

# Am29027

Arithmetic Accelerator

## ADVANCE INFORMATION

### DISTINCTIVE CHARACTERISTICS

- High-speed floating-point accelerator for the Am29000 processor
- Comprehensive floating-point and integer instruction sets
- Single-, double-, and mixed-precision operations
- Performs conversions between precisions and between data formats
- Compatible with industry-standard floating-point formats
  - IEEE P754 version 10.1
  - DEC F, DEC D, and DEC G formats
  - IBM system 370 format
- Exact IEEE compliance for denormalized numbers
- Simple interface requires no glue logic between Am29000 and Am29027
- Eight-deep register file for intermediate results and on-chip 64-bit datapath facilitate compound operations, e.g., Newton-Raphson division, sum-of-products, and transcendental
- Supports pipelined or flow-through operation
  - Performs single- and double-precision floating-point operations at 120-ns pipelined rate
- Full compiler and assembler support
- Fabricated with Advanced Micro Devices' 1.2 micron CMOS process

### GENERAL DESCRIPTION

The Am29027 Arithmetic Accelerator is a high-speed computational unit intended for use with the Am29000 Streamlined Instruction Processor (SIP). When added to a 29000-based system, the Am29027 can improve floating-point performance by an order of magnitude or more.

The Am29027 implements an extensive floating-point and integer instruction set, and 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 standard P754, with direct implementation of special features such as gradual underflow and trap handling.

The Am29027 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 datapath 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 orthogonal selection of input operands, and an output multiplexer that allows access to result data, operation status, flags, or accelerator state. The control unit interprets transaction requests from the Am29000, and sequences the ALU and datapath.

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, such as those found in graphics or signal processing.

The Am29027 connects directly to Am29000 system buses, and requires no additional interface circuitry.

Fabricated with AMD's 1.2 micron technology, the Am29027 is housed in a 169-lead pin-grid-array (PGA) package.

3

res.

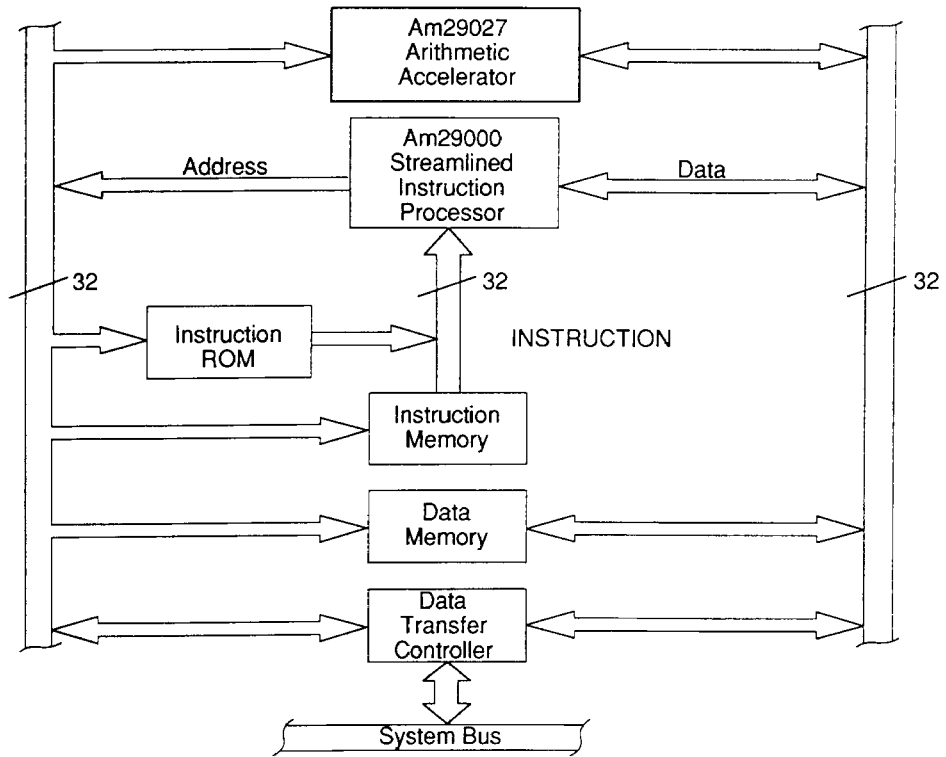
Orig

004980

4980

AMD

## SIMPLIFIED SYSTEM DIAGRAM

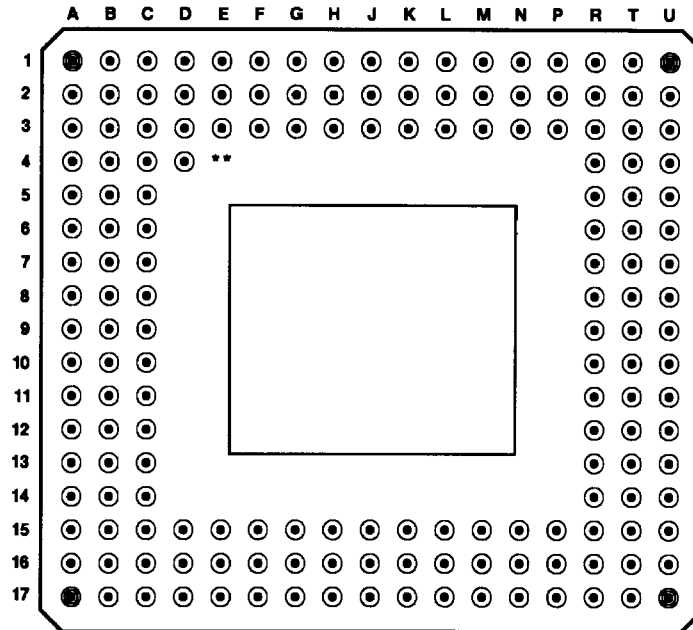


AF004740

## RELATED AMD PRODUCTS

Part No.	Description
Am29000	Streamlined Instruction Processor

### CONNECTION DIAGRAM 169-Lead PGA\* Bottom View



CD009761

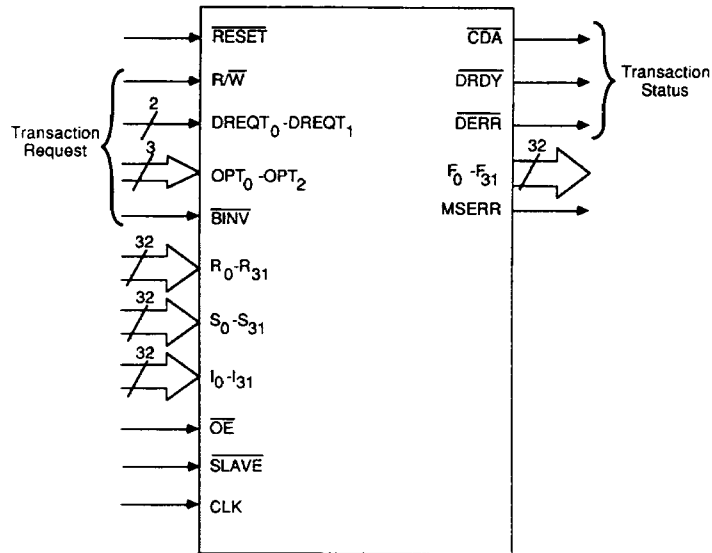
\*Pinout observed from pin side of package.

\*\*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



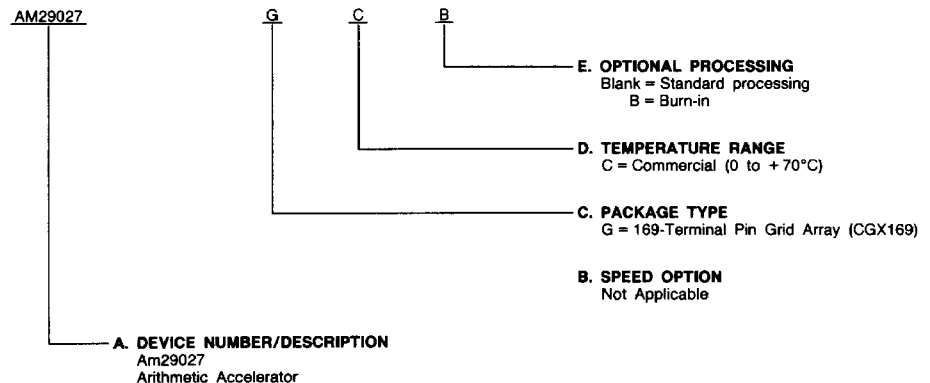
LS002960

## 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	
AM29027	GC, GCB

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

### **BINV Bus Invalid (Input; Active LOW)**

A logic LOW indicates that the Am29000 address bus is invalid, and that signals **DREQT<sub>1</sub>** and **DREQT<sub>0</sub>** are to be ignored.

### **CDA Accelerator Data Accept (Output; Active LOW)**

A logic LOW indicates that the Am29027 is ready to accept operands or operation codes from the Am29000.

### **CLK Clock (Input)**

### **DERR Data Error (Output; Active LOW)**

A logic LOW indicates that one or more unmasked flags were set by a previous operation.

### **DRDY Data Ready (Output; Active LOW)**

A logic LOW indicates that valid data is available on port F.

### **DREQT<sub>0</sub> Start Instruction/Suppress Errors (Input; Active HIGH)**

This signal, when accompanied by a valid write operand **R**, write operand **S**, write operands **R**, **S**, or write instruction transaction request, commands the Am29027 to begin a new operation. When accompanying a valid read result **LSBs**, read result **MSBs**, read flags, or read status transaction request, **DREQT<sub>0</sub>** suppresses the reporting of operation errors. This signal is considered valid only when signal **BINV** is inactive.

### **DREQT<sub>1</sub> Accelerator Transaction Request (Input; Active HIGH)**

A logic HIGH indicates that the Am29000 is making an accelerator transaction request. This signal is considered valid only when signal **BINV** is inactive.

### **F<sub>0</sub>–F<sub>31</sub> F Output Bus (Output)**

### **I<sub>0</sub>–I<sub>31</sub> Instruction Word (Input)**

Specifies the operation to be performed by the accelerator, including the opcode, integer/floating-point select,

multiplexer selects, register file select and enable, and the precision of the operands.

### **MSERR Master/Slave Error (Output)**

A logic HIGH indicates that a discrepancy has been detected between the outputs of the master and slave accelerators.

### **OE Output Enable (Input; Active LOW)**

A logic HIGH disables all accelerator outputs unconditionally. When **OE** is LOW, accelerator outputs are enabled and disabled by transaction requests. This signal is provided for test purposes.

### **OPT<sub>0</sub>–OPT<sub>2</sub> Transaction Type (Input)**

These signals, in conjunction with **R/W**, specify the type of accelerator transaction, if any, currently being requested by the Am29000.

### **R<sub>0</sub>–R<sub>31</sub> R Input Data Bus (Input)**

### **RESET Reset (Input; Asynchronous, Active LOW)**

Resets the state of the internal sequencing circuitry. When **RESET** is asserted, the state of the instruction and data registers is undefined; the status register is cleared. **RESET** must be connected to the signal line used to reset the Am29000.

### **R/W Read/Write (Input)**

Determines the direction of a transaction. When **R/W** is HIGH, data is being transferred from the Am29027 to the Am29000. When **R/W** is LOW, data is being transferred from the Am29000 to the Am29027.

### **S<sub>0</sub>–S<sub>31</sub> S Input Data Bus (Input)**

### **SLAVE Master/Slave Mode Select (Input; Active LOW)**

A logic **LOW** selects Slave mode; in this mode all outputs except **MSERR** are disabled. A logic **HIGH** selects Master mode.

## FUNCTIONAL DESCRIPTION

### Overview

The Am29027 is a high-performance, single-chip arithmetic accelerator for the Am29000 Streamlined Instruction Processor.

### Architecture

The Am29027 comprises a high-speed ALU, a 64-bit datapath, and control circuitry.

The core of the Am29027 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. Seven ALU flags report operation status. The ALU is completely combinatorial for reduced latency; optional pipelining is available to boost throughput for array operations.

The datapath consists of the 64-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; an output multiplexer that permits the selection of data, flags, operation status, or accelerator state; and a 64-bit output bus **F**. Input operands enter the floating-point accelerator 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 three-state output bus **F**. The results can also be stored in the register file for use in subsequent operations.

On-board control circuitry sequences the ALU and datapath during operations, and manages the transfer of data between the accelerator and its host.

### Instruction Set

The Am29027 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
- Saturation (clipping)
- 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.

### **Performance**

The Am29027 provides operation speeds several times greater than that of conventional floating-point coprocessors, by virtue of its extensive use of combinatorial, rather than sequential, logic. Most floating-point operations, whether single-, double-, or mixed-precision, can be performed in as few as six system clock cycles. Performance is further enhanced by the presence of the on-board register file, which can be used to hold intermediate operation results, thus reducing the amount of time needed to transfer operands between the Am29027 and its host.

