



PRELIMINARY

Am29112

DISTINCTIVE CHARACTERISTICS

- **Expandable**
8-bit slice, cascadable up to 16 bits
- **Deep stack**
A 33-deep on-chip stack is used for subroutine linkage, interrupt handling, and loop control.
- **Hold feature**
A hold pin facilitates multiple sequencer implementations.
- **Interruptible at the microprogram level**
Two kinds of interrupts: maskable and unmaskable.
- **Powerful loop control**
When cascaded, two counters can act as a single 16-bit counter or two independent 8-bit counters.
- **Powerful addressing modes**
Features direct, multiway, multiway relative, and program counter relative addressing.

GENERAL DESCRIPTION

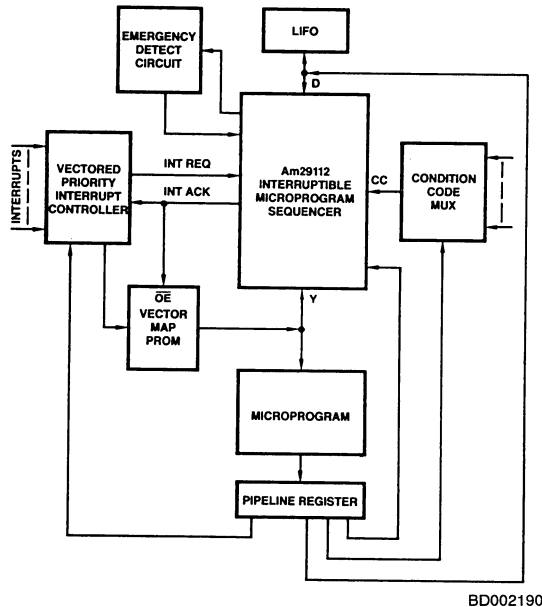
The Am29112 is a high-performance interruptible microprogram sequencer intended for use in very high-speed microprogrammed machines and optimized for the new state-of-the-art ALUs such as the Am29116A 16-Bit Bipolar Microprocessor.

The Am29112 is designed to operate in 10-MHz microprogrammed systems.

It has an instruction set featuring relative and multiway branching, a rich variety of looping constructs, and provision for loading and unloading the on-chip stack.

Interrupts are accepted at the microcycle level and serviced in a manner completely transparent to the interrupted microcode.

BLOCK DIAGRAM*

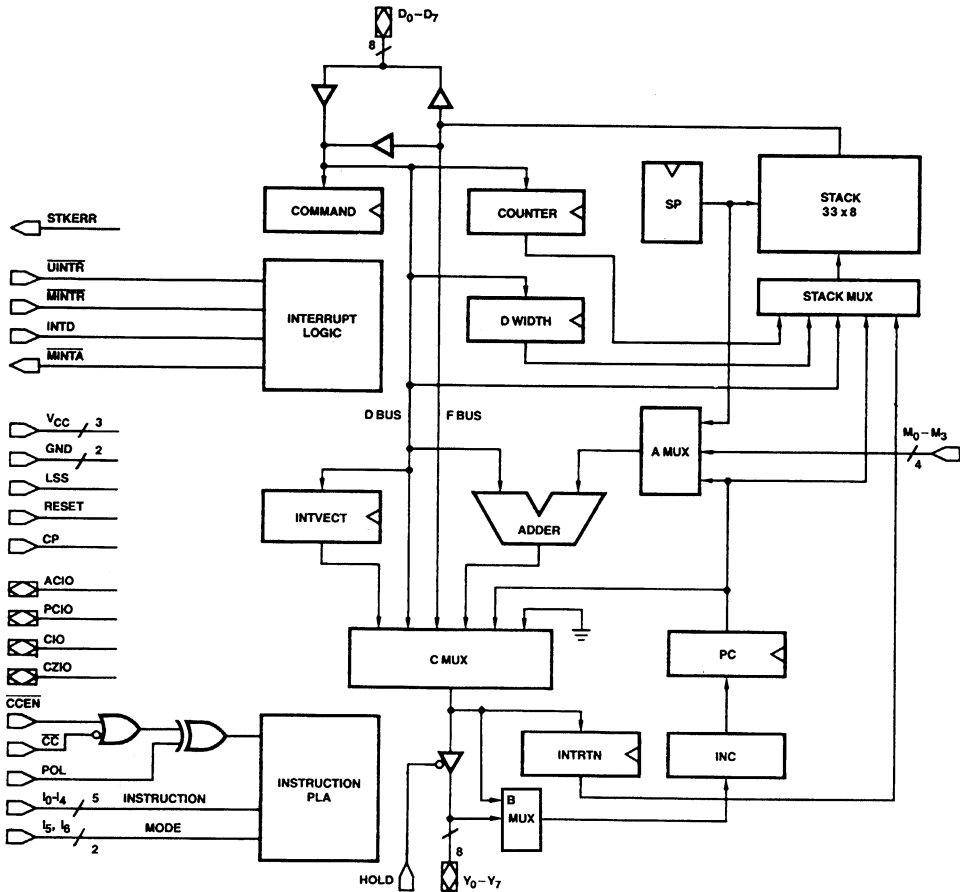


BD002190

*Typical System Diagram — Am29112 in a Single Pipelined System
(See Figure 1 for Detailed Block Diagram)

RELATED PRODUCTS

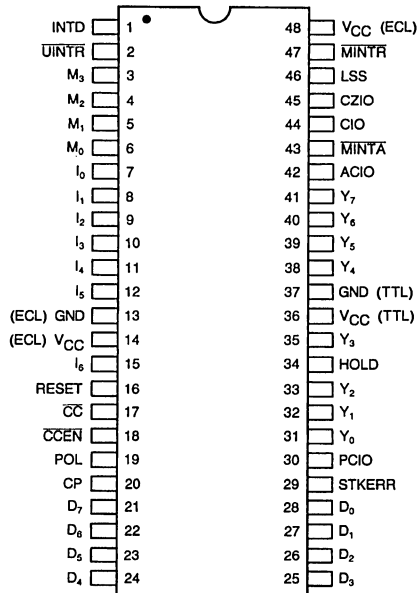
Part No.	Description
Am29114	Vectored Priority Interrupt Controller
Am29116	A 16-Bit Bipolar Microprocessor
Am29117	Two-Port Am29116
Am29118	8-Bit Bidirectional I/O Port/Accumulator
Am29PL141	Fuse-Programmable Controller
Am2950A/ 51A/52A/53A	8-Bit Bidirectional I/O Port
Am2925	System Clock Generator and Driver
Am2904	Status and Shift Control Unit
Am2940	DMA Address Generator
Am2942	Programmable Timer/Counter/DMA



BD001932

Figure 1. Detailed Block Diagram

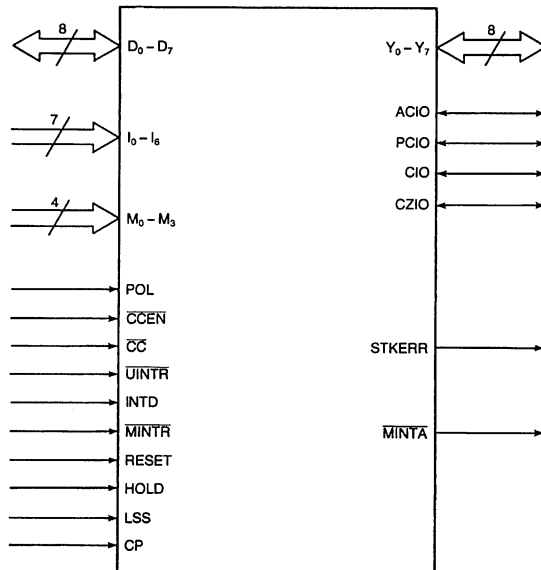
CONNECTION DIAGRAM Top View



CD004891

Note: Pin 1 is marked for orientation.

LOGIC SYMBOL



LS002630

ORDERING INFORMATION

Standard Products

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

- A. Device Number**
- B. Speed Option** (if applicable)
- C. Package Type**
- D. Temperature Range**
- E. Optional Processing**

AM29112

D

C

B

E. OPTIONAL PROCESSING
Blank = Standard processing
B = Burn-in

D. TEMPERATURE RANGE
C = Commercial (0 to +70°C)

C. PACKAGE TYPE
D = 48-Pin Sidebraced Ceramic DIP (SD 048)

B. SPEED OPTION
Not Applicable

A. DEVICE NUMBER/DESCRIPTION
Am29112
High-Performance 8-Bit Slice Microprogram Sequencer

Valid Combinations

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

Valid Combinations	
AM29112	DC, DCB

PIN DESCRIPTION

D₀-D₇ Data I/O Lines (8) (Input/Output)

Bidirectional bus inputs data to address multiplexer, counter, and other control registers and outputs stack.

Y₀-Y₇ Address I/O Lines (8) (Input/Output)

Bidirectional microprogram address bus outputs microprogram address and inputs interrupt vector.

M₀-M₃ Multiway Inputs (4) (Input)

Multiway input pins for up to 16-way branches.

HOLD Hold Control Pin (Input)

When this signal is HIGH, the Y-Bus is three-stated and the carry-in to the program counter incrementer is forced LOW. Also, the CMUX output is selected at the incrementer input.

CC Condition Code Pin (Input)

Test condition input for the sequencer (see Table 2).

CCEN Condition Code Enable Pin (Input)

Test enable for the sequencer (see Table 2).

POL Polarity Pin (Input)

Polarity Input for the test (see Table 2).

I₀-I₄ Instruction Lines (5) (Input)

Used to select the instruction to be performed by the Am29112.

I₅, I₆ Instruction Lines (2) (Input)

Mode control inputs. Selects one of three modes: normal, extended, or forced continue (see Table 1).

STKERR Stack Error Pin (Output)

Indicates stack overflow or underflow.

UNINTR Unmaskable Interrupt Pin (Input)

Unmaskable interrupt request input. Needs to remain active for at least one cycle.

MINTR Maskable Interrupt Pin (Input)

Maskable interrupt request input. Needs to remain active for at least one cycle. May remain active during entire interrupt routine. MINTR should go inactive for at least one cycle before it can go active again.

INTD Interrupt Disable Pin (Input)

Disable input for maskable interrupts.

MINTA Maskable Interrupt Acknowledge Pin (Output)

Acknowledges to interrupting device that the Am29112 has been interrupted.

LSS Least Significant Slice Pin (Input)

When LSS is HIGH, the Am29112 is programmed as the least significant slice. When LSS is LOW, the Am29112 is programmed as the most significant slice.

RESET Reset Pin (Input)

The reset input is registered and has no effect until the following cycle. On the second clock edge after reset, the stack pointer is reset, the command register is set to its default values, and maskable interrupts are disabled. A Jump-to-Zero is executed during the cycle following RESET becoming active.

CP Clock Pulse Pin (Input)

The clock input to the Am29112.

ACIO Adder Carry I/O (Input/Output)

Carry I/O line for the adder for two cascaded Am29112s. When not cascading two Am29112s, leave unconnected.

PCIO Program Counter I/O (Input/Output)

Carry I/O line for the program counter for two cascaded Am29112s. When not cascading two Am29112s, leave unconnected.

CIO Counter I/O (Input/Output)

Carry I/O line for the counter for two cascaded Am29112s. When not cascading two Am29112s, leave unconnected.

CZIO Counter Zero I/O (Input/Output)

Counter zero detect for two cascaded Am29112s. This is an open-collector output which should be OR-tied to another Am29112 using a 1K pull-up resistor. When not cascading two Am29112s, leave unconnected.

