interpreted as a signal or symbol. DEC-Reserved Operands are used to indicate invalid operations and operations whose results have overflowed the destination format. They may also be used to pass symbolic information from one calculation to another.

The DEC formats are fully described in the VAX Architecture Manual.

DEC G

The DEC G word is 64 bits wide and is arranged in the format shown below:



TB001090

The floating-point word is divided into three fields: a single-bit sign, an 11-bit biased exponent, and a 52-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; zero has a positive sign.

The biased exponent is an 11-bit unsigned integer representing a multiplicative factor of some power of two. The bias value is 1024. If, for example, the multiplicative value for a floating-point number is to be 2^a , the value of the biased exponent is $a + 1024$, where "a" is the true exponent.

The fraction is a 52-bit unsigned fractional field containing the 52 least-significant bits of the floating-point number's 53-bit mantissa. The weight of the fraction's most-significant bit is 2^{-2} . The weight of the least-significant bit is 2^{-53} .

A DEC G floating-point number is evaluated or interpreted as follows:

- If $e \neq 0$ value = $(-1)^s 2^{e-1024} (0.1f)$
- If $s = 0$ and $e = 0$ value = 0
- If $s = 1$ and $e = 0$ value = DEC-Reserved Operand

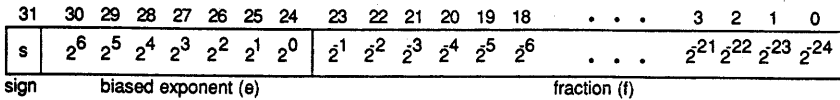
DEC-Reserved Operand: A DEC-Reserved Operand is interpreted as a signal or symbol. DEC-Reserved Operands are used to indicate invalid operations and operations whose results have overflowed the destination format. They may also be used to pass symbolic information from one calculation to another.

The DEC formats are fully described in the VAX Architecture Manual.

IBM Formats

IBM Single-Precision

The IBM single-precision word is 32 bits wide and is arranged in the format shown below:



TB001100

The floating-point word is divided into three fields: a single-bit sign, a 7-bit biased exponent, and a 24-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; a True-zero has a positive sign.

The biased exponent is a 7-bit unsigned integer representing a multiplicative factor of some power of 16. The bias value is 64. If, for example, the multiplicative value for a floating-point number is to be 16^a , the value of the biased exponent is $a + 64$, where "a" is the true exponent.

The fraction is a 24-bit unsigned fractional field containing the 24 least-significant bits of the floating-point number's 25-bit mantissa. The weight of the fraction's most-significant bit is 2^{-1} . The weight of the least-significant bit is 2^{-24} .

An IBM floating-point number is evaluated or interpreted as follows:

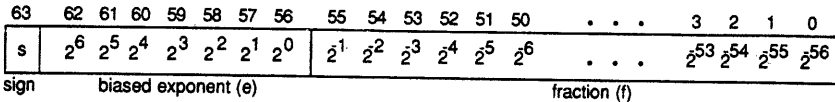
$$\text{value} = (-1)^s 16^e - 64 (0.f)$$

Zero: There are two possible classes of representations for zero. Since there is no leading bit in the IBM format, the range of the IBM fraction is equal to or greater than zero and less than one. If an operation causes the fraction of the result to cancel exactly, then the result is a floating-point zero. A True-zero has a positive sign, a biased exponent of zero, and a fraction of zero.

The IBM format is fully described in the IBM System/370 Principles of Operation Manual.

IBM Double-Precision

The IBM double-precision word is 64 bits wide and is arranged in the format shown below:



TB001110

The floating-point word is divided into three fields: a single-bit sign, a 7-bit biased exponent, and a 56-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; a True-zero has a positive sign.

The biased exponent is a 7-bit unsigned integer representing a multiplicative factor of some power of 16. The bias value is 64. If, for example, the multiplicative value for a floating-point number is to be 16^a , the value of the biased exponent is $a + 64$, where "a" is the true exponent.

The fraction is a 56-bit unsigned fractional field containing the 56 least-significant bits of the floating-point number's 57-bit mantissa. The weight of the fraction's most-significant bit is 2^{-1} . The weight of the least-significant bit is 2^{-56} .

An IBM floating-point number is evaluated or interpreted as follows:

$$\text{value} = (-1)^s 16^e - 64 (0.f)$$

Zero: There are two possible classes of representations for zero. Since there is no leading bit in the IBM format, the range of the IBM fraction is equal to or greater than zero and less than one. If an operation causes the fraction of the result to cancel exactly, then the result is a floating-point zero. A True-zero has a positive sign, a biased exponent of zero, and a fraction of zero.

The IBM format is fully described in the IBM System/370 Principles of Operation Manual.

APPENDIX B — ROUNDING MODES

The Am29C327 provides six rounding modes for floating-point operations, and for integer multiplication:

RM2	RM1	RM0	Round Mode
0	0	0	Round to Nearest (IEEE)
0	0	1	Round to Minus Infinity
0	1	0	Round to Plus infinity
0	1	1	Round to Zero
1	0	0	Round to Nearest (DEC)
1	0	1	Round Away From Zero
1	1	X	Illegal Value

Round to Nearest IEEE (Unbiased)

The infinitely precise result of an operation is rounded to the closest representable value in the destination format. If the infinitely precise result is exactly halfway between two representations, it is rounded to the representation having a least-significant bit of zero. This rounding mode conforms to the "round to nearest" mode described in the IEEE Floating-Point Standard.

Round to Minus Infinity

The infinitely precise result of an operation is rounded to the closest representable value in the destination format that is less than or equal to the infinitely precise result. This rounding mode conforms to the "round to minus infinity" mode described in the IEEE Floating-Point Standard.

Round to Plus Infinity

The infinitely precise result of an operation is rounded to the closest representable value in the destination format that is greater than or equal to the infinitely precise result. This round mode conforms to the "round to plus infinity" mode described in the IEEE Floating-Point Standard.

Round to Zero

The infinitely precise result of an operation is rounded to the closest representable value in the destination format whose magnitude is less than or equal to the infinitely precise result. This rounding mode conforms to the "round to zero" mode described in the IEEE Floating-Point Standard.

Round to Nearest DEC (Biased)

The infinitely precise result of an operation is rounded to the closest representable value in the destination format. If the infinitely precise result is exactly halfway between two representations, it is rounded to the representation having the greater magnitude. This rounding mode is used by DEC VAX computers.

Round Away from Zero

The infinitely precise result of an operation is rounded to the closest representable value in the destination format whose magnitude is greater than or equal to the infinitely precise result.

A graphical representation of these rounding modes is shown in Figures B1-1 and B1-2.

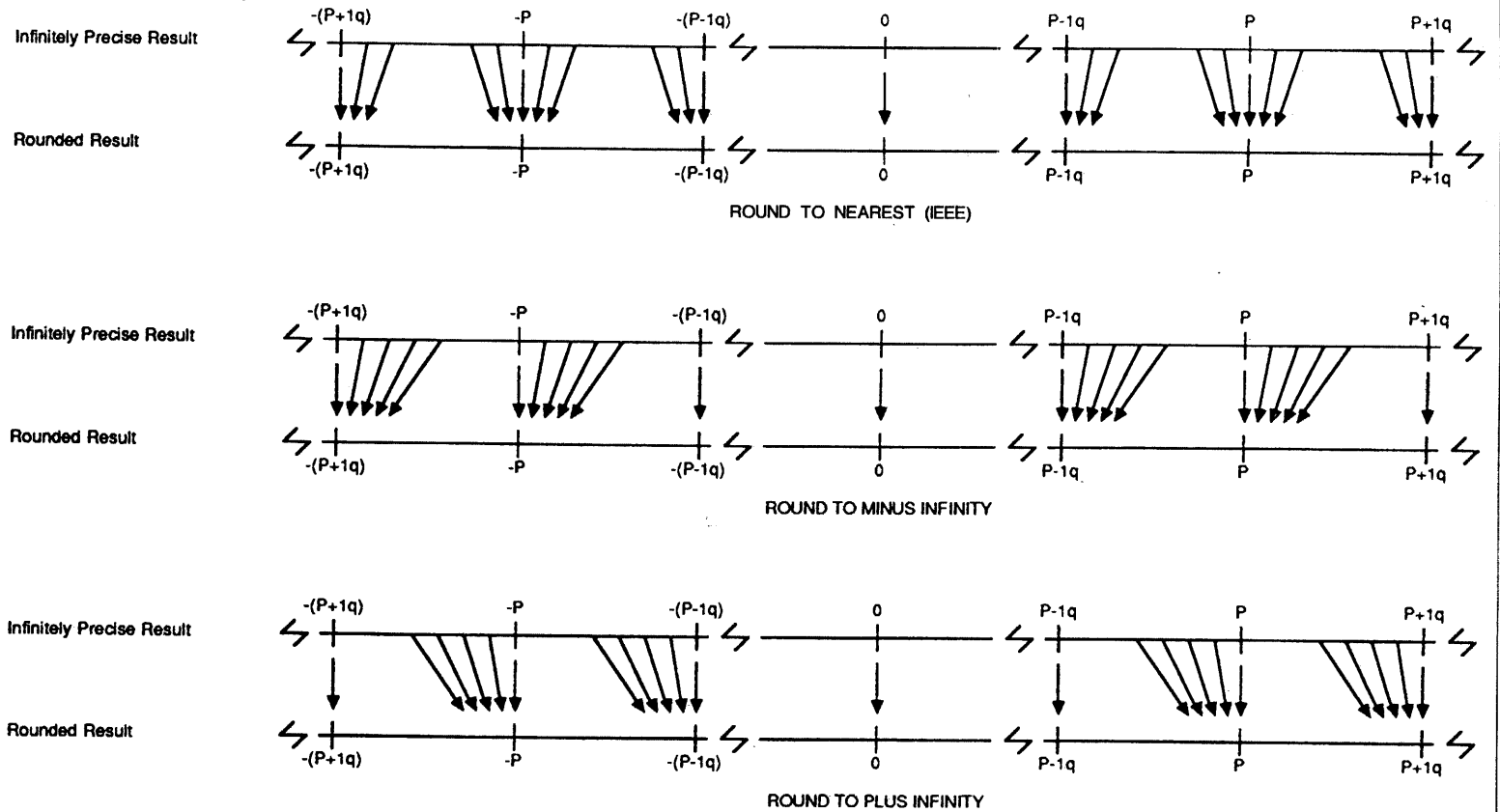


Figure B1-1. Graphical Interpretation of IEEE Round-to-Nearest, Round-to-Minus-Infinity, and Round-to-Plus-Infinity Rounding Modes

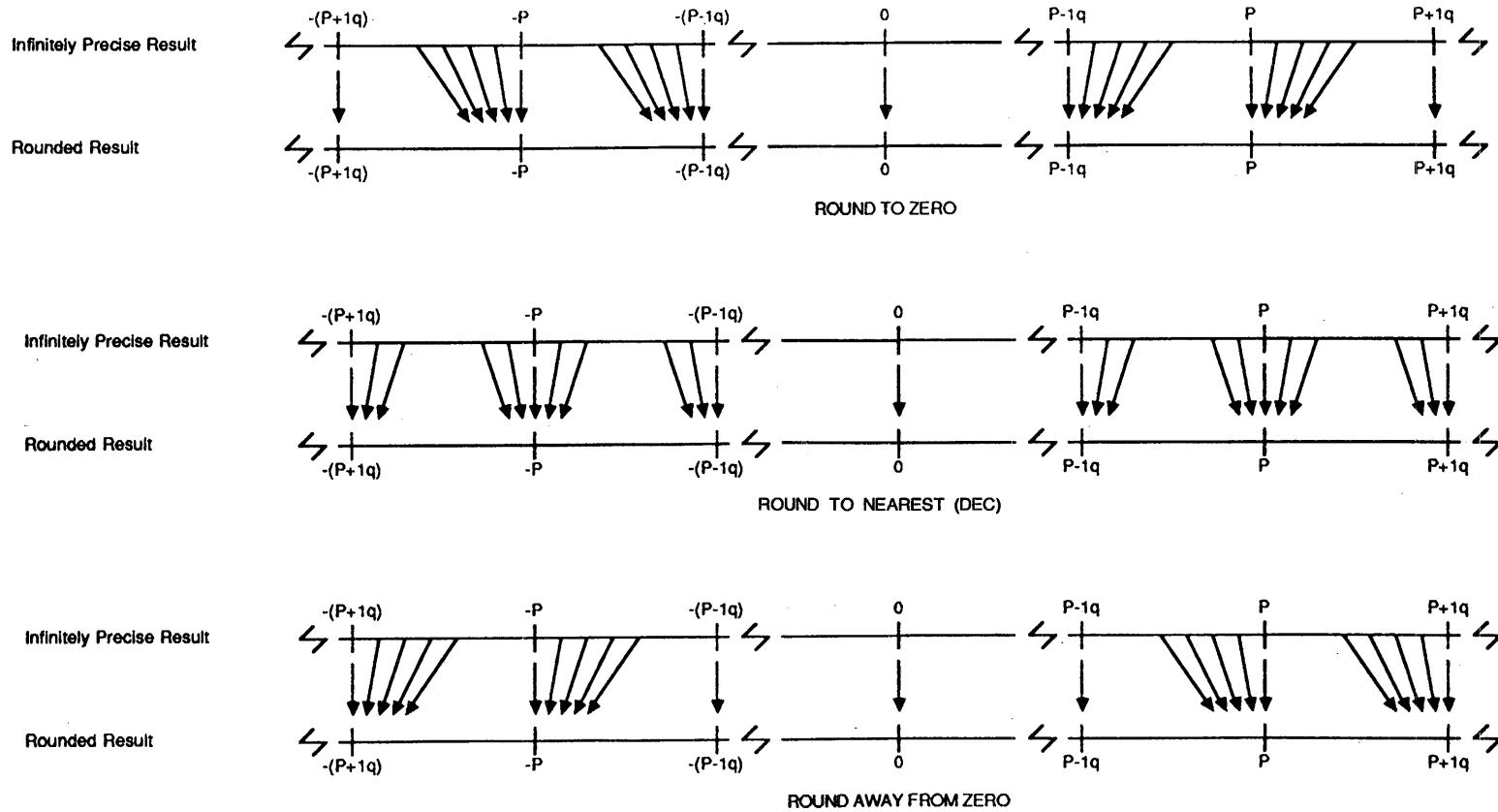


Figure B1-2. Graphical Interpretation of Round-to-Zero, DEC Round-to-Nearest, and Round-Away-from-Zero Rounding Modes

APPENDIX C — ADDITIONAL OPERATION DETAILS

Differences Between IEEE Floating-Point Standard and Am29C327 IEEE Operation

The IEEE floating-point standard recommends that a trapped overflow on conversion from a binary format return a result in that or a wider format, rounded to the destination format. The Am29C327 returns an operand in the destination format,

rounded to that format. Note that trapped operation is an optional aspect of the IEEE floating-point standard, and as such, is not necessary for compliance.

Differences Between IBM 370 Floating-Point Arithmetic and Am29C327 IBM Operation

For all arithmetic operations, the Am29C327 in general will produce a more precise result than the IBM 370.

Differences Between DEC Floating-Point Arithmetic and Am29C327 DEC Operation

The Am29C327 and DEC VAX floating-point formats contain identical information, but the sub-fields of the floating-point words are arranged differently:

The Am29C327 DEC F format is:

sign - bit 31
exponent - bits 30 - 23
mantissa - bits 22 - 0

The Am29C327 DEC D format is:

sign - bit 63
exponent - bits 62 - 55
mantissa - bits 54 - 0

The VAX format is:

sign - bit 15
exponent - bits 14 - 7
mantissa - bits 6 - 0,
bits 31 - 16

The VAX format is:

sign - bit 15
exponent - bits 14 - 7
mantissa - bits 6 - 0,
bits 31 - 16,
bits 47 - 32,
bits 63 - 48
bit 6 = MSB,
bit 48 = LSB

ABSOLUTE MAXIMUM RATINGS

Storage Temperature	-65 to +150°C
Ambient Temperature (T _A)	
Under Bias	-55 to +125°C
Supply Voltage to	
Ground Potential Continuous	-0.5 to +7.0 V
DC Voltage Applied to	
Outputs for HIGH State	-0.5 V to +V _{CC} Max.
DC Input Voltage	-0.5 to +5.5 V
DC Output Current, Into Outputs	30 mA
DC Input Current	-10 to +10 mA

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to absolute maximum ratings for extended periods may affect device reliability.

OPERATING RANGES

Commercial (C) Devices	
Temperature (T _A)	0 to +70°C
Supply Voltage (V _{CC})	+5 V ± 5%
Min.	+4.75 V
Max.	+5.25 V
Military (M) Devices	
Temperature (T _A)	-55 to +125°C
Supply Voltage (V _{CC})	+5 V ± 10%
Min.	+4.5 V
Max.	+5.5 V

Operating ranges define those limits between which the functionality of the device is guaranteed.

DC CHARACTERISTICS over operating range unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions (Note 1)	Min.	Max.	Unit
V _{OH}	Output HIGH Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH} V _{IL} = 0.8 V V _{IH} = 2.0 V I _{OH} = -0.4 mA	2.4		V
V _{OL}	Output LOW Voltage	V _{CC} = Min. V _{IN} = V _{IL} or V _{IH} V _{IL} = 0.8 V V _{IH} = 2.0 V I _{OL} = 4.0 mA		0.5	V
V _{IH}	Input HIGH Level	Guaranteed Input Logical-HIGH Voltage for All Inputs	2.0		V
V _{IL}	Input LOW Level	Guaranteed Input Logical-LOW Voltage for All Inputs		0.8	V
V _I	Input Clamp Voltage	V _{CC} = Min. I _{IN} = -18 mA		-1.5	V
I _{IL}	Input LOW Current	V _{CC} = Max. V _{IN} = 0.4 V		-0.4	mA
I _{IH}	Input HIGH Current	V _{CC} = Max. V _{IN} = 2.4 V		75	μA
I _I	Input HIGH Current	V _{CC} = Max. V _{IN} = 5.5 V		1	mA
I _{OZH}	Off-State (High-Impedance) Output Current	V _{CC} = Max.	V _O = 2.4 V	25	μA
I _{OZL}			V _O = 0.4 V	-25	
I _{SC} (Note 2)	Output Short-Circuit Current	V _{CC} = Max. V _O = 0 V All Outputs	-3	-30	mA
I _{CC} (Note 3)	Power Supply Current		COM'L	300	mA
			MIL	350	
I _{CCQ1} (Note 4)	Quiescent Power Supply Current		COM'L		mA
			MIL		
I _{CCQ2} (Note 5)	Quiescent Power Supply Current		COM'L		mA
			MIL		

- Notes: 1. For conditions shown as Min. or Max., use the appropriate value specified under Electrical Characteristics for the applicable device type.
 2. Not more than one output should be shorted at a time. Duration of the short-circuit test should not exceed one second.
 3. I_{CC} is measured with clock frequency = 8 MHz and with outputs disabled. Inputs should be presented with random logic-HIGHs and LOWs to assure the toggling of internal nodes.
 4. V_{IN} ≥ V_{IH}, V_{IN} ≤ V_{IL}
 5. V_{IN} ≥ V_{CC} - 0.2 V, V_{IN} ≤ 0.2 V

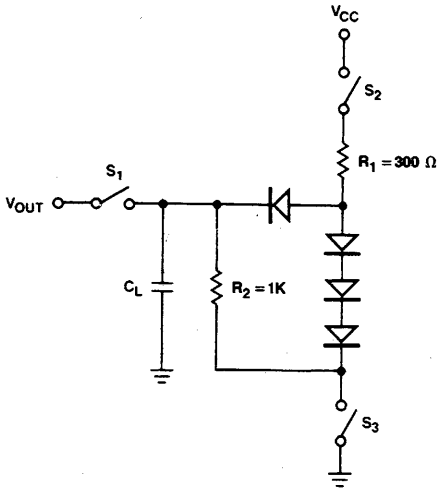
SWITCHING CHARACTERISTICS over operating range unless otherwise specified

No.	Parameter Description	Test Conditions	Min.	Max.	Unit
1	CLK Period	(Note 1)			
	Flow-Through Mode Multiply-Accumulate All Other Operations		360 240	DC DC	ns ns
	Single-Pipelined Mode Multiply-Accumulate All Other Operations		240 120	DC DC	ns ns
	Double-Pipelined Mode Multiply-Accumulate		120	DC	ns
2	CLK LOW Time				ns
3	CLK HIGH Time				ns
4	CLK Rise Time	(Note 2)			ns
5	CLK Fall Time	(Note 2)			ns
6	Data/Instruction Setup Time	(Note 3)	15		ns
7	Data/Instruction Hold Time	(Note 3)	0		ns
8	Control Lines Setup Time	(Note 4)	15		ns
9	Control Lines Hold Time	(Note 4)	0		ns
10	F ₀₋₃₁ CLK-to-Output-Valid F Register Clocked			20	ns
11	FLAG ₁₋₆ SIGN CLK-to-Output-Valid Register Clocked			20	ns
12	F ₀₋₃₁ CLK-to-Output-Valid F Register Transparent				
	Flow-Through Mode Multiply-Accumulate All Other Operations			380 260	ns ns
	Single-Pipelined Mode Multiply-Accumulate All Other Operations			260 140	ns ns
	Double-Pipelined Mode Multiply-Accumulate			140	ns
13	FLAG ₁₋₆ SIGN CLK-to-Output-Valid S Register Transparent				
	Flow-Through Mode Multiply-Accumulate All Other Operations			380 260	ns ns
	Single-Pipelined Mode Multiply-Accumulate All Other Operations			260 140	ns ns
	Double-Pipelined Mode Multiply-Accumulate			140	ns
14	OE _F , OE _S , Disable Time HIGH to Z			15	ns
15	OE _F , OE _S , Disable Time LOW to Z			15	ns
16	OE _F , OE _S , Enable Time Z to HIGH			20	ns
17	OE _F , OE _S , Disable Time Z to LOW			20	ns
18	FSEL to F ₀₋₃₁			20	ns
19	MSERR Data-to-Valid Delay			20	ns

- Notes:** 1. CLK switching characteristics are made relative to 2.5 V.
 2. CLK rise time and fall time measured between 0.8 V and (V_{CC} - 1.0 V).
 3. Data/Instruction signals include R₀₋₃₁, S₀₋₃₁, S/DR, S/DS, S/DF, RM₀₋₂, PSEL₀₋₃, QSEL₀₋₃, TSEL₀₋₃ and I0-13.
 4. Control signals include ENR, ENS, ENF, ENRF, RFSEL₀₋₂, FSEL, ENI, OE_F, and OE_S.

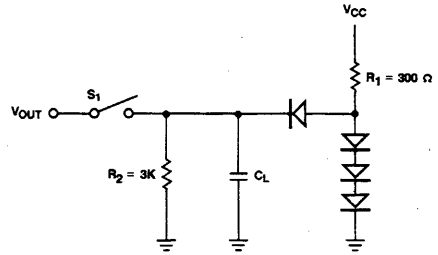
- Conditions:** A. All inputs/outputs except CLK are TTL-compatible for V_{IH}, V_{IL}, and V_{OL}.
 B. All outputs are driving 80 pF unless otherwise noted.
 C. All setup, hold, and delay times are measured relative to CLK at V_{CC}/2 volts unless otherwise noted.

SWITCHING TEST CIRCUITS



TCR01331

A. Three-State Outputs

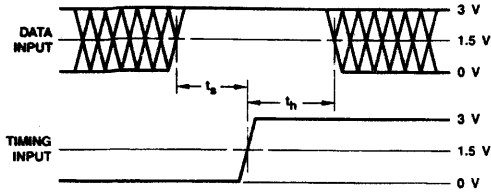


TC000421

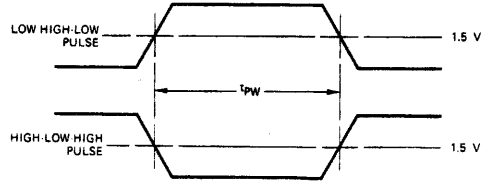
B. Normal Outputs

- Notes:
1. $C_L = 50$ pF includes scope probe, wiring, and stray capacitances without device in test fixture.
 2. S_1, S_2, S_3 are closed during function tests and all AC tests except output enable tests.
 3. S_1 and S_3 are closed while S_2 is open for t_{pZH} test.
 4. $C_L = 5.0$ pF for output disable tests.

SWITCHING TEST WAVEFORMS



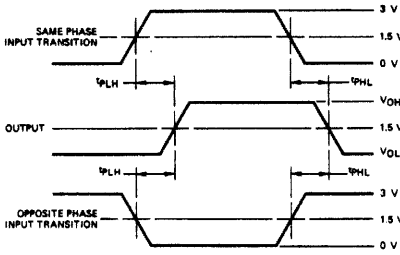
WFR02970



WFR02790

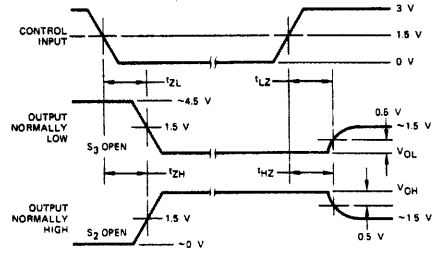
- Notes: 1. Diagram shown for HIGH data only. Output transition may be opposite sense.
2. Cross-hatched area is don't care condition.

Setup, Hold, and Release Times



WFR02980

Pulse Width



WFR02660






- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
2. S_1 , S_2 and S_3 of Load Circuit are closed except where shown.

Propagation Delay

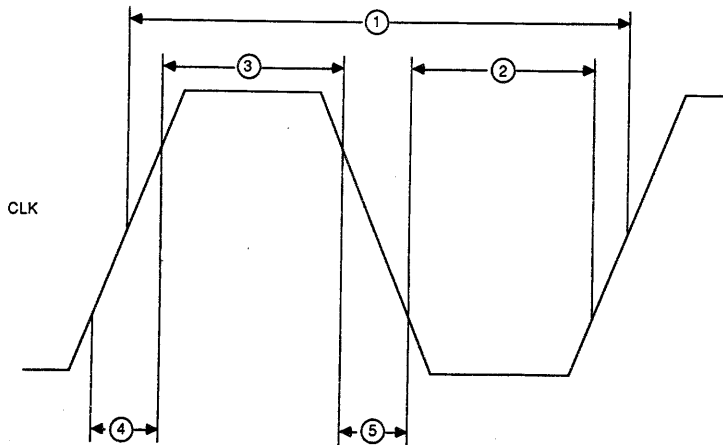
Enable and Disable Times

SWITCHING WAVEFORMS

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	MUST BE STEADY	WILL BE STEADY
	MAY CHANGE FROM H TO L	WILL BE CHANGING FROM H TO L
	MAY CHANGE FROM L TO H	WILL BE CHANGING FROM L TO H
	DON'T CARE; ANY CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	DOES NOT APPLY	CENTER LINE IS HIGH IMPEDANCE "OFF" STATE

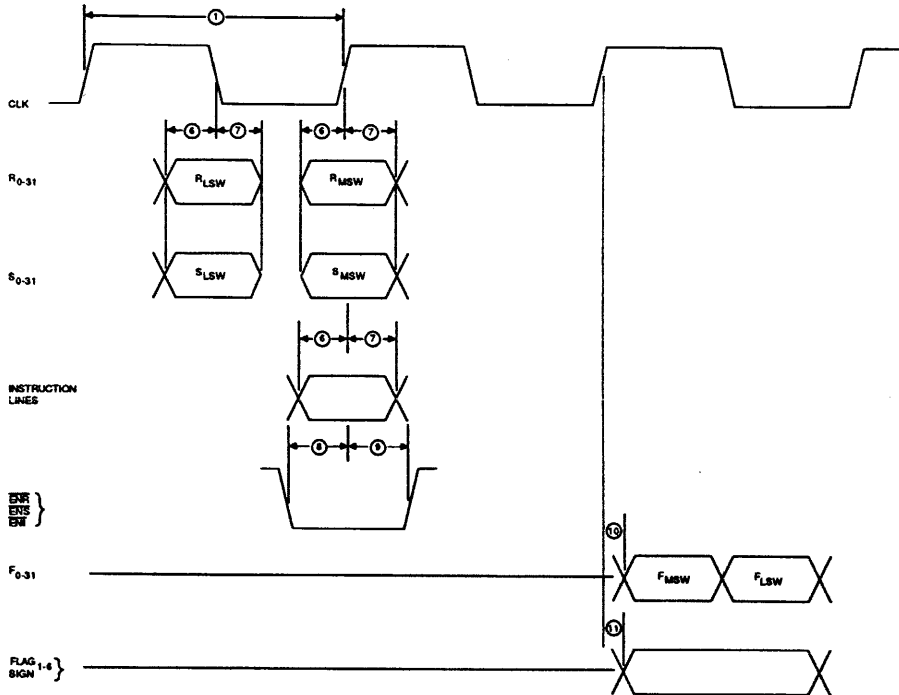
KS000010



WF025010

Input Clock Timing

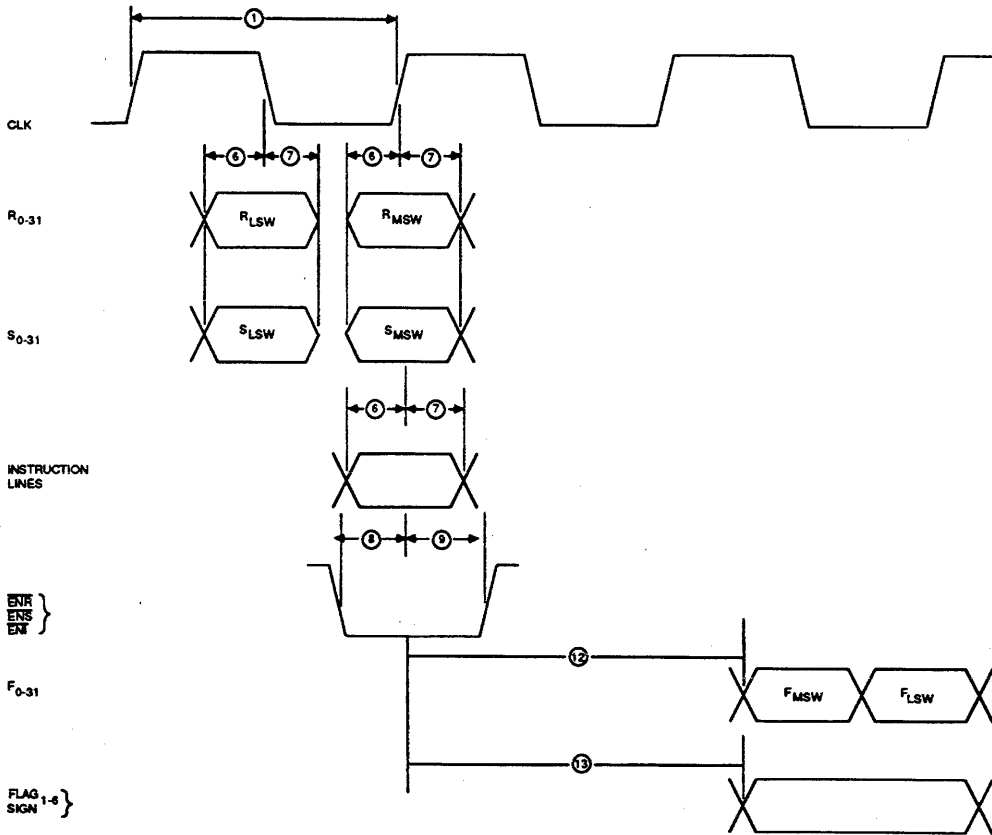
SWITCHING WAVEFORMS (Cont'd.)



WF025020

Timing of Operations with F Register and Status Clocked. Assumes 32-Bit Bus, Single-Cycle, LSW-First Input Mode and Flow-Through Operation

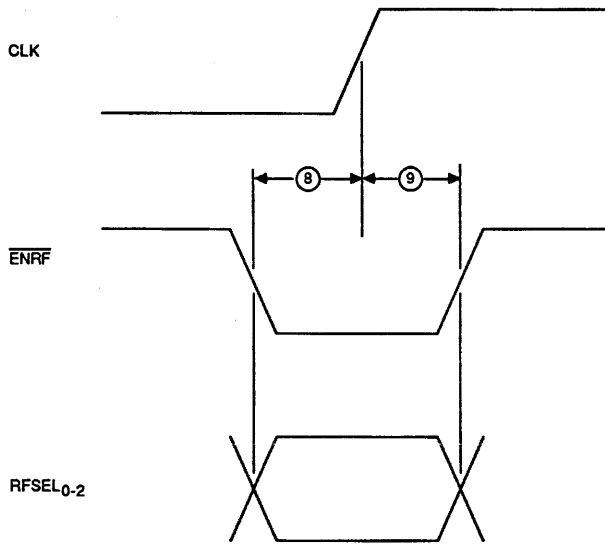
SWITCHING WAVEFORMS (Cont'd.)



WF025030

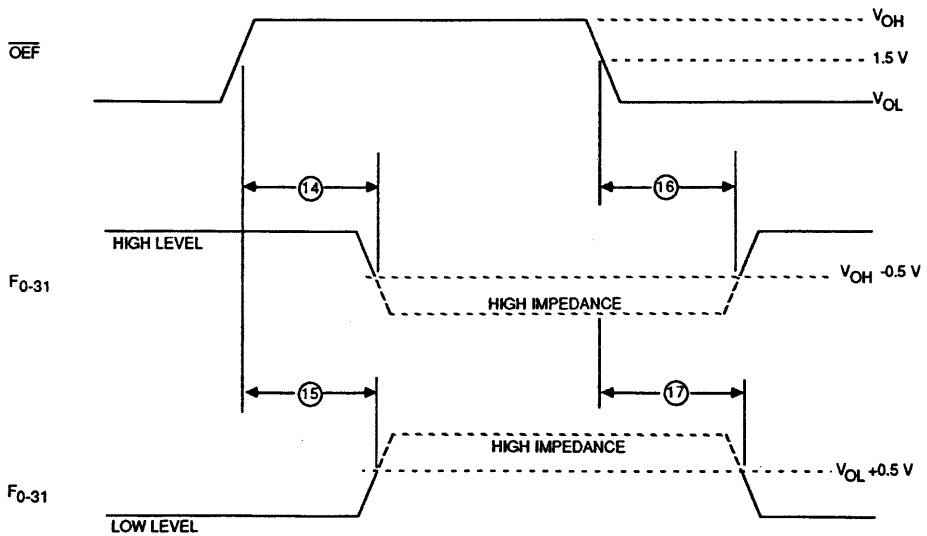
Timing of Operations with F-Register and Status Register in Feedthrough Mode. Assumes 32-Bit Bus, Single-Cycle, LSW-First Input Mode and Flow-Through Operation.

SWITCHING WAVEFORMS (Cont'd.)



WF025040

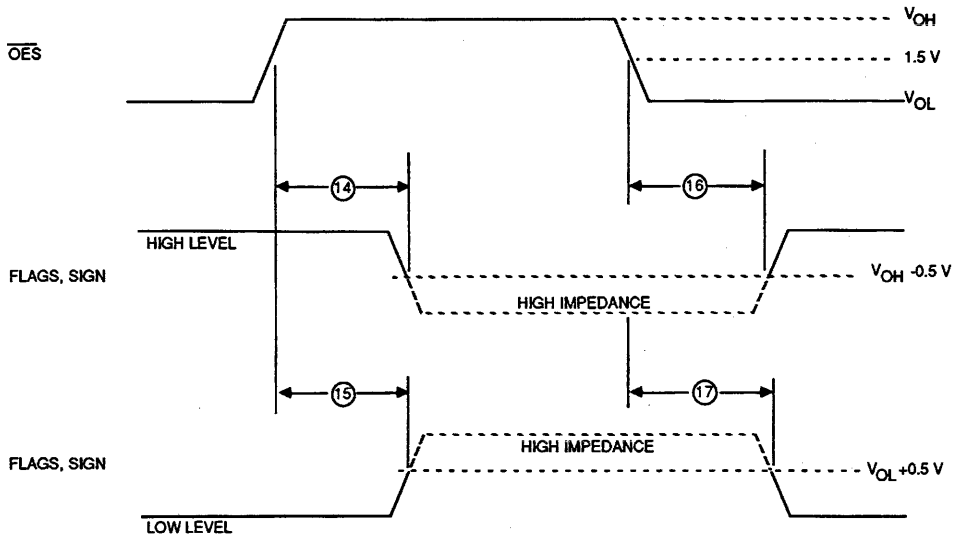
Register File Control Timing



WF025050

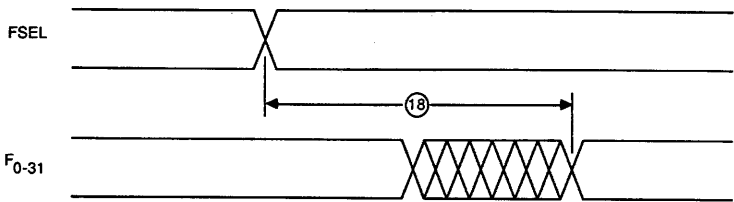
Enable/Disable Timing for F_0-31

SWITCHING WAVEFORMS (Cont'd.)



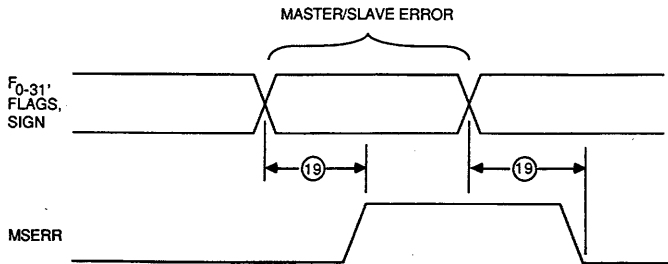
WF025060

Enable/Disable Timing for FLAG₁₋₆ and SIGN



WF025070

Output Selection Timing



WF025080

Master/Slave Timing (Assumes SLAVE Mode)

CHAPTER 5

Support Tools	
5.1 Am29C300 EVALUATION BOARD	5-1
5.2 Am29300 TEST BOARD	5-4
5.3 Am29300 DEFINITION FILE	5-7
5.4 MICROCODE DEVELOPMENT	5-23
5.4.1 Step Engineering 32-Bit Development Tools	5-23
5.4.2 Microtech mcASM Structured Microcode Assembler	5-31
5.4.3 Hilevel Technology	5-37
5.4.4 Hewlett-Packard Microprogram Development Support	5-43
5.5 SIMULATION MODELS	5-48
5.6 C COMPILER SUPPORT	5-54
5.7 WRITABLE CONTROL STORE	5-59
5.7.1 Agility AG-11B Microprogram Development	5-59

CHAPTER 5

Support Tools



Advanced Micro Devices is recognized as the pioneer and leading supplier of fast microprogrammable bit-slice and related integrated circuits used in a wide variety of high-performance systems

Because of their flexibility, these microprogrammable ICs require a deeper understanding of hardware than required by a typical MOS microprocessor. But there is no reason to shy away from microprogramming: it is not difficult, and there are several hardware and software tools available.

Tools that help the systems engineer design his system can be in the form of hardware, software, written materials, and even professional advice. The importance of support to any design approach, and the relative difficulty of microcoded design, require a detailed explanation.

As more support is provided to the customer, ease-of-design improves and time-to-market decreases. The design process becomes less tedious, risk is reduced, and a lower skill level is required of the designer to implement a successful system. In general, the more rigid a device family becomes (i.e., fixed architecture/fixed instruction set), the easier it is to support.

When assessing the support available for a design approach, considerations need to be given to the realities of the situation. For instance, building blocks offer a flexibility in architecture and programming that can only be equaled in gate arrays (which can be even more versatile). The informed engineer would not ask the question, "Can I get compiler support for what I build with gate arrays?" The answer would obviously be, "Only if you emulated something that was already supported, or targeted a compiler to your new creation." Until tools become available that automatically generate compilers, it will remain the case that more flexible approaches get you closer to the hardware and away from higher level language, and usually result in better performance.

It is impossible to even imagine all of the various ways a microcoded system might be constructed. Further, since the architecture is not fixed, it is not possible to pre-define a compiler or assembler for the system. If the full flexibility of the microprogrammed-building-block approach is to be maintained, then a penalty must be paid in terms of a lack of high-level language support. Fortunately, a good meta assembler greatly alleviates the programming task. Of course, once a system is defined, a compiler may be developed, but not cheaply. With these tradeoffs now in mind, we can present tools available to the Am29300/29C300 family.

5.1 Am29C300 EVALUATION BOARD

The Am29C300 Evaluation Board is an educational tool to help the user understand the Am29C300 32-bit building-block family. With all the major devices of the Am29C300 family and an on-board debug monitor, the board provides an excellent tool for those who would like to learn more about the Am29C300 family. A block diagram of the board is shown in Figure 5-1.

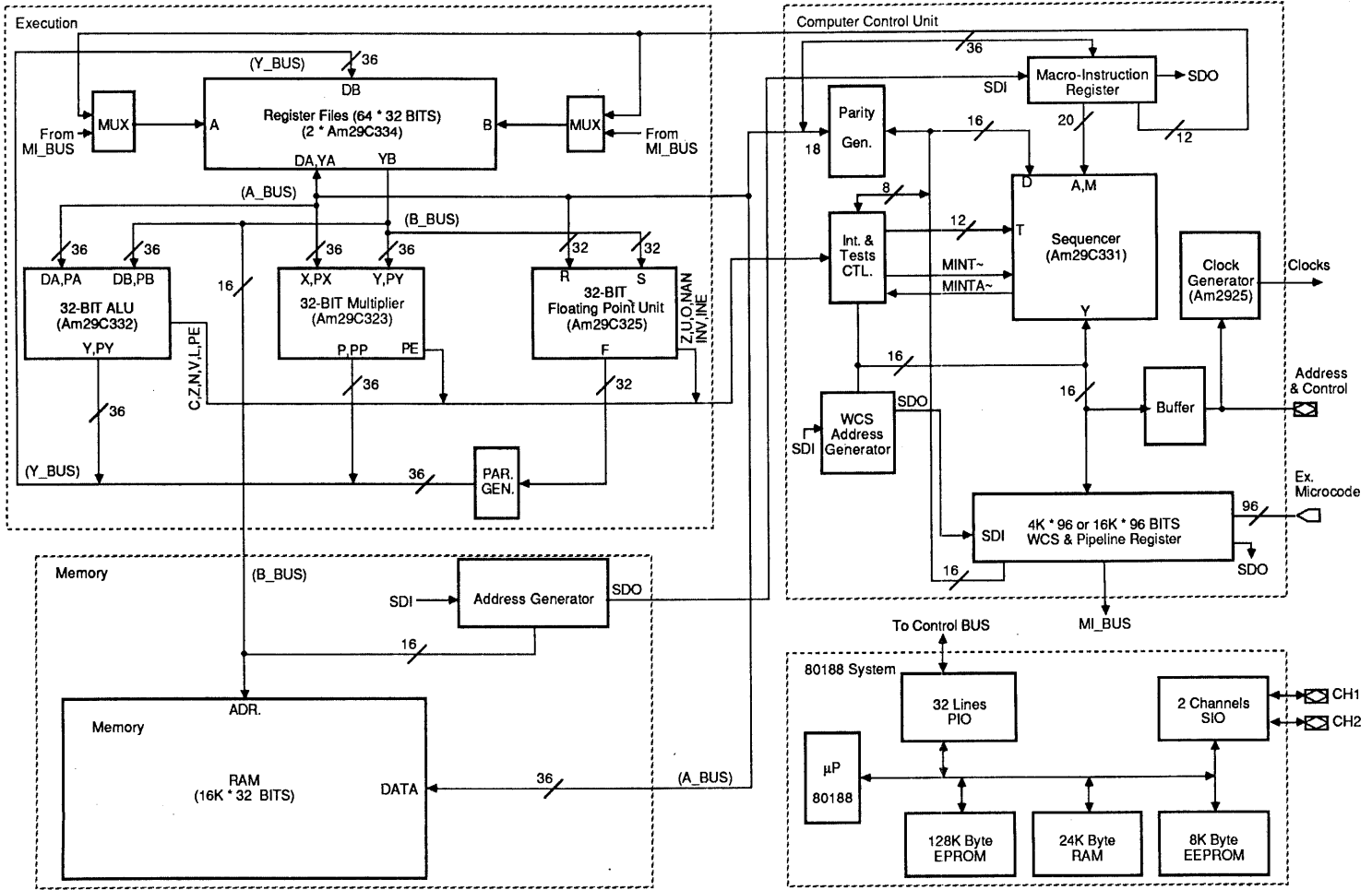
The board consists of two systems: the 80188 and Am29C300 system. The 80188 system is a front-end processor which provides the necessary interface between the board and external sources, such as a CRT terminal. Through a parallel interface between the 80188 system and the Am29C300 system, the 80188 system can control and monitor the activity of the Am29C300 system, which is a 32-bit system with three major parts: a computer control unit, an execution unit, and memory.

Am29C300 System

As a standard computer architecture, the computer control unit provides all the control signals for the Am29C300 system. It includes several major hardware logics: sequencer (Am29C331), writable control store, pipeline register, interrupt controller, and macro instruction register. Its operation is a very standard procedure. First, it fetches and stores a macro instruction into the macro-instruction register; then, the opcode of the macro instruction is decoded to find a correct micro-routine for the macro instruction. Finally, the selected micro-routine controls the operations of the execution unit and the memory.

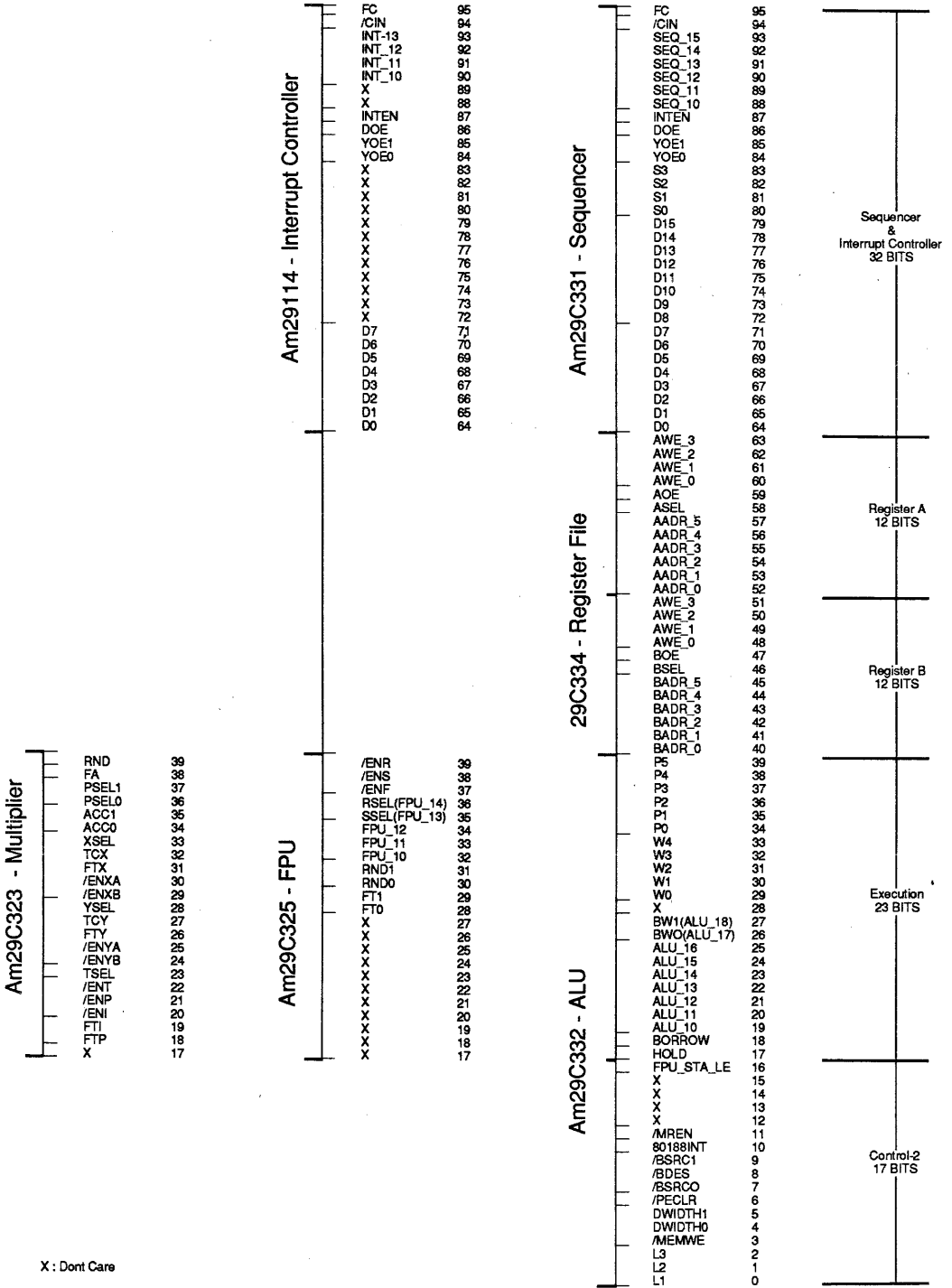
With the building blocks of the Am29C300 family, a powerful execution unit has been implemented on the board. The execution unit is able to handle 32-bit arithmetic and logic operations, multi-precision multiplication and division, and single-precision floating-point calculations within a reasonable time period. Also, the execution unit has 64 32-bit registers in which to store data. The following Am29C300 building blocks have been included in the execution unit:

- Am29C334 – 64 x 18 Bit Dual-Access Four-Port Register File
- Am29C332 – 32-Bit Arithmetic Logic Unit
- Am29C323 – 32-Bit Parallel Multiplier
- Am29C325 – Single-Precision Floating-Point Processor



09372A 5.1-1

Figure 5-1. Am29C300 Evaluation Board



X : Dont Care

06372A 5.1-2

Figure 5-2. Am29C300 EVB Microcode BIT Map

The memory architecture is very straightforward. It includes 12 static RAMs and a control PAL. Three bits of the microcode are decoded by the control PAL to generate chip selects and write pulses for the RAMs. A register in the execution unit should act as a program counter to provide addresses for the RAMs.

Microcode

The 96-bit wide microcode is divided into five major fields: sequencer and interrupt field, register A field, register B field, execution field, and control field. A detailed microcode format is shown in Table 5-1.

Monitor

The monitor of the Am29C300 evaluation board is implemented in C and controlled by the 80188 system. It provides a limited microcode assembler and disassembler, a download and upload utility, and a microcode

debugger. The debugger includes various useful features such as single step, break point, and display of register contents.

5.2 Am29300 TEST BOARD

With the increasing complexity of integrated circuits, it is often necessary to check the functionality of an IC. The IBM PC board allows the user to functionally check any Am29300 family device by writing input test vectors. The software accompanying the board takes these input vectors one at a time, applies them to the device under test, clocks the device, and produces output vectors. Figure 5-3 shows the architecture of the board. As stated above, the intention is to allow users to familiarize themselves with the functionality of the part. AC specs cannot be verified. Sample input and output files for the Am29331 are also shown.

Table 5-1

32 Bits	12 Bits	12 Bits	23 Bits	17 Bits
Sequencer & Interrupt Controller Am29C331 Am29114	Register A (Source) Am29C334_A_Port	Register B (Source & Destination) Am29C334_B_Port	Execution Am29C332 Am29C323 Am29C325	Control Am2925

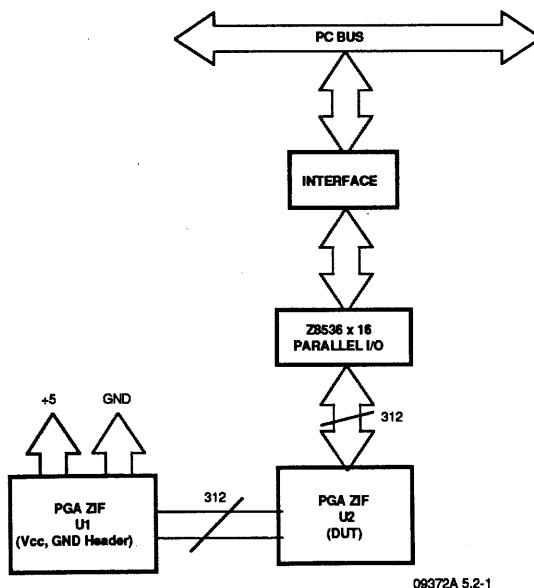


Figure 5-3. Am29300 Testboard – Block Diagram

Am29331 Input File

```

socket 120
63,76,120,96,95,83,82
107,93,79,80,81,67
69,94,68,62
61,55,39,57,56,43,44,45,37,38,25,26
65,60,48,28,64,53,40,27,58,52,41,14,59,47,42,1
108,85,86,100,114,90,104,105,24,10,8,20,19,30,29,2
109,98,99,88,115,103,117,106,12,23,22,33,6,18,4,15
46,77
97,111,113,101,102,91,92,119,13,36,35,21,7,31,17,16
84,72,78,71
73,74,89,34,66,75,11,3,118,110
32,87,54,49,51,50,5,116,9,112,70;
:
:           M M M M
:           3 2 1 0
:           T . . . . D A Y A
:           S I I S 1 3 3 3 3 1 1 1 / - E E
:           / H L N I 5 3 1 - - - - 5 5 / 5 I F R Q
:           R O A T N - - - 3 2 1 0 - - C O - N U R U V G
:           C S F L V E T I S T . . . . D A I E Y T L O A C N
:           P T C D E N R 0 0 0 0 0 0 0 0 0 0 N D 0 A L R L C D
:
:specify base for each column
:
& B B B B B B B QH H HHH H H H H HHHH HHHH B B HHHH B B B B QHH OHH
:
:specify pin direction for each column
:
% I I I I I I I II I III I I I I IIII IIII I I OOOO O O O O OOO OOO
:RESET
001 w 0 X 0 X X X X XX X XXX X X X X XXXX XXXX X 0 OOOO O O O O OOO OOO - 001 A
:
:CONTINUE, BRCC_D, CONTINUE
:
002 w 1 0 0 0 1 0 30 X XXX X X X X XXXX XXXX 0 0 OOOO O O O O OOO OOO - 001 A
003 w 1 0 0 0 1 0 00 0 001 X X X X 8971 XXXX 0 0 OOOO O O O O OOO OOO - 001 A
004 w 1 0 0 0 1 0 30 X XXX X X X X XXXX XXXX 0 0 OOOO O O O O OOO OOO - 004 A
005 w 1 0 0 0 1 0 30 X XXX X X X X XXXX XXXX 0 0 OOOO O O O O OOO OOO - 003 L

```


Am29331 Output File

```

socket 120
63,76,120,96,95,83,82
107,93,79,80,81,67
69,94,68,62
61,55,39,57,56,43,44,45,37,38,25,26
65,60,48,28,64,53,40,27,58,52,41,14,59,47,42,1
108,85,86,100,114,90,104,105,24,10,8,20,19,30,29,2
109,98,99,88,115,103,117,106,12,23,22,33,6,18,4,15
46,77
97,111,113,101,102,91,92,119,13,36,35,21,7,31,17,16
84,72,78,71
73,74,89,34,66,75,11,3,118,110
32,87,54,49,51,50,5,116,9,112,70;
:
:
:           M M M M
:           3 2 1 0
:           T . . . . D A Y A
:           S I I S I 3 3 3 3 1 1 / - E E
: / H L N I 5 3 1 - - - - 5 5 / 5 I F R Q
: R O A T N - - - 3 2 1 0 - - C O - N U R U V G
: C S F L V E T I S T . . . . D A I E Y T L O A C N
: P T C D E N R 0 0 0 0 0 0 0 0 0 0 N D 0 A L R L C D
:
:specify base for each column
:
& B B B B B B B QH H HHH H H H H HHHH HHHH B B HHHH B B B B QHH OHH
:
:specify pin direction for each column
:
% I I I I I I I II I III I I I I IIII IIII I I O000 O O O O OOO OOO
:RESET
001 w 0 0 0 0 0 0 0 00 0 000 0 0 0 0 0000 0000 0 0 0000 1 0 0 0 3FF 000 - 001
:
:CONTINUE, BRCC_D, CONTINUE
:
002 w 1 0 0 0 1 0 30 0 000 0 0 0 0 0000 0000 0 0 0001 1 0 0 0 3FF 000 - 001
003 w 1 0 0 0 1 0 00 0 001 0 0 0 0 8971 0000 0 0 8971 1 0 0 0 3FF 000 - 001
004 w 1 0 0 0 1 0 30 0 000 0 0 0 0 0000 0000 0 0 8972 1 0 0 0 3FF 000 - 001
004 w 1 0 0 0 1 0 30 0 000 0 0 0 0 0000 0000 0 0 8973 1 0 0 0 3FF 000 - 002
004 w 1 0 0 0 1 0 30 0 000 0 0 0 0 0000 0000 0 0 8974 1 0 0 0 3FF 000 - 003
004 w 1 0 0 0 1 0 30 0 000 0 0 0 0 0000 0000 0 0 8975 1 0 0 0 3FF 000 - 004
005 w 1 0 0 0 1 0 30 0 000 0 0 0 0 0000 0000 0 0 8978 1 0 0 0 3FF 000 - 003

```

5.3 Am29300 DEFINITION FILE

Introduction

The definition file contains the description of the micro machine for which assemblies are to be performed. Its innate flexibility allows the assembler to be retargetted to support any given bit-slice microprocessor machine and instruction format. The definition file is composed of:

- Instruction Definition
- Macro Definitions

The definition file is stored on a floppy disk and can be requested from your local AMD sales office.

Instruction Definition

The instruction definition defines a name for the instruction, the length of the instruction, the fields of the instruction and variation in format, allowable values for each field, and default values for each field.

The instruction definition contains:

- Field Definitions
- Case Definitions

Field Definition

A field in a microinstruction is a group of bits that are logically related and are manipulated as a unit. The form of the field definition is:

```
<fielddef l> <descript 1>
  <descript 2> (<const 1> : <id 1>,
               <const 2> : <id 2>,
               .           .
               .           .
               <const m> : <id m>)
```

<fielddef l> is a name of a field definition to be defined. <const i> is an integer-valued expression of an identifier. <id i> defines a name of an identifier. A descriptor <descript> specifies the size and location of the field and assigns valid values for the field. Valid descriptors are as follows:

Bits:	Bits that make up a field
Length:	Length of a field
Default:	Default values for a field
Values:	Definitions of names for field values

Invert:	One's complement field values
Complement:	Two's complement field values
Mask:	Use low bits of value, ignore high order bits
Reverse:	Reverse order of bits in field
Valid:	A list of valid values for the field
Display:	Display mode for debugging

The following is an example of the field definition for the Am29332:

```
Am29332:length (7)
  values (H'00' : ZERO-EXTA
         H'01' : ZERO-EXTB
         H'5F' : SMULFIRST)
```

The name of the field may be any sequence of characters. Constants may be specified in hexadecimal, decimal, octal, binary, or ASCII characters. Each of the 'values' definitions consists of a constant followed by a colon and a symbol that will represent the constant's value when assigned to the field.

Case Definitions

The case definition is used to describe multiple formats for the microinstruction word. A microinstruction may have different interpretations of certain fields, depending upon other fields. The case definition provides a way of making this form of differentiation formal. The specification is such that if the selector field has a specific value, only one of the alternate field definitions is valid and all the others are undefined.

The case statement is introduced by 'case' and followed by an optional field selector field name. Following this are one or more case entries. A case entry consists of a value or list of values of the selector field and a 'begin-end' block containing the description of the fields that are defined for this value.

The form of a case definition is as follows:

```
Case {<selector>} of
  <casevalue1> :begin
    <fielddescriptors>
  end;
  <casevalue2> :begin
    <fielddescriptors>
  end;
endcase;
```

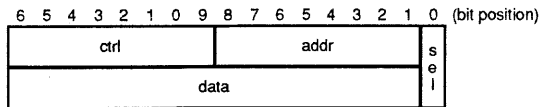
<selector> is an optional field that is set depending upon

CHAPTER 5
Support Tools

which case branch is selected. <casevalue1> is a value of the selector that selects the branch and is used for verification. <fielddescrs> is a field definition. An example is:

```
sel : length (1);
case sel of
  0 : begin
    addr : length (8);
    cntrl : length (8);
  end;
  1 : begin
    data : length (8);
  end
endcase;
```

This structure corresponds to the following overlaid microconstruction:



Macrodefinitions

Macrodefinition is a very simple language, consisting of

the field assignment. It is based upon the instruction definitions discussed above and is user-definable, depending upon any particular architecture.

All instructions are a sequence of phrases, each of which is either a field assignment or a macro call. The following is the form of macrodefinitions:

```
macro <op> &<var 1> &<var 2> .....;
  begin
    <fielddef 1>=<id k>,...,<fielddef i>
    =&<var j>
  endm;
```

<op> is a name of the macro. &<var j> is a macro variable that may be local to a particular macro or accessible by any other macro that defines the same global macro name. The following is an example for the Am29331:

```
macro call &dest;
  begin
    data=&dest, Am29331=CALL
  endm;
```

In this case, the Am29331 is set for a subroutine call instruction CALL and the microprogram branches to the address specified by &dest. Other conditions are default as given by the Am29331 instruction definition.

```
-----
;
;
;  AMDASM definitions for Am29114 Real Time Interrupt Controller
;
;-----
```

WORD 4

```
MCLR:          EQU      H#0      ; Master clear
CHSR:          EQU      H#1      ; Clear highest in service reg
CCIR:          EQU      H#2      ; Clear highest in interrupt reg
NOOP:          EQU      H#3      ; No operation
BSMK:          EQU      H#4      ; Set mask reg from D-Bus
BCMK:          EQU      H#5      ; Clear mask reg from D-Bus
LDMK:          EQU      H#6      ; Load mask reg from D-Bus
RDMK:          EQU      H#7      ; Read mask reg to D-Bus
BSSR:          EQU      H#8      ; Set in service reg from D-Bus
BCSR:          EQU      H#9      ; Clear in service reg fr D-Bus
LDSR:          EQU      H#A      ; Load in service reg from D-Bus
RDSR:          EQU      H#B      ; Read in service reg to D-Bus
BSIR:          EQU      H#C      ; Set interrupt reg from D-Bus
BCIR:          EQU      H#D      ; Clear interrupt reg from D-Bus
LDIR:          EQU      H#E      ; Load interrupt reg from D-Bus
RDIR:          EQU      H#F      ; Read interrupt reg to D-Bus

INT.CNTL:     DEF      4VH#3     ; Default to no operation
```

```

;
;
; AMDASM definitions for Am29331 Microprogram Sequencer
;
;

```

WORD 14

```

; Am29331 bit fields:
;           0      - FC- Force continue
;           1      - CIN          - Increment carry in
;           2-7    - I0-I5        - Instruction
;           8      - INTEN        - Interrupt enable
;           9      - OE           - D-Bus Output enable
;           10-13 - S0-S3        - Test select
;

```

; FC values:

```

FCONT:      EQU      B#1      ; Force continue

```

; CIN values:

```

CINCR:      EQU      B#0      ; Increment by one
CNINCR:     EQU      B#1      ; Don't increment

```

; Condition control (COND) (I4-I5)

```

TRUE:       EQU      B#00     ; Branch on true
FALSE:      EQU      B#01     ; Branch on false
ALWAYS:     EQU      B#10     ; Branch always

```

; Address source (ADDR) (I2-I3)

```

D.BUS:      EQU      B#00     ; Address source - D-Bus
A.BUS:      EQU      B#01     ; Address source - A-Bus
MULTW:      EQU      B#10     ; Address source - Multiway
STACK:      EQU      B#11     ; Address source - Stack

```

; Sequencer operation (SEQ) (I0-I1)

```

BRA:        EQU      H#00     ; Branch
CALL:       EQU      H#01     ; Call
EXIT:       EQU      H#10     ; Exit
DJMP:       EQU      H#11     ; Decrement counter and jump

```

CHAPTER 5 Support Tools

; Sequencer special instructions (I0-I5)

CONT:	EQU	6H#30:	; Continue
FOR.D:	EQU	6H#31:	; For D ...
DECR:	EQU	6H#32:	; Decrement counter
LOOP:	EQU	6H#33:	; Loop ...
POP.D:	EQU	6H#34:	; Pop stack to D
PUSH.D:	EQU	6H#35:	; Push D on stack
RESET.SP:	EQU	6H#36:	; Reset stack pointer
FOR.A:	EQU	6H#37:	; For A ...
POP.A:	EQU	6H#38:	; Pop stack to Counter
PUSH.C:	EQU	6H#39:	; Push Counter to stack
SWAP:	EQU	6H#3A:	; Exchange Ctr and TOS
STACK.C:	EQU	6H#3B:	; Push Ctr & Load Ctr D
LOAD.D:	EQU	6H#3C:	; Load Ctr from D
LOAD.A:	EQU	6H#3D:	; Load Ctr from A
BSET:	EQU	6H#3E:	; Load Comp Reg from D
CLEAR:	EQU	6H#3F:	; Disable Comparator

; Test conditions (S0-S3)

T0:	EQU	H#0	; Test T0
T1:	EQU	H#1	; Test T1
T2:	EQU	H#2	; Test T2
T3:	EQU	H#3	; Test T3
T4:	EQU	H#4	; Test T4
T5:	EQU	H#5	; Test T5
T6:	EQU	H#6	; Test T6
T7:	EQU	H#7	; Test T7
T8:	EQU	H#8	; Test T8 ==
CARRY:	EQU	H#8	; Carry
T9:	EQU	H#9	; Test T9 ==
SIGN:	EQU	H#9	; Negative sign
T10:	EQU	H#10	; Test T10 ==
OVER:	EQU	H#10	; Overflow
T11:	EQU	H#11	; Test T11 ==
ZERO:	EQU	H#11	; Zero or Equal
ULTB:	EQU	H#12	; C+Z Uns LT, borrow
ULT:	EQU	H#13	; ~C+Z Uns LT
LT:	EQU	H#14	; $N \wedge V$ - Signed LT
LE:	EQU	H#15	; $(N \wedge V) + Z - LE$

```
;
; Definitions for conditional sequencer operations
; (interrupts disabled)
SEQ:          DEF      B#0,B#1,2VB#11,2VB#00,2VB#00,B#0,B#1,4VH#0
;            FC CIN COND ADDR SEQ INTEN DOE TEST

; (interrupts enabled)
SEQI:         DEF      B#0,B#1,2VB#11,2VB#00,2VB#00,B#1,B#1,4VH#0
;            FC CIN COND ADDR SEQ INTEN DOE TEST

; Definitions for special sequencer operations
; (interrupts disabled)
SSEQ:         DEF      B#0,B#1,6VH#30:,B#0,B#1,4VH#0
;            FC CIN I0-I5 INTEN DOE TEST

; (interrupts enabled)
SSEQI:        DEF      B#0,B#1,6VH#30:,B#1,B#1,4VH#0
;            FC CIN I0-I5 INTEN DOE TEST

END
```

CHAPTER 5
Support Tools

```
{*****}
{
{      MCASM (Microtec Assembler)
{      Definitions for Am29323 32-bit Parallel Multiplier
{
{*****}

rnd:          length (1),          { Round control      }
              values (0 : inactive, 1 : active),
              default (inactive);

format:       length (1),          { Format adjust      }
              values (0 : fractional, 1 : signed),
              default (signed);

psel:         length (2),          { Output control    }
              values (0 : temp,      { Temp reg          }
                    1 : low,         { Lower half       }
                    2 : high,        { Upper half       }
                    3 : none),       { No output        }
              default (none);

acc:          length (2),          { Accumulator control }
              values (0 : pass,
                    1 : accum,
                    3 : shift),
              default (pass);

xsel:         length (1),          { Select X register  }
              values (0 : XB, 1 : XA),
              default (XA);

tcx:          length (1),          { X mode control    }
              values (0 : unsigned, 1 : signed),
              default (signed);

ftx:          length (1),          { Feedthru control for X regs}
              values (0 : registered, 1 : transparent),
              default (registered);

enx:          length (2),          { Load XA and XB regs }
              values (0 : both,
                    1 : XA,
                    2 : XB,
                    3 : none),
              default (none);

ysel:         length (1),          { Select Y register  }
              values (0 : YB, 1 : YA),
              default (YA);

tcy:          length (1),          { Y mode control    }
              values (0 : unsigned, 1 : signed),
              default (signed);
```

```

fty:      length (1),                { Feedthru control for Y regs}
          values (0 : registered, 1 : transparent),
          default (registered);

eny:      length (2),                { Load YA and YB regs      }
          values (0 : both,
                  1 : YA,
                  2 : YB,
                  3 : none),
          default (none);

tsel:     length (1),                { Temporary reg load select }
          values (0 : low,            { Lower half      }
                  1 : high),        { Upper half      }
          default (low);

ent:      length (1),                { Load temporary reg }
          values (0 : load, 1 : hold),
          default (hold);

eni:      length (1),                { Load instruction reg  }
          values (0 : load, 1 : hold),
          default (hold);

enp:      length (1),                { Load accumulator}
          values (0 : load, 1 : hold),
          default (hold);

fti:      length (1),                { Feedthru control for inst reg }
          values (0 : registered, 1 : transparent),
          default (registered);

ftp:      length (1),                { Feedthru control for accum }
          values (0 : registered, 1 : transparent),
          default (registered);

```


CHAPTER 5
Support Tools

```
{*****}
{
{      MCASM (Microtec Assembly)
{      Macros for Am29323 32-bit Parallel Multiplier
{
{*****}

{*****}
{
{      Load X Register
{
{*****}
macro      loadX      &X &mode;
begin
output      ("enx = &X, tcx = &mode");
end

{*****}
{
{      Load Y Register
{
{*****}
macro      loadY      &Y &mode;
begin
output      ("eny = &Y, tcy = &mode");
end

{*****}
{
{      Load Temp Register
{
{*****}
macro      loadT      &mode;
begin
output      ("ent = load, tsel = &mode");
end

{*****}
{
{      Select X & Y registers
{
{*****}
macro      selXY      &X &Y;
begin
output      ("xsel = &X, ysel = &Y");
end

{*****}
{
{      Multiplier function
{
{*****}
macro      mul      &A &mode;
begin
output      ("acc = &A, enp = load, psel = &mode, eni = load");
end
```

```

{*****}
{
{   MCASM (Microtech Assembler)
{   Definitions for Am29332 32-bit Arithmetic Logic Unit
{
{*****}

position: length (6),          { LSB Position or shift count          }
         default (0);

width:   length (5),          { Width of field              }
         default (31);

case of
0 : begin
    b_width:length (2),          { Byte width of data      }
      values (0 : four,
              0 : long,
              1 : one,
              1 : byte,
              2 : two,
              2 : short,
              3 : three),
      default (four);

    Am29332:length (7),          { Instruction }
      values (H'00': ZERO-EXTA,  { Zero extend A   }
              H'01': ZERO-EXTB, { Zero extend B   }
              H'02': SIGN-EXTA, { Sign extend A   }
              H'03': SIGN-EXTB, { Sign extend B   }
              H'04': PASS-STAT, { Pass status to Y }
              H'05': PASS-Q,   { Pass Q reg to Y }
              H'06': LOADQ-A,  { Load A into Q   }
              H'07': LOADQ-B,  { Load B into Q   }
              H'08': NOT-A,    { Not A           }
              H'09': NOT-B,    { Not B           }
              H'0A': NEG-A,    { 2's complement A }
              H'0B': NEG-B,    { 2's complement B }
              H'0C': PRIOR-A,  { Output priority A }
              H'0D': PRIOR-B,  { Output priority B }
              H'0E': MERGEA-B, { Merge A with B   }
              H'0F': MERGEB-A, { Merge B with A   }
              H'10': DECR-A,   { A - 1           }
              H'11': DECR-B,   { B - 1           }
              H'12': INCR-A,   { A + 1           }
              H'13': INCR-B,   { B + 1           }
              H'14': DECR2-A,  { A - 2           }
              H'15': DECR2-B,  { B - 2           }
              H'16': INCR2-A,  { A + 2           }
              H'17': INCR2-B,  { B + 2           }
              H'18': DECR4-A,  { A - 4           }
              H'19': DECR4-B,  { B - 4           }
              H'1A': INCR4-A,  { A + 4           }
              H'1B': INCR4-B,  { B + 4           }
              H'1C': LDSTAT-A, { Load A into status }
              H'1D': LDSTAT-B, { Load B into status }
              H'1E': undefined1, { RESERVED        }
              H'1F': undefined2, { RESERVED        }
              H'20': DN1-0F-A,  { A >> 1, zero fill }
              H'21': DN1-0F-B,  { B >> 1, zero fill }
              H'22': DN1-0F-AQ, { AQ >> 1, zero fill }

```

```

H'23': DN1-0F-BQ, { BQ >> 1, zero fill }
H'24': DN1-1F-A, { A >> 1, one fill }
H'25': DN1-1F-B, { B >> 1, one fill }
H'26': DN1-1F-AQ, { AQ >> 1, one fill }
H'27': DN1-1F-BQ, { BQ >> 1, one fill }
H'28': DN1-LF-A, { A >> 1, link fill }
H'29': DN1-LF-B, { B >> 1, link fill }
H'2A': DN1-LF-AQ, { AQ >> 1, linkfill }
H'2B': DN1-LF-BQ, { BQ >> 1, linkfill }
H'2C': DN1-AR-A, { A >> 1, sign fill }
H'2D': DN1-AR-B, { B >> 1, sign fill }
H'2E': DN1-AR-AQ, { AQ >> 1, sign fill }
H'2F': DN1-AR-BQ, { BQ >> 1, sign fill }
H'30': UP1-0F-A, { A << 1, zero fill }
H'31': UP1-0F-B, { B << 1, zero fill }
H'32': UP1-0F-AQ, { AQ << 1, zero fill }
H'33': UP1-0F-BQ, { BQ << 1, zero fill }
H'34': UP1-1F-A, { A << 1, one fill }
H'35': UP1-1F-B, { B << 1, one fill }
H'36': UP1-1F-AQ, { AQ << 1, one fill }
H'37': UP1-1F-BQ, { BQ << 1, one fill }
H'38': UP1-LF-A, { A << 1, link fill }
H'39': UP1-LF-B, { B << 1, link fill }
H'3A': UP1-LF-AQ, { AQ << 1, link fill }
H'3B': UP1-LF-BQ, { BQ << 1, link fill }
H'3C': ZERO, { Zeros to Y }
H'3D': SIGN, { -1 to Y if N == 1 }
H'3E': OR, { A or B }
H'3F': XOR, { A exclusive or B }
H'40': AND, { A and B }
H'41': XNOR, { A exclusive nor B }
H'42': ADD, { A + B }
H'43': ADDC, { A + B + carry }
H'44': SUB, { A - B }
H'45': SUBR, { B - A }
H'46': SUBC, { A - B - carry }
H'47': SUBRC, { B - A - carry }
H'48': SUM-CORR-A, { Correct BCD A for add }
H'49': SUM-CORR-B, { Correct BCD B for add }
H'4A': DIFF-CORR-A, { Correct BCD A for sub }
H'4B': DIFF-CORR-B, { Correct BCD B for sub }
H'4E': SDIVFIRST, { First step signed }
H'4F': UDIVFIRST, { First step unsigned }
H'50': SDIVSTEP, { Iter step signed }
H'51': SDIVLAST1, { Last step signed / + }
H'52': MPDIVSTEP1, { First step multi / }
H'53': MPSDIVSTEP3, { Last step multi signed }
H'54': UDIVSTEP, { Iter step unsigned / }
H'55': UDIVLAST, { Last step unsigned / }
H'56': MPDIVSTEP2, { Iter step multi / }
H'57': MPUDIVSTEP3, { Last step multi uns }
H'58': REMCORR, { Correct rem after / }
H'59': QUOCORR, { Correct quo after / }
H'5A': SDIVLAST2, { Last step signed / - }
H'5B': UMULFIRST, { First step unsigned * }
H'5C': UMULSTEP, { Iter step unsigned * }
H'5D': UMULLAST, { Last step unsigned * }
H'5E': SMULSTEP, { Iter step signed * }
H'5F': SMULFIRST, { First step signed * }
default (ADD);

```

end;

```

1 : begin
    pos_src:length (1),           { Source for position }
        values (0 : pins, 1 : reg),
        default (pins);

    wid_src:length (1),          { Source for width   }
        values (0 : pins, 1 : reg),
        default (pins);

    Am29332:length (7),          { Instruction   }
    values
        (H'60': NB-SN-SHA, { A << pos, sign fill } )
        H'61': NB-SN-SHB, { B << pos, sign fill } )
        H'62': NB-OF-SHA, { A << pos, zero fill } )
        H'63': NB-OF-SHB, { B << pos, zero fill } )
        H'64': NBROT-A,    { Rotate A up pos bits } )
        H'65': NBROT-B,    { Rotate B up pos bits } )
        H'66': EXTBIT-A,   { Extract A<pos> } )
        H'67': EXTBIT-B,   { Extract B<pos> } )
        H'68': SETBIT-A,   { A<pos> = 1 } )
        H'69': SETBIT-B,   { B<pos> = 1 } )
        H'6A': RSTBIT-A,   { A<pos> = 0 } )
        H'6B': RSTBIT-B,   { B<pos> = 0 } )
        H'6C': SETBIT-STAT, { STAT<pos> = 1 } )
        H'6D': RSTBIT-STAT, { STAT<pos> = 0 } )
        H'6E': NOTF-AL-B,   { Comp B field } )
        H'6F': PASSF-AL-B,  { Pass B, set Z flag } )
        H'70': NOTF-A,      { Comp A field, unalgn'd } )
        H'71': NOTF-AL-A,   { Comp A field, aligned } )
        H'72': PASSF-A,     { Pass A field, unalgn'd } )
        H'73': PASSF-AL-A,  { Pass A field, aligned } )
        H'74': ORF-A,       { A or B, unaligned } )
        H'75': ORF-AL-A,    { A or B, aligned field } )
        H'76': XORF-A,      { A xor B, unaligned } )
        H'77': XORF-AL-A,   { A xor B, aligned field } )
        H'79': ANDF-AL-A,   { A and B, aligned field } )
        H'78': ANDF-A,      { A and B, unaligned } )
        H'7A': EXTF-A,      { Extract field in A } )
        H'7B': EXTF-B,      { Extract field in B } )
        H'7C': EXTF-AB,     { Extract field in AB } )
        H'7D': EXTF-BA,     { Extract field in BA } )
        H'7E': EXTBIT-STAT, { Extract STAT<pos> } )
        H'7F': PASS-MASK); { Generate mask pattern }

    end;
endcase;

borrow:    length (1),           { Borrow mode   }
           default (0);

hold:      length (1),           { Hold status & Q}
           default (0);

```

CHAPTER 5
Support Tools

```
{*****}
{
    Macros for MCASM (Microtec Assembler)
    Macros for Am29332 32-bit ALU
}
{*****}

{*****}
{
    datasize - set data size for subsequent operations
}
{*****}
macro    datasize &sz;
        global  &dsiz;
        begin
            &dsiz = &sz;
        end

{*****}
{
    ALU      - set alu operation with fixed data size
}
{*****}
macro    ALU      &op;
        global    &dsiz;
        begin
            output ("b_width = &dsiz, Am29332 = &op");
        end

{*****}
{
    preg     - set position source to register
}
{*****}
macro    preg     ;
        begin
            output ("pos_src = reg");
        end

{*****}
{
    wreg     - set width source to register
}
{*****}
macro    wreg     ;
        begin
            output ("wid_src = reg");
        end

{*****}
{
    ALUv     - set alu operation for variable data size
}
{*****}
macro    ALUv     &op &pos &width ;
        begin
            output ("position = &pos, width = &width, Am29332 = &op");
        end
```

```

/*****
/*
/*      MetaStep (Step Assembler)
/*      Definitions for Am29325 32-bit Floating Point Processor
/*
/*****

enr:          length (1),          /* Load Register A   */
              values (0 : LOAD , 1 : NOP),
              default (NOP);

ens:          length (1),          /* Load Register S   */
              values (0 : LOAD , 1 : NOP),
              default (NOP);

enf:          length (1),          /* Load Register F   */
              values (0 : LOAD , 1 : NOP),
              default (NOP);

R_Select:    length (1),          /* R Source Select    */
              values (0 : BUS , 1 : F-Reg),
              default (BUS);

S_Select:    length (1),          /* S Source Select    */
              values (0 : S-Reg , 1 : F-Reg),
              default (S-Reg);

Am29325:     length (3),          /* FPU Instruction    */
              values (0 : PLUS, /* F = R + S */
                    1 : MINUS, /* F = R - S */
                    2 : MUL, /* F = R * S */
                    3 : 2MINUS, /* F = 2 - S */
                    4 : FLOAT, /* F = float R */
                    5 : INT, /* F = int R */
                    6 : DEC, /* F = dec R */
                    7 : IEEE, /* F = ieee R */
                    default (0);

round:       length (2),          /* Rounding Mode     */
              values (0 : NEAREST,
                    1 : DOWN,
                    2 : UP,
                    3 : ZERO,
                    default (NEAREST);

```

CHAPTER 5
Support Tools

```

/*****
/*
/*      Macros for MetaStep (Step Assembler)
/*      Macros for Am29325 32-bit Floating Point Processor
/*
*****/

/*****
/*
/*      Load R Register
/*
*****/
macro    loadr &src;
        begin
            R_select = &src, enr = LOAD
        endm;

/*****
/*
/*      Load S Register
/*
*****/
macro    loads ;
        begin
            ens = LOAD
        endm;

/*****
/*
/*      Load F Register
/*
*****/
macro    loadf ;
        begin
            enf = LOAD
        endm;

/*****
/*
/*      Do all 1 operand FPU operations
/*
*****/
macro    fpu &op &s ;
        begin
            Am29325 = &op, S_select = &s
        endm;

/*****
/*
/*      Do all 0 operand FPU operations
/*
*****/
macro    fcvt &op ;
        begin
            Am29325 = &op
        endm;

```

```

/*****
/*
/*      MetaStep (Step Assembler)
/*      Definitions for Am29334 Four-Port Register File
/*
/*
/*****

Wrt_enable_A:      length (4),          /* Write enable for port A */
                   values (H'0' : double,
                           H'8' : 3byte,
                           H'3' : high-word,
                           H'C' : low-word,
                           H'7' : byte3,
                           H'B' : byte2,
                           H'D' : bytel,
                           H'E' : byte0,
                           H'F' : none),
                   default (none);

OEA:               length (1),          /* Port A output enable */
                   values (0 : enable,
                           1 : disable),
                   default (disable);

A-write:          length (6);          /* A write address */

A-read:           length (6);          /* A read address */

Wrt_enable_B:     length (4),          /* Write enable for port B */
                   values (H'0' : double,
                           H'8' : 3byte,
                           H'3' : high-word,
                           H'C' : low-word,
                           H'7' : byte3,
                           H'B' : byte2,
                           H'D' : bytel,
                           H'E' : byte0,
                           H'F' : none),
                   default (none);

OEB:              length (1),          /* Port B output enable */
                   values (0 : enable,
                           1 : disable),
                   default (disable);

B-write:          length (6);          /* B write address */

B-read:           length (6);          /* B read address */

```


CHAPTER 5
Support Tools

```

/*****/
/*
/*      MACROS for MetaStep (Step Assembler)
/*      Macros for Am29334 Four-Port Register
/*
/*****/

/*****/
/*
/*      SrcA - select A register source
/*
/*
/*****/
macro      SrcA      &n ;
          begin
              A-read = &n, OEA = enable
          endm;

/*****/
/*
/*      SrcB - select B register source
/*
/*
/*****/
macro      SrcB      &n ;
          begin
              B-read = &n, OEB = enable
          endm;

/*****/
/*
/*      DestA - select A register destination and size
/*
/*
/*****/
macro      DestA      &n &size;
          begin
              A-write = &n, Wrt_enable_A = &size
          endm;

/*****/
/*
/*      DestB - select B register destination and size
/*
/*
/*****/
macro      DestB      &n &size;
          begin
              B-write = &n, Wrt_enable_B = &size
          endm;

```

5.4 MICROCODE DEVELOPMENT

5.4.1 Step Engineering 32-Bit Development Tools

Step Engineering offers an integrated set of powerful development tools for the design and development of microprogram-based systems. In particular, these development tools are well suited for use with 32-bit building block devices such as the Am29300 family of components from AMD.

For the 32-bit system designer, the MetaStep Language System provides a powerful and flexible language definition, design, and development system for the development of customized microinstructions and microprograms. An important feature of the language is the ability to support both high order language constructs and bit-vector level operations. In addition, comprehensive source level debug facilities are inherent in the language, with a link to the STEP-40 SDT hardware debug stations.

The STEP-40 SDT is Step's system-level development tool for Am29300 32-bit microprogram-based design. It offers a comprehensive array of hardware tools and user interface software that supports every level of the development task.

The MetaStep Language System

The MetaStep Language System from Step Engineering is a powerful new microprogramming tool for the programmer/designer who wishes to utilize microprogram-based devices such as the Am29300 family as well as the Am2901, the Am2910, the Am29116, and many other bit-slice or microprogrammable units. MetaStep is a full-featured and well-structured microprogram meta-assembler with advanced features that give the programmer great power and flexibility. Both an elegant high order and a powerful bit-level language system, MetaStep includes five interrelated language modules and an AMDASM-to-MetaStep translator program.

A unique feature of the MetaStep Language is the MetaStep QuickLearn Environment. This integral environment expedites the development and debug of microprograms by providing a menu driven, interactive program that gives the user instant access to a user-selected editor, a file display program, a directory listing, an automated definition file generator and the MetaStep assembler. This program lets the user easily generate a definition file, assemble a program, quickly move from an assembly error directly to the line in his source code that contains the error, correct that error and return to

assembly. With single keystrokes the user can select from a variety of options and move quickly from one programming environment to another.

These features can greatly increase the speed and accuracy of definition file and microprogram generation by eliminating much of the tedious, time-consuming and error-prone task of catching and correcting syntactical errors.

Unlike earlier, more primitive microprogram assemblers, the MetaStep language system provides both high level and low level programming constructs for the designer/programmer. For the hardware designer/debugger, MetaStep supports any "close to the hardware" programming style with total control of bit level field constructs. This is termed bit vector level coding. MetaStep is also the ONLY microprogram meta-assembler to support true source level debug when linked to a STEP-40 SDT system.

MetaStep supports a full range of macro instruction features that let the programmer easily and quickly take full advantage of the power inherent in devices such as the Am29332 ALU, the Am29331 Sequencer, the Am29334 Register File, the Am29C323 Multiplier and Am29325 Floating Point Processor.

This flexible language provides the ability to create complex high level language constructs specifically tailored to your application. These constructs can be of any complexity, up to and including those of a custom language compiler. Of particular interest is the ability to intersperse bit-level instructions freely among high order constructs. This allows performance-critical code to be hand-crafted and placed within high order assembly or even high level language statements.

Design rule constraint management, error checking, data field validation, user-defined warning messages, and automatic pipeline compensation mechanisms provide a rich, defensive programming environment that permits error detection at assembly time, rather than at debug or runtime.

MetaStep features include a free-form and position-independent syntax, informative listings of macro expansions, field assignments, default assignments, symbol cross references, and symbol table listings, automatic hardware-to-software bit position mapping, field checking facilities, pipeline delay facilities, constraint management, consumption of AMDASM code, 28 expression operators, close interface to runtime debug facilities, and generation of files that give runtime information in symbolic form. MetaStep also supports meta-disassembly.

MetaStep is presently distributed for use on five different types of systems: CPM/68K-based systems, MS/DOS-based systems, VAX/UNIX-based systems, VAX/VMS-based systems, and SUN UNIX-based workstations. Support for other operating systems will be added in the future.

The five MetaStep language modules are called the Definition Processor, the Assembler Processor, the Linker Processor, the Format Processor and the UDS or User-Defined Symbolics Processor.

The Definition processor is used to define a language for a given target architecture, field by field, with logical groupings where appropriate. The definition processor defines constraints over fields, groups of fields, and entire instructions. Included in the definition processor is the ability to define macroinstructions, constants, and variables only once, and to then make those values available to the entire language system.

The Assembler processor is a macro-driven, relocating and constraint maintaining microprogram assembler. It produces relocatable object modules, error, warning, and user-defined messages, and symbolic output for use by the linker and system debuggers.

The Linker processor generates absolute code as well as debug, symbol and structure tables from definition processor and assembler processor output files.

The Formatter processor takes the absolute object file output of the linker and extracts several different types of information. These include a binary output file loadable into a STEP-40 SDT development tool, a hexadecimal output file, a symbol file with user program global labels and addresses, and a debug file for on-line assembly/disassembly and source level debug.

The User-Defined-Symbolics processor automatically generates User-Defined-Symbolics or UDS files. This frees the debug engineer who wishes to perform debug functions at the source level from the task of redefining the symbolics of the language every time he does a re-assembly.

The AMDASM-to-MetaStep translator offers the ability to take current AMDASM assembly source code and automatically translate that source into a syntactic form that is accepted by the MetaStep assembler.

MetaStep can be configured to execute in two environments: the station model, intended for use on a STEP-40 SDT development station; and the no-station model, intended for use in environments that do not use the STEP development stations or MetaStep language system debug and symbol files.

