Preliminary

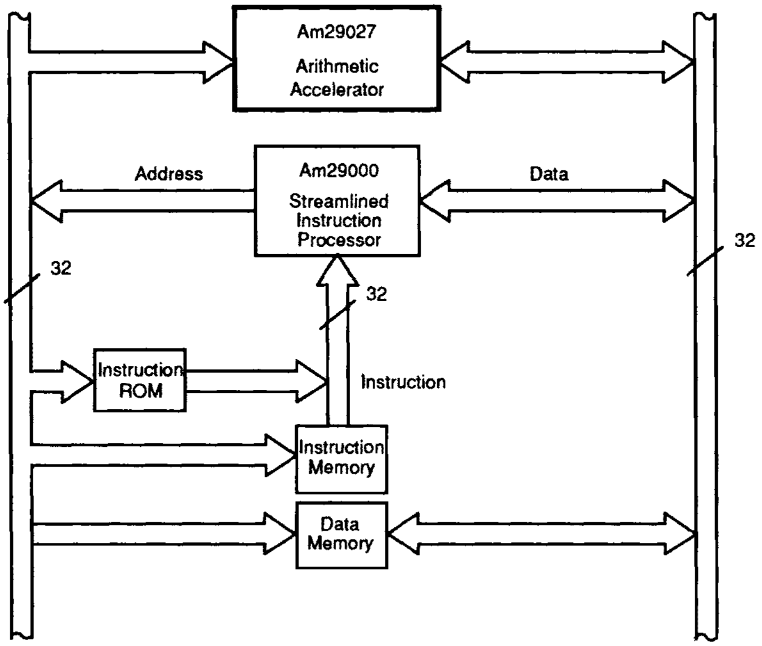
Am29027
Arithmetic Accelerator

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- **High-speed floating-point accelerator for the Am29000™ processor**
- **Comprehensive floating-point and integer instruction sets, including addition, subtraction, and multiplication**
- **Single-, double-, and mixed-precision operations**
- **Performs conversions between precisions and between data formats**
- **Complies with seven industry-standard floating-point formats:**
  - –IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE std 754-1985), single- and double-precision
  - –DEC™ F, DEC D, and DEC G Standards
  - –IBM® System/370 single- and double-precision
- **Exact IEEE compliance for denormalized numbers with no speed penalty**
- **Simple interface requires no glue logic between Am29000 and Am29027™**
- **Eight-deep register file for intermediate results and on-chip 64-bit data path facilitate compound operations, for example, Newton-Raphson division, sum-of-products, and transcendentals**
- **Supports pipelined or flow-through operation**
- **Full compiler and assembler support for IEEE format**
- **Fabricated with Advanced Micro Devices' 1.2-micron CMOS process**

SIMPLIFIED SYSTEM DIAGRAM



09114-001C

Publication # 09114 Rev. C Amendment /0
Issue Date: October 1989

TABLE OF CONTENTS

GENERAL DESCRIPTION

The Am29027 Arithmetic Accelerator is a high-performance computational unit intended for use with the Am29000 Streamlined Instruction Processor. When added to an Am29000-based system, the Am29027 improves 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 widely used floating-point formats—IEEE, DEC, and IBM—are supported. IEEE operations fully comply with the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE standard 754-1985), with direct implementation of special features such as gradual underflow and exception handling.

The Am29027 consists of a 64-bit ALU, a 64-bit data path, and a control unit. The ALU has three data input ports, and can perform operations requiring one, two, or three input operands. The data path comprises two 64-bit input operand registers, an 8-by-64-bit register file for storage of intermediate results, three operand selection multiplexers that provide for orthogonal selection of input operands, and an output multiplexer that allows access to the result data, the operation status, the flags, or the accelerator state. The control unit interprets transaction requests from the Am29000, and sequences the ALU and data path.

Operations can be performed in either of two modes: flow-through or pipeline. In flow-through mode, the ALU is completely combinatorial; this mode is best suited to scalar operations. Pipeline mode divides the ALU into two or three 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 CMOS technology, the Am29027 is housed in two packages: a 169-lead pin-grid-array (PGA) package, and a 164-lead ceramic-quad-flat-pack (CQFP) package for military applications.

Related AMD Products

Part No.	Description
Am29000	Streamlined Instruction Processor

29K™ Family Development Support Products

Contact your local AMD representative for information on the complete set of development support tools.

Software development products on several hosts:

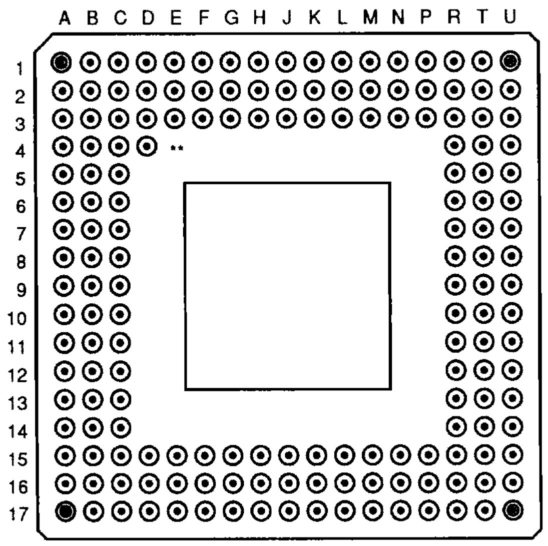
- Optimizing compilers for common high-level languages
- Assembler and utility packages
- Source- and assembly-level software debuggers
- Target-resident development monitors
- Simulators

Hardware Development:

- ADAPT29K™ Advanced Development and Prototyping Tool

CONNECTION DIAGRAMS

169-Lead PGA*
Bottom View



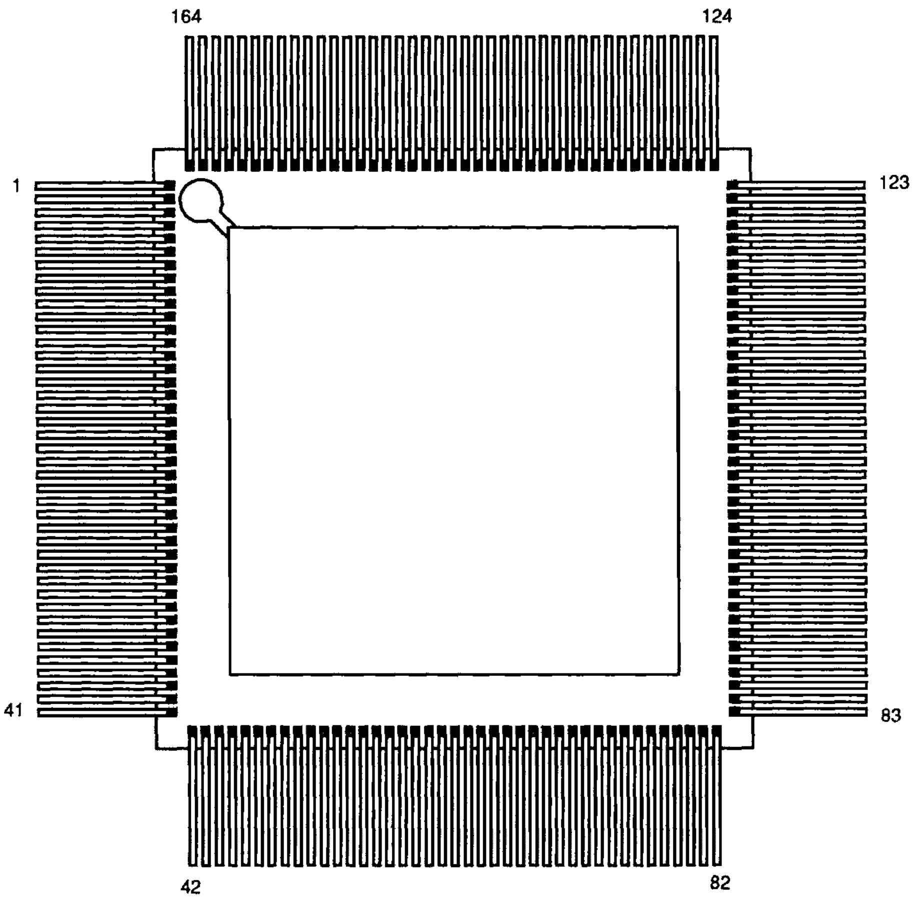
* Pinout observed from pin side of package.
**Alignment pin (not connected internally).

CD009761

CONNECTION DIAGRAMS (continued)

164-Lead CQFP*

Top View
(Lid Facing Viewer)



PGA PIN DESIGNATIONS (sorted by Pin No.)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
A-1	S_{31}	C-10	F_{20}	J-16	I_{16}	R-12	$DREQT_0$
A-2	F_4	C-11	V_{CCO}	J-17	I_{18}	R-13	$\overline{RESET}$
A-3	F_6	C-12	GNDO	K-1	S_9	R-14	$\overline{DREQ}$
A-4	F_8	C-13	F_{29}	K-2	S_{10}	R-15	I_{29}
A-5	F_{10}	C-14	GNDO	K-3	GND	R-16	I_{27}
A-6	F_{12}	C-15	V_{CCO}	K-15	I_{21}	R-17	I_{24}
A-7	F_{14}	C-16	I_2	K-16	I_{20}	T-1	R_{28}
A-8	F_{16}	C-17	I_6	K-17	I_{19}	T-2	R_{23}
A-9	F_{18}	D-1	S_{24}	L-1	S_8	T-3	R_{21}
A-10	F_{21}	D-2	S_{25}	L-2	S_7	T-4	R_{18}
A-11	F_{22}	D-3	S_{29}	L-3	S_6	T-5	R_{16}
A-12	F_{24}	D-4	(see note)	L-15	GNDO	T-6	R_{13}
A-13	F_{27}	D-15	I_0	L-16	I_{23}	T-7	R_{10}
A-14	F_{28}	D-16	I_3	L-17	I_{22}	T-8	R_7
A-15	F_{31}	D-17	I_8	M-1	S_5	T-9	R_5
A-16	$\overline{SLAVE}$	E-1	S_{21}	M-2	S_4	T-10	R_3
A-17	I_1	E-2	S_{23}	M-3	S_2	T-11	R_0
B-1	S_{30}	E-3	S_{26}	M-15	V_{CCO}	T-12	OPT_1
B-2	F_1	E-15	I_4	M-16	$\overline{DRDY}$	T-13	$DREQT_1$
B-3	F_3	E-16	I_7	M-17	$\overline{CDA}$	T-14	$\overline{BINV}$
B-4	F_5	E-17	I_9	N-1	S_3	T-15	I_{31}
B-5	F_7	F-1	S_{18}	N-2	S_1	T-16	I_{28}
B-6	F_9	F-2	S_{20}	N-3	R_{30}	T-17	I_{25}
B-7	F_{13}	F-3	S_{22}	N-15	NC	U-1	R_{25}
B-8	F_{15}	F-15	V_{CC}	N-16	$\overline{EXCP}$	U-2	R_{22}
B-9	F_{17}	F-16	I_{10}	N-17	$\overline{DERR}$	U-3	R_{19}
B-10	F_{19}	F-17	I_{12}	P-1	S_0	U-4	R_{17}
B-11	F_{23}	G-1	S_{15}	P-2	R_{29}	U-5	R_{15}
B-12	F_{25}	G-2	S_{17}	P-3	R_{26}	U-6	R_{14}
B-13	F_{26}	G-3	S_{19}	P-15	I_{26}	U-7	R_{11}
B-14	F_{30}	G-15	GND	P-16	NC	U-8	R_9
B-15	GND	G-16	I_{11}	P-17	NC	U-9	R_6
B-16	MSERR	G-17	I_{14}	R-1	R_{31}	U-10	R_4
B-17	I_5	H-1	S_{13}	R-2	R_{27}	U-11	R_2
C-1	S_{27}	H-2	S_{14}	R-3	R_{24}	U-12	R_1
C-2	S_{28}	H-3	S_{16}	R-4	R_{20}	U-13	OPT_0
C-3	F_0	H-15	GND	R-5	V_{CC}	U-14	OPT_2
C-4	F_2	H-16	I_{13}	R-6	GND	U-15	$R/\overline{W}$
C-5	V_{CCO}	H-17	I_{15}	R-7	R_{12}	U-16	$\overline{OE}$
C-6	GNDO	J-1	S_{11}	R-8	R_8	U-17	I_{30}
C-7	F_{11}	J-2	S_{12}	R-9	GND		
C-8	GNDO	J-3	V_{CC}	R-10	V_{CC}		
C-9	V_{CCO}	J-15	I_{17}	R-11	CLK		

Note: Pin Number D-4 = Alignment Pin.
V_{CCO} and GNDO are power and ground pins for the output buffers.
V_{CC} and GND are power and ground pins for the rest of the logic.

PGA PIN DESIGNATIONS (sorted by Pin Name)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
T-14	$\overline{BINV}$	G-15	GND	B-16	MSERR	P-1	S_0
M-17	$\overline{CDA}$	H-15	GND	N-15	NC	N-2	S_1
R-11	CLK	K-3	GND	P-16	NC	M-3	S_2
N-17	$\overline{DERR}$	R-6	GND	P-17	NC	N-1	S_3
M-16	$\overline{DRDY}$	R-9	GND	U-16	$\overline{OE}$	M-2	S_4
R-14	$\overline{DREQ}$	C-6	GNDO	U-13	OPT_0	M-1	S_5
R-12	$DREQT_0$	C-8	GNDO	T-12	OPT_1	L-3	S_6
T-13	$DREQT_1$	C-12	GNDO	U-14	OPT_2	L-2	S_7
N-16	$\overline{EXCP}$	C-14	GNDO	T-11	R_0	L-1	S_8
C-3	F_0	L-15	GNDO	U-12	R_1	K-1	S_9
B-2	F_1	D-15	I_0	U-11	R_2	K-2	S_{10}
C-4	F_2	A-17	I_1	T-10	R_3	J-1	S_{11}
B-3	F_3	C-16	I_2	U-10	R_4	J-2	S_{12}
A-2	F_4	D-16	I_3	T-9	R_5	H-1	S_{13}
B-4	F_5	E-15	I_4	U-9	R_6	H-2	S_{14}
A-3	F_6	B-17	I_5	T-8	R_7	G-1	S_{15}
B-5	F_7	C-17	I_6	R-8	R_8	H-3	S_{16}
A-4	F_8	E-16	I_7	U-8	R_9	G-2	S_{17}
B-6	F_9	D-17	I_8	T-7	R_{10}	F-1	S_{18}
A-5	F_{10}	E-17	I_9	U-7	R_{11}	G-3	S_{19}
C-7	F_{11}	F-16	I_{10}	R-7	R_{12}	F-2	S_{20}
A-6	F_{12}	G-16	I_{11}	T-6	R_{13}	E-1	S_{21}
B-7	F_{13}	F-17	I_{12}	U-6	R_{14}	F-3	S_{22}
A-7	F_{14}	H-16	I_{13}	U-5	R_{15}	E-2	S_{23}
B-8	F_{15}	G-17	I_{14}	T-5	R_{16}	D-1	S_{24}
A-8	F_{16}	H-17	I_{15}	U-4	R_{17}	D-2	S_{25}
B-9	F_{17}	J-16	I_{16}	T-4	R_{18}	E-3	S_{26}
A-9	F_{18}	J-15	I_{17}	U-3	R_{19}	C-1	S_{27}
B-10	F_{19}	J-17	I_{18}	R-4	R_{20}	C-2	S_{28}
C-10	F_{20}	K-17	I_{19}	T-3	R_{21}	D-3	S_{29}
A-10	F_{21}	K-16	I_{20}	U-2	R_{22}	B-1	S_{30}
A-11	F_{22}	K-15	I_{21}	T-2	R_{23}	A-1	S_{31}
B-11	F_{23}	L-17	I_{22}	R-3	R_{24}	A-16	$\overline{SLAVE}$
A-12	F_{24}	L-16	I_{23}	U-1	R_{25}	F-15	V_{CC}
B-12	F_{25}	R-17	I_{24}	P-3	R_{26}	J-3	V_{CC}
B-13	F_{26}	T-17	I_{25}	R-2	R_{27}	R-5	V_{CC}
A-13	F_{27}	P-15	I_{26}	T-1	R_{28}	R-10	V_{CC}
A-14	F_{28}	R-16	I_{27}	P-2	R_{29}	C-5	V_{CCO}
C-13	F_{29}	T-16	I_{28}	N-3	R_{30}	C-9	V_{CCO}
B-14	F_{30}	R-15	I_{29}	R-1	R_{31}	C-11	V_{CCO}
A-15	F_{31}	U-17	I_{30}	R-13	$\overline{RESET}$	C-15	V_{CCO}
B-15	GND	T-15	I_{31}	U-15	$R/\overline{W}$	M-15	V_{CCO}

Note: Pin Number D-4 = Alignment Pin.
V_{CCO} and GNDO are power and ground pins for the output buffers.
V_{CC} and GND are power and ground pins for the rest of the logic.

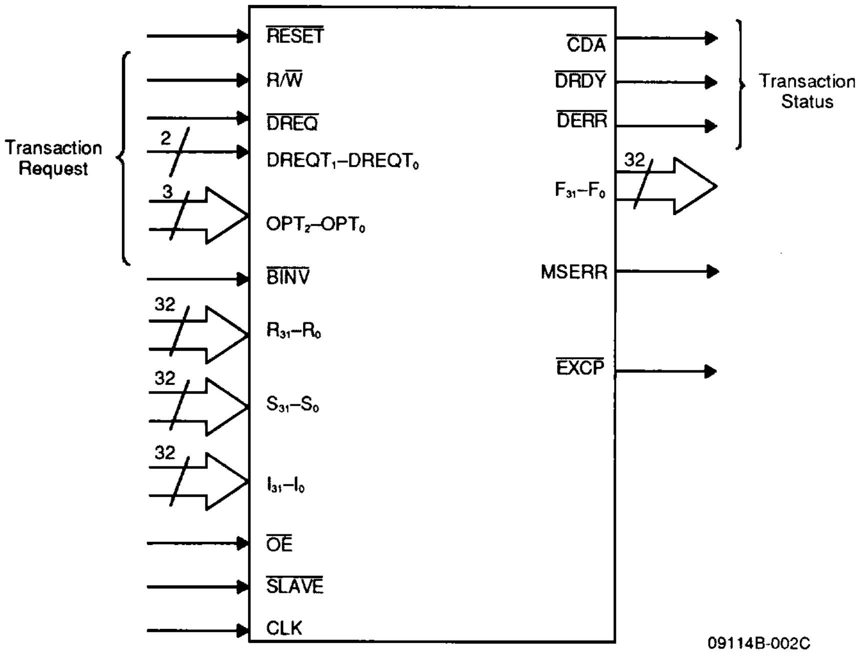
CQFP PIN DESIGNATIONS (sorted by Pin No.)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	F_0	42	V_{CC}	83	I_{29}	124	R_{25}
2	F_1	43	GND	84	I_{28}	125	R_{26}
3	F_2	44	I_0	85	I_{31}	126	R_{27}
4	F_3	45	I_1	86	$\overline{DREQ}$	127	R_{28}
5	F_4	46	I_2	87	$\overline{OE}$	128	R_{29}
6	V_{CCO}	47	I_3	88	$\overline{BINV}$	129	R_{30}
7	GNDO	48	I_4	89	$\overline{RESET}$	130	R_{31}
8	F_5	49	I_5	90	$R/\overline{W}$	131	S_0
9	F_6	50	I_6	91	$DREQT_1$	132	S_1
10	F_7	51	I_7	92	$DREQT_0$	133	S_2
11	F_8	52	I_8	93	OPT_2	134	S_3
12	F_9	53	I_9	94	OPT_1	135	S_4
13	F_{10}	54	I_{10}	95	OPT_0	136	S_5
14	F_{11}	55	I_{11}	96	CLK	137	S_6
15	F_{12}	56	I_{12}	97	R_0	138	S_7
16	F_{13}	57	I_{13}	98	R_1	139	S_8
17	F_{14}	58	GND	99	R_2	140	S_9
18	F_{15}	59	I_{14}	100	R_3	141	S_{10}
19	GNDO	60	I_{15}	101	R_4	142	S_{11}
20	V_{CCO}	61	I_{16}	102	V_{CC}	143	GND
21	F_{16}	62	I_{17}	103	GND	144	V_{CC}
22	F_{17}	63	I_{18}	104	R_5	145	S_{12}
23	F_{18}	64	I_{19}	105	R_6	146	S_{13}
24	F_{19}	65	I_{20}	106	R_7	147	S_{14}
25	F_{20}	66	I_{21}	107	R_8	148	S_{15}
26	F_{21}	67	I_{22}	108	R_9	149	S_{16}
27	F_{22}	68	I_{23}	109	R_{10}	150	S_{17}
28	F_{23}	69	$\overline{CDA}$	110	R_{11}	151	S_{18}
29	F_{24}	70	$\overline{DRDY}$	111	R_{12}	152	S_{19}
30	F_{25}	71	$\overline{DERR}$	112	R_{13}	153	S_{20}
31	F_{26}	72	GNDO	113	R_{14}	154	S_{21}
32	V_{CCO}	73	V_{CCO}	114	R_{15}	155	S_{22}
33	GNDO	74	$\overline{EXCP}$	115	R_{16}	156	S_{23}
34	F_{27}	75	NC	116	R_{17}	157	S_{24}
35	F_{28}	76	NC	117	R_{18}	158	S_{25}
36	F_{29}	77	NC	118	R_{19}	159	S_{26}
37	F_{30}	78	I_{24}	119	R_{20}	160	S_{27}
38	F_{31}	79	I_{25}	120	R_{21}	161	S_{28}
39	GND	80	I_{26}	121	R_{22}	162	S_{29}
40	$\overline{SLAVE}$	81	I_{27}	122	R_{23}	163	S_{30}
41	MSERR	82	I_{30}	123	R_{24}	164	S_{31}

CQFP PIN DESIGNATIONS (sorted by Pin Name)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
88	$\overline{BINV}$	39	GND	41	MSERR	130	R_{31}
69	$\overline{CDA}$	43	GND	75	NC	40	$\overline{SLAVE}$
96	CLK	58	GND	76	NC	131	S_0
71	$\overline{DERR}$	103	GND	77	NC	132	S_1
86	$\overline{DREQ}$	143	GND	87	$\overline{OE}$	133	S_2
92	$\overline{DREQT_0}$	7	GNDO	95	OPT_0	134	S_3
91	$\overline{DREQT_1}$	19	GNDO	94	OPT_1	135	S_4
70	$\overline{DRDY}$	33	GNDO	93	OPT_2	136	S_5
74	$\overline{EXCP}$	72	GNDO	89	$\overline{RESET}$	137	S_6
1	F_0	44	I_0	90	$R/\overline{W}$	138	S_7
2	F_1	45	I_1	97	R_0	139	S_8
3	F_2	46	I_2	98	R_1	140	S_9
4	F_3	47	I_3	99	R_2	141	S_{10}
5	F_4	48	I_4	100	R_3	142	S_{11}
8	F_5	49	I_5	101	R_4	145	S_{12}
9	F_6	50	I_6	104	R_5	146	S_{13}
10	F_7	51	I_7	105	R_6	147	S_{14}
11	F_8	52	I_8	106	R_7	148	S_{15}
12	F_9	53	I_9	107	R_8	149	S_{16}
13	F_{10}	54	I_{10}	108	R_9	150	S_{17}
14	F_{11}	55	I_{11}	109	R_{10}	151	S_{18}
15	F_{12}	56	I_{12}	110	R_{11}	152	S_{19}
16	F_{13}	57	I_{13}	111	R_{12}	153	S_{20}
17	F_{14}	59	I_{14}	112	R_{13}	154	S_{21}
18	F_{15}	60	I_{15}	113	R_{14}	155	S_{22}
21	F_{16}	61	I_{16}	114	R_{15}	156	S_{23}
22	F_{17}	62	I_{17}	115	R_{16}	157	S_{24}
23	F_{18}	63	I_{18}	116	R_{17}	158	S_{25}
24	F_{19}	64	I_{19}	117	R_{18}	159	S_{26}
25	F_{20}	65	I_{20}	118	R_{19}	160	S_{27}
26	F_{21}	66	I_{21}	119	R_{20}	161	S_{28}
27	F_{22}	67	I_{22}	120	R_{21}	162	S_{29}
28	F_{23}	68	I_{23}	121	R_{22}	163	S_{30}
29	F_{24}	78	I_{24}	122	R_{23}	164	S_{31}
30	F_{25}	79	I_{25}	123	R_{24}	42	V_{CC}
31	F_{26}	80	I_{26}	124	R_{25}	102	V_{CC}
34	F_{27}	81	I_{27}	125	R_{26}	144	V_{CC}
35	F_{28}	84	I_{28}	126	R_{27}	6	V_{CCO}
36	F_{29}	83	I_{29}	127	R_{28}	20	V_{CCO}
37	F_{30}	82	I_{30}	128	R_{29}	32	V_{CCO}
38	F_{31}	85	I_{31}	129	R_{30}	73	V_{CCO}

LOGIC SYMBOL



ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The ordering 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

AM29027 -25 G C B

e. OPTIONAL PROCESSING
Blank = Standard Processing
B = Burn-in

d. TEMPERATURE RANGE
C = Commercial (0 to +85°C)

c. PACKAGE TYPE
G = 169-Lead Pin Grid Array without Heatsink (CGX169)

b. SPEED OPTION
–25 = 25 MHz
–20 = 20 MHz
–16 = 16 MHz

a. DEVICE NUMBER/DESCRIPTION
Am29027
Arithmetic Accelerator

Valid Combinations	
AM29027-25	GC, GCB
AM29027-20	
AM29027-16	

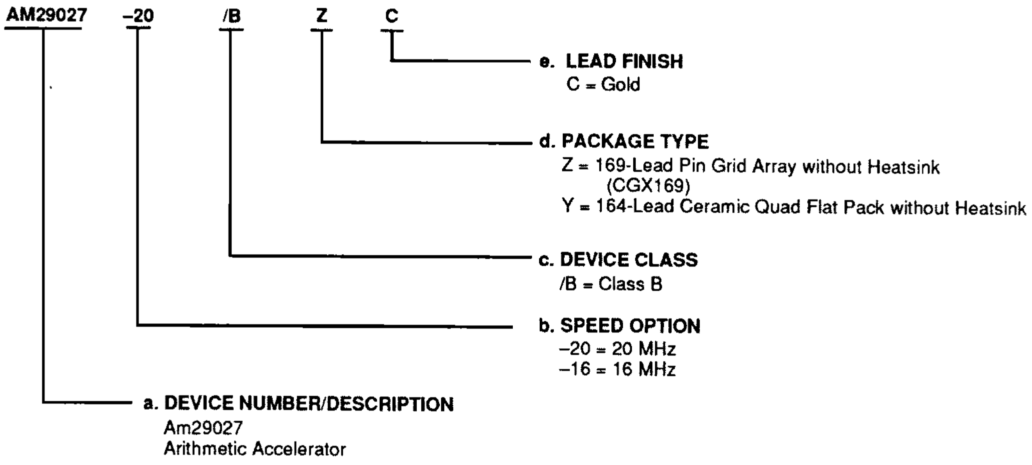
Valid Combinations

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

MILITARY ORDERING INFORMATION
APL Products

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

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



e. LEAD FINISH
C = Gold

d. PACKAGE TYPE
Z = 169-Lead Pin Grid Array without Heatsink (CGX169)
Y = 164-Lead Ceramic Quad Flat Pack without Heatsink

c. DEVICE CLASS
/B = Class B

b. SPEED OPTION
–20 = 20 MHz
–16 = 16 MHz

a. DEVICE NUMBER/DESCRIPTION
Am29027
Arithmetic Accelerator

Valid Combinations	
AM29027-20	/BZC, /BYC
AM29027-16	

Valid Combinations

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

Group A Tests

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

PIN DESCRIPTION

$\overline{\text{BINV}}$

Bus Invalid (Synchronous Input)

A logic Low indicates that the Am29000 address bus and related control signals are invalid. The Am29027 will ignore signal DREQT$_1$ when $\overline{\text{BINV}}$ is Low.

$\overline{\text{CDA}}$

Coprocessor Data Accept (Three-State Output)

A logic Low indicates that the Am29027 is ready to accept data from the Am29000. This signal is normally driven by the Am29027, and assumes a high-impedance state only if input signal $\overline{\text{OE}}$ is High or input signal $\overline{\text{SLAVE}}$ is Low.

CLK

Clock (Input)

$\overline{\text{DERR}}$

Data Error (Three-State Output)

A logic Low indicates that an unmasked exception occurred during or preceding the current transaction request. This signal is normally driven by the Am29027, and assumes a high-impedance state only if input signal $\overline{\text{OE}}$ is High or input signal $\overline{\text{SLAVE}}$ is Low.

$\overline{\text{DRDY}}$

Data Ready (Three-State Output)

A logic Low indicates that data is available on Port F. This signal is normally driven by the Am29027, and assumes a high-impedance state only if input signal $\overline{\text{OE}}$ is High or input signal $\overline{\text{SLAVE}}$ is Low.

$\overline{\text{DREQ}}$

Data Request (Synchronous Input)

A logic Low indicates that the Am29000 is making a data access. The Am29027 will ignore signal DREQT$_1$ when $\overline{\text{DREQ}}$ is High.

DREQT$_0$

Start Instruction/Suppress Errors (Synchronous Input)

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$_0$ suppresses the reporting of operation errors. DREQT$_0$ also modifies the action of the write status transaction request to retime an operation in flow-through mode, or to invalidate the ALU pipeline in pipeline mode.

DREQT$_1$

Accelerator Transaction Request (Synchronous Input)

A logic High indicates that the Am29000 is making an accelerator transaction request. This signal is considered valid only when signal $\overline{\text{BINV}}$ is High and signal $\overline{\text{DREQ}}$ is Low.

$\overline{\text{EXCP}}$

Exception (Three-State Output)

Indicates that the status register contains one or more unmasked exception bits. This signal can be used as an interrupt or trap signal by the Am29000. $\overline{\text{EXCP}}$ is normally driven by the Am29027, and assumes a high-impedance state only if input signal $\overline{\text{OE}}$ is High or input signal $\overline{\text{SLAVE}}$ is Low.

