Am29112

A High-Performance 8-Bit Slice Microprogram Sequencer

ADVANCED INFORMATION

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 ALU's and other processing components.

The Am29112 is designed to operate in 10MHz 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.

APPLICATION NOTES REFERENCE

- Microprogrammed CPU using Am29116
- An intelligent fast disk controller
- Am29116 architecture speeds pixel manipulation in interactive bit-mapped graphics

BLOCK DIAGRAM

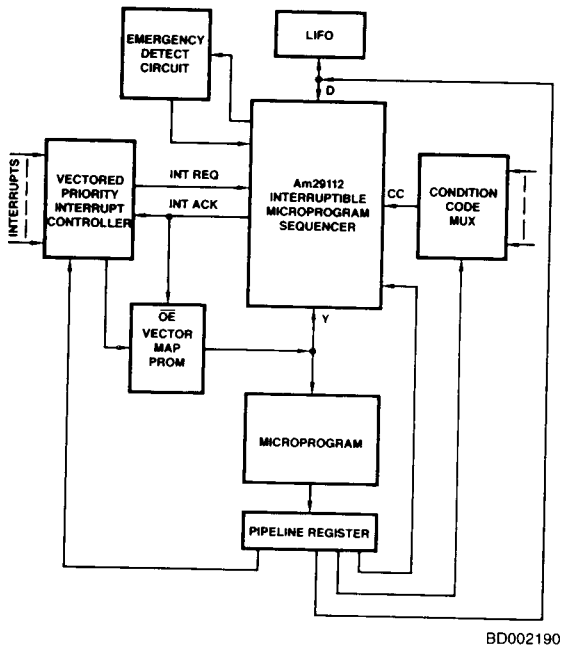


Figure 1. Am29112 in a Single Pipelined System.

RELATED PRODUCTS

Part No.	Description
Am29116	A 16-Bit Bipolar Microprocessor
Am2904	Status and Shift Control Unit
Am2910A	Microprogram Controller
Am29114	Vectored Priority Interrupt Controller
Am2925	System Clock Generator and Driver
Am2940	DMA Address Generator
Am2942	Programmable Timer/Counter/DMA
Am2950/ 51/52/53	8-Bit Bidirectional I/O Port
Am29118	8-Bit Bidirectional I/O Port/Accumulator

PIN DESCRIPTION

Pin No.	Name	I/O	Description
	D_0-D_7	I/O	Bidirectional data input for direct input to address multiplexer, counter and other control registers and stack output.
	Y_0-Y_7	I/O	Bidirectional microprogram address bus outputs microprogram address and inputs interrupt vector.
	M_0-M_3	I	Multiway input pins for up to 16-way branches.
	HOLD	I	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.
	$\overline{CC}$	I	Test input for the sequencer. (See Table 2.)
	CCEN	I	Test enable for the sequencer. (See Table 2.)
	POL	I	Polarity input for test. (See Table 2.)
	I_0-I_4	I	Instruction input.
	I_5-I_6	I	Mode control input. Select one of three modes: normal, extended or forced continue. (See Table 1.)
	STKERR	O	Indicates stack overflow or underflow.
	$\overline{UINTR}$	I	Unmaskable interrupt request input.
	$\overline{MINTR}$	I	Maskable interrupt request input.
	INTD	I	Disable for maskable interrupts.
	$\overline{MINTA}$	O	Maskable interrupt acknowledge.
	LSS	I	Programs the least significant chip when high, most significant chip when low.
	RST	I	Reset input. Selects zero as the next microprogram address, resets the stack pointer and interrupt logic, and disables maskable interrupts.
	CP	I	Clock input.
	ACIO	I/O	Bidirectional adder I/O line for cascaded Am29112s.
	PCIO	I/O	Bidirectional program counter I/O line for cascaded Am29112s.
	CIO	I/O	Bidirectional counter I/O line for cascaded Am29112s.
	CZIO	I/O	Bidirectional counter zero I/O line for cascaded Am29112s.