Some of the more important features of MetaStep are:

- Free-form, non-positional keyword syntax
- Powerful macro facility
- Symbolic field names
- Data types such as strings, integer, and enumeration
- If and for assembler directives
- Case statements
- Recursive expression facility
- Attribute operators
- Modular programming support
- Design rule management
- Automatic pipeline delay compensation
- Relocatable object code
- Any order bit-to-field assignments
- Link to true source level debug
- Easy integration to hardware debug station
- Consumes AMDASM source code
- Fast (10,000 fields/minute) one-pass operation

MetaStep solves the problems associated with older positional microprogram assemblers, i.e., the difficulties in keeping track of fields and field values by rote and precise positioning, the lack of any value or error checking mechanisms, the lack of a link to a hardware debug system at the symbolic level, and the lack of any means of reconstructing backwards from the microword to the bit fields that comprise it.

MetaStep provides the non-positional capability to define fields in logical order rather than simply by microcode instruction address, and includes support for nested macros, case structures and keyword parameters. The following is an illustration of a partial MetaStep program. As can be seen below, MetaStep has the ability to support both bit vector and high level coding techniques. The upper program segment illustrates a field by field programming style that uniquely declares each pertinent field in the microinstruction word. The lower segment shows a second MetaStep example that uses only high level statements to perform the same operation! As can be imagined, utilizing high level language constructs

greatly eases the programming task. For convenience and power, the programmer can intermix low level and high level program statements and/or start his programming task with simplistic statements and then grow into more complex usages as his experience grows.

Two illustrative MetaStep program statements:

Should the programmer/designer wish to program at the bit vector level, a simple MetaStep bit vector level program could be written like:

```

    ....
    ....
;
OP116 = TORAA,
SRCDST = OR, REG = R1,
CTLYEN = YEN_L, CCMUX = T1,
2910INST = CONT, TCONTROL = N1,
JMPADR = WALK, DLE = DLE_H, OET = OET_H,
SRE = SRE_L, IEN = IEN_L,
OEY = OEY_L
;
    ....
    ....
    ....

```

A comparable MetaStep partial program using High Order Language or HOL constructs would look like this:

```

    ....
;
    ACC ← ACC OR R1
;
    ....

```

While the previous example illustrates the simplicity of using MetaStep, the microprogrammer may very well be more concerned with power and flexibility. Devices like the Am29332 are complex devices with powerful instruction sets. To best take advantage of their power, MetaStep can incorporate all of the possible configurations of an Am29332 instruction into one clear MetaStep instruction.

For example, there are numerous options available to the programmer on each Am29332 instruction. Fixed length and variable length instructions such as MOVES, SHIFTS, ADDS, SUBTRACTS, MULTIPLY/DIVIDES, offer several different source and destination locations depending upon the class of instruction. With MetaStep, a programmer need define each Am29332 instruction only once, using high level constructs such as the CASE directive to define all of the possible configurations of the instruction. Then throughout his program, he can utilize that definition with a simple high order instruction mne-

monic that takes into account all of the various complications associated with that instruction and data and source combinations.

In addition, he can prevent microprogramming errors by providing error checking conditions within the instruction definition, so that illegal conditions are flagged at the assembly level, not at the debug level.

In this way, the programmer can reduce a large and complex instruction set to a few easy to remember mnemonics. This frees the programmer to concentrate on the logic of his program. In this way, microprogrammers can quickly apply all of the power of the Am29300 family to his design.

MetaStep system components share a common database and utilize common control constructs. The definition processor provides the capability to define variables, a string facility that allows concatenation, and it supports cohesion operations as well as 28 expression operators. The definition processor's ability to nest macros, pass variables through macro expansions, and perform recursion makes it a powerful facility for creating custom languages.

Constraint management facilities include a check descriptor that may be utilized to test constraints on a single field, a case branch, an entire microinstruction, or between microinstructions. Most importantly, rules of the target architecture may be embedded in the language facilities to detect bugs at assembly time rather than debug time. This facility allows user-defined procedural-based design rules to be enforced.

With MetaStep, memory space controls allow code to be generated for not only multiple segments, but multiple memory segments. This allows a single program to generate code for modern architecture class machines such as Harvard class machines and data flow architectures that typically contain multiple program stores.

A significant advantage offered by MetaStep is that the database files generated from the definition, assembler and linker are common and provide a method to pass all language constructs to debug tools such as the STEP-40 SDT. This means that the STEP-40 development tools can now have the capability to use the language definition files and all symbol tables to create true meta-disassembly. Powerful source level debug can greatly speed the development of any microprogram design and, in particular, as microprogram-based systems increase in complexity, true source level debug is a necessity.

MetaStep Quick Reference

MetaStep System Overview

- Common system elements shared between MetaStep processors
- Five Processors
 - Definition processor
 - Assembler processor
 - Linker processor
 - Format processor
 - User defined symbolics processor
- AMDASM to MetaStep translator program
- COMMON ELEMENTS: All processors share data files and common structures.
 - Common syntax and semantics: include forms of names, constants, directives and legal and illegal value definitions.
 - Common directives include:
 - Source Control Directives,
 - Listings - forms control, summary information,
 - Include - source inclusion,
 - Format - listing headings, trailers, and control,
 - Flow-of-Control Directives
 - If - fully nested conditional control
 - For - repetitive conditional control statement
 - Macro facilities, including nested macro capability and parameter passing and expansion.
 - Specification of assembly time constructs,
 - Shorthand specification of logical groupings of assignments.
 - Generation of warning and error messages.
- DEFINITION PROCESSOR: accepts a definition of the target system architecture and development environment.
 - Micro-architecture description: by means of instruction/field formats.
 - Instruction directive: names the architecture and specifies instruction length. Maximum instruction length is 1024 bits.
 - Field Description: defines a field as a group of bits (not necessarily contiguous) that perform a common function. Each field must be given a field description.

A full set of field descriptors is as follows:

 - bits - define absolute bit locations of field in microinstruction
 - check - constraint check on assignment to this field

- complement - two's complement field value
- default - provide value when field is not assigned
- display - provide debugger and default radix information
- invert - one's complement field value
- length - specify length of field
- mask - truncate values to field length
- parity - this field is the parity field
- reverse - reverse bits in field
- valid - specify legal values for field
- values - specify symbolic values for field

VALUES, VALID, AND CHECK provide syntactic, semantic, and pragmatic verifications on a per field basis.

VALUES provide syntactic information indicating what are acceptable values for assignment to a field.

VALID provides semantic information, listing all the acceptable values for the field.

CHECK provides a way of examining assigned values in the context of other field values or other state information.

- The Case Definition: alternative field interpretations. A case definition can be specified for each field. It is a powerful mechanism for defining alternative bit values for overlapping fields.
- The Environment Description: allows the programmer to specify the development environment, with constraints on field values, sequences of microinstructions, and the relationship between field values.

Features include:

- bitMap
- macros
- EQU symbols
- variables
- Constraints are provided in three general ways:
 - Symbolic values
 - Case branch constraints
 - Check descriptors - The check descriptor associates a constraint macro with one of the following:
 - a single field
 - a case branch
 - the entire microinstruction
- Validations: numerous checks performed at definition time verify that field names and values in case branches are consistent.

- **THE METASTEP ASSEMBLER:** supports coding styles ranging from bit vector specification through high order language expression and each stage in between. Allows mixing of bit vector and HOL expressions during coding.
 - Instructions: a series of comma-separated phrases. A phrase may be a field assignment, a macro-invocation, or a flow-of-control directive.
 - Field Assignments: consists of field name, followed by an equal sign, followed by an expression.
 - Macro Phrases: a macro-invocation is a macro name, optionally followed by parameters. Macros may be nested.
 - Relocation Facilities
 - org
 - align
 - reserve
 - segment
 - entry
 - point
 - external
- **METASTEP LINKER:** combines all system elements into absolute code that can be loaded into ROMs or simulators. It also produces debug tables.
 - Directives:
 - load
 - name
 - locate
 - reserve
 - fill
 - mapPoint
 - analyze
 - set
 - parity
- **AMDASM TO METASTEP TRANSLATOR:** produces MetaStep source statements from AMDASM source statements.

The Step-40 SDT

The STEP-40 SDT is the premier hardware-based development tool for any microprogram development task. In particular, it offers a comprehensive system for the design and debug of Am29300-based systems. It offers in one integrated chassis all of the development and debug tools needed for such an effort. With high reliability cabling and interconnect technology, the hardware

chassis permits the plug-in addition of a wide range of distinct but interrelated hardware tools. An IBM-PC/AT computer system provides the human interface, mass storage, and I/O devices.

Key Features of the STEP-40 SDT:

- Fully supports 32-bit Am29300-based system development and debug.
- Supports other microprogrammed products such as bit-slice, ASIC, DSP, or VLSI.
- Completely integrated hardware/software development station.
- Powerful IBM-PC/AT-based microprogram support instrument.
- Supports MetaStep, the first true high level language for microprogram development with in-line bit vector level support.
- SOURCE LEVEL DEBUG available at all levels of hardware and software debug.
- Reconfigurable, ultra-reliable 10 to 70 ns writable control store supports up to 64K x 512-bit arrays.
- Real-time emulators for popular bit-slice AMD ALUs and sequencers.
- Logic state analysis with trace memory and sophisticated multi-level control.
- Performance analysis tools like histograms, timing analysis, access tracking and predicate analysis.
- Regression Test tools for design validation.
- Meta-Disassembly coupled with source edit, source management, version control, and on-line patch management.
- User-Defined Symbolics allows conditional disassembly of trace or any system data.
- Sophisticated, easy-to-use screen-oriented editor with pop-up help menus.

HARDWARE resources include writable control store modules with the widest range of speeds and widths; real-time emulators for popular bit-slice parts such as the

Am2910, and Am29116; logic state analysis trace memory modules with flexible clock and breakpoint control modules; a histogram/timing analysis module for performance analysis tasks; and high speed memory simulation modules for more than 450 popular ROMs, RAMs, and PROMs. With a powerful high speed bus and modular hardware design, the STEP-40 SDT presents no hardware limitations for designers utilizing the most advanced microprogrammed devices.

SOFTWARE tools include a sophisticated, easy-to-use, screen-oriented editor; a powerful turbo programmers environment for fast, error free program development and debug; MetaStep for superior high level and bit-vector level programming; User-Defined Symbolics for comprehensive on-line symbolic debug; Meta-Disassembly for true interactive symbolic debug with full access to MetaStep symbol tables; and performance analysis tools like histogram and time stamping, regression testing and automated test suite generation tools. The STEP-40 SDT is the first system to offer source level debug throughout the development and debug environment.

Because the STEP-40 SDT is an IBM-PC/AT based development station, it gives you the best of both worlds: a wide range of comprehensive hardware debug resources coupled with a fast, convenient and well-supported computer system. The IBM AT, in particular, offers the widest range of software support of any lab-based system in the industry. The IBM-PC/AT workstations have the power to match the STEP-40 SDT debug station. As intelligent hosts they can support advanced user interfaces and control the multiple hardware resources. In addition, system updates and new features can be added quickly thanks to the flexibility inherent in these standard workstations. As hardware needs change, the user need only add hardware modules to the STEP-40 SDT specialized hardware chassis.

Hardware Tools

Plug-in writable control store modules are available with flexible array configurations from 1K x 64 to 16K x 128 per module. Modules can be mapped into arrays of up to 64K x 512 bits in size. Access times vary from 70 ns to 10 ns (and even faster when RAM technology permits).

The Writable Control Store (WCS) is a dual-port memory accessible from either the STEP-40 SDT or the target system. Both ECL and TTL RAM are supported with the industry's most comprehensive array of memory emulation. Having up to 16K x 128 bits on a single WCS versus having many small boards connected with many cables, dramatically improves reliability and signal integrity. The

user can configure to meet his design objective without sacrificing reliability or performance. Further, the STEP-40 SDT can support up to 32 independent arrays controlled by either a single or multiple clocks.

Available Modules:

- WCS-64 is the fastest STEP WCS. It uses 10 ns ECL RAMs and connects to the target via address and data pods containing ECL to TTL translators. Organized by 1K x 64 or 2K x 32 bits.
- WCS-128 provides twice the density of the WCS-64 with 10 ns ECL RAMs. Organized in 2K x 64, 4K x 32, or 8K x 16 bits.
- WCS-256 and WCS-1024 provide even larger memories for applications with less demanding speed requirements. WCS-256 is configured as 4K x 64, 8K x 32, or 16K x 16. WCS-1024 is configured as 16K x 64, 32K x 64, or 64K x 16 bits. Interface circuitry matches exact user memory specifications.

LOGIC STATE ANALYSIS (LSA) - provides trace memory modules with sophisticated clock, breakpoint and trace control. With true conditional bit-mapped disassembler (User Defined Symbolics or UDS), the LSA provides real-time 3-way branching using a 54-bit matchword to trigger the 25 MHz or 50 MHz trace memory. Linkage is provided to the symbol table of the user's source code for access to symbolic debug information. Source code can be interleaved with trace samples for easy cause (microinstruction) and effect (traced sample) readability and comparison.

TRACE MEMORY is provided with either 4K (TM-256) or 16K (TM-1024) bits of real-time trace memory at speeds of 16 MHz, 25 MHz or 50 MHz. These memories act as a circular buffer storing the last 4K or 16K store samples. Store clock filtering extends the effective buffer depth substantially by filtering out unwanted samples. Triggering and sampling is controlled by the trace control module.

TRACE CONTROL modules include the sophisticated clock and breakpoint controls. With a screen editor display, the user can set up to five 54-bit (16 address, 32 data, and 6 external qualifiers) matchwords per level to qualify trace memory sampling. Up to 16 independent levels for trace triggering or breakpoint are possible, with each level allowing for three way branching on an IF, ELSE-IF, ELSE-IF basis. A delay counter can be used on each IF branch to count occurrences of the 54-bit matchword or store cycles.

IN-CIRCUIT EMULATORS permit real-time emulation of popular bit-slice circuits such as the Am2910, Am29116 and other popular devices. The user can directly observe the internal states of these chips as they execute his program. The user can examine and modify registers and stacks. Execution control includes single step, multiple step and run program commands. Multiple emulators can be simultaneously controlled from a single emulator control module. STEP in-circuit emulators will operate in real-time at the full rated speed of the emulated circuit.

MEMORY EMULATOR modules support a wide range of RAM, ROM and PROM devices. Over 450 popular memory devices can be emulated.

PERFORMANCE ANALYSIS modules provide the hardware support for software features like histogram and time stamping. Time analysis can be performed with 12.5 ns resolution. Histograms can be in absolute time or in microcycles for precise execution measurements. A 48-bit timer/counter permits continuous analysis over hours and days, not just seconds.

Software Tools

The STEP-40 SDT fully supports METASTEP, thus providing the world's first truly high level microcode development language in a fully integrated development station.

METASTEP QUICKLEARN PROGRAMMING ENVIRONMENT is a unique facility that speeds the development of MetaStep programs. The user can quickly switch from facility to facility without losing his place in his code. This is particularly useful during program debug and patch.

SOURCE LEVEL DEBUG is another unique capability of the STEP-40 SDT. With the MetaStep language as the foundation, a microcode-based project can be greatly speeded by utilizing symbolic information throughout the debug cycle. A truly interactive symbolic debug capability, source level debug permits on-line meta-assembly, meta-disassembly on-line, run-time editing at the source level, and directly readable displays.

All STEP-40 SDT commands can reference symbolic labels defined in MetaStep. Thus, the user need enter and define his labels only once. Later he can use them throughout his debug tasks without reentering or redefining them. This is a requirement for convenient debug of relocatable microcode. Other systems require that the user spend endless hours defining his symbolic information each time he reassembles his code. Source Level Debug also means that he can control his hardware debug resources using this symbolic capability.

User Defined Symbolics (UDS) provides complete display and control of microcode, trace data and emulator data. Any arbitrary digital word can be conditionally disassembled into any symbolic representation. Unlike older systems that merely allow permutation of some fields in groups of contiguous bits, UDS gives the user a general purpose bit mapping (binary to symbolics) capability unmatched by any other system. UDS has great utility in hardware trace situations.

META-DISASSEMBLER capability allows the source definition to be accessed by the debug process and provides the user the abilities of disassembling his source code in-line, assembling in-line, plus insertion of additional microcode.

PERFORMANCE ANALYSIS capabilities include histograms and time stamping.

HISTOGRAMS permit absolute time or microcycle analysis of your microcode execution. With a 48-bit counter, time analysis can be performed over days and weeks if necessary, not just seconds. This analysis can give you graphical information showing where code optimization can best help overall system performance.

TIME STAMPING includes a 12.5 ns resolution to easily measure time between captured system events and provides both absolute and relative time stamping in both time and microcycles.

QUALITY ASSURANCE TOOLS aid in reducing overall system costs and in rapid test development. These include access tracking, predicate analysis and MetaStep facilities for maintenance of source and version control.

REGRESSION TESTS such as AUTOSTEP provide the capability to generate, store and reuse system validation tests from design definition throughout the life of the product.

Hardware Specifications

6-Slot Mainframe:
6-user slots available per chassis
Expandable backplane

MetaMachines:
Up to 32 per mainframe, each with separate data, address and/or clock inputs.

Writable Control Store:
Total Address Space: 64K deep x 512 bits wide.

CHAPTER 5
Support Tools

Modules:

- WCS-64 - 1K x 64/2K x 32,
10ns or 15ns RAM speed.
- WCS-128 - 2K x 64/4K x 32/8K x 16,
15ns or 25ns RAM speed.
- WCS-256 - 4K x 64/8K x 32/16K x 16,
25ns or 35ns RAM speed.
- WCS-512 - 4K x 128/8K x 64,
10ns, 15ns, and 25ns RAM speed.
- WCS-1024 - 16K x 64/32K x 32/64K x 16
35ns or 70ns RAM speed.
- WCS-2048 - 16K x 128/32K x 64,
25ns, 30ns or 70ns RAM speed.

Simulation Pods:

ECL to TTL/ TTL to ECL conversion
TTL specifications
Unlimited number of arrays

Trace Memory:

Sizes: 4K x 64 bits or 16K x 16 bits
Number: up to 8 modules per trace controller.

Clock, Trace and Breakpoint Controller:

16-level, 54-bit match word, conditional trace and break supported.

Logic State Analysis Control:

16-states, comprehensive control through counters, timers, conditionals, triggers, and unlimited break-points.

Additional information about MetaStep, the STEP-40 SDT and other Step tools for developing Am29300-based systems is available upon request from Step Engineering. Please contact:

Step Engineering, Inc.
661 East Arques Ave.
P.O. Box 61166
Sunnyvale, CA 94088
(408) 733-7837
(800) 538-1750
TWX: 910-339-9506

5.4.2 Microtech Research mcASM Structured Microcode Assembler

The mcASM microcode assembler provides software support for the Am29300 family. A second generation Structured Microcode Assembler, mcASM was the result of a joint effort between Advanced Micro Devices and Microtec Research. Ten years of bit-slice and microcode assembler experience within both companies has been combined with the latest software technology to produce this advanced implementation of a relocatable microcode assembler.

Special support is provided for the variable formats found in the Am29300 family. This support is an additional benefit as it provides constraint management for the entire microcode word. New features make mcASM faster and easier to use than previous microcode assemblers. These features allow the programmer to concentrate on the target system algorithm, thereby achieving a more competitive target system.

mcASM Features

- Am29300 family mnemonic definitions included
- Hosted on VMS/VMS and PC/DOS
- PROM programmer, Microtec, AMD, and STEP output formats
- Relocatable code segments
- Overlay support
- Macros with keyword parameters
- Automatic selection of word format
- Keyword syntax
- Local symbols for each field
- Fields defined with non-contiguous or contiguous bits

Description

As a meta-assembler, mcASM is used to assemble source programs targeted for a user defined set of hardware. First, a model definition program, mcDEF, is used to define the target mnemonics and their corresponding bit patterns for the assembler, mcASM. Then, mcASM assembles the user's source program into microinstructions for the target.

This meta-assembler is optimized for microcode applications where very wide word widths (up to 1024 bits) are not uncommon. A library of pre-defined part definitions is included with mcASM for the Am29300 family and other

AMD microcode driven products to help the user quickly build the hardware definition file.

Four related programs make up the product: mcDEF, mcASM, mcLINK, and mcPROM.

A model of the target system is defined using the mcDEF definition language. The model is then compressed into a lookup table by the definition program, mcDEF.

The model lookup table allows the microcode assembler, mcASM, to translate the user's assembly language source code into microcode bit patterns that drive the target system. Object modules generated by mcASM are in a relocatable format. Thus, smaller, more manageable source files can be generated. These can be independently updated and quickly reassembled.

Relocatable object modules are linked together with mcLINK to form an absolute executable microcode program. The program may include overlaid segments to conserve target system memory. Four formats may be selected as the mcLINK output format. These include mcFMT, AMDASM, Microtec META29, and STEP Engineering GENHEX.

A fourth program, mcPROM, converts the linker output into PROM files that can be downloaded into a PROM programmer. DATA I/O ASCII format and BNPF format are supported.

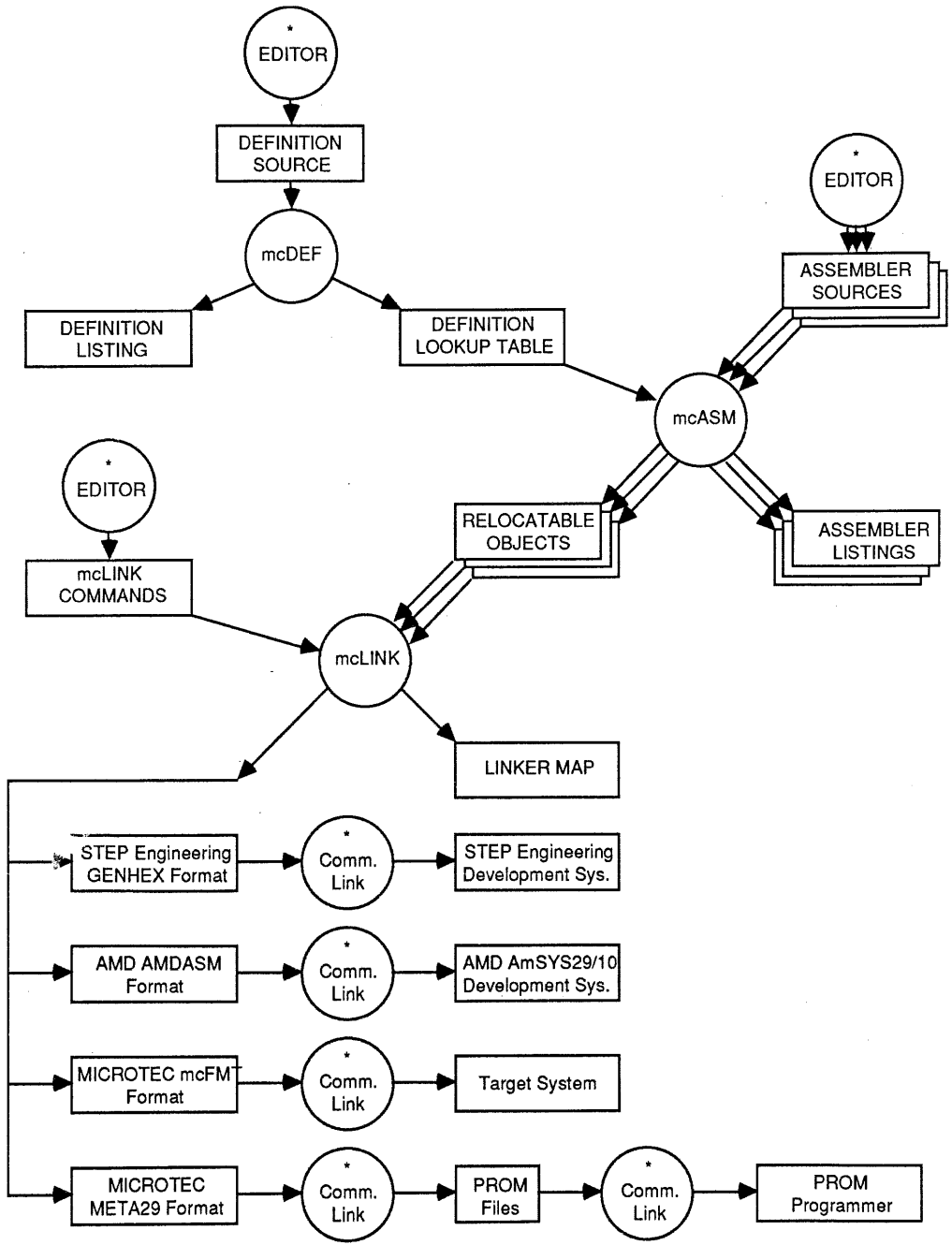
Figure 5-4 shows an overview of the mcASM development process and the following sections describe each component of the mcASM package.

mcDEF - Definition Program

The mcDEF definition program is a table builder that converts a model of the target hardware into a compact lookup table for later use by the assembler. The model is required by the assembler to describe how mnemonic names, used by the programmer, are converted into bit fields in a microcode word.

mcDEF accepts an input file that describes the field structure of the microcode word. Each field is independently described so it can be uniquely referenced by name in the assembly source code. The programmer can then directly reference any field and assign a value without having to put the value in a prescribed position in a source statement.

Each field can also be assigned a default value so that all fields do not need to be encoded in each line of source code. Mnemonics assigned a value for a field are local to that field. The same mnemonic can be assigned a different value in another field. A partial example of a processor model is shown in Figure 5-5.



* User Supplied Program

09372A 5-4

Figure 5-4. Overview of the Microtec Research mcASM Development Process

Sample Microword

Mem	MAR	Pos	Width	Am29332	Borrow	Hold	Data
-----	-----	-----	-------	---------	--------	------	------

Microword Definition

Mem: bit(40), length (1),
values (0:read, 1:write), default (read);
MAR: bit(38), length (2),
values (0:nop,
1:load,
2:enable,
3:ld-en);
Position: bit(32), length(6), default (0);
Width: bit(27), length(5), default (31);
Am29332: bit(18),
values(see file Am29332.def);
Borrow:bit(17), length(1), default (0);
Hold: bit(16), length(1), default (0);
Data bit(0), length(16), default (0);

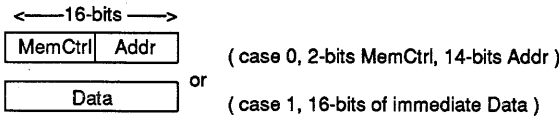
Figure 5-5. Sample Microword Organization

In some cases fields may overlap, resulting in several independent formats being defined for the same bits. mcDEF provides a structured case statement that describes each of the formats independently. This allows very simple selection of the required format within the assembly source code. Selection may be made by a

specific bit setting, use of a unique field name, or assigning a value unique to one of the cases.

A case statement demonstrating field overlaying is illustrated in Figure 5-6.

MICROWORD LAYOUT



mcDEF DEFINITION

```

case of
0:   begin           ( two fields )
      addr:          length(12);    ( address field )
      MemCtrl:       length(4);     ( memory control field )
    end
1:   begin           ( or one field )
      data: bits (16) ( immediate data field )
    end
endcase;
```

Figure 5-6. A Variable Format and Case Structure Definition

In the source program, the format is chosen by specifying 'data', or by specifying 'addr' and 'MemCtrl'. Any attempt to select both formats will result in an error at assembly time.

mcASM - Assembler Program

Source microcode is assembled by using mcASM, a structured microcode macro assembler that produces relocatable object modules as output. mcASM reads the source file and the model definition table as input. Each statement of source code is then converted into one or more microcode words as defined by the definition table. The output object module format is relocatable, thereby allowing separate modules to be linked into a larger executable program.

Microcode instructions are generated by assigning values to the fields that were defined in mcDEF. Assignment statements are used to assign values (i.e. fieldname = value), allowing the fields to be referenced in any order. Fields with acceptable default values do not need to be encoded. An example, using the model defined above, is shown below.

```
loop: Am29332 = INCR-A
      MAR = enable, Addr = fetch ;
```

Several features are demonstrated by this example.

- A single instruction can be continued on several lines without special notation.
- Field references can be grouped so that they refer to a common device or action. Fields with acceptable default values (such as Mem = read) do not have to be encoded.
- A reference to the Data field in the microword would generate an error because it conflicts with the case selection caused by the use of the Addr field.

An extensive macro facility allows the user to simplify the coding task by representing a large collection of field assignments with a single name and a few parameters. Macros also allow several microcode words to be generated with a single macro definition. The ability of mcASM macros to support assignment statements allows the user to define a higher level language that greatly reduces coding errors and coding time. For example, the instruction in the example above can be replaced with:

```
loop: ALU INCR-A;
```

where ALU is the macro name. The macro ALU assigns the parameter INCR-A to a variable field and fixes the values of the rest of the fields such as MemCtrl and Mem. Macros can also test the parameter values or names and then conditionally generate one of several outputs.

mcASM allows the programmer to structure microcode source into segments. Labels used within a segment are local to that segment allowing the labels to be reused in other segments with new values. Individual segments and collections of segments (modules) are separately assembled so that the whole program does not have to be reassembled for each change in source code.

mcLINK - Linking Loader Program

mcLINK collects the separate segments generated by the assembler and combines them into one executable program module. In addition, mcLINK supports generation of overlays that can be separately loaded into a common memory area.

Four absolute output formats are provided. Standard formats supported by mcASM include AMD AMDASM, STEP Engineering GENHEX, and Microtec META29. These three formats allow mcASM code to be used with existing development systems. A fourth format, called mcFMT, includes complete information for implementing overlays and performing symbolic debugging.

While the mcLINK program can generate separate overlay files in addition to the root program files in these three standard formats, a single file including overlays and symbol information is generated when the mcFMT output is selected.

mcPROM - PROM Formatter Program

Microcode is generally stored in PROMs in target machines. mcPROM is provided to divide the absolute linker output into separate PROM sized files. These files can then be downloaded to a PROM programmer through a user supplied communication package.

Program Features

The Microtec mcASM structured microcode assembler system has the following features.

Definition Program Features

- Microword lengths up to 1024 bits
- Variable formats, with multiple fields, predefined in cases statement
- Field definition attributes :
 - BIT - a field may start at any microword bit
 - LENGTH - total field length (max 16-bits) is specified
 - VALUE - local mnemonics are assigned to field values
 - VALID - only values in this list can be used
 - DEFAULT - the field is assigned a default value

Value modification operators :

- COMPLEMENT - uses two's complement of the value
- INVERT - inverts all the bits
- MASK - removes high bits to set size
- REVERSE - reverses the bit order

Definition program directives :

- TITLE - adds text string to top of each page
- INSTRUCTION - defines the width of the micro-word
- (NO)LIST - (does not generate) generates a listing
- (NO)OUTPUT - (does not generate) generates definition table
- (NO)XREF - (does not add) adds cross reference
- EJECT - advances listing to next page
- END - marks end of definition program

- ENTRY - lists all entry points to a segment
- EXTERNAL - lists all labels defined outside the file

Assembler directives :

- PROGRAM - names first segment and definition file
- EQU - assigns a constant to a name
- GLOBAL - defines variable available to all segments
- INCLUDE - adds additional source file inline
- ORG - sets location counter to new value
- TITLE - adds a text string to each listing page
- (NO)LIST - (does not generate) generates listing file
- (NO)OUTPUT - (does not produce) produces output file
- (NO)XREF - (does not generate) generates cross reference
- EJECT - advances listing to next page
- END - marks end of assembly source

Assembly Program Features

Symbolic addressing

Conditional assembly facility

Values assigned to field names

Powerful macro definition commands :

- MACRO - specifies macro name and parameters
- BEGIN - marks the start of the macro definition
- LOCAL - defines symbols local to this macro
- GLOBAL - defines symbols global to program
- OUTPUT - outputs source code
- IF - processes a statement if variable is true
- WARN - issues text string to output listing
- ERROR - sends text to listing, ends macro
- END - marks end of the macro definition

Flexible macro reference :

Parameter may precede macro name
(P1 macro_name P2)

Positional parameters are assigned values

Keyword parameters have default values

Relocatable output with multiple segments :

- SEGMENT - starts or restarts a user-named segment

Link Program Features

Combines independently assembled relocatable object modules

Resolves external references

Adjusts relocatable addresses into absolute addresses

Versatile user commands :

- LINK - loads specified segments from specified file
- ORG - changes value of location counter
- ALIGN - starts next segment at an address module n
- OVERLAY - starts and names an overlay
- SET - defines external symbols at link time
- TRANSFER - reads commands from another file
- END - marks end of command entry

Output listing controls :

- Load map - area and overlay name, base addresses
- Defined and undefined symbol references
- Optional symbol cross reference

Object module output in one of four formats
Microtec mcFMT with overlays and symbols
Microtec META29
STEP Engineering GENHEX
AMD AMDASM and AmSYS29

Conversion Utility Features

- Separates absolute file into PROM size modules
- Format is DATA I/O ASCII hexadecimal or BNPF
- Column overlaying
- Column switching
- Automatic parity generation

Minimum Hardware Required

Any Digital Equipment Corporation VAX System that operates under VAX/VMS. The software product typically requires 450K bytes of diskstorage after installation.

An IBM PC or compatible system that includes at least 512K bytes of total main memory and one (1) megabyte of disk storage. Typically the product requires 600K bytes of disk space for permanent installation with additional disk storage required for temporary files. Size of temporary files depends on the volume of user input.

Prerequisite Software

For distributions pre-installed for Digital Equipment Corporation computer systems, the appropriate VAX/VMS operating system.

For distributions pre-installed for IBM PC or compatible systems PC-DOS or MS-DOS versions 2.1 and newer.

Support Category – Microtec Research Supported

During the warranty period, Microtec Research Inc., provides the following standard services if the customer encounters a problem with the Software Product:

1. If Microtec Research determines the problem to be a defect in the software product, Microtec Research will provide remedial service by telephone if necessary (1) to apply a temporary correction or make a reasonable attempt to develop an emergency bypass if the software is inoperable, and (2) to assist the customer in preparing a Software Performance Report (SPR).
2. If customer diagnosis indicates the problem is caused by a defect in the software product, he may submit an SPR. Microtec Research will respond to problems reported in SPRs that are caused by de-

fects in the current, unaltered release of the Software Product via a newsletter. The newsletter provides notice of the availability of corrected code.

Any updates to this product released by Microtec Research during this warranty period will be provided to the customer on standard distribution media at prices specified in the prevailing Standard License Fee List. Non-standard media can be supplied upon request for an additional fee.

Service required because of customer use of other than the current, unaltered release of the Software Product operated in accordance with the Software Product Description (SPD) will be provided at Microtec Research's current rates, terms and conditions.

Ordering Information

All binary licensed software, including any subsequent updates, is furnished under the licensing provisions of Microtec Research's Standard Terms and Conditions of Sale. These terms provide, in part, that the software and any part thereof may be used on only the single CPU on which the software is first installed, and may be copied, in whole or in part, (with the proper inclusion of the copyright notice and any proprietary notices on the software) only for use on this CPU.

Refer to the Standard License Fee List for further ordering and media information or consult Microtec Research.

Software Product Service

Post warranty service for this product is available to licensed customers by purchasing a Software Product Service Agreement.

Full Documentation

Technical reference manuals are included as part of the software product. These manuals provide the information needed to use the software product and are written to be used in combination with the language reference materials provided by the manufacturer of the microprocessor. Manuals included are:

- Microtec mcASM User's Guide
- Microtec mcASM Reference Manual
- Microtec mcASM Installation Guide

For additional information contact:

Microtec Research, Inc.
3930 Freedom Circle, Suite 101
Santa Clara, CA. 95054
(408)733-2919

5.4.3 Hilevel Technology, Inc. Emulyzer and Hale

Hilevel's DS3700 Series Emulyzers provide full microcode development support for Advanced Micro Devices Am29300 Series building blocks. The DS3700 combined with HALE (an advanced retargetable Macro-Meta Assembler), with software for firmware integration and debug, and with a host computer provides a complete microcode development system.

DS Series Emulyzers

The DS3700 system employs an internal bit-slice architecture combined with ECL design to achieve high speed, decrease system latency, facilitate product upgrades, and implement unique features. The DS3700 range of features includes:

- HALE, an Advanced Macro-Meta assembler
- 10 ns WCS provides 25 ns access times at target
- 50 MHz logic state analyzer
- 50 MHz pattern generator
- Full software support for PC or VAX based operation
- Interactive source code debugging
- Source presentation of WCS and trace
- 16 level unrestricted triggering
- Microcode performance analysis
- User-defined display formats with bit permutation for both WCS and logic analyzer data
- Command language and command file execution of system operations
- Up to 512 bit wide WCS and trace

The DS3700 Emulyzer is available in three different configurations to accommodate varied Am29300 development needs:

- 1) as an integrated microcode development system connecting to an IBM-PC/XT/AT or compatible
- 2) as a stand-alone microcode development workstation connecting to your host computer.
- 3) as an Emulyzer using a VT100 compatible terminal providing memory emulation and logic analysis.

The Emulyzer can be remotely operated from virtually any host computer, over either the IEEE-488 or RS232 standard interfaces. A series of specific computer commands provides a high degree of Emulyzer control and programming flexibility, with provisions for rapid data transfer.

Writable Control Stores

The Writable Control Store (WCS) portion of the DS3700 Emulyzer is a high-speed memory which can be written to or read from by the DS3700 operator, the development workstation, the host computer, and your target machine. For RAM emulation, the microprogrammer may read and write to the WCS from the target processor. WCS memory options with access times of 25 ns at the target are ideal for high speed Am29300 operation.

A choice of fifteen different WCS memory modules are available to provide the user with a selection of speeds and densities to fill any microprogramming application. Memory boards are designed to optimize access times. All memory modules are 16 bits wide and are available in depths of 1K, 4K, or 16K. Modules may be configured in parallel for widths up to 512 bits.

The DS3700 Series can support WCS arrays up to 16K deep or 512 bits wide. Additionally, the WCS may be configured to support multiple arrays with each array configured for a unique size and speed.

Logic Analyzer

The DS3700 Series Logic Analyzer section is configured in 16 bit increments. Each increment may be clocked independently, or any number of these can be clocked synchronously. Trigger words may be defined across the entire trace width and qualified with ANDs, ORs, complement, and not equal. Up to 256 trace channels are available in a single chassis; however, chassis may be chained for greater widths. Either 4K or 16K deep trace memories are available at 25 MHz, 35 MHz, and 50 MHz.

Trace synchronization is nominally provided via selection of one of five clocks. Alternatively, each channel group (16 data channels/one clock per group) can be synchronized to compensate for clock delays, skewing, and multiple timebases. The DS3700 clocking scheme allows address (or data) to be delayed one clock cycle to align the address trace with its associated data.

Symbols for trace disassembly and triggering are automatically created by HALE (Hilevel's Assembler). Additional symbols may be defined and stored in the symbol table. The symbol table can be saved and restored for future use.

The DS3700 has four triggering modes.

Single Trigger: Single matchword defined across all address and data trace bits with don't care bits.

External Trigger: A hardware input may be programmed to act as a trigger, conditional trigger, or arming condition.

Multi-Level Trigger: Provides 16 levels of trace control with up to 4 conditions per level. Multiple commands (thirteen total) may be executed on the current clock cycle in real-time for any of the 4 conditions. Trigger patterns may be specified across the entire address and data fields including "don't care" bits.

Unlimited Break Points: Provides either 16K, 64K, or 1M of address breakpoints/triggers.

The DS3700 provides 16 active user-defined trace display headings and data formats. Any 4 bits of the trace data may be used to change display formats dynamically. In addition, symbols may be defined across the entire address and data fields and displayed along with the formatted data.

Trace masking is achieved by entering mask addresses in a table and then toggling the trace mask function on or off.

Trace permutations (as well as WCS permutations) are available to permute the order of display for clear presentations of the data.

During debug, using the Interactive Trace Disassembler with the DS3700 allows viewing of both the formatted trace with symbols and the related source code with comments.

Additionally, trace data may be displayed graphically as waveforms. Movement of linear cursors permit comparison of waveforms and viewing of timing information.

Microcode Performance Analyzer

The TIM-1E option provides an asynchronous clock for time-tag and performance analysis operations. Resolution of the clock may be set to either 15 ns or 250 ns in three operating modes:

Absolute Time: Allows elapsed time to be measured from any selected event; multiple reference points may be defined.

Time Interval: Provides a measurement of the time interval between adjacent trace data or any locations in the trace buffer.

Performance Analysis: Up to 15 groups of addresses may be defined as performance groups.

Performance groups of addresses can be defined to generate statistical performance analysis histograms, address vs. frequency of address and address groups vs. time spent in groups, to allow the engineer to measure firmware efficiency. For example, time spent in subroutines, interrupt handlers, and in arithmetic functions can be measured. Dynamic graphing is available to actually view the performance in real time.

Pattern Generator

The PG201 Option allows the Emulyzer to function as a digital stimulus response tester. Sequential or programmed vectors (or instructions) may be applied to the target and the response recorded. Using the Emulyzer Programming Language, the trace may be uploaded and compared to a known good file. The multilevel trigger may be used to set conditions for the pattern generator so that different vectors may be applied after a certain response has been recorded. The PG201 card also allows fast firmware-generated patterns to be inserted anywhere within the WCS. Walking ones, walking zeros, checkerboard, and random patterns may be merged with writable control store or used to fill the WCS. The PG201 may be used to emulate a controller, such as the Am29PL141, which controls or sequences the target hardware.

Hale - An Advanced Retargetable Macro-Meta Assembler

- Includes Am29300 Definition Files
- Increases User Productivity
- Allows Coding Optimization
- Pipeline Macros Ideal for Am29300 Blocks
- Assembles on Several Computers
- Relocatable Linkable Code
- Matched to Development System

HALE provides the microprogrammer with a set of facilities to rapidly create instruction sets and quickly write, assemble, and check his programs against design rules. For building custom instruction sets or emulating instruction sets, HALE increases programming efficiency and gets the job done fast.

HALE supports several programming techniques to accommodate varied programming styles and architectural requirements. Free-formatting, fixed-format instructions, position-independent code, macros, and pipeline macros each provide specific programming benefits. Techniques are often mixed in programs to provide the optimum control and ease of programming.

Am29300 programmers using HALE receive the benefits of an assembler that allows source presentations (your actual instruction), comments, and symbolic debug when used with a HILEVEL DS Series Emulyzer. These integration tools speed development.

HALE is easy to use and is a quickly learned assembler. Generating productive code with HALE begins within the first few minutes of use. Straight forward coding and simple definitions of powerful high-level macros permit code to be tested right away.

Pipeline macros allow the programmer to optimize the utilization of his hardware resources. By permitting macros for fields, combinations of fields, or along functional boundaries, and allowing multiple invocations of the macros while the earlier calls are still generating code allows highly overlapped, and compacted code to be written.

Pipeline macros are particularly useful for the Am29300 series since they are designed along functional boundaries. Pipeline macros written for the multiplier (Am29C323), a floating point processor (Am29325), and an arithmetic logic unit (Am29332) in an architecture combining these resources would allow tight control and economy of code for their independent and interdependent operations.

Pipeline macros are well suited for n-stage pipelined architectures, DSP algorithms, pipelined multiplier operations, and adding programming elegance. Once pipeline macros are written for an element, they are invoked and closed out with two simple commands. Up to eight pipeline macros can be operated simultaneously. Pipeline macros are position independent.

Calls to pipeline macros are limited only by the processing element's latency period, allowing maximum data flow processing. Pipeline macros also simplify coding for elements that introduce pipeline delays into the target hardware.

Pipeline macros may contain conditional assembly statements allowing the automatic selection of microcode sequences for a given operation.

User definable errors allow the microprogrammer to assert design rules and check his code against them. This saves time by catching errors during assembly rather than at debug and integration time. When microarchitectural constraints change, the program may be reassembled with new rules and checked against them. Instead of searching for potential errors, valuable time is saved by the automatic detection of errors.

User definable warnings allow the programmer to write non-assembling messages at any location in the source program. These messages may be used to follow assembly program flows or flag untested routines. Incomplete cases within macros may be detected by inserting a warning message as the last case. If an undefined case is called, the warning will be displayed. Warning messages assist the programmer in directing his attention to areas of concern and correcting them before they show up as problems during firmware integration time.

While and Endwhile looping directives allow code between these directives to be generated as long as a user specified boolean equation is true. While $A < B$, While

$A + B < C$, and While $A = B$ are examples showing the versatility of this directive. "While loops" may be nested up to 15 levels deep. "While loops" are also particularly useful in pattern generation applications.

ASCII statements convert ASCII code to its binary equivalent, which may then be imbedded within the microcode. Data may be coded directly into microcode in ASCII format. ASCII conversions are useful for passing messages, strings, or variables from one part of your target to another.

Macro facilities allow the assignment of a name to either a single microinstruction or to a sequence of microinstructions. Macros allow parameters to be passed to points within the macro body. A multiply macro may consist of 100 lines of code, yet may be invoked by a single call (i.e., Mult A,B.). Macros permit the generation of assembly language for your target or even higher level languages if one builds macros from macros. Macros may be nested up to 15 levels deep. Macros may call pipeline macros to generate extremely powerful code.

Conditional assembly statements can be used to generate high-order instructions that can accomplish a number of things based upon variable inputs: for example, executing either signed or unsigned functions, selecting the correct microcode for a specific task (automatic instruction selection), or interrogating the hardware and conditionally executing different microcode sequences (context switching). Conditional assembly statement allows the construction of powerful macros.

String facilities are used to identify variables and compare entire or whole portions of strings with each other. When combined with other assembly directives, different routines based upon the results of the compares can be invoked.

Expressions, operators, and modifiers allow versatile assembly program control. Addition, subtraction, multiplication, division, less than, greater than, equal to, and combinations thereof can be used to generate and modify variables. Other commands available include shifting, negation, modulo addressing, relative addressing, and absolute addressing.

HALE's PROM formatter outputs in HILEVEL ASCII, AMDASM, DATA I/O, and Intel Intellec Hex to adapt to your specific PROM programming needs.

HALE allows the linking of relocatable code so that several software modules may be developed in parallel, allowing completion of the programming task sooner.

Over 4000 source and definition symbols allow virtually unlimited amounts of code to be written. Word widths of up to 256 bits are supported accommodating highly parallel architectures.

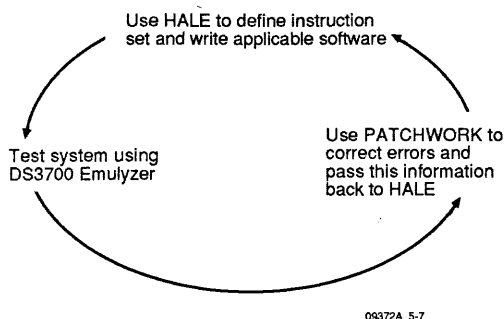


Figure 5-7

HALE runs on the IBM-PC/XT/AT, VAX, and Apollo computers. HALE runs all programs developed using AMDASM or Microtec Meta Assemblers, assuring the best possible return on your software investment.

Software Tools for Firmware Integration and Debug

Patchwork for fast effective microcode changes

Patchwork is an interactive assembler that permits the user to write the patches in assembly mnemonics and immediately test them. Temporary patches can be easily made and removed based upon the date they were made. Patchwork records each change, comments, date and time. Each change that creates new object code is appended to the listing and source files. In addition, a log file maintains a complete record of the entire editing session.

Alternatively, the user can utilize the object code editor in the DS3700 to make changes in the microcode residing in the WCS. In this mode, the WCS data is displayed in the same format as the HALE Macro-Meta Assembler object code listing.

Single-Step for tough debugging problems

The Single-Step program allows examination of the trace, source code, and comments together on a line by line basis. Each line shows what instruction was executed and what in fact happened. Using Single-Step, problems stand out and solutions often become apparent. Invoke patchwork, make the desired changes, and Single-Step again. For programmers writing code or maintaining it, the line by line comments allow quick recognition and interpretation of the instructions, thus reducing debug time.

Formatted Trace for full speed debugging

Formatted trace helps find errors that occur during real time execution. After a full speed run Formatted Trace allows stepping through the trace buffer presenting source code and comments together. This allows fast identification of problem areas, and points to instructions causing problems.

Trace Waveform for full logic analysis

Trace Waveform conveys a visual historical record of target board operation at a glance. It allows converting all or any combination of trace channels into timing diagrams. Labels may be assigned to each trace channel for clarity and recognition. A label file (containing the names of your traces) and a setup file (which holds parameters such as magnifications and scroll modes) can be created, saved and conveniently accessed in future uses. Cursor controls make comparison of non adjacent waveform edges easy. Channel order may be permuted.

Screen Driven

The Hilevel Emulyzer provides screens for convenient system set-up and operation. Each screen may be configured, saved and restored by the operator or by the Emulyzer Programming Language. The full range of Emulyzer operations are contained within the screens. For example, the writing of multilevel trigger programs, setting the logical analyzers breakpoints, running and tracing the microcode program, and analyzing microcode performance. Each screen is designed for maximum utility and optimum information display.

Automated Emulyzer Operation

EPL (Emulyzer Programming Language) automates the Emulyzer operation through the use of high-level commands. EPL permits the execution of command files that are used to setup the development environment (download the WCS, download multilevel trigger programs, download display format, etc.) and later save it. This allows multiple users fast and easy access to the development system while managing their files safely.

Microcode Quality Control

Microcode Quality can be assured by repetitive testing. EPL provides commands that allow looping, uploading trace data and comparisons against known good files. Using EPL, extended tests can be used to catch illusive program bugs.

System Software

Hilevel's system software allows the user to customize his development system. Keys may be assigned to invoke any program including HALE, EPL, Patchwork, Single Step, Formatted Trace, and Waveform. Often-used keyboard routines may be defined as keyboard macros and are invoked with a single keystroke.

In-Circuit Emulators

HILEVEL In-Circuit Emulators are available for a variety of microcoded processors and support devices. Emulation is accomplished by placing the target device in a socket on the appropriate emulation pod and plugging the pod into the device socket in the system. The pod is controlled by the EC1000 controller, which can accommodate up to four pods simultaneously. The EC1000 features a built-in keyboard and LCD display to support stand-alone operation.

The EC1000 may be connected to the DS3700 Development System, allowing the microprogrammer to control the Emulzyer and review data using the development system console. Using the EC1000 in concert with the development system also takes advantage of the DS3700's multi-level triggering capabilities.

All control and display capabilities necessary for comprehensive device emulation are designed into the EC1000:

- Decimal, Hex, Octal, Binary, ASCII
- Target single step or multiple step capability
- Displays registers whose contents match specified data
- Allows changes to any part of any register
- Allows control to be transferred to DS3700 or VT100 compatible terminal
- EEPROM allows customization of default parameters
- External trigger allows external logic or test equipment to halt the Emulator

Emulation pods currently offered by Hilevel for Advanced Micro Devices are the Am2910 sequencer, Am29116 ALU, and Am29PL141 Fuse Programmable Controller.

For additional information contact:

Hilevel Technology, Inc.
18902 Bardeen
Irvine, CA. 92715
(714) 752-5215
TLX 655-316

DS3700 SERIES SPECIFICATIONS

Writable Control Store (WCS)

Depth: 1K to 64K; depending on memory configuration.

Array Width: 0 to 512 bits in 16-bit increments.

RAM Speed: 10 ns to 120 ns; depending on memory module selected.

System Access Time: 25 ns to 140 ns; depending on memory module and pod selected.

Number of Independent Arrays: 16 maximum.

Target Control: Break (Halt), clear, single-step, continuous slow step, full speed emulation, break on event(s), PROM enable.

Editing Modes:

DS3700: Screen oriented editing with full search, scroll, page and window operation.

DS3700/CS: Full Interactive Source Code Debug.

WCS MEMORY MODULES: See following page.

WCS INTERFACE PODS

Logic Type: TTL, 10K ECL, or 100K ECL.

POD Types: Data, Address, Master Pods.

Output Signals:

Data Pods: 16 Data bits per pod.

Master Pods: 16 Data bits, clock enable, target reset, 2925 run control.

Address Pods: Clock enable, target reset, ROM enable, 2925 run control.

Signal Inputs:

Address Pods: 16 Address bits, clock input.

Master Pods: 16 Address bits, clock input, PROM enable.

Target Connection: Connector or PROM socket.

Type of Memories Emulated: ROM, PROM, SRAM.

Additional Support:

Registered Memories: Yes, with initialization.

Chip Select/Chip Enable: Up to 3.

Pod Size:

Data and Address Pods: 0.75" H x 2.75" W x 4" L

Master Pods: 1.5" H x 2.75" W x 4" L

Logic State Analyzer (Trace)

Number of Input Channels:

DS3700 Mainframe: 0 to 80 channels in 16 channel increments.

DT37XX Mainframes: 0 to 256 channels in 16 channel increments.

Maximum Clock Rate: 25 or 35 MHz; depending on type of trace memory selected.

TRACE MEMORY MODULES:

Model	Depth	Speed	Width
TRC/MLT-25	4K	25 MHz	16 bits
TRC/MLT-35	4K	35 MHz	16 bits
TRC16/MLT-25 16K	25 MHz	16 bits	

TRIGGER, BREAKPOINT AND TRACE CONTROL MODES

Modes: External trigger, single event trigger. Unlimited break/trigger (UBE option) and Multi-level trigger/trace control.

Mode Combinations: Any combination except single event trigger and multi-level trigger, can be used simultaneously.

(continued on following page)

DS3700 SERIES SPECIFICATIONS (continued)

External Trigger:

Input: BNC connector

Level: TTL

Active State: Negative going transition

Single Event Trigger: Single level condition specified across entire address and data fields.

Unlimited Break/Trigger:

Description: Address field can be used to specify trigger/breakpoint events for simultaneous monitoring.

Address Range:

Option	Range
UBE-16	16K
UBE-64	64K

Type of Trigger: Any address or address range may be specified as a trigger, conditional trigger or arming word.

Multi-Level Trigger/Trace Control:

Number of Levels: 16 independent levels.

Conditional Patterns: 4 per level across entire address and data fields.

Condition Formats: Bit patterns with user defined format, and symbols (user defined or assembler generated).

Boolean combination of symbols: Symbols may be combined with the following expressions: AND, OR, COMPLEMENT, NOT EQUAL

Multiple Action Commands: Up to 9 concurrent commands per condition

Action Commands: 13; as shown below.

1. Trigger
2. Conditional Trigger
3. Arm Trigger
4. Unarm Trigger
5. Reset Trigger
6. Disable Trace
7. Enable Trace
8. Override Trace Disable
9. Disable Trace Mask
10. Zero Timer
11. Jump to level <N>
12. Initialize loop/event counter
13. Assert Pattern Generator

Conditional Control

Loop/Event Counter: Up to 65,535 events

Trigger Delay: 0 to 4095 clock cycles

Breakpoints: Independent on/off control

TRACE MODES

Modes: State analysis; State timing, absolute elapsed time; State timing, Interval; Performance analysis and Dynamic performance graphing

State Timing (absolute and interval):

Resolution: 15 ns or 250 ns, selectable

Maximum Time:

Low Resolution: 16 minutes

High Resolution: 1 minute

Using Trace control: ≥16 hours

Performance Analysis (TIM-1E and UBE options):

Number of Groups: 15

Group definition: Any subset of the address range.

Address Range:

Option	Range
UBE-16	16K
UBE-64	64K

Operation: Logic analyzer stores group transitions.

Display: Both histogram and absolute time chart.

Histogram: Relative % of execution time used by each defined group.

Absolute: Total execution time of each group.

Group Name: Up to 15 characters.

Time Resolution: 15 ns or 250 ns, selectable.

Dynamic Performance Graphing

Number of Groups: 15

Group definition: Any subset of the address space.

Address Range: 64K

Operation: Logic analyzer dynamically updates trace memory and displays graph of percentage of events within each group.

Display: Histogram

SYMBOLIC TRACE

Description: Symbols may be defined using entire address and data fields.

Display: symbols will be displayed along with user formatted data.

Use: symbols may be used for trace display, trace control/trigger condition statements, search/locate operations, and time interval measurements.

Source: Symbols may be defined using DS3700 menu or downloaded from HALE definition files.

Maximum Characters per Symbol: 15

Maximum Number of Symbols: Depends on number of characters per symbol and width of data fields. ≥1000 symbols with average of 7 characters when defined on address field

TRACE MASK (UBE OPTION)

Description: Unconditionally masks from trace any user specified address or range of addresses.

Maximum Mask: Any subset of address range.

Address Range:

Option	Range
UBE-16	16K
UBE-64	64K

TRACE PODS

Logic Type: TTL, 10K ECL, or 100K ECL.

Signal Inputs: 16 data bits, clock.

Display Formatting

DS3700: Any user selected combination of hexadecimal, binary, and/or octal.

DS3700/CS: Full interactive WCS and Trace Disassembly.

Multiple Formats: Any 4 bits of each array and trace may be used to select between 16 user specified formats.

User Defined Headings:

Maximum number of characters: 256

Multiple headings: Up to 16 to match multiple formats.

Display Permutation: Any bit may be displayed in any position within WCS and Trace displays.

DS3700 Mainframe

WCS Size: Accepts up to 8 WCS memory modules (128 bits).

Number of Arrays: One

Trace size: Accepts up to 5 trace memory modules (80 channels).

(continued on following page)

DS3700 SERIES SPECIFICATIONS (continued)

Interfaces:

RS232: 3 ports

High Speed Parallel: 1 port

GPIB (IEEE-Std-488): 1 port (Optional)

BNC Inputs: External clock, external trigger

BNC Outputs: Arm output, trigger output.

Annunciation: Front panel LEDs show status of trigger, GPIB interface, clocks, and operational controls.

Interfaces:

RS232: 3 ports

High Speed Parallel: 1 port

GPIB (IEEE-Std-488): 1 port (Optional)

BNC Inputs: External clock, external trigger

BNC Outputs: Arm output, trigger output.

Annunciation: Front panel LEDs show status of trigger, GPIB interface, clocks, and operational controls.

DT37XX Mainframe

WCS Size: None, requires EXP3700 for WCS operation.

Trace Size: Accepts up to 16 trace memory modules (256 channels).

EXP3700 Expansion Chassis

WCS Size: Accepts up to 16 WCS memory modules (256 bits).

Number of Arrays: May be configured as one or two arrays.

Operating Specifications

(DS3700, DT37XX, EXP3700 chassis)

Chassis Size: 7" H x 18" W x 23" D

Weight: 60 to 70 lbs depending on options included.

Operating Temperature: 15°C to 35°C

Operating Humidity: 10 to 80 % RH

Power Requirements: 90 to 132 VAC, or 180 to 250 VAC; 50 or 60 Hz.

Warranty: 1 year limited warranty.

For additional information contact:

Hilevel Technology, Inc.
18902 Bardeen
Irvine, CA. 92715
(714) 752-5215
TLX 655-316

WCS MEMORY MODULES

Model	Depth			Emulation		RAM Speed (ns)	System Speed (ns)**							
	1K	4K	16K	PROM	RAM		25	35	40	50	90	140		
E1K-10	X			X		10	X							
M1K-20*	X			X		20		X						
M1K-35*	X			X		35					X			
E4K-10		X		X		10	X							
E4KW-10		X		X	X	10	X							
E4K-25		X		X		25				X				
E4KW-25		X		X	X	25				X				
M4K-25*		X		X		25				X				
M4K-35*		X		X		35					X			
M4K-120*		X		X		120								X
E16K-25			X	X		25				X				
E16KW-25			X	X	X	25				X				
M16K-35*			X	X	X	35					X			
M16K-70*			X	X	X	70						X		
M16K-120*			X	X	X	120								X

*M Series memory modules requires EXP370-4 expansion chassis.

**Access times specified at target side of pod.

5.4.4 Hewlett-Packard Microprogram Development Support

HP 64276 Microprogram Development Subsystem

Description

The HP 64276 Microprogram Development Subsystem and the HP 64320S 25 MHz Logic State/Software Analyzer provide run control and real-time analysis for the AMD Am29300 family. As integrated subsystems of the HP 64000 Logic Development System, the HP 64276 and the HP 64320S add the power of run control and analysis to all phases of the design, development, and maintenance of Am29300-based products.

The Microprogram Development Subsystem consists of three components: a Run Control module, a Writable Control Store (WCS), and a 25 MHz Logic State/Software Analyzer. Run Control provides program flow control, clock control, and break event detection. Writable Control Store provides high speed RAM for storing the microcode to be executed. A 25 MHz Logic State/Software Analyzer monitors systems buses and provides trigger, store, and sequencing functions for locating problems in the microprogram. Integration of the Microprogram Development Subsystem with other powerful HP 64000 analysis and emulation tools allow for interactive, cross-triggered measurements in complex multiprocessor environments.

Features

- The choice of clock control or real-time address jam at break detection offers flexible target system control.
- Address ranging and two-level sequencing provide powerful break event specification.
- Real-time, nonintrusive analysis of microprogrammed system activity reduces software development time.
- Flexible user-definable microassembler provides support for a wide variety of Am29300-based designs.
- Microcode source interleaved with analyzer trace data speeds software debugging.
- Linking of separately assembled microcode modules accelerates software turnaround time.
- MACRO instruction feature of the microassembler improves software engineering productivity.
- Modular architecture permits specific Writable Control Store configurations for customized development tool needs.
- Integration of Run Control and analysis capabilities simplifies operation.

- Interaction with other HP 64000 System Emulators and analyzers provides real-time analysis in multiprocessor environments.

Run Control

Run control provides system clock control, break event specification, and address jamming. These important features improve debugging of Am29300-based systems.

Architecture

The Run Control module taps into the clock lines on the target system to obtain the greatest level of clock control. Clock control functions allow you to start and stop the clock, single step, and break on a specific clock edge or pattern.

The Run Control module provides 20 I/O lines to probe the address bus, monitor status bits, or drive control lines. These I/O lines are bused internally to the Writable Control Store and the state analysis data probe connectors on the Run Control module.

Both single lead or coaxial cable leads are supplied for probing the clock and control lines between the target system and the Run Control module. Coaxial leads are recommended for use with higher clock rates to ensure better signal quality.

Clock Control

Precise specification of clock edges and relationships is critical for breaking or halting the clock in target systems with multiple clock signals. The Run Control Module allows you to specify complex clock signal characteristics for use in break events.

Address Jamming

Address jamming forces program execution at a specific address if a starting point other than a system reset vector location is desired. For example, to force the execution of a monitor routine that displays the registers, an address is jammed onto the address bus, causing the program to jump to the monitor routine. With the HP 64276 Microprogram Development Subsystem, you can jam either 8, 12, 16, or 20 address lines.

Break Events

The HP 64276 allows you to initiate a break event after the detection of any of the following occurrences: an address pattern (up to four can be specified), an address range, or a two-term sequence of an address pattern, range, or both. The state analysis trigger also can enable break event detection. When a break event occurs, an address can be jammed onto the address bus (e.g., to a monitor program) or the system clock can be stopped.

Writable Control Store

The Writable Control Store (WCS), the memory array for the system microcode, consists of a dual port RAM that allows easy microcode downloading from the assembly environment and high-speed access of the microcode by the microprogram target system. Target system development and debugging is more efficient using the WCS instead of the target system control store.

Architecture

The Writable Control Store (WCS) contains either one or two 32 kbyte memory boards. Each board can be configured into one of three array sizes: (bits wide by words deep) 16 by 16K, 32 by 8K, or 64 by 4K. With two WCS boards in the subsystem, the microword widths are doubled.

The WCS address is obtained from the Run Control module, eliminating the need to probe the target system a second time. By using one of the WCS address lines as an enable control to three-state the WCS output, you can toggle between target memory and subsystem memory.

Load

Once microcode has been assembled and linked, it is downloaded from the software development environment to the Writable Control Store for execution. Transferring microcode is fast and easy with the integrated development and hardware execution environments of the Microprogram Development Subsystem.

List

When debugging microcode, you can examine the contents of the WCS and list them to a destination file, a printer, or a display. A single list command specifies from one to four addresses or groups of contiguous WCS addresses. Displaying the address ranges allows you to examine and compare the microcode in different subroutines.

Modify

While debugging, you can modify the absolute code and continue debugging. Modify can be specified for up to 32 bits at a time for either a single WCS address or a range of addresses.

Save

The absolute code stored in WCS can be saved to a disc file for later reloading or for verifying the correctness of changes to source microcode.

User-defined

You can design a custom WCS array and combine it with the other modules of the Microprogram Development

Subsystem. The combination of the HP 64000 Logic Development System, the HP 64276 Run Control, and the user-defined WCS array provides an integrated development solution for all Am29300 microprogram target systems.

The user-defined WCS interface supports any array size between 16 by 512K and 1024 by 8K (bits wide by words deep). The interface between the HP 64000 mainframe and the user-definable WCS consists of control lines and parallel address and data buses that allow data to be written to or read from the WCS. User-definable control sequences can be transmitted to the user's WCS preceding and following an upload or download operation.

25 MHz Logic State/Software Analyzer

The HP 64320S 25 MHz Logic State/Software Analyzer adds high-speed, real-time, nonintrusive software analysis to the HP 64000 Logic Development System. This flexible analyzer works well in microprogram software analysis, general-purpose software analysis, and system integration. Measurement results are displayed in source microcode (including MACROs and comment lines) or in user-defined symbols that minimize the need to decode captured data. The analyzer can also reference symbols from the microprogram source files for easy specification and interpretation.

Architecture

The analyzer can be configured for 30, 60, or 90 channels of data acquisition. Each configuration must have a control card and from one to three data acquisition cards containing 30 data acquisition channels. The following table contains the analyzer's configurations.

Number of Input Channels	Control Cards	30-Channel Card
30	1	1
60	1	2
90	1	3

Format Specification

The Format Specification establishes the conditions and relationships of target system signals transmitted to the analyzer through the clock and data input channels. User-defined labels up to fifteen characters long can be assigned to signal groups from one to 32 contiguous channels wide. Saving the Format Specification to the disc eliminates respecifying data channel labels, threshold levels, and clock characteristics each time the analyzer is used. After a label is assigned to a group of input channels, it also appears on the analyzer softkeys.

To avoid confusion caused when both positive and negative true data are present in the system under test, the 25 MHz analyzer can automatically complement any group of data channels. You do not need to invert these signals on the target system or complement data as measurements are specified and results are interpreted.

The analyzer has two separate clock inputs. Data can be captured on the positive and negative edges of both clocks. With two clocks, you can analyze systems with multiple CPUs by capturing data on each processor's address strobe signal.

Data and clock signal switching threshold voltages can also be varied. Appropriate thresholds for TTL and ECL logic families have been preprogrammed. You can also select other values between -10 and +10 volts, in 100 mV increments for monitoring several different logic families. Independent threshold specifications can be made for each acquisition board (30 data channels).

Map Specifications

The Map Specification greatly simplifies measurement setups and trace data interpretation by replacing raw captured data with user-defined symbols. A "symbol map" can be associated with any labeled input channel via the Format Specification. Entries in a symbol map appear as part of the analyzer's softkey syntax and in the displays of measurement results. Map symbols are defined as constants, patterns, or ranges. A map symbol can be defined in terms of source file line numbers or user-symbols from microprogram source files.

Trace Specification

The Trigger function determines **when** the analyzer will capture data. Complex triggering conditions can be implemented using sequence terms. A "term" is defined as "AND'ed" constants and patterns. A constant can be an integer, map symbol, or symbol from the microprogram source file. A pattern is an integer with embedded "don't cares" (e.g., 0100xxxxB). Four sequence terms (trigger being the fourth) are available. Each sequence term can be set up to occur from 1 to 65,536 times before it is satisfied. A restart term is also available for resetting the sequencer.

The Trigger Enable function specifies **when** the analyzer monitors data for a trigger event. The trigger event can be stored anywhere within the trace memory buffer, allowing trace data to be stored either preceding, surrounding, or following the trigger event. The Store function determines **what** data should be stored. You can specify up to four OR'ed terms with each term consisting of

AND'ed constants and patterns. When the restart term is used for sequencing, the maximum number of OR'ed terms is three. The optional store with "sequence protect"

specifies that the sequence events be saved before any pre-trigger events are stored.

Measurement Results

The HP 64320S 25 MHz Logic State/Software Analyzer provides a high degree of display flexibility. When using source display, the microcode is visible without having to probe the microword: microword fields, MACRO invocations, and comments from source files are displayed. The display shows these source level statements combined with target data probed by the analyzer. This combination of program and data makes microcode debug more productive and efficient. Displays can also include user-defined symbols specified in the symbol maps and can automatically reference microassembler symbol tables generated during software development. These symbols can be displayed in the trace listings.

Flexible Probing Capability

The HP 64320S analyzer's clock cable and two of its data probes plug directly into the HP 64276 Microprogram Development Subsystem to eliminate double probing of the Am29300-based target system. Run Control, WCS, and the other state analysis data probes connect to the target system by general-purpose wire grabbers or D-type coaxial cables. The coaxial cables offer better high-frequency signal quality and a more reliable connection to the target system.

Measurement Involving Multiple Analyzers

Measurements with the HP 64320S and other HP 64000 analysis subsystems relate microcode execution to other software and hardware events. These interactive measurements are conducted via the high-speed intermodule bus (IMB). The IMB carries the following five signals between the analysis subsystems:

IMB Signal	Received by HP 64320S	Driven by HP 64320S
Master Enable	yes	yes
Trigger Enable	yes	yes
Trigger	yes	yes
Storage Enable	yes	no
Delay Clock	no	yes

The Master Enable signal coordinates measurement starts with other analyzer and emulators. When the analyzer is set up to receive this signal and the Master Enable is "false," the analyzer is completely disabled and will not capture data. When Master Enable becomes "true," the analyzer begins examining data.

The Trigger Enable operates in the same way as Master Enable by informing the receiving analysis module when it can begin looking for its trigger condition.

The Trigger signal, when received, causes the analyzer to immediately trigger and complete its measurement. For example, this is valuable for using the HP 64610S high-speed Timing/State Analyzer in conjunction with the 25 MHz Logic State/Software Analyzer to determine if a spurious signal pulse is related to a microcode event. By triggering the 25 MHz analyzer on a hardware event, the microcode execution surrounding the pulse is quickly pinpointed and evaluated.

The Storage Enable signal exercises hierarchical control over the store specification.

Microassembler

The HP 64276 Microprogram Development Subsystem includes a user-definable microassembler and linker capable of generating microwords up to 128 bits in width which support Am29300 family devices. The linker allows assembly of separate modules, reducing turn-around time for source microcode changes.

The definition language operates on a 32 bit, 40 register pseudo machine with standard instructions for the movement and manipulation of data. In addition, higher level commands for standard tasks are also provided (i.e., commands such as GET_TOKEN, FIND_DELIMITER, and GET_OPCODE support lexical analysis). The user-definable microassembler can also generate relocatable code with the use of the GEN_CODE command. The ERROR and WARNING commands print messages from a fixed table to the listing file to simplify error detection and correction. Field names and their values are easily specified (e.g., SEQ = CONT).

The definition language is powerful enough to allow the creation of a customized microassembler capable of:

- Generating code
- Specifying default values for missing fields
- Issuing errors for missing fields not having a default value
- Issuing errors for overlapping field definitions
- Issuing errors and warnings for architectural inconsistencies, such as a microinstruction that could cause bus contention

The resulting customized microassembler recognizes the syntax specified in the definition stage. Standard capabilities are predefined for the microassembler and need not be explicitly specified in the definition stage. For example, standard pseudo-ops are provided for storage allocation, location counter control, and listing format control. In addition, a powerful MACRO facility is supported.

5.5 SIMULATION MODELS

Logic Automation, Inc. Simulation Models for Hardware and Software Verification

The freedom and flexibility that have always been the benefits of designing with microprogrammed devices are now supported by a new generation of computer-aided design tools.

Advanced Micro Devices, Inc. and Logic Automation Incorporated have entered into a Library Development Relationship. This agreement has made it possible to model many of the latest AMD devices and make them available to designers. Table 5-2 includes all the Am29300 family.

Many other Advanced Micro Devices models are also available from Logic Automation; the entire AMD model list appears at the end of this section. These simulation models have been developed by Logic Automation with the cooperation of Advanced Micro Devices. Each model is based on information provided by AMD and verified

with the same vectors that are used to test the actual part. Each model is a SmartModel, capable of performing usage and timing checks that will significantly improve your ability to debug, verify, and optimize your designs.

SmartModel Simulation Benefits

Simulation models from Logic Automation are called SmartModels because they are behavioral language models with built-in intelligence. This concept—that information about VLSI devices is most effective when it is available inside the models used to simulate complex systems—was introduced and pioneered by Logic Automation. SmartModels allow you to use a workstation and logic simulator to verify your designs at the systems level.

Design cycles are shorter because the simulations catch many errors—both subtle and obvious—before the first prototype is built. Cycles are shortened because Smart-Model simulations are fast. They are easy to use and they are designed to maximize the effects of your simulation runs. Simulation runs are also critical as the first step in developing test vectors that must be used later to verify production systems.

Table 5-2

Description	TTL	CMOS	ECL
32-Bit Integer Multiplier		Am29C323	
Floating Point Processor	Am29325	Am29C325	
16-Bit Sequencer	Am29331	Am29C331	
32-Bit ALU	Am29332	Am29C332	
Register File	Am29334	Am29C334	Am29434
Bounds Checker	Am29337		
Byte Queue	Am29338		

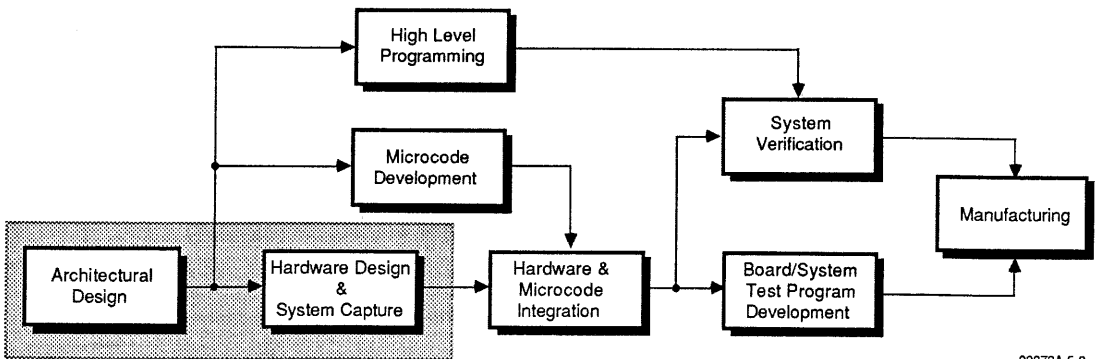
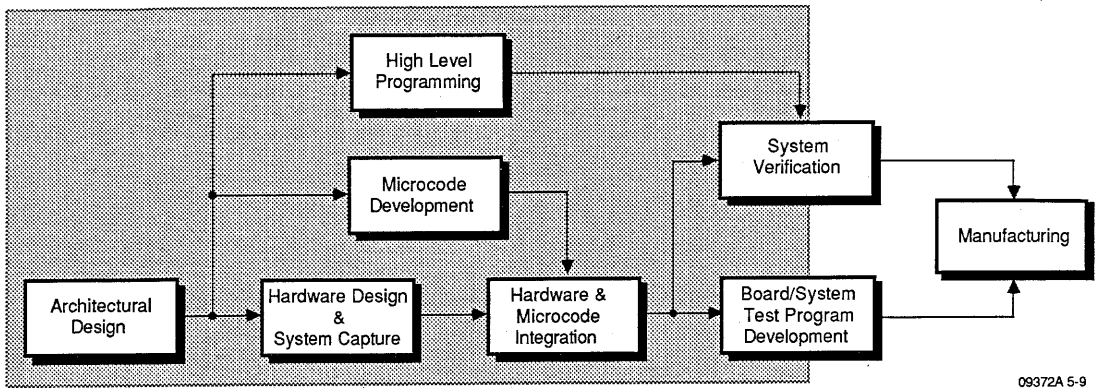


Figure 5-8. Microprogrammed Product Development Cycle (without simulating)



09372A 5-9

Figure 5-9. Microprogrammed Product Development Cycle (with simulating)

SmartModel Simulations Postpone Prototyping

Without simulating, the microprogrammed product development requires hardware prototype development very early in the process. As shown by the shading in the diagram's process blocks, Figure 5-8, only the overall design and hardware design (plus schematic capture) can be completed without breadboarding. Contrast this situation with the same process diagrammed in Figure 5-9.

Simulating permits far more of the product development cycle to take place before the first hardware prototypes are necessary. First of all, the simulation takes the place of the breadboarded hardware that would have been necessary for integration. In addition, short sections of code generated in a high level language using existing software development tools can also be executed in the simulation environment to help in the initial phase of system verification.

SmartModel Simulations Are Fast

Simulations with behavioral language models run fast. The demonstration circuit used below is a simple graphics processor designed using AMD's new 32-bit building block Am29300 family: the Am29331 sequencer, Am29332 ALU, Am29325 floating point processor, and two Am29334 dual-port register files. There are a total of 39 ICs in the schematic including 4 Am29827 10-bit buffers, 12 Am29841 10-bit latches, and 8 Am27S35 registered PROMs. In addition the design contains an abstracted behavioral language model of a display memory that is equivalent to eight SRAMs.

Figure 5-10 is a screen print of a simulation running under Mentor Graphics QuickSim 5.1. A timing diagram in a trace window occupies the width of the screen at the top. The QuickSim menu window is below left; next is

a list window showing a few of the circuit lines against simulation time. In the lower lefthand corner, there is a transcript window containing messages written by one of the Smart Models in the circuit. The lower righthand corner of the screen shows the schematic.

The circuit executes microcode out of ROM to plot the pixels that make up a line on a display. The pseudo-code for the line-plotting algorithm is below.

```

x, y, deltax, deltax <- FIFO (1,2,3,4)
e <- 2 * deltax - deltax
for i = 1 to deltax do begin
  plot(x,y){XOR in pixel(x,y)into bitmap}
  if e > 0 then begin
    y <- y + 1
    e <- e + (2 * deltax - 2 * deltax)
  end
  else
    e <- e + 2 * deltax
    x <- x + 1
end for
  
```

Run on an Apollo DN000 with Mentor Graphics QuickSim Version 5.1, the circuit ran through that algorithm executing the equivalent microcode at a rate of 34 microcode instructions per minute at a 1 ns resolution. Note that this was an exercise of the entire design, a true system-level benchmark.

SmartModels Are Easy To Use

SmartModel simulations are effective because these models are designed to make the most of every simulation run. For example, some users of simulation techniques have noted that analyzing computer printouts of logic values is tedious and very time-consuming. Using SmartModels eliminates that problem. During the initial stages the models' functional checks pinpoint usage

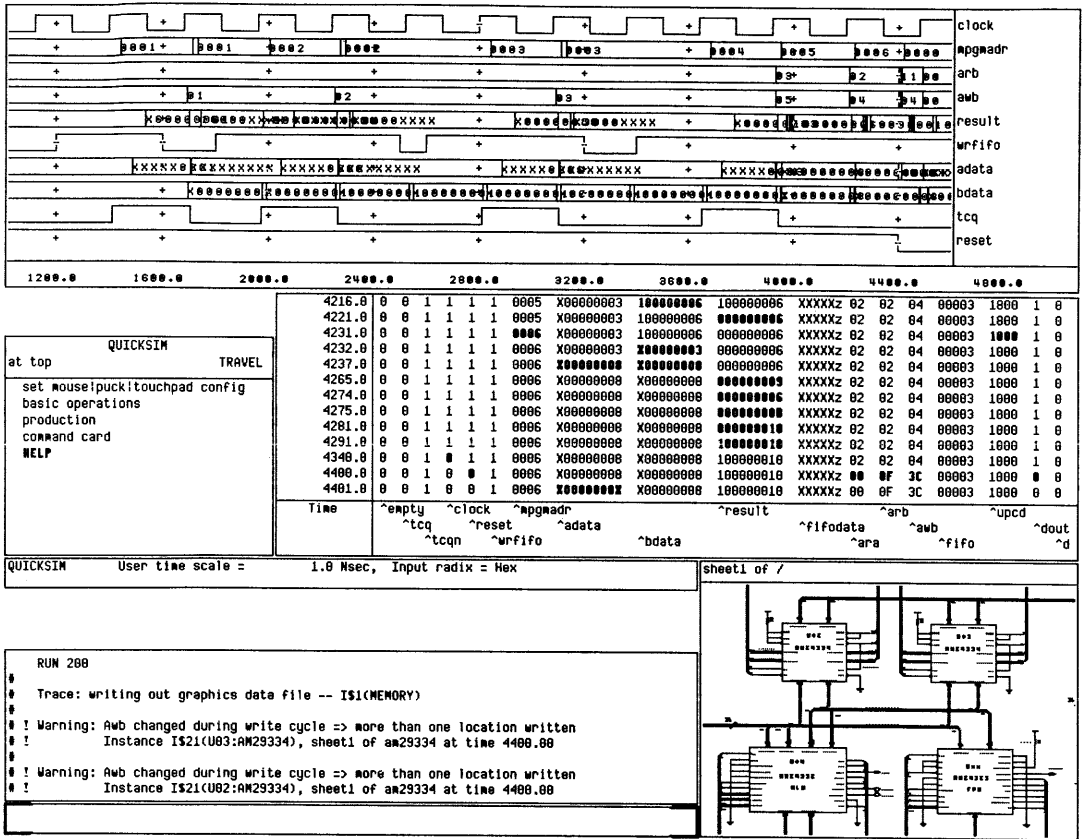


Figure 5-10

errors. Later in the design process, the timing checks are usually more pertinent. In both cases, the models use messages on the workstation screen to pinpoint the exact problem by time and schematic instance. This unique feature of simulation models from Logic Automation is called Symbolic Hardware Debugging.

Symbolic Hardware Debugging is a series of checks which write error or warning messages in the transcript window during your simulation runs. There are two types: functional checks and timing checks. The function checks vary greatly with the device type, but essentially

they help make sure a chip is being used correctly. For example, a DMA controller will include a check on whether or not all internal modes and registers were initialized. A DRAM check will produce a message like: "WE was low at the RAS falling edge."

The timing checks can include set-up, hold, frequency, pulse width, recovery time, etc., as applicable to the component and as specified by the semiconductor

vendor's current data sheet. A 1 megabit x 1 DRAM model, for example, contains about 50 different timing checks.

Both kinds of checks produce Symbolic Hardware Debugging messages that are very specific. A setup time violation, for example, will cause an error message that documents: pin name; device, by instance, reference designator, and component name; sheet name; design name; simulation time; signals and edges, as appropriate; and setup times, both as they occurred and as required by the vendor's data sheet.

Symbolic Hardware Debugging means your simulation runs give you answers, not just binary data which you have to painstakingly decode and compare to the IC data books.

Messages like that during your simulation runs speed your design debugging and verification. In this case, a check for an illegal operation has been built into the model; the operation can occur if the first instruction in an

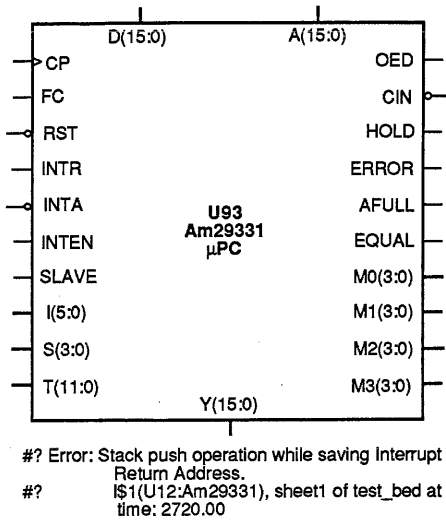


Figure 5-11. Symbolic Hardware Debugging in the AMD 32-Bit Building Block Family SmartModels

interrupt service routine is a stack operation. Besides a service routine that starts with a stack operation, this error message might be caused by an incorrect interrupt vector that caused a jump to any location that contained a stack operation. Similarly, the Am29334 SmartModel will signal if the write address changes during a write cycle; the model will issue a warning and write the data to all the locations involved so that the simulation run can continue. Many other function checks are built into these models. For the Am29300 family SmartModels, there are setup and hold timing checks for each input pin except the clock. For the clock, there are pulse width and frequency checks built into the models. Pulse width checks for the Write Enable and Data Latch Enable pins are also written into the Am29334 model.

SmartModels Make Your Simulations More Efficient

SmartModels maximize your simulations because they are adept at handling X's (unknowns). Depending on where it occurs in the circuit, one unknown can spread X's throughout your simulation. When that happens, your run is less useful than it could be because later events are buried in X's. To gain more information, you fix the first problem and rerun the simulation. SmartModels are designed not to generate or propagate X's unnecessarily—with Symbolic Hardware Debugging, the use of X's can be very judicious. Our engineers anticipate when an "X" is truly a "don't care" and keep your simulations useful

as long as possible while always issuing a warning message to document the event.

SmartModels Are Accurate

The Logic Automation and Advanced Micro Devices Library Development Relationship means that AMD supplies our model builders with advance information and with the test vectors used for the actual chips. We use the test vectors to certify that the SmartModels are accurate simulations of the AMD components.

SmartModels Represent Good Values

Multiple Timing Versions

Every SmartModel includes the correct timing for all available speed versions. An example is the Am29C323; the SmartModel for that part contains the Am29C323, Am29C323-1, and Am29C323-2 timing versions.

Maintenance

A maintenance agreement will keep your models current automatically. When CAE companies update their simulators and workstation operating systems, your models will be updated. Because Logic Automation works with the CAE companies prior to the new software release, you will generally have new SmartModels in your hands before you're ready to upgrade your system. If you have a maintenance agreement, Logic Automation will also automatically update your SmartModels when the manufacturer changes specifications or adds new timing versions.

Documentation and Support

SmartModels are very easy to install and use. Full documentation is provided with each set ordered. Included are: installation instructions; SmartModel Library Users Guide; data sheets on each model; and relevant application notes. In addition, our Applications Engineers are ready to help you with any questions at 503-690-6900.

SmartModels Are Available For Designs Now

Logic Automation has more than 250 timing versions of about 100 Advanced Micro Devices components that run on popular CAE workstations available now.

EPROMs

- Am27128A 16Kx8
- Am27LS191, included with Am27S191
- Am27PS191/A, included with Am27S191
- Am27S191/A/SA 2Kx8

PROMs

Am27S19/A 32x8
Am27S25 512x8
Am27S291A 2Kx8
Am27S35/A 1Kx8
Am27S37/A 1Kx8
Am27S45/A 2Kx8
Am27S47/A 2Kx8

Static RAMs

Am2130 1Kx8, dual port
Am2168 4Kx4
Am2169, included with Am2168
Am27519 64Kx1
Am9114 1Kx4
Am9124, included with Am9114
Am9128 2Kx8
Am9150 1Kx4
Am9151 1Kx4
Am91L14, included with Am9114
Am91L24, included with Am9114
Am93L422 256x4

Support

Am29114 real-time interrupt controller
Am2914 interrupt controller
Am2952 8-bit bidirectional I/O port
Am2953/A 8-bit bidirectional I/O port
Am2965 octal driver
Am2966 octal driver
Am8237A DMA controller
Am9513A system controller
Am9517A DMA controller
AmZ8073 system controller
AmZ8530 serial controller

32-Bit Building Blocks

Am29C323 32-bit multiplier
Am29325 floating point processor
Am29C325 floating point processor
Am29C331 16-bit sequencer
Am29331 16-bit sequencer
Am29332 32-bit ALU
Am29C332 32-bit ALU
Am29334 register file
Am29434 register file
Am29337 bounds register
Am29338 byte queue

Bit-Slice Family

Am2901B/C 4-bit slice
Am2902A carry/look-ahead
Am2903A 4-bit slice
Am2909 microprogram sequencer
Am2910/A microprogram controller
Am29116/A 16-bit microcontroller
Am2911A microprogram sequencer
Am2940 DMA address generator
Am2942 timer/counter/DMA address generator
Am29520 pipeline register
Am29521 pipeline register
Am2960 error detection and correction
Am29C10 microprogram controller
Am29L116, included with Am29116/A

Multipliers & ALUs

Am25S557 8-bit multiplier
Am25S558 8-bit multiplier
Am29C323, see 32-bit building blocks category
Am29332, see 32-bit building blocks category
Am29516 16-bit multiplier
Am29517 16-bit multiplier
Am29L516 16-bit multiplier
Am29L517 16-bit multiplier

Programmable Logic Devices

AmPAL18P8 PAL
AmPAL22V10/A PAL
Am29PL141 fuse programmable controller

Am29800 Family

Am29806 6-bit chip select decoder
Am29809 9-bit equal-to comparator
Am29818 shadow register/WCS pipeline register
Am29821/A/Am29C821 10-bit register
Am29822/A 10-bit register (inverting)
Am29823/A/Am29C823 9-bit register
Am29824/A 9-bit register (inverting)
Am29825/A 8-bit register
Am29826/A 8-bit register (inverting)
Am29827/A/Am29C827 10-bit bus buffer
Am29828/A/Am29C828 10-bit bus buffer (inverting)
Am29833/A/Am29C833 parity bus transceiver
Am29834/A/Am29C834 parity bus transceiver (invert register)
Am29841/A/Am29C841 10-bit bus interface latch
Am29842/A 10-bit latch (inverting)

Am29800 Family (continued)

Am29843/A/Am20C843 9-bit latch
Am29844/A 9-bit latch (inverting)
Am29845/A 8-bit latch
Am29846/A 8-bit latch (inverting)
Am29853/A/Am29C853 parity bus transceiver
(noninverting latch)
Am29854/A/Am29C854 parity bus transceiver
(inverting latch)
Am29861/A/Am29C861 10-bit transceiver
Am29862/A 10-bit transceiver (inverting)
Am29863/A/Am29C863 9-bit transceiver
Am29864/A 9-bit transceiver (inverting)

Models are added every week, so call to get the latest catalog or price and delivery information:

Logic Automation Incorporated
P. O. Box 310
Beaverton, OR 97075
Tel: (503)690-6900. Fax: (503)690-6906.