F$_{31}$–F$_0$

F Output Bus (Three-State Output)

I$_{31}$–I$_0$

Instruction Bus (Synchronous Input)

Used to specify the operation to be performed by the accelerator.

MSERR

Master/Slave Error (Output)

Reports the result of the comparison of processor outputs with the signals provided internally to the off-chip drivers. If there is a difference for any enabled driver, MSERR assumes the logic High state.

$\overline{\text{OE}}$

Output Enable (Asynchronous Input)

A logic High forces all accelerator outputs except MSERR to assume a high-impedance state unconditionally; master/slave comparison circuitry is also disabled. This signal is provided for test purposes.

OPT$_2$–OPT$_0$

Transaction Type (Synchronous Input)

These signals, in conjunction with R/$\overline{\text{W}}$, specify the type of accelerator transaction, if any, currently being requested by the Am29000.

R$_{31}$–R$_0$

R Data Bus (Synchronous Input)

$\overline{\text{RESET}}$

Reset (Asynchronous Input)

Resets the Am29027. When $\overline{\text{RESET}}$ is a logic Low, the state of internal sequencing circuitry is initialized, and the status register is cleared. $\overline{\text{RESET}}$ must be connected to the signal line used to reset the Am29000.

R/$\overline{\text{W}}$

Read/Write (Synchronous Input)

Determines the direction of a transaction. When R/$\overline{\text{W}}$ is High, data is transferred from the Am29027 to the Am29000; when R/$\overline{\text{W}}$ is Low, data is transferred from the Am29000 to the Am29027.

S_{31}–S_0
S Data Bus (Synchronous Input)

$\overline{\text{SLAVE}}$
Master/Slave Mode Select (Synchronous Input)

A logic Low selects Slave mode; in this mode all outputs except MSERR assume a high-impedance state. 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 data path, and control circuitry.

The core of the Am29027 is a 64-bit floating-point/integer ALU. The 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 minimum latency; optional pipelining is available to boost throughput for vector operations.

The data path consists of two 32-bit input buses, R and S; two 64-bit input registers; two 64-bit temporary input registers; a 64-bit result register; an 8-word-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 selects data, operation flags, operation status, or other accelerator state; and a 32-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 the input registers. The operand selection multiplexers route the operands to the ALU; operation results and status leave the device on Output Bus F. Operation results also can be stored in the register file for use in subsequent operations.

On-board control circuitry sequences the ALU and data path during operations, and manages the transfer of data between the accelerator and the Am29000. A 32-bit instruction register and a 32-bit temporary instruction register hold the instruction words for current and pending operations.

Instruction Set

The Am29027 implements 57 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 maximum or minimum of two numbers
- rounding to integral value
- absolute value, negation, pass
- reciprocal seed generation
- conversion between any of the supported floating-point formats, including conversions between precisions
- conversion of a floating-point number to an integer format, with an optional scale factor

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 maximum or minimum of two numbers
- absolute value, negation, pass
- 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 an optional scale factor
- pass operand

One special instruction is provided to move data.

Performance

The Am29027 provides operation speeds several times greater than conventional floating-point processors 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 that can be used to hold intermediate results, thus reducing the amount of time needed to transfer operands between the Am29027 and the Am29000. The input operand registers and the instruction register are double-buffered, so that a new operation can be specified while the current operation is being completed.

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. Eight input signals specify the transaction to be performed; 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 accelerator (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 IEEE format is fully supported by those hardware and software tools available for the Am29000, including:

- HighC29K Cross-Development Toolkit
- ASM29K Cross-Development Toolkit
- ADAPT29K, a general-purpose hardware development system. The ADAPT29K permits single-step operation, break-point insertion, and other standard debugging techniques.

Block Diagram Description

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

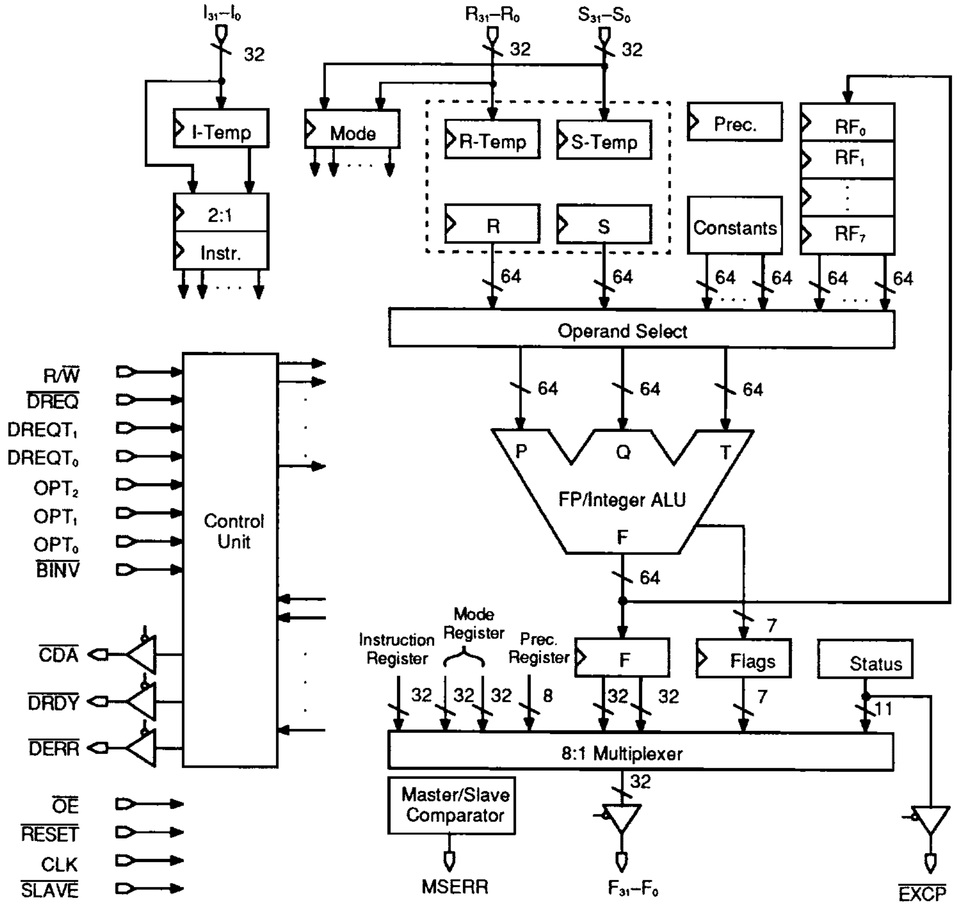


Figure 1. Am29027 Block Diagram

09114-003C

Input Registers

Operands are loaded into the accelerator via the 32-bit 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 or one double-precision operand may be written to the input registers in a single system clock cycle. Accompanying the input registers are two 64-bit temporary registers, R-Temp and S-Temp, that permit 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 RF_7–RF_0, also have access to a set of floating-point and integer constants. These constants are double-precision preprogrammed numbers for use in ALU operations, and are automatically provided in the appropriate 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, operand format (integer or floating-point), sign-change selects for ALU input and result operands, operand precisions, operand sources, and register file controls. The instruction register is preceded by the 32-bit temporary register, I-Temp, permitting the overlapping of instruction transfers and ALU operations. Instructions enter the accelerator via 32-bit Instruction Bus I.

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. Some operations require a single input operand, for example, conversion operations. Others, such as addition or multiplication, require two input operands. The multiplication-accumulation and funnel shift operations require three input operands. 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.

The ALU can be configured in either flow-through mode, for which the ALU is completely combinatorial, or pipeline mode, for which ALU operations are divided into one or two pipeline stages.

Output Register/Register File

Operation results are stored in 64-bit output register F; results can also be stored in the 8-by-64-bit register file for use in subsequent operations. A precision register, part of the register file, contains bits indicating the precisions of the operands stored in each register file location, thus permitting the ALU to correctly process these operands in later operations.

Flag Register

The 32-bit flag register stores flags pertaining to the most recently performed operation. The flags indicate error conditions, such as underflow or overflow, and also report results for operations that produce result flags, such as comparisons.

Status Register

The 32-bit status register contains information regarding the status of past, current, and pending operations.

Six exception bits report operation error conditions. These exception bits are individually latched; once a given bit is set, it remains set until reset by the Am29000 or by system reset. The exception bits indicate error conditions of overflow, underflow, zero result, reserved operand, invalid operation, and inexact result. At the user's option, the presence of an exception can be used to report a data error to the Am29000, or to halt Am29027 operation; exception bits can be individually enabled or disabled by programming the corresponding mask bit in the mode register.

Exception bit activity is summarized by a seventh bit, Exception Status, which indicates that one or more unmasked status bits are set. If desired, the state of this bit can be placed on signal $\overline{\text{EXCP}}$, which can be used to interrupt the Am29000.

The status register contains four additional bits—R-Temp Valid, S-Temp Valid, I-Temp Valid, and Operation Pending—that pertain to the state of pending operands and operations.

Output Multiplexer

The output multiplexer routes operation results and accelerator's internal state to the Am29000 through the 32-bit F bus. This multiplexer can select Register F, the flag register, status register, instruction register, mode register, or precision register.

Mode Register

The 64-bit mode register contains accelerator control parameters that change infrequently or not at all, such as floating-point format, round mode, and operation timing information. These parameters are initialized by the Am29000 during system start-up, and are modified as required during operation.

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 oversees operation of the Am29027 by issuing one of thirteen commands, or transaction requests, to the control unit via eight signal lines. Each transaction request specifies an action on the part of the Am29027, such as writing an operand to an input register or returning a result to the Am29000. The control unit interprets the transaction request and sequences the Am29027 to produce the desired action. Three transaction status lines are generated by the control unit to indicate transaction completion, or to indicate the existence of an accelerator error condition.

Master/Slave Comparator

Each Am29027 output signal has associated logic that compares that signal with the signal that the accelerator provides 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 $\overline{\text{SLAVE}}$. The second accelerator detects open-circuit signals, and provides a check of the outputs of the first accelerator.

System Interface

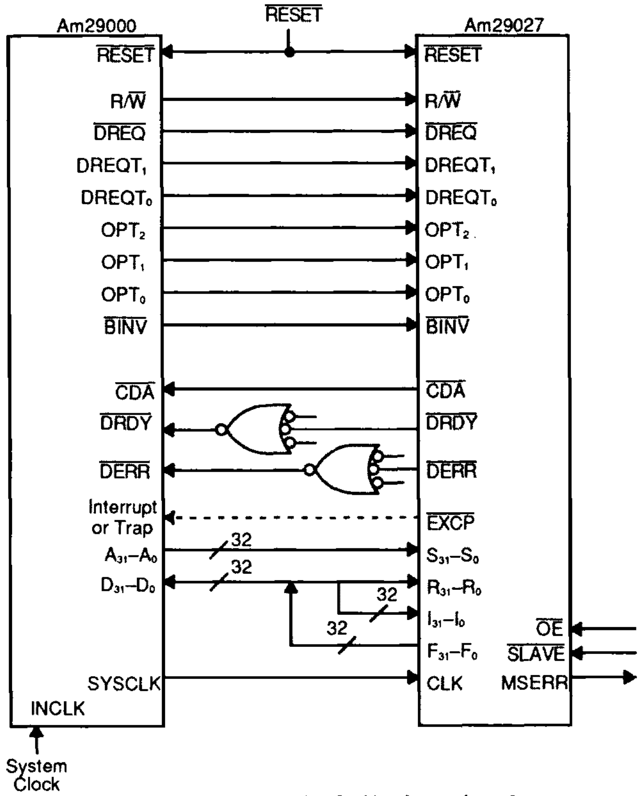
Am29000/Am29027 signal interconnects are depicted in Figure 2.

Three Am29027 buses—R_{31}–R_0, I_{31}–I_0, and F_{31}–F_0—are connected to Am29000 Data Bus D_{31}–D_0; the remaining Am29027 bus, S_{31}–S_0, is connected to Am29000 Address Bus A_{31}–A_0. Through these connections, the Am29000 can transfer to the Am29027 a 32-bit instruction, two 32-bit operands, or a 64-bit operand in a single cycle, or can receive a 32-bit result from the Am29027 in a single cycle.

Twelve additional signals govern communication between the Am29000 and Am29027. Eight Am29000 output signals—R/$\overline{\text{W}}$, $\overline{\text{DREQ}}$, DREQT_1, DREQT_0, OPT_2–OPT_0, and $\overline{\text{BINV}}$—are connected to the corresponding Am29027 signals and are used to issue transaction requests to the Am29027. Three Am29027 signals—$\overline{\text{CDA}}$, $\overline{\text{DRDY}}$, and $\overline{\text{DERR}}$—report transaction status. $\overline{\text{CDA}}$ is directly connected to the corresponding input of the Am29000, while $\overline{\text{DRDY}}$ and $\overline{\text{DERR}}$ must be ORed with like signals from other resources. A fourth Am29027 signal, $\overline{\text{EXCP}}$, may be connected to an Am29000 trap or interrupt input to signal the presence of Am29027 operation exceptions at the user's option.

The Am29027 takes its clock input from the Am29000 SYSCLK system clock output.

The signal used to reset the Am29000 must also be connected to the Am29027 $\overline{\text{RESET}}$ input.



09114-004C

Figure 2. Am29000/Am29027 Hardware Interface

Special-Purpose Registers

The Am29027 contains six special-purpose registers: the mode register, status register, flag register, precision register, instruction register, and I-Temp register.

Mode Register

The 64-bit mode register stores 24 infrequently changed parameters pertaining to accelerator operation; its format is shown in Figure 3. The Am29000 modifies the accelerator parameter set by issuing a write mode register transaction request.

The mode register should be initialized after hardware reset, and may be written with new parameters when a new mode of accelerator operation is required; mode changes take effect immediately. The Am29027 does not alter the contents of the mode register in the course of operation.

Bits 63–47—Reserved for future use. This field must be set to 0 to assure future compatibility.

Bit 46—$\overline{\text{EXCP}}$ Enable (EX): When EX is High, reporting of unmasked exceptions via signal $\overline{\text{EXCP}}$ is enabled. When EX is Low, signal $\overline{\text{EXCP}}$ is forced inactive (logic High).

Bit 45—Halt On Error Enable (HE): When HE is High, the Am29027 will halt operation in the presence of an unmasked exception.

Bit 44—Advance $\overline{\text{DRDY}}$ (AD): When AD is High, signal $\overline{\text{DRDY}}$ is advanced one cycle in flow-through mode. This bit has no effect in pipeline mode.

Bits 43–40—Timer Count for the MOVE P Operation (MVTC): In flow-through mode, MVTC specifies the number of clock cycles needed for data to traverse the ALU for base operation code MOVE P; in pipeline mode, it has no effect. This field can assume values between 3 and 15, inclusive.

Bits 39–36—Timer Count for the Multiply-Accumulate Operation (MATC): In flow-through mode, MATC specifies the number of clock cycles needed for data to traverse the ALU for base operation code $F' = (P' \times Q') + T'$; in pipeline mode, it has no effect. This field can assume values between 3 and 15, inclusive.

Bits 35–32—Pipeline Timer Count (PLTC): In flow-through mode, PLTC specifies the number of clock cycles needed for data to traverse the ALU for any base operation code except $F' = (P' \times Q') + T'$ or MOVE P; in pipeline mode, it specifies the number of cycles needed for data to traverse a single pipeline stage for any base operation code. This field can assume values between 3 and 15, inclusive, in flow-through mode, and between 2 and 15, inclusive, in pipeline mode.

Bits 31–28—Reserved for future use. This field must be set to 0 to assure future compatibility.

Bit 27—Zero Result Exception Mask (ZMSK): When ZMSK is High, the status register zero result exception bit is masked and will not contribute to the detection of an error condition.

Bit 26—Inexact Result Exception Mask (XMSK): When XMSK is High, the status register inexact result exception bit is masked and will not contribute to the detection of an error condition.

Bit 25—Underflow Exception Mask (UMSK): When UMSK is High, the status register underflow exception bit is masked and will not contribute to the detection of an error condition.

Bit 24—Overflow Exception Mask (VMSK): When VMSK is High, the status register overflow exception bit is masked and will not contribute to the detection of an error condition.

Bit 23—Reserved Operand Exception Mask (RMSK): When RMSK is High, the status register reserved operand exception bit is masked and will not contribute to the detection of an error condition.

Bit 22—Invalid Operation Exception Mask (IMSK): When IMSK is High, the status register invalid operation exception bit is masked and will not contribute to the detection of an error condition.

Bit 21—Reserved for future use. This bit must be set to 0 to assure future compatibility.

Bit 20—Pipeline Mode Select (PL): When PL is High, pipeline mode is selected; when PL is Low, flow-through (unpipelined) mode is selected.

Bits 19–17—Reserved for future use. This field must be set to 0 to assure future compatibility.

Bits 16–14—Round Mode Select (RMS): Selects one of six rounding modes as follows:

RMS	Round Mode
0 0 0	Round to nearest (IEEE)
0 0 1	Round to minus infinity
0 1 0	Round to plus infinity
0 1 1	Round to zero
1 0 0	Round to nearest (DEC)
1 0 1	Round away from zero
1 1 X	Illegal value

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

Bits 13–12—Integer Multiplication Format Adjust (MF): Selects the output format for integer multiplication. The user may select either the MSBs or the LSBs of an integer multiplication result, with optional format adjust. MF is encoded as follows:

MF	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.

Bit 11—Integer Multiplication Signed/Unsigned Select (MS): If MS is High, input operands for integer multiplication operations are treated as two's complement numbers. If MS is Low, the input operands are treated as unsigned numbers.

Bit 10—Reserved for future use. This bit must be set to 0 to assure future compatibility.

Bit 9—IBM Underflow Mask Enable (BU): If BU is High, certain underflowed IBM operations will produce a normalized result with a biased exponent increased by 128. If BU is Low, these operations will produce a final result of true zero. BU affects only those operations that produce a result in IBM format and that use the following base operation codes:

F′ = P′ + T′	Convert T to Alternate F.P. Format
F′ = P′ × Q′	Convert T from Alternate F.P.
Compare P, T	Format
F′ = (P′ × Q′) + T′	Scale T to Floating-point by Q

Bit 8—IBM Significance Mask Enable (BS): If BS is High, certain IBM operations having intermediate results of 0 will produce a final result of 0 with the biased exponent unchanged. If BS is Low, these operations will produce a final result of true zero. BS affects only those operations that produce a result in IBM format and that use the F′ = P′ + Q′ and COMPARE P, T base operation codes.

Bit 7—IEEE Sudden Underflow Enable (SU): If SU is High, all IEEE denormalized results are replaced by a 0 of the same sign; if SU is Low, the appropriate denormalized number will be produced. If IEEE traps are enabled (mode register bit TRP High), sudden underflow is disabled.

Bit 6—IEEE Trap Enable (TRP): If TRP is High, IEEE trapped operation is enabled; the Saturate Enable (SAT) and Sudden Underflow (SU) bits are ignored. For an underflowed result, the biased exponent is increased by 192 (single precision) or 1536 (double precision), with the significand unchanged. For an overflowed result, the biased exponent is decreased by a like amount with the significand unchanged. If TRP is Low, IEEE trapped operation is disabled. This bit affects only those operations that produce a result in IEEE floating-point format.

Bit 5—IEEE Affine/Projective Select (AP): If AP is High, IEEE addition or subtraction operations having infinite input operands are performed in affine mode; if AP is Low, these operations are performed in projective mode. In affine mode, it is permissible to add infinities of like sign or subtract infinities of opposite sign, producing an infinite result with the appropriate sign. In projective mode these operations will produce an invalid operation exception. This bit affects only those operations that produce a result in IEEE floating-point format.

Bit 4—Saturate Enable (SAT): If SAT is High, overflowed results are replaced by the largest representable value in the selected format of the same sign as the overflowed result; if SAT is Low, the result produced depends on the overflow conventions for the selected floating-point format. If IEEE traps are enabled (mode register bit TR High), saturation is disabled for any operation that produces a result in IEEE floating-point format.

Bits 1–0 Primary Floating-Point Format (PFF), Bits 3–2 Alternate Floating-Point Format (AFF): The primary format is used as the source and destination format for all floating-point operations except conversions, and as the source or destination format for operations that convert between floating-point and integer formats. The alternate format is used as a source or destination format in operations that convert one floating-point format to another. Both the PFF and AFF fields are encoded as follows:

High Bit	Low Bit	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.

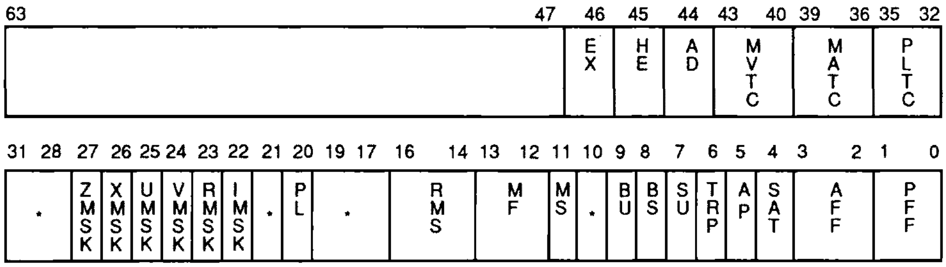


Figure 3. Mode Register

09114-005C

Status Register

The status register contains operation exception status, as well as the status of pending operands and operations; its format is shown in Figure 4. The Am29000 can initialize or modify the contents of the status register by issuing a write status transaction request, and can read current status register contents by issuing a read status transaction request or as part of a save state sequence.

All status register bits are initialized to a logic Low after hardware reset.

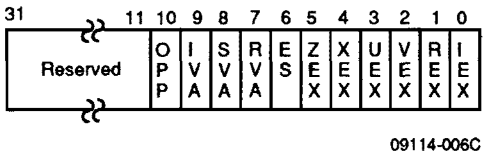


Figure 4. Status Register

Bits 31–11—Reserved for future use. This field must be set to 0 when written to assure future compatibility.

Bit 10—Operation Pending (OPP): A logic High indicates that an operation awaits execution.

Bit 9—I-Temp Valid (IVA): A logic High indicates that register I-Temp contains an instruction for a pending operation.

Bit 8—S-Temp Valid (SVA): A logic High indicates that register S-Temp contains an operand for a pending operation.

Bit 7—R-Temp Valid (RVA): A logic High indicates that register R-Temp contains an operand for a pending operation.

Bit 6—Exception Status (ES): A logic High indicates that status register bits 0–5 contain an unmasked exception.

Bit 5—Zero Result Flag (ZEX): A logic High indicates that an operation produced a zero result. Latches until cleared.

Bit 4—Inexact Result Bit (XEX): A logic High indicates that an operation result had to be rounded to fit the destination format. Latches until cleared.

Bit 3—Underflow Exception Bit (UEX): A logic High indicates that an operation result has underflowed the destination format. Latches until cleared.

Bit 2—Overflow Exception Bit (VEX): A logic High indicates that an operation result overflowed the destination format. Latches until cleared.

Bit 1—Reserved Operand Exception Bit (REX): A logic High indicates that a reserved operand appeared as an input operand to an operation or was generated as a result. Latches until cleared.

Bit 0—Invalid Operation Exception Bit (IEX): A logic High indicates that input operands are unsuitable for the operation performed (e.g., $\infty \times 0$). Latches until cleared.

Flag Register

The flag register contains 7 flag bits that report exception or Boolean results for the most recently performed operation; its format is shown in Figure 5. The remaining 25 register bits are reserved for future use. The Am29000 can read the current flag register contents by issuing a read flags transaction request.

Flag register bits 6–0 correspond to Flag 6–Flag 0 (FL_6–FL_0).

These flags assume a meaning that is operation-dependent, as discussed in the Operation Flags section.

The flag register is made transparent in flow-through mode.

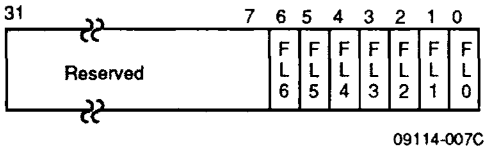


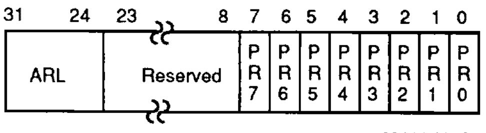
Figure 5. Flag Register

Precision Register

The precision register contains 8 bits that report the precision of operands stored in the register file; its format is shown in Figure 6. Bit 0 (PR_0) reports the precision of register file location 0 (RF_0), bit 1 the precision of location 1 (RF_1), and so on. A logic High indicates a single-precision value, logic Low a double-precision value.

The precision register also contains the Accelerator Release Level (ARL), an 8-bit, read-only identification number that specifies the accelerator version. The ARL field occupies bits 31–24.

The remaining 16 bits of the precision word are reserved for future use, and must be set to 0 when written to assure future compatibility.

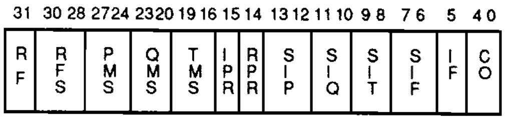


09114-008A

Figure 6. Precision Register

Instruction Register, I-Temp Register

The instruction register contains a 32-bit instruction word that specifies the ALU operation; its format is shown in Figure 7.



09114-009A

Figure 7. Instruction Register

Bit 31—Register File Enable (RF): Enables a write to the register file. When RF is High, the operation result is written to the register file location specified by RFS and the resulting precision is written to the corresponding bit of the precision register. When RF is Low, no write is performed either to the register file or the precision register.

Bits 30–28—Register file select (RFS): Selects the register file location (RF_7–RF_0) to which the operation result is to be written. If bit RF is Low, the value of RFS is a "don't care."

Bits 27–24—Select for P Operand Multiplexer (PMS): Selects the data input for the ALU P port.

Bits 23–20—Select for Q Operand Multiplexer (QMS): Selects the data input for the ALU Q port.

Bits 19–16—Select for T Operand Multiplexer (TMS): Selects the data input for the ALU T port.

Bit 15—Input Precision (IPR): Precision of the operands in Registers R and S; single precision when High, double precision when Low.

Bit 14—Result Precision (RPR): Precision of the ALU output; single precision when High, double precision when Low.

Bits 13–12—Sign P (SIP): Sign-change control for the ALU P input.

Bits 11–10—Sign Q (SIQ): Sign-change control for the ALU Q input.

Bits 9–8—Sign T (SIT): Sign-change control for the ALU T input.

Bits 7–6—Sign F (SIF): Sign-change control for the ALU output.

Bit 5—Integer/Floating-point Select (IF): A logic Low selects a floating-point operation, a logic High an integer operation.

Bits 4–0—Core Operation (CO): Specifies the core operation to be performed by the ALU.

The function of the instruction word fields is discussed in further detail in the Accelerator Instruction Set section.

The I-Temp register has a format identical to that of the instruction register; this register is used to temporarily buffer instructions for pending operations, thus allowing the overlap of operation specification and execution.

The Am29000 can write to the instruction and I-Temp registers by issuing the write instruction transaction request, and can read the contents of these registers as part of the save state sequence.

Operand Registers

The Am29027 holds operands in thirteen 64-bit registers. Four registers—R, S, R-Temp, and S-Temp—store ALU input operands; a fifth register, F, stores ALU results. Eight remaining registers, RF_7–RF_0, are arranged as a file into which operation results can be written, and from which operands can be taken for use in subsequent operations.

All operand registers share common data formats; any register can hold a single- or double-precision floating-point number, or a single- or double-precision integer.

Floating-point numbers are stored with the sign bit in the most significant bit (bit 63) of the operand register. For single-precision numbers, the 32 LSBs of the register are unused; the value of these unused bits is a "don't care."

Integer numbers are stored with the least significant bit placed in the least significant bit (bit 0) of the operand register. For single-precision numbers, the 32 MSBs of the register are unused; the value of these unused bits is a "don't care." Floating-point and integer formats are described in further detail in Appendix A.

Accelerator Transaction Requests

The Am29000 controls the Am29027 with 13 transaction requests. Transaction request type is indicated by the state of four signals: $R/\overline{W}$ and OPT_2–OPT_0. Table 1 lists the transaction types and corresponding signal states.

Transaction requests are conditioned by signal $DREQT_1$ (which when High indicates an accelerator transaction) and signals $\overline{BINV}$ and $\overline{DREQ}$. The Am29027 will recognize a transaction request only if $DREQT_1$ and $\overline{BINV}$ are High and $\overline{DREQ}$ is Low.

Signal $DREQT_0$ modifies the execution of most transaction requests. For transaction requests that transfer operands or instructions to the Am29027, asserting $DREQT_0$ will start the execution of an accelerator operation. For transaction requests that transfer operation results, status, or flags to the Am29000, asserting $DREQT_0$ will suppress the reporting of unmasked exceptions via signal $\overline{DERR}$. For the write status transaction request, asserting $DREQT_0$ either retimes the operation currently described by the instruction register (flow-through mode) or invalidates the ALU pipeline (pipeline mode).

Write Transaction Requests

Write transactions transfer data from the Am29000 to the Am29027, or cause the Am29027 to transfer data internally. To perform a write request, the Am29000:

- Issues the appropriate transaction request on signals OPT_2–OPT_0, and asserts signal $R/\overline{W}$ Low
- Places the data to be transferred, if any, on output signals D_{31}–D_0 and A_{31}–A_0

The Am29027 responds to the request by asserting one (and only one) of two status signals:

- $\overline{CDA}$ indicates that the Am29027 will take the specified action and clock in the data accompanying the transaction request, if any, on the next rising edge of clock.
- $\overline{DERR}$ indicates that the Am29027 is unable to accept the data, due to the presence of an unmasked exception.

Timing for write transactions is illustrated in Appendix D.

Table 1. Transaction Requests

$R/\overline{W}$	OPT_2	OPT_1	OPT_0	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	Advance Temp Registers
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

There are eight write transactions:

Write Operand R: An operand is written to Input Register R and/or R-Temp. The most significant half of the 64-bit operand to be written is placed on Input Bus R, the least significant half on Input Bus S. The action taken depends on signal $DREQT_0$ and on whether an accelerator operation will be in progress during the next clock cycle.