### **Interface**

The Am29027 connects directly to the Am29000 system buses. Am29027 operations are specified by a series of operand and instruction transactions issued by the Am29000. Seven input signals specify the transaction requested by the Am29000; three output signals report transaction status.

### **Master/Slave**

The Am29027 contains special comparison hardware to allow the operation of two accelerators 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.

### **Support**

The Am29027 is fully supported by those hardware and software tools available for the Am29000, including:

- Am29000 C, Pascal, and Fortran Compilers
- Am29000 Assembler
- Am29000 Functional Simulator
- Hardware Debug Module that interfaces the target system to a logic analyzer and a host. The Module permits single-step operation, breakpoint insertion, and other standard debugging techniques.

### **Low Power**

The Am29027 has a maximum power dissipation of 1.5 W, guaranteed over the operating range.

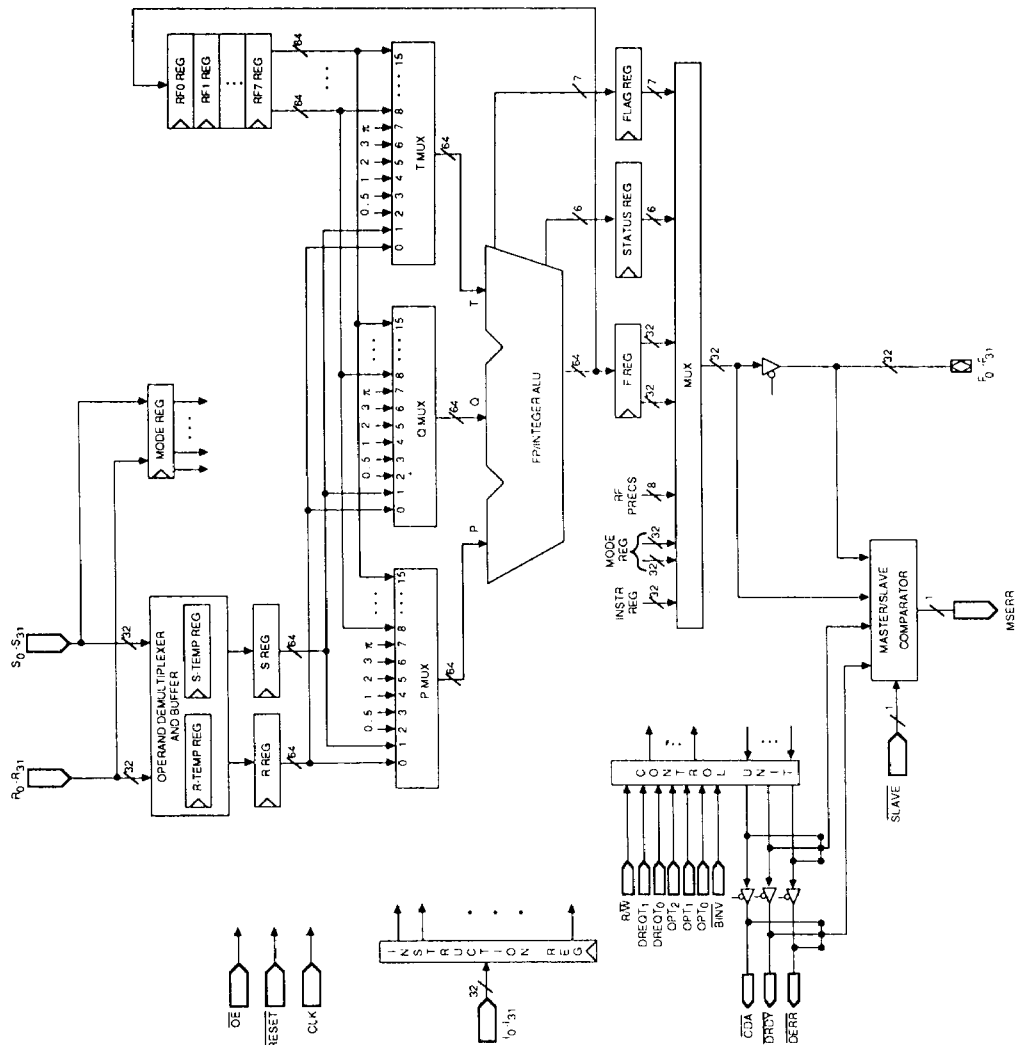


Figure 1. Am29027 Block Diagram



## Block Diagram Description

A block diagram of the Am29027 is shown in Figure 1. The Am29027 comprises input registers, operand selection multiplexers, instruction register, ALU, output register/register file, flag register, status register, output selection multiplexer, control unit, mode register, and the master/slave comparator.

### Input Registers

Operands enter the accelerator through the R and S buses, and are demultiplexed and buffered for subsequent storage in 64-bit registers R and S. Input operands may be either single-precision (32-bit) or double-precision (64-bit). Two single-precision operands or one double-precision operand may be loaded in a single system clock cycle. 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.

### 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 accelerator operation. Included in the instruction word are fields that specify: the core operation to be performed by the ALU; sign-change selects for ALU input and result operands; the operands to be selected as ALU inputs; the register file write location and write enable; and operand precisions. A detailed description appears in the Accelerator Instruction Set section.

### 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  $(P \cdot Q) + T$ . 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. Most ALU operations allow the user to modify operand signs, thus greatly increasing the number of arithmetic expressions that can be evaluated in a single ALU pass. A detailed description of ALU operations appears in the Accelerator Instruction Set section.

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.

### 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.

## Flag Register

The flag register is a 7-bit register that stores flags pertaining to the most recently performed operation. A detailed description of the flag register is provided in the Accelerator Instruction Set section.

### Status Register

The status register stores six operation status bits generated by the ALU during the course of an operation. These status bits are individually latched; once a given bit is set, it remains set until the status register is cleared by the host processor. The operation status bits indicate conditions of overflow, underflow, zero result, reserved operand, invalid operation, and inexact result. A detailed description of the status register is provided in the Accelerator Instruction Set section.

### Mode Register

The mode register contains accelerator parameters that change infrequently. The 64-bit mode word is loaded into the register via the R and S buses. A detailed description of the mode register is provided in the Mode Register Description section.

### Output Multiplexer

The output multiplexer routes operation results and accelerator internal state to the Am29000 through the F bus. This multiplexer selects from the ALU output register (32 LSBs or 32 MSBs); flag register; status register; register file precisions; instruction register; and the mode register (32 LSBs or 32 MSBs).

### Control Unit

The control unit manages the transfer of data between the Am29000 and the Am29027, as well as the timing of operation execution.

The Am29000 host processor oversees the operation of the Am29027 by issuing one of thirteen commands, or transaction requests, to the control unit via seven signal lines. Each transaction request specifies an action on the part of the Am29027, such as loading an operand into an input register or returning a result to the host. The control unit then sequences the remainder of the Am29027 to produce the desired action. Three transaction status lines are generated by the control unit to indicate whether a transaction has been completed, and whether returned data contains an error.

A detailed description of transaction requests is given in the Accelerator Transactions section.

### Master/Slave Comparator

Each Am29027 output signal has associated logic that compares that signal with the signal that the accelerator is providing internally to the output driver; any discrepancies are indicated by assertion of signal **MSERR**.

For a single accelerator, 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 accelerator in parallel with the first, with the second accelerator's outputs disabled by assertion of signal **SLAVE**. The second accelerator detects open-circuit signals, as well as providing a check of the outputs of the first accelerator.

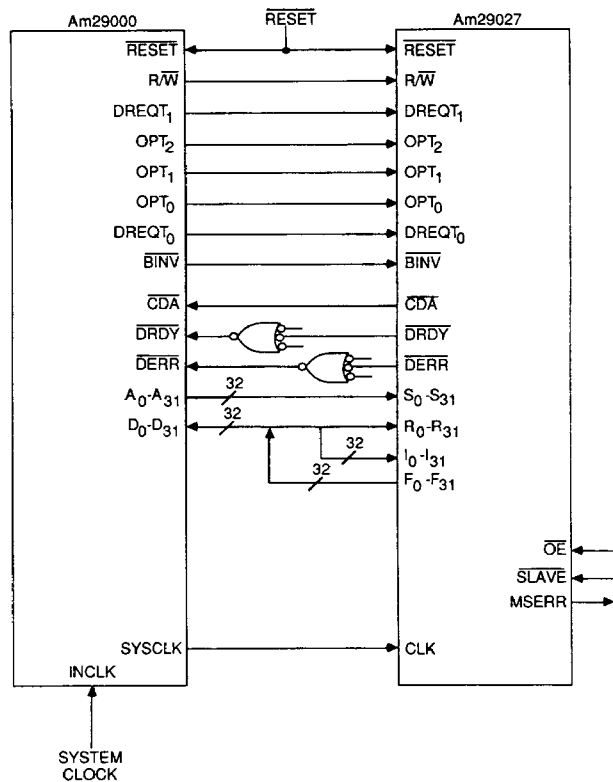


Figure 2. Am29000/Am29027 Hardware Interface

## Mode Register Description

The 64-bit mode register stores twenty-three infrequently changed parameters pertaining to accelerator operation. The Am29000 modifies the accelerator parameter set by placing the desired values on accelerator ports R and S while issuing a **write mode register** transaction request; further details on this transaction are given in the Accelerator Transactions section. The mode register should be loaded after reset, or whenever a new mode of accelerator operation is required.

### Mode Register Bits M0 – M63

Mode register bits M0 – M63 are organized as follows:

#### 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

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, 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. This bit has no effect for formats other than IEEE.

**M6 — IEEE Trap Enable:** If M6 is HIGH, 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, IEEE trapped operation is disabled. This bit has no effect for formats other than IEEE.

**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 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 formats other than IBM.

**M9 — IBM Underflow Mask Enable:** If M9 is HIGH, certain underflowed IBM operations will produce a normalized result with an exponent 128 larger than the correct one. If M9 is LOW, these operations will produce a final result of true-zero. This bit has no effect for formats other than IBM.

**M10: Reserved for future use (don't care).**

**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 — Round Mode Select:** Selects one of six rounding modes:

M16	M15	M14	Round Mode
0	0	0	Round to Nearest (unbiased)
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 (biased)
1	0	1	Round Away From Zero
1	1	X	Illegal Value

Additional information on round modes can be found in Appendix B.

**M17 — M19: Unused (don't care).**

**M20 — Pipeline Mode Select:** When M20 is HIGH, pipelined mode is selected; when M20 is LOW, flow-through (unpipelined) mode is selected.

**M21: Logic 0**

**M22 — Invalid Operation Mask Bit:** When M22 is HIGH, the invalid operation flag will not contribute to signal **DERR**.

**M23 — Reserved Operand Mask Bit:** When M23 is HIGH, the reserved operand flag will not contribute to signal **DERR**.

**M24 — Overflow Mask Bit:** When M24 is HIGH, the overflow flag will not contribute to signal **DERR**.

**M25 — Underflow Mask Bit:** When M25 is HIGH, the underflow flag will not contribute to signal **DERR**.

**M26 — Inexact Result Mask Bit:** When M26 is HIGH, the inexact result flag will not contribute to signal **DERR**.

**M27 — Zero Mask Bit:** When M27 is HIGH, the zero flag will not contribute to signal **DERR**.

**M28 — M31: Reserved for future use (don't care).**

**M32 — M35 — Pipeline Timer Count:** In flow-through mode, this parameter specifies the number of clock cycles needed for data to traverse the ALU when performing any operation except multiply-accumulate; in pipelined mode, it specifies the number of cycles needed for data to traverse a single pipeline stage for any operation. The field can assume values between 3 and 15, inclusive, in flow-through mode; between 2 and 15, inclusive, in pipelined mode.

**M36 — M39 — Timer Count for the Multiply-Accumulate Operation:** In flow-through mode, this parameter specifies the number of clock cycles needed for data to traverse the ALU when performing a multiply-accumulate operation. The field can assume values between 3 and 15, inclusive.

**M40 — M43 — Timer Count for the Save State Transaction Request:** This parameter specifies the number of clock cycles needed to move data through the floating-point ALU when performing a **save state** transaction request. The field can assume values between 2 and 15, inclusive.

**M44 — Advance **DRDY**:** When M44 is HIGH, signal **DRDY** is advanced one clock cycle in flow-through mode. This bit has no effect in pipelined mode.

**M45 — M63: Reserved for future use (don't care).**

## Accelerator Transactions

The Am29000 controls the Am29027 with 13 transaction requests. Transaction type is indicated by the state of four signals: **R/W** and **OPT<sub>0</sub> — OPT<sub>2</sub>**. Table 1 lists the transaction types and the corresponding signal states.