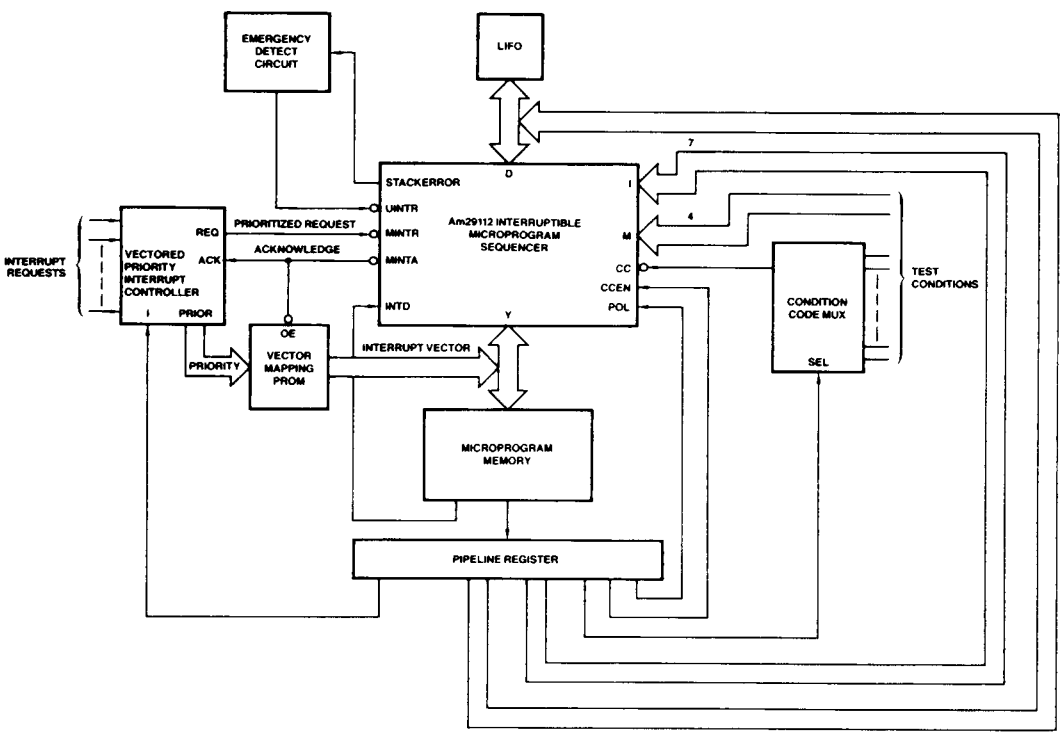
PRODUCT OVERVIEW

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

Branch addresses, constants for the various registers, and stack pointer values 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 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 acknowledges the request via the $\overline{MINTA}$ pin. Upon receiving the acknowledge, the priority interrupt control 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 turns off the Y port, enabling the interrupt vector onto the microprogram address bus. In the Am29112, internal states are 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.



BD001921

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 3. 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 power-up 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 and the stack pointer, 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, carry ripples from the LSS adder to the MSS adder over the CIO 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.

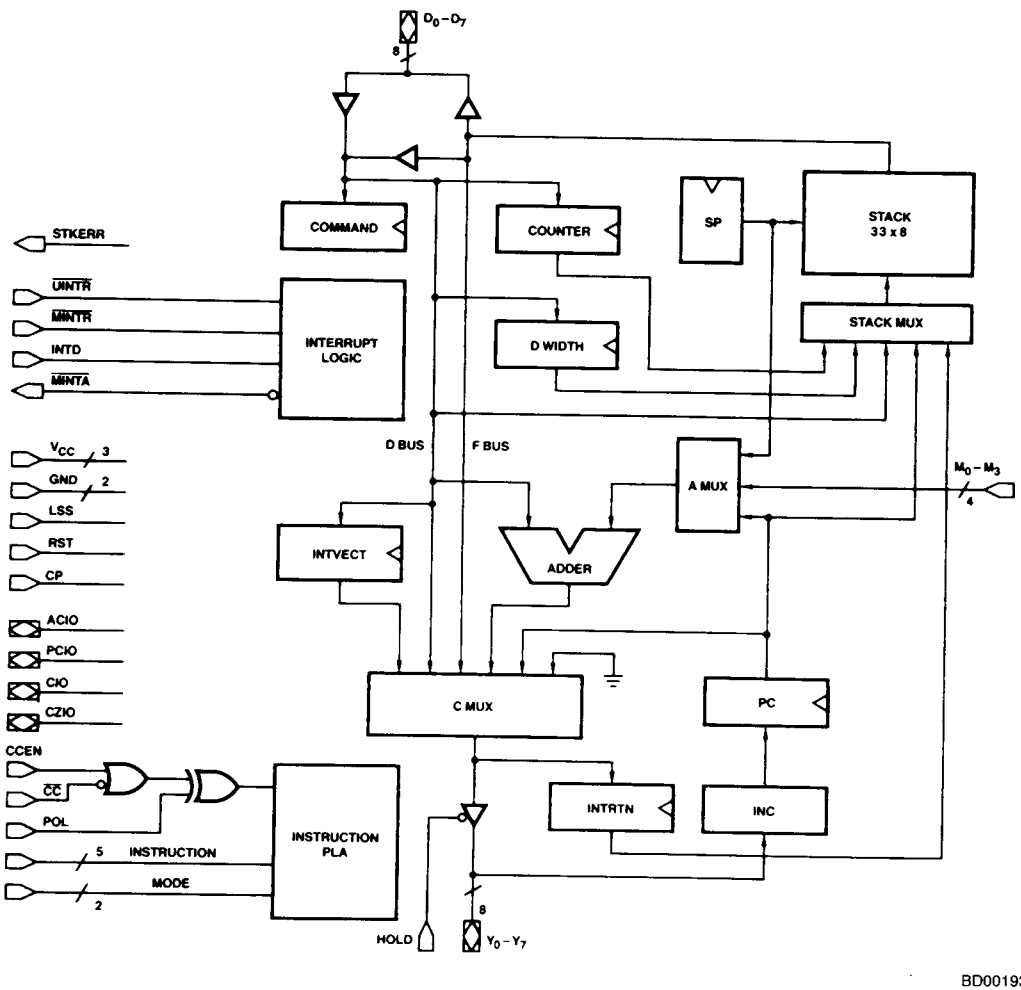


Figure 3. Am29112 48-Pin Package.

INSTRUCTION SET OF THE Am29112

MODE BITS ($I_{6,5}$)

The Am29112 is controlled by 5 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

$I_{6,5}$	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.