East Coast sales office:

Park View Office Building, Suite 400
10480 Little Patuxent Parkway
Columbia, MD 21044-3502
Tel: (301)740-8704.

5.6 C COMPILER SUPPORT

Introduction

With the advent of the Am29300 Family, it has become relatively easy to design bit slice systems controlled by very large amounts of microcode.

When it is expected that a fair amount of application microcode must be written, when speed of application development is important, or when some measure of portability is desired, then a microcode compiler can be an invaluable, if not essential, tool.

In this section, we discuss compiler implementations from two different angles. To begin with, we will discuss some of the decisions to be made when implementing a compiler for a specific architecture. Then we will discuss what hardware features are desirable to support the implementation of a compiler.

Before going any further, we should note that we do not believe that a microcode compiler can by itself provide a complete solution to the problem of writing code for bit slice systems. If you want to implement a general purpose language, you must design a general purpose processor. If you have not designed a general purpose processor, then it may be pointless to try to implement a compiler for your hardware. Even if your hardware is an ideal target for a compiler, there will inevitably be a need to code some small portion, at least, in assembler. In short, a microcode compiler is a tool, but not a panacea.

The Microcode C Compiler

The language we use is called *Microcode C*. It is similar enough to the C language that a programmer who already knows C can start programming in Microcode C after as little as one day's study.

The Microcode C compiler must be customized, which basically means that we have to write a code generator for your hardware, after making certain design decisions based on your needs and the capabilities of your hardware.

The compiler generates micro-assembler code as its output. If you already have a microcode assembler, then we can arrange to generate the mnemonics used by your assembler. Otherwise, we can generate code for Bit Slice Software's standard microcode assembler.

To date, we have developed about 12 different Microcode C compilers. These have variously been installed under PC-DOS, VMS, and/or Unix.

Types

All Microcode C compilers support a common data type - the signed integer whose width corresponds to the width of the processor. Typically, the width is 16 or 32 bits. Usually the types **short** and **long** are treated the same as **int**. Structures, unions, and arrays are supported, but sometimes with restrictions.

Other types are supported if desired and if the hardware permits. The type **char** can be reasonably supported if the basic memory architecture allows byte addressing. Since most microarchitectures use word oriented addressing, **char** is most often simply treated as **int**. The type **unsigned** can be supported if condition codes for unsigned comparisons are efficiently implemented. The types **float** and **double** are usually implemented only if there is floating point hardware to support them. However, they can also be implemented if software floating point routines are written.

Storage class

All Microcode C implementations support the storage class **static**. The **auto** storage class is only supported if the hardware allows a reasonable implementation of a run time stack. If it is not possible to support a stack, then local variables (which are normally allocated on a stack) are treated as **static** and recursive calls are not allowed. The **extern** storage class is supported if the assembler for which the compiler is generating code supports external references and definitions.

Most micro-programmers lay great stress on maximizing their use of the machine registers. Microcode C supports their desires by allowing them to declare variables with **register** storage class. *Microcode C allows registers to be declared globally*, as well as locally. Local register variables must be saved when a function call is made. Global registers never need to be saved or restored. They can be used to pass data between procedures in registers.

Initialization

The standard C syntax for static initialization of variables is supported.

Expressions

Each implementation supports all the standard C operations defined for its supported types. Binary operations supported include integer addition, integer subtraction, logical left and right shifts, bitwise and, bitwise or, bitwise

exclusive or, logical and, and logical or. Unary operations include take address, indirect through address, one's complement, logical negation, integer negation, and pre- and post-increment and decrement. Integer multiplication, division, and remainder are supported when the micro-architecture encourages them.

Statements

All of the standard C statement types are supported, including **for**, **while**, **do**, **go to**, **switch**, **if**, **else**, **break**, **continue**, **case**, and **default**. The **switch** statement will generate a jump table if the micro-architecture permits. The compiler also supports a **switchf** statement, which is like a **switch** except that it does not do a bounds check on the switch value before passing it through the jump table. Use of **switchf** instead of **switch** can save four or five micro-instructions if the switch value is known to be or forced to be in the range of the **switch**. For systems whose sequencers (such as the Am29331) have a hardware loop counter, the compiler supports a **loop** statement, which is very useful for coding fast inner loops. For Am29331-based systems, the compiler allows loop statements to be nested.

Built-in functions

Each micro-architecture has a unique interface to external buses, registers, and signals. Each Microcode C implementation supports this interface by providing a set of built-in hardware functions designed specifically for the particular implementation. These built-in functions behave like macros in that they are expanded in-line. A basic set of built-in functions might include:

```
data = input( source ); - gets data from an external
                        register
output( sink, data ); - sends data to an external
                      register
cc( condition_code ); - tests a hardware condition
                       code
memcycle( type ); - initiates a memory cycle
```

In this case, "source", "sink", "condition_code", and "type" would be chosen from a set of constants contained in a standard file supplied with the compiler. Any special timing constraints (such as "you must wait two cycles to read back data after cycling the memory") are enforced automatically by the compiler.

One of the advantages of using built-in functions, as opposed to adding new keywords to the language, is that it is possible to debug microcode programs on the host

system using the standard C compiler, simply by writing a small library of functions which are equivalent to the built-in ones and which simulate the operation of the target hardware.

Scratchpad RAM

In order to allocate non-register variables, there must be some sort of an external scratchpad memory accessible to the compiler. When reference is made to a non-register variable, the compiler automatically generates the micro-operations needed to set up the address and write out or read back the data.

Compaction

All microcode compilers must do some form of compaction in order to take advantage of the parallelism usually inherent in the micro-architecture. Microcode C uses resource-based compaction on straight line code segments. Operations are compacted in the order that they are generated by the compiler. An operation can be moved to precede a previously compacted operation if there is space for it and if no resource dependencies are detected while trying to move it.

In-line assembler code

If it is necessary to code key sections of a program in assembler, the compiler allows the user to include assembler code in-line. In order for in-line micro-assembler code to share data with compiled code, there is also a mechanism for in-line code to refer to register variables by the names they were declared with (rather than by number).

The overall aim is to provide a compiler which is inexpensive to build, simple and robust in construction, and can be relied upon to generate correct code. Although the compiler does take care of a great many housekeeping details (such as register number assignment and "constant folding"), it does not attempt to perform complex global flow analysis and optimization. Instead, the burden of doing so is placed on the programmer. Fortunately, the C language is designed to permit you to perform in source code the kinds of optimizations that optimizing compilers usually do. For instance, it is easy to recode array references in inner loops to use pointer operations instead.

There are many advantages to using Microcode C to write microcode. Programs are more readable, more comprehensible, and more maintainable. The use of a

high level language dramatically increases productivity and makes it much, much easier to try out different approaches during software development.

Hardware Design Considerations

If you are in the fortunate position of being in the process of designing new hardware and you want to know how to make it easy for a compiler to produce code for it, here are a few ideas.

ALU

To begin with, it is always nice if the ALU supports “three address code”, which means you can add register A to register B and place the result in register C in one instruction.

Second best, but also acceptable, is two address code, in which you add register A to register B and place the result in register B in one instruction.

In general, it is preferable for compiling purposes if any of the following can be accomplished in one instruction:

- add a register to a register
- move the contents of a register to a second register
- add a constant to a register

Although these would seem to be fairly simple things to do, it is suprising how many micro-architectures are unable to carry them out. You should not get the idea that it would not be possible to generate a microcode compiler for a given micro-architecture if it cannot perform the operations outlined above in one instruction. We recognize that many other factors, such as cost and board space, must be taken into account in your particular design and we are well aware of the dangers of over-specifying a design.

For two address architectures, you should try if possible to avoid putting any restrictions on the second address, such as “the upper two bits of the second address must be the same as the upper two bits of the first address”. Such restrictions can be worked around successfully, but they can be a rich source of bugs and are acceptable only if you are sure that the saving of a couple of bits in the microword will be worth all the trouble it will cause to both compiler writer and micro-programmer!

Constant Field

Most micro-architectures provide at least one constant field in the micro-instruction word. This field is set with constant data for the sequencer (jump addresses) or the ALU. This field should be at least as wide as the maximum of the sequencer address width and the data address width. In the best of all possible worlds, it should

also be as wide as the ALU and internal data paths. On a machine with a 32 bit ALU, it may be too expensive to reserve 32 microword bits for a constant field. One solution is to reserve only 16 bits and load all constants in two steps (load an upper data register from the constant field and then source the constant field combined with the upper data register). This solution can be made somewhat more satisfactory if it were also possible to treat the 16 bit data field as a 32 bit number in one or more of the following ways:

- zero extend the 16 bit constant on the left
- zero extend the 16 bit constant on the right
- sign extend the 16 bit constant on the left

Sequencer

In order to implement jump tables for SWITCH statements and to allow computation of addresses for indirect function calls, it is desirable if an address for the sequencer chip can be computed in the ALU. Typically this can be done by providing an external register which can be written to from the ALU's Y bus and then read into the sequencer using its “direct” inputs.

Similarly, if the sequencer contains a loop counter (as the Am29331 does), it would be nice if it could be loaded with an arbitrary value computed at run time in the ALU. This could be done using much the same mechanism as described above.

For branching within the microprogram, it is most desirable if there is a field in the micro-instruction which is big enough to hold the maximum microcode address. It should be possible to branch to an arbitrary microcode location in one micro-instruction. The address should be in one contiguous field of the micro-instruction. Although these ideas may seem obvious, we have seen several systems which ignored them. For instance, one system required the branch address to be loaded into a special register, with the actual jump in a subsequent instruction. Another system used a 4 bit “page register” with a 12 bit sequencer to address a 16 bit microcode address space. Although it was feasible to develop a compiler for both of these systems, the hardware design made all branches relatively expensive in the first case and all subroutine calls relatively expensive in the second case.

In order to achieve the maximum possible instruction rate, most systems are designed so that a conditional branch in one instruction is made based on condition codes computed in the immediately previous instruction. In some systems, all condition codes are latched in a register at the end of the first instruction, so that any one can be tested in the second. In other systems, the condition code to be tested is selected at the end of the

first instruction and only the one selected bit is latched, in order to save a couple of chips. A microcode compiler can be made to cope with either way of doing things, although the first is preferable.

In general, compiled code cannot always benefit from this pipelining of ALU and sequencer operations. A nice feature, which you might consider including in your design, would be to have an extra bit in the instruction which, when set, would cause the cycle length to be doubled. If the condition code were available halfway through the double cycle, then it would be possible to code a conditional test and a branch in the same instruction. Although this would not save any time, it would save on expensive microword space.

Floating Point

It is a relatively simple task to generate code for low latency parts, such as the Am29325.

Integer Multiplier

Multiplications are often generated by compilers during subscript calculations, if the size of the object being subscripted is not a power of 2. In order of increasing cost and speed, there are three ways to provide for multiplication in a bit slice design. The cheapest is to simply use the integer ALU to perform the standard shift and add algorithm, which costs one machine cycle per result bit (e.g. 32 cycles for a 32 by 32 bit multiplication). The next option is to provide a multiplier which can multiply address offsets, but not data, in one cycle. For instance, if the data paths were 32 bits, but the address width was only 16 bits, you could provide a 16 by 16 bit multiplier. This would take one cycle to compute a 16 bit offset, but would require four cycles to compute a 32 bit result. The fastest option is to use a multiplier, such as the Am29C323, which can handle either address or data calculations in one cycle.

Scratchpad Memory

In order to be able to declare non-register variables, there must be a memory somewhere to hold them. In most systems, this takes the form of a small, fast, local memory. In others, the bit slice processor uses memory on the main system bus.

If the memory is on the main system bus (a VME Bus or a Multibus, for instance), then it is usually a byte addressable memory. If your processor is to perform only word accesses on such a memory, then you might consider setting up the addressing so that the processor puts out a word address to the bus interface, which converts the address to a byte address. For instance, suppose the bus has 24 address lines. If you use byte addresses in the

processor, then any time some C code needs to do the subscript calculation

a[]],

it has to multiply the subscript by the size of the object being subscripted. Although, this multiplication can be converted into a shift if the size is 16 or 32 bits, this still imposes an unnecessary penalty for such a routine operation. A better scheme (for a processor whose word size is 16 bits) would be to use 23 bit addresses in the processor and have the bus interface in effect shift the address left by one and always supply a least significant bit of zero. For a processor which is 32 bits wide, you would use a 22 bit address in the processor, shift the address by two, and force the two least significant bits to zero.

Multiple Memories

One of the fundamental features of C is that it assumes that all memory accesses are identical and that a pointer can point to any addressable memory location. This makes it very tricky to support a system with memories with overlapping address spaces. For instance, if you have a pointer stored somewhere and you want to indirect through it, there are two problems. First, you must identify the memory in which the pointer is stored. Second, you must identify the memory to which the pointer points.

In most bit slice designs, the problem of overlapping address spaces usually comes up in one of two ways.

In the first and simplest case, memory address space overlap almost always occurs with control store memory and scratch pad memory. However, it is easy to tell which is which if control store memory contains only code and scratch pad memory contains only data (which may include pointers to functions in control store memory).

In the second case, the problem may arise if the hardware can operate on a host bus, such as a VME bus.

While it is conceptually possible to support an architecture featuring multiple memories of different granularities, the implementation of the concept would add a great deal of complexity to the code generator, because objects have different sizes in different memories. For instance a structure in one memory would have a different set of offsets to its members than the same structure in a memory with different granularity.

Usually, when Microcode C is implemented on a processor, one memory is picked to be the default system memory, as far as the microcode is concerned. All declared variables are stored in this memory. Space is also allocated within the memory for the run-time stack,

if that is required for the implementation. All addressing operations generate addresses in this memory. All indirect operations (including array/structure/union references) generate addresses within this memory.

Built-in Functions

Other memories (if any) are treated as peripheral devices and built-in functions are implemented to support them. For instance, a very common configuration might include a word-addressed 4K static memory and an interface to a byte-addressed VME bus. A Microcode C implementation for such a machine would designate the static memory as the main memory. The VME bus would be supported by a set of built-in functions, such as

```
set_vme_address( expr );
result = read_vme__bus(); /* at address */
write_vme_bus_byte( expr ); /* at address */
write_vme_bus_word( expr ); /* at address */
write_vme_bus_long( expr ); /* at address */
```

The disadvantage of this scheme is that it makes it impossible to use C structure references to refer to such external data. However, it does make it easier to support some of the more esoteric interfaces, such as those which support pre-fetching of data through FIFOs.

Addressing

In general, the ALU should be at least as wide as the memory address register of the main system memory. If it is not, then it is necessary to resort to either segmenting the address space or using very expensive double precision integer arithmetic for all address calculations. Neither of these two alternatives is very attractive!

In some micro-architectures, the main integer ALU handles all the work of generating memory addresses. In others, there is a separate functional unit, often featuring pointer and offset registers. These units are usually very effective for the special purposes for which they are designed but often lack certain fundamental functionality which is very useful to the C compiler.

The main deficiency, which we have seen in some systems, is the lack of the ability to generate an address based on taking a constant offset from a pointer register, *without* writing the resultant address back into the pointer register.

Given that MAR stands for "Memory Address Register" and that "constant" could be negative, the basic functionality which is desirable for the compiler would include

```
MAR = constant
MAR = arbitrary expression result
```

```
MAR = pointer register + constant
MAR = pointer register + arbitrary expression result
pointer register = constant
pointer register = arbitrary expression result
```

Note that this by no means excludes additional functionality, such as offset registers or multiple MARs. An actual hardware implementation could provide several variations on this scheme, such as providing operations in which a small constant is implicit in the operation, rather than having to be placed into a literal field. This allows certain memory addressing operations to be combined with operations which use the literal field.

To efficiently support pre-increment and pre-decrement operations we add

```
MAR = pointer register = pointer register + constant
```

To efficiently support post-increment and post-decrement operations, we add

```
MAR = pointer register
pointer register = pointer register + constant
```

with the sense that this is done in one operation.

The Stack

Since the stack pointer (SP) is simply a dedicated pointer register, all the operations on pointer registers described above also apply to the SP.

Most modern microprocessors reserve two registers to control the stack: the SP (which points to the top of the stack) and the Frame Pointer (FP) which points to the base of the current stack frame. The use of the FP allows a compiler to use stack offsets which are constant irrespective of how much has been pushed onto the stack (for temporaries or called function arguments).

In the interest of avoiding extra overhead on function entry and exit and at the expense of some extra internal housekeeping, the Microcode C compiler dispenses with the use of an FP and uses the SP only. The disadvantage of not keeping a separate FP is that the task of generating a stack trace back becomes much more complicated.

Bit Slice Software
321 Auburn Drive
Waterloo, Ontario, N2K 2X7
(519)885-4313
© 1987 by R. Preston Gurd

5.7 WRITABLE CONTROL STORE

5.7.1 Agility

AG-11B Microprogram Development

The AG-11B combines with your IBM personal computer to create a complete development station for microprogram-based designs. Its high performance and very low cost open new design opportunities for using flexible bit slice, ASIC, DSP, and 32-bit building block architectures. The AG-11B provides high speed in-circuit emulation of your design target's ROM or PROM.

Writable Control Store

The heart of the AG-11B is the Writable Control Store module (WCS) resident within your IBM PC. Each WCS has a memory array 96 bits wide by 4096 words deep which can be increased in width and/or depth with additional modules to suit virtually any size microprogrammed application. You microcode is loaded into WCS memory using your personal computer and AG-11B software. The WCS utilizes high-speed static RAM which provides a 50 ns maximum access time to your target.

Configurable Buffer Interface and Software

The AG-11B offers maximum flexibility in configuring for your particular design. The WCS interfaces to your target through the Target Interface Board. The hardware is complemented by the AG-11B software, which allows easy software control of your configuration variables. The AG-11B software, which is either menu-driven or command-line driven, provides control of breakpoint and target control signals and complete WCS card diagnostics.

mcASM Microcode Assembler

Included optionally with the Ag-11B is the mcASM Structured Microcode Assembler. Developed as a joint effort between Microtec Research and Advanced Micro Devices, this assembler features macro support, design

rule checking, nonpositional keyword syntax, and relocatable segments. mcASM lets you define your target's architecture and assembly mnemonics, and then produces executable microcode for your target in a format that is easily loaded into the WCS.

Applications

Microprogrammed architectures are increasingly used to boost performance in applications such as graphics, peripheral controllers, communications, military, robotics, and industrial automation. The AG-11B supports all architectures which use microprogramming, including bit slice as well as ASIC, DSP, and 32-bit building block devices. And since it is not designed for any specific architecture, the AG-11B is adaptable to any microprogrammed product.

Cost and Time Savings

The AG-11B:

- uses the computing power of an inexpensive IBM PC
- comes at a fraction of the cost of other microcode development stations
- is a cost-effective way to set up multiple development stations so that microcode development work can proceed in parallel
- lets you avoid the time and expense of burning new PROMs after each change to your microcode
- increases the productivity and morale of firmware engineers
- is available immediately and can be set up quickly and easily

For more information, contact Agility, 1290 Lawrence Station Road, Sunnyvale, CA 94089, (408) 744-0806.

CHAPTER 6

Articles/Application Notes

6.1 BIPOLAR BUILDING BLOCKS DELIVER SUPERMINI SPEED TO MICROCODED SYSTEMS	6-1
6.2 Am29300 DEMONSTRATION SYSTEM APPLICATION NOTE	6-12
6.3 THE FAST WAY TO BUILD A RISC PROCESSOR	6-92
6.4 FAULT-TOLERANT CHIPS INCREASE SYSTEM RELIABILITY	6-97
6.5 FLOATING-POINT MATH HANDLES ITERATIVE AND RECURSIVE ALGORITHMS	6-102
6.6 FLOATING-POINT ARRAY PROCESSOR IMPROVES COMPUTATIONAL POWER	6-109
6.7 FLOATING-POINT μ P IMPLEMENTS HIGH-SPEED MATH FUNCTIONS	6-116
6.8 OPTIMIZE YOUR GRAPHICS SYSTEM FOR BOTH 2D AND 3D	6-123
6.9 VARIABLE-WIDTH FIFO BUFFER SEQUENCES LARGE DATA WORDS	6-136
6.10 DIGITAL SYSTEMS VME 29300-1	6-141
6.11 BIBLIOGRAPHY	6-144

Bipolar building blocks deliver supermini speed to microcoded systems

As CMOS processes start to encroach on the performance of bipolar circuits, bipolar technology is taking the next step to keep itself in the lead for the highest speed systems. A family of five bipolar VLSI computational circuits—fabricated with a scaled,

ion-implanted, oxide-isolated process and three levels of metal interconnections for high density—provides a set of functionally partitioned microprogrammable VLSI building blocks for systems such as superminicomputers, digital signal processors, high-speed controllers, and many others. The modularity of the system functions ensures that the chips can meet the performance requirements of a general-purpose superminicomputer, as well as those of an image processor, which are radically different from each other.

Dhaval Ajmera, Ole Moller, and David Sorensen
Advanced Micro Devices Inc.

Since the beginning of last year, Dhaval Ajmera has been a design engineer in product planning at Advanced Micro Devices in Sunnyvale, Calif. He holds an MSEE from the University of Florida.

Ole Moller is also a design engineer in AMD's product planning operation. He holds an MSEE from the Technical University of Denmark.

Another engineer in product planning, David Sorensen specializes in programmable processors. He holds a BSEE from Arizona State University.

Included in the family are three parts that form the core of a general-purpose microprogrammed system: a 32-bit arithmetic and logic unit (ALU), a 16-bit microprogram sequencer, and a 64-by-18 four-port, dual-access RAM. And, for systems that do a large number of multiplications or floating-point

operations, two performance accelerators—a 32-by-32-bit multiplier and a 32-bit floating-point processor will be available to tie onto the buses (see Design Entry, p. 246).

The chips offer high performance, a flexible architecture, and microprogrammability, and even address the problem of fault detection for data integrity. These circuits can thus support an extremely fast microcycle—about 80 ns (projected). That high speed is the result of several design considerations: Each part is designed internally with emitter-coupled logic but has TTL-compatible inputs and outputs. Second, more power was allocated to the logic circuits used in the critical paths than for logic in the noncritical paths on each chip, to maximize the speed. Third, by integrating highly specialized logic on chip it is possible to execute very complex operations in a single cycle.

The microprogrammability of this chip set offers several benefits to the system designer. It provides a structured and systematic approach for implementing the control mechanism of the system, and like the bit slices, it allows the instruction set to be customized to suit the designer's application (see "Architectural Limitations of Bit Slices," opposite). And several versions of the initial design can be tested, or current designs can be enhanced simply by changing the microcode.

Thus, the functionally partitioned Am29300 family overcomes all of the performance penalties of bit-slice structures, while maintaining its ability to form a wide variety of architectures. Even though the chips are designed to work together as a family, each can also be used independently in an application that requires its unique capabilities.

Pipelines are out

The flexibility of the Am29300 family is largely due to a decision not to place pipeline stages within the functional blocks. Not including the pipeline registers inside incurs some off-chip delays. This is a small price to pay to allow system designers to optimize the pipeline structure for their individual needs. Moving the register file out of the functional block for the ALU also slows things down. At the same time it does not force a fixed register size on the user, enabling systems to be created with dedicated

registers, register windows, or register banks—all with neither fixed depth nor width.

Additionally, the high level of integration helps eliminate the propagation delays often encountered when signals must go from chip to chip. The use of VLSI also results in fewer parts at the system level, which, in turn, conserves power (usually many watts in the case of bipolar systems) and board space. Lastly, a complete 32-bit solution is provided for applications that require increased precision for arithmetic operations, high memory bandwidth, and a

Architectural limitations of bit slices

The limited performance of bit-slice circuits can be improved by increasing the width of the slices. That higher level of integration results in higher performance by reducing the number of off-chip delays while preserving the flexibility that has made bit-slice systems so attractive. However, as higher levels of integration become possible, two inherent problems with bit-slice architectures will limit their ultimate speed. The first involves the off-chip delays inherent in cascading. For example, the carry chain is usually the slowest path of an ALU. Breaking this chain between slices introduces off-chip delays into the critical path.

The second problem is that the functional needs of many systems do not slice well. Barrel shifters and prioritizers are especially difficult to cascade. Unfortunately, the ability to perform N-bit shifts and locate the position of leading 1s are of greatest importance in applications that require heavy number crunching and manipulation of data fields, such as image processing, graphics, database management, and controllers. These are precisely the applications whose need for speed forces the use of bit-slice devices. The system performance is compromised not only because these operations must be done bit by bit, but also because many high speed algorithms cannot be efficiently implemented.

DESIGN ENTRY

Microprogrammable 32-bit chips

large addressing capability (4 billion bytes) to support virtual memory systems (Fig. 1).

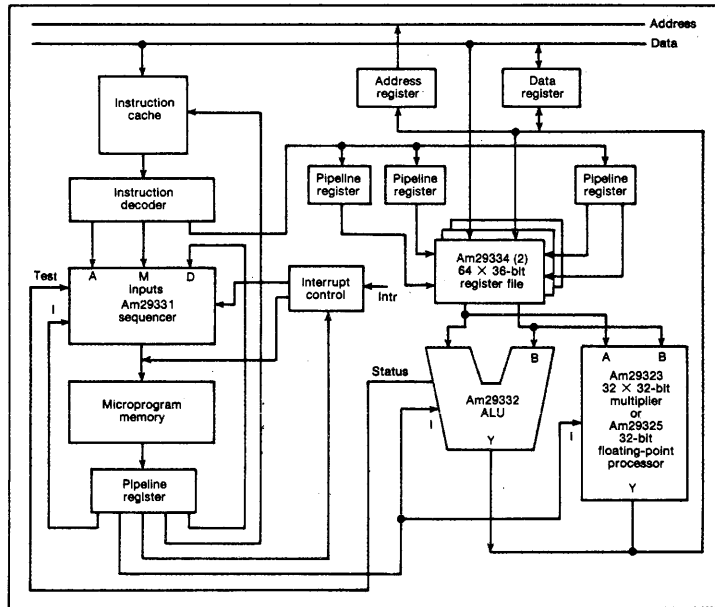
The performance of a system depends, not just on its raw computing speed, but on its ability to respond to events such as interrupts and traps. For example, the Am29331 sequencer responds to both interrupts and traps at the microprogram level very quickly, and its response is completely transparent to the interrupted microroutine. Also, the Am29332 ALU indirectly supports the handling of these events by allowing its internal state to be saved or restored.

The Am29332, a noncascadable 32-bit-wide, ALU, provides fast number crunching, high data transfer rates, and powerful bit-manipulation capabilities. Intended to be used with the Am29334 dual-ported RAM, which serves as an external register file, the ALU has two

32-bit input buses (DA and DB) and one 32-bit output bus (Y).

Internally, the device has a 32-bit data path that interconnects its various functional blocks. These blocks include various shifters and multiplexers, a mask generator, a funnel shifter, the ALU proper, a priority encoder, a parity generator and checker, a master-slave comparator, and the status and Q registers (Fig. 2). The ALU proper has three 32-bit inputs: R, S and M. The R input comes from the funnel shifter, the M input from the mask generator, and the S input from a variety of sources—the DA or DB buses, status register, or the Q register.

The power and flexibility of the Am29332 comes partly from its ability to perform operations on various data types. It can operate on



1. A conventional CPU, built with Am29300 building blocks, forms the focal point of an extremely compact system that cycles as fast as 80 ns.

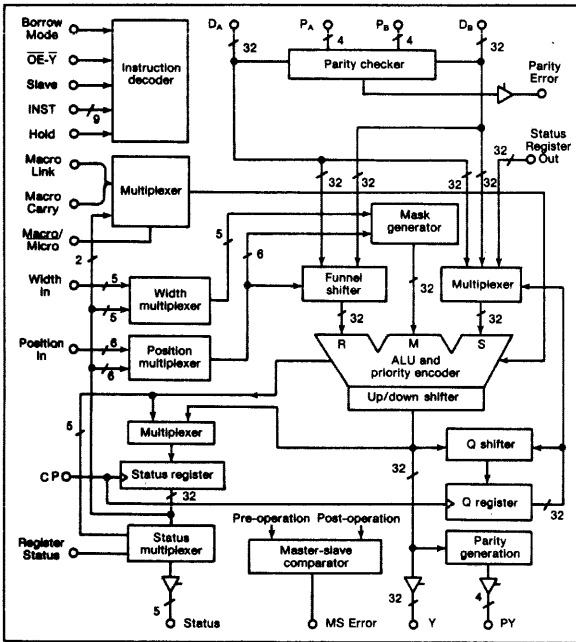
variable bytes, variable-length bit fields, or single bits. This is made possible by the internal mask generator, which creates a 32-bit mask for each instruction (with no time overhead). The mask is used as an additional operand in each instruction to allow the operation on only selected data widths.

The type of mask generated depends on the type of instruction. For instructions that operate on variable bytes (1, 2, 3 or 4 bytes) the mask is a fence of 1s (bit 0 aligned) for all low-order selected bytes with a fence of 0s for all high-order unselected bytes. Instructions that operate on variable-length bit fields require a mask that is a string of contiguous 1s for all selected bit positions and 0s for all unselected bit positions. In cases where the field exceeds the 32-bit boundary, the mask does not wrap around, thus

allowing operation on a contiguous field across a word boundary. For instructions that operate on a single bit, the mask is a 1 for the selected bit position and 0s for the other unselected bits.

For most single-operand instructions, the unselected bit positions pass the corresponding bits of the operand unmodified. For most two-operand instructions, the unselected bit positions pass the corresponding bits of the operand unmodified on the DB input. Thus, for two-operand instructions the mask allows the merging of two operands in a single cycle. In addition to being used internally, the mask can be sent out over the Y bus, permitting the generator to be used as a pattern generator for testing purposes.

To speed various mathematical and logical operations, many circuits have started to in-



2. To connect its various internal functional blocks, the Am29332 ALU employs a 32-bit bus. Among the chip's major features are a 64-bit funnel shifter, parity checking and generation, and a basic 32-bit ALU that has three input ports. The processor also has three 32-bit ports through which it transfers data into and out of the chip.

DESIGN ENTRY

Microprogrammable 32-bit chips

clude a barrel shifter, which has an N-bit input and an N-bit output. The barrel shifter would be used to shift or rotate the operand either up or down from 0 to N bits in a single cycle. Such high-speed shifting is very useful in operations such as the normalization of a mantissa for floating-point arithmetic or in applications in which the packing and unpacking of data are frequent operations.

However, a more useful circuit is a funnel shifter, which can be thought of as having two N-bit inputs and one N-bit output. Just such a circuit (with 32-bit-wide ports) was included on the 29332. The circuit can perform all the operations of a barrel shifter with capabilities extended to two operands instead of one. In addition, it can extract a 32-bit contiguous field across its two operands, a function very useful in several graphics applications. And any of its operations can be followed by a logical operation, with both completed in a single cycle.

Setting the priorities

Prioritization, useful to control N-way branches, perform normalizations, and in graphic operations such as polygon fills, can readily be handled by the ALU chip. The built-in priority encoder sends out a 5-bit binary weighted code that signifies the relative position of the most-significant 1 from the most-significant bit position of the byte width selected. That allows prioritization on either 8-, 16-, 24-, or 32-bit operands. The priority encoder output can be passed on to the Y bus or stored in the status register.

If, for example, prioritization is used to normalize a mantissa during a floating-point arithmetic operation, it requires two cycles. In the first, the mantissa is prioritized to determine the number of leading 0s that need to be stripped off. In the next cycle, the mantissa is shifted up by the amount specified by the priority encoder output.

Relevant information for each operation performed by the chip is stored in the 32-bit status register after each microcycle. Each byte of the status word holds different information. The least-significant byte holds the position specifier. The next most-significant byte holds the width specifier and three other bits that are used to test the comparison of unsigned and

signed operands. The next byte contains the Carry, Negative, Overflow, Link, Zero, M and S flags. The M flag stores the multiplier bit for multiply or the sign compare bit for signed division, and the S flag stores the sign of the partial remainder for unsigned division. The most significant byte stores the nibble carries for BCD operations.

The states of the Carry, Negative, Overflow, Link and Zero flags are available on the status pins, and the status multiplexer allows the user to select either the status of the previous instruction (register status) or the status of the current instruction (raw status) to appear on the status pins. The raw status could be used to update an external macro status register. This also allows branching at either the micro- or macro-level.

The Q shifter and Q register are primarily used to assemble the partial product or partial quotient in multiplication and division operations. Variable bytes of the status and Q register can either be loaded via the DA and DB inputs or can be read over the Y bus. Thus saving and restoring of the registers allows efficient interrupt handling after any microcycle. It is also possible to inhibit the update of both these registers by asserting the Hold pin.

Powerful and orthogonal instructions

The power of the ALU chip's instruction set comes directly from the integration of several functional blocks mentioned earlier. The commands are symmetrical as well as orthogonal, to make it easier for a compiler to generate efficient code. Thus, any operation on the DA input is also possible on the DB input, and each instruction is completely independent of its data type.

Three-fourths of the instruction set consists of variable byte-width (one, two, three or four) operand instructions. The byte-width is selected by two bits in the instruction. For these operands, the instruction set supports all conventional arithmetic, logical and shift operations. Arithmetic operations can be performed on both signed and unsigned binary integers.

Additionally, the instruction set supports multiprecision arithmetic such as addition with carrying and subtraction with carrying or

DESIGN ENTRY

Microprogrammable 32-bit chips

borrowing. For all subtract operations it provides the convenience of using borrowing instead of carrying by asserting the borrow pin. In this mode the carry flag is updated with the true Borrow. To allow efficient execution of macroinstructions the chip contains a Macro mode pin. When the chip asserts this pin, it allows the external Macro-Carry and Macro-Link bits instead of their microcounterparts to participate in the operation.

Instructions that execute algorithms for the multiplication and division of signed and unsigned integers are multiple cycles are also provided. For multiplication, the circuit supports the modified Booth algorithm, yielding two product bits in one cycle. Both single-precision and multiprecision division of signed and unsigned integers are supported at the rate of one quotient bit in every cycle.

Besides binary integers the instruction set provides basic arithmetic operations for binary-coded decimal (BCD) numbers. By operating directly on the decimal numbers created

in most business applications, significant processing time is saved by eliminating the need to convert from binary to BCD and vice versa. Also, the round-off errors involved in converting from one base to the other are eliminated.

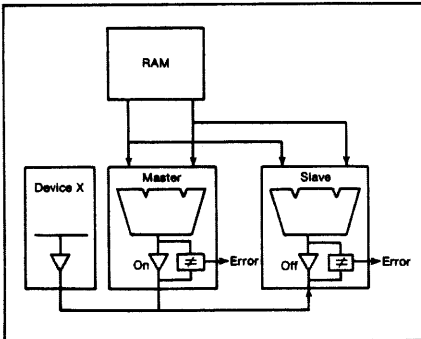
The last group of instructions was created to support variable-length bit fields (1 to 32) and single-bit operands. The position and width of the field can be specified by either the position and width inputs or by fields in the status register, thereby saving bits in the microcode. Most of the time, the position and width are determined dynamically. It is therefore difficult to supply them via the microinstructions. For single bit operations only the position specifier is needed.

Bit-manipulation instructions include setting, resetting, or extracting a single bit of the operand or the status register. Logical operations on either aligned or nonaligned fields in the two operands include OR, AND, NOT and XOR. In the case of nonaligned fields it is assumed that at least one of the fields is aligned to bit position 0. It is also possible to extract a field from one operand and insert it into another operand or extract a field across two operands.

Enhancing system integrity

The growing need for data integrity has been addressed at both the system and the chip level by including hardware for fault detection. During calculations, byte-wide even parity is generated for the data result by the ALU and stored with the data in the external RAM. Byte-wide even parity is also checked at the ALU inputs and any error is flagged.

Even parity is specifically used to check for a floating TTL bus. Thus, all interchip connections are checked out. In addition, hardware for functional verification is also provided on the sequencer and the ALU functional verification can be implemented by using two similar devices in the master and slave mode (Fig. 3). In that setup, both chips perform the same operation, with any difference in their outputs being flagged as an error. The slave-mode chip's bidirectional buses operate in their input mode, allowing the master to compare its own internal result with that of the slave on every cycle. Additionally, the master checks the output bus to



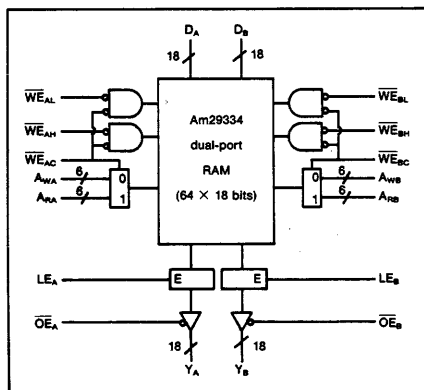
3. To help ensure system integrity, two Am29332 processors can be set for master and slave operation. Both chips perform the same operation in parallel, and any difference in their results is flagged as an error. The master also checks its internal result against the data on the output bus to make sure that no other device (such as device X) is turned on at the same time.

DESIGN ENTRY

Microprogrammable 32-bit chips

make sure that no other device is turned on at the same time.

As mentioned earlier, the ALU architecture was designed to use an external register file. Keeping the file external to the chip permits the user to expand it to meet any system need. The Am29334, a high-speed 64-word-by-18-bit dual-access RAM, provides two independent data input ports and two independent data output ports (Fig. 4). Each port can be read from or written to using the separate inputs and outputs. The two accesses are independent except for the case when simultaneous write operations are done to the same word—in which case the result is undefined. The read address inputs and the write address inputs of each side are se-



4. The dual-access RAM serves as an external register file for the arithmetic processor chip. The Am29334 holds 64 words, each 18 bits long. Two chips are often connected to build a RAM block with four data outputs, two data inputs, and six address lines. Each port of the RAM can be independently accessed to read or write.

parate in order to save the cost and time delay of external multiplexing between a read address and a write address.

The word width of 18 bits allows the RAM to store two bytes plus a parity bit for each. Each side has separate write enable for the lower and upper nine-bit bytes and a common write enable that also switches the address multiplexer. The actual write is delayed internally to allow the write address to set up internally before writing starts.

It is possible to build a RAM with four data outputs, two data inputs and six addresses by using two dual-access RAMs and on each side connecting the data input, write address and write enables of one RAM in parallel with the corresponding inputs of the other RAM. This expanded RAM may be used in concurrent processing applications in which an ALU and an adder (which generates the address) do their computations—this yields a result and an address in parallel. The two values can then be fed simultaneously to the multiport memory.

The sequencer controls the show

The cycle time of the microprogrammed system is dependent on both the control path (i.e., sequencer and microprogram memory) and the data path (i.e., register file and ALU). Traditionally, the system bottleneck has been the control path, especially the critical paths associated with conditional branching. Special care has been taken in the design of the Am29300 family to balance control and data-path timing.

A key device contributing to the improved control-path timing is the Am29331 16-bit microprogram sequencer. It is designed for high speed, and that speed has been attained by the elimination of functions that would slow down the microaddress selection and by including the test logic and the test multiplexer in the sequencer (Fig. 5). As in most previous generation sequencers, the address register, the incrementer, the address multiplexer, the stack, and the counter are standard functions. The sequencer has multiway branch instructions that allow 1 of 16 consecutive addresses to be selected as the branch target in a single cycle.

The address register in most other sequencers is called a program counter, but this name is not correct if a strict definition is applied. In

DESIGN ENTRY

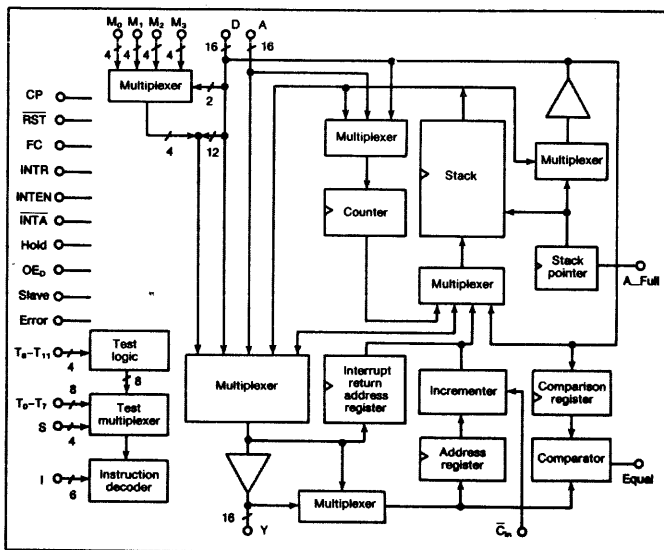
Microprogrammable 32-bit chips

the Am29331, the incrementing counter is placed after the address register, which thus allows for the handling of traps. The stack stores return addresses, loop addresses and loop counts. It has 33 levels to permit the deep nesting of subroutines, loops and interrupts. An output, Almost Full (A-Full), indicates when 28 or more of the levels are in use.

Available for use in iterative loops, the counter can be loaded with an iteration count at the beginning of a loop, and the count is tested and then decremented at the end of the loop.

The loop is terminated if the count is equal to one; otherwise a jump to the beginning of the loop is executed.

There are three buses that carry microaddresses. The bidirectional D bus can be connected to the pipeline register, providing branch addresses or loop counts, or used for two-way communication with the data processing part of the system. The A bus, called an alternate bus, can be connected to a mapping PROM to provide starting microaddresses for instructions in a computer. The Y bus sends out



5. To aid in handling trap operations, the incrementer is placed after the address register in the Am29331 microsequencer. Additionally, the chip has a 16-bit address bus, which enables it to access up to 64 kwords of control memory and handle interrupts and multiple-path branches.

DESIGN ENTRY

Microprogrammable 32-bit chips

selected microaddresses to the microprogram memory and accepts interrupt or trap addresses if interrupt or trap is employed.

Four sets of 4-bit multiway inputs provide a simultaneous test capability of up to 4 bits. And, one way to use those inputs would be to decode mode bits in changing positions in macroinstructions. The four select lines select 1 of 16 tests to be used in conditional instructions. There are twelve test inputs. Four of these may be used for C (Carry), N (Negative), V (Overflow) and Z (Zero), generating internally the tests $C+Z$, $C+Z, N \text{ XOR } V$, and $N \text{ XOR } V+Z$, which are used for comparison of signed and unsigned numbers.

Relative addressing was the only somewhat useful function that was removed in order to maximize speed. The sequencer supports interrupts and traps with single-level pipelining, but may also be used with two levels of pipelining in the control path. It has a 16-bit-wide address path and cannot be cascaded, which thus limits the addressable memory depth to 64 kwords of microcode. That, however, is sufficient for the vast majority of applications—a typical computer, for instance, that has a microprogrammed instruction set, might use only about 1 to 2 kwords. However, for systems in which the microprogram is the sole program level, its size is generally larger.

Microprogram interrupts supported

The Am29331 sequencer supports interrupts at the microprogram level. Like polling, interrupts handle asynchronous events. However, polling requires explicit tests in the microprogram for events, thus leading to long response times, lower throughput, and larger microprograms. Interrupts, on the other hand, have a response time equal to the cycle time of the system (approximately 80 ns), measured from the Interrupt Request input (INTR). The sequencer accepts interrupts at every microinstruction boundary when the Interrupt Enable input (INTEN) is asserted.

An actual interrupt turns off the Y bus driver and asserts the Interrupt Acknowledge output (INTA), which should be used to enable an external interrupt address onto the Y bus, thus driving the microprogram memory. The interrupt also causes the interrupt return address to

be saved on the stack; this permits nested interrupts to be handled (Fig. 6).

The Am29331 is also the first sequencer that can handle traps. A trap is an unexpected situation caused by the current microinstruction, which must be handled before the microinstruction completes and changes the state of the system. An attempt to read a word from memory across a word boundary in a single cycle is an example of such a situation. When a trap occurs, the current microinstruction must be aborted and re-executed after the execution of a trap routine, which will take corrective measures.

Execution of a trap requires that the sequencer ignore the current microinstruction and push the trap return address—the address of the ignored microinstruction—on the stack. The trap address must be transferred onto the Y bus at the same time. All this can be accomplished by disabling the carry-in to the incrementer (C_{in}) and asserting the Force Continue input (FC) and the Interrupt Request input (INTR).

Also built into the sequencer is an address comparator, which allows detection of breakpoint in the microprogram. An output signal from the comparator indicates when the content of the comparator register is equal to the address on the Y bus. There is an instruction that loads the comparator register from the D bus and enables the comparator, which may later be disabled by another instruction.

Parallel microprocesses are useful when the system must deal with peripheral devices that are controlled at the microcode level. Normally only one processor is present and it must be time multiplexed between the concurrent operations that must be performed. When a process is suspended its private state must be saved, so that it can be restored when the process resumes execution. That, in turn, requires that the state of the sequencer be saved and restored, or each process must have its own sequencer that is active when the associated process is active. The first approach is the least expensive, but the second offers the advantage of shorter response time, because no time is spent on saving and restoring the state.

The Am29331 supports the first approach with its bidirectional D bus, through which the

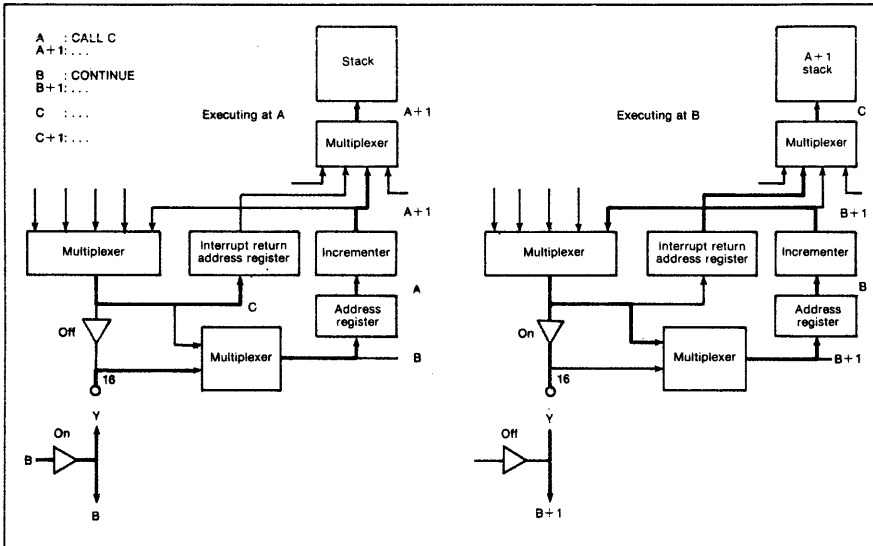
DESIGN ENTRY

Microprogrammable 32-bit chips

entire state, with the exception of the comparator register, can be saved and restored. The sequencer also supports the multiple sequencer arrangement, in which the three-state Y buses from the sequencers are tied together driving a single microprogram memory. One of the sequencers is active, while the remaining sequencers are put on hold by asserting their Hold inputs. The Hold input disables most outputs (the D bus synchronously), disables the incrementer, and enables an internal Force Continue. This effectively detaches the sequencer

from the system and preserves its state.

The sequencer has a 6-bit instruction input that is internally decoded to yield a set of 64 instructions. There are 16 basic branch instructions, each in an unconditional version, a conditional version, and a conditional version with complemented test. In addition there are 16 special instructions like Continue and Push C (push counter on stack). The branching instructions handle jumps, subroutines, various kinds of loops and exits out of loops, and FC actually overrides the instruction inputs with a continue



6. Because it can accept interrupts at any microinstruction boundary, the sequencer responds faster than most other microprogrammed systems. For example, while the instruction at point A in memory is being executed, the sequencer is directed to point B. The only restriction on the programmer is that the first instruction of the interrupt routine cannot use the stack, since the interrupt return address is pushed onto it at the start of the procedure.

DESIGN ENTRY

Microprogrammable 32-bit chips

instruction. FC is useful in field sharing and support for writable microprogram memory.

The Am29331 is one of the few sequencers where the stack is accessible from outside through the bidirectional D bus. This indirectly allows access to the whole state of the sequencer except the comparator register. This is useful when testing the device, and during

system debugging, in which, for example, the contents of the counter and the stack may be examined and altered. By including the troubleshooting instructions in the microcode, the sequencer may aid in debugging itself and the rest of the system. The access to the state is also useful for changing context or extending the stack outside.□

Am29300 DEMONSTRATION SYSTEM

Application Note

By Mark McClain

This application note describes the design of a high performance microprogrammed 32-bit processor using the Am29300 family of 32-bit building blocks. Basic design philosophy for a microprogrammed processor is discussed as the design choices made for this system are explained. Support circuitry used with the Am29300 family components is also covered in detail. This circuitry includes: Writable Control Store, Serial Shadow Register diagnostics, and Programmable Array Logic.

Table of Contents

SECTION 1	Overview	6-15
	SYSTEM LAYOUT	6-16
	DATA FLOW	6-16
	Memory and I/O Sections	6-16
	Data Section	6-16
	Control Section	6-17
SECTION 2	Nomenclature	6-19
SECTION 3	Data Section Description	6-21
	REGISTER FILE	6-21
	ARITHMETIC LOGIC UNIT	6-21
	Am29332	6-21
	Macro Status Register	6-21
	FLOATING POINT PROCESSOR	6-24
	Am29325	6-24
	FPP External Status Register	6-26
	Seed Look-Up Table	6-26
	PARALLEL MULTIPLIER	6-28
SECTION 4	Memory and External System Interface	6-31
	EXTERNAL BUS INTERFACE CONTROL	6-32
	Host Access Definition	6-32
	Host Interface Block Diagram	6-33
	Event Signals	6-34
	Memory Enable	6-35
	AmPAL22V10 Support Logic	6-35
	SSR Diagnostics	6-35
	Controller Description	6-37
	MEMORY	6-40
	Memory Components	6-40
	Addressing Scheme	6-40
	CPU - Memory Buffers	6-42
	External System Buffers	6-43
SECTION 5	Control Section Description	6-45
	MACRO OPCODE SUPPORT	6-45
	Macro Opcode Register	6-45
	Macro Opcode Format Restrictions	6-46
	Macro Opcode Decoding Method	6-47
	Macro Opcode Map RAM	6-47
	WCS Port	6-48
	Macro Operand Address Counters	6-48
	REGISTER FILE ADDRESS MULTIPLEXER	6-50
	Read Ports A and B	6-50
	Write Port A	6-51
	Write Port B	6-52

	POSITION AND WIDTH MULTIPLEXERS	6-52
	SEQUENCER	6-52
	D BUS TRANSCEIVER	6-56
	INTERRUPT CONTROL	6-56
	Interrupt and Trap Philosophy	6-56
	Interrupt Operations	6-57
	Trap Operation	6-59
	MICROCODE CONTROL STORE AND CONTROL PIPELINE REGISTER	6-60
	Control Store Function	6-60
	Pipeline Register Function	6-60
	Control Store Implementation	6-60
	CLOCK CONTROL	6-62
	Clock Qualification Circuit	6-62
	Clock Generator	6-64
	MICROCODE WORD	6-66
	Control Philosophy	6-66
	Microcode Word Field Descriptions	6-66
	Alternate Arrangements	6-72
	CONTROL DECODE	6-74
	What Is It Good For?	6-74
	Control Logic Description	6-74
SECTION 6	System Timing and Critical Path Analysis	6-77
	DEFINITIONS	6-77
	CONTROL AND DATA PATHS	6-77
	WORST CASE PATHS	6-78
	Case Definitions	6-78
	FINAL RESULTS	6-80
SECTION 7	Physical Issues	6-87
	ELECTRICAL LAYOUT ISSUES FOR POWER SUPPLY DECOUPLING CAPACITORS	6-87
	SOCKETS	6-87
SECTION 8	Conclusion	6-91

SECTION 1 Overview

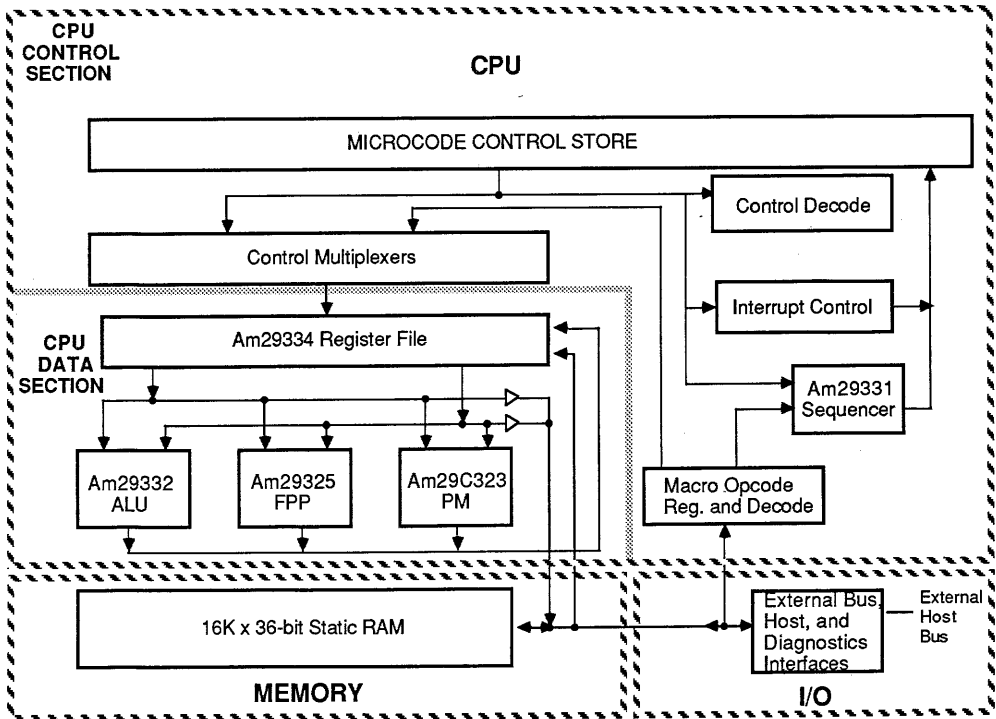
This application note describes the design of a high performance microprogrammed 32-bit processor using the Am29300 family of 32-bit building blocks.

Basic design philosophy for a microprogrammed processor is discussed as the design choices made for this system are explained. Issues of microprogram sequence control, interrupt handling, microprogram memory options, microword layout, macroprogramming, high speed multiply, and clock control are covered.

Support circuitry used with the Am29300 family components is also covered in detail. This circuitry includes: Writable Control Store, Serial Shadow Register diagnostics, and Programmable Array Logic.

The use of the following Advanced Micro Devices components is illustrated in extensively documented examples:

- Am29331** - 16-bit Address Sequencer,
- Am29332** - 32-bit Arithmetic Logic Unit,
- Am29334** - 64 x 18-bit Four Port Register File,
- Am29C323** - 32-bit Parallel (Integer) Multiplier Accumulator,
- Am29325** - 32-bit Floating Point Unit,
- Am29114** - Interrupt Controller,
- Am29800** - Family of Interface and Diagnostics Logic Devices,
- Am29PL141** - Fuse Programmable State Machine,
- AmPAL18P8** - Programmable Output 20-pin Combinatorial PAL,
- AmPAL22V10** - Output Macrocell 24-pin PAL,
- Am9151** - Registered RAM with SSR™,
- Am99C165** - 16K x 4-bit CMOS high speed RAM.



09856A 1-1

Figure 1-1. System Components

SYSTEM LAYOUT

As with all processors, this system contains three main portions: Central Processing Unit (CPU), memory, and input/output (I/O) (see Figure 1-1).

The CPU consists of a control section and a data section:

The data section manipulates data via operations such as addition, subtraction, shifting, merging, multiplication, and division. These functions are implemented with the Am29332 Arithmetic Logic Unit (ALU), Am29325 Floating Point Processor (FPP), and Am29C323 Parallel Multiplier (PM). The data section also stores operands and intermediate results in Am29334 register files.

The control section directs the operations performed by the data section and determines the order in which the operations are performed. This section contains the Am29331 Microprogram Sequencer, macro opcode register & decode, interrupt control logic, microcode control store, control decoding logic, and control multiplexers for the register file and ALU.

The memory contains a 16K word by 36-bit static RAM. Included as part of the memory block are two address registers/counters, which may be used to speed up sequential reads and writes made by the CPU.

The I/O portion is a simple connection to a host system's address and data bus. It is assumed that the Am29300 demonstration system operates as a peripheral processor to a larger host system, as might be the case with an array or digital signal co-processor. Information to be processed by the demonstration system is loaded into the memory portion via Direct Memory Access (DMA). When processing of the data is complete, the host system unloads the memory portion via DMA.

A diagnostics port is also provided as part of the I/O section. This port allows control over the demonstration system clock for single stepping, and it allows for serial diagnostics to display and control the state of the system.

Throughout the remainder of this application note, it is assumed that the reader has some previous experience with microprogrammed processor design and is familiar with the Am29300 family data sheets. For those readers not familiar with microprogrammed design, some reference material is listed in Appendix A.

DATA FLOW

The system data paths are illustrated in the block diagram of Figure 1-2.

Memory and I/O Sections

Information processed by the Am29300 system is exchanged between the host system and the memory via the external bus interface. The information may be both data and macroinstructions.

From the external bus, the host system is able to address the memory via the bus driver connected to the memory address bus. Data is moved over the memory data bus. The host system's only access to the Am29300 system is via these buses to the memory. Therefore, all data to the system flows through the memory via DMA accesses by the host system.

Diagnostic control and information flows through the external bus interface via the host interface controller. It controls the clocking and single stepping of the system while loading and reading serial diagnostics via Serial Shadow Registers (SSR) that are placed in key locations throughout the system.

(SSR is a trademark of Advanced Micro Devices, Inc.)

Data Section

Data must be moved from the memory to the register file to be available to the ALU and multipliers for processing.

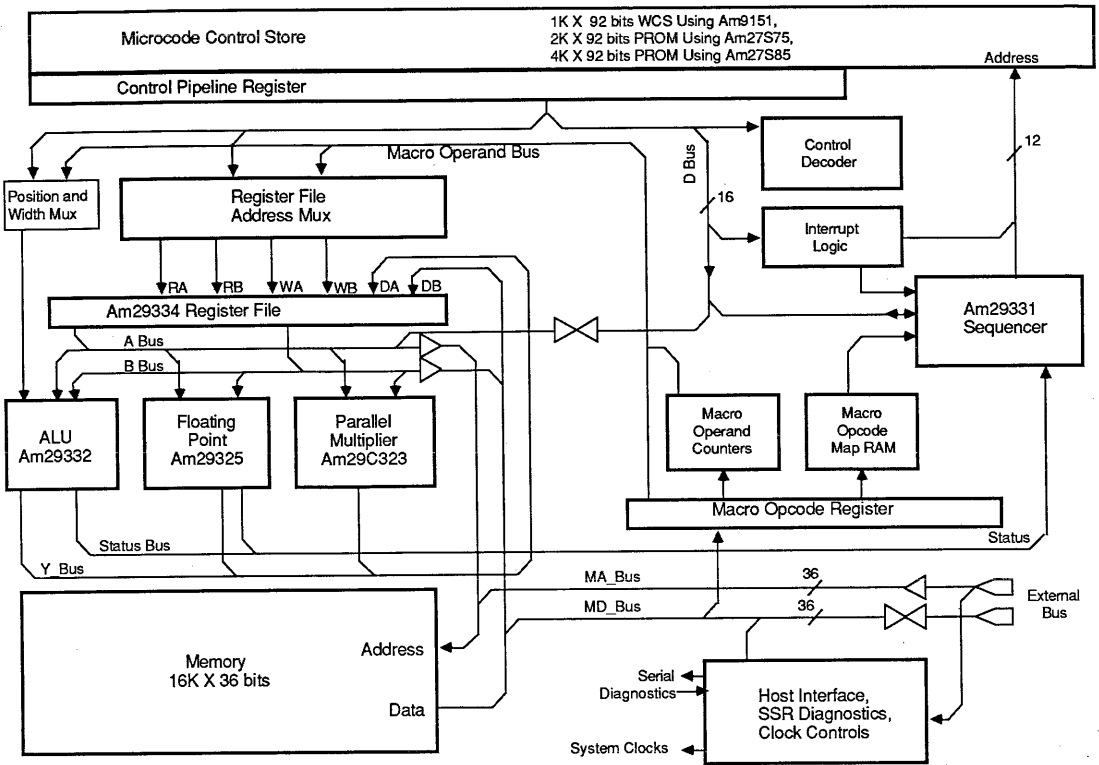
The register file has four access ports, two ports for writing data into the file and two ports for reading data out to the ALU and multipliers. This arrangement allows two operands to be read from the file in the same cycle as two operands are being written. The two read operands are used either as A and B operands for the ALU, FPP, or PM, or as address and data inputs to the memory.

To move data from the memory to the register file, an address to the memory is selected from the register file on the A read port. This address selects a word from the memory that is transferred on the memory data bus to the B write port of the register file.

Once data is loaded into the register file, it can then be selected for use on either the A or B read ports for input to the ALU, FPP, or PM.

Data processing results from the ALU, FPP, or PM are then placed on the Y bus for return to the register file A write port.

Finally, processed data is moved back to the memory via the B read port of the register file, while the location to be written in the memory is addressed by the value on the A read port of the register file.



09856A 1-2

Figure 1-2. Am29300 Demonstration System

(NOTE: The advantage of using both write ports on the register file is that it is possible to perform calculations and write the results via the A write port at the same time that new data is being moved into the register file from the memory via the B write port. This will be illustrated in more detail later in this document.)

Control Section

D Bus

The D bus is a highway for information flow between the microcode control store, interrupt control sequencer, and data section of the CPU.

Branch addresses or constants from the microcode can pass to the sequencer via the D bus. The interrupt controller's interrupt vector base address register may also be loaded via the D bus.

Constants from the microcode can pass to the data section for use in calculations via the D bus to A bus transceiver. Microcode constants can also be used as

addresses to the memory, via a D bus to A bus to memory address bus connection.

Variable data can be passed from the register file to the sequencer. The sequencer can also return data to the register file, via the A bus to ALU Y bus to A write port path. The D bus path to the sequencer is valuable for storing and retrieving the state information in the sequencer when interrupts, traps, or context switches occur.

Control Decode

This section of logic expands encoded microcode fields into individual control lines used throughout the system.

Interrupt Logic

This circuit monitors interrupt and trap conditions such as parity errors and breakpoints. When an interrupt condition is detected, an interrupt request to the sequencer is made and an interrupt address vector generated.

Sequencer

The sequencer is an address multiplexer with an on-chip address incrementer and stack. It selects the address for each microinstruction word read from the control store. The address selected depends on the instruction to the sequencer and on the state of test conditions. The sequencer can select addresses from the branch field of the control pipeline register, the macro opcode map, the internal stack, the increment of the last microinstruction address, or one of four status condition driven multi-way branch inputs.

Macro Opcode Support

Macro vs. Micro Programs: A microprogram is the definition for the state of the primary system control signals during each system clock cycle. Each word of microcode usually has a large number of bits so that many parallel operations may be controlled simultaneously. Each microcode word must deal with the intricate details of system operation. The writing of microcode is a slow tedious process that must take into account every facet of system operation in order to provide the most efficient use of system resources.

The advantage of microcode is that, very often, different system operations can be overlapped (done in parallel) since there is parallel control over all the system resources.

A "macroprogram" is a series of microcode subroutine calls. Each macroinstruction has an opcode field that is simply a value that can be translated into the starting address of a microcode subroutine within the system microprogram. The macroinstruction may include parameters that are passed to the microprogram. These parameters might be register addresses, loop counter values, immediate data, or memory addresses.

The advantage of a macroprogram is that the instructions are very simple and require relatively few bits to define as compared to a microcode word. The macroinstructions are simpler because all the details of system operation are specified by the underlying microcode instructions. The simpler instructions allow macroprograms to be written much more quickly than microprograms. Therefore, once a set of microcode subroutines are developed to perform the most often needed system operations, a wide variety of macroprogram applications can be quickly written. Macroinstructions remove the system programmer's concern over every detail of system operation.

The disadvantage of a macroprogram is that each instruction must be fetched from memory and decoded (translated to a microcode subroutine address) before

each microcode subroutine is executed. When each subroutine execution is long compared to the overhead of fetching and decoding the macroinstruction, the macroprogram will run nearly as fast as an equivalent microprogram with the advantage being a much easier programming task. When the microcode subroutines are short compared to the macroinstruction overhead, the system speed can drop significantly.

So, if macroprogramming concepts are used carefully, a macroprogrammed approach to system design can yield a significant improvement in the ease of system use without a large decline in system performance.

For that reason, the Am29300 demonstration system includes the features described below, which allow a macroprogrammed approach. These features are intended to show how basic macroprogramming can be implemented.

Macro Opcode Register: When macro-instructions are executed, the instructions are addressed in the memory via the A read port of the register file in the same way as described earlier for data. The selected instruction is read from the memory via the memory data bus and written into the macro opcode register. The instruction can also be written into the register file via the B write port in the same cycle (which may be useful for instructions that contain immediate operands that would be used by the data section).

Macro Opcode Map RAM: The macro opcode map RAM is made of three Am9150 high speed SRAMs. The opcode portion of the macro opcode register addresses a microcode entry point table in the map RAM. This entry point is then used by the Am29331 sequencer as a branch address to the microcode routine that performs the function required by the macroinstruction.

Macro Operands: The operand portion of the macro opcode register is loaded into the macro operand counters. The macroinstruction operands allow the direct specification of register file addresses, ALU shift values, or ALU field masks to be used by the microcode routines.

Register File Address, Position, and Width Multiplexers: Register file addresses are passed to the register file via the register file address multiplexer. Position and width information for shift values and field masks are passed to the ALU via the position and width multiplexers. These multiplexers allow either the microcode or the macroinstructions to control the register file and ALU.

SECTION 2
Nomenclature

Throughout the remaining figures in this application note, some naming and drawing conventions are used as noted below.

All signal names are written as single word identifiers with underlines used to provide visual space between sections of a multi-word identifier.

Signals that are active low have names that end with an asterisk. In some of this document's programmable logic definition files, this convention is not allowed. In those situations, the active low signal names will begin with an exclamation point or end with an underline character.

Clock and qualified clock signals have names that begin with CLK_.

Groups of signals that form buses are shown as single lines with an associated number that indicates how many lines are involved. Bus lines are drawn with 45 degree turns and intersections instead of the usual right angle turns and intersections used with individual signal lines, in order to highlight buses visually. Major data highways such as the A_BUS, B_BUS, and Y_BUS have signal names that end in _BUS. The lines of a bus are numbered from least significant to most significant with the least significant identified as line zero (0). Where a subset of the lines in a bus is shown, the bus signal name will be followed by parentheses containing numbers that show the range of lines in use. The numbers of a continuous range are separated by a colon (:), non-contiguously numbered lines are separated by a comma (,). Where lines of a bus are split out to show the specific connection of bus lines in a circuit, a small number that indicates the line number within the bus will be shown near each line that is split off.

Four major buses in the system share a common structure. The A_BUS, B_BUS, Y_BUS, and MD_BUS all have the same layout. Each bus carries a 36-bit data word, which is arranged as four 8-bit bytes, each byte having its own parity bit. Byte zero (least significant) is

located in bits 0:7; bit 32 is the parity bit for byte zero. Byte one is in bits 8:15 with its parity in bit 33. Byte two is in bits 16:23 with parity in bit 34. Byte three is in bits 24:31 with parity in bit 35.

Signals that come directly from the microcode memory pipeline register have signal names that begin with "P_".

Ground symbols (zero volt points) are drawn as downward pointing triangles, or the signal name GND is used.

Points tied to +5 volts are labeled with the signal name V_{CC} .

Components are shown with pin numbers immediately outside the rectangle that defines the component. Component-specific signal names related to component pins may be shown immediately inside the component rectangle. Where there are several components shown on a page with very similar connections, only one of the components will have pin numbers and signal names shown. The remaining components on the page are wired in the same manner.

Each component is assigned and labeled with a "U number" that uniquely identifies the component. This helps identify specific components for discussion and separates identical type devices in the system component list.

Because this demonstration system is complex by nature, it must be illustrated with many figures, each focusing on a different portion of the overall system. In order to show the signal interconnections between all parts of the system, each signal that leaves or enters a figure is given a name. Often the names are abbreviations in order to save space in the figures. Each name shows a relationship to the signal's use. Wherever the same signal name appears in different figures, a connection between the figures is defined. To help in identifying all the figures to which a signal travels, there is a signal-to-figure cross reference listing in Appendix B.

SECTION 3

Data Section Description

REGISTER FILE

Two Am29334 register files are used in tandem to provide a 64-register by 36-bit wide file. This allows the storage of 32-bit data plus parity (1 parity bit/byte). Each Am29334 contains 64 registers that are 18 bits wide; see Figure 3-1.

An Am29334 register file can both read and write data in the same cycle, but it does not perform the read and write simultaneously. The read must be performed during part of the system cycle and the write during another part of the cycle. Since read data is needed by the ALU and multipliers as early in the cycle as possible and, since data values to be written are only available later in the cycle, the reading of data is done in the first half of the cycle and the writing done in the second half of the cycle. A convenient way to separate the two parts of the cycle is to use the system clock signal to control the internal address mux and write enable.

As connected in Figure 3-1, the read port latch enables (LEA and LEB) and write port common enables (WEAC* and WEBC*) are tied to the data section clock line (CLK_D). This causes read data to be accessed while CLK_D is high and read data to be latched when CLK_D is low. Data is written when CLK_D is low if the port write enables are active (WEAL* and WEAH*, or WEBL* and WEBH*). The high and low byte write enables for each port are tied together since only full 36-bit word writes will be done in this system.

The various read and write addresses are provided from the register file address multiplexers, which will be covered later.

The output enable (P_OEA*) and write enables (P_WEA* and P_WEB*) come directly from the microcode pipeline register.

ARITHMETIC LOGIC UNIT

Am29332

The Am29332 provides a 64-bit funnel (barrel) shifter, 32-bit mask generator, and 32-bit ALU. The ALU can perform binary and BCD add or subtract, multi-cycle multiply or divide, and logical operations. This single, highly-integrated chip provides the complete function of the ALU block in this system. The only added component is an external register used to maintain status bits for the macroprogram separate from status information used by the micro program. The ALU is shown in Figure 3-2.

Most of the control lines come directly from the microcode control pipeline register.

The ALU output enable (ALU_OE*) is decoded from the control pipeline register.

The POSITION and WIDTH signals come from the position and width multiplexers. These multiplexers select the position and width values from either the microcode pipeline or the macroinstruction in the macro opcode register.

The slave mode input is tied to ground since there will be no use of the slave mode comparisons in this system.

The HOLD input is used as an enable control over the clocking of the internal micro status register and Q register during times the ALU is not in use. Because the ALU, FPP, and PM share the same data source and destination buses (A_BUS, B_BUS, and Y_BUS), they generally cannot be used simultaneously due to bus contention. In recognition of this, the control fields for the ALU, FPP, and PM have been overlapped in the microcode to minimize the required width of each microcode word. This means that at certain times the control lines to the ALU will be meaningless to the ALU because the values on the lines are determined by the needs of the FPP or PM. Therefore, unless the hold input is used to prevent clocking of the status and Q register during these times, the ALU status could be lost whenever the FPP or PM are in use.

Note, however, that the hold input is not used as the general means to prevent clocking of the ALU registers when the whole system is halted (e.g., during single step mode). The data clock (CLK_D) that is distributed throughout the data section of the CPU is a qualified clock and will be used to control the state change of all registers in the data section, including those in the ALU at times when the whole system is halted.

Macro Status Register

There are two levels of status information that the programmer of a microprogrammed system must track if that system executes macroinstructions. These are referred to as the micro and macro status. The micro status of the system is updated at the end of each microcode step and is part of the system state. The macro status is part of the macroprogram state as reflected at the end of each macro step. Since many microinstructions may be executed to perform the function defined by a given macroinstruction, the macro status reflects the machine state

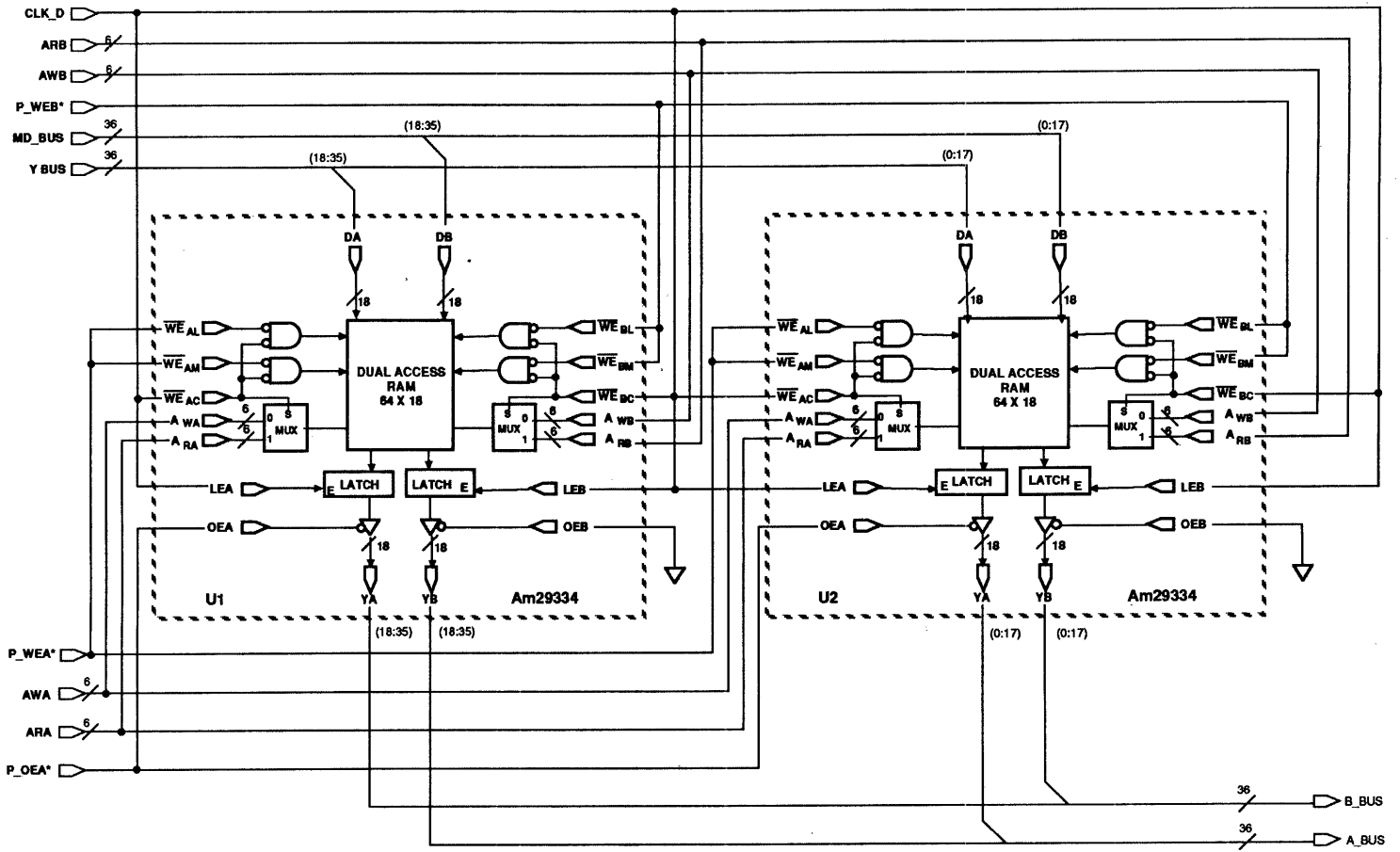


Figure 3-1 Register File

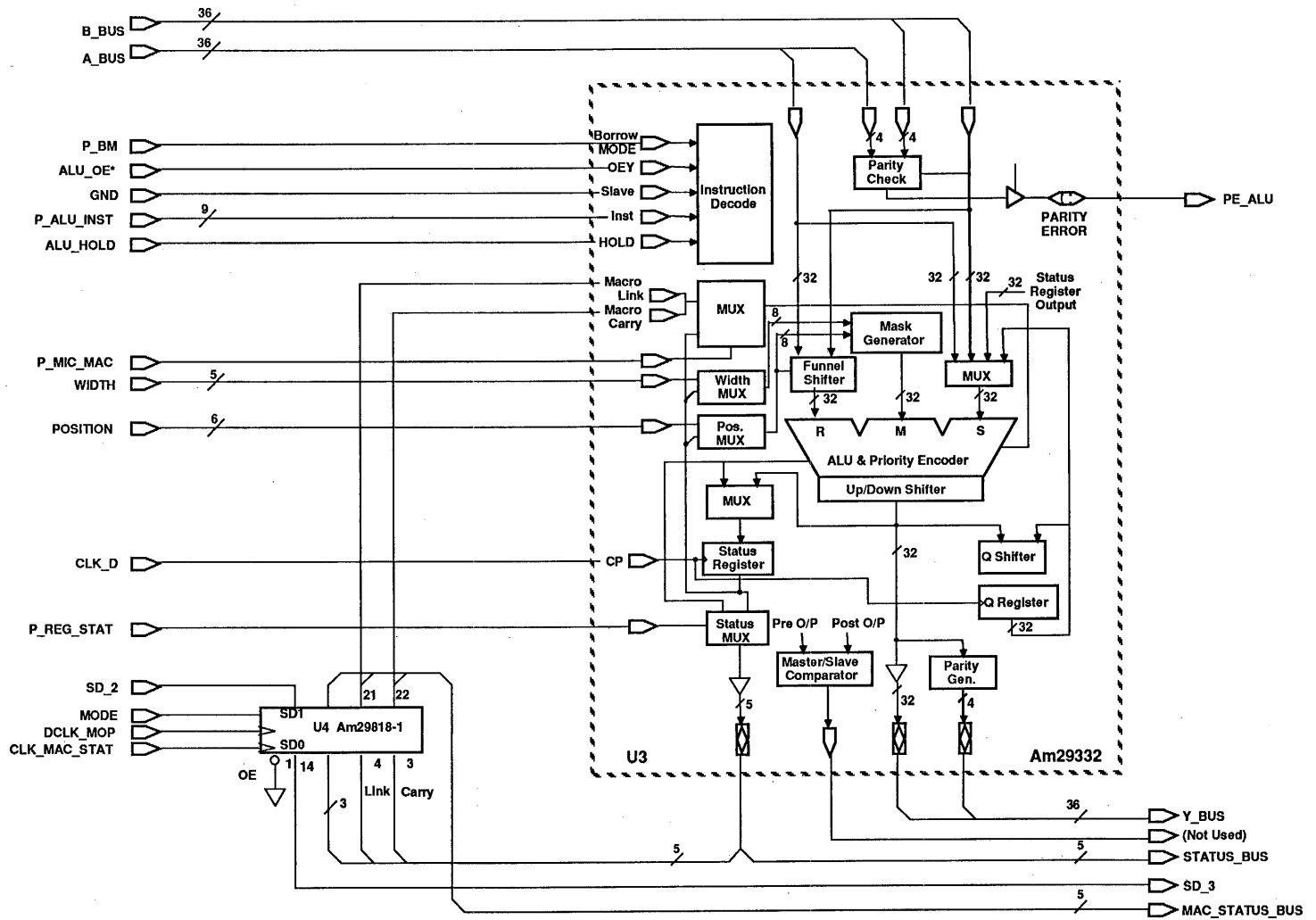


Figure 3-2 ALU Block

from the macroprogram viewpoint. The macro status may be carried across many microinstruction cycles without change. This requires a separate register to contain the macro status independent of the micro status. The Am29332 does not have an internal macro status register so one must be provided externally. The loading of the macro status register and the use of the macro status information by the microprogram must be controlled by microcode. The Am29332 does provide an on-board multiplexer to select between the micro and macro status inputs. Only the carry and link values are used directly by the Am29332 since these are the only status values normally used to modify data values. The macro status for the zero, sign, and overflow flags can be used by the sequencer as test conditions for branch instructions.

The register used for holding macro status is an Am29818-1. The register is loaded (clocked) by a qualified clock called CLK_MAC_STAT. This clock is qualified by the load macro status bit in the control pipeline register. The Am29818-1 is also used to provide a diagnostic ability to read and load the macro status register through the use of an internal serial shadow register (SSR).

FLOATING POINT PROCESSOR

Am29325

The Am29325 Floating Point Processor (FPP) performs 32-bit floating point multiplication, addition, or subtraction in a single cycle. Floating point division can be done in seven cycles using the Newton-Raphson method. The FPP is shown in Figure 3-3.

All the control lines for the FPP are driven directly by the microcode pipeline register with the exception of the FPP output enable and the register flow-through enables. Those signals are decoded from the data path select field of the microcode pipeline register. The output enable decode is done by the AmPAL22V10 in Figure 3-3. The register flow through enable decode is done by the control decode logic which is described later.

It should be noted that the Am29325 is not a full fledged member of the Am29300 family. It is different from the other Am29300 members with regard to three key characteristics: it is slower, does no data bus parity checking or generation, and has no slave mode capability.

The Am29325 flow through calculation time is 100 to 125 ns rather than the 42 or 70 ns for the ALU or PM (the current PM is at 120 ns, but the fastest version will be at 70 ns). This requires that whenever the FPP is used, the system clock cycle must be extended to allow

for the slower propagation time. This extended clock timing is covered later in more detail.

The lack of parity checking is not much of a problem for the rest of the system since it only affects the data integrity of information going through the FPP. The lack of parity generation isn't a problem as long as only the FPP is working on the data. The problem starts when floating point data is moved back to memory or is converted to integer values for use by the ALU.

If data from the FPP is read by the ALU or PM, parity errors will be detected and a system interrupt may result. That problem can be avoided if the system has kept track of which data resulted from FPP calculations and if the parity errors are ignored when that data is read. But if FPP data results are moved directly to the memory and then on to the host system, the parity errors will eventually be found.

So some means of adding parity generation to the FPP should be provided. One way is to add four 8-bit parity generator chips to the FPP output bus. This consumes power and boardspace while providing a benefit only when FPP data is moved directly through the register file to the memory. A better way is to use the parity generators already available in the Am29332 by requiring that FPP data be passed through the ALU before being moved to the memory. Even though the data may not be modified by the ALU, correct parity will be generated on the ALU output.

With the use of a little trick, there is a way to provide parity checking on the FPP data inputs. To do this, one of the data path select codes is used to control the output enables of both the ALU and FPP. This code (P_DSP = 11) causes the FPP outputs to be disabled and the ALU outputs enabled, even though the data path selected is the FPP. By turning on the ALU outputs, the ALU parity error output will also be enabled and any parity error on the A_BUS or B_BUS will be reported. At the same time, the control microcode for the FPP is still valid and may be used to load registers with the data present on the A_BUS and B_BUS. Of course the register file should not be loaded from the Y_BUS in the cycle where this scheme is used because the ALU is driving nonsense information onto the Y_BUS. Enabling the ALU outputs is only a trick used to make the ALU parity checker results available for this scheme. Note that the ALU hold input remains active even though the ALU output enable is active. This prevents any state change in the ALU when the FPP is the data path actually in use.

Finally, the issue of no slave error checking is unimportant, since the slave mode is not used in this system.

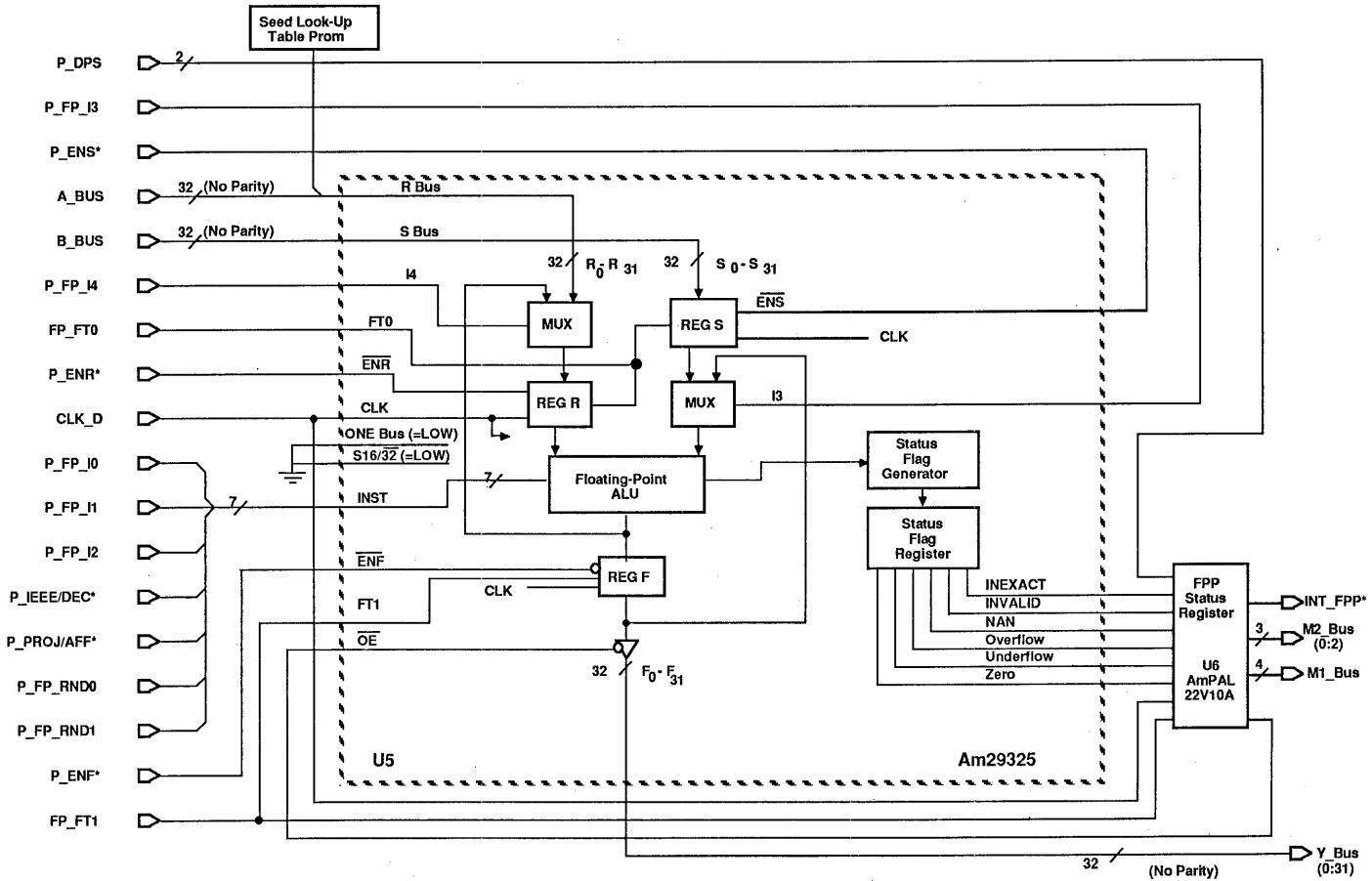


Figure 3-3 Floating Point Block

0956A 3-3

FPP External Status Register

Status Pipeline Issue

The FPP status flags appear at the status outputs along with data at the Y outputs. If the FPP "F" register is made transparent, the status flag register is also transparent. If the F register is clocked, so is the status register. In this demonstration system this presents a problem.

Normally, status conditions from the data section are registered before being used by the control section. This maintains the pipelined, parallel operation of the control and data sections. The control section bases its testing on registered status from the last data section cycle rather than being forced to wait for status results of the current execution cycle before determining the next microinstruction to execute.

To provide the same system for the FPP requires an external status register for cycles in which the F register is transparent to allow results to pass directly to the register file. In that situation the status flags are not registered by the FPP and thus, without an external register, there is no place to pipeline the status for the control section.

Multiple Status Flag Test Issue

Several of the FPP status flags signal events of equal importance such that it would be a convenience to be able to test multiple flags in a single cycle rather than basing branches on only one flag at a time.

A simple way to test multiple conditions at one time is to execute a multi-way branch based on the bits being tested. In the case of the FPP there are six flags, too many for a single multi-way branch which can be based on only four bits. A solution is to OR some of the flags together as one of the multi-way branch bits and use the remaining bits directly as part of the multi-way branch address. In that way, one multi-way branch can test all six flags.

When testing the status, if no flags are active, no abnormal condition exists, and the zero value destination of the multi-way branch continues. If one or more of the direct flags is active, the multi-way branch goes straight to a routine to handle the problem. If one of the ORed flags is active, the multi-way branch destination instruction can either ignore the flags or take a second multi-way branch that is based on direct inputs of the flags that were ORed in the first multi-way branch (an advantage of having more than one source for multi-way branch conditions). The second multi-way branch determines which of the ORed flags was active in the first multi-way branch.

FPP Status Register Implementation

An AmPAL22V10 Programmable Array Logic device is used to register the FPP status flags and perform the OR of some of the flags.

This external status register loads new status only as the result of cycles in which the FPP is the selected data path during an instruction execution. When the FPP "F" register is in transparent mode, the external status register is loaded with the flags at the end of an FPP cycle. This results in a one level deep pipeline on status in the same way that ALU status is pipelined one level internal to the ALU. When the F register is in clocked mode, the external status register will load in the cycle following an FPP cycle. This will capture the data that is loaded into the FPP on chip status register at the end of the FPP cycle. This causes the status to be double pipelined for cycles in which the F register is clocked.

The multi-way branch outputs for the first level branch are the following flags: Overflow, Underflow, Invalid, and the OR of the Inexact, OR, NAN, and Zero flags. The multi-way branch outputs for the second level branch are: Inexact, NAN, Zero, and Ground.