TABLE 1. TRANSACTION REQUESTS

R/W	OPT <sub>2</sub>	OPT <sub>1</sub>	OPT <sub>0</sub>	Request Type
0	0	0	0	Write Operand R
0	0	0	1	Write Operand S
0	0	1	0	Write Operands R, S
0	0	1	1	Write Mode
0	1	0	0	Write Status
0	1	0	1	Write RF Precisions
0	1	1	0	Write Instruction
0	1	1	1	Restore Instruction
1	0	0	0	Read Results MSBs
1	0	0	1	Read Results LSBs
1	0	1	0	Read Flags
1	0	1	1	Read Status
1	1	0	0	Save State

Transactions are conditioned by signals **DREQT<sub>1</sub>** (which, when HIGH, indicates an accelerator transaction) and **BINV**. The Am29000 will recognize a transaction only if both **DREQT<sub>1</sub>** and **BINV** are HIGH.

### Write Transactions

There are eight available write transactions:

**Write Operand R:** An operand is written to input register R. The most significant half of the 64-bit word to be written is placed on port R, the least significant half on port S. If signal **DREQT<sub>0</sub>** is HIGH, operation execution will begin on the next rising edge of CLK.

**Write Operand S:** An operand is written to input register S. The most significant half of the 64-bit word to be written is placed on port R, the least significant half on port S. If signal **DREQT<sub>0</sub>** is HIGH, operation execution will begin on the next rising edge of CLK.

**Write Operands R, S:** Operands are written to input registers R and S. If signal **DREQT<sub>0</sub>** is LOW, two 32-bit half-operands are written to registers R-Temp and S-Temp. If **DREQT<sub>0</sub>** is HIGH, two 32-bit half-operands are written to the upper halves of registers R and S, and the contents of R-Temp and S-Temp are transferred to the lower halves of R and S; instruction execution will begin on the next rising edge of CLK.

**Write Mode:** A 64-bit word is written to the mode register. The most significant half of the mode word to be written is placed on port R, the least significant half on port S.

**Write Status:** A 6-bit word is written to the status register. The status word to be written is placed on the six LSBs of the R port as follows:

- R5 — Zero
- R4 — Inexact Result
- R3 — Underflow
- R2 — Overflow
- R1 — Reserved Operand
- R0 — Invalid Operation

**Write Register File Precisions:** An 8-bit word containing the precisions of register file locations RF0 – RF7 is written to the register file. The precision word to be written is placed on the eight LSBs of the R port, with bit 0 representing the precision of location RF0, bit 1 representing the precision of RF1, etc. A logic HIGH represents single precision, a logic LOW double precision.

**Write Instruction:** An instruction is written to the instruction register. If signal **DREQT<sub>0</sub>** is HIGH, operation execution will begin on the next rising edge of CLK.

**Restore Instruction:** An instruction is written to the instruction register, and the internal timer used to generate signals **DRDY** and **DERR** is restarted. This transaction request is used to restore an instruction at the end of a state restore sequence.

When issuing a **write operand R**; **write operand S**; **write operands R, S**; or **write instruction** transaction request, the Am29000 can command the Am29027 to begin an operation by asserting signal **DREQT<sub>0</sub>**; the operation will begin after receipt of data.

### Read Transactions

Five read transactions transfer data from the Am29027 to the Am29000. When data is to be transferred, the Am29000:

- Issues the appropriate transaction request on signals **OPT<sub>0</sub> – OPT<sub>2</sub>** and **R/W**.
- Places its data bus drivers in a high-impedance state.

The Am29027 then places the requested data on signals **F<sub>0</sub> – F<sub>31</sub>**, and issues one of two signals:

- **DRDY** indicates that valid data is available on signals **F<sub>0</sub> – F<sub>31</sub>**.
- **DERR** indicates that an error has occurred during a previous operation, and that the data on signals **F<sub>0</sub> – F<sub>31</sub>** may be erroneous.

If desired, the Am29000 can suppress the reporting of errors by asserting signal **DREQT<sub>0</sub>** when a read transaction request is issued.

There are five available read transactions:

**Read Result LSBs:** The 32 LSBs of register F are placed on port F, followed by the 32 MSBs.

**Read Result MSBs:** The 32 MSBs of register F are placed on port F.

**Read Flags:** The seven flag register bits are placed on port F. When outputting flags, the Am29027 will place a logic LOW on the most significant 25 bits of port F.

**Read Status:** The six status register bits are placed on port F. When outputting status, the Am29027 will place a logic LOW on the most significant 26 bits of port F.

**Save State:** The contents of the instruction register, mode register, status register, register file, and operand registers R, R-Temp, S, and S-Temp are transferred to the Am29000. Error reporting is suppressed for this transaction request. Further details on the use of this request appear in the Pipelining of Accelerator Operations section.

The Am29027 can be configured as a pipelined or unpipelined (flow-through) accelerator. The flow-through mode is normally selected when performing scalar operations, and is invoked by setting the mode register's "pipeline mode select" field (M20) to a logic LOW. To provide high throughput for vector operations, the Am29027 is configured as a pipelined accelerator by setting M20 to a logic HIGH.

## Programmer's Model

A programmer's model of the Am29027 in flow-through mode is shown in Figure 3. Note that register F and the flag register are made transparent in this mode.

Operations in flow-through mode are performed by:

- 1) Storing instruction and data in the Am29027, and starting the operation.
- 2) Loading the result.

Storing instruction and data can be done in any of three different ways:

- 1) **Writing the instruction only, and starting the operation:** This is appropriate when all the necessary operands are already stored in Am29027, as is sometimes the case when using on-board constants or the results of previous operations.
- 2) **Writing the data only, and starting the operation:** This is appropriate when the desired instruction is already present in Am29027, as is the case when performing the second of two identical operations.
- 3) **Writing the instruction and data, and starting the operation:** This is appropriate whenever the next operation requires both new instructions and data.

Operands and instructions are written using the **write operand R**, **write operand S**, **write operand R**, **write operand S**, and **write instruction** requests.

Loading an operation result is performed using the **read result MSBs**, **read result LSBs**, and **read flag register** transaction requests. The specific request used depends on whether the result of an operation is a flag or flags (as is the case with comparison operations) or data (as is the case with most other operations). In cases where the result is stored in the register file, the user may elect not to read the result, but to proceed with the next operation.



### Figure 3. Programmer's Model for Flow-Through Mode

## Error Recovery

Six ALU flags — overflow, underflow, zero, reserved operand, invalid operation, and inexact result — are used to update the status register after every operation. The status register latches each flag individually, such that, once the status register bit corresponding to a given flag is set, that bit remains set until the status register is altered by a write status transaction request.

If a status register bit is set, and if the corresponding mask bit in the mode register is inactive, the Am29027 will signal an error to the Am29000 by asserting signal **DERR** when the Am29000 performs a **read result LSBs**, **read result MSBs**, or **read flags** transaction request. The user can determine the precipitating error or errors by reading the status register with the **read status** transaction request. The status register can then be cleared (thus allowing further data reads) with a **write status** transaction request. Note that when the Am29027 asserts **DERR**, the data requested appears on the F output bus.

## Saving and Restoring State

In flow-through mode, the state of the Am29027 can be saved and restored. Saving of states is initiated with the **save state** transaction request; the first such request will return the contents of the instruction register. Subsequent **save state** transactions will return the contents of registers R, S, R-Temp, S-Temp, the status register, register file locations RF0 – RF7, and the mode register. The user has the option of saving only part of the state by issuing only the number of save state transactions needed to save registers of interest. When issuing a series of save state requests, data is returned in the following order:

Request	Data Returned
1	Instruction
2	R LSBs
3	R MSBs
4	S LSBs
5	S MSBs
6	R-Temp
7	S-Temp
8	Status
9	Register file precisions
10	RF0 LSBs
11	RF0 MSBs
.	.
.	.
24	RF7 LSBs
25	RF7 MSBs
26	Mode LSBs
27	Mode MSBs

Sequencing for the **save state** transaction request is reinitialized when the Am29000 issues any transaction request other than **save state**. If, for example, the Am29000 issues a **write operand R** transaction request, the next **save state** transaction would return the contents of the instruction register.

It should be noted that the process of saving state alters the contents of several Am29027 registers.

Error reporting via signal **DERR** is suppressed for the **save state** transaction.

Accelerator state is restored using the transaction requests in concert with the existing Am29027 "move operand" operations. Data is restored in the following order, using the following transaction requests:

Register To Be Restored	Procedure For Restoring
Mode	Write using <b>write mode</b> transaction request
RF0	Write "move R to RF0" instruction using <b>write instruction</b> transaction request Write RF0 value to register R using <b>write operand R</b> transaction request, start operation
RF7	Write "move R to RF7" instruction using <b>write instruction</b> transaction request Write RF7 value to register R using <b>write operand R</b> transaction request, start operation
Register File Precisions	Write using <b>write register file precisions</b> transaction request
Status	Write using <b>write status</b> transaction request
R	Write using <b>write operand R</b> transaction request
S	Write using <b>write operand S</b> transaction request
R-Temp, S-Temp	Write using <b>write operands R, S</b> transaction request
Instruction	Write using <b>restore instruction</b> transaction request

## Determining Timer Counts

In order to provide optimum accelerator performance over a wide range of possible system clock frequencies, the timing of Am29027 operations has been made programmable. Three mode register fields — the pipeline timer count, timer count for the multiply-accumulate operation, and timer count for the **save state** transaction request — must be programmed according to system clock frequency and accelerator speed.

**Pipeline Timer Count:** The pipeline timer count, mode register field M32 – M35, specifies the number of cycles allotted for operations other than multiply-accumulate. This count can assume values between 3 and 15 inclusive, and must be given a value that satisfies the relationship:

$$[8] \leq (\text{pipeline timer count}) * [1],$$

where

$$[8] = \text{Operation time, flow-through mode, all other operations}$$

$$\text{and } [1] = \text{CLK period,}$$

as described in the Switching Characteristics table.

**Timer Count for the Multiply-Accumulate Operation:** The timer count for the multiply-accumulate operation, mode register field M36 – M39, specifies the number of cycles allotted for operations of the form  $(P*Q) + T$ . This count can assume values between 3 and 15 inclusive, and must be given a value that satisfies the relationship:

$$[6] \leq (\text{timer count for mpy-acc. operation}) * [1],$$

where

$$[6] = \text{Operation time, flow-through mode, multiply-accumulate}$$

$$\text{and } [1] = \text{CLK period,}$$

as described in the Switching Characteristics table.

**Timer Count for the Save State Transaction Request:** The timer count for the **save state** transaction request, mode register field **M40 – M43**, specifies the number of cycles allotted for the output multiplexer to select the appropriate data in response to a **save state** transaction request. This count can assume values between 2 and 15 inclusive, and must be given a value that satisfies the relationship:

$$[7] \leq (\text{timer count for save state tr. req.}) * [1],$$

where

$$[7] = \text{Operation time, flow-through mode, save state}$$

$$\text{and } [1] = \text{CLK period,}$$

as described in the Switching Characteristics table.

#### Advancing **DRDY**

Normally, an operation result produced by the Am29027 in flow-through mode is read by the Am29000 host no sooner than the clock cycle following operation completion. Depending on the system clock frequency used, it may be advantageous to overlap the reading of the result with the last cycle of the operation. Consider, for example, a system with a 50-ns clock cycle and an accelerator that performs an operation in 240 ns. The pipeline timer count **M32 – M35** will have to be set to a minimum of 5 for such a system; the Am29000 will therefore read the result no sooner than during the sixth clock cycle following the start of the operation.

Mode register bit **M44**, Advance **DRDY**, can be used in such a case to advance transaction status signals **DRDY** and **DERR** by a full clock cycle, thus allowing the Am29000 to read the result a clock cycle earlier than would otherwise be possible. For the example given above, the pipeline timer count remains at 5, but the Am29000 can read the result during the fifth cycle after the operation starts, rather than the sixth, thus saving a clock cycle.

In order to advance **DRDY** and **DERR**, the following system timing conditions must be met:

$$[19] \leq (\text{timer count for mpy-acc. operation}) * [1] - [*9]$$

$$[20] \leq ((\text{timer count for save state tr. req.}) * [1]) - [*9]$$

$$[21] \leq ((\text{pipeline timer count}) * [1]) - [*9]$$

where

$$[19] = \text{Data operation-start-to-output-valid delay, multiply-accumulate}$$

$$[20] = \text{Data operation-start-to-output-valid delay, save state}$$

$$[21] = \text{Data operation-start-to-output-valid delay, all other operations}$$

$$\text{and } [1] = \text{CLK period,}$$

as described in the Switching Characteristics table, and

$$[*9] = \text{Am29000 synchronous input setup time}$$

as described in the Switching Characteristics table of the Am29000 Advance Information Data Sheet (PID 09075A/0).

