AMD CUSTOMER EDUCATION

ED29300

Am29300 Family
of 32-Bit Building Blocks
for High-Performance Systems

Lecture

ED29300

Am29300 Family
of 32-bit Building Blocks
for High-Performance Systems

Lecture

Advanced Micro Devices, Inc.
Customer Education, MS 71
PO Box 3453
Sunnyvale, CA 94088

March 1987

Course Contents

Am29300 Family

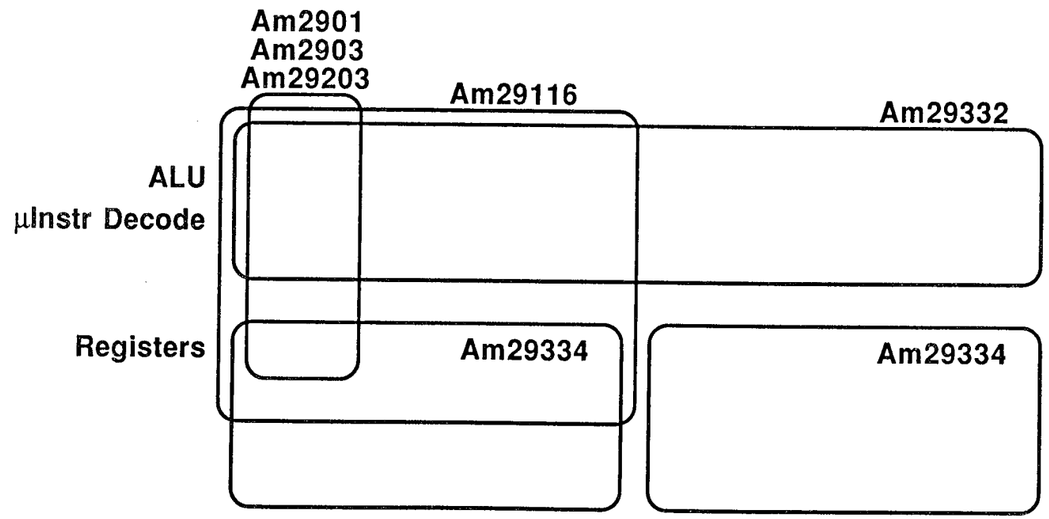
Unique Features

- Three-bus flow-through architecture
  - High throughput (no bidirectional busses)
  - Unlimited bus access
  - Unlimited register file expansion
  - Performance enhancing accelerators
- Functional Partitioning
  - Complete 32-bit functions (fewer devices)
  - Improved cycle time (80ns)
  - Supports concurrent processing architectures
- Regular, symmetric and powerful instruction set
  - Facilitates structured microprogramming
  - Clean interface to compiler for HLL support
  - All instructions are single cycle
- Complete interlocking fault detection
  - Parity detects interchip connection faults
  - Master/slave assures correct chip operation

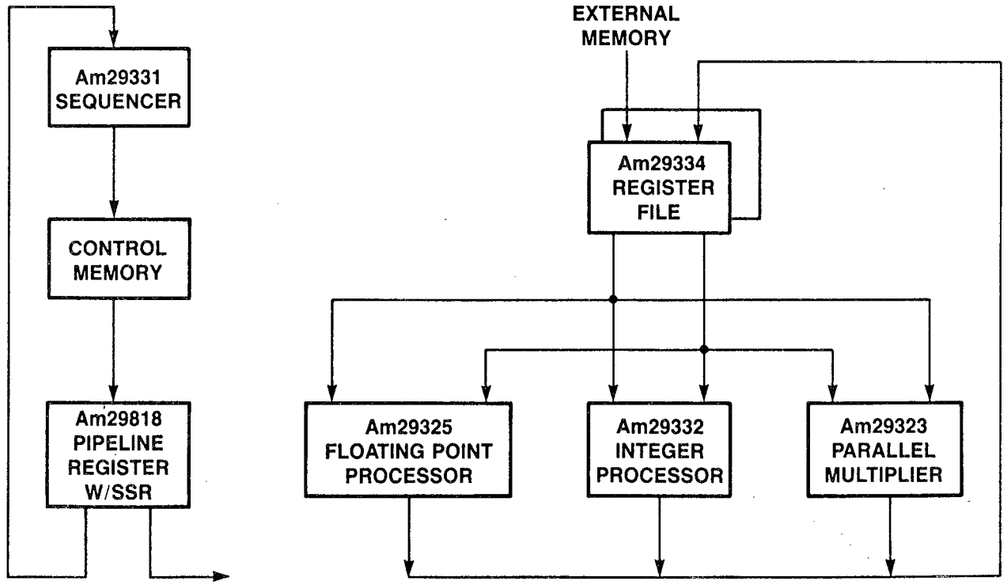
Am29300 Family

			Pins
Am29331	Sequencer		120
Am29332	32-Bit ALU		168
Am29325	Floating Point Processor		144
Am29323	32 x 32 Parallel Multiplier		168
Am29334	4 port Register File	4 port Register File	120