$DREQT_0$ asserted	Operation in progress next clock cycle	Data written to	R-Temp valid bit	Operation pending bit
No	X	R-Temp	Set	Unchanged
Yes	No	R-Temp, R	Reset	Reset
Yes	Yes	R-Temp	Set	Set

If $DREQT_0$ is asserted and no accelerator operation will be in progress during the next clock cycle, a new operation will be started on the next rising edge of CLK.

If mode register bit HE (Halt On Error Enable) is High and an unmasked exception has been detected, the Am29027 will respond to a write operand R request by asserting signal $\overline{DERR}$; the contents of Registers R and R-Temp will not be changed, and the R-Temp Valid and Operation Pending bits will retain their current values.

Write Operand S: An operand is written to Input Register S and/or S-Temp. The most significant half of the 64-bit operand to be written is placed on Input Bus R, the least significant half on Input Bus S. The action taken depends on signal $DREQT_0$ and on whether an accelerator operation will be in progress during the next clock cycle.

DREQT$_0$ asserted	Operation in progress next clock cycle	Data written to	S-Temp valid bit	Operation pending bit
No	X	S-Temp	Set	Unchanged
Yes	No	S-Temp, S	Reset	Reset
Yes	Yes	S-Temp	Set	Set

If DREQT$_0$ is asserted and no accelerator operation will be in progress during the next clock cycle, a new operation will be started on the next rising edge of CLK.

If mode register bit HE (Halt On Error Enable) is High and an unmasked exception has been detected, the Am29027 will respond to a write operand S request by asserting signal $\overline{DERR}$; the contents of Registers S and S-Temp will not be changed, and the S-Temp Valid and Operation Pending bits will retain their current values.

Write Operands R, S: Two 32-bit operands are written to Registers R and S and/or Registers R-Temp and S-Temp. The 32-bit operand to be written to Registers R or R-Temp is placed on Input Bus R; the 32-bit operand to be written to Registers S or S-Temp is placed on Input Bus S. Each 32-bit word is written to both the upper and lower halves of the target register. The action taken depends on signal DREQT$_0$ and on whether an accelerator operation will be in progress during the next clock cycle.

DREQT$_0$ asserted	Operation in progress next clock cycle	Data written to	R-, S-Temp valid bits	Operation pending bit
No	X	R-Temp S-Temp	Set	Unchanged
Yes	No	R-Temp S-Temp R, S	Reset	Reset
Yes	Yes	R-Temp S-Temp	Set	Set

If DREQT$_0$ is asserted and no accelerator operation will be in progress during the next clock cycle, a new operation will be started on the next rising edge of CLK.

If mode register bit HE (Halt On Error Enable) is High and an unmasked exception has been detected, the Am29027 will respond to a write operands R, S request by asserting signal $\overline{DERR}$; the contents of Registers R, R-Temp, S, and S-Temp will not be changed, and the R-Temp Valid, S-Temp Valid, and Operation Pending bits will retain their current values.

Write Mode: A 64-bit word is written to the mode register. The least significant half of the mode word is placed on Input Bus R, the most significant half on Input Bus S. The state of signal DREQT$_0$ is a "don't care" for this transaction request.

Write Status: A 32-bit word is written to the status register and the status word to be written is placed on Input Bus R. Asserting signal DREQT$_0$ will produce an additional action that is mode-dependent. In flow-through mode, asserting DREQT$_0$ will cause the operation currently specified by the instruction register to be retimed; operation results will not be written to the status register or the register file. In pipeline mode, asserting DREQT$_0$ will invalidate the ALU pipeline.

Write Register File Precisions: A 32-bit word indicating the precisions of register file locations RF_7–RF_0 is written to the precision register; the precision word to be written is placed on Input Bus R. The state of signal DREQT$_0$ is a "don't care" for this transaction request.

Write Instruction: A 32-bit accelerator instruction is written to the instruction register and/or Register I-Temp. The 32-bit instruction is taken from input signals I_{31}–I_0. The action taken depends on signal DREQT$_0$, and on whether an accelerator operation will be in progress during the next clock cycle.

DREQT$_0$ asserted	Operation in progress next clock cycle	Data written to	I-Temp valid bit	Operation pending bit
No	X	I-Temp	Set	Unchanged
Yes	No	I-Temp instruction register	Reset	Reset
Yes	Yes	I-Temp	Set	Set

If DREQT$_0$ is asserted and no accelerator operation will be in progress during the next clock cycle, a new operation will be started on the next rising edge of CLK.

If mode register bit HE (Halt On Error Enable) is High and an unmasked exception has been detected, the Am29027 will respond to a write instruction transaction request by asserting signal $\overline{DERR}$; the contents of Register I-Temp and the instruction register will not be changed, and the I-Temp Valid and Operation Pending bits will retain their current values.

Advance Temp Registers: The contents of the R-Temp, S-Temp, and I-Temp registers are transferred to Register R, Register S, and the instruction register, respectively. The state of signal DREQT$_0$ is a "don't care" for this transaction request. The advance temp registers transaction request is used during restoration of accelerator state.

Read Transaction Requests

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_2–OPT_0, and asserts signal $R/\overline{W}$ High.
- Places its data bus drivers in a high-impedance state.

The Am29027 then places the requested data on signals F_{31}–F_0 and issues two status signals:

- $\overline{DRDY}$ indicates that the data requested is available on Output Bus F_{31}–F_0.
- $\overline{DERR}$ indicates that the Am29027 has detected an unmasked exception; the exception may or may not be related to the data requested.

$\overline{DRDY}$ and $\overline{DERR}$ may both be active at the same time; if so, the Am29000 will respond to $\overline{DERR}$ and ignore $\overline{DRDY}$.

Timing for read transactions is illustrated in Appendix D.

There are five read transactions:

Read Result MSBs: The 32 MSBs of Register F are placed on output bus F. Asserting signal $DREQT_0$ will suppress the reporting of unmasked exceptions.

Read Result LSBs: The 32 LSBs and 32 MSBs of Register F are placed on Output Bus F in consecutive clock cycles. Asserting signal $DREQT_0$ will suppress the reporting of unmasked exceptions. The read result LSBs request must always be followed by a read result MSBs request.

Read Flags: The flag register contents are placed on Output Bus F; bits F_{31}–F_7 will be logic Low. Asserting signal $DREQT_0$ will suppress the reporting of unmasked exceptions.

Read Status: The status register contents are placed on Output Bus F; bits F_{31}–F_{11} will be logic Low. Asserting signal $DREQT_0$ will suppress the reporting of unmasked exceptions.

Save State: The contents of the instruction register, mode register, status register, register file, precision register, and Registers R, R-Temp, S, S-Temp, and I-Temp are transferred to the Am29000 via Output Bus F. Exception reporting via signal $\overline{DERR}$ is suppressed; the state of signal $DRETQ_0$ is a "don't care." Further details on the use of this request appear in the Saving and Restoring State sections.

Coprocessor Data Accept

The Coprocessor Data Accept ($\overline{CDA}$) signal indicates to the Am29000 that the Am29027 is able to accept new operands or instructions. $\overline{CDA}$ is normally Low (active), but will go High if:

- The Am29027 has an operation currently in progress and a completely specified pending operation waiting in the temporary registers,

or

- The Am29027 has halted in response to an unmasked exception (Halt On Error mode enabled).

If the Am29027 issues any write transaction request and $\overline{CDA}$ is active Low, the transaction request will complete in a single cycle. If $\overline{CDA}$ is High, response to a write transaction request depends on request type:

- For the write operand R, write operand S, write operands R, S, and write instruction transaction requests, the Am29027 will assert $\overline{CDA}$ active when it is able to accept new data. If it is not able to accept new data indefinitely due to presence of an unmasked exception (Halt On Error mode enabled), it will respond to the transaction request by asserting signal $\overline{DERR}$.
- For the write mode, write status, write register file precisions, and advance temp registers transaction requests, the Am29027 will temporarily assert $\overline{CDA}$ during the cycle after the request is issued, regardless of whether an operation is in progress or an unmasked exception has halted the accelerator.

$\overline{CDA}$ pertains only to write transaction requests; for read transaction requests, the Am29000 ignores the state of $\overline{CDA}$.

Data Ready

The Data Ready ($\overline{DRDY}$) signal indicates to the Am29000 that the Am29027 is placing data on the F output bus. The Am29027 generates $\overline{DRDY}$ in response to the read result MSBs, read result LSBs, read status, read flags, and save state transaction requests.

For the read result MSBs, read result LSBs, read flags, and read status transaction requests, there is usually a minimum of one cycle delay between the time the request is issued and the time that $\overline{DRDY}$ is asserted. The only exception to this rule is when a read result LSBs request is immediately followed by a read result MSBs request, in which case the Am29027 responds to the second request in a single cycle. If the Am29027 is unable to respond immediately to a read transaction request, as may be the case when an operation is in progress, the $\overline{DRDY}$ signal will be held inactive until such a time as the requested data can be output. For the save state transaction request, the delay between the issuance of the transaction request and the $\overline{DRDY}$ response varies according to the specific data requested.

$\overline{DRDY}$ pertains only to read transaction requests; for write transaction requests, $\overline{DRDY}$ remains inactive.

Data Error

The Data Error ($\overline{DERR}$) signal indicates to the Am29000 that the Am29027 is unable to respond to a transaction request normally, due to the presence of an unmasked exception bit in the status register.

For read transaction requests, read result LSBs, read result MSBs, read flags, and read status, the Am29027 asserts $\overline{DERR}$ if the status register contains an unmasked exception bit. The Am29000 may suppress error reporting for these requests by issuing them with signal DREQT$_0$ asserted.

For write transaction requests, write operand R, write operand S, write operands R, S, and write instruction, $\overline{\text{DERR}}$ is issued in the presence of an unmasked exception if Halt On Error Mode is enabled in such an event, the contents of the target registers are left unchanged.

$\overline{\text{DERR}}$ is never issued in response to transaction requests write mode, write status, write register file precisions, advance temp registers, and save state.

Accelerator Instruction Set

The ALU performs 57 arithmetic and logic instructions. Input operands for these instructions can be taken from Input Registers R and S, register file locations RF$_7$–RF$_0$, and on-board constant stores. At the user's option, results can be stored in register file locations RF$_7$–RF$_0$.

Instruction Word

The 32-bit instruction word, IN$_{31}$–IN$_0$, specifies the operation to be performed by the ALU. The instruction word is stored in the instruction register; instruction register format is shown in Figure 7. In flow-through mode, the instruction word specifies the operation to be performed by the entire ALU. In pipeline mode, the instruction word specifies the operation to be performed by the first pipeline stage; the remaining pipeline stage or stages are controlled by their respective pipeline registers. The instruction word also specifies input operand sources, result destination, and operand precisions.

An instruction word comprises five sections: base operation code, sign-change selects, operand precision selects, operand source selects, and register file controls.

Base Operation Code

The base operation code consists of the core operation field (CO), which specifies the type of operation to be performed, and the integer/floating-point select bit (IF), which specifies whether the operation is integer or floating-point. Available base operation codes and the corresponding values for CO and IF are listed in Table 2. Note that the value of IF is a "don't care" for base operation code MOVE P.

Sign-Change Selects

Each ALU input and output port has associated hardware that can be used to modify operand signs (see Figure 8). These sign-change blocks, when applied to base operations, greatly increase the number of available operations. The base operation code F′ = P′ + T′, for example, can be used to perform operations such as P – T, ABS(P) + ABS(T), ABS(P + T), and others, simply by modifying the signs of the input and output operands. The SIP, SIQ, and SIT instruction word fields control the sign-change blocks for the P, Q, and T input operands, respectively; the SIQ and SIF fields control the sign change block for output operand F.

Using the sign-change blocks, the sign of an input operand may be left unchanged, inverted, set Low, or set High; the sign of the output operand may be left unchanged, inverted, set Low, set High, set to the sign of the P input operand, or set to the sign of the T input operand. Select codes for the P, Q, T, and F sign-change blocks are shown in Tables 3, 4, 5, and 6, respectively.

Operand Precision Selects

The Am29027 supports mixed-precision operations; it is possible, for example, to perform an operation having single-precision inputs and a double-precision output, or one single- and one double-precision input, or any other combination.

The precision of the operands in Registers R and S is specified by instruction bit IPR, which is logic High for single-precision operands and logic Low for double-precision operands. Note that the operands in the R and S registers must have the same precision if they are to be used in the same operation. This restriction does not preclude performing an operation with mixed-precision input operands, as there are no restrictions on the precisions of operands stored in the register file. The precision of each operand stored in the register file is recorded in the precision register; this precision information is automatically supplied to the ALU when a register file location is specified as an input operand to an operation.

The precision of an operation result is specified by instruction bit RPR, which is set High for a single-precision result, and Low for a double-precision result. Should the instruction word specify that the result is to be written to the register file (instruction bit RF High), the resulting precision will be written to the appropriate precision register bit when the result is written to the register file.

Table 2. Operation Codes

IF	CO					
IN_5	IN_4	IN_3	IN_2	IN_1	IN_0	**Base Operation Code (Floating-Point)**
0	0	0	0	0	0	$F' = P$
0	0	0	0	0	1	$F' = P' + T'$
0	0	0	0	1	0	$F' = P' \times Q'$
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	$F' = (P' \times Q') + T'$
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
IN_5	IN_4	IN_3	IN_2	IN_1	IN_0	**Base Operation Code (Integer)**
1	0	0	0	0	0	$F = P$
1	0	0	0	0	1	$F = P + T$
1	0	0	0	1	0	$F = P \times Q$
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	F = P OR T
1	1	0	0	0	1	F = P AND T
1	1	0	0	1	0	F = 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
IN_5	IN_4	IN_3	IN_2	IN_1	IN_0	**Base Operation Code (Special)**
X	1	1	0	0	0	MOVE P

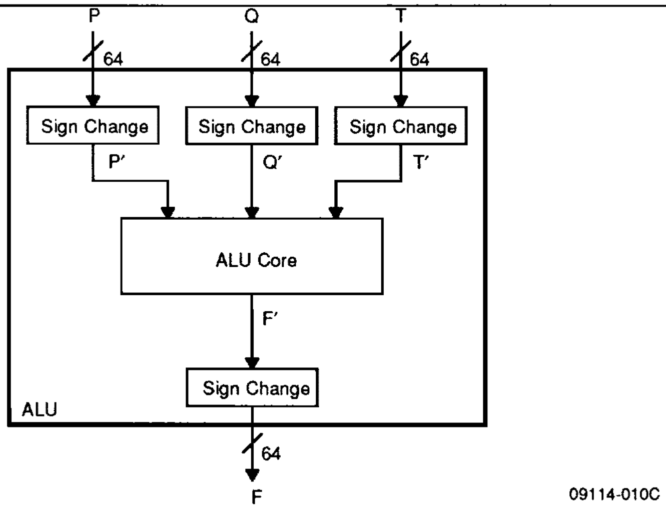


Figure 8. ALU Sign-Change Blocks

Table 3. Select Codes for P Operand Sign-Change Block

SIP		
IN_{13}	IN_{12}	SIGN (P′)
0	0	SIGN(P)
0	1	$\overline{\text{SIGN}}$ ($\overline{\text{P}}$)
1	0	0
1	1	1

Table 4. Select Codes for Q Operand Sign-Change Block

SIQ		
IN_{11}	IN_{10}	SIGN (Q′)
0	0	SIGN(Q)
0	1	$\overline{\text{SIGN}}$ ($\overline{\text{Q}}$)
1	0	0
1	1	1

Table 5. Select Codes for T Operand Sign-Change Block

SIT		
IN_9	IN_8	SIGN (T′)
0	0	SIGN(T)
0	1	$\overline{\text{SIGN}}$ ($\overline{\text{T}}$)
1	0	0
1	1	1

Table 6. Select Codes for F Operand Sign-Change Block

	SIQ		SIF		
Base Operation	IN_{11}	IN_{10}	IN_7	IN_6	**SIGN(F)**
F′ = P (Floating-Point)	0	X	0	0	SIGN(F′)
F = P (Integer)	0	X	0	1	$\overline{\text{SIGN}}$($\overline{\text{F}}$′)
<u>OR</u>	0	X	1	0	0
Maximum P, T	0	X	1	1	1
<u>OR</u>	1	0	X	X	SIGN(P)
Minimum P, T	1	1	X	X	SIGN(T)
	X	X	0	0	SIGN(F′)
<u>All Other</u> Base	X	X	0	1	$\overline{\text{SIGN}}$($\overline{\text{F}}$′)
Operations	X	X	1	0	0
	X	X	1	1	1

Operand Source Selects

Instruction fields PMS, QMS, and TMS specify the select codes for the P, Q, and T operand multiplexers, respectively; these codes are summarized in Table 7.

The P, Q, and T operand multiplexers can independently select Register R, Register S, register file locations RF_7–RF_0, or one of six predefined constants. For operations with floating-point inputs, constants 0, 0.5, 1, 2, 3, and pi are available; for operations with integer inputs, constants 0, –1, 1, 2, 3, and $-(2^{63})$ are available. These constants are supplied to the ALU as double-precision numbers, independent of the precisions specified for other input and result operands. Hexadecimal values for the constants are listed in Table 8.

Register File Controls

Instruction fields RF and RFS control the storing of operation results in the register file. If register file enable bit RF is High, the result of the operation specified by the instruction word will be stored in register file location RFS, where RFS is a number from 7 to 0; the precision of the result, as specified by the RPR bit, will be written to the appropriate bit in the precision register. If RF is Low, the operation result is written to neither the register file nor the precision register.

Accelerator Operations

Table 9 illustrates a number of possible ALU instructions and corresponding values for instruction word fields SIP, SIQ, SIT, SIF, IF, and CO. Note that the remaining instruction fields—RF, RFS, PMS, QMS, TMS, IPR, and RPR—can be specified independently.

The user may create additional instructions using instruction words other than those listed in Table 9. For some base operation codes, sign-change control settings SIP, SIQ, SIT, and SIF are completely arbitrary; for others, only the sign-change field values shown in Table 9 are valid. Table 10 summarizes permissible sign-change field values for each base operation code.

Table 7. Select Codes for P, Q, and T Operand Multiplexers

PMS	IN_{27}	IN_{26}	IN_{25}	IN_{24}	P
QMS	IN_{23}	IN_{22}	IN_{21}	IN_{20}	**Q**
TMS	IN_{19}	IN_{18}	IN_{17}	IN_{16}	**T**
	0	0	0	0	Register R
	0	0	0	1	Register S
	0	0	1	0	0 (Zero)
	0	0	1	1	0.5 (F.P.) – 1(integer)
	0	1	0	0	1
	0	1	0	1	2
	0	1	1	0	3
	0	1	1	1	π (F.P.) – 2^{63}(integer)
	1	0	0	0	RF_0
	1	0	0	1	RF_1
	1	0	1	0	RF_2
	1	0	1	1	RF_3
	1	1	0	0	RF_4
	1	1	0	1	RF_5
	1	1	1	0	RF_6
	1	1	1	1	RF_7

Table 8. Hexadecimal Values for On-Chip Constants

IEEE Floating-Point Constant	Hexadecimal Representation
0	0000000000000000
0.5	3FE0000000000000
1	3FF0000000000000
2	4000000000000000
3	4008000000000000
π	400921FB54442D18

DEC D Floating-Point Constant	Hexadecimal Representation
0	0000000000000000
0.5	4000000000000000
1	4080000000000000
2	4100000000000000
3	4140000000000000
π	41490FDAA22168C2

DEC G Floating-Point Constant	Hexadecimal Representation
0	0000000000000000
0.5	4000000000000000
1	4010000000000000
2	4020000000000000
3	4028000000000000
π	402921FB54442D18

IBM Floating-Point Constant	Hexadecimal Representation
0	0000000000000000
0.5	4080000000000000
1	4110000000000000
2	4120000000000000
3	4130000000000000
π	413243F6A8885A31

Integer Constant	Hexadecimal Representation
0	0000000000000000
–1	FFFFFFFFFFFFFFFF
1	0000000000000001
2	0000000000000002
3	0000000000000003
-2^{63}	8000000000000000

Table 9. Instruction Words for Typical ALU Operations

Operation	SIP	SIQ	SIT	SIF	IF	CO
FP P	00	00	XX	00	0	00000
FP – P	00	00	XX	01	0	00000
FP ABS(P)	00	00	XX	10	0	00000
FP Sign(T) × ABS(P)	00	11	XX	XX	0	00000
FP P + T	00	XX	00	00	0	00001
FP P – T	00	XX	01	00	0	00001
FP T – P	01	XX	00	00	0	00001
FP –P – T	01	XX	01	00	0	00001
FP ABS(P + T)	00	XX	00	10	0	00001
FP ABS(P – T)	00	XX	01	10	0	00001
FP ABS(P) + ABS(T)	10	XX	10	00	0	00001
FP ABS(P) – ABS(T)	10	XX	11	00	0	00001
FP ABS[ABS(P) – ABS(T)]	10	XX	11	10	0	00001
FP P × Q	00	00	XX	00	0	00010
FP (–P) × Q	01	00	XX	00	0	00010
FP ABS(P × Q)	00	00	XX	10	0	00010
FP Compare P, T	00	XX	01	00	0	00011
FP Max P, T	00	00	01	00	0	00100
FP Max ABS(P), ABS(T)	10	00	11	00	0	00100
FP Min P, T	01	00	00	00	0	00101
FP Min ABS(P), ABS(T)	11	00	10	00	0	00101
FP Limit P to Magnitude T	11	10	10	XX	0	00101
FP Convert T to Integer	XX	XX	00	00	0	00110
FP Scale T to Integer by Q	XX	00	00	00	0	00111
FP T + P × Q	00	00	00	00	0	01000
FP T – P × Q	01	00	00	00	0	01000
FP –T + P × Q	00	00	01	00	0	01000
FP –T – P × Q	01	00	01	00	0	01000
FP ABS(T) + ABS(P × Q)	10	10	10	00	0	01000
FP ABS(T) – ABS(P × Q)	11	10	10	00	0	01000
FP ABS(P × Q) – ABS(T)	10	10	11	00	0	01000
FP Round T to Integral Value	XX	XX	00	00	0	01001
FP Reciprocal Seed (P)	00	XX	XX	00	0	01010
FP Convert T to Alternate Floating-Point Format	XX	XX	00	00	0	01011
FP Convert T from Alternate Floating-Point Format	XX	XX	00	00	0	01100
int P	00	00	00	00	1	00000
int –P	00	00	00	01	1	00000
int ABS(P)	00	00	00	10	1	00000
int Sign(T) × ABS(P)	00	11	00	XX	1	00000
int P + T	00	XX	00	00	1	00001
int P – T	00	XX	01	00	1	00001
int T – P	01	XX	00	00	1	00001
int ABS(P + T)	00	XX	00	10	1	00001
int ABS(P – T)	00	XX	01	10	1	00001
int P × Q	00	00	XX	00	1	00010
int Compare P, T	00	XX	01	00	1	00011
int Max P, T	00	00	01	00	1	00100
int Min P, T	01	00	00	00	1	00101

Table 9. Instruction Words for Typical ALU Operations (continued)

Operation	SIP	SIQ	SIT	SIF	IF	CO
int Convert T to Float	XX	XX	00	00	1	00110
int Scale T to Float by Q	XX	00	00	00	1	00111
int P OR T	XX	XX	XX	XX	1	10000
int P AND T	XX	XX	XX	XX	1	10001
int P XOR T	XX	XX	XX	XX	1	10010
int NOT T (see Note 1)	XX	XX	XX	XX	1	10010
int Shift P Logical Q Places	00	00	XX	00	1	10011
int Shift P Arithmetic Q Places	00	00	XX	00	1	10100
int Funnel Shift PT Q Places	00	00	00	00	1	10101
MOVE P	XX	XX	XX	XX	X	11000

Note 1. NOT T is performed by XORing T with a word containing all 1s (integer – 1). When invoking NOT T the user must set instruction field PMS to 0011_2, thus selecting integer constant –1.

Table 10. Allowable Sign-Change Combinations

IF	CO	Operation	SIP	SIQ	SIT	SIF
0	00000	FP F′ = P	F	V	X	V
0	00001	FP F′ = P′ + T′	V	X	V	V
0	00010	FP F′ = 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 Convert T to Integer	X	X	F	F
0	00111	FP Scale T to Integer	X	F	F	F
0	01000	FP F′ = (P′ × Q′) + T	V	V	V	V
0	01001	FP Round T	X	X	F	F
0	01010	FP Reciprocal Seed P	F	X	X	F
0	01011	FP Convert T to Alt Format	X	X	F	F
0	01100	FP Convert T from Alt Format	X	X	F	F
1	00000	int F = P	F	F	F	F
1	00001	int F = P + T	F	X	F	F
1	00010	int F = 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 Convert T to F.P.	X	X	F	F
1	00111	int Scale T to F.P.	X	F	F	F
1	10000	int F = P OR T	X	X	X	X
1	10001	int F = P AND T	X	X	X	X
1	10010	int F = P XOR T	X	X	X	X
1	10011	int Shift P Logical	F	F	X	F
1	10100	int Shift P Arithmetic	F	F	X	F
1	10101	int Funnel Shift PT	F	F	F	F
X	11000	MOVE P	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 9.
X = Don't care; this field does not affect the operation or its result.

Base Operation Code Description

F′ = P (Floating-Point): The P-operand is passed through the ALU unchanged, except for any specified precision conversions. If the user specifies different input and output precisions, the operation may be used to perform single-to-double or double-to-single conversions. Instructions such as negation, absolute value extraction and sign transfer may be executed by setting the sign-change controls appropriately while executing this base operation.

F′ = P′ + T′ (Floating-Point): The two operands P′ and T′ are added, taking into account any specified precision conversions. Instructions such as subtraction, sum-of-absolute-values, difference-of-absolute-values, absolute-value-of-sum, and absolute-value-of-difference may be executed by setting the sign-change controls appropriately while executing this base operation.

F′ = P′ × Q′ (Floating-Point): The operands P′ and Q′ are multiplied, taking into account any specified precision conversions. Instructions such as negative-product and absolute-value-of-product may be executed by setting the sign-change controls appropriately while executing this base operation.

Compare P, T (Floating-Point): The two operands P and T are compared, taking into account any specified precision conversions. The output of the operation is the result of the subtraction (P – T). The flags are set appropriately to indicate the result of the comparison, conforming to the relevant parts of the floating-point standards. For IEEE and DEC operations, one of four flags (greater than, less than, equal to, or unordered) is set for any given compare operation. For IBM operations, the unordered flag does not apply since the format does not support reserved operands.

Maximum P, T (Floating-Point): The two operands P and T are compared, taking into account any specified precision conversions. The most positive operand is selected as the output. The Winner flag indicates which of the operands is selected. Additionally, the operation maximum-of-absolute-value may be performed by setting the appropriate sign-change controls.

Minimum P, T (Floating-Point): The two operands P and T are compared, taking into account any specified precision conversions. The most negative operand is selected as the output. The Winner flag indicates which of the two operands is selected. Additionally, the operations minimum-of-absolute-values and limit-P-to-magnitude-T may be performed by setting the appropriate sign-change controls. The limit-P-to-magnitude-T operation is useful for clipping a sequence of operands to ensure that their magnitude never exceeds a preset limit.

Convert T to Integer (Floating-Point): The operand T is converted from floating-point representation to two's complement integer representation, taking into account the specified precision of the floating-point operand. If the output precision is specified as single, the result is a 32-bit integer. If the output precision is specified as double, the result is a 64-bit integer.

Scale T to Integer by Q (Floating-Point): The operand T is converted from floating-point representation to two's complement integer representation, using the exponent of the floating-point operand Q as a scale factor and taking into account the specified precision of the floating-point operands. The unbiased exponent of the operand Q is added to the exponent of the operand T, permitting IEEE and DEC operands to be multiplied by any power of 2, and IBM operands by any power of 16, before the conversion is performed. If the output precision is specified as single, the result is a 32-bit integer. If the output precision is specified as double, the result is a 64-bit integer.

F′ = (P′ × Q′) + T′ (Floating-Point): The operands P′ and Q′ are multiplied, producing a double-precision product. This product is added to the operand T′, taking into account any specified precision conversions. Instructions such as P × Q – T, T – P × Q, ABS (P × Q) + ABS(T) and ABS(P × Q + T) may be executed by setting the sign-change controls appropriately while executing this base operation.

Round T to Integral Value (Floating-Point): The floating-point operand T is rounded to an integer-valued floating-point operand, using the specified rounding mode and taking into account any specified precision conversions. As an example, the operation converts a floating-point representation of Pi (3.14159 . . .) to a floating-point representation of 3.0 or 4.0, depending on the rounding mode selected. The final result of the operation is a floating-point number.

Reciprocal Seed of P (Floating-Point): An approximation to the reciprocal of the operand P is evaluated, taking into account any specified precision conversions. The reciprocal seed comprises an accurate sign, a fully-accurate exponent and a mantissa that is accurate to only one place. This operation can be used as the initial step in performing Newton-Raphson division; optionally, an external seed look-up table can be used for faster convergence.

Convert T to Alternate Floating-Point Format (Floating-Point): The floating-point operand T, assumed to be in the primary floating-point format, is converted to a floating-point operand in the alternate floating-point format, taking into account any specified precision conversions.

Convert T from Alternate Floating-Point Format (Floating-Point): The floating-point operand T, assumed to be in the alternate floating-point format, is converted to a floating-point operand in the primary floating-point format, taking into account any specified precision conversions.

F = P (Integer): The P-operand is passed through the ALU unchanged except for any specified precision conversions. If the user specifies different input and output precisions, the operation may be used to perform single-to-double or double-to-single conversions. Instructions such as negation, absolute value extraction, and sign transfer may be performed by setting the sign-change control appropriately while executing this base operation.

F = P + T (Integer): The two operands P and T are added, taking into account any specified precision conversions. Instructions such as subtraction, absolute-value-of-sum, and absolute-value-of-difference may be performed by setting the sign-change controls appropriately while executing this base operation.

F = P × Q (Integer): The two operands P and Q are multiplied, taking into account any specified precision conversions. Either 32-bit multiplication or 64-bit multiplication may be performed, and the user may select either the MSBs or the LSBs of the product as the final result. In addition, format-adjusting may be implemented if required, and the operands may be considered as signed (two's complement) or unsigned.