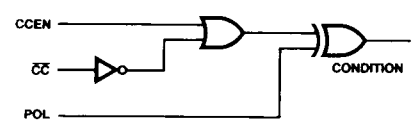
FORCED CONTINUE MODE

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 writeable control store 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 40 different instructions. In order to encode these instructions with only 5 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).



PF001060

Figure 4. Condition Code Circuit.

TABLE 2. CONDITION CODE TABLE

CCEN	$\overline{CC}$	POL	Condition
0	0	0	PASS
0	1	0	FAIL
0	0	1	FAIL
0	1	1	PASS
1	0	0	PASS
1	1	0	PASS
1	0	1	FAIL
1	1	1	FAIL

Am29112 Instruction Set

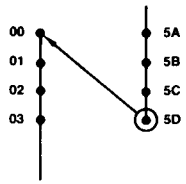
Opcode (I_{40})	Condition	Mnemonic	Description
0	X	JZ.U	UNCONDITIONAL JUMP ZERO
1	PASS	PUSHD.P	PUSH D (PASS)
1	FAIL	LDCMD.F	LOAD COMMAND REGISTER FROM D (FAIL)
2	COND	POP.C	POP; CONDITIONAL STACKOUT TO D
3	COND	CJD.C	CONDITIONAL JUMP D
4	COND	CJSD.C	CONDITIONAL JUMP SUBROUTINE D
5	COND	CJMW.C	CONDITIONAL JUMP MULTIWAY D
6	COND	CJSMW.C	CONDITIONAL JUMP SUBROUTINE MULTIWAY D
7	COND	CRTN.C	CONDITIONAL RETURN
8	COND	PUSHPL.C	PUSH PC; COND LOAD LOWER COUNTER
9	COND	LDLC.C	LOAD LOWER COUNTER; COND PUSH COUNTER
10	X	POPLC.U	POP TO LOWER COUNTER
11	PASS	RSTSP.P	RESET STACK POINTER (PASS)
11	FAIL	LDINTV.F	LOAD UNMASKABLE INTERRUPT VECTOR (FAIL)
12*	PASS	RFCTU.P	REPEAT LOOP, UPPER COUNTER = 0 (PASS)
12*	FAIL	RFCTL.F	REPEAT LOOP, LOWER COUNTER = 0 (FAIL)
13**	PASS	RPCTU.P	REPEAT PIPELINE, UPPER COUNTER = 0 (PASS)
13**	FAIL	RPCTL.F	REPEAT PIPELINE, LOWER COUNTER = 0 (FAIL)
14	COND	LOOP.C	TEST END LOOP
15	PASS	ENINT.P	ENABLE INTERRUPTS (PASS)
15	FAIL	DISINT.F	DISABLE INTERRUPTS (FAIL)
16***	COND	TWBL.C	THREE-WAY BRANCH, LOWER COUNTER
17***	COND	TWBU.C	THREE-WAY BRANCH, UPPER COUNTER
18	PASS	TSTSP.P	TEST SP WITH D (PASS)
18	FAIL	TSTMT.F	JUMP D IF STACK NOT EMPTY
19	COND	CJDF.C	COND JUMP D/STACK AND POP
20	COND	CJSDF.C	COND JUMP SUBROUTINE D/STACK AND POP
21	COND	CJMWR.C	COND JUMP MULTIWAY RELATIVE D
22	COND	CJSMWR.C	COND JUMP SUBROUTINE MULTIWAY RELATIVE D
23	COND	CJPP.C	COND JUMP PIPELINE AND POP
24	COND	PUSHPU.C	PUSH PC; COND LOAD UPPER COUNTER
25	COND	LDUC.C	LOAD UPPER COUNTER; COND PUSH COUNTER
26	PASS	POPUC.P	POP TO UPPER COUNTER (PASS)
26	FAIL	POPDW.F	POP TO DISPLACEMENT WIDTH (FAIL)
27	COND	LDDW.C	LOAD DISPLACEMENT WIDTH; COND PUSH DW
28	COND	CJR.C	COND JUMP D PC REL
29	COND	CJRN.C	COND JUMP D PC REL NEGATIVE
30	COND	CJSR.C	COND JUMP SUBROUTINE D PC REL
31	COND	CJSRN.C	COND JUMP SUBROUTINE D PC REL NEGATIVE

* These instructions are identical in the extended mode.
**These too.
***These too.

Extensions: U – unconditional; C – conditional; P – PASS condition; F – FAIL condition.
Note: PASS/FAIL condition can be produced as follows. P stands for polarity and I for input:

$\overline{CC}$	CCEN	POL	Condition
X	1	0	PASS
X	1	1	FAIL
I	0	P	COND

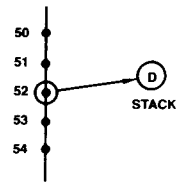
0 Jump Zero (JZ.U)



PF000600

UNCONDITIONAL

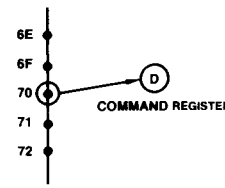
1 Push D (PUSHD.P)



PF000620

FORCED PASS

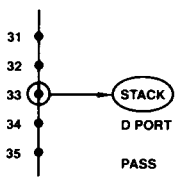
1 Load Command Register from D (LDCMD.F)



