T-49-17-32

Advance Information

Am29005

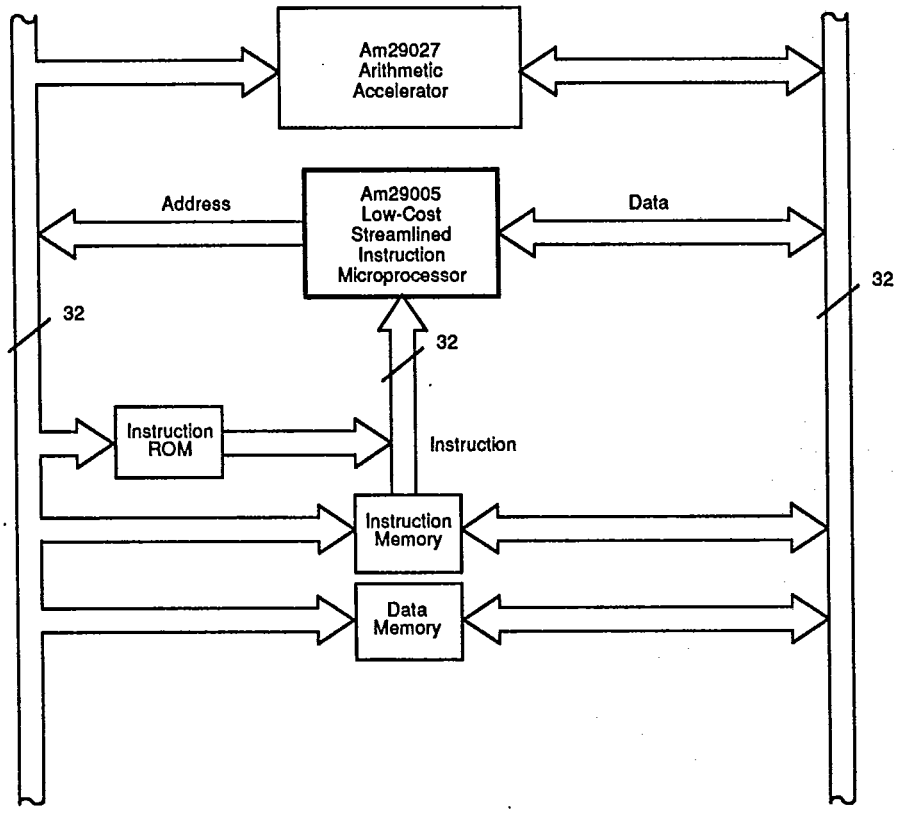
Low-Cost Streamlined Instruction Microprocessor

Advanced Micro Devices

DISTINCTIVE CHARACTERISTICS

- Full 32-bit, three-bus architecture
- Nine million instructions per second (MIPS) sustained at its 16-MHz operating frequency
- Efficient execution of high-level language programs
- CMOS technology
- Concurrent instruction and data accesses
- Burst-mode access support
- 192 general-purpose registers
- Demultiplexed, pipelined address, instruction, and data buses
- Three-address instruction architecture
- On-chip byte-alignment support allows optional byte/half-word accesses

SIMPLIFIED BLOCK DIAGRAM



09075B-003A

Publication # 13089 Rev. B Amendment 0
Issue Date: August 1990

A

Table of Contents

B

GENERAL DESCRIPTION

The Am29005™ Low-Cost Streamlined Instruction Processor is a high-performance, general-purpose, 32-bit microprocessor implemented in CMOS technology. It supports a variety of applications by virtue of a flexible architecture and rapid execution of simple instructions that are common to a wide range of tasks.

The Am29005 microprocessor efficiently performs operations common to all systems, while deferring most decisions on system policies to the system architect. It is well-suited for application in cost-sensitive embedded systems like laser printers, communications and graphics controllers, or other applications where high performance, flexibility, and the ability to program using standard software tools is important.

The Am29005 microprocessor instruction set has been influenced by the results of high-level language, optimizing compiler research. It is appropriate for a variety of languages because it efficiently executes operations that are common to all languages. Consequently, the Am29005 microprocessor is an ideal target for high-level languages such as C, FORTRAN, Pascal, Ada®, and COBOL.

The processor is available in a 168-lead plastic-quad-flat-pack (PQFP) package. The package has 141 signal pins and 27 power and ground pins. A representative system diagram is shown on page 1.

29K™ Family Development Support Products

Contact your local AMD® representative for information on AMD's 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

RELATED AMD PRODUCTS

Am29005 Microprocessor Peripheral Devices

Part No.	Description
Am29027™	Arithmetic Accelerator

PQFP PIN DESIGNATIONS
(Sorted by Pin Number)

T-49-17-32

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	$\overline{DRDY}$	43	V_{CC}	85	GND	127	GND
2	$\overline{CDA}$	44	I_3	86	A_{31}	128	OPT_0
3	INCLK	45	I_2	87	A_{30}	129	OPT_1
4	PWRCLK	46	I_1	88	A_{29}	130	OPT_2
5	SYSCLK	47	GND	89	A_{28}	131	SUP/$\overline{US}$
6	GND	48	I_0	90	A_{27}	132	IREQT
7	V_{CC}	49	D_0	91	A_{26}	133	$STAT_0$
8	GND	50	D_1	92	A_{25}	134	$STAT_1$
9	$\overline{RESET}$	51	D_2	93	A_{24}	135	$STAT_2$
10	$CNTL_0$	52	D_3	94	A_{23}	136	MSERR
11	$CNTL_1$	53	D_4	95	A_{22}	137	$DREQT_0$
12	$\overline{TEST}$	54	D_5	96	A_{21}	138	$DREQT_1$
13	I_{31}	55	D_6	97	A_{20}	139	$\overline{LOCK}$
14	I_{30}	56	D_7	98	A_{19}	140	R/$\overline{W}$
15	I_{29}	57	D_8	99	A_{18}	141	$\overline{DREQ}$
16	I_{28}	58	D_9	100	A_{17}	142	$\overline{PDA}$
17	I_{27}	59	D_{10}	101	A_{16}	143	$\overline{PIA}$
18	I_{26}	60	D_{11}	102	A_{15}	144	$\overline{IREQ}$
19	I_{25}	61	D_{12}	103	GND	145	$\overline{BGRT}$
20	I_{24}	62	D_{13}	104	V_{CC}	146	$\overline{DBREQ}$
21	GND	63	D_{14}	105	V_{CC}	147	$\overline{IBREQ}$
22	V_{CC}	64	V_{CC}	106	A_{14}	148	$\overline{BINV}$
23	I_{23}	65	GND	107	A_{13}	149	V_{CC}
24	I_{22}	66	D_{15}	108	A_{12}	150	V_{CC}
25	I_{21}	67	D_{16}	109	A_{11}	151	GND
26	I_{20}	68	D_{17}	110	A_{10}	152	V_{CC}
27	I_{19}	69	D_{18}	111	A_1	153	GND
28	I_{18}	70	D_{19}	112	A_0	154	$\overline{TRAP_0}$
29	I_{17}	71	D_{20}	113	$MPGM_0$	155	$\overline{TRAP_1}$
30	I_{16}	72	D_{21}	114	$MPGM_1$	156	$\overline{INTR_0}$
31	I_{15}	73	D_{22}	115	V_{CC}	157	$\overline{INTR_1}$
32	I_{14}	74	D_{23}	116	V_{CC}	158	$\overline{INTR_2}$
33	I_{13}	75	D_{24}	117	A_9	159	$\overline{INTR_3}$
34	I_{12}	76	D_{25}	118	A_8	160	$\overline{WARN}$
35	I_{11}	77	D_{26}	119	A_7	161	$\overline{IBACK}$
36	I_{10}	78	D_{27}	120	A_6	162	$\overline{IRDY}$
37	I_9	79	D_{28}	121	A_5	163	$\overline{IERR}$
38	I_8	80	D_{29}	122	A_4	164	$\overline{DERR}$
39	I_7	81	D_{30}	123	A_3	165	$\overline{DBACK}$
40	I_6	82	D_{31}	124	A_2	166	$\overline{PEN}$
41	I_5	83	GND	125	GND	167	$\overline{BREQ}$
42	I_4	84	V_{CC}	126	GND	168	GND

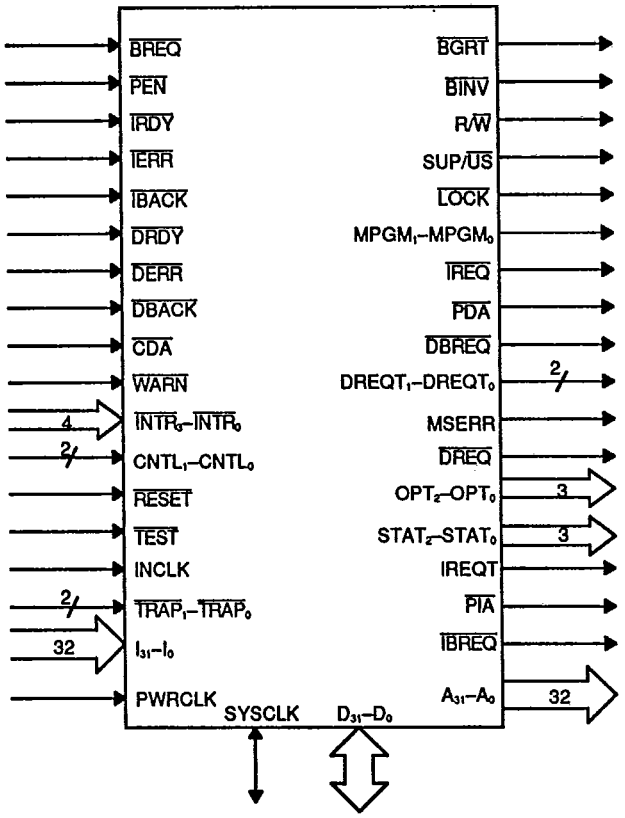
PQFP PIN DESIGNATIONS
(Sorted by Pin Name)

T-49-17-32

Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.
A_0	112	D_4	53	GND	103	INCLK	3
A_1	111	D_5	54	GND	125	$\overline{INTR_0}$	156
A_2	124	D_6	55	GND	126	$\overline{INTR_1}$	157
A_3	123	D_7	56	GND	127	$\overline{INTR_2}$	158
A_4	122	D_8	57	GND	151	$\overline{INTR_3}$	159
A_5	121	D_9	58	GND	153	$\overline{IRDY}$	162
A_6	120	D_{10}	59	GND	168	$\overline{IREQ}$	144
A_7	119	D_{11}	60	I_0	48	IREQT	132
A_8	118	D_{12}	61	I_1	46	$\overline{LOCK}$	139
A_9	117	D_{13}	62	I_2	45	$MPGM_0$	113
A_{10}	110	D_{14}	63	I_3	44	$MPGM_1$	114
A_{11}	109	D_{15}	66	I_4	42	MSERR	136
A_{12}	108	D_{16}	67	I_5	41	OPT_0	128
A_{13}	107	D_{17}	68	I_6	40	OPT_1	129
A_{14}	106	D_{18}	69	I_7	39	OPT_2	130
A_{15}	102	D_{19}	70	I_8	38	$\overline{PDA}$	142
A_{16}	101	D_{20}	71	I_9	37	$\overline{PEN}$	166
A_{17}	100	D_{21}	72	I_{10}	36	$\overline{PIA}$	143
A_{18}	99	D_{22}	73	I_{11}	35	PWRCLK	4
A_{19}	98	D_{23}	74	I_{12}	34	$R/\overline{W}$	140
A_{20}	97	D_{24}	75	I_{13}	33	$\overline{RESET}$	9
A_{21}	96	D_{25}	76	I_{14}	32	$STAT_0$	133
A_{22}	95	D_{26}	77	I_{15}	31	$STAT_1$	134
A_{23}	94	D_{27}	78	I_{16}	30	$STAT_2$	135
A_{24}	93	D_{28}	79	I_{17}	29	$SUP/\overline{US}$	131
A_{25}	92	D_{29}	80	I_{18}	28	SYSCLK	5
A_{26}	91	D_{30}	81	I_{19}	27	$\overline{TEST}$	12
A_{27}	90	D_{31}	82	I_{20}	26	$\overline{TRAP_0}$	154
A_{28}	89	$\overline{DBACK}$	165	I_{21}	25	$\overline{TRAP_1}$	155
A_{29}	88	$\overline{DBREQ}$	146	I_{22}	24	V_{CC}	7
A_{30}	87	$\overline{DERR}$	164	I_{23}	23	V_{CC}	22
A_{31}	86	$\overline{DRDY}$	1	I_{24}	20	V_{CC}	43
$\overline{BGRT}$	145	$\overline{DREQ}$	141	I_{25}	19	V_{CC}	64
$\overline{BINV}$	148	$DREQT_0$	137	I_{26}	18	V_{CC}	84
$\overline{BREQ}$	167	$DREQT_1$	138	I_{27}	17	V_{CC}	104
$\overline{CDA}$	2	GND	6	I_{28}	16	V_{CC}	105
$CNTL_0$	10	GND	8	I_{29}	15	V_{CC}	115
$CNTL_1$	11	GND	21	I_{30}	14	V_{CC}	116
D_0	49	GND	47	I_{31}	13	V_{CC}	149
D_1	50	GND	65	$\overline{IBACK}$	161	V_{CC}	150
D_2	51	GND	83	$\overline{IBREQ}$	147	V_{CC}	152
D_3	52	GND	85	$\overline{IERR}$	163	$\overline{WARN}$	160

LOGIC SYMBOL

T-49-17-32



PIN DESCRIPTION

Although certain outputs are described as being three-state or bidirectional outputs, all outputs (except MSERR) may be placed in a high-impedance state by the Test mode. The three-state and bidirectional terminology in this section is for those outputs (except SYSCLK) that are disabled when the processor grants the channel to another master.

A_{31}–A_0
Address Bus (Three-state Outputs, Synchronous)

The Address Bus transfers the byte address for all accesses except burst-mode accesses. For burst-mode accesses, it transfers the address for the first access in the sequence.

$\overline{BGRT}$
Bus Grant (Output, Synchronous)

This output signals to an external master that the processor is relinquishing control of the channel in response to $\overline{BREQ}$.

$\overline{BINV}$
Bus Invalid (Output, Synchronous)

This output indicates that the address bus and related controls are invalid. It defines an idle cycle for the channel.

$\overline{BREQ}$
Bus Request (Input, Synchronous)

This input allows other masters to arbitrate for control of the processor channel.

$\overline{CDA}$
Coprocessor Data Accept (Input, Synchronous)

This signal allows the coprocessor to indicate the acceptance of operands or operation codes. For transfers to the coprocessor, the processor does not expect a $\overline{DRDY}$ response; an active level on $\overline{CDA}$ performs the function normally performed by $\overline{DRDY}$. $\overline{CDA}$ may be active whenever the coprocessor is able to accept transfers.

$CNTL_1$–$CNTL_0$
CPU Control (Inputs, Asynchronous)

These inputs control the processor mode:

$CNTL_1$	$CNTL_0$	Mode
0	0	Load Test Instruction
0	1	Step
1	0	Halt
1	1	Normal

D_{31}–D_0
Data Bus (Bidirectionals, Synchronous)

The Data Bus transfers data to and from the processor for load and store operations.

$\overline{DBACK}$
Data Burst Acknowledge (Input, Synchronous)

This input is active whenever a burst-mode data access has been established. It may be active even though no data are currently being accessed.

$\overline{DBREQ}$
Data Burst Request (Three-state Output, Synchronous)

This signal is used to establish a burst-mode data access and to request data transfers during a burst-mode data access. $\overline{DBREQ}$ may be active even though the address bus is being used for an instruction access. This signal becomes valid late in the cycle, with respect to $\overline{DREQ}$.

$\overline{DERR}$
Data Error (Input, Synchronous)

This input indicates that an error occurred during the current data access. For a load, the processor ignores the content of the data bus. For a store, the access is terminated. In either case, a Data Access Exception trap occurs. The processor ignores this signal if there is no pending data access.

$\overline{DRDY}$
Data Ready (Input, Synchronous)

For loads, this input indicates that valid data is on the data bus. For stores, it indicates that the access is complete, and that data need no longer be driven on the data bus. The processor ignores this signal if there is no pending data access.

$\overline{DREQ}$
Data Request (Three-state Output, Synchronous)

This signal requests a data access. When it is active, the address for the access appears on the address bus.

$DREQT_1$–$DREQT_0$
Data Request Type (Three-state Outputs, Synchronous)

These signals specify the address space of a data access, as follows (the value "x" is a "don't care"):

$DREQT_1$	$DREQT_0$	Meaning
0	0	Instruction/data memory access
0	1	Input/output access
1	x	Coprocessor transfer

An interrupt/trap vector request is indicated as a data-memory read. If required, the system can identify the vector fetch by the $STAT_2$–$STAT_0$ outputs. $DREQT_1$–$DREQT_0$ are valid only when $\overline{DREQ}$ is active.

I_{31}–I_0
Instruction Bus (Inputs, Synchronous)

The Instruction Bus transfers instructions to the processor.

$\overline{IBACK}$
Instruction Burst Acknowledge (Input, Synchronous)

This input is active whenever a burst-mode instruction access has been established. It may be active even though no instructions are currently being accessed.

$\overline{IBREQ}$
Instruction Burst Request (Three-state Output, Synchronous)

This signal is used to establish a burst-mode instruction access and to request instruction transfers during a burst-mode instruction access. $\overline{IBREQ}$ may be active even though the address bus is being used for a data access. This signal becomes valid late in the cycle with respect to $\overline{IREQ}$.

$\overline{IERR}$
Instruction Error (Input, Synchronous)

This input indicates that an error occurred during the current instruction access. The processor ignores the content of the instruction bus, and an Instruction Access Exception trap occurs if the processor attempts to execute the invalid instruction. The processor ignores this signal if there is no pending instruction access.

INCLK
Input Clock (Input)

When the processor generates the clock for the system, this is an oscillator input to the processor at twice the processor's operating frequency. In systems where the clock is not generated by the processor, this signal must be tied High or Low, except in certain master/slave configurations.

$\overline{INTR_3}$–$\overline{INTR_0}$
Interrupt Request (Inputs, Asynchronous)

These inputs generate prioritized interrupt requests. The interrupt caused by $\overline{INTR_0}$ has the highest priority, and the interrupt caused by $\overline{INTR_3}$ has the lowest priority. The interrupt requests are masked in prioritized order by the Interrupt Mask field in the Current Processor Status Register.

$\overline{IRDY}$
Instruction Ready (Input, Synchronous)

This input indicates that a valid instruction is on the instruction bus. The processor ignores this signal if there is no pending instruction access.

$\overline{IREQ}$
Instruction Request (Three-state Output, Synchronous)

This signal requests an instruction access. When it is active, the address for the access appears on the address bus.

IREQT
Instruction Request Type (Three-state Output, Synchronous)

This signal specifies the address space of an instruction request when $\overline{IREQ}$ is active:

IREQT	Meaning
0	Instruction/data memory access
1	Instruction read-only memory access

$\overline{LOCK}$
Lock (Three-state Output, Synchronous)

This output allows the implementation of various channel and device interlocks. It may be active only for the duration of an access, or active for an extended period of time under control of the Lock bit in the Current Processor Status.

MSERR
Master/Slave Error (Output, Synchronous)

This output shows 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, this line is asserted.

$MPGM_1$–$MPGM_0$
MMU Programmable (Three-state Outputs, Synchronous)

These outputs have no function in the Am29005 microprocessor, and always are driven Low on an access. They are defined to ensure pin compatibility with the Am29000™ microprocessor.

OPT_2–OPT_0
Option Control
(Three-state Outputs, Synchronous)

These outputs reflect the value of bits 18–16 of the load or store instruction that begins an access. Bit 18 of the instruction is reflected on OPT_2, bit 17 on OPT_1, and bit 16 on OPT_0.

The standard definitions of these signals (based on DREQT) are as follows (the value "x" is a "don't care"):

$DREQT_1$	$DREQT_0$	OPT_2	OPT_1	OPT_0	Meaning
0	x	0	0	0	Word-length access
0	x	0	0	1	Byte access
0	x	0	1	0	Half-word access
0	0	1	0	0	Instruction ROM access (as data)
0	0	1	0	1	Cache control
0	0	1	1	0	Hardware-development system accesses
-all others-					Reserved

During an interrupt/trap vector fetch, the OPT_2–OPT_0 signals indicate a word-length access (000). Also, the system should return an entire aligned word for a read, regardless of the indicated data length.

The Am29005 microprocessor does not explicitly prevent a store to the instruction ROM. OPT_3–OPT_0 are valid only when $\overline{DREQ}$ is active.

$\overline{PDA}$
Pipelined Data Access
(Three-state Output, Synchronous)

If $\overline{DREQ}$ is not active, this output indicates that a data access is pipelined with another in-progress data access. The indicated access cannot be completed until the first access is complete. The completion of the first access is signaled by the assertion of $\overline{DREQ}$.

$\overline{PEN}$
Pipeline Enable (Input, Synchronous)

This signal allows devices that can support pipelined accesses (i.e., that have input latches for the address and required controls) to signal that a second access may begin while the first is being completed.

$\overline{PIA}$
Pipelined Instruction Access
(Three-state Output, Synchronous)

If $\overline{IREQ}$ is not active, this output indicates that an instruction access is pipelined with another in-progress instruction access. The indicated access cannot be completed until the first access is complete. The completion of the first access is signaled by the assertion of $\overline{IREQ}$.

R/$\overline{W}$
Read/Write (Three-state Output, Synchronous)

This signal indicates whether data is being transferred from the processor to the system, or from the system to the processor. R/$\overline{W}$ is valid only when the address bus is valid. R/$\overline{W}$ will be High when $\overline{IREQ}$ is active.

$\overline{RESET}$
Reset (Input, Asynchronous)

This input places the processor in the Reset mode.

$STAT_2$–$STAT_0$
CPU Status (Outputs, Synchronous)

These outputs indicate the state of the processor's execution stage on the previous cycle. They are encoded as follows:

$STAT_2$	$STAT_1$	$STAT_0$	Condition
0	0	0	Halt or Step Modes
0	0	1	Pipeline Hold Mode
0	1	0	Load Test Instruction Mode, Halt/Freeze
0	1	1	Wait Mode
1	0	0	Interrupt Return
1	0	1	Taking Interrupt or Trap
1	1	0	Non-sequential Instruction Fetch
1	1	1	Executing Mode

SUP/$\overline{US}$
Supervisor/User Mode
(Three-state Output, Synchronous)

This output indicates the program mode for an access.

The processor does not relinquish the channel (in response to $\overline{BREQ}$) when $\overline{LOCK}$ is active.

SYSCLK
System Clock (Bidirectional)

This is either a clock output with a frequency that is half that of INCLK, or an input from an external clock generator at the processor's operating frequency.

$\overline{TEST}$
Test Mode (Input, Asynchronous)

When this input is active, the processor is in Test mode. All outputs and bidirectional lines, except MSERR, are forced to the high-impedance state.

$\overline{TRAP_1}$–$\overline{TRAP_0}$
Trap Request (Inputs, Asynchronous)

These inputs generate prioritized trap requests. The trap caused by $\overline{TRAP_0}$ has the highest priority. These trap requests are disabled by the DA bit of the Current Processor Status Register.

WARN

Warn (Input, Asynchronous, Edge-sensitive)

A High-to-Low transition on this input causes a non-maskable $\overline{\text{WARN}}$ trap to occur. This trap bypasses the normal trap vector fetch sequence, and is useful in situations where the vector fetch may not work (e.g., when data memory is faulty).

The following pin is not a signal pin, but is named in Am29005 microprocessor documentation because of its special role in the processor and system.

PWRCLK

Power Supply for SYSCLK Driver

This pin is a power supply for the SYSCLK output driver. It isolates the SYSCLK driver, and is used to determine whether or not the Am29005 microprocessor generates the clock for the system. If power (+5 V) is applied to this pin, the Am29005 microprocessor generates a clock on the SYSCLK output. If this pin is grounded, the Am29005 microprocessor accepts a clock generated by the system on the SYSCLK input.

FUNCTIONAL DESCRIPTION

Product Overview

The Am29005 microprocessor contains a high-function execution unit, a large register file (192 locations), and a high-bandwidth, pipelined external channel with separate instruction and data buses. The flexible register file may be used as a cache for run-time variables during program execution, or as a collection of register banks allocated to separate tasks in multitasking applications.

The Am29005 microprocessor provides a significant margin of performance over other processors designed for cost-sensitive situations, since the majority of processor features were defined with the maximum achievable performance in mind. This section describes the features of the Am29005 microprocessor from the point of view of system performance.

Cycle Time

The processor operates at a frequency of 16 MHz. The processor cycle time is a single, 62.5-ns clock period. The processor interface drivers can drive 80-pF loads at this frequency (for loads greater than 80 pF, refer to the Capacitive Output Delay table).

The Am29005 microprocessor architecture and system interfaces are designed so that the processor cycle time can decrease with technology improvements.

Four-Stage Pipeline

The Am29005 microprocessor utilizes a four-stage pipeline, allowing it to execute one instruction every clock cycle. The processor can complete an instruction on every cycle, even though four cycles are required from the beginning of an instruction to its completion.

At the 16-MHz operating frequency, the maximum instruction execution rate is 16 million instructions per second (MIPS). The Am29005 microprocessor pipeline is designed so that the Am29005 microprocessor can operate at the maximum instruction execution rate a significant portion of the time.

Pipeline interlocks are implemented by processor hardware. Except for a few special cases, it is not necessary to rearrange programs to avoid pipeline dependencies.

System Interface

The Am29005 microprocessor accesses external instructions and data using three non-multiplexed buses. These buses are referred to collectively as the channel. The channel protocol minimizes the logic chains involved in a transfer, and provides a maximum transfer rate of 128 Mb/s.

Separate Address, Instruction, and Data Buses

The Am29005 microprocessor incorporates two 32-bit buses for instruction and data transfers, and a third address bus that is shared between instruction and data accesses. This bus structure allows simultaneous instruction and data transfers, even though the address bus is shared. The channel achieves the performance of four separate 32-bit buses at a much-reduced pin count.

Pipelined Addresses

The Am29005 microprocessor address bus is pipelined so that it can be released before an instruction or data transfer is completed. This allows a subsequent access to begin before the first has been completed, and allows the processor to have two accesses in progress simultaneously.

Support of Burst Devices and Memories

Burst-mode accesses provide high transfer rates for instructions and data at sequential addresses. For such accesses, the address of the first instruction or datum is sent, and subsequent requests for instructions or data at sequential addresses do not require additional address transfers. These instructions or data are transferred until either party involved in the transfer terminates the access.

Burst-mode accesses can occur at the rate of one access per cycle after the first address has been processed. At 16 MHz, the maximum achievable transfer bandwidth for either instructions or data is 64 Mb/s.

Burst-mode accesses may occur to input/output devices if the system design permits.

Interface to Fast Devices and Memories

The processor can be interfaced to devices and memories that complete accesses within one cycle. The channel protocol takes maximum advantage of such devices and memories by allowing data to be returned to the processor during the cycle in which the address is transmitted. This allows a full range of memory-speed tradeoffs to be made within a particular system.

Register File

An on-chip Register File containing 192 general-purpose registers allows most instruction operands to be fetched without the delay of an external access. The Register File incorporates several features that aid the retention of data required by an executing program. Because of the number of general-purpose registers, the frequency of external references for the Am29005 microprocessor is significantly lower than the frequency of references in processors having only 16 or 32 registers.

Triple-port access to the Register File allows two source operands to be fetched in one cycle while a previously computed result is written. Three 32-bit internal buses prevent contention in the routing of operands. All operand fetches and result write-backs for instruction execution can be performed in a single cycle.

The registers allow efficient procedure linkage by caching a portion of a compiler's run-time stack. On the average, procedure calls and returns can be executed 5 to 10 times faster (on a cycle-by-cycle basis) than in

processors that require the implementation of a run-time stack in external memory (with the attendant loading and storing of registers on procedure call and return).

The registers can contain variables, constants, addresses, and operating-system values. In multitasking applications, they can be used to hold the processor status and variables for as many as eight different tasks. A register-banking option allows the Register File to be divided into segments, which can be individually protected. In this configuration, a task switch can occur in as few as 17 cycles.

Instruction Execution

The Am29005 microprocessor uses an Arithmetic/Logic Unit, a Field Shift Unit, and a Prioritizer to execute most instructions. Each of these is organized to operate on 32-bit operands and provide a 32-bit result. All operations are performed in a single cycle.

Instruction operations are overlapped with operand fetch and result write-back to the Register File. Pipeline forwarding logic detects pipeline dependencies and routes data as required, avoiding delays that might arise from these dependencies.

Branching

Branch conditions in the Am29005 microprocessor are based on Boolean data contained in general-purpose registers rather than on arithmetic condition codes. Using a condition-code register for the purpose of branching—which is common in other processors—inhibits certain compiler optimizations because the condition-code register is modified by many different instructions. It is difficult for an optimizing compiler to schedule this shared use. By treating branch conditions as any other instruction operand, the Am29005 microprocessor avoids this problem.

Loads and Stores

The performance degradation of load and store operations is minimized in the Am29005 microprocessor by overlapping them with instruction execution, by taking advantage of pipelining, and by organizing the flow of external data onto the processor so that the impact of external accesses is minimized.

Overlapped Loads and Stores

In the Am29005 microprocessor, a load or store is performed concurrently with execution of instructions that do not have dependencies on the load or store operation. An optimizing compiler can schedule loads and stores in the instruction sequence so that, in most cases, data accesses are overlapped with instruction execution.

Overlapped load and store operations can achieve up to a 30% improvement in performance when data memory has a two-cycle access time. Processor hardware detects dependencies while overlapped loads and stores are being performed, so dependencies have no software implications.

The Am29005 microprocessor exception restart mechanism automatically saves information required to restart any load or store until the operation is successfully completed. Thus, it allows the overlapped execution of loads and stores while properly handling address-translation exceptions.

The Am29005 microprocessor data-flow organization avoids the one-cycle penalty that would result from the contention between load data and the results of overlapped instruction execution. Load data is buffered in a latch while awaiting an opportunity to be written into the register file. This opportunity is guaranteed to arise before the next load is executed. While the data is buffered in this latch, it may be used as an instruction operand in place of the destination register for the load.

Load Multiple and Store Multiple

Load Multiple and Store Multiple instructions allow the transfer of the contents of multiple registers to or from external memories or devices. This transfer can occur at a rate of one register content per cycle.

The advantage of Load Multiple and Store Multiple is best seen in task switching, in register-file saving and restoring, and in block data moves. In many systems, such operations require a significant percentage of execution time.

The Load Multiple and Store Multiple sequences are interruptible so that they do not affect interrupt latency.

Forwarding of Load Data

Data that are sent to the processor at the completion of a load are forwarded directly to the appropriate execution unit if the data are required immediately by an instruction. This avoids the common one-cycle delay from bus transfer to use of data, and reduces the access latency of external data by one cycle.

Interrupts and Traps

When the Am29005 microprocessor takes an interrupt or trap, it does not automatically save its current state information. This greatly improves the performance of temporary interruptions such as floating-point emulation or other simple operating-system calls that require no saving of state information.

In cases where the processor state must be saved, the saving and restoring of state information is under the control of software. The methods and data structures used to handle interrupts—and the amount of state saved—may be tailored to the needs of a particular system.

Interrupts and traps are dispatched through a 256-entry Vector Area, which directs the processor to a routine to handle a given interrupt or trap. The Vector Area may be

relocated in memory by the modification of a processor register. There may be multiple Vector Areas in the system, though only one is active at any given time.

The Vector Area is either a table of pointers to the interrupt and trap handlers, or a segment of instruction memory (possibly read-only memory) containing the handlers themselves. The choice between the two possible Vector Area definitions is determined by the cost/performance trade-offs made for a particular system.

If the Vector Area is a table of vectors in data memory, it requires only 1 kb of memory. However, this structure requires that the processor perform a vector fetch every time an interrupt or trap is taken. The vector fetch requires at least three cycles in addition to the number of cycles required for the basic memory access.

If the Vector Area is a segment of instruction memory, it requires a maximum of 64 kb of memory. The advantage of this structure is that the processor begins the execution of the interrupt or trap handler in the minimum amount of time.

Floating-Point Arithmetic Unit

The Am29027 arithmetic accelerator is a double-precision, floating-point arithmetic unit for the Am29005 microprocessor. It can provide an order-of-magnitude performance increase over floating-point operations performed in software. It performs both single-precision and double-precision operations using IEEE and other floating-point formats. The Am29027 arithmetic accelerator also supports 32- and 64-bit integer functions.

The Am29027 arithmetic accelerator performs floating-point operations using combinatorial—rather than sequential—logic; therefore, operations with the Am29027 arithmetic accelerator require only five Am29005 microprocessor cycles. Floating-point operations may be overlapped with other processor operations. Furthermore, the Am29027 arithmetic accelerator incorporates pipeline registers and eight operand registers, permitting very high throughput for certain types of operations (such as array computations).

The Am29027 arithmetic accelerator attaches directly to the Am29005 microprocessor using the coprocessor interface. The Am29005 microprocessor can transfer two 32-bit quantities to the Am29027 arithmetic accelerator in one cycle.

The Am29027 arithmetic accelerator is described in detail in the Am29027 Arithmetic Accelerator Data Sheet (order #09114) and the Am29027 Handbook (order #11852).

ARCHITECTURE HIGHLIGHTS

This section introduces the principle architectural elements, hardware features, and system interfaces of the Am29005 microprocessor.

Architecture Overview

This section gives a brief description of the Am29005 microprocessor from a programmer's point of view. It introduces the processor's program modes, registers, and instructions. An overview of the processor's data formats and handling is given. This section also briefly describes interrupts and traps, and the coprocessor interface. Finally, the Timer Facility and Trace Facility are introduced.

Program Modes

There are two mutually exclusive modes of program execution: the Supervisor mode and the User mode. In the Supervisor mode, executing programs have access to all processor resources. In the User mode, certain processor resources may not be accessed; any attempted access causes a trap.

Visible Registers

The Am29005 microprocessor incorporates two classes of registers that are accessed and manipulated by instructions: general-purpose registers, and special-purpose registers. (Refer to the Register Description section for greater detail of the register categories.)

General-Purpose Registers

The Am29005 microprocessor has 192 general-purpose registers. With a few exceptions, general-purpose registers are not dedicated to any special use and are available for any appropriate program use.

Most processor instructions are three-address instructions. An instruction specifies any three of the 192 registers for use in instruction execution. Normally, two of these registers contain source operands for the instruction, and a third stores the result of the instruction.

The 192 registers are divided into 64 global and 128 local registers. Global registers are addressed with absolute register numbers, while local registers are addressed relative to an internal Stack Pointer.

For fast procedure calling, a portion of a compiler's run-time stack can be mapped into the local registers. Statically allocated variables, temporary values, and operating-system parameters are kept in the global registers.