FUNCTIONAL DESCRIPTION

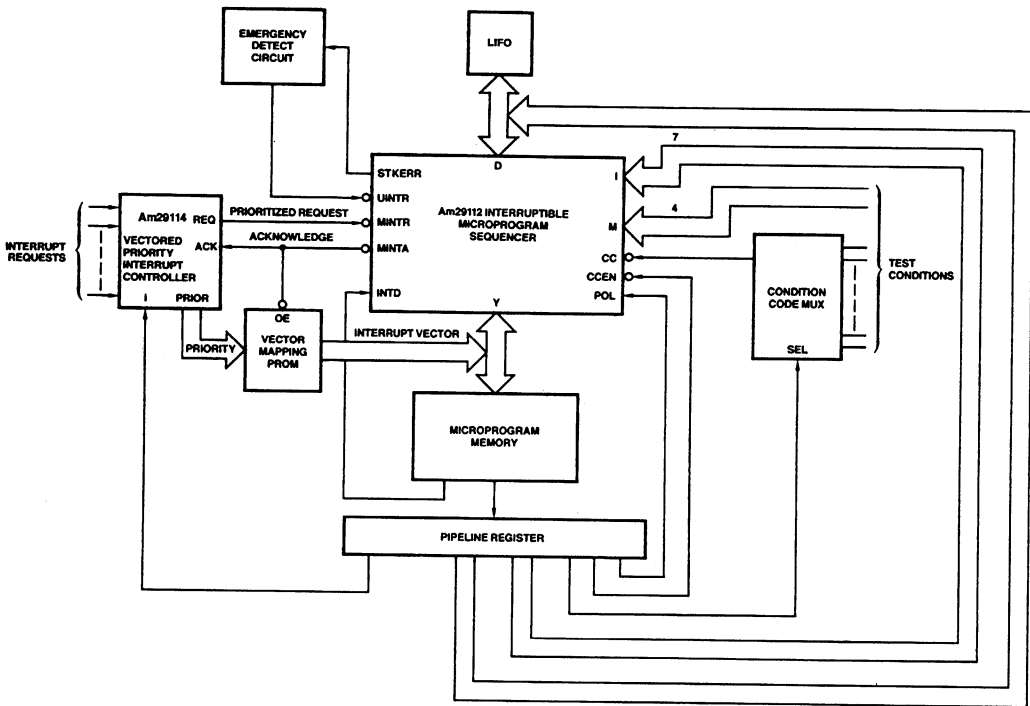
The Am29112 is designed for use in single-level pipelined systems. A typical configuration is shown in Figure 2.

Branch addresses and constants for the various registers are supplied to the Am29112 through the D port which is bidirectional to allow the stack to be unloaded onto an external LIFO. The next address generated by the sequencer is output on the Y port and directly drives the microprogram memory. A single pipeline register at the output of the microprogram memory contains the microinstruction being executed, while the next is being fetched. External conditions are applied to the \overline{CC} input of the Am29112 via the condition code MUX and also to the multiway inputs.

A vectored priority interrupt controller generates a prioritized interrupt request (\overline{MINTR}) to the Am29112, which acknowl-

edges the request via the \overline{MINTA} pin. Upon receiving the acknowledge, the priority interrupt controller puts out the encoded priority of the interrupt, which is translated to a vector by the vector mapping PROM. The \overline{MINTA} output of the Am29112 turns on the PROM output and simultaneously switches the Y port from output to input, enabling the interrupt vector onto the microprogram address bus. In the Am29112, the next address of the interrupted sequence is automatically saved on the stack while the interrupt vector is transmitted through the Y port and incremented to form the next microprogram address.

The emergency detect circuit generates an unmaskable interrupt request upon power failure or stack error. On receiving an unmaskable interrupt, the sequencer branches to the unmaskable interrupt routine; the address of this routine is stored on the Am29112 in the INTVECT register. Detailed interrupt handling is discussed in a later section.



BD001922

Figure 2. Control Path in a Single Pipelined System Using the Am29112

Architecture of the Am29112

The internal organization of the Am29112 is shown in Figure 1. The most important control loop inside the sequencer consists of the CMUX, incrementer, and PC register. The CMUX selects the next microprogram address based on the instruction and condition code inputs. The next microprogram address is selected from the PC register for a continue, the D port for a branch, the adder for relative and multiway branches, the interrupt register for unmaskable interrupts, the stack for subroutine returns or loop repeats, or all zeros for the JUMP ZERO instruction.

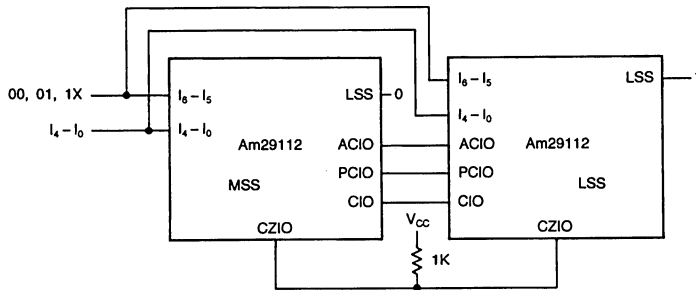
The Am29112 has many registers other than the PC register and the interrupt register. There is an 8-bit counter used for loop control; the DWIDTH register is a 4-bit register which programs the number of least significant bits of the D port that are added to the PC in relative addressing modes; the stack pointer is a 6-bit counter/register that points to the top of stack element; the 3-bit command register is used to program the chip on reset for compatibility with the external hardware

configuration; finally, there is the INTRTN register which is used for saving the CMUX output on the stack when an interrupt occurs.

With the exception of the INTRTN register, the stack pointer, and the PC register, each of the above registers can be loaded directly from the D port of the Am29112.

The Am29112 features a high-speed adder with full carry lookahead across 8-bits. The adder is used for PC relative addressing (branch address is PC + D), multiway relative addressing (branch address is D + M, where M is the 4-bit multiway input), and for testing the stack pointer against the D bus. In cascaded configurations (see Figure 3), carry ripples from the LSS adder to the MSS adder over the ACIO line.

The on-chip stack is 33 deep, and the Am29112 has instructions to save the D inputs, counter, multiway register, and PC-register on the stack. The stack output bus is connected via three-state buffers to the D port. It is possible to pop the stack to the D port. It is also possible to push data from the D port onto the stack.



LD000320

Figure 3. 16-Bit Configuration

Instruction Set of The Am29112

Mode Bits (I₆, I₅)

The Am29112 is controlled by five instruction inputs, two mode inputs, and the condition code. In typical applications it is expected that the instruction inputs are driven directly from the pipeline, whereas the mode inputs are either permanently wired high or low to select the desired operating mode, or driven indirectly via external logic. (In some applications it might be justifiable to drive the mode bits directly from the pipeline.) The two mode bits select among three operating modes: normal (00), extended (01), and forced continue (10 and 11). In the normal mode, the entire instruction set of the Am29112 applies.

TABLE 1. MODE CONTROLS

I ₆ , I ₅	Mode	Description
00	Normal	For cascaded Am29112s, two independent 8-bit counters
01	Extended	For cascaded Am29112s, one 16-bit counter
10	Forced Continue	The Am29112 executes a continue instruction regardless of instruction, condition code, and multiway inputs.
11		

Extended Mode

The instruction set includes instructions that differentiate between upper and lower counters (when there are two cascaded Am29112s). In the normal mode, the two counters on cascaded Am29112s function independently, and it is possible to set up a doubly nested loop without having to save and restore counter values on the stack. In the extended mode, however, the counters on cascaded Am29112s behave like one 16-bit counter and instructions that differentiate between the counters degenerate into identical instructions. Hence in a system with only one Am29112 there is no use for the extended mode.

Forced Continue Mode

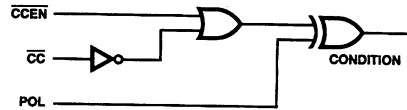
In the forced continue mode the Am29112 executes a continue in every cycle regardless of the instruction bits, condition code, and multiway inputs. The simplest application (if mode bits are driven directly from the pipeline) is to use forced continue for straight-line segments of code, thereby permitting most of the sequencer control fields of the pipeline

to be shared. The forced continue also has an important application in systems with a writable control store (WCS) where it is necessary to step through the addresses sequentially while loading the WCS.

The instructions of the Am29112 are classified into four groups:

- Branching and subroutine linkage
- Looping
- Stack and register
- Interrupt

The sequencer has an instruction repertoire of altogether thirty-nine different instructions. In order to encode these instructions with only five instruction lines, the condition code is used in some cases to differentiate between two distinct instructions sharing the same opcode. This way of encoding is used for the stack and register, and interrupt groups of instructions. For these instructions, therefore, the condition code multiplexer is not used to select an external condition. However it is required to force the condition code to unconditional Pass or Fail. The condition code enable and polarity logic has been designed with this in mind. Using the enable and polarity, it is possible to generate both unconditional Pass and unconditional Fail (regardless of the condition code input). Hence the condition code for these instructions is like a sixth instruction line, and the condition code multiplexer field of the pipeline can be shared for these instructions (see Figure 4 and Table 2).



PF001062

Figure 4. Condition Code Circuit.

TABLE 2. CONDITION CODE TABLE

CCEN	CC	POL	Condition
0	0	0	PASS
0	1	0	FAIL
0	0	1	FAIL
0	1	1	PASS
1	X	0	PASS
1	X	1	FAIL

TABLE 3. Am29112 INSTRUCTION SET

Opcode (I₄₀)	Condition	Mnemonic (Note 1)	Description
00 (00 _H)	Unconditional	JZ.U	Unconditional Jump Zero
01 (01 _H)	Forced Pass	PUSHD.P	Push D (Pass)
01 (01 _H)	Forced Fail	LDCMD.F	Load Command Register from D (Fail)
02 (02 _H)	Conditional	POP.C	Pop; Conditional Stack out to D
03 (03 _H)	Conditional	CJD.C	Conditional Jump D
04 (04 _H)	Conditional	CJSD.C	Conditional Jump Subroutine D
05 (05 _H)	Conditional	CJMW.C	Conditional Jump Multiway D
06 (06 _H)	Conditional	CJSMW.C	Conditional Jump Subroutine Multiway D
07 (07 _H)	Conditional	CRTN.C	Conditional Return
08 (08 _H) (Note 2)	Conditional	PUSHPL.C	Push PC; Conditional Load Lower Counter
09 (09 _H) (Note 3)	Conditional	LDLC.C	Load Lower Counter; Conditional Push Counter
10 (0A _H)	Unconditional	POPLC.U	Pop to Lower Counter
11 (0B _H)	Forced Pass	RSTSP.P	Reset Stack Pointer (Pass)
11 (0B _H)	Forced Fail	LDINTV.F	Load Unmaskable Interrupt Vector (Fail)
12 (0C _H) (Note 4)	Forced Pass	RFCTU.P	Repeat Loop, Upper Counter $\neq 0$ (Pass)
12 (0C _H) (Note 4)	Forced Fail	RFCTL.F	Repeat Loop, Lower Counter $\neq 0$ (Fail)
13 (0D _H) (Note 5)	Forced Pass	RPCTU.P	Repeat Pipeline, Upper Counter $\neq 0$ (Pass)
13 (0D _H) (Note 5)	Forced Fail	RPCTL.F	Repeat Pipeline, Lower Counter $\neq 0$ (Fail)
14 (0E _H)	Conditional	LOOP.C	Test End Loop
15 (0F _H)	Forced Pass	ENINT.P	Enable Interrupts (Pass)
15 (0F _H)	Forced Fail	DISINT.F	Disable Interrupts (Fail)
16 (10 _H) (Note 6)	Conditional	TWBL.C	Three-Way Branch, Lower Counter
17 (11 _H) (Note 6)	Conditional	TWBU.C	Three-Way Branch, Upper Counter
18 (12 _H)	Forced Pass	TSTSP.P	Test SP with D (Pass)
18 (12 _H)	Forced Fail	TSTMT.F	Jump D if Stack not Empty
19 (13 _H)	Conditional	CJDF.C	Conditional Jump D/Stack and Pop
20 (14 _H)	Conditional	CJSDF.C	Conditional Jump Subroutine D/Stack and Pop
21 (15 _H)	Conditional	CJMWR.C	Conditional Jump Multiway Relative D
22 (16 _H)	Conditional	CJSMWR.C	Conditional Jump Subroutine Multiway Relative D
23 (17 _H)	Conditional	CJPP.C	Conditional Jump Pipeline and Pop
24 (18 _H) (Note 2)	Conditional	PUSHPU.C	Push PC; Conditional Load Upper Counter
25 (19 _H) (Note 3)	Conditional	LDUC.C	Load Upper Counter; Conditional Push Counter
26 (1A _H)	Forced Pass	POPUC.P	Pop to Upper Counter (Pass)
26 (1A _H)	Forced Fail	POPWF.F	Pop to Displacement Width (DW) Register (Fail)
27 (1B _H)	Conditional	LDDW.C	Load DW Register; Conditional Push DW Register
28 (1C _H)	Conditional	CJR.C	Conditional Jump D PC Rel
29 (1D _H)	Conditional	CJRN.C	Conditional Jump D PC Rel Negative
30 (1E _H)	Conditional	CJSR.C	Conditional Jump Subroutine D PC Rel
31 (1F _H)	Conditional	CJSRN.C	Conditional Jump Subroutine D PC Rel Negative

- Notes: 1. Extensions: U = Unconditional; C = Conditional; P = Pass Condition; F = Fail Condition.
 2. For two cascaded Am29112s in the extended mode, the two instructions are identical.
 3. For two cascaded Am29112s in the extended mode, the two instructions are identical.
 4. For two cascaded Am29112s in the extended mode, the two instructions are identical.
 5. For two cascaded Am29112s in the extended mode, the two instructions are identical.
 6. For two cascaded Am29112s in the extended mode, the two instructions are identical.