## Operation in Pipelined Mode

### Programmer's Model

A programmer's model of Am29027 in pipelined mode is shown in Figure 4. In pipelined mode, register F and the flag register are made non-transparent, thus permitting the overlap of the current operation(s) with the reading of the results of the previous operation.

### Pipeline Delays

When placed in pipelined mode, the floating-point ALU of Am29027 incurs two pipeline delays for multiplication-accumulation, and a single pipeline delay for all other operations. The number of clock cycles allotted to each pipeline stage is determined by the pipeline timer count (mode register bits **M32 – M35**).

### Pipeline Advance Criteria

Pipelined operation in Am29000 accelerator mode is somewhat different than for flow-through mode. For the pipelined Am29000 accelerator mode, the pipe is advanced only when one of two events occur:

- 1) The Am29000 writes operands to registers R or S, or writes an instruction to the instruction register. Note that writing to R-Temp or S-Temp does not advance the pipe.
- 2) The Am29000 commands the Am29027 to begin a new operation by writing an instruction or operand with signal **DREQT<sub>0</sub>** HIGH.

Should these conditions occur in unison, the pipe is advanced only once. It is important to note that once the pipe is advanced, it cannot be advanced again until the next operation has begun.

One consequence of this conditional advance is that data does not fall through the pipe but instead is "pushed" through it. If, for example, an addition is performed in pipelined mode, the pipe must be advanced twice (by either of the means listed above) before the result of the addition is loaded into the register F, the flag register, the status register, and, optionally, register file locations **RF0 – RF7**.

### Performing Operations

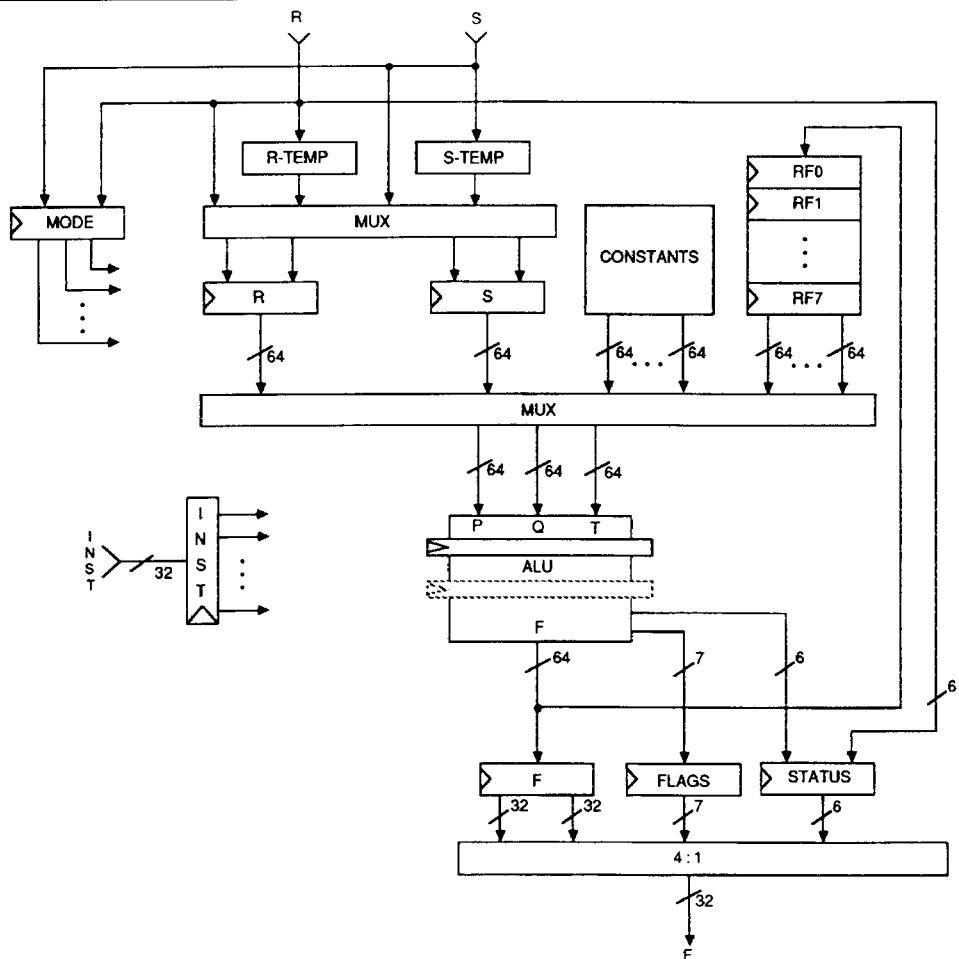
Operations in pipelined mode are performed by storing instruction and operand in the Am29027, and starting the operation. Optionally, the result of a previous operation may be read by the Am29000 before or after the current operation is started; the position of the transaction requests that read the result relative to transaction requests that advance the pipe will determine just which result is loaded.

### Writing Instructions and/or Operands

Same as for flow-through mode.

### Error Recovery

Same as for flow-through mode.



BD007131

Figure 4. Programmer's Model for Pipelined Mode

### Saving and Restoring State

It is not possible to save the complete state of the Am29027 when operating in pipelined mode. Pipelined operations may therefore be interrupted only under special circumstances, such as:

- 1) If the interrupting routine does not use the floating-point accelerator.
- 2) If the current series of pipelined operations has been completed, and any register contents needed for future operations have already been transferred to the Am29000.

The **save state** and **restore instruction** transaction requests cannot be used in pipelined mode.

### Determining Timer Counts

As with flow-through mode, the timing of operations in pipelined mode is programmable, to accommodate variations in system timing. Operation timing in pipelined mode is

controlled by the pipeline timer count, a field in the mode register.

The pipeline timer count, mode register field **M32-M35**, specifies the number of cycles allotted for all operations. This count can assume values between 2 and 15 inclusive, and must be given a value that satisfies the relationship:

$$[9] \leq (\text{pipeline timer count}) * [1],$$

where

[9] = Operation time, pipelined mode, all operations  
and [1] = CLK period,

as described in the Switching Characteristics table.

### Advancing **DRDY**

Because the Am29027 F register and flag register are non-transparent in pipelined mode, it not possible (nor advantageous) to advance **DRDY**. Accordingly, mode register bit **M44** has no effect in pipelined mode.



## Restrictions on Instruction Ordering

When using the Am29027 in pipelined mode, two conditions must be observed:

1) The mode register bits are not pipelined: when the mode register is loaded, any differences between the current mode and the previous mode take effect immediately. When changing mode, then, the user must take care to first read the result of all operations currently in the pipe, or risk corrupting those results. The user can read the results of all operations currently in the pipe by executing the requisite number of "NO-OP" operations needed to push data out of the pipe, a "NO-OP" operation being any operation whose result is not stored in the register file.

2) A multiplication-accumulation operation cannot be immediately followed by another type of operation (e.g., an addition or multiplication). This problem can be avoided by executing an extra multiplication-accumulation operation at the end of a series of such operations.

## Accelerator Instruction Set

### Instruction Word

The 32-bit instruction word ( $I_0 - I_{31}$ ) specifies the instructions to be performed by the ALU, and comprises six distinct sections: opcode, integer/floating-point control, sign-change controls, operand selection multiplexer controls, register file controls, and operand precision controls. The instruction word is loaded into the instruction register by the Am29000 via the Instruction bus, using the **write instruction** or **restore instruction** transaction requests. The format of the instruction register is shown in Figure 5, and is described below:

**I0 - I4 — Opcode:** Specifies the operation to be performed by the ALU.

**I5 — Integer/Floating-Point Select:** Specifies whether an operation is performed in integer or floating-point format.

**I6, I7 — Sign F:** Sign change control for the ALU output.

**I8, I9 — Sign T:** Sign change control for the ALU T input.

**I10, I11 — Sign Q:** Sign change control for the ALU Q input.

**I12, I13 — Sign P:** Sign change control for the ALU P input.

**I14 - I17 — Select for T Operand Multiplexer:** Selects the data input to the ALU T-port.

**I18 - I21 — Select for Q Operand Multiplexer:** Selects the data input to the ALU Q-port.

**I22 - I25 — Select for P Operand Multiplexer:** Selects the data input to the ALU P-port.

**I26 - I28 — Register File Select:** Selects the register file location (RF0 - RF7) to which the result is to be written.

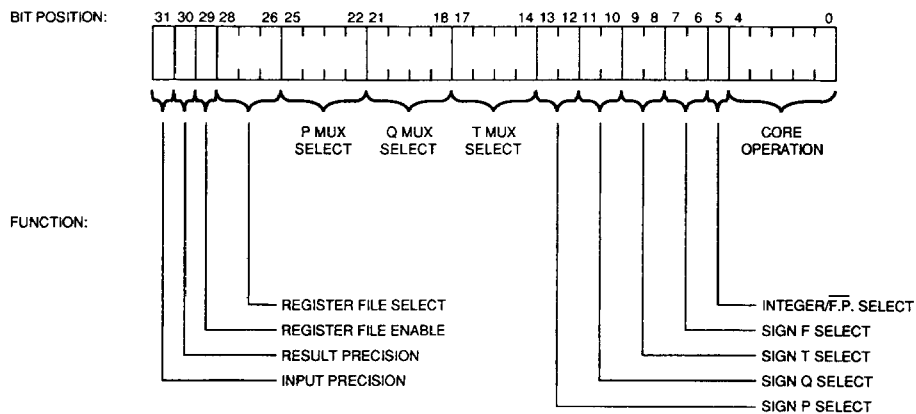
**I29 — Register File Enable:** Enables a write to the register file at end of operation. When I29 is LOW, the operation result is written to the register file location specified by I26 - I28. When I29 is HIGH, register file writes are disabled.

**I30 — Result Precision:** Precision of the ALU output; single-precision when HIGH, double-precision when LOW.

**I31 — Input Precision:** Precision of the input operands loaded into registers R and S; single-precision when HIGH, double-precision when LOW.

### Instruction Register Format

The Instruction Register Format is given in Figure 5.



TB001020

Figure 5. Instruction Register Format

## Floating-Point and Integer Opcodes

floating-point or integer format. The core operations and their corresponding opcodes are listed below:

The opcode field, I4 – I0, specifies the core operation to be performed by the ALU; instruction bit I5 selects between

I5	I4	I3	I2	I1	I0	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
I5	I4	I3	I2	I1	I0	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

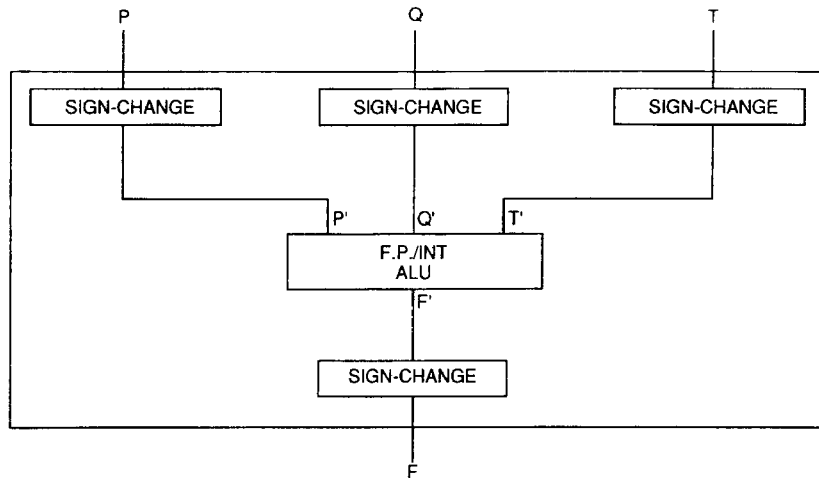
Core operations MOVE P and LOAD MODE REGISTER can both be performed in either floating-point or integer format:

I5	I4	I3	I2	I1	I0	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 sign (see Figure 6). 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.