The Stack Pointer for local registers is mapped to Global Register 1. The Stack Pointer is a full 32-bit virtual address for the top of the run-time stack.

The general-purpose registers may be accessed indirectly, with the register number specified by the content of a special-purpose register (see below) rather than by an instruction field. Three independent indirect register numbers are contained in three separate special-purpose registers. Indirect addressing is accomplished by specifying Global Register 0 as an

instruction operand or result register. An instruction can specify an indirect register access for any or all of the source operands or result.

General-purpose registers may be partitioned into segments of 16 registers for the purpose of access protection. A register in a protected segment may be accessed only by a program executing in the Supervisor mode. An attempted access (either read or write) by a User-mode program causes a trap to occur.

Special-Purpose Registers

The Am29005 microprocessor contains 25 special-purpose registers. These registers provide controls and data for certain processor functions.

Special-purpose registers are accessed by data movement only. Any special-purpose register can be written with the contents of any general-purpose register, and any general-purpose register can be written with the contents of any special-purpose register. Operations cannot be performed directly on the contents of special-purpose registers.

Some special-purpose registers are protected, and can be accessed only in the Supervisor mode. This restriction applies to both read and write accesses. An attempt by a User-mode program to access a protected register causes a trap to occur.

The protected special-purpose registers are defined as follows:

1. Vector Area Base Address—Defines the beginning of the Interrupt/trap Vector Area.
2. Old Processor Status—Receives a copy of the Current Processor Status (see below) when an interrupt or trap is taken. It is later used to restore the Current Processor Status on an Interrupt return.
3. Current Processor Status—Contains control information associated with the currently executing process, such as interrupt disables and the Supervisor Mode bit.
4. Configuration—Contains control information that normally varies only from system to system, and usually is set only during system initialization.
5. Channel Address—Contains the address associated with an external access, and retains the address if the access is not completed successfully. The Channel Address Register, in conjunction with the Channel Data and Channel Control registers described below, allows the restarting of unsuccessful external accesses. This might be necessary for an access encountering a page fault in a demand-paged environment, for example.
6. Channel Data—Contains data associated with a store operation, and retains the data if the operation is not completed successfully.
7. Channel Control—Contains control information associated with a channel operation, and retains this information if the operation is not completed successfully.
8. Register Bank Protect—Restricts access of user-mode programs to specified groups of 16 registers. This facilitates register banking for multitasking applications, and protects operating system parameters kept in the global registers from corruption by user-mode programs.
9. Timer Counter—Supports real-time control and other timing-related functions.
10. Timer Reload—Maintains synchronization of the Timer Counter. It includes control bits for the Timer Facility.
11. Program Counter 0—Contains the address of the instruction being decoded when an interrupt or trap is taken. The processor restarts this instruction upon interrupt return.
12. Program Counter 1—Contains the address of the instruction being executed when an interrupt or trap is taken. The processor restarts this instruction upon interrupt return.
13. Program Counter 2—Contains the address of the instruction just completed when an interrupt or trap is taken. This address is provided for information only, and does not participate in an interrupt return.

The unprotected special-purpose registers are defined as follows:

1. Indirect Pointer C—Allows the indirect access of a general-purpose register.
2. Indirect Pointer A—Allows the indirect access of a general-purpose register.
3. Indirect Pointer B—Allows the indirect access of a general-purpose register.
4. Q—Provides additional operand bits for multiply step, divide step, and divide operations.
5. ALU Status—Contains information about the outcome of integer arithmetic and logical operations, and holds residual control for certain instruction operations.
6. Byte Pointer—Contains an index of a byte or half-word within a word. This register is also accessible via the ALU Status Register.

7. Funnel Shift Count—Provides a bit offset for the extraction of word-length fields from double-word operands. This register is also accessible via the ALU Status Register.
8. Load/Store Count Remaining—Maintains a count of the number of loads and stores remaining for Load Multiple and Store Multiple operations. The count is initialized to the total number of loads or stores to be performed before the operation is initiated. This register is also accessible via the Channel Control Register.
9. Floating-Point Environment—Controls the operation of floating-point arithmetic, such as rounding modes and exception reporting.
10. Integer Environment—Enables and disables the reporting of exceptions that occur during integer multiply and divide operations.
11. Floating-Point Status—Contains information about the outcome of floating-point operations.
12. Exception Opcode—Reports the operation code of an instruction causing a trap. This register is provided primarily for recovery from floating-point exceptions, but is also set for other instructions that cause traps.

Instruction Set Overview

The three-address architecture of the Am29005 microprocessor instruction set allows a compiler or assembly-language programmer to prevent the destruction of operands, and aids register allocation and operand reuse. Instruction operands may be contained in any 2 of the 192 general-purpose registers, and instruction results may be stored in any of the 192 general-purpose registers.

The compiler or assembly-language programmer has complete freedom to allocate register usage. There is no dedication of a particular register or register group to a particular class of operations. The instruction set is designed to minimize the number of side effects and implicit operations of instructions.

Most Am29005 microprocessor instructions can specify an 8-bit constant as one of the source operands. Larger constants are constructed using one or two additional instructions and a general-purpose register. Relative branch instructions specify a 16-bit, signed, word offset. Absolute branches specify a 16-bit word address.

The Am29005 microprocessor instruction set contains 113 instructions. These instructions are divided into nine classes:

1. Integer Arithmetic—Perform integer add, subtract, multiply, and divide operations.
2. Compare—Perform arithmetic and logical comparisons. Some instructions in this class allow the generation of a trap if the comparison condition is not met.
3. Logical—Perform a set of bit-wise Boolean operations.
4. Shift—Perform arithmetic and logical shifts, and allow the extraction of 32-bit words from 64-bit double words.
5. Data Movement—Perform movement of data fields between registers, and the movement of data to and from external devices and memories.
6. Constant—Allow the generation of large constant values in registers.
7. Floating-Point—Included for floating-point arithmetic, comparisons, and format conversions. These instructions are not currently implemented directly in processor hardware.
8. Branch—Perform program jumps and subroutine calls.
9. Miscellaneous—Perform miscellaneous control functions and operations not provided by other classes.

The Am29005 microprocessor executes all instructions in a single cycle, except for interrupt returns, Load Multiple, and Store Multiple.

Figure 1 shows a complete list of Am29005 microprocessor instructions, listed alphabetically by instruction mnemonic (refer to the Instruction Set section for more detail).

Mnemonic	Instruction Name
ADD	Add
ADDC	Add with Carry
ADDCS	Add with Carry, Signed
ADDCU	Add with Carry, Unsigned
ADDS	Add, Signed
ADDU	Add, Unsigned
AND	AND Logical
ANDN	AND-NOT Logical
ASEQ	Assert Equal To
ASGE	Assert Greater Than or Equal To
ASGEU	Assert Greater Than or Equal To, Unsigned
ASGT	Assert Greater Than
ASGTU	Assert Greater Than, Unsigned
ASLE	Assert Less Than or Equal To
ASLEU	Assert Less Than or Equal To, Unsigned
ASLT	Assert Less Than
ASLTU	Assert Less Than, Unsigned
ASNEQ	Assert Not Equal To
CALL	Call Subroutine
CALLI	Call Subroutine, Indirect
CLASS	Classify Floating-Point Operand
CLZ	Count Leading Zeros
CONST	Constant
CONSTH	Constant, High
CONSTN	Constant, Negative
CONVERT	Convert Data Format
CPBYTE	Compare Bytes
CPEQ	Compare Equal To
CPGE	Compare Greater Than or Equal To
CPGEU	Compare Greater Than or Equal To, Unsigned
CPGT	Compare Greater Than
CPGTU	Compare Greater Than, Unsigned
CPLE	Compare Less Than or Equal To
CPLEU	Compare Less Than or Equal To, Unsigned
CPLT	Compare Less Than
CPLTU	Compare Less Than, Unsigned
CPNEQ	Compare Not Equal To
DADD	Floating-Point Add, Double-Precision
DDIV	Floating-Point Divide, Double-Precision
DEQ	Floating-Point Equal To, Double-Precision
DGE	Floating-Point Greater Than or Equal To, Double-Precision
DGT	Floating-Point Greater Than, Double-Precision
DIV	Divide Step
DIV0	Divide Initialize
DIVIDE	Integer Divide, Signed
DIVIDU	Integer Divide, Unsigned
DIVL	Divide Last Step
DIVREM	Divide Remainder
DMUL	Floating-Point Multiply, Double-Precision
DSUB	Floating-Point Subtract, Double-Precision
EMULATE	Trap to Software Emulation Routine
EXBYTE	Extract Byte
EXHW	Extract Half-Word
EXHWS	Extract Half-Word, Sign-Extended
EXTRACT	Extract Word, Bit-Aligned
FADD	Floating-Point Add, Single-Precision
FDIV	Floating-Point Divide, Single-Precision
FDMUL	Floating-Point Multiply, Single-to-Double Precision
FEQ	Floating-Point Equal To, Single-Precision
FGE	Floating-Point Greater Than or Equal To, Single-Precision

Figure 1. Am29005 Microprocessor Instruction Set

Mnemonic	Instruction Name
FGT	Floating-Point Greater Than, Single-Precision
FMUL	Floating-Point Multiply, Single-Precision
FSUB	Floating-Point Subtract, Single-Precision
HALT	Enter Halt Mode
INBYTE	Insert Byte
INHW	Insert Half-Word
IRET	Interrupt Return
JMP	Jump
JMPF	Jump False
JMPFDEC	Jump False and Decrement
JMPFI	Jump False Indirect
JMPI	Jump Indirect
JMPT	Jump True
JMPTI	Jump True Indirect
LOAD	Load
LOADL	Load and Lock
LOADM	Load Multiple
LOADSET	Load and Set
MFSR	Move from Special Register
MTSR	Move to Special Register
MTSRIM	Move to Special Register Immediate
MUL	Multiply Step
MULL	Multiply Last Step
MULTIPLU	Integer Multiply, Unsigned
MULTIPLY	Integer Multiply, Signed
MULTM	Integer Multiply Most-Significant Bits, Signed
MULTMU	Integer Multiply Most-Significant Bits, Unsigned
MULU	Multiply Step, Unsigned
NAND	NAND Logical
NOR	NOR Logical
OR	OR Logical
SETIP	Set Indirect Pointers
SLL	Shift Left Logical
SQRT	Square Root
SRA	Shift Right Arithmetic
SRL	Shift Right Logical
STORE	Store
STOREL	Store and Lock
STOREM	Store Multiple
SUB	Subtract
SUBC	Subtract with Carry
SUBCS	Subtract with Carry, Signed
SUBCU	Subtract with Carry, Unsigned
SUBR	Subtract Reverse
SUBRC	Subtract Reverse with Carry
SUBRCS	Subtract Reverse with Carry, Signed
SUBRCU	Subtract Reverse with Carry, Unsigned
SUBRS	Subtract Reverse, Signed
SUBRU	Subtract Reverse, Unsigned
SUBS	Subtract Signed
SUBU	Subtract Unsigned
XNOR	Exclusive-NOR Logical
XOR	Exclusive-OR Logical

Figure 1. Am29005 Microprocessor Instruction Set (continued)

Data Formats and Handling

This section introduces the data formats and data-manipulation mechanisms that are supported by the Am29005 microprocessor.

Data Types

A word is defined as 32 bits of data. A half-word consists of 16 bits, and a double word consists of 64 bits. Bytes are 8 bits in length. The Am29005 microprocessor has direct support for word-integer (signed and unsigned), word-logical, word-Boolean, half-word integer (signed and unsigned), and character (signed and unsigned) data.

Other data types, such as character strings, are supported with sequences of basic instructions and/or external hardware. Single- and double-precision floating-point types are defined for the Am29005 microprocessor, but are not supported directly by hardware.

The format for Boolean data used by the processor is such that the Boolean values TRUE and FALSE are represented by 1 and 0, respectively, in the most-significant bit of a word.

Figure 2 illustrates the numbering conventions for data units contained in a word. Within a word, bits are numbered in increasing order from right to left, starting with the number 0 for the least-significant bit. Bytes and half-words within a word are numbered in increasing order, starting with the number 0. However, bytes and half-words may be numbered right-to-left or left-to-right, as controlled by the Configuration Register.

Note that the numbering of bits within words is strictly for notational convenience. In contrast, the numbering conventions for bytes and half-words within words affect processor operations.

External Data Accesses

External accesses move data between the processor and external devices and memories. These accesses occur only as a result of load and store instructions.

Load and store instructions move words of data to and from general-purpose registers. Each load and store instruction moves a single word. There are load and store instructions that support interlocking operations necessary for multiprocessor exclusion, synchronization, and communication.

For the movement of multiple words, Load Multiple and Store Multiple instructions move the contents of sequentially addressed external locations to or from sequentially numbered general-purpose registers. The Load Multiple and Store Multiple allow the movement of up to 192 words at a maximum rate of one word per processor cycle. The multiple load and store sequences may be interrupted, and restarted at the point of interruption.

Load and store instructions provide no mechanism for computing the address associated with the external data access. All addresses are contained in a general-purpose register at the beginning of the access, or are given by an 8-bit instruction constant. Any address computation must be performed explicitly before the load or store instruction is executed. Since address computations are expressed directly, they are exposed for compiler optimizations as any other computations are.

External data accesses are overlapped with instruction execution. Processor performance is improved if instructions that follow loads do not immediately use externally referenced data. In this manner, the time required to perform the external access is overlapped with subsequent instruction execution. Because of hardware interlocks, this concurrency has no effect on the logical behavior of an executing program.

Addressing and Alignment

External instructions and data are contained in one of four 32-bit address spaces:

1. Instruction/Data Memory
2. Input/Output
3. Coprocessor
4. Instruction Read-Only Memory (Instruction ROM)

Bits contained within load and store instructions distinguish between the instruction/data memory, input/output, and coprocessor address spaces. The Current Processor Status register determines whether instruction accesses are directed to the instruction/data memory address space or to the instruction ROM address space.

The Am29005 microprocessor does not support data accesses directly to the instruction ROM address space. However, this capability is possible as a system option.

All addresses are interpreted as byte addresses, although accesses are word-oriented. The number of a byte within a word is given by the two least-significant address bits. The number of a half-word within a word is given by the next-to-least-significant address bit.

Since only byte addressing is supported, it is possible that an address for the access of a word or half-word is not aligned to the desired word or half-word. For a word access, an unaligned address has a 1 in either or both of the 2 least-significant address bits. For a half-word access, an unaligned address has a 1 in the least-significant address bit. In many systems, address alignment can be ignored, with addresses truncated to access the word or half-word of interest. However, as a user option, the Am29005 microprocessor creates a trap when a non-aligned access is attempted. The trap allows software emulation of nonaligned accesses.

In the Am29005 microprocessor, all instructions are 32 bits in length, and are aligned on word-address boundaries.

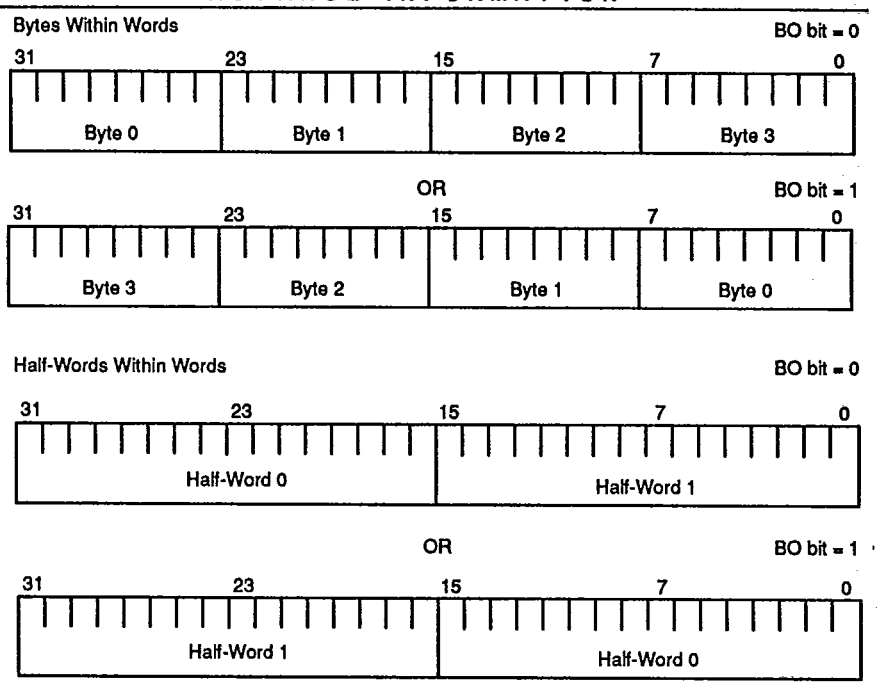


Figure 2. Data-Unit Numbering Conventions

Byte and Half-Word Accesses

The Am29005 microprocessor supports the direct external access of bytes and half-words as an option. If this option is enabled, the Am29005 microprocessor selects a byte or half-word within a word on a load, and aligns it to the low-order byte or half-word of a register. On a store, the low-order byte or half-word of a register is replicated in all byte or half-word positions, so that the external memory can easily write the required byte or half-word in memory. This option requires that the external memory system be able to write individual bytes and half-words within words.

To avoid the memory-system complexity caused by writing individual bytes and half-words, the Am29005 microprocessor can perform byte and half-word accesses using software alone. The Am29005 microprocessor can set a byte-position indicator in the ALU Status Register as an option for load instructions, with the two least-significant bits of the address for the load. To load a byte or half-word, a word load is first performed. This load sets the byte-position indicator, and a subsequent instruction extracts the byte or half-word of interest from the accessed word. To store a byte or half-word, a load is also first performed; the byte or half-word of interest is inserted into the accessed word, and the resulting word then is stored. Even if the Am29005 microprocessor is configured to perform byte and half-word accesses in hardware, this software-only technique operates correctly; this allows software to be upwardly compatible from simpler systems to more complex systems.

Interrupts and Traps

Normal program flow may be preempted by an interrupt or trap for which the processor is enabled. The effect on the processor is identical for interrupts and traps; the distinction is in the different mechanisms by which interrupts and traps are enabled. It is intended that interrupts be used for suspending current program execution and causing another program to execute, while traps are used to report errors and exceptional conditions.

The interrupt and trap mechanism supports high-speed, temporary context switching and user-defined interrupt-processing mechanisms.

Temporary Context Switching

The basic interrupt/trap mechanism of the Am29005 microprocessor supports temporary context switching. During the temporary context switch, the interrupted context is held in processor registers. The interrupt or trap handler can return immediately to this context.

Temporary context switching is useful for instruction emulation, floating-point operations, and so forth. Many of its features are similar to microprogram execution; processor context does not have to be saved, interrupts are disabled for the duration of the program, and all processor resources are accessible, even if the context

that was interrupted is in the User mode. The associated routine may execute from instruction/data memory or instruction ROM.

User-Defined Interrupt Processing

Since the basic interrupt/trap mechanism for the Am29005 microprocessor keeps the interrupted context in the processor, dynamically nested interrupts are not supported directly. The context in the processor must be saved before another interrupt or trap can be taken.

The interrupt or trap handler executing during a temporary context switch is not required to return to the interrupted context. This routine optionally may save the interrupted context, load a new one, and return to the new context.

The implementation of the saving and restoring of contexts is completely user-defined. Thus, the context save/restore mechanism used (e.g., interrupt stack, program status word area, etc.) and the amount of context saved may be tailored to the needs of the system.

Vector Area

Interrupt and trap dispatching occur through a relocatable Vector Area, which accommodates as many as 256 interrupt and trap handling routines. Entries into the Vector Area are associated with various sources of interrupts and traps; some are predefined while others are user-defined.

The Vector Area is either a table of vectors in data memory where each vector points to the beginning of an interrupt or trap handler, or it is a segment of instruction/data memory (or instruction ROM) containing the actual routines. The latter configuration for the Vector Area yields better interrupt performance with the cost of additional memory.

Coprocessor Programming

The coprocessor interface for the Am29005 microprocessor allows a program to communicate with an off-chip coprocessor for performing operations not supported by processor hardware directly.

The coprocessor interface allows the program to transfer operands and operation codes to the coprocessor, and then perform other operations while the coprocessor operation is in progress. The results of the operation are read from the coprocessor by a separate transfer. The processor may transfer multiple operands to the coprocessor without retransferring operation codes or reading intermediate results. As many as 64 bits of information can be transferred to the coprocessor in a single cycle.

The Am29005 microprocessor includes features that support the definition of the coprocessor as a system option. In this case, coprocessor operations are emulated by software when the coprocessor is not present in a system.

Timer Facility

The Timer Facility provides a counter for implementing a real-time clock or other software timing functions. This facility comprises two special-purpose registers: the Timer Counter Register, which decrements at a rate equal to the processor operating frequency, and the Timer Reload Register, which reinitializes the Timer Counter Register when it decrements to 0. The Timer Facility optionally may create an interrupt when the Timer Counter decrements to 0.

Trace Facility

The Trace Facility allows a debug program to emulate single-instruction stepping in a program under test. This facility allows a trap to be generated after the execution of any instruction in the program being tested.

Using the Trace Facility, the debug program can inspect and modify the state of the program at every instruction boundary. The Trace Facility is designed to work properly in the presence of normal system interrupts and traps.

FUNCTIONAL OPERATION

This section briefly describes the operation of Am29005 microprocessor hardware. It introduces the processor pipeline and the two major internal functional units: the Instruction Fetch Unit, and the Execution Unit. Finally, the processor's operational modes are described.

Four-Stage Pipeline

The Am29005 microprocessor implements a four-stage pipeline for instruction execution. The four stages are: fetch, decode, execute, and write-back. The pipeline is organized so that the effective instruction execution rate is as high as one instruction per cycle. Data forwarding and pipeline interlocks are handled by processor hardware.

Fetch Stage

During the fetch stage, the Instruction Fetch Unit determines the location of the next processor instruction and issues the instruction to the decode stage. The instruction is fetched either from the Instruction Prefetch Buffer or an external instruction memory.

Decode Stage

During the decode stage, the Execution Unit decodes the instruction selected during the fetch stage and fetches and/or assembles the required operands. It also evaluates addresses for branches, loads, and stores.

Execute Stage

During the execute stage, the Execution Unit performs the operation specified by the instruction.

Write-Back Stage

During the write-back stage, the results of the operation performed during the execute stage are stored. In the case of branches, loads, and stores, the physical address generated during the execute stage is transmitted to an external device or memory.

Function Organization

Figure 3 shows the Am29005 microprocessor internal data-flow organization. The following sections refer to the various components on this data-flow diagram.

Instruction Fetch Unit

The Instruction Fetch Unit fetches instructions and supplies instructions to other functional units. It incorporates the Instruction Prefetch Buffer and the Program Counter Unit. All components of the Instruction Fetch Unit operate during the fetch stage of the processor pipeline.

Instruction Prefetch Buffer

Most instructions executed by the Am29005 microprocessor are fetched from external instruction/data memory. The processor prefetches instructions so that they are requested at least four cycles before they are required for execution.

Prefetched instructions are stored in a four-word Instruction Prefetch Buffer while awaiting execution. An instruction prefetch request occurs whenever there is a free location in this buffer (if the processor is otherwise enabled to fetch instructions). When a nonsequential instruction fetch occurs, prefetching is terminated, and then restarted for the new instruction stream.

Instruction prefetching uncouples the instruction fetch rate from the instruction access latency. For example, an instruction may be transferred to the processor two cycles after it is requested. However, as long as instructions are supplied to the processor at an average rate of one instruction per cycle, this latency has no effect on the instruction execution rate.

Program Counter Unit

The Program Counter Unit creates and sequences addresses of instructions as they are executed by the processor.

Execution Unit

The Execution Unit executes instructions. It incorporates the Register File, the Address Unit, the Arithmetic/Logic Unit, the Field Shift Unit, and the Prioritizer. The Register File and Address Unit operate during the decode stage of the pipeline. The Arithmetic/Logic Unit, Field Shift Unit, and Prioritizer operate during the execute stage of the pipeline. The Register File operates during the write-back stage.

Register File

The general-purpose registers are implemented by a 192-location Register File. The Register File can perform two read accesses and one write access in a single cycle. Normally, two read accesses are performed during the decode-pipeline stage to fetch operands required by the instruction being decoded. The write access during the same cycle completes the write-back stage of a previously executed instruction.

Addressing logic associated with the Register File distinguishes between the global and local general-purpose registers, and it performs the Stack-Pointer addressing for the local registers. Register File addressing functions are performed during the decode stage.

Address Unit

The Address Unit evaluates addresses for branches, loads, and stores. It also assembles instruction-immediate data and computes addresses for Load Multiple and Store Multiple sequences.

Arithmetic/Logic Unit

The ALU performs all logical, comparative, and arithmetic operations (including multiply step and divide step).

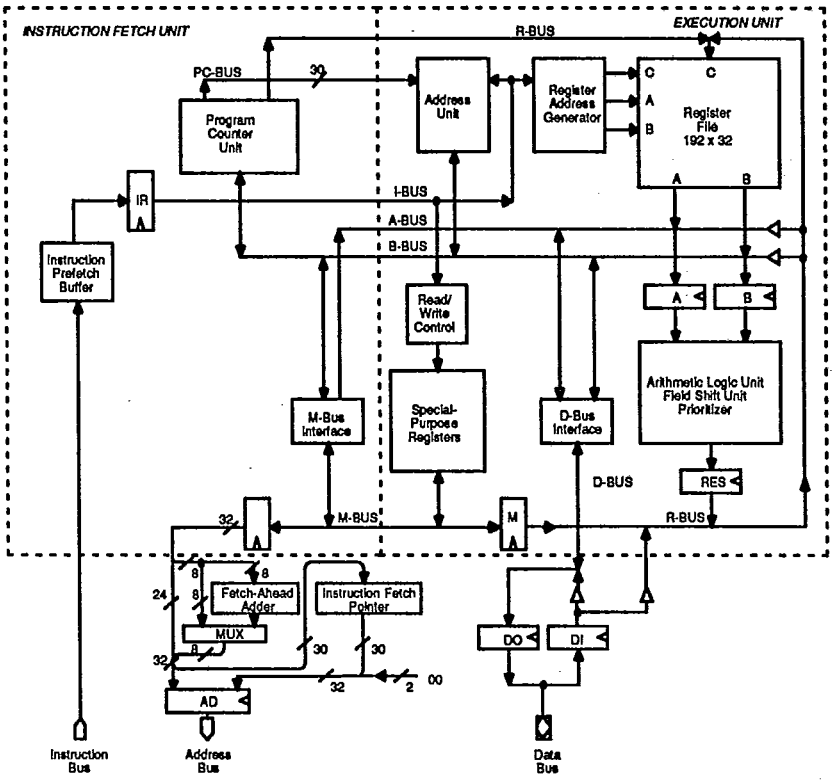


Figure 3. Am29005 Microprocessor Data Flow

Field Shift Unit
The Field Shift Unit performs N-bit shifts. The Field Shift Unit also performs byte and half-word extract and insert operations, and it extracts words from double words.

Prioritizer
The Prioritizer provides a count of the number of leading 0 bits in a 32-bit word; this is useful for performing floating-point normalization, for example. It can also be used to implement prioritization in a multilevel interrupt handler.

Processor Modes

The Am29005 microprocessor operates in several different modes to accomplish various processor and system functions. All modes except for Pipeline Hold (see below) are under direct control of instructions and/or processor control inputs. The Pipeline Hold mode normally is determined by the relative timing between the processor and its external system for certain types of operations. The processor provides an external indication of its operational mode.

Executing
When the processor is in the Executing mode, it fetches and executes instructions as described in this manual. External accesses occur as required.

Wait
When the processor is in the Wait mode, it does not execute instructions and it performs no external accesses. The Wait mode is controlled by the Current Processor Status Register. The processor leaves this mode when an interrupt or trap for which it is enabled occurs, or when a reset occurs.

Pipeline Hold
Under certain conditions, processor pipelining might cause nonsequential instruction execution or timing-dependent results of execution. For example, the processor might attempt to execute an instruction that has not been fetched from instruction/data memory.

For such cases, pipeline-interlock hardware detects the anomalous condition and suspends processor

execution until execution can proceed properly. While execution is suspended by the interlock hardware, the processor is in the Pipeline Hold mode. The processor resumes execution when the pipeline-interlock hardware determines that it is correct to do so.

Halt

The Halt mode is provided so that the processor may be placed under the control of a hardware-development system for the purposes of hardware and software debugging. The processor enters the Halt mode as the result of instruction execution, or as the result of external controls. In the Halt mode, the processor neither fetches nor executes instructions.

Step

The Step mode allows a hardware-development system to step through processor pipeline operation on a stage-by-stage basis. The Step mode is nearly identical to the Halt mode, except that it enables the processor to enter the Executing mode while the pipeline advances by one stage.

Load Test Instruction

The Load Test Instruction mode permits a hardware-development system to access data contained in the processor or system. This is accomplished by allowing the hardware-development system to supply the processor with instructions, instead of having the processor fetch instructions from instruction/data memory. The Load Test Instruction mode is defined so that, once the processor has completed the execution of instructions provided by the hardware-development system, it may resume the execution of its normal instruction sequence.

Test

The Test mode facilitates testing of hardware associated with the processor by disabling processor outputs so that they may be driven directly by test hardware. The Test mode also allows the addition of a second processor to a system to monitor the outputs of the first and to signal detected errors.

Reset

The Reset mode provides initialization of certain processor registers and control state. This is used for power-on reset, for eliminating unrecoverable error conditions, and for supporting certain hardware debugging functions.

System Interface

This section briefly describes the features of the Am29005 microprocessor that allow it to be connected to other system components.

The two major interfaces of the Am29005 microprocessor, introduced in this section, are the channel and the Test/Development interfaces. The other topics briefly described here are clock generation, master/slave checking, and coprocessor attachment.

Channel

The Am29005 microprocessor channel consists of the following 32-bit buses and related controls:

1. an Instruction Bus, which transfers instructions into the processor;
2. a Data Bus, which transfers data to and from the processor;
3. an Address Bus, which provides addresses for both instruction and data accesses. The address bus also is used to transfer data to a coprocessor.

The channel performs accesses and data transfers to all external devices and memories, including instruction/data memories, instruction caches, instruction read-only memories, data caches, input/output devices, bus converters, and coprocessors.

The channel defines three different access protocols: simple, pipelined, and burst-mode. For simple accesses, the Am29005 microprocessor holds the address valid throughout the entire access. This is appropriate for high-speed devices that can complete an access in one cycle, and for low-cost devices that are accessed infrequently (such as read-only memories containing initialization routines). Pipelined and burst-mode accesses provide high performance with other types of devices and memories.

For pipelined accesses, the address transfer is uncoupled from the corresponding data or instruction transfer. After transmitting an address for a request, the processor may transmit one more address before receiving the reply to the first request. This allows address transfer and decoding to be overlapped with another access.

On the other hand, burst-mode accesses eliminate the address-transfer cycle completely. Burst-mode accesses are defined so that once an address is transferred for a given access, subsequent accesses to sequentially increasing addresses may occur without retransfer of the address. The burst may be terminated at any time by either the processor or responding device.

The Am29005 microprocessor determines whether an access is simple, pipelined, or burst-mode on a transfer-by-transfer (i.e., generally device-by-device) basis. However, an access that begins as a simple access may be converted to a pipelined or burst-mode access at any time during the transfer. This relaxes the timing constraints on the channel-protocol implementation, since addressed devices do not have to respond immediately to a pipelined or burst-mode request.

Except for the shared address bus, the channel maintains a strict division between instruction and data accesses. In the most common situation, the system supplies the processor with instructions using burst-mode accesses, with instruction addresses transmitted

to the system only when a branch occurs. Data accesses can occur simultaneously without interfering with instruction transfer.

The Am29005 microprocessor contains arbitration logic to support other masters on the channel. A single external master can arbitrate directly for the channel, while multiple masters may arbitrate using a daisy chain or other method that requires no additional arbitration logic. However, to increase arbitration performance in a multiple-master configuration, an external channel arbiter should be used. This arbiter works in conjunction with the processor's arbitration logic.

Test/Development Interface

The Am29005 microprocessor supports the attachment of a hardware-development system. This attachment is made directly to the processor in the system under development, without the removal of the processor from the system. The Test/Development Interface makes it possible for the hardware-development system to gain control over the Am29005 microprocessor, and inspect and modify its internal state (e.g., general-purpose register contents, etc.). In addition, the Am29005 microprocessor may be used to access other system devices and memories on behalf of the hardware-development system.

The Test/Development Interface is made up of controls and status signals provided on the Am29005 microprocessor, as well as the instruction and data buses. The Halt, Step, Reset, and Load Test Instruction modes allow the hardware-development system to control the operation of the Am29005 microprocessor. The hardware-development system may supply the processor with instructions on the instruction bus using the load test instruction mode. The internal processor state can be inspected and modified via the data bus.

Clocks

The Am29005 microprocessor generates and distributes a system clock at its operating frequency. This clock is specially designed to reduce skews between the system clock and the processor's internal clocks. The internal clock-generation circuitry requires a single-phase oscillator signal at twice the processor operating frequency.

For systems in which processor-generated clocks are not appropriate, the Am29005 microprocessor also can accept a clock from an external clock generator.

The processor decides between these two clocking arrangements based on whether the power supply to the clock-output driver (PWRCLK) is tied to +5 V or to Ground.

Master/Slave Operation

Each Am29005 microprocessor output has associated logic that compares the signal on the output with the signal that the processor is providing internally to the output driver. The processor signals situations where the output of any enabled driver does not agree with its input.

For a single processor, the output comparison detects short circuits in output signals, but does not detect open circuits. It is possible to connect a second processor in parallel with the first, where the second processor has its outputs disabled due to the Test mode. The second processor detects open-circuit signals, as well as provides a check of the outputs of the first processor.

Coprocessor Attachment

A coprocessor for the Am29005 microprocessor attaches directly to the processor channel. However, this attachment has features that are different from those of other channel devices. The coprocessor interface is designed to support a high operand transfer rate and to support the overlap of coprocessor operations with other processor operations, including other external accesses.

The coprocessor is assigned a special address space on the channel. This permits the transfer of operands and other information on the address bus without interfering with normal addressing functions. Since both the address bus and data bus are used for data transfer, the Am29005 microprocessor can transfer 64 bits of information to the coprocessor in one cycle.

Program Modes

All system-protection features of the Am29005 microprocessor are based on two mutually exclusive program modes: the Supervisor mode and the User mode.

Supervisor Mode

The processor is in the Supervisor mode whenever the Supervisor Mode (SM) bit of the Current Processor Status Register (see Register Description section) is 1. In this mode, executing programs have access to all processor resources.

During the address cycle of a channel request, the Supervisor mode is indicated by the SUP/$\overline{US}$ output being High.