TABLE 4. JUMP INSTRUCTIONS

Instruction Functions		Mnemonic	Opcode	Condition	Y	Stack
Direct	One-Way	JZ.U (Note 1)	00 (00H)	U	0	RESET
		CJD.C	03 (03H)	P	D	-
				F	(PC)	
		CJPP.C	23 (17H)	P	D	Pop
	F			(PC)	-	
	Two-Way	CJDF.C	19 (13H)	P	D	Pop
				F	(Stack)	
	Multi-Way	CJMW.C	05 (05H)	P	$Y_4 - Y_7 = D_4 - D_7$, $Y_0 - Y_3 = M_0 - M_3$ (Note 2)	-
F				(PC)		
Relative	One-Way	CJR.C	28 (1CH)	P	(PC) + D\$ (Note 3)	-
				F	(PC)	
		CJRN.C	29 (1DH)	P	(PC) + D* (Note 4)	-
				F	(PC)	
	Multi-Way	CJMWR.C	21 (15H)	P	D + M (Note 5)	-
				F	(PC)	

- Notes: 1. JZ.U also resets the Command Register and the interrupt logic.
 2. $Y_0 - Y_7$ is as shown if $CR_1 = 1$. If $CR_1 = 0$, then $Y_0 - Y_7 = D_0 - D_7$.
 3. D\$ represents the number of D bits used as displacement; the remaining high-order bits are zero-extended.
 4. D* represents the number of D bits used as displacement; the remaining high-order bits are one-extended. D should be a two's-complement number.
 5. $Y_0 - Y_7 = D + M$ if $CR_1 = 1$. If $CR_1 = 0$, then $Y_0 - Y_7 = D_0 - D_7$.

TABLE 5. SUBROUTINE INSTRUCTIONS

Instruction Functions		Mnemonic	Opcode	Condition	Y	Stack
Direct	One-Way	CJSD.C	04 (04H)	P	D	Push
				F	(PC)	-
		CRTN.C	07 (07H)	P	(Stack)	Pop
				F	(PC)	-
	Two-Way	CJSDF.C	20 (14H)	P	D	Pop and Push
				F	(Stack)	
	Multi-Way	CJSMW.C	06 (06H)	P	$Y_4 - Y_7 = D_4 - D_7,$ $Y_0 - Y_3 = M_0 - M_3$ (Note 1)	Push
				F	(PC)	-
Relative	One-Way	CJSR.C	30 (1EH)	P	(PC) + D\$ (Note 2)	Push
				F	(PC)	-
		CJSRN.C	31 (1FH)	P	(PC) + D* (Note 3)	Push
				F	(PC)	-
	Multi-Way	CJSMWR.C	22 (16H)	P	D + M (Note 4)	Push
				F	(PC)	-

- Notes: 1. $Y_0 - Y_7$ is as shown if $CR_1 = 1$. If $CR_1 = 0$, then $Y_0 - Y_7 = D_0 - D_7$.
 2. D\$ represents the number of D bits used as displacement; the remaining high-order bits are zero-extended.
 3. D* represents the number of D bits used as displacement; the remaining high-order bits are one-extended. D should be a two's-complement number.
 4. $Y_0 - Y_7 = D + M$ if $CR_1 = 1$. If $CR_1 = 0$, then $Y_0 - Y_7 = D_0 - D_7$.

TABLE 6. REPEAT INSTRUCTIONS

Instruction Functions	Mnemonic	Opcode	Condition	Upper Count	Lower Count	Y	Stack
Loop	RFCTU.P (Note 1)	12 (0C _H)	P	> 0		(Stack)	-
				= 0		(PC)	Pop
	RFCTL.F (Note 1)		F	> 0		(Stack)	-
				= 0		(PC)	Pop
	RPCTU.P (Note 2)	13 (0D _H)	P	> 0		D	-
				= 0		(PC)	
	RPCTL.F (Note 2)		F	> 0		D	-
				= 0		(PC)	
LOOP.C	14 (0E _H)	P			(PC)	Pop	
		F			(Stack)	-	
Loop and Branch	TWBU.C (Note 3)	17 (11 _H)	P			(PC)	Pop
			F	> 0		(Stack)	-
				= 0		D	Pop
	TWBL.C (Note 3)		16 (10 _H)	P			(PC)
		F		> 0		(Stack)	-
			= 0		D	Pop	

Notes: 1. For two cascaded Am29112s in the extended mode, the two instructions are identical.
 2. For two cascaded Am29112s in the extended mode, the two instructions are identical.
 3. For two cascaded Am29112s in the extended mode, the two instructions are identical.

TABLE 7. TESTING INSTRUCTIONS

Instruction Functions	Mnemonic	Opcode	Condition		Y	Stack
Test SP and Branch	TSTSP.P	18 (12 _H)	P	$SP + D \leq 33$	(PC)	-
				$SP + D > 33$	(Stack)	Pop
	TSTMT.F		F	$SP \neq 0$	D	-
				$SP = 0$	(PC)	

TABLE 8. OTHER INSTRUCTIONS

Opcode	Mnemonic	Condition	Functions	Stack
01 (01H)	PUSHD.P	P	D → Stack	Push
	LDCMD.F	F	D → Command Register	–
02 (02H)	POP.C	P	(Stack) → D	Pop
		F		
08 (08H)	PUSHPL.C (Note 1)	P	(PC) → Stack D → Lower Counter	Push
		F	(PC) → Stack	
09 (09H)	LDLC.C (Note 2)	P	D → Lower Counter (Counter) → Stack	Push
		F	D → Lower Counter	–
10 (0AH)	POPLC.U	U	(Stack) → Lower Counter	Pop
11 (0BH)	RSTSP.P	P	Reset SP	–
	LDINTV.F	F	D → Interrupt Vector Register	
15 (0FH)	ENINT.P	P	Enable Maskable Interrupt	–
	DISINT.F	F	Disable Maskable Interrupt	
24 (18H)	PUSHPU.C (Note 1)	P	(PC) → Stack D → Upper Counter	Push
		F	(PC) → Stack	
25 (19H)	LDUC.C (Note 2)	P	D → Upper Counter (Counter) → Stack	Push
		F	D → Upper Counter	–
26 (1AH)	POPUC.P	P	(Stack) → Upper Counter	Pop
	POPDW.F	F	(Stack) → Displacement Width (DW) Register	
27 (1BH)	LDDW.C	P	D → DW Register (DW Register) → Stack	Push
		F	D → DW Register	–

Notes: 1. For two cascaded Am29112s in the extended mode, the two instructions are identical.
 2. For two cascaded Am29112s in the extended mode, the two instructions are identical.

INSTRUCTION SET DEFINITION OF THE Am29112

Jump Instructions

Key: ● = Other Instruction
 ○ = Instruction Being Described
 ○ = Register in Part
 P = Pass

F = Fail
 U = Unconditional
 FP = Forced Pass
 FF = Forced Fail

Opcode	Mnemonic	Condition	Execution Example
00(00H)	JZ.U	Unconditional	<p>Description: Unconditional Jump Zero This instruction causes an Unconditional Jump to address location 0. This instruction also resets the stack pointer and interrupt logic (maskable interrupt disabled), as well as setting the Command Register as follows: CR₀ = CR₁ = CR₂ = 1.</p>

Opcode	Mnemonic	Condition	Execution Example
03(03H)	CJD.C	Conditional	<p>Description: Conditional Jump D Conditionally Jump to the address input on the D port. If the condition is a Pass, the next address is obtained from the D port. If the condition is a Fail, a Continue is executed.</p>

Opcode	Mnemonic	Condition	Execution Example
23(17H)	CJPP.C	Conditional	

Description: Conditional Jump Pipeline and Pop
 If the condition is a Pass, a Jump to D is executed and the Stack is popped.
 If the condition is a Fail, a Continue is executed.

Opcode	Mnemonic	Condition	Execution Example
19(13H)	CJDF.C	Conditional	

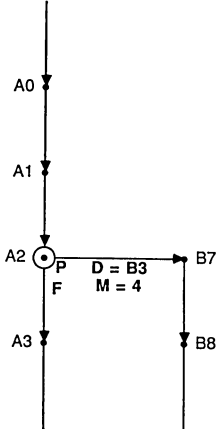
Description: Conditional Jump D/Stack and Pop
 If the condition is a Pass, a Jump to D is executed and the Stack is popped.
 If the condition is a Fail, a Jump is made to the address on the top of Stack and the Stack is popped.

Opcode	Mnemonic	Condition	Execution Example
05(05H)	CJMW.C	Conditional	

Description: Conditional Jump Multiway D
 If the condition is a Pass, a branch is taken to an address provided by the D and M inputs. The four M inputs directly replace the four LSBs of the branch address input at D. If the condition is a Pass and multiway branching is disabled in the Command Register, then a branch is taken to the address input at D. If the condition is a Fail, then a Continue is executed.

Opcode	Mnemonic	Condition	Execution Example
28(1C _H)	CJR.C	Conditional	
<p>Description: Conditional Jump D PC Rel If the condition is a Pass, a Jump is made to an address computed from the PC and D inputs. The specified D bits are added to the PC to form the branch address. The remaining D bits are ignored. If the condition is a Fail, a Continue is executed.</p> <p>Note: The number of D bits used as displacement is stored in the Displacement Width (DW) Register. The remaining high order bits are zero-extended.</p>			

Opcode	Mnemonic	Condition	Execution Example
29(1D _H)	CJRN.C	Conditional	
<p>Description: Conditional Jump D PC Rel Negative If the condition is a Pass, a Jump is made to an address computed from the PC and D inputs. The specified D bits are subtracted from the PC to form the branch address. The remaining bits are ignored. If the condition is a Fail, a Continue is executed.</p> <p>Note: The number of D bits used as displacement is stored in the DW Register. The remaining high order bits are one-extended. The D inputs should therefore present a two's complement number.</p>			

Opcode	Mnemonic	Condition	Execution Example
21(15H)	CJMWR.C	Conditional	 <p>The diagram illustrates the execution flow for the CJMWR.C instruction. It shows a vertical sequence of addresses A0, A1, A2, and A3. At address A2, there is a branch point. A horizontal arrow points to B7, and a vertical arrow points to B8. The branch point is labeled with 'P' (Pass) and 'F' (Fail). To the right of the branch point, the text 'D = B3' and 'M = 4' is shown.</p>
Description: Conditional Jump Multiway Relative D			
If the condition is a Pass, a branch is taken to an address computed from the D and M inputs. The constant on the M inputs is added to the D inputs to form the branch address. If the condition is a Pass and multiway branching is disabled in the Command Register, then a branch is taken to the address input at D. If the condition is a Fail, a Continue is executed.			