AF004730

Figure 6. ALU Sign-Change Blocks

TABLE 2-1. SELECT DECODE FOR P OPERAND SIGN-CHANGE BLOCK

I13	I12	Sign (P')
0	0	SIGN (P)
0	1	$\overline{\text{SIGN}} (P)$
1	0	0
1	1	1

TABLE 2-2. SELECT DECODE FOR Q OPERAND SIGN-CHANGE BLOCK

I11	I10	Sign (Q')
0	0	SIGN (Q)
0	1	$\overline{\text{SIGN}} (Q)$
1	0	0
1	1	1

TABLE 2-3. SELECT DECODE FOR T OPERAND SIGN-CHANGE BLOCK

C9	C8	Sign (T')
0	0	SIGN T
0	1	$\overline{\text{SIGN}} T$
1	0	0
1	1	1

TABLE 2-4. SELECT DECODE FOR F OPERAND SIGN-CHANGE BLOCK

Core Operation	I11	I10	I7	I6	Sign (F)
P,	0	0	x	x	SIGN (F')
Max P, T	0	1	x	x	$\overline{\text{SIGN}} (F')$
or	1	0	x	x	SIGN (P)
Min P, T	1	1	x	x	SIGN (T)
Other	x	x	0	0	SIGN (F')
	x	x	0	1	$\overline{\text{SIGN}} (F')$
	x	x	1	0	0
	x	x	1	1	1

## Operand Multiplexer Selects

Instruction fields I22 – I25, I18 – I21, and I14 – I17 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**

I25 I21 I17	I24 I20 I16	I23 I19 I15	I22 I18 I14	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 Am29027 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 instruction bit I31. A logic HIGH indicates a single-precision operand or operands; a LOW, double precision. Note that the operands in the R and S registers must have the same precision. This does not preclude performing an operation with operands of different precision, as there are no restrictions on the precisions of operands stored in the register file.

Precision of an operation result is specified by instruction bit I30. 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 information is automatically supplied to the ALU when a register file location is used as an input operand to an operation.

## Accelerator 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; register file selects and enable; and the input and output operand precisions — can be specified independently.

The user may create his own 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/F	Opcode
	P	Q	T	F		
FP <b>P</b>	00	00	xx	00	0	00000
FP <b>-P</b>	00	00	xx	01	0	00000
FP <b>ABS (P)</b>	00	00	xx	10	0	00000
FP <b>Sign (T)*ABS (P)</b>	00	11	xx	xx	0	00000
FP <b>P + T</b>	00	xx	00	00	0	00001
FP <b>P - T</b>	00	xx	01	00	0	00001
FP <b>T - P</b>	01	xx	00	00	0	00001
FP <b>-P - T</b>	01	xx	01	00	0	00001
FP <b>ABS (P + T)</b>	00	xx	00	10	0	00001
FP <b>ABS (P - T)</b>	00	xx	01	10	0	00001
FP <b>ABS (P) + ABS (T)</b>	10	xx	10	00	0	00001
FP <b>ABS (P) - ABS (T)</b>	10	xx	11	00	0	00001
FP <b>ABS (ABS (P) - ABS (T))</b>	10	xx	11	10	0	00001
FP <b>P * Q</b>	00	00	xx	00	0	00010
FP <b>(-P) * Q</b>	01	00	xx	00	0	00010
FP <b>ABS (P * Q)</b>	00	00	xx	10	0	00010
FP <b>Compare P, T</b>	00	xx	01	00	0	00011
FP <b>Max P, T</b>	00	00	01	00	0	00100
FP <b>Max ABS (P), ABS (T)</b>	10	00	11	00	0	00100
FP <b>Min P, T</b>	01	00	00	00	0	00101
FP <b>Min ABS (P), ABS (T)</b>	11	00	10	00	0	00101
FP <b>Limit P to Magnitude T</b>	11	10	10	xx	0	00101
FP <b>Convert T to Integer</b>	xx	xx	00	00	0	00110
FP <b>Scale T to Integer by Q</b>	xx	00	00	00	0	00111
FP <b>T + P*Q</b>	00	00	00	00	0	01000
FP <b>T - P*Q</b>	01	00	00	00	0	01000
FP <b>-T + P*Q</b>	00	00	01	00	0	01000
FP <b>-T - P*Q</b>	01	00	01	00	0	01000
FP <b>ABS (T) + ABS (P*Q)</b>	10	10	10	00	0	01000
FP <b>ABS (T) - ABS (P*Q)</b>	11	10	10	00	0	01000
FP <b>ABS (P*Q) - ABS (T)</b>	10	10	11	00	0	01000
FP <b>Round T to Integral Value</b>	xx	xx	00	00	0	01001
FP <b>Reciprocal Seed (P)</b>	00	xx	xx	00	0	01010
FP <b>Convert T to Alternate Floating-point Format</b>	xx	xx	00	00	0	01011
FP <b>Convert T from Alternate Floating-point Format</b>	xx	xx	00	00	0	01100

TABLE 4. INSTRUCTION WORDS (Cont'd.)

Operation	Sign				I/F	Opcode
	P	Q	T	F		
Int <b>P</b>	00	00	00	00	1	00000
Int <b>-P</b>	00	00	00	01	1	00000
Int <b>ABS (P)</b>	00	00	00	10	1	00000
Int <b>sign (T)*ABS (P)</b>	00	11	00	xx	1	00000
Int <b>P + T</b>	00	xx	00	00	1	00001
Int <b>P - T</b>	00	xx	01	00	1	00001
Int <b>T - P</b>	01	xx	00	00	1	00001
Int <b>ABS (P + T)</b>	00	xx	00	10	1	00001
Int <b>ABS (P - T)</b>	00	xx	01	10	1	00001
Int <b>P * Q</b>	00	00	xx	00	1	00010
Int <b>Compare P, T</b>	00	xx	01	00	1	00011
Int <b>Max P, T</b>	00	00	01	00	1	00100
Int <b>Min P, T</b>	01	00	00	00	1	00101
Int <b>Convert T to Float</b>	xx	xx	00	00	1	00110
Int <b>Scale T to Float by Q</b>	xx	00	00	00	1	00111
Int <b>P OR T</b>	xx	xx	xx	xx	1	10000
Int <b>P AND T</b>	xx	xx	xx	xx	1	10001
Int <b>P XOR T</b>	xx	xx	xx	xx	1	10010
Int <b>NOT T (see Note 1)</b>	xx	xx	xx	xx	1	10010
Int <b>Shift P Logical Q Places</b>	00	00	xx	00	1	10011
Int <b>Shift P Arithmetic Q Places</b>	00	00	xx	00	1	10100
Int <b>Funnel Shift PT Q Places</b>	00	00	00	00	1	10101
<b>Move P</b>	xx	xx	xx	xx	x	11000
<b>Load Mode Register</b>	xx	xx	xx	xx	x	11111

Notes: 1. NOT T is performed by XORing T with a word containing all 1s (integer -1). When invoking NOT T the user must set I22-I25 to 0011<sub>2</sub>, 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 larger 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 smaller 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 value T is converted to integer format. The operation therefore permits T to be scaled by any multiple of 2 when the source format is IEEE or DEC; and by any multiple 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; alternately, an external seed look-up table can be used for faster convergence.

**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 I22 - I25 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 no flags are set.

## Primary and Alternate Floating-Point Formats

All floating-point operations except format conversions are performed in the primary format selected by mode register bits M0 and M1. Conversions to the alternate format convert floating-point numbers from the primary format to the alternate format selected by mode register bits M2 and M3. Conversions from the alternate format convert floating-point data from the alternate format to the primary format. Conversions and scale operations to and from integer format operate on floating-point numbers in the primary format.

## 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 flag 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 arguments is the larger or the smaller 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 describes the flags reported for each operation.

It should be noted that in flow-through mode, the flag register is made transparent, and the selected flags are presented directly to the output multiplexer; however, flag selection is not affected.



TABLE 6. ORGANIZATION OF FLAGS

Operations		Opcode I4-I0	Flag Register						
			LSB						MSB
			0	1	2	3	4	5	
IEEE	Non-arithmetic single-operand	00000	I	R	V	U	X	Z	S
IEEE	Operations using add	00001	I	R	V	U	X	Z	S
IEEE	Operations using multiply	00010	I	R	V	U	X	Z	S
IEEE	Compare	00011	I	R	#	<	>	=	S
IEEE	Maximum, minimum, limit	0010x	I	R		W		Z	S
IEEE	Convert/scale to integer	0011x	I	R	V		X	Z	S
IEEE	Multiply/accumulate	01000	I	R	V	U		Z	S
IEEE	Round to integral value	01001	I	R	V		X	Z	S
IEEE	Reciprocal seed	01010	I	R	V	U		Z	S
IEEE	Convert to alt f.p. format	01011	I	R	V	U	X	Z	S
IEEE	Convert from alt. f.p. format	01100	I	R	V	U	X	Z	S
DEC D	Non-arithmetic single-operand	00000		R	V		X	Z	S
DEC D	Operations using add	00001		R	V	U	X	Z	S
DEC D	Operations using multiply	00010		R	V	U	X	Z	S
DEC D	Compare	00011		R	#	<	>	=	S
DEC D	Maximum, minimum, limit	0010x		R		W		Z	S
DEC D	Convert/scale to integer	0011x	I	R	V		X	Z	S
DEC D	Multiply/accumulate	01000		R	V	U		Z	S
DEC D	Round to integral value	01001		R	V		X	Z	S
DEC D	Reciprocal seed	01010	I	R	V	U		Z	S
DEC D	Convert to alt f.p. format	01011	I	R	V	U	X	Z	S
DEC D	Convert from alt. f.p. format	01100	I	R	V	U	X	Z	S
DEC G	Non-arithmetic single-operand	00000		R	V	U	X	Z	S
DEC G	Operations using add	00001		R	V	U	X	Z	S
DEC G	Operations using multiply	00010		R	V	U	X	Z	S
DEC G	Compare	00011		R	#	<	>	=	S
DEC G	Maximum, minimum, limit	0010x		R		W		Z	S
DEC G	Convert/scale to integer	0011x	I	R	V		X	Z	S
DEC G	Multiply/accumulate	01000		R	V	U		Z	S
DEC G	Round to integral value	01001		R	V		X	Z	S
DEC G	Reciprocal seed	01010	I	R	V	U		Z	S
DEC G	Convert to alt f.p. format	01011	I	R	V	U	X	Z	S
DEC G	Convert from alt. f.p. format	01100	I	R	V	U	X	Z	S
IBM	Non-arithmetic single-operand	00000			V		X	Z	S
IBM	Operations using add	00001			V	U	X	Z	S
IBM	Operations using multiply	00010			V	U	X	Z	S
IBM	Compare	00011				<	>	=	S
IBM	Maximum, minimum, limit	0010x				W		Z	S
IBM	Convert/scale to integer	0011x			V		X	Z	S
IBM	Multiply/accumulate	01000			V	U		Z	S
IBM	Round to integral value	01001			V		X	Z	S
IBM	Reciprocal seed	01010	I		V	U		Z	S
IBM	Convert to alt f.p. format	01011			V	U	X	Z	S
IBM	Convert from alt. f.p. format	01100	I	R	V	U	X	Z	S
Integer	Non-arithmetic single-operand	00000			V			Z	S
Integer	Sign transfer	00000			V			Z	S
Integer	Operations using add	00001	C		V			Z	S
Integer	Operations using multiply	00010			V			Z	S
Integer	Compare operations	00011	C		V	<	>	=	S
Integer	Maximum, minimum, limit	0010x				W		Z	S
Integer	Convert to float	00110					X	Z	S
Integer	Scale to float	00111		R	V	U	X	Z	S
Integer	Logical operations	100xx						Z	S
Integer	Arithmetic shift	10100			V			Z	S
Integer	Funnel shift	10101						Z	S
Move operand		11000							
Load mode register		11111							

Note: unused flags assume the LOW state.

## Status Register

The status register latches flags over a series of operations, so that errors occurring during an operation sequence can be reported after the last operation. This ability is particularly useful for those applications in which intermediate results are not read by the Am29000 (at which time an error would be reported), or for situations where checking each operation result for errors would be prohibitively time-consuming.

Six of the previously described ALU flags — I, R, U, V, X, and Z — are fed to corresponding bits in the status register at the conclusion of each operation. The status register latches each flag individually, such that, once the status register bit corresponding to a given flag is set, that bit remains set until the status register is altered by a **write status** transaction request.

If a status register bit is set, and if the corresponding mask bit in the mode register is inactive, the Am29027 will signal an error to the Am29000 by asserting signal **DERR** when the Am29000 performs a read transaction request other than **save state**; if signal **DREQT0** is HIGH, error reporting is suppressed. The user can determine the precipitating error or

errors by reading the status register using the **read status** transaction request.

## Master/Slave Operation

Two Am29027 accelerators 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 **SLAVE** and **MSERR**, are tied to the corresponding signals of the master. The master is selected by asserting signal **SLAVE LOW**; the slave, by asserting signal **SLAVE HIGH**.

The slave accelerator, by comparing its outputs to the outputs of the master accelerator, performs a comprehensive check of the operation of the master accelerator. In addition, if the slave accelerator is connected at the proper position on the channel, it may detect open circuits and other faults in the electrical path between the master accelerator and its host. Note that the master accelerator 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

$2^{31} 2^{30} 2^{29} 2^{28} 2^{27} 2^{26} 2^{25} . . . . . 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0$

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

$2^{63} 2^{62} 2^{61} 2^{60} 2^{59} 2^{58} 2^{57} . . . . . 2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0$

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 inte-

ger 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:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	. . .	3	2	1	0
s	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	$2^1$	$2^2$	$2^3$	$2^4$	$2^5$	. . .	$2^{20}$	$2^{21}$	$2^{22}$	$2^{23}$
sign	biased exponent (e)								fraction (f)									

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 ≠ 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 ≠ 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**.