Compare P, T (Integer): The two operands P and T are compared, taking into account any specified precision conversions. The output of the operation is the result of the subtraction (P – T). The flags are set appropriately to indicate the result of the comparison, one of three flags (greater than, less than, or equal to) being set for any given compare operation.

Maximum P, T (Integer): The two operands P and T are compared, taking into account any specified precision conversions. The most positive operand is selected as the output. The Winner flag indicates which of the two operands is selected.

Minimum P, T (Integer): The two operands P and T are compared, taking into account any specified precision conversions. The most negative operand is selected as the output. The Winner flag indicates which of the two operands is selected.

Convert T to Floating-Point (Integer): The operand T is converted from two's complement integer representation to floating-point representation, taking into account the specified precision of the integer operand. If the output precision is specified as single, the result is a 32-bit floating-point operand. If the output precision is specified as double, the result is a 64-bit floating-point operand.

Scale T to Floating-Point by Q (Integer): The operand T is converted from two's complement integer representation to floating-point representation, using the exponent of the floating-point operand Q as a scale factor and taking into account the specified precision of the integer operand. The unbiased exponent of the operand Q is added to the exponent of the floating-point result, permitting IEEE and DEC operands to be multiplied by any power of 2, and IBM operands by any power of 16 after the conversion is performed. If the output precision is specified as single, the result is a 32-bit floating-point operand. If the output precision is specified as double, the result is a 64-bit floating-point operand.

F = P OR T (Integer): The operand P is logically ORed with the operand T. Before the operation is performed, the inputs, if 32-bit, are sign-extended to 64 bits.

F = P AND T (Integer): The operand P is logically ANDed with the operand T. Before the operation is performed, the inputs, if 32-bit, are sign-extended to 64 bits.

F = P XOR T (Integer): The operand P is logically exclusive-ORed with the operand T. Before the operation is performed, the inputs, if 32-bit, are sign-extended to 64 bits. This operation may be used to invert an operand by selecting the second operand to be the integer constant, –1, so that all bits of this second operand are 1. Exclusive-ORing an operand with –1 is equivalent to inverting each bit in the operand.

Shift P Logical Q Places (Integer): This operation cannot be performed in mixed-precision mode. The precision of the result is the same as the precision of the input operand P. A two's-complement shift length in the range –64 to +63 (double-precision) or –32 to +31 (single-precision) is extracted from the LSBs of the operand Q. The operand P is logically right-shifted by the number of places specified by the shift length. A negative shift length therefore produces a left-shift. If a right-shift is performed, 0s fill vacated bit positions to the left of the input operand. If a left-shift is performed, 0s fill vacated bit positions to the right of the input operand.

Shift P Arithmetic Q Places (Integer): This operation cannot be performed in mixed-precision mode. The precision of the result is the same as the precision of the input operand P. A two's-complement shift length in the range –64 to +63 (double-precision) or –32 to +31 (single-precision) is extracted from the LSBs of the operand Q. The operand P is arithmetically right-shifted by the number of places specified by the shift length. A negative shift length therefore produces a left-shift. If a right-shift is performed, the MSB (bit 63 or 31) is replicated to fill vacated bit positions to the left of the input operand. If a left-shift is performed, 0s fill vacated bit positions to the right of the input operand.

Funnel Shift PT Q Places (Integer): This operation cannot be performed in mixed-precision mode. The operand T is interpreted as having the same precision as the input operand P, and the precision of the result is also the same as the precision of the input operand P. A two's-complement shift length in the range –64 to +63 (double-precision) or –32 to +31 (single-precision) is extracted from the LSBs of the operand Q. A triple-width operand (96-bit or 192-bit) is formed by concatenating the input operands into the arrangement P-T-P, with the 32-bit or 64-bit result field initially aligned with the T-operand. The triple-width operand is logically right-shifted by the number of places specified by the shift length. A negative shift length therefore produces a left-shift.

Move P (Floating-Point or Integer): The 64-bit operand P is passed unchanged through the ALU. No exceptions are detected or signaled.

Primary and Alternate Floating-Point Formats

Two mode register fields, PFF and AFF, specify the primary and alternate floating-point formats used by the ALU. All floating-point operations except format conversions are performed in the format specified by PFF. For format conversion operations, either primary floating-point format PFF or alternate floating-point format AFF are used as follows:

- For conversions between floating-point and integer formats (base operation codes Convert T to integer, Convert T to floating-point, Scale T to integer by Q, Scale T to floating-point by Q), the floating-point source or destination format is specified by PFF; for the scale operations, the format of operand Q is also specified by PFF.
- When converting from the primary floating-point format to the alternate floating-point format (base operation code Convert T to alternate F. P. format), an operand in format PFF is converted to format AFF.
- When converting from the alternate floating-point format to the primary floating-point format (base operation code Convert T to primary F.P. format), an operand in format AFF is converted to format PFF.

Operation Precision

The ALU performs all operations in double-precision format. All single-precision input operands are converted to double-precision equivalents by the ALU at the start of an operation. If the operation is to report a single-precision result, the ALU converts the double-precision internal result to single-precision at the end of the operation.

Note that operation flags and exception bits pertain to the source and destination precisions. If, for example, an operation produces a single-precision overflowed result, an overflow is indicated regardless of whether that result overflows the double-precision internal format.

Operation Flags

For each operation, the ALU produces thirteen flags. Of these, a maximum of seven are relevant to any given operation. The relevant flags are placed in the flag register in the manner shown in Table 11. All flags are active High. In flow-through mode the flag register is made transparent, and the selected flags are presented directly to the output multiplexer.

The ALU flags are:

C—CARRY: Carry-out bit produced by integer addition, subtraction, or comparison.

I—INVALID OPERATION: Indicates that the input operands are unsuitable for the operation performed (e.g., $\infty \times 0$).

R—RESERVED OPERAND: Indicates that the operation result is a reserved operand. Reserved operands include signaling or quiet NaNs in IEEE format, and DEC reserved operands in DEC D or G formats.

S—SIGN: Result sign; Low for a non-negative result, High for a negative result.

U—UNDERFLOW: Indicates that the operation result underflowed the destination format.

V—OVERFLOW: Indicates that the operation result overflowed the destination format.

W—WINNER: Indicates which of two input operands is reported as the result of the MAX P, T and MIN P, T operations. A logic High indicates that operand T is reported as the result, a logic Low operand P.

X—INEXACT RESULT: Indicates that the operation result had to be rounded to fit the destination format.

Z—ZERO RESULT: Indicates that the operation produced a zero result. Note that the result is exactly zero only if the Z flag is High and the X flag is Low.

>, =, <, #—GREATER THAN, EQUAL TO, LESS THAN, UNORDERED: Used to report the result of an operation with the Compare P, T base operation code. The Greater Than flag indicates that $P > T$, the Equal To flag that $P = T$, and the Less Than flag that $P < T$. The Unordered flag indicates that one or both input operands are reserved operands and cannot be compared. Note that the Unordered flag cannot arise when comparing IBM floating-point operands or integers. Exactly one comparison flag will be active per comparison operation.

Table 11. Organization of Flags

Format	Operation	CO IN_4–IN_0	Flag Register FL 6	FL 5	FL 4	FL 3	FL 2	FL 1	FL 0
IEEE	F′ = P′	00000	S	Z	X	U	V	R	I
IEEE	F′ = P′ + T′	00001	S	Z	X	U	V	R	I
IEEE	F′ = P′ × Q′	00010	S	Z	X	U	V	R	I
IEEE	Compare P, T	00011	S	=	>	<	#	R	I
IEEE	Maximum P, T	00100	S	Z		W		R	I
IEEE	Minimum P, T	00101	S	Z		W		R	I
IEEE	Convert T to Integer	00110	S	Z	X		V	R	I
IEEE	Scale T to Integer	00111	S	Z	X		V	R	I
IEEE	F′ = (P′ × Q′) + T′	01000	S	Z		U	V	R	I
IEEE	Round T to Integral Value	01001	S	Z	X		V	R	I
IEEE	Reciprocal Seed of P	01010	S	Z		U	V	R	I
IEEE	Convert T to Alt F.P. Format	01011	S	Z	X	U	V	R	I
IEEE	Convert T from Alt F.P. Format	01100	S	Z	X	U	V	R	I
DEC D	F′ = P′	00000	S	Z	X		V	R	
DEC D	F = P′ + T′	00001	S	Z	X	U	V	R	
DEC D	F′ = P′ × Q′	00010	S	Z	X	U	V	R	
DEC D	Compare P, T	00011	S	=	>	<	#	R	
DEC D	Maximum P, T	00100	S	Z		W		R	
DEC D	Minimum P, T	00101	S	Z		W		R	
DEC D	Convert T to Integer	00110	S	Z	X		V	R	I
DEC D	Scale T to Integer	00111	S	Z	X		V	R	I
DEC D	F′ = (P′ × Q′) + T′	01000	S	Z		U	V	R	
DEC D	Round T to Integral Value	01001	S	Z	X		V	R	
DEC D	Reciprocal Seed of P	01010	S	Z		U	V	R	I
DEC D	Convert T to Alt F.P. Format	01011	S	Z	X	U	V	R	I
DEC D	Convert T from Alt F.P. Format	01100	S	Z	X	U	V	R	I
DEC G	F′ = P′	00000	S	Z	X	U	V	R	
DEC G	F = P′ + T′	00001	S	Z	X	U	V	R	
DEC G	F′ = P′ × Q′	00010	S	Z	X	U	V	R	
DEC G	Compare P, T	00011	S	=	>	<	#	R	
DEC G	Maximum P, T	00100	S	Z		W		R	
DEC G	Minimum P, T	00101	S	Z		W		R	
DEC G	Convert T to Integer	00110	S	Z	X		V	R	I
DEC G	Scale T to Integer	00111	S	Z	X		V	R	I
DEC G	F′ = (P′ × Q′) + T′	01000	S	Z		U	V	R	
DEC G	Round T to Integral Value	01001	S	Z	X		V	R	
DEC G	Reciprocal Seed of P	01010	S	Z		U	V	R	I
DEC G	Convert T to Alt F.P. Format	01011	S	Z	X	U	V	R	I
DEC G	Convert T from Alt F.P. Format	01100	S	Z	X	U	V	R	I
IBM	F′ = P′	00000	S	Z	X		V		
IBM	F = P′ + T′	00001	S	Z	X	U	V		
IBM	F′ = P′ × Q′	00010	S	Z	X	U	V		
IBM	Compare P, T	00011	S	=	>	<			
IBM	Maximum P, T	00100	S	Z		W			
IBM	Minimum P, T	00101	S	Z		W			
IBM	Convert T to Integer	00110	S	Z	X		V		
IBM	Scale T to Integer	00111	S	Z	X		V		
IBM	F′ = (P′ × Q′) + T′	01000	S	Z		U	V		
IBM	Round T to Integral Value	01001	S	Z	X		V		
IBM	Reciprocal Seed of P	01010	S	Z			V		I
IBM	Convert T to Alt F.P. Format	01011	S	Z	X	U	V	R	
IBM	Convert T from Alt F.P. Format	01100	S	Z	X	U	V	R	I

Table 11. Organization of Flags (continued)

Format	Operation	CO IN_4–IN_0	Flag Register FL 6	FL 5	FL 4	FL 3	FL 2	FL 1	FL 0
Integer	F = P	00000	S	Z			V		
Integer	F = P + T	00001	S	Z			V		C
Integer	F = P × Q	00010	S	Z			V		
Integer	Compare P, T	00011	S	=	>	<	V		C
Integer	Maximum P, T	00100	S	Z		W			
Integer	Minimum P, T	00101	S	Z		W			
Integer	Convert T to Floating-Point	00110	S	Z	X				
Integer	Scale T to Floating-Point	00111	S	Z	X	U	V	R	
Integer	F = P OR T	10000	S	Z					
Integer	F = P AND T	10001	S	Z					
Integer	F = P XOR T	10010	S	Z					
Integer	Logical Shift P by Q Places	10011	S	Z					
Integer	Arithmetic Shift P by Q Places	10100	S	Z			V		
Integer	Funnel Shift P T by Q Places	10101	S	Z					
	MOVE P	11000	S						

Note: Unused flags assume the Low state.

Updating the Status Register

The status register exception bits are updated at the conclusion of each operation in flow-through mode, and at the start of each operation in pipeline mode. An exception bit is updated only if the operation reports that exception with a flag. For example, an IEEE floating-point addition operation produces an overflow flag and would therefore update the overflow exception bit; an IEEE floating-point comparison operation, on the other hand, does not produce an overflow flag and would therefore leave the overflow exception bit unchanged.

The mode register exception mask bits do not affect the updating of the status register exception bits—masked exceptions still appear in the status register. However, a masked exception will not set the exception status bit (ES).

Operation Sequencing

The Am29027 can be configured for either pipelined or flow-through (unpipelined) operation. Flow-through mode is normally selected for performing scalar operations; pipeline mode provides high throughput for vector operations. The manner in which operations are sequenced depends on the mode currently invoked.

Operation in Flow-Through Mode

Flow-through mode is invoked by setting mode register bit PL (Pipeline Mode Select) to logic Low.

Programmer's Model

A programmer's model of the Am29027 in flow-through mode is shown in Figure 9. Note that Output Register F and the flag register are made transparent in this mode.

Performing Operations

Flow-through mode operations are performed by:

- Storing instructions and/or operands in the Am29027 and starting the operation
- Loading the result

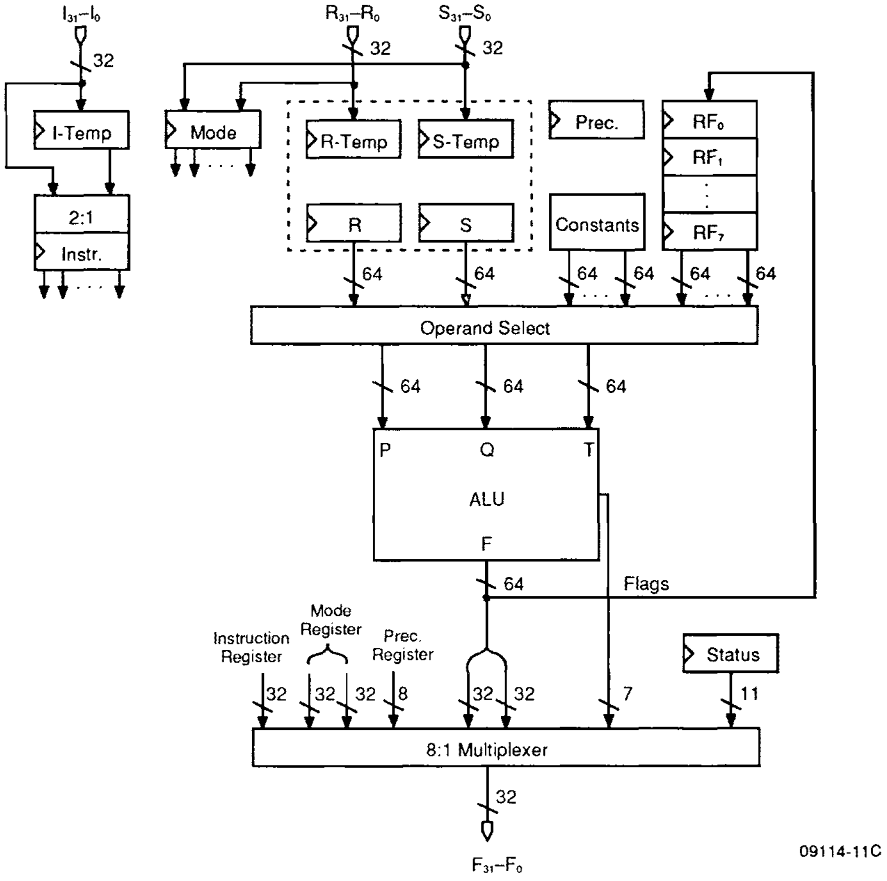


Figure 15. Programmer's Model for Flow-Through Mode

Storing instructions and operands can be done in any of three ways:

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

Operands and instructions are written using the write operand R, write operand S, write operands R, S, and write instruction transaction requests. Operands and instructions can be written to the Am29027 in any order, with the operation start bit ($DREQT_0$ High) accompanying the last of the transaction requests.

Loading an operation result is performed using the read result MSBs, read result LSBs, and read flags 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 operation result is stored in the register file, the user may elect not to read the result but to proceed with the next operation.

Operation Timing

The Am29027 will usually start a flow-through operation during the first cycle following the receipt of a write operand R, write operand S, write operands R, S, or write instruction transaction request having signal $DREQT_0$ set High.

Operation execution begins with the transfer of the contents of the R-Temp, S-Temp, and I-Temp registers to Register R, Register S, and the instruction register, respectively; only those temporary registers written to as part of the operation specification will be transferred. The operand or instruction accompanying the transaction request that starts the operation (that is, the transaction request for which signal $DREQT_0$ is High) is written directly to the appropriate working register, that is, Register R, Register S, or the instruction register.

Once started, an operation will proceed for the number of cycles specified by mode register fields MATC, MVTC, and PLTC; MATC specifies the number of cycles for base operation code $(P \times Q) + T$, MVTC the number of cycles for base operation code MOVE P, and PLTC the number of cycles for all other base operation codes. At the end of the last operation cycle, the status register exception bits and exception status bit will be updated and, optionally, the operation result will be written to the register file and precision register.

There are two conditions for which the Am29027 will not start an operation immediately. The first condition is when an operation is already in progress. In this case the new operation is kept pending in the I-Temp, R-Temp, and S-Temp registers until the current operation is completed, at which time the new operation begins. The second condition is when a previous operation creates an unmasked exception in Halt On Error mode (mode register bit HE High). In this case the new operation is kept in the I-Temp, R-Temp, and S-Temp registers until the exception is cleared, at which time the new operation begins.

Timing for typical accelerator operations in the flow-through mode is illustrated in Appendix D.

Availability of Operation Results

In order to directly read the result of an operation, the operation specification should be followed by the appropriate read transaction request. Should the Am29000 attempt to read an operation result before the operation is completed, the Am29027 will withhold acknowledging the transaction request by holding signals $\overline{DRDY}$ and $\overline{DERR}$ inactive until the operation has been completed. All read transaction requests, including save state, will be held off in this manner.

Overlapping Operations

Due to the presence of the R-Temp, S-Temp, and I-Temp registers, it is possible to partially or completely specify a new operation while the previously specified operation is being performed. Execution of the new operation will begin immediately after the previous operation is completed. Execution begins with the transfer of the contents of the R-Temp, S-Temp, and I-Temp registers to the corresponding working registers; only those temporary registers that have been written to as part of the operation specification are transferred.

It is important to note that, once the new operation is completely specified, any attempt to read a result will be held off until the new operation is completed. This means that it is not possible to directly read the result of an operation if another operation is completely specified before the results of the first operation are read. If, for example, specification of operation $2.0 + 3.0$ is immediately followed by specification of operation 4.0×5.0, subsequent read result LSBs and read result MSBs transaction requests will return value 20.0, the result of the second operation. Similarly, a read flags transaction request will return flags for the second operation, and a read status transaction request will return status reflecting the completion of the second operation. This delayed read feature is provided to eliminate ambiguity in the correspondence between operations and results.

Should two operations be overlapped, and should the first operation have as its target a register file location, the second operation can be completely specified before the first operation is completed. If the first operation produces a result that is to be read directly by the Am29000, the second operation can be partially specified before the result of the first operation is read. A partial operation specification is one that includes all but the last operand or instruction.

Timing for typical overlapped operations in flow-through mode is illustrated in Appendix D.

Saving and Restoring State

In flow-through mode, the complete state of the Am29027 can be saved and restored with the save state transaction request. The first save state transaction request will return the contents of the instruction register; subsequent requests will return the contents of Registers I-Temp, R, S, R-Temp, S-Temp, the status register, the precision register, register file locations RF_7–RF_0, and the mode register. The user has the option of saving only part of the state by issuing only the number of save state transaction requests needed to save registers of interest. When issuing a series of save state transaction requests, data is returned in the following order:

Request	Data Returned
1	Instruction
2	I-Temp
3	R LSBs
4	R MSBs
5	S LSBs
6	S MSBs
7	R-Temp LSBs
8	R-Temp MSBs
9	S-Temp LSBs
10	S-Temp MSBs
11	Status
12	Precision
13	RF_0 LSBs
14	RF_0 MSBs
.	.
.	.
.	.
27	RF_7 LSBs
28	RF_7 MSBs
29	Mode LSBs
30	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 after a series of save state requests, the next save state request will return the contents of the instruction register.

It should be noted that the process of saving state alters the contents of the instruction register and Registers R and S.

Error reporting via signal $\overline{\text{DERR}}$ is suppressed for the save state transaction request.

Accelerator state is restored using transaction requests in concert with the MOVE P base operation code. Before restoring state, all status register bits should be set to logic Low using the write status transaction request to prevent the possibility of an unmasked exception bit inhibiting the restore sequence. The accelerator operand and instruction registers can then be restored, followed by restoration of the status register using the write status transaction request, with signal $DREQT_0$ asserted to indicate the end of the restore sequence. When state restoration is complete, the Am29027 will retime the operation specified by current instruction register contents.

Accelerator state is restored in the following order:

Register to be restored	Procedure for restoring
Status	Set all bits in the status register to a logic Low using the write status transaction request.
Mode	Write using write mode transaction request.
RF_0	Write "Move R to RF_0" instruction using write instruction transaction request.
	Write RF_0 value to Register R using write operand R transaction request, start operation.
	.
	.
	.
RF_7	Write "Move R to RF_7" instruction using write instruction transaction request.
	Write RF_7 value to Register R using write operand R transaction request, start operation.
Precision	Guarantee that "Move R to RF_7" operation has been completed by performing a read result MSBs transaction request.
	Write precisions using write register file precisions transaction request.
R, S, Instruction	Write R value to Register R-Temp using the write operand R transaction request.
	Write S value to Register S-Temp using the write operand S transaction request.
	Write instruction value to Register I-Temp using write instruction transaction request.
	Transfer contents of Registers R-Temp, S-Temp, and I-Temp to Register R, Register S, and the instruction register, respectively, using the advance temp registers transaction request.
R-Temp, S-Temp, I-Temp	Write R-Temp value to Register R-Temp using the write operand R transaction request.
	Write S-Temp value to Register S-Temp using the write operand S transaction request.
	Write I-Temp value to Register I-Temp using the write instruction transaction request.
Status	Write status to status register using the write status transaction request, with signal $DREQT_0$ asserted to indicate that the restore sequence is complete.

The user may elect to restore only those registers relevant to a particular application by omitting parts of the state restoration sequence. The only mandatory portions of state restoration are the initial clearing of the status register, and restoration of the status register with signal $DREQT_0$ asserted to indicate completion of the restore sequence.

Error Recovery

Six exception bits—invalid operation, reserved operand, overflow, underflow, inexact result, and zero result—are maintained in the status register; these bits are updated upon completion of an operation. Exception bits can be masked individually by programming the appropriate bits in the mode register; if the corresponding mask bit is inactive (logic Low), the exception bit is said to be unmasked and contributes to error reporting. The Am29027 provides three mechanisms with which unmasked exceptions can be handled.

Reporting Errors Upon Read

If an unmasked status register exception bit is set, the Am29027 will signal an error by asserting signal $\overline{DERR}$ when the Am29000 performs a read result LSBs, read result MSBs, read flags, or read status transaction request. Error reporting can be suppressed by issuing any of these transaction requests with signal $DREQT_0$ asserted.

Halt On Error Mode

Should the application require, the Am29027 can be configured to halt operation upon detection of an unmasked exception; this mode is invoked by setting mode register bit HE (Halt On Error) High. Once configured this way, the Am29027 will respond to an unmasked exception as follows:

- Signal $\overline{CDA}$ will become inactive upon completion of the operation producing the unmasked exception.
- Should the operation producing the unmasked exception specify that the operation result be stored on-chip, that is, in the register file, the result will not be written to its destination.
- A pending operation will not be started; the operands and/or instruction for that operation will remain in the appropriate temporary registers.
- If the Am29000 attempts to start a new operation during the last cycle of the operation that produces the unmasked exception by issuing a write operand R, write operand S, write operands R, S, or write instruction transaction request with $DREQT_0$ asserted, and if no other operation is pending, the operand or instruction will be written to the appropriate temporary register rather than to the R, S, or instruction register.
- Once $\overline{CDA}$ is deasserted, the Am29027 will respond to the write operand R, write operand S, write operands R, S, and write instruction transaction requests by asserting signal $\overline{DERR}$ one cycle after the request is issued; the contents of the target register or registers will remain unchanged.

Through these measures, the Am29027 will retain the input operands and instructions for the operation causing the exception. The input operands will be retained in the R register, S register, or register file locations, and the instructions will be retained in the instruction register. Additionally, the R-Temp, S-Temp, and I-Temp registers may contain the operands and instructions for a partially or fully specified pending operation. The Am29000 can recover these operands and instructions with the save state transaction request; this information can then be given to an error-handling routine for resolution.

The error halt condition is removed by clearing the status register exception status (ES) bit and the exception bit or bits responsible for producing the halt.

Reporting Errors via $\overline{\text{EXCP}}$

Signal $\overline{\text{EXCP}}$ will go active Low in the presence of an unmasked exception. This signal can be connected to an Am29000 trap or exception input signal, and is enabled or disabled independent of other exception handling mechanisms with mode register bit EX.

Writing to the Mode, Status, and Precision Registers

Unlike the R, S, and instruction registers, the mode, status, and precision registers are not preceded by temporary registers. Accordingly, writing to these registers may produce undesirable or unpredictable side effects if an accelerator operation is in progress at the time. To avoid such side effects, a write to any of these registers should be preceded by a read transaction request, which will guarantee that any current or pending accelerator operations will have been completed before the write transaction request is issued.

Writing to the Register File

The numerical result of any operation may be written to the register file by specifying the desired destination in instruction field RFS and setting instruction bit RF High. The result can then be used as an input operand for subsequent operations.

It is permissible for an operation result to be placed in a register file location that previously contained an input operand for that operation. In such a case, however, it is not permissible for the Am29000 to directly read the result, status, or flags for that operation, as the writing of the result modifies the operation performed by the ALU.

Determining Timer Counts

To provide optimum accelerator performance over a range of possible system clock frequencies, the timing of Am29027 operations is programmable. Three mode register fields—pipeline timer count (PLTC), timer count for the Multiply-Accumulate Operation (MATC), and timer count for the MOVE P Operation (MVTC)—must be programmed according to system clock frequency and accelerator speed.

PLTC

PLTC specifies the number of cycles allotted to operations other than those using base operation codes $(P \times Q) + T$ or MOVE P. This count can assume values between 3 and 15, inclusive, and must be given a value that satisfies the relationship:

$$[8] \leq \text{PLTC} \times [1],$$

where

[8] = Operation time, flow-through mode, all other base operation codes

and [1] = CLK period,

as described in the Switching Characteristics table.

MATC

MATC specifies the number of cycles allotted to operations that use base operation code $F' = (P' \times Q') + T'$. This count can assume values between 3 and 15, inclusive, and must be given a value that satisfies the relationship:

$$[6] \leq \text{MATC} \times [1],$$

where

[6] = Operation time, flow-through mode, $F' = (P' \times Q') + T'$

and [1] = CLK period,

as described in the Switching Characteristics table.

MVTC

MVTC specifies the number of cycles allotted to operations that use the MOVE P base operation code. This count can assume values between 3 and 15, inclusive, and must be given a value that satisfies the relationship:

$$[7] \leq \text{MVTC} \times [1],$$

where

[7] = Operation time, flow-through mode, MOVE P

and [1] = CLK period,

as described in the Switching Characteristics table.

ADVANCING $\overline{\text{DRDY}}$

Normally, an operation result produced by the Am29027 in flow-through mode is read by the Am29000 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 45-ns clock cycle and an Am29027 that performs an operation in 240 ns. The pipeline timer count PLTC will have to be set to a minimum of 6 for such a system, and the Am29000 will read a result no sooner than during the seventh clock cycle after the start of an operation.

Mode register bit DA, $\overline{\text{DRDY}}$ Advance, can be used to advance transaction status signals $\overline{\text{DRDY}}$ and $\overline{\text{DERR}}$ by a full clock cycle, thus allowing the Am29000 to read data one clock cycle earlier than would otherwise be possible. For the example given above PLTC remains at 6, but the Am29000 can read data during the sixth clock cycle after the operation starts rather than the seventh, thus saving a clock cycle.

In order to advance $\overline{\text{DRDY}}$ and $\overline{\text{DERR}}$, the following system timing conditions must be met:

$$[19] \le (\text{MATC} \times [1]) - [\times 9B] - [\text{gate}]$$
$$[20] \le (\text{MVTC} \times [1]) - [\times 9B] - [\text{gate}]$$
$$[21] \le (\text{PLTC} \times [1]) - [\times 9B] - [\text{gate}]$$

where [19] = Data operation-start-to-output valid delay, $F' = P' \times Q' + T'$
[20] = Data operation-start-to-output valid delay, MOVE P
[21] = Data operation-start-to-output valid delay, all other operations
and [1] = CLK period

as described in the Switching Characteristics table and

[x9] = Synchronous input setup time

as described in the Switching Characteristics table of the Am29000 Preliminary Data Sheet (order #09075).

The term [gate] represents the delay of the external gate through which the $\overline{\text{DERR}}$ signal passes.

Timing for a typical accelerator operation with $\overline{\text{DRDY}}$ advanced is illustrated in Appendix D.

Operation In Pipeline Mode

Pipeline mode is invoked by setting mode register bit PL (Pipeline Mode Select) to logic High.

Programmer's Model

A programmer's model of the Am29027 in pipeline mode is shown in Figure 10. Note that Output Register F and the flag register are non-transparent in this mode, thus permitting the overlap of the current operation(s) with the reading of the result for a previous operation.

Pipeline Delays

When placed in pipeline mode, the ALU is divided into three pipeline stages for multiply-accumulate operations, and into two stages for all other operations. The ALU configuration for pipeline mode is shown in Figure 11. Note that for multiplication-accumulation operations, multiplicand P and multiplier Q enter the first pipeline stage, while addend T enters the second pipeline stage. As a consequence, the source for operands P and Q must be specified in the corresponding multiply-accumulate instruction, while the source for operand T must be specified in the following instruction.