INSTRUCTION SET DEFINITION OF THE Am29112

Subroutine Instructions

Key: ● = Other Instruction

○ = Instruction Being Described

○ = Register in Part

P = Pass

F = Fail

U = Unconditional

FP = Forced Pass

FF = Forced Fail

Opcode	Mnemonic	Condition	Execution Example
04(04H)	CJSD.C	Conditional	

Description: Conditional Jump Subroutine D

Conditionally Jump to Subroutine. The subroutine address is input over the D port. If the condition is a Pass, the PC is saved on the Stack and a branch is taken to the subroutine address. If the condition is a Fail, a Continue is executed.

Opcode	Mnemonic	Condition	Execution Example
07(07H)	CRTN.C	Conditional	

Description: Conditional Return

If the condition is a Pass, a return is executed from the Interrupt or Subroutine. The return address is obtained from the Stack and the Stack is popped. If the condition is a Fail, a Continue is executed.

Opcode	Mnemonic	Condition	Execution Example
31(1FH)	CJSRN.C	Conditional	<p>Description: Conditional Jump Subroutine PC Rel Negative</p> <p>If the condition is a Pass, a subroutine call is made to an address computed from the PC and D inputs. The specified D bits are subtracted from the PC to form the subroutine starting address. The remaining D bits are ignored. The subroutine Return Address is saved on the Stack. If the condition is a Fail, a Continue is executed.</p> <p>Note: The number of bits of D used as displacement is stored in the DW Register. The remaining high order bits are one-extended. The D inputs should, therefore, present two's-complement number.</p>

Opcode	Mnemonic	Condition	Execution Example
22(16H)	CJSMWR.C	Conditional	<p>Description: Conditional Jump Subroutine Multiway Relative D</p> <p>If the condition is a Pass, a branch to subroutine is made. The subroutine starting address is computed from the D and M inputs. The constant on the M inputs is added to the D inputs to form the starting address. If the condition is a Pass and multiway branching is disabled in the Command Register, then a subroutine call is made to the address input at D. If the condition is a Fail, a Continue is executed.</p>

INSTRUCTION SET DEFINITION OF THE Am29112

Repeat Instructions

Key: ● = Other Instruction

○ = Instruction Being Described

○ = Register in Part

P = Pass

F = Fail

U = Unconditional

FP = Forced Pass

FF = Forced Fail

Opcode	Mnemonic	Condition	Execution Example
12(0C _H)	RFCTU.P	Forced Pass	

Description: Repeat Loop, Upper Counter \neq 0

On a Forced Pass condition, the upper counter value is checked. If found not equal to zero, execution branches to the address provided from the top of Stack and the counter is decremented. If found equal to zero, a Continue is executed and the Stack is popped. For two cascaded Am29112s in the Normal mode, the lower slice loops back but its counter is unaffected. For two cascaded Am29112s in the Extended mode, the instruction operates on the 16-bit concatenated counter.

Note: The counter is checked before being decremented. A count of 1 causes two iterations.

Opcode	Mnemonic	Condition	Execution Example
12(0C _H)	RFCTL.F	Forced Fail	

Description: Repeat Loop, Lower Counter \neq 0

On a Forced Fail condition, the lower counter value is checked. If found not equal to zero, execution branches to the address provided from the top of Stack and the counter is decremented. If found equal to zero, a Continue is executed and the Stack is popped. For two cascaded Am29112s in the Normal mode, the upper slice loops back but its counter is unaffected. For two cascaded Am29112s in the Extended mode, the instruction operates on the 16-bit concatenated counter.

Note: The counter is checked before being decremented. A count of 1 causes two iterations.

Opcode	Mnemonic	Condition	Execution Example
13(0D _H)	RPCTU.P	Forced Pass	

Description: Repeat Pipeline, Upper Counter \neq 0

On a Forced Pass condition, the upper counter value is checked. If found not equal to zero, execution branches to the address provided from the D inputs and the counter is decremented. If found equal to zero, a Continue is executed. For two cascaded Am29112s in the Normal mode, the lower slice loops back but its counter is unaffected. For two cascaded Am29112s in the Extended mode, the instruction operates on the 16-bit concatenated counter.

Note: The counter is checked before being decremented. A count of 1 causes two iterations.

Opcode	Mnemonic	Condition	Execution Example
13(0D _H)	RPCTL.F	Forced Fail	

Description: Repeat Pipeline, Lower Counter \neq 0

On a Forced Fail condition, the lower counter value is checked. If found not equal to zero, execution branches to the address provided from the D inputs and the counter is decremented. If found equal to zero, a Continue is executed. For two cascaded Am29112s in the Normal mode, the upper slice loops back but its counter is unaffected. For two cascaded Am29112s in the Extended mode, the instruction operates on the 16-bit concatenated counter.

Note: The counter is checked before being decremented. A count of 1 causes two iterations.

Opcode	Mnemonic	Condition	Execution Example
14(0E _H)	LOOP.C	Conditional	

Description: Test End Loop

If the condition is a Fail, program execution branches to the address obtained from the top of Stack. If the condition is a Pass, program execution continues and the Stack is popped.

Opcode	Mnemonic	Condition	Execution Example
17(11H)	TWBU.C	Conditional	

Description: Three-Way Branch, Upper Counter
 If the condition is a Pass, a Continue is executed and the Stack is popped. If the condition is a Fail, the upper counter value is checked. If found not equal to zero, execution branches to the address provided from the top of Stack and the counter is decremented. If the counter is equal to zero, execution branches to the address provided from the D inputs and the Stack is popped. For two cascaded Am29112s in the Normal mode, the lower slice branches or loops back as the case may be, but its counter remains unaffected. For two cascaded Am29112s in the Extended mode, the instruction operates on the 16-bit concatenated counter.

Note: The counter is checked before being decremented. A count of 1 causes two iterations.

Opcode	Mnemonic	Condition	Execution Example
16(10H)	TWBL	Conditional	

Description: Three-Way Branch, Lower Counter
 If the condition is a Pass, a Continue is executed and the Stack is popped. If the condition is a Fail, the lower counter value is checked. If found not equal to zero, execution branches to the address provided from the top of Stack and the counter is decremented. If the counter is equal to zero, execution branches to the address provided from the D inputs and the Stack is popped. For two cascaded Am29112s in the Normal mode, the upper slice branches or loops back as the case may be, but its counter remains unaffected. For two cascaded Am29112s in the Extended mode, the instruction operates on the 16-bit concatenated counter.

Note: The counter is checked before being decremented. A count of 1 causes two iterations.

INSTRUCTION SET DEFINITION OF THE Am29112

Testing Instructions

Key: ● = Other Instruction

○ = Instruction Being Described

○ = Register in Part

P = Pass

F = Fail

U = Unconditional

FP = Forced Pass

FF = Forced Fail

Opcode	Mnemonic	Condition	Execution Example
18(12H)	TSTSP.P	Forced Pass	<p>Description: Test SP with D On a Forced Pass condition, the sequencer tests the Stack to see if there is enough space, as determined by a constant input at the D port. (The constant must be a hexadecimal number.) If the available number of Stack locations is greater than or equal to the constant, a Continue will be executed. If the available number of Stack locations is less than the constant, a Subroutine Return is executed.</p>

Opcode	Mnemonic	Condition	Execution Example
18(12H)	TSTMT.F	Forced Fail	<p>Description: Jump D if Stack Not Empty On a Forced Fail condition, the Stack is checked to see if it is empty. If empty, a Continue is executed. If not empty, a Jump to D is executed.</p>

INSTRUCTION SET DEFINITION OF THE Am29112

Other Instructions

Key: ● = Other Instruction

○ = Instruction Being Described

○ = Register in Part

P = Pass

F = Fail

U = Unconditional

FP = Forced Pass

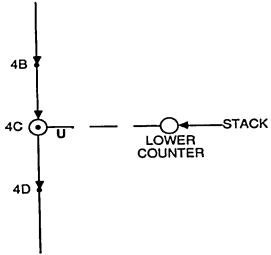
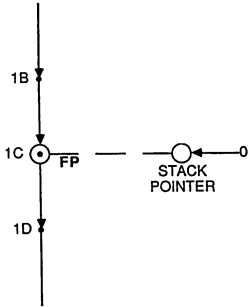
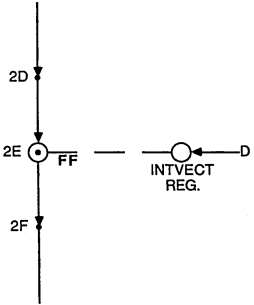
FF = Forced Fail

Opcode	Mnemonic	Condition	Execution Example
01(01H)	PUSHD.P	Forced Pass	
<p>Description: Push D Push the D port onto the top of Stack under the Forced Pass condition. The next address is a Continue.</p>			
Opcode	Mnemonic	Condition	Execution Example
01(01H)	LDCMD.F	Forced Fail	
<p>Description: Load Command Register from D On a Forced Fail condition, the Command Register is loaded from the least significant three bits of the D input. The remaining bits on the D Port are "don't cares". The three bits of the Command Register are described below: CR₀ — Interrupt acknowledge on Stack full CR₀ = 1 : inhibit acknowledge on Stack full (default) CR₀ = 0 : generate acknowledge on Stack full CR₁ — Multiway Enable CR₁ = 1 : Enable multiway branching (default for LSS) CR₁ = 0 : Disable multiway branching (default for MSS) CR₂ — Interrupt post-delay CR₂ = 1 : No post-delay (default) CR₂ = 0 : Post-delay Note: When CR₀ = 1, the Stack is checked for a minimum of five locations before acknowledging an interrupt.</p>			

Opcode	Mnemonic	Condition	Execution Example
02(02H)	POP.C	Conditional	<p>Description: Pop; Conditional Stack Out to D This instruction unconditionally pops the Stack. Also, on a Pass condition, the value popped is output on the D port. Care should be taken to avoid bus contention on the D Port when this is done.</p>

Opcode	Mnemonic	Condition	Execution Example
08(08H)	PUSHPL.C	Conditional	<p>Description: Push PC; Conditional Load Lower Counter If the condition is a Pass, the PC is pushed onto the Stack and the lower counter is loaded from the D inputs. If the condition is a Fail, the PC is pushed onto the Stack and the lower counter remains unchanged. In either case, a Continue is executed. For two cascaded Am29112s in the Normal mode, the PC is pushed onto the Stack in both devices and the high order counter remains unchanged. For two cascaded Am29112s in the Extended mode, the PC is pushed onto the Stack in both devices. If the condition is a Pass, the concatenated 16-bit counter is loaded from the respective D inputs.</p>

Opcode	Mnemonic	Condition	Execution Example
09(09H)	LDLC.C	Conditional	<p>Description: Load Lower Counter; Conditional Push Counter If the condition is a Pass, the lower counter is loaded from the D inputs and the previous value in the counter is pushed onto the Stack. If the condition is a Fail, the lower counter is loaded from the D inputs and the previous value in the counter is lost. In either case, a Continue is executed. For two cascaded Am29112s in the Normal mode, the high order counter is conditionally pushed onto the Stack, but its value remains unchanged. For two cascaded Am29112s in the Extended mode, the 16-bit counter is loaded from the respective D inputs, and the previous 16-bit value from the counter is conditionally pushed onto the Stack.</p>

Opcode	Mnemonic	Condition	Execution Example
10(0A _H)	POPLC.U	Unconditional	<p>Description: Pop to Lower Counter The Stack is popped to the lower counter unconditionally. For two cascaded Am29112s in the Normal mode, the Stack of the high order slice is also popped but the higher counter remains unchanged. For two cascaded Am29112s in the Extended mode, the Stack is unconditionally popped to the 16-bit concatenated counter.</p> 
11(0B _H)	RSTSP.P	Forced Pass	<p>Description: Reset Stack Pointer On a Forced Pass condition, the Stack Pointer is reset to 0. All previous Stack contents are lost.</p> 
11(0B _H)	LDINTV.F	Forced Fail	<p>Description: Load Unmaskable Interrupt Vector On a Forced Fail condition, the Interrupt Vector Register is loaded from the D inputs.</p> 

Opcode	Mnemonic	Condition	Execution Example
15(0FH)	ENINT.P	Forced Pass	<p>Description: Enable Interrupts On a Forced Pass condition, enable maskable interrupts.</p>