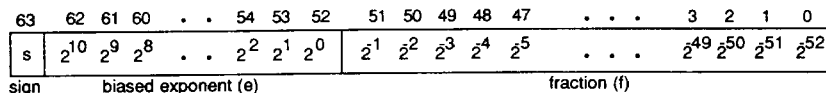
**NaN:** A NaN is interpreted as a signal or symbol. NaNs are used to indicate invalid operations, and as a means of passing process status through a series of calculations. They arise in

two ways: either generated by the Am29027 to indicate an invalid operation, or provided by the user as an input. A **signalling 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 P754.

## 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

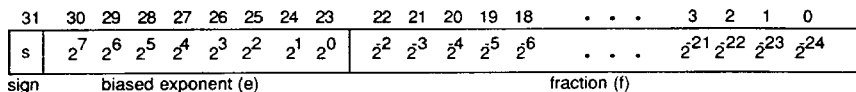
two ways: either generated by the Am29027 to indicate an invalid operation, or provided by the user as an input. A **signalling NaN** has the MSB of its fraction set to 0 and at least one of the remaining fraction bits set to 1. A **quiet NaN** has the MSB of its fraction set to 1.

The IEEE format is fully described in IEEE Standard P754.

## 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:

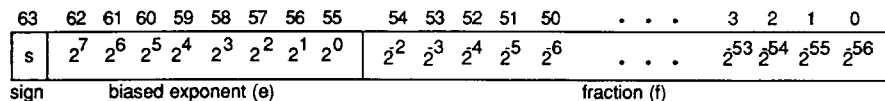
If $e \neq 0$ .....	value $\neq (-1)^s 2^{e-128}(0.f)$
If $s = 0$ and $e = 0$ .....	value = 0
If $s = 1$ and $e = 0$ .....	value = DEC-Reserved Operand

**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 =  $(-1)^s 2^e - 128 (0.f)$

If  $s = 0$  and  $e = 0$  ..... value = 0

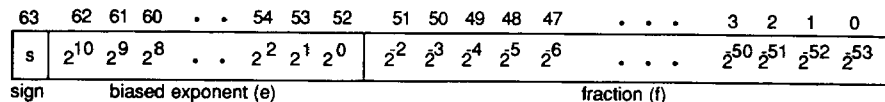
If  $s = 1$  and  $e = 0$  ..... value = DEC-Reserved Operand

**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 =  $(-1)^s 2^e - 1024 (0.f)$

If  $s = 0$  and  $e = 0$  ..... value = 0

If  $s = 1$  and  $e = 0$  ..... value = DEC-Reserved Operand

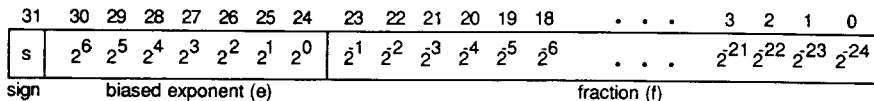
**DEC-Reserved Operand:** A Dec-Reserved Operand is interpreted as a signal or symbol. DEC-Reserved Operands are used to indicate invalid operations and operations whose results have overflowed the destination format. They may also be used to pass symbolic information from one calculation to another.

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:

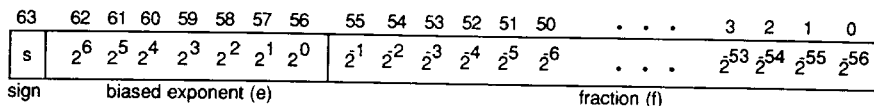
$$\text{value} = (-1)^s 16^e - 64(0.f)$$

**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 Am29027 provides six rounding modes for floating-point operations, and for integer multiplication:

### Round to Nearest (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 round 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 round 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 round mode conforms to the "round to zero" mode described in the IEEE Floating Point Standard.

### Round to Nearest (Biased)

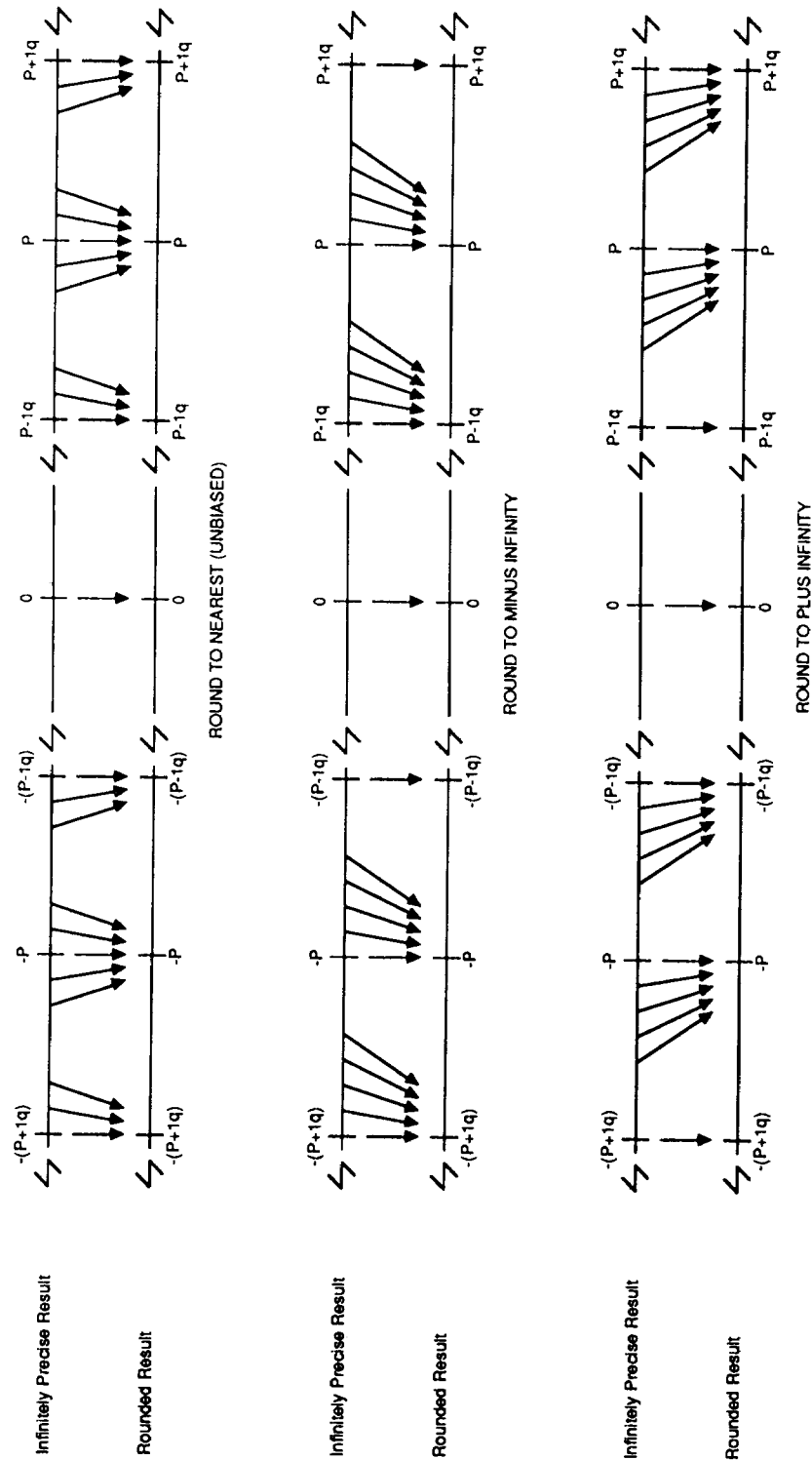
The infinitely precise result of an operation is rounded to the closest representable value in the destination format. If the infinitely precise result is exactly halfway between two representations, it is rounded to the representation having the greater **magnitude**. This round mode is used by DEC 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 round modes are shown in Figures B1-1 and B1-2.

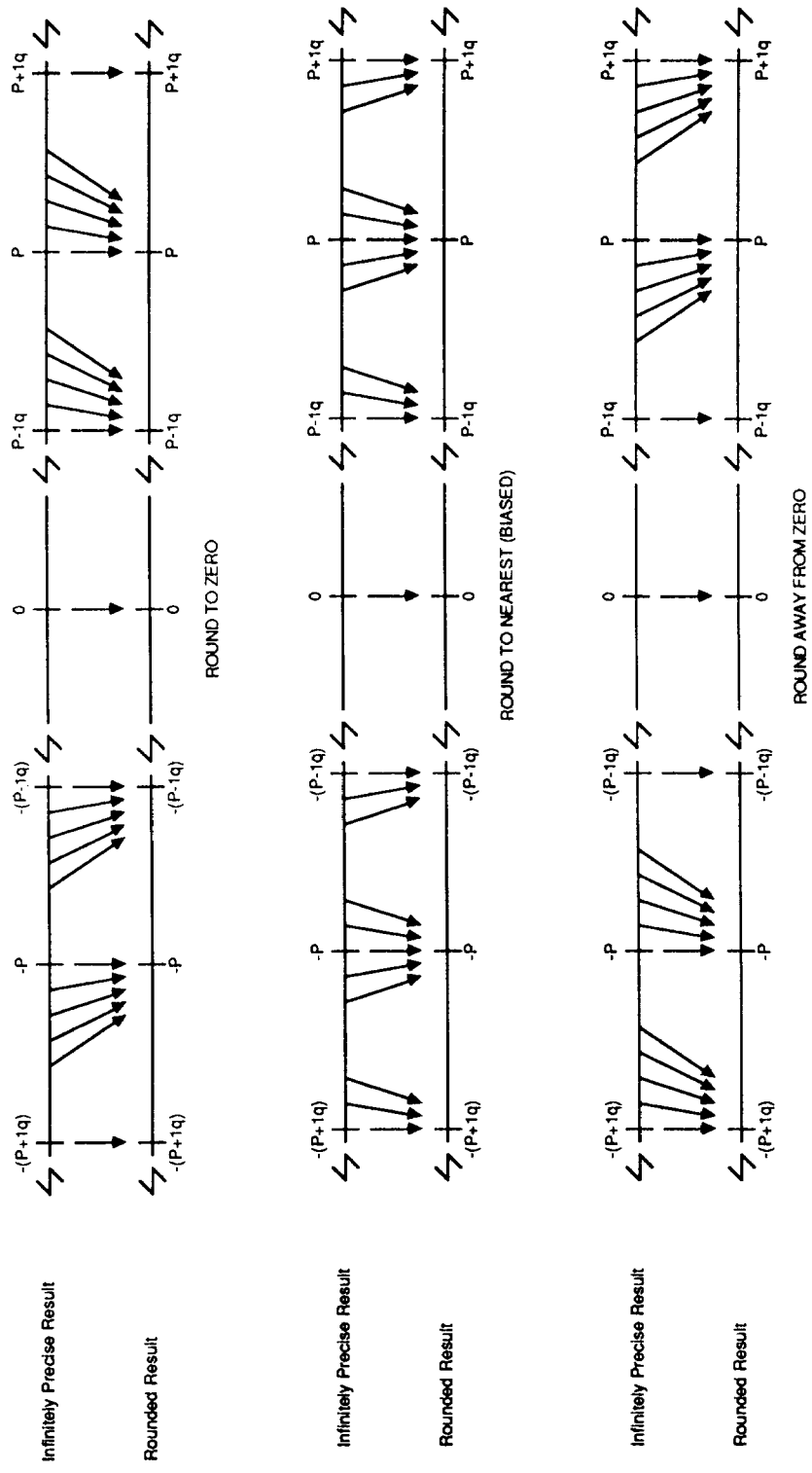
Round mode is selected with mode register bits M14 – 16, as described in Mode Register Description section.



PF002470

**Figure B1-1. Graphical Interpretation of Round-to-Nearest (Unbiased), Round-to-Minus-Infinity, and Round-to-Plus-Infinity Rounding Modes**





PF002480

Figure B1-2. Graphical Interpretation of Round-to-Zero, Round-to-Nearest, and Round-Away-from-Zero Rounding Modes

## Appendix C — Additional Operation Details

### Differences Between IEEE Floating-Point Standard and Am29027 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 Am29027 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 Am29027 IBM Operation

For addition operations, the IBM 370 retains only one guard digit. The Am29027 retains one or more guard digits, so that

the final result of an addition is the infinitely precise result, rounded to the destination format.