PF000611

FORCED FAIL

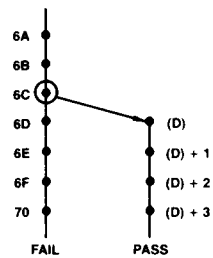
2 Pop and Conditional Stack-out to D (POP.C)



PF000571

CONDITIONAL

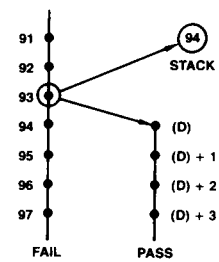
3 Conditional Jump D (CJD.C)



PF000580

CONDITIONAL

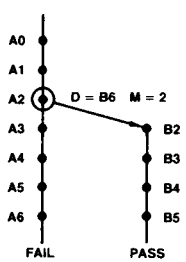
4 Conditional Jump Subroutine D (CJSD.C)



PF000590

CONDITIONAL

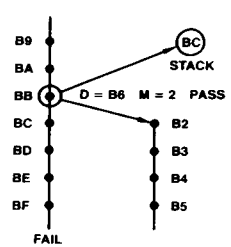
5 Conditional Jump Multiway D (CJMW.C)



PF000560

CONDITIONAL

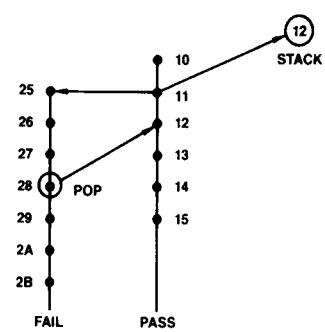
6 Conditional Jump Subroutine Multiway D (CJSMW.C)



PF000550

CONDITIONAL

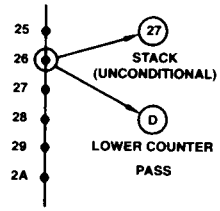
7 Conditional Return (CRTN.C)



PF000540

CONDITIONAL

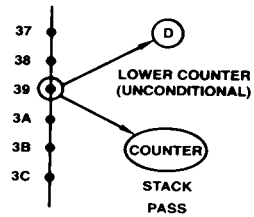
8 Push PC and Conditional Load Lower Counter (PUSHPL.C)



PF000511

CONDITIONAL

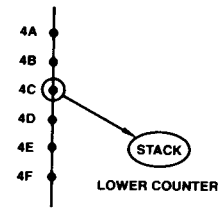
9 Load Lower Counter and Conditional Push Counter (LDLC.C)



PF000521

CONDITIONAL

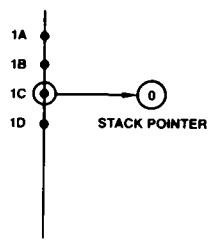
10 Pop to Lower Counter (POPLC.U)



PF000531

UNCONDITIONAL

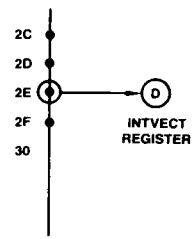
11 Reset Stack Pointer (RSTSP.P)



PF000460

FORCED PASS

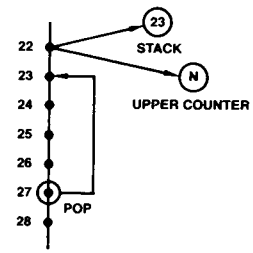
11 Load Unmaskable Interrupt Vector (LDINTV.F)



PF000470

FORCED FAIL

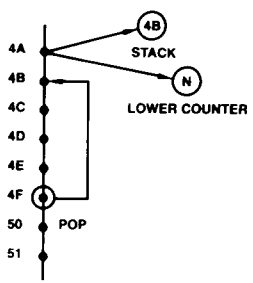
12 Repeat Loop, Upper Counter (RFCTU.P)



PF000790

FORCED PASS

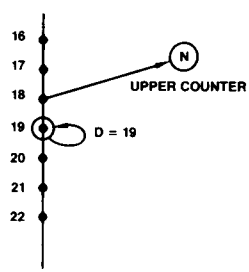
12 Repeat Loop, Lower Counter (RFCTL.F)



PF000440

FORCED FAIL

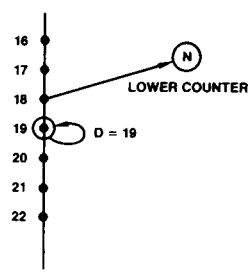
13 Repeat Pipeline, Upper Counter (RPCTU.P)



PF000451

FORCED PASS

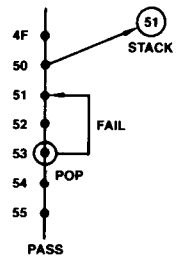
13 Repeat Pipeline, Lower Counter (RPCTL.F)



PF000781

FORCED FAIL

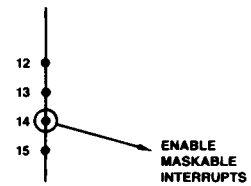
14 Test End Loop (LOOP.C)



PF000421

CONDITIONAL

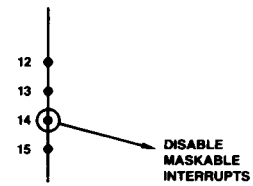
15 Enable Interrupts (ENINT.P)