Opcode	Mnemonic	Condition	Execution Example
15(0FH)	DISINT.F	Forced Fail	<p>Description: Disable Interrupts On a Forced Fail condition, disable maskable interrupts.</p>

Opcode	Mnemonic	Condition	Execution Example
24(18H)	PUSHPU.C	Conditional	<p>Description: Push PC; Conditional Load Upper Counter If the condition is a Pass, the PC is pushed onto the Stack and the upper counter is loaded from the D inputs. If the condition is a Fail, the PC is pushed onto the Stack and the upper counter remains unchanged. In either case, a Continue is executed. For two cascaded Am29112s in the Normal mode, the PC is pushed onto the Stack in both devices and the low order counter remains unchanged. For two cascaded Am29112s in the Extended mode, the PC is pushed onto the Stack in both devices and if the condition is a Pass, the concatenated 16-bit counter is loaded from the respective D inputs.</p>

Opcode	Mnemonic	Condition	Execution Example
25(19H)	LDUC.C	Conditional	<p>Description: Load Upper Counter; Conditional Push Counter</p> <p>If the condition is a Pass, the upper counter is loaded from the D inputs and the previous value in the counter is pushed onto the Stack. If the condition is a Fail, the upper counter is loaded from the D inputs and the previous value in the counter is lost. In either case, a Continue is executed. For two cascaded Am29112s in the Normal mode, the low order counter is conditionally pushed onto the Stack, but its value remains unchanged. For two cascaded Am29112s in the Extended mode, the 16-bit counter is loaded from the respective D inputs and the previous 16-bit value from the counter is conditionally pushed onto the Stack.</p>

Opcode	Mnemonic	Condition	Execution Example
26(1AH)	POPUC.P	Forced Pass	<p>Description: Pop to Upper Counter</p> <p>On a Forced Pass condition, the Stack is popped to the upper counter. For two cascaded Am29112s in the Normal mode, the Stack of the low order slice is also popped but the low order counter remains unchanged. For two cascaded Am29112s in the Extended mode, on a Forced Pass condition, the Stack is popped to the 16-bit concatenated counter.</p>

Opcode	Mnemonic	Condition	Execution Example
26(1AH)	POPDW.F	Forced Fail	<p>Description: Pop to Displacement Width Register</p> <p>On a Forced Fail condition, the Stack is popped to the 4-bit Displacement Width (DW) Register. The four MSBs are ignored.</p>

Opcode

Mnemonic

Condition

Execution Example

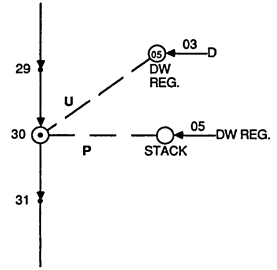
27(1B_H)

LDDW.C

Conditional

Description: Load Displacement Width (DW) Register; Conditional Push DW Register

The 4-bit Displacement Width (DW) Register is unconditionally loaded from the four LSBs of the D input. (The four MSBs of the D input are ignored.) If the condition is a Pass, the previous value of the Displacement Width Register is pushed onto the Stack. If the condition is a Fail, the previous value of the Displacement Width Register is lost.



Branching Instructions

Direct Branching

Instruction 0 is the unconditional jump to zero instruction. This instruction also executes an immediate reset (it is not delayed until the following clock cycle)

Direct branching is implemented by instruction 3 (COND JUMP D) and 4 (COND JSB D). The branch address is input through the D port. If the condition is PASS, the branch is taken, otherwise the sequencer executes a continue. Two-way direct branching is implemented by instruction 19 (COND JMP D/STACK) and instruction 20 (COND JSB D/STACK). If the condition is Pass, the branch address is taken from the D input port, otherwise, the branch address is taken from the stack. In either case the stack is popped. This instruction assumes that the alternative address was pushed on the stack by a previous instruction. Jump to subroutine differs from JUMP in that the PC register is pushed on the stack. This enables the subroutine to use COND RETURN (7) to return to the point of call. Note that the two-way jump to subroutine (20) causes a simultaneous pop and push so that the stack pointer is unaffected but the top of stack element is replaced by the return address.

Relative Branching

In the relative branch instructions, a dynamically alterable subfield of the D inputs is added to the PC to form the branch address. The remaining most significant bits of the D inputs are ignored and internally converted to all 0's for forward branches and all 1's for backward branches. The displacement width (DW) Register in the Am29112 holds the number of least significant bits of D that participate in the relative branch as the displacement, and can be loaded from the lower four bits of the D port. In cascaded systems, the displacement width has to be loaded consistently in the two chips. For example, for a displacement width of 9, the lower order chip gets a displacement width of 8 and the higher order chip gets a displacement width of 1. As another example, if the lower order chip has a displacement width of less than 8 bits, the higher order chip must have a displacement width of zero. If the displacement width register is loaded with any value greater than 8, it is exactly as if it were loaded with 8.

Instruction 28 (29) is the relative jump (jump back) instruction, and instruction 30 (31) is the relative jump to subroutine (jump back to subroutine) instruction. For backward relative branches, the displacement must be coded as a two's complement negative number. When the displacement width is the same as the microaddress width the forward and backward relative branch instructions are identical. When the displacement width is less than the microaddress width, the more significant bits of D outside the displacement are forced to all zeros for positive branches and to all ones for negative branches. This is effectively sign extension except that the sign information is contained in the instruction rather than the displacement, and there is no need for sign information to propagate between cascaded chips since it is assumed that the displacement width registers in the two chips have been consistently loaded.

The disadvantage of having the sign information in the instruction rather than the displacement can be overcome by a judicious choice of instruction format. The opcodes for forward and backward relative branch instructions have been chosen to differ in the least significant bit position only, with a "0" in that bit for forward branches and a "1" for backward branches. If the sequencer instruction field is contiguous with and on the more significant side of the displacement field in the pipeline register, then the least significant instruction bit is like the sign bit for the displacement for relative branch instructions. This permits the assembler to use the same

opcode for forward and backward relative branch instructions, but *overlap* the displacement field (now declared to be one bit longer than the actual displacement field in the pipeline) with the sequencer instruction field by one bit. If the assembler now generates a negative displacement, the sequencer opcode formed is the backward branch; while if the displacement is positive, the sequencer opcode formed is forward branch.

When the instruction is executed, the PC already has been incremented and points to the next sequential instruction, hence a forward branch with a displacement of 0 causes the next sequential instruction to be executed.

Multway Branching

Two variants of multiway branching are available on the Am29112 – multiway substitute D (see Figure 5) and multiway relative D (see Figure 6). In multiway substitute D the 4 multiway inputs directly replace the 4 least significant bits of the branch address input at D. Instruction 5 is a conditional multiway branch and instruction 6 a conditional multiway subroutine call. In these instructions, the least significant 4 bits of the D input port are not used by the sequencer, and may be shared, for instance to select among different sets of multiway inputs.

1. Concatenation

(a) With one Am29112

$$Y_7 - Y_0 = D_7 - D_4 \bullet M_3 - M_0$$

(b) With two Am29112

$$Y_{15} - Y_0 = D_{15} - D_{12} \bullet M_7 - M_4 \bullet D_7 - D_4 \bullet M_3 - M_0$$

(*dot) denotes concatenation

Figure 5. Multiway Branch — Concatenation

2. Relative to data inputs

(a) With one Am29112

$$Y_7 - Y_0 = D_7 - D_0 + M_3 - M_0$$

(b) With two Am29112

$$Y_{15} - Y_8 = D_{15} - D_8 + M_7 - M_4 + ACIO$$

$$Y_7 - Y_0 = D_7 - D_0 + M_3 - M_0$$

Figure 6. Multiway Branch — Relative to Data Inputs

Multway branching has the disadvantage that the jump table must be aligned on a 16-word boundary. This disadvantage is overcome in the Am29112 multiway relative branching instructions. In these instructions, the number input on the multiway pins is added to the branch address input at D. Instruction 21 is a conditional multiway relative branch and instruction 22 a conditional multiway relative subroutine call.

One of the advantages of multiway branching is that it enables a 16-way decision to be made in exactly one microcycle. However, the 16 target addresses are constrained to be contiguous in memory. Hence, if the target routines need more than one microword each, as is very likely, they are addressed indirectly through a table of 16 contiguous branch instructions. For very high-speed applications, the extra microcycle needed to branch indirectly off the jump table may not be acceptable. This penalty is avoidable if the multiway bits are offset with respect to the D inputs. When two cascaded Am29112s are used, there are two sets of 4-bit multiway inputs. The least significant chip has a multiway input with no offset, while the

most significant chip has a multiway input with an 8-bit offset. The Am29112 command register has a bit CR_1 that enables or disables multiway branching on the chip. In a system with two cascaded Am29112s, each chip has a command register bit. Multiway branching may be disabled in either chip by resetting the command register bit on that chip, or enabled by setting the command register bit. When multiway branching is disabled on a chip, for that chip both multiway and multiway relative branches are converted to direct branches, and the multiway inputs are a Don't Care. Multiway branching with an 8-bit offset is implemented by disabling multiway in the least significant slice and enabling it in the most significant slice. In this case, the 16 target addresses are dispersed in memory, separated by 256 locations each. Another useful configuration is obtained by enabling multiway on both chips. In this case, up to 16 sets of target addresses are dispersed in memory, separated by 256 locations each (see Figure 7).

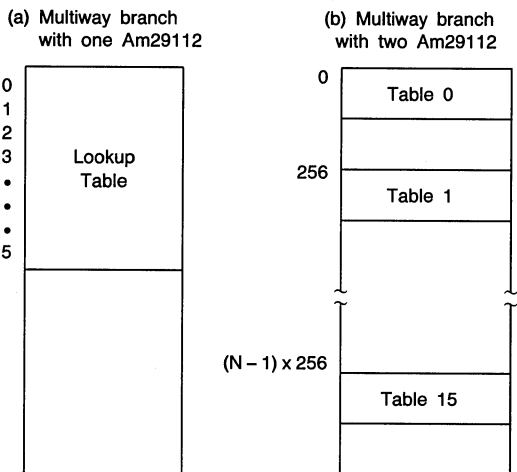


Figure 7. Table Lookup Using Multiway

The Am29112 does not have an unconditional continue in its instruction set. This is not expected to be a drawback because the instruction set requires that both Unconditional Pass and Unconditional Fail are programmable by the sequencer to select among different instructions sharing the same opcode. Hence, a continue is obtained by executing instruction 3 (COND JUMP D) with a Forced Fail condition.

Looping Instructions

The looping instructions on the Am29112 are of two kinds: conditional, which depend on an external condition to signal loop termination, and iterative, which decrement the Am29112 counter and check for a count of zero. There is also a three-way branch instruction that combines the check for external condition with the check for count of zero in a single instruction.

All the looping instructions are similar in two respects. Firstly, the check for the loop condition is done at the end of the loop. This implies that the loop body is always executed at least once. Secondly, in the case that the loop has to be repeated, a backward branch to the loop head is made by using the address on top of stack. This frees the D inputs for other use, but makes it necessary to push the address of the start of the loop on the stack before entering the loop. Also, if the loop is iterative, it is necessary to load a count value in the counter at the same time. Instructions 24 (PUSH PC; COND LOAD UPPER COUNTER) and 8 (PUSH PC; COND LOAD LOWER COUNTER) combine both these requirements.

Instruction 14 implements a simple conditional repeat loop. If the condition is FAIL the sequencer loops back using the top of stack address, and if the condition is PASS, the sequencer performs a continue to the next sequential address, and simultaneously pops the stack to remove the address of the loop head. The instruction may be described in Pascal-like syntax as:

```
repeat PUSH PC
LOOP BODY
until condition = TRUE;
```

Instruction 23 (COND JUMP PIPELINE AND POP) implements a loop exit that may be used with any of the Am29112 loop instructions. It is a conditional jump to D, which simultaneously pops the stack. If the condition is FAIL, it simply performs a continue.