### Differences Between DEC Floating-Point Arithmetic and Am29027 DEC Operation

The Am29027 and DEC VAX floating-point formats contain identical information, but the sub-fields of the floating-point words are arranged differently:

The Am29027 DEC F format is:  
sign – bit 31  
exponent – bits 30 – 23  
mantissa – bits 22 – 0

The Am29027 DEC D format is:  
sign – bit 63  
exponent – bits 62 – 55  
mantissa – bits 54 – 0

The Am29027 DEC G format is:  
sign – bit 63  
exponent – bits 62 – 52  
mantissa – bits 51 – 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

The VAX format is:  
sign – bit 15  
exponent – bits 14 – 4  
mantissa – bits 3 – 0,  
bits 31 – 16,  
bits 47 – 42,  
bits 63 – 48

**ABSOLUTE MAXIMUM RATINGS**

Storage Temperature ..... -55 to +150°C  
 Voltage on Any Pin  
 with Respect to GND ..... -0.5 to +7.0 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<sub>A</sub>) ..... 0 to +70°C  
 Supply Voltage (V<sub>CC</sub>) ..... +4.5 to +5.5 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	Min.	Max.	Units
V <sub>IL</sub>	Input LOW Voltage		-0.5	0.8	V
V <sub>IH</sub>	Input HIGH Voltage		2.0	V <sub>CC</sub> + 0.5	V
V <sub>ILC</sub>	CLK Input LOW Voltage		-0.5	0.6	V
V <sub>IHC</sub>	CLK Input HIGH Voltage		3.8	V <sub>CC</sub> + 0.5	V
V <sub>OL</sub>	Output LOW Voltage Command Outputs, Control Outputs			.45	V
V <sub>OH</sub>	Output HIGH Voltage Command Outputs, Control Outputs		2.4		V
I <sub>F</sub>	Input Current (S0, S1, and M/I <sub>O</sub> Inputs)	V <sub>F</sub> = 4.5 V		-0.3	mA
I <sub>IL</sub>	Input Leakage Current (All Other Inputs)	0 V ≤ V <sub>IN</sub> ≤ V <sub>CC</sub>		±10	μA
I <sub>LO</sub>	Output Leakage Current	.45 V ≤ V <sub>OUT</sub> ≤ V <sub>CC</sub>		±10	μA
I <sub>CCSB</sub>	Standby Power-Supply Current	V <sub>CC</sub> = 5.5 V, V <sub>IN</sub> = V <sub>CC</sub> or GND, Outputs Open			μA
I <sub>CCOP</sub>	Operating Power-Supply Current	V <sub>CC</sub> = 5.5 V, Outputs Open			mA/MHz

# SWITCHING CHARACTERISTICS over operating ranges unless otherwise specified






No.	Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Units
1		CLK Period	(Note 1)	40	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		Operation Time, Low-Latency Mode				
7		Multiply-Accumulate			360	ns
8		Save State			120	ns
		All Other Operations			240	ns
9		Operation Time, Pipelined Mode				
		All Operations			120	ns
10		Transaction Request Setup Time	(Note 3)			ns
11		Transaction Request Hold Time	(Note 3)		0	ns
12		BINV Setup Time				ns
13		BINV Hold Time			0	ns
14		Data/Instruction Setup Time	(Note 4)			ns
15		Data/Instruction Hold Time	(Note 4)		0	ns
16		CDA CLK-to-Output-Valid Delay				ns
17		F <sub>0</sub> - F <sub>31</sub> CLK-to-Output-Valid Delay				ns
18		F <sub>0</sub> - F <sub>31</sub> Three-State CLK-to-Output-Inactive Delay				ns
19		Data Operation-Start-to-Output				
20		Valid Delay	(Note 5)			ns
		Multiply-Accumulate	(Note 5)			ns
21		Save State	(Note 5)			ns
		All Other Operations	(Note 5)			ns
22		DRDY, DERR CLK-to-Output-Valid Delay				ns
23		MSERR CLK-to-Output-Valid Delay				ns

- Notes:**
1. CLK switching characteristics are made relative to 2.5 V.
  2. CLK rise time/fall time measured between 0.8 V and (V<sub>CC</sub> - 1.0 V).
  3. Transaction request signals include R/W, DREQT<sub>0</sub> - DREQT<sub>1</sub>, and OPT<sub>0</sub> - OPT<sub>2</sub>.
  4. Data/instruction signals include R<sub>0</sub> - R<sub>31</sub>, S<sub>0</sub> - S<sub>31</sub>, and I<sub>0</sub> - I<sub>31</sub>.
  5. This parameter relevant to flow-through mode only.

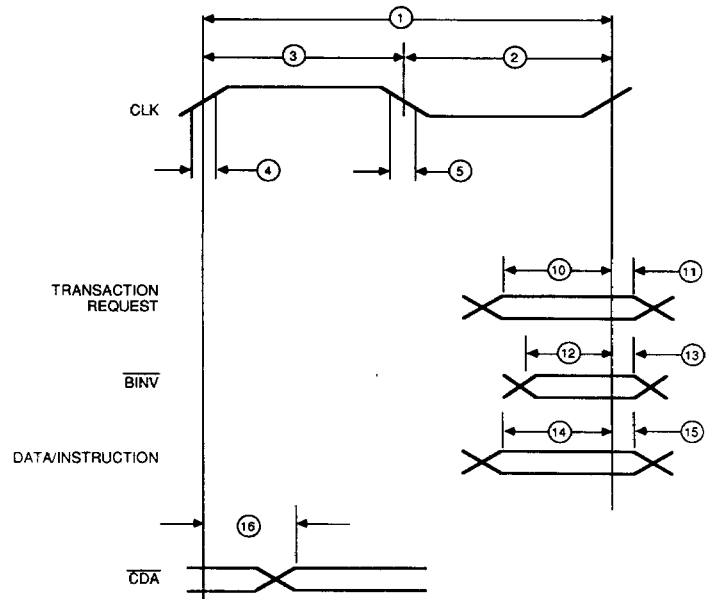
- Conditions:**
- A. All inputs/outputs except CLK are TTL-compatible for V<sub>IH</sub>, V<sub>IL</sub>, and V<sub>OL</sub>.
  - B. All outputs are driving 80 pF unless otherwise noted.
  - C. All setup, hold, and delay times are measured relative to CLK at V<sub>CC</sub>/2 volts unless otherwise noted.

# 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

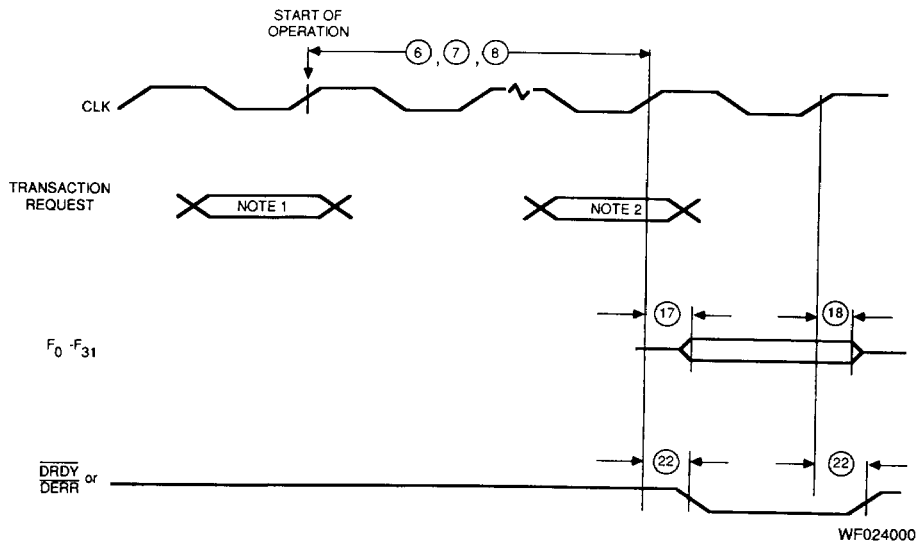
KS000010



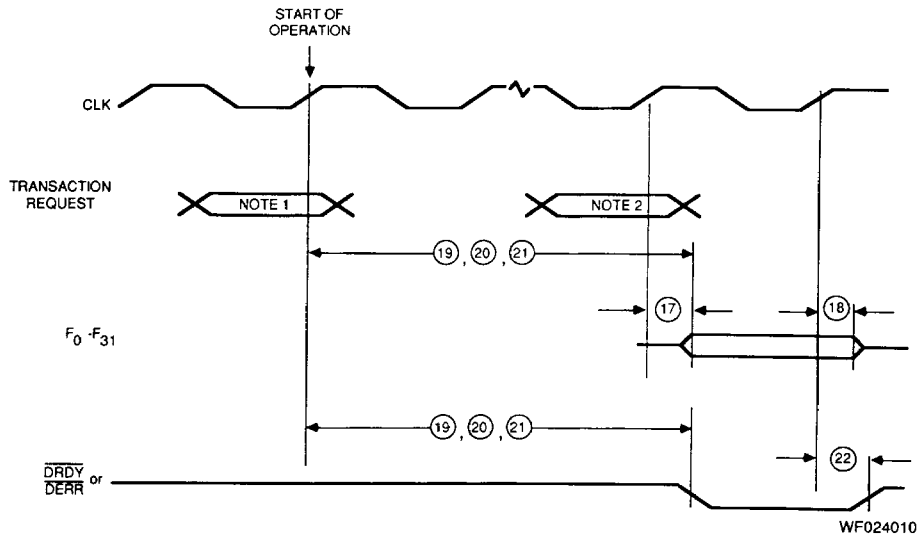
WF023850

Input Signal Timing, CDA Timing

## SWITCHING WAVEFORMS (Cont'd.)



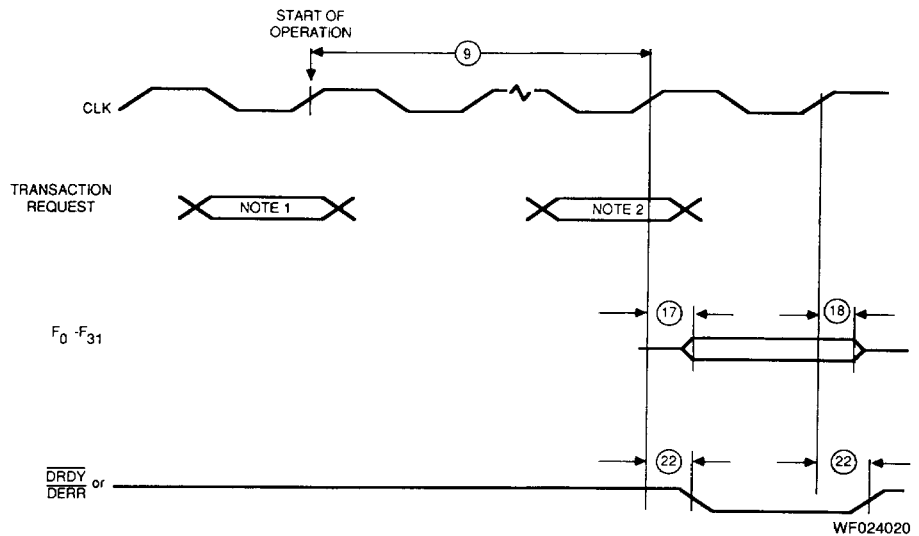
Operation Timing for Flow-Through Mode,  $\overline{DRDY}$ ,  $\overline{DERR}$  Not Advanced  
(Mode Register Bit M44 = LOW)



Operation Timing for Flow-Through Mode,  $\overline{DRDY}$ ,  $\overline{DERR}$  Advanced  
(Mode Register Bit M44 = HIGH)

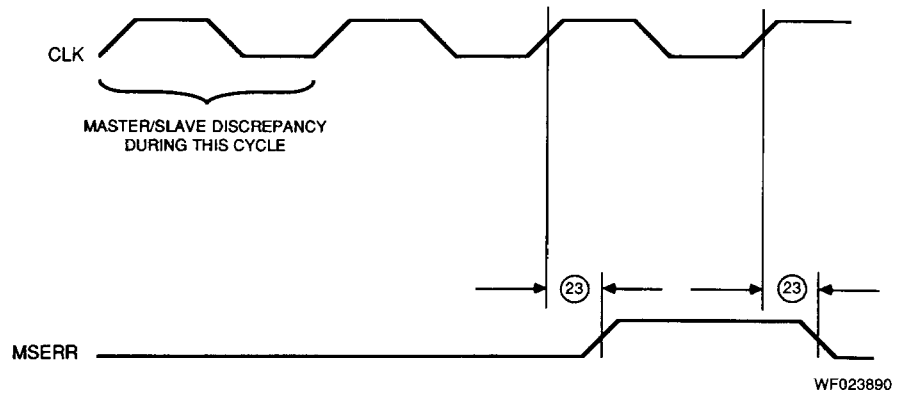
- Notes:**
1. Transaction request **Write Operand R**; **Write Operand S**; **Write Operands R, S**; or **Write Instruction** with Signal DREQT<sub>0</sub> active.
  2. Transaction Request **Read Result MSBs**, **Read Result LSBs**, **Read Flags**, **Read Status**, or **Save State**. If request **Read Result LSBs** is issued, the Am29027 produces two data outputs in two consecutive cycles, with  $\overline{DRDY}$  or  $\overline{DERR}$  active for both cycles.