These groups of four bits are substituted for the least significant four bits of a branch address to act as a multi-way branch.

In addition to the multi-way branch test for flags, an added output of the status PAL ORs together the Overflow, Underflow, and Invalid flags for use as an interrupt signal to the system interrupt controller, thus giving one additional way to monitor the FPP error flags. Using the interrupt approach eliminates the need to follow floating point operations with multi-way branches in order to test for error conditions. Execution of instructions can proceed, assuming no major problems exist in an FPP cycle. If one of the above mentioned error flags is active, the resulting interrupt will deal with the error.

One last element of the status PAL is that it acts as part of the system control decode by decoding the data path select bits of the control pipeline to enable the FPP output when the FPP is the selected data path.

The logic definition file for the status PAL is listed in Appendix C.

Seed Look-Up Table

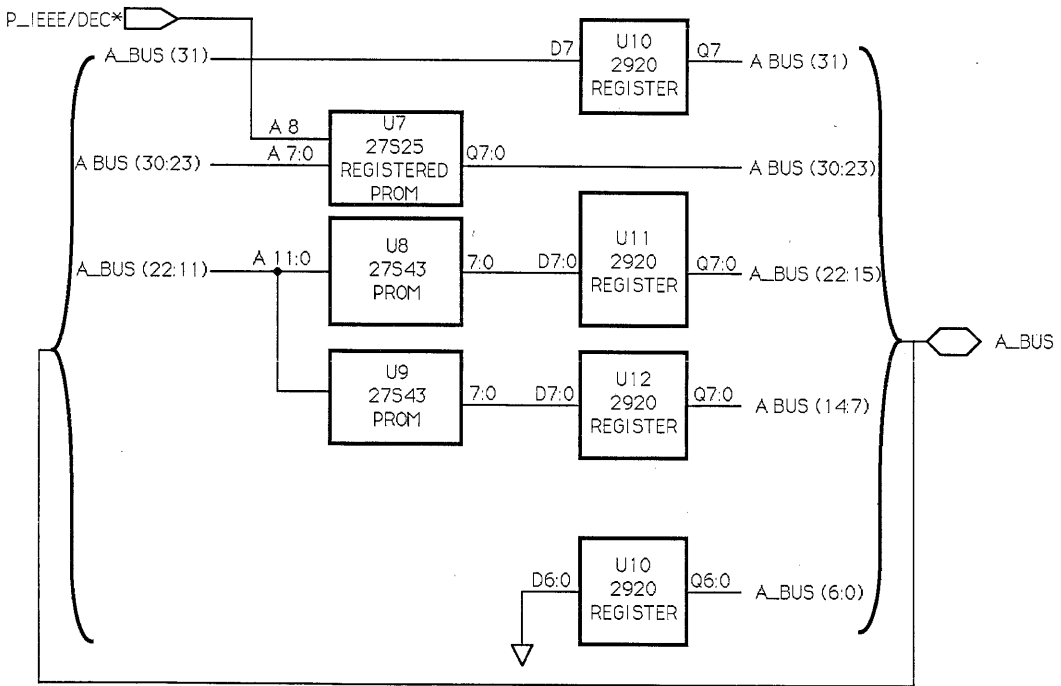
The Newton-Raphson division algorithm does a division of A by B by finding the inverse of B (i.e., 1/B) and performing a multiply against A. This scheme works with the Am29325 since finding the inverse of B requires only

a series of multiplies and subtracts which the Am29325 can do in single cycles. But, these multiplies and subtracts are performed only to refine the accuracy of a precalculated seed value (a rough approximation of the inverse of B). So a table of seed values must be available to support division with the Am29325.

This seed table is stored in PROM memory external to the FPP. The B variable is used to address the seed table, and the resulting seed value is fed into the FPP to be refined.

Placing the seed table in the path to one of the FPP inputs normally requires a 32-bit multiplexer to select between the PROM and the direct input bus for loading normal operands in multiply, add, and subtract operations. Building this multiplexer would require at least six hex-2-to-1 multiplexer chips. The PROM and multiplexer would also increase the propagation time needed to load the FPP, thereby requiring the cycle timing to be extended even more than is already required by the FPP.

The implementation of the seed table in this system has been modified to save chips and cycle length. Instead of placing the seed table between the A_BUS and the FPP, it is placed to the side as an appendage of the A_BUS (see Figure 3-3). The inputs and outputs of the table are tied together and to the A_BUS. The internal structure of the table is shown in Figure 3-4. It contains three PROMs, each of which is followed by a three-state output register (the Am27S25 has an internal register). In this arrangement the PROMs can be accessed by the value present on the A_BUS in one cycle and the resulting seed loaded into the registers. In the following cycle the registers can drive the A_BUS with the seed value. This scheme requires three fewer chips and no extension to the FPP cycle time. It is true that two cycles are now required to load the seed value but the cycle used to access the seed table can be combined with the operation of checking for a zero divisor. This operation is generally done during the setup for a divide.



09856A 3-4

Figure 3-4. Floating Point Block Seed Look-Up Table -- Data Flow Diagram

The detailed connections of the seed table are shown in Figure 3-5. The Am27S25 contains the seed values for the exponent and the two Am27S43s contain the seed for the fraction. The seed table output enable (SEED_OE*) signal is a decoded output of the microcode control pipeline register. The output register of the seed look-up table is clocked by the data section clock.

PARALLEL MULTIPLIER

The entire Parallel Multiplier (PM) block's function is provided by the single chip Am29C323 Parallel Multiplier. This chip performs 32-bit, 64-bit, 96-bit, and 128-bit integer multiplies. It also can perform multiply accumulate using an internal 67-bit accumulator. The PM is shown in Figure 3-6.

Most of the control signals come directly from the control pipeline register. The Parallel Multiplier output enable (PM_OE*) is decoded from the data path select field of the microcode pipeline register. The enable and flow through controls for the instruction register (ENI* and FTI) are tied respectively to GND and VCC to allow instructions to flow directly from the microcode pipeline register to the multiplier, since the microcode pipeline register already provides the one level of pipeline required in the system. The flow through enable on the product register is enabled only when the PM data path is selected via the control decode logic.

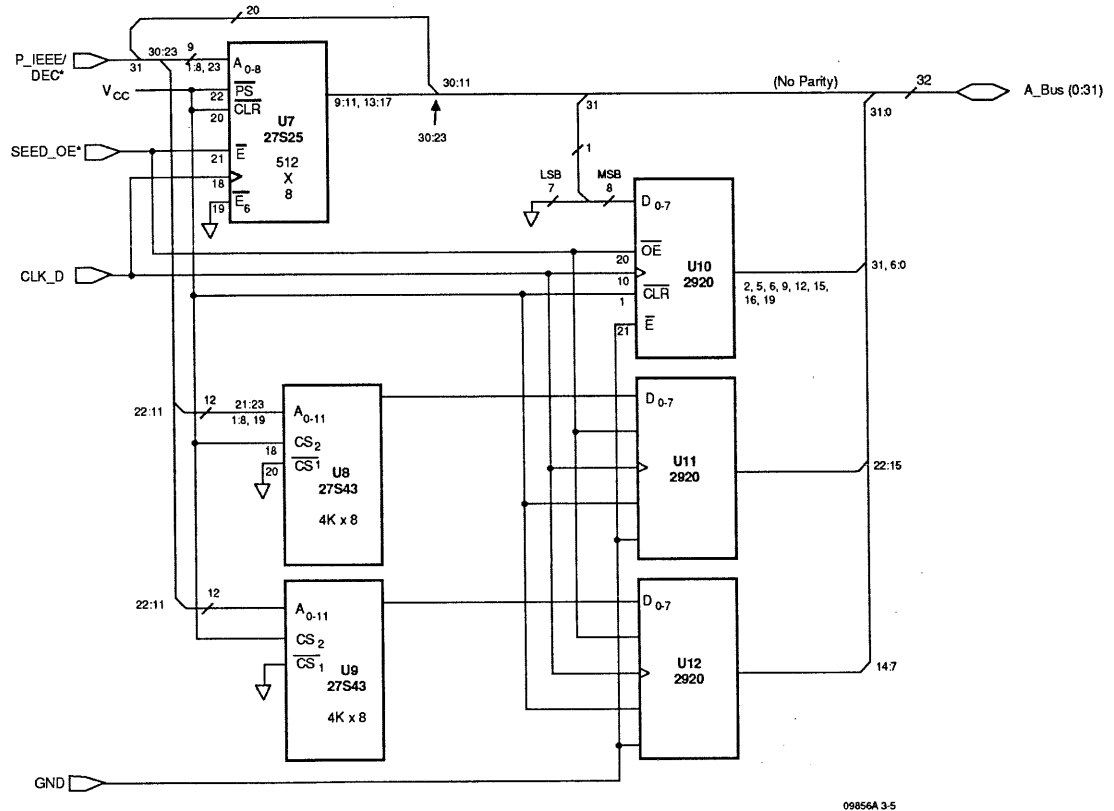


Figure 3-5. Floating Point Block Seed Look-Up Table -- Implementation

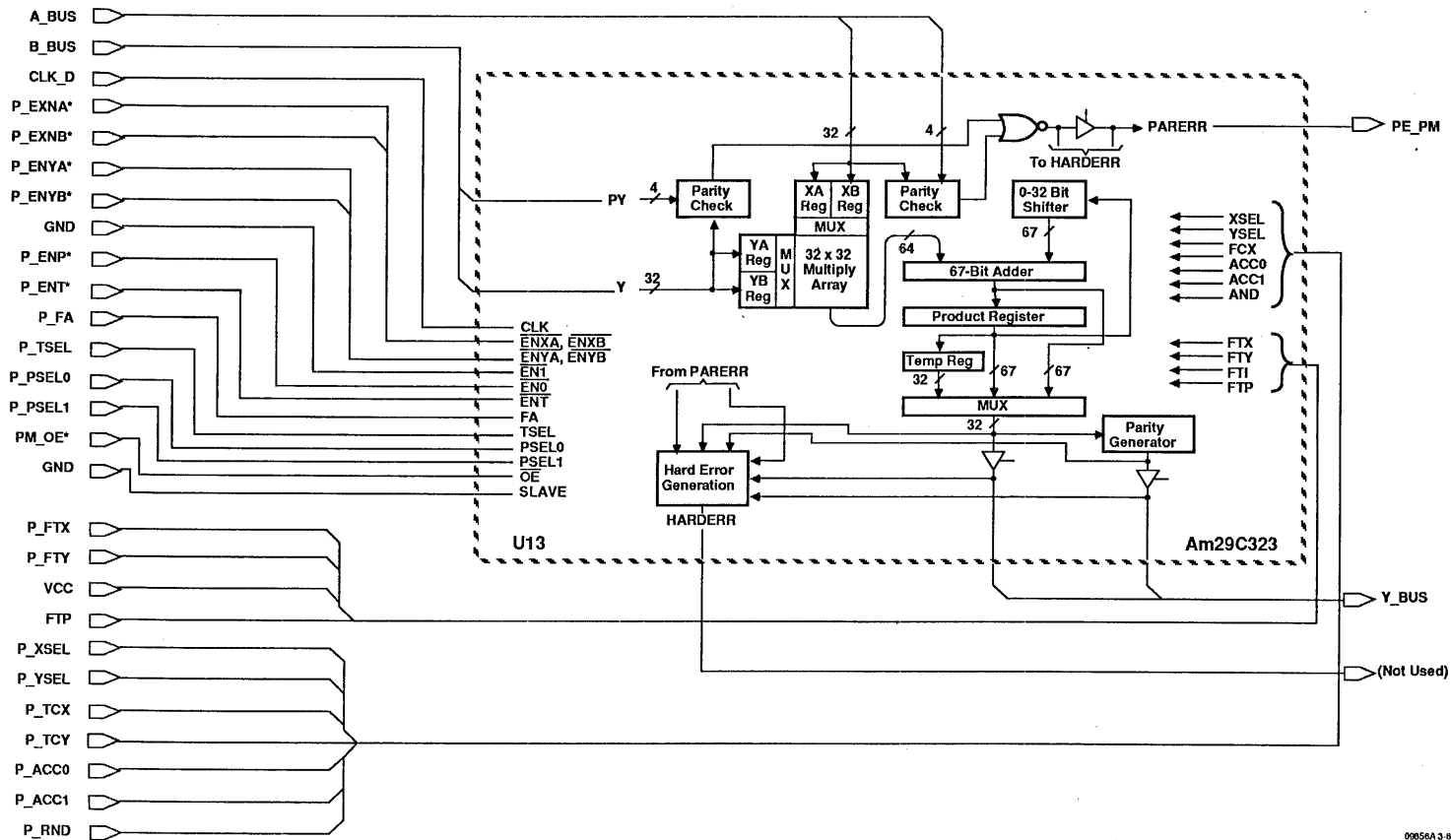


Figure 3-6. Integer Multiplier Block

SECTION 4

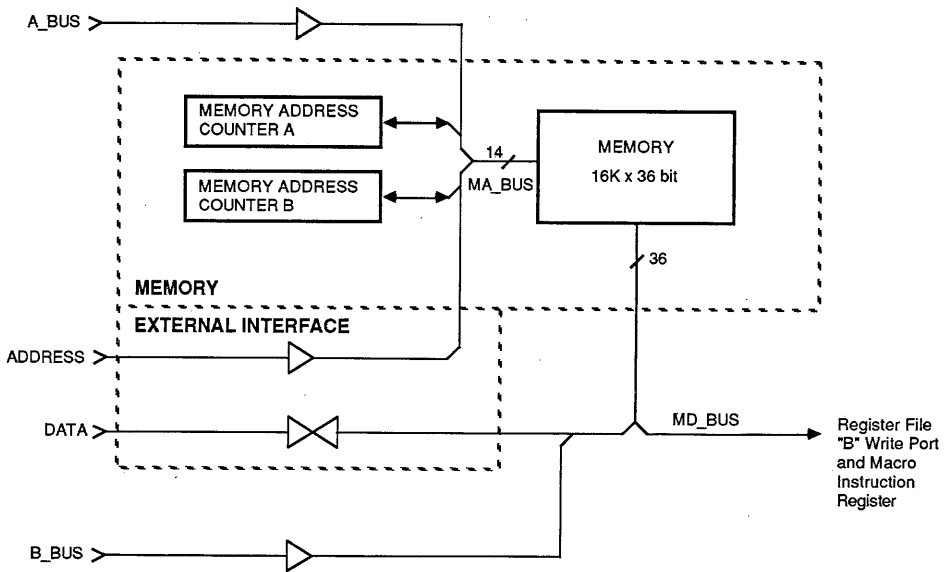
Memory and External System Interface

The memory block and external system interface are discussed together in this chapter because of the tight interconnection between these areas. It is helpful to view the two blocks together in order to understand the shared use these blocks make of the memory address bus (MA_BUS) and the memory data bus (MD_BUS). Figure 4-1 shows a block diagram of the data and address paths used in these sections.

One thing to note is that both the memory and the external interface are not elaborate in design. Essentially the external I/O section of this system is just a second port on the system memory. This system does little more than provide a simple arbitration scheme on access to the memory that allows an externally supplied DMA device to load and retrieve data from the memory. Event

or interrupt signaling between the CPU and host system is limited to a single pair of interrupt signals, one from host to CPU, one from CPU to host. Memory itself is only a simple bank of static RAM with two address counters on the input that help speed up array calculation.

The reason for this simple approach is that the design to the CPU using the Am29300 family of building blocks is the focus of this application note. Every reader who may find the information in this application note useful will have different memory and I/O requirements to handle and will very likely design individual approaches to memory and I/O. Therefore, only this simple approach is covered here so that more time can be spent discussing the CPU design.



09856A 4-1

Figure 4-1. Memory and External Interface Address and Data Paths

EXTERNAL BUS INTERFACE CONTROL

Host Access Definition

A block diagram of the host interface controller and its connection to the MA_BUS and MD_BUS buffers is shown in Figure 4-2.

The Am29300 demonstration system is treated as a co-processor to some host system. It ultimately gets all of its instructions, data, and control from the external host system. To provide communication with the host using a minimum of design effort and special hardware, only two portals into the Am29300 system are allowed.

One portal is the Am29300 memory, which is treated as a dual port memory with all words directly mapped into the host bus address space. With this, the host has complete access to macroinstructions and data going into and out of the system.

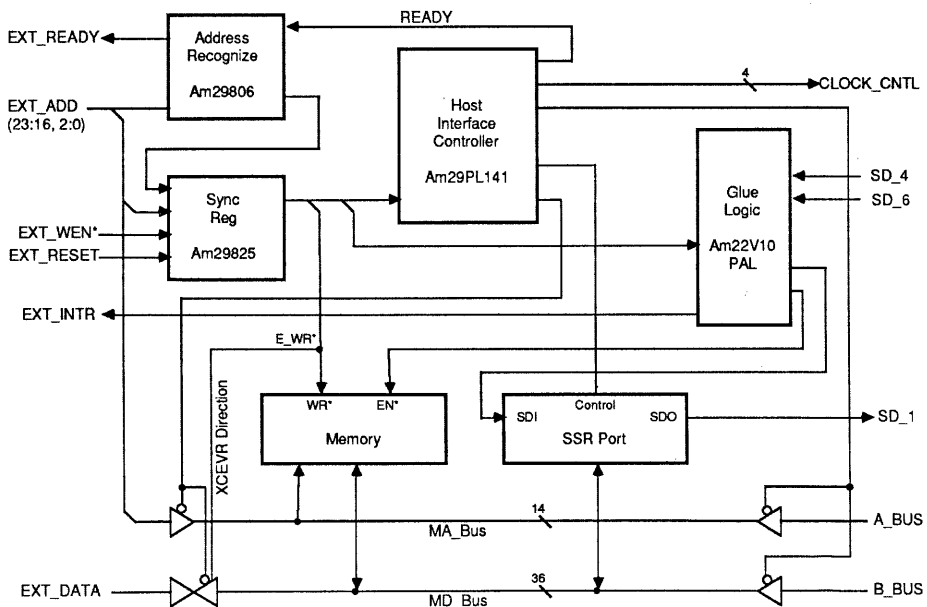
The second port is a serial diagnostics shift chain that runs through key control registers of the system. This serial pathway gives access to loading and reading the microcode writable control store, to the control pipeline register, to loading and reading the macro opcode map RAM, to the macro opcode register, to the macro status register, and to the interrupt base address register.

Through this serial port, the microinstructions are loaded by the host before program execution begins. Also, the system clocks can be controlled by the host to allow diagnostics and code debugging via single stepping and breakpoints.

These portals are controlled by a state machine that is separate from the Am29300 system. The state machine is referred to as the host interface controller. It constantly monitors the external host address bus. When the host presents an address that matches a preset address on the Am29300 system board, the host interface controller is selected to perform one of several interface functions.

Any function requested by the host takes priority over anything that the Am29300 CPU is doing. The host always gains control of the memory address and data buses as soon as the CPU clocks can be stopped and the CPU to memory bus buffers disabled.

The function performed is dependent on the address used, thus the commands from the host to the interface controller are memory mapped. A 24-bit address from the host is assumed for this design. The 6 most significant bits (23:18) of the address are matched to the Am29300 system board address to select the host interface controller. The next two most significant bits (17:16) are used to select a command mode. The 3 least significant bits (2:0)



09856A 4-2

Figure 4-2. Host Interface Block Diagram

are used to select a specific command function within two of the command modes.

Host Interface Block Diagram

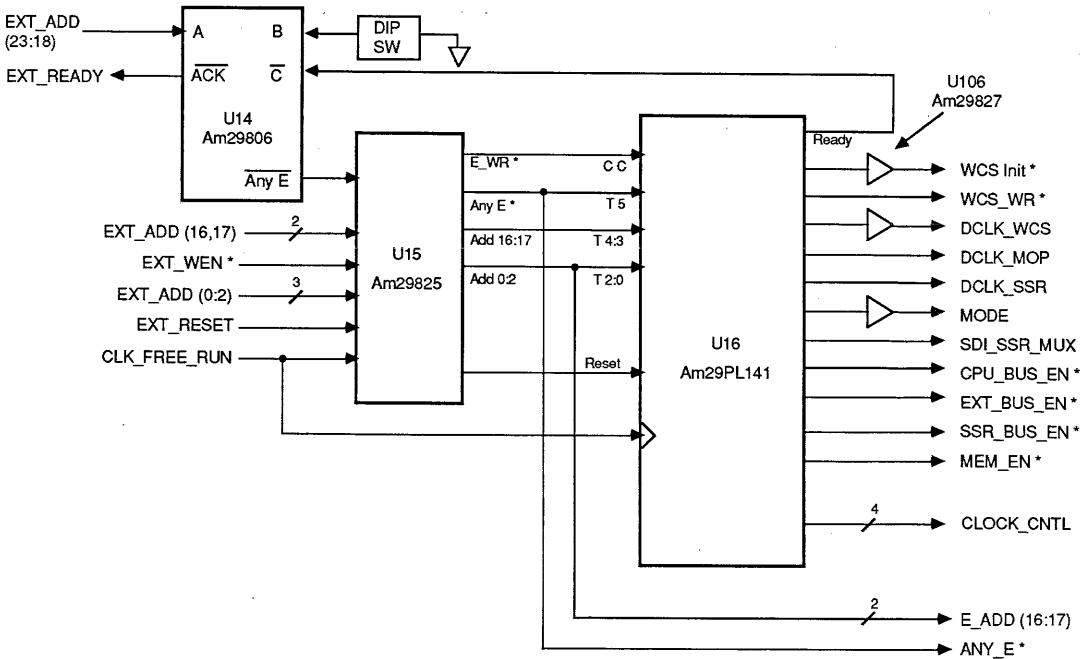
The 6 most significant bits of the host address are checked by the address recognition block: if the address matches the board address, then the match signal is fed into the input of a synchronizing register. Also fed into this register are: the external bus write enable line (EXT_WEN*); the external address bits 17, 16, 2:0 [EXT_ADD(17,16,2:0)]; and the host system reset line.

The synchronizing register is clocked by a free-running version of the Am29300 system clock. The register used has special meta-stable hardened circuitry that prevents the outputs from oscillating, regardless of the timing relationship of input data to clock. This register allows the entire Am29300 system to run asynchronously with regard to the host system clock. All the interaction between the host system and the Am29300 system is synchronized to the Am29300 system clock by the register. Each command to the host interface controller is thus presented at the output of this register in synchronization with the host interface controller clock.

The heart of the host interface is an Am29PL141 Fuse Programmable Controller. It is a microprogrammed sequencer with on-chip microcode memory and pipeline register. This sequencer implements the state machine functions needed to control the interaction between the host and the Am29300 system. Used with the Am29PL141 is an Am22V10 PAL. This PAL collects together some glue logic functions: an interrupt signal latch, a multiplexer, and some encoding logic, all of which are described later.

The Am29PL141 provides control signals to the clock gating and distribution section of the Am29300 system. It also controls the enabling of all the buffers and transceivers that connect with the MA_BUS and MD_BUS. The controller acts as a "traffic cop" that allows only one driver on those buses at a time to prevent contention. The controller also manages the loading, reading, and shifting of the Serial Shadow Register diagnostic chain.

The Serial Shadow Register (SSR) diagnostics port is a 32-bit-wide parallel read and write register that also functions as a shift register. Data to be read or written to the SSR diagnostic chain is loaded or read via this port. The port is connected to the host via the MD_BUS. The



09856A 4-3

Figure 4-3. Host Interface Controller

port is built from four Am29818-1 SSR diagnostic pipeline registers. These registers, like all the registers in the diagnostics chain in this system, contain one normal parallel input and output pipeline register that is backed-up or "shadowed" by a second parallel input and output register that also acts as a serial shift register. The pipeline register can be loaded from the shadow register and the shadow register can be loaded from the outputs of the pipeline register. This gives the ability to move data into or out of the pipeline register via the shadow register. Data in the shadow register can be serially shifted to other similar registers in the system. By connecting all the diagnostic serial shadow registers together in a serial chain, data can be moved serially through a large number of key registers in the system using very few wires.

The SSR diagnostics port is just an extra section of the diagnostics chain that runs throughout the Am29300 system. This extra section is connected to the MD_BUS to serve as a parallel input and output port that gives access to the serial shadow register chain.

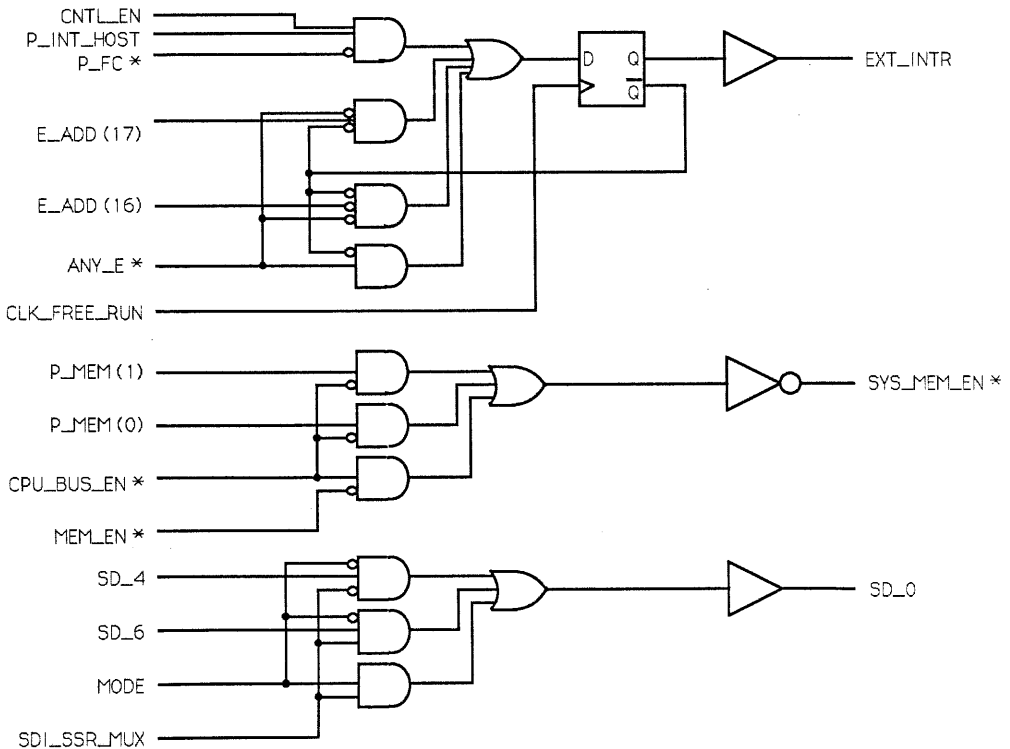
A slightly more detailed view of the Host Interface Controller is shown in Figures 4-3 and 4-4.

Event Signals

The host and the Am29300 system need to be able to signal each other when important events occur, such as the transfer of ownership over sections of the dual port memory. To allow this, a simple interrupt setting and clearing scheme is provided.

The host interrupts the Am29300 system with a command to the host interface controller. The controller in turn sets an interrupt flag in the Am29300 system interrupt controller. The interrupt is cleared when the Am29300 services its interrupt controller.

The Am29300 interrupts the host by using a microcode bit to set a latch that drives an interrupt line on the external bus. The interrupt is cleared whenever the host does an operation on the SSR port. The interrupt latch is implemented in the AmPAL22V10, as shown in Figure 4-4.



09856A 4-4

Figure 4-4. U17 Am22V10A Host Interface Glue Logic

Memory Enable

The Am29300 system memory can be enabled by either the Am29300 microcode or by the host interface controller. A simple multiplexer is needed to direct the correct control signal to the memory enable input. This logic is also implemented in the AmPAL22V10 shown in Figure 4-4.

AmPAL22V10 Support Logic

Figure 4-4 shows the logic for the AmPAL22V10 that integrates the interrupt signal latch, SDI multiplexer, and memory enable logic. The logic equation definition file for this PAL is listed in Appendix D.

SSR Diagnostics

SSR Shift Path

Figure 4-5 shows a block diagram of how the serial shadow registers in the system are linked together and how they relate to the macro opcode map RAM, se-

quencer, and microcode control store. Most of these registers are also depicted in other Figures throughout this application note in their rôles as parallel input and output pipeline registers. Figure 4-5 emphasizes the serial in and out and control connections of the shadow registers also contained in these registers.

The SSR diagnostics port is shown as the starting and ending point for the entire shift chain (or loop as seen here). Data to be loaded into the SSR loop is parallel loaded into this register from the MD_BUS via the bidirectional outputs of the registers in this port (note: the shadow register in the Am29818-1 gets its input from the output pins of the Am29818-1 pipeline register).

Data loaded into this shadow register is then shifted into one of two branches of the SSR loop. One branch flows through the Writable Control Store (WCS) port and the microcode control store pipeline shadow registers. The WCS port is used to address the microcode control store or to receive (load) data from (to) the macro opcode map RAM. The microcode control store shadow register is used to write data into the microcode writable control store or to read the contents of the control pipeline

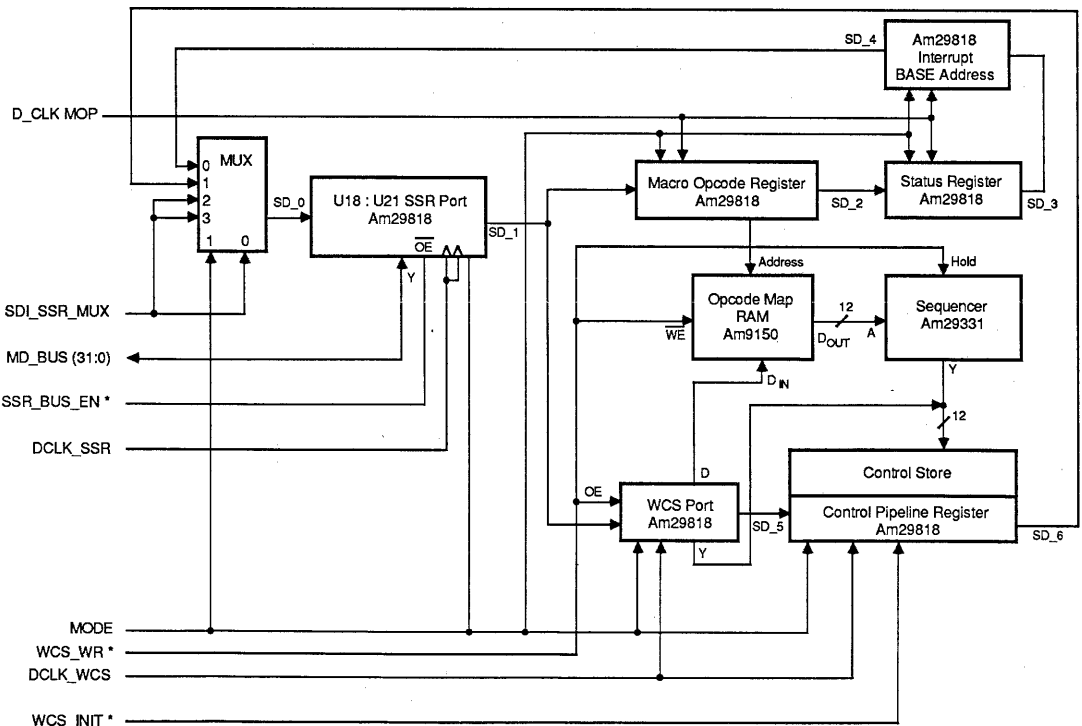


Figure 4-5. Serial Diagnostics Shift Path

09856A 4-5

register. The second branch flows through the macro opcode, macro status, and the interrupt base address registers. The macro opcode register is used in part to address the macro opcode map RAM .

These branches are separate because it helps to shorten the shift chain length by using branches and because the shift chain clock to the writable control store and WCS port must be separate from the shift clocks to the rest of the diagnostics chain. The shift clocks must be separate because of the way the writable control store is loaded.

The data outputs of the control store are connected to the inputs of the pipeline register as required for normal use in the system. To write the memory, the inputs must be driven with the data to be written, turning the input pins into outputs. In the Writable Control Store (WCS) pipeline register this is fine, since the memory outputs are disabled during the write.

If other diagnostic registers in the system were tied to the same shift clock and mode control lines as the WCS pipeline, there could be a problem every time the WCS is written. The other diagnostic registers not involved in the WCS write would see the same control signals as the WCS registers and would drive their input pins. Depending on what the other registers were connected to, this situation could cause serious contention problems through the system.

For this reason, the SSR used to load WCS is treated separately from other SSR registers in the system. It is worth noting that the only control signal that need be separate is the shift clock. The mode and serial path may be shared with all SSR in the system. Putting the SSR into WCS loading mode, requires the shift clock to load an internal mode flip flop. If the shift clock is active only to the SSR used for WCS when the MODE and Serial Data In (SDI) signals are set high, only the WCS SSR will go into the input pin driving mode.

The end of each branch in the SSR loop returns to a multiplexer at the serial data input (SDI) of the SSR diagnostics port. This multiplexer allows the selection of the shifted branch into the port when the SSR loop is being read rather than written. It also allows the SDI value to be forced when the MODE signal is high. When the MODE signal is high, all the SSRs in the system pass

their SDI directly to their Serial Data Output (SDO). This causes the SDI value forced at the input of the SSR port to be passed directly to all SSRs in the system (note: significant propagation time from SDI to SDO for each SSR is involved). In this way the forced value of SDI becomes an additional control signal to all the SSRs in the system. The function of this multiplexer is integrated into the AmpAL22V10 as shown in Figure 4-4.

SSR Reading and Writing

To read the contents of the pipeline registers in the Am29300 system, the host must first send a command to load the SSR throughout the system from the pipeline registers. Then the host must shift the contents of the SSR into the SSR port register (up to 32 bits at a time). The host then performs a read of the SSR port. The host then repeats the shifting-and-reading process until the entire SSR chain has been read.

To write the system pipeline registers, the host reverses the above procedure. Data is first written into the SSR port. Then the SSR chain is shifted to move data into position. The SSR port loading and SSR chain shifting go on until the section of the SSR chain desired is filled. Finally a pipeline load command is issued by the host to load the contents of the SSR into the pipeline registers.

To write the macro opcode map RAM and the microcode writable control store (note: these are treated as a single WCS and must be written together), an address for the map RAM is first loaded into the macro opcode pipeline register via the method described above. Then the address for the microcode WCS is loaded into the WCS port pipeline register. Next, the data to be written into the map RAM and into the microcode WCS is shifted into the WCS port SSR and WCS SSR. A load WCS command is then given which performs the actual write of data into the memories. During the write operation the output of the WCS port is enabled and the Am29331 sequencer output is disabled (via its HOLD pin).

The only trick involved in the SSR Reading and Writing is knowing how much to shift the SSR during each read or write. The problem is that the SSR chain length in this system (and in nearly every real system) is not an even multiple of the SSR port size. During the first (or last) shift operation of either the read or the write of pipeline

registers, it will be necessary to shift fewer than the full 32 bits of the SSR port. The number of bits to be shifted depends on the chain length. One thing to note is that the chain length will be in a multiple of 4 bits because diagnostic pipeline registers are currently available only in 4-bit and 8-bit devices. So, when a shift operation is commanded by the host, the number of nibbles (4-bit shifts) to be shifted must be indicated.

A final note: during the shifting of the WCS SSR, the Am29300 system clocks must be halted. This is due to the fact that pipeline clock and shift clock to the Am9151 may not occur within 65 ns of each other. Since these clocks would occur within the above window in this system, the pipeline clock must not be active.

Controller Description

Function/Command Descriptions

The following is a list of the address values for functions that the host interface will perform when addressed by the host:

Memory Access: Reading and writing of the Am29300 system memory is done by selecting the address for the Am29300 system with address bits 16 and 17 equal to zero. The address for the specific word in memory is contained in address bits 0:15. The host interface controller, upon recognizing the host access, will stop the clocks to the Am29300 system and disable the CPU to MA_BUS and MD_BUS buffers. At the same time the external bus to MA_BUS and MD_BUS transceivers are enabled. This suspends the operation of the Am29300 system and gives memory access to the external host. The write enable line on the external bus determines whether a read or write occurs.

Note that by suspending the Am29300 system operation, the memory access is transparent to (or hidden from) the CPU. There is no action required on the part of the Am29300 microcode or interrupt control.

Serial Diagnostics Port Access: This access is very similar to that of a memory access. The difference is that the SSR port register is being read or written instead of memory.

ADDRESS BITS					FUNCTION
17	16	2	1	0	
0	0	x	x	x	Am29300 Memory Access
0	1	x	x	x	Serial Diagnostics Port Access
1	0	0	0	0	Illegal code
1	0	0	0	1	Halt CPU
1	0	0	1	0	Run CPU
1	0	0	1	1	Single Step CPU
1	0	1	0	0	Single Step CPU Control Section
1	0	1	0	1	Single Step CPU Data Section
1	0	1	1	0	Interrupt CPU
1	0	1	1	1	Reset CPU
1	1	0	0	0	Illegal code
1	1	0	0	1	Load Pipeline Register
1	1	0	1	0	Load Macro Opcode Register
1	1	0	1	1	Load Writable Control Store
1	1	1	0	0	Load Initialization Register
1	1	1	0	1	Load Serial Shadow Register
1	1	1	1	0	Shift WCS SSR Chain
1	1	1	1	1	Shift Macro Opcode SSR chain

Halt CPU: This command throws the Am29300 system clocks in to a continuous stop condition until the mode is cleared by the RUN CPU command or temporarily overridden by one of the single step commands.

Run CPU: This command starts the Am29300 system clocks running.

Single Step CPU: When the CPU is halted, this command will cause all the system clocks to cycle once to advance the state of the CPU one step. Note that gated clocks will be active during this cycle only if their enables are active (i.e., gated clocks operate as they would during a normal clock cycle; they are not forced to operate).

This mode is useful during diagnostic operations to single step the machine between serial load and unload of the SSR diagnostics.

Single Step CPU Control Section: This will step only the clocks in the control section of the CPU. The control pipeline, macro opcode, macro operand, status, sequencer, and interrupt registers may be affected.

This is useful for forcing the control section into a new state under the control of diagnostics, such as a forced branch to a new location in the microcode. This is done by first loading the control pipeline with an instruction to branch via the SSR diagnostics chain. The control section would then be single stepped to execute the branch. Note that during these operations, the data section is not affected and no data is modified.

Single Step CPU Data Section: This operation single steps the clocks only in the data section of the CPU. This may be useful for repetitive diagnostic operations involving only the data section.

Interrupt CPU: This command causes the host interface controller to set an interrupt input to the Am29300 system interrupt controller. The interrupt controller in turn prioritizes the interrupt and causes an interrupt to the CPU when that type of interrupt is enabled.

Reset CPU: This will make the reset line to the Am29300 system active and step all the ungated system clocks. The clocking is required by some parts of the system to affect reset state changes.

Load Pipeline Register: This command will step only the clock to the control pipeline and WCS port for one cycle while forcing the pipeline registers to load data from the SSR chain. This is used to control the state of the pipeline through serial diagnostics.

Load Macro Opcode Register: This steps only the clock to the macro opcode, macro operand, status, and interrupt base address pipeline registers while forcing the registers to load from the SSR chain.

Load Writable Control Store: This command initiates a series of clock cycles that cause data in the SSR chain to be loaded into the writable microcode control store and the macro opcode map RAM from the SSR chain. The address loaded is also specified in the SSR chain.

Load Initialization Register: Like the previous command, this operation loads the writable microcode store. The difference is that only the WCS (Am9151) initialize registers are loaded from the SSR chain.

Load Serial Shadow Register: This causes the contents of all diagnostic pipeline registers to be copied into the related SSR chain elements. This is used to read the Am29300 system state into the SSR chain so that it can be shifted out to the host.

Shift WCS SSR Chain: This command shifts the contents of the SSR port register into the SSR diagnostics chain used for the writable control store. It also brings the bits at the end of the WCS SSR chain into the SSR port register. This is the serial read and write operation of the WCS SSR chain (or loop).

Shift Macro Opcode SSR Chain: This is the same as the previous command but it affects the SSR chain associated with the macro opcode, status, and interrupt base address registers.

Illegal Code: Due to the way the host interface controller algorithm was implemented, this command (address combination) is illegal. If it is used, it will lock up the host interface controller in an infinite loop.

Access Timing

The speed of interaction between the host and the Am29300 system is regulated by both the host and the host interface controller.

Once the Am29300 system is addressed by the host, the host interface controller holds the external bus by driving EXT_READY inactive. This continues until the host interface controller completes the command requested. The EXT_READY signal is then made active and held active until the host stops addressing the Am29300 system. At that time, the host interface controller recognizes that the host has completed the transaction and the EXT_READY line is again made inactive.

In this fashion, either the host interface controller or the host can extend the length of the external bus transaction as required. The signal timing between the host and the host interface is treated as asynchronous. The timing of the host interface itself is synchronous with the Am29300 internal clock cycle.

An interaction diagram is shown below for a bus transaction between the host and the Am29300 system. The single-line dividers indicate one clock cycle of the

Am29300 system. The double-line dividers indicate one or more clocks as needed for synchronization or algorithm execution.

The length of an external bus transaction can vary from about 6 Am29300 system clock cycles for a memory access, to about 80 clock cycles for an SSR shift operation. Regardless of the transaction type, the Am29300 system looks to the host like a slave bus peripheral. Sometimes, as in the case of the SSR shift operation, it is a rather slow peripheral.

External Bus Activity	Am29300 System Activity
Address to Am29300 is active on the bus.	CPU is active. CPU owns MA and MD bus.
Address is clocked into the host interface controller synchronizing register.	CPU is still active. CPU still owns internal bus. Host interface controller performs branch to command routine.
External bus transceivers are enabled if needed.	CPU clocks are stopped. CPU bus buffers are disabled. Host interface executes first instruction of command routine. READY may or may not be made active depending on routine.
If READY is inactive, wait for host interface to complete algorithm and make READY active. CPU operation is still suspended.	If READY is active, then wait for host to release external bus by stopping selection of the Am29300 system.
External bus address no longer selects Am29300 system.	CPU still suspended. Host interface waiting to see host release bus.
Lack of external bus address is clocked into host interface sync register.	CPU still suspended. Host interface branches back to idle loop.
External bus transceiver is disabled.	CPU clocks are active. CPU has MA and MD bus access. Host interface waits in idle loop for next command.

Program Definition

A detailed definition of the host interface controller's algorithm is contained in Appendix E.

MEMORY

Memory Components

The memory device used to construct the 16K word x 36-bit memory is the Am99C165. This is a 16K x 4-bit CMOS static RAM memory. The 35 ns access time version is assumed in any timing estimates for the Am29300 demonstration system. Nine memories are used as shown in Figure 4-6.

The Am99C165 is used so that an additional output enable is available to help prevent bus contention with other buffers on the MD_BUS. The memory outputs are disabled whenever the memory write enable line is active. The write enable line is also used to control the direction of the external bus data transceiver and the enable on the CPU data buffer. The delay of the inverter on the output enable input to the memory has been matched by a buffer in each of the other bus drivers just noted. This is so that when a write operation is signalled, each bus driver receives its bus enable or disable signal at the same time as the memory. This overlaps the turn off time of the memory outputs with the turn on time of the other bus drivers to minimize bus contention with the memory.

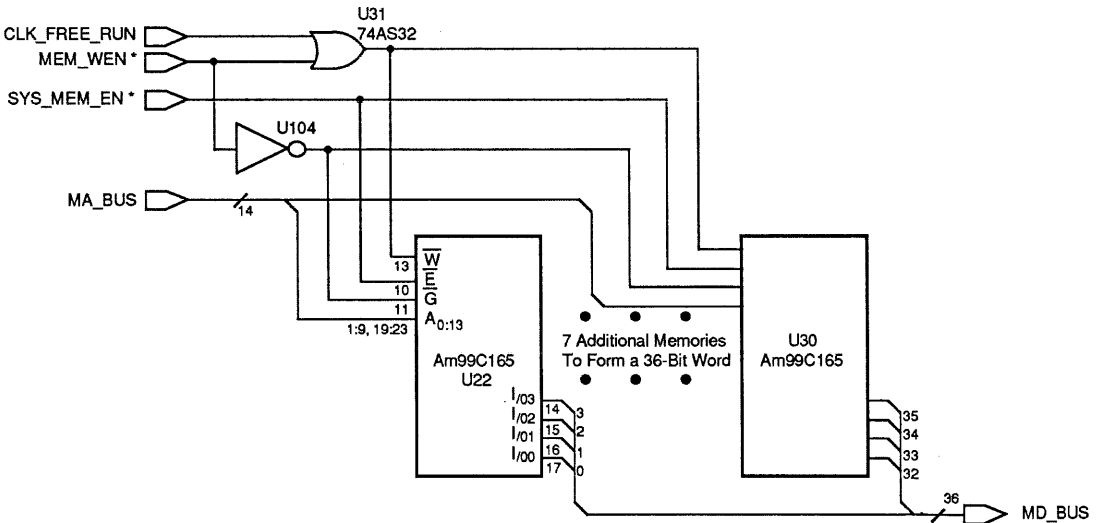
The enable line to the memory is used to power down the memory when it is not being selected by the Am29300 CPU.

The write enable line to the memory is gated with the Am29300 system free-running clock. This keeps the write line high (inactive) until late in the cycle when all the control signals that feed into the memory enable have settled. This is important for cycles in which there is a change of ownership on the memory address and data buses. The gating with clock ensures that unintended pulses on the write enable line that may occur early in the system cycle will not cause spurious writes in the memory.

Addressing Scheme

Description: With reference to Figure 4-1, the memory address bus (MA_BUS) is not only the address input to the memory, it is also a part of a 4 to 1 multiplexer. There are four address drivers tied to the MA_BUS. They are: the A_BUS to MA_BUS buffer, the External Bus address to MA_BUS buffer, and the two memory address counters. Each of these sources has three-state output drivers and, by careful control of which source is allowed to drive the MA_BUS at any one time, the sources form the 4 to 1 multiplexer.

In this way the memory can be addressed directly by the A_BUS or the External Bus. The memory can also be addressed indirectly by the A_BUS via the memory address counters.



The memory address counters are loadable up/down counters that can serve as address pipeline registers, sequencers, or stack pointers independent of the CPU's data section. They allow sequential reads or writes to memory by the CPU without requiring the CPU to calculate an address on every read or write cycle.

In fact, after loading a memory address counter with an initial address, the CPU can perform sequential read cycles while at the same time continuing to use the data section for other calculations. This is possible because of the dual write port design of the CPU register file. The memory data is loaded into the register file via the B write port while calculation results on the Y_BUS are stored through the A write port.

Two counters are provided to allow for consecutive A and B operand data fetches from two separate arrays of data without the need to constantly reload the counter values. Each counter is built from two AmPAL22V10 Programmable Array Logic (PAL) devices that act as two cascaded 7-bit loadable up/down counters. The counters are connected as shown in Figure 4-7. The logic definition file for the PALs is given in Appendix F.

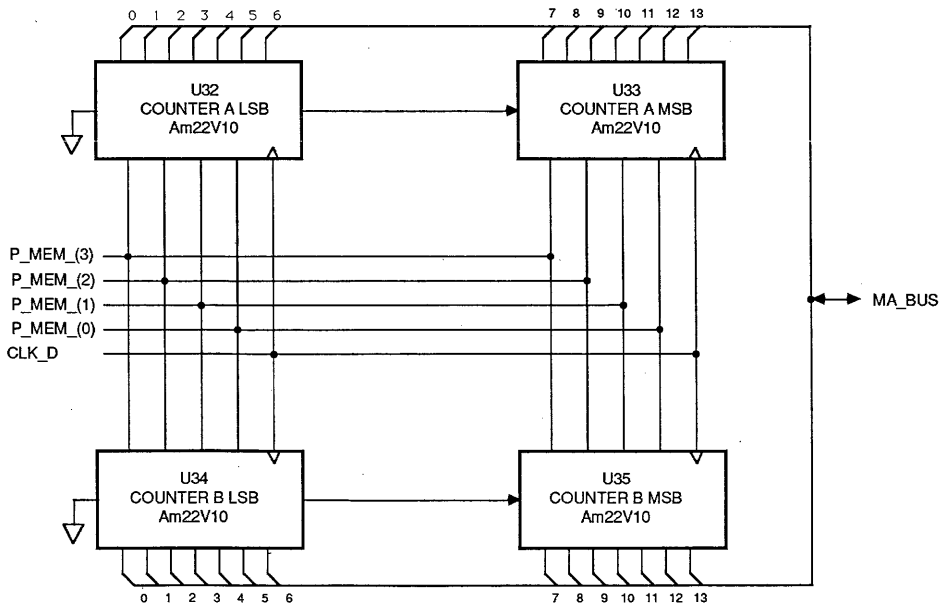
The two counters are only loaded from the A_BUS and not the External Bus, even though the connection of the counters to the MA_BUS would permit the latter. This is due to the difficulty in coordinating the use of the counters

between the CPU and the External Bus. The counters are simply viewed as a resource of the CPU only.

Why This Approach?: Why address the memory from the A_BUS? Doing so means that data in the memory is selected by an address previously stored in the register file. So one cycle must be used to calculate an address in the data section of the CPU, store the result in the register file, and take a second cycle to actually address the memory. Why not just take the address as it is calculated and feed it directly from the Y_BUS to the memory?

First, the access time is better from the A_BUS than from the Y_BUS. The A_BUS address is valid 45 ns into a cycle which still leaves time to access a fast static RAM in the same time that data would normally flow from the A_BUS through the ALU and back to the register file. An address on the Y_BUS would not be valid until 87 ns into a cycle, which would require either that the memory access extend the cycle length significantly or that the address be pipelined into a memory address register and be used to address the memory in a second cycle.

Second, since the register file can present two data words in one cycle it is possible to address the memory and provide write data in the same cycle; the address and data go from the register file to the memory. If the Y_BUS is used as the path to the memory in a write operation, a second cycle must be used to provide the write data.



09856A 4-7

Third, the above comments are trick answers. If the two approaches of A_BUS or Y_BUS as the memory address path are carefully examined it can be seen that it is really a situation of "six of one, or half a dozen of the other". Ultimately, in either case, a cycle is used to calculate the address and a second cycle is used to read or write the memory; there is only one data path in the system and only one calculation can occur in a cycle. Between the two approaches there are various ways to overlap other calculations with memory accesses to make the best use of the system's time but either approach takes the same time.

The real difference is that the A_BUS method is simpler from the microprogrammer's point of view. With the A_BUS method a memory read is done in one cycle and the resulting data is in the register file in the next cycle. With the Y_BUS approach there is a one cycle delay between a read access and the return of data, which requires that the microprogrammer "fill in the hole" in the microcode with other useful work to get the same system efficiency. So, as a designer's preference, the A_BUS for memory address approach is used.

CPU - Memory Buffers

The address buffers from the A_BUS to the MA_BUS and the data buffers from the B_BUS to the MD_BUS are shown in Figure 4-8. The address and data buffers are built from Am29827 10-bit-wide high speed buffers.

The address bus is 14-bits wide to address 16K words of 36-bit-wide memory. But these bits are taken from bit positions 2:15 of the A_BUS. This leaves the two least significant bits of the A_BUS unused and therefore treats the address as being in terms of bytes with the addressing restricted to four-byte (word) boundaries. This was done so that interface with an external host bus would be simpler. Many of the host systems with which this demonstration system could be mated use byte addressing. With the above address scheme, all the address line numbering is consistent between the host and CPU. In addition, if there were a future need to allow byte addressing of the CPU memory, it would be possible with only a minor change to the address buffer wiring. Also, it

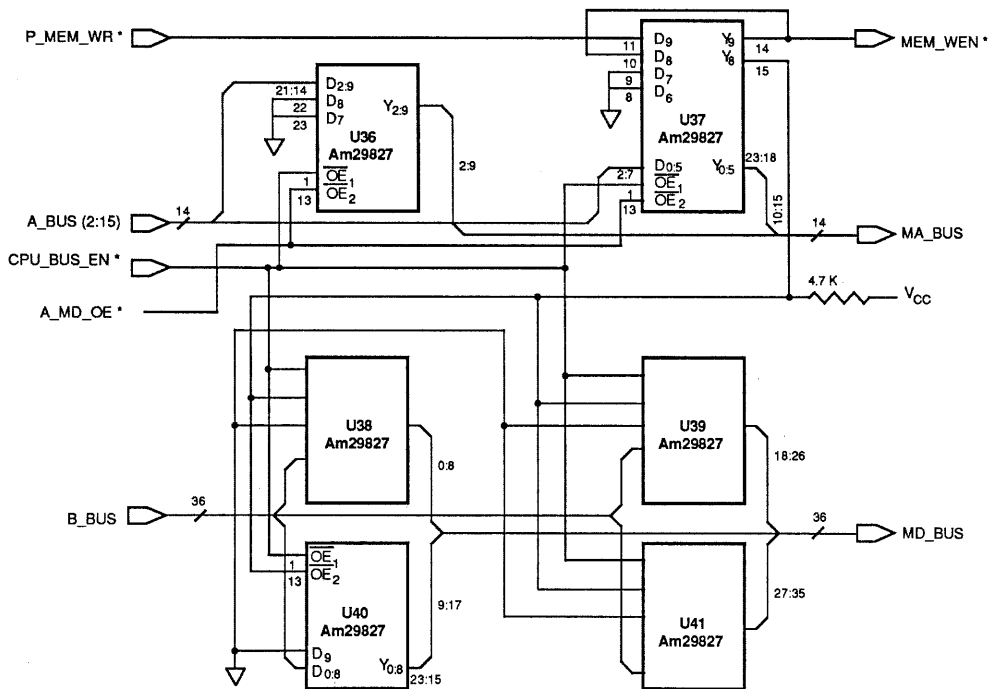


Figure 4-8. CPU to Memory Bus Buffers

09856A 4-8

may be noted that the parity bits on the A_BUS have been ignored in the MA_BUS since there is no parity checking implemented on the memory address.

The data buffers are arranged as one buffer per byte of the B_BUS (with parity on each byte). Note that, since the B_BUS provides only write data, and read data from the memory is received by the register file, only a unidirectional buffer is needed.

Whenever the external bus interface does not have the memory buses in use, the CPU to memory buffers receive the CPU_BUS_EN* signal to enable the buffers. If the operation is a write, the CPU_WEN* signal is provided by the CPU.

Note that the CPU_WEN* is routed through the address buffer twice and then to the data buffer to enable it on a write operation. This is done to help equalize the timing between this buffer and the output enable on the memory. Note also that the address buffers have a second enable input that is controlled by the control pipeline bits that manage whether the memory address comes from the A_BUS or from one of the memory address counters.

External System Buffers

The address buffers from the External Bus to the MA_BUS and the data buffers from the External Bus to the MD_BUS are shown in Figure 4-9. The address bus is built from Am29827 10-bit-wide high speed buffers. These buffers are connected in exactly the same way as described above for the CPU to memory address buffers.

The data buffers are, however, different from the earlier circuit description. These buffers are Am29863 non-inverting 9-bit high speed transceivers. The transceivers allow data to be both read and written by the external bus.

When the external host system addresses the Am29300 CPU memory, the external bus interface controller halts the system clocks in the CPU and disconnects the CPU from the MA_BUS and MD_BUS by making CPU_BUS_EN* inactive. Then the external bus is connected to the memory by making EXT_BUS_EN* active to enable the external bus buffers. The external bus supplies a write enable if the operation will be a write. Note again that the write enable timing is equalized with that of the write enable to the memory.

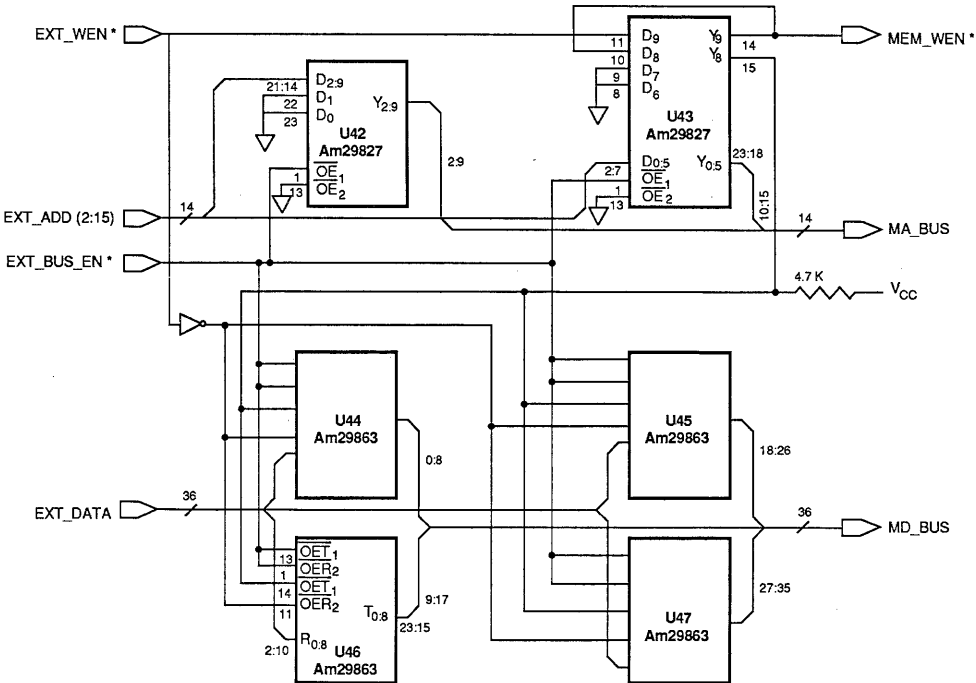


Figure 4-9. External Bus Buffers

09856A 4-9

SECTION 5

Control Section Description

MACRO OPCODE SUPPORT

Macro Opcode Register

In order for the control section of the CPU to make use of a macroinstruction, the instruction must be selected from memory and loaded into a register that is accessible to the control section.

This register is called the macro opcode register. It is a 32-bit register made from four Am29818-1 pipeline diagnostic registers. This register is shown in Figure 5-1.

The most significant 14 bits (bits 31:18) of the register output are used as the macro opcode. Bits 31:22 are connected to the address inputs of the macro opcode

map RAM. Bits 21:18 are connected to one of the Am29331 sequencer's multi-way branch inputs. These lower four bits may thus be used as an opcode modifier via a multi-way branch.

Bits 17:0 are the instruction operand register addresses. These bits are divided into three 6-bit fields, one for each register file port. Bits 17:12 are used as the register file 'A' read port address. Bits 11:6 are used as the 'B' read port address. Bits 5:0 are used as the register file 'A' write port address. These addresses are respectively referred to as the 'A', 'B', and 'C' operand register addresses.

These three addresses allow macroinstructions to specify directly three address operations with two read operands and a separate write operand. Note however that

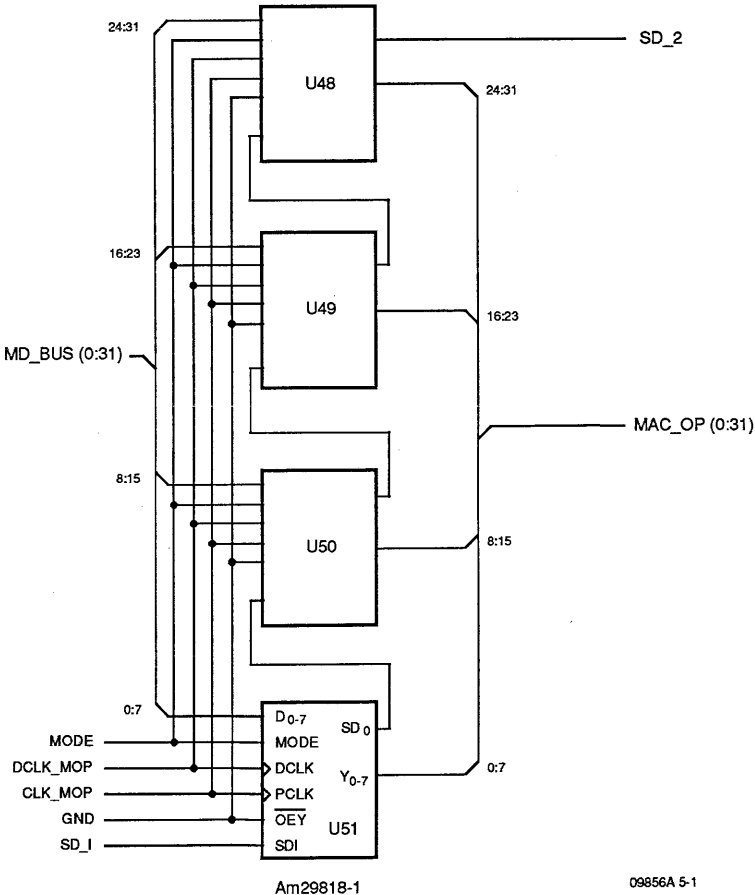


Figure 5-1. Macro Opcode Register

that these bits are connected to the macro operand address counters, which in turn are used to address the register file. This is more fully described in a later section.

In addition, bits 23:18 are connected to the position multiplexer. This allows macro instructions to specify directly the ALU position input as the lower bits of the opcode. Taking the position information from these bits still leaves all of the operand register addresses free for use in three address operations.

Also, bits 4:0 are connected to the width multiplexer. This allows macro instructions to specify directly the width input of the ALU for use in masked operations. Although this overrides this field of the opcode for use as the 'C' operand address, the 'C' operand address may internally be specified as the same as either the 'A' or 'B' operand register addresses. Thus two address macroinstructions involving width, or width and position specifiers are possible.

Macro Opcode Format Restrictions

Because of the large number of possible macroinstruction formats, this application note will not attempt to provide a detailed macroinstruction set definition. It is only important that the format restrictions imposed by the hardware design be stated.

As defined by connections of the macro opcode register, the macro opcode must always be located within bits

31:22. The size and position of the opcode within this field are determined by how the macro opcode map RAM is set up to interpret and map the opcode. The optional opcode modifier (multi-way branch input) must be in bits 21:18 if it is used.

The optional position field must be in bits 24:18 if used and the optional width field must come from bits 4:0 when used.

All three of the operand register addresses are optional and if used must come from the fields specified in the last section. The operand positions are fixed for the 'A' and 'B' operands since they may only come from the 'A' or 'B' operand bits of the macro opcode register. The 'C' operand address may come from any of the three operand fields.

The reason that the 'A' and 'B' operands do not share the positional flexibility of the 'C' operand is that the 'A' and 'B' operands specify registers to be read from the register file. These read addresses are in the critical timing path for the system, and any excess delay in selecting the address adds directly to the system cycle time. A multiplexer like that used for the 'C' operand address would add undesired cycle lengths. The 'C' operand address may afford its multiplexer delay since the 'C' operand address is not used by the register file until late in the machine cycle.

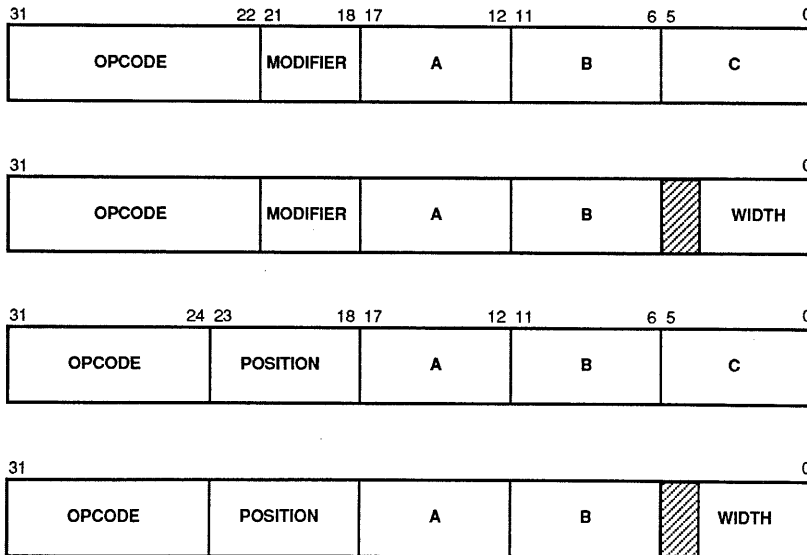


Figure 5-2. Example Macro Opcode Formats

09856A 5-2

Each operand address is optional, because the operand address may always be specified in the microcode.

Any optional field, even an unused portion of the opcode field, may be used as a data operand. Where a field is not used as part of the instruction control, it may be treated as data by loading the macroinstruction into the register file. Once the instruction is in the data section of the system, any data field may be extracted and used in calculations.

Some example macroinstruction formats are shown in Figure 5-2. The instructions are shown in a 32-bit word layout (byte parity is ignored for the moment).

Macro Opcode Decoding Method

The opcode portion of the macroinstruction is the index into the control store for the location of the first instruction of a microcode subroutine. Translating the bit pattern of the opcode into the microcode store address may be done several ways.

The opcode could be used directly to point to a table of first instructions at the base of the microcode store. In such a scheme all microcode routines longer than one word would require the first word of the routine to branch to the remaining part of the routine elsewhere in the microcode store. This would break up many routines into different parts of microcode store. It may also be inefficient, depending on what other functions the branch field of the microcode word could have performed if the first word of the routine did not have to be a branch.

The opcode could be used directly with zeros inserted at the least significant end to form an address that would point to microcode entry points separated by 2, 4, 8, 16, etc. words, depending on the number of zeros appended. This would allow more routines to be located in contiguous words. Only routines longer than the entry point spacing would have to be split by branching to other parts of microcode store. The disadvantage is that where routines are shorter than the entry point spacing, there would be unused holes in the microcode store. When microprograms are expanded and the microcode store gets full (as memories always seem to do), the microprograms will be split more and more times to fit into the unused holes in the microcode store. This will make the micro program more difficult to design and debug as the microcode store fills up.

A PAL may be programmed to decode the opcode into entry point addresses spaced to fit the microprograms. This allows the microcode words of the routines to be

kept together in consecutive locations, making design and debugging of programs easier. But each time routines are moved or expanded in size, a new program for the opcode mapping PAL must be defined.

A RAM or PROM memory may be used as a look-up table for entry points in the microcode store. This allows the greatest flexibility. Microcode routines may be located anywhere in control store, independent of the opcode value. The entry points may be spaced to fit each routine. As routines are changed or moved, it is very easy to reload the look-up table with new entry points.

The opcode mapping method chosen for this system is the RAM approach.

Macro Opcode Map RAM

The map RAM is shown in Figure 5-3. It is formed from three Am9150 1K x 4 bit separate I/O high speed RAMs.

Together, the three RAMs provide a 12-bit output which is used as the macroinstruction decode address. The address is limited to 12 bits since the maximum size of control store provided for in this system is 4K words.

This decode address is connected to the 'A' address input of the Am29331 sequencer. When this address is selected by the sequencer, a branch is made to the first microinstruction of the selected routine.

The address input to all the Am9150s comes from the most significant bits of the Macro Opcode Register (bits 31:22). This address selects the entry point into microcode control store from the map RAM when a macroinstruction is decoded. The macro opcode register is also used during diagnostics and WCS loading to address the map RAM.

The Am9150 RAMs are always selected and output enabled since no other device shares the 'A' input of the sequencer. Also the Am9151 has no power down mode, so there would be no advantage to deselecting the memory. Note: if lower power in the system is required, an alternate memory to use in implementing the map RAM would be the Am2148. That memory does save significant power when deselected and would increase map RAM access time only slightly.

When the Am9150 RAMs are loaded with data, they are written with data as though they were an extension of the microcode control store. The writable control store write enable line is connected to the Am9150's write enable input.

WCS Port

Also shown in Figure 5-3 is the Writable Control Store (WCS) port. This port is formed from two Am29818-1 pipeline diagnostics registers. The port was shown in block form in Figure 4-5. The port is used as part of the system serial diagnostics and writable control store loading scheme.

The bidirectional "inputs" of the Am29818-1 are connected to the macro opcode map RAM data inputs. When placed in a special mode, the port "inputs" are driven as data outputs. This data is then used as input to the map RAM during a WCS write operation. The data comes from the Am29818-1's internal shadow register.

The outputs of the WCS port are connected to the microcode control store address lines. The WCS port may thus be used as an alternate address source for the microcode control store. During a diagnostic read or write of the control store, the WCS port provides the needed address.

Note that the data for the outputs of the WCS port comes from the Am29818-1's internal pipeline register. The pipeline register contents are independent of the shadow

register contents. This allows an address for the microcode control store to be in the pipeline register at the same time data for the map RAM is in the shadow register. These separate registers allow the WCS and map RAM to be written in the same cycle as though they were one writable control store.

Macro Operand Address Counters

These are three identical loadable up/down binary counters made from AmPAL22V10 PALs. They are shown in Figure 5-4. The logic definition file for the PALs is shown in Appendix G.

One counter is used for each operand register address. The counters are loaded from the data outputs of the macro opcode register. The outputs of the counters are tied to the address inputs of the read and write ports of the Am29334 register file.

The counter load, count direction, output enable, and count enable functions are internally decoded from inputs that come from the control pipeline register. These counters are intended for use in array processing algorithms, one example being a digital signal processing algorithm for a filter.

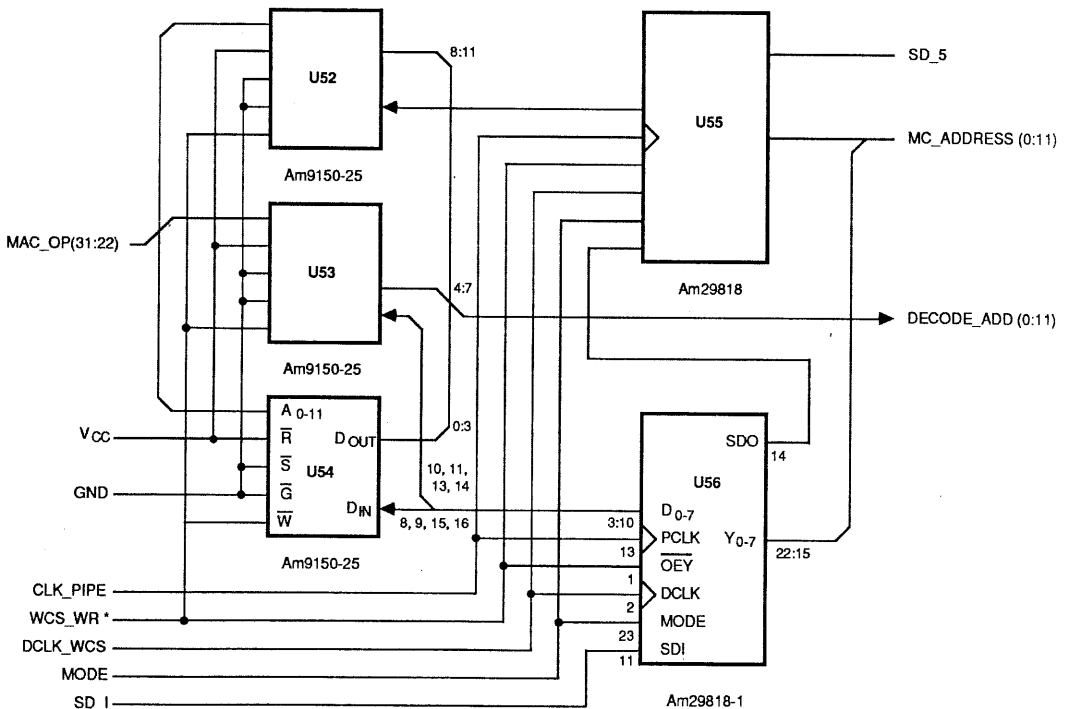


Figure 5-3. Macro Opcode Map RAM

The counters make it simple to perform the same calculation on arrays of data stored in the register file. One microinstruction or a short microinstruction routine can loop on an array calculation and at the end of each calculation cycle simply increment the operand address counters. In that way, new operands are fetched for each calculation on the array without the need for the microcode instructions to directly specify operand addresses.

Control pipeline bits determine whether the microcode operand address or the macro operand counter address is used. The selection is independent for each operand address. Thus, an example would be the operand 'A' address' coming from the microcode while the 'B' operand and 'C' operand addresses come from the counters.

An additional feature is that the 'C' operand counter address may be directed to the Am29334 register file 'B' write port address input. This allows the 'C' operand address to come from microcode while the 'C' operand counter address is used in writing data from system memory into the register file via the second write port. This means that CPU calculations may continue uninterrupted while new data is being loaded into the

register file. Also, as long as data is coming from sequential locations in memory and going to sequential locations in the register file, the memory address counter and 'C' operand counter may be incremented together, thus loading several memory words in sequence. This loading may be accomplished without repeated address calculation by the CPU.

Operand Counter Use Example

To help illustrate the use of the operand address counters a typical Finite Impulse Response (FIR) digital signal processing filter algorithm is described here.

An FIR digital filter takes in a stream of amplitude samples from an analog waveform. Each sample is processed through a series of calculations to produce an output value. The resulting stream of output amplitude values produces a waveform that is the result of a filter operation on the input waveform.

The calculations involved are a series of multiplies between different coefficient values and several past input samples. The result of each multiply is accumulated to produce one output value. The number of coefficients

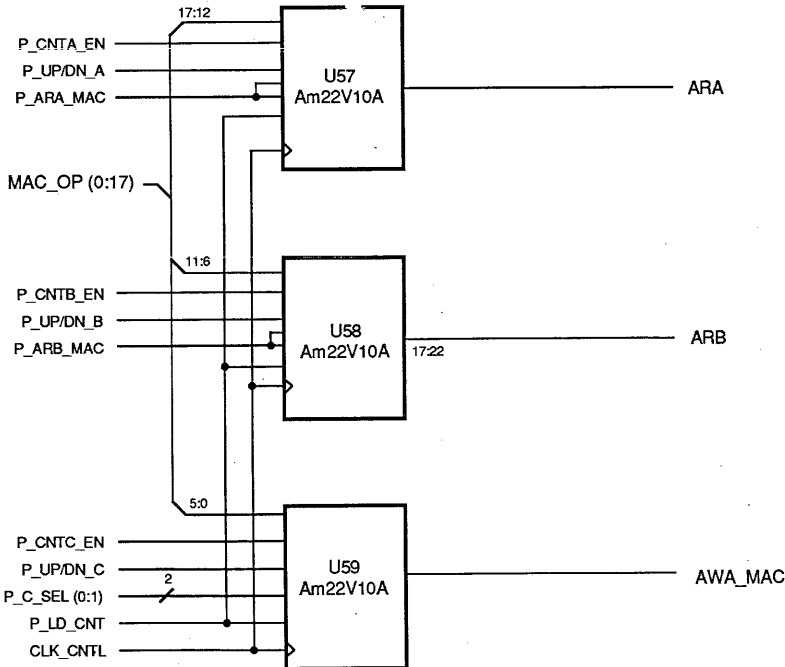


Figure 5-4. Macro Operand Address Counters

09856A 5-4

and retained past samples determines how selective the filter operation is. The values of the coefficients determine the type of filter operation; e.g., bandpass vs. lowpass.

The algorithm for calculating one output value would be the following:

```
Sum := 0;
for n = 0 to number_of_coefficients do
  Sum := Sum + (Sample(x - n) * Coefficient(n));
```

Each time a new input sample is acquired, the new sample becomes Sample(x), and all past samples shift down in the sample array such that Sample(x - 1) := Sample(x) for all x. Note that the number of retained past samples is equal to the number of coefficients.

This algorithm may be implemented with two arrays of data and a temporary register. One array contains coefficients and the other contains past input samples.

The coefficient and sample operands may be multiplied in a single system cycle by either the Parallel Multiplier or the Floating Point Processor. The Parallel Multiplier may also perform an accumulate in the same cycle. The Floating Point Processor requires a second cycle to do the accumulate function. So for each multiply and accumulate operation on a sample-coefficient pair, either one or two cycles are needed.

Obviously the operand counters may be used to address the data arrays. As each coefficient-sample pair is multiply-accumulated, the counters are incremented to point to the next pair of operands. This allows the inner multiply-accumulate loop to be only one or two microinstructions long.

One feature of the operand counters adds to the efficiency of this algorithm. When an operand counter reaches either the maximum or minimum count value, the counter will reload the original count value from the macro opcode register on the next increment. This creates a counter that may treat the register file as a circular buffer. The length of the buffer is determined by the distance from the original count value to either the base or upper limit of the register file address.

Note also that if one counter is always incremented while the other is decremented, two circular buffers may share the register file. One has a lower bound of zero and the other an upper bound of 63. With this scheme two equal size buffers could be up to 32 words each.

The circular buffer approach to the arrays works well with the FIR filter algorithm. At the end of each output value

calculation, the counter addresses will point back to the first coefficient-sample pair, ready for the next input sample iteration.

Note that if on the last multiply-accumulate cycle of an iteration the sample operand counter is not incremented, and the 'C' operand counter is used to load a new sample from memory into the oldest sample array location, the effect will be to shift all the samples down by one in the array while overlapping the new sample load with the last cycle of a sample iteration.

One additional cycle at the end of each iteration may move the output value from the register file to the memory. No memory address calculation cycle is needed since the memory address counter may be used to address the memory.

With this scheme only one cycle of overhead between iterations is needed. Therefore, assuming clocked multiply operation of the PM to achieve single cycle multiply-accumulate execution, a 31 coefficient FIR could complete one output value iteration in 32 cycles. Assuming a 100 ns cycle time (100 ns clocked multiply in the PM), that would allow over 312,000 samples per second or an input bandwidth of over 156 kHz. A 9 coefficient filter would have a 500 kHz bandwidth.

This is an example of how a microprogrammed system may have its architecture tuned to a particular application for the best possible performance. Much of the performance comes from the microprogrammed system's ability to control and perform several parallel functions at one time.

REGISTER FILE ADDRESS MULTIPLEXER

The Register File Address Multiplexer, shown in the block diagram of Figure 1-2, is made up of four separate multiplexers. One multiplexer is used for each register file address port; two read ports and two write ports.

Read Ports A and B

These multiplexers are shown in Figures 5-4 and 5-5. Each multiplexer is really a three-state bus that may be driven either from the control pipeline register via an Am29827 three-state buffer or from an operand counter output. A bit for each address from the control pipeline selects which source may drive each address bus.

The Am29827 three-state buffers are needed in addition to the three-state outputs of the control pipeline because each operand address is 6 bits. This number does not fit

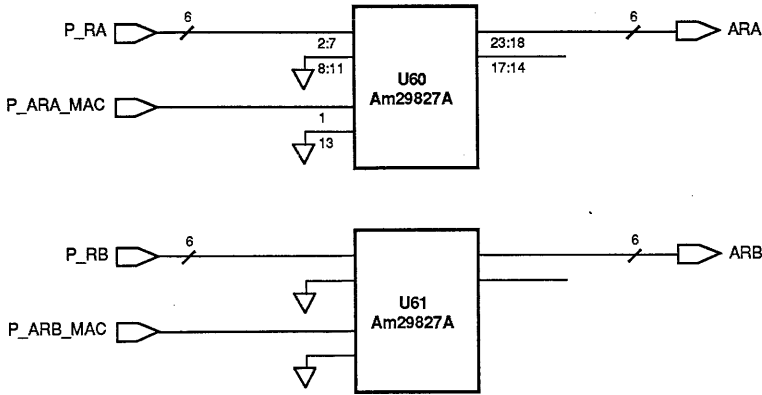


Figure 5-5. Register File Address MUX, Read Ports

09856A 5-5

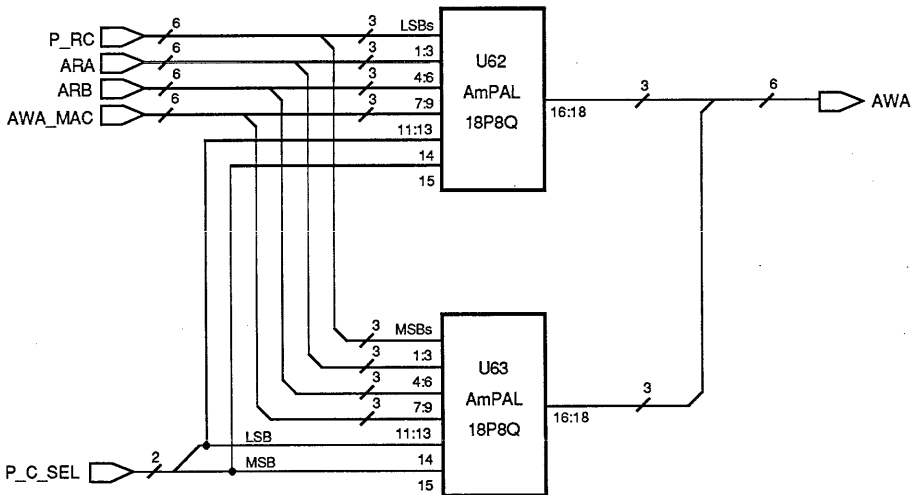


Figure 5-6. Register File Address MUX, Write Port A

09856A 5-6

well into the 4-bit boundaries of each slice of the microcode control store. So to avoid wasting control store bits, the external three-state buffer is used to gate the control pipeline address onto the register file address bus rather than trying to use the control store's own three-state outputs.

Write Port A

This multiplexer is implemented by a pair of AmPAL 18P8 PALs. It is shown in Figure 5-6. The logic definition file for the PAL is contained in Appendix H.

It is this four input hex multiplexer that allows the 'C' register file operand (i.e., register file 'A' write port) address to come from four possible sources. The address may be provided from the 'C' operand in the control store, 'C' operand counter, 'A' operand final address, or 'B' operand final address. The 'A' and 'B' operand addresses are referred to as final because the multiplexer input is taken from the register address buses after the choice between control pipeline or operand counter has been made for the 'A' and 'B' operand addresses. The select bits for the multiplexer come from the control pipeline.

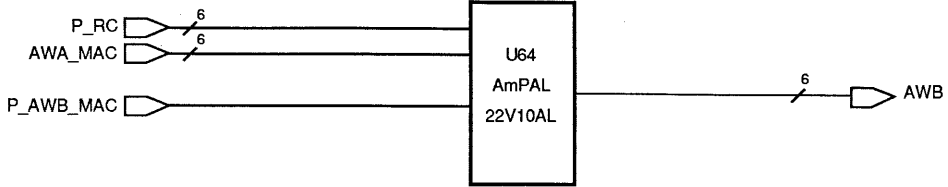


Figure 5-7. Register File Address MUX, Write Port B

09856A 5-7

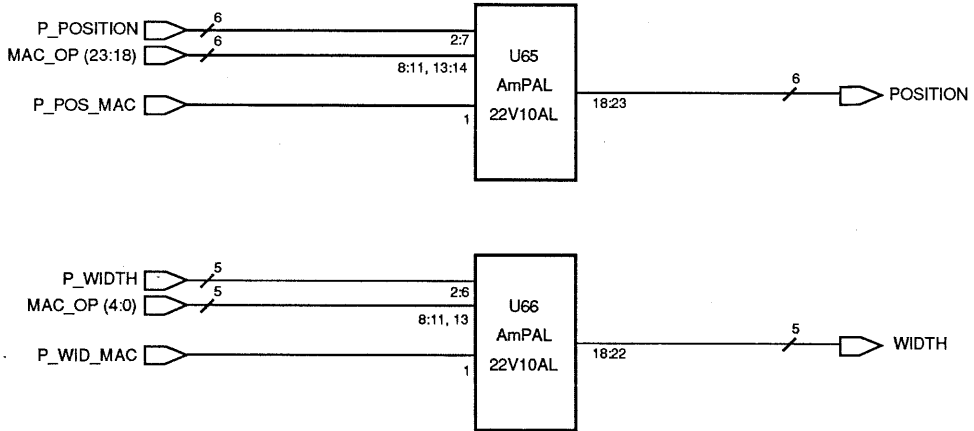


Figure 5-8. Position and Width MUX

09856A 5-8

Write Port B

This multiplexer is made from an AmPAL22V10. It operates as a two input hex multiplexer. It is shown in Figure 5-7. The logic definition file for the PAL is given in Appendix I.

It selects either the control pipeline 'C' operand address or the 'C' operand counter address as the source for the register file 'B' write port address. The select bit comes from the control pipeline register.

POSITION AND WIDTH MULTIPLEXERS

The position and width multiplexers are implemented with AmPAL22V10A PALs. They are shown in Figure 5-8. The logic definition file for the PALs is given in Appendix I.

Each is a two input hex multiplexer, identical to the multiplexer used for the B Write Port Mux. They select

from the Position and Width values that may be provided either from the control pipeline or the Macro Opcode Register. The select control comes from the control pipeline.

'A' speed PALs are used here since these multiplexers are in the critical path to the ALU. They must use 7 ns less delay than the combined delay of the 'A' Read Port Mux and Register File access time. The required 7 ns advantage is consumed by the ALU's longer propagation delay from Position input to Y output vs. Data input to Y output.

SEQUENCER

The sequencer is a 16-bit-wide address generator that controls the execution sequence of microinstructions stored in the microcode control store. It may handle interrupts or traps at any microinstruction boundary. An interrupt or trap is treated like an unexpected procedure call.

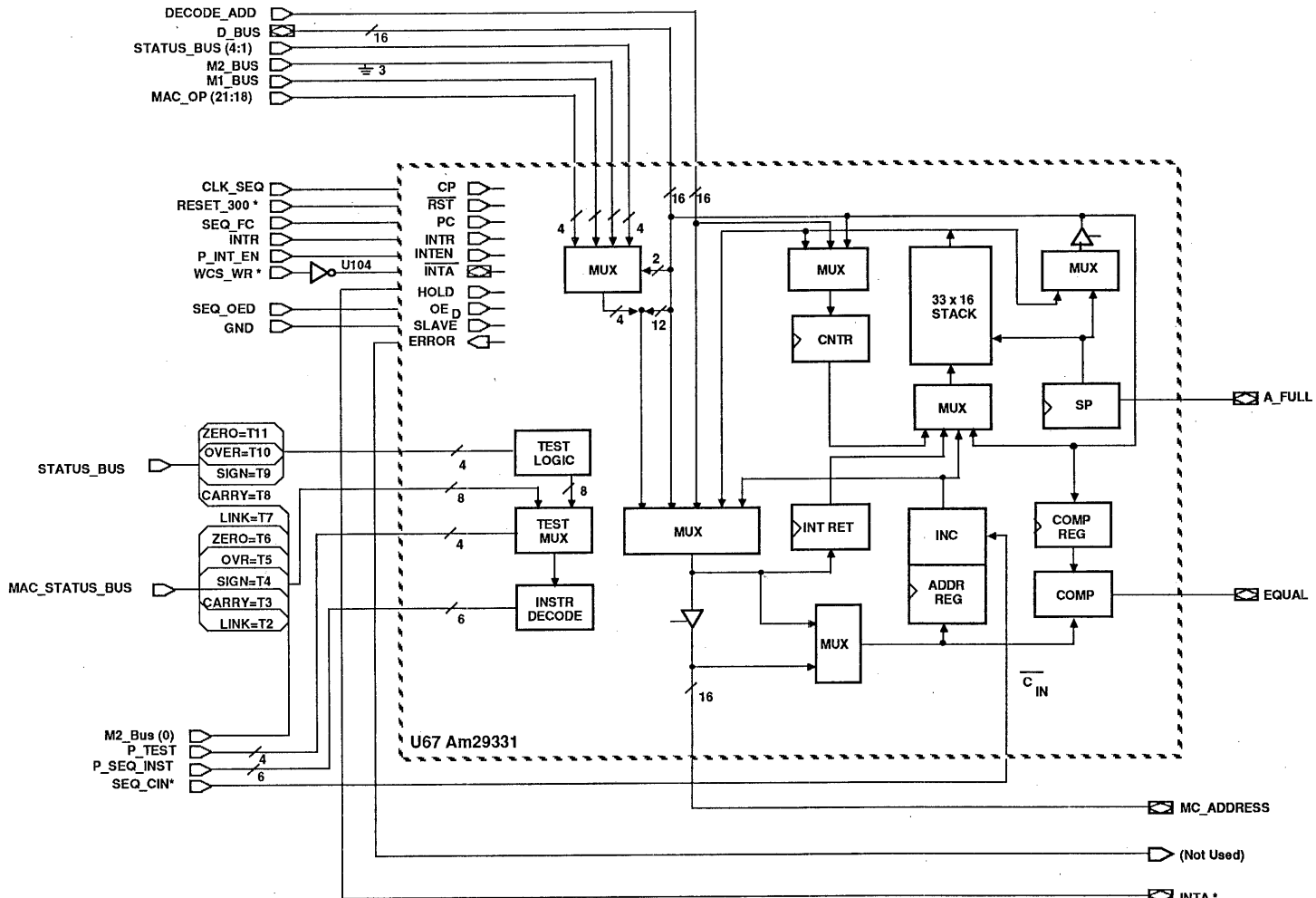


Figure 5-9. Sequencer Block

06656A 5-9

Two independent branch inputs as well as four multi-way branch address sources are provided. One of the branch address inputs is bidirectional and may be used to read or write information in the sequencer's internal 33-level deep stack.

A 16-bit counter, test condition multiplexer, and break-point address comparator are also provided. The break-point comparator is used as a hardware aid to microcode debugging. The connections to the sequencer are shown in Figure 5-9.

The sequencer's 'A' branch address input is connected to the Macro Opcode map RAM output and is the path through which the macroinstruction specifies its entry point into microcode.

The 'D' branch address input is tied to the D_BUS. Through this path, branch addresses or constants come from the control pipeline register and data may be exchanged with the data section of the CPU.

The 'M0' multi-way branch address input is connected to the macro opcode register bits 21:18. These bits may be used as a modifier to the macro opcode via a multi-way branch based on these bits.