Traditional Vertical Slicing vs Am29300 Family



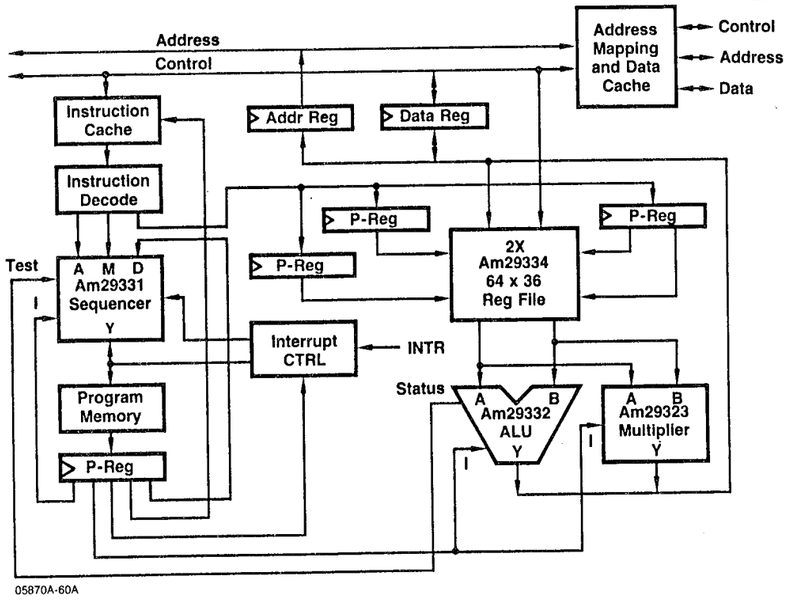
Am29300 Family



05870A-13

Am29300 Typical Application

High Performance 32-Bit CPU



05870A-60A

Comparision of Superminis

	VAX/780	MV/10000	MV/8000	Am29300
Cycle-Time	200ns	140ns	220ns	80-100ns
Address generator	no	yes	no	yes
Relative performance	1	1-2.5	1	2-4
Number of boards	26	12	9	1-2
Components	5000 IC Equivalent	2500 IC Equivalent	2000 IC	5 PGAs + 100-200 DIPs
Approx. power for CPU	1300 watts	800 watts	450 watts	200-300 watts

Am29332 Arithmetic Logic Unit

Am29332 Arithmetic Logic Unit

- ECL with TTL I/O
- 60 nsec delay or less
- Flow-through architecture
- 169-pin (168 + index) PGA package
- Cascadable for some operations
  (for example, MACRO-CARRY/LINK allow cascading ADD)

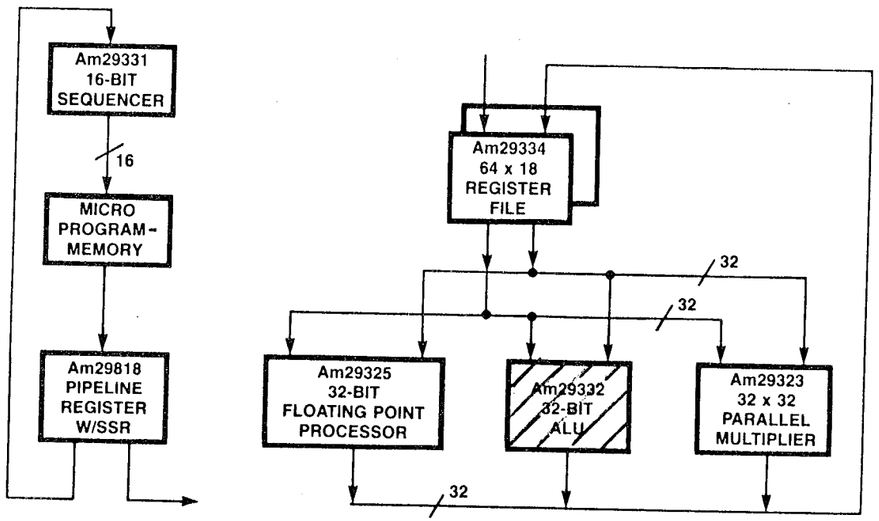
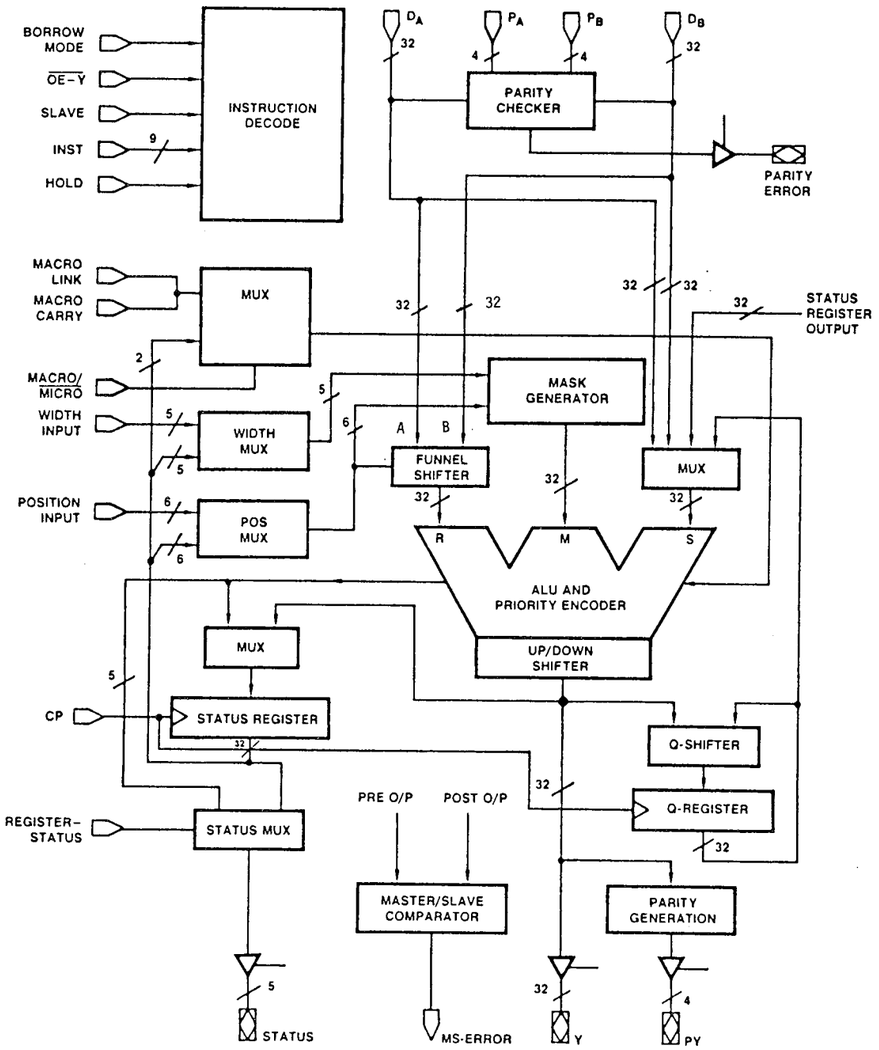


Figure 1. Am29300 Family High Performance System Block Diagram

Am29332



Am29332 Function Description

64-bit Funnel Shifter

- **64-bits input** (AB, AA, or BB concatenations)
- **Most significant 32 bits output** to ALU (R)
- Shifts up or down
- Position value gives number of bits to shift
  - Positive P_5-P_0 : **shift up** (0 to 31)
  - Negative P_5-P_0 : **shift down** (1 to 32)
- AA or BB inputs provide **rotates**