## SWITCHING WAVEFORMS (Cont'd.)



### Operation Timing for Pipelined Mode

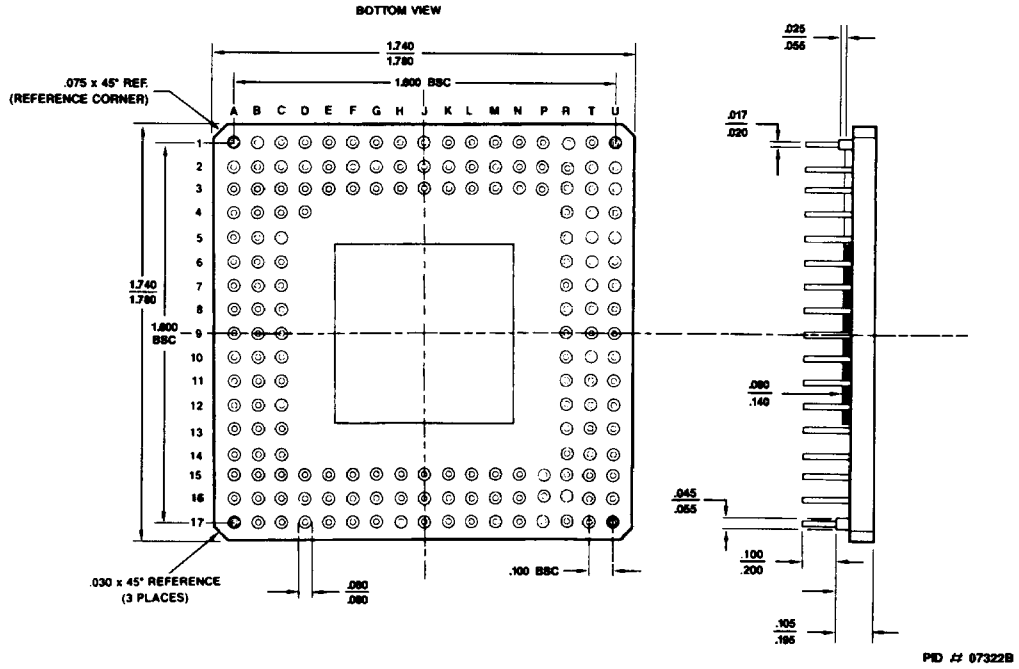
- Notes:**
1. Transaction request **Write Operand R**; **Write Operand S**; **Write Operands R, S**; or **Write Instruction** with Signal **DREQ<sub>0</sub>** active.
  2. Transaction Request **Read Result MSBs**, **Read Result LSBs**, **Read Flags**, **Read Status**, or **Save State**. If request **Read Result LSBs** is issued, the Am29027 produces two data outputs in two consecutive cycles, with **DRDY** or **DERR** active for both cycles.



### Master/Slave Timing

# PHYSICAL DIMENSIONS\*

CGX169



\*For reference only.



# ADVANCED MICRO DEVICES

## U.S. SALES OFFICES

ALABAMA	(205) 882-9122	MASSACHUSETTS	(617) 273-3970
ARIZONA		MINNESOTA	(612) 938-0001
Tempe	(602) 242-4400	MISSOURI	(314) 275-4415
CALIFORNIA		NEW JERSEY	(201) 299-0002
Culver City	(213) 645-1524	NEW YORK	
Newport Beach	(714) 752-6262	Liverpool	(315) 457-5400
San Diego	(619) 560-7030	Poughkeepsie	(914) 471-8180
Sunnyvale	(408) 720-8811	Woodbury	(516) 364-8020
Woodland Hills	(818) 992-4155	NORTH CAROLINA	(919) 847-8471
COLORADO	(303) 741-2900	OREGON	(503) 245-0080
CONNECTICUT	(203) 264-7800	OHIO	(614) 891-6455
FLORIDA		PENNSYLVANIA	
Clearwater	(813) 530-9971	Allentown	(215) 398-8006
Ft Lauderdale	(305) 484-8600	Willow Grove	(215) 657-3101
Melbourne	(305) 729-0496	TEXAS	
Orlando	(305) 859-0831	Austin	(512) 346-7830
GEORGIA	(404) 449-7920	Dallas	(214) 934-9099
ILLINOIS	(312) 773-4422	Houston	(713) 785-9001
INDIANA	(317) 244-7207	WASHINGTON	(206) 455-3600
KANSAS	(913) 451-3115	WISCONSIN	(414) 782-7748
MARYLAND	(301) 796-9310		

## INTERNATIONAL SALES OFFICES

BELGIUM		ITALY, Milano	TEL: (02) 3390541
Bruxelles	TEL: (02) 771 99 93		FAX: (02) 3498000
	FAX: (02) 762-3716		TLX: 315286
	TLX: 61028		
CANADA, Ontario		JAPAN	
Kanata	TEL: (613) 592-0090	Tokyo	TEL: (03) 345-8241
Willowdale	TEL: (416) 224-5193		FAX: 3425196
	FAX: (416) 224-0056		TLX: J24064AMDTKQJ
FRANCE		Osaka	TEL: 06-243-3250
Paris	TEL: (01) 45 60 00 55		FAX: 06-243-3253
	FAX: (01) 46 86 21 85	KOREA, Seoul	TEL: 82-733-1021/7
	TLX: 202053F		FAX: 82-733-1028
GERMANY			TLX: K22652
Hannover area	TEL: (05143) 50 55	LATIN AMERICA	
	FAX: (05143) 55 53	Ft. Lauderdale	TEL: (305) 484-8600
	TLX: 925287		FAX: (305) 485-9736
München	TEL: (089) 41 14-0		TLX: 5109554261 AMDFTL
	FAX: (089) 406490	SWEDEN, Stockholm	TEL: (08) 733 03 50
	TLX: 523883		FAX: (08) 733 22 85
Stuttgart	TEL: (0711) 62 33 77		TLX: 11602
	FAX: (0711) 625187	UNITED KINGDOM	
	TLX: 721882	Manchester area	TEL: (0925) 828008
HONG KONG			FAX: (0925) 827693
Kowloon	TEL: 3-695377		TLX: 628524
	FAX: 1234276	London area	TEL: (04862) 22121
	TLX: 50426		FAX: (04862) 22179
			TLX: 859103

## NORTH AMERICAN REPRESENTATIVES

CALIFORNIA		NEW MEXICO	
I <sup>2</sup> INC	OEM (408) 988-3400	THORSON DESERT STATES	(505) 293-8555
	DIST (408) 496-6868	NEW YORK	
IDAHO		NYCOM, INC	(315) 437-8343
INTERMOUNTAIN TECH MKGT	(208) 888-6071	OHIO	
INDIANA		Dayton	
SAI MARKETING CORP	(317) 241-9276	DOLFUSS ROOT & CO	(513) 433-6776
IOWA		Strongsville	
LORENZ SALES	(319) 377-4666	DOLFUSS ROOT & CO	(216) 238-0300
MICHIGAN		PENNSYLVANIA	
SAI MARKETING CORP	(313) 750-1922	DOLFUSS ROOT & CO	(412) 221-4420
NEBRASKA		UTAH	
LORENZ SALES	(402) 475-4660	R <sup>2</sup> 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 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088, USA  
TEL: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450

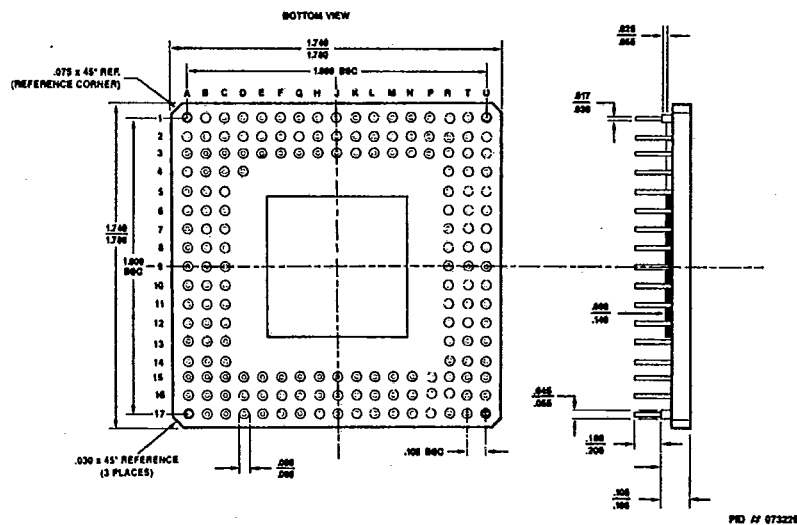
© 1987 Advanced Micro Devices, Inc.  
Printed in U.S.A. AIS-WCP-30M-2/87-0

## General Information

T. 90-20

## PACKAGE OUTLINES\*

### CGX169



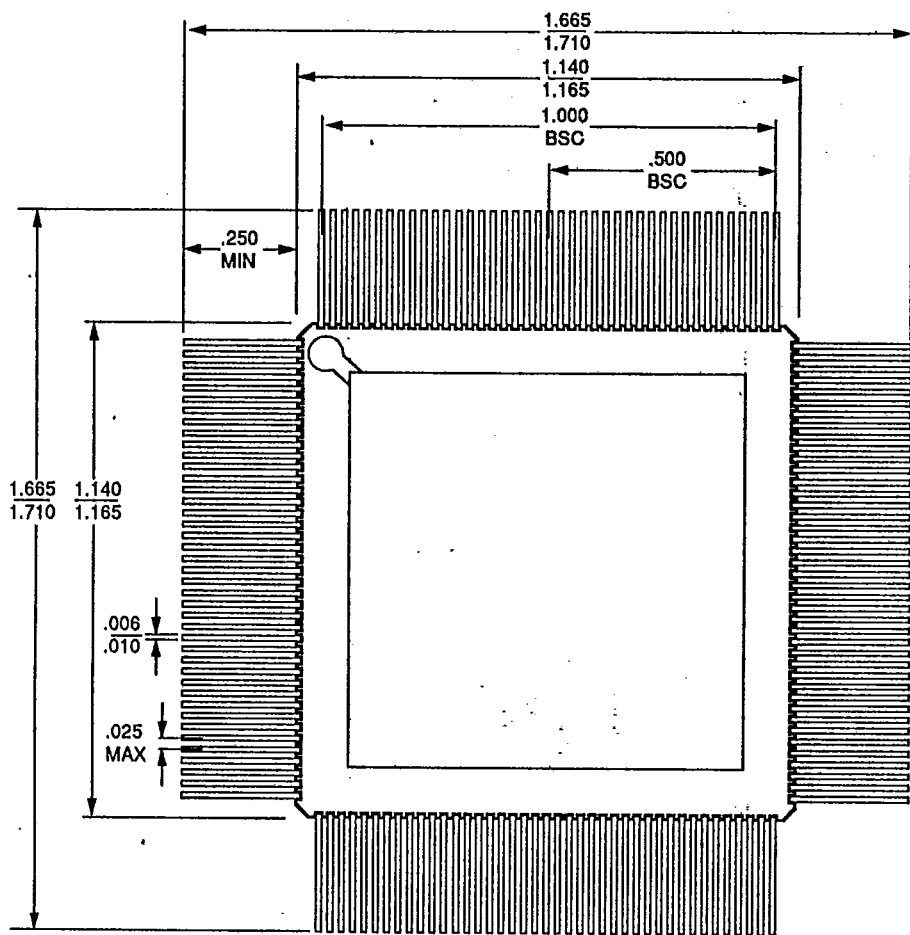
\*For reference only.

**\*For reference only. All dimensions are measured in inches. BSC is an ANSI standard for Basic Space Centering.**

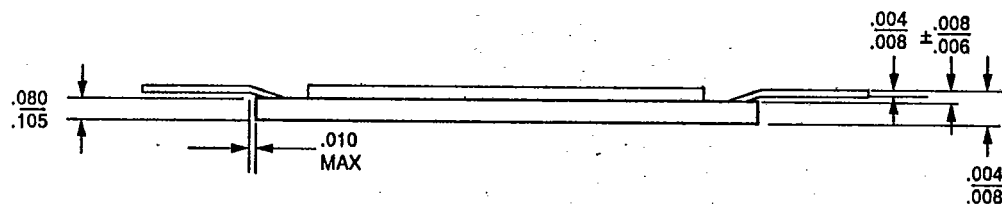
T-90-20

Package Outlines

CQ164



TOP VIEW



13092A