User Mode

The processor is in the User mode whenever the SM bit in the Current Processor Status Register is 0. In this mode, any of the following actions by an executing program causes a Protection Violation trap to occur:

1. An attempted access of any general-purpose register for which a bit in the Register Bank Protect Register is 1.
2. An attempted execution of a load or store instruction for which the UA bit is 1.
3. An attempted execution of one of the following instructions: Interrupt Return, Interrupt Return

and Invalidate, Invalidate, or Halt. However, a hardware-development system can disable protection checking for the Halt instruction, so this instruction may be used to implement instruction breakpoints in User-mode programs.

4. An attempted access of one of the following registers: Vector Area Base Address, Old Processor Status, Current Processor Status, Configuration, Channel Address, Channel Data, Channel Control, Register Bank Protect, Timer Counter, Timer Reload, Program Counter 0, Program Counter 1, or Program Counter 2.

5. An attempted execution of an assert or Emulate instruction that specifies a vector number between 0 and 63, inclusive.

Devices and memories on the channel also can implement protection and generate traps based on the value of the SM bit. During the address cycle of a channel request, the User mode is indicated by the SUP/$\overline{US}$ output being Low.

REGISTER DESCRIPTION

The Am29005 microprocessor has two classes of registers that are accessible by instructions. These are general-purpose registers and special-purpose registers. Any operation available in the Am29005 microprocessor can be performed on the general-purpose registers, while special-purpose registers are accessed only by explicit data movement to or from general-purpose registers. Various protection mechanisms prevent the access of some of these registers by User-mode programs.

General-Purpose Registers

The Am29005 microprocessor incorporates 192 general-purpose registers. The organization of the general-purpose registers is diagrammed in Figure 4.

General-purpose registers hold the following types of operands for program use:

1. 32-bit data addresses
2. 32-bit signed or unsigned integers
3. 32-bit branch-target addresses
4. 32-bit logical bit strings
5. 8-bit signed or unsigned characters
6. 16-bit signed or unsigned integers
7. Word-length Booleans
8. Single-precision floating-point numbers
9. Double-precision floating-point numbers (in two register locations)

Because a large number of general-purpose registers are provided, a large amount of frequently used data can be kept on-chip, where access time is fastest.

Am29005 microprocessor instructions can specify two general-purpose registers for source operands, and one general-purpose register for storing the instruction result. These registers are specified by three 8-bit instruction fields containing register numbers. A register may be specified directly by the instruction, or indirectly by one of three special-purpose registers.

Register Addressing

The general-purpose registers are partitioned into 64 global registers and 128 local registers, differentiated by the most-significant bit of the register number. The distinction between global and local registers is the result of register-addressing considerations.

The following terminology is used to describe the addressing of general-purpose registers:

1. Register number—this is a software-level number for a general-purpose register. For example, this is the number contained in an instruction field. Register numbers range from 0 to 255.

2. Global register number—this is a software-level number for a global register. Global register numbers range from 0 to 127.

3. Local register number—this is a software-level number for a local register. Local register numbers range from 0 to 127.

4. Absolute register number—this is a hardware-level number used to select a general-purpose register in the Register File. Absolute register numbers range from 0 to 255.

Global Registers

When the most-significant bit of a register number is 0, a global register is selected. The 7 least-significant bits of the register number give the global register number. For global registers, the absolute register number is equivalent to the register number.

Global Registers 2 through 63 are unimplemented. An attempt to access these registers yields unpredictable results; however, they may be protected from User-mode access by the Register Bank Protect Register.

The register numbers associated with Global Registers 0 and 1 have special meaning. The number for Global Register 0 specifies that an indirect pointer is to be used as the source of the register number; there is an indirect pointer for each of the instruction operand/result registers. Global Register 1 contains the Stack Pointer, which is used in the addressing of local registers as explained below.

Local Register Stack Pointer

The Stack Pointer is a 32-bit register that may be an operand of an instruction as any other general-purpose register. However, a shadow copy of Global Register 1 is maintained by processor hardware to be used in local register addressing. This shadow copy is set only with the results of Arithmetic and Logical instructions. If the Stack Pointer is set with the result of any other instruc-

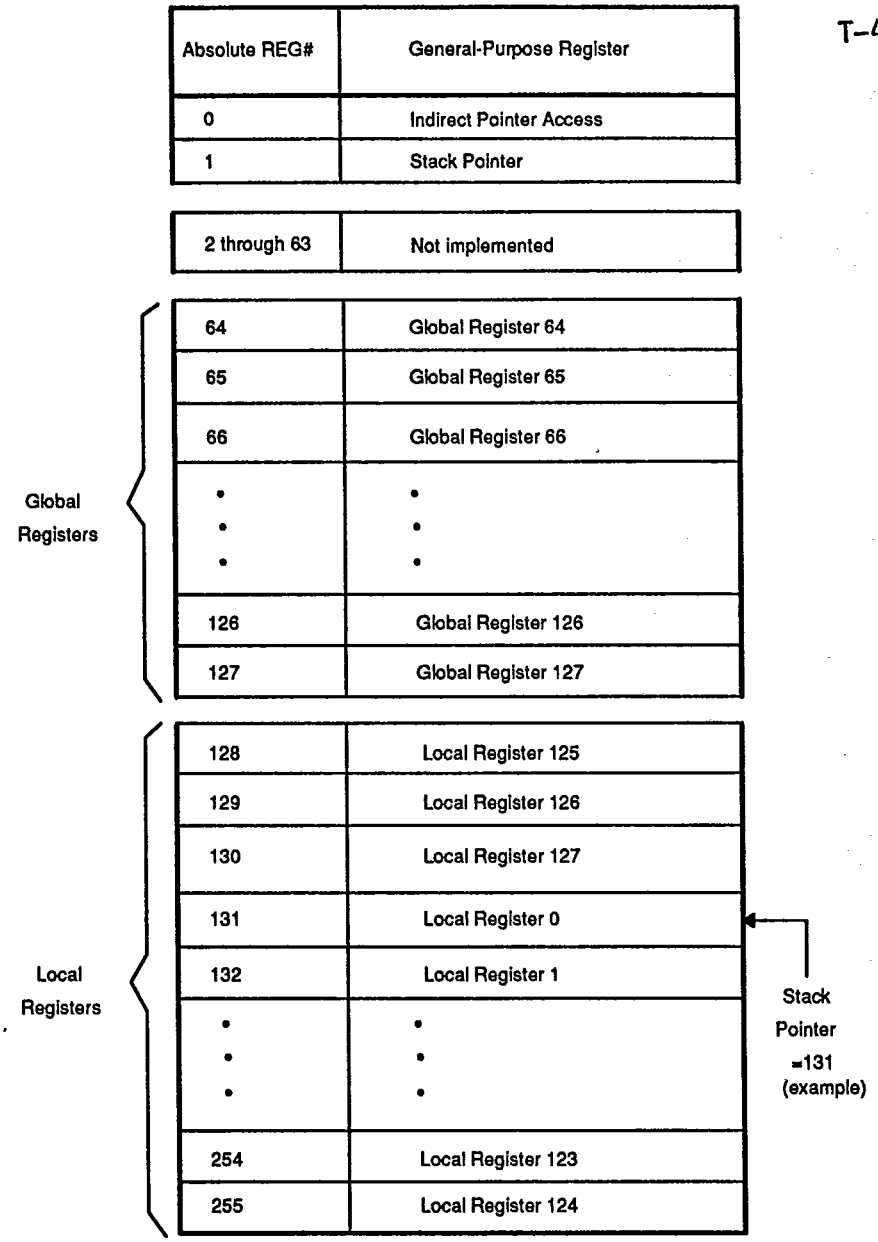


Figure 4. General-Purpose Register Organization

tion class, local registers cannot be accessed predictably until the Stack Pointer is set once again with an Arithmetic or Logical instruction.

Local Registers

When the most-significant bit of a register number is 1, a local register is selected. The 7 least-significant bits of

the register number give the local-register number. For local registers, the absolute register number is obtained by adding the local register number to bits 8–2 of the Stack Pointer and truncating the result to 7 bits; the most-significant bit of the original register number is unchanged (i.e., it remains a 1).

The Stack Pointer addition applied to local register numbers provides a limited form of base-plus-offset addressing within the local registers. The Stack Pointer contains the 32-bit base address. This assists run-time storage management of variables for dynamically nested procedures.

Register Banking

For the purpose of access restriction, the general-purpose registers are divided into register banks. Register banks consist of 16 registers (except for Bank 0, which contains Unimplemented Registers 2 through 15) and are partitioned according to absolute register numbers, as shown in Figure 5.

The Register Bank Protect Register contains 16 protection bits, where each bit controls User-mode accesses (read or write) to a bank of registers. Bits 0–15 of the

Register Bank Protect Register Bit	Absolute-Register Numbers	General-Purpose Registers
0	2 through 15	Bank 0 (unimplemented)
1	16 through 31	Bank 1 (unimplemented)
2	32 through 47	Bank 2 (unimplemented)
3	48 through 63	Bank 3 (unimplemented)
4	64 through 79	Bank 4
5	80 through 95	Bank 5
6	96 through 111	Bank 6
7	112 through 127	Bank 7
8	128 through 143	Bank 8
9	144 through 159	Bank 9
10	160 through 175	Bank 10
11	176 through 191	Bank 11
12	192 through 207	Bank 12
13	208 through 223	Bank 13
14	224 through 239	Bank 14
15	240 through 255	Bank 15

Figure 5. Register Bank Organization

Register Bank Protect Register protect Register Banks 0 through 15, respectively.

When a bit in the Register Bank Protect Register is 1 and a register in the corresponding bank is specified as an operand register or result register by a User-mode instruction, a Protection Violation trap occurs. Note that protection is based on absolute register numbers; in the case of local registers, Stack-Pointer addition is performed before protection checking.

When the processor is in Supervisor mode, the Register Bank Protect Register has no effect on general-purpose register accesses.

Indirect Accesses

Specification of Global Register 0 as an instruction-operand register or result register causes an indirect access to the general-purpose registers. In this case, the absolute register number is provided by an indirect pointer contained in a special-purpose register.

Each of the three possible registers for instruction execution has an associated 8-bit indirect pointer. Indirect register numbers can be selected independently for each of the three operands. Since the indirect pointers contain absolute register numbers, the number in an indirect pointer is not added to the Stack Pointer when local registers are selected.

The indirect pointers are set by the Move To Special Register, Floating-Point, MULTIPLY, MULTM, MULTIPLU, MULTMU, DIVIDE, DIVIDU, SETIP, and EMULATE instructions.

For a Move To Special Register instruction, an indirect pointer is set with bits 9–2 of the 32-bit source operand. This provides consistency between the addressing of words in general-purpose registers and the addressing of words in external devices or memories. A modification of an indirect pointer using a Move To Special Register has a delayed effect on the addressing of general-purpose registers.

For the remaining instructions, all three indirect pointers are set, simultaneously, with the absolute register numbers derived from the register numbers specified by the instruction. For any local registers selected by the instruction, the Stack-Pointer addition is applied to the register numbers before the indirect pointers are set.

Register numbers stored into the indirect pointers are checked for bank-protection violations—except when an indirect pointer is set by a Move-To-Special-Register instruction—at the time that the indirect pointers are set.

Special-Purpose Registers

The Am29005 microprocessor contains 25 special-purpose registers. The organization of the special-purpose registers is shown in Figure 6.

Special-purpose registers provide controls and data for certain processor operations. Some special-purpose registers are updated dynamically by the processor, independent of software controls. Because of this, a read of a special-purpose register following a write does not necessarily get the data that was written.

Some special-purpose registers have fields that are reserved for future processor implementations. When a special-purpose register is read, a bit in a reserved field is read as a 0. An attempt to write a reserved bit with a 1 has no effect; however, this should be avoided because of upward-compatibility considerations. The only exceptions to this rule are bits 6–5 of the Current Processor Status Register and bit 0 of the Configuration Register. These correspond to implemented bits in the Am29000 microprocessor, and should be written with values of 1 for upward compatibility with that microprocessor.

The special-purpose registers are accessed by explicit data movement only. Instructions that move data to or from a special-purpose register specify the special-purpose register by an 8-bit field containing a special-purpose register number. Register numbers are specified directly by instructions.

An attempted read of an unimplemented special-purpose register yields an unpredictable value. An attempted write of an unimplemented special-purpose register has no effect; however, this should be avoided, because of upward-compatibility considerations.

The special-purpose registers are partitioned into protected and unprotected registers. Special-purpose registers numbered 0–127 and 160–255 are protected (note that not all of these are implemented). Special-purpose registers numbered 128–159 are unprotected (again, not all are implemented).

Protected special-purpose registers numbered 0–127 are accessible only by programs executing in the Supervisor mode. An attempted read or write of a protected special-purpose register by a User-mode program causes a Protection Violation trap to occur. Protected special-purpose registers numbered 160–255 are not accessible by programs in either the User mode or the Supervisor mode. These register numbers identify virtual registers in the floating-point architecture.

The Floating-Point Environment Register, Integer Environment Register, Floating-Point Status Register, and Exception Opcode Register are not implemented in processor hardware. These registers are implemented via a virtual floating-point interface provided on the Am29005 microprocessor.

Unprotected special-purpose registers are accessible by programs executing in both the User and Supervisor modes.

Vector Area Base Address (Register 0)

This protected special-purpose register (see Figure 7) specifies the beginning address of the interrupt/trap Vector Area. The Vector Area is either a table of 256 vectors that points to interrupt and trap handling

Register Number	Protected Registers	Mnemonic
0	Vector Area Base Address	VTB
1	Old Processor Status	OPS
2	Current Processor Status	CPS
3	Configuration	CFG
4	Channel Address	CHA
5	Channel Data	CHD
6	Channel Control	CHC
7	Register Bank Protect	RBP
8	Timer Counter	TMC
9	Timer Reload	TMR
10	Program Counter 0	PC0
11	Program Counter 1	PC1
12	Program Counter 2	PC2

	Unprotected Registers	
128	Indirect Pointer C	IPC
129	Indirect Pointer A	IPA
130	Indirect Pointer B	IPB
131	Q	Q
132	ALU Status	SR
133	Byte Pointer	BPR
134	Funnel Shift Count	FCR
135 ...	Load/Store Count Remaining	MC
160	Floating-Point Environment	FPE
161	Integer Environment	INTE
162 ...	Floating-Point Status	FPS
164	Exception Opcode	EXOP

Figure 6. Special-Purpose Registers

routines, or a segment of 256 64-instruction blocks that directly contains the interrupt and trap handling routines.

The organization of the Vector Area is determined by the Vector Fetch (VF) bit of the Configuration Register. If the VF bit is 1 when an interrupt or trap is taken, the vector number for the interrupt or trap (see Interrupts and Traps section) replaces bits 9–2 of the value in the Vector Area Base Address Register to generate the physical address for a vector contained in instruction/data memory.

If the VF bit is 0, the vector number replaces bits 15–8 of the value in the Vector Area Base Address Register to generate the physical address of the first instruction of the interrupt or trap handler. The instruction fetch for this instruction is directed either to instruction memory or instruction read-only memory, as determined by the ROM Vector Area (RV) bit of the Configuration Register.

Bits 31–16: Vector Area Base (VAB)—The VAB field gives the beginning address of the Vector Area. This address is constrained to begin on a 64-kb address-boundary in instruction data memory or instruction read-only memory.

Bits 15–0—These bits contain 0s; they force the alignment of the Vector Area.

Old Processor Status (Register 1)

This protected special-purpose register has the same format as the Current Processor Status described below. The Old Processor Status stores a copy of the Current Processor Status when an interrupt or trap is taken. This is required since the Current Processor Status will be modified to reflect the status of the interrupt/trap handler.

During an interrupt return, the Old Processor Status is copied into the Current Processor Status. This allows the Current Processor Status to be set as required for the routine that is the target of the interrupt return.

Current Processor Status (Register 2) T-49-17-32

This protected special-purpose register (see Figure 8) controls the behavior of the processor and its ability to recognize exceptional events.

Bits 31–16—Reserved.

Bit 15: Coprocessor Active (CA)—The CA bit is set and reset under the control of load and store instructions that transfer information to and from a coprocessor. This bit indicates that the coprocessor is performing an operation at the time that an interrupt or trap is taken. This notifies the interrupt or trap handler that the coprocessor contains state information to be preserved. Note that this notification occurs because the CA bit of the Old Processor Status is 1 in this case, not because of the value of the CA bit of the Current Processor Status.

Bit 14: Interrupt Pending (IP)—This bit allows software to detect the presence of external interrupts while they are disabled. The IP bit is set if one or more of the external signals $\overline{INTR_3}$–$\overline{INTR_0}$ is active, but the processor is disabled from taking the resulting interrupt due to the value of the DA, DI, or IM bits. If all external interrupt signals subsequently are deasserted while still disabled, the IP bit is reset.

Bits 13–12: Trace Enable, Trace Pending (TE, TP)—The TE and TP bits implement a software-controlled, instruction single-step facility. Single stepping is not implemented directly, but rather emulated by trap sequences controlled by these bits. The value of the TE bit is copied to the TP bit whenever an instruction execution is completed. When the TP bit is 1, a Trace trap occurs.

Bit 11: Trap Unaligned Access (TU)—The TU bit enables checking of address alignment for external data-memory accesses. When this bit is 1, an Unaligned Access trap occurs if the processor either generates an ad-

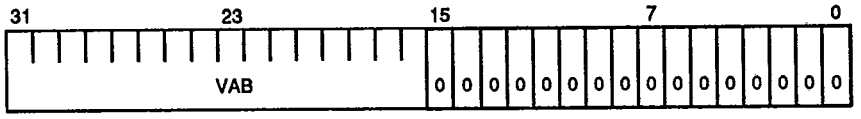


Figure 7. Vector Area Base Address Register

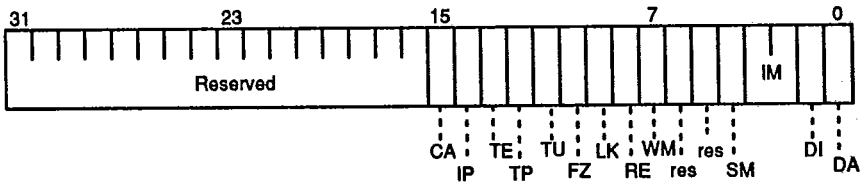


Figure 8. Current Processor Status Register

dress for an external word that is not aligned on a word address boundary (i.e., either of the least-significant 2 bits is 1), or generates an address for an external half-word that is not aligned on a half-word address boundary (i.e., the least-significant address bit is 1). When the TU bit is 0, data-memory address alignment is ignored.

Alignment is ignored for input/output accesses and coprocessor transfers. The alignment of instruction addresses is also ignored (unaligned instruction addresses can be generated only by indirect jumps). Interrupt/trap vector addresses always are aligned properly.

Bit 10: Freeze (FZ)—The FZ bit prevents certain registers from being updated during interrupt and trap processing, except by explicit data movement. The affected registers are: Channel Address, Channel Data, Channel Control, Program Counter 0, Program Counter 1, Program Counter 2, and the ALU Status Register.

When the FZ bit is 1, these registers hold their values. An affected register can be changed only by a Move To Special Register instruction. When the FZ bit is 0, there is no effect on these registers, and they are updated by processor instruction execution as described in this datasheet.

The FZ bit is set whenever an interrupt or trap is taken, holding critical state in the processor so that it is not modified unintentionally by the interrupt or trap handler.

Bit 9: LOCK (LK)—The LK bit controls the value of the $\overline{LOCK}$ external signal. If the LK bit is 1, the $\overline{LOCK}$ signal is active. If the LK bit is 0, the $\overline{LOCK}$ signal is controlled by the execution of the instructions Load and Set, Load and Lock, and Store and Lock. This bit is provided for the implementation of multiprocessor synchronization protocols.

Bit 8: ROM Enable (RE)—The RE bit enables instruction fetching from external instruction read-only memory (ROM). When this bit is 1, the IREQT signal directs all instruction requests to ROM. When this bit is 0, off-chip requests for instructions are directed to instruction/data memory.

Bit 7: WAIT Mode (WM)—The WM bit places the processor in the Wait mode. When this bit is 1, the processor performs no operations. The Wait mode is reset by an interrupt or trap for which the processor is enabled, or by the Reset mode.

Bit 6—Reserved (Physical Addressing/Data bit in Am29000 microprocessor).

Bit 5—Reserved (Physical Addressing/Instruction bit in Am29000 microprocessor).

Bit 4: Supervisor Mode (SM)—The SM bit protects certain processor context, such as protected special-purpose registers. When this bit is 1, the processor is in the Supervisor mode, and access to all processor context is allowed. When this bit is 0, the processor is in the User mode, and access to protected processor context is not allowed; an attempt to access (either read or write) protected processor context causes a Protection Violation trap.

For an external access, the User Access (UA) bit in the load or store instruction also controls access to protected processor context. When the UA bit is 1, the channel performs the access as though the program causing the access was in User mode.

Bits 3–2: Interrupt Mask (IM)—The IM field is an encoding of the processor priority with respect to external interrupts. The interpretation of the interrupt mask is specified by the following table:

IM Value	Result
0 0	$\overline{INTR_0}$ enabled
0 1	$\overline{INTR_1}$–$\overline{INTR_0}$ enabled
1 0	$\overline{INTR_2}$–$\overline{INTR_0}$ enabled
1 1	$\overline{INTR_3}$–$\overline{INTR_0}$ enabled

Bit 1: Disable Interrupts (DI)—The DI bit prevents the processor from being interrupted by external interrupt requests $\overline{INTR_3}$–$\overline{INTR_0}$. When this bit is 1, the processor ignores all external interrupts. However, note that traps (both internal and external), Timer interrupts, and Trace traps will be taken. When this bit is 0, the processor will take any interrupt enabled by the IM field, unless the DA bit is 1.

Bit 0: Disable all Interrupts and Traps (DA)—The DA bit prevents the processor from taking any interrupts and most traps. When this bit is 1, the processor ignores interrupts and traps, except for the $\overline{WARN}$, Instruction Access Exception, Data Access Exception, and Coprocessor Exception traps. When this bit is 0, all traps will be taken, and interrupts will be taken if otherwise enabled.

Configuration (Register 3)

This protected special-purpose register (see Figure 9) controls certain processor and system options. Most fields normally are modified only during system initialization. The Configuration Register definition follows.

Bits 31–24: Processor Release Level (PRL)—The PRL field is an 8-bit, read-only identification number that specifies the processor version.

Bits 23–6—Reserved.

Bit 5: Data Width Enable (DW)—The DW bit enables and disables byte and half-word external accesses. If the DW bit is 0, byte and half-word accesses are not performed in hardware, and these accesses must be emulated by software. If the DW bit is 1, byte and half-word accesses are performed by hardware: this requires that external devices and memories be able to write individual bytes and half-words within a word.

Bit 4: Vector Fetch (VF)—The VF bit determines the structure of the interrupt/trap Vector Area. If this bit is 1, the Vector Area is defined as a block of 256 vectors that specify the beginning addresses of the interrupt and trap

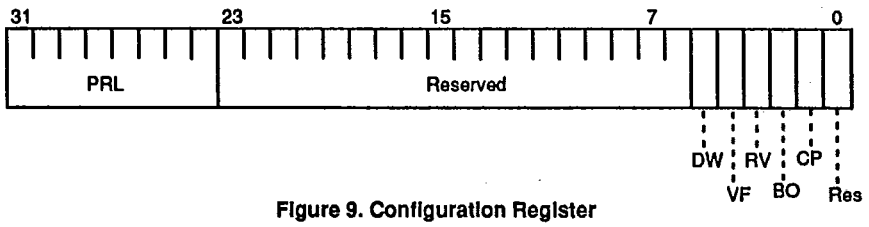


Figure 9. Configuration Register

handling routines. If the VF bit is 0, the Vector Area is a segment of 256 64-instruction blocks that contain the actual routines.

Bit 3: ROM Vector Area (RV)—If the VF bit is 0, the RV bit specifies whether the Vector Area is contained in instruction memory (RV = 0) or instruction read-only memory (RV = 1). The value of the RV bit is irrelevant if the VF bit is 1.

Bit 2: Byte Order (BO)—The BO bit determines the ordering of bytes and half-words within words. If the BO bit is 0, bytes and half-words are numbered left-to-right within a word. If the BO bit is 1, bytes and half-words are numbered right-to-left.

Bit 1: Coprocessor Present (CP)—The CP bit indicates the presence of a coprocessor that may be used by the processor. If this bit is 1, it enables the execution of load and store instructions that have a Coprocessor Enable (CE) bit of 1. If the CP bit is 0 and the processor attempts to execute a load or store instruction with a CE bit of 1, a Coprocessor Not Present trap occurs. This feature may be used to emulate coprocessor operations as well as to protect the state of a coprocessor shared between multiple processes.

Bit 0—Reserved (Branch Target Cache Disable Bit in Am29000 microprocessor).

Channel Address (Register 4)

This protected special-purpose register (Figure 10) is used to report exceptions during external accesses or coprocessor transfers. It also is used to restart interrupted Load Multiple and Store Multiple operations, and to restart other external accesses when possible.

The Channel Address Register is updated on the execution of every load or store instruction, and on every load or store in a Load Multiple or Store Multiple sequence, except when the Freeze (FZ) bit in the Current Processor Status Register is 1.

Bits 31–0: Channel Address (CHA)—This field contains the address of the current channel transaction (if the FZ bit of the Current Processor Status Register is 0). For transfers to the coprocessor, the CHA field contains data transferred to the coprocessor.

Channel Data (Register 5)

This protected special-purpose register (Figure 11) is used to report exceptions during external accesses or coprocessor transfers. It is also used to restart the first store of an interrupted Store Multiple operation and to restart other external accesses when possible.

The Channel Data Register is updated on the execution of every load or store instruction, and on every load or store in a Load Multiple or Store Multiple sequence, except when the Freeze (FZ) bit in the Current Processor Status Register is 1. When the Channel Data Register is updated for a load operation, the resulting value is unpredictable.

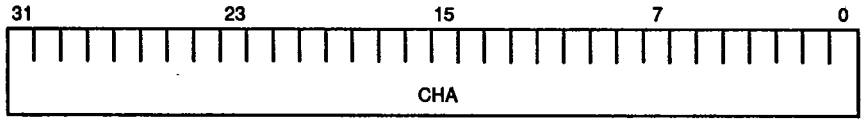


Figure 10. Channel Address Register

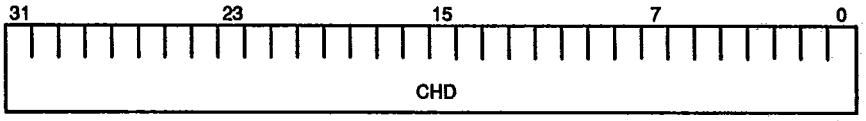


Figure 11. Channel Data Register

Bits 31–0: Channel Data (CHD)—This field contains the data (if any) associated with the current channel transaction (if the FZ bit of the Current Processor Status Register is 0). If the current channel transaction is not a store or a transfer to the coprocessor, the value of this field is irrelevant.

Channel Control (Register 6)

This protected special-purpose register (Figure 12) is used to report exceptions during external accesses or coprocessor transfers. It also is used to restart interrupted Load Multiple and Store Multiple operations, and to restart other external accesses when possible.

The Channel Control Register is updated on the execution of every load or store instruction, and on every load or store in a Load Multiple or Store Multiple sequence, except when the Freeze (FZ) bit in the Current Processor Status Register is 1.

Bits 31–24—These bits are a direct copy of bits 23–16 from the load or store instruction that started the current channel transaction.

Bits 23–16: Load/Store Count Remaining (CR)—The CR field indicates the remaining number of transfers for a Load Multiple or Store Multiple operation that encountered an exception or was interrupted before completion. This number is zero-based; for example, a value of 28 in this field indicates that 29 transfers remain to be completed. If the fault or interrupt occurs on the last transaction, the CR field contains a value of 0 and the ML bit is 1 (see below).

Bit 15: Load/Store (LS)—The LS bit is 0 if the channel transaction is a store operation, and 1 if it is a load operation.

Bit 14: Multiple Operation (ML)—The ML bit is 1 if the current channel transaction is a partially complete Load Multiple or Store Multiple operation; otherwise it is 0.

Bit 13: Set (ST)—The ST bit is 1 if the current channel transaction is for a Load and Set instruction; otherwise it is 0.

Bit 12: Lock Active (LA)—The LA bit is 1 if the current channel transaction is for a Load and Lock or Store and Lock instruction; otherwise it is 0. Note that this bit is not set as the result of the Lock (LK) bit in the Current Processor Status Register.

Bit 11—Reserved.

T-49-17-32

Bit 10: Transaction Faulted (TF)—The TF bit indicates that the current channel transaction was not complete due to some exceptional circumstance. This bit is set only for exceptions reported via the $\overline{\text{DERR}}$ input, and it causes a Data Access Exception or Coprocessor Exception trap to occur (depending on the value of the CE bit) when it is 1.

The TF bit allows the proper sequencing of externally reported errors that get preempted by higher-priority traps; it is reset by software that handles the resulting trap.

Bits 9–2: Target Register (TR)—The TR field indicates the absolute register number of data operand for the current transaction (either a load target or store data source). Since the register number in this field is absolute, it reflects the Stack-Pointer addition when the indicated register is a local register.

Bit 1: Not Needed (NN)—The NN bit indicates that, even though the Channel Address, Channel Data, and Channel Control registers contain a valid representation of an uncompleted load operation, the data requested is not needed. This situation arises when a load instruction is overlapped with an instruction that writes the load target register.

Bit 0: Contents Valid (CV)—The CV bit indicates that the contents of the Channel Address, Channel Data, and Channel Control registers are valid.

Register Bank Protect (Register 7)

This protected special-purpose register (Figure 13) protects banks of general-purpose registers from User-mode program accesses.

The general-purpose registers are partitioned into 16 banks of 16 registers each (except that Bank 0 contains 14 registers). The banks are organized as shown in Figure 4.

Bits 31–16—Reserved.

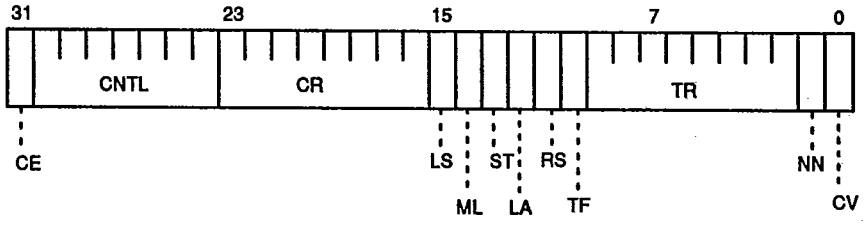


Figure 12. Channel Control Register

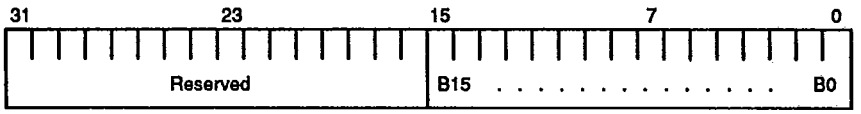


Figure 13. Register Bank Protect Register

Bits 15–0: Bank 15 through Bank 0 Protection Bits (B15–B0)—In the Register Bank Protect Register, each bit is associated with a particular bank of registers and the bit number gives the associated bank number (e.g., B11 determines the protection for Bank 11).

When a protection bit is 1, the corresponding bank is protected from access by programs executing in the User mode. A Protection Violation trap occurs when a User-mode program attempts to access (either read or write) a register in a protected bank. When a bit in this register is 0, the corresponding bank is available to programs executing in the User mode.

Supervisor-mode programs are not affected by the Register Bank Protect Register.

Register protection is based on absolute register numbers. For local registers, the protection checking is performed after the Stack-Pointer addition is performed.

Timer Counter (Register 8)

This protected special-purpose register (Figure 14) contains the counter for the Timer Facility.

Bits 31–24—Reserved.

Bits 23–0: Timer Count Value (TCV)—The 24-bit TCV field decrements by one on each processor clock. When the TCV field decrements to 0, it is reloaded with the content of the Timer Reload Value field in the Timer Reload Register. At this time, the Interrupt bit in the Timer Reload Register is set.

Timer Reload (Register 9)

This protected special-purpose register (Figure 15) maintains synchronization of the Timer Counter Register, enables Timer interrupts, and maintains Timer Facility status information.

Bits 31–27—Reserved.

Bit 26: Overflow (OV)—The OV bit indicates that a Timer interrupt occurred before a previous Timer interrupt was serviced. It is set if the Interrupt (IN) bit is 1 (see below) when the Timer Count Value (TCV) field of the Timer Counter Register decrements to 0. In this case, a Timer interrupt caused by the IN bit has not been serviced when another interrupt is created.

Bit 25: Interrupt (IN)—The IN bit is set whenever the TCV field decrements to 0. If this bit is 1 and the IE bit is also 1, a Timer interrupt occurs. Note that the IN bit is set when the TCV field decrements to 0, regardless of the value of the IE bit. The IN bit is reset by software that handles the Timer interrupt.

The TCV field is 0-based with respect to the Timer interrupt interval; for example, a value of 28 in the TCV field causes the IN bit to be set in the 29th subsequent pro-

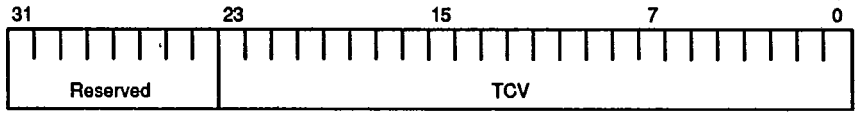


Figure 14. Timer Counter Register

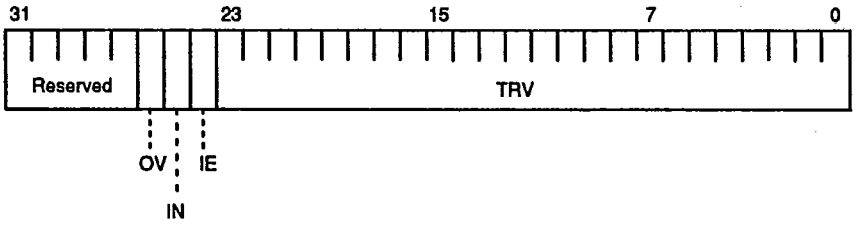


Figure 15. Timer Reload Register

cessor cycle. The reason for this is that the TCV field is 0 for a complete cycle before the IN bit is set.

Bit 24: Interrupt Enable (IE)—When the IE bit is 1, the Timer interrupt is enabled, and the Timer interrupt occurs whenever the IN bit is 1. When this bit is 0, the Timer interrupt is disabled. Note that Timer interrupts may be disabled by the DA bit of the Current Processor Status Register regardless of the value of the IE bit.

Bits 23–0: Timer Reload Value (TRV)—The value of this field is written into the Timer Count Value (TCV) field of the Timer Counter Register when the TCV field decrements to 0.