Priority Encoder

- Generates 5-bit binary code indicating **number of leading zeros.**
- Operates on 1-, 2-, 3-, or 4-byte data
- Example

31	23	15	7
0 . . . 0	0 . . . 0	00011010	00100000

Priority Encoded Output for selected width of:

1 byte = 2

2 byte = 3

3 byte = 11

4 byte = 19

Am29332

Mask Generator

- Generates mask to **select operand** of desired **width and position**
- **Byte-boundary-aligned** operands
  - 1, 2, 3, or 4 bytes (right adjusted)
  - I_8, I_7 instruction bits select width

I_8	I_7	Width (Bytes)
0	0	4
0	1	1
1	0	2
1	1	3

- **Variable-width** operands
  - 1 to 32-bit widths
  - Width determined by W_4-W_0
  - Position of LSB of field determined by P_5-P_0
- **Generates fence** of all 0's length W_4-W_0 or byte width I_8I_7
- **Upshifts fence** P_5-P_0 bits, 1 LSB extend

  (bits do not wrap around)
- Can be used as a **pattern generator**
  - Use Pass-Mask instruction
  - Example: Memory Test Sequences

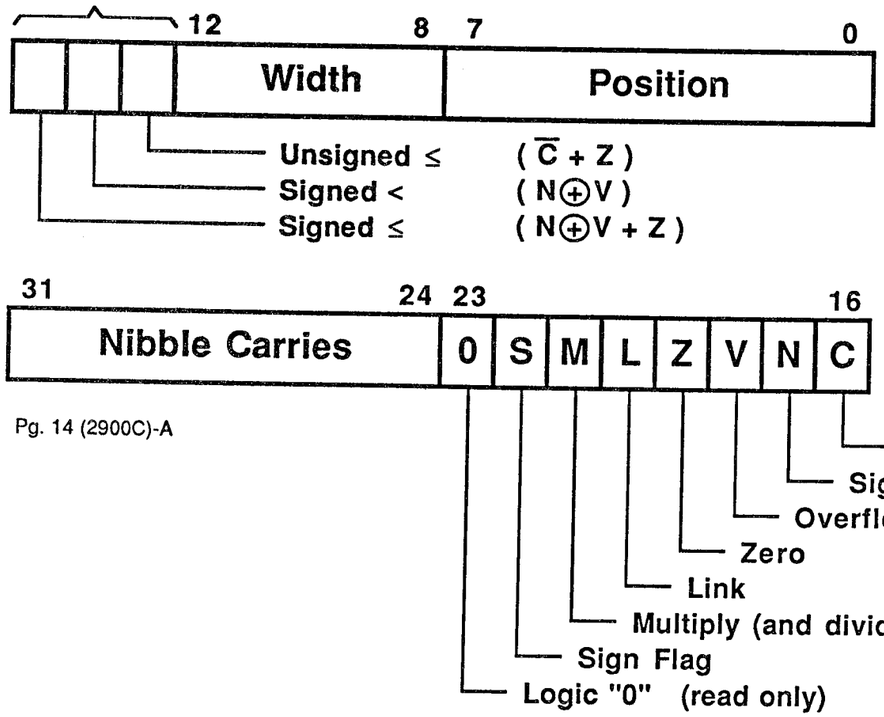
Instr. Sequence*	Y(32-bits)
0, PASS-MASK, 1, 0	00 ------- 001
0, PASS-MASK, 1, 1	00 ------- 010
0, PASS-MASK, 1, 2	00 ------- 100
.	.
.	.
.	.

*Format: <byte-width>, <op-code>,

Am29332 Status Register

Status Register

Read Only



Pg. 14 (2900C)-A

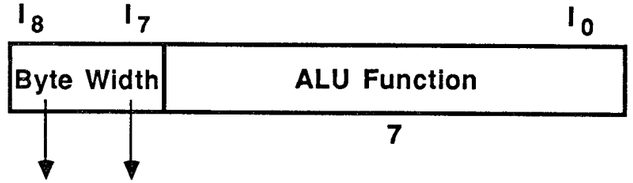
Notes:

1. Link bit used by single-bit shift and divide instructions.
2. Sign flag used primarily by sign extend instructions.
3. Nibble carries used for BCD operations.

Am29332 Instruction Formats

Instruction Format 1

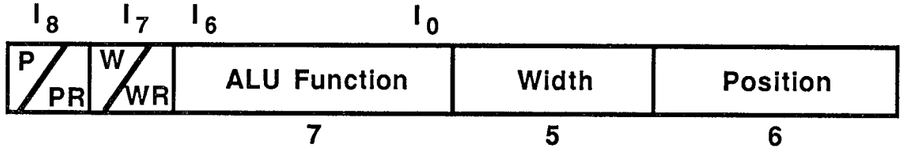
Byte Aligned Operators



0	0	-	Four Bytes
0	1	-	One Byte
1	0	-	Two Bytes
1	1	-	Three Bytes

Instruction Format 2

Field and Bit Operators

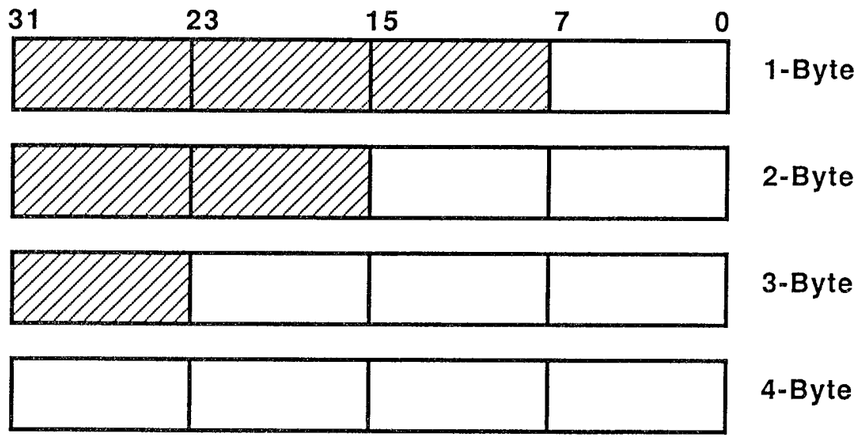


P	=	Position from input pins	($I_8 = 0$)
PR	=	Position from Status Register	($I_8 = 1$)
W	=	Width from input pins	($I_7 = 0$)
WR	=	Width from Status Register	($I_7 = 1$)