The 'M1' multi-way branch address inputs come from the Floating Point Processor (FPP) external status register. These bits are the overflow, underflow, invalid, and 'extra' status flags from the FPP. The 'extra' status flag is the OR of the zero, NAN, and inexact status flags from the FPP. A single multi-way branch on these inputs may be used to detect and handle quickly any of the catastrophic status conditions from the FPP. If the 'extra' flag is active, it indicates that a second multi-way branch may be used to determine which of the 'extra' status flags is active.

The FPP zero, NAN, and inexact status flags are connected to the 'M2' multi-way branch input of the sequencer.

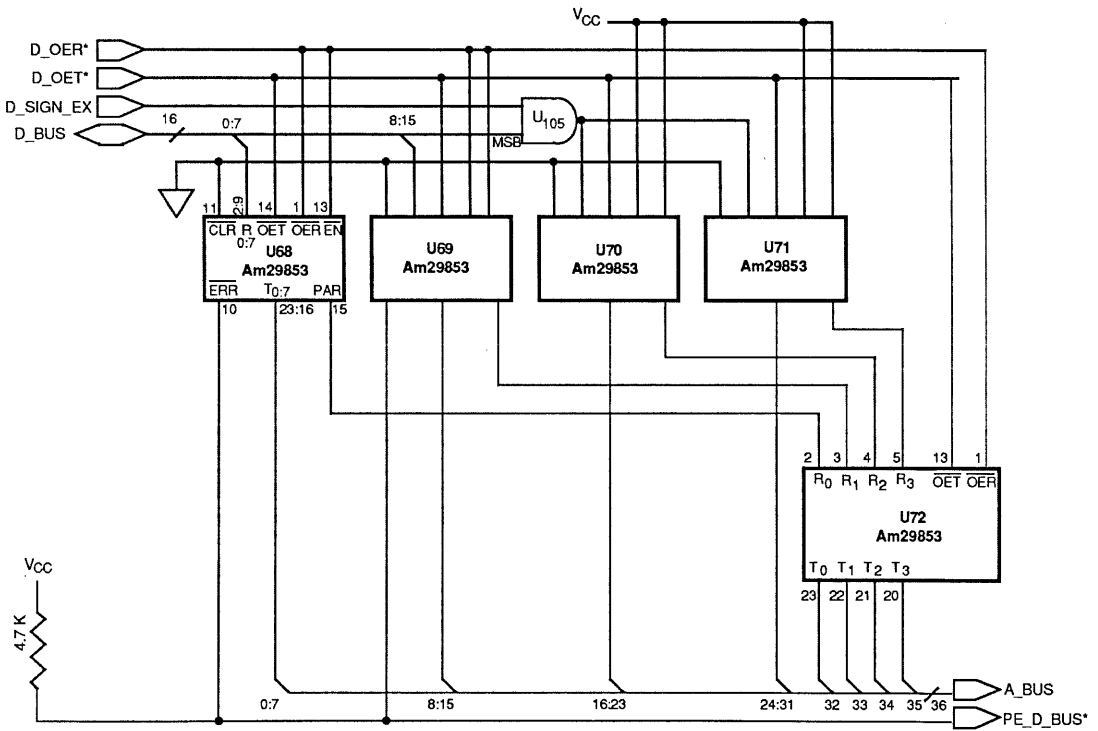


Figure 5-10. D Bus Transceiver

09856A 5-10

The 'M3' multi-way branch input is tied to the ALU microprogram status outputs so that an alternate means of checking ALU status is available. A multi-way branch based on these bits is able to check multiple condition flags in a single cycle.

The Force Continue and Carry-In inputs of the sequencer are active in a trap operation to prevent state change in the sequencer and capture the address of the trapped instruction in the interrupt return address register. Carry-in (CIN*) is driven high by a trap event signal from the trap logic in Figure 5-11. The trap event signal is also ORed with a signal from the control pipeline (P_FC) so that either signal will cause Force Continue to go high. The interrupt request input comes from the Trap circuit shown in Figure 5-11.

The sequencer's HOLD input is driven by the inverted value of the WCS_WR* signal from the host interface controller shown in Figure 4-3. When this signal is

active, the sequencer's output will be three-stated so the WCS Port may drive the microcode control store address lines without contending with the sequencer's output drivers.

The Slave input is grounded since no use of the mode is made in this demonstration system.

The test condition inputs of the sequencer come from three sources. Conditions 11 through 7 are the ALU status bits for zero, overflow, sign, carry, and link. Conditions 6 through 2 come from the Macro Status Register; these bits are the macro version of the same ALU status bits. Condition 1 comes from the FPP external status register bit for zero. Condition 0 is unused.

Control for the sequencer's interrupt enable, test condition select, and instruction input comes from the control pipeline register.

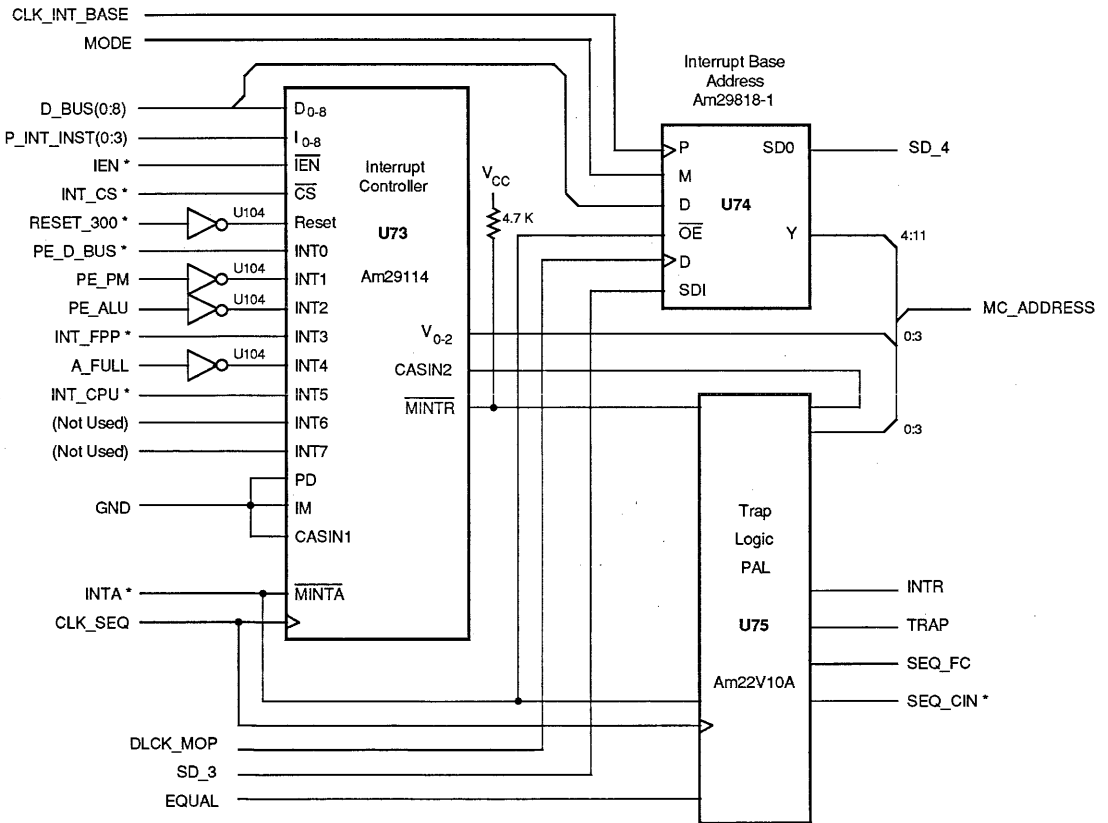


Figure 5-11. Interrupt and Trap Logic

09856A 5-11

The sequencer's D_BUS output enable comes from the control decode logic.

The sequencer A_FULL signal is used as an interrupt signal to the system interrupt controller.

The Equal (breakpoint) signal is used as a trap event signal to the Trap Logic.

Interrupt acknowledge goes to the interrupt controller and trap logic to enable the interrupt and trap vectors onto the microcode control store address bus when an interrupt is executed.

The 'Y' outputs of the sequencer drive the microcode control store address lines to select each microinstruction.

D BUS TRANSCEIVER

The transceiver between the A_BUS and the D_BUS is shown in Figure 5-10.

The D_BUS has no parity bits included where as the A_BUS does contain parity. It is therefore necessary to provide parity generation for the data moved from the D_BUS to the A_BUS.

The D_BUS is only 16 bits wide vs. the 32-bit-wide A_BUS. Thus it is also necessary to provide bus drivers and parity generators for the upper two bytes of the A_BUS, even though no variable data is passed to the A_BUS from the D_BUS through those bits.

The transceiver and parity generator/checker function are combined in a single device type: the Am29853. Four of these are used in addition to an Am29862 inverting transceiver. The inverting transceiver is used on the parity bits because the Am29853 uses odd parity while the Am29300 system uses even parity.

As an added convenience for when numeric constants are passed from the D_BUS to the A_BUS, an AND gate is provided to drive the inputs of the upper two bytes of transceiver. If the AND gate is enabled by the control pipeline, the most significant bit of the D_BUS will be copied to all the upper bits on the A_BUS, thus performing a sign extend for two's complement numbers. If the AND gate is disabled, the upper bits of the A_BUS are forced to zero.

INTERRUPT CONTROL

Interrupt and Trap Philosophy

What is a Trap?

Traps are events that require the immediate attention of the CPU. The urgency of the event is so great that the CPU must not even complete the execution of the instruction in progress in the cycle that the trap request happens. The CPU must not change any machine state in that cycle; it must store the address of the instruction that was to have been executed and must branch to a routine that services the trap event.

The implication here is that the trap will prevent some disastrous change in machine state from which no recovery would be possible. Also implied is that the trap servicing routine may repair what ever the problem is and then return to complete the execution of the instruction where the trap occurred.

One additional implication is that the trap event may be signaled early enough in the instruction cycle to prevent the clocking (change of machine state) that normally occurs at the end of each instruction.

An example of a trap event could be a miss on cache memory access. To complete an instruction when the data being accessed from a cache is invalid would be a disaster with little chance for recovery. If a trap routine to update the cache may be executed instead of completing the instruction, the program may be saved. After the cache has the correct data, the trap routine may return to the aborted instruction to continue execution of the program as if no problem had existed.

Another example of a trap would be a program breakpoint. When debugging a program it is very useful to be able to stop execution of a program just before executing a particular instruction. If this is done, the state of the machine before executing the breakpoint instruction may be examined. To do this the address of the breakpoint instruction is recognized as the instruction is fetched from microcode control store. In the next cycle before the instruction may complete, a trap occurs which branches to a debugging routine. When the programmer is ready to continue the program, a return from trap completes the execution of the breakpoint instruction. The breakpoint trap operation is easy to do, and hardware to implement

it is already provided in the Am29331 sequencer. The breakpoint trap operation will be shown in the Trap Logic described later.

What is an Interrupt?

Interrupts are events that require the attention of the CPU soon.

“Soon” is defined as faster than might happen if the event were polled by a CPU program but later than a few microinstruction execution cycles.

Interrupt events and the resolution of an interrupt are not directly tied to the CPU state. No disasters occur if a few cycles pass by before the interrupt may be handled.

Examples of events handled via interrupt could be: external mechanical events such as switches being opened or closed, an impending stack-full situation, a message signal from another processor, or a peripheral delay timer indicating time-out.

In this demonstration system one other class of interrupt source is included. It is the parity error. A parity error implies corrupted data in a program that cannot be corrected. Since the influence of corrupted data on the program is difficult to determine or correct for, the affected program should be aborted. A parity error is, therefore, important to detect so that the program in which it occurs may be terminated and perhaps rerun with corrected data.

Parity errors are treated as interrupts rather than traps for two reasons. The indication that an error has occurred comes fairly late in an instruction cycle and is therefore difficult to use as a trigger for a trap. When a parity error occurs, the program is generally corrupted and will be terminated; whether the termination happens in the cycle following the error as would be the case with a trap, or within a few cycles, as with an interrupt, is unimportant.

Interrupt Operations

There is no need to design an interrupt circuit from scratch when one already exists. The Am29114 interrupt controller is used in this system. It provides interrupt latching, priority, masking, and vector generation for eight interrupt inputs.

Interrupt Controller

Six interrupt sources are used in this Am29300 system; the two remaining interrupt source inputs are available for software generated interrupts.

The interrupt and trap circuit block diagram is shown in Figure 5-11.

The three highest priority interrupts are parity error signals from the D_BUS, the Am29C323 Parallel Multiplier, and the Am29332 ALU.

The next priority interrupt is a signal from the FPP external status PAL, which indicates that one of the following status flags is active: Overflow, Underflow, or Invalid.

The next priority interrupt is the A_FULL signal from the Am29331 sequencer. This interrupt indicates that the sequencer stack will be full if three additional stack pushes occur.

The next interrupt is the external bus interrupt signal from the host interface controller. This is a “tap on the shoulder” from the host that requests the Am29300 CPU take some previously agreed on action, such as reading a message from the host out of memory.

The two least significant interrupts are unused by hardware and are available for use as software interrupts. These interrupts would be set by the CPU writing into the Am29114 interrupt register.

The interrupt mode is set for capturing asynchronous low going pulses as interrupt signals. This is done because most of the interrupt signals are only guaranteed to be active for a single clock cycle. Therefore, the interrupts must be latched and held by the interrupt controller until acknowledged by the CPU.

The D_BUS is connected to the interrupt controller data pins so that the internal interrupt, mask, and in-service registers may be read and written.

The interrupt controller is selected and given instructions via outputs of the control pipeline register.

Interrupt Sequence

During a given clock, one of the interrupt inputs goes active. At the end of that cycle (active edge of clock), the interrupt signal is clocked into the interrupt register of the Am29114.

During the second clock cycle, the interrupt is ANDed with the interrupt mask register and, if the interrupt is allowed, its priority is compared to any currently in-service interrupt. If the new interrupt is of higher priority than any in-service interrupt, the MINTR* (interrupt request) will go active at the next active clock edge.

During the third clock cycle, the Am29114 interrupt request is externally ORed with the interrupt request from the trap logic. The combined interrupt request is then loaded into a delay flip flop. The delay flip flop is needed to synchronize the final interrupt request with the system clock. The reason for this is that the interrupt request from the Am29114 is stable too late (41 ns) in the third cycle to be useful in selecting an interrupt address. The set-up time for the microcode control store address could not be met if the Am29114 interrupt request were used directly with the Am29331 sequencer.

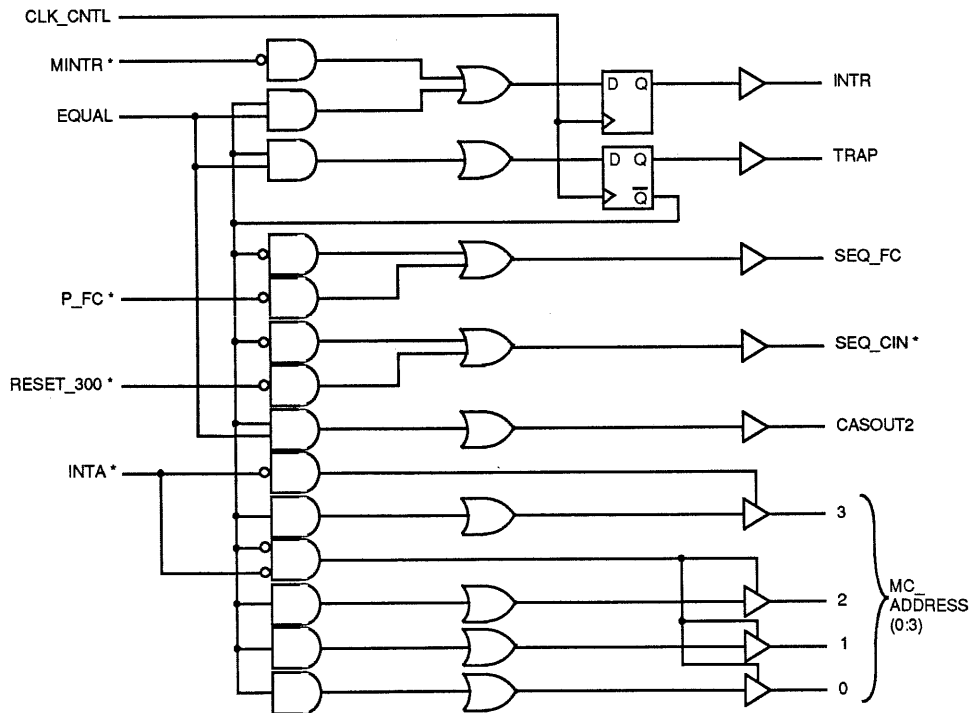
The external OR and delay functions are implemented in an AmPAL22V10A, whose logic is shown in Figure 5-12.

During the fourth clock cycle, the INTR* (interrupt request) input of the sequencer is driven by the delay flip flop. The sequencer then returns INTA* (interrupt acknowledge) if micro-interrupts are allowed. The INTA* signal enables the interrupt vector onto the microcode control store address lines.

The LSB three bits of the interrupt vector are provided by the Am29114 interrupt priority encoder. Bit 3 of the interrupt vector is provided by the trap logic. The bit is low for an interrupt and high for a trap vector. The upper bits (4:11) of the vector are provided by an external Am29818-1 register. This register provides a variable base address for a nine entry point table look-up (multi-way branch), which is based on the four bits of interrupt vector from the Am29114. The Am29818-1 register is loaded via the D_BUS or through the diagnostics SSR chain. The need for a nine entry point table is explained in the section on trap operation.

During the fifth clock cycle of the interrupt sequence, the first instruction of the interrupt routine will execute. During this cycle the interrupt return address will be pushed onto the sequencer stack.

In summary, from the time an interrupt signal becomes active until the interrupt service routine begins execution, four instructions in the main program will complete execution.



09856A 5-12

Figure 5-12. U75 AmPAL 22V10A Trap Logic PAL

Trap Operation

Trap Issues

A trap requires extremely fast response to the trap event signal.

The ideal situation is for the trap event signal to cause the abortion of the instruction in execution at the time the event signal appears.

This is extremely difficult in a high clock frequency system. To succeed, the trap event signal must be stable at least in time to prevent clocking of the data section of the CPU, which would otherwise change the system state (i.e., complete execution of the instruction). This implies that the trap event signal is stable one clock control circuit set-up time before the high to low edge of the system clock. The high-to-low edge of clock is significant, because once the clock signal falls, the writing of any write enabled port on the Am29334 register file will begin. In addition, the trap event signal must be stable in time to cause the Am29331 sequencer force continue (FC), interrupt request (INTR), and carry in (CIN*) signals to go high soon enough to disable the sequencer microprogram address in time to meet the set-up time requirements of the microcode control store.

In a 100 ns cycle time system, such as the one being discussed here, the trap event signal must be valid no later than 25 ns into the cycle. For a trap event signal that is to be derived from the effects of the instruction in execution in that cycle, this requirement is very difficult to meet.

Fortunately there are trap events that may be signalled on the one or two cycles previous to the cycle in which the trap must occur. Some examples would be: a cache miss that may be detected from the cache address created in a cycle prior to that in which the cache data is used in a calculation; or a breakpoint in which the breakpoint target instruction address is detected by the sequencer in the cycle prior to the instruction being loaded into the control pipeline for execution.

If an instruction is a known potential trap, it is possible to execute the instruction so that no critical information is destroyed by completing its execution. This may be done by writing results back to a temporary register while allowing no other significant system state changes, such as updating the ALU Q register, or doing a return from procedure call. The instruction may then be allowed to execute and generate any trap event signals that might result from the execution, without concern for irrevocably destroying data because of some error condition.

In the above examples, the trap event signal may be loaded into a delay flip flop to synchronize the trap request with the beginning of the following cycle. This causes the trap operation to occur early in the cycle following the event and to complete successfully.

The only trap condition implemented in this design is the breakpoint.

Trap Logic

By definition, the response time between trap event signal and trap operation must be much faster than the four or more cycles that an interrupt takes to begin execution. This requires that the trap logic be different from the Am29114 interrupt controller. The trap logic design is implemented in an AmPAL22V10A. The logic is shown in Figure 5-12. The definition file for the PAL is shown in Appendix J.

The trap logic is in effect a simpler and faster interrupt controller. This "trap controller" is cascaded with the Am29114 interrupt controller so that the same address vector approach used with the interrupt controller may be extended to trap operations.

A trap is treated as a special form of interrupt with a higher priority. When a trap occurs, the trap logic generates a cascade out (CASOUT2) signal to the Am29114 to prevent any interrupt operation from beginning in the same cycle.

The trap logic also generates an INTR signal to the Am29331 sequencer. The INTR signal in turn causes the sequencer to three-state its microcode address outputs and return an INTA signal to the trap logic. The INTA signal enables a four bit vector from the trap logic and the interrupt base address from the Am29818-1 registers as shown in Figure 5-11.

The above steps essentially generate an interrupt and provide the interrupt vector. What makes a trap different is that the Trap Logic is also used to drive the Am29331 sequencer Force Continue and Carry-In inputs. This causes the sequencer to ignore the instruction being trapped and to perform a continue instruction instead, which changes no state in the sequencer. The CIN* signal's being high causes the trapped instruction address to not be incremented. Therefore, the trapped instruction's address will be loaded into the sequencer interrupt return address register. In addition, the TRAP signal is used to prevent any state change in the system other than in the sequencer, effectively aborting the trapped instruction.

Following are some other features to note in the trap logic.

Am29300 system RESET is used to generate the sequencer Carry-In signal (SEQ_CIN*). This is done to force SEQ_CIN* high during reset so that the first microcode instruction executed after reset will be at address zero rather than one.

In order for a trap operation to take effect, the instruction that is to be trapped must have its microcode interrupt enable bit active. This bit is used as the interrupt enable to the sequencer. If it is not active, then the microcode control store address from the sequencer will not be three-stated, and the interrupt vector will not be substituted. In addition, the TRAP signal will still occur, causing the trap target instruction not to execute correctly. Note that the interrupt enable bit could be externally forced active by the trap operation via an OR gate. But the added delay could cause the interrupt acknowledge to be too late to allow the interrupt vector address to meet required set-up times. (Of course, it is possible to design the system so that every trap causes all the system clocks to be stopped for one cycle. That would allow enough time for all kinds of tricks to be played. This design, however, will not explore that approach.)

MICROCODE CONTROL STORE AND CONTROL PIPELINE REGISTER

Control Store Function

The microcode control store is the high speed memory that contains the control bits comprising the instructions that the system may execute.

This system uses what is called "horizontal" microcode. Each microinstruction contains many control bits that manage a variety of different functions in parallel. "In parallel" is the key phrase. All the control information needed to manage the entire Am29300 system during the execution of one microinstruction is contained in one word of microcode control store.

The memory must be fast because its access time must be significantly shorter than the cycle time of the system. In general the access time must be less than half the cycle length. This is because of the time required by the sequencer to generate each new address to the control store, which takes up the remaining time in the cycle.

Pipeline Register Function

At the output of the microcode control store there is a register to hold the control information stable during the

execution of an instruction. With the control information held in the pipeline register, the control section of the CPU is free to begin reading the next microinstruction from the control store. In this way, the control section is operating in parallel with the data section. The control section fetches the next instruction while the data section executes the current instruction. This parallel operation, where one section of the system works on one step of a problem while another section works on the next step, is called pipelining, hence the name for the pipeline register.

Through parallel operation, pipelining nearly doubles the speed of the system over what might be the case if the control section and data section were directly tied together in a serial fashion.

Control Store Implementation

Because this method of pipelining the output of a microcode store is so popular, there are special memories available that combine a high speed memory with a pipeline register at its output. These combined memory and pipeline devices may significantly reduce the system parts count.

These memories are available as either RAM or PROM devices. RAM versions are used to make writable control stores.

These memories also include Serial Shadow Registers (SSR) along with the pipeline register. This allows diagnostic routines to read and control the pipeline register outputs. Where RAM versions are used, the SSR is used as a built in means to load the writable control store.

This system is designed to use one of the following for control store: Am9151-50, 1K x 4 RAM; Am27S65, 1K x 4 PROM; Am27S75, 2K x 4 PROM; or Am27S85, 4K x 4 PROM. These devices all share a similar pinout so that simple jumper connections allow any of them to be placed in the same sockets.

The connections to the control store are shown in Figures 5-13 and 5-14.

A total of 23 memories are used to form the needed 92-bit-wide microcode words.

Because this system is designed to use no more than a 4K word deep control store, only the lower 12 bits of microcode address from the sequencer are connected.

The memories in the control store which provide the microcode branch field are connected differently from the remaining memories. This is because the branch field

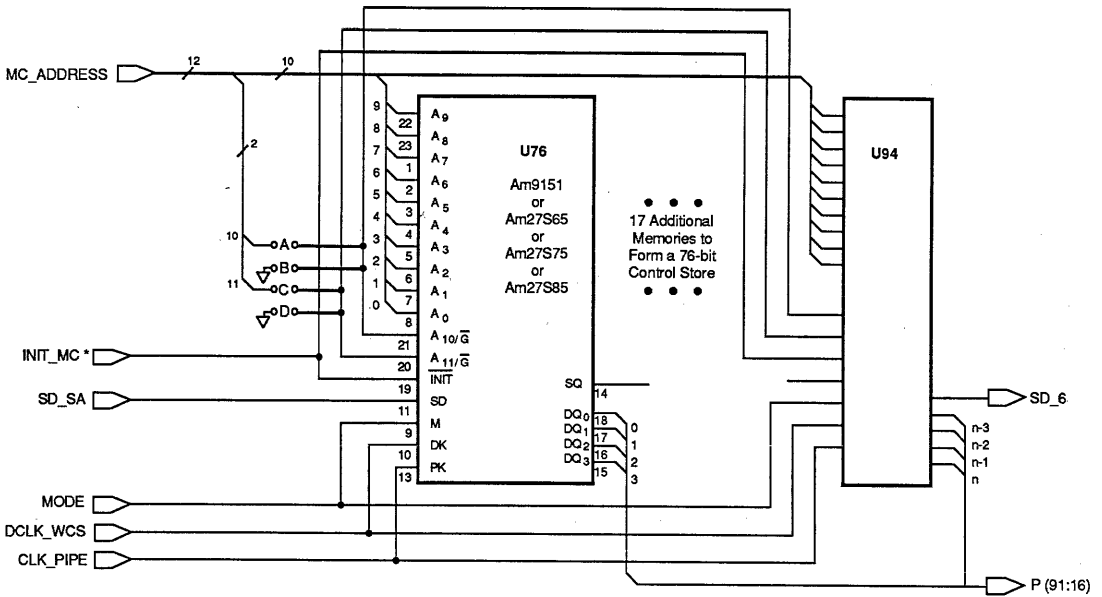


Figure 5-13. Microcode Control Store

09856A 5-13

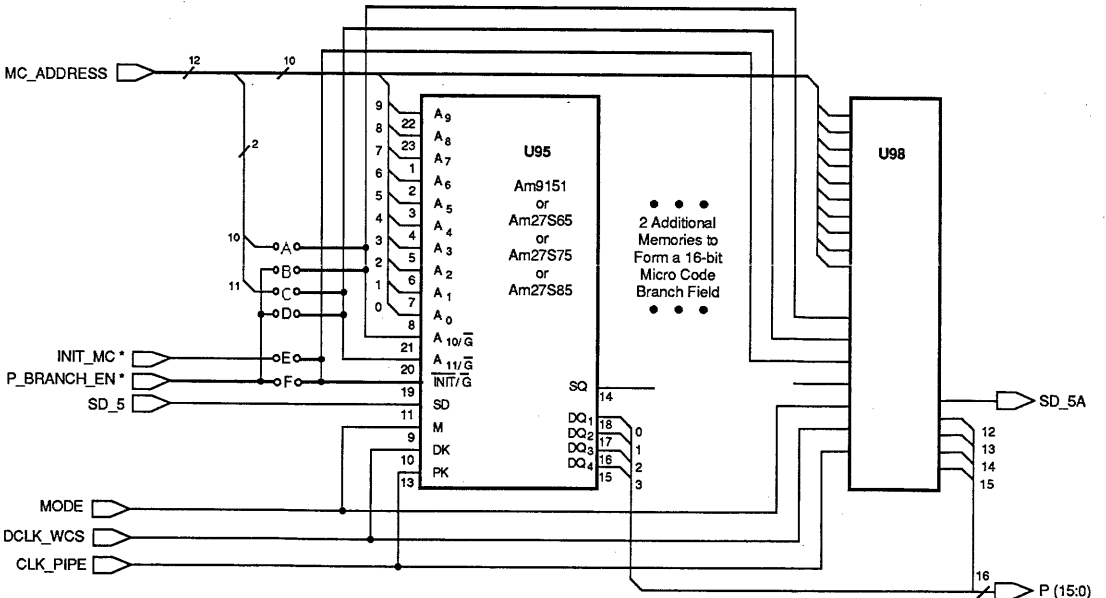


Figure 5-14. Microcode Control Store

09856A 5-14

outputs are connected to the D_BUS and must be three-stated when other devices drive the D_BUS. All the other outputs of the control store are always output enabled.

Figure 5-13 shows how the bulk of the control store is connected.

When the Am9151-50 or the Am27S65 is used, the jumper at location "B" is connected. This continuously enables the memory.

When the Am27S75 is used, the jumpers at locations A and D are connected. Also, the Am27S75 G/Gs* (pin 20) is internally programmed as an asynchronous enable. Those jumper connections will always enable the memory and connect address bit 10 to it.

When the Am27S85 is used, the jumpers at locations A and C are connected. The Am27S85 G/Gs//Is* (pin 19) is programmed as a synchronous initialize function. Those connections will always enable the memory and provide address bits 10 and 11 to it.

Figure 5-14 shows the connection for the memories that support the branch field.

When the Am9151-50 or the Am27S65 is used, the jumpers at location B and E are connected. This enables the memory when the control pipeline selects the control store to drive the D_BUS.

When the Am27S75 is used, the jumpers at locations A, D and E are connected. Also, the Am27S75 G/Gs* (pin 20) is internally programmed as an asynchronous enable. Those jumper connections will enable the memory when the control pipeline selects the control store to drive the D_BUS.

When the Am27S85 is used, the jumpers at locations A, C, and F are connected. The Am27S85 G/Gs//Is* (pin 19) is programmed as an asynchronous enable function. Those connections will enable the memory when the control pipeline selects the control store to drive the D_BUS. Also, these connections imply that when the Am27S85 is used, the branch field of the initialize word will not be valid.

CLOCK CONTROL

In almost every complex digital system there is a need to control and qualify selectively the system clock.

A register often needs a qualified clock that will clock (i.e., load) the register only when specified by some control signal. Sometimes a register will internally qualify its own

clock by providing a load enable input. But most often, registers have only data input and outputs, an output enable, and an unqualified clock input. It is up to the system designer to provide a means to restrict the clock to the register so that it receives clock only on those cycles when its load enable control signal is active.

Restricting a clock in this fashion is referred to as qualifying a clock. The controlling signal that enables the qualified clock is called the qualifier.

Most synchronous digital systems have a system clock with a single active edge. This means that the system state will only change on either the low-to-high or high-to-low edge of the clock. The opposite transition of the clock will have no state changing effect in the system. The opposite transition of the clock is referred to as the inactive edge of the clock. It should be noted, however, that, even though there is a single active edge for the clocking of registered states in the system, the level of the clock may have an effect on some multiplexers or latches in the system. The level of the clock may control the path selected by a multiplexer, whether a latch is flow-through or held, or the write enable of a memory.

To qualify a clock, there must be a way to prevent the active edge from occurring. This implies that the clock is held either high or low when it is prevented from cycling. The choice of whether the clock will be stopped (held) at its high level or low level may depend on what, if any, effect the level of the clock has on system multiplexers, latches, or memories. For example, if the low level of the clock enables a memory write line, it may be preferred to stop the clock at the high level rather than the low level to prevent any change in state of the memory.

Clock Qualification Circuit

In the Am29300 system described here, the system clock will be stopped at the high level. This is because the low level of the clock may start the writing of data into the Am29334 register file. The active edge of the clock will be the low-to-high transition.

This method of qualifying clocks is referred to as 'OR' qualification. Usually with this method the free-running (unqualified) version of the system clock is 'ORed' with a low active enable signal. Thus, if the enable is active (low) the resulting qualified clock is allowed to track the free running clock. If the enable is inactive (high) the qualified clock will be forced high, stopping the clock, until the enable again goes active. Because the free running clock is always high during the first portion of each clock cycle, the clock enable signal need not be stable until just before the inactive edge of the free running clock.

In this Am29300 demonstration system the following are the desired controls over the system clocks:

1. The ability to stop all clocks to the Am29300 CPU, both control and data sections. This will suspend operation of (halt) the system.
2. The ability further to qualify register loading (register clocks) with control pipeline signals. The controlled registers would be the Macro Status, Macro Opcode, and Interrupt Base Address register.
3. The ability to single step all the system clocks when the system clocks are in the halt mode. Note this implies only conditional single stepping on those register clocks that are further qualified by load enable controls.
4. The ability to single step the data section or the control section independently.
5. The ability to force the control pipeline or the Macro Status, Macro Opcode, and Interrupt Base Address registers to load. This capability is used to implement diagnostic control over these registers.

To implement this kind of control over the system clocks, a separately qualified version of the system free running clock must be created for each differently handled register. The general clock for the control section is different from that for the data section. Also, each qualified register clock is different.

The block diagram for the clock qualification circuit is shown in Figure 5-15. The logic equation definition file for the PAL in this circuit is shown in Appendix K.

The qualifiers for the system clocks come from either the control pipeline, trap logic or the host interface controller. The AmPAL22V10A Programmable Array Logic (PAL) device is used to combine the various qualifiers into the appropriate clock enables for each differently handled set of registers. The output of the PAL is then logically ORed with the system free running clock to form the various qualified clocks in the system.

In this system, the free running clock generator produces an active low clock with the enables active high. By using negative logic OR gates (NAND gates) the clock and enable signals are logically ORed together to produce active high qualified clocks. The negative logic OR gates are external to the clock qualifier PALs.

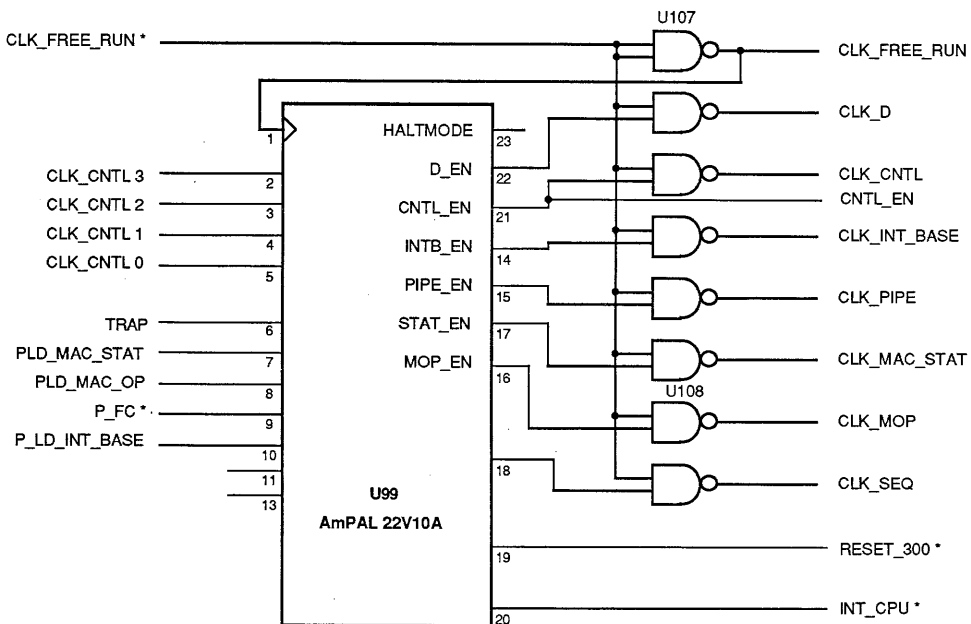


Figure 5-15. Clock Qualification Block Diagram

09856A 5-15

The NAND gates also serve as high output current buffers that allow the qualified clocks to drive many registers in the system. These NAND buffers also cause the clocks to have very high speed edges. This requires that clock lines be handled more carefully than other signal lines to help prevent noise, reflections, and ringing on the clock lines. Preventing these problems helps to ensure clean clock signals free from the glitches that may cause missed clocking or double clocking of registers. It is suggested that clock lines be routed serially, kept less than 12 inches in length, and terminated to the printed circuit board's characteristic impedance at the last point of use on each clock line.

Note that all the system clock lines, even the free-running clock line, pass through a NAND gate. This is done to equalize the delay of all clocks so that clock skew in the system is minimized.

Clock Generator

The unqualified (free running) source for all the clocks in the system comes from a clock generator implemented in an AmPAL16R6B. A diagram of the logic implemented in this PAL is shown in Figure 5-16. The logic equation definition file for this PAL is shown in Appendix L.

The only reason that a clock generator PAL is used in addition to a simple clock oscillator module is to provide the ability to vary dynamically the length of each system clock cycle. This ability allows the system to run at the maximum clock rate most of the time when the fastest data paths are in use and to run at a slower rate only when slower system data paths are in use. By slowing the system cycle time dynamically only when a slow data path is used, the average system speed is much higher than would be the case if the system clock rate were fixed at the rate required by the slowest data path.

A simple way to do this would be to divide the normal system clock by two and on each cycle select whether the normal length or the double length clock cycle would be used.

In this system, finer control over the length of each cycle is desired. Where the cycle need only be a little longer than usual, only a slightly longer cycle is used rather than doubling the cycle length.

This is done by dividing down a high speed clock, which runs three times faster than the normal system clock. It is then possible to extend a clock cycle in increments of the high speed clock. A cycle then may be 1, 1 1/3, 1 2/3, or 2 times the normal cycle length.

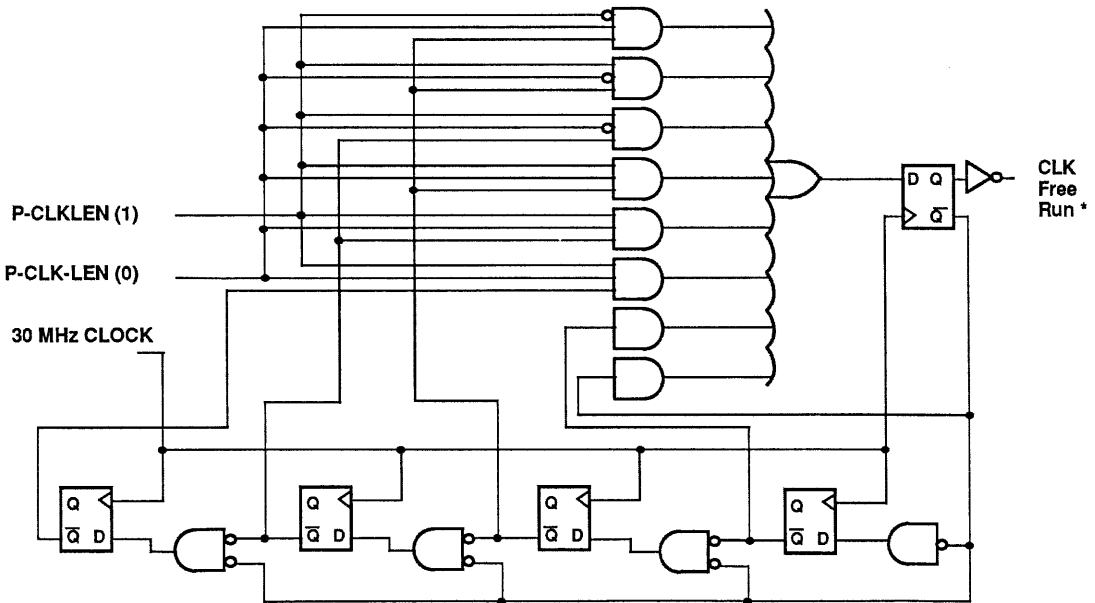


Figure 5-16. U100 AmPAL16R6B Clock Generator

09856A 5-16

The Am29300 demonstration system's normal clock is 10 MHz, or 100 ns, long. The high speed clock is then 30 MHz and is provided by a commercially available clock oscillator module.

The control over the cycle length comes from the control pipeline register and may thus be specified differently on each instruction. Two bits are provided to select one of the four cycle lengths. Each instruction may thus control its own cycle length based on the time required by the data paths that are used.

The waveform of the clock may be described in terms of the number of high speed clock periods during which it is active and then inactive.

Note that the output of the AmpAL16R6 is inverting. The logic internal to the PAL creates an "active high" clock with a low-to-high active edge. This waveform is inverted by the final output of the PAL and is later inverted once more in the clock qualifying circuit. The final system clocks are thus active high. When describing any system clock, it will be done in terms of an active high clock. The clock generator waveform is shown in Figure 5-17, where the outputs are shown active high, even though the actual PAL output is inverted.

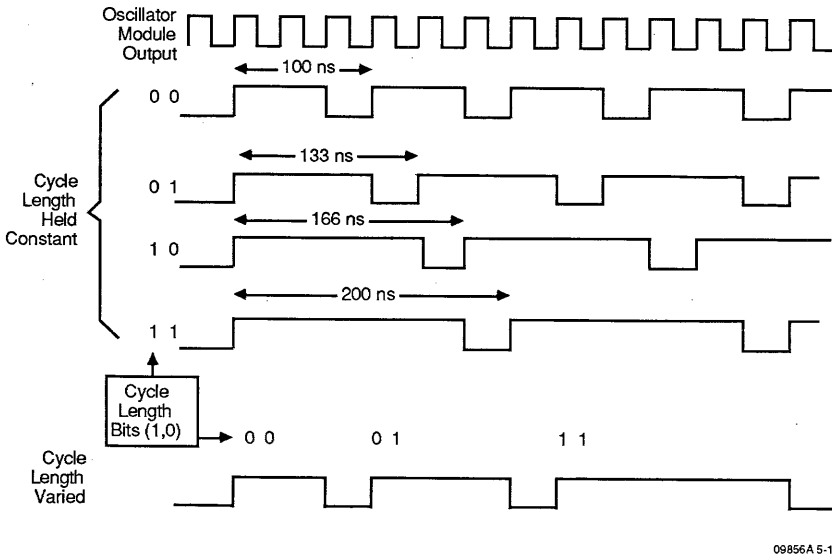
Each clock cycle has two or more active periods followed by one inactive period.

The clock generator PAL output is from a D flip flop. When the flip flop output is inactive (low), one term feeds back the inverted output. This will force the flip flop high on the next high speed clock. The output of this flip flop feeds a shift chain of four other flip flops, which act as a simple timer for the extended cycle lengths.

During the first active period of the clock output, the output of the first flip flop in the timing chain is still inactive. This first flip flop's output is inverted and fed back into the clock output flip flop to force the clock output to remain high for a second active period.

During the second active period, the clock cycle length bits from the control pipeline become stable and determine whether additional active periods will be inserted into the output clock.

Note that since the first two periods of active clock are forced by the logic, the control bits need not be stable for two high speed clock periods minus the PAL set-up time ($66.6 \text{ ns} - 15 \text{ ns} = 51.6 \text{ ns}$). This time margin is further reduced by the skew between the high speed clock and the qualified clock to the control pipeline which is equal to the clock-to-output time of the clock generator PAL plus the propagation delay of the qualifying NAND gate ($51.6 \text{ ns} - (10 \text{ ns} + 5.5 \text{ ns}) = 36.1 \text{ ns}$). Therefore, as long as the control pipeline register clock-to-output time does not exceed 36 ns, the clock generator will work as described here.



09856A 5-17

Figure 5-17. Clock Generator Outputs (Inverted)

If the clock cycle length bits are zero, no additional feedback terms are enabled and the clock output flip flop will go low in the next high speed clock period.

If the clock cycle length bits equal 1, the output of the second timing chain flip flop is fed back to the output flip flop to allow one additional active clock period.

Similarly, when the clock cycle length bits are equal to 2 or 3, an additional 2 or 3 active periods are inserted in the output clock waveform.

When the clock output flip flop again goes inactive, its output will force all of the timing chain flip flops to be cleared, thus beginning a new Am29300 clock cycle.

MICROCODE WORD

This section describes the structure and function of each field of bits in this system's microcode word. Included are some comments on how functions were determined and how they might vary in similar systems.

Control Philosophy

In a microprogrammed system, each word of the microcode functions as the determinate of all system action during one clock cycle of system operation. Each bit directly affects some aspect of the machine. Each field of bits may act independent of other fields to manage parallel data paths and simultaneous operations. This ability to manage parallel activities in each machine cycle gives a microprogrammed system high speed and flexibility. But the power of complete parallel control over nearly all the functions in a system comes at a cost.

The cost is wide control memory words. Fifty- to 150-bit-wide control words are common in microprogrammed systems. Three hundred-bit-wide control words have been used in large mainframe computers for years.

With each machine instruction's eating up 100 or more bits of memory, it doesn't take long to consume significant board space, power, and cost for high speed microcode memory.

The resulting dilemma between the need for parallel control and the cost, size, and power that accompanies it, is the basis of many a system designer's headache.

The usual approach used to strike a balance between the opposing issues is to determine carefully which functions must absolutely be able to occur in parallel, then to limit

the microcode word size to that absolute minimum. Control over other less frequently used functions or over alternate operations is then overlapped with the primary control fields.

Overlapping of control fields means that during certain operations, the meaning of the bits in the overlapped control field changes. The hardware controlled by the primary meaning of an overlapped field must be disabled during the time that the alternative meaning is in effect. This of course means that the functions controlled by the overlapped fields cannot occur in the same machine cycle.

This results in winning a little and losing a little. More control and thus more functions may be managed with less control memory, but some operations then take multiple cycles to complete, due to the use of functions that may not be managed in one instruction. Also, the need to enable and disable control field meanings and the associated hardware, will add control bits and decoding logic. The decode logic adds delay into the machine cycles and will cause the system to run a little slower.

Additional savings in control word size may be made by encoding fields rather than having each bit directly drive a control signal. This again adds decoding logic and its associated delay.

The job of deciding what control must be parallel and what must be overlapped is more art than science. No matter how the microcode word is defined, there will always be other interesting ways to rearrange and overlap the control fields. Each way will cost something either in word width or control decoding, thus providing endless trade-offs.

All these possible variations make it extremely important to have a thorough understanding of the algorithms to be handled by a particular machine. The better the understanding, the better the chance to optimize the system architecture and control to solve the problem at hand.

Microcode Word Field Descriptions

Throughout the figures that detail the design of this system, signals that travel from page to page have been given meaningful names that imply the function of the signal. This helps in understanding what is going on in each figure. Many of these signals are the direct outputs of the control store pipeline register. As it turns out, many of the bits in the microcode carry multiple meanings because the function of several fields are overlapped to save microcode word size.

The result is that more than one signal name may often be associated with a particular bit of the control pipeline. Physically, of course, all signal lines that ultimately connect to a particular pipeline bit are one piece of wire. The logical separation of lines, by using different names, only helps to understand the function of a given signal, when the hardware that uses the signal is enabled. The following three Figures show the physical and logical relationships between the microcode control store bits and the signal names (meanings) that are attached.

Each Figure is split into pairs of columns preceded by one column that indicates the individual bit numbers for each signal. Each column pair contains a Field Name column that describes the function of the bit and a Signal Name column that gives the signal name used throughout the Figures in this document for that meaning. The left most column pair shows the primary meaning of the control bits. Other column pairs to the right give alternate (overlapped) meanings for the control bits along with the signal name used with each meaning.

Unless a control bit is overlapped with an alternate meaning in one of the columns to the right, the function of the control bit is constant.

Register File Controls

Figure 5-18 shows the microcode word bits that affect the Am29334 register file.

It was decided that a three address machine would be the most appropriate way to obtain the best performance from the Am29300 family components. Because of the common three bus architecture these parts share, a three address register file fits nicely. Two addresses are used to read an A and B operand from the file while the third address specifies an independent write location. This allows writing back results without requiring the destruction of one of the read operands in a single cycle.

An address multiplexer on the C operand register address does allow for two and one address operations by allowing either the A or B operand address to be used for the write operand address in addition to its use as a read operand.

Also, to support macroinstruction execution, address multiplexers are used on the read addresses so that macroprogram supplied register addresses may be directed to the register file. When macroprogram supplied addresses are in use, the meaning of the register address fields changes to control signals for the macro operand address counters. With this alternate meaning, the macro addresses may be incremented or decremented at the end of each cycle.

Bits 91 and 84 select whether the microcode or the macro opcode addresses are directed to the register file. If either bit is high, the alternate definition for the related address field takes effect, and the macro opcode address is used.

Bits 76 and 77 are used to select one of four addresses to be supplied to the A write port of the register file. The selections are as follows:

Bit		
77	76	
0	0	C operand microcode address used.
0	1	A operand address, as specified by bit 91.
1	0	B operand address, as specified by bit 84.
1	1	C macro operand counter address used.

When any selection other than for the C operand microcode address is made, the field assumes the alternate meaning for control of the macro operand counter.

In addition to the three addresses used by the data section of the CPU, a fourth address is provided for the B write port of the register file so that data may be moved into the file via the second port while other calculations go on undisturbed.

The address for this fourth port comes from a multiplexer that may select either the C operand microcode address or the C macro opcode address counter as the source. Bit 69 is the select input for this fourth address multiplexer.

Bit 68 enables the register file A read port onto the A_BUS. If this bit is inactive and if the FPP seed register output is also inactive, the D_BUS to A_BUS transceiver is enabled so that constants, masks, and variables may be passed from the D_BUS to A_BUS.

Bits 67 and 66 are used as the write enable controls for the two write ports of the register file.

Data Path Controls

The data path controls are shown in Figure 5-19.

To provide a straightforward example of the usage of the PM and FPP, these devices have had their input and output buses paralleled with those of the ALU. In this arrangement it is not generally feasible to make use of more than one module in a given cycle. This is because the data buses may carry useful information to only one device at a time (this assumes that passing the same data to more than one device is of limited use). Also, only one device may drive the Y_BUS at a time.

Figure 5-18. Am29300 Demonstration System Microinstruction Word Layout -- Register File Controls

Control Pipeline Bit #	Primary Field Name Meaning	Primary Signal Name	Alternate 1 Field Name Meaning	Alternate 1 Signal Name	Alternate 2 Field Name Meaning	Alternate 2 Signal Name
P91	Reg A Macro/Micro*	P_ARA_MAC				
	If P91 = 0 then primary		If P91 = 1 then alternate 1			
P90	Register A Address (5)	P_RA (5)				
P89	Register A Address (4)	P_RA (4)				
P88	Register A Address (3)	P_RA (3)				
P87	Register A Address (2)	P_RA (2)				
P86	Register A Address (1)	P_RA (1)	RA Count Direction	P_UP/DN_A		
P85	Register A Address (0)	P_RA (0)	RA Count Enable	P_CNTA_EN		
P84	Reg B Macro/Micro*	P_ARB_MAC				
	If P84 = 0 then primary		If P84 = 1 then alternate 1			
P83	Register B Address (5)	P_RB (5)				
P82	Register B Address (4)	P_RB (4)				
P81	Register B Address (3)	P_RB (3)				
P80	Register B Address (2)	P_RB (2)				
P79	Register B Address (1)	P_RB (1)	RB Count Direction	P_UP/DN_B		
P78	Register B Address (0)	P_RB (0)	RB Count Enable	P_CNTB_EN		
P77	Reg C Add Source (1)	P_C_SEL (1)				
P76	Reg C Add Source (0)	P_C_SEL (0)				
	If P77:76 = 00 then primary		If P77:76 = 01, 10, 11 then alternate 1			
P75	Register C Address (5)	P_RC (5)				
P74	Register C Address (4)	P_RC (4)				
P73	Register C Address (3)	P_RC (3)				
P72	Register C Address (2)	P_RC (2)				
P71	Register C Address (1)	P_RC (1)	RC Count Direction	P_UP/DN_C		
P70	Register C Address (0)	P_RC (0)	RC Count Enable	P_CNTC_EN		
P69	B Write Port Select	P_AWB_MAC				
P68	A Bus Output Enable*	P_OEA*				
P67	A Port Write Enable*	P_WEA*				
P66	B Port Write Enable*	P_WEB*				

Figure 5-19. Am29300 Demonstration System Microinstruction Word Layout -- Data Path Controls

Control Pipeline Bit #	Primary Field Name Meaning	Primary Signal Name	Alternate 1 Field Name Meaning	Alternate 1 Signal Name	Alternate 2 Field Name Meaning	Alternate 2 Signal Name
P65	Data Path Select	(1) P_DPS	(1)			
P64	Data Path Select	(0) P_DPS	(0)			
ALU when P65:64 = 00			FPP when P65:64 = 10,11			PM when P65:64 = 01
P63	ALU Instruction	(8) P_ALU_INST	(8) FPU Instruction	(4) P_FP_I	(4) TCX	P_TCX
P62	ALU Instruction	(7) P_ALU_INST	(7) FPU Instruction	(3) P_FP_I	(3) TCY	P_TCY
P61	ALU Instruction	(6) P_ALU_INST	(6) FPU Instruction	(2) P_FP_I	(2) ACC	(1) P_ACC (1)
P60	ALU Instruction	(5) P_ALU_INST	(5) FPU Instruction	(1) P_FP_I	(1) ACC	(0) P_ACC (0)
P59	ALU Instruction	(4) P_ALU_INST	(4) FPU Instruction	(0) P_FP_I	(0) RND	P_RND
P58	ALU Instruction	(3) P_ALU_INST	(3) ENR*	P_ENR*	XSEL	P_XSEL
P57	ALU Instruction	(2) P_ALU_INST	(2) ENS*	P_ENS*	YSEL	P_YSEL
P56	ALU Instruction	(1) P_ALU_INST	(1) ENF*	P_ENF*	TSEL	P_TSEL
P55	ALU Instruction	(0) P_ALU_INST	(0) Feed Through	(1) P_FP_FT	(1) ENXA*	P_ENXA*
P54	Position Mac/Mic*	P_POS_MAC	(0) Feed Through	(0) P_FP_FT	(0) ENXB*	P_ENXB*
P53	Position	(5) P_POSITION	(5) IEEE/DEC*	P_IEEE/DEC*	ENYA*	P_ENYA*
P52	Position	(4) P_POSITION	(4) Seed Output Enable*	P_SEED_OE	ENYB*	P_ENYB*
P51	Position	(3) P_POSITION	(3) Projective/Affine	P_PROJ/AFF*	ENP*	P_ENP*
P50	Position	(2) P_POSITION	(2) Rounding Mode	(1) P_FP_RND	(1) ENT*	P_ENT*
P49	Position	(1) P_POSITION	(1) Rounding Mode	(0) P_FP_RND	(0) FA	P_FA
P48	Position	(0) P_POSITION	(0)			
P47	Width Mac/Mic*	P_WID_MAC		FTX	P_FTX	
P46	Width	(4) P_Width	(4)	FTY	P_FTY	
P45	Width	(3) P_Width	(3)	FTP	P_FTP	
P44	Width	(2) P_Width	(2)	PSEL	(1) P_PSEL	(1)
P43	Width	(1) P_Width	(1)	PSEL	(0) P_PSEL	(0)
P42	Width	(0) P_Width	(0)			
P41	Macro/Micro* Status	P_MIC/MAC				
P40	Register Status	P_REG_STAT				
P39	Load Macro Status	P_LD_MAC_STAT				
P38	Borrow Mode	P_BM				
P37	Memory Add Select	(3) P_MEM	(3)			
P36	Memory Add Select	(2) P_MEM	(2)			
P35	Memory Add Select	(1) P_MEM	(1)			
P34	Memory Add Select	(0) P_MEM	(0)			
P33	Memory Write En*	P_MEM_WR*				

Figure 5-20. Am29300 Demonstration System Microinstruction Word Layout -- Control Section Controls

Control Pipeline Bit #	Primary Field Name Meaning	Primary Signal Name	Alternate 1 Field Name Meaning	Alternate 1 Signal Name	Alternate 2 Field Name Meaning	Alternate 2 Signal Name
P32	Cycle Length (1)	P_CLK_LEN (1)				
P31	Cycle Length (0)	P_CLK_LEN (0)				
P30	Interrupt Enable	P_INT_EN				
P29	Force Continue If P29 = 1 then primary	P_FC*		If P29 = 0 then alternate 1		
P28	Seq Instruction (5)	P_SEQ_INST (5)	Interrupt Host	P_INT_HOST		
P27	Seq Instruction (4)	P_SEQ_INST (4)	Sign Extend A_BUS	P_SIGN_EX		
P26	Seq Instruction (3)	P_SEQ_INST (3)	Initialize	P_INIT		
P25	Seq Instruction (2)	P_SEQ_INST (2)	Load Interrupt Base Add	P_LD_INT_BASE		
P24	Seq Instruction (1)	P_SEQ_INST (1)				
P23	Seq Instruction (0)	P_SEQ_INST (0)				
	If P29 = 1 AND P28:27 != 11 then primary		If P29 = 0 OR P28:27 = 11 then alternate 1			
P22	Test Select (3)	P_TEST (3)	Am29114 Instruction (3)	P_INT_INST (3)		
P21	Test Select (2)	P_TEST (2)	Am29114 Instruction (2)	P_INT_INST (2)		
P20	Test Select (1)	P_TEST (1)	Am29114 Instruction (1)	P_INT_INST (1)		
P19	Test Select (0)	P_TEST (0)	Am29114 Instruction (0)	P_INT_INST (0)		
P18	Load Operand Counter	P_LD_CNT				
P17	Load Macro Op Reg	P_LD_MAC_OP				
P16	Branch Field Enable*	P_BRANCH_EN*				
P15	Branch Address (15)	D_BUS (15)				
P14	Branch Address (14)	D_BUS (14)				
P13	Branch Address (13)	D_BUS (13)				
P12	Branch Address (12)	D_BUS (12)				
P11	Branch Address (11)	D_BUS (11)				
P10	Branch Address (10)	D_BUS (10)				
P9	Branch Address (9)	D_BUS (9)				
P8	Branch Address (8)	D_BUS (8)				
P7	Branch Address (7)	D_BUS (7)				
P6	Branch Address (6)	D_BUS (6)				
P5	Branch Address (5)	D_BUS (5)				
P4	Branch Address (4)	D_BUS (4)				
P3	Branch Address (3)	D_BUS (3)				
P2	Branch Address (2)	D_BUS (2)				
P1	Branch Address (1)	D_BUS (1)				
P0	Branch Address (0)	D_BUS (0)				

If separate control bits were provided for the FPP or PM, they could perform multi-cycle operations such as Newton-Raphson division in the FPP or greater than 32 by 32 bit multiplies in the PM, while remaining detached from the input and output buses during most of the multi-cycle operation. If this were done, the ALU could operate in parallel during such operations. The cost of doing this would be an additional 15 to 35 bits added to the microcode word width. These bits would get full use only during those situations that parallel calculations are possible.

For this design it was decided to use a smaller microcode word by overlapping control bits for each of the three functional units.

Data Path Selection: Only one functional unit (data path) in the data section is chosen in any one cycle. Bits 65 and 64 select one of four options:

Bit		
65	64	
0	0	ALU enabled
0	1	PM enabled
1	0	FPP enabled
1	1	Special function

In the special function option, the FPP is enabled for calculation and the control bits are assumed to be set correctly for use by the FPP, but the output enable of the FPP is inactive with the ALU output enable active. The ALU is not enabled for calculation in the sense that its hold input is made active to prevent state change in the status or Q registers.

This odd-looking combination is used to provide input operand parity checking for the FPP. The FPP does not have its own parity checking circuits, so with this arrangement the ALU parity checkers will be enabled by the active output enable on the ALU. The FPP is still allowed to function and may complete its operation and store the result in its internal registers, while in the same cycle the input operand parity is checked by the ALU. The ALU state is left undisturbed by this operation.

How useful is this scheme? It may save a cycle once in a while, but mainly it illustrates the odd sort of opportunities one may find to use up an otherwise wasted control code.

ALU Path: When the data path select bits enable the ALU meaning for bits 63:38, bits 54 and 47 are used to select either the microcode or macroinstruction position and width fields. The macro supplied information is selected when these select bits are high. When the macro source is selected, the microcode position and width fields are unused.

Bit 41 selects macro or micro status inputs for the ALU. Bit 40 selects whether the status output of the ALU is flow-through or registered.

Bit 39 is used as a clock qualifier for the loading of the ALU external macro status register.

Bit 38 directly controls the Borrow mode of the ALU.

FPP Path: When the data path selects enable the FPP, the control bits shown directly manage the operation of the FPP as described by the Am29325 data sheet. Bit 52 is used to enable the output of the FPP external "division seed" registered PROM.

PM Path: When the data path selects enable the PM, the listed control bits are used as defined in the Am29C323 data sheet.

Data Path Enabling: What does it mean to enable or disable one of the functional units? The control bits that are shared between each functional unit are either high

or low every cycle, and they are connected to the ALU and multipliers all the time. There is no intervening logic that turns all the control bits "off" when a particular path is not selected. Each device sees a jumble of nonsense on its control lines whenever the control field meaning is intended for another device. Nonsense or not, each device will do whatever the control bits specify.

Enabling a data path means making the output enable of the selected device active so that it drives the Y_BUS and is able to write calculation results back into the register file. In the case of the ALU, enabling also means that the ALU hold input will be made inactive so that state change of the ALU status and Q registers is allowed. Enabling one path implies disabling the other paths.

For the PM and FPP, disabling means their output enables are inactive. It also means that the PM product register feed through pin is disabled by the control decode logic. For the FPP it means that both of its register feed through lines are disabled by control decode logic. These register feed through controls are disabled because, if they are allowed to be active, it is possible for the PM and FPP multipliers to feedback on themselves and begin to oscillate. This action would not damage the devices, but it could add to power consumption and system power plane noise. A simple prevention is just to disable the feed-throughs when the data paths are not selected. Note that the ALU has no internal feedback paths and does not need any similar treatment.

Memory Control: Bits 37:33 are available at all times to control the Am29300 system memory.

Bit 33 is the memory write enable control.

Bits 35:34 select the source of the address for the memory.

Bit		
35	34	
0	0	No memory address or operation is selected
0	1	A_BUS data is used to address memory
1	0	The A memory address counter is selected for address
1	1	The B memory address counter is selected for address

Bits 37:36 select the following:

Bit		
37	36	
0	0	Load counter A
0	1	Load counter B
1	0	Selected counter is incremented
1	1	Selected counter is decremented

The increment and decrement commands have effect only when a counter is selected as the MA_BUS source. The load commands have effect only when the A_BUS is the selected source.

Control Section Controls

Figure 5-20 shows the bit definitions for the control section.

Pipeline bits 32:31 control the length of each machine cycle.

Bit		
32	31	
0	0	Normal cycle length
0	1	1.33 x Normal cycle length
1	0	1.66 x Normal cycle length
1	1	2 x Normal cycle length

Bit 30 enables sequencer interrupts on a cycle by cycle basis.

Bit 29 is the Force Continue signal for the sequencer. When this bit is active, the sequencer will execute a continue instruction regardless of the state of the sequencer instruction or test select lines. This effectively enables the alternate meaning for the sequencer instruction and test select fields.

Bits 28:19 are normally the sequencer instruction and test select inputs. When Force Continue is active, the sequencer instruction field meaning changes.

When Force Continue is active, bits 28:25 are used to control four individual functions. Bit 28 will send an interrupt signal to the host system. Bit 27 will enable the sign extension of data going from the D_BUS to the A_BUS. Bit 26 will force the control pipeline register to load data from the control store initialize register at the next active system clock. Bit 25 will enable the loading of the interrupt base address register.

Bits 22:19 are used to control the sequencer test selection. When an unconditional sequencer instruction is in effect or when the Force Continue bit is active, bits 22:19 are used to control the Interrupt controller instruction.

Bit 18 is used to load the macro operand counters from the macro opcode register.

Bit 17 is used to load the macro opcode register.

Bit 16 enables the three-state outputs on the branch field bits of the control pipeline register. If these outputs are disabled, then the sequencer, A_BUS to D_BUS transceiver, or Interrupt Controller may drive the D_BUS. How a device is chosen to drive the D_BUS is explained in the control decode logic description. It is only important to note that if bit 16 is active, the branch field outputs will be active and will have priority over any other driver on the D_BUS.

Bits 15:0 are the branch address field to the sequencer. This field is also used to contain constants or masks. These may be used by the data section, sequencer, interrupt base register, or interrupt controller. It is a full 16 bits long in order to allow for constants or masks that fill half of the 32-bit data path. This allows 32-bit microcode supplied masks to be formed with two microinstructions.

Alternate Arrangements

The microcode word size just defined for this system totals 92 bits wide. Having so many bits allows the flexibility to change the control over most of the machine's functions on any or every cycle. But, this degree of control flexibility is not required for every application. The size of the control store may be reduced based on how the system is used most often. Following are a few comments on ways to rearrange and reduce the control store size.

Current Control Bit Usage

First let's look at how the control bits are used in this design.

Seven of the bits are used to control the selection of alternate field meanings (i.e., overlap control in bits 91, 84, 77:76, 65:64, and 29).

Eleven bits are used to control functions that are desired to operate in all cycles, independent of other system operations. These are the register file write and read enables (bits 69:66), memory controls (bits 37:33), and the cycle length controls (bits 32:31).

Eight bits generally do not change state frequently. Their existence in this design is a convenience that reduces the need for control decode logic and adds system flexibility. These bits are 41:38, 30, 18:16.

Three bit fields are used only with some instruction types. These are the position, width, and branch fields. Whenever a particular instruction does not use a field, those bits in the field are currently wasted in that instruction cycle.

Alternative Usage

The bits that change infrequently could be replaced by decode logic that provides these same control signals via set-reset flip flops. The flip flops would be controlled by overlapping set and reset commands with some other control store field. This would add to the decode logic complexity and would limit when the flip flops could be changed by restricting the control over them to certain instruction types. Since they change only infrequently, the requirement to use certain instruction types when setting or resetting them should not be a problem.

Those bit fields that are limited to certain instruction types could be overlapped. An example might be to overlap the position and width fields with the branch address field. This would restrict branches to instructions that do not require the position or width information.

When alternative field meanings are enabled, often the alternative definition does not make use of all the bits in the field. This presents the opportunity to overlap other control bits that may be valid in the same cycle as the alternate meaning of the field.

For example, some of the infrequently-used control bits could be overlapped with the unused bits of the register C address when the primary meaning of the C address field is not active. When a two address instruction is executed, the address for the C register comes from the A or B address, thus leaving the microcode field for the C register address available for other functions.

In another example, the bits in the position and width fields that are not used by the PM or FPP could be overlapped with other control functions when the alternate meanings for the field are in effect. An alternate branch address field might be placed in those bits to allow branch instructions in combination with FPP or PM operations without the need for the currently defined branch field.

Careful analysis of how each data path is used may also allow reductions through the elimination of controls that are not needed. As an example: if the PM were used

only in flow through mode, all the controls for register enables, flow through modes, and input multiplexers could be removed from the microcode word and those inputs to the PM tied to fixed voltage levels. If only two's complement mode is used then an additional two bits may be eliminated. This would leave only four necessary control bits, the accumulator controls, rounding mode, and format adjust. This reduction might allow PM operations to be overlapped with some multiply-accumulate operations in the FPP.

By combining these reduction techniques, the following changes could be made:

All of the eight infrequently used control bits could be moved to overlap with the C register address, with half in effect when the A address is substituted for the C address and half in effect when the B address is substituted.

The PM controls, except for flow through and two's complement mode, may be moved to overlap with the position, width, and memory control fields. Also, the fourth data path select field may be changed to disable the memory controls and select the ALU — minus the position and width fields — to be active along with the PM. In this mode the PM flow through and two's complement mode controls would be fixed with no flow through and two's complement mode active. The ALU position and width inputs would be set to 0 and 31 respectively by control decode logic (unless these fields were selected to come from the macro opcode).

The branch address field may be moved to overlap with the position, width, and memory control fields. When ever the sequencer instruction selects a branch operation, the position, width, and memory fields are disabled and the branch address meaning substituted.

If all of these changes are made, the currently defined branch address field and infrequently used control bits may be eliminated, which would save 24 bits of microcode word width. This would reduce the word size to 68 bits.

This savings would come at the cost of allowing branch instructions only when the ALU instruction does not need position or width information from the microcode (this information may still come from the macro opcode register) and when the system memory is not being used. Further, a PM operation could not occur with a memory access in the same cycle. Also, with these changes it would be possible to control the ALU and PM concurrently when the ALU does not need position or width information and when the PM operates on internally registered data.

There are many such combinations of microcode control field definition. Each one provides a different trade-off between word size and what operations may be concurrent. Each one requires a different degree of complexity in the control decode logic.

CONTROL DECODE

What Is It Good For?

The ideal microprogrammed system has a separate microcode control store bit for each control input that exists in the system. This kind of complete control over every aspect of the system directly from the control pipeline totally eliminates the need for decoding the meaning of any system control bits. It also requires a very large microcode word to manage most useful systems. So in the real world, most microprogrammed systems encode or overlap at least some control functions in the microcode word.

Encoded control or not, each control input in the system requires valid voltage levels during each machine cycle if the system is to operate as expected.

The control decode logic acts as the bridge between encoded or overlapped (i.e., sometimes unavailable) microcode control fields and the related control signals in the system. The control decode logic continuously provides valid logic levels for those control signals that cannot be directly driven by the control pipeline register.

If the control field for a particular function is encoded, the control logic translates the function codes into individual control signals. Where control fields are overlapped, the control logic may be used to disable logic affected by a control field when that field has a meaning different than that intended for the logic being disabled (i.e., when overlapped control is active).

In some cases, control logic is used to prevent harmful conflicts between the meaning of different control bits, for example when two separate control fields affect the three-state enables on different buffers which may drive the same signal line. Certain combinations of control bits might enable both buffers in the same cycle causing contention between the buffers. Allowed to continue for long periods, this kind of contention may destroy the buffers. Control logic may be used in this situation to disable one or both buffers when the combination of controls affecting them would otherwise cause damage. In fact it is strongly recommended that this kind of problem always be avoided by designing the control decode logic to prevent such disasters. The alternative is to watch hardware melt because of a software mistake.

Control Logic Description

Some of the control logic function in this demonstration system has been distributed into the devices being controlled. This is done when a PAL is used to implement a function. A PAL generally has excess inputs and internal logic that may be put to use in decoding the meaning of encoded control fields(e.g. the memory address counters). The memory address counters are implemented from AmPAL22V10 devices and are shown in Figure 4-7. The control for loading, incrementing, decrementing, and output enabling the counters is provided directly from the encoded memory control field. The PALs internally decode the meaning of the control bits.

When a device requires a decoded control signal, the signal must come from control decode logic that takes control pipeline bits as input and produces the needed control signal. In this system, the required control logic has been implemented in three AmPAL18P8B PALs. These PALs are fast to minimize the delay induced between the control pipeline register and the device controlled. The PALs also provide the convenience of having programmable output levels, either high or low active for each output, independent of other outputs.

The block diagram for these PALs is shown in Figures 5-21 and 5-22. The logic definition files for these PALs are in Appendix M.

The ALU output enable, ALU hold, and PM output enable are all direct results of the pipeline data path select bits.

The pipeline controls for seed register output enable, PM flow through, and FPP flow through are gated by the appropriate data path selection so that each control signal is active only when the related data path is selected.

The D_BUS to A_BUS direction of the D_BUS transceiver is enabled by the register file A output's being disabled in conjunction with the seed register output's being disabled.

The A_BUS to MD_BUS buffer is enabled by certain codes of the memory control field.

The control store initialize register select is enabled by the combination of the pipeline Force Continue and the pipeline control bit for the initialize select. It is also enabled by the WCS_INIT* signal from the host interface controller. Note that the initialize control is synchronous as used in this system so that the initialize word is loaded only at the next active clock.

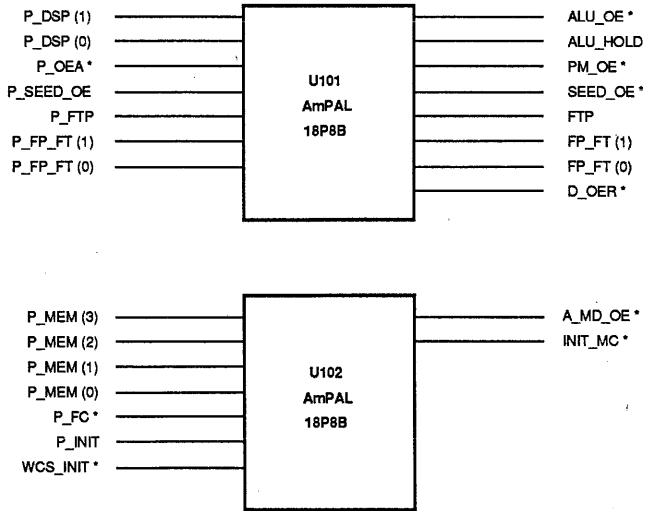


Figure 5-21. Control Decode Logic Part 1

09856A 5-21

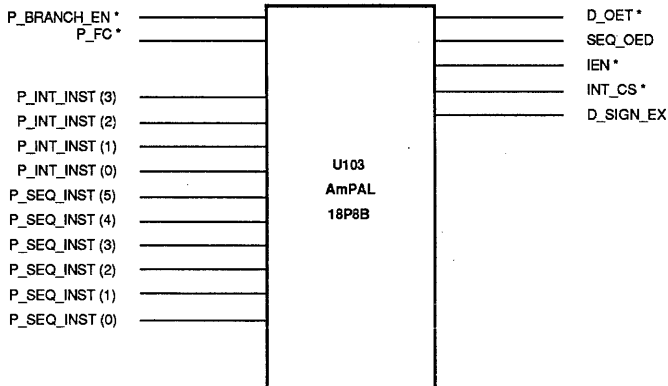


Figure 5-22. Control Decode Logic Part 2

09856A 5-22

The D_BUS sign extend, Sequencer output enable, Interrupt controller instruction and chip select enables, and A_BUS to D_BUS enable are all direct results of the pipeline sequencer instruction, interrupt controller instruction, branch enable, and Force Continue bits.

The Sequencer output enable, A_BUS to D_BUS enable, and interrupt controller chip select are used to control which device is allowed to drive the D_BUS in any given cycle. These output enables are arranged in a priority with only one output allowed to be active in any cycle; including the branch field of the control pipeline.

The highest priority output is the branch field. If it is enabled all other outputs are disabled.

If the branch field is disabled, then the Sequencer D output is enabled if either a Continue or a Pop D instruction is being executed.

If neither the branch field nor the sequencer are enabled, then the interrupt controller may drive the D bus if the interrupt controller instruction is one of three read operations.

If none of the above conditions exist to enable the other D_BUS devices, then the A_BUS to D_BUS transceiver path is enabled.

Note that the interrupt controller chip select is treated as both an instruction enable and as an output disable. The chip select is active whenever there is a valid interrupt instruction that would not cause a conflict with another driver of the D_BUS. This means that when there is a valid instruction, the chip select will be inactive only if a read instruction is selected and either the branch field or sequencer are already enabled on the D_BUS. If any other interrupt instruction is in effect, the interrupt controller does not drive its outputs.

The above scheme for managing the access rights to the D_BUS may seem a bit complex but it allows great flexibility in movement of information over the D_BUS. Information may be moved between the interrupt controller and sequencer, interrupt controller and A_BUS, or sequencer and A_BUS. Information may be loaded into the interrupt base address register from the pipeline, sequencer, or A_BUS. Also, the pipeline may provide data to the sequencer, interrupt controller, interrupt base address register, or A_BUS.

SECTION 6

System Timing and Critical Path Analysis

DEFINITIONS

The upper limit on system speed is set by the slowest signal propagation path in the system.

The length of a signal propagation path is measured from the output of one register to the input of another register, where all registers are loaded by the same clock.

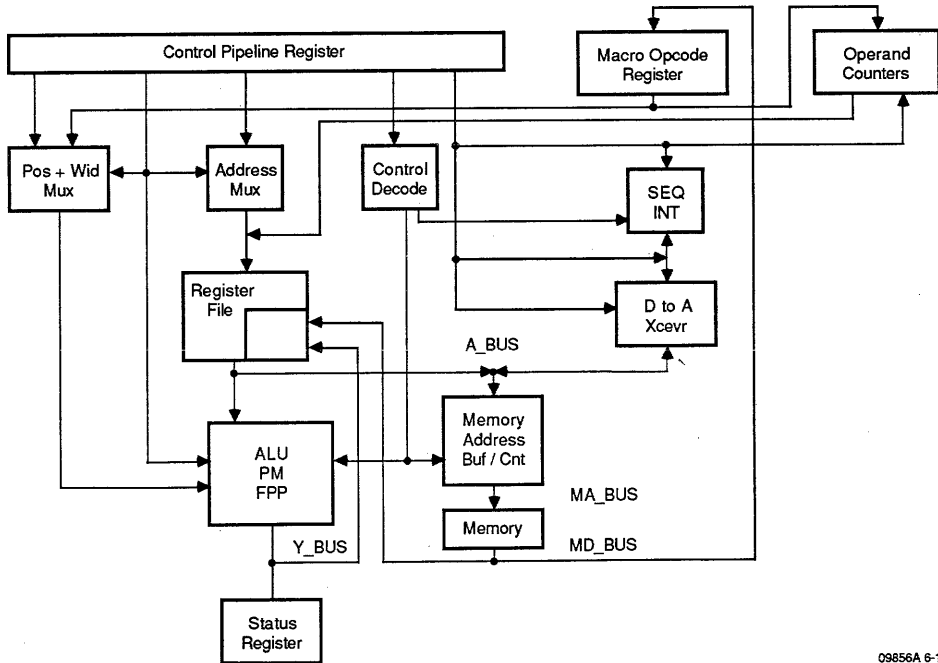
The slowest signal path will be different for different control states. An example would be the selection of the ALU data path vs. the FPP data path.

A signal path may be slower in the first cycle that control selects the path than it will be in a subsequent cycle that maintains the same path selection. This can be due to three-state enable or disable times being longer than normal propagation delays of the circuits involved.

CONTROL AND DATA PATHS

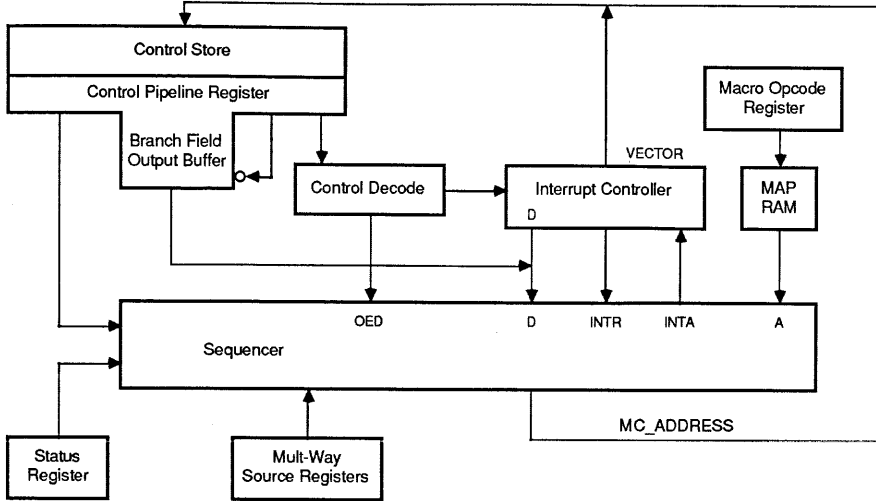
In determining the maximum system speed, every signal path must be analyzed. This requires tracing every control signal and every data signal and totaling the delay induced by each component along the path from source register to destination register. Where parallel paths exist, the time delay for the slowest path is used.

Most often, the critical (slowest) paths originate with the pipeline control register. In the data section the paths will end with data being loaded into the register file, an FPP or PM internal register, the system memory, or a D_BUS destination. In the control section the paths will end with loading of new control bits into the control pipeline register.



09856A 6-1

Figure 6-1. Data Section Timing Paths



09856A 6-2

Figure 6-2. Control Section Timing Paths

Since the control section and data section operate in parallel, the slowest path in either section will determine the cycle length required for a specific operation.

Figures 6-1 and 6-2 provide a block diagram view of significant signal pathways for both control and data lines in both the control and data sections.

Referring to these figures as critical timing paths are discussed may help in following the timing analysis.

In this and nearly any complex system, there are hundreds of pathways that must be traced in order to ensure finding all the worst case delays. To go through all of them here would require too much time and space. Many of the timing paths for this design have already been analyzed, and what appear to be the worst case paths will be shown here.

WORST CASE PATHS

Each case is shown in Table 6-1. The table is separated into several pages due to its length. It can be viewed as a long spreadsheet calculation in which the appropriate timing parameters that apply to each case have been selected and placed in the correct column. Only the worst case delay for each segment of a critical path is shown. Parallel but faster paths have been eliminated from each case so that the total of the times listed for a case represents the minimum time in which a path can be traveled.

Case Definitions

1. Basic flow-through calculation, data path.

Data is moved from the register file through the ALU and back to the register file. The timing path begins at the control pipeline where the register file address for the A and B read operands appear after the clock to output delay of the control pipeline register. These addresses flow through the Am29827 buffer that forms one side of the register file address multiplexer. The address accesses the register file and one access time later the data operands are presented to the ALU. By this time the control signals for the ALU instruction have been stable long enough that the flow through time of the data in the ALU will be the slower path. Once data is on the Y bus the last delay is the set-up time for the register file before clock can occur. Again, the control path to the register file (A port write address) is faster than the data path so the data path is the limiting factor.

The total delay for this path is 96 ns. If the PM path is substituted for the ALU the delay would be 174 ns. If the FPP were substituted, the delay is 179 ns. So flow through calculations with either of the multipliers will require extended cycle length.

2. Basic flow-through calculation, position control path.

This case is the same as Case 1 except that a careful look at the control path for the position input to the ALU is taken. This path turns out to be 97 ns worst case. This is an example where the control path is a little slower than the data path.
3. Flow-through calculation with address supplied by the Macro operand counter; counter output enabled same cycle.

Again this path is similar to Case 1. The difference is that the read addresses are assumed to come from the Macro operand counters. It is further assumed that the counters are selected during the cycle analyzed. This means that the output enable time of the counter must be added to the clock to output time for the pipeline bit that selects the macro opcode counter.

This increases the delay path to 115 ns, indicating that during the first cycle, in which a macro opcode counter is used as the address source, the cycle length will need to be extended.
4. Flow-through calculation with address supplied by the Macro operand counter; counter output enabled prior cycle.

This case is a comparison with Case 3, where the Macro operand counter was output enabled in the previous cycle. The counter delay is thus limited to the clock to output delay of the counter. This reduces the cycle time requirement to 90 ns. So, sequential register file address cycles, using an operand counter can be completed within the normal cycle time.
5. First cycle of FPP Newton-Raphson division, seed value load.

In this case the critical path starts at the control pipeline clock to output delay, and then goes through the control decode logic that enables the output of the Seed register. In this case it is assumed that the seed value is multiplied and stored in an FPP internal register to complete the first cycle of a Newton-Raphson division. This requires a total of 169 ns. Note that if the seed value had simply been moved into the input register of the FPP, the total delay would have only been 73 ns.
6. Memory read with address from the register file, selected by microcode.

This is a simple memory read with the time starting at the pipeline clock to output delay, followed by the address mux, register file access, A_BUS to MA_BUS buffer, memory, and register file data set-up time. The total time comes in at 99 ns, just under the desired 100 ns basic cycle time.
7. Memory read with address from a memory address counter.

Here the access time of the register file is essentially traded for the output enable time of a memory address counter. The total delay only improves to 94 ns, but there is a big advantage in the fact that for a sequential access the CPU did not need to calculate a memory address. This will save at least one cycle. Also, it is possible to overlap a memory read from an address counter with a calculation cycle in the CPU.
8. Memory write with data from register file, selected by operand counter.

In a memory write case, time is saved by needing only to meet the data set-up time of the memory rather than the memory access time plus the register file set-up time, as would be the case in a read operation. In this case the time gained is traded for the time required to output enable an operand counter. Even so, the total time is still 94 ns.
9. Move register file data to interrupt controller or sequencer, data selected by operand counter.

Here again, the long delay path of using a macro opcode counter as the register file address source is used. Even with the output enable delay of the counter in addition to the pipeline clock to output time, the total delay comes in at 89 ns.

10. Move sequencer or interrupt controller data to register file.

In the reverse of the above case, the time to get data from D_BUS is similar to the time in Case 9 to access data from the register file. The big delay here is the need to move the data from the A_BUS, through the ALU and back to the registerfile. Not having a direct path to the Y_BUS has cost a good bit of time. The total comes in at 127 ns. Fortunately this type of data move is not likely to be a commonly executed cycle.

11. Sequencer branch, conditional or unconditional.

In this case much of the delay is in the pipeline clock to output time for the branch field enable bit, cascaded with the output enable time of the branch field in the control pipeline register. This is followed by the branch address flow through time of the sequencer and the access time of the control store. Even with all the delay, this path is significantly faster than most of the data section paths. The total time is 84 ns.

12. Sequencer interrupt or trap cycle.

In this case the pipeline output doesn't turn out to be in the main delay path. The interrupt starts at the clock to output delay of the trap logic where the interrupt request is generated. The sequencer then responds with interrupt acknowledge, which in turn output enables bit 3 of the interrupt vector from the trap logic. The interrupt

vector then accesses the control store. The total for this cycle is 81 ns.

13. Sequencer branch to macro opcode specified instruction.

Here the initial delay is the clock to output delay of the macro opcode register, followed by the access time of the map RAM. Next is the branch flow through time for the sequencer and the access of the control store. This cycle comes in at 85 ns.

FINAL RESULTS

Several cases were shown here to help give an idea of how fast the system is for different instructions. These cases were some of the worst identified during the critical analysis of this design. All but three of the cases shown fit within the desired 100 ns basic clock cycle. Two of the cases would only require a cycle 1 1/3 times normal. Case 5 officially needs a double length cycle.

As noted in the discussion of Case 1, both the PM and FPP require much longer cycles for flow through calculations. If the PM and FPP are used in clocked multiply mode for sequential pipelined multiplies, as would occur in array calculations, the cycle time can be significantly reduced. In clocked multiply mode the PM or the FPP requires only 100 ns cycle times.

With a dynamically variable clock cycle length, this system can run most instructions at the basic 100 ns cycle rate, but will still handle the occasional extended execution time instructions.

Table 6-1A

Data Path Element Parameter Description	Worst Case Time Delay in Nanoseconds, Over Commercial Operating Range														
	Symbol	Value	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9	Case 10	Case 11	Case 12	Case 13
Control Store/Register - Am9151-50															
Clock to Output	Tpkhdqv1	15	15	15			15	15	15	15	15	15	15		
OE to Output Valid	Tgldqv	20											15		
Synchronous!													20		
I to Clock Set-up	Tivpkh	25													
Address to Clock Set-up	Tavpkh	30											30	30	30
Control Decode Logic - AmPAL18P8B															
Input to Output	Tpd	15					15					15			
Macro Opcode Register - Am29818-1															
Clock to Output	Tpd	11													11
Input to Clock Set-up	Ts	6													
Macro Operand Counters - AmPAL22V10A															
Clock to Output	Tco	15				15									
Input to Clock Set-up	Ts	20													
OE to Output Valid	Tea, Ter	25			25					25	25				
Reg File A or B Read Add Mux - Am29827A															
Input to Output	Tph	6	6							6					
OE to Output Valid	Tzh	10													
Reg File C Write Add Mux - AmPAL18P8Q															
Input to Output	Tpd	35													

Table 6-1C

Data Path Element Parameter Description	Worst Case Time Delay in Nanoseconds, Over Commercial Operating Range														
	Symbol	Value	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9	Case 10	Case 11	Case 12	Case 13
Floating Point Processor - Am29325															
Unclocked Operations		125													
Clock Operation		100													
Clock Multiplied, Data to Clock Set-up	Tsd1	13													
Clock Multiplied, Data to Clock Set-up	Tsd2	104					104								
FPP Seed Register - Am2920 & Am27S25															
OE to Output Valid	Tzh	35					35								
FPP External Status Register -AmPAL22V10A															
Clock to Output	Tco	15													
Input to Clock Set-up	Ts	20													
Macro Status Register - Am29818-1															
Clock to Output	Tpd	11													
Input to Clock Set-up	Ts	6													
Memory Address or Data Buffer -Am29827															
Input to Output	Tph	10						10		10					
OE to Output Valid	Tzh	17													
Memory Address Counters - AmPAL22V10															
Clock to Output	Tco	25													
Input to Clock Set-up	Ts	30													
OE to Output Valid	Tea, Ter	35							35						

Table 6-1E

Data Path Element Parameter Description	Worst Case Time Delay in Nanoseconds, Over Commercial Operating Range														
	Symbol	Value	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9	Case 10	Case 11	Case 12	Case 13
Sequencer - Am29331															
Branch Input to Y Output		19											19		19
Instruction to Y Output		25													
Instruction to D Output		31													
Force Continue to Y Output		21													
Interrupt Request to Interrupt Ack		11												11	
OE D to D Valid		25										25			
Minimum Cycle Time per Case			96	97	115	90	169	99	94	94	89	127	84	81	85

Table 6-1F

Case Definitions

1. Basic flow through calculation, data path.
Pipeline, Tco; Address Mux, Tpd; Register File, Tpd; ALU, Tpd; Register File, Set-up.
2. Basic flow through calculation, position control path.
Pipeline, Tco; Position Mux, Tpd; ALU, Tpd; Register File, Set-up.
3. Flow through calculation with address supplied by operand counter; counter output enabled same cycle.
Pipeline, Tco; Operand Counter, Tea; Register File, Tpd; ALU, Tpd; Register File, Set-up.
4. Flow through calculation with address supplied by operand counter; counter output enabled prior cycle.
Pipeline, Tco; Operand Counter, Tco; Register File, Tpd; ALU, Tpd; Register File, Set-up.
5. First cycle of FPP Newton-Raphson division, seed value load.
Pipeline, Tco; Control Decode, Tpd; Seed Register, Tzh; FPP Internal Register Set-up, Tsd2.
6. Memory read with address from the register file, selected by microcode.
Pipeline, Tco; Address Mux, Tpd; Register File, Taa; Memory Address Buffer, Tpd; Memory, Taa; Register File, Set-up.
7. Memory read with address from a memory address counter.
Pipeline, Tco; Control Decode, Tpd; Memory Address Counter, Tzh; Memory, Taa; Register File, Set-up.
8. Memory Write with data from register file, selected by operand counter.
Pipeline, Tco; Operand Counter, Tea; Register File, Taa; Memory Address Buffer, Tpd; Memory, Write Set-up.
9. Move register file data to interrupt controller or sequencer, data selected by operand counter.
Pipeline, Tco; Operand Counter, Tea; Register File, Taa; A to D Bus Xceiver, Tpd; Interrupt Controller, Data Set-up.
10. Move sequencer or interrupt controller data to register file.
Pipeline, Tco; Control Decode, Tpd; Sequencer, OED to D; D to A Bus Xceiver, Tpd; Parity Buffer, Tpd; ALU, Tpd; Register File, Set-up.
11. Sequencer branch, conditional or unconditional.
Pipeline, Tco; Pipeline Branch Field, Tzh; Sequencer, D to Y; Control Store, Address Set-up.
12. Sequencer interrupt or trap cycle.
Trap Logic, Clock to INTR; Sequencer, INTR to INTA; Trap Logic, Tea; Control Store, Address Set-up.
13. Sequencer branch to macro opcode specified instruction.
Macro Opcode Register, Tco; Map RAM, Taa; Sequencer A to Y, Control Store, Address Set-up.