PF000430

FORCED PASS

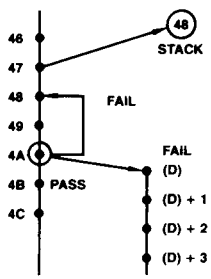
15 Disable Interrupts (DISINT.F)



PF000390

FORCED FAIL

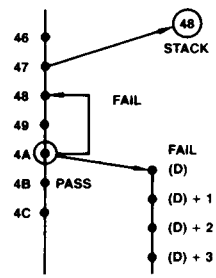
16 Three-Way Branch, Lower Counter (TWBL.C)



PF000411

CONDITIONAL

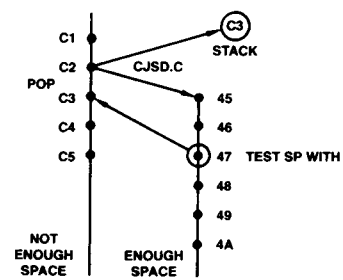
17 Three-Way Branch, Upper Counter (TWBU.C)



PF000411

CONDITIONAL

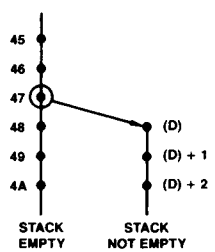
18 Test SP with D (TSTSP.P)



PF000400

FORCED PASS

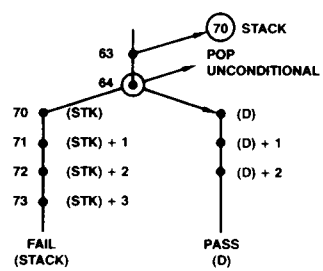
18 Jump D if Stack Not Empty (TSTMT.F)



PF000660

FORCED FAIL

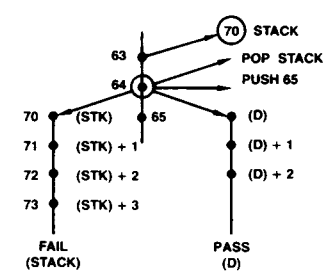
19 Conditional Jump D/Stack and Pop (CJDF.C)



PF000691

CONDITIONAL

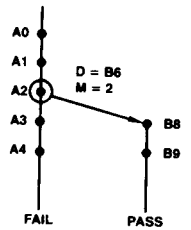
20 Conditional Jump Subroutine D/Stack and Pop (CJSDF.C)



PF000681

CONDITIONAL

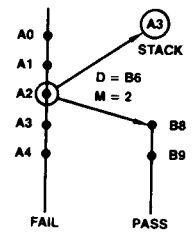
21 Conditional Jump Multiway Relative D (CJMWR.C)



PF000670

CONDITIONAL

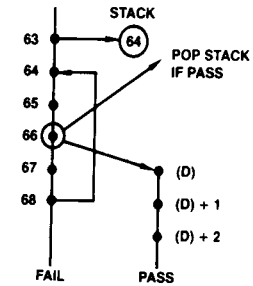
22 Conditional Jump Subroutine Multiway Relative D (CJSMWR.C)



PF000700

CONDITIONAL

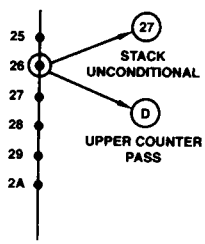
23 Conditional Jump Pipeline and Pop (CJPP.C)



PF000760

CONDITIONAL

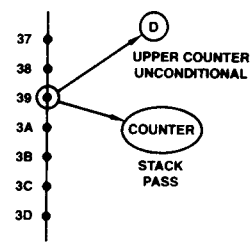
24 Push PC and Conditional Load Upper Counter (PUSHPU.C)



PF000730

CONDITIONAL

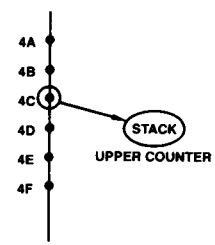
25 Load Upper Counter and Conditional Push Counter (LDUC.C)



PF000710

CONDITIONAL

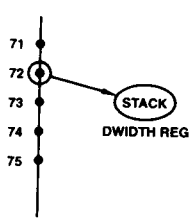
26 Pop to Upper Counter (POPUC.P)



PF000770

FORCED PASS

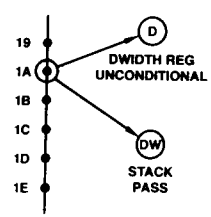
26 Pop to Displacement Width (POPDW.F)



PF000720

FORCED FAIL

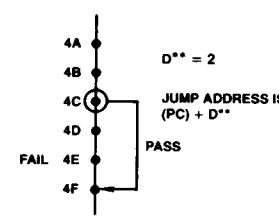
27 Load Displacement Width and Conditional Push DW (LDDW.C)



PF000740

CONDITIONAL

28 Conditional Jump D PC Relative (CJR.C)

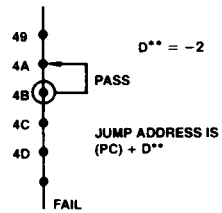


PF000750

D** is displacement (see Note 1).