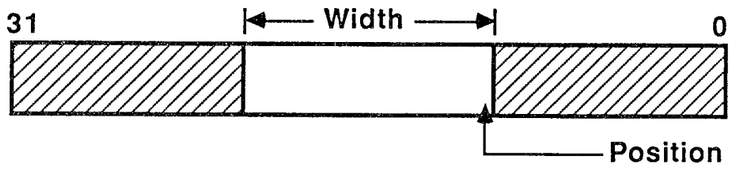
Pg. 15 (2900C)-A

Am29332 Data Types

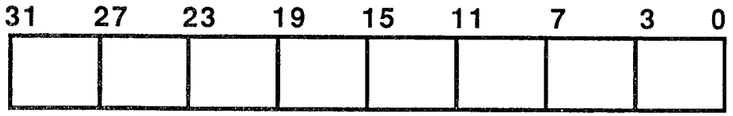
1. Byte Boundary Aligned Operands



2. Variable Length Bit Field



3. BCD (2, 4, 6, 8 digits)



INSTRUCTION SET SUMMARY

Operand Size: Variable Byte Width: 1, 2, 3, 4 Bytes		
Type	**Operation**	**Data Type**
Arithmetic	• Increment by one, two, four • Decrement by one, two, four • Add, addc (carry = macro/micro) • Sub, subr • Subc, subrc (carry/borrow) • BCD sum and difference correct steps	Binary Integer and BCD
	• Negate (two's complement) • Multiply steps (modified Booth) • Divide steps (non-restoring) { (Signed and unsigned)	Binary Integer
Prioritize	• Prioritize	Binary
Logical	• Not, OR, AND, XOR, XNOR, zero, sign	Binary
Single-Bit Shifts	• Upshift with 0, 1, link fill • Downshift with 0, 1, link, sign fill { (Single and double precision)	Binary
Data Movement	• Zero extend • Sign extend • Pass-status, Q-Reg • Load-status, Q-Reg • Merge	Binary

Operand Size: 32 Bits		
Type	**Operation**	**Data Type**
N-Bit Shifts N-Bit Rotates	• Upshift by 0 to 31 bits with 0 fill • Downshift by 1 to 32 bits with 0, sign fill • Rotate by 0 to 31 bits	Binary

Operand Size: Single Bit		
Type	**Operation**	**Data Type**
Bit Manipulation	• Extract • Set • Reset	Binary

Operand Size: Variable Length Bitfield: 1 to 32 Bits		
Type	**Operation**	**Data Type**
Field Logical (aligned and non-aligned)	• Not, OR, XOR, AND, extract, insert	Binary
Mask	• Pass-mask	Binary

Instruction Format Used in Examples

Byte Aligned Operands:

```
<byte-width> , <op-code>

0 - 32-bit word      move instructions
1 -  8-bit word      logical instructions
2 - 16-bit word      single-bit shift instructions
3 - 24-bit word      prioritize instructions
                     arithmetic instructions
```

Examples:

0,LOADQ-A	Load 32 bits of A-input into the Q register.
2,AND	AND the A-input with the B-input where A and B are 16-bit quantities.

Instruction Format Used in Examples (Cont.)

Field and Bit Operations:

<position-width-select> , <op-code> , <width> , <position>

1 to 31, 0 -32 to +31

Value	Position	Width	Instructions
0	input pins	input pins	Shift/rotate
1	input pins	status reg	Single-bit
2	status reg	input pins	Bit-field
3	status reg	status reg	Logical
			Mask

Note: (1) If <width> or <position> are not needed in an operation or are obtained from STATUS leave it out, but include all commas.

(2) <position> is the only field which has a sign.

Example:

0,PASSF-A,14,4

- four bits to right of field
- field is 14 bits wide
- output consists of field bits taken from A input, rest of bits taken from the B input
- position and width comes from input pins to chip rather than the status register

1. Move Instructions

(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
ZERO-EXTA	0 0	zero extend A	ZERO	Y				*		*	
ZERO-EXTB	0 1	zero extend B	ZERO	Y				*		*	
SIGN-EXTA	0 2	sign extend A	SIGN	Y				*		*	
SIGN-EXTB	0 3	sign extend B	SIGN	Y				*		*	
PASS-STAT	0 4	pass Status to Y	B	Y							
PASS-Q	0 5	pass Q register to Y	B	Y							
MERGEA-B	0 E	merge A with B	B	Y				*		*	
MERGEB-A	0 F	merge B with A	A	Y				*		*	
ZERO	3 C	pass 0's to Y	B	Y				1		0	
SIGN	3 D	pass -1 if N = 1; 0 otherwise	B	Y				$\overline{N}$			
LOADQ-A	0 6	load A into Q	Q	Q, Y				*		*	
LOADQ-B	0 7	load B into Q	Q	Q, Y				*		*	
LDSTAT-A	1 C	load A into Status register	S	S, Y	+	+	+	+	+	+	+
LDSTAT-B	1 D	load B into Status register	S	S, Y	+	+	+	+	+	+	+

Pg. 18 (ED2900C)-A

Examples:

2, ZERO-EXTB Pass lower two bytes of B to Y with zero fill on upper two bytes

0, LOADQ-A Load all four bytes of A into Q register, pass updated Q register to Y

Legend:

+: updated only if bytewidth = 0 or 3
*: Status is modified
A: A input
B: B input
S: Status Register
Q: Q Register

MOVE INSTRUCTIONS

Notes:

- Use 0,ZERO-EXTA or 0,ZERO-EXTB to sign extend an unsigned byte or word to a full 32-bit number.
- Use 0,ZERO to create upper 32-bit value of a 64-bit unsigned number.
- Use 0,SIGN-EXTA or 0,SIGN-EXTB to sign extend a signed byte or word to a full 32-bit number.
- Use 0,SIGN to create upper 32-bit value of a 64-bit signed number.
- Use MERGE-A (or MERGE-B) to pack least significant bytes of A into B (or B into A).
- For LDSTAT instructions, STATUS bits are loaded as follows:

STATUS bits	A (or B) bit
C	16
N	17
V	18
Z	19
L	20
M	21
S	22

Using the Example Coding Form

Fill in the columns as follows:

- For sequence control (Am29331) in the OP column, write the mnemonic for the sequencer instruction.