SECTION 7

Physical Issues

ELECTRICAL LAYOUT ISSUES FOR POWER SUPPLY

The TTL compatible, bipolar, Am29300 family components all use internal ECL circuitry with TTL compatible I/O buffers.

Each part has a large number of output buffers due to the 32-bit output bus, plus various status outputs.

These two facts can make the real world interesting.

When a large number of the output buffers switch simultaneously, the local Printed Circuit Board (PCB) power and ground, and the chip internal power supply lines can experience significant noise transients.

This power supply noise can couple into the internal logic's ECL VCC pins. Since the internal ECL circuitry is referenced to the ECL VCC, the power supply noise can cause short duration shifts in the threshold levels of the internal logic.

Due to the way ECL circuitry operates, it has much smaller noise margins than equivalent TTL circuits. The threshold shifts result in lower than normal noise margins in already sensitive high speed circuits. These reduced noise margins can result in noise-induced logic errors.

It is, therefore, very important to provide very good power distribution and decoupling in a system using the Am29300 family. It is strongly suggested that a multi-layer PCB be used to provide power and ground planes. It is also important to minimize coupling between the TTL and ECL VCC pins of any Am29300 bipolar device. This can be done in part through good power supply decoupling.

An additional way to decouple the ECL and TTL VCC pins is to introduce inductive isolation. The simplest way to do that is to place a cut in the VCC plane that separates the ECL supply pins from the TTL pins. This produces a

longer electrical path between the pins, which adds inductance between the pins. This inductive isolation will significantly reduce noise coupling.

Some suggested PCB layouts for use with the Am29300 family are shown in Figures 7-1a and 7-1b. The images are negatives where black indicates an absence of metal in the VCC plane.

Although significant noise can also occur on the TTL and ECL ground lines, the ECL circuits are much less sensitive to this noise. Attempting to isolate the TTL and ECL ground pins from each other can create more problems than it solves. Any isolation will reduce the noise in the ECL circuitry and thereby make the chip internal ECL ground "different" from the TTL ground. This can reduce the noise margin in the ECL to TTL conversion logic, introducing potential for noise induced errors. It is recommended that no isolation between grounds be used.

DECOUPLING CAPACITORS

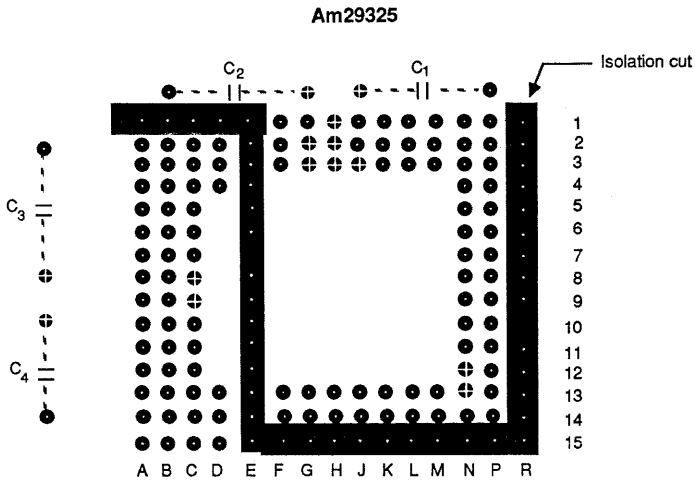
An added help in providing local VCC to ground decoupling is available in the form of under-chip capacitors.

Special capacitors for PGA device packages have been developed by Rogers Corporation, Q-PAC Div., 2400 South Roosevelt St., Tempe, AZ. 85282.

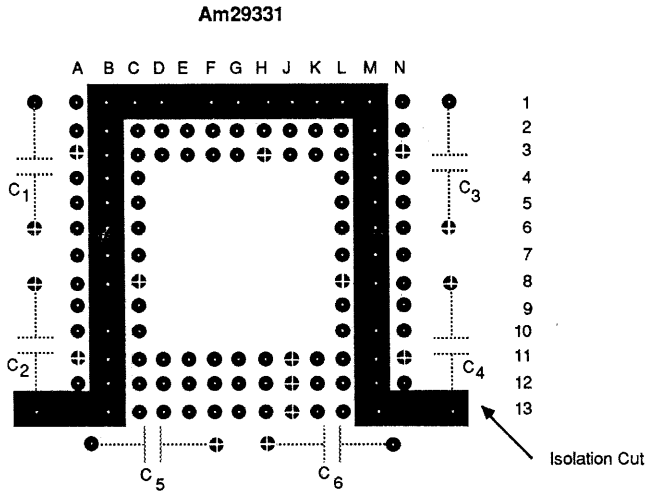
SOCKETS

Whenever high pin count, expensive VLSI components are used in a system, many hardware designers prefer to have the devices in sockets. This allows easy removal for repairs or upgrades and provides an additional test point in the system.

Sockets for the Am29300 family are available from Augat Corporation, Interconnection Component Div. 33 Perry Ave. Attleboro, MA. 02703.



05621D
29325



- = Through Hole
- ⊕ = V_{CC} Plane Connection
- C₁ = C₃ = C₅ = 10 μF
- C₂ = C₄ = C₆ = 0.1 μF

0572D-1
29331
09856A 7-1

Figure 7-1a. Layout Recommendations for the V_{cc} Plane

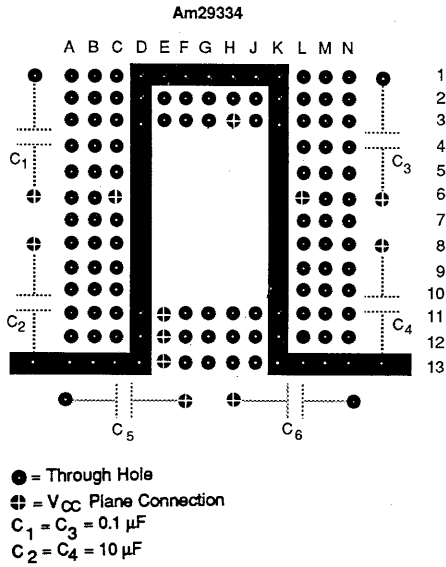
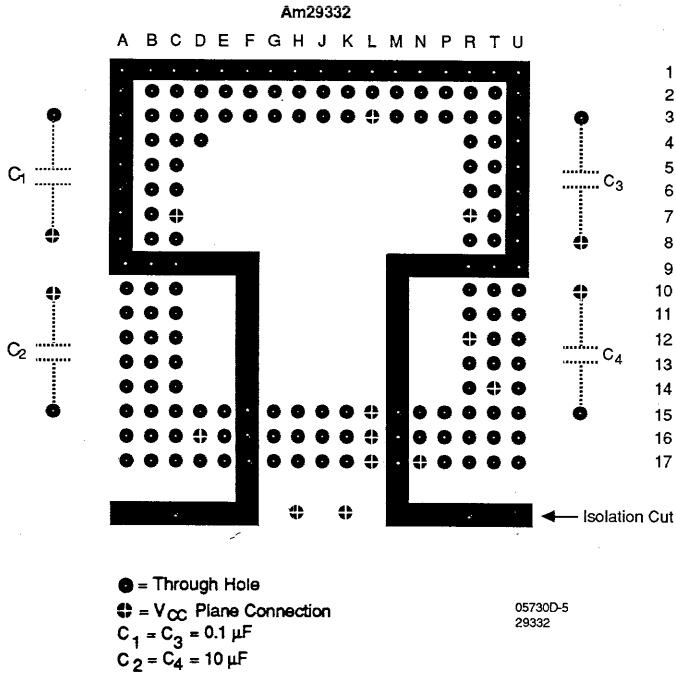


Figure 7-1b. Layout Recommendations for the V_{CC} Plane

SECTION 8

Conclusion

There are many ways to skin a cat and surprisingly, many more ways to build a computer. This application note has tried to guide the reader through just one simple implementation. The author hopes some of the reasons behind the design choices in a microprogrammed computer design were made clear during the course of the description.

Aside from some general notions about how a microprogrammed system works, the reader should walk away having noted the following thoughts:

This design is a full 32-bit processor capable of executing a full 32-bit add, barrel shift, logical, integer multiply, or even floating point multiply every 100 ns to 133 ns. That is a 7 to 10 Million Instructions Per Second (MIPS) rate, which is (loosely) comparable to 7 times the performance of a VAX 11/780.

For all that computing horsepower, the real core of this machine is made from only 6 chips: the Am29300 family of computer building blocks. That's an incredible degree

of integration as compared with previous approaches to high performance microprogrammed computer design.

Most of the logic surrounding the Am29300 family components is not required. The additional logic is used to add system flexibility and to show off different aspects of microprogrammed design. Very little glue is needed to hold this family together.

There is very little in the way of standard SSI logic used. Virtually all the MSI and SSI level logic functions were incorporated into Programmable Array Logic. This shows the versatility and integration that PALs can provide.

Due to use of Serial Shadow Registers throughout the system, there is a reasonable hope that enough of the system state can be read and controlled so that debugging in the factory or field will be simple. This access to the internal structure of the machine is gained with very little "excess" logic.

This application note, augmented by 60 pages of PAL and Am29PL141 definition files is available as a separate booklet; Publication No. 09856A.

CHAPTER 6 (continued)

6.3	The Fast Way to Build a RISC Processor	6-92
6.4	Fault-Tolerant Chips Increase System Reliability	6-97
6.5	Floating-Point Math Handles Iterative and Recursive Algorithms	6-102
6.6	Floating-Point Array Processor Improves Computational Power	6-109
6.7	Floating-Point μP Implements High-Speed Math Functions	6-116
6.8	Optimize Your Graphics System for Both 2D and 3D	6-123
6.9	Variable-Width FIFO Buffer Sequences Large Data Words	6-136
6.10	Digital Systems VME 29300-1	6-141
6.11	Bibliography	6-144

Product Application

The fast way to build a RISC processor

A family of 32-bit VLSI ICs yields reduced instruction-set computers with a variety of architectures

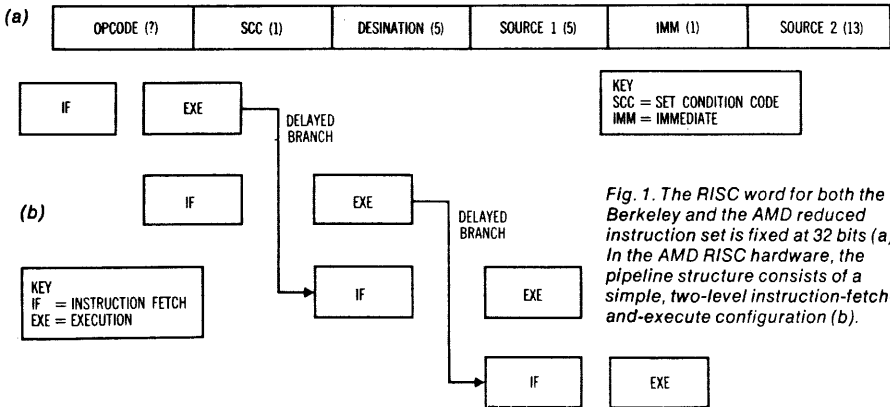
Dhaval Ajmera and Cheng-Gang Kong
Production Planning and Development Engineers
Microprogrammable Processes
Advanced Micro Devices, Inc., Sunnyvale, CA

Central processing units with reduced instruction sets fall into two categories. Single-chip versions are champion performers, but their fixed instruction sets mean that software compatibility can be a problem. Others are built from an army of discrete components and small-, medium-, and large-scale ICs (SSI, MSI, and LSI) and so suffer from high chip counts, long interchip delays, and great power dissipation.

A good compromise between the two is a team of a few very large-

scale IC (VLSI) parts—namely, the bipolar Am29300 and CMOS Am-29C300 families of VLSI building blocks (see box, "VLSI RISC"). By using these families, it is possible to adapt an operating system and instruction set to a reduced-instruction-set computer (RISC) architecture while maintaining software compatibility.

As a family, the 29300 can support the extremely fast cycle time of 80 ns, and both it and the 29C300 group have a 32-bit fixed word length. That



Reprinted with permission from Electronic Products, Vol. 29 No. 12, November 17, 1986. Copyright 1986, Hearst Business Communications Inc.

word length affords high precision for arithmetic operations as well as a wide bandwidth for memory and a large (4-gigabyte) addressing capability for virtual-memory operations.

Each family member fulfills a distinct function, allowing the RISC designer considerable freedom to configure them in a variety of architectures. Because, for example, the Am29334 register file building block is functionally separate from the Am29332 arithmetic logic unit (ALU), several Am29334 can be used to vary the size of the register file as required. In addition, data from the registers can be shared by other parallel devices besides the ALU.

The high level of integration of the 29300 and 29C300 family members favors higher performance because interchip delays are shorter. Also, systems need fewer and smaller boards to mount a lower parts count, and less power is dissipated—both factors that tend to reduce costs.

The AMD RISC architecture closely resembles the RISC I devel-

oped at the University of California at Berkeley, which has 33 instructions. Basic to both architectures is a fixed instruction format. Every instruction word is 32 bits wide (see Fig. 1a). Its op code occupies a field of 7 bits. Three fields totaling 23 more bits specify two source operands and a single destination. These fields are always in the same position in the instruction word—an arrangement that makes it

VLSI RISC

needed to build the RISC described in the accompanying article.

\$\$\$\$\$
The Am29332 ALU is housed in a 168-pin grid array and sells for \$495 each in 100-unit quantities. The Am29334 four-port, dual-access register file is packaged in the 120-pin grid array and sells for \$180 each in 100-unit quantities. The Am29337 bounds checker comes in 28-pin ceramic DIP and is priced at \$22 in 100-unit quantities. Other building blocks in the Am29300 family are available.

relatively simple to decode the op code in parallel with the operand access.

A two-level pipeline

The pipeline of the AMD RISC is a simple, two-level structure. One level fetches an instruction while the other is executing the instruction fetched immediately beforehand (see Fig. 1b).

This concurrency, however, creates difficulties with branch instructions. A conditional branch instruction cannot make its condition available until it has been executed. Therefore, the instruction fetched during its execution might not be the correct one.

To circumvent this pipeline lock-step dependency, a method called delayed branch is used. A code reorganizer (a program) rearranges the sequence of instructions so that the one immediately following the branch instruction is always executed despite the branching condition (see Fig. 1b again). In 9 out of 10 cases, a useful operation can be inserted. The rest of the time a NOP fills in. In other words, whatever the result of the branch instruction, it is executed only after an intervening

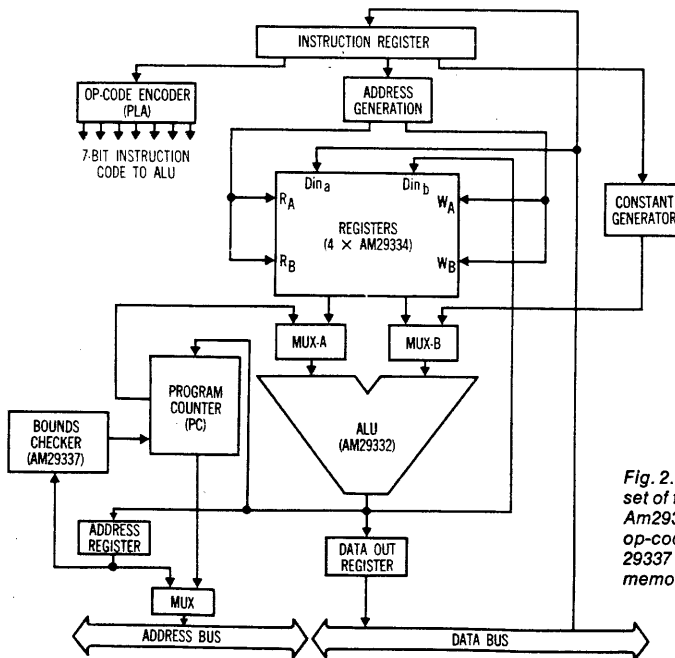
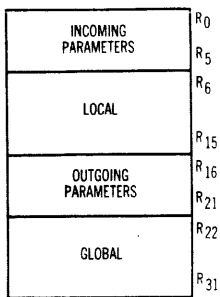


Fig. 2. The AMD RISC system includes a set of four Am29334 registers and an Am29332 ALU, which derives its 7-bit op-code controls from a PLA. The Am-29337 bounds checker identifies all memory references to the file registers.

instruction has been dealt with.

Exceptions are another pipeline hazard. When one occurs, the pipeline contents are duplicated by three registers in the program counter unit. This unit is routed to the ALU through the A multiplexer (see Fig. 2)—a feature that allows the return address to be saved when a call instruction is executed. During exception handling, this path also makes it possible to save the contents of the three program counter registers and to use them to restart the processor.



(a)

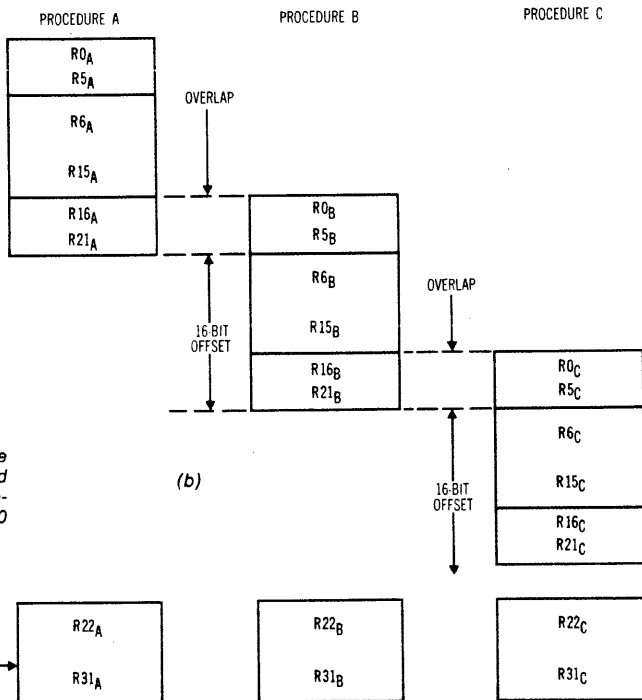
Fig. 3. The register window of the AMD RISC is functionally divided into four sections (a). Every procedure of the program shares the 10 global registers (b).

derived from the instruction's 7-bit op code through a programmable logic array (PLA). The Am29332 is a 32-bit-wide ALU that performs all arithmetic and Boolean operations. A high data-transfer rate is provided by a powerful, orthogonal instruction set. To enhance system performance, the device also features a 64-bit-in, 32-bit-out funnel shifter, as well as a 32-bit barrel shifter and a priority encoder.

in the execution of high-level languages.

Four Am29334s, with the aid of some SSI and MSI chips, provide seven register windows and 10 global registers. Altogether, they easily fit onto a standard hex card.

One register window is allocated to each procedure. Each window consists of 32 registers; thus at any time just 32 registers are visible to the currently executing procedure.



(b)

The instruction set enables constants to be formed through the instruction word directly. Before a constant can be fed into the ALU, however, some data has to be re-routed to generate it. This rerouting is done by the constant generator, which in essence uses 32 two-input multiplexers to produce the proper constant. The result is then fed via the B multiplexer to an ALU input.

The control section of the AMD RISC is relatively simple (see Fig. 2 again). All the control signals are

The Am29334 register file is a four-port, dual-access file that can be used to implement a distinctive feature of the Berkeley RISC—its so-called overlapped register windows. This overlapping improves the speed at which the procedures (or subroutines) in an application program can pass parameters among themselves and the main program in a call-return sequence. Berkeley researchers developed the technique after finding that parameter passing is one of the most time-consuming events

The 32 are functionally partitioned into four sections: 10 global and 10 local registers, as well as 6 apiece for incoming and outgoing parameters (see Fig. 3a). (In the Berkeley RISC, there are 138 registers grouped into 8 register windows.)

The 10 global registers (R₂₂ to R₃₁) are shared by every procedure of the program (see Fig. 3b). They are used primarily for globally referenced items such as a system's commonly applicable constants.

The 10 local registers (R₆ to

Integrated Circuits

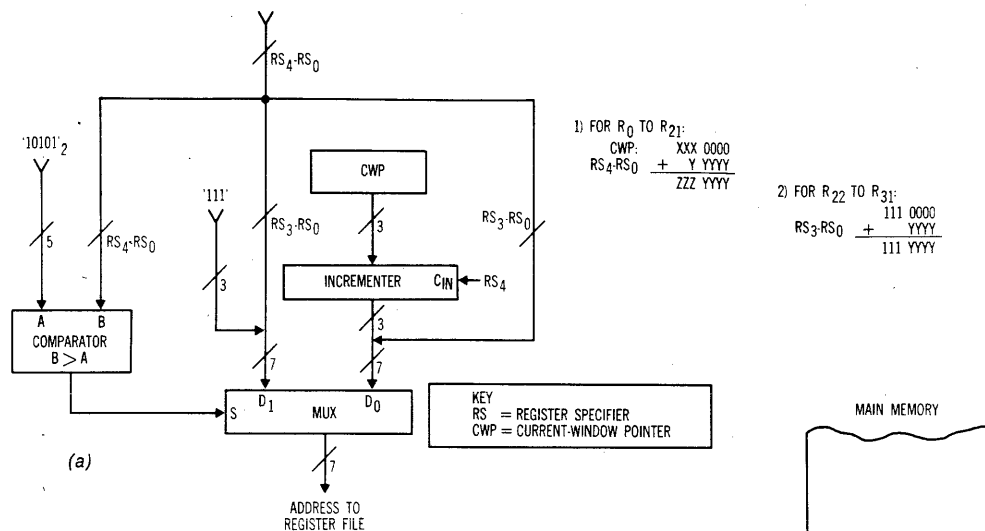


Fig. 4. The AMD and Berkeley RISC register numbering are 1's complements of each other (a). Also, either procedure can only be translated into the other if they are mapped one on one (b). Both the 1's complementing and the mapping are simple operations.

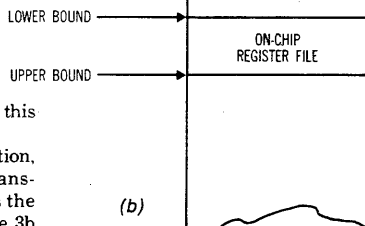
R_{15}), dedicated to the procedure itself, store local variables.

Six registers (R_0 to R_5) accept incoming parameters from the calling procedure for use by the called procedure. They are also used to return results from the called to the calling procedure.

When the called procedure in turn summons another, it puts its outgoing parameters in six registers (R_{16} to R_{21}) that then overlap the six in-

coming-parameter registers of this last procedure.

With such a register organization, parameters can be rapidly transferred between procedures, as the three register windows in Figure 3b illustrate. When procedure A calls procedure B, all the parameters pass through the outgoing-parameter registers of A to become the incoming-parameter registers of B, which can operate on these parameters without



accessing the stack memory. The same principle applies when B calls C. When C finishes, the results return through the outgoing parameters of B (or incoming of C). In turn, B also returns its results through the outgoing parameters of A.

The register numbering used in the AMD RISC for the windowing scheme is the 1's complement of its Berkeley RISC counterpart, a convention easily implemented with simple address-generation logic (see Fig. 4a). (A one-to-one mapping still remains between these two processors after this numbering change.)

The address generation logic maps any register number greater than 21 into the global register. The mapping is done by appending the lower 4 bits of the register specifier to three 1s. This operation maps it to a high address in the register file.

To generate the address of a local register, the pointer to the current

32-Bit Computer Performance Benchmarks

Benchmark	AMD RISC (ms)	Berkeley RISC 1 (ms)	Typical 32-bit superminicomputer (ms)
E-string search	0.115	0.46	0.59
F-bit test	0.015	0.06	0.29
H-linked list	0.025	0.10	0.12
K-bit matrix	0.108	0.43	1.29
I-quicksort	12.6	50.4	151.2
Ackerman (3.6)	800	3,200	5,120
Recursive Q sort	200	800	1,840
Puzzle (subscript)	1,175	4,700	9,400
Puzzle (pointer)	800	3,200	4,160
SED (batch editor)	1,275	5,100	5,610
Towers of Hanoi (18)	1,700	6,800	12,240
Average times faster	8	4	1

window (logically a 7-bit register) is added to the register specifier. The current-window pointer is the base pointer for the currently visible registers. It is advanced to the next window base pointer when a call instruction is executed; it is restored to the previous window base pointer when a return is executed. Since each register window is offset from the previous window by 16 registers (due to the overlap illustrated in Fig. 3b), the lower 4 bits of the current-window pointer are always zero. Therefore, an incrementer at the fifth bit position of this pointer can be used

to add in the register specifier. Thus connecting the fifth bit of the register specifier to the carry-in of the current-window pointer's incrementer generates the proper address for registers 0 to 21.

The comparator generates the proper select signal to gate the appropriate address (global or local) to the register file. With the projected 80 ns of the combined propagation delay of the Am29332 and Am29334, a 100-ns system cycle time can be easily obtained.

The register file, part of the system's run-time stack, is mapped into

the main memory (see Fig. 4b). The Am29337 bound-checking facility detects any memory reference to this section and reports it to the CPU. The CPU can then redirect the reference to the proper data store in the register file.

Performance evaluation

Usually it is hard to compare one architecture to another with any accuracy. The AMD RISC, though, is functionally compatible with Berkeley's RISC I, so that published parameters can serve as a basis for predicting their relative performance. The comparison is also predicated upon the following four assumptions:

mented by software subroutines.

- Use a fixed instruction format.

A fixed instruction format greatly simplifies instruction decoding and thus the hardware. Each field of the instruction word is dedicated to a particular function. For example, a fixed field is dedicated to the op code, and two or three fields are dedicated to operand specifiers. An added benefit is that an instruction with this format may allow some signals to be derived directly from it, permitting several operations to overlap.

- Employ a load/store architecture.

Memory references alone are done by load- or store-register operations. All the other operations are register-to-register. The simplicity of this addressing mode makes it easy to implement. The absence of complex addressing modes also makes it easier to restart instructions when an exception occurs.

- Support high-level languages.

The simple instruction set supplies the compiler with only the most primitive operations. From these the compiler can compose instruction sequences that are tailored to the exact requirements of the programming language. In some architectures, the hardware savings realized by the simple implementation is invested in speeding up some of the high-level language's more time-consuming operations. The University of California at Berkeley RISC processor, for instance, includes a large register file for speeding up the sequence of calling and returning from a procedure.

- A 100-ns cycle time. The Am29332 and Am29334 will contribute 80 ns to the total cycle time, and the register address generator and source multiplexer add another 20 ns (provided Schottky TTL components form the glue logic of the circuit).

- A 100-ns instruction cache. It has been established that an 8-Kbyte directly mapped instruction cache can provide a hit ratio of 99.8% on VAX-11 (programs written in C and running under Unix). High-speed RAMs (around 45 ns) are available from which a 100-ns instruction-cache memory with a good hit ratio can be easily constructed.

- The execution of the same instructions as RISC I. Register renaming of the code is easy.

- No adverse impact on performance due to the AMD's RISC having one fewer register window (Berkeley's RISC I has eight register windows versus seven for AMD).

For a simulated RISC I running 11 benchmark programs written in C, the system cycle time was 400 ns. For the AMD system running the same programs, it was 100 ns, or four times shorter. Further, as the table indicates, the AMD implementation averages about eight times faster than a typical 32-bit superminicomputer. □

RISC's minimalist philosophy

A new style of computer architecture has stirred a lot of attention recently. It's called RISC, for reduced instruction-set computer. Examples of it are the University of California at Berkeley's RISC I and RISC II, IBM's 801 project, and Stanford University's MIPS (for microprocessor without interlocked pipe stages).

The time-honored route in system design has been to leverage on progress in IC technology by increasing the complexity of computer architecture, with the goal of narrowing the "semantic gap" between the high-level languages of programming and the bit languages of machines. Complex instruction-set computers, or CISCs, are one result. But the side effects are unpleasant—longer design times, more numerous design errors, and inconsistent implementations.

This outcome triggered an about-turn in favor of simplicity. RISC designers try to select only the most frequently used, primitive instructions and to execute them very fast. Some of the main architectural design principles of the RISC are:

- Execute one instruction per cycle.

Program traces show that the most heavily used instructions are quite primitive. They also execute in one cycle. Hardwiring instead of microprogramming them enhances overall performance by eliminating the overhead incurred in microcode interpretation. The lengthy, highly complex, and infrequently summoned instructions provided by the CISC but omitted on the RISC can be imple-

FAULT-TOLERANT CHIPS INCREASE SYSTEM RELIABILITY

Using parity checking and a master/slave duplication technique, a bipolar chip set provides an interlocking fault-detection scheme that enhances fault tolerance.

by Tim Olson

Fault-tolerant computers have been used in satellites, aircraft, and industrial control and communications applications. The use of fault-tolerant techniques is currently being extended into other arenas, including on-line transaction processing and increasingly complex very large-scale integration circuitry. In addition, the rising cost of system maintenance and repair is causing a demand for fault-tolerant system building blocks that enhance system availability and reliability.

The Advanced Micro Devices 32-bit, microprogrammable chip set addresses these needs. The Am29300 family, which consists of the Am29332 arithmetic logic unit (ALU), Am29331 sequencer, Am29334 register file, Am29325 floating-point processor and Am29323 multiplier, uses an interlocking fault-detection scheme to provide fault tolerance. This detection scheme consists of a parity-check system and a master/slave duplication technique.

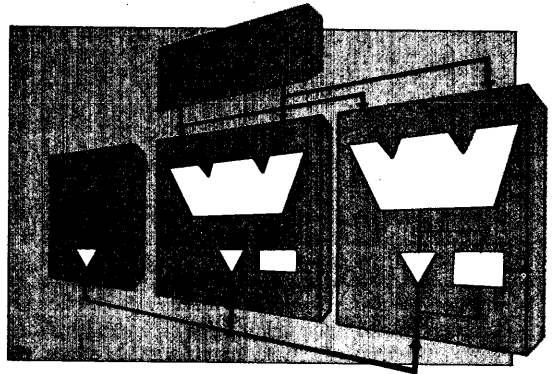
Add a bit

Parity-check codes are a form of error detection in which a single parity bit is appended to a group of data bits. The addition of this single bit changes the number of zeros and ones within the bit group. If, with the addition of the parity bit, the group has an even number of ones, the group has even parity;

Tim Olson is a product engineer for Advanced Micro Devices (Sunnyvale, CA). He holds an MS in electrical engineering from the University of Arizona.

Order # 08087A

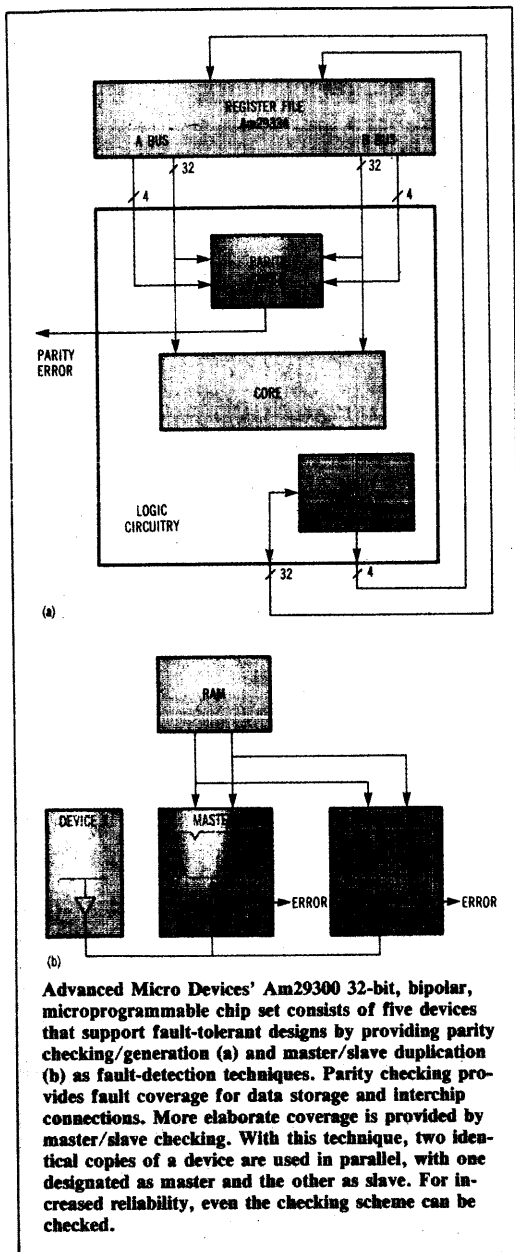
Reprinted with permission from Computer Design.



if it has an odd number of ones, the group has odd parity. Parity-check codes can detect all single-bit errors, as well as errors that involve an odd number of bits. For groups with an odd number of bits, even parity can detect the all-ones condition and odd parity, the all-zeros condition.

To detect data-transmission errors, the Am29300 family checks parity according to bytes. In this scheme, a parity bit is appended to each byte in the 32-bit word, resulting in four 9-bit groups. Each group contains a single parity bit. There are three reasons for using byte parity: fault coverage, decreased cycle time and byte-write capability. Fault coverage is increased by providing a single parity bit per byte. This technique catches many faults that would go undetected if a single parity bit per word were used.

Decreased cycle time refers to the fact that four parity bytes operating in parallel can generate and perform a parity check faster than a single 32-bit



parity-generation system. Byte-write capability provides other advantages. In byte parity, individual bytes can be written back into the register file without reading the rest of the 32-bit word to compute parity.

The Am29300 family uses even parity, which extends fault coverage to include a floating input bus. This parity scheme includes an all-ones failure mode, which occurs if a failure in the source device prevents it from driving the bus or if a failure in the control path prevents the source device from being accessed. Parity bits are stored in the register file, checked when input to the ALU and multiplier, and then generated as an output. If a parity error is detected on either of the two input buses, the Parity-Error output is asserted. This output is active high to provide fault detection for the error signals.

This parity scheme provides fault detection on both the data storage and the interchip connections. Since the Am29332 ALU and the Am29323 multiplier perform operations on data that cannot carry parity bits, however, a more elaborate checking scheme is used. This system is called master/slave checking.

More than one copy

Master/slave checking uses duplication as a fault-detection technique. Two identical copies of a device are used in parallel; one is designated as master, the other as slave. The master device computes a result from the inputs and moves its result to the chip outputs. The slave device also computes a result from the inputs, but all of its outputs (except for MS-Error) are changed to inputs that carry the results of the master.

The slave compares its result with the result of the master and signals any discrepancy on the MS-Error output. This output, like Parity-Error, is active high to provide fault detection for the error signal. Master/slave checking can detect multiple failures in both the master and the slave devices, as long as at least one failure is nonoverlapping. This checking system also detects output bus contention, which is indicated by the MS-Error output on the master device. This output is activated when the master result and the output bus fail to match.

For systems that must operate nonstop, master/slave techniques may also be applied at the board level. Two sets of master/slave pairs are used; one is active and the other is standby. If the slave of the active pair signals an MS-Error, the active pair is turned off and the standby pair is activated. The standby pair may also perform transactions while the active pair is running, resulting in twice the throughput of normal operation.

The ALU, multiplier and sequencer all have a master/slave operation mode. This mode, combined with parity checking of the data paths, provides complete interlocking fault detection on a cycle-by-cycle basis.

The fault recovery process can identify two types of faults: permanent or transient. Permanent, or hard faults, are caused by physical changes in the hardware (failures), while transient, or soft faults, are due to unstable hardware or temporary environmental conditions. Detection of a permanent fault may cause a standby unit to take over for the failed device.

When transient faults are detected, on the other hand, the microinstruction that faulted will be restarted after the transient condition disappears. In either case, the faulted microinstruction must be aborted, so that no state change occurs to disrupt the restarting of the microinstruction.

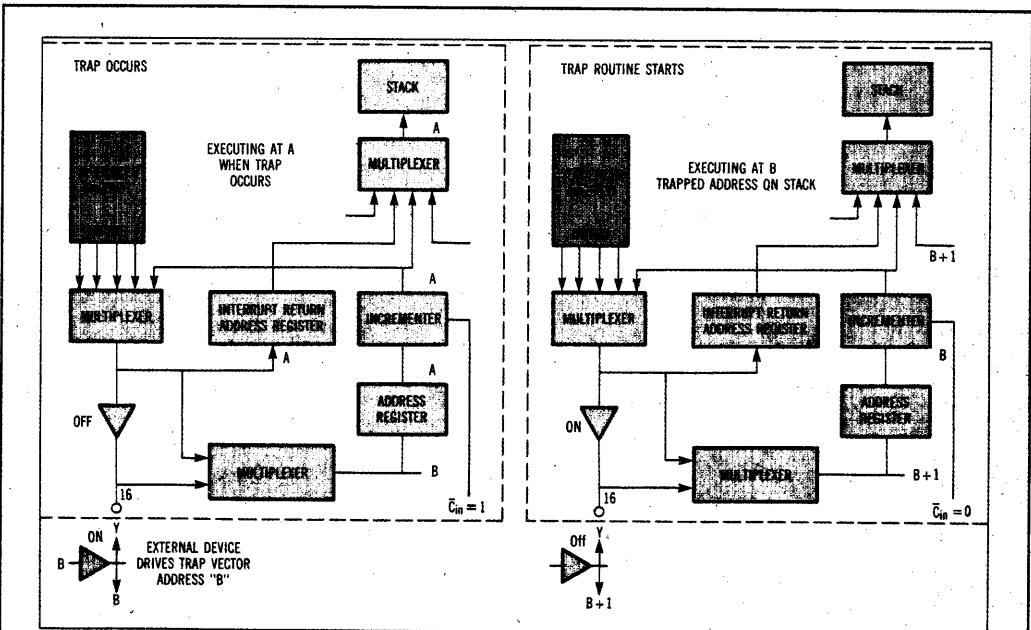
To restart the microinstruction, the sequencer performs traps at any microinstruction boundary. When a trap condition is signaled by the simultaneous assertion of the interrupt request and force continue signals with the Carry input (C_{in}) signal disabled, the address incrementer to pass the current address instead of the next address, the sequencer puts the Y output bus in a high-impedance state. This allows an external trap vector to be placed on it. The sequencer then pushes the trapped microinstruction

address onto the internal stack and starts fetching microinstructions, using the trap vector as the starting address. The aborted microinstruction is stored on top of the stack and is restarted by executing a return instruction. When the Hold input is asserted, updates of the ALU's internal state are inhibited. This ensures that the aborted microinstruction has no effect.

Fault-tolerant CPU design

In order to show how the Am29300 family members interact to perform fault detection, recovery and isolation, consider a simple CPU design. In this design, the data path consists of two sets of register files and two ALUs in a master/slave configuration. Because new data may already have been written to the register file before a fault is signaled, two register file sets are required. One register file set holds the working address and data registers, while the other set holds backup copies of these registers that are used in error recovery.

The ALUs perform address and data calculations, which are used to address memory via the data-out, data-in and address registers. These registers are built



Errors are signaled with Parity-Error and MS-Error outputs and prioritized by an interrupt controller. The sequencers then trap the microinstruction that is being executed. To restart a microinstruction when a failure occurs, the sequencer can perform traps at any microinstruction boundary. When a trap condition is signaled, the sequencer changes the Y output bus to a high-impedance state, allowing an external trap to be placed on it.

from Am29818 diagnostics registers that offer off-line testing and fault diagnosis. The control path starts with the instruction register, which consists of four serial shadow registers. The instruction is applied to a mapping PROM to derive the starting microcode address for the sequencer, which is built from two Am29331 sequencers in a master/slave configuration. The microinstruction is fetched from the writable control store and loaded into pipeline registers, which distribute control throughout the CPU.

Fault detection, recovery and isolation

During instruction execution, errors are detected on a cycle-by-cycle basis by the sequencer and ALU master/slave pairs. They are signaled with the Parity-Error and MS-Error outputs. These error signals are prioritized by a vectored priority-interrupt controller, which causes the sequencers to trap the microinstruction that is currently executing. The trap vector is then put on the Y output bus. The controller also asserts the Hold pin on the ALUs, which prevents the trapped microinstruction from updating the internal state of the ALU and disables writes to the

backup register file. Writes to the backup register file are disabled, keeping the state of the ALU prior to the trapped microinstruction intact.

Microinstruction processing then begins with the trap routine associated with the highest priority fault indication. This routine can determine whether the fault is transient or permanent. If the fault is transient, the trapped microinstruction must be restarted. The trap handler first restores the state of the register file by copying each of the registers in the backup register file into the working register file, restoring the registers to the values they held prior to the fault. Any other state that was saved during trap processing is also restored during this process. The sequencers then perform a return instruction, popping the trapped microinstruction address from the stack.

To increase system availability, permanent faults must be isolated quickly. This usually involves running a series of test patterns through the devices to determine which ones have failed. These patterns can be loaded and tested quickly using the serial shadow registers. All of the serial shadow registers in the CPU design are connected by a serial link that forms

a diagnostics loop. Arbitrary patterns can be loaded serially through the loop, then clocked through in a single system cycle. The resulting state can be read out from the loop for use in isolating the failed device.

Checking the checkers

Failures in checking devices are even more serious. A failed checker can give a false indication of error or a no-error condition. While false indications of failure are tolerable, a no-error condition often results in undetected faults.

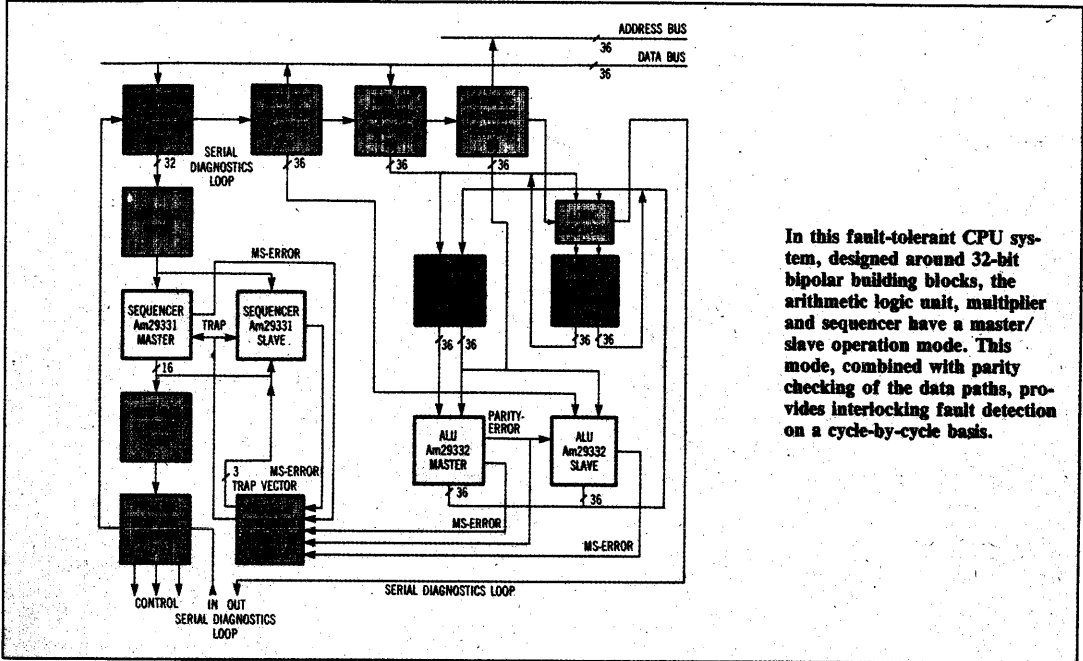
There are three basic fault detectors in the CPU design: the Am29332 parity checker, the Am29332 master/slave checker and the Am29331 master/slave checker. These fault-detection circuits must be verified during system initialization, and their operational status should be confirmed periodically during subsequent operation.

Fault injection, which is the process of deliberately causing a fault in the part of the system that is checked by the fault-detection hardware, can be used to perform this verification. The parity-check

circuitry can be tested by loading a word with bad parity into the data-in register via the serial link. It is then loaded into the register file and used in an ALU operation. This procedure should detect a parity error.

Another method of verifying the parity checker is to issue a microcode instruction that performs an ALU operation while the register-file outputs are in a high-impedance state. The parity checker should detect the all-ones condition and flag the error.

Master/slave checking can be verified on the ALU by using the Hold input. The status registers in the master and slave are first set to a known equivalent state. The next microinstruction alters that state, but asserts the Hold input on one of the devices, inhibiting the status update. A master/slave error, caused by the differing status outputs, should occur. Master/slave checking can also be verified on the Am29331 sequencers by executing a jump instruction while asserting the force-continue input on one of the parts. The part without the asserted force-continue input executes the jump, causing a nonsequential address for the next microinstruction. The



In this fault-tolerant CPU system, designed around 32-bit bipolar building blocks, the arithmetic logic unit, multiplier and sequencer have a master/slave operation mode. This mode, combined with parity checking of the data paths, provides interlocking fault detection on a cycle-by-cycle basis.

force continue asserted on the other sequencer overrides the jump instruction, causing the next microinstruction address to be sequential. This results in differing addresses which, in turn, causes a master/slave error.

The AMD family extends many of the concepts of fault-tolerant computing, including parity checking and master/slave duplication into the 32-bit arena. This fault-detection scheme can identify both permanent and transient faults, ensuring broad-based fault protection throughout the system. **CD**

Floating-point math handles iterative and recursive algorithms

Floating-point arithmetic gives you better dynamic range and precision than integer arithmetic, but it needs careful implementation. Part 1 of this 3-part series discusses possible sources of error you may encounter when using floating-point hardware, and it reviews the current standards. Part 2 will describe the advantages of fast array processors, and part 3 will discuss algorithmic options for floating-point processors and considerations when implementing a complete system.

Charlie Ashton, *Advanced Micro Devices Inc*

Many signal-processing algorithms, such as fast Fourier transforms, generate outputs whose magnitudes far exceed those of the inputs. Nevertheless, those outputs must retain the precision of the input operands if the accuracy of the computation is not to be so severely degraded as to render the results meaningless. For these and similar applications that use iterative or recursive algorithms, true floating-point operation often furnishes the only acceptable number representation.

Until recently, you needed a very good reason to give

your system floating-point hardware. It was large, expensive, power-hungry, and relatively slow (although faster than the software-based implementations needed to perform comparable operations). However, the introduction of fast VLSI array processors has changed the picture. These devices (such as Weitek's 1032/1033 and AMD's Am29325) can stand alone and are implemented on one or two chips. You can now economically use floating-point hardware in applications whose size and budget constraints would previously have forced the use of fixed-point hardware or floating-point software.

The new chips won't dissipate all your potential headaches, of course. Just one of the many choices you'll have to make is which standard to support. The four most commonly used standards (IEEE, DEC, IBM, and MIL-STD-1750A) have subtly different binary representations of floating-point numbers. Each standard has advantages and disadvantages for specific types of computational problems. This series of articles covers some of the theoretical considerations you'll have to take into account, as well as some specifics on the available chips.

The manner in which a system represents floating-point numbers clearly affects both the dynamic range and the precision of the system. The most obvious way

Reprinted with permission from EDN, January 9, 1986. Copyright 1986, Reed Publishing USA.

VLSI processors now make floating-point hardware cost effective in applications with severe budget or size constraints.

to represent numbers is to use a signed exponent and a signed fraction (Table 1). A large exponent field obviously supports a large dynamic range: A 2-digit exponent, for example, implies a dynamic range of 10^{100} , whereas a 3-digit exponent increases the dynamic range to 10^{1000} . Similarly, the more digits you can include in the fraction, the greater will be the precision of the number, especially if the number is normalized so that the left-most digit of the fraction is nonzero. Leading zeros in the fraction of an unnormalized number clearly reduce the precision of that number. As a general principle, then, the precision of a floating-point

fact, a 32-bit floating-point number in IEEE format has a dynamic range equivalent to that of a 276-bit 2's-complement integer.

Despite the high precision and large dynamic range of normalized floating-point numbers, floating-point systems do not altogether escape the effect of quantization (rounding) errors. You can think of a floating-point system as producing an infinitely precise result (ie, a fraction of unlimited length, abbreviated "IPR"), which is then rounded to fit into the destination format. Typically, this strategy means that some of the low-order fraction bits are lost. Consequently, whenever the destination format lacks enough bits to accommodate the IPR, rounding introduces quantization errors, which in turn result in system noise. Consider, for example, the multiplication of two numbers in a 4-digit decimal system:

$$(0.8102 \times 10^3) \times (0.8001 \times 10^{-7}) = 0.6410401 \times 10^{-4}$$

The IPR is rounded to 0.6410×10^{-4} to fit the destination format, thus introducing a quantization error. In practice, quantization errors during a long computation will be random, and the overall effect will be analogous to an increase in system white noise. If the quantization errors are *not* random, they may appear as system nonlinearities and, as a consequence, cause serious problems in such applications as spectral analysis.

TABLE 1—SIGNED vs BIASED EXPONENTS

DECIMAL NUMBER		SIGNED EXPONENT		FRACTION
-123.45	=	10^{-3}	x	-0.12345
+0.0000678	=	10^{-4}	x	0.678

DECIMAL NUMBER		BIASED EXPONENT		FRACTION
-123.45	=	5+3=8	x	0.12345
+0.0000678	=	5-4=1	x	0.678

number depends on the length of its fraction, and the dynamic range depends on the size of the exponent and the radix.

In practice, floating-point hardware generally uses a biased exponent for two reasons. First, use of a biased exponent avoids problems that follow from the need to handle negative numbers in the exponent circuitry. Second (and perhaps more important), a suitable choice of bias can ensure that you'll be able to compute the reciprocals of all the representable numbers without exponential overflow or underflow. You'll find that overflow and underflow cause plenty of problems in computing the fraction portion of the output (see box, "Dealing with underflow and overflow"). You certainly don't want to introduce them into exponential computations as well.

Biased exponents and normalized fractions are the features that give true floating-point representation a clear advantage over block floating-point and integer formats. To double the dynamic range of an integer word, you have to double the number of bits in it. To obtain the same result in true floating-point operation, you need to add only one bit to the exponential field. In

Are quantization errors data dependent?

Mathematical analysis of an integer system shows that quantization errors due to rounding have a mean value of one-quarter the value of the least significant bit. The relative error at each rounding thus depends on the magnitude of the operand being rounded. Therefore, as the magnitude of the operand decreases, the relative quantization error increases. The same is true of a block floating-point system, in which denormalized operands may contain leading zeros. In integer and block-floating-point systems, therefore, the errors are data-dependent, and for this reason error analysis is both difficult and time-consuming.

In true floating-point systems, however, operands are generally normalized, so the relative quantization errors are the same, regardless of the magnitude of the operands. Quantization error analysis in floating-point systems is thus data independent and therefore doesn't require complicated worst-case simulations.

Floating-point systems can suffer from a computational drawback known as the "operand ordering prob-

lem." Consider the addition of three floating-point numbers: $A (=1)$, $B (=2^{90})$, and $C (= -2^{90})$. You may find that $(A+B)+C=0$, although $A+(B+C)=1$. This result clearly violates the associative law of addition. The discrepancy occurs because the floating-point standard doesn't have enough bits to accommodate the intermediate result of the first calculation $(A+B)$. The hardware has to round the IPR, $2^{90}+1$, to the nearest representable number, which is 2^{90} . Errors of this kind are inevitable whenever the IPR has to be rounded to fit the destination format, although they would usually be considered so small as to be unimportant.

You can minimize rounding errors (although, as the previous example shows, you can't entirely remove them) by a judicious choice of rounding mode. Some floating-point standards allow you to select from among several rounding modes the one that best suits your operation. All of the commonly used floating-point standards support one or more of four modes:

- Round-to-nearest mode replaces the IPR with the closest representation that fits in the destination format. In the case of an IPR that falls exactly halfway between two representations, the IEEE standard rounds the IPR to the representation

Dealing with underflow and overflow

For the rare cases in which the result of a calculation is too large or too small to be represented, you must have previously specified the way in which your system will deal with that result. In short, your system must handle the related problems of underflow and overflow.

Underflow arises when the rounded result of an operation is a number between zero and the smallest representable normalized number. You can handle such a number in one of two ways: You can set the number to zero (sudden underflow), or you can represent the rounded result by a denormalized number (gradual underflow).

Overflow occurs when the rounded result of an operation is greater than the largest representable number. You can handle this problem by setting the result to infinity, which implicitly terminates a chain of calculations, or by saturating the result to the largest representable number (correctly signed).

It's important to know which of the various methods your system supports, because in some

applications sudden underflow or saturated overflow can destroy the accuracy of an entire series of calculations. The IEEE standard, for example, treats underflows by invoking the gradual underflow method, while the IBM and DEC standards deal with only sudden underflow.

Sudden underflow is generally the fastest method of treating underflows and is acceptable in the majority of systems because high accuracy is seldom required for very small numbers. Sudden underflow can produce quantization errors almost as large as the smallest normalized number, but usually you can treat these errors as insignificant.

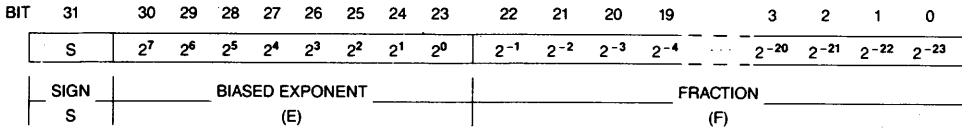
The gradual-underflow method creates much smaller errors because it rounds results to a normalized number. On the other hand, gradual underflow is more difficult and more expensive to implement than sudden underflow, a drawback you'll have to weigh against the advantage of accurate results over a wider range of numbers. Gradual underflow is generally best for iterative applications in which

you drive a residual value to zero and for which you require maximum possible accuracy. When such a residual value underflows gradually to zero, you know that it's negligible compared with every normalized number.

For handling overflow, data-processing applications generally set the result to infinity, because in a high-accuracy mathematical model a saturated result could destroy the accuracy of an entire series of calculations. In real-time digital signal processing, however, it's generally preferable to saturate the result and continue the chain of calculations. In the analysis of radar returns, for example, you would certainly not want a single anomalous return to bring the entire processing sequence to a halt by introducing an operand (an infinity) that would be useless in further processing. In this and similar applications, it's often better to have an approximately correct data point than no data point at all.

**TABLE 2—NUMBER REPRESENTATION
IN FOUR FLOATING-POINT STANDARDS**

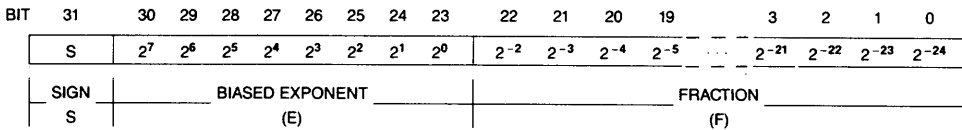
IEEE FORMAT



$E = 0 \text{ AND } F = 0 \dots\dots\dots V = (-1)^S \cdot 0 (-0, +0)$
 $E = 0 \text{ AND } F \neq 0 \dots\dots\dots V = (-1)^S \cdot 0.F \cdot 2^{-126} \text{ (DENORMALIZED)}$
 $0 < E < 255 \dots\dots\dots V = (-1)^S \cdot 1.F \cdot 2^{E-127} \text{ (NORMALIZED)}$
 $E = 255 \text{ AND } F = 0 \dots\dots\dots V = (-1)^S \cdot 00 (-00, +00)$
 $E = 255 \text{ AND } F \neq 0 \dots\dots\dots V = \text{NaN (NOT-A-NUMBER)}$

(a)

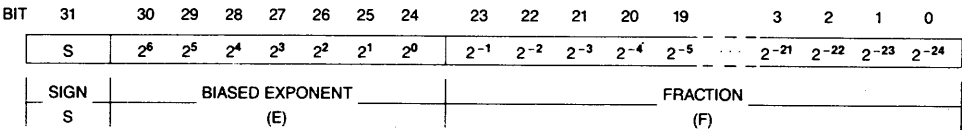
DEC FORMAT



$S = 1 \text{ AND } E = 0 \dots\dots\dots V = \text{DEC RESERVED OPERAND}$
 $S = 0 \text{ AND } E = 0 \dots\dots\dots V = 0$
 $E > 0 \dots\dots\dots V = (-1)^S \cdot 0.1F \cdot 2^{E-128} \text{ (NORMALIZED)}$

(b)

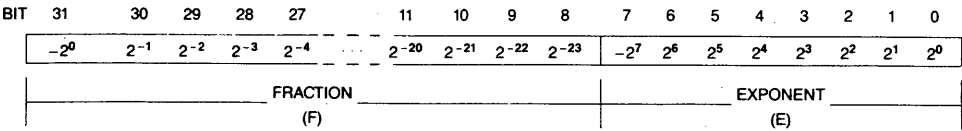
IBM FORMAT



$F = 0 \dots\dots\dots V = (-1)^S \cdot 0 (-0, +0)$
 $F \neq 0 \dots\dots\dots V = (-1)^S \cdot 0.F \cdot 16^{E-64}$

(c)

MIL-STD-1750A FORMAT



$V = F \cdot 2^E$

(d)

Biased exponents and normalized fractions give true floating-point systems a clear advantage over integer and block-floating-point systems.

having an LSB of zero, whereas the DEC standard rounds the IPR to the representation that has the greater magnitude.

- Round-to-minus-infinity mode rounds the IPR to the closest representable value that is less than or equal to the IPR.
- Round-to-plus-infinity mode rounds the IPR to the closest representable value that is greater than or equal to the IPR.
- Round-to-zero mode is analogous to truncation; it rounds the IPR to the closest representable value with a magnitude less than or equal to that of the IPR.

As noted earlier, the various floating-point standards specify different binary representations of floating-point numbers, and you'll have to match their respective advantages and disadvantages to your own computational problems. The four of the most common binary floating-point standards, the IEEE, DEC, IBM, and MIL-STD-1750A standards, all represent single-precision, floating-point numbers by means of 32-bit words having the formats shown in Table 2. All four standards support double-precision data, and some of these standards also support other data types, such as single-extended and double-extended data.

The IEEE working group presented the specifications contained in proposed standard P754, draft 10.1, as a robust standard for portable floating-point software. This proposed standard has received wide acceptance, and it's likely to form the basis of a large number of future hardware implementations. P754 has

several features that aren't found in other standards. In particular, +0, -0, and infinities are all valid operands. Operations performed on infinities signal no exceptions unless the operation itself is invalid. The standard allows the use of a special operand known as NaN (Not-a-Number). An implementation should interpret NaNs as signals rather than numbers, and it should use NaNs to indicate invalid operations or to pass status information through a series of calculations. Also, the standard accepts denormalized numbers as a representation of a result that is less than the smallest normalized number.

The DEC standard is implemented in all DEC VAX minicomputers; the VAX Architecture Manual contains the full specifications of the standard. Conceptually simpler than the IEEE standard, the DEC standard has no provisions for infinities or denormalized numbers, and it has only a single representation for zero. The DEC standard does, however, incorporate DEC reserved operands, which are analogous to IEEE NaNs.

An important feature common to both the IEEE and the DEC standards is the existence of a hidden bit. Both standards specify that all operands will be normalized (except for denormalized numbers in the IEEE format). This stricture implies that the leading fraction bit must always be a one. This bit would not only be redundant if included in the 32-bit representation, but it would actually reduce the precision of the number, so its presence is assumed. In the case of IEEE denormalized numbers, the biased exponent is zero, thereby

continued, page 6-108

TABLE 3—COMPARISON OF FLOATING-POINT STANDARDS

	IEEE	DEC	IBM	1750A
LARGEST POSITIVE NUMBER	$2^{128} - 2^{104}$	$2^{127} - 2^{103}$	$2^{253} - 2^{228}$	$2^{127} - 2^{103}$
SMALLEST POSITIVE NUMBER	2^{-149}	2^{-128}	2^{-280}	2^{-129}
LARGEST NEGATIVE NUMBER	$-2^{128} + 2^{104}$	$-2^{127} + 2^{103}$	$-2^{253} + 2^{228}$	-2^{127}
SMALLEST NEGATIVE NUMBER	-2^{-149}	-2^{-128}	-2^{-280}	-2^{-129}
DYNAMIC RANGE	2^{277}	2^{256}	2^{533}	2^{256}
PRECISION	2^{-23}	2^{-23}	2^{-20}	2^{-23}

VLSI floating-point μ P for recursive algorithms

One example of floating-point hardware that handles recursive algorithms is the Am29325 from Advanced Micro Devices. The processor integrates a 32-bit adder/subtractor, a multiplier, and a data path on a single chip. This level of integration reduces the processing overhead incurred by chip sets comprising separate ALU and multiplier chips. The internal feedback paths facilitate the implementation of such recursive algorithms as sum-of-products and Newton-Raphson division.

The processor supports both the IEEE and DEC floating-point formats. The instruction set includes instructions that convert data from IEEE format to DEC format and vice versa, as well as instructions that convert data to and from 32-bit integer format.

Three functional blocks

The processor has three main functional blocks (Fig A): a floating-point ALU, a status-flag generator, and a 32-bit internal data path. The ALU is fully combinatorial, and it performs all instructions in a single cycle. The eight instructions handle floating-point $R+S$, $R-S$, $R \times S$, and $2-S$ operations as well as the format conversions.

The $2-S$ instruction forms the core of the Newton-Raphson division algorithm, which performs division by a sequence of iterations. In this and other iterative algorithms, intermediate results are retained in the R or S register, thereby eliminating the need for any off-chip registers and minimizing the number of required data transfers.

Three programmable I/O modes allow the Am29325 to interface with a variety of systems. The 32-bit, 2-input-bus mode uses three separate 32-bit

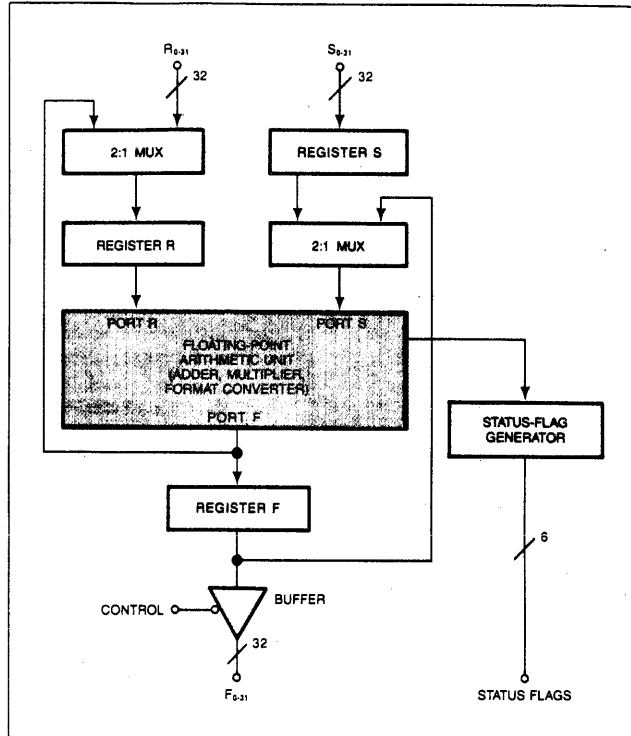


Fig A—This VLSI floating-point processor is fast because it contains all the major components for 32-bit operations on a single chip. It has one input for an external clock and 17 inputs for instruction-select and control functions.

buses (R, S, and F) for high-speed, nonmultiplexed operation; in this case, the R and S registers are configured as independent 32-bit ports. In the 32-bit, 1-input-bus mode, both the R and S registers are connected to a common 32-bit input bus; the host multiplexes operands onto this bus. In the 16-bit, 2-input-bus mode, 32-bit operands are multiplexed onto the corresponding 16-bit buses (low-order bits first).

Six flags and four modes

The status-flag generator provides six fully decoded flags. Four of these flags report exceptional conditions, as defined in

the IEEE standard. The remaining two flags identify zero-valued or nonnumerical results.

The Am29325 implements the four IEEE-mandated rounding modes: round-to-nearest, round-to-plus-infinity, round-to-minus-infinity, and round-to-zero. The same four modes are supported for the DEC standard, except that when the infinitely precise result is halfway between two representable numbers, the IEEE round-to-nearest mode rounds to the closest representation with an LSB of zero, whereas the DEC round-to-nearest mode rounds to the value with the larger magnitude.

instructing the system to assume that the value of the hidden bit is also zero.

The IBM floating-point standard differs from its IEEE and DEC counterparts in several respects. It has no provision for infinities or reserved operands, although it does accept denormalized numbers. More important, however, are the absence of a hidden bit and the use of radix 16 rather than radix 2. Because the exponent of an IBM number is expressed as a power of 16, the standard has a large dynamic range. For the same reason, however, numbers are spaced farther apart than in the other formats. This increased granularity results in less precision than is provided by the IEEE and DEC formats. Also, the use of radix 16 allows as many as three leading zeros in the binary fraction of a normalized number, even though the leading hexadecimal digit is nonzero if the number is expressed in hexadecimal format. The leading binary zeros can cause the precision to vary from one operand to another. This variation is known as wobbling.

The MIL-STD-1750A standard, developed for use in military systems, allows no reserved operands, infinities, or denormalized numbers. Furthermore, the use of a 2's-complement fraction, rather than a sign-magnitude representation as in the other three formats,

requires a somewhat different hardware architecture.

The applications to which each of the four standards is best suited differ quite widely. Nevertheless, you can make a simple comparison (Table 3) between the standards, based on factors such as the largest and smallest representable numbers, the dynamic range, and the precision. Such a comparison can be useful in selecting the most suitable format for a given application. In most cases, however, the format to be used is determined by outside constraints, such as compatibility with existing hardware or software. **EDN**

Author's biography

Charlie Ashton is a senior engineer in the product-planning division of Advanced Micro Devices Inc (Sunnyvale, CA). His duties include defining array-processing products. Charlie holds a BSc degree from Reading University, UK, and he is a member of the British IEE. In his spare time, he enjoys cricket, hiking, and swimming.

**Designer's Guide to:
Floating-point processing—Part 2**

Floating-point array processor improves computational power

Powerful math-processing chips configured with high-speed memories and controllers form the core of a floating-point math or array processor for small computers. This second part of EDN's 3-part floating-point math series discusses the tradeoffs you must make to add flexibility and speed to array-processor designs.

Robert M Perlman, *Advanced Micro Devices*

For such jobs as digital-signal processing, image processing, graphics, and scientific calculations, an array processor can take over repetitive arithmetic chores while your host computer performs control tasks and retrieves information. By employing a floating-point array processor, you also increase the math-processing power of your computer system.

The basic array-processor design (Fig 1) contains an arithmetic unit, a controller, data memory, program memory, and a host interface (see box, "Array processor vs general-purpose computer"). If you use newer control, memory, and math chips, you can fit the circuit on a single pc board. This array-processor design uses an Am29325 floating-point processor chip, which oper-

ates with either IEEE- or DEC-standard single-precision data. The chip performs single-cycle floating-point additions, subtractions, multiplications, and format conversions at an 8-MHz clock frequency.

Because the Am29325 chip contains a floating-point arithmetic unit (AU), three 32-bit registers, two data buses, and two data-selection multiplexers, you need only a small amount of external hardware to design a complete math- or array-processor circuit. In the array-processor design, the Am29325 receives operands from two high-speed memories. An 8k×32-bit RAM provides input data for your algorithms, and it stores intermediate and final results. An 8k×32-bit PROM provides constant values for the algorithms.

Although you can design a circuit that specifically controls the math chip and its associated memory chips, you'll find an equivalent circuit in the 2910A microprogrammable controller chip. The 2910A chip is a general-purpose controller; it's not dedicated to controlling the Am29325. The controller chip contains a program counter, a loop counter, a LIFO stack, and other circuits that access program instructions and control the array processor in the basic design. The controller provides an 11-bit address for the design's 2k×64-bit microprogram memory, which contains the instructions for your algorithms. Each algorithm instruction con-

Reprinted with permission from EDN, January 23, 1986. Copyright 1986, Reed Publishing USA.

A basic array processor speeds math operations by performing repetitive tasks quickly.

tains 64 bits that the circuit divides into seven groups of outputs:

- 11 jump address bits
- one address and write-enable multiplexer bit
- one write-enable control bit
- 13 RAM-address bits
- 13 PROM-address bits
- 24 miscellaneous control bits
- one interrupt-control line.

The microprogram memory routes its outputs through an internal register and then to the rest of the array-processing hardware. Although it may not be obvious, the register at the microprogram memory's output helps maintain high-speed data processing. By using a clocked register to hold the memory's output bits, the controller latches a 64-bit instruction while it

addresses the microprogram memory for the next instruction. The memory's output register therefore permits the overlap of the instruction-fetch and -execute operations, which saves processing time.

Because it holds information for a pending operation, the microprogram memory's output register is often referred to as a pipeline register. Array processors can contain a series of pipeline registers, the number of which depends on the architecture of the array processor and the maximum processing speed you need.

Host interface links processors

You must carefully choose your host-computer interface circuits according to the type of system bus in your computer. You can accommodate most general-purpose computers by providing bus buffers for the address,

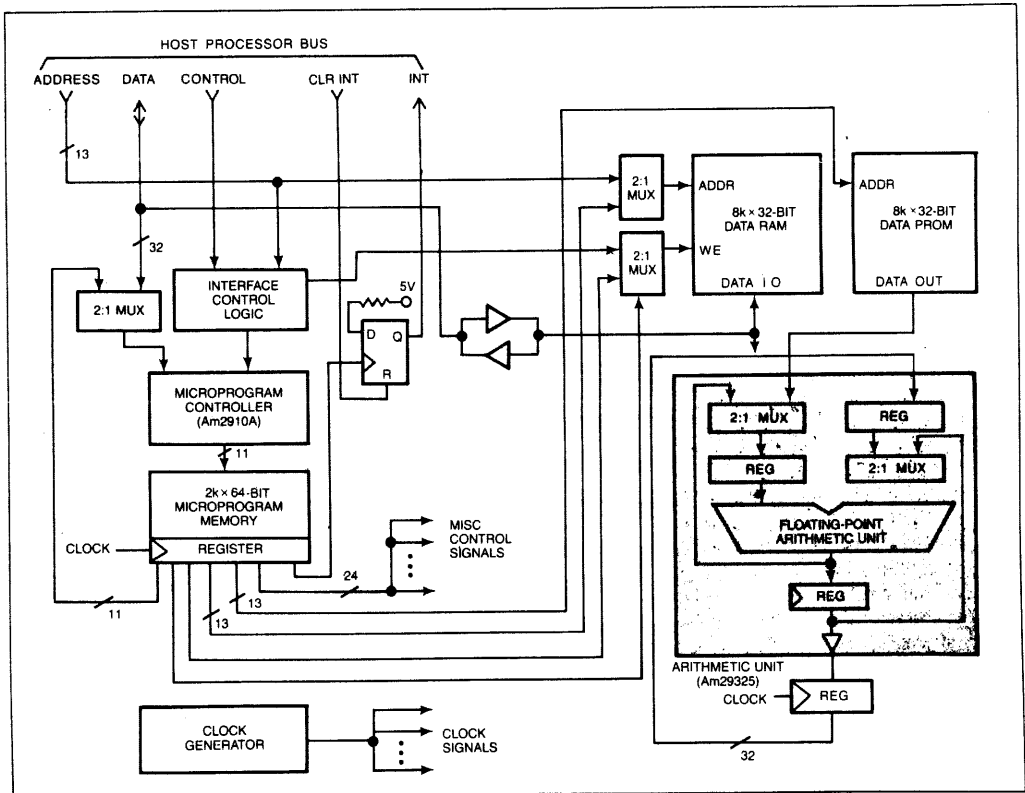


Fig 1—The Am29325 floating-point processor used in this design adheres to IEEE and DEC floating-point standards.

**TABLE 1—
BENCHMARK EXECUTION TIMES**

OPERATION	EXECUTION TIME
5-TAP FIR FILTER	1.125 μ SEC
RADIX-2 FFT BUTTERFLY	1.25 μ SEC
4x1 MATRIX ADDITION	1.0 μ SEC
4x4 MATRIX MULTIPLICATION	14.0 μ SEC

data, and control lines. You'll also need a small amount of control logic to manage the flow of information to and from the array processor and the host computer. For example, you can construct a Multibus interface by using octal bus buffers and PAL chips. If your host computer's data bus contains fewer than 32 data bits, you'll need to convert the data to and from the 32-bit format that the array processor requires. You can include double-buffer latch circuits for the data inputs to the array processor, and you can provide latches and multiplexers on the processor's data-output lines.

The host computer's data bus provides the main link between the host and the array processor. Your computer starts a math operation by loading the RAM with raw data and then signaling the array processor to start a math-processing algorithm. After the processor runs an algorithm program, your host computer reads the RAM's contents to obtain the results.

To simplify the data-transfer operations to and from the host computer, the array processor goes into an idle, or standby, state when it isn't running an algorithm program. Instead of controlling the processor's data and control lines, the microprogram controller continuously runs a 1-microinstruction program loop. In addition, the idle microinstruction switches the RAM's address and write-enable multiplexers so that the RAM appears to be part of the host computer's main memory. The host computer loads the desired input data into the data RAM, and it then loads the microprogram controller with the starting address of the algorithm you want to run. The microprogram controller then jumps to the preprogrammed sequence of microinstructions for the algorithm. The algorithm's first microinstruction reconfigures the data RAM so that only the array processor can address it. When the algorithm completes its tasks, it sends an interrupt signal to the host processor, switches the data RAM back to the host, and executes the 1-instruction standby loop.

Once you're sure the array processor is operating

properly, you can test the operating speed of your circuit by using benchmark programs tailored to specific tasks (Table 1). The benchmark times were calculated for the array processor with an 8-MHz clock frequency. The basic processor performs one data-RAM operation (read or write) per clock cycle.

Modifications improve performance

Although the basic array-processor circuit works well, you can improve its performance. The ability to take data addresses directly from the program memory in the simple array processor means that the program memory must contain a section of microcode for each iteration of an algorithm. For example, a program that performs 20 matrix multiplications contains a separate section of microprogram code for each multiplication

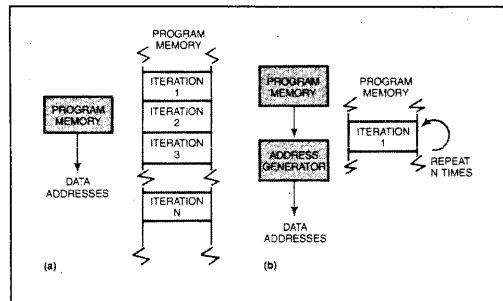


Fig 2—You can implement the program memory in two ways: Either you can include steps for each iteration of your algorithm (a), or you can add an address-generator circuit (b) that lets you use only one section of code for all iterations. The address generator locates specific values and coefficients in memory automatically.

step. Each code section contains specific addresses for data and coefficients (Fig 2a). The in-line coding approach therefore wastes program-memory space.

One improvement found in virtually every array processor is a data-address-generator circuit that generates the necessary data and coefficient addresses within the array processor. The address-generator hardware reduces the amount of microprogram memory you'll need for an algorithm. By using such hardware, the processor performs multiple iterations of an operation by looping through the same section of microcode as many times as necessary (Fig 2b).

Depending on your specific tasks, you can choose a data-address generator that fits a specific algorithm, such as the fast Fourier transform (FFT), or you can choose a general-purpose addressing device. Some

continued, page 6-114

Array processor vs general-purpose computer

To understand better what an array processor does, consider first the strengths and shortcomings of general-purpose computers. General-purpose computers incorporate the standard Von Neumann architecture and perform a variety of tasks. Such computers perform instruction-fetch and instruction-execution tasks sequentially, with instructions and data available in one memory array (Fig A).

Consider the calculation of the sum of products, a common task in signal-processing and matrix-manipulation algorithms. The basic sum-of-products equation is

$$Y = \sum_{i=1}^N k_i x_i,$$

where k_i and x_i represent coefficients and data stored in memory, respectively. The sum-of-products computation represents a large class of array-processing problems that share three fundamental characteristics: First, they involve repetitive computations on arrays of data. Second, the underlying control structure is simple, having many loops but no conditional branches. Third, the math steps are memory-intensive—each calculation requires one data point and one constant from memory.

To evaluate a product term, the computer fetches x_i and k_i , multiplies them, and then adds the result to the running total. Each step requires an instruction-fetch cycle and an instruc-

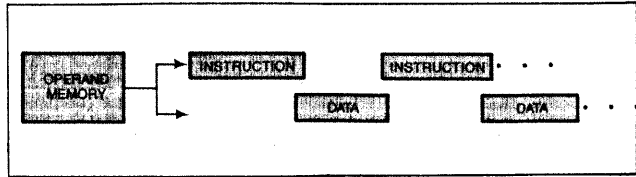


Fig A—A general-purpose computer memory stores instructions and data in the same block. The computer must access instruction and data values sequentially.

tion-execution cycle. Although specific details vary from computer to computer, in general even primitive math operations require many cycles.

Overlapping operation

Traditionally, Von Neumann-type computers perform each step sequentially. Array processors, however, provide a degree of parallelism by doing more than one thing at a time. When data and program steps reside in separate memories—an arrangement that fits the Harvard-architecture model—instruction- and data-fetch operations can overlap (Fig B). In the case of the sum-of-products operation, the array processor fetches the input operands at the same time that it fetches the instruction that performs the multiplication. Most array processors also overlap instruction-fetch and instruction-execution operations.

For highly regular, math-intensive algorithms, the overlapping results in high-speed operation, but such operation can be inefficient when the algorithm includes conditional branches. If,

for example, a program calls for a conditional branch to another instruction, the instruction following the branch instruction may be in the instruction queue. If it is in the queue, the computer discards it. Array processors are therefore best suited to the many number-crunching algorithms that require little or no conditional branching.

Because array processors provide parallel operation, you can optimize them for a specific math process. For example, an array processor designed for a sum-of-products operation may contain a multiplier and adder circuit, which evaluates a product term in one cycle. Because array processors perform parallel operations, programming the processors is more demanding than programming a general-purpose computer. However, the resulting increase in computational power often justifies the additional programming effort. Instead of programming in Basic or in assembly language, you'll use a microcode that controls individual circuits and operations in the array processor. Although such programming is demand-

ing, it gives you complete control of the array processor's internal operations.

Five functional blocks

Array processors typically receive data and instructions from a host machine—usually a general-purpose computer. Although specific array-processor architectures vary greatly, most processors contain at least five functional blocks: an arithmetic unit, data memory, a controller, program memory, and a host interface.

The heart of the processor is the arithmetic unit, which controls the data paths and performs arithmetic operations. Depending on your application, the arithmetic unit performs fixed-point operations, floating-point operations, or both. For some high-speed, real-time applications, such as radar- and video-information processing, array processors operate on 12-, 16-, or 24-bit fixed-point data. However, the trend is toward 32-bit

floating-point data processing.

The data-memory—usually banks of high-speed RAM or PROM—supplies operands to the arithmetic unit and stores results from the arithmetic unit. The data memory can have multiple data ports, depending on how fast the memory chips must supply operands and accept results. If it doesn't have enough ports or enough speed, the data memory can become a processing bottleneck, leaving the arithmetic unit starved for operands.

Controller is simple

The controller sequences the array processor through its operations. Because most array-processing algorithms have modest sequencing requirements, the controller isn't complex. Controllers provide a program counter (PC) that you increment to access the next program-memory word. You can also load the PC with the program memory's output to force the controller to jump to a different part of

the program. The controller includes a loop counter, which counts repeated operations. Depending on the array processor's sophistication, the controller may incorporate circuits that control nested subroutines, interrupts, and conditional-branch operations.

The program memory stores the array processor's microcode, which controls the other processor elements. Like the data memory, the program memory can be RAM or PROM. Use PROMs when the algorithms are well-defined and unlikely to change. Use RAM during algorithm development. The resources in the array processor determine the microcode memory's bit width. For example, a 60-bit-wide program memory provides 30 bits that control the arithmetic unit, 15 bits that transfer information to the controller (including a 12-bit jump address), and 15 bits that control other internal array-processor resources.

The host interface transfers data and instructions between the host computer and the array processor—usually by DMA operations. The host computer sends the array processor a block of data and an instruction word that selects a processing algorithm. After processing the data, the array processor transfers the results to the host computer.

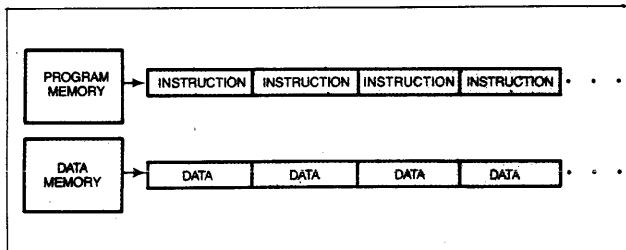


Fig B—An array processor's memory provides separate storage blocks for instructions and data. The separate storage areas let the control circuits access instructions and data in parallel.

An array processor can include pipeline registers that let the circuit overlap tasks.

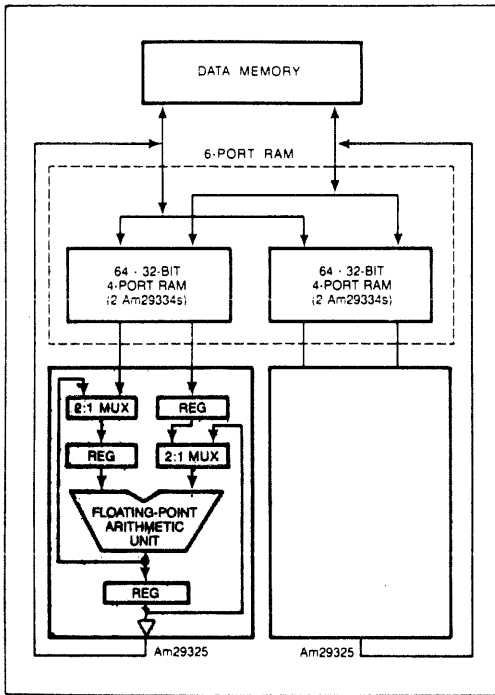


Fig 3—A 6-port RAM speeds data transfers so that two math-processor chips can operate independently. The chips can process data from the memory or from one another.

array processors provide both a general-purpose and a dedicated address-generator circuit. You'll find separate address generators for data and coefficient memories in array processors that provide extremely high processing speeds.

An address generator reduces the size of your array processor's program memory, and it increases the processor's speed. To increase processing speed further, consider adding arithmetic hardware to your design so the processor can do several computations in parallel. In the basic array-processor design, the arithmetic unit performs one operation at a time—for example, sums of products, which involve alternate addition and multiplication operations. The array processor performs the multiplication and addition operations sequentially.

The throughput of the basic array processor is 250 nsec per floating-point product term; to increase that

speed you can gang two 29325 floating-point math processors (Fig 3). The processors communicate through a 6-port RAM. When the circuit incorporates a multiport RAM, the floating-point processors can each access two input operands and store one result during each clock cycle. Because data produced by one floating-point processor is accessible to the other, you can double the processing speed for such algorithms as sum-of-products: One processor produces product terms, while the other processor sums and accumulates them. Of course, you can choose other math-chip configurations that better suit specific array-processing tasks. Keep in mind, however, that although you gain higher-speed operations by providing parallel math chips, your programming tasks grow. Coordinating the software operations of several parallel math chips can be difficult.

Memory expansion increases throughput

When you upgrade the arithmetic unit by adding parallel math chips, you must improve the data memory as well. The data-memory configuration in the basic array processor limits processing speed because the processor only accesses one constant and only performs one RAM-read or -write operation per clock cycle. To let the array processor perform operations that require two operands from RAM in the same cycle, or that require RAM-read and -write operations during the same cycle, you must upgrade the memory. Possible enhancements include converting the coefficient PROM to high-speed RAM, running the data RAM at twice the processor's speed to allow single-cycle reading and writing, or replacing the data RAM with a 2-port RAM.

In addition to high processing speeds, some applications may require rapid data transfers between the array processor and the host computer. There are at least two ways of speeding the transfer of data from the host to the array processor. First, you can replace the array processor's data RAM with a 2-section memory (Fig 4) that gives the host computer access to one section while the array processor uses the other. When the array processor completes its task, it switches between the buffers. The host obtains the results from the array processor's old buffer, while the processor operates with the data in the host's old buffer. The host computer's and the array processor's operations are no longer sequential; instead, they overlap. You'll have to pay careful attention to the manner in which the array processor controls the 2-section memory, because you

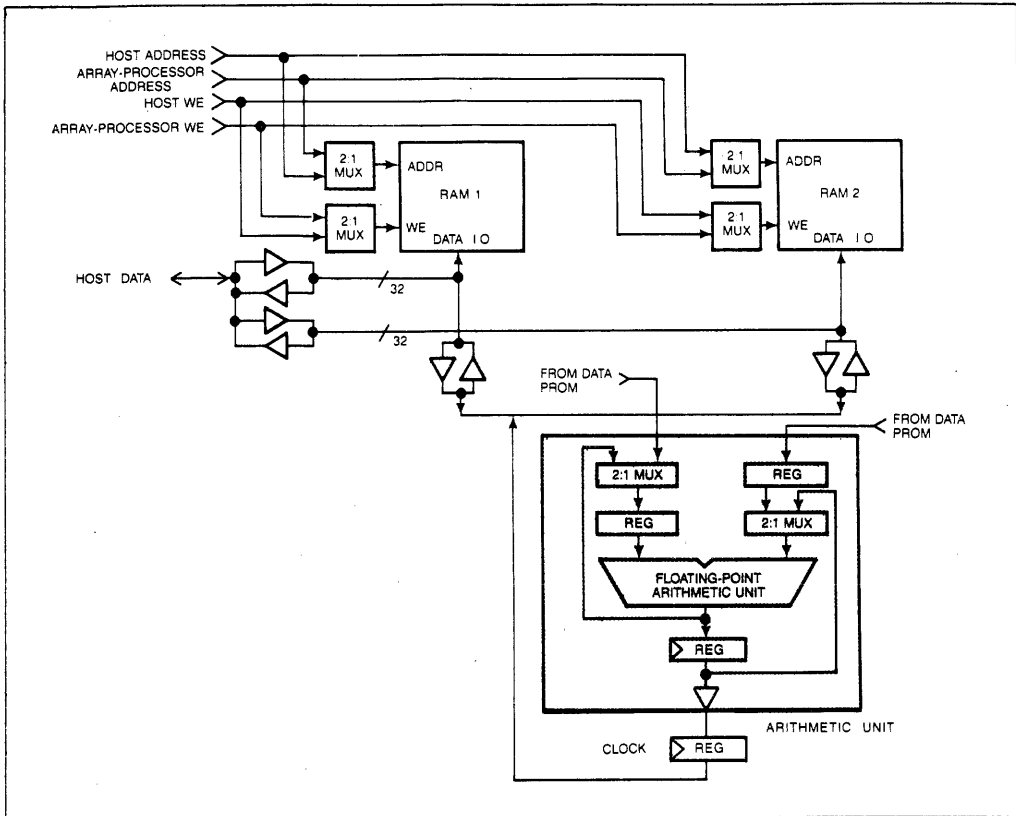


Fig 4—A 2-section memory offers a speed enhancement. The host processor reads or writes from one section, while the array processor processes the data in the other section.

don't want to switch buffers while the host or the array processor is still using one.

A second approach involves bypassing the host computer and letting the array processor take data directly from the data source—for example, an A/D converter. The processor uses the data and passes results to the host computer.

The 2-section-memory and direct-data-input techniques aren't mutually exclusive. In a given application, you might send data from an A/D converter directly to a 2-section memory. In this case, when the A/D converter's memory is full, it switches the memory section to the array processor.