Pipeline Advance

The ALU pipeline is advanced whenever a new operation begins. One consequence of this advance criterion is that data does not fall through the pipe but instead is "pushed" through. If, for example, an addition is performed in pipeline mode, the pipe must be advanced twice (by starting two operations) before the result of the addition appears in Register F, the flag register, the status register, and, optionally, a register file location.

Performing Operations

Pipeline mode operations are performed by:

- Storing instructions and/or operands in the Am29027, and starting the operation
- Loading the result of a previous operation

Storing instructions and operands can be done in any of three ways:

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

Operands and instructions are written using the write operand R, write operand S, write operands R, S, and write instruction transaction requests. Operands and instructions can be written to the Am29027 in any order, with the operation start bit (DREQT_0 High) accompanying the last of the transaction requests.

Loading the result of a previous operation is performed using the read result MSBs, read result LSBs, and read flags transaction requests. The specific request used depends on whether the result 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 operation result is stored in the register file, the user may elect not to read the result, but to proceed with the next operation.

Operation Timing

The Am29027 will usually start a pipelined operation during the first cycle following the receipt of a write operand R, write operand S, write operands R, S, or write instruction transaction request having signal DREQT_0 set High.

Operation execution begins with the transfer of the contents of the R-Temp, S-Temp, and I-Temp registers to Register R, Register S, and the instruction register, respectively; data is transferred only from those temporary registers written to as part of the operation specification. The operand or instruction accompanying the

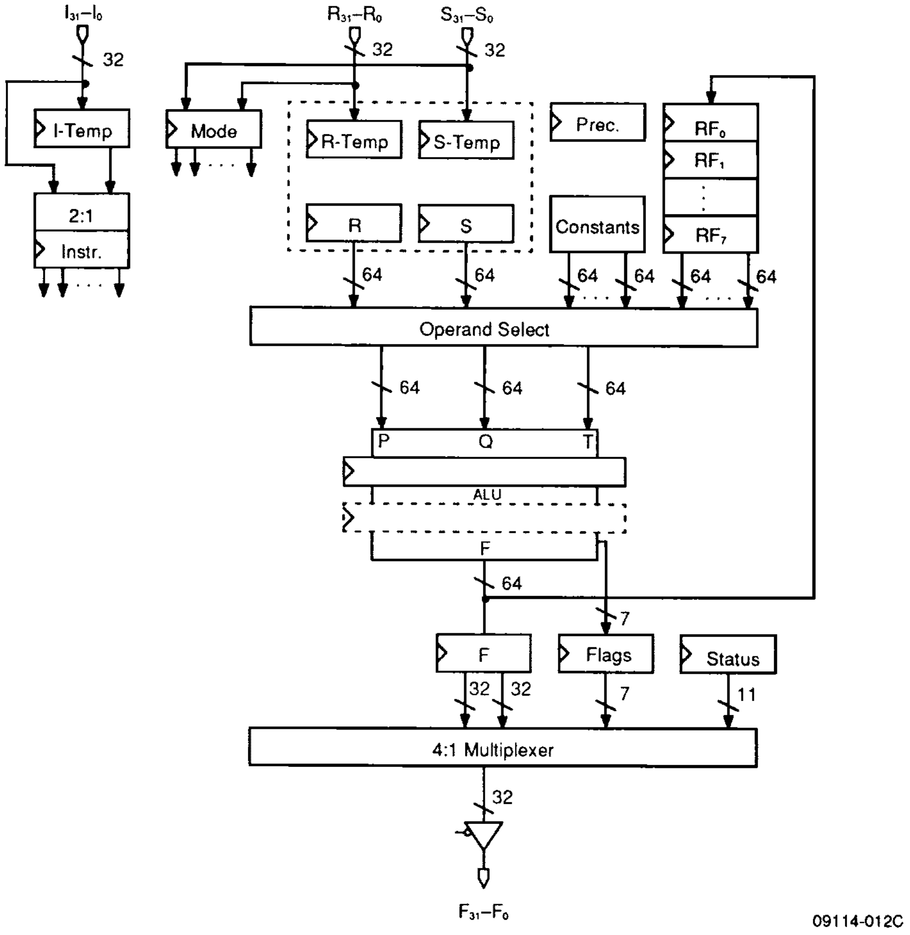
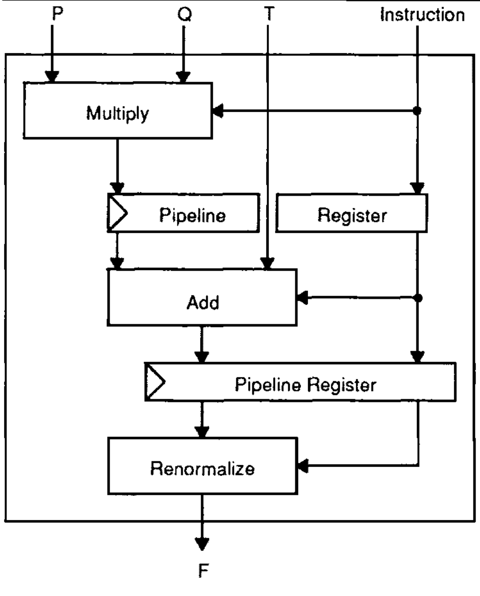


Figure 16. Programmer's Model for Pipeline Mode

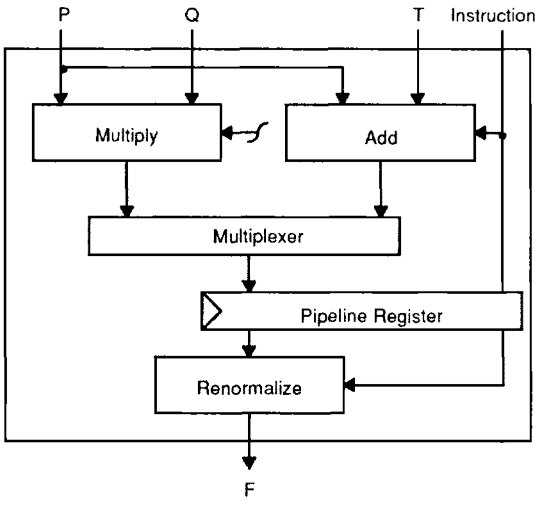
transaction request that starts the operation (that is, the transaction request for which signal $DREQT_0$ is High) is written directly to the appropriate working register, that is, Register R, Register S, or the instruction register. At the start of the operation, the output of the last ALU pipeline stage is transferred to Register F, the flag register, and, optionally, to a register file location; the status register exception status and exception bits are updated. The outputs of all other ALU pipeline stages are written to their respective pipeline registers.

Once started, an operation will proceed for the number of cycles specified by mode register field PLTC, which denotes the number of cycles needed for data to traverse a single pipeline stage.

There are two conditions for which the Am29027 will not start an operation immediately. The first condition is when an operation has been started recently and has not yet had time to settle at the output of the first pipeline stage. In this case the new operation is kept pending in the I-Temp, R-Temp, and S-Temp registers until the previous operation completes the first pipeline stage. The second condition is when a previous operation creates an unmasked exception in Halt On Error mode (mode register bit HE High). In this case the new operation is kept in the I-Temp, R-Temp, and S-Temp registers until the exception is cleared, at which time the new operation will begin.



a. Multiply-Accumulate



b. Other Operations

09114-013C

Figure 17. ALU Configuration for Pipeline Mode

Timing for typical accelerator operations in the pipeline mode is illustrated in Appendix D.

Availability of Operation Results

Because Register F, the flag register, and the status register are updated at the beginning of an operation, these registers can be read at any time after an operation begins.

Overlapping Operations

Due to the presence of the R-Temp, S-Temp, and I-Temp registers, it is possible to partially or completely specify a new operation while the previously specified operation is propagating through the first ALU pipeline stage. Execution of the new operation will begin immediately after the previous operation completes the first pipeline stage. Execution begins with the transfer of the contents of the R-Temp, S-Temp, and I-Temp registers to the corresponding working registers; only those temporary registers that have been written to as part of operation specification are transferred.

It is important to note that, once the new operation is completely specified, any attempt to read a result will be held off until the new operation begins; this means that it is not possible to read the result that is placed in the output registers when the first operation begins. If, for example, result X is placed in Register F when an operation starts and if another operation is completely specified thereafter, subsequent read result MSBs and read result LSBs transaction requests will return not X, but the result placed in the F register when the second operation begins; the read flags and read status transaction requests will behave in like manner. This delayed read feature is provided to eliminate ambiguity in the correspondence between operations and results.

Saving and Restoring State

Due to the presence of ALU pipeline registers, it is not possible to save the complete state of the Am29027 in pipeline mode. Pipeline operations may therefore be interrupted only under special circumstances, such as:

- If the interrupting routine does not use the floating-point accelerator

or

- If the current series of pipelined operations has been completed, and any operands needed for future operations have already been transferred to the Am29000

The save state transaction request is disabled in pipeline mode. It is permissible to switch to flow-through mode and use the save state transaction request, but doing so does not permit the saving of Register F, the flag register, or the ALU pipeline registers.

Error Recovery

As for flow-through mode, the Am29027 provides three mechanisms with which unmasked exceptions can be handled.

Reporting Errors Upon Read

If an unmasked status register exception bit is set, the Am29027 will signal an error by asserting signal $\overline{\text{DERR}}$ when the Am29000 performs a read result LSBs, read result MSBs, read flags, or read status transaction request. Error reporting can be suppressed by issuing any of these transaction requests with signal DREQT$_0$ asserted.

Halt On Error Mode

Should the application require it, the Am29027 can be configured to halt operation upon detection of an unmasked exception; this mode is invoked by setting mode register bit HE (Halt On Error) High. Once configured this way, the Am29027 will respond to an unmasked exception as follows:

- Signal $\overline{\text{CDA}}$ will become inactive when the results of the operation producing the unmasked exception are transferred from the last pipeline stage to Register F, the flag register, and the status register.
- Once $\overline{\text{CDA}}$ is deasserted, the Am29027 will respond to the write operand R, write operand S, write operands R, S, and write instruction transaction requests by asserting signal $\overline{\text{DERR}}$ one cycle after the request is issued; the contents of the target register or registers will remain unchanged.

Through these measures, the Am29027 will retain the input operands and instructions for the most recently started operation. The input operands for that operation will be retained in the R register, S register, or register file locations, and the instructions will be retained in the instruction register. Additionally, the R-Temp, S-Temp, and I-Temp registers may contain the operands and instructions for a partially or fully specified pending operation. Note that the input operands and instructions words for the operation causing the exception, as well as for operations currently in the ALU pipeline, will not be available. At the user's option, this information can be stored in a circular queue in the Am29000 register file so that full recovery from a pipelined exception is possible.

The Am29000 can read the contents of Am29027 operand and instruction registers by invoking flow-through mode and using the save state transaction request. Note that the contents of Register F, the flag register, and the ALU pipeline registers will be lost. This information can then be given to an error-handling routine for resolution.

The error halt condition is removed by clearing the status register exception status (ES) bit and the exception bit or bits responsible for producing the halt.

Reporting Errors via $\overline{\text{EXCP}}$

Same as for the flow-through mode.

Pipeline Invalidation

There are several situations for which the ALU pipeline stages may contain invalid data. The Am29027 recognizes these situations and invalidates results automatically; results marked as invalid will not update the status register, register file locations RF$_7$–RF$_0$, or the precision register. Results are invalidated for the following conditions:

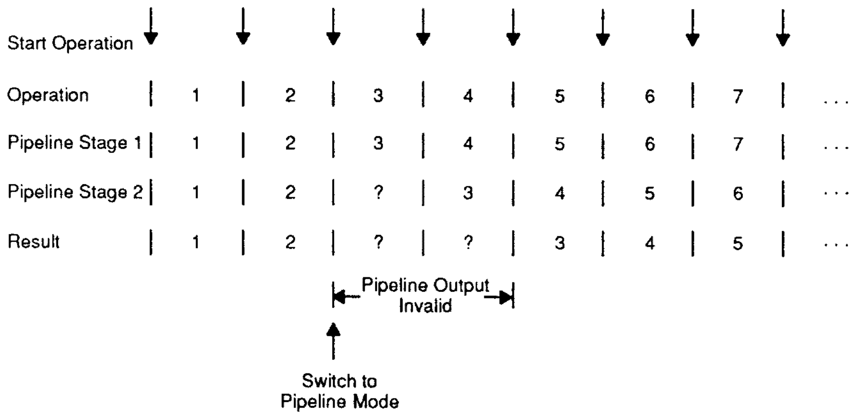
- The Am29027 is switched from flow-through mode to pipeline mode. Any data present in the ALU at the time of the switch is marked as invalid. This invalidation is illustrated in Figure 12a.
- The Am29027 performs a multiply-accumulate operation that is preceded by an operation other than multiply-accumulate. The multiply-accumulate operation result and the result that precedes it will be separated by a spurious result, due to the insertion of an additional pipeline stage for the multiply-accumulate operation. The spurious result is marked invalid. This invalidation is illustrated in Figure 12b.

The pipeline may also be invalidated manually by issuing a write status transaction request with signal DREQT$_0$ asserted High; this request invalidates all current pipeline contents. Pipeline invalidation does not apply to operation in flow-through mode.

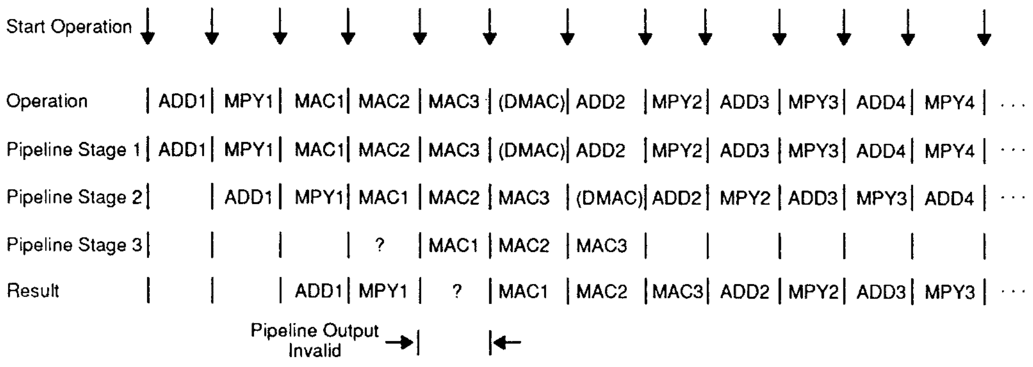
Writing to the Mode, Status, and Precision Registers

Unlike the R, S, and instruction registers, the mode, status, and precision registers are not preceded by temporary registers. Accordingly, writing to these registers may produce undesirable or unpredictable side effects if an accelerator operation is pending at the time. To avoid such side effects, a write to any of these registers should be preceded by a read transaction request, which will guarantee that any pending accelerator operation will have started before the write transaction request is issued.

The mode register outputs are not pipelined in the ALU, that is, all pipeline stages receive mode information directly from the mode register. Accordingly, writing to the mode register may produce undesirable or unpredictable side effects for operations currently in the ALU pipeline. To avoid such side effects, a write to the mode register should be performed only if the contents of the ALU pipeline are a "don't care," that is, only after the last operation result of interest has been written to Register F, the flag register, or a register file location. If, for exam-


Start Operation
Operation | 1 | 2 | 3 | 4 | 5 | 6 | 7 | . . .
Pipeline Stage 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | . . .
Pipeline Stage 2 | 1 | 2 | ? | 3 | 4 | 5 | 6 | . . .
Result | 1 | 2 | ? | ? | 3 | 4 | 5 | . . .
Pipeline Output Invalid
Switch to Pipeline Mode

a. Pipeline invalidation timing for switch from flow-through to pipeline mode. Operations shown incur two pipe-line delays in pipeline mode [all base operations except F′ = (P′ × Q′) + T′].


Start Operation
Operation | ADD1 | MPY1 | MAC1 | MAC2 | MAC3 | (DMAC) | ADD2 | MPY2 | ADD3 | MPY3 | ADD4 | MPY4 | . . .
Pipeline Stage 1 | ADD1 | MPY1 | MAC1 | MAC2 | MAC3 | (DMAC) | ADD2 | MPY2 | ADD3 | MPY3 | ADD4 | MPY4 | . . .
Pipeline Stage 2 | | ADD1 | MPY1 | MAC1 | MAC2 | MAC3 | (DMAC) | ADD2 | MPY2 | ADD3 | MPY3 | ADD4 | . . .
Pipeline Stage 3 | | | | ? | MAC1 | MAC2 | MAC3 | | | | | |
Result | | | ADD1 | MPY1 | ? | MAC1 | MAC2 | MAC3 | ADD2 | MPY2 | ADD3 | MPY3 | . . .
Pipeline Output Invalid

b. Pipeline invalidation timing for multiply-accumulate operations in pipeline mode.

Notes:		
ADDx	=	addition operation
MPYx	=	multiplication operation
MACx	=	multiply-accumulate operation
(DMAC)	=	dummy multiply-accumulate operation

09114-014C

Figure 18. Pipeline Invalidation Timing

ple, the last in a series of addition operations has just been started, the mode register should not be written until the pipeline is advanced twice, placing that operation's results in the F register, flag register, and, optionally, a register file location.

Writing to the Register File

The numerical result of any operation may be written to the register file by specifying the desired destination in instruction field RFS and setting instruction bit RF High. The result may then be used as an input operand in subsequent operations. Because all ALU operations incur one or more pipeline delays, the result of an operation will not be available for use by the very next operation.

It is permissible for an operation result to be placed in a register file location that previously contained an input operand for that operation.

Multiplication-Accumulation Operations

The pipeline structure of the Am29027 permits the evaluation of sum-of-products expressions in a canonically efficient manner by interleaving the evaluation of two sum-of-product expressions. Operation sequencing is described in Figure 13.

Determining Timer Counts

As for flow-through mode, the timing of operations in pipeline mode is programmable to accommodate variations in system timing. A single mode register field—pipeline timer count (PLTC)—specifies the timing of all pipelined operations; fields MATC and MVTC are not used.

PLTC specifies the number of cycles allotted for data to traverse a single pipeline stage. This count can assume values between 2 and 15, inclusive, and must be given a value that satisfies the relationship:

$$[9] \leq \text{PLTC} \times [1],$$

where

[9] = Operation time, pipeline mode, all operations

and [1] = CLK period,

as described in the Switching Characteristics table.

Advancing $\overline{\text{DRDY}}$

Because the Am29027 F register and flag register are non-transparent in pipeline mode, it is not possible (nor advantageous) to advance $\overline{\text{DRDY}}$. Accordingly, mode register bit M44 has no effect in pipeline mode.

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 $\overline{\text{SLAVE}}$ and MSERR, are connected directly to the corresponding signals of the master. The master is selected by asserting signal $\overline{\text{SLAVE}}$ Low, the slave by asserting signal $\overline{\text{SLAVE}}$ High.

The slave accelerator, by comparing its outputs to the outputs of the master accelerator, performs a comprehensive check of master accelerator logic. In addition, if the slave accelerator is connected at the proper position on the Am29000 buses, it may detect open circuits and other faults in the electrical path between the master accelerator and the Am29000.

Note that the master accelerator also performs a comparison between its outputs and its own internally generated results, and is therefore able to detect faults in its output drivers, which it reports with its MSERR signal.

Initialization and Reset

The accelerator is in an unknown state when power is first applied and must be initialized before processing can begin. This is accomplished by asserting the $\overline{\text{RESET}}$ signal, which initializes accelerator state as follows:

- All bits in the status register are cleared
- The accelerator is placed in flow-through mode
- Signal $\overline{\text{CDA}}$ is active; signals $\overline{\text{DRDY}}$ and $\overline{\text{DERR}}$ are inactive
- All internal circuitry controlling operation timing is initialized

The $\overline{\text{RESET}}$ signal does not initialize the operand and instruction registers and may corrupt existing register contents. It is the responsibility of the user to initialize these registers, if needed.

Applications

Suggestions for Power and Ground Pin Connections

The Am29027 operates in an environment of fast signal rise times and substantial switching currents. Therefore, care must be exercised during circuit board design and layout, as with any high-performance component. The following is a suggested layout, but since systems vary widely in electrical configuration, an empirical evaluation of the intended layout is recommended.

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

Printed Circuit-Board Layout Suggestions

1. Use of a multilayer PC board with separate power, ground, and signal planes is highly recommended.
2. All V_{CC} and V_{CCO} pins should be connected to the V_{CC} plane. V_{CCO} pins should be isolated from V_{CC} pins by means of an isolation slot which is cut in the V_{CC} plane (see Figure 14). By physically separating the V_{CC} and V_{CCO} pins, coupled noise will be reduced.
3. All GND and GNDO pins should be connected directly to the ground plane.
4. The V_{CCO} pins should be decoupled to ground with a 0.1-µF ceramic capacitor and a 10-µF electrolytic capacitor, placed as closely to the Am29027 as is practical. V_{CC} pins should be decoupled to ground in a similar manner.

A suggested layout is shown in Figure 14.

Operation	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC	MAC*		
Register R	a11	a21	a12	a22	a13	a23	a14	a24	a31	a41	a32	a42	a33	a43	a34	a44			
Register S	b1	b1	b2	b2	b3	b3	b4	b4	b1	b1	b2	b2	b3	b3	b4	b4			
Pipeline Stage 1	a11 x b1	a21 x b1	a12 x b2	a22 x b2	a13 x b3	a23 x b3	a14 x b4	a24 x b4	a31 x b1	a41 x b1	a32 x b2	a42 x b2	a33 x b3	a43 x b3	a34 x b4	a44 x b4			
Pipeline Stage 2		a11 x b1	a21 x b1	a12 x b2 + (c1)	a22 x b2 + (c2)	a13 x b3 + (c1)	a23 x b3 + (c2)	a14 x b4 + (c1)	a24 x b4 + (c2)	a31 x b1	a41 x b1	a32 x b2 + (c3)	a42 x b2 + (c4)	a33 x b3 + (c3)	a43 x b3 + (c4)	a34 x b4 + (c3)	a44 x b4 + (c4)		
Pipeline Stage 3			a11 x b1	a21 x b1	a12 x b2 + (c1)	a22 x b2 + (c2)	a13 x b3 + (c1)	a23 x b3 + (c2)	a14 x b4 + (c1)	a24 x b4 + (c2)	a31 x b1	a41 x b1	a32 x b2 + (c3)	a42 x b2 + (c4)	a33 x b3 + (c3)	a43 x b3 + (c4)	a34 x b4 + (c3)	a44 x b4 + (c4)	
RF_0				(c1)	(c2)	(c1)	(c2)	(c1)	(c2)	c1	c2	(c3)	(c4)	(c3)	(c4)	(c3)	(c4)	c3	c4
Register F				(c1)	(c2)	(c1)	(c2)	(c1)	(c2)	c1	c2	(c3)	(c4)	(c3)	(c4)	(c3)	(c4)	c3	c4

Calculate matrix product C = A x B, where:

$$A = \begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix} \quad B = \begin{bmatrix} b1 \\ b2 \\ b3 \\ b4 \end{bmatrix} \quad C = \begin{bmatrix} c1 \\ c2 \\ c3 \\ c4 \end{bmatrix}$$

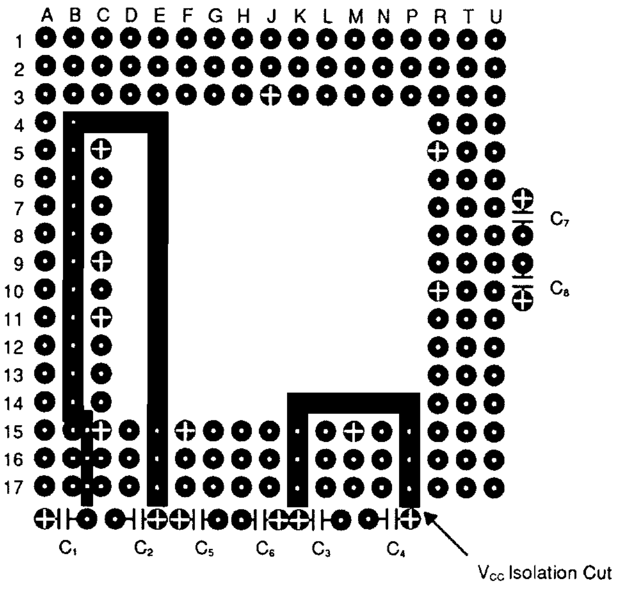
c1 = a11 x b1 + a12 x b2 + a13 x b3 + a14 x b4
c2 = a21 x b1 + a22 x b2 + a23 x b3 + a24 x b4
c3 = a31 x b1 + a32 x b2 + a33 x b3 + a34 x b4
c4 = a41 x b1 + a42 x b2 + a43 x b3 + a44 x b4

09114-015C

Notes:
1. Register file location RF_0 is used as the accumulator.
2. Parentheses are used to indicate partial sums of products.

*Additional MAC operation needed to terminate sequence.

Figure 13. Canonically Efficient Sum-of-Products Evaluation in Pipeline Mode



● = Through Hole

⊕ = V_{CC} Plane Connection

CD011711

$C_1 = C_3 = C_5 = C_7 = 0.1\ \mu F$ (ceramic or monolithic capacitor)

$C_2 = C_4 = C_6 = C_8 = 10\ \mu F$ (electrolytic or tantalum capacitor)

Figure 20. Suggested Printed Circuit-Board Layout (power and ground connections)

ABSOLUTE MAXIMUM RATINGS

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

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

OPERATING RANGES

Commercial (C) Devices

Case Temperature (T_C)	0 to +85°C
Supply Voltage (V_{CC})	+4.75 V to +5.25 V

Military* (M) Devices

Case Temperature (T_C)	−55 to +125°C
Supply Voltage (V_{CC})	+4.5 V to +5.5 V

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

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

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

Parameter Symbol	Parameter Description	Test Conditions (Note 1)			Min.	Max.	Unit
V_{OH}	Output High Voltage	V_{CC} = Min. $V_{IN} = V_{IH}$ or V_{IL}		I_{OH} = −4.0 mA	2.4		V
V_{OL}	Output Low Voltage	V_{CC} = Min. $V_{IN} = V_{IH}$ or V_{IL}		I_{OL} = 4.0 mA		0.45	V
V_{IH}	Guaranteed Input Logical High Voltage (Note 2)				2.0		V
V_{IL}	Guaranteed Input Logical Low Voltage (Note 2)					0.8	V
V_{IH}(F)	Guaranteed Input Logical High Voltage (Notes 2, 6)	F Bus, Slave Operation Only			V_{CC} −0.5		V
V_{IL}(F)	Guaranteed Input Logical Low Voltage (Notes 2, 6)	F Bus, Slave Operation Only				0.5	V
I_{IL}	Input Leakage Current	$0.450 \le V_{IN} \le V_{CC} -0.450$, V_{CC} = Max.				±10	µA
I_{LO}	Output Leakage Current	$0.450 \le V_{OUT} \le V_{CC} -0.450$, V_{CC} = Max.				±10	µA
I_{CC} Static	Static Power Supply Current	V_{CC} = Max. I_O = 0 µA	Com'l T_C = 0 to +85°C	(Note 3) CMOS $V_{IN} = V_{CC}$ or GND		240	mA
				(Note 3) TTL V_{IN} = 0.5 V or 2.4 V		275	
			MIL T_C = −55 to +125°C	(Note 3) CMOS $V_{IN} = V_{CC}$ or GND			
				(Note 3) TTL V_{IN} = 0.5 V or 2.4 V			
I_{CCOP}	Operating Power Supply Current	V_{CC} = Max. Outputs floating				9.0	mA/MHz

Notes: 1. V_{CC} conditions shown as Min. or Max. refer to ±5% V_{CC} (commercial) and ±10% V_{CC} (military).
2. These input levels provide zero noise immunity and should only be statically tested in a noise-free environment (not functionally tested).
3. Use CMOS I_{CC} when the device is driven by CMOS circuits and TTL I_{CC} when the device is driven by TTL circuits.
4. I_{CC} (Total) = I_{CC} (Static) + $I_{CCOP} \times f$, where f is in MHz. This is tested on a sample basis only.
5. Tested on a sample basis only.
6. These levels guaranteed compatible with F bus output levels.

CAPACITANCE

Parameter Symbol	Parameter Description	Test Conditions	Min.	Max.	Unit
C_{IN}	Input Capacitance	fc = 1 MHz (Note 5)		12	pF
C_{OUT}	Output Capacitance			20	pF
$C_{I/O}$	I/O Pin Capacitance			20	pF

SWITCHING CHARACTERISTICS over COMMERCIAL operating range

No.	Parameter Description	Test Conditions	25 MHz		20 MHz		16 MHz		Unit
			Min.	Max.	Min.	Max.	Min.	Max.	
1	CLK Period	(Note 1)	40	DC	50	DC	60	DC	ns
2	CLK Low Time		18		20		22		ns
3	CLK High Time		18		20		22		ns
4	CLK Rise Time	(Note 2)		5		5		5	ns
5	CLK Fall Time	(Note 2)		5		5		5	ns
6	Operation Time, Low-Latency Mode, $F' = (P' \times Q') + T'$			280		300		360	ns
7	MOVE P			120		150		180	ns
8	(All Other Base Operation Codes)			200		250		300	ns
9	Operation Time, Pipeline Mode All Operations			150		150		180	ns
10	Transaction Request Setup Time	(Note 3)	20		24		26		ns
11	Transaction Request Hold Time	(Note 3)	0		0		0		ns
12	$\overline{\text{BINV}}$ Setup Time		11		13		15		ns
13	$\overline{\text{BINV}}$ Hold Time		2		2		2		ns
14	Data Setup Time	(Note 4)	18		22		24		ns
15	Data Hold Time	(Note 4)	2		2		2		ns
16	Instruction Setup Time	(Note 5)	18		22		24		ns
17	Instruction Hold Time	(Note 5)	2		2		2		ns
18	$\overline{\text{CDA}}$ CLK-to-Output-Valid Delay			20		24		26	ns
19	F_{31}–F_0 CLK-to-Output-Valid Delay			30		35		37	ns
20	F_{31}–F_0 Three-State CLK-to-Output-Inactive Delay	(Note 6)		22		25		27	ns
21	Data Operation-Start-to-Output-Valid Delay $F' = (P' \times Q') + T'$			270		285		340	ns
22	MOVE P			110		135		160	ns
23	(All Other Base Operation Codes)			190		235		280	ns
24	$\overline{\text{DRDY}}$ CLK-to-Output-Valid Delay			18		21		23	ns
25	$\overline{\text{DERR}}$ CLK-to-Output-Valid Delay			18		21		23	ns
26	$\overline{\text{EXCP}}$ CLK-to-Output-Valid Delay			18		21		23	ns
27	MSERR CLK-to-Output-Valid Delay			20		25		30	ns

Notes: 1. CLK switching characteristics are made relative to 1.5 V.