As discussed earlier, the counters present in cascaded Am29112s may be used independently or cascaded as a single 16-bit counter under microprogram control. The mode bits select the cascaded configuration only in the extended mode. There are separate repeat and three-way branch instructions for upper and lower counters. In the case of the repeat instructions, the condition code is used to differentiate between the repeat on the upper and the repeat on lower counter (a condition of PASS selects the upper counter). In the case of the three-way branch, which needs the condition code input for the external condition, there are two separate opcodes for three-way branch on upper (opcode 17) and three-way branch on lower (opcode 16) counters. When a single Am29112 is used only the repeat on lower counter instructions are useful; and when two Am29112s are cascaded but operated in the extended mode, the repeat instructions on upper and lower counter are identical in effect and both operate on the 16-bit cascaded counter.

Instruction 12 (REPEAT LOOP IF COUNTER NOT ZERO) is the iterative analog of instruction 14 (TEST END LOOP). Instruction 8 (PUSH PC; COND LOAD LOWER COUNTER) is used with condition code as forced PASS and the desired count in the D field of pipeline. This causes the address of the loop head to be pushed on the stack, and the lower counter loaded with the count. At the end of the loop body, the repeat instruction checks if the count is zero. If it is not zero, it performs a loop back using the top of stack address and simultaneously decrements the counter; if it is zero, it pops the address of the loop head off the stack and simultaneously selects the next sequential address thereby exiting the loop. A repeat loop on the upper counter can be set up using instruction 24 instead of 8 to push PC and load upper counter and using instruction 12 to loop back with condition code as forced PASS. Note the potential off-by-one error: since the count is checked before it is decremented, a count of 1 causes two iterations: the first iteration finds a count of 1 and decrements; on the second iteration the count is found to be zero and the loop terminates. Hence, the value of count loaded should be *one less* than the desired number of iterations. In the example above, loading the counter with 7 resulted in 8 iterations.

The single instruction repeat (instruction 13) is provided for applications where the loop body is a single microinstruction, for example, an ALU shift. The loop is set up as before using instruction 9 (LOAD LOWER COUNTER; COND PUSH COUNTER) or 25 (LOAD UPPER COUNTER; COND PUSH COUNTER). The repeat instruction then presents its own address to the D inputs of the sequencer. As with the repeat loop instruction, the single instruction repeat checks for counter = 0. If the counter is equal to zero, it continues to the next sequential instruction; otherwise it repeats the address presented to the D inputs, and decrements the count by one.

Instruction 13 can also be used in place of instruction 12 where there is no stack location available to hold the address of the loop head.

Often it is necessary to repeat an action until either some external condition becomes true or a predetermined count is reached: for example, searching a character string for an occurrence of some character. The three-way branch instructions of the Am29112 combine the test for count and external condition in one cycle. At any loop iteration, if the condition becomes PASS when the three-way branch is executed, then the sequencer performs a continue to the next sequential instruction, and pops the stack. If the condition is FAIL when the three-way branch is executed, the sequencer tests the count. If the count is zero, then the search is unsuccessful and the sequencer performs a branch to the address input at the D port, simultaneously popping the stack. If the count is not zero, and the condition is FAIL, the sequencer performs a loop back via the stack. The instruction always decrements the counter by one if the counter is non-zero.

Since interrupts may occur at any point in the execution of microcode, it is necessary to be able to save counter values on the stack so that the interrupt routines can use the counter without interfering with the operation of the interrupted code. The sequencer provides instructions that permit arbitrary nesting of loops and subroutine calls. Instruction 9 (LOAD LOWER COUNTER; CONDITIONAL PUSH COUNTER) can be used to load the lower counter from the D port. If the condition is PASS, then the instruction also causes the old counter value to be pushed on the stack. To restore the counter from the stack, instruction 10 (POP TO LOWER COUNTER) can be used. Instructions 25 (LOAD UPPER COUNTER; CONDITIONAL PUSH COUNTER) and 26 (COND POP TO UPPER COUNTER/POP TO DISPLACEMENT WIDTH) are the counterparts for operating on the upper counter. Note that in cascaded systems, when the counter is pushed, regardless of whether instruction 25 or instruction 9 is executed, the entire counter is pushed to keep the stack balanced in the two Am29112s.

Stack and Register Instructions

In addition to all the instructions mentioned earlier that explicitly or implicitly alter the stack, the Am29112 has some specialized instructions for stack manipulation.

The stack on the Am29112 is 33 levels deep. Attempting to push when the stack is full will cause the top of the stack to be overwritten. Attempting to pop when the stack is empty will return undefined data. In either case STKERR will go HIGH and will persist until the sequencer is reset or the stack is popped in case of an overflow (or pushed in the case of an underflow).

The stack on the Am29112 can be loaded through the D port using instruction 1 (COND PUSH D/LOAD COMMAND REGISTER) with condition as forced PASS and unloaded out of the D port using instruction 2 (POP; COND STACK OUT TO D) with a forced PASS condition. In the stackout instruction the D port becomes an output port. Care must be taken to avoid contention on the D bus when this instruction is executed. The D bus is output enabled while CP is low for this instruction. The ability to load and unload the stack is useful for implementing context switches. For fast unloading of the stack, a tight two-instruction loop can be set up using instruction 2 (POP; COND STACKOUT TO D) with a Forced Pass condition and instruction 18 (COND TEST SP/BRANCH STACK NOT EMPTY) also with a Forced Fail condition. The branch instruction performs a branch to D if the stack is not empty.

The stack nesting level in an interruptible sequencer varies dynamically. Hence, the Am29112 is provided with instructions

for checking the available stack space: instruction 18 (COND TEST SP/BRANCH STACK NOT EMPTY). Two distinct instructions for testing the Stack Pointer have been packed into the same opcode and are differentiated by the condition code. A condition code of PASS selects the Test Stack Pointer instruction. In this instruction, the sequencer tests the stack to see if there is enough space, as determined by a constant input at the D port; if there is enough space, the sequencer performs a continue, whereas if there is not enough space, the sequencer performs a subroutine return (the stack is popped). The number of stack locations required is input at the D port. In a system with only one Am29112, the least significant 6 bits of the D are used within the chip for this instruction. In a system with two cascaded Am29112s the determination is made *independently* in the two chips (since the stack pointer is at all times identical in the two chips). Hence, the same number must be presented to the two chips. The adders in the two Am29112s are not cascaded for this instruction but function independently. In both Am29112s only the 6 LSBs of the D port are actually used in the comparison.

Interrupt Handling

The Am29112 recognizes two kinds of interrupts: maskable and unmaskable. Maskable interrupts cause automatic saving of the interrupt return address on the internal stack and can be inhibited, either externally via the INTERRUPT DISABLE pin, or internally via instruction 15 (COND ENABLE/DISABLE INTERRUPT). In addition, maskable interrupts are disabled when there is not enough space on the stack to service the interrupt, though this internal inhibit can be overridden by clearing a bit in the command register. The unmaskable interrupt, on the other hand, cannot be disabled and does not save the interrupt return address on the internal stack. It is intended for handling abnormal and irrecoverable situations like power failure or stack overflow. When an unmaskable interrupt occurs, the sequencer branches to the address of the unmaskable interrupt routine stored in the INTVECT register. This address is stored on-chip at system initialize time using instruction 11 (COND RESET SP/LOAD INTERRUPT REGISTER) with a condition of FAIL. If a maskable interrupt is being processed when the unmaskable interrupt occurs, the unmaskable interrupt may be delayed at most one cycle to prevent contention on the Y bus. In any case, the unmaskable interrupt request should persist for at least one clock edge.

The Am29112 contains an interrupt disable flip-flop on-chip. The flip-flop is set by the DISABLE INTERRUPT instruction (opcode 15 with forced FAIL) and reset by the ENABLE INTERRUPT instruction (opcode 15 with forced PASS). The flip-flop output performs the same function as the interrupt disable pin. On reset, or on receiving an unmaskable interrupt, the flip-flop is set thereby disabling maskable interrupts. Hence, at the end of initialization, the ENABLE INTERRUPT instruction will have to be executed to reset the flip-flop and enable maskable interrupts.

In the case of maskable interrupts, the interrupt return address is saved on the stack automatically using the INTRTN register. The INTRTN register is loaded with the CMUX output with every clock. When an interrupt is acknowledged, the Am29112 output is turned off and the vector applied externally. However, the sequencer executes the instruction which is in the pipeline register in that cycle. The result of executing the interrupted instruction, namely the next address, does not come out of the Am29112 Y bus because the Y bus is used to input the interrupt vector. It is clocked into the INTRTN register. On the first cycle of the interrupt routine, the sequencer pushes the return address on the stack so that the interrupt routine returns by doing a COND RETURN, like any other subroutine.

THE INVISIBLE STACK PUSH THAT THE SEQUENCER EXECUTES WHEN IT IS INTERRUPTED OCCURS IN THE FIRST CYCLE OF THE INTERRUPT SERVICE ROUTINE. HENCE, THE FIRST INSTRUCTION OF THE INTERRUPT SERVICE ROUTINE MAY NOT BE ANY INSTRUCTION THAT USES THE STACK.

Before acknowledging an interrupt, the sequencer checks the stack to see if there is a minimum of five levels available to handle the interrupt. If there is insufficient space on the stack, the acknowledge is not generated. This feature may be disabled by a bit in the command register.

Maskable Interrupts

The branch vector for maskable interrupts is applied externally to the Y port of the Am29112. This section discusses the system timing considerations and their impact on interrupt handling in the Am29112.

Figures 12-1 & 12-2 show general system configurations highlighting the interrupt portion of the circuitry and the control loop. A priority interrupt controller generates an interrupt request for the highest priority pending interrupt. This request is applied to the MINT \bar{R} pin of the Am29112. If the request is not masked, the Am29112 puts out an acknowledge on the MINT \bar{A} pin. The interrupt controller then puts out the encoded priority of the highest priority interrupt to the vector PROM, which maps the priority code into a vector.

The MINT \bar{A} line turns on the vector PROM output at the same time as the Y port on the Am29112 is three-stated. Hence, the interrupt vector gets onto the micromemory address bus and is also input into the Am29112, and incremented to form the next address. The Am29112 saves the return address on the stack so that when the interrupt service routine does a subroutine return, control returns to the instruction following the interrupted instruction.

The maskable interrupt request is synchronized on the Am29112. If there is no disable, therefore, the acknowledge always is active in the cycle following the request. However, the acknowledge to Y bus three-stating delay is programmable: the Y bus three-stating signal can occur either in the same cycle as, or in the cycle following, the MINT \bar{A} acknowledge, depending on a bit in the command latch of the Am29112.

The command register bit that programs the postdelay option is bit 2, the third least significant bit. The command register has 3 bits altogether and is loaded from the 3 LSBs of the D inputs using instruction 1 (COND PUSH D/LOAD COMMAND REGISTER) with a condition of FAIL. Note that in a system with two cascaded Am29112s, bits 0 and 2 of the command registers in the two chips must both be loaded with the same data on system initialization. The postdelay bit in the command register selects the postdelay option when it is zero.

Figure 12-1 shows the configuration without postdelay, including a simplified view of the acknowledge circuit. The acknowledge is granted at the same time the Y output of the Am29112 is three-stated and the vector PROM enabled by the MINT \bar{A} signal out of the Am29112. The critical delay path in this case is clock to acknowledge (Am29112) + acknowledge to priority out (interrupt controller) + vector PROM access time + microprogram memory access time + pipeline setup time. Obviously, this delay will have a significant impact on overall

cycle time. However, in slow systems or in systems where the vector is always available immediately with acknowledge, this configuration is acceptable. It is also acceptable if the vector mapping PROM is made part of the microprogram memory by dedicating the locations in low memory addressed by the priority to hold vectors to the corresponding interrupt routines. Care must be taken in disabling interrupts. Disabling of interrupts using the INTD input or the DISABLE INTERRUPTS instruction does not prevent an interrupt address from being put on the Y-bus by an external device in the current cycle. This interrupt address will be translated by the microprogram memory into the first instruction of an interrupt routine, which will probably begin by disabling interrupts and end by enabling interrupts. Therefore, when the interrupted microcode resumes control, interrupts will be enabled despite the fact that interrupts should have been disabled at this point.