Program Counter 0 (Register 10)

This protected special-purpose register (Figure 16) is used on an interrupt return to restart the instruction that was in the decode stage when the original interrupt or trap was taken.

Bits 31-2: Program Counter 0 (PC0)—This field captures the word address of an instruction as it enters the decode stage of the processor pipeline, unless the Freeze (FZ) bit of the Current Processor Status Register is 1. If the FZ bit is 1, PC0 holds its value.

When an interrupt or trap is taken, the PC0 field contains the word address of the instruction in the decode stage; the interrupt or trap has prevented this instruction from executing. The processor uses the PC0 field to restart this instruction on an interrupt return.

Bits 1–0—These bits are 0 since instruction addresses are always word-aligned.

Program Counter 1 (Register 11)

This protected special-purpose register (Figure 17) is used on an interrupt return to restart the instruction that was in the execute stage when the original interrupt or trap was taken.

Bits 31–2: Program Counter 1 (PC1)—This field captures the word address of an instruction as it enters the execute stage of the processor pipeline, unless the Freeze (FZ) bit of the Current Processor Status Register is 1. If the FZ bit is 1, PC1 holds its value.

When an interrupt or trap is taken, the PC1 field contains the word address of the instruction in the execute stage; the interrupt or trap has prevented this instruction from completing execution. The processor uses the PC1 field to restart this instruction on an interrupt return.

Bits 1–0—These bits are 0 since instruction addresses are always word-aligned.

Program Counter 2 (Register 12)

This protected special-purpose register (Figure 18) reports the address of certain instructions causing traps.

Bits 31–2: Program Counter 2 (PC2)—This field captures the word address of an instruction as it enters the write-back stage of the processor pipeline, unless the Freeze (FZ) bit of the Current Processor Status Register is 1. If the FZ bit is 1, PC2 holds its value.

When an interrupt or trap is taken, the PC2 field contains the word address of the instruction in the write-back stage. In certain cases, PC2 contains the address of the

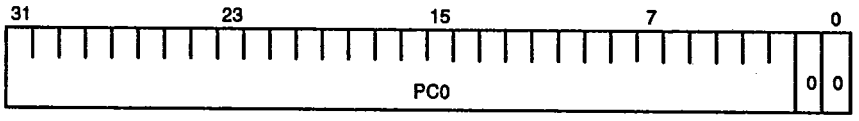


Figure 16. Program Counter 0 Register

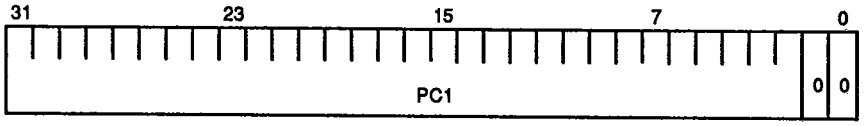


Figure 17. Program Counter 1 Register

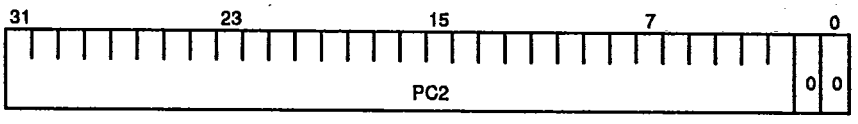


Figure 18. Program Counter 2 Register

instruction causing a trap. The PC2 field is used to report the address of this instruction, and has no other use in the processor.

Bits 1–0—These bits are 0 since instruction addresses are always word-aligned.

Indirect Pointer C (Register 128)

This unprotected special-purpose register (Figure 19) provides the RC-operand register number when an instruction RC field has the value 0 (i.e., when Global Register 0 is specified).

Bits 31–10—Reserved.

Bits 9–2: Indirect Pointer C (IPC)—The 8-bit IPC field contains an absolute register number for a general-purpose register. This number directly selects a register (Stack-Pointer addition is not performed in the case of local registers).

Bits 1–0—These bits contain 0s. The IPC field is aligned for compatibility with word addresses.

Indirect Pointer A (Register 129)

This unprotected special-purpose register (Figure 20) provides the RA-operand register number when an instruction RA field has the value 0 (i.e., when Global Register 0 is specified).

Bits 31–10—Reserved.

Bits 9–2: Indirect Pointer A (IPA)—The 8-bit IPA field contains an absolute register number for either a general-purpose register or a local register. This number directly selects a register (Stack-Pointer addition is not performed in the case of local registers).

Bits 1–0—These bits contain 0s. The IPA field is aligned for compatibility with word addresses.

Indirect Pointer B (Register 130)

This unprotected special-purpose register (Figure 21) provides the RB-operand register number when an instruction RB field has the value 0 (i.e., when Global Register 0 is specified).

Bits 31–10—Reserved.

Bits 9–2: Indirect Pointer B (IPB)—The 8-bit IPB field contains an absolute register number for a general-purpose register. This number directly selects a register (Stack-Pointer addition is not performed in the case of local registers).

Bits 1–0—These bits contain 0s. The IPB field is aligned for compatibility with word addresses.

Q (Register 131)

The Q Register is an unprotected special-purpose register (Figure 22).

Bits 31–0: Quotient/Multiplier (Q)—During a sequence of divide steps, this field holds the low-order bits of the dividend; it contains the quotient at the end of the divide. During a sequence of multiply steps, this field holds the multiplier; it contains the low-order bits of the result at the end of the multiply.

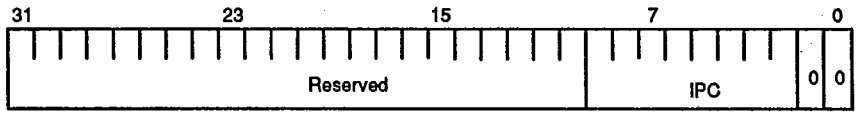


Figure 19. Indirect Pointer C Register

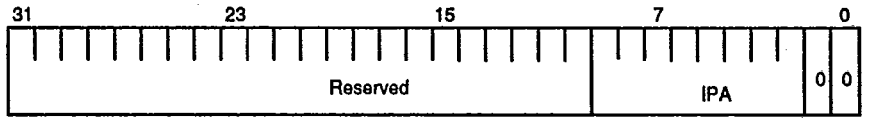


Figure 20. Indirect Pointer A Register

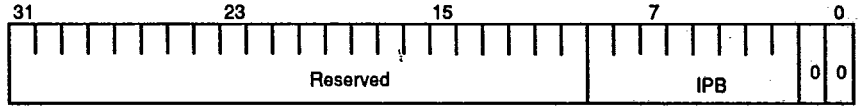


Figure 21. Indirect Pointer B Register

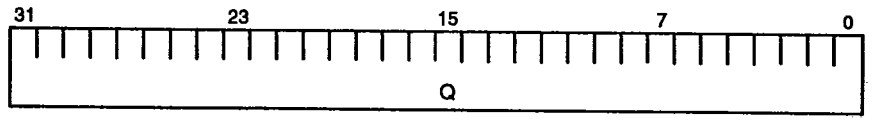


Figure 22. Q Register

For an integer divide instruction, the Q field contains the high-order bits of the dividend at the beginning of the instruction, and contains the remainder upon completion of the instruction.

ALU Status (Register 132)

This unprotected special-purpose register (Figure 23) holds information about the outcome of Arithmetic/Logic Unit (ALU) operations as well as control for certain operations performed by the Execution Unit.

Bits 31–12—Reserved.

Bit 11: Divide Flag (DF)—The DF bit is used by the instructions that implement division. This bit is set at the end of the division instructions either to 1 or to the complement of the 33rd bit of the ALU. When a Divide Step instruction is executed, the DF bit then determines whether an addition or subtraction operation is performed by the ALU.

Bit 10: Overflow (V)—The V bit indicates that the result of a signed, twos-complement ALU operation required more than 32 bits to represent the result correctly. The value of this bit is determined by exclusive ORing the ALU carry-out with the carry-in to the most-significant bit for signed, twos-complement operations. This bit is not used for any special purpose in the processor, and is provided for information only.

Bit 9: Negative (N)—The N bit is set with the value of the most-significant bit of the result of an arithmetic or logical operation. If twos-complement overflow occurs, the N bit does not reflect the true sign of the result. This bit is used in divide operations.

Bit 8: Zero (Z)—The Z bit indicates that the result of an arithmetic or logical operation is zero. This bit is not used for any special purpose in the processor, and is provided for information only.

Bit 7: Carry (C)—The C bit stores the carry-out of the ALU for arithmetic operations. It is used by the add-with-carry and subtract-with-carry instructions to generate the carry into the Arithmetic/Logic Unit.

Bits 6–5: Byte Pointer (BP)—The BP field holds a 2-bit pointer to a byte within a word. It is used by Insert Byte and Extract Byte instructions. The exact mapping of the pointer value to the byte position depends on the value of the Byte Order (BO) bit in the Configuration Register.

The most-significant bit of the BP field is used to determine the position of a half-word within a word for the Insert Half-Word, Extract Half-Word, and Extract Half-Word, Sign-Extended instructions. The exact mapping of the most-significant bit to the half-word position depends on the value of the BO bit in the Configuration Register.

The BP field is set by a Move To Special Register instruction with either the ALU Status Register or the Byte Pointer Register as the destination. It is also set by a load or store instruction if the Set Byte Pointer (SB) bit in the instruction is 1. A load or store sets the BP field either with the two least-significant bits of the address (if the DW bit of the Configuration Register is 0) or with the complement of the Byte Order bit of the Configuration Register (if DW is 1).

Bits 4–0: Funnel Shift Count (FC)—The FC field contains a 5-bit shift count for the Funnel Shifter. The Funnel Shifter concatenates two source operands into a single 64-bit operand and extracts a 32-bit result from this 64-bit operand; the FC field specifies the number of bit positions from the most-significant bit of the 64-bit operand to the most-significant bit of the 32-bit result. The FC field is used by the Extract instruction.

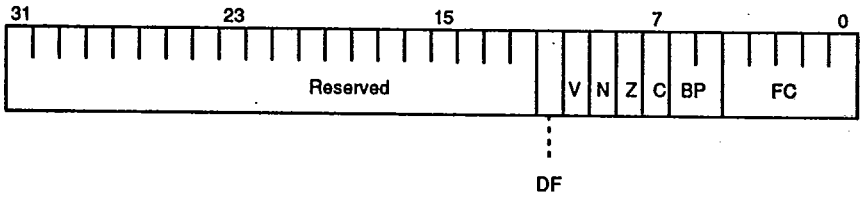


Figure 23. ALU Status Register

The FC field is set by a Move To Special Register instruction with either the ALU Status Register or the Funnel Shift Count Register as the destination.

Byte Pointer (Register 133)

This unprotected special-purpose register (Figure 24) provides an alternate access to the BP field in the ALU Status Register.

Bits 31–2—These bits contain 0s.

Bits 1–0: Byte Pointer (BP)—This field allows a program to change the BP field without affecting other fields in the ALU Status Register.

Funnel Shift Count (Register 134)

This unprotected special-purpose register (Figure 25) provides an alternate access to the FC field in the ALU Status Register.

Bits 31–5—These bits contain 0s.

Bits 4–0: Funnel Shift Count (FC)—This field allows a program to change the FC field without affecting other fields in the ALU Status Register.

Load/Store Count Remaining (Register 135)

This unprotected special-purpose register (Figure 26) provides alternate access to the CR field in the Channel Control Register.

Bits 31–8—These bits contain 0s.

Bits 7–0: Load/Store Count Remaining (CR)—This field allows a program to change the CR field without affecting other fields in the Channel Control Register, and is used to initialize the value before a Load Multiple or Store Multiple instruction is executed.

Floating-Point Environment (Register 160)

This unprotected special-purpose register (Figure 27) contains control bits that affect the execution of floating-point operations.

Bits 31–9—Reserved.

Bit 8: Fast Float Select (FF)—The FF bit being 1 enables fast floating-point operations, in which certain requirements of the IEEE floating-point specification are not met. This improves the performance of certain operations by sacrificing conformance to the IEEE specification.

Bits 7–6: Floating-Point Round Mode (FRM)—This field specifies the default mode used to round the results of floating-point operations, as follows:

FRM1–0	Round Mode
0 0	Round to nearest
0 1	Round to -∞
1 0	Round to +∞
1 1	Round to zero

Bit 5: Floating-Point Divide-By-Zero Mask (DM)—If the DM bit is 0, a Floating-Point Exception trap occurs when the divisor of a floating-point division operation is zero and the dividend is a non-zero, finite number. If the DM bit is 1, a Floating-Point Exception trap does not occur for divide-by-zero.

Bit 4: Floating-Point Inexact Result Mask (XM)—If the XM bit is 0, a Floating-Point Exception trap occurs when the result of a floating-point operation is not equal to the infinitely precise result. If the XM bit is 1, a Floating-Point Exception trap does not occur for an inexact result.

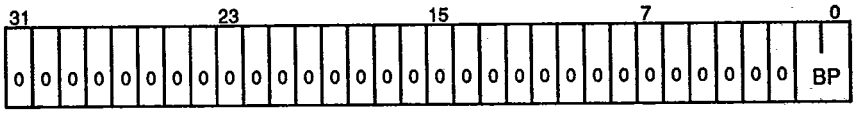


Figure 24. Byte Pointer

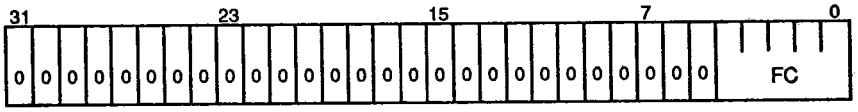


Figure 25. Funnel Shift Count

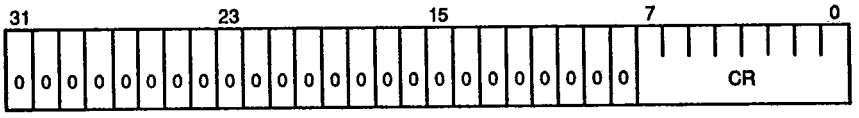


Figure 26. Load/Store Count Remaining

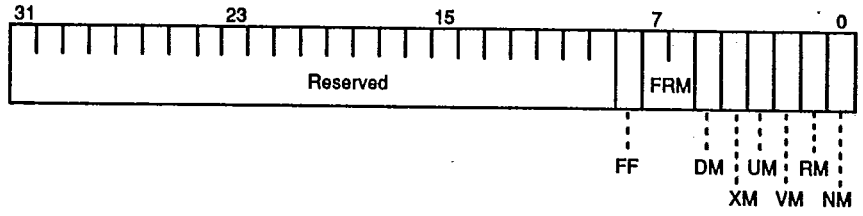


Figure 27. Floating-Point Environment

Bit 3: Floating-Point Underflow Mask (UM)—If the UM bit is 0, a Floating-Point Exception trap occurs when the result of a floating-point operation is too small to be expressed in the destination format. If the UM bit is 1, a Floating-Point Exception trap does not occur for underflow.

Bit 2: Floating-Point Overflow Mask (VM)—If the VM bit is 0, a Floating-Point Exception trap occurs when the result of a floating-point operation is too large to be expressed in the destination format. If the VM bit is 1, a Floating-Point Exception trap does not occur for overflow.

Bit 1: Floating-Point Reserved Operand Mask (RM)—If the RM bit is 0, a Floating-Point Exception trap occurs when one or more input operands to a floating-point operation is a reserved value, or when the result of a floating-point operation is a reserved value. If the RM bit is 1, a Floating-Point Exception trap does not occur for reserved operands.

Bit 0: Floating-Point Invalid Operation Mask (NM)—If the NM bit is 0, a Floating-Point Exception trap occurs when the input operands to a floating-point operation produce an indeterminate result (e.g., ∞ times 0). If the NM bit is 1, a Floating-Point Exception trap does not occur for invalid operations.

Integer Environment (Register 161)

This unprotected special-purpose register (Figure 28) contains control bits that affect the execution of integer operations.

Bits 31–2—Reserved.

Bit 1: Integer Division Overflow Mask (DO)—If the DO bit is 0, an Out of Range trap occurs when overflow of a signed or unsigned 32-bit result occurs during DIVIDE or DIVIDU instructions, respectively. If the DO bit is 1, an Out of Range trap does not occur for overflow during integer divide operations.

The DIVIDE and DIVIDU instructions always cause an Out of Range trap upon division by zero, regardless of the value of the DO bit.

Bit 0: Integer Multiplication Overflow Exception Mask (MO)—If the MO bit is 0, an Out of Range trap occurs when overflow of a signed or unsigned 32-bit result occurs during MULTIPLY or MULTIPLU instructions, respectively. If the DO bit is 1, an Out of Range trap does not occur for overflow during integer multiply operations.

Floating-Point Status (Register 162)

This unprotected special-purpose register (Figure 29) contains status bits indicating the outcome of floating-point operations. The bits of the Floating-Point Status Register are divided into two groups of status bits. The bits in each group correspond to the causes of Floating-Point Exception traps that are enabled and disabled by bits 5–0 of the Floating-Point Environment Register.

The first group of status bits (bits 13–8) are trap status bits that report the cause of a Floating-Point Exception trap. The trap status bits are set only when a Floating-Point Exception trap occurs, and indicate all conditions that apply to the trapping operation. All other operations leave the status bits unchanged. A trap status bit is set regardless of the state of the corresponding mask bit of the Floating-Point Environment Register, except that at least one of the mask bits must be 0 for the trap to occur. When a Floating-Point Exception trap occurs, all trap status bits not relevant to the trapping operation are reset.

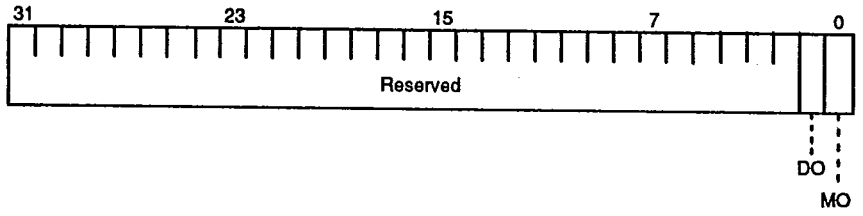


Figure 28. Integer Environment

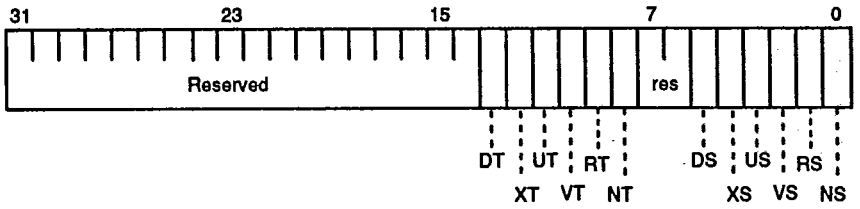


Figure 29. Floating-Point Status

The second group of status bits (bits 5–0) are sticky status bits that, once set, remain set until explicitly cleared by a Move to Special Register (MTSR) or Move to Special Register Immediate (MTSRIM) instruction. A sticky status bit is set only when a floating-point exception is detected and the corresponding mask bit of the Floating-Point Environment Register is 1. That is, the sticky status bit is set only if the corresponding cause of a Floating-Point Exception trap is disabled. Normally, this means that sticky status bits are not set when a Floating-Point Exception trap is taken. However, if multiple exceptions are detected, a sticky status bit corresponding to a masked exception may still be set if a Floating-Point Exception trap occurs for an unmasked exception.

Bits 31–14—Reserved.

Bit 13: Floating-Point Divide-By-Zero Trap (DT)—The DT bit is set when a Floating-Point Exception trap occurs, and the associated floating-point operation is a divide with a zero divisor and a non-zero, finite dividend. Otherwise, this bit is reset when a Floating-Point Exception trap occurs.

Bit 12: Floating-Point Inexact Result Trap (XT)—The XT bit is set when a Floating-Point Exception trap occurs, and the result of the associated floating-point operation is not equal to the infinitely precise result. Otherwise, this bit is reset when a Floating-Point Exception trap occurs.

Bit 11: Floating-Point Underflow Trap (UT)—The UT bit is set when a Floating-Point Exception trap occurs, and the result of the associated floating-point operation is too small to be expressed in the destination format. Otherwise, this bit is reset when a Floating-Point Exception trap occurs.

Bit 10: Floating-Point Overflow Trap (VT)—The VT bit is set when a Floating-Point Exception trap occurs, and the result of the associated floating-point operation is too large to be expressed in the destination format. Otherwise, this bit is reset when a Floating-Point Exception trap occurs.

Bit 9: Floating-Point Reserved Operand Trap (RT)—The RT bit is set when a Floating-Point Exception trap occurs, and either one or more input operands to the associated floating-point operation is a reserved value or the result of this floating-point operation is a reserved value. Otherwise, this bit is reset when a Floating-Point Exception trap occurs.

Bit 8: Floating-Point Invalid Operation Trap (NT)—The NT bit is set when a Floating-Point Exception trap occurs, and the input operands to the associated floating-point operation produce an indeterminate result. Otherwise, this bit is reset when a Floating-Point Exception trap occurs.

Bits 7–6—Reserved.

Bit 5: Floating-Point Divide-By-Zero Sticky (DS)—The DS bit is set when the DM bit of the Floating-Point Environment Register is 1, the divisor of a floating-point division operation is a zero, and the dividend is a non-zero, finite number.

Bit 4: Floating-Point Inexact Result Sticky (XS)—The XS bit is set when the XM bit of the Floating-Point Environment Register is 1, and the result of a floating-point operation is not equal to the infinitely precise result.

Bit 3: Floating-Point Underflow Sticky (US)—The US bit is set when the UM bit of the Floating-Point Environment Register is 1, and the result of a floating-point operation is too small to be expressed in the destination format.

Bit 2: Floating-Point Overflow Sticky (VS)—The VS bit is set when the VM bit of the Floating-Point Environment Register is 1, and the result of a floating-point operation is too large to be expressed in the destination format.

Bit 1: Floating-Point Reserved Operand Sticky (RS)—The RS bit is set when the RM bit of the Floating-Point Environment Register is 1, and either one or more input operands to a floating-point operation is a reserved value or the result of a floating-point operation is a reserved value.

Bit 0: Floating-Point Invalid Operation Sticky (NS)—The NS bit is set when the NM bit of the Floating-Point Environment Register is 1, and the input operands to a floating-point operation produce an indeterminate result.

Exception Opcode (Register 164)

This unprotected special-purpose register (Figure 30) reports the operation code (opcode) of an instruction causing a trap. It is provided primarily for recovery from floating-point exceptions, but reports the opcode of any trapping instruction.

Bits 31–8—Reserved.

Bits 7–0: Instruction Opcode (IOP)—This field captures the opcode of an instruction causing a trap as a result of instruction execution; the opcode is captured as the instruction enters the write-back stage of the processor pipeline. Instructions that do not trap as a consequence of execution do not modify the IOP field.

INSTRUCTION SET

The Am29005 microprocessor implements 113 instructions. All instructions execute in a single cycle except for IRET, LOADM, STOREM, and the trapping arithmetic instructions such as floating-point instructions.

Most instructions deal with general-purpose registers for operands and results; however, in most instructions, an 8-bit constant can be used in place of a register-based operand. Some instructions deal with special-purpose registers, external devices and memories, and coprocessors.

This section describes the nine instruction classes in the Am29005 microprocessor, and provides a brief summary of instruction operations.

If the processor attempts to execute an instruction that is not implemented, an Illegal Opcode trap occurs.

Integer Arithmetic

The Integer Arithmetic instructions perform add, subtract, multiply, and divide operations on word-length integers. Certain instructions in this class cause traps if signed or unsigned overflow occurs during the execution of the instruction. There is support for multi-precision arithmetic on operands whose lengths are multiples of words. All instructions in this class set the ALU Status Register. The integer arithmetic instructions are shown in Figure 31.

The instructions MULTIPLU, MULTMU, MULTIPLY, MULTM, DIVIDE, and DIVIDU are not implemented directly by processor hardware, but cause traps to occur in instruction-emulation routines.

Compare

The Compare instructions test for various relationships between two values. For all Compare instructions except the CPBYTE instruction, the comparisons are performed on word-length signed or unsigned integers. There are two types of Compare instructions. The first type places a Boolean value reflecting the outcome of the compare into a general-purpose register. For the second type (assert instructions), instruction execution continues only if the comparison is true; otherwise a trap occurs. The assert instructions specify a vector for the trap.

The assert instructions support run-time operand checking and operating-system calls. If the trap occurs in the User mode and a trap number between 0 and 63 is specified by the instruction, a Protection Violation trap occurs. The Compare instructions are shown in Figure 32.

Logical

The Logical instructions perform a set of bit-by-bit Boolean functions on word-length bit strings. All instructions in this class set the ALU Status Register. These instructions are shown in Figure 33.

Shift

The Shift instructions (Figure 34) perform arithmetic and logical shifts. All but the Extract instruction operate on word-length data and produce a word-length result. The Extract instruction operates on double-word data and produces a word-length result. If both parts of the double word for the Extract instruction are from the same source, the Extract operation is equivalent to a rotate operation. For each operation, the shift count is a 5-bit integer, specifying a shift amount in the range of 0 to 31 bits.

Data Movement

The Data Movement instructions (Figure 35) move bytes, half-words, and words between processor registers. In addition, they move data between general-purpose registers and external devices, memories, and the coprocessor.

Constant

The Constant instructions (Figure 36) provide the ability to place half-word and word constants into registers. Most instructions in the instruction set allow an 8-bit con-

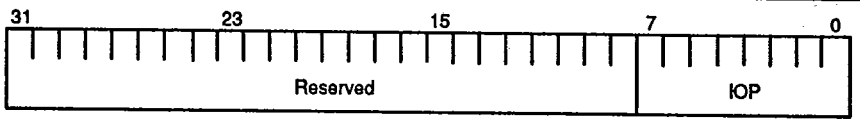


Figure 30. Exception Opcode

Mnemonic	Operation Description
ADD	DEST <-SRCA + SRCB
ADDS	DEST <-SRCA + SRCB IF signed overflow THEN Trap (Out Of Range)
ADDU	DEST <-SRCA + SRCB IF unsigned overflow THEN Trap (Out Of Range)
ADDC	DEST <-SRCA + SRCB + C
ADDCS	DEST <-SRCA + SRCB + C IF signed overflow THEN Trap (Out Of Range)
ADDCU	DEST <-SRCA + SRCB + C IF unsigned overflow THEN Trap (Out Of Range)
SUB	DEST <-SRCA – SRCB
SUBS	DEST <-SRCA – SRCB IF signed overflow THEN Trap (Out Of Range)
SUBU	DEST <-SRCA – SRCB IF unsigned underflow THEN Trap (Out Of Range)
SUBC	DEST <-SRCA – SRCB –1 + C
SUBCS	DEST <-SRCA – SRCB –1 + C IF signed overflow THEN Trap (Out Of Range)
SUBCU	DEST <-SRCA – SRCB –1 + C IF unsigned underflow THEN Trap (Out Of Range)
SUBR	DEST <-SRCB – SRCA
SUBRS	DEST <-SRCB – SRCA IF signed overflow THEN Trap (Out Of Range)
SUBRU	DEST <-SRCB – SRCA IF unsigned underflow THEN Trap (Out Of Range)
SUBRC	DEST <-SRCB – SRCA –1 + C
SUBRCS	DEST <-SRCB – SRCA –1 + C IF signed overflow THEN Trap (Out Of Range)
SUBRCU	DEST <-SRCB – SRCA –1 + C IF unsigned underflow THEN Trap (Out Of Range)
MULTIPLU	DEST <-SRCA * SRCB (unsigned)
MULTIPLY	DEST <-SRCA * SRCB (signed)
MUL	Perform 1-bit step of a multiply operation (signed)
MULL	Complete a sequence of multiply steps
MULTM	DEST <-SRCA * SRCB (signed), most-significant bits
MULTMU	DEST <-SRCA * SRCB (unsigned), most-significant bits
MULU	Perform 1-bit step of a multiply operation (unsigned)
DIVIDE	DEST <-(Q//SRCA)/SRCB (signed) Q <-Remainder
DIVIDU	DEST <-(Q//SRCA)/SRCB (unsigned) Q <-Remainder
DIV0	Initialize for a sequence of divide steps (unsigned)
DIV	Perform 1-bit step of a divide operation (unsigned)
DIVL	Complete a sequence of divide steps (unsigned)
DIVREM	Generate remainder for divide operation (unsigned)

Figure 31. Integer Arithmetic Instructions

Mnemonic	Operation Description
CPEQ	IF SRCA = SRCB THEN DEST <-TRUE ELSE DEST <-FALSE
CPNEQ	IF SRCA <> SRCB THEN DEST <-TRUE ELSE DEST <-FALSE
CPLT	IF SRCA < SRCB THEN DEST <-TRUE ELSE DEST <-FALSE
CPLTU	IF SRCA < SRCB (unsigned) THEN DEST <-TRUE ELSE DEST <-FALSE
CPLE	IF SRCA <= SRCB THEN DEST <-TRUE ELSE DEST <- FALSE
CPLEU	IF SRCA <= SRCB (unsigned) THEN DEST <-TRUE ELSE DEST <-FALSE
CPGT	IF SRCA > SRCB THEN DEST <-TRUE ELSE DEST <-FALSE
CPGTU	IF SRCA > SRCB (unsigned) THEN DEST <-TRUE ELSE DEST <-FALSE
CPGE	IF SRCA >= SRCB THEN DEST <-TRUE ELSE DEST <-FALSE
CPGEU	IF SRCA >= SRCB (unsigned) THEN DEST <-TRUE ELSE DEST <-FALSE
CPBYTE	IF (SRCA.BYTE0 = SRCB.BYTE0) OR (SRCA.BYTE1 = SRCB.BYTE1) OR (SRCA.BYTE2 = SRCB.BYTE2) OR (SRCA.BYTE3 = SRCB.BYTE3)THEN DEST <-TRUE ELSE DEST <-FALSE
ASEQ	IF SRCA = SRCB THEN Continue ELSE Trap (VN)
ASNEQ	IF SRCA <> SRCB THEN Continue ELSE Trap (VN)
ASLT	IF SRCA < SRCB THEN Continue ELSE Trap (VN)
ASLTU	IF SRCA < SRCB (unsigned) THEN Continue ELSE Trap (VN)
ASLE	IF SRCA <= SRCB THEN Continue ELSE Trap (VN)
ASLEU	IF SRCA <= SRCB (unsigned) THEN Continue ELSE Trap (VN)
ASGT	IF SRCA > SRCB THEN Continue ELSE Trap (VN)
ASGTU	IF SRCA > SRCB (unsigned) THEN Continue ELSE Trap (VN)
ASGE	IF SRCA >= SRCB THEN Continue ELSE Trap (VN)
ASGEU	IF SRCA >= SRCB (unsigned) THEN Continue ELSE Trap (VN)

Figure 32. Compare Instructions

Mnemonic	Operation Description
AND	DEST <-SRCA & SRCB
ANDN	DEST <-SRCA & ~ SRCB
NAND	DEST <-~ (SRCA & SRCB)
OR	DEST <-SRCA | SRCB
NOR	DEST <-~ (SRCA | SRCB)
XOR	DEST <-SRCA ^ SRCB
XNOR	DEST <-~ (SRCA ^ SRCB)

Figure 33. Logical Instructions

Mnemonic	Operation Description
SLL	DEST <-SRCA << SRCB (zero fill)
SRL	DEST <-SRCA >> SRCB (zero fill)
SRA	DEST <-SRCA >> SRCB (sign fill)
EXTRACT	DEST <-high-order word of (SRCA//SRCB << FC)

Figure 34. Shift Instructions

stant as an operand. The Constant instructions allow the construction of larger constants.

Floating-Point

The Floating-Point instructions (Figure 37) provide operations on single-precision (32-bit) or double-precision (64-bit) floating-point data. In addition, they provide conversions between single-precision, double-precision, and integer number representations. In the current processor implementation, these instructions cause traps to occur in routines that perform the floating-point operations.

Branch

The Branch instructions (Figure 38) control the execution flow of instructions. Branch target addresses may be absolute, relative to the Program Counter (with the offset given by a signed instruction constant), or contained in a general-purpose register. For conditional jumps, the outcome of the jump is based on a Boolean value in a general-purpose register. Procedure calls are unconditional and save the return address in a general-purpose register. All branches have a delayed effect; the instruction following the branch is executed regardless of the outcome of the branch.

Miscellaneous

The Miscellaneous instructions (Figure 39) perform various operations that cannot be grouped into other instruction classes. In certain cases, these are control functions available only to Supervisor-mode programs.

Reserved Instructions

Sixteen Am29005 microprocessor operation codes are reserved for instruction emulation. These instructions cause traps, much like the floating-point instructions, but currently have no specified interpretation.