2. CLK rise time/fall time measured between 0.8 V and (V_{CC} −1.0 V). Tested on a sample basis only.

3. Transaction request signals include R/$\overline{\text{W}}$, $\overline{\text{DREQ}}$, DREQT_1–DREQT_1, and OPT_2–OPT_0.

4. Data signals include R_{31}–R_0 and S_{31}–S_0.

5. Instruction signals include I_{31}–I_0.

6. Three-State Output Inactive Test Load. Three-State CLK-to-Output-Inactive Delay is measured as the time to a ±500 mV change from prior output level.

Conditions: A. All inputs/outputs are TTL-compatible for V_{IH}, V_{IL}, and V_{OL} unless otherwise noted.

B. All outputs are driving 80 pF unless otherwise noted.

C. All setup, hold, and delay times are measured relative to CLK at 1.5 V unless otherwise noted.

SWITCHING CHARACTERISTICS over MILITARY operating range

No.	Parameter Description	Test Conditions	20 MHz		16 MHz		Unit
			Min.	Max.	Min.	Max.	
1	CLK Period	(Note 1)	50	DC	60	DC	ns
2	CLK Low Time		20		22		ns
3	CLK High Time		20		22		ns
4	CLK Rise Time	(Note 2)		5		5	ns
5	CLK Fall Time	(Note 2)		5		5	ns
	Operation Time, Low-Latency						
6	Mode, $F' = (P' \times Q') + T'$			300		360	ns
7	MOVE P			150		180	ns
8	(All Other Base Operation Codes)			250		300	ns
	Operation Time, Pipeline Mode						
9	All Operations			150		180	ns
10	Transaction Request Setup Time	(Note 3)	24		26		ns
11	Transaction Request Hold Time	(Note 3)	0		0		ns
12	$\overline{BINV}$ Setup Time		14		16		ns
13	$\overline{BINV}$ Hold Time		2		2		ns
14	Data Setup Time	(Note 4)	22		24		ns
15	Data Hold Time		2		2		ns
16	Instruction Setup Time	(Note 5)	22		24		ns
17	Instruction Hold Time		2		2		ns
18	$\overline{CDA}$ CLK-to-Output-Valid Delay			24		26	ns
19	F_{31}–F_0 CLK-to-Output-Valid Delay			35		40	ns
20	F_{31}–F_0 Three-State CLK-to-Output-Inactive Delay	(Note 6)		26		30	ns
	Data Operation-Start-to-Output-Valid Delay						
21	$F' = (P' \times Q') + T'$			285		340	ns
22	MOVE P			135		160	ns
23	(All Other Base Operation Codes)			235		280	ns
24	$\overline{DRDY}$ CLK-to-Output-Valid Delay			21		23	ns
25	$\overline{DERR}$ CLK-to-Output-Valid Delay			21		23	ns
26	$\overline{EXCP}$ CLK-to-Output-Valid Delay			21		23	ns
27	MSERR CLK-to-Output-Valid Delay			25		30	ns

Notes: 1. CLK switching characteristics are made relative to 1.5 V.

2. CLK rise time/fall time measured between 0.8 V and (V_{CC}−1.0 V). Tested on a sample basis only.

3. Transaction request signals include R/$\overline{W}$, $\overline{DREQ}$, $DREQT_1$–$DREQT_0$, and OPT_2–OPT_0.

4. Data signals include R_{31}–R_0 and S_{31}–S_0.

5. Instruction signals include I_{31}–I_0.

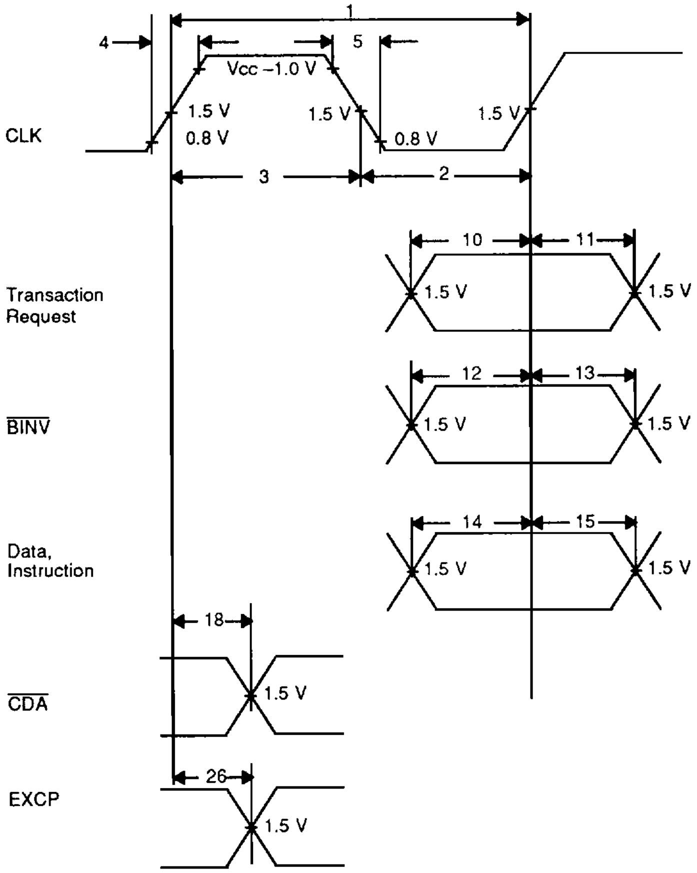
6. Three-State Output Inactive Test Load. Three-State CLK-to-Output-Inactive Delay is measured as the time to a ±500 mV change from prior output level.

Conditions: A. All inputs/outputs are TTL-compatible for V_{IH}, V_{IL}, and V_{OL} unless otherwise noted.

B. All outputs are driving 80 pF unless otherwise noted.

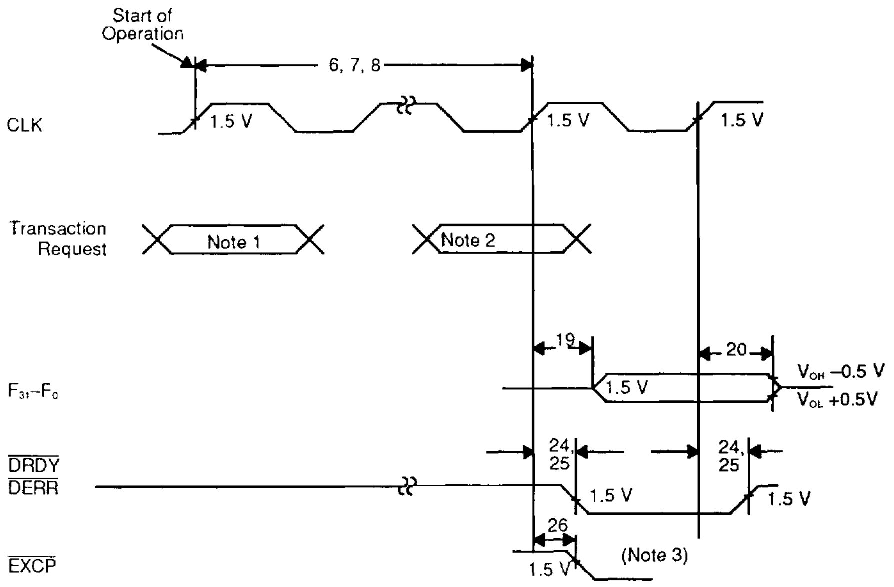
C. All setup, hold, and delay times are measured relative to CLK at 1.5 V unless otherwise noted.

SWITCHING WAVEFORMS



Input Signal Timing; $\overline{\text{CDA}}$, $\overline{\text{EXCP}}$ Timing

SWITCHING WAVEFORMS (continued)



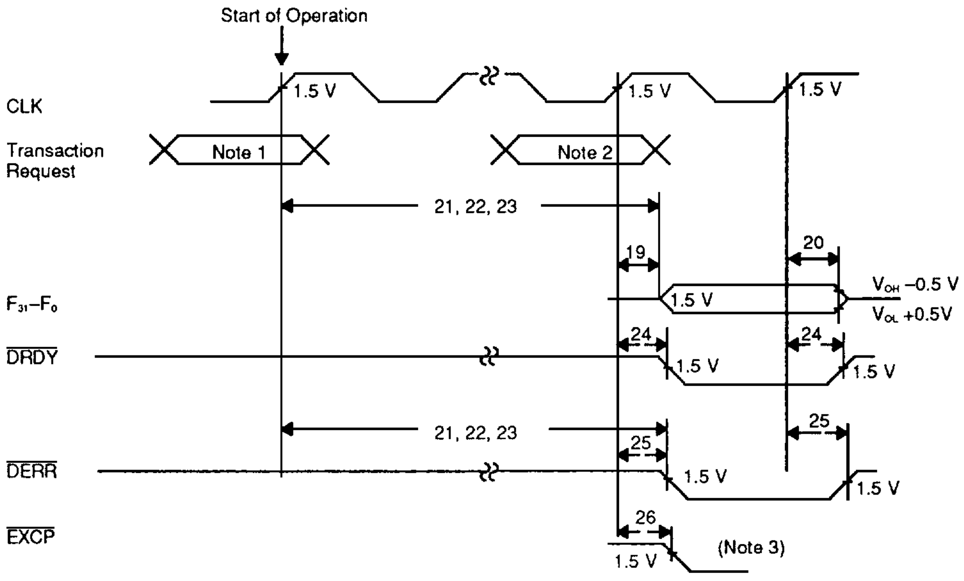
Operation Timing for Flow-Through Mode, $\overline{\text{DRDY}}$, $\overline{\text{DERR}}$ Not Advanced (Mode Register Bit AD = 0)

Notes: 1. Transaction request Write Operand R; Write Operand S; Write Operands R, S; or Write Instruction with Signal DREQT_0 asserted.

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 $\overline{\text{DRDY}}$ or $\overline{\text{DERR}}$ active for both cycles.

3. Signal $\overline{\text{EXCP}}$ is asserted in the presence of unmasked exception.

SWITCHING WAVEFORMS (continued)



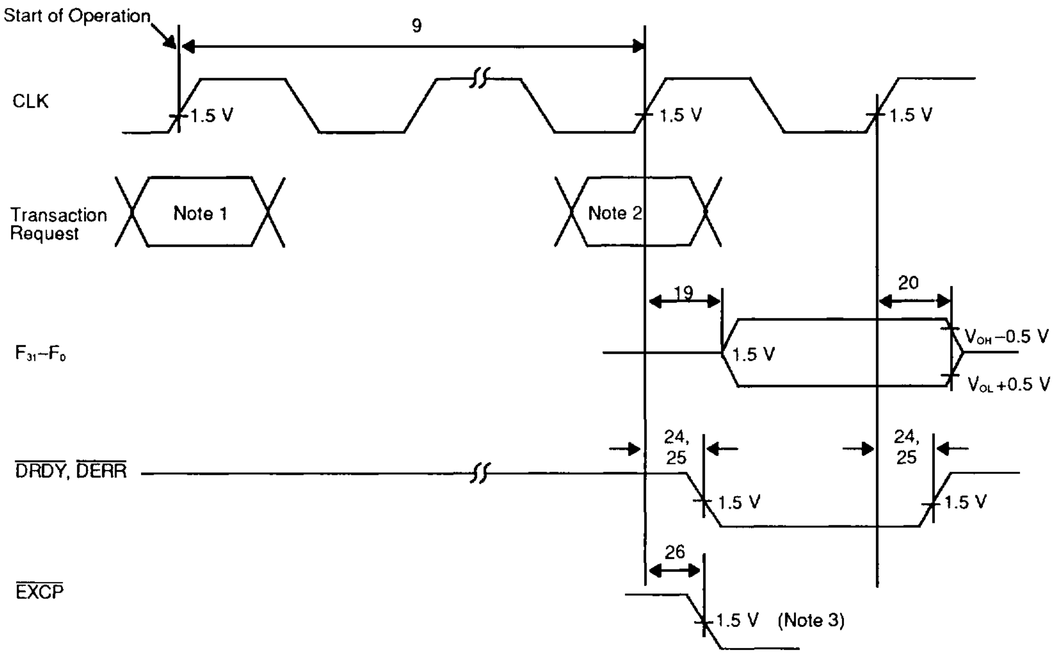
Operation Timing for Flow-Through Mode, $\overline{DRDY}$, $\overline{DERR}$ Advanced (Mode Register Bit AD = 1)

Notes: 1. Transaction request Write Operand R; Write Operand S; Write Operands R, S; or Write Instruction with Signal $DREQT_0$ asserted.

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 consecutive cycles, with $\overline{DRDY}$ or $\overline{DERR}$ active for both cycles.

3. Signal $\overline{EXCP}$ is asserted in the presence of an unmasked exception.

SWITCHING WAVEFORMS (continued)

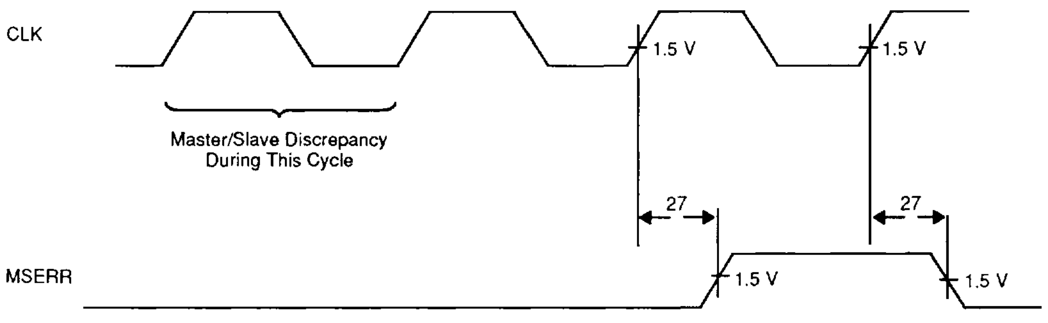


Operation Timing for Pipeline Mode

Notes: 1. Transaction request Write Operand R; Write Operand S; Write Operands R, S; or Write Instruction with signal $DREQT_0$ asserted.

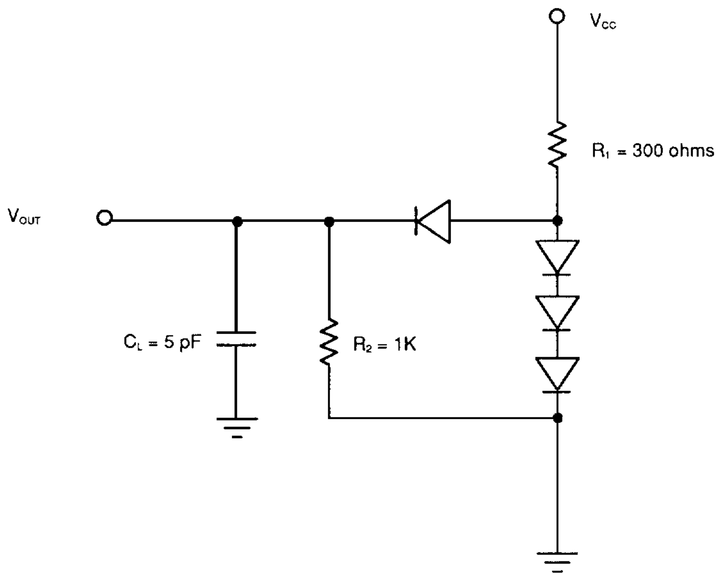
2. Transaction Request Read Result MSBs, Read Result LSBs, Read Flags, Read Status, or Save State. If request Read Result LSBs is issued, the Am29027 produces two data outputs in consecutive cycles, with $\overline{DRDY}$ or $\overline{DERR}$ for both cycles.

3. Signal $\overline{EXCP}$ is asserted in the presence of an unmasked exception.

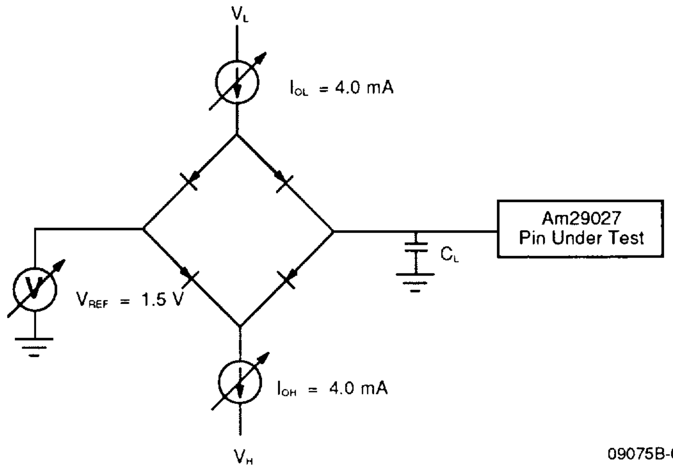


Master/Slave Timing

SWITCHING TEST CIRCUIT



Three-State Output Inactive Test



09075B-001A

C_L is guaranteed to 80 pF.

TEST PHILOSOPHY AND METHODS

The following nine points describe AMD's philosophy for high-volume, high-speed automatic testing.

1. Ensure that the part is adequately decoupled at the test head. Large changes in V_{CC} current as the device switches may cause erroneous function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may start to oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an output transition, ground current may change by as much as 400 mA in 5–8 ns. Inductance in the ground cable may allow the ground pin at the device to rise by hundreds of millivolts momentarily.
4. Use extreme care in defining point input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins and they may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq 0$ V and $V_{IH} \geq 3.0$ V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. Capacitive Loading for AC Testing.

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

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

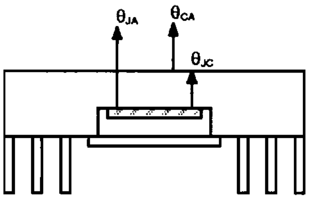
   The noise associated with automatic testing (due to the long, inductive cables) and the high gain of the tested device when in the vicinity of the actual device threshold, frequently give rise to oscillations when testing high-speed circuits. These oscillations are not indicative of a reject device,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

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

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

Am29027 Thermal Characteristics

Pin-Grid-Array Package



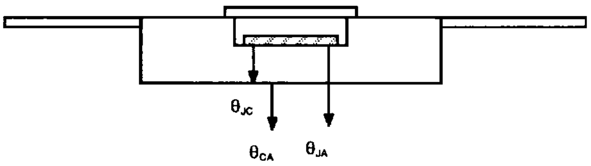
$\theta_{JA} = \theta_{JC} + \theta_{CA}$

Thermal Resistance – °C/Watt

	Airflow—ft./min. (m/sec)					
Parameter	**0 (0)**	**150 (0.76)**	**300 (1.53)**	**480 (2.45)**	**700 (3.58)**	**900 (4.61)**
θ_{JC} Junction-to-Case	4	4	4	4	4	4
θ_{CA} Case-to-Ambient (no Heatsink)	16	14	12	11	9	8
θ_{CA} Case-to-Ambient (with omnidirectional 4-Fin Heatsink, Thermalloy [illegible])	10	6	3	2	2	2
θ_{CA} Case-to-Ambient (with unidirectional Pin Fin Heatsink, Wakefield 840-20)	10	6	3	2	2	2

Am29027 Thermal Characteristics

Ceramic Quad-Flat-Pack Package



$\theta_{JA} = \theta_{JC} + \theta_{CA}$

Thermal Resistance – °C/Watt

	Airflow—ft./min. (m/sec)					
Parameter	**0 (0)**	**150 (0.76)**	**300 (1.53)**	**480 (2.45)**	**700 (3.58)**	**900 (4.61)**
θ_{JC} Junction-to-Case						
θ_{CA} Case-to-Ambient (no Heatsink)						

Note: This is for reference only.

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.

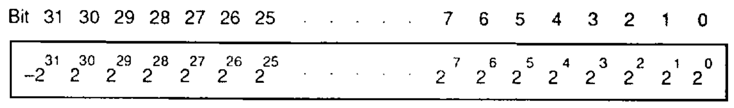
The primary and alternate floating-point formats are selected by mode register fields PFF and AFF. The user may select between floating-point operations and integer operations by means of instruction bit IN_5.

The nine supported formats are described below:

Integer Formats

32-Bit Integer

The 32-bit integer word is arranged as follows:

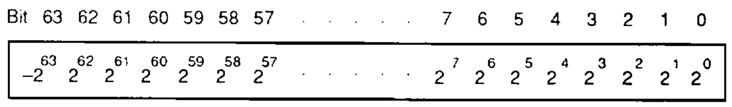


TB001030

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:



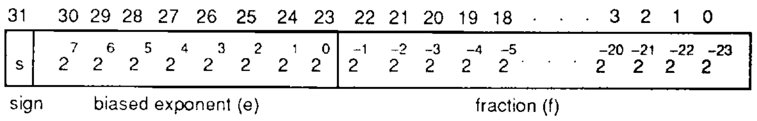
TB001040

The 64-bit word is interpreted as a two's-complement integer. For integer multiplications, the user has the option of interpreting integers as unsigned. An unsigned double-precision integer has a format similar to that of the two's-complement integer, but with an MSB weight of 2^{63}.

IEEE Formats

IEEE Single Precision

The IEEE single-precision word is 32 bits wide and is arranged in the format shown below:



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. 0 may have either sign.

The biased exponent is an 8-bit unsigned integer representing a multiplicative factor of some power of 2. The bias value is 127. If, for example, the multiplicative value for a floating-point number is to be 2^a, the value of the biased exponent is a + 127, where "a" is the true exponent.

The fraction is a 23-bit unsigned fractional field containing the 23 least significant bits of the floating-point number's 24-bit mantissa. The weight of the fraction's most significant bit is 2^{-1}. The weight of the least significant bit is 2^{-23}.

An IEEE floating-point number is evaluated or interpreted as follows:

If $e = 255$ and $f \neq 0$	value = NaN	Not a Number
If $e = 255$ and $f = 0$	value = $(-1)^s\infty$	Infinity
If $0 < e < 255$	value = $(-1)^s 2^{e-127}$ (1.f)	Normalized number
If $e = 0$ and $f \neq 0$	value = $(-1)^s 2^{-126}$ (0.f)	Denormalized number
If $e = 0$ and $f = 0$	value = $(-1)^s 0$	Zero

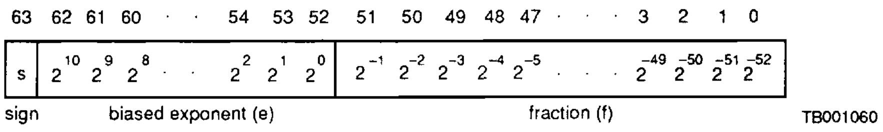
Infinity: Infinity can have either a positive or negative sign. The interpretation of infinities is determined by mode register bit AP.

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 signaling NaN has the MSB of its fraction set to 0 and at least one of the remaining fraction bits set to 1. A quiet NaN has the MSB of its fraction set to 1.

The IEEE format is fully described in ANSI/IEEE Standard 754-1985.

IEEE Double Precision

The IEEE double-precision word is 64 bits wide and is arranged in the format shown below:



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; 0 may have either sign.

The biased exponent is an 11-bit unsigned integer representing a multiplicative factor of some power of 2. 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 mode register bit AP.

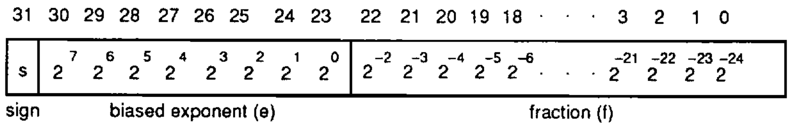
NaN: A NaN is interpreted as a signal or symbol. NaNs are used to indicate invalid operations and as a means of passing process status through a series of calculations. They arise in two ways: either generated by the Am29027 to indicate an invalid operation, or provided by the user as an input. A signaling NaN has the MSB of its fraction set to 0 and at least one of the remaining fraction bits set to 1. A quiet NaN has the MSB of its fraction set to 1.

The IEEE format is fully described in ANSI/IEEE Standard 754-1985.

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; 0 has a positive sign.

The biased exponent is an 8-bit unsigned integer representing a multiplicative factor of some power of 2. 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:

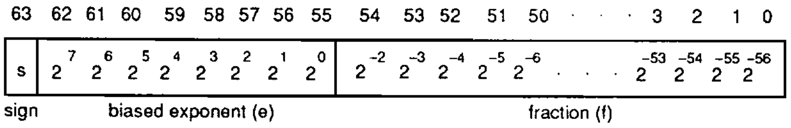
If e≠0 .... value $\neq (-1)^S 2^{e-128}$ (0.1f)
If s = 0 and e = 0 ...... value = 0
If s = 1 and e = 0 .... value = DEC-Reserved Operand

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

The DEC formats are fully described in the VAX™ Architecture Manual.

DEC D

The DEC D word is 64 bits wide and is arranged in the format shown below:



TB001080

The floating-point word is divided into three fields: a single-bit sign, an 8-bit biased exponent, and a 55-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; 0 has a positive sign.

The biased exponent is an 8-bit unsigned integer representing a multiplicative factor of some power of 2. 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:

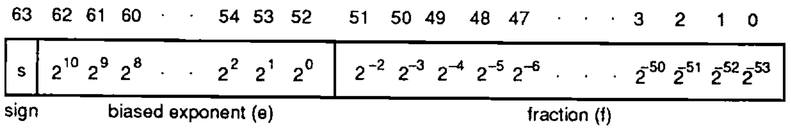
If e≠0 .... value $= (-1)^S 2^{e-128}$ (0.1f)
If s = 0 and e = 0 .... value = 0
If s = 1 and e = 0 .... value = DEC-Reserved Operand

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

The DEC formats are fully described in the VAX Architecture Manual.

DEC G

The DEC G word is 64 bits wide and is arranged in the format shown below:



TB001090

The floating-point word is divided into three fields: a single-bit sign, an 11-bit biased exponent, and a 52-bit fraction.

The sign bit is 0 for positive numbers and 1 for negative numbers; 0 has a positive sign.

The biased exponent is an 11-bit unsigned integer representing a multiplicative factor of some power of 2. The bias value is 1024. If, for example, the multiplicative value for a floating-point number is to be 2^a, the value of the biased exponent is a + 1024, where "a" is the true exponent.

The fraction is a 52-bit unsigned fractional field containing the 52 least significant bits of the floating-point number's 53-bit mantissa. The weight of the fraction's most significant bit is 2^{-2}. The weight of the least significant bit is 2^{-53}.

A DEC G floating-point number is evaluated or interpreted as follows:

If $e \neq 0$	value $= (-1)^S 2^{e-1024}$ (0.1f)
If $s = 0$ and $e = 0$	value $= 0$
If $s = 1$ and $e = 0$	value = DEC-Reserved Operand

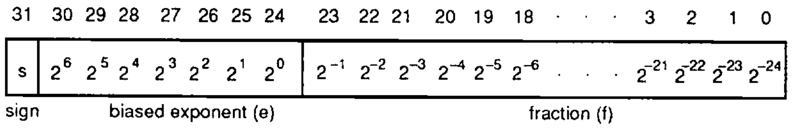
DEC-Reserved Operand: A DEC-Reserved Operand is interpreted as a signal or symbol. DEC-Reserved Operands are used to indicate invalid operations and operations whose results have overflowed the destination format. They may also be used to pass symbolic information from one calculation to another.

The DEC formats are fully described in the VAX Architecture Manual.

IBM Formats

IBM Single Precision

The IBM single-precision word is 32 bits wide and is arranged in the format shown below:



TB001080

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 0 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:

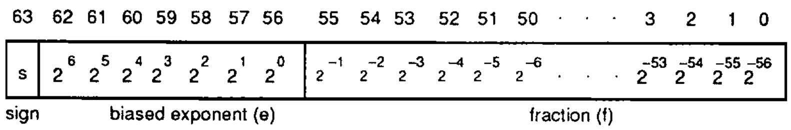
Value $= (-1)^S 16^{e-64}(0.f)$

Zero: There are two classes of zero. If the sign, biased exponent, and fraction are all zero, the operand is known as a "True Zero." If the fraction is zero, but the sign and biased exponent are not both zero, the operand is known as a "Floating-point 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:



TB00110

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 0 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 classes of zero. If the sign, biased exponent, and fraction are all zero, the operand is known as a "True Zero." If the fraction is zero, but the sign and biased exponent are not both zero, the operand is known as a "Floating-point Zero."

The IBM format is fully described in the IBM System/370 Principles of Operation Manual.

APPENDIX B—ROUNDING MODES

The round mode is selected by mode register field RMS as follows:

RMS	Round Mode
000	Round to Nearest (IEEE)
001	Round to Minus Infinity (IEEE)
010	Round to Plus Infinity (IEEE)
011	Round to Zero (IEEE)
100	Round to Nearest (DEC)
101	Round Away from Zero
11X	Illegal Value

Round to Nearest (IEEE)

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 0.

Round to Minus Infinity (IEEE)

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.

Round to Plus Infinity (IEEE)

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.

Round to Zero (IEEE)

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.

Round to Nearest (DEC)

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.

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 is shown in Figures B1 and B2.

The IEEE standard specifies that all four "IEEE" modes be available so that the user may select the mode most appropriate for the algorithm being executed. The DEC standard specifies that two rounding modes be available—Round-to-Nearest (DEC) and Round-to-Zero. The IBM standard specifies that all operations be performed using the Round-to-Zero mode.

It should be noted, however, that the Am29027 permits *any* of the supported rounding modes to be selected, regardless of the format of the operation. It is permissible to use one of the IEEE rounding modes with an IBM operation, or DEC rounding with an IEEE operation, or any other possible combination. For those integer operations where rounding is performed, any rounding mode may be chosen. This flexibility allows the user to select the mode most appropriate for the arithmetic environment in which the processor is operating.