Dividing the work load

By adding both direct-data input and output ports to your array-processor design, you can connect several processors in series, letting each one perform a subset of your algorithm. After it processes a piece or block of information, each processor passes results to the next processor in the chain.

The basic array processor performs addition, subtraction, multiplication, and format-conversion operations. For complex and transcendental operations, you'll need specific microcode routines that offer cosine, sine, and other functions. Standard algorithms are available, so your programming tasks aren't insurmountable. Part 3 of EDN's floating-point series will explore transcendental functions and tell how to implement them.

EDN

Author's biography

Robert M Perlman is the section manager for arithmetic accelerators at Advanced Micro Devices Inc (Sunnyvale, CA). Bob has been with AMD for 2½ years. His work involves developing and defining new products. He holds a BSEE from RPI, and he obtained his MSEE from Johns Hopkins University in 1981. Bob is an extra-class amateur-radio operator (KG6AF).

Floating-point μP implements high-speed math functions

This final article in a 3-part series describes how to incorporate a floating-point processor into your system. It discusses criteria for the selection of the algorithms you'll use, and in particular it details the methods used to implement transcendental functions.

David Quong, *Advanced Micro Devices*

If your application must perform a variety of math functions at high speeds on a wide range of input data, consider designing a math subsystem based upon a VLSI floating-point processor. A floating-point processor, a microsequencer, RAM, and ROM, configured as shown in Fig 1, together with the appropriate algorithms, will allow you to perform most math functions at real-time speeds with high precision and a very large dynamic range. A system of this type will outperform even the fastest floating-point coprocessor.

The choice of algorithms is an important step in the realization of your math processor. You can choose from a variety of methods for implementing transcendental and other math functions: The Taylor series, the Chebyshev series expansion, and the Newton-Raphson approximation are just a few of the many possible approaches. Which algorithm is the best one for your particular application will depend upon what functions you want to perform, the hardware architecture you are

using, and the system throughput and accuracy you expect to receive.

Many designers select the Taylor series for performing math functions. This well-known method allows you to find equations for various functions in most books of math tables. The Taylor series has a major drawback, however: It has a nonuniform convergence rate in the number of terms needed to achieve a desired accuracy. Consider, for example, the Taylor series expansion of the sine function:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \dots$$

For values of x near zero radians, this equation converges very quickly, but as x becomes larger, you'll need a larger number of terms to evaluate $\sin(x)$ to the same accuracy that you obtained for the smaller values.

The Chebyshev expansion method, like the Taylor method, produces a polynomial approximation, but it's not so well known. The generation of the Chebyshev approximation for a particular function is more complex than for the Taylor series, but the resulting polynomial is just as easy to implement. The major advantage of the Chebyshev method is that it has uniform convergence. Moreover, for any given function, over the operating range of the Chebyshev series this method yields smaller errors than almost any other method. You can usually determine by inspection the upper bound of the error; the error of the truncated series

Reprinted with permission from EDN, February 6, 1986. Copyright 1986, Reed Publishing USA.

A math-processing subsystem incorporating a VLSI floating-point processor will outperform even the fastest available floating-point coprocessor.

cannot exceed the sum of the absolute values of the remaining Chebyshev coefficients. (For details of the derivation of the Chebyshev series, see box, "Deriving a Chebyshev series.")

Iteration handles simple functions

For some simple functions such as division and square-root extraction, the Newton-Raphson method, an iterative approach for approximating such functions, works well. When using this or any other iterative method, you have to start with a seed, or initial approximation. The better this approximation is, the faster will be the convergence. You can store predetermined seed values in a look-up table. This method usually requires extra hardware (in the form of ROMs), but it gives you flexibility, because you can store seed values that are as accurate as you want.

The chief attraction of the Newton-Raphson method is its rapid convergence; the number of iterations required is low. The method converges quadratically,

ie, the order of the error is squared by each iteration. For example, if the seed is accurate to eight bits, the first iteration improves the accuracy to 16 bits, and the second iteration improves it to approximately 32 bits (variance depends on the magnitude of the error).

The math processor shown in Fig 1 evaluates Chebyshev and Newton-Raphson approximations very efficiently. The system performs transcendental (trigonometric, logarithmic, and exponential) functions by the Chebyshev method and division and square-root extraction by the Newton-Raphson method.

Understand the algorithms

The algorithms for 10 very common math functions are described below. You'll need these functions for applications associated with navigation, guidance, image processing, signal processing, and many other areas. The algorithms for the transcendental functions are based on the Chebyshev method and consist of a 3-stage process. The first stage reduces the range of

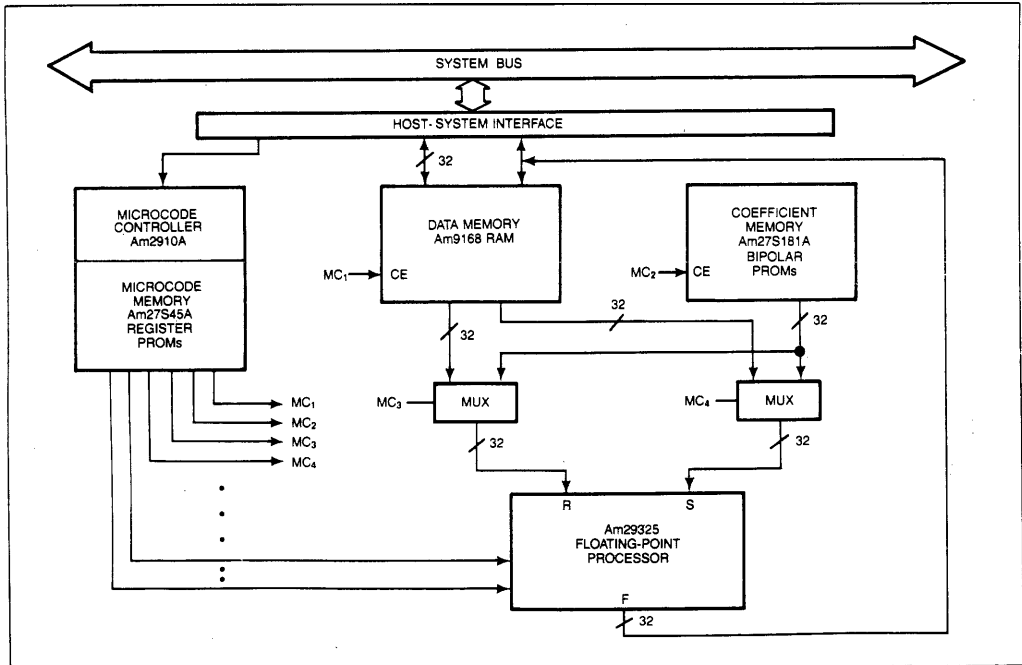


Fig 1—This math subsystem is based on a VLSI floating-point processor. It performs math functions with high precision and a large dynamic range.

Deriving a Chebyshev series

The Chebyshev series expansion is a procedure for generating a polynomial approximation for a given math function, $f(x)$. To expand the function, you must express it as a Chebyshev series:

$$f(x) = 0.5C_0 + C_1T_1(x) + C_2T_2(x) + \dots$$

for $-1 \leq x \leq 1$, where $T_n(x)$ is the Chebyshev polynomial of degree n given by

$$T_n(x) = \cos(n \times \arccos(x))$$

and C_n is a coefficient of the Chebyshev series. The value of C_n is dependent upon the function $f(x)$. You can determine the value of C_n by evaluating the following relationship:

$$C_n = \frac{2}{\pi} \int_{-1}^{+1} \frac{f(x) T_n(x)}{\sqrt{1-x^2}} \delta x.$$

Alternatively, you can obtain the C_n coefficients in tabular form, for a wide variety of functions, from books on mathematical tables (Ref 2).

Examples of the $T_n(x)$ polynomial include the following:

$$\begin{aligned} T_0(x) &= \cos(0) = 1 \\ T_1(x) &= \cos(\arccos(x)) = x \\ T_2(x) &= \cos(2\arccos(x)) \\ &= 2 \cos^2(\arccos(x)) \\ &= 2x^2 - 1. \end{aligned}$$

You can generate a polynomial equation for a function by combining the above equations and combining terms with common exponents. The accuracy of the result depends

upon the number of terms you use. (If you are interested in a formal derivation of the Chebyshev method, see Refs 1 and 2.)

Expansion for sine function

If you want to find the Chebyshev expansion for the sine function, first go to the coefficient tables in Ref 2 and look up the coefficients for the sine function (or calculate them from the formula given above). Next, determine the number of coefficients required to provide the accuracy you want. For example, to achieve 24 bits of accuracy, the error should be no greater than one part in 17 million. Compare the magnitude of this largest acceptable error with each of the coefficients.

The first term that contains a coefficient that's less than the error can be the last term in the series. It's common practice, however, to include one extra term in the series.

Using the above criteria, you need only six coefficients for the sine function using $\sin(\frac{1}{2}\pi x)$ in order to obtain a result that's accurate to 24 bits. These coefficients are

- $C_0 = C_{\sin 0} = +2.552557925$
- $C_1 = C_{\sin 1} = -0.285261569$
- $C_2 = C_{\sin 2} = +9.118016007 \times 10^{-03}$
- $C_3 = C_{\sin 3} = -1.365875135 \times 10^{-04}$
- $C_4 = C_{\sin 4} = +1.184961858 \times 10^{-06}$
- $C_5 = C_{\sin 5} = -6.702792 \times 10^{-09}$

Substituting the $T_n x$ polynomials into the Chebyshev series gives

$$\begin{aligned} \sin(\frac{1}{2}\pi x) &= \\ &0.5C_0 + C_1x + C_2(2x^2 - 1) \\ &+ C_3(4x^3 - 3x) \\ &+ C_4(8x^4 - 8x^2 + 1) \\ &+ C_5(16x^5 - 20x^3 + 5x). \end{aligned}$$

Simplifying the terms gives

$$\sin(\frac{1}{2}\pi x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5.$$

where

- $a_0 = (0.5)C_0 - C_2 + C_4$
- $a_1 = C_1 - 3C_3 + 5C_5$
- $a_2 = 2C_2 - 8C_4$
- $a_3 = 4C_3 - 20C_5$
- $a_4 = 8C_4$
- $a_5 = 16C_5$

The final result for the sine function is a simple polynomial equation that you'll find easy to implement. You can precalculate the coefficients a_0 through a_5 and store them in a ROM table. You can apply the same procedure to any well-behaved function for which you can find or compute the Chebyshev coefficients.

The Chebyshev expansion method, like the Taylor method, produces a polynomial approximation, but it's not so well known.

the input arguments to values between +1 and -1, because the Chebyshev expansion operates only over this range. The second stage evaluates the polynomial derived from the Chebyshev expansion. The third stage performs any postprocessing that may be required, such as correction of the sign.

The detailed descriptions were developed by Clenshaw, Miller, and Woodger (Ref 1). They use the terms RND and CSERIES: RND indicates that the result of the operation must be rounded towards minus infinity, and CSERIES indicates that the Chebyshev series for the input must be evaluated.

Range reduction prepares arguments

The range-reduction steps for the sine function are

- $x = x(2/\pi)$
- $x = x - 4(\text{RND}(0.25(x+1)))$
- If $x > 1$ then $x = 2 - x$.

As noted, these steps reduce the input argument to the range $-1 \leq x \leq 1$. You then evaluate the sine function by summing the terms of the following polynomial equation derived for the sine function:

$$\sin(x) = x(\text{CSERIES}_{\sin}(2x^2 - 1)).$$

The range-reduction steps for the cosine function are

- $x = x(2/\pi)$
- $x = 4(\text{RND}(0.25(x+2))) - x + 1$
- If $x > 1$ then $x = 2 - x$.

You then evaluate the cosine function by using the same polynomial equation as for the sine function:

$$\cos(x) = x(\text{CSERIES}_{\cos}(2x^2 - 1)).$$

The range-reduction steps for the tangent function are

- $x = x(2/\pi)$
- $x = x - 4(\text{RND}(0.25(x+1)))$
- $y = x$
- If $x > 1$ then $x = 2 - x$.

The Chebyshev polynomial evaluation for the tangent function is

$$\tan(x) = x(\text{CSERIES}_{\tan}(2x^2 - 1)).$$

You have to perform one postprocessing step:

$$\text{If } y > 1 \text{ then } \tan(x) = 1/\tan(x).$$

You don't need any range-reduction steps for the

arcsine function, because all values outside the range $-1 \leq x \leq 1$ indicate an error condition. For input arguments in the range $x^2 \leq 1/2$, you evaluate the arcsine as follows:

$$\text{asin}(x) = x(\sqrt{2}(\text{CSERIES}_{\text{asin}}(4x^2 - 1))).$$

For input arguments in the range $1/2 < x^2 \leq 1$, you evaluate the arcsine as follows:

$$\text{asin}(x) = \text{sign}(x)(\pi/2)(\sqrt{2 - 2x^2})(\text{CSERIES}_{\text{asin}}(3 - 4x^2)),$$

where $\text{sign}(x)$ is the sign of x .

You use the following trigonometric identity to evaluate the arc-cosine function:

$$\text{acos}(x) = \pi/2 - \text{asin}(x).$$

The range-reduction steps for the arctangent function are

- $u = x$
- If $|\text{ABS}(x)| > 1$ then $x = 1/x$,

where $|\text{ABS}(x)|$ is the absolute value of x . The Chebyshev polynomial evaluation is

$$\text{atan}(x) = x(\text{CSERIES}_{\text{atan}}(2x^2 - 1)).$$

The postprocessing steps are

$$\text{If } u > 1 \text{ then } \text{atan}(x) = +(\pi/2) - \text{atan}(x)$$

and

$$\text{If } u < -1 \text{ then } \text{atan}(x) = -(\pi/2) - \text{atan}(x).$$

The range-reduction steps for the exponentiation function are

- $x = x(\log_2 e)$
- $N = 1 + \text{RND}(x)$.

The Chebyshev polynomial evaluation is

$$\exp(x) = 2^N(\text{CSERIES}_{\exp}(2(N-x) - 1)).$$

Only positive values are valid input arguments for the natural-log function; a zero or a negative value should be flagged as an error:

$$\ln(x) = (\text{CSERIES}_{\ln}(4(\text{mant}(x)) - 3)) + (\text{expo}(x) - 1)(\ln(2)),$$

where $\text{mant}(x)$ is the mantissa value of x , $\text{expo}(x)$ is the exponent value of x , and $\ln(2)$ is a constant value.

You perform division operations by evaluating the

reciprocal function. For example, you can express the division operation $C=A/B$ in its reciprocal form, $C=A(1/B)$. By using the Newton-Raphson method, you can find an iterative expression for the reciprocal function. This expression is

$$x_{i+1} = x_i(2 - Bx_i),$$

where x_0 is the initial divisor reciprocal (seed value) for $i=0$, and X_i is the i th approximation.

The square-root function also uses the Newton-Raphson method. The iterative expression for the inverse square-root function is

$$x_{i+1} = 0.5(x_i(3.0 - Ax_i^2)).$$

You then evaluate the square root of A by the equation

$$B = A(x_{i-1}),$$

where A is the input argument, B is the square root of A , x_0 is the initial approximation (seed value) for $i=0$, and x_i is the i th approximation.

The principal component of the math-processor subsystem described here is the Am29325 floating-point processor. The subsystem also contains RAM, bipolar PROMs to store coefficients, a subsystem controller, and a host interface. The floating-point processor performs all computations under control of the subsystem controller; microcoded programs to perform the functions you need reside in the subsystem controller's PROM. If you wish to modify existing functions or add new functions, you merely change the microprogrammed PROM.

The Am29325 floating-point processor (Fig 2) provides many features that simplify subsystem design. The 3-port, 32-bit I/O structure of the Am29325 avoids data multiplexing and allows efficient transfer of information. The 32-bit internal registers and data paths allow the chip to store the results of intermediate calculations for use in subsequent operations, thereby avoiding the delays that transfer of these results to and from off-chip storage would entail. Many functions don't need to send data out of the chip until the final results of an operation are ready.

The floating-point-processor hardware detects exceptional conditions and, rather than compounding the error until the end of the calculation, immediately notifies the host system. The chip notifies the host by means of flags that indicate underflow, overflow, inva-

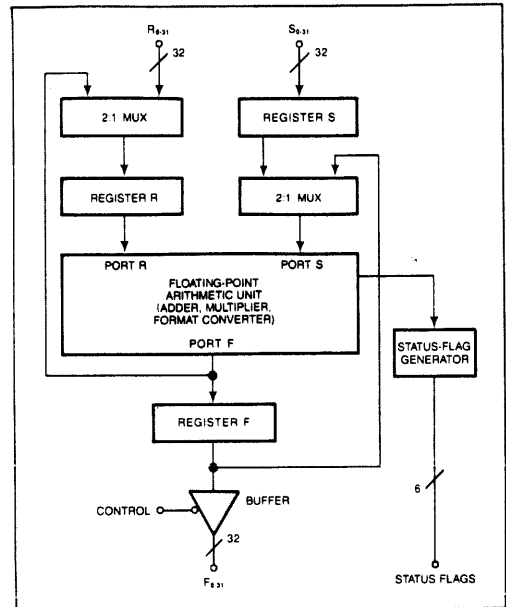


Fig 2—This VLSI floating-point processor is fast because it contains all the major components for 32-bit operations on a single chip. It has one input for an external clock and 17 inputs for instruction-select and control functions.

lid operation, and other error conditions.

Subsystem data storage consists of a high-speed, 4-port RAM. You can load the data memory from the host computer (using DMA), from the floating-point processor, or from an integer processor. You'll need to process integers during operations such as isolating the exponent and mantissa portions of a floating-point word. You can have the host processor perform integer processing, or you can arrange it so that the math subsystem performs the required operations by incorporating an integer processor chip in your design.

Learn to microprogram the processor

Two examples of how to implement math functions on the Am29325 floating-point processor will give you an introduction to the microcoding procedures you'll use in the math processor. Recall, that, for a given division operation ($C=A/B$), the Newton-Raphson division algorithm begins by obtaining the reciprocal of the divisor by means of an iterative equation. A single iteration requires just three arithmetic operations:

- multiplication: $B(x_i) = u$
- subtraction: $2 - u = v$
- multiplication: $v(x_i) = x_{i+1}$.

You can microcode this procedure with a 3-instruction loop that you repeat until you obtain a sufficiently accurate value of x_{i-1} . You then perform a single multi-

The math processor uses the Newton-Raphson method to execute the division and square-root functions.

plication, $A \times x_{i-1}$, to obtain the quotient.

The conventional way to obtain a seed is to use the most significant 16 or so bits of the divisor as a pointer into a look-up table in ROM; the contents of the address to which the divisor bits point become the seed output, which usually has approximately the same number of bits. You might think that use of a 16-bit address would require a ROM that's 64k words deep, but this is not so. In floating-point division, you can reciprocate the exponent and significand separately, each from its own table, and then recombine them. Consequently, for an 8-bit exponent and the eight most significant bits of the significand, you require only two tables, each just 256 words deep.

You can also trade ROM word width for execution time (ie, the number of iterations); doubling the width of the significand stored in ROM will reduce reciprocal refinement time by roughly one iteration. Convergence is specified by the inequality $2/B > |x_0| > 0$.

The microcoding for the complete Newton-Raphson division is shown in Table 1. The operation requires six lines of microcode. In cycle 1, you load the seed into register R of the floating-point processor and load the divisor into register S. In cycle 2, you multiply the contents of registers R and S; the result appears in register F.

In cycle 3, you perform the subtraction, using the 2-S instruction of the floating-point processor. The

input for port S comes from register F via the internal feedback path. The result of the subtraction appears in register F.

In cycle 4, you perform the second multiplication. This operation multiplies the contents of register F (via port S) by x_i (from register R). The result, x_{i+1} , replaces x_i in register R. In parallel with the multiplication, the microsequencer executes a jump back to cycle 2 to begin the next iteration.

Cycle 5 begins after the last iteration of cycles 2 through 4. In this cycle, you load the dividend (A) into register S and multiply it by the contents of register R to produce the final result. This result appears in register F, from which you can unload it via the F bus to local data storage or to the host.

The second implementation example uses the Chebyshev method to perform a sine calculation. In the polynomial equation that evaluates the sine function,

$$CSERIES_{\sin} = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5.$$

The range-reduction steps require eight or nine operations. Evaluation of the polynomial equation requires 23 additional operations, including processing of the $2x^2-1$ expression. One final operation multiplies the result of the polynomial evaluation by x . The sine function therefore requires 32 or 33 operations.

You can, however, save 10 cycles in the evaluation of

**TABLE 1—INSTRUCTION SEQUENCE FOR
NEWTON-RAPHSON DIVISION ON THE Am29325**

CLOCK CYCLE	10-ALU SELECT	11-ALU SELECT	12-ALU SELECT	13-SMUX CONTROL	14-RMUX CONTROL	ENR-R REG ENABLE	ENS-S REG ENABLE	ENF-F REG ENABLE	OE-OUTPUT ENABLE	ALU OPERATION	CONTENT OF REG R	CONTENT OF REG S	CONTENT OF REG F	COMMENT
1	X	X	X	X	0	0	0	X	X	X	?	?	?	LOAD B AND SEED INTO Am29325
2	0	1	0	0	X	1	1	0	X	R'S	X(0)	B	?	BEGIN FIRST ITERATION
3	1	1	0	1	X	1	1	0	X	2-S	X(0)	B	B*X(0)	
4	0	1	0	1	1	0	0	0	X	R'S	X(0)	B	2-B*X(0)	X(1)=X(0)[2-B*X(0)], LOAD A
5	0	1	0	0	X	1	1	0	X	R'S	X(1)	A	X(1)	A*X(1), X(1)=1/B
6	X	X	X	X	X	X	X	X	0	X	X(1)	A	A*X(1)	OUTPUT RESULT. A/B

X=DONT CARE ?=UNKNOWN

The floating-point processor hardware detects exceptional conditions and, rather than compounding the error, immediately notifies the host system.

the polynomial equation by applying Horner's Rule, an algebraic method for rearranging components in a polynomial. The polynomial equation then becomes

$$C\text{SERIES}_{\sin} = (((a_5x + a_4)x + a_3)x + a_2)x + a_1)x + a_0.$$

The total number of operations in the sine function then decreases to 22 or 23. Evaluation of the rearranged polynomial equation is complete in 10 clock cycles.

In cycle 1, you load x into the S register and a_5 into the R register. Multiply these two operands to produce $a_5 \times x$. In cycle 2, you load the result of the multiplication into the F register, load a_4 into the R register, and add the contents of the F and R registers to yield

$$(a_5 \times x) + a_4.$$

In cycle 3, you load the result of the addition into the R register; the S register still contains x . Perform $R \times S$ to obtain

$$((a_5 \times x) + a_4)x.$$

Cycles 4 through 10 perform similar addition and multiplication operations, progressively using the terms a_3 through a_0 . The final result of evaluating the polynomial equation is available in the F register after cycle 10.

The ability to perform both simple and complex math functions rapidly is critical in systems that process data in real time. You won't yet find many simple, compact solutions to this problem on the market. Math-coprocessor ICs are available, but they are still in the low- to medium-performance range, and they limit you to a microprocessor environment. (Table 2 shows compara-

tive timings for two floating-point coprocessor chips and the Am29325 floating-point processor.)

You can design and build your own MSI chip, but such a product will require much development time and cost, and it will probably be large and consume lots of power. Another possible approach is to compute the values of the math functions you will need and to store these values in ROM, but such a look-up-table method is adequate only for small amounts of data. At the present time, the use of a math subsystem based upon a VLSI floating-point processor with a relatively small amount of support circuitry appears to be the most cost-effective solution. **EDN**

References

1. Clenshaw, CW, Miller, GF, and Woodger, M, "Algorithms for Special Functions I," *Numerische Mathematik*, Vol 4, 1963, pg 403. See also Miller, GF, "Algorithms for Special Functions II," *Numerische Mathematik*, Vol 7, 1965, pg 194.
2. Clenshaw, C W, "Chebyshev series for mathematical functions," Vol 5 of the *Mathematical Tables of the National Physical Laboratory*, HM Stationery Office, London, 1962.

Author's biography

David Quong is an engineer in the product planning division of Advanced Micro Devices (Sunnyvale, CA). He received a BSEE from California State University at Sacramento, and in his spare time enjoys fishing, hiking, and skiing.

**TABLE 2—TIMING COMPARISON
OF SINGLE-PRECISION FLOATING-POINT FUNCTIONS**

FLOATING-POINT CHIP	SPEED (MHz)	ADD (μ SEC)	MULTIPLY (μ SEC)	DIVISION (μ SEC)	SQUARE ROOT (μ SEC)	SINE (μ SEC)	COSINE (μ SEC)	TANGENT (μ SEC)
INTEL 8087 ¹	8.0	12.5	18.1	25.4	23.3	NOTE 3	NOTE 3	67.5
MOTOROLA 68881 ²	16.67	2.8	3.1	3.8	N/A	23.0	23.0	27.2
AMD Am29325	8.0	0.125	0.125	1.125	1.625	2.875	3.125	4.750

NOTES:

N/A = TIMES NOT AVAILABLE.

1. TIMES FOR THE INTEL 8087 WERE DERIVED FROM THE INSTRUCTION CLOCK COUNT GIVEN IN THE INTEL DATA PAMPHLET (1984) ALL TIMES LISTED ARE WORST CASE.

2. TIMES FOR THE MOTOROLA MC68881 WERE TAKEN FROM A NEWS ITEM IN *ELECTRONIC PRODUCTS*, FEBRUARY 15, 1985, PG 43.

3. THIS OPERATION IS NOT COVERED BY THE INSTRUCTION SET AND MUST BE IMPLEMENTED BY USING OTHER INSTRUCTIONS.

Optimize your graphics system for 2-D and 3-D

The design of a graphics system that's both 2-dimensional and 3-dimensional poses some conflicting requirements. You can reconcile some of these conflicts, however, through careful design of the frame-buffer structure, and you can achieve adequate speed for 3-D applications by using parallel processors for computation-intensive tasks.

Anoop S Khurana and Olivier Garbe,
Advanced Micro Devices Inc

A graphics system that will handle both 2- and 3-dimensional applications presents design requirements that are at odds with one another. These conflicts arise from the fundamental differences in the nature of the geometry-, pixel-, and display-processing tasks required by the two systems. A system with a microprogrammed architecture can help you avoid the difficulties you'd encounter in reconciling these differences.

You'd use a 2-D graphics system with such graphics editors as MacDraw, MacPaint, and Interleaf, or with CAE programs such as schematic-capture packages or layout editors for pc-board design. You'd need a 3-D system, on the other hand, to display 3-D wire-frame

models, to model solids for mechanical design, or to produce visually pleasing 3-D pictures for animation.

One of the major differences lies in the size of the frame buffer needed, and the speed with which the host computer can obtain access to it. Most 2-D systems need only eight bits to define a pixel color as one of 256 simultaneously displayable colors. A 3-D system, on the other hand, needs eight bits each for red (R), green (G), and blue (B)—a total of 24 bits per pixel. Also, 2-D pixel-processing operations require fast access to multiple pixels during the same frame-buffer cycle. In a 3-D system, by contrast, pixel-processing operations (such as Gouraud shading) are computation-intensive but require access to only one pixel at a time.

Similarly, geometry-processing operations are more arithmetic-intensive in 3-D than in 2-D systems. Fixed-point, 32-bit arithmetic provides adequate computational power and speed for many 2-D applications, whereas 3-D applications need the speed and versatility of fast floating-point arithmetic.

Most of the graphics systems available today, including engineering workstations, are optimized for 2-D graphics operations; if they have 3-D capabilities, they perform the required processing mainly in software, which is slow. To obtain adequate speed, then, serious users of 3-D graphics find that they need a separate system that's optimized for 3-D graphics, resulting in an expensive duplication of hardware and software.

You can avoid these disadvantages by designing a single graphics system that provides all the features

A 2-dimensional graphics system can handle diagrams, but you need 3-dimensional capability for mechanical modeling.

necessary for both 2-D and 3-D graphics. You'll find a microprogrammed architecture ideal for such a system, because such an architecture lets you customize the data paths and computational resources to a particular application and to the performance level that you want. It also lets you integrate both fast integer and fast floating-point arithmetic capabilities, both of which are necessary for complex graphics operations, into a single system.

As an example of such a system, consider the design of a graphics peripheral for a conventional minicomputer. This peripheral can act as a bus master on the host's system bus, but it need not do so. The application program runs on the host computer and generates a display list, defining the image, which the CPU passes to the graphics peripheral via a DMA channel (or by any other appropriate means). The graphics peripheral processes this display list to generate the image. (The

steps that convert a display list to an image on the screen are collectively referred to as the "graphics pipeline"; see box, "From object to image: the graphics pipeline.") The three main functional blocks of the system are the communications and display-list handler; an update processor that performs geometry and pixel processing; and a display controller (Fig 1).

A conventional, general-purpose, 16- or 32-bit μP , which has its own memory and DMA channel, receives and executes commands issued by the host. This communications processor can directly execute some host commands, such as Load Display-List. Other commands, such as Render Display-List, involve the rest of the graphics system; the communications processor analyzes these commands and dispatches appropriate commands to the update processor, using a message-based protocol and a fast, dual-access memory block that serves as a mailbox.

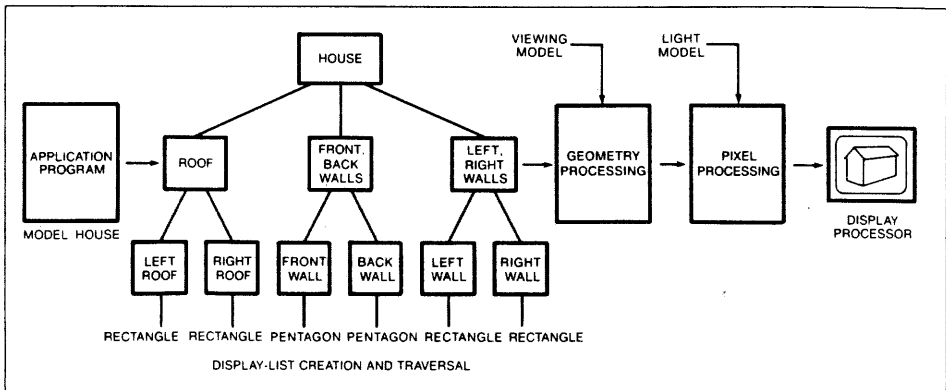
From object to image: the graphics pipeline

The graphics pipeline is the sequence of operations that translates the user's description of a scene into a viewable image. The four stages in this process are display-list handling, geometry processing, pixel processing, and display control.

The display-list handler helps the user or the application program decompose objects to be depicted into a display list. The display list is usually hierarchical, and it embodies the structure inherent in the object being modeled. Leaf nodes in the hier-

archy are drawing primitives provided by the graphics system.

The geometry processor performs viewing- and perspective-transformation operations on the display list, and it clips objects against the boundaries of the



The graphics pipeline consists of the processing steps needed to convert a graphics object description, in digital form, into a viewable image on the screen.

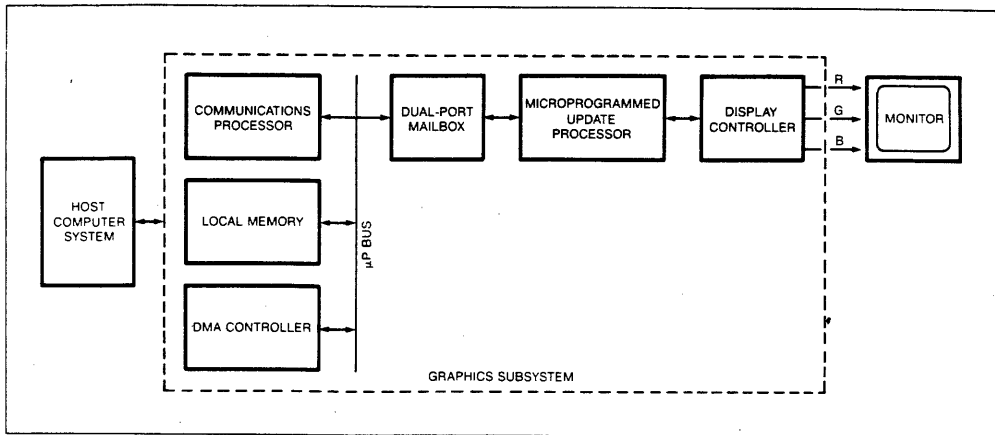


Fig 1—A graphics subsystem is ideally an intelligent peripheral that accepts a display list from the host computer and converts the digital representation of an image into a standard video signal that creates a screen display.

The dual ports of the mailbox allow the update processor to read a command while the communications processor is sending a subsequent command. Semaphores, also located in the mailbox RAM, govern both command chaining and the allocation of memory to message buffers.

The microprogrammed update processor executes all

commands that are related to geometry or pixel processing. Such operations may update the pixel data in the frame buffer, or they may pass a message back to the communications processor.

The frame buffer uses video RAM (VRAM) ICs, both to maximize bandwidth and to minimize the quantity of hardware needed for refreshing the image. The frame-

viewing volume. You can decompose the complex primitives used by the geometry processor, such as patches or cubic curves, into simpler primitives, such as polygons or lines.

The pixel processor physically writes all the pixels affected by a primitive into their correct locations in the frame buffer. It also performs all operations, such as pixel-block transfers, that require pixels to be read from or written to the frame buffer.

The display controller converts the pixel values stored in the frame buffer into a standard video signal. This video signal, when transmitted to a suitable monitor, builds the desired image on the screen.

A single, general-purpose processor, such as the Intel 80286, along with the 80287 numeric coprocessor, can perform all the operations in the graphics pipe-

line sequentially. In such a system, the main processor writes the final value of each pixel to the frame buffer, which forms part of the address space of the main processor. This configuration is relatively slow, however, and the speed may be inadequate for 3-D applications.

You can achieve improved performance by using specialized VLSI peripheral devices, such as the Am95C60 Quad Pixel Dataflow Manager, to speed some of the operations in the graphics pipeline. Most current graphics peripherals relieve the main processor of most of the pixel-processing tasks. Typical functions performed by such peripherals are line drawing, polygon filling, and block transfer of pixels. Because these tasks are relatively standard and are well suited to implementation in high-performance silicon, graphics peripherals yield a substan-

tial improvement in system performance. You can achieve a similar improvement by using high-performance floating-point processors to speed the computation-intensive geometry-processing tasks.

For even higher performance and functionality, you should consider the use of multiprocessing systems that provide one or more processors for each stage in the graphics pipeline. Two factors contribute to the improvement in performance that such systems yield. First, because most graphics operations are vector operations, the concurrent performance of several parts of a task can yield a speed increase that's proportional to the number of processors available. Second, you can fine-tune the system by customizing it for highest performance in just those operations that the applications require.

A microprogrammed architecture lets you customize the resources of the system to the problem you're trying to solve.

buffer controller provides all the signals needed for reading, writing, and refreshing the VRAMs, and for performing all video-refresh functions.

You'll need to organize the structure of the frame buffer carefully to make the most efficient use of the available storage. As noted, for 2-D displays you need only eight bits per pixel, which allows you to display the pixel in one of 256 colors. For 3-D displays, you need at least 24 bits per pixel (eight each for the R, G, and B channels); you may also need, for each pixel, an additional eight bits for the alpha channel and 16 or 32 bits for the Z buffer (a maximum of 64 bits/pixel).

You can reduce the total number of bits per pixel by mapping the Z buffer into a portion of the frame buffer. For example, in a $2k\text{-pixel} \times 1k\text{-line}$ buffer, you could map a $1k \times 1k$ -pixel screen into the first $1k$ pixels of each line and the Z buffer into the second $1k$ pixels. Consequently, you could access the Z value of a pixel by adding an offset of 1024 to the pixel address. You would need two memory cycles to access both the RGB and the Z values of the pixel. This structure, however, has the great advantage that no bits are irrevocably dedicated to the Z buffer. If you don't need a Z buffer, this memory becomes available for general use.

You'll still have to resolve the discrepancy between the eight bits/pixel needed for 2-D and the 24 bits/pixel needed for 3-D. Your first thought might be to allocate a 32-bit memory word for each pixel, but then you'd be wasting 24 bits in 2-D operations. A better solution is to allow each 32-bit word to be treated as four adjacent 8-bit pixels in 2-D. You could then reorganize a $2k \times 1k \times 32$ -bit memory as a frame buffer of $8k \times 1k \times 8$ bits. This organization allows you to store one 3-D screen with a resolution of $1024\text{ pixels} \times 1024\text{ lines} \times 32$ planes, or several 2-D screens at once.

The frame buffer in our example consists of $64k \times 4$ -bit VRAMs and uses the shifter port of each VRAM for video refreshing; the update processor therefore has virtually unlimited access to the frame buffer. It's possible to organize each VRAM as a $256 \times 256 \times 4$ -bit square area of memory; using this area as a building block, you can create a $2k \times 1k \times 4$ -bit memory array having four rows and eight columns (Fig 2). If you want to extend the depth of the array to 32 bits/pixel, you'll need eight VRAMs in each element (called a bank) of the array.

The video display controller (VDC) provides complete control of the frame buffer, both for update operations and for video-refresh operations. In response to a read or write memory-cycle request from

the update processor, the VDC generates the appropriate VRAM-control signals ($\overline{\text{RAS}}$, $\overline{\text{CAS}}$, etc). If a dynamic-RAM refresh cycle or a transfer cycle for video refresh is already in progress, however, the VDC delays execution of the update cycle until the higher-priority cycle is finished.

Because each access to the frame buffer reads or writes a 32-bit word, the $2k \times 1k \times 32$ -bit frame buffer requires 21 address lines, of which 11 define the X address and the other 10 define the Y address within the array. In the 3-D 32-bit/pixel mode, each 32-bit word in the frame buffer represents one pixel.

In the 2-D 8-bit/pixel mode, each 32-bit word represents four pixels. The 18 most significant address bits select the 8-bit row address, the 8-bit column address, and $\overline{\text{RAS}}$ strobe signals. Decoding the three least significant bits yields a decode signal that selects one of eight adjacent pixels.

The capacitive loading imposed by the VRAMs makes it necessary to buffer the address and control outputs of the display controller. To reduce skew between signals, and thereby achieve a shorter memory-cycle time, you can buffer the address, $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and XF/G signals within a single IC package, such as the Am2976 11-bit dynamic memory driver used in this example.

Select one of eight pixels

Each of the eight rows in the frame memory receives a separate $\overline{\text{RAS}}$ signal. You can therefore connect to a common 32-bit bus the data ports of all four banks of VRAMs within a column. Each memory cycle now gives access to eight pixels, one from each column. The update processor operates on only 32 bits at a time, however, so you'll need a mechanism to select just one of the eight available words.

You can perform this 8:1 multiplexing quite simply by decoding the three least significant address bits to obtain the $\overline{\text{CAS}}$ signal. As a result, only one bank in memory receives both $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$. Consequently, you can tie together the outputs of all 32 banks in memory, but only the selected bank will drive the bus. To access eight sequential pixels, then, you'd need eight memory cycles.

There's another way to perform the multiplexing, however—one that gives the update processor very rapid random access to any or all of the eight adjacent pixels addressed in a single memory cycle. This method requires eight 32-bit, bidirectional, bus-interface registers. You connect the eight 32-bit words, accessed in parallel from the memory, independently to one port of

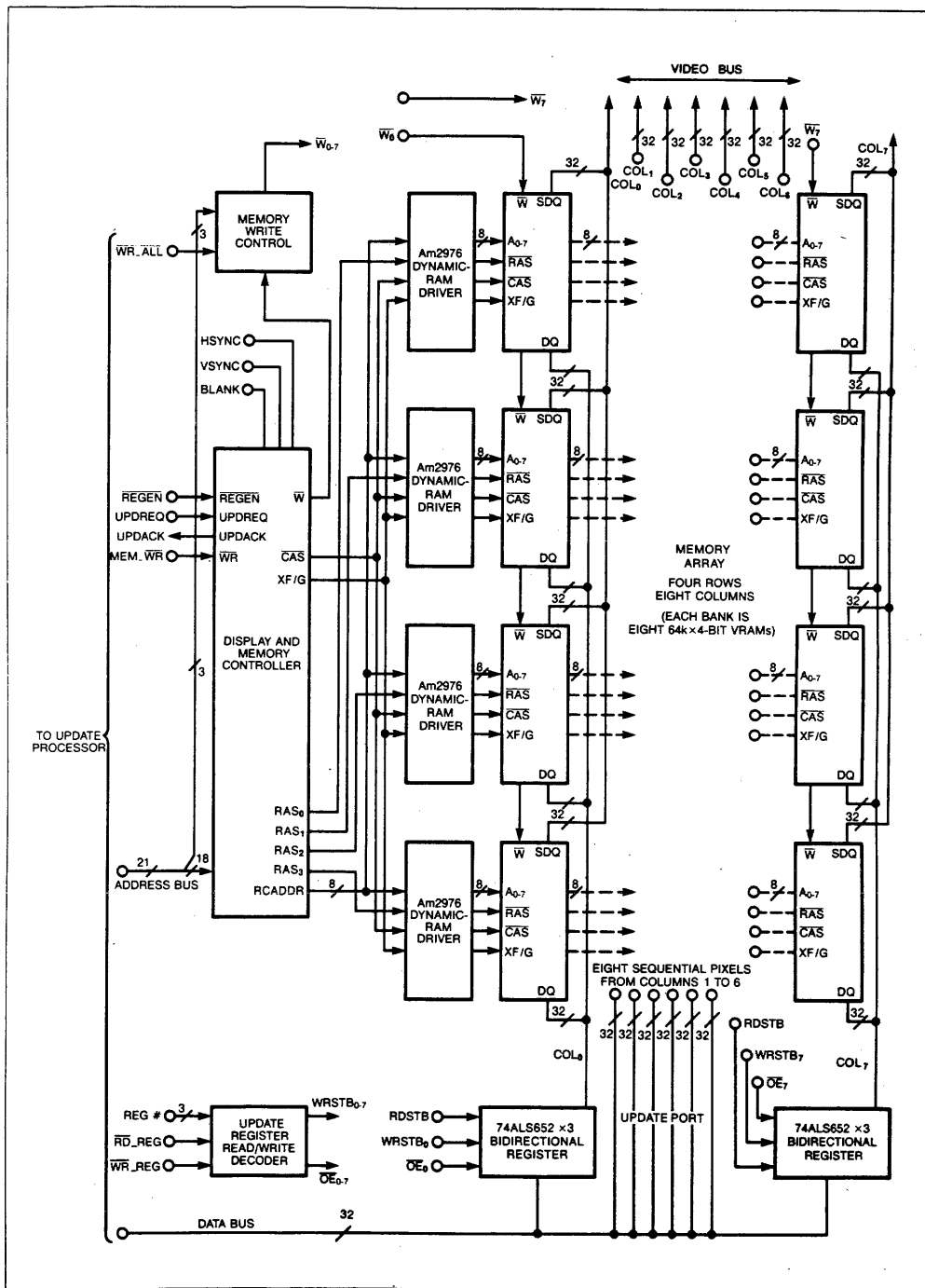


Fig 2—This frame buffer is organized as 2k pixels x 1k lines x 32 bits. Three-dimensional applications can read or write eight adjacent pixels at one time. For 2-D applications, each 32-bit word represents four 8-bit pixels.

A microprogrammed graphics system acts as a peripheral on the host computer's system bus.

these registers. To the other port you tie corresponding bits of each register together to form a single 32-bit bus that leads to the update processor. You then perform the 8:1 multiplexing by controlling the output-enable signals of the registers.

The update processor regards the registers as independent 8-pixel input and output buffers. A memory-read operation fills the input buffer, and the update processor can fetch any or all of the eight pixels much more quickly than if a separate memory cycle were required for each one. You can also provide two different write modes. In the first mode, the update processor writes just one pixel to the appropriate place in memory. In the second mode, the update processor fills all eight registers, and the memory cycle writes their contents to eight different pixels simultaneously.

Refreshing the video display is easy when the display

memory consists of VRAMs. At every vertical-sync (Vsync) pulse, the display controller resets an internal video-refresh counter to the address of the upper-left corner of the screen. At every horizontal-sync (Hsync) pulse, the controller initiates a transfer cycle that transfers data for the next scan line into the VRAMs' shift registers and then increments its internal address counter to point to the start of the data for the next line. You can perform panning and scrolling simply by changing the address held in the controller's top-of-frame register.

Given that there are eight memory banks per row, and that each VRAM is capable of shifting at a clock speed of 25 MHz, a total bandwidth of 200M pixels/sec is possible in 3-D mode. In 2-D mode, the available bandwidth becomes 800M pixels/sec. The maximum pixel bandwidth is therefore limited mainly by the

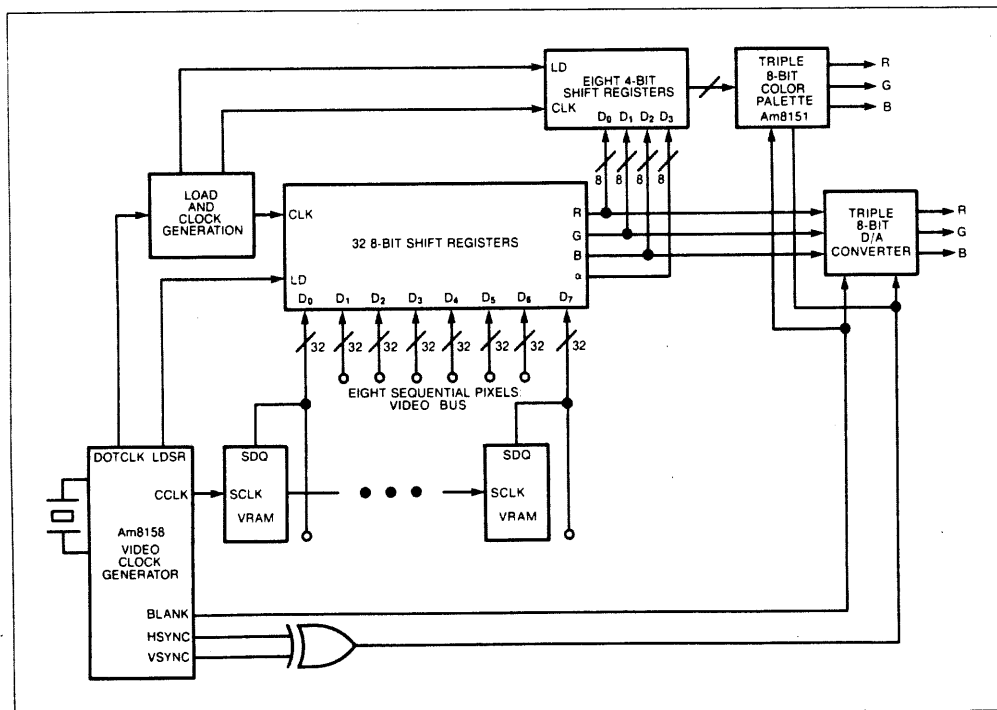


Fig 3—You'll need two video shift registers if you want to reconfigure the frame buffer from 32-bit, 3-D pixels to 8-bit, 2-D pixels or vice versa. The main register handles eight sequential 32-bit pixels; the secondary register reformats the RGB bit streams from the main register into RGB streams representing 8-bit pixels.

characteristics of the shift registers and the associated D/A converter, not by those of the memory.

In 32-bit/pixel mode, strobe signals generated by the video clock generator—in this example, an Am8158—load into the video shift registers the eight sequential 32-bit pixels that are in parallel on the video bus (Fig 3). The video shift registers consist of 16 dual, 8-bit, parallel-in, serial-out ECL shift-register ICs. These ICs produce serial bit streams of the R, G, and B values of each pixel and forward these bit streams to a triple 8-bit D/A converter.

In 8-bit/pixel mode, the 32 bits that appear at the R, G, and B outputs of the shift registers actually represent four pixels. Four 4-bit ECL shift registers convert the 32-bit data into four 8-bit pixels for use by the Am8151 ECL color palette. To change from one mode to the other, you need only make the appropriate modifications to the Shift and Load signals to the shift registers.

The Am8158 generates the pixel clock pulse and some of the Shift and Load signals used by the shift registers. This IC also generates the Vsync, Hsync, and Blank pulses. The display controller uses these signals to initiate VRAM transfer cycles, and the D/A converters use them to force the video signals to the appropriate sync or blank levels. You can program all the important parameters of these signals using registers contained in the Am8158.

The update processor is microprogrammed

The update processor performs all pixel- and geometry-processing functions for both 2-D and 3-D graphics. These functions require powerful and versatile data-transfer capability coupled with fast integer and floating-point arithmetic. Implementing the update processor as a microprogrammed subsystem allows you to achieve the high performance that you need.

The major functional blocks and buses of the update processor are shown in Fig 4. The main data path in this example consists of the Am29332 integer ALU, the Am29323 integer multiplier, and the vector floating-point arithmetic unit, which consists of two Am29325 ICs. Each of these units accepts data from two common 32-bit input buses and places its results on one common 32-bit output bus (the main data bus).

An Am29334 register file provides storage for frequently accessed data. Its read ports supply data to the arithmetic unit's input buses. It also has two write ports, one of which accepts data from the main data bus, while the other transfers the result of an ALU

operation back to the register file without using the main data bus. The system timing is such that the ALU can fetch two operands from the register file, process them, and write the result back to the register file within a single microcycle.

The update processor addresses 64k 32-bit words of high-speed local data memory, which consists of static RAM. An Am2131 dual-port message-buffer IC occupies 1k words of the 64k-word address space. To allow the main ALU to process video data at maximum efficiency, an auxiliary Am29C101 16-bit ALU performs all local-memory address computation; the outputs of this ALU are captured in a 16-bit address register. Random accesses to local memory therefore take two microcycles—one to compute and latch the address, and another to access the RAM. During consecutive memory accesses, however, next-word computation overlaps the current RAM access, so that the second and subsequent memory accesses are completed in a single microcycle.

The frame-buffer-address generator consists of pre-settable up/down counters (an 11-bit counter for the X address and a 10-bit counter for the Y address). The sequencer loads these counters via the main data bus. Although the main ALU is primarily responsible for generating frame-buffer addresses, use of the counters speeds the critical loops in curve drawing and other pixel-processing functions.

The update processor is configured with a single level of pipelining, so that next-address computation overlaps execution of the current microinstruction. The Am29331 sequencer computes the address of the next instruction in response to its instruction inputs, and it places the result on its Y output bus. For access to sequential microcode addresses, this result is simply the contents of the program counter. The sequencer uses an internal stack to store count values for nested loops and return addresses for calls to microcode sub-routines.

To execute a jump to an address defined by the microcode, the sequencer connects the address section of the microinstruction word back into its program counter via the A bus. To allow the computation of jump addresses at run time, and to allow external examination of the sequencer's stack and stack pointer, the D bus connects to the main system bus.

An internal condition-code multiplexer, controlled by microcode, selects and enables one of the condition inputs of the sequencer; the sequencer can then test that condition and jump according to the state of the

The organization of the frame buffer is the key to resolving conflicts between 2-D and 3-D requirements.

selected input. For testing as many as four conditions simultaneously, a PAL device accepts all the signals that need to be tested simultaneously and encodes them into four fields of four bits each. A base address is assigned to each field, and the state of the field defines one of 16 sequential locations as an offset from the base address. The sequencer can then examine one of these fields and jump to the location defined by the state of that field. You can use this capability to advantage in a line-clipping algorithm.

In the 2-D mode, one of the most important pixel-processing operations is the movement of a rectangular block of pixels from one area of the frame buffer to another. This process, also known as BitBlt, may also require the execution of a logical operation during the transfer. The update processor transfers data one row at a time from the source block to the destination block.

Within a row, the processor may transfer data either left to right or right to left. The sole reason for including the feature that provides fast access to eight pixels in the frame buffer is to speed block transfer. In the 32-bit/pixel mode, the algorithm that transfers one row of the source block to the corresponding row in the destination block has four steps, as illustrated in Fig 5a and described as follows:

- Read memory with $X=24$. This operation transfers pixels 24 through 31 into the frame buffer's read registers. Next, read pixels 31 and 32 into the register file. Then read memory again with $X=32$. Read five pixels (32 through 36) into the register buffer. You have now transferred the first seven pixels from the source region into the register file (there are only seven valid pixels in the first destination read cycle).
- Read memory with $X=96$. This operation trans-

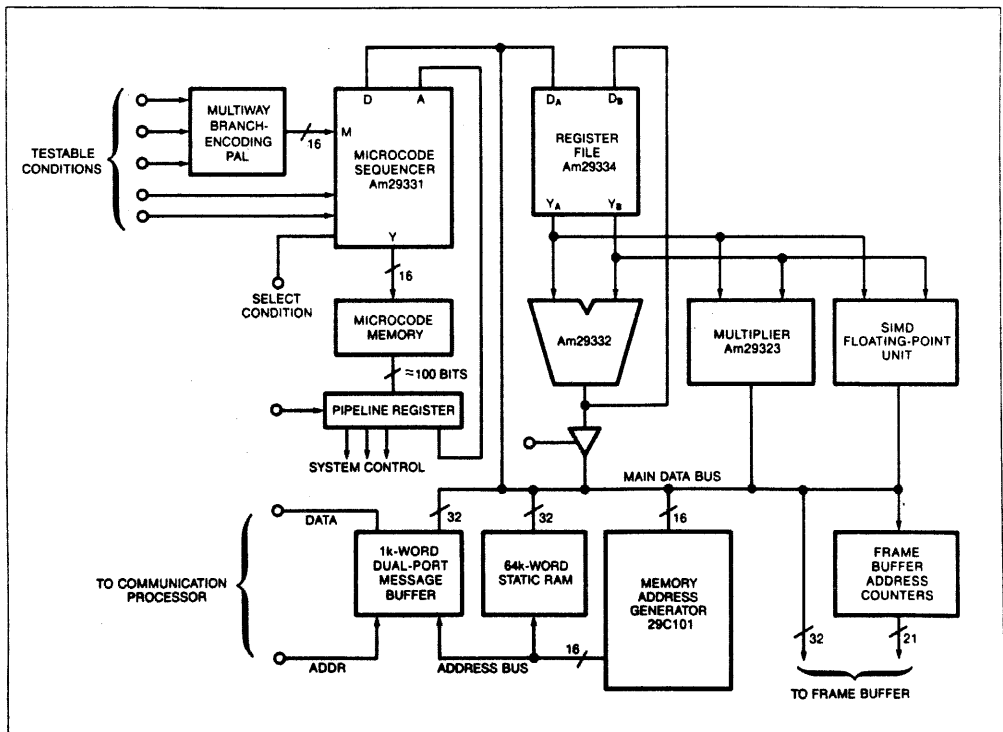


Fig 4—This update processor, which handles all geometry- and pixel-processing operations, uses a microprogrammed sequencer for control and parallel floating-point processors for vector operations.

fers seven valid destination pixels into the frame buffer's registers.

- Read each valid destination pixel, one at a time, and perform any required logical operation with the corresponding source pixel in the register file. Write the resulting pixel back into the frame buffer's write registers. Copy each unread destination pixel from the input register to the output register.

- Write the eight destination pixels in the output registers back to memory. Repeat the sequence until you have transferred the entire row.

Assuming that a memory-read cycle takes 300 nsec and that each frame-buffer read or write operation takes 100 nsec, the total transfer time is 500 nsec/pixel. Using this algorithm, an average covering all possible alignments of source and destination turns out to be approximately 600 nsec/pixel. This time is a substantial improvement over the time of 1200 nsec/pixel for the case in which each memory cycle accesses a single pixel, and it's an acceptable data-transfer speed for 32-bit pixels.

In the 8-bit/pixel mode, the block-transfer algorithm must take into account different alignments of the source and destination within a 32-bit word, and it requires a modification of the procedure. The modified

algorithm, illustrated in Fig 5b, is as follows:

- Read source words 1 and 2 simultaneously from both output ports of the register file. Using the Am29332 funnel shifter, extract four bytes aligned with the destination, and write this 32-bit word back to a temporary location in the register file. In the example shown, you need to extract the last three pixels of word 1 and pixel S2 from word 2.

- Read this aligned source location, using one register-file port. Read the destination pixel from the frame buffer via the main bus into the second register-file port.

- Perform the logical operation on the aligned-source and destination pixels, using the mask generated internally by the ALU; doing so leaves the first pixel unchanged by the logical operation. Write the result, which appears at the ALU's outputs, back to the frame buffer's input registers at the end of the cycle.

Step 3 of the algorithm now takes three microcycles per word instead of two, and it changes the average transfer time to just over 600 nsec per word. Because each word contains four pixels, the average pixel-transfer time is $600 \div 4 = 150$ nsec/pixel. This pixel-transfer rate allows an entire $1k \times 1k$ -pixel screen to be updated in 150 msec, or about 10 frame times, and is

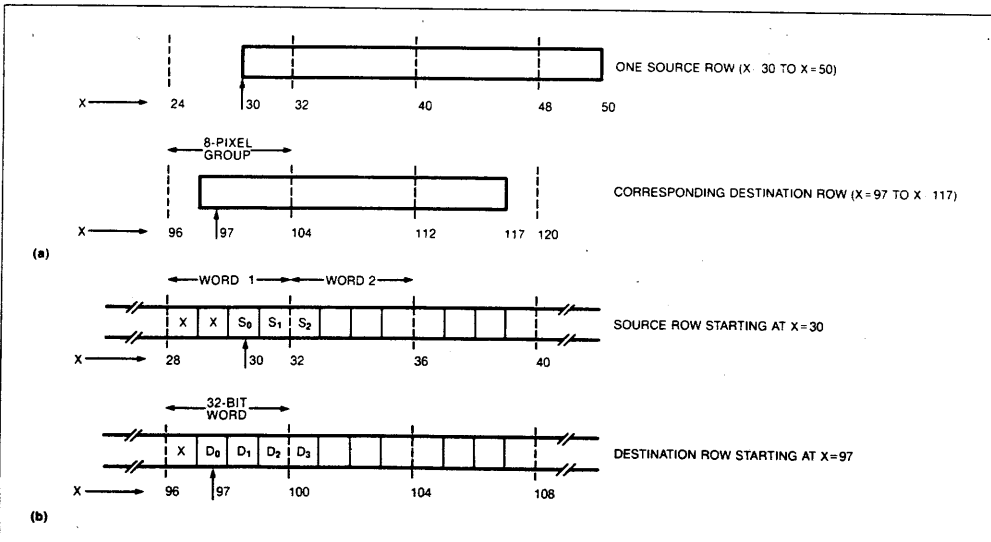


Fig 5—Pixel block-transfers need careful alignment of the source and destination within a group of pixels. In 32-bit/pixel mode (a), the group is eight pixels wide. In 8-bit/pixel mode (b), the group is four pixels wide.

The update processor needs fast access to several pixels at a time in the frame buffer.

sufficient for displaying text and manipulating windows.

It's not difficult to implement line- and circle-drawing algorithms, such as those of Bresenham, in microcode. The inner loop of Bresenham's line-drawing algorithm will require three microcycles. Because this time is equal to the time needed to access a pixel in the frame buffer, you can plot pixels at the pixel-access speed of the memory. However, because this algorithm does not profit from the fast access to sequential pixels, the plotting speed will be about the same in both the 32-bit/pixel and the 8-bit/pixel modes. The inner loop of Bresenham's circle-drawing algorithm will require four microcycles, and because each iteration through the loop generates eight points that must be plotted in separate memory cycles, circles too are drawn at the

rate of about one pixel in every frame-buffer access time.

Typical pixel- and geometry-processing operations in a 3-D system are computation-intensive and require that you carefully consider the design of the arithmetic unit. Integer arithmetic, although fast, is unsuitable for these graphics operations. Fixed-point arithmetic has disadvantages as well. Although you can readily perform most pixel-processing functions using 32-bit fixed-point arithmetic, fixed-point geometry-processing operations require time-consuming pre- and postscaling operations. For this reason, floating-point operations are easier to develop and are more general in character. Furthermore, there are now many inexpensive floating-point chips, which are almost as fast as integer units and provide all the computation power you need.

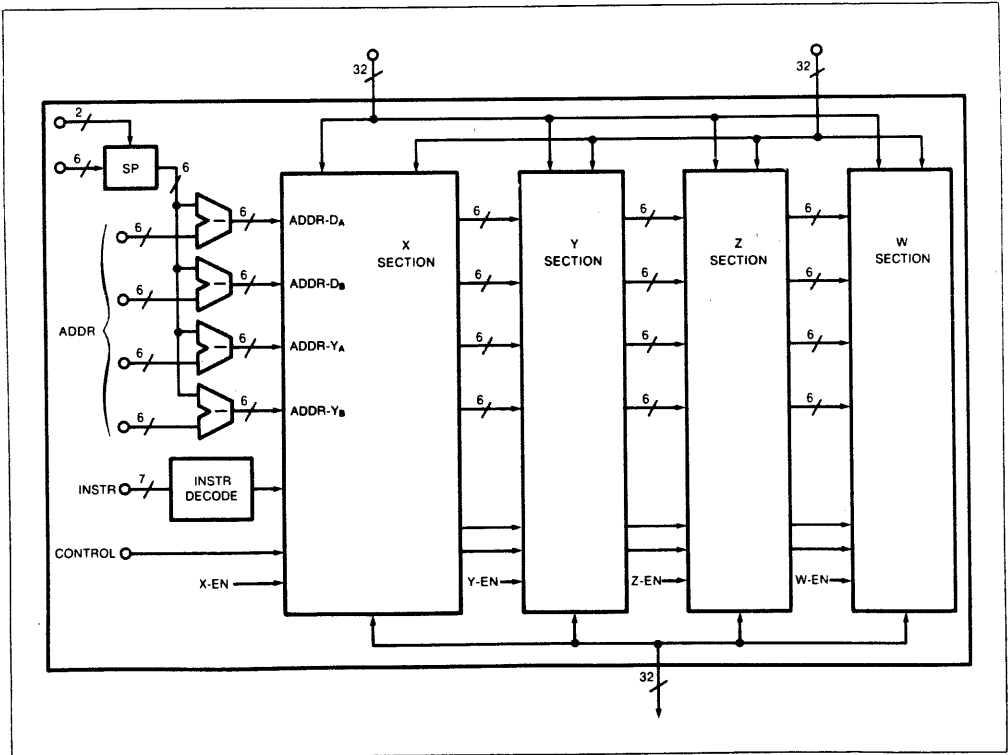


Fig 6—This SIMD floating-point unit has four sections that share a common control bus. All four sections concurrently perform the same operation on different data.

TABLE 1—TRANSFORMATION OF A 3-D POINT

CYCLE	EXECUTE	READ/WRITE
1		READ: $Y_A=R=ST(0)$, $Y_B=S=ST(4)$
2	EXECUTE: $F=R \cdot S$	READ: $Y_A=R=ST(1)$, $Y_B=S=ST(5)$
3	EXECUTE: $R=R \cdot S$	
4	EXECUTE: $F=F+R$	READ: $Y_A=R=ST(2)$, $Y_B=S=ST(6)$
5	EXECUTE: $R=R \cdot S$	
6	EXECUTE: $F=F+R$	READ: $Y_A=R=ST(3)$, $Y_B=S=ST(7)$
7	EXECUTE: $R=R \cdot S$	
8	EXECUTE: $F=F+R$	
9		WRITE: $D_A=F$, OUTPUT REGISTER=F (OPTIONAL)

In a graphics system, most of the arithmetic computations are vector operations, because points, plane-equations, transformation matrices, and other common data structures are all vectors. For example, you can represent a point in 3-D space, in homogeneous form, as the vector $(x \ y \ z \ w)$. Although a single processor can perform vector operations sequentially, a multiple-processor system that uses four ICs (in this example, Am29325s) is much faster. If you can distribute the computation tasks among the four processors in such a way that you keep each processor busy all of the time, you can expect to achieve four times the performance of a single processor.

Fortunately, it's quite easy to distribute the simple vector operations that are useful in graphics. For example, perspective division on a point $(x \ y \ w \ z)$ in homogeneous coordinates yields $(x/w \ y/w \ z/w \ 1)$. Consequently, you can perform these divisions in parallel on four different processors, and you can arrange for algorithms that do not map onto such an architecture to run (though more slowly) on a single processor as a sequence of scalar operations. Furthermore, the fact that all processors perform the same operation (division, in this example) at the same time (but on different data) suggests that you should design the floating-point unit as a single-instruction, multiple-data (SIMD) machine, whose processors share a common instruction bus.

You can see the overall structure of a 4-processor SIMD floating-point unit in Fig 6. Each section consists of a floating-point processor, a register file, and a seed ROM (Fig 7). In each section, a 64-word area of the stack constitutes the register file, and you can address data in the register file with a 6-bit negative displacement from the stack pointer. The microcode word therefore contains four 6-bit fields to specify the addresses of the four ports on the register file. The stack-addressing capability allows microcode subroutines to be completely general in character, and if you first load the stack pointer with zero, you can use the microcode-word displacement fields to specify absolute addresses.

The seven instruction bits of the main microcode word, when decoded, provide all the output-enable and multiplexer-select signals needed to reflect all possible arithmetic-operation and source/destination combinations. Twenty-four bits specify the addresses for the four ports of the register file, two bits control write operations on the D_A and D_B ports of the register file, and one bit switches the source-select multiplexer located at the register file's D_A input. Two additional bits

determine whether the stack pointer is to be left unchanged, incremented, decremented, or loaded from the data bus.

A data-access microcycle consists of three time slots. In the first slot, the address hardware computes register-file addresses by adding the displacement specified in the microcode word to the current contents of the stack pointer. In the second slot, data is written into the register file. In the last slot, data required for the next execution cycle is read from the register file.

The pipelined structure of the floating-point unit allows the overlapping of arithmetic operations with operations that access data from the register file. As a rule, the floating-point unit must access data from the register file one microcycle before using that data in an arithmetic operation. In many cases, however, the data needed for the next operation is already held in the Am29325's internal registers, so that a register-access cycle is unnecessary. Furthermore, most graphics operations allow execution cycles to overlap data-access cycles in a similar manner. Consequently, the effective throughput of the floating-point unit remains close to one operation per microcycle.

Guidelines for coding typical operations

As an example of how you can distribute portions of an operation among the four processors, consider the transformation of a 3-D point in homogeneous coordinates, using a $\times 4$ matrix. The first step is to broadcast all four coordinates of the point to be transformed, and to write them into the register files of all four sections of the floating-point unit simultaneously. Because the register file also acts as the matrix stack, the transformation matrix is already established in the floating-point unit. You then distribute the transformation matrix among the four sections, storing only one column of the matrix in each section.

Assume that the point to be transformed is on top of the stack at $\{ST(0) \ ST(1) \ ST(2) \ ST(3)\}$, and that the matrix column is at $\{ST(4) \ ST(5) \ ST(6) \ ST(7)\}$, where $ST(n)$ refers to the data n words down from the current stack pointer. You perform the transformation by computing the dot product of the point and a column of the transformation matrix. You can now compute, in parallel, the four dot products needed to transform each

The update processor is configured with a single level of pipelining, so that next-address computation overlaps execution of the current microinstruction.

component of the vector, one in each section of the floating-point unit. The entire transformation can complete within nine microcycles (Table 1).

You can use the same approach to perform matrix-matrix multiplication. In this case, assume that the current transformation is on top of the stack, with one column in each section. You can now treat a row of the new matrix as a point and transform it by the matrix held on top of the stack to yield a row of the trans-

formed matrix. You repeat this procedure four times (once for each row) to obtain the complete result. A matrix-matrix multiplication therefore takes 36 microcycles.

You can also perform parallel interpolation, using forward differences, when drawing cubic curves such as splines and Bezier curves. In this case, each iteration requires three addition operations, and because each component of the vector requires an identical computa-

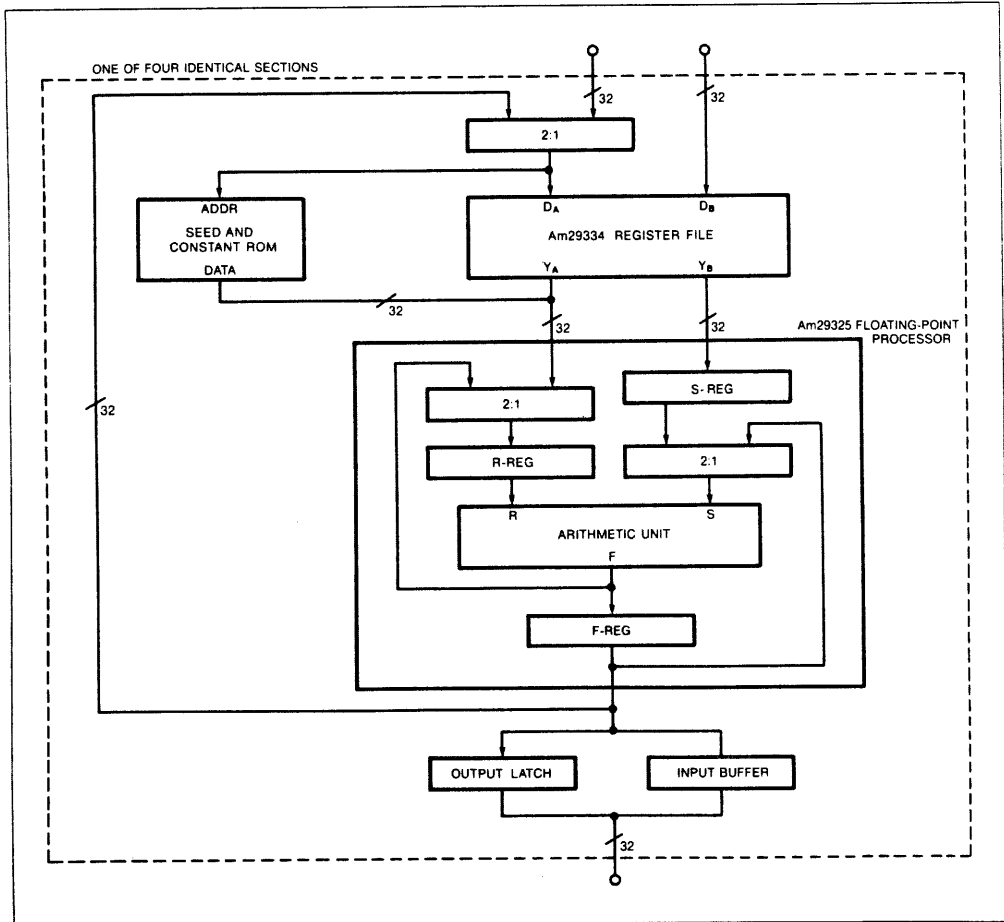


Fig 7—Each section of the SIMD floating-point unit is identical with the others, and each has its own register file, seed and constant table, and floating-point processor.

tion, you can perform the four computations in parallel in the four sections. Consequently, you can compute a new point every four microcycles. In the computation shown below, D_x , D_{2x} , and D_{3x} are the first-, second-, and third-order forward differences for the X coordinate:

$$[X D_x D_{2x} D_{3x}] = [X D_x D_{2x} D_{3x}] + [D_x D_{2x} D_{3x} 0]$$

$$[Y D_y D_{2y} D_{3y}] = [Y D_y D_{2y} D_{3y}] + [D_y D_{2y} D_{3y} 0]$$

$$[X D_z D_{2z} D_{3z}] = [Z D_z D_{2z} D_{3z}] + [D_z D_{2z} D_{3z} 0]$$

Perspective division requires a division operation, and the normalization of an interpolated vector, in the inner loop of Phong shading, requires square-root operations. The Am29325 does not perform division and square roots directly, however. Instead, it uses Newton-Raphson iteration to obtain the corresponding results. The seed ROM provides the seed (or first approximation) to start the iteration procedure. Each iteration requires three microcycles for division and five microcycles for square roots. Refining the seed to approximately single-precision accuracy requires another three microcycles. Consequently, each division operation requires a total of ten microcycles, and each square-root operation requires sixteen microcycles. Furthermore, because each processor in the floating-point unit has its own seed table, four such computations can proceed in parallel. **EDN**

Author's biography

Anoop S Khurana is a product planning engineer for advanced graphics at Advanced Micro Devices (Sunnyvale, CA). He's responsible for specifying architectures for peripheral graphics controllers. He holds a BSEE from the University of Roorkee in India and an MSEE from the University of Florida (Gainesville). His hobbies include squash, reading, and hiking.

Olivier Garbe is department manager for graphics and office automation in AMD's product-planning group. He is responsible for new-product definition. Olivier has a Diplôme d'Ingénieur from L'Ecole des Travaux Publics in Paris. His hobbies are PC hardware, wind surfing, water skiing, and scuba diving.

EDN March 18, 1987

DESIGN APPLICATIONS

Variable-width FIFO buffer sequences large data words

Tim Olson

Advanced Micro Devices Inc., 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088; (408) 732-2400.

First-in, first-out (FIFO) buffers are a popular means of matching different data rates in large digital systems. I/O controllers for character-oriented devices like terminals, for example, usually return or receive one 8-bit byte on a slow but regular basis. In contrast, block-oriented devices, such as high-speed disks, must move large chunks of data from peripherals to the host bus with great speed.

The demand for larger, denser data-processing systems has spurred the development of FIFO buffers

with deeper memory but unchanged width. Cascading these buffers horizontally or vertically is still the most common and cost efficient method of expanding both the width and depth of a data queue.

Even this solution has shortcomings. FIFO buffers usually

Fast systems gain from a cascadable device supporting everything from instruction pipelines to peripheral host adapters.

link devices of like width but do not possess the requisite logic to cope with, say, transferring data between a 32-bit-wide memory, 16- or 32-bit data buses, and an 8-bit peripheral bus. To further complicate matters, some of the newer variable-width instruction architectures must buffer instruction words varying in width from 8 to 128 bits at any particular cycle.

In short, as both synchronous and asynchronous systems push toward larger or disparate data widths, it becomes more difficult to cascade with typical 8- and 9-bit-wide FIFO buffers in a rudimentary fashion. Designers are seeking an efficient solution for matching data widths as well as data rates.

One of the best devices for such matching is the Am29338 Byte Queue FIFO buffer. The general-purpose, 32-bit-wide buffer is organized as four dual-ported RAMs, each 9 bits (1 byte plus parity) wide and 32 bytes deep (Fig. 1a). Each RAM section has its own queue (load) and dequeue (unload)

pointers (Fig. 1b) and supplies byte-wise (that is, byte-by-byte) parity checking at the buffer's input and output. A Byte Count output shows the current number of bytes in the queue. The RAMs are organized so that a variable number of bytes can be queued or dequeued at any cycle. The device can queue or dequeue from zero to four 8-bit bytes of data in one 80-ns cycle. Ultimately, this feature can be used to queue data at one width and dequeue it at another. For example, two 16-bit half words may be queued sequentially and dequeued as one 32-bit word. In addition, the Am29338 can be cascaded horizontally to release up to 16 data bytes (128 bits) per cycle.

The Am29338 also addresses the problem of byte ordering, a side effect of the evolution of memory word widths from 8 to 16 to 32 bits. Byte ordering is simply the order in which bytes appear in a word. The Am29338 performs byte swapping to effect any type of byte-ordering scheme. Two signals, for example, allow bytes to be swapped within 16-bit half words and 32-bit half words, respectively. Together, they make possible four separate byte orderings (Fig. 2).

Like the rest of the Am29300 family of 32-bit microprogrammable building blocks, the Am29338 is implemented in ECL (packaged in a 120-pin pin-grid-array) but is interfaced with TTL-level devices. Because it is RAM-based, the buffer has an almost zero fall-through delay, suiting it to applications where data must be immediately available after a queuing operation.

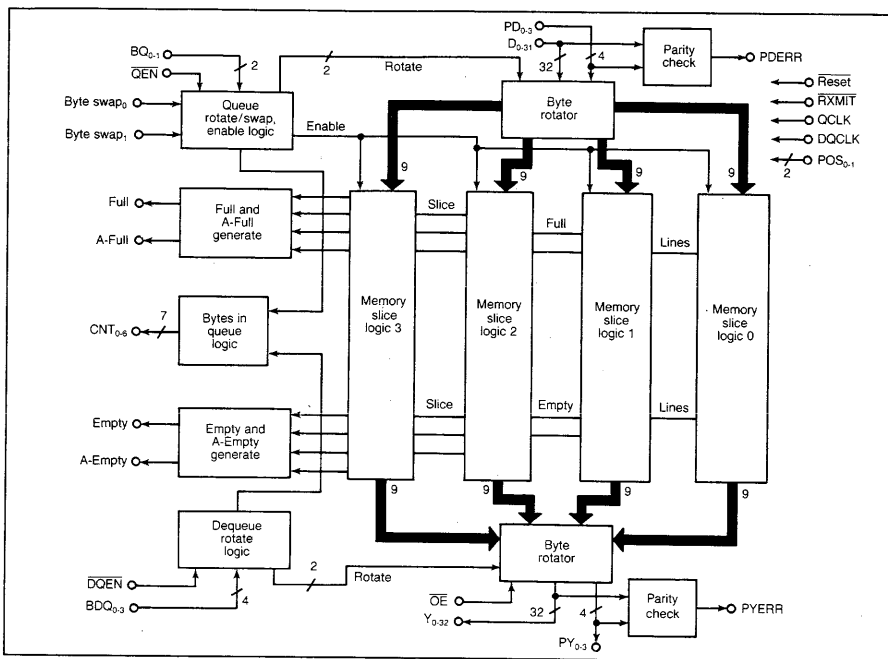
This feature best suits systems with variable data widths, especially instruction-prefetching pipelines, I/O peripheral buffers, and hardware mailboxes.

AN INSTRUCTION-PREFETCH QUEUE

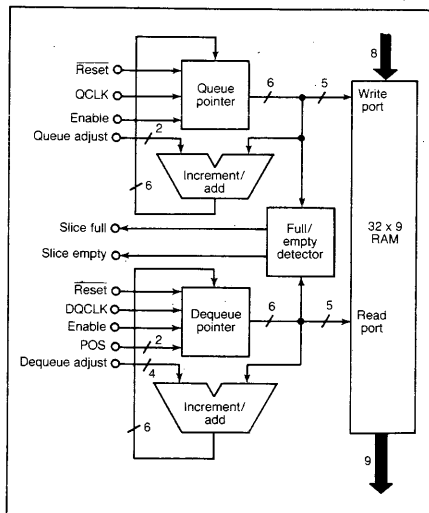
Instruction-prefetch queues, of course, separate instruction fetching from instruction execution for parallel execution of the two tasks. Between jumps from one operation to the other, a sequential instruction stream is fetched from memory and

"Reprinted with permission from Electronic Design, Vol. 35 No. 14, July 11, 1987. Copyright 1987 Hayden Publishing Co., Inc."

DESIGN APPLICATION ■ Variable-width FIFO buffer



1. The Am29338 Byte Queue from AMD is a general-purpose, 32-bit FIFO buffer with four 8-by-32-bit RAM memory stacks. It works in either the synchronous or asynchronous mode, can transmit data blocks, and performs error checking at both input and output. Up to four bytes can be queued or dequeued in one cycle (a). Each stack has its own pointers: queue and dequeue logic enabling variable-width data to enter and leave the FIFO buffer (b).



placed in the prefetch queue. This occurs independently of the rate at which the instructions are decoded and executed. Because many computer architectures work with variable-length instructions, the Am29338, which can release data of different widths, greatly simplifies prefetch-queue designs. Fixed-width words can be queued from memory while variable-length instructions are dequeued.

The Am29338 buffer can function as an instruction-prefetch queue, where it is synchronized with a separate instruction-fetch unit (Fig. 3). In operation, sequential 32-bit memory locations are fetched by the instruction-fetch unit and are stacked in the byte queue. Each time the CPU needs an instruction, it takes the next bytes in the byte queue rather than addressing main memory. The CPU can determine the instruction length from the first byte of the instruction and updates the dequeue pointer in the byte queue; that is, it tells the byte queue which bytes it wants to see. The instruction length is determined by the 4-bit word on the Bytes Dequeued (BDQ) lines while the Dequeue Clock (DQCLK) line releases the bytes from the queue. If a jump in the instruction sequence (the program) occurs, the instruction-fetch unit must flush the byte queue by asserting the Reset line and issuing a new instruction address.

EXECUTING SMALL LOOPS

The Byte Count (CNT) indicator can serve as a tool to limit the buffer's depth. For instance, jump or branch instructions usually account for about 20% of a typical instruction mix. When a jump occurs, instructions stored in the instruction-prefetch queue are discarded. To limit instruction-prefetching operations and conserve memory bandwidth, the user can sound an alarm when the fetch buffer's depth surpasses five or six instructions.

Many operations, however, can be executed with small loops, which fit entirely in the prefetch queue and can be controlled with the assertion of the retransmit lines (RXMIT) and with a small amount of external hardware. The Am29338 buffer can rapidly retransmit stored block data without requeuing from main memory, assuming that 128 bytes or less have been queued since the last assertion of a Reset command. This is done by first bringing the RXMIT line low. When this happens, the chip's internal dequeue pointers are directed to the first RAM location, and the internal queue pointers are not reset. The data in the locations between the old queue pointers and the new dequeue pointers can then be unloaded. RXMIT is useful for redundant instruction sequences because the CPU can run faster without having to refetch instructions from memory or cache.

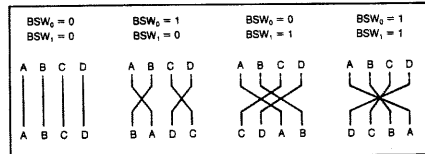
New applications open the door for instructions far in excess of 32 bits, particularly in systems that use large, variable-length instructions spanning many bytes. To meet this challenge in the synchronous mode, up to four Am29338s may be cascaded horizontally to free up to 16

consecutive bytes (one 128-bit word) for dequeuing in one cycle (Fig. 4a). Because each cascaded part is connected to a common 32-bit input bus, each chip holds the same information (Fig. 4b). When the Reset (or RXMIT) line is asserted, however, the internal dequeue pointers are offset by the value programmed on the chip's position inputs, POS.

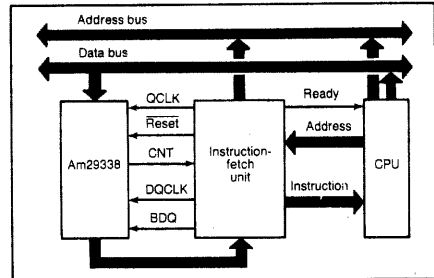
Another frequent task for first-in, first-out buffers is as a straightforward I/O buffer. Many processor-memory systems have expanded their word length from 8 to 32 bits, though the peripheral-controller chips have for the most part remained at 8 bits. The Am29338 buffer supplies a buffered path between peripherals and memory while making the necessary conversion from one word size to another.

MESSAGE IN THE MAIL

A communication mailbox usually serves to link two or more loosely coupled devices in a multiprogramming system. With the help of a first-in, first-out buffer, messages from one device to another are queued in the mailbox. If the mailbox happens to be full, the sending process blocks data transfer until the mailbox has a slot free. If the mailbox is empty, the receiving process is blocked until the mailbox receives a message from the sending end.



2. The data stacks make possible four different combinations of byte swapping. As a result, data can be queued at one width and dequeued at another.



3. The FIFO buffer can function as an instruction-prefetch queue by coupling it with a separate instruction-fetch unit. The CPU runs faster by reading repetitive instruction loops from the byte queue without addressing main memory.

DESIGN APPLICATION ■ Variable-width FIFO buffer

Otherwise, the sending and receiving processes run concurrently.

When devices are run on separate processors in a multiprocessor system, a hardware mailbox is needed. The Am29338 can help create such mailboxes (Fig. 5), serving to transfer variable-length messages from one processor to another.

In this design example, two AmPAL16R4 programmable-logic arrays serve as the interface to the Am29338, one each for the sending and receiving processors. The arrays serve as a conduit to examine the status of the FIFO buffer and also enable a programmable interrupt. In operation, the processor wishing to send a message to the mailbox calls a special operating-system routine. This routine first reads the status of the mailbox; if it is not full, the message is written. Then the routine returns to the calling process. If the mailbox is full, the operating-system routine blocks the calling process and enables interrupts from the mailbox. When a slot becomes available, the sending processor is interrupted. The interrupt routine sends the message, disables interrupts from the mailbox, and blocks the sending process. The receiving side of

the mailbox, of course, operates in an inverse manner.

From the practical standpoint, the state of the mailbox is first examined by asserting the Chip Select (CS), Read/Write (R/W) and Control/Data (C/D) lines of the appropriate PAL device and monitoring the buffer's Full flag. An interrupt enable can then be written by bringing the R/W line low. The actual message may be transmitted from the processor to the mailbox by bringing the PAL's CS and R/W lines low.

Conversely, messages from the mailbox are sent to the receiving end by asserting CS and R/W of the appropriate PAL device, and bringing its C/D line low. The mailbox status is examined by asserting CS, R/W and C/D. The interrupt-enable bit can be written by bringing CS and C/D high, and R/W low.

The mailbox, finally, can be extended to operate in a heterogeneous multiprocessing system. In that system, processes with both disparate data-block widths and clock frequencies are interconnected—an easy task for this FIFO buffer.

SYNCHRONOUS OR ASYNCHRONOUS OPERATION

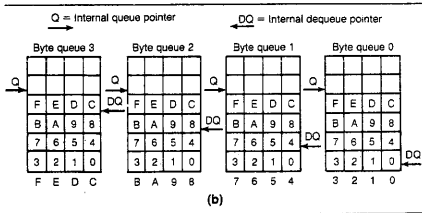
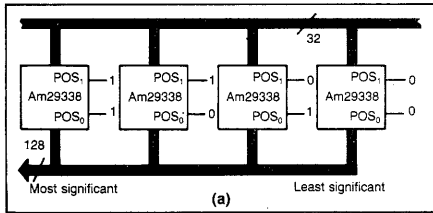
The Am29338 operates as most FIFO buffers do in the asynchronous mode, as well as in the synchronous mode. For the asynchronous mode, the Queue Clock input (QCLK) and DQCLK lines serve as strobes to queue or dequeue data and are generally independent of one another. As a result, the buffer can connect two asynchronous subsystems or to an asynchronous bus such as the VMEbus.

In a synchronous system, however, Enable signals are easier to generate than strobes. Thus, the QCLK and DQCLK signals may be simply derived from the common subsystem clock. Queueing and dequeuing may then be ordered with the Queue Enable (QEN) and Dequeue Enable (DQEN) inputs. This technique makes it easy to interface the buffer to a single subsystem or synchronous bus, such as Multibus II.

As long as the FIFO buffer is neither full nor empty, the rates at which data flows in and out of the buffer are independent of each other. The user stays abreast of the chip buffers' states by means of four status indicators: Full, Almost Full (A-Full), Empty, and Almost Empty (A-Empty). This is the role of the byte-count output.

Besides the basic flags such as Full and Empty for indicating chip state, the Am29338 supplies indicators to warn of the exact condition of its buffers. The A-Full and A-Empty outputs, for example, show that there are less than 4 bytes of space available, or more than 4 bytes of data in the buffer. These indicators, like Full and Empty, are valid only for synchronous operation.

Finer control over the amount of data stored is possible with the 7-bit Byte Count output, which monitors the number of bytes currently in the buffer. Unlike the other status indicators, Byte Count is valid only in the synchro-



4. Up to four FIFO buffers can be horizontally cascaded to support large word-width computer applications. Up to four devices can create one 128-bit word or a combination of 8-bit bytes (a). Buffers are combined by offsetting the internal queue and dequeue pointers.

DESIGN APPLICATION ■ Variable-width FIFO buffer

nous mode. In asynchronous operation, Byte Count is undefined.

An example of applying the Byte Count indicator is illustrated by its use in control tasks. For instance, various system devices may need some minimum amount of data on hand before a given function can be carried out. In this particular case, an external comparator informs the system that the required information is indeed in the buffer.

In all operations, the chip is first initialized by bringing the Reset line low. In tasks like instruction-prefetch queues, asserting Reset flushes the queue when a jump or branch instruction occurs. This action discards any prefetched instructions.

DATA-BIT MECHANICS

The number of bytes to be queued into the buffer is set by means of the Bytes Queued (BQ) inputs, and the corresponding data is presented to the data (D) and data parity (PD) inputs aligned to the least significant byte. When the QEN line is asserted, data will be entered on the falling edge of the QCLK input. The device's internal pointers will then be updated on the low-to-high transition of the clock.

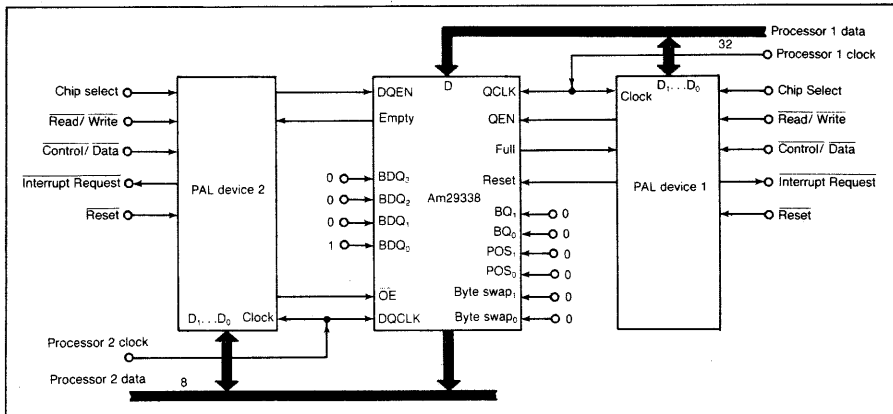
The number of bytes to be dequeued is determined by the Bytes Dequeued (BDQ) input. If the Dequeue Enable line (\overline{DQEN}) is brought low, the state of the byte queue is updated and data is off-loaded on the low-to-high transition of the \overline{DQCLK} signal.