Mnemonic	Operation Description
LOAD	DEST <-EXTERNAL WORD [SRCB]
LOADL	DEST <-EXTERNAL WORD [SRCB] assert *LOCK output during access
LOADSET	DEST <-EXTERNAL WORD [SRCB] EXTERNAL WORD [SRCB] <-h'FFFFFFFF', assert $\overline{\text{LOCK}}$ output during access
LOADM	DEST.. DEST + COUNT <- EXTERNAL WORD [SRCB] .. EXTERNAL WORD [SRCB + COUNT * 4]
STORE	EXTERNAL WORD [SRCB] <-SRCA
STOREL	EXTERNAL WORD [SRCB] <-SRCA assert $\overline{\text{LOCK}}$ output during access
STOREM	EXTERNAL WORD [SRCB] .. EXTERNAL WORD [SRCB + COUNT * 4] <- SRCA .. SRCA + COUNT
EXBYTE	DEST <-SRCB, with low-order byte replaced by byte in SRCA selected by BP
EXHW	DEST <-SRCB, with low-order half-word replaced by half-word in SRCA selected by BP
EXHWS	DEST <- half-word in SRCA selected by BP, sign-extended to 32 bits
INBYTE	DEST <-SRCA, with byte selected by BP replaced by low-order byte of SRCB
INHW	DEST <-SRCA, with half-word selected by BP replaced by low-order half-word of SRCB
MFSR	DEST <-SPECIAL
MTSR	SPDEST <-SRCB
MTSRIM	SPDEST <-0I16

Figure 35. Data Movement Instructions

Mnemonic	Operation Description
CONST	DEST <-0I16
CONSTH	Replace high-order half-word of SRCA by I16
CONSTN	DEST <-1I16

Figure 36. Constant Instructions

Mnemonic	Operation Description
FADD	DEST (single-precision) <-SRCA (single-precision) + SRCB (single-precision)
DADD	DEST (double-precision) <-SRCA (double-precision) + SRCB (double-precision)
FSUB	DEST (single-precision) <-SRCA (single-precision) – SRCB (single-precision)
DSUB	DEST (double-precision) <-SRCA (double-precision) – SRCB (double-precision)
FMUL	DEST (single-precision) <-SRCA (single-precision) * SRCB (single-precision)
FDMUL	DEST (double-precision) <-SRCA (single-precision) * SRCB (single-precision)
DMUL	DEST (double-precision) <-SRCA (double-precision) * SRCB (double-precision)
FDIV	DEST (single-precision) <-SRCA (single-precision)/ SRCB (single-precision)
DDIV	DEST (double-precision) <-SRCA (double-precision)/ SRCB (double-precision)
FEQ	IF SRCA (single-precision) = SRCB (single-precision) THEN DEST <-TRUE ELSE DEST <-FALSE
DEQ	IF SRCA (double-precision) = SRCB (double-precision) THEN DEST <-TRUE ELSE DEST <-FALSE
FGE	IF SRCA (single-precision) >= SRCB (single-precision) THEN DEST <-TRUE ELSE DEST <-FALSE
DGE	IF SRCA (double-precision) >= SRCB (double-precision) THEN DEST <-TRUE ELSE DEST <-FALSE
FGT	IF SRCA (single-precision) > SRCB (single-precision) THEN DEST <-TRUE ELSE DEST <-FALSE
DGT	IF SRCA (double-precision) > SRCB (double-precision) THEN DEST <-TRUE ELSE DEST <-FALSE
SQRT	DEST (single-precision, double-precision, extended-precision) <-SQRT[SRCA (single-precision, double-precision, extended-precision)]
CONVERT	DEST (integer, single-precision, double-precision) <-SRCA (integer, single-precision, double-precision)
CLASS	DEST (single-precision, double-precision, extended-precision) <-CLASS[SRCA (single-precision, double-precision, extended-precision)]

Figure 37. Floating-Point Instructions

Mnemonic	Operation Description
CALL	DEST <-PC//00 + 8 PC <-TARGET Execute delay instruction
CALLI	DEST <-PC//00 + 8 PC <-SRCB Execute delay instruction
JMP	PC <-TARGET Execute delay instruction
JMPI	PC <-SRCB Execute delay instruction
JMPT	IF SRCA = TRUE THEN PC <-TARGET Execute delay instruction
JMPTI	IF SRCA = TRUE THEN PC <-SRCB Execute delay instruction
JMPF	IF SRCA = FALSE THEN PC <-TARGET Execute delay instruction
JMPFI	IF SRCA = FALSE THEN PC <-SRCB Execute delay instruction

JMPFDEC	IF SRCA = FALSE THEN
	SRCA <-SRCA – 1
	PC <-TARGET
	ELSE
	SRCA <-SRCA – 1
	Execute delay instruction

Figure 38. Branch Instructions

Mnemonic	Operation Description
CLZ	Determine number of leading 0s in a word
SETIP	Set IPA, IPB, and IPC with operand register numbers
EMULATE	Load IPA and IPB with operand register numbers, and Trap (VN)
IRET	Perform an interrupt return sequence
HALT	Enter Halt mode on next cycle

Figure 39. Miscellaneous Instructions

The relevant operation codes and the corresponding trap vectors are:

Operation Codes (hexadecimal)	Trap Vector Numbers (decimal)
D8–DD	24–29
E7–E9	39–41
F8	56
FA–FF	58–63

These instructions are intended for future processor enhancements, and users desiring compatibility with future processor versions should not use them for any purpose.

The Am29000 microprocessor instructions Move To TLB (op code BE), Move From TLB (B6), Invalidate (9F), and Interrupt Return and Invalidate (86) are reserved in the Am29005 microprocessor. However, for compatibility with the Am29000 microprocessor, these are not defined as illegal instructions. An attempted execution of any of these instructions in the User mode causes a Protection Violation trap, because they are privileged in the Am29000 microprocessor. When executed in Supervisor mode:

- the Move To TLB instruction has no effect;
- the Move From TLB instruction places an undefined value in the destination register;
- the Invalidate instruction has no effect;
- the Interrupt Return and Invalidate instruction is equivalent to an Interrupt Return instruction.

DATA FORMATS AND HANDLING

This section describes the various data types supported by the Am29005 microprocessor, and the mechanisms for accessing data in external devices and memories. The Am29005 microprocessor includes provisions for the external access of bytes, half-words, unaligned words, and unaligned half-words, as described in this section.

Integer Data Types

Most Am29005 microprocessor instructions deal directly with word-length integer data; integers may be either signed or unsigned, depending on the instruction. Some instructions (e.g., AND) treat word-length operands as strings of bits. In addition, there is support for character, half-word, and Boolean data types.

Byte Operations

The processor supports character data through load, store, extraction, and insertion operations on word-length operands, and by a compare operation on byte-length fields within words. The format for unsigned and signed characters is shown in Figure 40; for signed characters, the sign bit is the most-significant bit of the character. For sequences of packed characters within words, bytes are ordered either left-to-right or right-to-left, depending on the BO bit of the Configuration Register (see Special Floating-Point Values section).

If the Data Width Enable (DW) bit of the Configuration Register is 1, the Am29005 microprocessor is enabled to load and store byte data. On a load, an external packed byte is converted to one of the character formats shown in Figure 40. On a store, the low-order byte of a word is packed into every byte of an external word. The External Data Accesses section describes external byte accesses in more detail.

The Extract Byte (EXBYTE) instruction replaces the low-order character of a destination word with an arbitrary byte-aligned character from a source word. For the EXBYTE instruction, the destination word can be a 0 word, which effectively 0-extends the character from the source operand.

The Insert Byte (INBYTE) instruction replaces an arbitrary byte-aligned character in a destination word with the low-order character of a source word. For the INBYTE instruction, the source operand can be a character constant specified by the instruction.

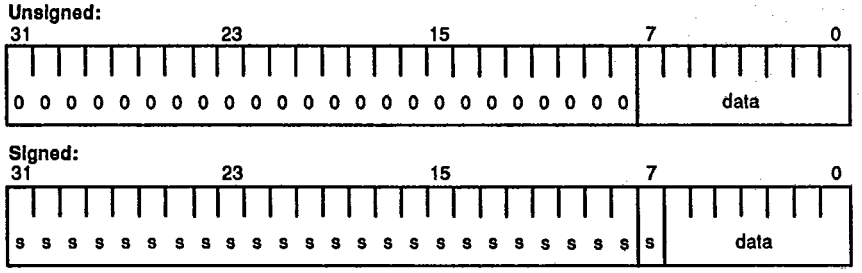


Figure 40. Character Format

The Compare Bytes (CPBYTE) instruction compares two word-length operands and gives a result of True if any corresponding bytes within the operands have equivalent values. This allows programs to detect characters within words without first having to extract individual characters, one at a time, from the word of interest.

Half-Word Operations

The processor supports half-word data through load, store, insertion, and extraction operations on word-length operands. The format for unsigned and signed half-words is shown in Figure 41; for signed half-words, the sign bit is the most-significant bit of the half-word. For sequences of packed half-words within words, half-words are ordered either left-to-right or right-to-left, depending on the Byte Order (BO) bit of the Configuration Register (see Addressing and Alignment section).

If the Data Width Enable (DW) bit of the Configuration Register is 1, the Am29005 microprocessor is enabled to load and store half-word data. On a load, an external packed half-word is converted to one of the formats shown in Figure 41. On a store, the low-order half-word of a word is packed into every half-word of an external word.

The Extract Half-Word (EXHW) instruction replaces the low-order half-word of a destination word with either the low-order or high-order half-word of a source word. For the EXHW instruction, the destination word can be a 0 word, which effectively 0-extends the half-word from the source operand.

The Extract Half-Word, Sign-Extended (EXHWS) instruction is similar to the EXHW instruction, except that it sign-extends the half-word in the destination word (i.e., it replaces the most-significant 16 bits of the destination word with the most-significant bit of the source half-word).

The Insert Half-Word (INHW) instruction replaces either the low-order or high-order half-word in a destination word with the low-order half-word of a source word.

Boolean Data

Some instructions in the Compare class generate word-length Boolean results. Also, conditional branches are conditional upon Boolean operands. The Boolean format used by the processor is such that the Boolean values True and False are represented by a 1 or 0, respectively, in the most-significant bit of a word. The remaining bits are unimportant; for the compare instructions, they are reset. Note that twos-complement negative integers are indicated by the Boolean value True in this encoding scheme.

Floating-Point Data Types

The Am29005 microprocessor defines single- and double-precision floating-point formats that comply with the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std. 754-1985). These data types are not supported directly in processor hardware, but can be implemented by a virtual floating-point interface provided in the Am29005 microprocessor.

In this section, the following nomenclature is used to denote fields in a floating-point value:

- s: sign bit
- bexp: biased exponent
- frac: fraction
- sig: significand

Single-Precision Floating-Point

The format for a single-precision floating-point value is shown in Figure 42.

Typically, the value of a single-precision operand is expressed by:

$$(-1)^{**}s * 1.\text{frac} * 2^{**}(\text{bexp}-127).$$

The encoding of special floating-point values is given in the Special Floating-Point Values section.

Double-Precision Floating-Point

The format for a double-precision floating-point value is shown in Figure 43.

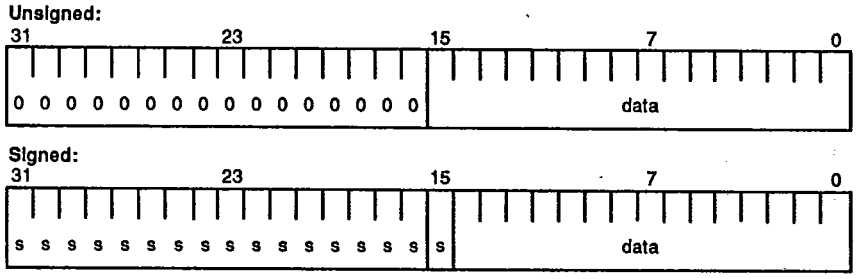


Figure 41. Half-Word Format

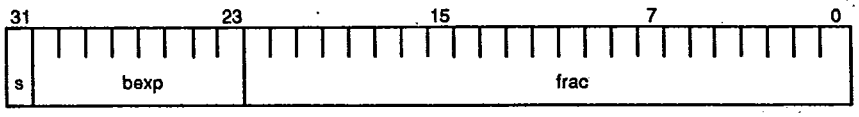


Figure 42. Single-Precision Floating-Point Format

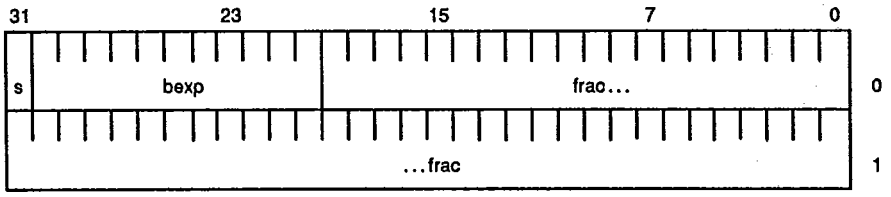


Figure 43. Double-Precision Floating-Point Format

Typically, the value of a double-precision operand is expressed by:

(–1)**s * 1.frac * 2**(bexp – 1023).

The encoding of special floating-point values is given in the Special Floating-Point Values section.

In order to be properly referenced by a floating-point instruction, a double-precision floating-point value must be double-word aligned. The absolute register number of the register containing the first word (labeled "0" in Figure 43) must be even. The absolute register number of the register containing the second word (labeled "1" in Figure 43) must be odd. If these conditions are not met, the results of the instruction are unpredictable. Note that the appropriate registers for a double-precision value in the local registers depend on the value of the Stack Pointer.

Special Floating-Point Values

The Am29005 microprocessor defines floating-point values that are encoded for special interpretation. The values are described in this section.

Not-a-Number

A Not-a-Number (NaN) is a symbolic value used to report certain floating-point exceptions. It also can be used to implement user-defined extensions to floating-point operations. A NaN comprises a floating-point number with maximum biased exponent and non-zero fraction. The sign bit can be either 0 or 1 and has no significance. There are two types of NaN: signaling NaNs and quiet NaNs. A signaling NaN causes an Invalid Operation exception if used as an input operand to a floating-point operation; a quiet NaN does not cause an exception. The Am29005 microprocessor distinguishes signaling and quiet NaNs by the most-significant bit of the fraction: a 1 indicates a quiet NaN, and a 0 indicates two signaling NaN.

An operation never generates a signaling NaN as a result. A quiet NaN result can be generated in one of two ways:

- as the result of an invalid operation that cannot generate a reasonable result, or
- as the result of an operation for which one or more input operands are either signaling or quiet NaNs.

In either case, the Am29005 microprocessor produces a quiet NaN having a fraction of 11000...0; that is, the two most-significant bits of the fraction are 11, and the remaining bits are 0. If desired, the Reserved Operand exception can be enabled to cause a Floating-Point Exception trap. The trap handler in this case can implement a scheme whereby user-defined NaN values appear to pass through operations as results, providing overall status for a series of operations.

Infinity

Infinity is an encoded value used to represent a value that is too large to be represented as a finite number in a given floating-point format. Infinity comprises a floating-point number with maximum biased exponent and zero fraction. The sign bit of an infinity distinguishes $+\infty$ from $-\infty$.

Denormalized Numbers

The IEEE Standard specifies that, wherever possible, a result that is too small to be represented as a normalized number be represented as a denormalized number. A denormalized number may be used as an input operand to any operation. For single- and double-precision formats, a denormalized number comprises a floating-point number with a biased exponent of zero and a non-

zero fraction field; the sign bit can be either 1 or 0. The value of a denormalized number is expressed by:

(–1)**s * 0.frac * 2**(–bias + 1),

where "bias" is the exponent bias for the format in question.

Zero

A zero comprises a floating-point number with a biased exponent of zero and a zero fraction field. The sign bit of a zero can be either 0 or 1; however, positive and negative zero are both exactly zero, and are considered equal by comparison operations.

External Data Accesses

All processor external accesses occur between general-purpose registers and external devices and memories. Accesses occur as the result of the execution of load and store instructions. The load and store instructions specify which general-purpose register receives the data (for a load) or supplies the data (for a store). The format of the load and store instructions is shown in Figure 44.

Addresses for accesses are given either by the content of a general-purpose register or by a constant value specified by the load or store instruction. The load and store instructions do not perform address computation directly. Any required address computations are performed explicitly by other instructions.

In the load or store instruction, the Coprocessor Enable (CE) bit (bit 23) determines whether or not the access is directed to the coprocessor. If the CE bit is 0, the access is directed to an external device or memory. If the CE bit is 1, data is transferred to or from the coprocessor. The CE bit affects the interpretation of the Control (CNTL) field as well as the channel protocol. This section deals with all external accesses other than coprocessor accesses.

The format of the instructions that do not perform coprocessor data transfers (i.e., in which the CE bit is 0) is shown in Figure 45.

In load and store instructions, the "RB or I" field specifies the address for access. The address is either the content of a general-purpose register, with register number RB, or a constant with a value I (zero-extended to 32 bits). The M bit determines whether the register or the constant is used.

The data for the access is written into the general-purpose register RA for a load, and is supplied by register RA for a store.

The definitions for other fields in the load or store instruction are given below:

Bit 23: Coprocessor Enable (CE)—The CE bit is 0 for a non-coprocessor load or store.

Bit 22: Address Space (AS)—If the AS bit is 0 for an untranslated load or store, the access is directed to instruction/data memory. If the AS bit is 1 for an untranslated load or store, the access is directed to input/output. The AS bit must be 0 for a translated load or store; if the AS bit is 1 for a translated load or store, a Protection Violation trap occurs.

Bit 21—Reserved.

Bit 20: Set Byte Pointer/Sign Bit (SB)—If the Data Width Enable (DW) bit of the Configuration Register is 0 and the SB bit is 1, the Byte Pointer Register is written

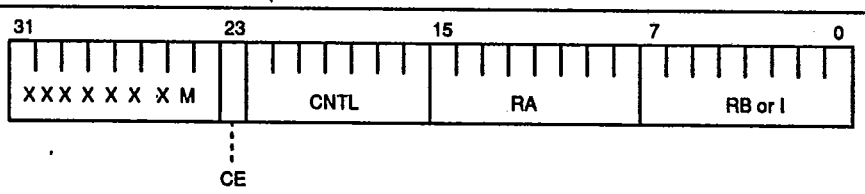


Figure 44. Load/Store Instruction Format

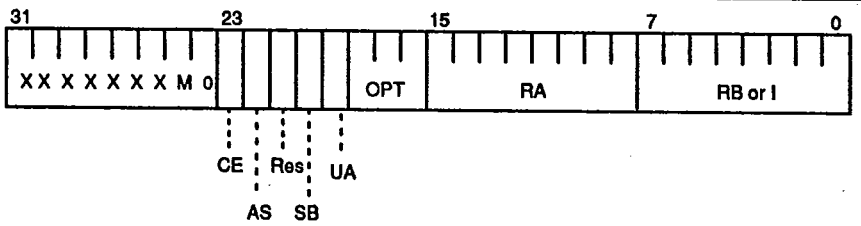


Figure 45. Non-Coprocessor Load/Store Format

T-49-17-32

with the two least-significant bits of the address for the access. These address bits can control subsequent character and half-word operations. If the BP bit is 0, the Byte Pointer Register is not affected.

If the Data Width Enable (DW) bit of the Configuration Register is 1 and the SB bit is 1 for a load, the loaded byte or half-word is sign-extended in the destination register; if the SB bit is 0, the byte or half-word is 0-extended. If the DW bit is 1 and the SB bit is 1 for either a load or store, then each bit of the Byte Pointer Register is written with the complement of the Byte Order bit of the Configuration Register. The Byte Pointer Register is set in this case to provide software compatibility across different types of memory systems. If the SB bit is 0, the Byte Pointer Register is not affected.

Bit 19: User Access (UA)—The UA bit allows programs executing in the Supervisor mode to emulate User-mode accesses. This allows checking of the authorization of an access requested by a User-mode program.

If the UA bit is 1 for a Supervisor-mode load or store, the access associated with the instruction is performed in the User mode. In this case, the User mode affects only the SUP/$\overline{US}$ output; it has no effect on the registers that can be accessed by the instruction. If the UA bit is 0, the program mode for the access is controlled by the SM bit.

If the UA bit is 1 for a User-mode load or store, a Protection Violation trap occurs.

Bits 18–16: Option (OPT)—This field is placed on the OPT_2–OPT_0 outputs during the address cycle of the access. There is a one-to-one correspondence between the OPT field and the OPT_2–OPT_0 outputs; that is, the most-significant OPT bit is placed on OPT_2, and so on.

The OPT field controls system functions as described below.

Bits 15–8: (RA)—The data for the access is written into the general-purpose register RA for a load, and is supplied by register RA for a store.

Bits 7–0: (RB or I)—In load and store instructions, the "RB or I" field specifies the address for the access. The address is either the content of a general-purpose register with register number RB, or a constant value I (0-extended to 32 bits). The M bit of the operation code (bit 24) determines whether the register or the constant is used.

Load and store operations are overlapped with the execution of instructions that follow the load or store instruction. Only one load or store may be in progress on any given cycle. If a load or store instruction is encountered while another load or store operation is in progress, the processor enters the Pipeline Hold mode until the first operation is completed. However, the address for the second operation may appear on the address bus if the first operation is to a device or memory that supports pipelined operations (see Pipelined Accesses section).

Load Operations

The processor provides the following instructions for performing load operations: Load (LOAD), Load and Lock (LOADL), Load and Set (LOADSET), and Load Multiple (LOADM). All of these instructions transfer data from an external device or memory into one or more general-purpose registers.

The LOADL instruction supports the implementation of device and memory interlocks in a multiprocessor configuration. It activates the $\overline{LOCK}$ output during the address cycle of the access.

The LOADSET instruction implements a binary semaphore. It loads a general-purpose register and automatically writes the accessed location with a word that has 1 in every bit position (that is, the write is indivisible from the read). The $\overline{LOCK}$ output is asserted during both the read and write accesses.

The LOADM loads a specified number of registers from sequential addresses, as explained below.

Load operations are overlapped with the execution of instructions that follow the load instruction. The processor detects any dependencies on the loaded data that subsequent instructions may have, and, if such a dependency is detected, enters the Pipeline Hold mode until the data are returned by the external device or memory. If a register that is the target of an incomplete load is written with the result of a subsequent instruction, the processor does not write the returning data into the register when the load is completed; the Not Needed (NN) bit in the Channel Control Register is set in this case.

Store Operations

The processor provides the following instructions for performing store operations: Store (STORE), Store and Lock (STOREL), and Store Multiple (STOREM). All of these instructions transfer data from one or more general-purpose registers to an external device or memory.

The STOREL instruction supports the implementation of device and memory interlocks in a multiprocessor configuration. It activates the $\overline{LOCK}$ output during the address cycle of the access.

The STOREM instruction stores a specified number of registers to sequential addresses, as explained below.

Store operations are overlapped with the execution of instructions that follow the store instruction. However, no data dependencies can exist since the store prevents any subsequent accesses until it is completed.

Multiple Accesses

Load Multiple (LOADM) and Store Multiple (STOREM) instructions move contiguous words of data between general-purpose registers and external devices and memories. The number of transfers is determined by the Load/Store Count Remaining Register.

The Load/Store Count Remaining (CR) field in the Load/Store Count Remaining Register specifies the number of transfers to be performed by the next LOADM or STOREM executed in the instruction sequence. The CR field is in the range of 0 to 255 and is 0-based; a count value of 0 represents one transfer, and a count value of 255 represents 256 transfers. The CR field also appears in the Channel Control Register.

Before a LOADM or STOREM is executed, the CR field is set by a Move To Special Register. A LOADM or STOREM uses the most recently written value of the CR field. If an attempt is made to alter the CR field and the Channel Control Register contains information for an external access that has not yet been completed, the processor enters the Pipeline Hold mode until the access is completed. Note that since the CR is set independently of the LOADM and STOREM, the CR field may represent a valid state of an interrupted program even if the Contents Valid (CV) bit of the Channel Control Register is 0.

Because of the pipelined implementation of LOADM and STOREM, at least one instruction (e.g., the instruction that sets the CR field) must separate two successive LOADM and/or STOREM instructions.

After the CR field is set, the execution of a LOADM or STOREM begins the data transfer. As with any other load or store operation, the LOADM or STOREM waits until any pending load or store operation is complete before starting. The LOADM instruction specifies the starting address and starting destination general-purpose register. The STOREM instruction specifies the starting address and the starting source general-purpose register.

During the execution of the LOADM or STOREM instruction, the processor updates the address and register number after every access, incrementing the address by four and the register number by one. This continues until either all accesses are completed or an inter-rupt or trap is taken.

For a Load Multiple or Store Multiple address sequence, addresses wrap from the largest possible value (hexadecimal FFFFFFFC) to the smallest possible value (hexadecimal 00000000).

The processor increments absolute register numbers during the Load Multiple or Store Multiple sequence. Absolute register numbers wrap from 127 to 128, and from 255 to 128. Thus, a sequence that begins in the global registers may make a transition to the local registers, but a sequence that begins in the local registers remains in the local registers. Also, note that the local registers are addressed circularly.

The normal restrictions on register accesses apply for the Load Multiple and Store Multiple sequences. For example, if a protected general-purpose register is encountered in the sequence for a User-mode program, a Protection Violation trap occurs.

Intermediate addresses are stored in the Channel Address Register, and register numbers are stored in the Target Register (TR) field of the Channel Control Register. For the STOREM instruction, the data for every access is stored in the Channel Data Register (this register also is set during the execution of the LOADM instruction, but has no interpretation in this case). The CR field is updated on the completion of every access so that it indicates the number of accesses remaining in the sequence.

Load Multiple and Store Multiple operations are indicated by the Multiple Operation (ML) bit in the Channel Control Register. This bit may be 1 even though the CR field has a value of 0 (indicating that one transfer remains to be performed). The ML bit is used to restart a multiple operation on an interrupt return; if it is set independently by a Move To Special Register before a load or store instruction is executed, the results are unpredictable.

While a multiple load or store is executing, the processor is in the Pipeline Hold mode, suspending any subsequent instruction execution until the multiple access is completed. If an interrupt or trap is taken, the Channel Address, Channel Data, and Channel Control registers contain the state of the multiple access at the point of interruption. The multiple access may be resumed at this point, at a later time, by an interrupt return.

The processor attempts to complete multiple accesses using the burst-mode capability of the channel (see Burst-Mode Accesses section). For this reason, multiple accesses of individual bytes and half-words are not supported. If the burst-mode access is preempted, the processor retransmits the address at the point of preemption. If the external device or memory cannot support burst-mode accesses, the processor transmits an address for every access.

The last load or store is executed as a simple access. The processor will preempt burst-mode transfer immediately prior to the last word of the transfer.

Option Bits

The Option field in the load and store instructions supports system functions, such as byte and half-word accesses. The definition of this field for a load or store, depending on the AS bit of the instruction, is as follows:

AS	OPT_2	OPT_1	OPT_0	Meaning
x	0	0	0	Word-length access
x	0	0	1	Byte access
x	0	1	0	Half-word access
0	1	0	0	Instruction ROM access (as data)
0	1	0	1	Cache control
0	1	1	0	hardware-development system accesses
	-all others -			Reserved

Note that some of these encodings do not affect processor operation, and could have other interpretations in a particular system. For example, the OPT values 000, 001, and 010 affect processor operation only if the DW bit of the Configuration Register is 1. However, nonstandard uses of the OPT field have an implication on the portability of software between different systems.

Addressing and Alignment

Address Spaces

External instructions and data are contained in one of four 32-bit address spaces:

1. Instruction/Data Memory
2. Input/Output
3. Coprocessor
4. Instruction Read-Only Memory (Instruction ROM).

It is possible to partition physical instruction and data addresses into two separate physical address spaces.

The coprocessor address space is not an address space in the strictest sense. The coprocessor address space is defined so that transfers of operands and operation codes to the coprocessor do not interfere with other external devices and memories.

The processor does not directly support the access of the instruction ROM address space using loads and stores; this capability is defined as a system option requiring external hardware.

Bits contained in load and store instructions distinguish between the instruction/data memory, input/output, and coprocessor address spaces.

For instruction fetches, the ROM Enable (RE) bit of the Current Processor Status Register distinguishes between the instruction/data and instruction ROM address spaces.

Byte and Half-Word Addressing

The Am29005 microprocessor generates word-oriented byte addresses for accesses to external devices and memories. Addresses are word-oriented because loads, stores, and instruction fetches access words. However, addresses are byte addresses because they are sufficient to select bytes packed within accessed words. For load and store operations, the processor provides means for using the least-significant address bits to access bytes and half-words within external words.

The selection of a byte within a word is determined by the two least-significant bits of an address and the Byte Order (BO) bit of the Configuration Register. The selection of a half-word within a word is determined by the next-to-least-significant bit of an address and the BO bit. Figure 46 illustrates the addressing of bytes and half-words when the BO bit is 0, and Figure 47 illustrates the addressing of bytes and half-words when the BO bit is 1. In Figure 46 and Figure 47, addresses are represented in hexadecimal notation.

In the processor, the two least-significant bits of an external address can be reflected in the Byte Pointer (BP) field of the ALU Status Register when the DW bit of the Configuration Register is 0. Alternatively, the two least-significant bits of the address can be used to control byte and half-word accesses when the DW bit is 1. The BO bit affects only the interpretation of the BP field and the two least-significant address bits.

If the BO bit is 0, bytes are ordered within words such that a 00 in the BP field or in the two least-significant address bits selects the high-order byte of a word, and a 11 selects the low-order byte. If the BO bit is 1, a 00 in the BP field or in the two least-significant address bits selects the low-order byte of a word, and a 11 selects the high-order byte.

If the BO bit is 0, half-words are ordered within words such that a 0 in the most-significant bit of the BP field or the next-to-least-significant address bit selects the high-order half-word, and a 1 selects the low-order half-word. If the BO bit is 1, a 0 in the most-significant bit of the BP field or the next-to-least-significant address bit selects the low-order half-word of a word, and a 1 selects the high-order half-word. Note that since the least-significant bit of the BP field or an address does not participate in the selection of half-words, the alignment of half-words is forced to half-word boundaries in this case.

Alignment of Words and Half-Words

Since only byte addressing is supported, it is possible that an address for the access of a word or half-word is not aligned to the desired word or half-word. The Am29005 microprocessor either ignores or forces alignment in most cases. However, some systems may require that unaligned accesses be supported for compatibility reasons. Because of this, the Am29005 microprocessor provides an option that creates a trap when a nonaligned access is attempted. This trap allows software emulation of the non-aligned accesses in a manner that is appropriate for the particular system.

The detection of unaligned accesses is activated by a 1 in the Trap Unaligned Access (TU) bit of the Current Processor Status Register. Unaligned access detection is based on the data length as indicated by the OPT field of a load or store instruction, and on the 2 least-significant bits of the specified address. Only addresses for instruction/data memory accesses are checked; alignment is ignored for input/output accesses and coprocessor transfers.

An Unaligned Access trap occurs only if the TU bit is 1 and any of the following combinations of OPT field and

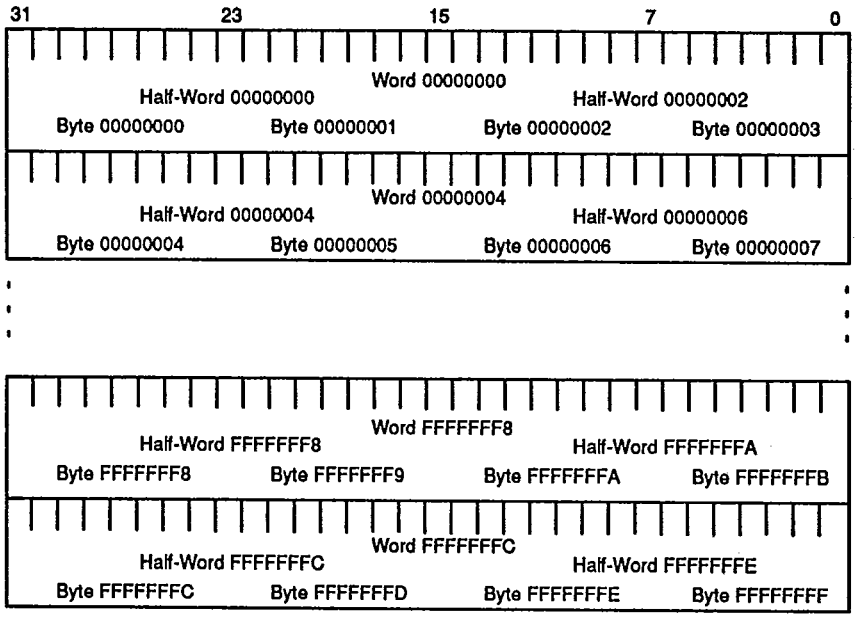


Figure 46. Byte and Half-Word Addressing with BO = 0

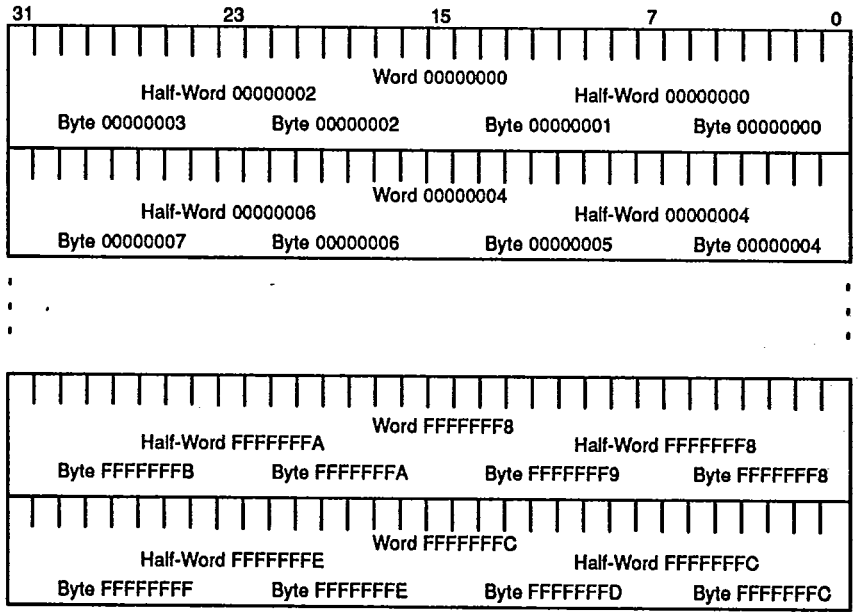


Figure 47. Byte and Half-Word Addressing with BO = 1

address bits is detected for a load or store to instruction/data memory:

OPT_2	OPT_1	OPT_0	A_1	A_0	
0	0	0	1	0	Unaligned
0	0	0	0	1	word access
0	0	0	1	1	
0	1	0	0	1	Unaligned
0	1	0	1	1	half-word access

The trap handler for the Unaligned Access trap is responsible for generating the correct sequence of aligned accesses and performing any necessary shifting, masking and/or merging. Note that a virtual page-boundary crossing also may have to be considered.

Alignment of Instructions

In the Am29005 microprocessor, all instructions are 32 bits in length, and are aligned on word-address boundaries. The processor's Program Counter is 30 bits in length, and the least-significant 2 bits of processor-generated instruction addresses are always 00. An unaligned address can be generated by indirect jumps and calls. However, alignment is ignored by the processor in this case, and it expects the system to force alignment (i.e., by interpreting the 2 least-significant address bits as 00, regardless of their values).

Accessing Instructions as Data

To aid the external access of instructions and data on separate buses, the processor distinguishes between instruction and data accesses. In systems where it is necessary to access instructions as data, this function should be performed via the shared address space. The OPT field provides a means for loads to access instructions in the Instruction read-only memory (ROM) address space. The Am29005 microprocessor does not take any action to prevent a store to the Instruction ROM address space.

Byte and Half-Word Accesses

The Am29005 microprocessor can perform byte and half-word accesses in either software or hardware under control of the Data Width Enable (DW) bit of the Configuration Register. Software byte and half-word accesses are selected by a DW bit of 0, and hardware byte and half-word accesses are selected by a DW bit of 1. Software byte and half-word accesses are less efficient than hardware byte and half-word accesses, but hardware accesses require that the system be able to selectively write individual byte and half-word positions within external devices and memories. The software-only technique is compatible with systems designed to provide hardware support for byte and half-word accesses.

This section describes the operation of both software and hardware byte and half-word accesses. Byte and half-word accesses operate as described here for memory and input/output accesses, but not for coprocessor transfers. Coprocessor transfers are unaffected by the DW bit.

The DW bit is cleared by a processor reset. It must explicitly be set to 1 by software before hardware byte and half-word accesses can be performed.

Software Byte and Half-Word Accesses

If the DW bit is 0, the Am29005 microprocessor allows the Byte Pointer Register to be set with the least-significant bits of an address specified by any load or store instruction, except those that transfer information to and from the coprocessor. Insert and extract instructions can then be used to access the byte or half-word of interest, after the external word has been accessed. This provides a general-purpose mechanism for manipulating external byte and half-word data, without the need for external hardware support.