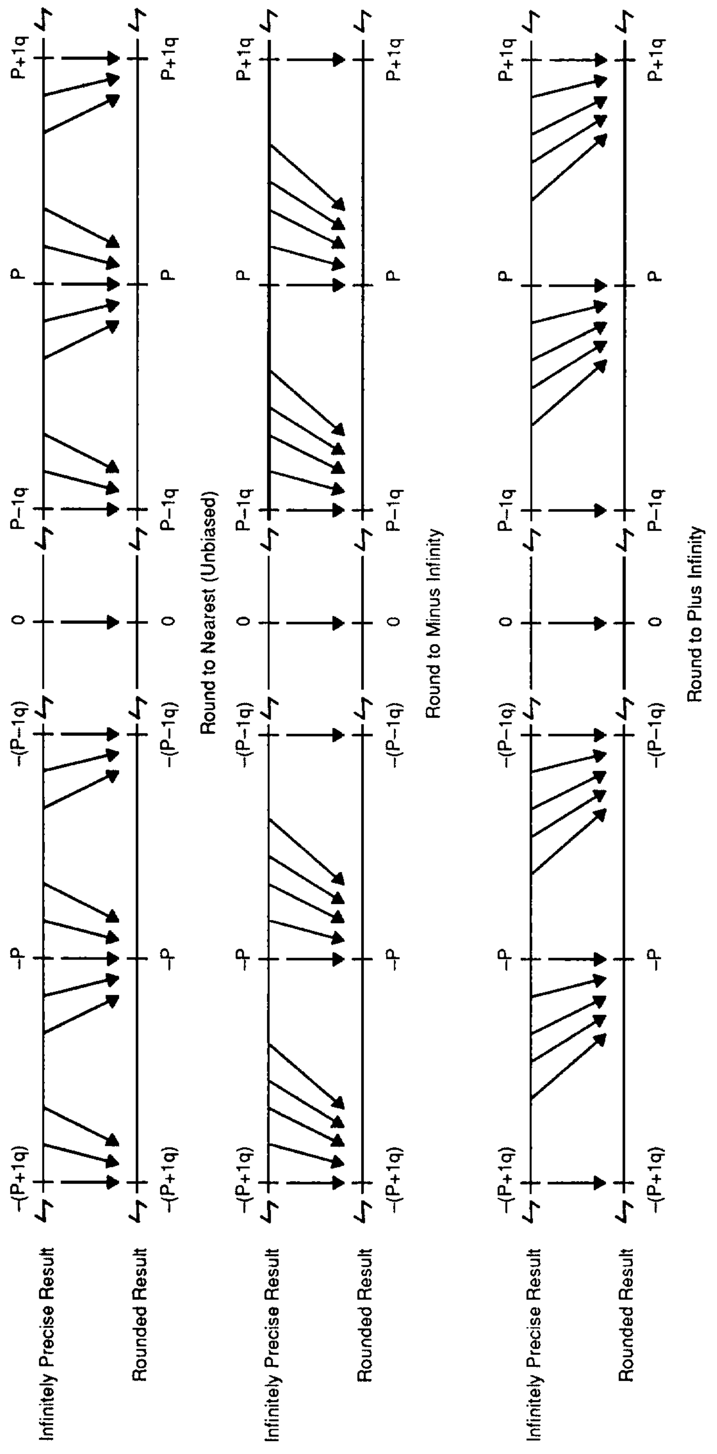


Figure B1. Graphical Interpretation of Round-to-Nearest (Unbiased), Round-to-Minus-Infinity, and Round-to-Plus-Infinity Rounding Modes

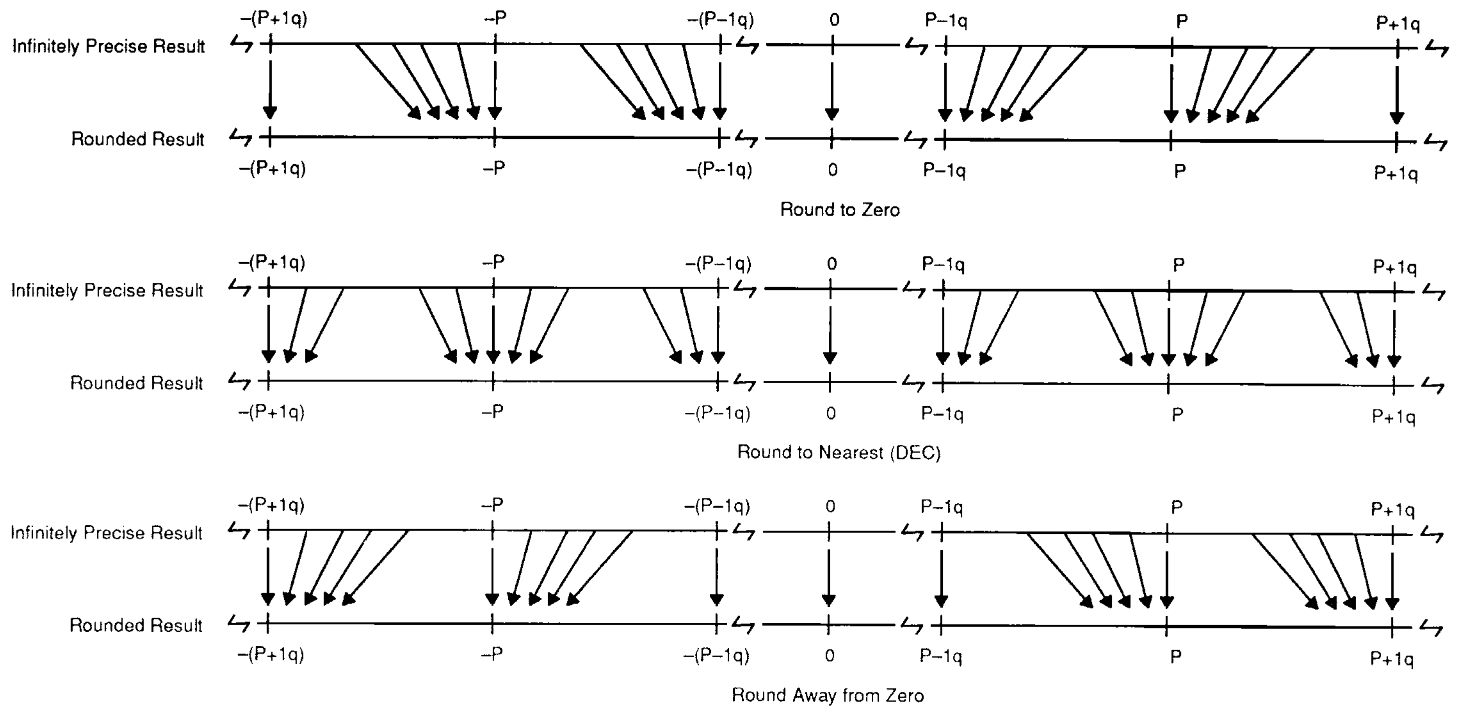


Figure B2. Graphical Interpretation of Round-to-Zero, Round-to-Nearest (DEC), and Round-Away-from-Zero Rounding Modes

APPENDIX C—ADDITIONAL OPERATION DETAILS

There are several cases in which the implementation of the IEEE, DEC, and IBM floating-point standards in the Am29C327 differs from the formal definitions of those standards. This appendix describes these differences.

Differences Between Floating-Point Arithmetic and Am29027 IEEE Operation

Section 7.3 of the IEEE-754 standard specifies that "Trapped overflow on conversion from a binary floating-point format shall deliver to the trap handler a result in that *or a wider format*, possibly with the exponent bias adjusted, but rounded *to the destination's precision*."

According to the IEEE standard, then, if a double-to-single IEEE operation overflows while traps are enabled, the result is a *double-precision* operand, rounded to single-precision width (23-bit fraction), together with a correctly adjusted (double-precision) exponent and the appropriate flags for a trapped overflow.

In the case of an overflow in *any* IEEE operation, the Am29027 returns a result in the destination format specified by the user, rounded to that destination format.

In the case of the double-to-single overflow described above, the result from the Am29027 is a *single-precision* operand, together with a correctly adjusted (single-precision) exponent and the appropriate flags for a trapped overflow.

A simple example serves to illustrate the discrepancy by describing the conversion of the double-precision IEEE number 52B123456789ABCD to single-precision, with traps enabled, and the round-to-nearest rounding mode selected. This number is too large to be represented in single-precision format.

According to the IEEE standard, the result of this operation is the double-precision number 52B1234560000000, comprising the double-precision exponent of the input and a fraction truncated to 23 bits, together with flags V and X.

When the operation is performed in the Am29027, however, using the F′ = P′ operation with appropriate precision controls, the result is the single-precision number 75891A2B, comprising the single-precision (overflowed) exponent reduced by 192 (decimal) and a single-precision fraction, together with flags V and X.

It should be noted that trapped operation is an optional part of the IEEE standard. Full adherence to the IEEE specification of trapped operation is therefore not necessary to ensure compliance with IEEE-754.

Differences Between DEC Floating-Point Arithmetic and Am29027 DEC Operation

The DEC F, DEC D, and DEC G standards, as implemented in the Am29027, differ from the implementations in a VAX only in the way in which the subfields of the floating-point word are arranged. The differences are listed in Table C1.

Table C1. Differences in Am29027 and DEC Floating-Point Formats

	Am29027 Arrangement	VAX Arrangement
DEC F	sign: bit 31 exponent: bits 30–23 fraction: bits 22–0	sign: bit 15 exponent: bits 14–7 fraction: bits 6–0, bits 31–16
DEC D	sign: bit 63 exponent: bits 62–55 fraction: bits 54–0	sign: bit 15 exponent: bits 14–7 fraction: bits 6–0, bits 31–16, bits 47–32, bits 63–48
DEC G	sign: bit 63 exponent: bits 62–52 fraction: bits 51–0	sign: bit 15 exponent: bits 14–4 fraction: bits 3–0, bits 31–16, bits 47–32, bits 63–48

Differences Between IBM 370 Floating-Point Arithmetic and Am29027 IBM Operation

The Am29027's deviations from the IBM standard may be summarized as follows, *assuming that the user has selected the round-to-nearest rounding mode*:

1. The Am29027 provides more guard bits in its internal format than specified by the IBM standard. With certain combinations of input operands, the Am29027 produces more accurate results than a standard IBM processor for instructions based on addition operations and comparisons.
2. The discrepancies are much larger for single-precision operations than double-precision operations, because the difference in the number of guard bits is much greater (33 more for single, one more for double).
3. There is no universal rule for determining whether a given set of input operands will result in a discrepancy. Pro vided the conditions in (1) above are met, the user must examine each operation on a case-by-case basis, taking into account the input operands and the internal formats discussed in this section.
4. The Am29027 does not produce unnormalized results from additions. The results of all addition operations are renormalized. Am29027 internal formats are compared with IBM internal formats in Figure C1.

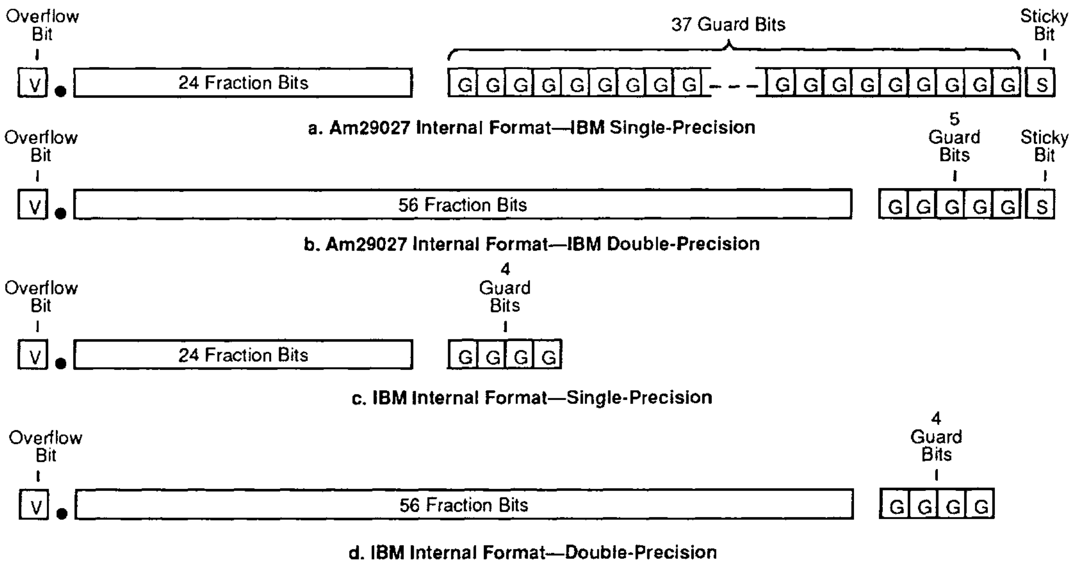
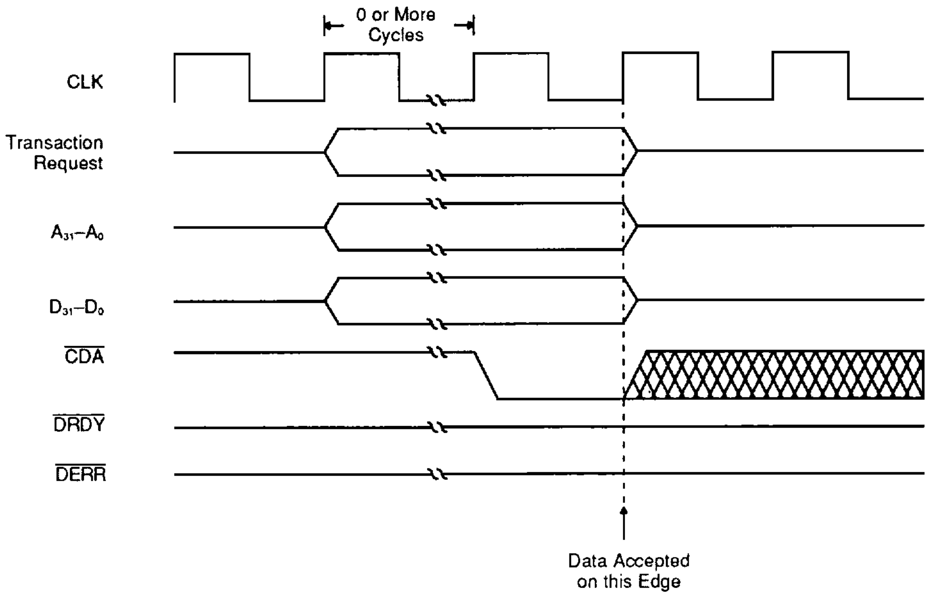
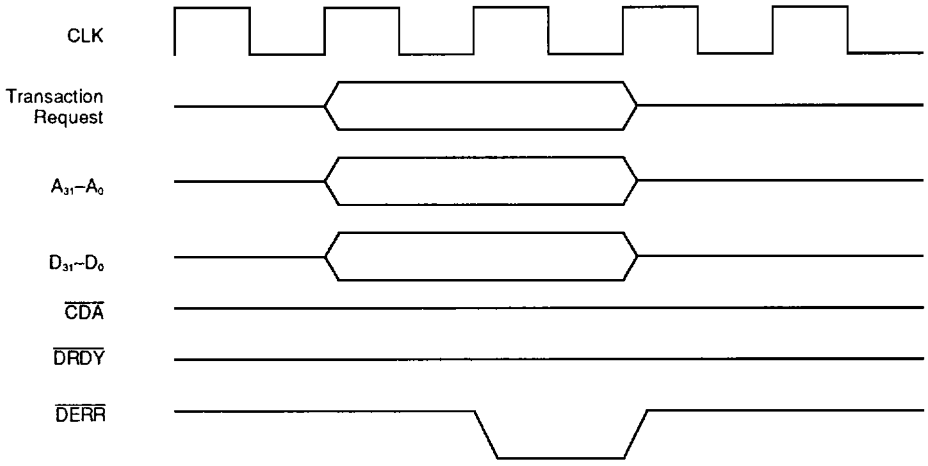


Figure C1. Differences in Internal Mantissa Formats of an IBM CPU and the Am29027

APPENDIX D—TRANSACTION REQUEST/OPERATION TIMING



a. Normal Operation, Data Accepted

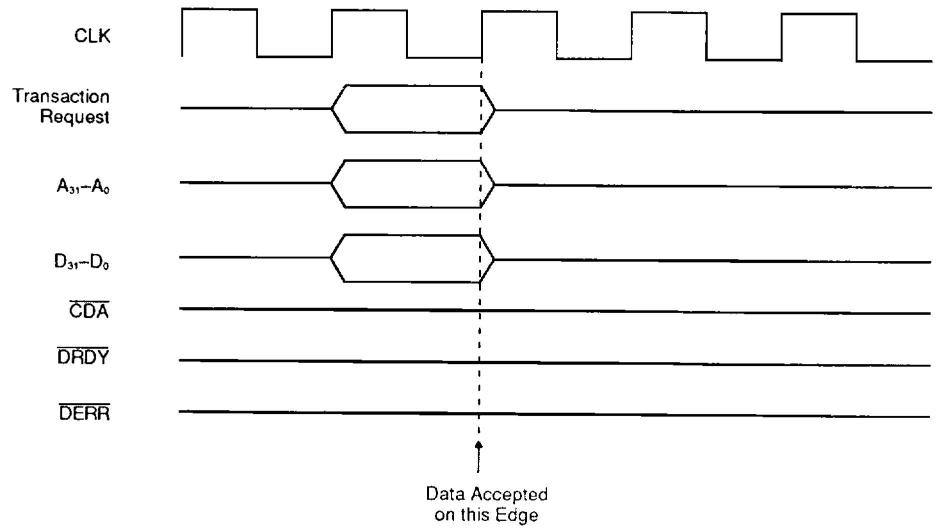


b. Halt On Error Mode, Unmasked Exception Present

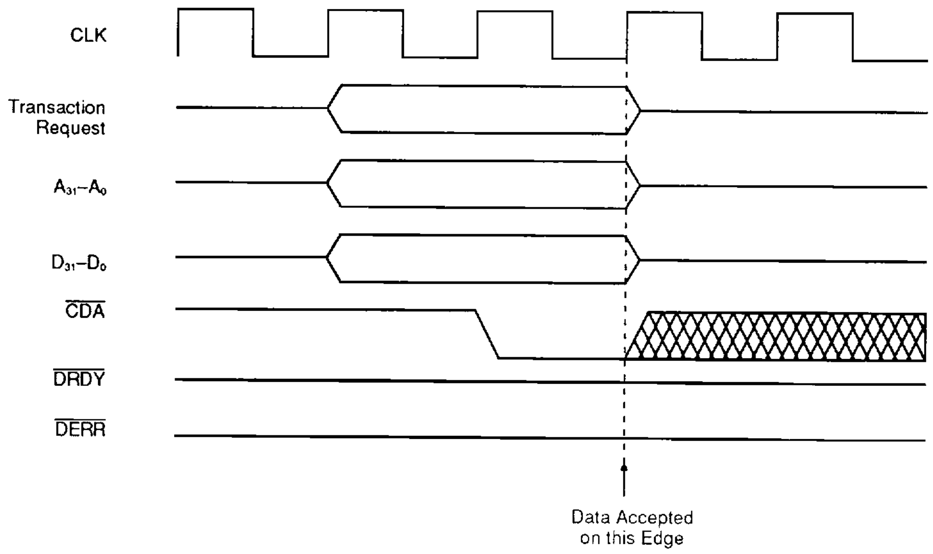
091148-017C

Note: Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D1. Timing for the Write Operand R, Write Operand S, Write Operands R, S, and Write Instruction Transaction Requests



a. $\overline{\text{CDA}}$ Low

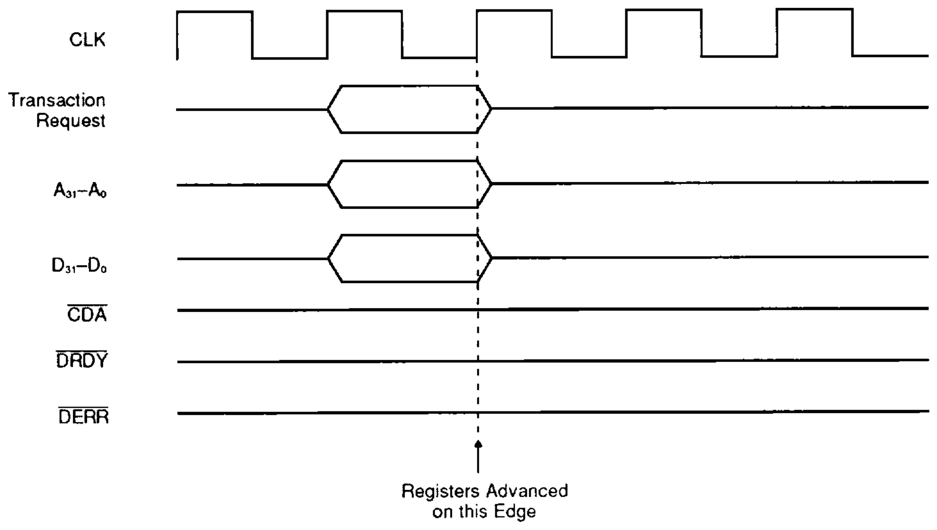


b. $\overline{\text{CDA}}$ High Initially

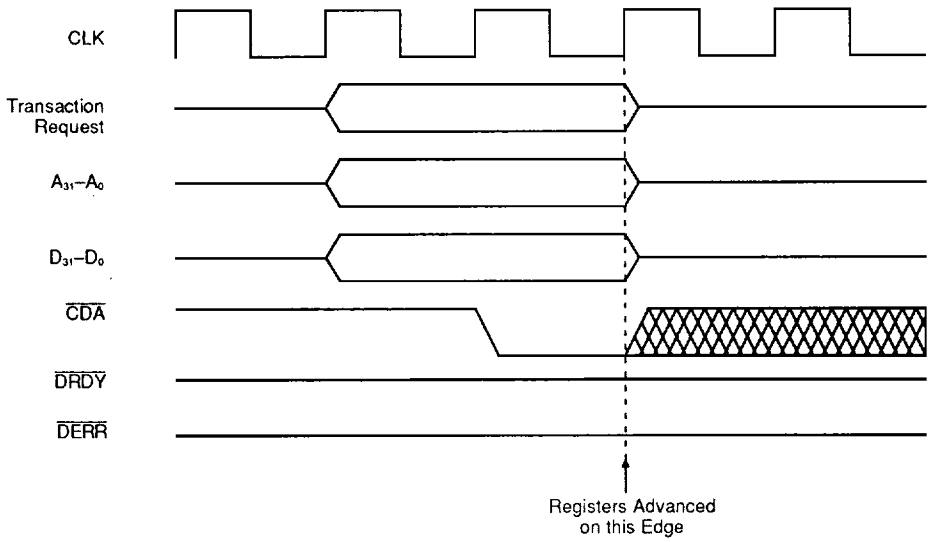
Note: Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

09114-018C

Figure D2. Timing for the Write Mode, Write Status, and Write Register File Precisions Transaction Requests



a. $\overline{\text{CDA}}$ Low

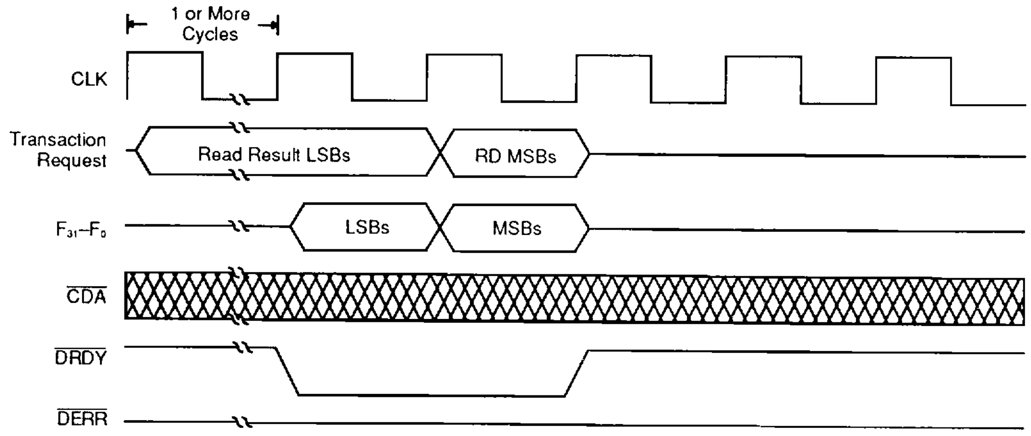


b. $\overline{\text{CDA}}$ High Initially

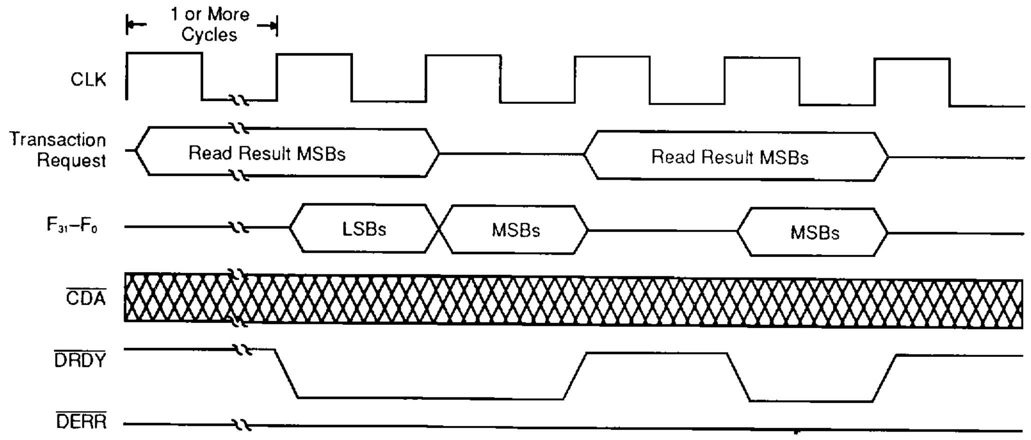
09114-019C

Note: Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D3. Timing for the Advance Temp. Registers Transaction Request



a. Read Result MSBs Request Issued in Cycle after Read Result LSBs Request



b. Read Result MSBs Request Issued Two or More Cycles after Read Result LSBs Request

09114-020C

Figure D4. Timing for the Read Result LSBs Transaction Request, No Unmasked Exceptions

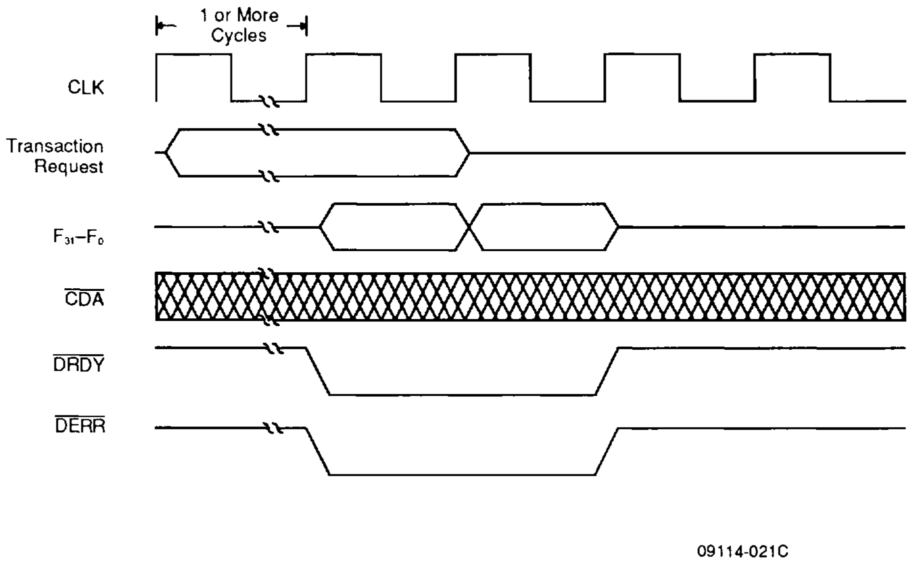
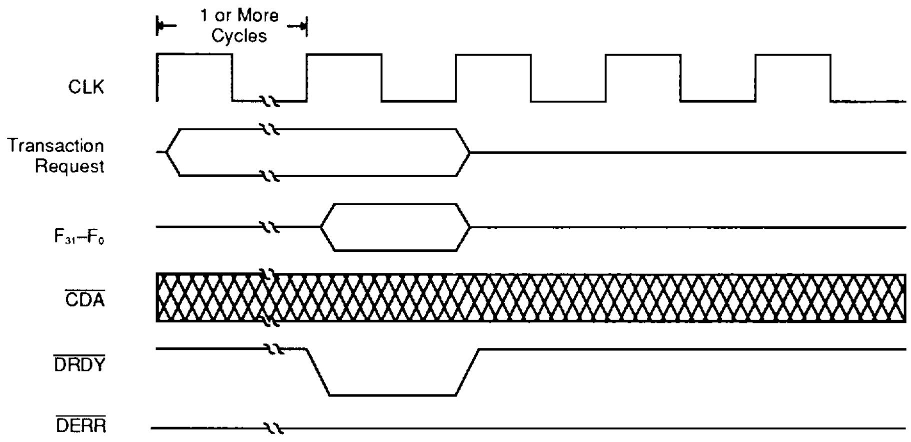
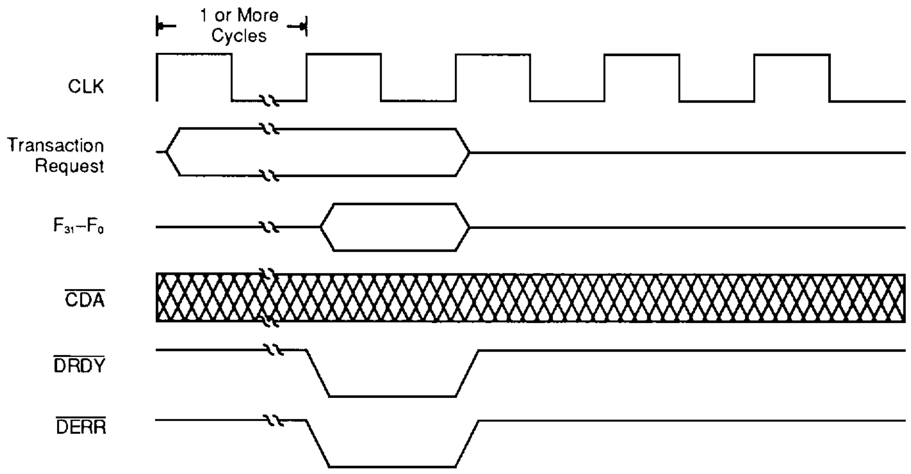