This problem can be solved in the no-postdelay mode by letting the INTD signal from the microprogram memory bypass the microinstruction pipeline register (see Figure 12-1). The instruction lines, however, cannot bypass the microinstruction pipeline register. Therefore the DISABLE INTERRUPT instruction must be accompanied by an asserted INTD bit in the same microinstruction. Since it is normally known at assembly time whether individual microinstructions can be interrupted, the INTD bit can be set at that time.

Figure 12-2 shows a simplified view of the Am29112 configured with postdelay active. An external D-type flip-flop adds a one cycle delay to the MINT \bar{A} signal before it switches the output enable on the vector register. The interrupt request to acknowledge delay is the same as in the circuit with postdelay inactive, but the Y bus three-stating signal occurs one cycle later than the acknowledge. The critical path has been broken into two with the register at the vector PROM output. In this case the critical delay path is cut short by the microprogram memory access time. While the vector PROM accesses the interrupt vector, the microprogram memory accesses the next sequential instruction. This implies that one more instruction of the interrupted code executes after the cycle in which the acknowledge is granted.

The Command Register bits are summarized below:

CR $_0$: Interrupt acknowledge on insufficient space

CR $_0$ = 1 : inhibit acknowledge on insufficient space
CR $_0$ = 0 : generate acknowledge on insufficient space

CR $_1$: Multiway enable

CR $_1$ = 1 : enable multiway branching (default for LSS)
CR $_1$ = 0 : disable multiway branching (default for MSS)

CR $_2$: Interrupt postdelay flip-flop

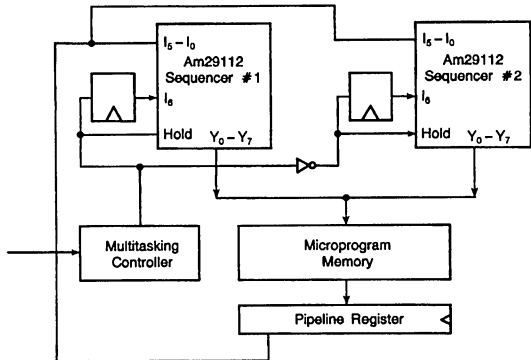
CR $_2$ = 1 : no postdelay (default)
CR $_2$ = 0 : postdelay

On reset & JZ.U: CR $_0$ = 1
CR $_1$ = LSS
CR $_2$ = 1

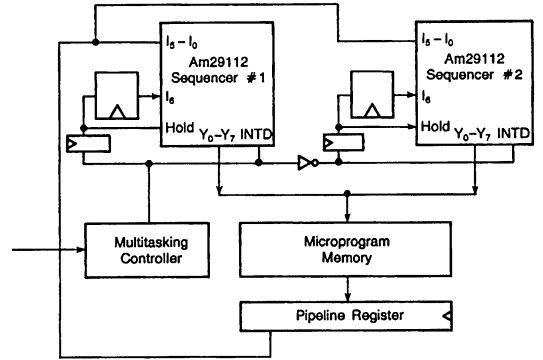
Hold

The Am29112 is equipped with a HOLD pin for configurations utilizing more than one sequencer driving a common microprogram address bus (see Figures 8 & 9). In such situations, it is necessary to cause the unselected sequencer to hold its internal state while some other sequencer executes, so that it can resume execution at the point where it was held. The HOLD pin, when asserted, three-states the Y bus, forces low the carry into the PC incrementer, and selects the internal

CMUX output (instead of the Y bus) at the incrementer input. To complete the HOLD function, it is also necessary to disable interrupts and to put the sequencer into the forced continue mode (see Figures 10 & 11). Under these conditions, the value of the PC is recirculated through the CMUX and the incrementer until the HOLD is released, and all the remaining state bits in the sequencer are not altered because of the forced continue. HOLD does not disable $\overline{\text{UINTR}}$; all of the sequencers will accept it and put out the $\overline{\text{UINTR}}$ address when they leave the Hold mode.



LD000330

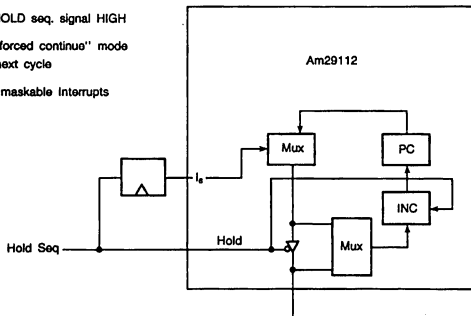


LD000290

Figure 8. Multitasking System Without Interrupts

Figure 9. Multitasking System With Interrupts

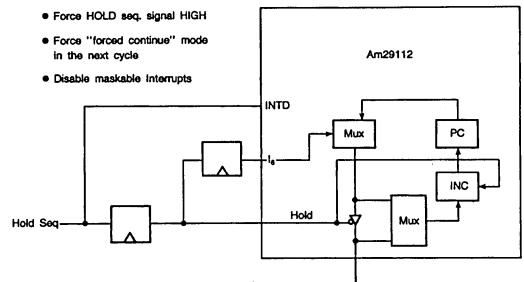
- Force HOLD seq. signal HIGH
- Force "forced continue" mode in the next cycle
- Disable maskable interrupts



LD000310

Figure 10. Hold Control Without Interrupts

- Force HOLD seq. signal HIGH
- Force "forced continue" mode in the next cycle
- Disable maskable interrupts



LD000300

Figure 11. Hold Control With Interrupts

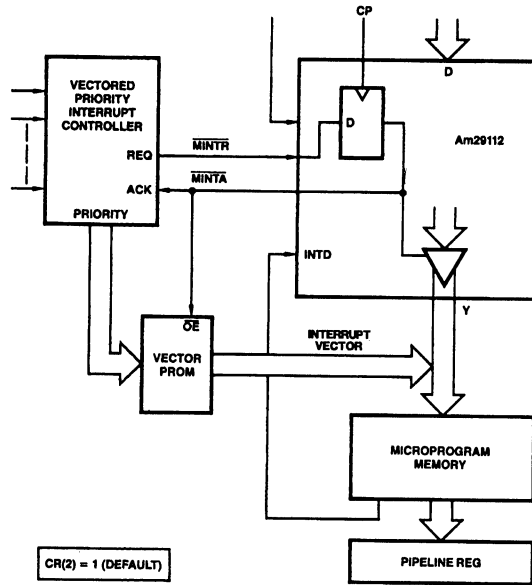


Figure 12-1. Interrupt Control Loop — No Postdelay

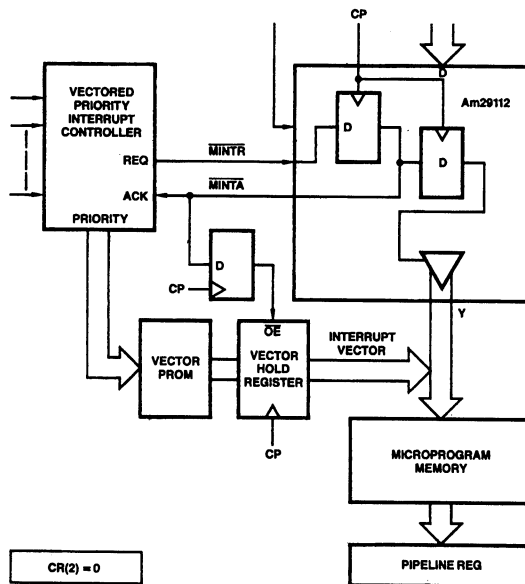


Figure 12-2. Interrupt Control Loop with Postdelay

ABSOLUTE MAXIMUM RATINGS

Storage Temperature -65 to +150°C
 Case Temperature Under Bias (T_C) -55 to +125°C
 Supply Voltage to Ground Potential -0.5 V to +7.0 V
 DC Voltage Applied to Outputs For
 High Output State -0.5 V to + V_{CC} Max.
 DC Input Voltage -0.5 V to +5.5 V
 DC Output Current, Into Outputs 30 mA
 DC Input Current -30 mA to +5.0 mA

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

OPERATING RANGES

Commercial (C) Devices
 Temperature (T_A) 0 to +70°C
 Supply Voltage (V_{CC}) +4.75 V to +5.25 V

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

DC CHARACTERISTICS over operating range unless otherwise specified

Parameter Symbol	Parameter Description	Test Conditions (Note 1)			Min.	Max.	Units
V_{OH}	Output HIGH Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IH}$ or V_{IL}		$I_{OH} = -1.6 \text{ mA}$	2.4		V
V_{OL}	Output LOW Voltage	$V_{CC} = \text{Min.}$ $V_{IN} = V_{IH}$ or V_{IL}	$Y_0 - Y_7$	$I_{OL} = 12 \text{ mA}$		0.5	V
			All Others	$I_{OL} = 8 \text{ mA}$			
V_{IH}	Guaranteed Input Logical HIGH Voltage (Note 5)		All Inputs		2.0		V
V_{IL}	Guaranteed Input Logical LOW Voltage (Note 5)		All Inputs			0.8	V
V_I	Input Clamp Voltage	$V_{CC} = \text{Min.}$	All Inputs	$I_{IN} = -18 \text{ mA}$		-1.5	V
I_{IL}	Input LOW Current	$V_{CC} = \text{Max.}$ $V_{IN} = 0.5 \text{ Volts}$ (Note 3)	All I/O Pins	$V_{IN} = 0.5 \text{ V}$		-0.55	mA
			Hold, LSS		-1.0		
			All Others		-0.5		
I_{IH}	Input HIGH Current	$V_{CC} = \text{Max.}$ $V_{IN} = 2.4 \text{ Volts}$ (Note 3)	All I/O Pins	$V_{IN} = 2.4 \text{ V}$		100	μA
			Hold, LSS		100		
			All Others		50		
I_I	Input HIGH Current	$V_{CC} = \text{Max.}$ $V_{IN} = 5.5 \text{ Volts}$	All Inputs				mA
I_{OZH}	Off State (High Impedance) Output Current	$V_{CC} = \text{Max.}$ $V_O = 2.4 \text{ Volts}$ (Note 3)		$V_{IN} = 2.4 \text{ V}$		100	μA
I_{OZL}	Off State (High Impedance) Output Current	$V_{CC} = \text{Max.}$ $V_O = 0.5 \text{ Volts}$ (Note 4)		$V_{IN} = 0.5 \text{ V}$		-0.55	μA
I_{OS}	Output Short Circuit Current	$V_{CC} = \text{Max.} + 0.5 \text{ Volts}$ $V_O = 0.5 \text{ Volts}$ (Note 2)	All Except CZIO			-50	mA
I_{CC}	Power Supply Current (Note 4)	$V_{CC} = \text{Max.}$	COM'L	$T_{AL} = 0 \text{ to } 70^\circ\text{C}$ (Note 6)			mA
				$T_A = 70^\circ\text{C}$		545	

- Notes: 1. For conditions shown as Min. or Max., use the appropriate value specified under Operating Ranges for the applicable device type.
 2. Not more than one output should be shorted at a time. Duration of the short circuit test should not exceed one second.
 3. Three-state outputs are internally connected to TTL inputs. Input characteristics are measured under conditions such that the outputs are in the OFF state.
 4. Worst case I_{CC} is at minimum temperature.
 5. These input levels provide zero noise immunity and should be tested only in a static, noise-free environment.
 6. Cold start.