To load a byte or half-word, a word load is first performed. This load sets the BP field with the 2 least-significant bits of the address. A subsequent EXBYTE, EXHW, or EXHWS instruction extracts the byte or half-word of interest from the accessed word.

To store a byte or half-word, a load is first performed, setting the BP field with the 2 least-significant bits of the address. A subsequent INBYTE or INHW instruction inserts the byte or half-word of interest into the accessed word, and the resulting word is then stored.

Software that relies on loads and stores setting the BP field cannot operate correctly when the Freeze (FZ) bit of the Current Processor Status Register is 1, because the ALU Status Register is frozen.

Hardware Byte and Half-Word Accesses

If the DW bit is 1 on a load, the Am29005 microprocessor selects a byte or half-word from the loaded word depending on the Option (OPT) bits of the load instruction, the Byte Order (BO) bit of the Configuration Register, and the 2 least-significant bits of the address (for bytes) or the next-to-least-significant bit of the address (for half-words). The selected byte or half-word is right-justified within the destination register. If the SB bit of the load instruction is 0, the remainder of the destination register is 0-extended. If the SB bit is 1, the remainder of the destination register is sign-extended with the sign bit of the selected byte or half-word.

If the DW bit is 1 on a store, the Am29005 microprocessor replicates the low-order byte or half-word in the source register into every byte and half-word position of the stored word. The system is responsible for generating the appropriate byte and/or half-word strobes, based on the OPT_2–OPT_0 signals and the 2 least-significant bits of the address, to write the appropriate byte or half-word in the selected device or memory (the system byte order must also be considered). The SB bit does not affect the operation of a store, except for setting the BP field as described below.

If the SB bit is 1 for either a load or store and the DW bit is also 1, both bits of the BP field are set to the complement of the BO bit when the load or store is executed. This does not directly affect the load or store access, but supports compatibility for software developed for word-write-only systems. Hardware byte and half-word accesses—in contrast to software byte and half-word accesses—can be performed when the FZ bit is 1, because these accesses do not rely on the BP field.

System Alternatives and Compatibility

The two mechanisms for performing byte and half-word accesses create the possibility of two types of systems. These are named for convenience:

- Type 1: simple, word-only accesses in external devices and memories; software byte and half-word accesses.
- Type 2: byte/half-word strobes in external devices and memories; hardware byte and half-word accesses by the Am29005 microprocessor.

The provision for hardware byte and half-word accesses encourages Type 2 systems. Software for Type 1 systems can execute on Type 2 systems, but the reverse is not true. Software compatibility is possible primarily because of the DW bit and because the Am29005 microprocessor sets the BP field with an appropriate byte pointer even when it performs byte and half-word accesses with internal hardware. Also, the system must return a full word in either type of system, regardless of the access data-width. The DW bit must be 0 in Type 1 systems and must be 1 in Type 2 systems. To illustrate compatibility between systems, consider the following steps of an unsigned byte load compiled for a Type 1 system, but executing on a Type 2 system:

1. Perform a load with OPT = 001 and SB = 1.
   - Type 1 system: The addressed word is accessed and placed into the destination register. The BP field is set with the 2 least-significant bits of the address.
   - Type 2 system: The addressed byte is accessed, aligned, padded, and placed into the destination register. The BP field is set to point to the low-order byte, reflecting the alignment that has been performed (the pointer depends on the value of the BO bit).
2. Perform a byte extract on the loaded word.
   - Type 1 system: The byte selected by the BP field is aligned to the low-order byte of the destination register and the remainder of the word is 0-extended. The selected byte may be in any byte position.
   - Type 2 system: The byte selected by the BP field (set to point to the low-order byte) is aligned to the low-order byte of the destination register and the remainder of the word is 0-extended. (Note that the selected byte was already in the low-order byte position. This operation does not change the program state but merely allows software compatibility.)

The recommended instruction sequences for all types of byte and half-word accesses and for both types of systems are enumerated below. Compatibility between these systems follows the above example, but for brevity, compatibility is not described in detail here.

Byte read, unsigned:

Type 1	Comments
load 0,17,temp,addr	; OPT = 001, SB = 1
exbyte temp,temp,0	; get byte

Type 2	Comments
load 0,1,temp,addr	; OPT = 001, SB = 0

Byte read, signed:

Type 1	Comments
load 0,17,temp,addr	; OPT = 001, SB = 1
exbyte temp,temp,0	; get byte
sll temp,temp,24	; sign extend
sra temp,temp,24	

Type 2	Comments
load 0,17,temp,addr	; OPT = 001, SB = 1 (sign extended)

Byte Write:

Type 1	Comments
load 0,17,temp,addr	; OPT = 001, SB = 1
inbyte temp,temp, data	; insert byte
store 0,1,temp,addr	; store

Type 2	Comments
store 0,1,data,addr	; OPT = 001, SB = 0

Half-word read, unsigned:

Type 1	Comments
load 0,18,temp,addr	; OPT = 010, SB = 1
exhw temp,temp,0	; get half-word unsigned

Type 2	Comments
load 0,2,temp,addr	; OPT = 010, SB = 0

Half-word read, signed:

Type 1	Comments
load 0,18,temp,addr	; OPT = 010, SB = 1
exhws temp,temp	; get half-word sign-extend

Type 2	Comments
load 0,18,temp,addr	; OPT = 010, SB = 1 (sign-extend)

Half-word write:

Type 1	Comments
load 0,18,temp,addr	; OPT = 010, SB = 1
inhw temp,temp,data	; insert half-word
store 0,2,temp,addr	; store

Type 2	Comments
store 0,2,data,addr	; OPT = 010, SB = 0

INTERRUPTS AND TRAPS

Interrupts and traps cause the Am29005 microprocessor to suspend the execution of an instruction sequence and to begin the execution of a new sequence. The processor may or may not later resume the execution of the original instruction sequence.

The distinction between interrupts and traps is largely one of causation and enabling. Interrupts allow external devices and the Timer Facility to control processor execution, and are always asynchronous to program execution. Traps are intended to be used for certain exceptional events that occur during instruction execution, and are generally synchronous to program execution.

Throughout this manual, a distinction is made between the point at which an interrupt or trap occurs and the point at which it is taken. An interrupt or trap is said to occur when all conditions that define the interrupt or trap are met. However, an interrupt or trap that occurs is not necessarily recognized by the processor, either because of various enables or because of the processor's operational mode (e.g., Halt mode). An interrupt or trap is taken when the processor recognizes the interrupt or trap and alters its behavior accordingly.

Interrupts

Interrupts are caused by signals applied to any of the external inputs $\overline{INTR_3}$–$\overline{INTR_0}$, or by the Timer Facility. The processor may be disabled from taking certain interrupts by the masking capability provided by the Disable All Interrupts and Traps (DA) bit, Disable Interrupts (DI) bit, and Interrupt Mask (IM) field in the Current Processor Status Register.

The DA bit disables all interrupts and most traps. The DI bit disables external interrupts without affecting the recognition of traps and Timer interrupts. The 2-bit IM field selectively enables external interrupts as follows:

IM Value	Result
0 0	$\overline{INTR_0}$ enabled
0 1	$\overline{INTR_1}$–$\overline{INTR_0}$ enabled
1 0	$\overline{INTR_2}$–$\overline{INTR_0}$ enabled
1 1	$\overline{INTR_3}$–$\overline{INTR_0}$ enabled

Note that the $\overline{INTR_0}$ interrupt cannot be disabled by the IM field. Also, note that no external interrupt is taken if either the DA or DI bit is 1. The Interrupt Pending bit in the Current Processor Status indicates that one or more of the signals $\overline{INTR_3}$–$\overline{INTR_0}$ is active, but that the corresponding interrupt is disabled due to the value of either DA, DI, or IM.

Traps

Traps are caused by signals applied to one of the inputs $\overline{TRAP_1}$–$\overline{TRAP_0}$, or by exceptional conditions such as protection violations. Except for the Instruction Access Exception, Data Access Exception, and Coprocessor Exception traps, traps are disabled by the DA bit in the Current Processor Status; a 1 in the DA bit disables traps, and a 0 enables traps. It is not possible to selectively disable individual traps.

Wait Mode

A wait-for-interrupt capability is provided by the Wait mode. The processor is in the Wait mode whenever the Wait Mode (WM) bit of the Current Processor Status is 1. While in Wait mode, the processor neither fetches nor executes instructions and performs no external accesses. The Wait mode is exited when an interrupt or trap is taken.

Note that the processor can take only those interrupts or traps for which it is enabled, even in the Wait mode. For example, if the processor is in the Wait mode with a DA bit of 1, it can leave the Wait mode only via the Reset mode or a $\overline{WARN}$ trap.

Vector Area

Interrupt and trap processing rely on the existence of a user-managed Vector Area in external instruction/data memory or instruction read-only memory (instruction ROM). The Vector Area begins at an address specified by the Vector Area Base Address Register, and provides for as many as 256 different interrupt and trap handling routines. The processor reserves 24 routines for system operation and 40 routines for instruction emulation. The number and definition of the remaining 192 possible routines are system-dependent.

The Vector Area has one of two possible structures as determined by the Vector Fetch (VF) bit in the Configuration Register. The first structure, as described below, requires less external memory than the second, but imposes the performance penalty of the vector-table lookup.

If the VF bit is 1, the structure of the Vector Area is a table of vectors in instruction/data memory. The layout of a single vector is shown in Figure 48. Each vector gives the beginning word-address of the associated interrupt or trap handling routine, and specifies, by the R bit, whether the routine is contained in instruction/data memory (R = 0) or instruction ROM (R = 1).

If the VF bit is 0, the structure of the Vector Area is a segment of contiguous blocks of instructions in instruction/data memory or instruction ROM. The ROM Vector Area (RV) bit of the Configuration Register determines whether the Vector Area is in instruction/data memory (RV = 0) or instruction ROM (RV = 1). A 64-instruction block contains exactly one interrupt or trap handling routine, and blocks are aligned on 64-instruction address boundaries.

Vector Numbers

When an interrupt or trap is taken, the processor determines an 8-bit vector number associated with the interrupt or trap. The vector number gives either the number of a vector table entry or the number of an instruction block, depending on the value of the VF bit.

If the VF bit is 1, the physical address of the vector table entry is generated by replacing bits 9–2 of the value in the Vector Area Base Address Register with the vector number.

If the VF bit is 0, the physical address of the first instruction of the handling routine is generated by replacing bits 15–8 of the value in the Vector Table Base Address Register with the vector number.

Vector numbers are either predefined or specified by an instruction causing the trap. The assignment of vector numbers is shown in Figure 49 (vector numbers are in decimal notation). Vector numbers 64 to 255 are for use by trapping instructions; the definition of the routines associated with these numbers is system-dependent.

Interrupt and Trap Handling

Interrupt and trap handling consists of two distinct operations: taking the interrupt or trap, and returning from the interrupt or trap handler. If the interrupt or trap handler returns directly to the interrupted routine, the interrupt or trap handler need not save and restore processor state.

Taking an Interrupt or Trap

The following operations are performed in sequence by the processor when an interrupt or trap is taken:

1. Instruction execution is suspended.
2. Instruction fetching is suspended.
3. Any in-progress load or store operation is completed. Any additional operations are canceled in the case of Load Multiple and Store Multiple.
4. The contents of the Current Processor Status Register are copied into the Old Processor Status Register.
5. The Current Processor Status register is modified as shown in Figure 50 (the value "u" means unaffected). Note that setting the Freeze (FZ) bit freezes the Channel Address, Channel Data, Channel Control, Program Counter 0, Program Counter 1, Program Counter 2, and ALU Status Registers.
6. The address of the first instruction of the interrupt or trap handler is determined. If the VF bit of the Configuration Register is 1, the address is obtained by accessing a vector from instruction/data memory, using the physical address obtained from the Vector Area Base Address Register and the vector number. This access appears on the channel as a data access, and the OPT_2–OPT_0 signals indicate a word-length access. If the VF bit is 0, the instruction address is given directly by the Vector Area Base Address Register and the vector number.

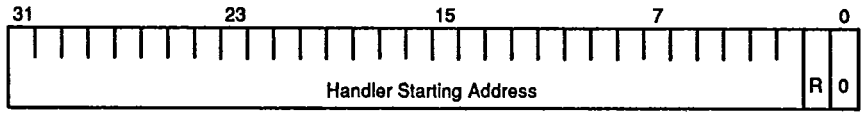


Figure 48. Vector Table Entry

Number	Type of Trap or Interrupt	Cause
0	Illegal Opcode	Executing undefined instruction
1	Unaligned Access	Access on unnatural boundary, TU = 1
2	Out of Range	Overflow or underflow
3	Coprocessor Not Present	Coprocessor access, CP = 0
4	Coprocessor Exception	Coprocessor $\overline{DERR}$ response
5	Protection Violation	Invalid User-mode operation
6	Instruction Access Exception	$\overline{IERR}$ response
7	Data Access Exception	$\overline{DERR}$ response, not coprocessor
8	Reserved	
9	"	
10	"	
11	"	
12	"	
13	"	
14	Timer	Timer Facility
15	Trace	Trace Facility
16	$\overline{INTR_0}$	$\overline{INTR_0}$ input
17	$\overline{INTR_1}$	$\overline{INTR_1}$ input
18	$\overline{INTR_2}$	$\overline{INTR_2}$ input
19	$\overline{INTR_3}$	$\overline{INTR_3}$ input
20	$\overline{TRAP_0}$	$\overline{TRAP_0}$ input
21	$\overline{TRAP_1}$	$\overline{TRAP_1}$ input
22	Floating-Point Exception	Unmasked floating-point exception
23	Reserved	
24–29	Reserved for instruction emulation (Op codes D8–DD)	
30	MULTM	MULTM instruction
31	MULTMU	MULTMU instruction
32	MULTIPLY	MULTIPLY instruction
33	DIVIDE	DIVIDE instruction
34	MULTIPLU	MULTIPLU instruction
35	DIVIDU	DIVIDU instruction
36	CONVERT	CONVERT instruction
37	SQRT	SQRT instruction
38	CLASS	CLASS instruction
39–41	Reserved for instruction emulation (Op codes E7–E9)	
42	FEQ	FEQ instruction
43	DEQ	DEQ instruction
44	FGT	FGT instruction
45	DGT	DGT instruction
46	FGE	FGE instruction
47	DGE	DGE instruction
48	FADD	FADD instruction
49	DADD	DADD instruction
50	FSUB	FSUB instruction
51	DSUB	DSUB instruction
52	FMUL	FMUL instruction
53	DMUL	DMUL instruction
54	FDIV	FDIV instruction
55	DDIV	DDIV instruction
56	Reserved for instruction emulation (Op code F8)	
57	FDMUL	FDMUL instruction
58–63	Reserved for instruction emulation (Op codes FA-FF)	
64–255	Assert and EMULATE instruction traps (vector number specified by instruction)	

Figure 49. Vector Number Assignments

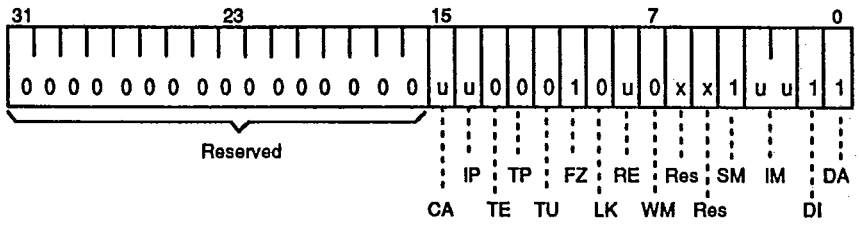


Figure 50. Current Processor Status after an Interrupt or Trap

7. If the VF bit is 1, the R bit in the vector fetched in Step 6 is copied into the RE bit of the Current Processor Status Register. If the VF bit is 0, the RV bit of the Configuration Register is copied into the RE bit. This step determines whether or not the first instruction of the interrupt handler is in Instruction ROM.
8. An instruction fetch is initiated using the instruction address determined in Step 6. At this point, normal instruction execution resumes.

Note that the processor does not explicitly save the contents of any registers when an interrupt is taken. If register saving is required, it is the responsibility of the interrupt or trap-handling routine. For proper operation, registers must be saved before any further interrupts or traps may be taken. The FZ bit must be reset at least two instructions before interrupts or traps are reenabled to allow the program state to be reflected properly in processor registers if an interrupt or trap is taken.

Returning from an Interrupt or Trap

An Interrupt Return (IRET) is used to resume the execution of an interrupted program. In some situations, the processor state must be set properly by software before the interrupt return is executed. The following is a list of operations normally performed in such cases:

1. The Current Processor Status is configured as shown in Figure 50 (the value "x" is a "don't care"). Note that setting the FZ bit freezes the registers listed below so that they may be set for the interrupt return.
2. The Old Processor Status is set to the value of the Current Processor Status for the target routine.
3. The Channel Address, Channel Data, and Channel Control registers are set to restart or resume uncompleted channel operations of the target routine.
4. The Program Counter 1 and Program Counter 0 registers are set to the addresses of the first and second instructions, respectively, to be executed in the target routine.
5. Other registers are set as required. These may include registers such as the ALU Status, Q, and so forth, depending on the particular situation. Some of these registers are unaffected by the FZ bit, so they must be set in such a manner that they are not modified unintentionally before the interrupt return.

Once the processor registers are configured properly, as described above, an interrupt return instruction (IRET) performs the remaining steps necessary to return to the target routine. The following operations are performed by the interrupt return instruction:

1. Any in-progress load or store operation is completed. If a Load Multiple or Store Multiple sequence is in progress, the interrupt return is not executed until the sequence is completed.

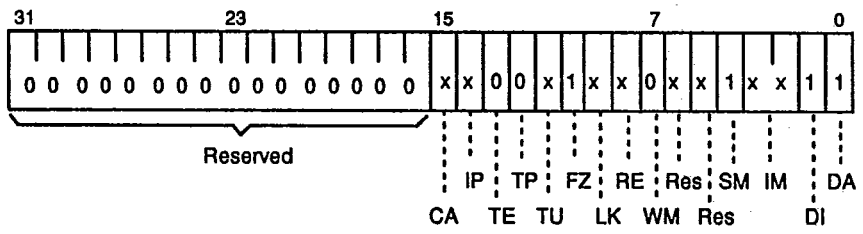


Figure 51. Current Processor Status Before Interrupt Return

2. Interrupts and traps are disabled, regardless of the settings of the DA, DI, and IM fields of the Current Processor Status, for Steps 3 through 9.
3. The contents of the Old Processor Status Register are copied into the Current Processor Status Register. This normally resets the FZ bit allowing the Program Counter 0, 1, 2, Channel Address, Data, Control, and ALU Status registers to update normally. Since certain bits of the Current Processor Status Register always are updated by the processor, this copy operation may be irrelevant for certain bits (e.g., the Interrupt Pending bit).
4. If the Contents Valid (CV) bit of the Channel Control Register is 1, and the Not Needed (NN) and Multiple Operation (ML) bits are both 0, an external access is started. This operation is based on the contents of the Channel Address, Channel Data, and Channel Control registers. The Current Processor Status Register conditions the access—as is normally the case. Note that Load Multiple and Store Multiple operations are not restarted at this point.
5. The address in Program Counter 1 is used to fetch an instruction. The Current Processor Status Register conditions the fetch.
6. The instruction fetched in Step 5 enters the decode stage of the pipeline.
7. The address in Program Counter 0 is used to fetch an instruction. The Current Processor Status Register conditions the fetch.
8. The instruction fetched in Step 5 enters the execute stage of the pipeline, and the instruction fetched in Step 7 enters the decode stage.
9. If the CV bit in the Channel Control Register is a 1, the NN bit is 0, and the ML bit is 1, a Load Multiple or Store Multiple sequence is started, based on the contents of the Channel Address, Channel Data, and Channel Control registers.
10. Interrupts and traps are enabled per the appropriate bits in the Current Processor Status Register.
11. The processor resumes normal operation.

Fast Interrupt Processing

The registers affected by the FZ bit of the Current Processor Status Register are those that are modified by almost any usual sequence of instructions. Since the FZ bit is set by an interrupt or trap, the interrupt or trap handler is able to execute while not disturbing the state of the interrupted routine, though its execution is somewhat restricted. Thus, it is not necessary in many cases for the interrupt or trap handler to save the registers that are affected by the FZ bit.

The processor provides an additional benefit if the Program Counter 0 and Program Counter 1 registers are not modified by the interrupt or trap handler. If Program Counters 0 and 1 contain the addresses of sequential instructions when an interrupt or trap is taken, and if they are not modified before an interrupt return is executed, Step 8 of the interrupt return sequence above occurs as a sequential fetch—instead of a branch—for the interrupt return. The performance impact of a sequential fetch is normally less than that of a nonsequential fetch.

Because the registers affected by the FZ bit are sometimes required for instruction execution, it is not possible for the interrupt or trap handler to execute all instructions unless the required registers are first saved elsewhere (e.g., in one or more global registers). Most of the restrictions due to register dependencies are obvious (e.g., the Byte Pointer for byte extracts), and will not be discussed here. Other less obvious restrictions are listed below:

1. Load Multiple and Store Multiple. The Channel Address, Channel Data, and Channel Control registers are used to sequence Load Multiple and Store Multiple operations, so these instructions cannot be executed while the registers are frozen. However, note that other external accesses may occur; the Channel Address, Channel Data, and Channel Control registers are required only to restart an access after an exception, and the interrupt or trap handler is not expected to encounter any exceptions.
2. Loads and stores that set the Byte Pointer. If the Set Byte Pointer (SB) of a load or store instruction is 1 and the FZ bit is also 1, there is no effect on the Byte Pointer. Thus, the execution of external byte and half-word accesses using this mechanism is not possible.
3. Extended arithmetic. The Carry bit of the ALU Status Register is not updated while the FZ bit is 1.
4. Divide step instructions. The Divide Flag of the ALU Status Register is not updated when the FZ bit is 1.

If the interrupt or trap handler does not save the state of the interrupted routine, it cannot allow additional interrupts and traps. Also, the operation of the interrupt or trap handler cannot depend on any trapping instructions (e.g., Floating-Point instructions, illegal operation codes, arithmetic overflow, etc.) since these are disabled. There are certain cases, however, where traps are unavoidable; these are discussed in the Arithmetic Exceptions section.

$\overline{\text{WARN}}$ Trap

The processor recognizes a special trap, caused by the activation of the $\overline{\text{WARN}}$ input, that cannot be masked. The $\overline{\text{WARN}}$ trap is intended to be used for severe

system-error or deadlock conditions. It allows the processor to be placed in a known, operable state, while preserving much of its original state for error reporting and possible recovery. Therefore, it shares some features in common with the Reset mode as well as features common to other traps described in this section.

The major differences between the $\overline{\text{WARN}}$ trap and other traps are:

1. The processor does not wait for an in-progress external access to be completed before taking the trap, since this access might not be completed. However, the information related to any outstanding access is retained by the Channel Address, Channel Data, and Channel Control registers when the trap is taken.
2. The vector-fetch operation is not performed, regardless of the VF bit of the Configuration Register, when the $\overline{\text{WARN}}$ trap is taken. Instead, the ROM Enable (RE) bit in the Current Processor Status is set, and instruction fetching begins immediately at Address 16 in the instruction ROM. The trap handler executes directly from the instruction ROM without the need to access external (and possibly nonfunctional or invalid) instruction/data memory.

Note that $\overline{\text{WARN}}$ trap may disrupt the state of the routine that is executing when it is taken, prohibiting this routine from being restarted.

Sequencing of Interrupts and Traps

On every cycle, the processor decides either to execute instructions or to take an interrupt or trap. Since there are multiple sources of interrupts and traps, more than one interrupt or trap may be pending on a given cycle.

To resolve conflicts, interrupts and traps are taken according to the priority shown in Figure 52. In this table, interrupts and traps are listed in order of decreasing priority. This section discusses the first three columns of Figure 52. The last two columns are discussed in the Exception Reporting and Restarting section.

In Figure 52, interrupts and traps fall into one of two categories depending on the timing of their occurrence relative to instruction execution. These categories are indicated in the third column by the labels "inst" and "async." These labels have the following meanings:

1. Inst—Generated by the execution or attempted execution of an instruction.
2. Async—Generated asynchronous to and independent of the instruction being executed, although it may be a result of an instruction executed previously.

The principle for interrupt and trap sequencing is that the highest priority interrupt or trap is taken first. Other interrupts and traps remain active until they can be taken, or are regenerated when they can be taken. This is accomplished, depending on the type of interrupt or trap, as follows:

1. All traps in Figure 52 with Priority 13 or 14 are regenerated by the re-execution of the causing instruction.
2. Most of the interrupts and traps of Priorities 4 through 12 must be held by external hardware until they are taken. The exceptions to this are listed in (3) below.
3. The exceptions to (2) above are the Data Access Exception trap, the Coprocessor Exception trap, the Timer interrupt, and the Trace trap. These are caused by bits in various registers in the processor and are held by these registers until taken or cleared. The relevant bits are: the Transaction Faulted (TF) bit of the Channel Control Register for Data Access Exception and Coprocessor Exception traps, the Interrupt (IN) bit of the Timer Reload Register for Timer interrupts, and the Trace Pending (TP) bit of the Current Processor Status Register for Trace traps.
4. All traps of Priorities 2 and 3 in Figure 52, except for the Unaligned Access trap, are not regenerated. These traps are mutually exclusive and are given high priority because they cannot be regenerated; they must be taken if they occur. If one of these traps occurs at the same time as a reset or $\overline{\text{WARN}}$ trap, it is not taken, and its occurrence is lost.
5. The Unaligned Access trap is regenerated internally when an external access is restarted by the Channel Address, Channel Data, and Channel Control registers. Note that this trap is not necessarily exclusive to the traps discussed in (4) above.

Note that the Channel Address, Channel Data, and Channel Control registers are set for a $\overline{\text{WARN}}$ trap only if an external access is in progress when the trap is taken.

Exception Reporting and Restarting

When an instruction encounters an exceptional condition, the Program Counter 0, Program Counter 1, and Program Counter 2 registers report the relevant instruction address(es), and allow the instruction sequence to be restarted once the exceptional condition has been remedied (if possible). Similarly, when an external access or coprocessor transfer encounters an exceptional condition, the Channel Address, Channel Data, and Channel Control registers report information on the access or transfer, and allow it to be restarted. This section describes the interpretation and use of these registers.

The "PC1" column in Figure 52 describes the value held in the Program Counter 1 Register (PC1) when the interrupt or trap is taken. For traps in the "inst" category, PC1

Priority	Type Of Interrupt Or Trap	Inst/Async	PC1	Channel Regs
1 (highest)	$\overline{\text{WARN}}$	async	next	see Note
2	Unaligned Access	inst	next	all
	Coprocessor not Present	inst	next	all
	Out of Range	inst	next	N/A
	Floating-Point Exceptions	inst	next	N/A
	Assert Instructions	inst	next	N/A
	Floating-Point Instructions	inst	next	N/A
	MULTIPLY	inst	next	N/A
	MULTM	inst	next	N/A
	DIVIDE	inst	next	N/A
	MULTIPLU	inst	next	N/A
	MULTMU	inst	next	N/A
	DIVIDU	inst	next	N/A
	EMULATE	inst	next	N/A
3	Data Access Exception	async	next	all
	Coprocessor Exception	async	next	all
4	$\overline{\text{TRAP}}_0$	async	next	multiple
5	$\overline{\text{TRAP}}_1$	async	next	multiple
6	$\overline{\text{INTR}}_0$	async	next	multiple
7	$\overline{\text{INTR}}_1$	async	next	multiple
8	$\overline{\text{INTR}}_2$	async	next	multiple
9	$\overline{\text{INTR}}_3$	async	next	multiple
10	Timer	async	next	multiple
11	Trace	async	next	multiple
12	Instruction Access Violation	inst	curr	N/A
13 (lowest)	Illegal Opcode	inst	curr	N/A
	Protection Violation	inst	curr	N/A

Note: The Channel Address, Channel Data, and Channel Control registers are set for a $\overline{\text{WARN}}$ trap only if an external access is in progress when the trap is taken.

Figure 52. Interrupt and Trap Priority Table

contains either the address of the instruction causing the trap, indicated by "curr," or the address of the instruction following the instruction causing the trap, indicated by "next."

For interrupts and traps in the "async" category, PC1 contains the address of the first instruction, which was not executed due to the taking of the interrupt or trap. This is the next instruction to be executed upon interrupt return, as indicated by "next" in the PC1 column.

Instruction Exceptions

For traps caused by the execution of an instruction (e.g., the Out of Range trap), the Program Counter 2 Register contains the address of the instruction causing the trap. In all of these cases, PC1 is in the "next" category. The Exception Opcode Register contains the operation code of the instruction causing the trap.

The traps associated with instruction fetches (i.e., those of Priority 13) occur only if the processor attempts the execution of the associated instruction. An exception may be detected during an instruction prefetch, but the associated trap does not occur if a nonsequential fetch occurs before the processor attempts the execution of the invalid instruction. This prevents the spurious indication of instruction exceptions.

Data Exceptions

The "Channel Regs" column of Figure 52 indicates the cases for which the Channel Address, Channel Data, and Channel Control registers contain information related to an external access or coprocessor transfer (these registers collectively are termed "channel registers" in the following discussion). For the cases indicated, the access or transfer was not completed because of some exceptional condition. Note that the Channel Data Register contains relevant information only in the case of a store.

For the $\overline{\text{WARN}}$ trap, the channel registers are valid only if a load or store were in progress when the trap was taken. Recall that the $\overline{\text{WARN}}$ trap does not wait for any in-progress access to be completed.

For the traps with an "all" in the "Channel Regs" column of Figure 52, the channel registers contain information relevant to the trap in all cases. These traps are associated with exceptional events during external accesses or coprocessor transfers.

For the traps with a "multiple" in the "Channel Regs" column, the channel registers might contain information for restarting an interrupted Load Multiple or Store Multiple operation. In these cases, the operation did not encounter an exception, but was simply canceled for latency considerations.

The information contained in the channel registers allows the processor to restart the related operation during an interrupt return sequence, without any special assistance by software. Software must only ensure that the relevant information is retained in, or restored to, the channel registers before an interrupt return is executed.

Arithmetic Exceptions

Integer and floating-point instructions can cause Out of Range or Floating-Point Exception traps, respectively, if an exception is detected during the arithmetic operation. This section describes the conditions under which these traps occur and the additional operations performed beyond those described in the Interrupt and Trap Handling section.

Integer Exceptions

Some integer add and subtract instructions—ADDS, ADDU, ADDCS, ADDCU, SUBS, SUBU, SUBCS, SUBCU, SUBRS, SUBRU, SUBRCS, and SUBRCU—cause an Out of Range trap upon overflow or underflow of a 32-bit signed or unsigned result, depending on the instruction.

Two integer multiply instructions—MULTIPLY and MULTIPLU—cause an Out of Range trap upon overflow of a 32-bit signed or unsigned result, respectively, if the MO bit of the Integer Environment Register is 0. If the MO bit is 1, these multiply instructions cannot cause an Out of Range trap.

Two integer divide instructions—DIVIDE and DIVIDU—take the Out of Range trap upon overflow of a 32-bit signed or unsigned result, respectively, if the DO bit of the Integer Environment Register is 0. If the DO bit is 1, the divide instructions cannot cause an Out of Range trap unless the divisor is zero. If the divisor is zero, an Out of Range trap always occurs, regardless of the DO bit.

In addition to the operations described in the Interrupt and Trap Handling section, the following operations are performed when an Out of Range trap is taken:

1. The operation code of the instruction causing the exception is placed in the IOP field of the Exception Opcode Register.
2. For the MULTIPLY, MULTIPLU, DIVIDE, and DIVIDU instructions, the absolute register numbers of the excepting instruction's source and destination registers are placed into the Indirect Pointer A, Indirect Pointer B, and Indirect Pointer C registers.
3. For the MULTIPLY, MULTIPLU, DIVIDE, and DIVIDU instructions, the destination register or registers are unchanged.

Floating-Point Exceptions

A Floating-Point Exception trap occurs when an exception is detected during a floating-point operation, and the exception is not masked by the corresponding bit of the Floating-Point Mask Register. In this context, a floating-point operation is defined as any operation that accepts a floating-point number as a source operand, that produces a floating-point result, or both. Thus, for example, the CONVERT instruction may create an exception while attempting to convert a floating-point value to an integer value.

In addition to the operations described in the Interrupt and Trap Handling section, the following operations are performed when a Floating-Point Exception trap is taken:

1. The operation code of the instruction causing the exception is placed in the IOP field of the Exception Opcode Register.
2. The status of the trapping operation is written into the trap status bits of the Floating-Point Status Register. The status bits that are written do not depend on the values of the corresponding mask bits in the Floating-Point Environment Register.
3. The absolute register numbers of the excepting instruction's source and destination registers are placed into the Indirect Pointer A, Indirect Pointer B, and Indirect Pointer C registers. If the RB or RC fields specify a function code, that

code is transferred to the corresponding indirect pointer. Note that if the most-significant bit of this function code is 1, the value of the Stack Pointer has been added to the RB field and must be subtracted to recover the original field.

4. The destination register or registers are left unchanged.

Exceptions During Interrupt and Trap Handling

In most cases, interrupt and trap handling routines are executed with the DA bit in the Current Processor Status having a value of 1. It is assumed that these routines do not create many of the exceptions possible in most other processor routines, so most of these are ignored.

If the assumption of no exceptions is not valid for a particular interrupt or trap handler, it is important that the handler save the state of the processor and reset the FZ bit of the Current Processor Status, so that the handler itself may be restarted properly. This must be accomplished before any interrupts or traps can be taken. In this case, the state (or the state of some other process) must be restored before an interrupt return is executed.

It is possible that errors reported via the $\overline{IERR}$ and $\overline{DERR}$ signals are associated with hardware errors, independent of any routine being executed. For this reason, the Instruction Access Exception, Data Access Exception, and Coprocessor Exception traps cannot be disabled by the DA bit, and the processor may take one of these traps even while handling another interrupt or trap.

If the processor does take an unmaskable trap while handling another interrupt or trap, and the state of the interrupt or trap handler is not reflected in processor registers, it is not possible to return to the point at which the unmaskable trap is taken. When the unmaskable trap is taken, the processor state saved is that state associated with the original interrupt or trap, not with the unmaskable trap; however, the Old Processor Status Register is modified to reflect the Current Processor Status Register of the interrupt or trap handler. This situation, indicated by the DA bit being 1 in the Old Processor Status Register, may not be recoverable.

CHANNEL DESCRIPTION

The processor channel provides the bandwidth required for performance, while permitting the connection of many different types of devices. This section describes the channel and methods of connecting devices and memories to the processor.

The channel consists of three 32-bit synchronous buses with associated control and status signals: the Address Bus, Data Bus, and Instruction Bus. The Address Bus transfers addresses and control information to devices and memories. The Data Bus transfers data to and from devices and memories. The Instruction Bus transfers instructions to the processor from instruction memories. In addition, a set of signals allows control of the channel to be relinquished to an external master.