  If the specified instruction requires a branch address or count value, write it in the "Branch" column.

  If the instruction is conditioned, write the condition code in the "Cond Select" column.

  If the instruction uses the Multiway facility, enter the number (0...3) in the "Multi Sel" column.

- For ALU control (Am29332) in the OP column, write the mnemonic for the ALU instruction.

  In the B/W column indicate the number of bytes to be used. (Note that 0 means 4 bytes.)

  If the Field Logical facility is being used, enter values in the "Width" and "Position" columns.

  When writing to the register file, put a "0" in the Y-OUT $\overline{OE}$ column, otherwise enter a "1".

- For the Register File (Am29334) specify the register address(es) for the A and/or B source operands in the A-IN and B-IN columns.

  Specify the destination register address in the Y-OUT column.

MOVE INSTRUCTION EXERCISES

(All operands are 32-bits unless otherwise noted)

Using the Am29300 Example Coding Form, give the Am29332 and Am29334 fields to perform the following functions:

1. Store the 16-bit contents of R7 in the Q register.

2. Store the 32-bit contents of R12 in R24.

3. Merge the lower two bytes of R17 with the upper two bytes of R1, and store the result in R1.

4. Convert the single-precision 24-bit signed number in R3 to a double-precision 48-bit number. Store the result in R3 (least significant 24 bits) and R4 (most significant 24-bits).

5. Load the STATUS width and position fields from R4.

6. Clear the Q register to zero.

29300 Example Coding Form

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
CONT				2	LOADQ-A			R7			

Note: Assume Y-OUT feeds the B-Input to the Am29334.

29300 CF-8

2. Logical Instructions

(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
NOT-A	0 8	one's complement of A	A	Y				*		*	
NOT-B	0 9	one's complement of B	B	Y				*		*	
OR	3 E	OR of A and B	B	Y				*		*	
XOR	3 F	EXCLUSIVE OR of A and B	B	Y				*		*	
AND	4 0	AND of A and B	B	Y				*		*	
XNOR	4 1	EXCLUSIVE NOR of A and B	B	Y				*		*	

Pg. 19 (ED2900C)-A

Examples:

2, NOT-A — Complement low order two bytes of A and output to Y with high order two bytes of A uncomplemented

1, AND — AND first byte of A and B. Output to Y with high three bytes of B.

LOGICAL INSTRUCTION EXERCISES

(All operands are 32-bits unless otherwise noted)

Using the Am29300 Example Coding Form, give the Am29332 and Am29334 fields to perform the following functions. Move instructions may be needed in some solutions.

1. Complement the three-byte contents of R14. Leave the result in R14.

2. Exclusive-OR R10 with R11. Leave the result in R11.

3. AND R21 with R30 and put result in R25.

4. Merge bits 16-31 of R16 with bits 0-15 of R9 and put results in R0.

5. Set bits 16 to 23 of R31 to zero. Leave the other bits alone in R31.

29300 Example Coding Form

Am29331					Am29332			Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multl Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	OE

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF

3a. Single-bit Shift Instructions (Single Precision)

(Aligned Format)

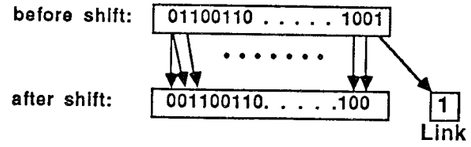
Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
DN1-0F-A	2 0	downshift A by 1 bit, zero fill	A	Y			*	*		*	
DN1-1F-A	2 4	downshift A by 1 bit, one's fill	A	Y			*	*		*	
DN1-LF-A	2 8	downshift A by 1 bit, link bit fill	A	Y			*	*		*	
DN1-AR-A	2 C	downshift A by 1 bit, sign bit fill	A	Y			*	*		*	
DN1-0F-B	2 1	downshift B by 1 bit, zero fill	B	Y			*	*		*	
DN1-1F-B	2 5	downshift B by 1 bit, one's fill	B	Y			*	*		*	
DN1-LF-B	2 9	downshift B by 1 bit, link bit fill	B	Y			*	*		*	
DN1-AR-B	2 D	downshift B by 1 bit, sign bit fill	B	Y			*	*		*	
UP1-0F-A	3 0	upshift A by 1 bit, zero fill	A	Y			*	*	*	*	
UP1-1F-A	3 4	upshift A by 1 bit, one's fill	A	Y			*	*	*	*	
UP1-LF-A	3 8	upshift A by 1 bit, link bit fill	A	Y			*	*	*	*	
UP1-0F-B	3 1	upshift B by 1 bit, zero fill	B	Y			*	*	*	*	
UP1-1F-B	3 5	upshift B by 1 bit, one's fill	B	Y			*	*	*	*	
UP1-LF-B	3 9	upshift B by 1 bit, link bit fill	B	Y			*	*	*	*	

Pg. 24 (ED2900C)-A

Example:

2, UP1-IF-A — Shift lower two bytes of A up one bit. Set LSB to 1. Fill unselected bytes with the unshifted upper two bytes of A.

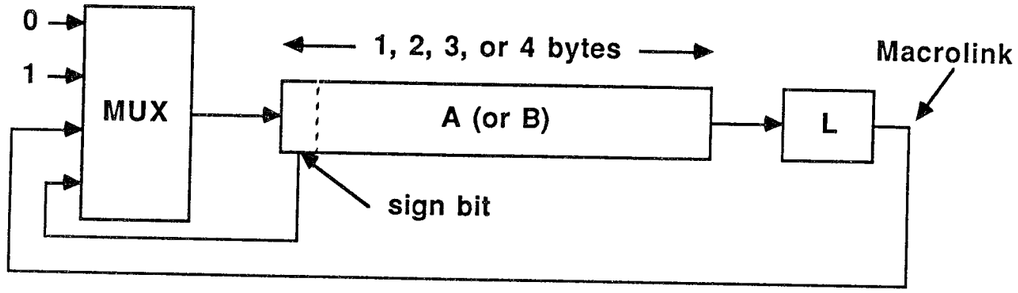
0, DN1-AR-B — Shift all 32-bits of B down one bit. Set the MSB of the result to the original sign bit.