When the Output Enable line (\overline{OE}) goes low, the next four bytes available for unloading and their corresponding parity bits are brought out on the data output (Y) and data parity (PY) lines. When \overline{OE} moves high, the D and PY pins assume a high-impedance state.

As mentioned earlier, the chip relies on byte-wise parity checking for error correction. Parity bits are checked at the input, stored with the data, and checked again at the output. Dual checking lends great flexibility to the error-checking operation. In an task involving an instruction-prefetch queue, for example, the designer may choose to check parity only at the output. Then, only executed instructions are checked. As a result, instructions that were prefetched but never used (such as those prefetched after a jump operation) will not cause spurious interrupts.

In typical operation, the data input parity-error output (PDERR) will go high if any of the bytes being queued have a parity error. The output parity-error line (PYERR) goes high if any of the bytes on the output bus have a parity error. Only valid bytes are checked for data anomalies; bytes on the data-input bus which are not being queued or undefined bytes which are sent out when the byte queue is almost empty are not included in the checking for errors. □

Tim Olson, a senior planning engineer at Advanced Micro Devices, is in charge of developing microprocessor architectures and Am29300 family building blocks. Olson has a BSEE-computer science degree from the University of Colorado at Boulder and an MSEE from the University of Arizona at Tucson.



5. Circuitry for a simple hardware mailbox needs only one Am29338 FIFO buffer and two programmable-logic arrays for links to transmit and receive controllers. Three signal lines collectively check chip status and control information flow: CS, R/W, and C/D. A fourth line (IREQ) indicates interrupt requests.

6.10 DIGITAL SYSTEMS VME 29300-1

Digital Systems offers the VME-29300-1, an Am29300-Family-based CPU, designed for those applications requiring the high performance of a 32-bit processor. Intended for use in emulating other computers or special-purpose computing such as graphics, encoding/decoding, and data reduction, the processor can be supplied with or without firmware. Its key features are:

- 100 ns per micro-instruction
- 4K words of Writable-Control-Storage
- 88-bit-wide microcode loaded from 27512 EPROM.
- On-board firmware address lights (single-stepping provided)
- N-way branching up to 64 ways
- 64 registers, 32 bits, 3-ported
- Calculated register address to 16-way
- Handles all seven interrupt levels
- Under firmware control: A16/A24/A32 and D8/D16/D32

Introduction

The VME-29300-1 CPU comes in a double-high two-board set. Both boards have P1 and P2 connectors for backplane connections, and in addition, control lines are interconnected between boards using two ribbon cables. The Instruction Board contains the Am29331 Sequencer, address read-out, microprogram memory, pipeline registers, and writable-control-storage circuitry.

The Arithmetic Board contains the Am29332 ALU, the Am29334 Register File, the calculation registers and latches, the constants ROM, and the address and data I/O circuitry. Board positions and spacing within the VME rack can be customized.

Am29331—Microprogram Sequencer

The Am29331 chip is configured as a 12-bit microprogram sequencer. The sequencer has multiway branch instructions that allow 1-of-N consecutive addresses to be selected as the branch target in a single cycle. The N-way branching can be chosen as 4-way, 8-way, 16-way, or 64-way by the microcode. Combinations of M, A, and D input lines of the Am29331 are used for this choice. A stack within the sequencer stores return addresses, loop addresses, and loop counts. It has 33 levels to permit the deep nesting of subroutines and loops. The lower 12 output lines address the 4096-word microprogram memory, each word of which has a width of 88 bits. (The upper 4 address bits are not used.) Output data from the memory are fed to the pipeline registers.

Writable-Control-Storage

The Writable-Control-Storage (WCS) circuitry consists of a 27512 EPROM and the associated circuitry to control loading. At power-on time, the loader brings the microprogram into the 4Kx88 random-access memory, stepping the Am29331 sequencer through a series of addresses. Then each word of the microprogram is checked back against the EPROM bit pattern. When this task is complete, the WCS loader is disabled and the sequencer takes control. For debugging purposes the microprogram can be single-stepped, and the WCS loader again controls the Am29331 sequencer. The address readout displays each address (in a readable fashion) during single-stepping.

Am29334—Register File

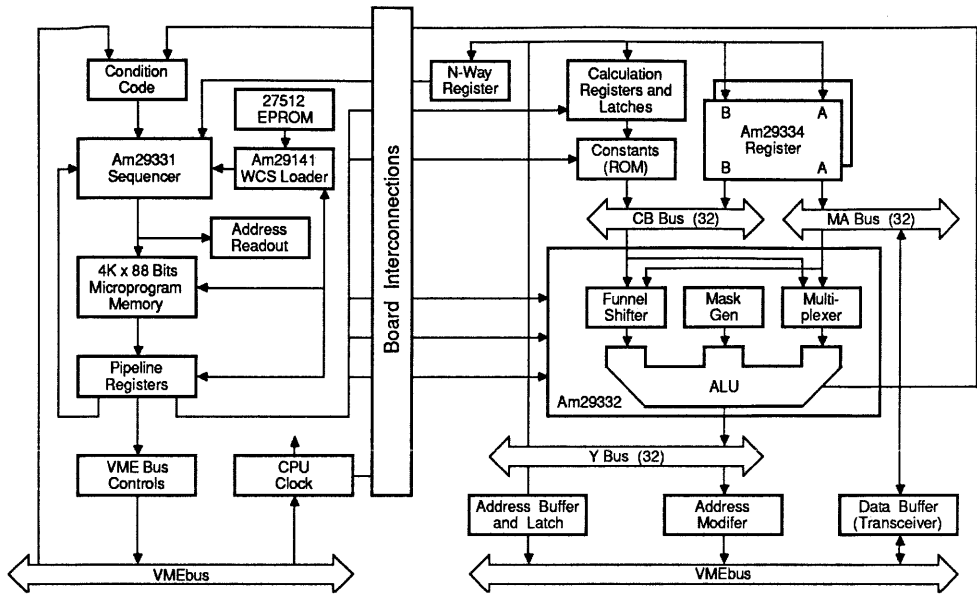
The two Am29334 chips serve as a 64x32 external register file for the ALU. Each of these is a high-speed, random-access memory configured with one write port (D) and two read ports (A,B). The D port is fed from the 32-bit wide Y bus, while the A port feeds the MA bus and the B port feeds the CB bus. Control of write operations is done with the common write enable to each chip. This allows the lower-16 or upper-16 bits to be stored separately and gives the four different write options:

- Write no data at all
- Write only the lower 16 bits
- Write only the upper 16 bits
- Write all 32 bits simultaneously

Read operations are controlled by a common output enable for reading all 32 bits to the A or B port. The A address bus originates in the writable control store (WCS) while the B and D address buses originate in the address calculation circuitry. By calculating the B and D addresses the CPU achieves a high degree of microprogram flexibility.

Am29332—ALU

The Arithmetic Logic Unit (ALU) processes 32-bit-wide data paths. This means that it allows one-, two-, three-, or four-byte data in arithmetic and logic operations as well as multiprecision arithmetic and multiple-bit shift operations. The data flow uses two input buses, MA and CB, and one output bus, Y. Operation on data of variable byte length, variable-length bit fields, or even single bits is made possible by the internal mask generator. This circuit creates a 32-bit mask for each instruction while using no overhead time. The mask is used as an additional operand in each instruction to allow operation on the selected data widths. Instructions that operate on variable-length bit fields require a mask that is a contiguous string of 1s for all selected bit positions and 0s for all



unselected bit positions. In cases where the field exceeds the 32-bit boundary, the mask does not wrap around, allowing operation on a contiguous field across a word boundary.

For most single-operand instructions, the unselected bit positions pass the corresponding bits of the operand unmodified. For most two-operand instructions, the unselected bit positions pass the corresponding bits of the operand unmodified on the CB input. Thus, for two-operand instructions the mask allows the merging of the two operands in a single cycle. In addition to being used internally, the mask can be sent over the Y bus as a pattern for testing purposes.

The Am29332 uses a funnel shifter with two 32-bit input ports and one 32-bit output port. This circuit can perform all of the operations of a barrel shifter (one N-bit input port and one N-bit output port) extended to two operands instead of one. Such a circuit is used to shift or rotate the operand up or down from 0 to 32 bits in a single cycle. This is very useful in operations such as the normalization of a mantissa for floating-point arithmetic or in applications where the packing and unpacking of data are frequent operations. In addition, it can extract a 32-bit contiguous field across the two operands, a function which is very useful in some graphics applications. Also, any of its operations can be followed by a logical operation with both completed in a single cycle.

The Am29332 easily handles prioritization which is useful in controlling N-way branches, performing normalizations, and in graphic operations such as polygon fills. The built-in priority encoder sends out a 5-bit binary weighted code that signifies the relative position of the most significant 1 of the byte width selected. This allows prioritization on either 8-, 16-, 24-, or 32-bit operands. The priority encoder output can be passed on to the Y bus or stored in the status register.

The Complete VME-29300-1

The VME-29300-1 is a complete 32-bit processor when firmware is in place. It will operate on the VMEbus as a master or an interrupt-handler. Since it is not a fixed-instruction-set processor, firmware must be designed for proper operation. However, this is its outstanding advantage over other processors. Firmware options are almost limitless, giving the processor its high degree of adaptability to virtually any computing job. Chief among the suitable applications of this CPU is its ability to emulate other computing systems. This capability is not limited to 32-bit processors, of course. Eight-bit and 16-bit systems are also easily emulated. Other complex computing jobs are also possible such as reducing large amounts of data and executing graphics programs.

Digital Systems will design the firmware and deliver it with your system or provide design advice at an hourly rate by phone call or site visit.

12-bit Microprogram Sequencer

- Provides 100-ns microcycle time to support 32-bit high performance system
- Supports 4-way, 8-way, 16-way, and 64-way branching chosen by the microcode
- Contains built-in conditional test logic for use with the ALU status bits
- A 33-level stack provides support for loops and subroutine nesting
- Supports single-stepping for the purpose of debugging
- 12-bit address readout provided

Microprogram Memory

- Provides 4096-word capacity with a word width of 88 bits of writable-control-storage
- A 27512 EPROM allows customized firmware to be easily replaced or modified

Register File

- Two cascaded high-speed RAM chips for 64x32-bit register capacity
- Write control allows independent lower-16 or upper-16 bits of storage
- Provides one WRITE port (D) and two READ ports (A, B) and four WRITE options
- Calculated B and D addresses provide high degree of microprogram flexibility

ALU

- A combinatorial architecture with equal cycle time for all instructions, two input ports, and one output port
- Funnel shifter allows N-bit shift-up, shift-down, 32-bit barrel shift or 32-bit field extract
- Supports one-, two-, three-, and four-byte data for all operations and variable length fields for logical operations

VME Characteristics

- Double-high, two-board set occupies 4 slots
- Power requirements: +5 VDC @ 3 A (max), +12 VDC @ 0 A, -12 VDC @ 0 A
- Operating range: 0-70°C, 80% relative humidity, forced cooling required
- Interrupt handler options: 1-7
- Requester option: R(3) used
- Master data transfer options: A16/A24/A32 and D8/D16/D32

Additional information is available upon request from:
Digital Systems Corporation
3 North Main Street
Walkersville, MD 21793
(301)845-4141

6.11 BIBLIOGRAPHY

- Ajmera, Dhaval, Ole Møller and David Sorenson. "Bipolar Building Blocks Deliver Supermini Speed to Microcoded Systems," *Electronic Design*, November 15, 1984.
- Baker, Stan. "AMD: 1st 32-Bit Bipolar Microprocessor Line," *Electronic Engineering Times*, November 12, 1984.
- Barney, Clifford. "32-Bit Chip Integrates Bit-Slice Functions," *Electronics Week*, November 12, 1984.
- Campbell, Steve, Timothy Flaherty. "Floating-Point Trends in Digital-Signal Processing Systems," *Electronic Engineering Times*, September 2, 1985.
- Case, Brian. "Building Blocks Yield Fast 32-Bit RISC Machines," *Computer Design*, July 1, 1985.
- Chu, Paul. "What is Microprogramming?" *Electronic Products*, November 3, 1986.
- Chu, Paul, Bernard New. "Microprogrammable Chips Blend Top Performance with 32-Bit Structures," *Electronic Design*, November 15, 1984.
- DiDio, Laura. "AMD Unveils Bipolar 32-Bit MPU," *Electronic Buyer's News*, November 12, 1984.
- Flaherty, Timothy. "Building Blocks Stack Up to High Performance," *Computer Design*, February 1, 1985.
- Freidin, Phillip. "Building 32-Bit Systems with Microprogram Blocks," *Electronic Engineering Times*, March 17, 1986.
- Freidin, Phillip. "Microcodable Blocks Yield Flexible Systems," *Electronic Engineering Times*, July 7, 1986.
- Mendelsohn, Alex. "32-Bit Bipolar Building Blocks Debut at AMD," *Integrated Circuits*, November 1984.
- Pertman, Robert. "Improve your Number-Crunching Powers with a Floating Point....," *EDN*, January 23, 1986.
- Pertman, Robert, Dave Quong. "Single-Chip Accelerators Speed Floating-Point and Binary Computations," *Electronic Design*, November 15, 1984.
- Quong, Dave. "Using a Floating-Point Processor to Implement High Speed....," *EDN*, February 6, 1986.
- Wilson, Dave. "32-Bit ICs Enhance Array Processor Performance," *Digital Design*, November 12, 1984.
- "Advanced Micro Devices Offers 32-Bit Bipolar MPU Family," *Electronic News*, November 12, 1984.
- "First 32-Bit Bipolar CPU," *Engineering Manager*, December 1984.
- "Microprocessors Soar to New Performance Dimensions," *EDN*, November 15, 1984.
- "32-Bit Floating-Point IC Heralds Appearance of High-Performance Family," *Electronic Products*, February 15, 1985.

CHAPTER 7

Technical Information

7.1 THE Am29300/29C300 TIMING ANALYSIS	7-1
7.2 THERMAL CHARACTERISTICS/AIR FLOW	7-6
7.3 CMOS/BIPOLAR RELIABILITY	7-10
7.4 CMOS LATCH-UP TEST METHODS AND RESULTS	7-17
7.5 TEST PHILOSOPHY AND METHODS	7-18

CHAPTER 7

Technical Information



7.1 THE Am29300/29C300 TIMING ANALYSIS

With the Am29300, you can construct a system with a family cycle time of 80 ns or faster. This is especially true with the Am29300A. This section discusses the various critical paths in determining the fastest family cycle time. The following systems configuration was assumed:

Control Path

Am29331/29C331	16-bit Microprogram Sequencer
Am29818A	Pipeline Register
Am99C68	Control Memory
Am27S55A	Registered PROM

Data Path

Am29332/29C332	32-Bit ALU
Am29334/29C334	68 x 18 Dual Port Register File
Am29818A	Status Register

Non-Pipelined Operation

The block diagram surrounding the Am29300/29C300 family is shown in Figure 7-1 and its critical timing analysis is described in Tables 7-1 and 7-2. This timing analysis shows that a system cycle time of 75 ns is possible with the Am29300/29300A family, and 90 ns is possible with the Am29C300/29C300-1 family. The summary of the performance is listed in Table 7-5.

Pipelined Operation

With the two pipelined stages in the Am29C334 (PIPE=HIGH), you can construct the pipelined systems with the Am29C300. As an example for this operation, the following describes a double-pipelined system. In this example, the Am27S55A, the registered PROM is utilized to improve the control path. Figure 7-2 shows an example of the pipelined system.

Writing the Data into the Register File

It takes two cycles to write data into the register file. In the first cycle, the data from the main memory is latched into the input pipeline register. Then in the second cycle, the data is written into the RAM location in the Am29C334. (See cycle 1-2 in Table 7-3.)

Data Calculation and Storage

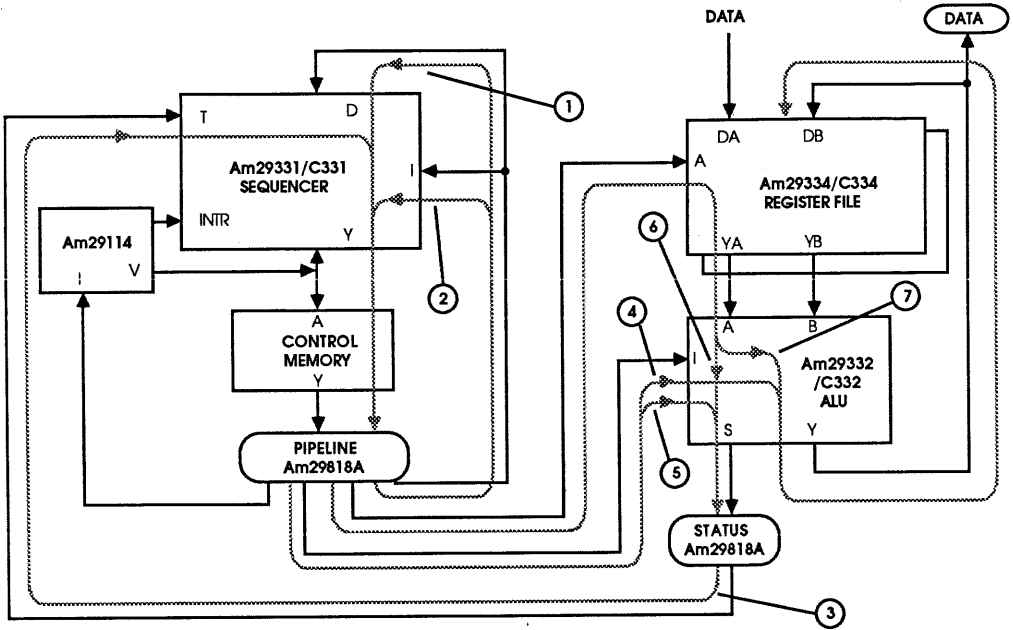
In the first cycle, data (A1) to be operated upon is latched from the RAM location onto the output pipeline register of the Am29C334. In the second cycle, the operation is performed on the data (A1,B1) by the Am29C332. The result (C1) is then set up on the input pipeline register of the Am29C334. In the last cycle, the result is written into the RAM location of the Am29C334. For an example, refer to cycle 3-6 of Table 7-3.

The second of the path cycles is the most critical of the three. The maximum propagation delay incurred on this timing then has to be compared with the maximum control path timing. The cycle time is determined by the longest of the two. The speed and choice of the main memory has to be based on the cycle time.

It is possible to time-share the above two operations. In other words, data can be written into the register file at the same time the operation is performed on the data from the register file. See Table 7-3 for an example.

Table 7-4 shows the calculation of the pipelined Am29C300 system. As you notice, testing of the ALU status through the Am29C331 is critical for the control path, and the data path involving I-Y of the Am29C332 is also critical. The table shows that the data path determines the cycle time. The result is shown in Table 7-5.

It is quite possible to improve the cycle time further with combinations of the Am29300, Am29300A, Am29C300, and Am29C300-1.



08902A-650

Figure 7-1. Am29300/29C300 System Timing Analysis

Table 7-1. Bipolar Am29300 Timing Analysis

Loop	Device		Path	Am29300	Am29300A ³
1	Am27S55A ¹	Pipeline Reg.	CP-Q	10	10
	Am29331	Sequencer	D-Y	19	17
	Am27S55A	RPROM	A-Q	<u>20</u>	<u>20</u>
	Total:			49	47
2	Am27S55A	Pipeline Reg.	CP-Q	10	10
	Am29331	Sequencer	I-Y	25	22
	Am27S55A	RPROM	A-Q	<u>20</u>	<u>20</u>
	Total:			55	52
3	Am29818A ²	Status Register	CP-Q	11	11
	Am29331	Sequencer	T-Y	25	22
	Am27S55A	RPROM	A-Q	<u>20</u>	<u>20</u>
	Total:			56	53
4	Am27S55A	Pipeline Reg.	CP-Q	10	10
	Am29332	ALU	I-Y	47	40
	Am29334	Reg. File	D-CP	<u>9</u>	<u>9</u>
	Total:			66	59
5	Am27S55A	Pipeline Reg.	CP-Q	10	10
	Am29332	ALU	I-C,Z,N,L	48	41
	Am29818A	Status Reg.	Y-CP	<u>6</u>	<u>6</u>
	Total:			64	57
6	Am27S55A	Pipeline Reg.	CP-Q	10	10
	Am29334	Reg. File	A-Y	24	24
	Am29332	ALU	D-C,Z,N,L	43	37
	Am29818A	Status Reg.	D-CP	<u>6</u>	<u>6</u>
	Total:			83	77
7	Am27S55A	Pipeline Reg.	CP-Q	10	10
	Am29334	Reg. File	A-Y	24	24
	Am29332	ALU	D-Y	35	30
	Am29334	Reg. File	D-CP	<u>9</u>	<u>9</u>
	Total:			78	73

Note: 1. In this timing analysis, a registered PROM is used to store microcodes. WCS can be also implemented as replacement for the registered PROM.

2. The specifications can be improved by choices of the pipeline registers.

3. This is only applicable for the Am29331A and the Am29332A.

Table 7-2. CMOS Am29C300 Timing Analysis (Non-pipelined Mode)

Loop	Device		Path	Am29C300	Am29C300-1
1	Am29818A ²	Pipeline Reg.	CP-Y	11	11
	Am29C331	Sequencer	D-Y	22	20
	Am99C68 ¹	WCS	A-Y	40	40
	Am29818A	Pipeline Reg.	D-CP	<u>6</u>	<u>6</u>
	Total:			79	77
2	Am29818A	Pipeline Reg.	CP-Q	11	11
	Am29C331	Sequencer	I-Y	24	22
	Am99C68	WCS	A-Y	40	40
	Am29818A	Pipeline Reg.	D-CP	<u>6</u>	<u>6</u>
	Total:			81	79
3	Am29818A	Status Reg.	CP-Q	11	11
	Am29C331	Sequencer	T-Y	24	22
	Am99C68	WCS	A-Y	40	40
	Am29818A	Pipeline Reg.	D-CP	<u>6</u>	<u>6</u>
	Total:			81	79
4	Am29818A	Pipeline Reg.	CP-Q	11	11
	Am29C332	ALU	I-Y	66	47
	Am29C334	Reg. File	D-CP	<u>15</u>	<u>13</u>
	Total:			92	71
5	Am29818A	Pipeline Reg.	CP-Q	11	11
	Am29C332	ALU	I-C,Z,N,L	67	48
	Am29818A	Status Reg.	Y-CP	<u>6</u>	<u>6</u>
	Total:			84	65
6	Am29818A	Pipeline Reg.	CP-Q	11	11
	Am29C334	Reg. File	A-Y	32	26
	Am29C332	ALU	D-C,Z,N,L	60	43
	Am29818A	Status Reg.	D-CP	<u>6</u>	<u>6</u>
	Total:			109	86
7	Am29818A	Pipeline Reg.	CP-Q	11	11
	Am29C334	Reg. File	A-Y	32	26
	Am29C332	ALU	D-Y	49	35
	Am29C334	Reg. File	D-CP	<u>15</u>	<u>13</u>
	Total:			107	85

- Notes:**
1. WCS is used to store microcodes. The registered PROM can be utilized as a replacement for the WCS.
 2. The specifications can be improved by choices of the pipeline register.
 3. An external register is used to store status output of the ALU. If the internal status register is used, the cycle time will be faster by eliminating the setup time of the external register.

Table 7-3. Pipelined Timing Sequence (Data Path)

Cycle	1	2	3	4	5	6
Am29C334 I/P	A1'	A2	A3	A4	A5/C1	A6/C2
RAM (write)		A1	A2	A3	A4	A5/C1
RAM (read)		A1/B1 ²	A2/B2	A3/B3	A4/B4	A5/B5
O/P			A1/B1	A2/B2	A3/B3	A4/B4
Am29C332 ALU				C1	C2	C3

Legend: I/P = Input Pipeline Register
 O/P = Output Pipeline Register
 Ci = Ai op Bi (op = Am29C332 Operation)

Note: 1. For example, A1/B1 stands for (data derived from A port)/(data derived from B port).
 2. Assumption is made that data Bi is already stored in the Am29C334.

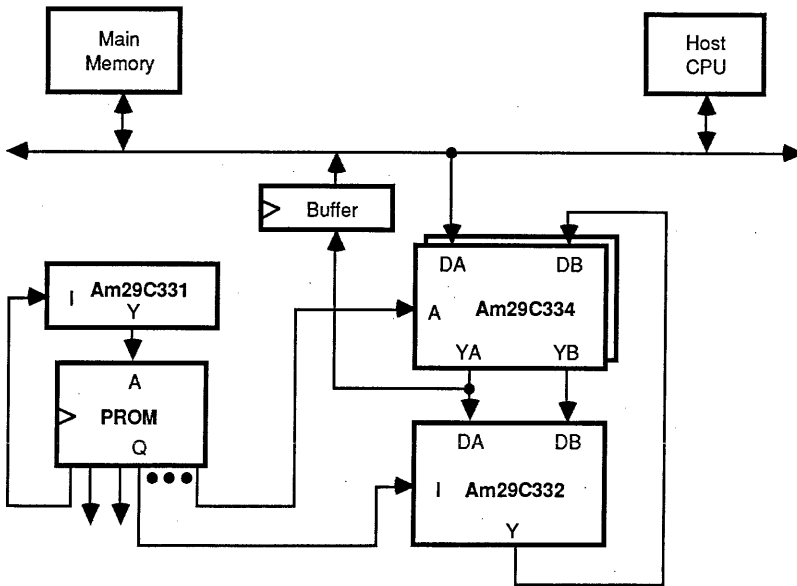


Figure 7-2. Block Diagram

Table 7-4. Pipelined Cycle Time Calculation

Control Path				Data Path			
		Am29C300	Am29C300-1			Am29C300	Am29C300-1
Am29818A	CP-Q	11	11	Am29818A	CP-Q	11	11
Am29C331	T-Y	24	22	Am29C332	I-Y	66	47
Am27S55A	Add. Setup	20	20	Am29C334	D-CP	<u>15</u>	<u>13</u>
Am29818A	D-CP	<u>6</u>	<u>6</u>	Total:		92	71
Total:		61	59				

Table 7-5. Am29300/29C300 Family Cycle Time (ns)

	Am29300	Am29300A	Am29C300	Am29C300-1
Non-Pipelined	83	77	109	86
Pipelined	N/A	N/A	92	71

7.2 THERMAL CHARACTERISTICS/ AIR FLOW

DEFINITION OF THERMAL RESISTANCE

The reliability of an integrated circuit is largely dependent on the maximum temperature which the device will attain during operation. Because the stability of a semiconductor junction declines with increasing temperature, knowledge of the thermal properties of the packaged device becomes an important factor during device design. In order to increase the operating lifetime of a given device, the junction temperatures must be minimized. This demands knowledge of the thermal resistance of the completed assembly and specification of the conditions in which the device will function properly. As devices become both smaller and more complex and the requirement for high speed operation becomes more important, heat dissipation will become an ever more critical parameter.

Thermal resistance is defined as the temperature rise per unit power dissipation above some referenced condition. The unit

of measure is typically °C/watt. The relationship between junction temperature and thermal resistance is given by:

$$T_J = T_x + P_D \theta_{JX} \quad (1)$$

- where: T_J = junction temperature
- T_x = reference temperature
- P_D = power dissipation
- θ_{JX} = thermal resistance
- X = some defined test condition

In general, one of three conditions is defined for measurement of thermal resistance:

- θ_{JC} - thermal resistance measured with reference to the temperature at some specified point on the package surface.
- θ_{JA} (still air) - thermal resistance measured with respect to the temperature of a specified volume of still air.
- θ_{JA} (moving air) - thermal resistance measured with respect to the temperature of air moving at a specified velocity.

The relationship between θ_{JC} and θ_{JA} is

$$\theta_{JA} = \theta_{JC} + \theta_{CA}$$

where θ_{CA} is a measure of the heat dissipation due to natural convection (still air) or forced convection (moving air) and the effect of heat radiation and mounting techniques. θ_{JC} is dependent solely on material properties and package geometry; θ_{JA} includes the influence of the surface area of the package and environmental conditions. Each of these definitions of thermal resistance is an attempt to simulate some manner in which the package device may be used.

The thermal resistance of a packaged device, however measured, is a summation of the thermal resistances of the individual components of the assembly. These in turn are functions of the thermal conductivity of the component materials and the geometry of the heat flow paths. Like other material properties, thermal conductivity is usually temperature dependent. For alumina and silicon, two common package materials, this dependence can amount to a 30% variation in thermal conductivity over the operating temperature range of the device. The thermal resistance of a component is given by

$$\theta = \frac{L}{K(T)A} \quad (2)$$

where: L = length of the heat flow path
 A = cross sectional area of the heat flow path
 K(T) = thermal conductivity as a function of temperature

and the overall thermal resistance of the assembly (discounting convective effects) will be:

$$\theta = \sum \theta_n = \sum \frac{L_n}{K_n A_n}$$

but since the heat flow path through a component is influenced by the materials surrounding it, determination of L and A is not always straightforward.

A second factor that affects the thermal resistance of a packaged device is the power dissipation level and, more particularly, the relationship between power level and die geometry, i.e., power distribution and power density. By rearrangement of equation 1 to

$$P_d = \frac{1}{\theta_{JX}} (T_J - T_X) = \frac{1}{\sum \theta_N} (T_J - T_X) \quad (3)$$

the relationship between P_d and T_J can be more clearly seen. Thus, to dissipate a greater quantity of heat for a given geometry, T_J must increase and, since the individual θ_n will also increase with temperature, the increase in T_J will not be a linear function of increasing power levels.

A third factor of concern is the quality of the material interfaces. In terms of package construction, this relates specifically to the die attach bond, and for those packages having a heatsink, the heatsink attach bond. The quality of the die attach bond will most severely influence the package thermal resistance as this is the area which first impedes the transfer of heat out of the silicon die. Indeed, it seems likely that the initial thermal response of a powered device can be directly related to the quality of the die attach bond.

EXPERIMENTAL METHOD

The technique for measurement of thermal resistance involves the identification of a temperature-sensitive parameter on the device and monitoring this parameter while the device is powered. For bipolar integrated circuits the forward voltage of the substrate isolation diode provides a convenient parameter to measure and has the advantage of a linear dependence on temperature. MOS devices which do not have an accessible substrate diode present greater measurement difficulties and may require simulation through use of a specially designed thermal test die. Choice of the parameter to be measured must be made with some care to ensure that the results of the measurement are truly representative of the thermal state of the device being investigated. Thus measurement of the substrate isolation diode which is generally diffused across the area of the die yields a weighted average of the condition of the individual junctions across the die surface. Measurement of a more local source would yield a less generalized result.

For MOS devices, simulation is accomplished using the thermal test die. The basis for this test die is a 25 mil square cell containing an isolated diode and a 1 K Ω resistor. The resistors are interconnected from cell to cell on the wafer before it is cut into multiple arrays of the basic unit cell. In use the device is powered via the resistors with voltage or current adjusted for the proper level and the voltage drop of the individual diodes is monitored as in the case of actual devices.

Prior to the thermal resistance test, the diode voltage/temperature calibration must be determined. This is done by measuring the forward voltage at 1 mA current level at two different temperatures. The diode calibration factor is then:

$$K_T = \frac{T_2 - T_1}{V_2 - V_1} = \frac{\Delta T}{\Delta V} \quad (4)$$

in units of $^{\circ}\text{C}/\text{mV}$. For most diodes used for this test the voltage/temperature relationship is linear and these two measurement points are sufficient to determine the calibration.

The actual thermal resistance measurement has two alternating phases: measurement and power on. The device under test is pulse powered with an ON duty cycle of 99% and a repetition rate of < 100 Hz. During the brief OFF states the device is reverse-biased with a 1 mA current and the voltage drop is measured. The series of voltage readings are averaged over short periods and compared to the voltage reading obtained before the device was first powered ON. The thermal resistance is then computed as:

$$\theta_{JK} = \frac{K_F(V_F - V_I)}{V_{HH}} = \frac{K_T \Delta V}{P_D} \quad (5)$$

where: K_F = calibration factor
 V_I = initial forward voltage value
 V_F = current forward voltage value
 V_H = heating voltage
 I_H = heating current

CHAPTER 7
Technical Information

The pulsing measurement is continued until the device has reached thermal equilibrium and the final value measured is the equilibrium thermal resistance of the device under test.

When the end result desired is θ_{JA} (still air), the device and the test fixture (typically a standard burn-in socket) are enclosed in a box containing approximately 1 cubic foot of air. For θ_{JC} measurements the device is attached to a large metal

heatsink. This ensures that the reference point on the device surface is maintained at a constant temperature. The requirements for measurement of θ_{JA} (moving air) are rather more complex and involve the use of a small wind tunnel with capability for monitoring air pressure, temperature and velocity in the area immediately surrounding the device tested. Standardization of this last test requires much careful attention.

WAVEFORMS FOR PULSED THERMAL RESISTANCE TEST

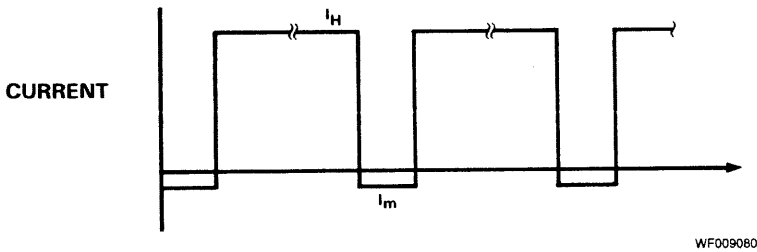
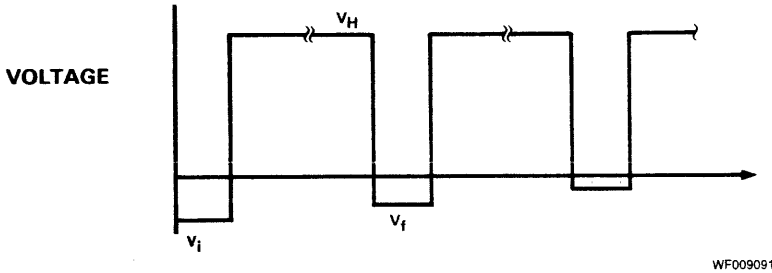
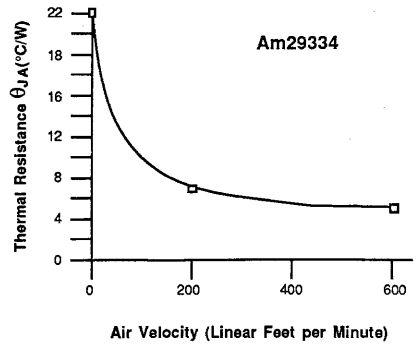
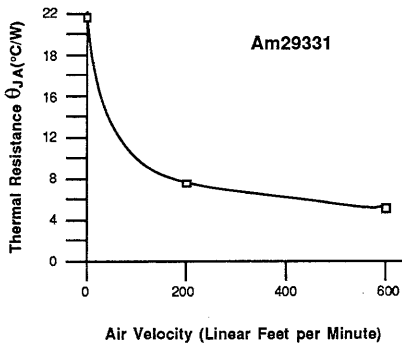
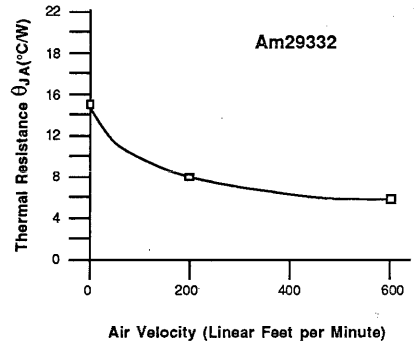
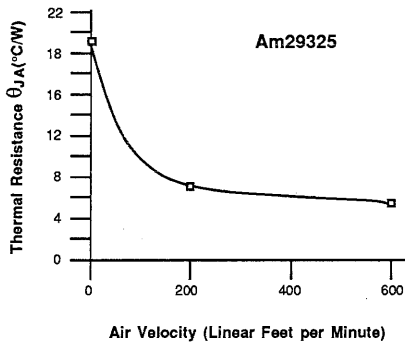


Table 7-6. Am29300 Thermal Resistance ($^{\circ}\text{C}/\text{W}$)¹

	Am29325GC	Am29331GC	Am29332GC	Am29334GC
θ_{JA} , Junction-to-Ambient, Still Air	19.0	21.8	15.0	22.0
θ_{JA} , 200 Linear Feet per Minute	7.0	7.7	8.0	7.5
θ_{JA} , 600 Linear Feet per Minute	5.5	5.1	6.0	5.0
θ_{JC} , Junction-to-Case ²	2.5	2.5	2.5	2.5

- Notes: 1. The air flow should be measured at the vicinity of the heatsink.
2. This is the measured value based on a 144-pin PGA with heatsink attached. The value should not vary significantly over the family.



7.3 CMOS/BIPOLAR RELIABILITY

Reliability Monitor Program

AMD Specification 01-011

The Reliability Monitor Program (RMP) is an extensive effort to measure the reliability of all process families at AMD on a regular basis. Typically 7,000 to 10,000 devices per month are tested in a variety of environmental stresses.

The Reliability Monitor Program has two purposes:

Improved Reliability Performance: Each reject found undergoes failure analysis. Results are used by AMD to identify and establish corrective actions to eliminate failure mechanisms.

Generation of Reliability Data: Reliability results are utilized in many ways. Typical applications include assessing the benefits of burn-in, providing estimates of typical lifetimes, modeling field applications, and determining suitability of plastic and hermetic packaging in various temperature and humidity environments. This information is available to the customer.

The stress tests employed are listed in Table 2:

Table 2. Reliability Monitor Stress Conditions

STRESS	DURATION	SAMPLE SIZE	CONDITIONS	
			HERMETIC	PLASTIC
Early Life	160 hours	300	125°C	125°C or 85°C
Operating Life	1000 hours	120	150°C and 125°C	125°C or 85°C
Extended Operating Life (Biannual)	2000 hours	120	150°C and 125°C	125°C or 85°C
Temperature Cycle	1000 cycles	50	-65°C to 150°C	-65°C to 150°C
Biased Temperature and Humidity	1000 hours	50	N/A	85°C & 85% RH 5v alt bias
Pressure Cooker	160 hours	50	N/A	121°C, 15 psig, no bias

The results from the Reliability Monitor Program form the basis of the failure rate calculations presented in the appendix.

The Estimation of Field Reliability

In this section, a modeling procedure is described for estimating reliability under field conditions, based on the lifestest data generated in the Reliability Monitor Program. The summaries of the lifestest results and the actual failure-rate projections are contained in the appendix.

A General Reliability Model

In order to evaluate the reliability of the product in the field, a general reliability model is utilized. The modeling procedure is described by authors Paul A. Tobias and David C. Trindade in the text **Applied Reliability** (New York: Van Nostrand Reinhold, 1986, pp. 173-182).

The failure probability $F(t)$ may be viewed as the probability that a random unit drawn from the population fails by time t . Thus, $F(t)$ may be represented in terms of a cumulative distribution function (CDF) of the times to failure.

To understand the general reliability model it is useful to think of failures in terms of the three D's: dead, defective, or deficient. The general model encompasses (1) the discovery of functionally dead test escapes, (2) the defective subpopulations, and (3) the typical competing failure modes of the main population, which are typically indicative of design, material, or process deficiencies.

The complete model for the field use CDF may be represented as:

$$F_T = \alpha F_e + \beta F_d + (1 - \alpha - \beta) F_N$$

where F_e is the discovery distribution for the proportion α of test escapes, F_d is the life distribution for the proportion β of units in the defective subpopulations, and F_N is the life distribution derived from the N typical competing failures modes.

For F_N , the competing nature arises because a unit is viewed as a series system of different potential failure mechanisms such that the occurrence of any one failure mechanism results in failure of the unit. Thus, $F_N = 1 - R_1 R_2 R_3 \dots R_N$, where R_i is the reliability function for a specific failure mechanism. For the series model, failure rates at any point in time are additive.

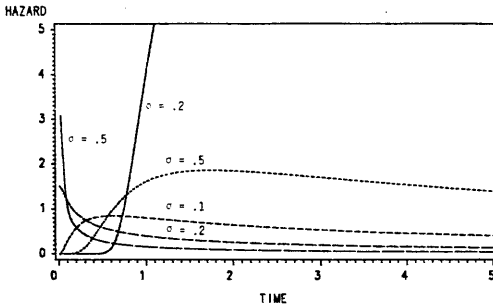
The distribution for the test escapes is not an actual life distribution, but describes the application dependent rate at which the escapes may be discovered in use. This category also includes good units damaged in test or handling.

Failure Distributions

The lognormal and Weibull CDF's are the distributions most often used to represent reliability failure mechanisms. The exponential distribution, characterized by a constant failure rate, is a special case of the Weibull. The lognormal distribution is specified by two parameters: T_{50} , the median time to failure, and sigma, the shape parameter. Similarly, the Weibull distribution, which can be written in closed form as $F(t) = 1 - \exp[-(t/c)^m]$, is characterized by a characteristic life c and a shape parameter m . The value of the shape parameter determines whether the failure rate is increasing ($m > 1$), decreasing ($m < 1$), or constant ($m = 1$). The exponential distribution, $F(t) = 1 - \exp[-(t/c)]$, is specified completely by the one parameter c called the mean time to failure (MTTF). Figures below show failure rates for several values of the scale parameters of the lognormal and Weibull distributions, respectively.

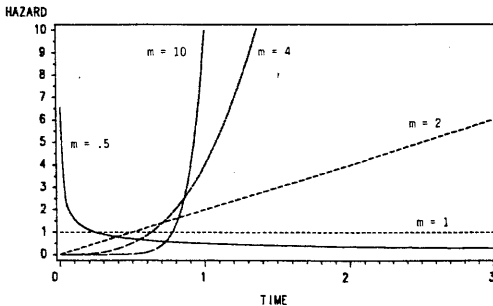
Lognormal Failure Rate (Hazard)

($T_{50} = 1$)



Weibull Failure Rate (Hazard)

(Characteristic Life = 1)



For the general reliability model to be applied, the distributions and associated parameters must be determined, either through reliability studies or a review of the reliability literature. In addition, if the experimentation is performed under accelerated conditions, acceleration models are needed to relate the results to field use. For distributions such as the lognormal or Weibull, acceleration factors are applied to the scale parameter (such as the median or characteristic life, respectively), in order to generate a new scale parameter from which failure rates at various field conditions may be estimated. Under true linear acceleration, the type of distribution and the shape parameter do not change between stress and field conditions.

Calculation of Failure Rates

To estimate field failure rates from reliability studies, many factors must be considered. One primary requirement is the identification of individual failure mechanisms in order to ascribe the failures to the proper categories used in the general reliability model.

Considerations and Assumptions

1. The fraction of test escapes and the underlying discovery distribution:

The fraction of test escapes and contributions from damage occurring as a result of testing and handling procedures at the vendor or customer are estimable only from actual field usage, since the underlying discovery distribution is application dependent. To model these test escapes, a Weibull distribution with a decreasing failure rate may be used. In the appendix, test escapes, which represent an unknown early adder to the model, are assumed negligible. Temperature acceleration considerations do not apply to test escapes since the units are basically inoperative.

2. The fraction of defective subpopulations and the underlying distribution:

The lifetimes for the fraction defective subpopulations may be modeled by the exponential distribution. Reliability results from stress testing must be carefully analyzed in order to identify the true defect related failure modes. From such studies at AMD, the mean time to failure (MTTF) for the defective subpopulations has been found to be approximately 100 hours at 125°C. The fraction β of product with defects is computed from the CDF estimate of defect related failures at readout time t by the following equation:

$$\beta = \text{CDF} / (1 - e^{-t/100}).$$

To combine the results from lifetests at different temperatures or from dissimilar readout times, a pooled estimate of β may be calculated as the weighted mean of the individual β estimates. Sample size is the weighting factor. Based on the reliability literature, an activation energy of 0.45 eV has been chosen as representative.

3. The distributions of the competing failure mechanisms in the main population:

Competing failure mechanisms may occur during either early fail or long term lifetesting. The distribution of lifetimes is modeled by a lognormal distribution with a sigma specific to each failure mechanism. The sigma value may be determined from the reliability literature and checked for reasonableness against values estimated from the data. Also from the reliability data giving the fraction failed for various mechanisms at stress readouts, the median time to fail (T_{50}) at stress conditions may be estimated. To combine the results for a specific mechanism from several lifetests, a pooled median time to fail, weighted by sample size, is computed from the individual $\ln T_{50}$ estimates.

The acceleration factors specific to a failure mechanism may be applied to the pooled stress T_{50} to estimate the field T_{50} . This field median life estimate may then be used with the same sigma to estimate the expected CDF in the field for a given mechanism at a chosen time. The individual failure rates for each mechanism may be summed to arrive at the total device failure rate.

4. The treatment of zero rejects for a possible failure mechanism:

Just because failures for a given mechanism are not observed does not mean such mechanisms are non-existent. The sample size may be insufficient or the acceleration may be inadequate to reveal all possible low level reliability concerns. In fact, if the potential failure mecha-

nisms have low thermal activation energies, the demonstration of reliability performance may be limited by mechanisms with no observed failures!

For example, time dependent dielectric breakdown (TDDB) for MOS devices has a lognormal distribution with sigma around 5.5 and activation energy of 0.3 eV. If no TDDB failures are observed in a HTOL stress, it is still possible to calculate a non-zero, upper confidence level for the CDF based on the given sample size. The use of such a low activation energy may be a significant factor when combining failure rates across all possible mechanisms having higher activation energies.

5. The incorporation of unknown failure mechanisms:

Another significant factor in calculating failure rates is the manner in which unidentified mechanisms are incorporated into the failure rate calculations. If the failure mechanism is unknown, the rejects may be pooled into a category that uses fairly conservative activation energies of 0.3 eV for MOS and 0.5 eV for bipolar. Even though failure mechanisms are unidentified, it may still be possible to estimate the lognormal sigmas from the data.

6. Overall activation energies and the exponential distribution.

In the reliability literature, it is common to see the use of **overall** activation energies, such as 0.7 eV for MOS and 1.0 eV for bipolar technologies. In addition, the exponential distribution is often assumed for all mechanisms. The use of an overall activation energy neglects those mechanisms which are known to have lower activation energies and can result in estimates which are impressively low but may be misleading. Furthermore, the use of the exponential distribution for all cases may also result in inaccurate projections, since it is well established in the literature that most failure rate mechanisms have non-constant failure rates.

CMOS

Channel Length: 1.5 μm

Gate Oxide Thickness: 250Å

Metal Pitch: 4-6 μm

Product Types Tested: Static RAMs – Am99C68, Am99C88

Non-Volatile Memory Division - Am27C1024

Microprocessor - Am29C10A

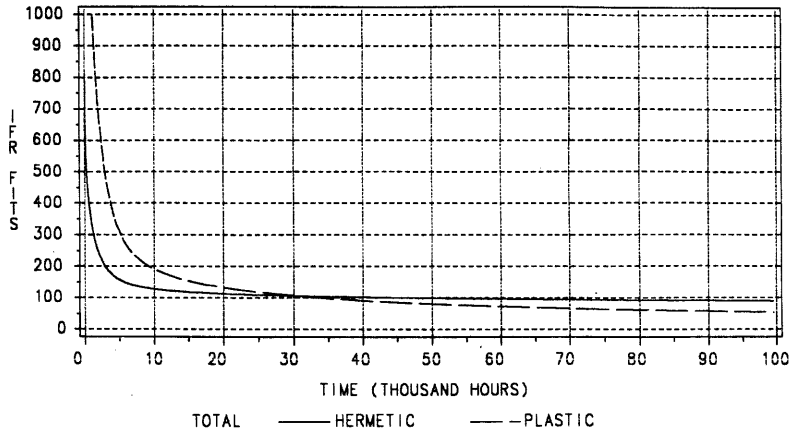
Fixed Instruction Processor - Am82C288

Data Summary and Failure Rate Estimation for General Reliability Model

Package Type	Term of Model	Failure Mechanism	Test Results			Reliability Modeling		Average Failure Rate (AFR)				
			168 hrs 125°C	1000 hrs 125°C	150°C	E _A (eV)	Parameters @ 55° C	FITs @ 55°C				
			Sample Size			MTTF (hrs)	Fraction Defective β (PPM)	0-4khrs				
			Number of Rejects					4-30khrs				
						30-100khrs						
Hermetic	Defective Subpopulations	Cause not found	1	0	0	0.45	1645	178	41	1	0	
			Totals			2	4	8			262	120
	Competing Mechanisms	Corroded Metal	0	2	0	0.50	2.5	18	13	39	53	
		Cracked Oxide	0	1	1	1.00	9.0	44	9	2	1	
		Ionic Contamination	1	0	0	1.00	9.0	45	6	1	1	
		Charge Gain/Loss	0	0	1	0.80	9.0	44	11	2	1	
		Oxide Pinholes	0	0	1	0.30	5.5	28	51	22	12	
		Cause not found	0	1	5	0.30	5.5	27	94	37	19	
		0 Rejects 50% conf.	0	0	0	0.30	5.5	28	38	17	9	
		Totals			2	4	8			262	120	96

Package Type	Competing Mechanisms	Sample Size			Sigma	ln(T50)	Average Failure Rate (AFR)			
		516	216	0			0-4khrs			
		Number of Rejects			4-30khrs					
					30-100khrs					
Plastic	0 Rejects 50% conf.	0	0	.	0.30	5.5	24	516	159	73
	Totals		0	0	.			516	159	73

**Instantaneous Failure Rate at Field Conditions.
 Curves Derived from General Reliability Model.**



Traditional Method for Reliability Projection

Single Exponential Distribution Assumed $E_A = 0.7eV$
 Stress Junction Temperature to Field Junction Temperature

Package Type	Stress	Sample Size	Equivalent Device Hours at 55° C	Rejects (60% Confidence)	Failure Rate (FITS)
Hermetic	168 hrs 125°C	6,403	83,841,423	2	
	1000 hrs 125°C	2,655	206,933,299	4	
	1000 hrs 150°C	1,477	384,626,879	8	
	Totals	10,535	675,401,600	14	23
Plastic	168 hrs 125°C	516	6,756,548	0	
	1000 hrs 125°C	216	16,835,251	0	
	Totals	732	23,591,799	0	39

Package Related Tests

Stress	Package Type	Sample Size	Failure Mechanism	Number of Rejects	Percent Rejected
Temperature Cycle	Hermetic	150		0	0.00
			Totals	0	0.00
Pressure Pot	Plastic	50		0	0.00
			Totals	0	0.00

IMOXII

Channel Length: N/A Gate Oxide Thickness: N/A Metal Pitch: 4-7 μm

Product Types Tested: Bipolar RAM- Am93422, Am93L412, Am93L422, Am93L425

Field Programmable Logic- AmPAL16H8, AmPAL16HD8, AmPAL16L8, AmPAL16L8L, AmPAL16R4,
AmPAL16R4L, AmPAL16R6, AmPAL16R6L, AmPAL16R8, AmPAL16R8L, AmPAL22V10

Bipolar Prom- Am27S25, Am27S29, Am27S31, Am27S33, Am27S181, Am27PS191, Am27S191

Interface and Logic Products- Am29827, Am29828, Am29833, Am29841, Am29843,
Am29844, Am29845, Am29853, Am29863, Am25LS14A

Microprocessor- Am2901C, Am2910A, Am29705A

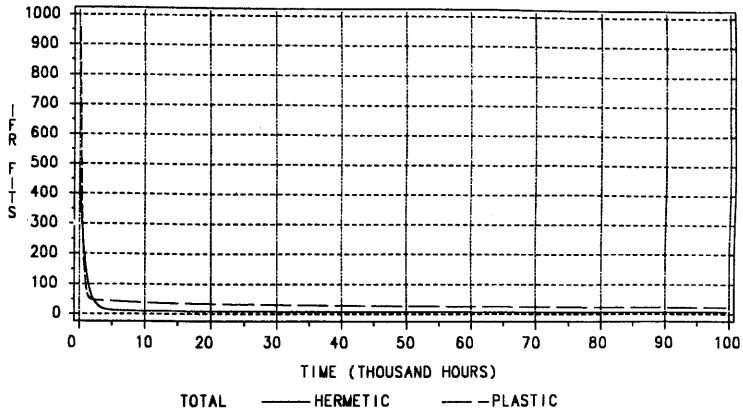
Microcontroller- Am29116

Peripheral Products- Am8177

Data Summary and Failure Rate Estimation for General Reliability Model

Package Type	Term of Model	Failure Mechanism	Test Results			Reliability Modeling		Average Failure Rate (AFR)			
			168 hrs 125°C	1000 hrs 125°C	5,709 150°C	E A (eV)	Parameters @ 55° C	FITs @ 55°C			
			Sample Size			MTTF (hrs)	Fraction Defective β (PPM)	0-4khrs	4-30khrs	30-100khrs	
			22,718	7,060	5,709						
Hermetic	Defective Subpopulations	Damaged Metal	1	0	0	0.45	848	50	13	0	0
		Foreign Material Oxide	3	0	0	0.45	848	151	38	0	0
		Wire Heel Broken	1	0	0	0.45	848	50	13	0	0
		Cause not Found	2	0	0	0.45	848	101	25	0	0
	Competing Mechanisms						Sigma In(T50)				
		Crystal Defects	1	0	0	0.70	9.0	45	5	1	1
		Cracked Oxide	1	0	1	1.00	9.0	46	3	1	0
		0 Rejects 50% conf.	0	0	0	0.50	4.0	24	8	8	6
		Totals	9	0	1				103	10	7
	Plastic	Defective Subpopulations	Glassivation Damaged	1	0	-	0.45	275	64	16	0
Damaged Metal			1	0	-	0.45	275	64	16	0	0
Wire Clearance			1	0	-	0.45	275	64	16	0	0
Cause not Found			3	0	-	0.45	275	191	48	0	0
Competing Mechanisms							Sigma In(T50)				
		Ionic Contamination	1	0	-	1.00	9.0	46	4	1	0
		0 Rejects 50% conf.	0	0	-	0.50	4.0	23	44	34	25
		Totals	7	0	-				144	35	25

**Instantaneous Failure Rate at Field Conditions.
 Curves Derived from General Reliability Model.**



Single Exponential Distribution Assumed $E_A = 1.0eV$
 Stress Junction Temperature to Field Junction Temperature

Package Type	Stress	Sample Size	Equivalent Device Hours at 55° C	Rejects (60% Confidence)	Failure Rate (FITS)
Hermetic	168 hrs 125°C	22,718	931,618,604	9	
	1000 hrs 125°C	7,060	1,724,695,417	0	
	1000 hrs 150°C	5,709	6,530,194,964	1	
	Totals	35,487	9,186,508,985	10	1
Plastic	168 hrs 125°C	18,338	368,584,446	7	
	1000 hrs 125°C	6,580	740,419,943	0	
	Totals	24,918	1,109,004,389	7	8

Package Related Tests

Stress	Package Type	Sample Size	Failure Mechanism	Number of Rejects	Percent Rejected
Temperature Cycle	Hermetic	2,849	Lifted Metal	4	0.14
			Cracked Oxide	3	0.11
			Package Seal Cracks	1	0.04
			Package Seal Voids	1	0.04
			Cause not found	4	0.14
	Totals		13	0.46	
	Plastic	2,603	Die Cracked	1	0.04
			Glassivation Cracked	2	0.08
			Corroded Metal	1	0.04
			Metal-Metal Short	1	0.04
Cracked Oxide			4	0.15	
Water in Package	2	0.08			
Wire Neck Broken	1	0.04			
Intermetallics	5	0.19			
Totals		17	0.65		
Temperature Humidity	Plastic	2,201	Cause not found	1	0.05
			Totals	1	0.05
Pressure Pot	Plastic	2,959	Die Cracked	1	0.03
			Corroded Leads	1	0.03
			Corroded Metal	1	0.03
			Totals	3	0.10

7.4 CMOS LATCH-UP TEST METHODS AND RESULTS

Latch-up is a phenomenon that occurs when a parasitic PNP structure on an IC chip is triggered and behaves like an SCR between the V_{CC} and GND rails. Once initiated, the latch-up condition will persist until either the power supply is removed or the device is destroyed. In virtually all cases, the device is destroyed because of the large current that can flow from the V_{CC} to the ground pin (the ON resistance of the SCR is very low).

Interior nodes of an IC could conceivably be prone to latch-up, but this intrinsically rare condition would be found during normal device testing and screening. Circuit nodes interfacing with the "outside world" are much more susceptible to latch-up because unusual transient conditions may occur – in particular, overshoot or ringing that pull the pin above the supply voltage or below GND.

To induce latch-up, the conditions on these pins must meet two criteria: a) there must be sufficient voltage to forward bias-critical junctions in the SCR, and b) the available current must be in excess of the SCR trigger current. If these conditions exist, and if a suitable parasitic PNP structure is connected to that pin, latch-up will occur.

Some thought must be given to the test values of voltage and current when determining susceptibility of a part to latch-up. Reasonable test values would seem to be those experienced in an actual system under worst-case conditions.

Most AMD devices are designed to work with a nominal +5V supply. In such a system, voltage transients result-

ing from transmission line effects, etc., will not exceed +5V in magnitude. Therefore, testing at a +10V extreme (V_{CC} plus 5V transient) and a -5V extreme (GND minus 5V transient) will simulate a worst-case system environment.

Current levels for latch-up testing are governed by the maximum current available from any device in the system. The maximum drive capability of any output pin is approximately 100 mA; adding some margin to this, the test value becomes 300 mA. Any current derived from the voltage transient magnitude divided by the transmission line impedance will be considerably less than this.

Latch-Up Testing

Testing was performed by forcing 300 mA into and out of each device pin, whether input or output, while monitoring I_{CC} for any indication of latch-up. The current sources were voltage-limited at +10V and -5V, per the discussion above. The test configurations are shown in Figures 7-4 and 7-5.

Normal outputs were set to the HIGH state when current was forced into the pin (positive current) and set to the LOW state when the current was pulled out of the device (negative current). Outputs with three-state capability were additionally tested in the high-impedance state.

The test results are summarized in Table 7-7. For the test limits indicated, **no latch-up was induced for any pin of any part of any device type tested.**

Note that there was no positive current flow into the input pins since the inputs remained high-impedance up to the +10V clamp level.

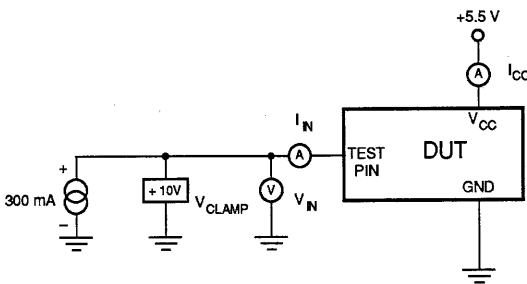


Figure 7-4.

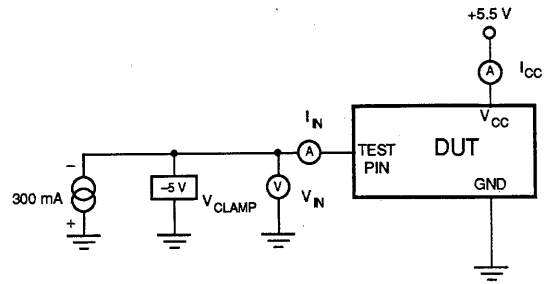


Figure 7-5.

Table 7-7. CMOS Latch-Up Testing Summary
 (Am29C01, Am29C10A, Am29C101)

Tested Pin	Test Figure	Max II (mA)	Max VI (V)	Latch-Up
Inputs	1	0	+10	No
	2	-18	-5	No
Normal Outputs	1	+300	+6.5	No
	2	-300	-1.4	No
Three-State Outputs (active)	1	+300	+6.6	No
	2	-300	-1.8	No
Three-State Outputs(High-Z)	1	+300	+10	No
	2	-300	-1.8	No

7.5 TEST PHILOSOPHY AND METHODS

The following nine points describe AMD's philosophy for high volume, high speed automatic testing.

1. Ensure that the part is adequately decoupled at the test head. Large changes in V_{CC} current as the device switches may cause erroneous function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may start to oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an output transition, ground current may change by as much as 400 mA in 5-8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily.
4. Use extreme care in defining point input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins and they may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \geq 3.0$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.

6. Capacitive Loading for AC Testing

Automatic testers and their associated hardware have stray capacitance that varies from one type of tester to another but is generally around 50 pF. This, of course, makes it impossible to make direct measurements of parameters which call for smaller capacitive load than the associated stray capacitance. Typical examples of this are the so-called "float delays," which measure the propagation delays into the high-impedance state and are usually specified at a load capacitance of 5.0 pF. In these cases, the test is performed at the higher load capacitance (typically 50 pF) and engineering correlations based on data taken with a bench setup are used to predict the result at the lower capacitance.

Similarly, a product may be specified at more than one capacitive load. Since the typical automatic tester is not capable of switching loads in mid-test, it is impossible to make measurements at **both** capacitances even though they may both be greater than the stray capacitance. In these cases, a measurement is made at one of the two capacitances. The result at the other capacitance is predicted from engineering correlations based on data taken with a bench setup and the knowledge that certain DC measurements (I_{OH} , I_{OL} for example) have already been taken and are within spec. In some cases, special DC tests are performed in order to facilitate this correlation.

7. Threshold Testing

The noise associated with automatic testing (due to the long, inductive cables) and the high gain of the tested device when in the vicinity of the actual device threshold, frequently give rise to oscillations when testing high speed circuits. These oscillations are not indicative of a reject device, but instead of an over-taxed test system. To minimize this problem, thresholds are tested at least once for each input pin. Thereafter, "hard" high and low levels are used for other tests. Generally this means that function and AC testing are performed at "hard" input levels rather than at V_{IL} Max. and V_{IH} Min.

8. AC Testing

Occasionally, parameters are specified that cannot be measured directly on automatic testers because of tester limitations. Data input hold times often fall into this category. In these cases, the parameter in question is guaranteed by correlating these tests with other AC tests that have been performed. These correla-

tions are arrived at by the cognizant engineer by using precise bench measurements in conjunction with the knowledge that certain DC parameters have already been measured and are within spec.

In some cases, certain AC tests are redundant, since they can be shown to be predicted by some other tests which have already been performed. In these cases, the redundant tests are not performed.

9. Output Short-Circuit Current Testing

When performing I_{OS} tests on devices containing RAM or registers, great care must be taken that undershoot caused by grounding the high-state output does not trigger parasitic elements which in turn cause the device to change state. In order to avoid this effect, it is common to make the measurement at a voltage (V_{OUTPUT}) that is slightly above ground. The V_{CC} is raised by the same amount so that the result (as confirmed by Ohm's law and precise bench testing) is identical to the $V_{OUT} = 0, V_{CC} = \text{Max.}$ case.

CHAPTER 8

General Information

8.1 PHYSICAL DIMENSIONS

8-1

8.2 ORDERING INFORMATION

8-8

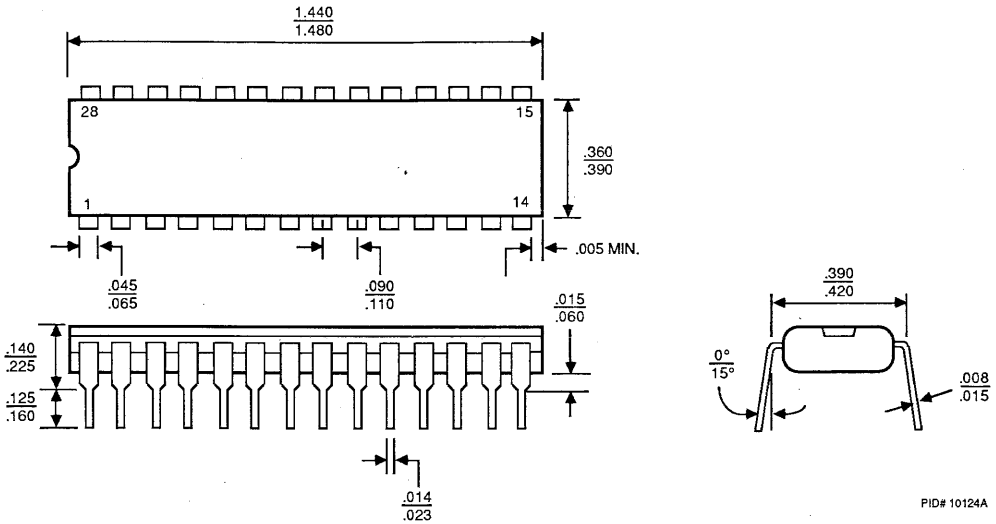
CHAPTER 8

General Information

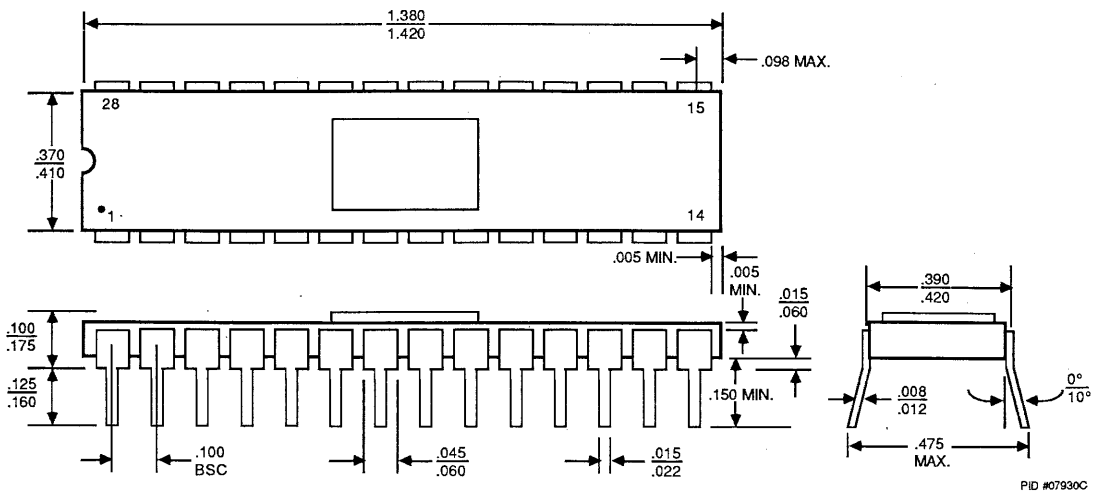


8.1 PHYSICAL DIMENSIONS*

Plastic DIP (PD) PD4028



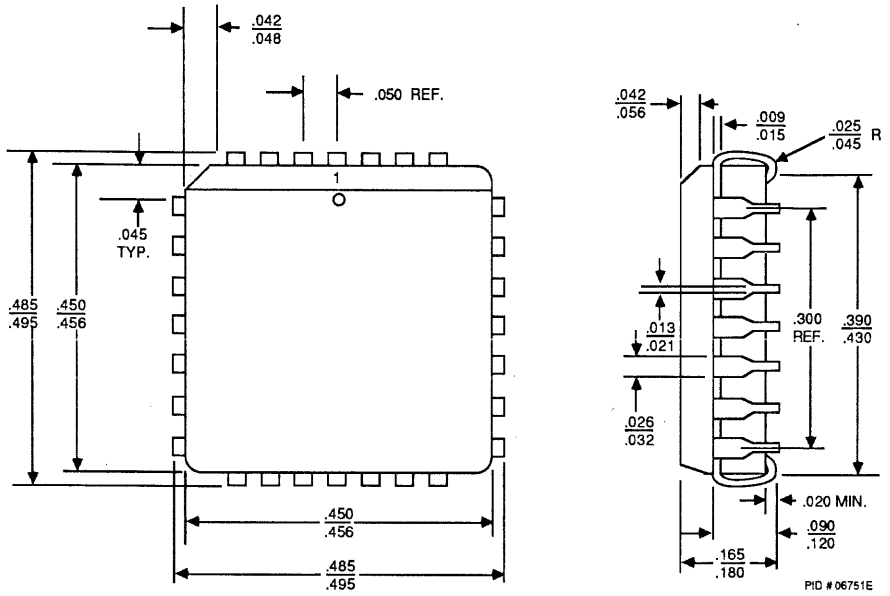
Ceramic Sidebrazed DIP (SD) SD4028



* For reference only.

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

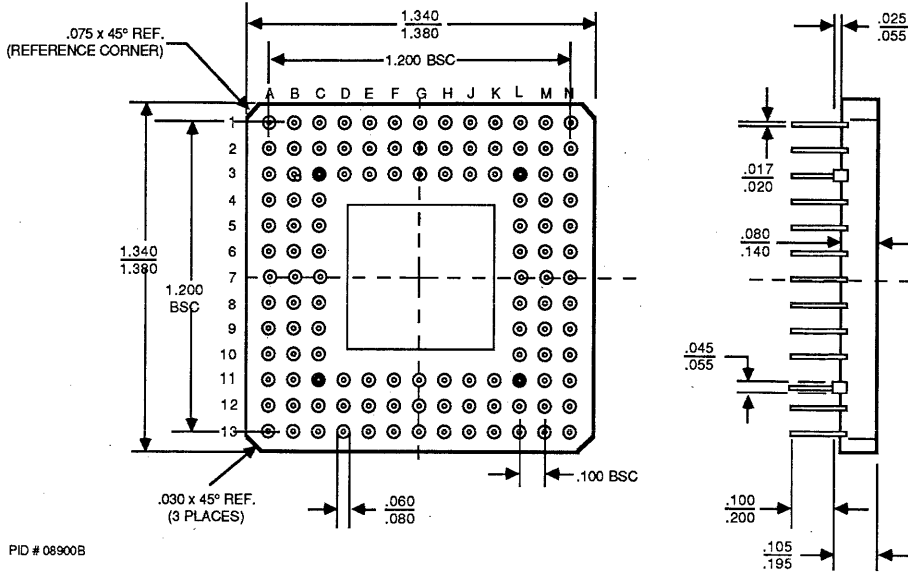
Plastic Leaded Chip Carrier (PC)
PL 028



NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

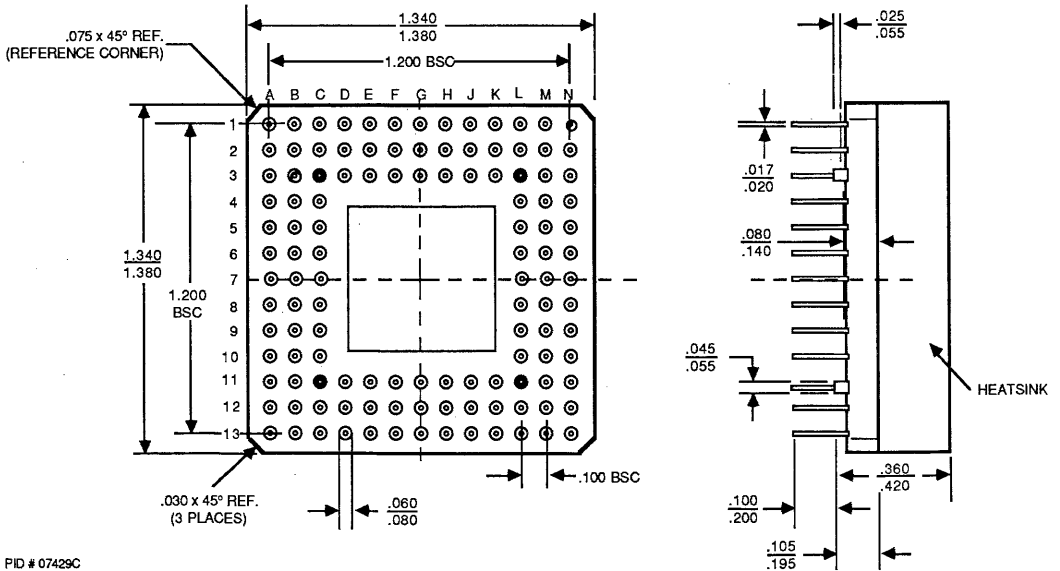
Ceramic Pin-Grid-Array Packages (CG/CGX)
CGX120

BOTTOM VIEW



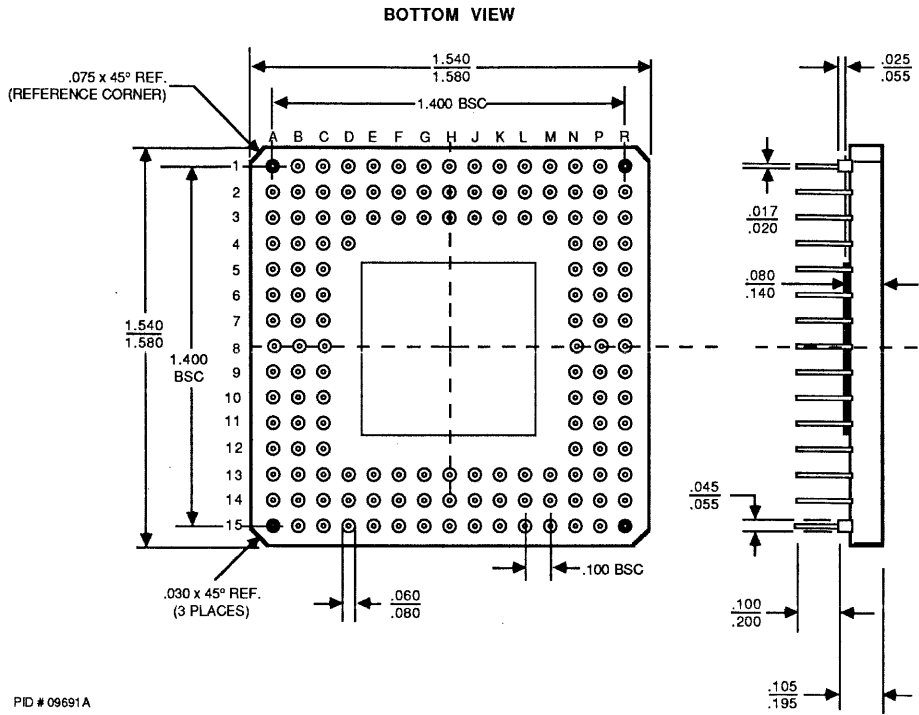
CG 120

BOTTOM VIEW



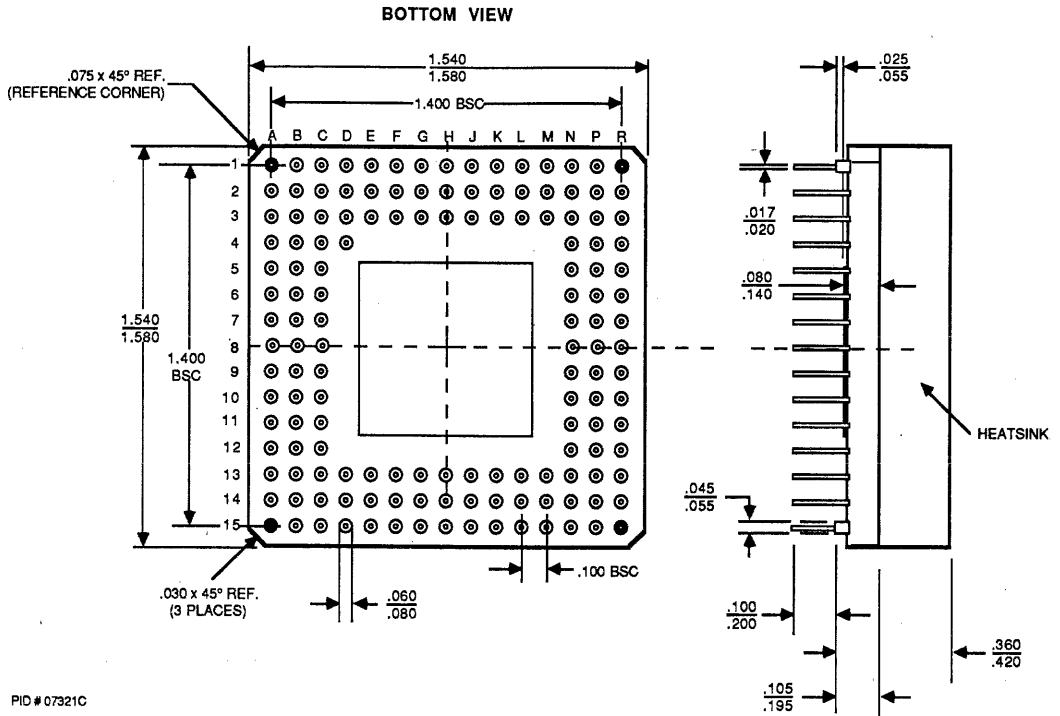
NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

Ceramic Pin-Grid-Array Packages (CG/CGX) (Continued)
CGX145



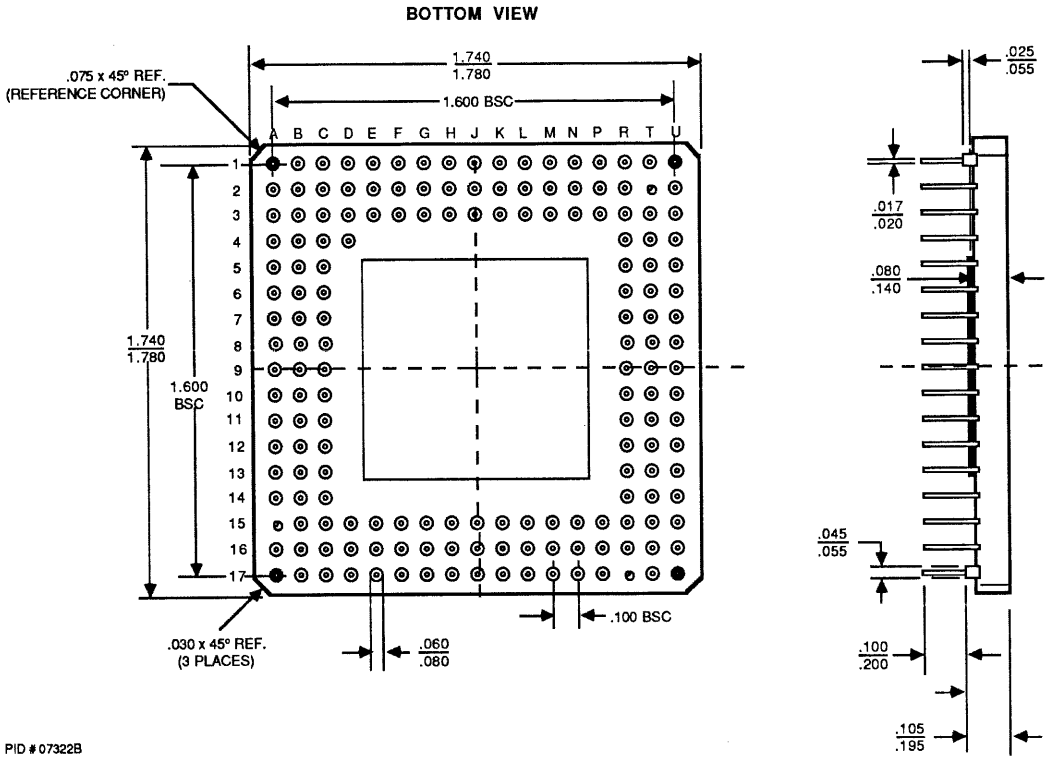
NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

Ceramic Pin-Grid-Array Packages (CG/CGX) (Continued)
CG 145



NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

Ceramic Pin-Grid-Array Packages (CG/CGX) (Continued)
CGX169

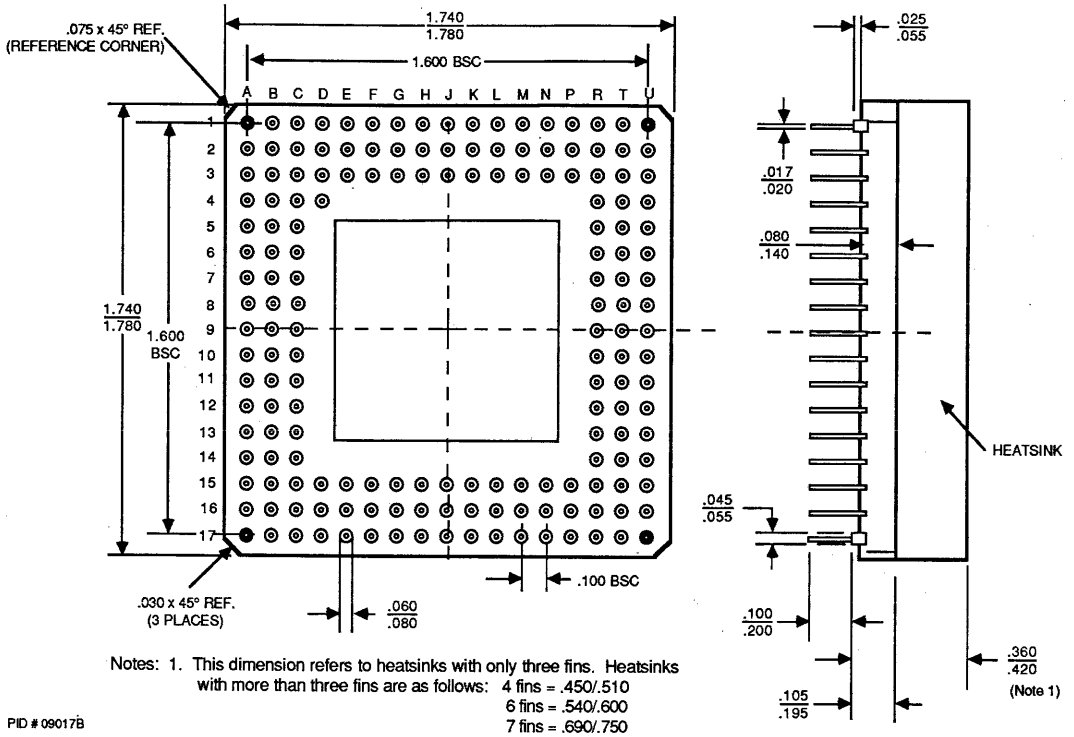


NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

Ceramic Pin-Grid-Array Packages (CG/CGX) (Continued)

CG 169

BOTTOM VIEW



PID # 09017B

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

8.2 ORDERING INFORMATION

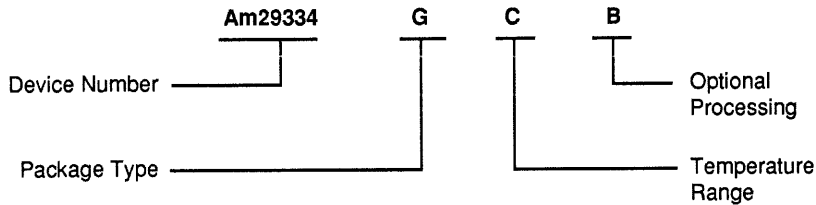
All Advanced Micro Devices' products listed are stocked locally and distributed nationally by Franchised Distributors. See back of this book for the location nearest you. Please consult them for the latest price revisions. For direct factory orders, call your local AMD Sales Office or Sales Representative. See the back of this book for the location nearest you.

Minimum Order

The minimum direct factory order is \$100.00 for a standard product. The minimum direct factory order for burn-in product is \$250.00.

Product Ordering, Package and Temperature Range Codes

The following scheme is used to identify Advanced Micro Devices' Standard products:



Package Type

P = Plastic DIP
D = Ceramic DIP
G = Pin Grid Array
J = Plastic Leaded Chip Carrier

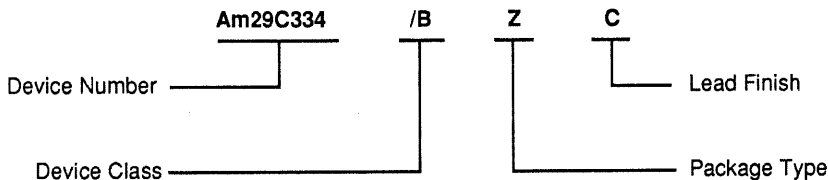
Temperature Range

C = Commercial
(0 to +70°C)

Optional Processing

Blank = Standard Processing
B = Burn-in

The following scheme is used to identify Advanced Micro Devices' Military (APL) products:



Device Class

/B = Class B

Package Type

X = DIP Packages
Z = All Other Configurations
(PGAs, etc.)

Lead Finish

C = Gold

NOTES

NOTES

ADVANCED MICRO DEVICES' NORTH AMERICAN SALES OFFICES

ALABAMA	(205) 882-9122	KANSAS	(913) 451-3115
ARIZONA	(602) 242-4400	MARYLAND	(301) 796-9310
CALIFORNIA		MASSACHUSETTS	(617) 273-3970
Culver City	(213) 645-1524	MINNESOTA	(612) 938-0001
Newport Beach	(714) 752-6262	MISSOURI	(314) 275-4415
San Diego	(619) 560-7030	NEW JERSEY	(201) 299-0002
San Jose	(408) 249-7766	NEW YORK	
Santa Clara	(408) 727-3270	Liverpool	(315) 457-5400
Woodland Hills	(818) 992-4155	Poughkeepsie	(914) 471-8180
CANADA, Ontario		Woodbury	(516) 364-8020
Kanata	(613) 592-0060	NORTH CAROLINA	(919) 847-8471
Willowdale	(416) 224-5193	OHIO	(614) 891-6455
COLORADO	(303) 741-2900	Columbus	(614) 891-6455
CONNECTICUT	(203) 264-7800	Dayton	(513) 439-0470
FLORIDA		OREGON	(503) 245-0060
Clearwater	(813) 530-9971	PENNSYLVANIA	
Ft Lauderdale	(305) 776-2001	Allentown	(215) 398-8006
Melbourne	(305) 729-0496	Willow Grove	(215) 657-3101
Orlando	(305) 859-0831	TEXAS	
GEORGIA	(404) 449-7920	Austin	(512) 346-7830
ILLINOIS		Dallas	(214) 934-9099
Chicago	(312) 773-4422	Houston	(713) 785-9001
Naperville	(312) 505-9517	WASHINGTON	(206) 455-3600
INDIANA	(317) 244-7207	WISCONSIN	(414) 792-0590

ADVANCED MICRO DEVICES' INTERNATIONAL SALES OFFICES

BELGIUM		KOREA, Seoul	TEL	82-2-784-7598
Bruxelles	TEL		FAX	82-2-784-8014
	FAX	LATIN AMERICA,		
	TLX	Ft. Lauderdale	TEL	(305) 484-8600
FRANCE			FAX	(305) 485-9736
Paris	TEL		TLX	5109554261 AMDFTL
	FAX	NORWAY		
	TLX	Hovik	TEL	(47) 2 537810
GERMANY			FAX	(47) 2 591959
Hannover area	TEL		TLX	79079
	FAX	SINGAPORE	TEL	65-2257544
	TLX		FAX	2246113
	TEL		TLX	RS55650 MMI RS
	FAX	SWEDEN, Stockholm	TEL	(08) 733 03 50
	TLX		FAX	(08) 733 22 85
	TEL		TLX	11602
	FAX	TAIWAN	TLX	886-2-7122066
	TLX		FAX	886-2-7122017
HONG KONG,		UNITED KINGDOM,		
Kowloon	TEL	Farnborough	TEL	(0252) 517431
	FAX		FAX	(44252) 521041
	TLX		TLX	858051
ITALY, Milano	TEL		TEL	(0925) 828008
	FAX		FAX	(0925) 827693
	TLX		TLX	628524
JAPAN,			TEL	(04862) 22121
Tokyo	TEL		FAX	(04862) 22179
	FAX		TLX	859103
	TLX			
	TEL			
	FAX			

NORTH AMERICAN REPRESENTATIVES

CALIFORNIA		MICHIGAN		
² INC	OEM (408) 988-3400	SAI MARKETING CORP	(313) 750-1922	
	DISTI (408) 498-6868	MISSOURI		
CANADA		LORENZ SALES	(314) 997-4558	
Calgary, Alberta		NEBRASKA		
VITEL ELECTRONICS	(403) 278-5833	LORENZ SALES	(402) 475-4660	
Kanata, Ontario		NEW MEXICO		
VITEL ELECTRONICS	(613) 592-0090	THORSON DESERT STATES	(505) 293-8555	
Mississauga, Ontario		NEW YORK		
VITAL ELECTRONICS	(416) 676-9720	NYCOM, INC	(315) 437-8343	
Quebec		OHIO		
VITEL ELECTRONICS	(514) 636-5951	Columbus		
IDAHO		DOLFUSS ROOT & CO	(614) 885-4844	
INTERMOUNTAIN TECH MKGT	(208) 888-6071	Dayton		
INDIANA		DOLFUSS ROOT & CO	(513) 433-6776	
SAI MARKETING CORP	(317) 253-1668	Strongsville		
IOWA		DOLFUSS ROOT & CO	(216) 238-0300	
LORENZ SALES	(319) 377-4666	PENNSYLVANIA		
KANSAS		DOLFUSS ROOT & CO	(412) 221-4420	
LORENZ SALES	(913) 384-6556	UTAH		
		R ² MARKETING	(801) 595-0631	

Advanced Micro Devices reserves the right to make changes in its product without notice in order to improve design or performance characteristics. The performance characteristics listed in this document are guaranteed by specific tests, guard banding, design and other practices common to the industry. For specific testing details, contact your local AMD sales representative. The company assumes no responsibility for the use of any circuits described herein.





**ADVANCED
MICRO
DEVICES, INC.**

901 Thompson Place
P.O. Box 3453
Sunnyvale
California 94088-3453
(408) 732-2400
TELEX: 34-6306
TOLL FREE
(800) 538-8450

**APPLICATIONS
HOTLINE
(800) 222-9323**