There are five logical groups of signals performing five distinct functions, as follows (since some signals perform more than one function, a signal may appear in more than one group):

1. Instruction Address Transfer and Instruction Access Requests: A_{31}–A_0, SUP/$\overline{US}$, $\overline{PEN}$, $\overline{IREQ}$, IREQT, $\overline{PIA}$, $\overline{BINV}$
2. Instruction Transfer: I_{31}–I_0, $\overline{IBREQ}$, $\overline{IRDY}$, $\overline{IERR}$, $\overline{IBACK}$
3. Data Address Transfer and Data Access Requests: A_{31}–A_0, R/$\overline{W}$, SUP/$\overline{US}$, $\overline{LOCK}$, $\overline{PEN}$, $\overline{DREQ}$, $DREQT_1$–$DREQT_0$, OPT_2–OPT_0, $\overline{PDA}$, $\overline{BINV}$
4. Data Transfer: D_{31}–D_0, $\overline{DBREQ}$, $\overline{DRDY}$, $\overline{DERR}$, $\overline{DBACK}$, $\overline{CDA}$
5. Arbitration: $\overline{BREQ}$, $\overline{BGRT}$, $\overline{BINV}$

User-Defined Signals

There are user-defined outputs on the processor to control devices and memories directly in a system-dependent manner. Each of these outputs is valid simultaneously with—and for the same duration as—the address for an access.

The set of signals OPT_2–OPT_0 is determined by bits 18–16 of the load or store instruction that initiates an access. These signals are valid only for data accesses, and have a predefined interpretation for coprocessor data transfers.

Standard interpretations of OPT_2–OPT_0 are given in the Pin Description section. Since the OPT_2–OPT_0 signals are determined by instructions, they have an impact on application-software compatibility, and system hardware should use the given definitions of OPT_2–OPT_0. The OPT_2–OPT_0 signals are used to encode byte and half-word accesses. However, for a load, the system should return an entire aligned word, regardless of the indicated data width.

Note that the standard interpretations of OPT_2–OPT_0 apply only to accesses to instruction/data memory and input/output. Other interpretations may be used for coprocessor transfers.

For interrupt and trap vector fetches, the OPT_2–OPT_0 outputs are all Low.

Instruction Accesses

Instruction accesses occur to one of two address spaces: instruction/data memory and instruction read-only memory (instruction ROM). The distinction between these address spaces is made by the IREQT signal, which is in turn derived from the ROM Enable (RE)

bit of the Current Processor Status Register. These are truly distinct address spaces; each may be populated independently based on the needs of a particular system.

Instruction/data memory contains both instructions and data. In certain systems, it may be required to access instructions via loads and stores, even though instructions may be contained in physically separate memories. For example, this requirement might be imposed because of the need to load instructions into memory. Note also that the OPT_2–OPT_0 signals may be used to allow the access of instructions in instruction ROM, using loads; the Am29005 microprocessor does not prevent a store to the instruction ROM, and protection against stores to the instruction ROM must be provided externally, if required.

All processor instruction fetches are read accesses, and the R/$\overline{W}$ signal is High for all instruction fetches.

Data Accesses

Data accesses occur to one of three address spaces: instruction/data memory, input/output (I/O), and the coprocessor. The distinction between these spaces is made by the $DREQT_1$–$DREQT_0$ signals, which are in turn determined by the load or store instruction that initiates a data access. Each of these address spaces is distinct from the others.

The protocol for data transfers to and from the coprocessor is slightly different than the protocol for instruction/data memory and I/O accesses.

Data accesses may occur either from a slave device or memory to the processor (for a load), or from the processor to a slave device or memory (for a store). The direction of transfer is determined by the R/$\overline{W}$ signal. In the case of a load, the processor requires that data on the data bus be held valid only for a short time before the end of a cycle. In the case of a store, the processor drives the data bus as soon as the bus is available and holds the data valid until the slave device or memory signals that the access is complete.

Reporting Errors

The successful completion of an instruction access is indicated by an active level on the $\overline{IRDY}$ input, and the successful completion of a data access is indicated by an active level on the $\overline{DRDY}$ input. If there are exceptional conditions for which an instruction or data access cannot be completed successfully, the unsuccessful completion is indicated by an active level on the $\overline{IERR}$ or $\overline{DERR}$ input, as appropriate.

If the processor receives an $\overline{IERR}$ or $\overline{DERR}$ in response to an instruction or data access, it ignores the content of the instruction or data bus and the value of $\overline{IRDY}$ or $\overline{DRDY}$. An $\overline{IERR}$ response causes an Instruction Access Exception trap, unless it is associated with an instruction that the processor does not ultimately execute (because of a nonsequential instruction fetch). A $\overline{DERR}$ response always causes either a Data Access Exception trap or a Coprocessor Exception Trap.

The processor supports the restarting of unsuccessful accesses upon an interrupt return. In the case of an unsuccessful instruction access, the restart is performed by the Program Counter 0 and Program Counter 1 registers. In the case of an unsuccessful data access, the restart is performed by the Channel Address, Channel Data, and Channel Control registers. In any event, the control program must determine whether or not an access can and/or should be restarted.

The Instruction Access Exception and Data Access Exception traps cannot be masked. If one of these traps occurs within an interrupt or trap handler, the processor state may not be recoverable.

Access Protocols

Figure 53 shows a control flowchart for accesses performed by the Am29005 microprocessor. This control flow applies independently to both instruction and data accesses. Since the processor performs concurrent instruction and data accesses, these accesses may be at different points in the control flow at any given point in time.

Note that the items on the flowchart of Figure 53 do not represent actual states and have no particular relationship to processor cycles. The flowchart provides only a high-level understanding of the control flow. Also, exceptions and error conditions are not shown.

The channel supports three protocols for accesses: simple, pipelined, and burst-mode. These are described in the following sections. The various protocols are defined to accommodate minimum-latency accesses as well as maximum-transfer-rate accesses. The protocols allow an access to complete in a single cycle, although they support accesses requiring arbitrary numbers of cycles. Address transfers for accesses may be independent of instruction or data transfers.

Simple Accesses

For a simple access, the processor holds the address valid throughout the entire access. This protocol is used for single-cycle accesses, and for accesses to simple devices and memories.

On any cycle before the completion of the access, a simple access may be converted to a pipelined access (by the assertion of $\overline{PEN}$) or to a burst-mode access (by the assertion of $\overline{IBACK}$ or $\overline{DBACK}$, if the processor is asserting $\overline{IBREQ}$ or $\overline{DBREQ}$). Thus, the protocol for simple accesses also may be used during the initial cycles of pipelined and/or burst-mode accesses. This is advantageous, for example, in cases where the slave device or memory either requires the address to be held for multiple cycles at the beginning of the pipelined or burst-mode access, or cannot respond to the pipelined or burst-mode request within one cycle.

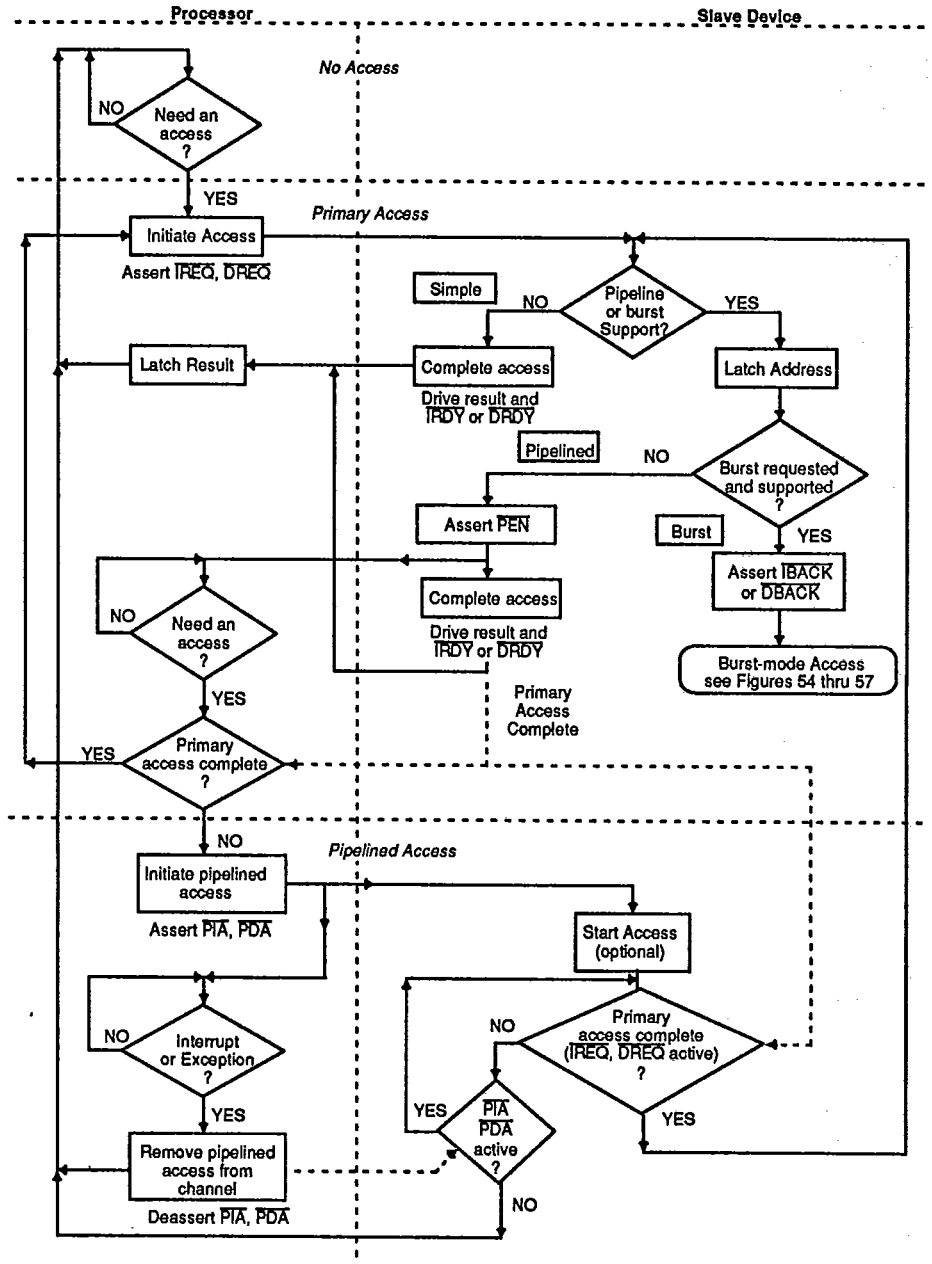


Figure 53. Channel Flowchart

Pipelined Accesses

A pipelined access is one that starts before an earlier in-progress access is completed. The in-progress access is called a primary access and the second access is called a pipelined access. A pipelined access is of the same type as the primary access. For example, an instruction access that begins before the completion of a data access is not considered to be a pipelined access, whereas a second data access is.

The Am29005 microprocessor allows only one pipelined access at any given time.

Tradeoffs

For accesses that require more than one cycle to complete, pipelined accesses perform better than simple accesses because they allow the overlap of portions of two accesses. In addition, the ability to latch addresses in support of pipelined accesses reduces utilization of the address bus, thereby reducing contention between instruction and data accesses. However, devices and memories that support pipelined accesses are somewhat more complex than devices and memories that support only simple accesses.

Support for pipelined operations is required for both the primary access and the pipelined access. The slave performing the primary access must contain some means for storing the address and other information about the access. The slave performing the pipelined access must be able to restrict its use of the Instruction bus or data Bus, and must be prepared to cancel the access (as explained below).

Pipelined Operation

Pipelined accesses are controlled by the signals $\overline{PEN}$, $\overline{PIA}$, and $\overline{PDA}$. Because of internal data-flow constraints, the Am29005 microprocessor does not perform a pipelined store operation while a load is in progress. However, the protocol does not restrict pipelined operations. Other channel masters may perform a pipelined store during a load.

Except as noted above, the processor attempts to perform pipelining for every access; the input $\overline{PEN}$ indicates whether or not pipelining is supported for a given access. The $\overline{PEN}$ input can be driven by individual devices, or can be tied active or inactive to enable or disable system-wide pipelined accesses. The processor ignores the value of $\overline{PEN}$ unless it is performing an access.

The processor samples $\overline{PEN}$ on every cycle during a primary access. If $\overline{PEN}$ is active on any cycle, the processor ceases to drive the address and associated controls for the primary access in the next cycle. If the processor requires another access before the primary access is completed, it drives the address and controls for the second access, asserting $\overline{PIA}$ or $\overline{PDA}$ to indicate that the second access is a pipelined access.

The output $\overline{IREQ}$ or $\overline{DREQ}$, as appropriate, is not asserted for a pipelined access. Devices and memories that cannot support pipelined accesses should therefore ignore $\overline{PIA}$ and/or $\overline{PDA}$, and base their operation upon $\overline{IREQ}$ and/or $\overline{DREQ}$.

A device or memory that receives a request for a pipelined access may treat it as any other access, with one exception: the pipelined access cannot use the instruction and data buses or the associated controls (e.g., $\overline{IRDY}$ or $\overline{DRDY}$). In the case of a data read or instruction access, the results of the pipelined access cannot be driven on the appropriate bus. In the case of a data write, the data do not appear on the data bus. Any other operations for the access, such as address decoding, can occur.

When the primary access is completed (as indicated by $\overline{IRDY}$ or $\overline{DRDY}$), the pipelined access becomes a primary access. The processor indicates this by asserting $\overline{IREQ}$ or $\overline{DREQ}$, depending on the type of access. The device or memory performing the pipelined access may complete the access as soon as $\overline{IREQ}$ or $\overline{DREQ}$ is asserted (possibly in the same cycle). When the access becomes a primary access, it controls the channel as any other primary access. For example, it may determine whether or not another pipelined access can be performed.

When the pipelined access becomes a primary access, the output $\overline{PIA}$ or $\overline{PDA}$ remains asserted for one cycle to ensure continuity of control within the slave device or memory. In the cycle after $\overline{IREQ}$ or $\overline{DREQ}$ is asserted, $\overline{PIA}$ or $\overline{PDA}$ is deasserted unless the processor initiates another pipelined access, in which case $\overline{PIA}$ or $\overline{PDA}$ remains asserted for the new access.

Cancellation of Pipelined Accesses

If the processor takes an interrupt or trap before a pipelined access becomes a primary access, the request for the pipelined access is removed from the channel. This may occur, for example, when $\overline{IERR}$ or $\overline{DERR}$ is signaled for the primary access.

If the pipelined access is removed from the channel, the slave device or memory does not receive an $\overline{IREQ}$ or $\overline{DREQ}$ for the pipelined access. Hence, the pipelined access does not become a primary access, and cannot be completed. A pipelined access may be canceled in this manner at any time before it becomes a primary access. Because of this, a pipelined access should not change the state of a slave device or memory until the pipelined access becomes a primary access.

Burst-Mode Accesses

A burst-mode access allows multiple instructions or data words at sequential addresses to be accessed with a single address transfer. The number of accesses performed and the timing of each access within the sequence are controlled dynamically by the burst-mode protocol. Burst-mode accesses take advantage of sequential addressing patterns, and provide several benefits over simple and pipelined accesses:

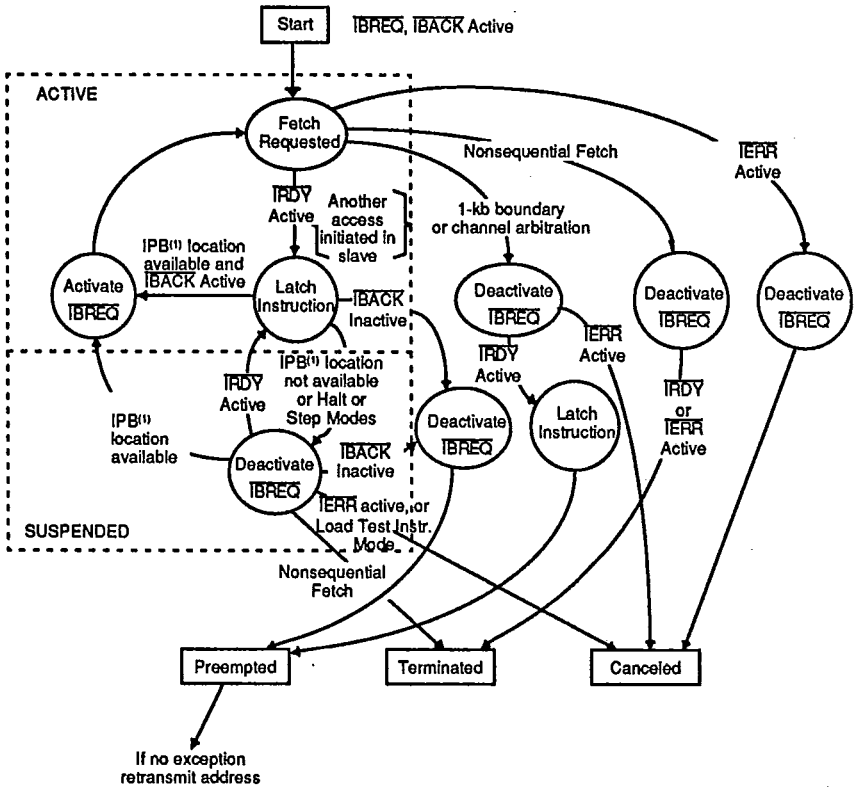
1. Simultaneous instruction and data accesses. Burst-mode accesses reduce the utilization of the address bus. This is especially important for instruction accesses, which are normally sequential. Burst-mode instruction accesses eliminate most of the address transfers for instructions, allowing the address bus to be used for simultaneous data accesses.
2. Faster access times. By eliminating the address-transfer cycle, burst-mode accesses allow addresses to be generated in a manner that improves access times.
3. Faster memory access modes. Many memories have special high-bandwidth access modes (e.g., fast page mode DRAM). These modes generally require a sequential addressing pattern, even though addresses may not be presented explicitly to the memory for all accesses. Burst-mode accesses allow the use of these access modes without hardware to detect sequential addressing patterns.

Burst-Mode Overview

The control-flow diagrams in Figure 54 and Figure 55 illustrate the operation of the processor and an instruction memory during a burst-mode instruction access. The control-flow diagrams in Figure 56 and Figure 57 illustrate the operation of the processor and a data memory or device during a burst-mode data access. These diagrams are for illustration only; nodes on these diagrams do not necessarily correspond to processor or slave states, and transitions on these diagrams do not necessarily correspond to processor cycles.

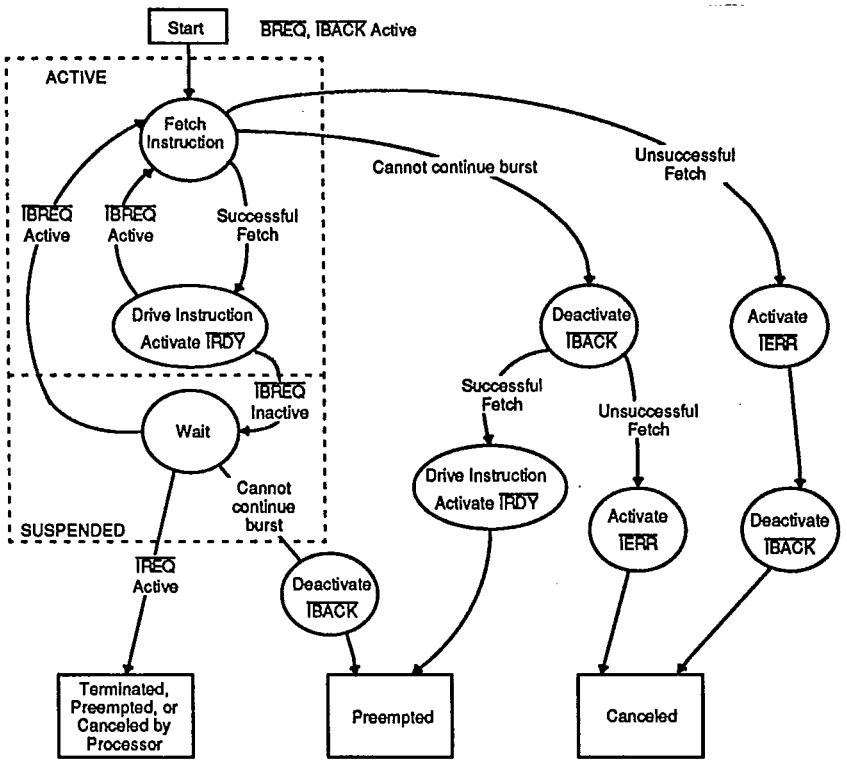
A burst-mode access is in one of the following operational conditions at any given time:

1. Established: The processor and slave device have successfully initiated the burst-mode access. A burst-



(1) IPB = Instruction Prefetch Buffer

Figure 54. Processor Burst-Mode Instruction Accesses: Control Flow



Note: A similar state transition may be used to support suspended burst-mode data accesses or a channel master other than the processor.

Figure 55. Slave Burst-Mode Instruction Accesses: Control Flow

mode access that has been established is either active or suspended. An established burst-mode access may become preempted, terminated or canceled.

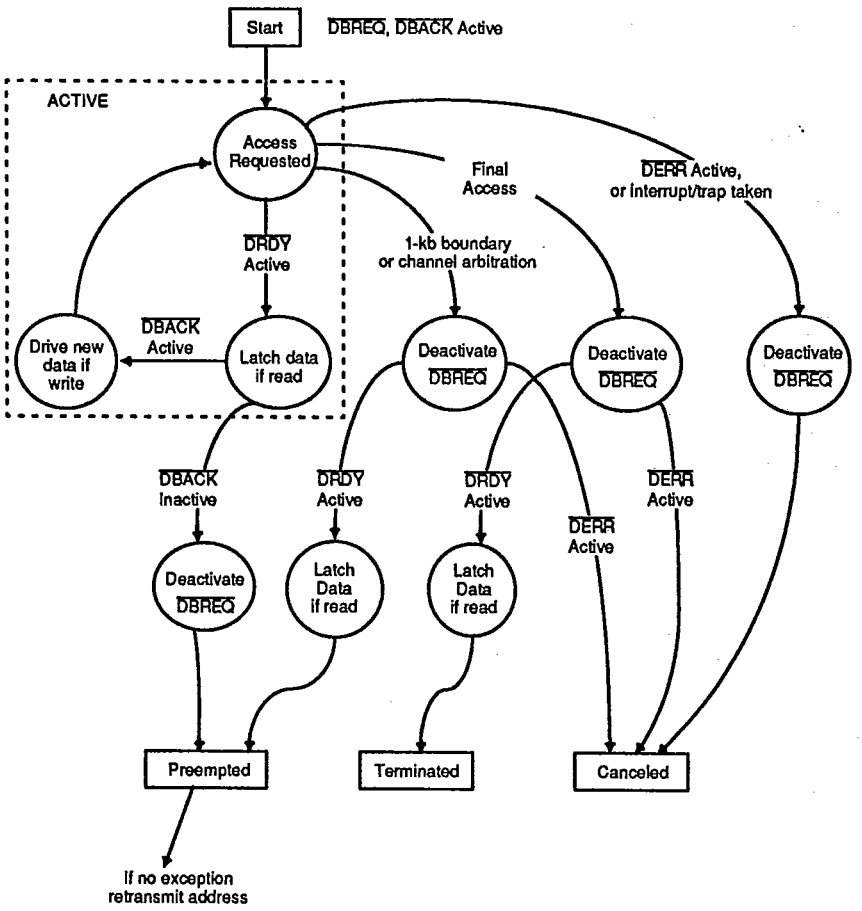
2. Active: Instruction or data accesses and transfers are being performed as the result of the burst-mode access. An active burst-mode access may become suspended.

3. Suspended: No accesses or transfers are being performed as the result of the burst-mode access, but the burst-mode access remains established. Additional accesses and transfers may occur at some later time (i.e., the burst-mode access may become active) without the retransmission of the address for the access.

4. Preempted: The burst-mode access can no longer continue because of some condition, but the burst-mode access can be reestablished within a short amount of time.

5. Terminated: All required accesses have been performed.

6. Canceled: The burst-mode access can no longer continue because of some exceptional condition. The access may be reestablished only after the exceptional condition has been corrected, if possible.



Note: The Am29005 microprocessor does not suspend burst-mode data accesses.

Figure 56. Processor Burst-Mode Data Accesses: Control Flow

Each of the above conditions, except for the terminated condition, is under the control of both the processor and slave device or memory. The terminated condition is determined by the processor, because only the processor can determine that all required accesses have been performed. The following sections discuss each of the above conditions with respect to the burst-mode protocol.

Establishing Burst-Mode Accesses

The Am29005 microprocessor attempts to perform all instruction prefetches using burst-mode accesses, except for instruction fetches at the last word before a 1-kb address boundary. For data accesses, the processor attempts to perform Load Multiple and Store Multiple operations using burst-mode accesses. The processor indicates that it desires a burst-mode access by asserting $\overline{IBREQ}$ or $\overline{DBREQ}$ during the cycle in which the initial address is placed on the address bus (however, note that these signals become valid later in the cycle than the address).

The inputs $\overline{IBACK}$ and $\overline{DBACK}$ indicate that a requested burst-mode access is supported. The processor ignores the value of $\overline{IBACK}$ unless $\overline{IBREQ}$ is asserted, and it ignores the value of $\overline{DBACK}$ unless $\overline{DBREQ}$ is asserted.

When it desires a burst-mode access, the processor continues to drive $\overline{IBREQ}$ or $\overline{DBREQ}$ on every cycle for which the address is valid on the address bus. During this time, the device or memory involved in the access may assert $\overline{IBACK}$ or $\overline{DBACK}$ to indicate that it can perform the burst-mode access. If $\overline{IBACK}$ or $\overline{DBACK}$ (as appropriate) is asserted while the initial address appears

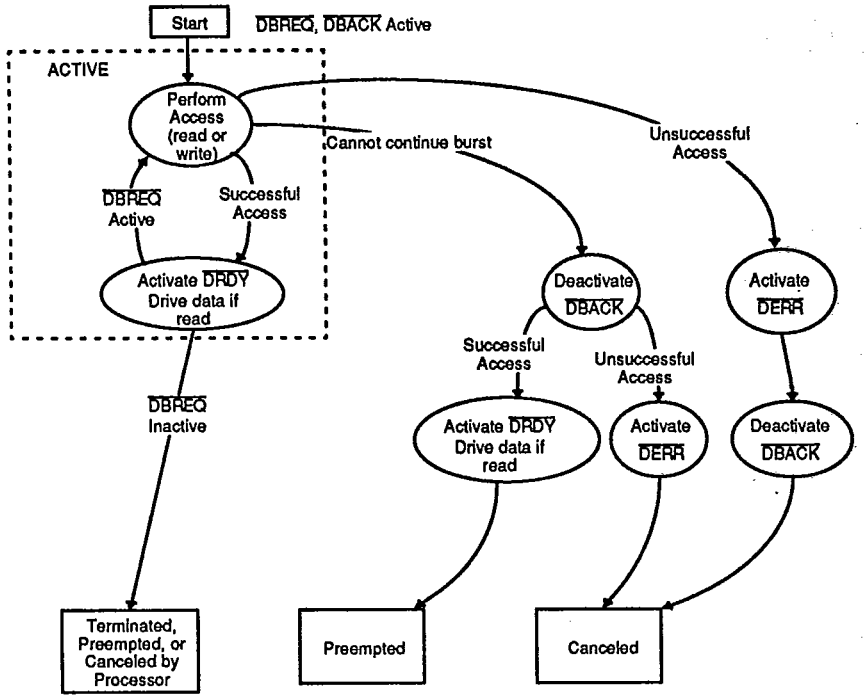


Figure 57. Slave Burst-Mode Data Accesses: Control Flow

on the address bus, the burst-mode access is established. In the following cycle, the processor removes the request address and deasserts $\overline{IREQ}$ or $\overline{DREQ}$. However, it continues to assert $\overline{IBREQ}$ or $\overline{DBREQ}$.

If the burst-mode access is not established on the first access, the processor attempts to establish a burst-mode access on each subsequent address transfer, as long as there are more accesses yet to be performed. During any subsequent access, the addressed device or memory may establish a burst-mode access by asserting $\overline{IBACK}$ or $\overline{DBACK}$. If the burst-mode access is never established, the default behavior is to have the processor transmit an address for every access.

Active and Suspended Burst-Mode Accesses

After the burst-mode access is established, $\overline{IBREQ}$ and $\overline{DBREQ}$ are used during subsequent accesses to indicate that the processor requires at least one more access. If $\overline{IBREQ}$ or $\overline{DBREQ}$ is active at the end of the cycle in which an access is successfully completed (i.e., when $\overline{IRDY}$ or $\overline{DRDY}$ is active), the processor requires another access. If the slave device or memory previously has not preempted the burst-mode access, and does not preempt (by deasserting $\overline{IBACK}$ or $\overline{DBACK}$) or cancel (by asserting $\overline{IERR}$ or $\overline{DERR}$) the burst-mode access in the cycle that the access completes, the additional access must be performed.

The execution rate of instructions is known only dynamically, so that in certain situations, a burst-mode instruction access must be suspended. If $\overline{IBREQ}$ is inactive during the cycle in which an instruction access is completed, the burst-mode access is suspended (if it is neither preempted nor canceled at the same time). The burst-mode access remains suspended unless the processor requests a new instruction access (in which case $\overline{IREQ}$ is asserted), or unless the instruction memory preempts the burst-mode access.

A suspended burst-mode instruction access becomes active whenever the processor can accept more instructions. The processor activates the burst-mode access by asserting $\overline{IBREQ}$. If the instruction memory does not preempt the burst-mode access during this cycle, an instruction access must be performed.

When a suspended burst-mode instruction access is activated, the resulting instruction access is not permitted to be completed in the cycle in which $\overline{IBREQ}$ is asserted, but may be completed in the next cycle. The reason for this restriction is that the burst-mode protocol is defined such that the combination of an active level on $\overline{IBREQ}$ and $\overline{IRDY}$ causes an instruction access (as previously discussed). If the instruction access is completed immediately in the cycle where a suspended burst-mode access is activated, there is an ambiguity in the protocol:

it is possible to interpret a single-cycle assertion of $\overline{IBREQ}$ as a request for two instructions.

The above ambiguity is resolved by delaying the instruction access resulting from a reactivated burst-mode access for a cycle. Since this restriction applies only when the Instruction Prefetch Buffer is full and the instruction memory is capable of a very fast access, the delayed instruction response has no performance impact.

The Am29005 microprocessor does not suspend burst-mode data accesses because the data transfers occur to and from general-purpose registers, which are always available. However, other channel masters may suspend burst-mode data accesses (during direct memory accesses, for example). The principles for suspending burst-mode accesses are the same as those for instruction accesses discussed above.

Processor Preemption, Termination, and Cancellation

The processor may preempt, terminate or cancel a burst-mode access by deasserting $\overline{IBREQ}$ or $\overline{DBREQ}$ and asserting $\overline{IREQ}$ or $\overline{DREQ}$ at some later point. Normally, the processor receives one more instruction or data word after $\overline{IBREQ}$ or $\overline{DBREQ}$ is deasserted. However, this access may be completed in the same cycle that $\overline{IBREQ}$ or $\overline{DBREQ}$ is deasserted. During the period after $\overline{IBREQ}$ or $\overline{DBREQ}$ is deasserted and before $\overline{IREQ}$ or $\overline{DREQ}$ is asserted, the burst-mode access is in a suspended condition.

The slave device or memory cannot distinguish between preempted, terminated, and canceled burst-mode accesses, when these are caused by the processor, until the processor asserts $\overline{IREQ}$ or $\overline{DREQ}$. If the slave continues to assert $\overline{IBACK}$ or $\overline{DBACK}$ after $\overline{IBREQ}$ or $\overline{DBREQ}$ is deasserted, the slave should be prepared to accept any new request during the cycle in which $\overline{IREQ}$ or $\overline{DREQ}$ is asserted to begin the new access. The reason for this is that the processor may attempt to establish a burst-mode access for the new access: if the slave is asserting $\overline{IBACK}$ or $\overline{DBACK}$ because of a previously preempted, terminated, or canceled burst-mode access, the processor interprets the active $\overline{IBACK}$ or $\overline{DBACK}$ as establishing the new burst-mode access and removes the request in the following cycle.

The processor preempts a burst-mode access when an external channel master arbitrates for the channel, or when a burst-mode fetch crosses a 1-kb address boundary. The burst-mode access is reestablished using the new address.

Note that the preemption resulting from 1-kb address boundaries is advantageous for devices or memories that require counters to follow the burst-mode address sequence. Since all burst-mode accesses are word accesses and the processor retransmits an address at every 1-kb address boundary, an 8-bit counter in the slave device or memory is sufficient to follow the burst-mode address sequence. Additional address bits are simply latched.

The processor terminates a burst-mode access whenever all required instructions or data have been accessed. In the case of instruction accesses, the burst-mode access is terminated when a nonsequential fetch occurs. In the case of data accesses, the burst-mode access is terminated when the count indicates a single load or store remains. The last load or store is executed as a simple access.

The processor cancels a burst-mode access when an interrupt or trap is taken. Note that a trap may be caused by the burst-mode access, for example when a XDERR response is received to an access in the sequence. If the processor cancels a burst-mode access when an access in the sequence remains to be completed, this access must be completed in spite of the cancellation.

Canceled burst-mode data accesses may be restarted at some (possibly much later) point in execution via the Channel Address, Channel Data, and Channel Control registers. In this case, the burst-mode access is restarted at the point at which it was canceled, rather than at the beginning of the original address sequence.

Slave Preemption and Cancellation

The slave device or memory involved in a burst-mode access may preempt the access by deasserting $\overline{IBACK}$ or $\overline{DBACK}$. The processor samples $\overline{IBACK}$ and $\overline{DBACK}$ when $\overline{IRDY}$ and $\overline{DRDY}$ are active so that $\overline{IBACK}$ and $\overline{DBACK}$ may be deasserted as the last supported access is completed. However, $\overline{IBACK}$ and $\overline{DBACK}$ also may be deasserted in any cycle before the access is completed. If $\overline{IBACK}$ or $\overline{DBACK}$ is deasserted when the processor is in a state where it expects an access, the access must be completed.

In general, the slave device or memory preempts the burst-mode access whenever it cannot support any further accesses in the burst-mode sequence. This normally occurs whenever an implementation-dependent address boundary is encountered (e.g., a cache-block boundary), but may occur for any reason. By preempting the burst-mode access, the slave receives a new request with the address of the next instruction or data word required by the processor.

The slave device or memory may cancel a burst-mode access by asserting $\overline{IERR}$ or $\overline{DERR}$ in response to a requested access. The signals $\overline{IBACK}$ or $\overline{DBACK}$ need not be deasserted at this time, but should be deasserted in the next cycle.