Note: In opcodes above L is logical shift and A is arithmetic shift

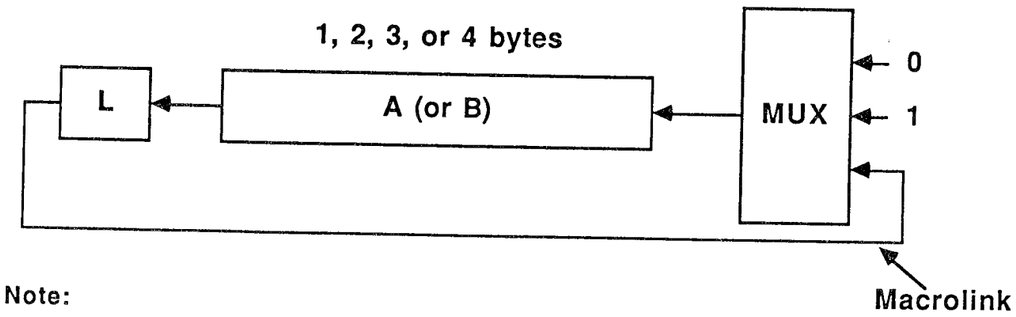
Single Precision Single-Bit Shift

Shift Down:



Shift with sign bit fill implements arithmetic shift.

Shift Up:



Note:

1. Use of the sign bit as fill bit implements arithmetic shifts.
2. For shift instructions using the L bit fill, either the L bit can be used for the fill bit, or the bit on the external MACROLINK can be used (when MACRO/MICRO = high).

Pg. 127 (2900C)

3b. Single-bit Shift Instructions (Double Precision)

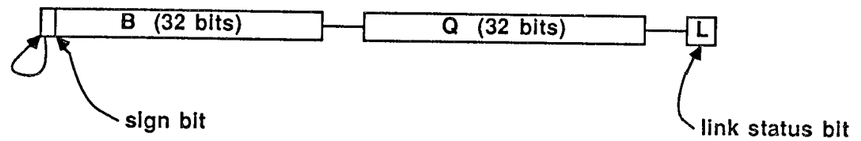
(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
DN1-0F-AQ	2 2	downshift A, Q by 1 bit, zero fill	A,Q	Y,Q			*	*		*	
DN1-1F-AQ	2 6	downshift A, Q by 1 bit, one fill	A,Q	Y,Q			*	*		*	
DN1-LF-AQ	2 A	downshift A, Q by 1 bit, link bit fill	A,Q	Y,Q			*	*		*	
DN1-AR-AQ	2 E	downshift A, Q by 1 bit, sign bit fill	A,Q	Y,Q			*	*		*	
DN1-0F-BQ	2 3	downshift B, Q by 1 bit, zero fill	B,Q	Y,Q			*	*		*	
DN1-1F-BQ	2 7	downshift B, Q by 1 bit, one fill	B,Q	Y,Q			*	*		*	
DN1-LF-BQ	2 B	downshift B, Q by 1 bit, link bit fill	B,Q	Y,Q			*	*		*	
DN1-AR-BQ	2 F	downshift B, Q by 1 bit, sign bit fill	B,Q	Y,Q			*	*		*	
UP1-0F-AQ	3 2	upshift A, Q by 1 bit, zero fill	A,Q	Y,Q			*	*	*	*	
UP1-1F-AQ	3 6	upshift A, Q by 1 bit, one fill	A,Q	Y,Q			*	*	*	*	
UP1-LF-AQ	3 A	upshift A, Q by 1 bit, link bit fill	A,Q	Y,Q			*	*	*	*	
UP1-0F-BQ	3 3	upshift B, Q by 1 bit, zero fill	B,Q	Y,Q			*	*	*	*	
UP1-1F-BQ	3 7	upshift B, Q by 1 bit, one fill	B,Q	Y,Q			*	*	*	*	
UP1-LF-BQ	3 B	upshift B, Q by 1 bit, link bit fill	B,Q	Y,Q			*	*	*	*	

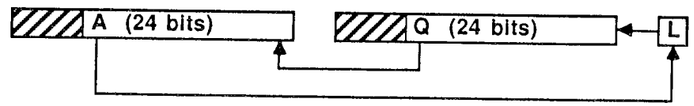
Pg. 20 (ED2900C)-A

Example:

0, DN1-AR-BQ Shift 64 bits (all 32 bits of both B and Q) down by one bit. LSB of B fills MSB of Q. MSB of B set to sign bit (bit N of status register)



3, UP1-LF-AQ Shift 48 bits (24-bits of A and 24-bits of Q) up by one bit. MSB of 24-bit Q fills LSB of A. MSB of 24-bit A sets link status bit. LSB of Q is filled with original link value.



SINGLE-BIT SHIFT INSTRUCTION EXERCISES

(All operands are 32-bits unless otherwise noted)

Using the Am29300 Example Coding Form, give the Am29332 and Am29334 fields to perform the following functions. Move and Logical instructions may be needed in some solutions.