Figure D5. Timing for Read Result LSBs Transaction Request, Unmasked Exception Present



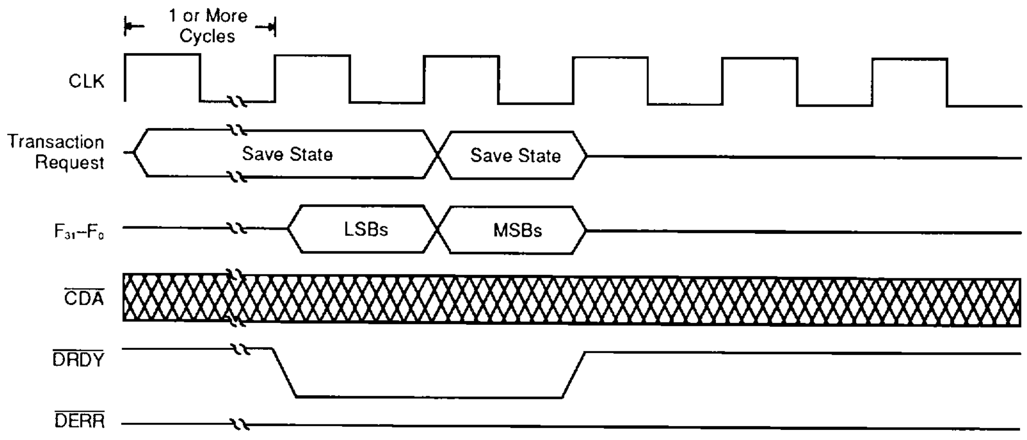
a. No Unmasked Exceptions Present



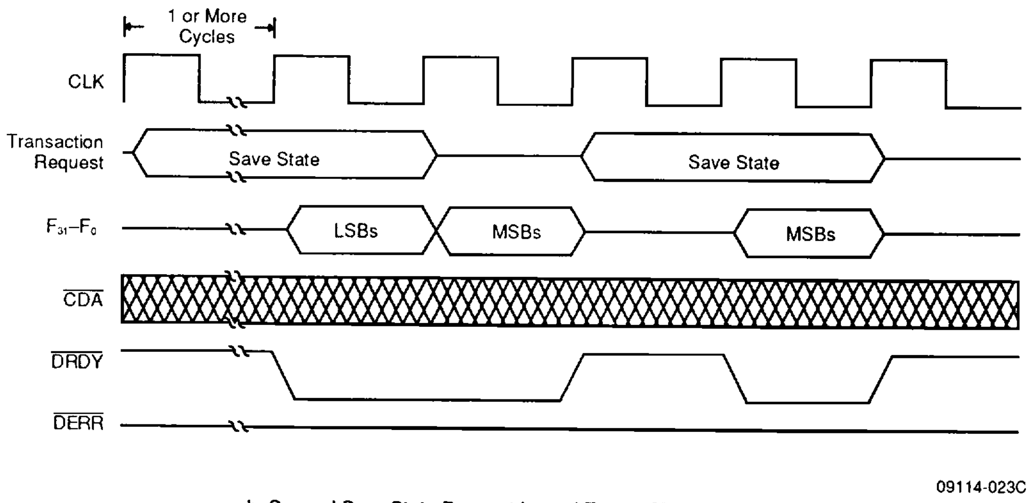
b. Unmasked Exceptions Present

09114-022C

Figure D6. Timing for Read Result MSBs, Read Flags, and Read Status Transaction Requests

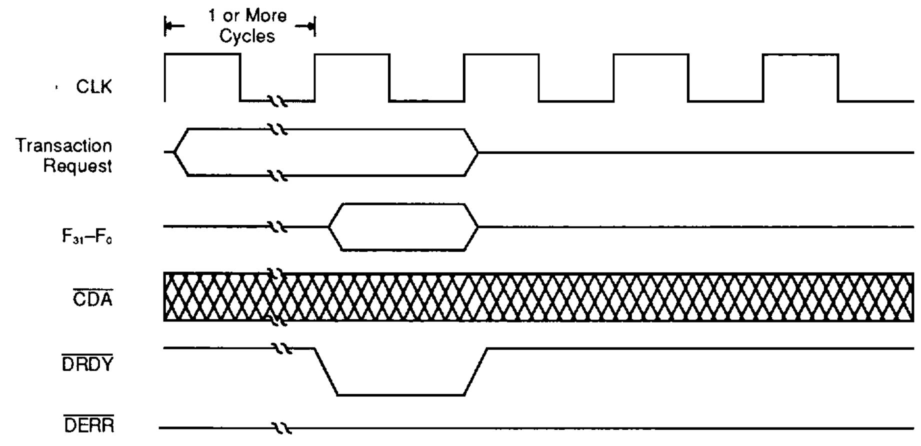


a. Second Save State Request Issued In Cycle Following First Request



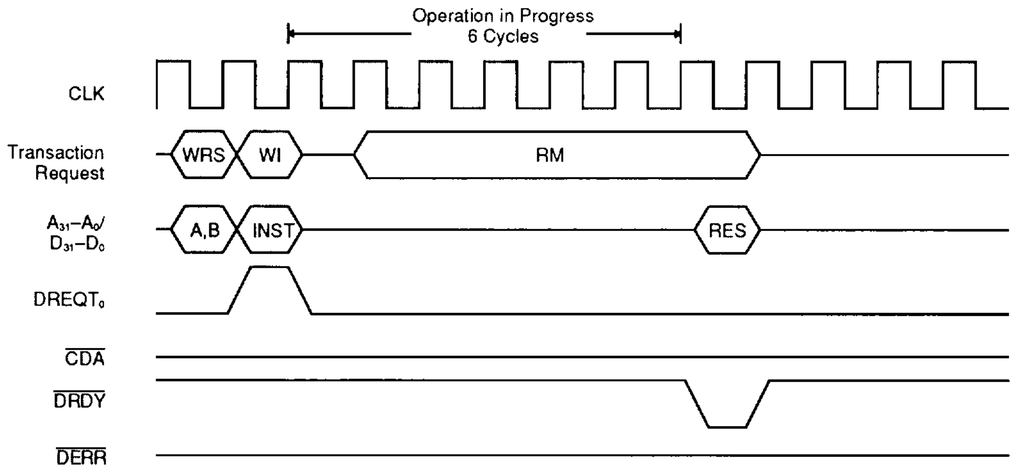
b. Second Save State Request Issued Two or More Cycles after First Request

Figure D7. Timing for the Save State Transaction Request, 64-Bit Resources (Registers R, R-Temp, S, S-Temp; Register File Locations RF_7–RF_0: Mode Register)



09114-024C

Figure D8. Timing for the Save State Transaction Request, 32-Bit Resources (Instruction Register, Register I-Temp, Status Register, Precision Register)

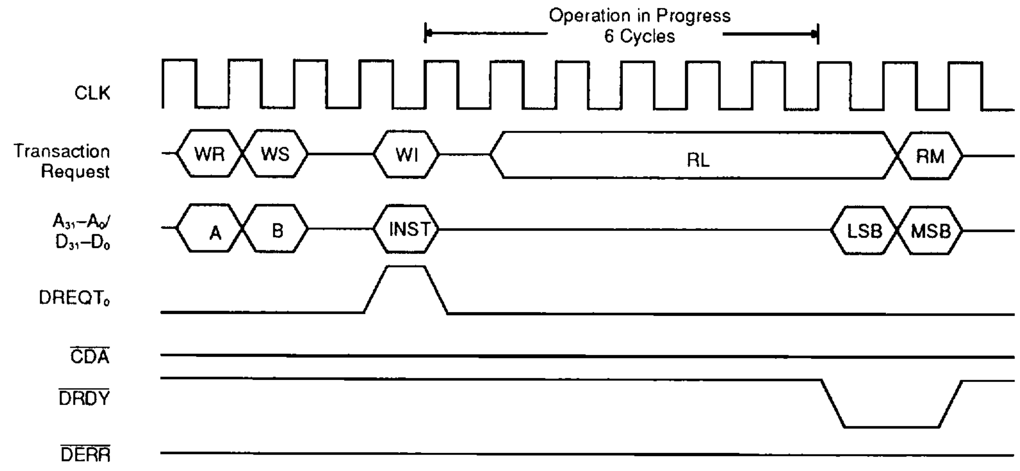


Notes:	WRS	= Write Operands R, S	WI	=	Write Instruction
	RM	= Read MSBs	A, B	=	Operands A, B
	INST	= Addition Instruction	RES	=	Result

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

09114-025C

Figure D9. Typical Timing for Single-Precision Operation in Flow-Through Mode—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

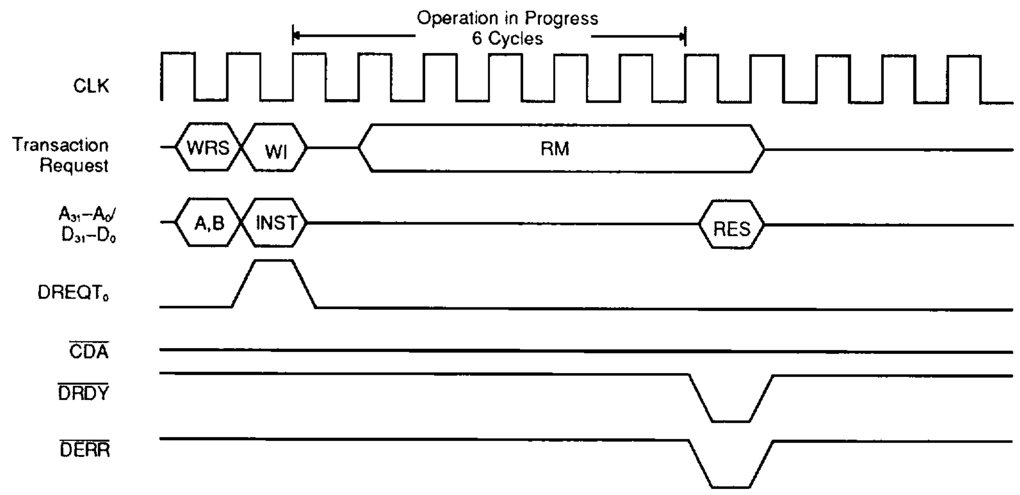


Notes:			
WR	= Write Operand R	WS	= Write Operand S
WI	= Write Instruction	RL	= Read LSBs
RM	= Read MSBs	A	= Operand A
B	= Operand B	INST	= Addition Instruction
LSB	= Result LSBs	MSB	= Result MSBs

09114-026C

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D10. Typical Timing for the Double-Precision Operation in Flow-Through Mode—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

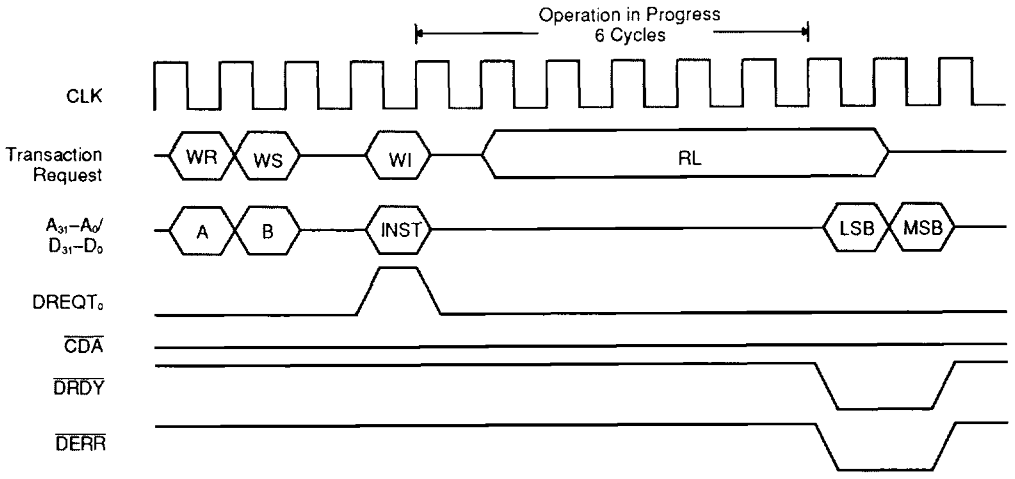


Notes:			
WRS	= Write Operands R, S	WI	= Write Instruction
RM	= Read MSBs	A, B	= Operands A, B
INST	= Addition Instruction	RES	= Result

09114-027C

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D11. Typical Timing for Single-Precision Operation in Flow-Through Mode, with Unmasked Exception Present—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

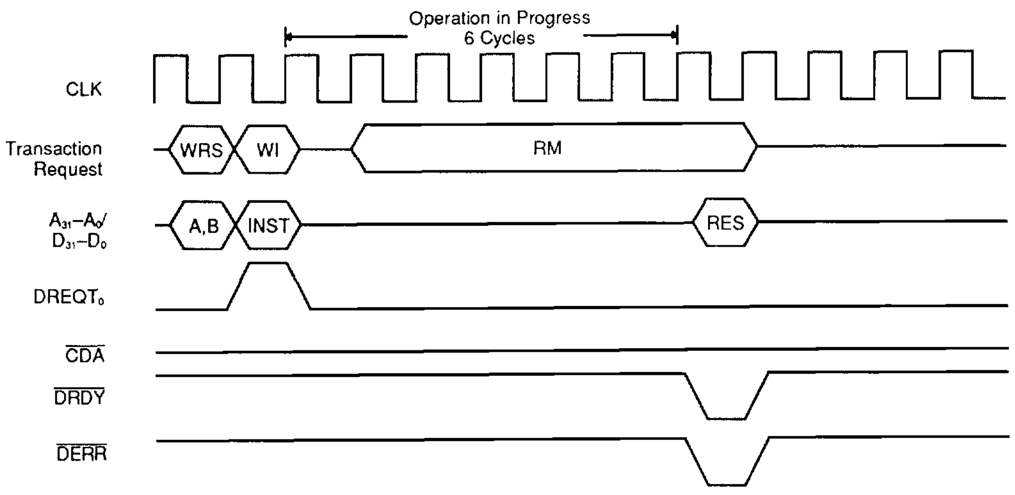


Notes: WR	= Write Operand R	WS	= Write Operand S
WI	= Write Instruction	RL	= Read LSBs
A	= Operand A	B	= Operand B
INST	= Addition Instruction	LSB	= Result LSBs
MSB	= Result MSBs		

09114-028C

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D12. Typical Timing for Double-Precision Operation in Flow-Through Mode, with Unmasked Exception Present—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

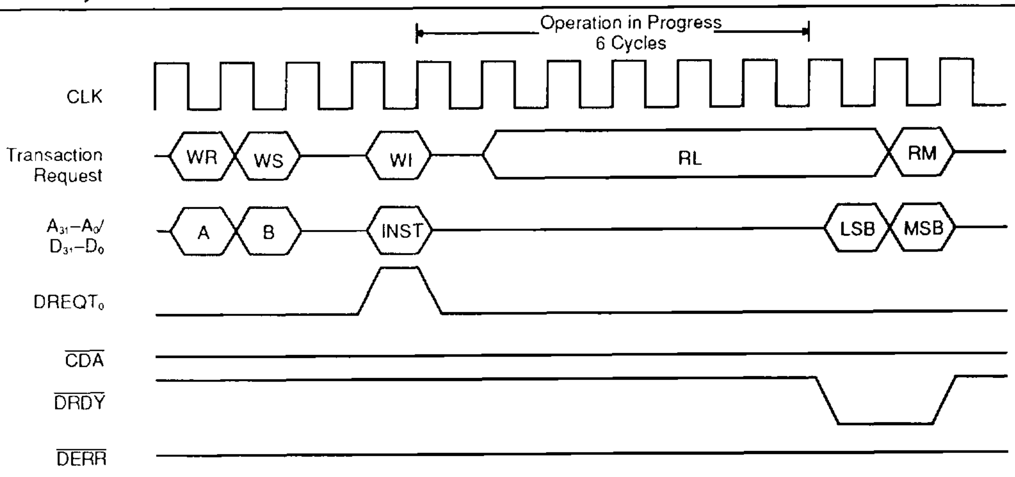


Notes: WRS	= Write Operands R, S	WI	= Write Instruction
RM	= Read MSBs	A, B	= Operands A, B
INST	= Addition Instruction	RES	= Result

09114-029C

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D13. Typical Timing for Single-Precision Operation in Flow-Through Mode, with $\overline{\text{DRDY}}$ Advanced—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

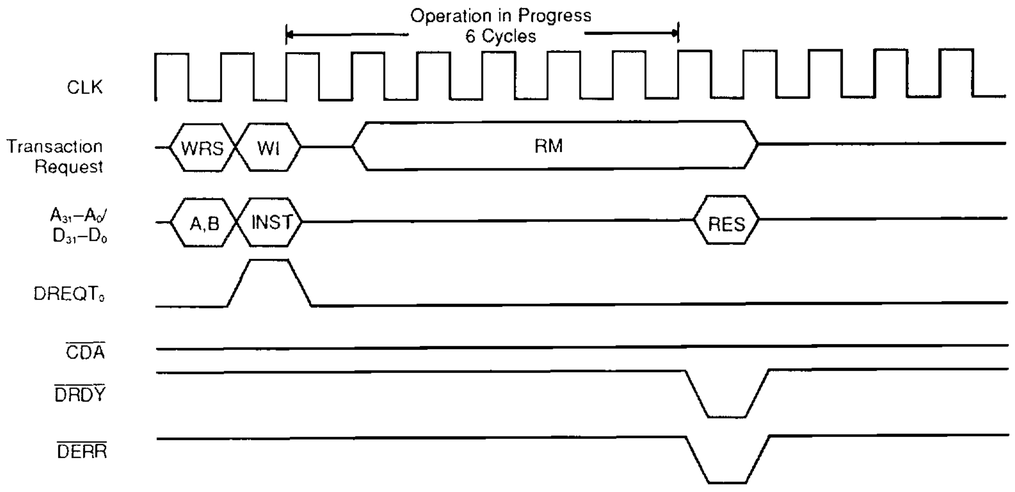


Notes:			
WR	= Write Operand R	WS	= Write Operand S
WI	= Write Instruction	RL	= Read LSBs
RM	= Read MSBs	A	= Operand A
B	= Operand B	INST	= Addition Instruction
LSB	= Result LSBs	MSB	= Result MSBs

09114-030C

Signals $A_{31}-A_0$ and $D_{31}-D_0$ are the Am29000 address and data buses, respectively.

Figure D14. Typical Timing for Double-Precision Operation in Flow-Through Mode, with $\overline{\text{DRD}}$ Advanced—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

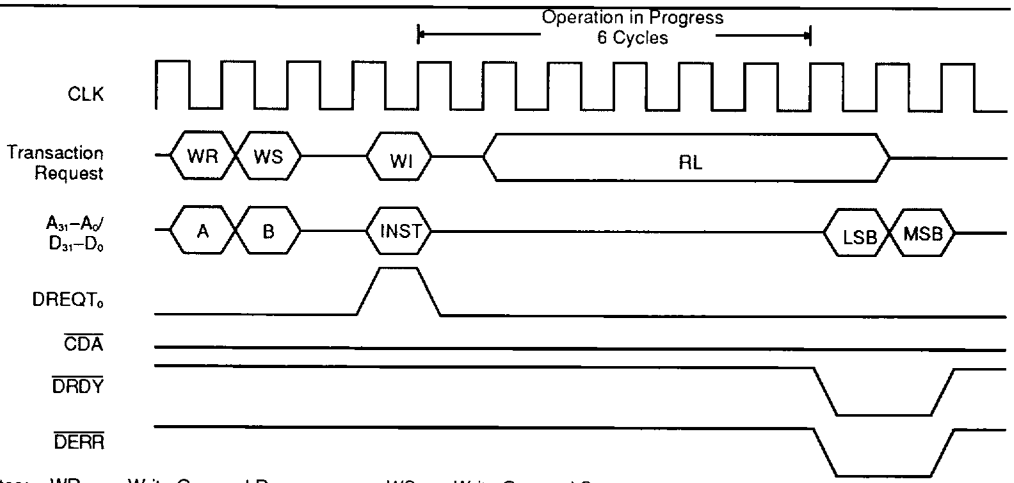


Notes:			
WRS	= Write Operands R, S	WI	= Write Instruction
RM	= Read MSBs	A, B	= Operands A, B
INST	= Addition Instruction	RES	= Result

09114-031C

Signals $A_{31}-A_0$ and $D_{31}-D_0$ are the Am29000 address and data buses, respectively.

Figure D15. Typical Timing for Single-Precision Operation in Flow-Through Mode, with $\overline{\text{DRDY}}$ Advanced and Unmasked Exception Present—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

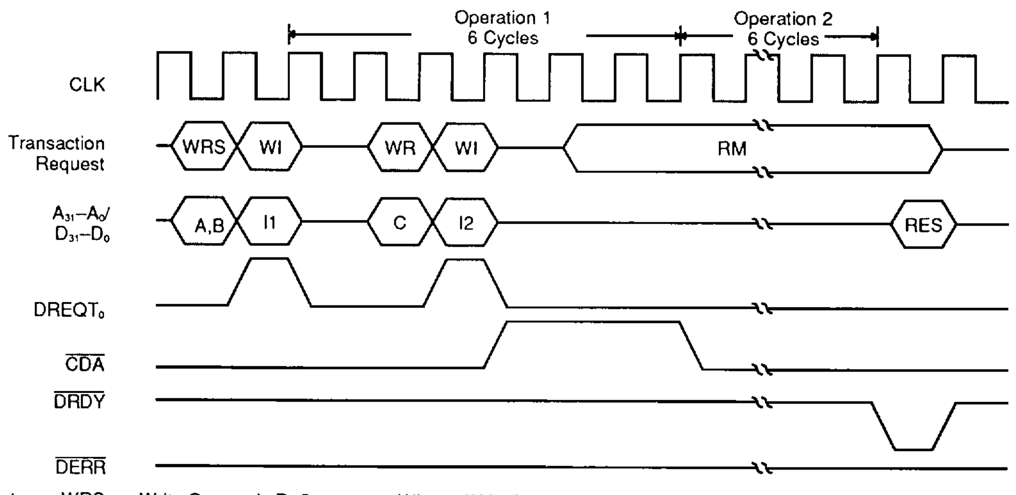


Notes: WR	= Write Operand R	WS	= Write Operand S
WI	= Write Instruction	RL	= Read LSBs
A	= Operand A	B	= Operand B
INST	= Addition Instruction	LSB	= Result LSBs
MSB	= Result MSBs		

09114-037C

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D16. Typical Timing for Double-Precision Operation in Flow-Through Mode, with $\overline{DRDY}$ Advanced and Unmasked Exception Present—Perform the Operation A PLUS B, Read the Result; Mode Register Field PLTC = 6

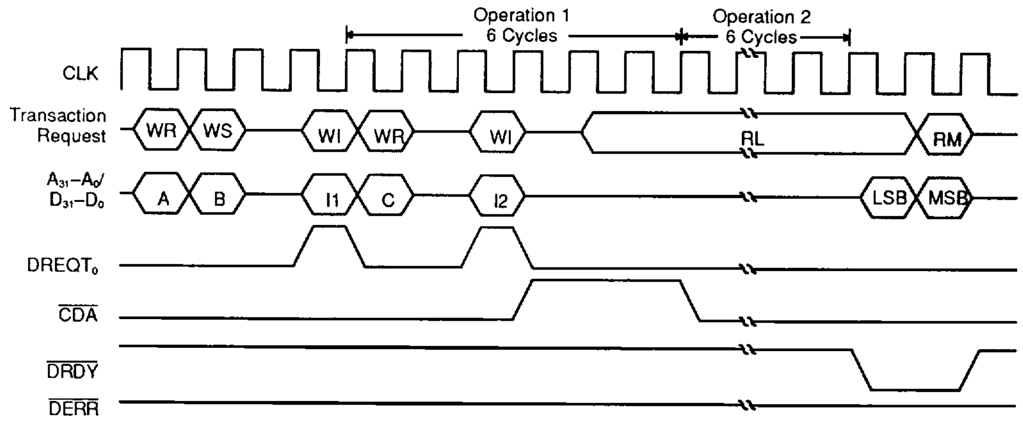


Notes: WRS	= Write Operands R, S	WI	= Write Instruction
WR	= Write Operand R	RM	= Read MSBs
A, B	= Operands A, B	I1	= Addition Instruction
C	= Operand C	I2	= Multiplication Instruction
RES	= Result		

09114-032C

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D17. Typical Timing for Overlapped Single-Precision Operations in Flow-Through Mode; Perform the Compound Operation (A PLUS B) × C by Performing Operations: (1) $RF_0 \leftarrow$ A PLUS B, (2) $RF_0 \times$ C Mode Register Field PLTC = 6

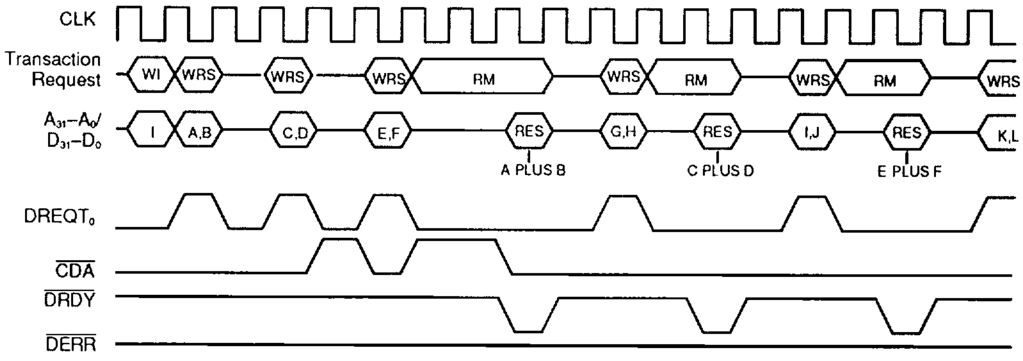


Notes:		
WR = Write Operand R	WS = Write Operand S
WI = Write Instruction	RL = Read LSBs
RM = Read MSBs	A = Operand A
B = Operand B	C = Operand C
I1 = Addition Instruction	I2 = Multiplication Instruction
LSB = Result LSBs	MSB = Result MSBs

Signals A_{31}–A_{01} and D_3–D_0 are the Am29000 address and data buses, respectively.

09114-033C

Figure D18. Typical Timing for Overlapped Double-Precision Operations in Flow-Through Mode; Perform the Compound Operation (A PLUS B) × C by Performing Operations: (1) $RF_0 \leftarrow$ A PLUS B, (2) $RF_0 \times$ C; Mode Register Field PLTC = 6 Mode Register Field PLTC = 6

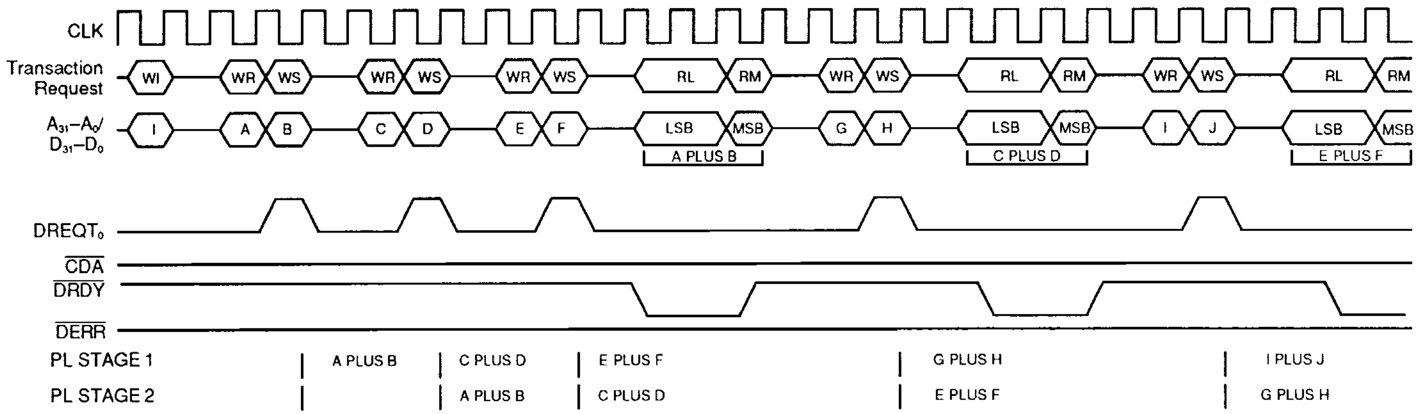


PL STAGE 1	A PLUS B	C PLUS D	E PLUS F	G PLUS H	I PLUS J	
PL STAGE 2		A PLUS B	C PLUS D	E PLUS F	G PLUS H	

Notes:		
WI = Write Instruction	WRS = Write Operands R, S
RM = Read MSBs	I = Addition Instruction
A, B, . . . = Operands	RES = Result

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

Figure D19. Typical Timing for Single-Precision Operations in Pipeline Mode; Perform a Series of Addition Operations A PLUS B, C PLUS D, E PLUS F, . . . Mode Register Field PLTC = 3



Notes: WI = Write Instruction
WS = Write Operand S
RM = Read MSBs
A, B, . . . = Operands
MSB = Result MSBs
WR = Write Operand R
RL = Read LSBs
I = Addition Instruction
LSB = Result LSBs

Signals A_{31}–A_0 and D_{31}–D_0 are the Am29000 address and data buses, respectively.

09114-035C

Figure D20. Typical Timing for Double-Precision Operations in Pipeline Mode; Perform a Series of Addition Operations A PLUS B, C PLUS D, E PLUS F, . . . Mode Register Field PLTC = 3