Note that the $\overline{IERR}$ and $\overline{DERR}$ signals cause non-maskable traps, except in the case where $\overline{IERR}$ is asserted for an instruction that the processor does not execute.

Arbitration

External masters can gain access to the address, data, and instruction buses by asserting the $\overline{BREQ}$ input. The processor completes any pending access, preempts any burst-mode access, and asserts the $\overline{BGRT}$ output. At this time, the processor places all channel outputs associated with the address, data, and instruction buses in the high-impedance state.

For the first cycle in which $\overline{BGRT}$ is asserted, the output $\overline{BINV}$ is also asserted. If the external master cannot control the address bus and associated controls in the cycle where $\overline{BGRT}$ is asserted, the active level on $\overline{BINV}$ may be used to define an idle cycle for the channel (i.e., any spurious access requests are ignored). The $\overline{BINV}$ signal is asserted only for a single cycle, so the external master must take control of the channel in the cycle after $\overline{BGRT}$ is asserted.

While the $\overline{BREQ}$ input remains asserted, the processor continues to assert $\overline{BGRT}$. The external master has control over the channel during this time.

To release the channel to the processor, the external master deasserts $\overline{BREQ}$, but must continue to control the channel for the first cycle in which $\overline{BREQ}$ is deasserted. In the cycle after $\overline{BREQ}$ is deasserted, the processor asserts $\overline{BINV}$ and deasserts $\overline{BGRT}$; the external master should release control of the channel at this time. On the following cycle, the processor deasserts $\overline{BINV}$ and is able to use the channel. The processor reestablishes any burst-mode access preempted by arbitration.

The processor does not relinquish the channel when the $\overline{LOCK}$ signal is active. This prevents external masters from interfering with exclusive accesses.

Bus Sharing—Electrical Considerations

When buses are shared among multiple masters and slaves, it is important to avoid situations where these devices are driving a bus at the same time. This may occur when more than one master or slave is allowed to drive a bus in the same cycle if bus arbitration is incompletely or incorrectly performed. However, it also occurs when a master or slave releases a bus in the same cycle that another master or slave gains control, and the first master or slave is slow in disabling its bus drivers, compared to the point at which the second master or slave begins to drive the bus. The latter situation is called a bus collision in the following discussion.

In addition to the logical errors that can occur when multiple devices drive a bus simultaneously, such situations may cause bus drivers to carry large amounts of electrical current. This can have a significant impact on driver reliability and power dissipation. Since bus collisions usually occur for a small amount of time, they are of less concern, but may contribute to high-frequency electromagnetic emissions.

The Am29005 microprocessor channel is defined to prevent all situations where multiple drivers are driving a bus simultaneously. However, bus collisions may be allowed to occur, depending on the system design.

In the case of the Am29005 microprocessor channel, arbitration for the channel prevents the processor from driving the address and data buses at the same time as another channel master. If there is more than one external master, the system design must include some means for ensuring that only one external master gains control of the channel, and that no external master gains control of the channel at the same time as the processor.

When the processor relinquishes control of the channel to an external master, bus collisions may be prevented by not allowing the external master to drive any bus while $\overline{BINV}$ is active. This ensures that all processor outputs are disabled by the time the external master takes control of the channel. However, there is nothing in the channel protocol to prevent the external master from taking control as soon as $\overline{BGRT}$ is asserted.

Slave devices and memories are prevented from simultaneously driving the instruction bus or data bus by allowing only the device or memory performing a primary access to drive the appropriate bus. When a pipelined access becomes a primary access, it may drive the instruction or data bus immediately, so there is a potential bus collision if the pipelined access is performed by a slave other than the slave performing the original primary access. This bus collision may be prevented by restricting all slaves to driving the instruction and data buses in the second half-cycle (using SYSCLK, for example). Since the processor samples data only at the end of a cycle, this restriction does not affect performance.

When the processor performs a store immediately following a load, it drives the data bus for the store in the second cycle following the cycle in which the data for the load appears on the data bus. This provides a complete cycle for the slave involved in the load to disable its data drivers. The processor continues to drive the data bus until it receives a $\overline{DRDY}$ or $\overline{DERR}$ in response to the store; it ceases to drives the data bus in the cycle following the response.

Channel Behavior for Interrupts and Traps

If an interrupt or trap is taken, any burst-mode accesses are canceled. If a request for a pipelined access is on the address bus, this request is removed. Any other accesses are completed and no new accesses are started, other than those required for the interrupt or trap. Note that any accesses that the processor expects to complete must be completed, even though burst-mode and pipelined accesses are canceled.

When interrupt or trap processing is complete, any canceled burst-mode access transactions are reestab-

lished using the address of the access that was to be performed next when the interrupt or trap was taken. Uncompleted pipelined accesses are restarted, either by the interrupt return sequence in the case of an instruction access, or by restarting the initiating instruction in the case of a data access.

Note that the restarting of a pipelined access is not performed by the Channel Address, Channel Data, and Channel Control registers, since these registers may be required to restart the primary access. The instruction initiating the pipelined access is not allowed to be completed until the primary access is completed, so that the Program Counter 1 (PC1) register contains the address of the initiating instruction when a pipelined access is canceled. The address in PC1 can restart this instruction on interrupt return.

Effect of the $\overline{\text{LOCK}}$ Output

The $\overline{\text{LOCK}}$ output provides synchronization and exclusion of accesses in a multiprocessor environment. $\overline{\text{LOCK}}$ has no predefined effect for a system, other than the fact that the Am29005 microprocessor does not grant the channel to an external master while $\overline{\text{LOCK}}$ is active.

The $\overline{\text{LOCK}}$ output is asserted for the address cycle of the Load-and-Lock and Store-and-Lock instructions, and is asserted for both the read and write accesses of a Load and Set instruction. $\overline{\text{LOCK}}$ may also be active for an extended period of time under control of the Lock bit in the Current Processor Status Register (this capability is available only to Supervisor-mode programs).

$\overline{\text{LOCK}}$ may be defined to provide any level of resource locking for a particular system. For example, it may lock the channel, an individual device or memory, or a location within a device or memory.

When a resource is locked, it is available for access only by the processor with the appropriate access privilege. The mechanisms for restricting accesses and the methods for reporting attempted violations of the restrictions are system-dependent.

Initialization and Reset

When power is first applied to the processor, it is in an unknown state and must be placed in a known state. Also, under certain circumstances, it may be necessary to place the processor in a defined state. This is accomplished by the Reset mode, which is invoked by activating the $\overline{\text{RESET}}$ pin for the required duration. The Reset mode configures the processor state as follows:

1. Instruction execution is suspended.
2. Instruction fetching is suspended.
3. Any interrupt or trap conditions are ignored.
4. The Current Processor Status Register is set as shown in Figure 58.
5. The Data Width Enable bit of the Configuration Register is reset.
6. The Contents Valid bit of the Channel Control Register is reset.

Except as previously noted, the contents of all general-purpose registers and special-purpose registers are undefined.

The Reset mode also configures the processor to initiate an instruction fetch using an address of 0. Since the ROM enable (RE) bit of the Current Processor Status is 1, this fetch is directed to external instruction read-only memory. This fetch occurs when the Reset mode is exited (i.e., when the $\overline{\text{RESET}}$ input is deasserted).

The Reset mode is invoked by asserting the $\overline{\text{RESET}}$ input and can be entered only if the SYSCLK pin is operating normally, whether or not the SYSCLK pin is being driven by the processor. The Reset mode is entered within four processor cycles after $\overline{\text{RESET}}$ is asserted. The $\overline{\text{RESET}}$ input must be asserted for at least four processor cycles to accomplish a processor reset.

The Reset mode can be entered from any other processor mode (e.g., the Reset mode can be entered from the Halt mode). If the $\overline{\text{RESET}}$ input is asserted at the time that power is first applied to the processor, the processor enters the Reset mode only after four cycles have occurred on the SYSCLK pin.

The Reset mode is exited when the $\overline{\text{RESET}}$ input is deasserted. Either three or four cycles after $\overline{\text{RESET}}$ is deasserted (depending on internal synchronization time), the processor performs an initial instruction access on the channel. The initial instruction access is directed to

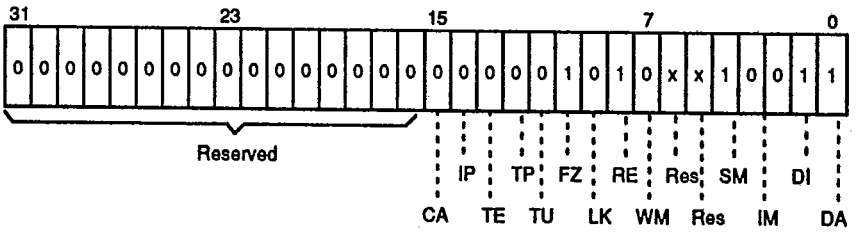


Figure 58. Current Processor Status Register in Reset Mode

Address 0 in the instruction read-only memory (instruction ROM). If instruction ROM is not implemented in a particular system, another device or memory must respond to this instruction fetch.

If the $CNTL_1$–$CNTL_0$ inputs are 10 or 01 when $\overline{RESET}$ is deasserted, the processor enters the Halt or Step mode, respectively. If the processor enters the Halt mode immediately after reset, the protection checking that normally applies to the Halt instruction is disabled so that the Halt instruction can be used as an instruction breakpoint in a User-mode program. The Load Test Instruction mode cannot be directly entered from the Reset mode. If the $CNTL_1$–$CNTL_0$ inputs are 00 immediately after $\overline{RESET}$ is deasserted, the effect on processor operation is unpredictable. If the $CNTL_1$–$CNTL_0$ inputs are 11, the processor enters the Executing mode.

The processor samples the $STAT_0$ output internally when $\overline{RESET}$ is asserted. A High level on $STAT_0$ in this case is used to enable a special test configuration and causes the processor to be inoperable. When $\overline{RESET}$ is asserted, the processor drives $STAT_0$ Low in order to disable this test configuration. However, if processor outputs are disabled by the Test mode, the processor is not able to drive $STAT_0$. Thus, if $\overline{RESET}$ is asserted when the processor is in the Test mode, the $STAT_0$ pin must be driven Low externally. (In a master/slave configuration, $STAT_0$ is driven Low by the master processor when $\overline{RESET}$ is asserted.)

ABSOLUTE MAXIMUM RATINGS

Storage Temperature ..... −65 to +150°C
Voltage on any Pin with
Respect to GND ..... −0.5 to V_{CC} +0.5 V

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

OPERATING RANGES T-49-17-32

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

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

DC CHARACTERISTICS over COMMERCIAL operating range

Parameter Symbol	Parameter Description	Test Conditions	Advance Information Min	Advance Information Max	Unit
V_{IL}	Input Low Voltage		−0.5	0.8	V
V_{IH}	Input High Voltage		2.0	V_{CC} +0.5	V
$V_{ILINCLK}$	INCLK Input Low Voltage		−0.5	0.8	V
$V_{IHINCLK}$	INCLK Input High Voltage		2.0	V_{CC} +0.5	V
$V_{ILSYSCLK}$	SYSCLK Input Low Voltage		−0.5	0.8	V
$V_{IHSYSCLK}$	SYSCLK Input High Voltage		V_{CC} −0.8	V_{CC} +0.5	V
V_{OL}	Output Low Voltage for All Outputs except SYSCLK	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Output High Voltage for All Outputs except SYSCLK	I_{OH} = −400 µA	2.4		V
I_{LI}	Input Leakage Current	0.45V ≤ V_{IN} ≤ V_{CC} −0.45V		±10	µA
I_{LO}	Output Leakage Current	0.45V ≤ V_{OUT} ≤ V_{CC} −0.45V		±10	µA
I_{CCOP}	Operating Power-Supply Current	V_{CC} = 5.25V, Outputs Floating; Holding RESET active with externally supplied SYSCLK		22	mA/MHz
V_{OLC}	SYSCLK Output Low Voltage	I_{OLC} = 20 mA		0.6	V
V_{OHC}	SYSCLK Output High Voltage	I_{OHC} = 20 mA	V_{CC} −0.6		V
I_{OSGND}	SYSCLK GND Short Circuit Current	V_{CC} = 5.0 V	100		mA
I_{OSVCC}	SYSCLK V_{CC} Short Circuit Current	V_{CC} = 5.0 V	100		mA

CAPACITANCE

Parameter Symbol	Parameter Description	Test Conditions	Min	Max	Unit
C_{IN}	Input Capacitance	fC = 1 MHz		15	pF
C_{INCLK}	INCLK Input Capacitance			20	pF
C_{SYSCLK}	SYSCLK Capacitance			90	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	Advance Information 16 MHz Min	Advance Information 16 MHz Max	Unit
1	System Clock (SYSCLK) Period (T)	Note 1	60	1000	ns
1A	SYSCLK at 1.5 V to $\overline{\text{SYSCLK}}$ at 1.5 V when used as an output	Note 13	0.5T – 2	0.5T + 2	ns
2	SYSCLK High Time when used as an input	Note 13	27		ns
3	SYSCLK Low Time when used as an input	Note 13	22		ns
4	SYSCLK Rise Time	Note 2		5	ns
5	SYSCLK Fall Time	Note 2		5	ns
6	Synchronous SYSCLK Output Valid Delay	Notes 3, 12	3	16	ns
6A	Synchronous SYSCLK Output Valid Delay for D_{31}–D_0	Note 12	4	20	ns
7	Three-State Synchronous SYSCLK Output Invalid Delay	Notes 4, 14, 15	3	30	ns
8	Synchronous $\overline{\text{SYSCLK}}$ Output Valid Delay	Notes 5, 12	3	16	ns
8A	Three-State $\overline{\text{SYSCLK}}$ Synchronous Output Invalid Delay	Notes 5, 14, 15	3	30	ns
9	Synchronous Input Setup Time	Note 7	15		ns
9A	Synchronous Input Setup Time for D_{31}–D_0, I_{31}–I_0		8		ns
9B	Synchronous Input Setup Time for $\overline{\text{DRDY}}$		16		ns
10	Synchronous Input Hold Time	Note 6	2		ns
11	Asynchronous Input Minimum Pulse Width	Note 8	T +10		ns
12	INCLK Period		30	500	ns
12A	INCLK to SYSCLK Delay		2	15	ns
12B	INCLK to $\overline{\text{SYSCLK}}$ Delay		2	15	ns
13	INCLK Low Time		12		ns
14	INCLK High Time		12		ns
15	INCLK Rise Time			5	ns
16	INCLK Fall Time			5	ns
17	INCLK to Deassertion of $\overline{\text{RESET}}$ (for phase synchronization of SYSCLK)	Note 9	0	5	ns
18	$\overline{\text{WARN}}$ Asynchronous Deassertion Hold Minimum Pulse Width	Note 10	4T		ns
19	$\overline{\text{BINV}}$ Synchronous Output Valid Delay from $\overline{\text{SYSCLK}}$	Note 12	1	9	ns
20	Three-State Synchronous SYSCLK Output Invalid Delay for D_{31}–D_0	Notes 11, 14, 15	3	25	ns

T-49-17-32

Notes:

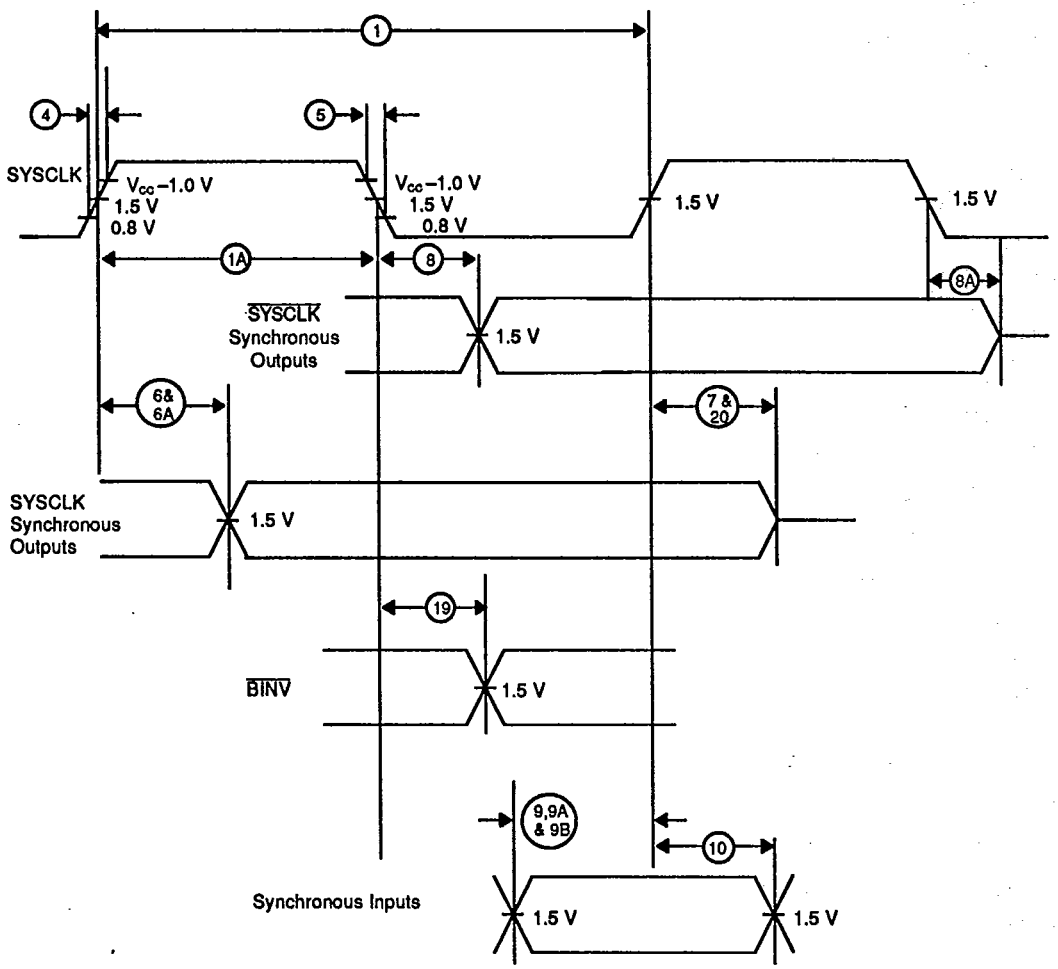
1. AC measurements made relative to 1.5 V, except where noted.
2. SYSCLK rise and fall times measured between 0.8 V and (V_{CC} – 1.0 V).
3. Synchronous Outputs relative to SYSCLK rising edge include: A_{31}–A_0, $\overline{BGRT}$, R/$\overline{W}$, SUP/$\overline{US}$, $\overline{LOCK}$, $MPGM_1$–$MPGM_0$, $\overline{IREQ}$, IREQT, $\overline{PIA}$, $\overline{DREQ}$, $DREQT_1$–$DREQT_0$, $\overline{PDA}$, OPT_2–OPT_0, $STAT_2$–$STAT_0$, and MSERR.
4. Three-state Synchronous Outputs relative to SYSCLK rising edge include: A_{31}–A_0, R/$\overline{W}$, SUP/$\overline{US}$, $\overline{LOCK}$, $MPGM_1$–$MPGM_0$, $\overline{IREQ}$, IREQT, $\overline{PIA}$, $\overline{DREQ}$, $DREQT_1$–$DREQT_0$, $\overline{PDA}$, and OPT_2–OPT_0.
5. Synchronous Outputs relative to SYSCLK falling edge ($\overline{SYSCLK}$): $\overline{IBREQ}$, $\overline{DBREQ}$.
6. Synchronous Inputs include: $\overline{BREQ}$, $\overline{PEN}$, $\overline{IRDY}$, $\overline{IERR}$, $\overline{IBACK}$, $\overline{DERR}$, $\overline{DBACK}$, $\overline{CDA}$, I_{31}–I_0, $\overline{DRDY}$ and D_{31}–D_0.
7. Synchronous Inputs include: $\overline{BREQ}$, $\overline{PEN}$, $\overline{IRDY}$, $\overline{IERR}$, $\overline{IBACK}$, $\overline{DERR}$, $\overline{DBACK}$, and $\overline{CDA}$.
8. Asynchronous Inputs include: $\overline{WARN}$, $\overline{INTR_3}$–$\overline{INTR_0}$, $\overline{TRAP_3}$–$\overline{TRAP_0}$, and $CNTL_1$–$CNTL_0$.
9. $\overline{RESET}$ is an asynchronous input on assertion/deassertion. As an option to the user, $\overline{RESET}$ deassertion can be used to force the state of the internal divide-by-two flip-flop to synchronize the phase of SYSCLK (if internally generated) relative to $\overline{RESET}$/INCLK.
10. $\overline{WARN}$ has a minimum pulse width requirement upon deassertion.
11. To guarantee Store/Load with one-cycle memories, D_{31}–D_0 must be asserted relative to SYSCLK falling edge from an external drive source.
12. Refer to Capacitive Output Delay table when capacitive loads exceed 80 pF.
13. When used as an input, SYSCLK presents a 90-pF max. load to the external driver. When SYSCLK is used as an output, timing is specified with an external load capacitance of ≤ 200 pF.
14. Three-State Output Inactive Test Load. Three-State Synchronous Output Invalid Delay is measured as the time to a ±500 mV change from prior output level.
15. When a three-state output makes a synchronous transition from a valid logic level to a high-impedance state, data is guaranteed to be held valid for an amount of time equal to the lesser of the minimum Three-State Synchronous Output Invalid Delay and the minimum Synchronous Output Valid Delay.

Conditions:

a. All inputs/outputs are TTL compatible for V_{IH}, V_{IL}, V_{OH}, and V_{OL} unless otherwise noted.
b. All output timing specifications are for 80 pF of loading.
c. All setup, hold, and delay times are measured relative to SYSCLK or INCLK unless otherwise noted.
d. All input Low levels must be driven to 0.45 V and all input High levels must be driven to 2.4 V except SYSCLK.

SWITCHING WAVEFORMS

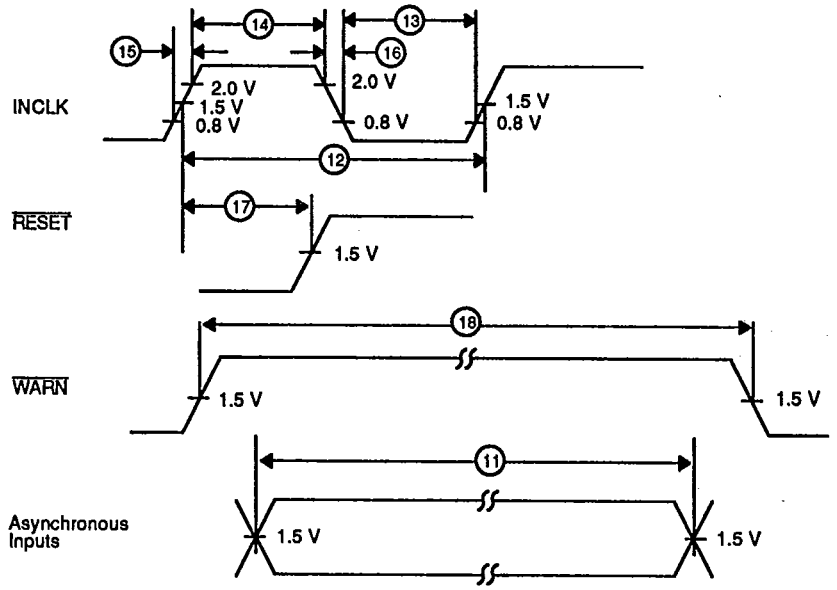
T-49-17-32



Relative to SYSCLK

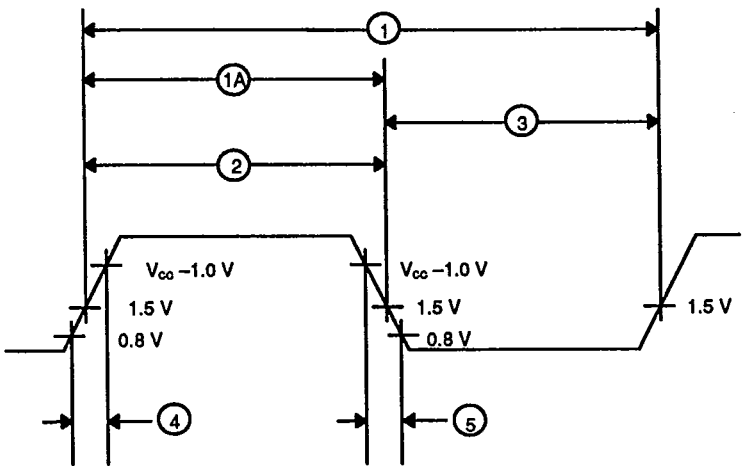
SWITCHING WAVEFORMS (continued)

T-49-17-32

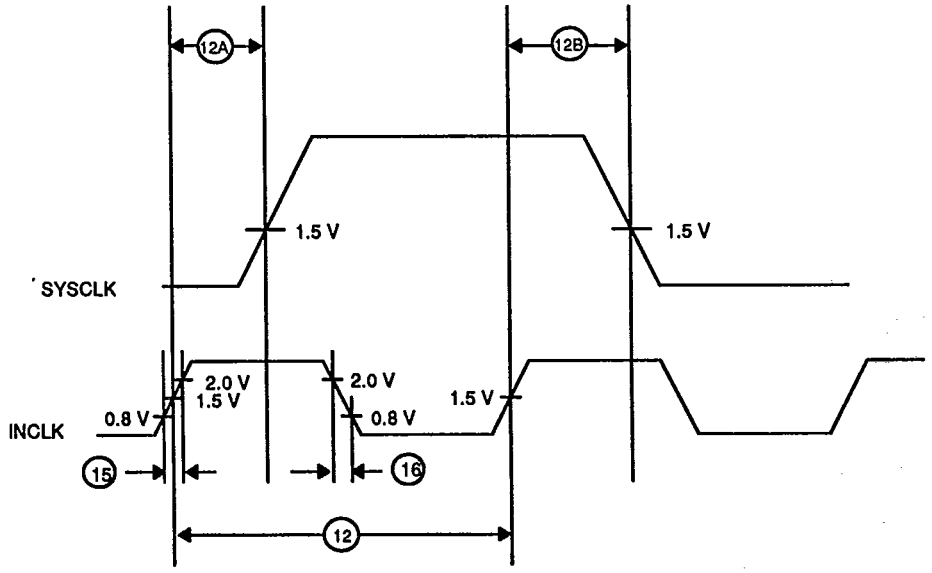


INCLK and Asynchronous Inputs

SWITCHING WAVEFORMS (continued)



SYSCLK Definition



INCLK to SYSCLK Delay

Capacitive Output Delays

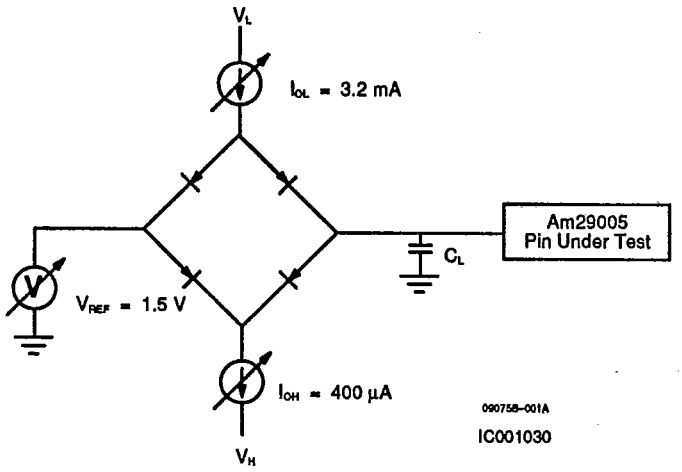
T-49-17-32

For loads greater than 80 pF

This table describes the additional output delays for capacitive loads greater than 80 pF. Values in the Maximum Additional Delay column should be added to the value listed in the Switching Characteristics table. For loads less than or equal to 80 pF, refer to the delays listed in the Switching Characteristics table.

No.	Parameter Description	Total External Capacitance	Maximum Additional Delay
6	Synchronous SYSCLK Output Valid Delay	100 pF 150 pF 200 pF 250 pF 300 pF	+1 ns +2 ns +4 ns +6 ns +8 ns
6A	Synchronous SYSCLK Output Valid Delay for D_{31}–D_0	100 pF 150 pF 200 pF 250 pF 300 pF	+1 ns +6 ns +10 ns +15 ns +19 ns
8	Synchronous $\overline{\text{SYSCLK}}$ Output Valid Delay	100 pF 150 pF 200 pF 250 pF 300 pF	+1 ns +2 ns +4 ns +6 ns +8 ns
19	$\overline{\text{BINV}}$ Synchronous Output Valid Delay from $\overline{\text{SYSCLK}}$	100 pF 150 pF 200 pF 250 pF 300 pF	+1 ns +3 ns +4 ns +6 ns +7 ns

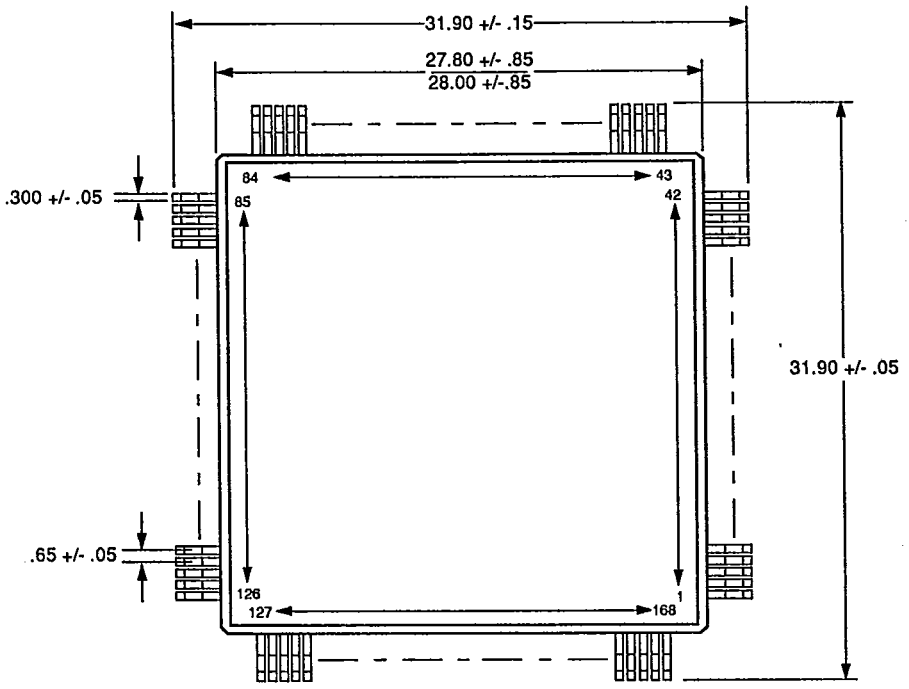
SWITCHING TEST CIRCUIT



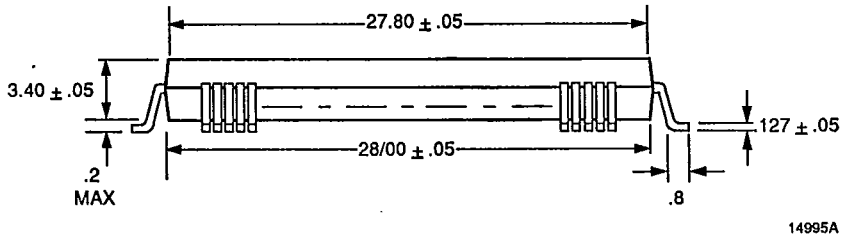
C_L is guaranteed to 80 pF. For capacitive loading greater than 80 pF, refer to the Capacitive Output Delay table.

PHYSICAL DIMENSIONS

QFP168 (PQFP)
(EIAJ)



TOP VIEW

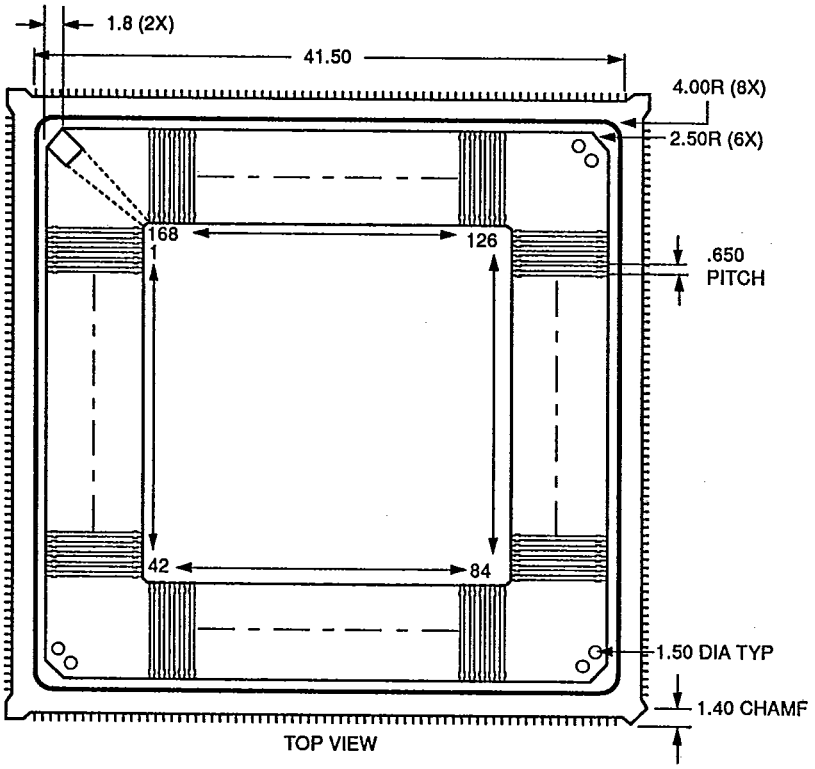


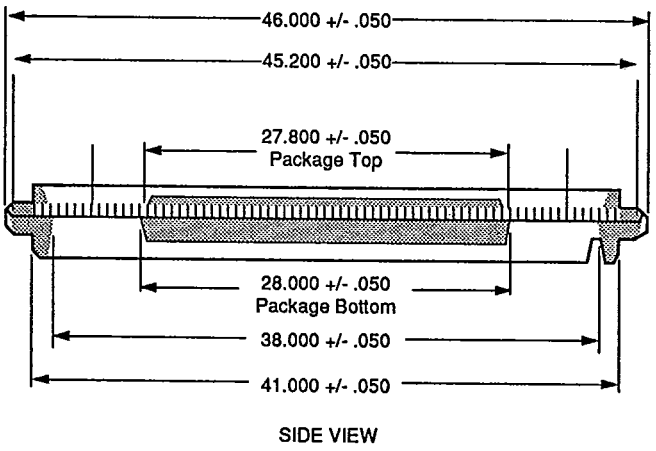
SIDE VIEW

PHYSICAL DIMENSIONS (continued)

T-49-17-32

QFP168 (MCQFP)
MOLDED CARRIER QUAD FLAT PACK





15000A