Am29112 SWITCHING CHARACTERISTICS
GUARANTEED CHARACTERISTICS OVER COMMERCIAL OPERATING RANGE
 (T_A = 0 to +70°C, V_{CC} = 4.75 to 5.25 V, C_L = 50 pF)



A. Setup and Hold Times (ns)

Input	t _s	t _h
D ₀ - D ₇	34	2
M ₀ - M ₇	33	0
Y ₀ - Y ₇	14	0
I ₀ - I ₆	45	0
I ₅	45	0
I ₆	45	0
\overline{CC}	26	0
\overline{CCEN}	26	0
POL	26	0
RESET	7	5
UINTR	8	5
MINTR	7	5
INTD	8	5
HOLD	—	—
LSS	—	—
ACIO	19	0
PCIO	10	2
CIO	20	0
CZIO	7	5

B. Combinational Delays (ns)

Input	Output							
	D	Y	MINTA	STKERR	ACIO	PCIO	CIO	CZIO
D ₀ - D ₇	—	38	—	—	24	37	—	25
M ₀ - M ₃	—	38	—	—	34	41	—	—
I ₀ - I ₆	—	50	34	37	43	40	39	38
I ₅	—	50	—	—	—	40	31	40
I ₆	—	50	—	—	—	—	40	38
\overline{CC}	—	29	—	37	—	40	40	37
\overline{CCEN}	—	29	—	37	—	40	40	37
POL	—	29	—	37	—	40	40	37
CP	23	35	23	33	33	41	26	33
HOLD	—	—	—	—	—	23	—	—
LSS	—	—	—	—	—	—	—	24
ACIO	—	25	—	—	—	31	—	—
CIO	—	—	—	—	—	—	—	37
Y ₀ - Y ₇	—	—	—	—	—	25	—	—

C. Enable/Disable Times (ns)

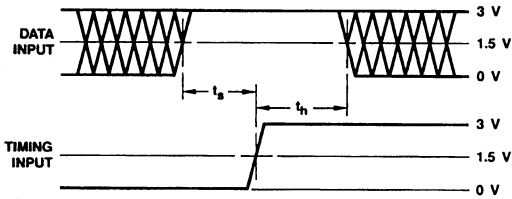
Input	Output				
	D	Y	PCIO	ACIO	CZIO
I ₀ - I ₆	36	—	—	—	—
I ₆	40	—	—	—	—
\overline{CC}	36	—	—	—	—
\overline{CCEN}	36	—	—	—	—
POL	36	—	—	—	—
CP 	—	34	—	—	—
CP 	43	—	—	—	—
HOLD	—	28	—	—	—
LSS	—	—	25	25	25

D. Clock Requirements (ns)

Minimum Clock LOW Time	
Minimum Clock HIGH Time	
Minimum Clock Period	

SWITCHING TEST WAVEFORMS

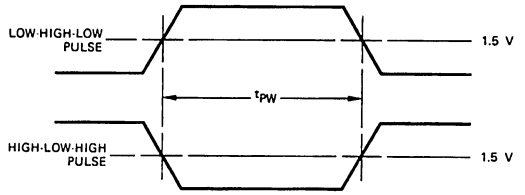
SET-UP, HOLD, AND RELEASE TIMES



WFR02970

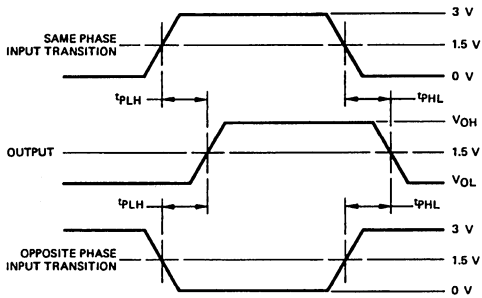
- Notes: 1. Diagram shown for HIGH data only.
Output transition may be opposite sense.
2. Cross hatched area is don't care condition.

PULSE WIDTH



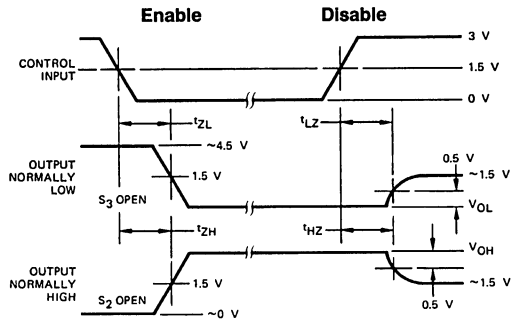
WFR02790

PROPAGATION DELAY



WFR02980

ENABLE AND DISABLE TIMES



WFR02660

- Notes: 1. Diagram shown for Input Control Enable-LOW and Input Control Disable-HIGH.
2. S₁, S₂ and S₃ of Load Circuit are closed except where shown.

Notes on Test Methods

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

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

Capacitive Loading for A.C. Testing

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

with a bench set up are used to predict the result at the lower capacitance.

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

Threshold Testing






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

A.C. Testing

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

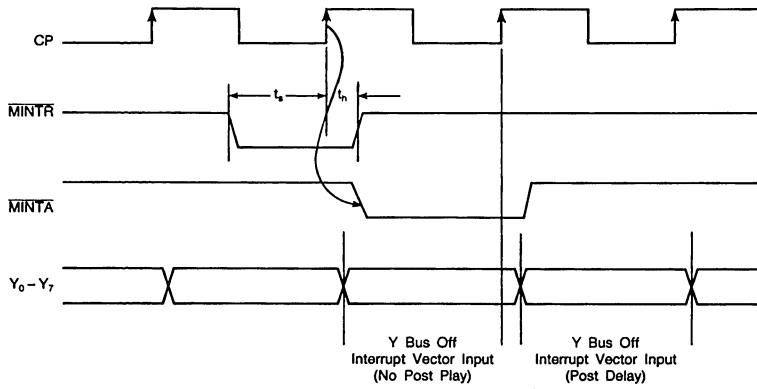
In some cases, certain A.C. tests are redundant since they can be shown to be predicted by other tests which have already been performed. In these cases, the redundant tests are not performed.

KEY TO SWITCHING WAVEFORMS

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

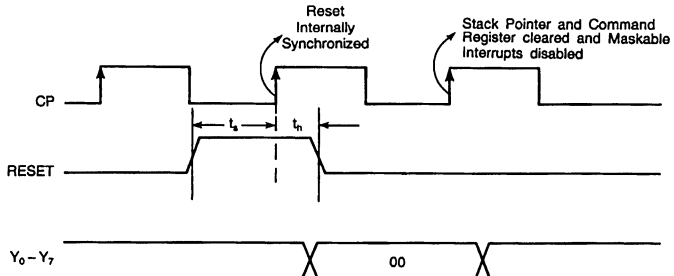
KS000010

SWITCHING WAVEFORMS



WF022320

Maskable Interrupt

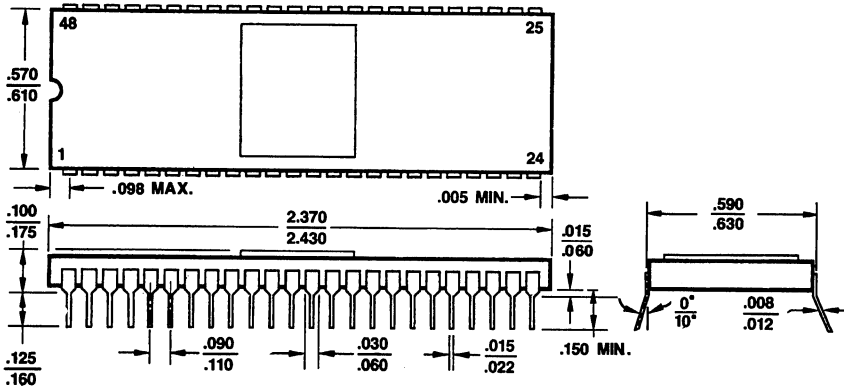


WF022330

RESET

PHYSICAL DIMENSIONS

SD 048



PID # 07546A

The International Standard of
Quality guarantees a 0.05% AQL on all
electrical parameters, AC and DC,
over the entire operating range.

INT·STD·500

ADVANCED MICRO DEVICES DOMESTIC SALES OFFICES

ALABAMA	(205) 882-9122	MARYLAND	(301) 796-9310
ARIZONA,		MASSACHUSETTS	(617) 273-3970
Tempe	(602) 242-4400	MINNESOTA	(612) 938-0001
Tucson	(602) 792-1200	NEW JERSEY	(201) 299-0002
CALIFORNIA,		NEW YORK,	
El Segundo	(213) 640-3210	Liverpool	(315) 457-5400
Newport Beach	(714) 752-6262	Poughkeepsie (IBM only)	(914) 471-8180
San Diego	(619) 560-7030	Woodbury	(516) 364-8020
Sunnyvale	(408) 720-8811	NORTH CAROLINA,	(919) 847-8471
Woodland Hills	(818) 992-4155	OREGON	(503) 245-0080
COLORADO	(303) 741-2900	OHIO,	
CONNECTICUT,	(203) 264-7800	Columbus	(614) 891-6455
FLORIDA,		PENNSYLVANIA,	
Altamonte Springs	(305) 339-5022	Allentown (AT&T only)	(215) 398-8006
Clearwater	(813) 530-9971	Willow Grove	(215) 657-3101
Ft. Lauderdale	(305) 484-8600	PUERTO RICO	(809) 764-4524
Melbourne	(305) 729-0496	TEXAS,	
GEORGIA	(404) 449-7920	Austin	(512) 346-7830
ILLINOIS	(312) 773-4422	Dallas	(214) 934-9099
INDIANA	(317) 244-7207	Houston	(713) 785-9001
KANSAS	(913) 451-3115	WASHINGTON	(206) 455-3600
		WISCONSIN	(414) 782-7748


INTERNATIONAL SALES OFFICES

BELGIUM,		HONG KONG,	
Bruxelles	TEL:	Kowloon	TEL:
	(02) 771 99 93		3-695377
	FAX:		FAX:
	(02) 762-3716		1234276
	TLX:		TLX:
61028	50426
CANADA, Ontario,		ITALY, Milano	TEL:
Kanata	TEL:		(02) 3390541
Willowdale	TEL:		FAX:
	(416) 224-5193		(02) 3498000
	FAX:		TLX:
	(416) 224-0056	315286
FRANCE,		JAPAN, Tokyo	TEL:
Paris	TEL:		(03) 345-8241
	(01) 45 60 00 55		FAX:
	FAX:		3425196
	(01) 46 86 21 85		TLX:
	TLX:		J24064 AMDTKOJ
202053F	LATIN AMERICA,	
GERMANY,		Ft. Lauderdale,	TEL:
Hannover area	TEL:		(305) 484-8600
	(05143) 50 55		FAX:
	FAX:		(305) 485-9736
	(05143) 55 53		TLX:
925287		5109554261 AMDFTL
	TLX:	SWEDEN, Stockholm	TEL:
	(089) 41 14-0		(08) 733 03 50
	FAX:		FAX:
	(089) 406490		(08) 733 22 85
	TLX:		TLX:
523883	11602
	TEL:	UNITED KINGDOM,	
	(0711) 62 33 77	Manchester area	TEL:
	FAX:		(0925) 828008
	(0711) 625187		FAX:
	TLX:		(0925) 827693
721882		TLX:
		628524
		London area	TEL:
			(04862) 22121
			FAX:
			(04862) 22179
			TLX:
		859103

NORTH AMERICAN REPRESENTATIVES

CALIFORNIA		NEW MEXICO	
I ² INC	OEM (408) 988-3400	THORSON DESERT STATES	(505) 293-8555
	DISTI (408) 496-6868	NEW YORK	
IDAHO		NYCOM, INC	(315) 437-8343
INTERMOUNTAIN TECH MKGT	(208) 888-6071	OHIO	
INDIANA		Dayton	
SAI MARKETING CORP	(317) 241-9276	DOLFUSS ROOT & CO	(513) 433-6776
IOWA		Strongsville	
LORENZ SALES	(319) 377-4666	DOLFUSS ROOT & CO	(216) 238-0300
MICHIGAN		PENNSYLVANIA	
SAI MARKETING CORP	(313) 227-1786	DOLFUSS ROOT & CO	(412) 221-4420
NEBRASKA		UTAH	
LORENZ SALES	(402) 475-4660	R ² MARKETING	(801) 595-0631

Advanced Micro Devices reserves the right to make changes in its product without notice in order to improve design or performance characteristics. The performance characteristics listed in this document are guaranteed by specific tests, correlated testing, guard banding, design and other practices common to the industry. For specific testing details, contact your local AMD sales representative. The company assumes no responsibility for the use of any circuits described herein.

 **ADVANCED MICRO DEVICES** 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088, USA
TEL: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450

© 1986 Advanced Micro Devices, Inc.
Printed in U.S.A. AIS-WCP-15M-07/86-0