1. Shift 16-bit R4 down by one bit and set the MSB of the result to zero.

2. Multiply R9 by 2.

3. Divide the 24-bit signed value in R21 by 2 and store the result in R0.

4. Shift the 64-bit unsigned value in R2,Q down by 1 bit.

5. Shift the 64-bit signed value in R2,R3 down by 1 bit.

29300 Example Coding Form

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF

4. Prioritize Instructions

(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
PRIOR-A	0 C	Number of leading 0-bits in A	—	Y, note 1				*			
PRIOR-B	0 D	Number of leading 0-bits in B	—	Y, note 1				*			

Pg. 21 (ED2900C)-A

Example:

3, PRIOR-A Value placed on Y is 2

Assume A is	01001011	00100010	00000000	00000000

Note:

1. Priority also loaded into STATUS <7:0>
2. Priority is zero if selected bytes of A (or B) is zero.

5. Arithmetic Instructions

(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
NEG-A	0 A	two's complement A	A	Y				*	*	*	*
NEG-B	0 B	two's complement B	B	Y				*	*	*	*
INCR-A	1 2	increment A by 1	A	Y				*	*	*	*
INCR-B	1 3	increment B by 1	B	Y				*	*	*	*
INCR2-A	1 6	increment A by 2	A	Y				*	*	*	*
INCR2-B	1 7	increment B by 2	B	Y				*	*	*	*
INCR4-A	1 A	increment A by 4	A	Y				*	*	*	*
INCR4-B	1 B	increment B by 4	B	Y				*	*	*	*
DECR-A	1 0	decrement A by 1	A	Y				*	*	*	*
DECR-B	1 1	decrement B by 1	B	Y				*	*	*	*
DECR2-A	1 4	decrement A by 2	A	Y				*	*	*	*
DECR2-B	1 5	decrement B by 2	B	Y				*	*	*	*
DECR4-A	1 8	decrement A by 4	A	Y				*	*	*	*
DECR4-B	1 9	decrement B by 4	B	Y				*	*	*	*

Pg. 22 (ED2900C)-A

Example:

2, DECR4-A Decrement lower two bytes of A by 4

Notes:

1. Borrow, rather than carry, is generated if BORROW-MODE pin is high (borrow = carry).
2. Nibble bits are set by these instructions. NEG-A (or NEG-B) and DIFF-CORR may be used to form 10's complement of a BCD number. Use SUM-CORR (for increment) or DIFF-CORR (for decrement) to increment for decrement a BCD number.

5. Arithmetic Instructions (continued)

(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
ADD	4 2	add A to B	B	Y				*	*	*	*
ADDC	4 3	add A to B with carry input	B	Y				*	*	*	*
SUB	4 4	subtract B from A	B	Y				*	*	*	*
SUBR	4 6	subtract A from B	B	Y				*	*	*	*
SUBC	4 5	subtract B from A with carry input	B	Y				*	*	*	*
SUBRC	4 7	subtract A from B with carry input	B	Y				*	*	*	*
SUM-CORR-A	4 8	correct nibbles in BCD word	A	Y				*	*	*	*
SUM-CORR-B	4 9	correct nibbles in BCD word	B	Y				*	*	*	*
DIFF-CORR-A	4 A	correct nibbles in BCD word	A	Y				*	*	*	*
DIFF-CORR-B	4 B	correct nibbles in BCD word	B	Y				*	*	*	*

Pg. 23 (ED2900C)-A

Example:

0, ADD-AB Add two 32-bit 2's cmplement integers A and B

Note:

1. For subtract operations, a borrow rather than a carry is stored in STATUS if BORROW-MODE pin is high. Carry is always generated for ADD regardless of BORROW-MODE.
2. Use SUM-CORR or DIFF-CORR to add or subtract BCD numbers.
3. Use add and subtract operations with carry to perform operations on integers longer than 32 bits.
4. Carry input operations obtain carry bit from the MACRO-CARRY pin if MACRO/MICRO pin is high. Otherwise, carry is obtained from the C status bit.

PRIORITY AND ARITHMETIC INSTRUCTION EXERCISES

(All operands are 32-bits unless otherwise noted)

Using the Am29300 Example Coding Form, give the Am29332 and Am29334 fields to perform the following functions. Move, Logical and Single-Bit Shift instructions may be needed in some solutions.

1. What is the value of Y for 2,PRIOR-A where the A-input is 032FH?
2. Determine the priority of the double precision number in A,B where it is known that the A-input is always zero and the MSB of B is 1. A is the MSB half of the number.
3. Put the two's complement of R17 in R0.
4. Increment R2 by 3.
5. Add R30 to R31 and store the result in R31.
6. Subtract R30 from R31 and store the result in R31.
7. Subtract the double-precision number in R7,R8 from the number in R9,R10 and put the result in R11,R12.
8. Subtract the BCD number in R30 from the BCD number in R31 and store the result in BCD in R31.

29300 Example Coding Form

Am29331					Am29332			Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF

Am29332 Division

- 8-, 16-, 24- or 32-bit data
- Signed and unsigned integers
- Non-restoring algorithm
- 32-bit/32-bit Divide (Worst Case):
  - Signed integer - 39 cycles @ 80ns = 3.12 us
  - Unsigned integer - 37 cycles @ 80ns = 2.96 us
- Supports Multiprecision Divide

In the algorithms which follow:

given: $\frac{\text{DIVDND}}{\text{DIVSR}}$

want: QUOT + REM

6. Divide Instructions

(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
		Signed Divide Steps									
SDIVFIRST	4 E	first instruction for signed divide	B	Y, Q	*	*	*	*		*	
SDIVSTEP	5 0	iterate step (#bits - 1 times)	B	Y, Q		*	*	*		*	*
SDIVLAST1	5 1	last divide instruction unless	B	Y, Q		*		*		*	*
SDIVLAST2	5 A	dividend & remainder negative	B	Y				*			
		Unsigned Divide Steps									
UDIVFIRST	4 F	first instruction for unsigned divide	B	Y, Q			*	*		*	
UDIVSTEP	5 4	iterate step (#bits - 1 times)	B	Y, Q	*	*	*	*			*
UDIVLAST	5 5	last instruction	B	Y, Q	0	*		*		*	*
		Multiprecision Divide Steps									
MPDIVSTEP1	5 2	first instruction	B	Y, Q							
MPDIVSTEP2	5 6	executed 0 times for double	B	Y, Q							
MPSDIVSTEP3	5 3	last instruction of inner loop	B	Y, Q							
MPUDIVSTP3	5 7	used for unsigned divide	B	Y, Q							
		Correction Steps									
REMCORR	5 8	correct remainder after divide	B	Y							*
QUOCORR	5 9	correct quotient after divide	B	Y					*		*

Example:

See Next Page

Note: Divisor in A, Dividend in A
Quotient in B, Remainder in B

Signed Division

Let DIVDND be the dividend
DIVSR be the divisor
REM be the remainder

Then the algorithm for divide using Am29332 instructions is:

1. Q <- DIVDND; (use LOAD-Q) N gets updated
2. REM <- SIGNEXTEND (00); (use SIGN with width 00) spread sign flag N into REM
3. FOR (BW*8-1), REM <- SDIVFIRST (DIVSR, REM); updates M flag with sign compare; single-bit upshift of REM Q.
4. ENDFOR, REM <- SDIVSTEP (DIVSR, REM); single cycle inner loop produces 1 quotient bit/cycle
5. REM <- SDIVLAST1 (DIVSR, REM); updates N based on final remainder value; forces 1 into LS quotient bit
6. If Z GOTO DONE, Y <- SDIVLAST2 (DIVSR, REM); updates Z for correction step
7. QUOT <- Q, use PASS-Q,
8. QUOT <- QUOCORR (QUOT); corrects the quotient
9. REM <- REMCORR (DIVSR, REM); adjust remainder

Done:10. ...