CONDITIONAL

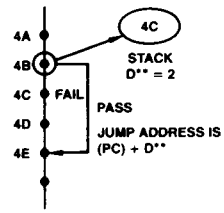
29 Conditional Jump D PC Relative Negative (CJRN.C)



PF000490

D** = −2, should be two's complement (see Note 2).
CONDITIONAL

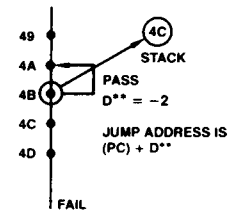
30 Conditional Jump Subroutine D PC Relative (CJSR.C)



PF000480

D**is displacement (see Note 1).
CONDITIONAL

31 Conditional Jump Subroutine D PC Relative Negative (CJSRN.C)



PF000500

D** = −2, should be two's complement (see Note 2).
CONDITIONAL

Notes: 1. The number of bits of D used as displacement is stored in DWIDTH register. The remaining high order bits are zero-extended.
2. The number of bits of D used as displacement is stored in DWIDTH register. The remaining high order bits are one-extended.

BRANCHING INSTRUCTIONS

Direct Branching

Instruction 0 is the unconditional jump to zero instruction. This instruction also resets the stack pointer and the interrupt logic as well as setting command register as follows: CR(0) = 1, CR(1) = LSS, CR(2) = 1.

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 (DWIDTH) 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.

Multiway Branching

Two variants of multiway branching are available on the Am29112 – multiway substitute D and multiway relative D. 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.

Multiway 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.

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 LOOP EXIT) 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). 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 (CONDITIONAL REPEAT LOOP). Instruction 8 (PUSH PC; COND LOAD 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 14 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 or 25 (LOAD COUNTER AND 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, which is its own address, 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 'xecuted, 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 with a forced FAIL condition. 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 10 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 deep. Attempting to push when the stack is full or to pop when the stack is empty causes the STACK ERROR signal out of the Am29112 to be generated. The error is latched internally and persists until either the chip is reset or the stack is popped in case of overflow or pushed in case of underflow. When the stack overflows, the stack pointer does not wrap around, and all subsequent pushes on the full stack write over the top-of-stack location.

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 STACKOUT 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 12 (POP; COND STACKOUT TO D) with a forced FAIL 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 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 status 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 be clearing a bit in the command register. The unmaskable interrupt, on the other hand, cannot be disabled and does not cause saving of status 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 locations 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.

CR(0) = 1 INHIBIT ACKNOWLEDGE ON STACK FULL (DEFAULT)

CR(0) = 0 GENERATE ACKNOWLEDGE ON STACK FULL

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.

Figure 5(a) shows a general system configuration 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 $\overline{\text{MINTR}}$ pin of the Am29112. If the request is not masked, the Am29112 puts out an acknowledge on the $\overline{\text{MINTA}}$ 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 $\overline{\text{MINTA}}$ 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 $\overline{\text{MINTA}}$ 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 PASS. Note that in a system with two cascaded Am29112s, the 0 and 2 bits 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 5(b) 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 MINTA 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.

Figure 5(c) shows a simplified view of the Am29112 configured with postdelay active. An external D-type flip-flop adds a one cycle delay to the $\overline{\text{MINTA}}$ 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. (If that instruction happens to be a DISABLE INTERRUPT instruction, then even though no more interrupts will be accepted by the Am29112, the interrupt which has been acknowledged goes through and the corresponding interrupt service routine may enable interrupts again using the ENABLE INTERRUPT instruction.)

The command register bits are summarized below:

CR(0) : Interrupt acknowledge on stack full

- CR(0) = 1 : inhibit acknowledge on stack full (default)
- CR(0) = 0 : generate acknowledge on stack full

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 & JZU: 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. 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. 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.

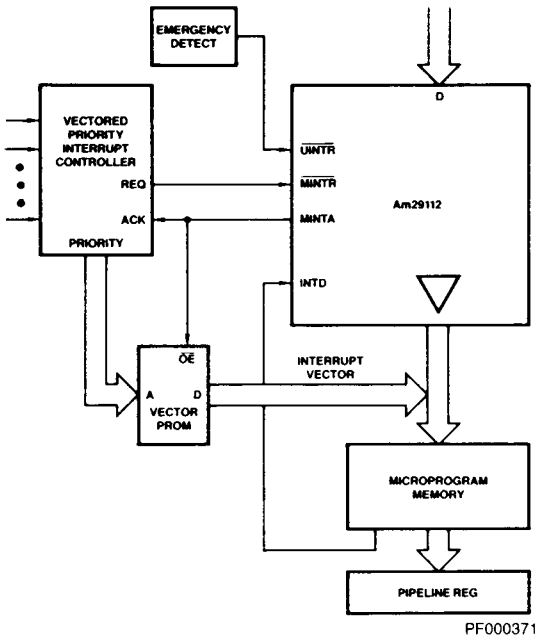


Figure 5a. Interrupt Control Loop.

Note: The INTD connection directly from microprogram memory.

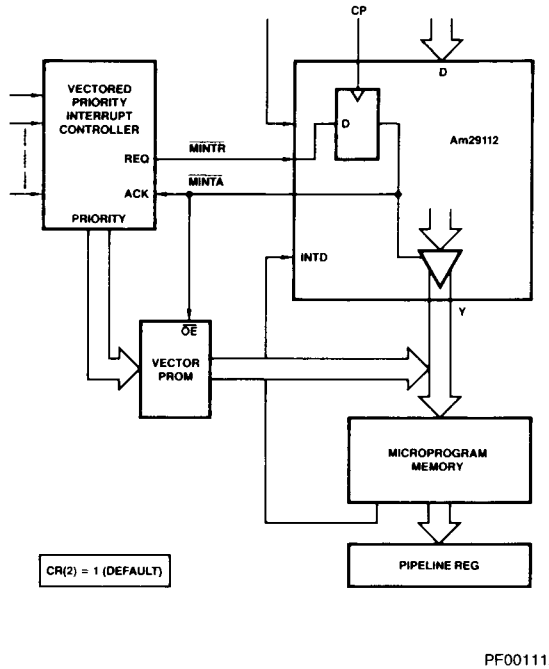


Figure 5b. No Postdelay.

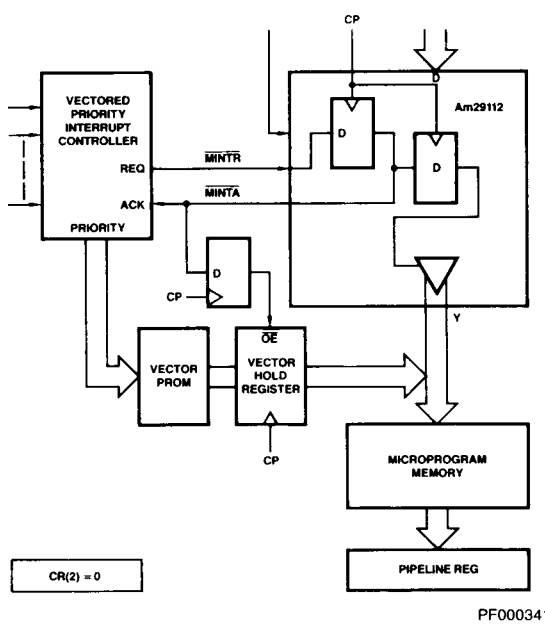
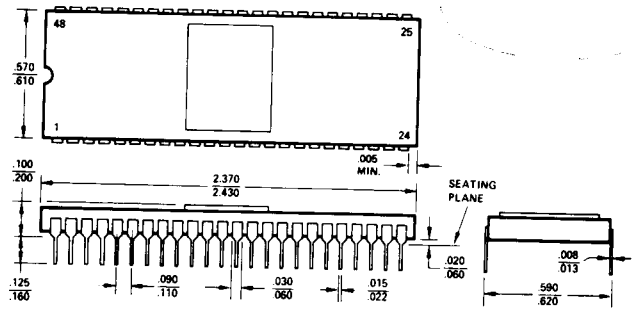


Figure 5c. With Postdelay.

PHYSICAL DIMENSIONS

D-48-2



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

U.S. AND CANADIAN SALES OFFICES

NORTHEAST AREA

Advanced Micro Devices
6 New England Executive Park
Burlington, Massachusetts 01803
Tel: (617) 273-3970

Advanced Micro Devices (Canada) Ltd.
2 Sheppard Avenue East
Suite 1610
Willowdale, Ontario
Canada M2N5Y7
Tel: (416) 224-5193

Advanced Micro Devices (Canada) Ltd.
AMD
4019 Carling #301
Kanata, Ottawa
Canada K2K2A3

Advanced Micro Devices
290 Elwood Davis Road
Suite 316
Liverpool, New York 13088
Tel: (315) 457-5400

MID-ATLANTIC AREA

Advanced Micro Devices
40 Crossways Park Way
Woodbury, New York 11797
Tel: (516) 364-8020

Advanced Micro Devices
Waterview Plaza, Suite 303
2001 U.S. Route #46
Parsippany, New Jersey 07054
Tel: (201) 299-0002

Advanced Micro Devices
110 Gibralter Road #110
Horsham, Pennsylvania 19044
Tel: (215) 441-8210
TWX: 510-665-7572

Advanced Micro Devices
Commerce Plaza
5100 Tilghman Street, Suite 320
Allentown, Pennsylvania 18104
Tel: (215) 398-8006
FAX: 215-398-8090

Advanced Micro Devices
205 South Avenue
Poughkeepsie, New York 12601
Tel: (914) 471-8180
TWX: 510-248-4219

Advanced Micro Devices
10 Main Street South
Southbury, Connecticut 06488
Tel: (203) 264-7600

Advanced Micro Devices
7223 Parkway Drive #203
Dorsey, Maryland 21076
Tel: (301) 796-9310
FAX: 796-2040

SOUTHEAST AREA

Advanced Micro Devices
4740 North State Road #7
Suite 102
Ft. Lauderdale, Florida 33319
Tel: (305) 484-8600

Advanced Micro Devices
7850 Ulmerton Road, Suite 1A
Largo, Florida 33541
Tel: (813) 535-9811

Advanced Micro Devices
15 Technology Parkway #200
Norcross, Georgia 30092
Tel: (404) 449-7920

Advanced Micro Devices
8 Woodlawn Green, Suite 220
Woodlawn Road
Charlotte, North Carolina 28210
Tel: (704) 525-1875

Advanced Micro Devices
303 Williams Avenue Southwest
Suite 118
Huntsville, Alabama 35801
Tel: (205) 536-5505

Advanced Micro Devices
6501 Six Forks, Suite 150
Raleigh, North Carolina 27609
Tel: (919) 847-8471

MID-AMERICA AREA

Advanced Micro Devices
500 Park Boulevard, Suite 940
Itasca, Illinois 60143
Tel: (312) 773-4422

Advanced Micro Devices
9900 Bren Road East, Suite 601
Minnetonka, Minnesota 55343
Tel: (612) 938-0001

Advanced Micro Devices
3592 Corporate Drive, Suite 108
Columbus, Ohio 43229
Tel: (614) 891-6455

Advanced Micro Devices
16985 West Blue Mound Road,
Suite 201
Brookfield, Wisconsin 53005
Tel: (414) 782-7748
FAX: (414) 782-8041

NORTHWEST AREA

Advanced Micro Devices
1245 Oakmead Parkway
Suite 2900
Sunnyvale, California 94086
Tel: (408) 720-8811

Advanced Micro Devices
One Lincoln Center, Suite 230
10300 Southwest Greenburg Road
Portland, Oregon 79223
Tel: (503) 245-0080

Advanced Micro Devices
Honeywell Ctr., Suite 1002
600 108th Avenue N.E.
Bellevue, Washington 98004
Tel:(206) 455-3600

MID-CALIF AREA

Advanced Micro Devices
360 N. Sepulveda, Suite 2075
El Segundo, California 90245
Tel: (213) 640-3210

Advanced Micro Devices
21600 Oxnard Street, Suite 675
Woodland Hills, California 91367
Tel: (213) 992-4155

SOUTHERN CALIF AREA

Advanced Micro Devices
4000 MacArthur Boulevard
Suite 5000
Newport Beach, California 92660
Tel: (714) 752-6262

Advanced Micro Devices
9619 Chesapeake Drive #210
San Diego, California 92123
Tel: (619) 560-7030

MOUNTAIN WEST AREA

Advanced Micro Devices
14755 Preston Road, Suite 700
Dallas, Texas 75240
Tel: (214) 934-9099

Advanced Micro Devices
8240 MoPac Expressway
Two Park North, Suite 385
Austin, Texas 78759
Tel: (512) 346-7830

Advanced Micro Devices
1873 South Bellaire Street
Suite 920
Denver, Colorado 80222
Tel: (303) 691-5100

Advanced Micro Devices
40 W. Baseline Road #206
Tempe, Arizona 85283
Tel: (602) 242-4400

Advanced Micro Devices
1955 W. Grant Road #125
Tucson, Arizona 85745
Tel: (602) 792-1200

INTERNATIONAL SALES OFFICES

BELGIUM

Advanced Micro Devices
Belgium N.V.—S.A.
Avenue de Tervueren, 412, bte 9
B-1150 Bruxelles
Tel: (02) 771 99 93
TELEX: 61028
FAX: 7623712

FRANCE

Advanced Micro Devices, S.A.
Silic 314, Immeuble Helsinki
74, rue d'Arcueil
F-94588 Rungis Cedex
Tel: (01) 687.36.66
TELEX: 202053
FAX: 686.21.85

GERMANY

Advanced Micro Devices GmbH
Rosenheimer Str. 143B
8000 Muenchen 80
West Germany
Tel: 49 89 41140
TELEX: 05-23883
FAX: 406 490

Advanced Micro Devices GmbH
Feuerseeplatz 4/5
D-7000 Stuttgart 1
Tel: (0711) 62 33 77
TELEX: 07-21882
FAX: 625 187

Advanced Micro Devices GmbH
Zur Worth 6
D-3108 Winsen/Aller
Tel: (05143) 53 62
TELEX: 925287
FAX: 5553

HONG KONG

Advanced Micro Devices
1303 World Commerce Centre
Harbour City
11 Canton Road
Tsimshatsui, Kowloon
Tel: (852) 3 695377
TELEX: 50426
FAX: (852) 123 4276

ITALY

Advanced Micro Devices S.R.L.
Centro Direzionale
Via Novara, 570
I-20153 Milano
Tel: (02) 3533241
TELEX: 315286
FAX: (39) 349 8000

JAPAN

Advanced Micro Devices, K.K.
Dai 3 Hoya Building
8-17, Kamitakaido 1 chome
Suginami-ku, Tokyo 168
Tel: (03) 329-2751
TELEX: 2324064
FAX: (03) 326 0262

SWEDEN

Advanced Micro Devices AB
Box 7013
S-172 07 Sundbyberg
Tel: (08) 98 12 35
TELEX: 11602
FAX: 298087

UNITED KINGDOM

Advanced Micro Devices (U.K.) Ltd.
A.M.D. House,
Goldsworth Road,
Woking,
Surrey GU21 1JT
Tel: Woking (04862) 22121
TELEX: 859103
FAX: 22179

Advanced Micro Devices (U.K.) Ltd.
The Genesis Centre
Garrett Field
Science Park South
Birchwood
Warrington WA3 7BH
Tel: Warrington (0925) 828008
TELEX: 628524
FAX: 827693

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

© 1984 Advanced Micro Devices, Inc.
Printed in U.S.A. SOG-B-2M-1/85-0

Advanced Micro Devices cannot assume responsibility for use of any circuitry described other than circuitry embodied in an Advanced Micro Devices' product.