Note: Sequencer instruction in Step 3 sets up the Am29331 stack and counter for a loop starting at Step 4.

29300 Example Coding Form

Signed Division

Example for 16-bit valves

R1 - Quotient
R2 - Dividend
R3 - Remainder
R4 - Divisor

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch or Counter Valve	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
CONT				2	LOADQ-A			R2			1
CONT				0	SIGN					R3	0
FOR_D	15			2	SDIVFIRST			R4	R3	R3	0
DJMP_S				2	SDIVSTEP			R4	R3	R3	0
CONT				2	SDIVLAST1			R4	R3	R3	0
BRCC_D	DONE	Z									1
CONT				2	SDIVLAST2			R4	R3	R3	0
CONT				2	PASS-Q					R1	0
CONT				2	QUOCORR				R1	R1	0
CONT				2	REMCORR			R4	R3	R3	0
DONE:											

Pg. 27 (ED2900C)-A

Unsigned Division

1. Q <- DIVOND; use LOAD-Q.
2. REM <- ZEROEXTEND (00); use ZERO with width 00 to spread zeros into REM.
3. FOR (BW*8-1), REM <- UDIVFIRST (DIVSR, REM); updates S and M.
4. ENDFOR, REM <- UDIVSTEP (DIVSR, REM); single cycle inner loop produces 1 quotient bit/cycle.
5. REM <- UDIVLAST (DIVSR, REM); updates N flag based in final remainder value.
6. If $\overline{N}$ GOTO DONE, QUOT <- Q; use PASS-Q to move quotient to QUOT.
7. REM <- REMCORR (DIVSR, REM); correct remainder.

DONE: 8 . . .

Note: The S flag holds the sign of the partial remainder after each UDIVSTEP in Step 4.

Multiprecision Divide

(Algorithm assumes divisor and dividend are each two words long.)

1. Q <- $DIVDND_H$, BYTE-WIDTH=BW_H; use LOAD-Q to put MS Dividend in Q; update N
2. REM_L <- SIGN (00); use SIGN with width=00 to spread N across REM_L
3. REM_H <- SIGN (BYTE-WIDTH=BW_H); leaves high bytes of REM_H intact
4. FOR (BW_H*8), REM_L <- SDIVFIRST ($DIVSR_H$, REM_L), BYTE-WIDTH=BW_H; set up flags
5. REM_L <- MPDIVSTEP1 ($DIVSR_L$, REM_L), BYTE-WIDTH=BW_H
6. ENDFOR, REM_H <- MPSDIVSTEP3 ($DIVSR_H$, REM_H); BYTE-WIDTH=BW_H
7. $QUOT_H$ <- Q, BYTE-WIDTH=BW_H; use PASS-0
8. $QUOT_H$ <- Q, BYTE-WIDTH=00; midpoint adjust
9. Q <- $DIVDND_L$, BYTE-WIDTH=00; load next dividend
10. FOR (32), REM_L.Q <- UP 1-0F (REM_L), BYTE-WIDTH=00
11. REM_L <- MPDIVSTEP1 ($DIVSR_L$, REM_L), BYTE-WIDTH=00
12. ENDFOR, REM_H <- MPSDIVSTEP3 ($DIVSR_H$, REM_H), BYTE-WIDTH=BW_H;
13. $QUOT_L$ <- Q, BYTE-WIDTH=00;
14. REM_H <- DN1-LF (REM_H), BYTE-WIDTH=BW_H;
15. REM_L <- DN1-LF (REM_L), BYTE-WIDTH=00; adjust remainder position

    (correction microcode)

Multiprecision Divide (Cont.)

Notes:

1. BW_H stands for the number of bytes in the most significant half of the divisor/dividend.
2. There are two loops for quotient computation. The first loop starts with the high bytes of dividend in Q and computes the high bytes of Quotient. The Q register is then read out and loaded with the low word of dividend, and a second loop entered to compute the low bytes of quotient.
3. Step 5 contains a tricky use of SDIVFIRST for double precision signed divide set up.
4. Unsigned division can be coded along the same lines, using UDIVFIRST instead of SDIVFIRST, and MPUDIVSTEP3 instead of MPSDIVSTEP3.
5. For operands longer than 64 bits, the inner loop can still be coded efficiently using MPDIVSTEP2. For example, for triple precision division, the inner loop looks like:

   LOOP: 1. REM_L <- MPDIVSTEP1 ($DIVSR_L$, REM_L), BYTE-WIDTH=00;
   2. REM_{INT} <- MPDIVSTEP2 ($DIVSR_{INT}$, REM_{INT}), BYTE-WIDTH=00;
   3. REM_H <- MPSDIVSTEP3 ($DIVSR_H$, REM_H), BYTE-WIDTH=BW_H;

   Any number of MPDIVSTEP2 microinstructions may be used for very long dividends.
6. Before quotient correction is applied, the least significant quotient bit should be set (use SETBIT on QUOT) so that the quotient is algebraically correct, i.e. quotient * divisor = dividend - remainder.

Am29332 Multiplication

- 8- 16-, 24- or 32-bit data
- Signed and unsigned integers
- Modified Booth algorithm
- Two product bits per cycle
- 32 x 32 Multiply (worst case)
  - Signed integer - 17 cycles @ 80ns = 1.36 us
  - Unsigned integer - 18 cycles @ 80ns = 1.44 us

For the algorithms which follow:

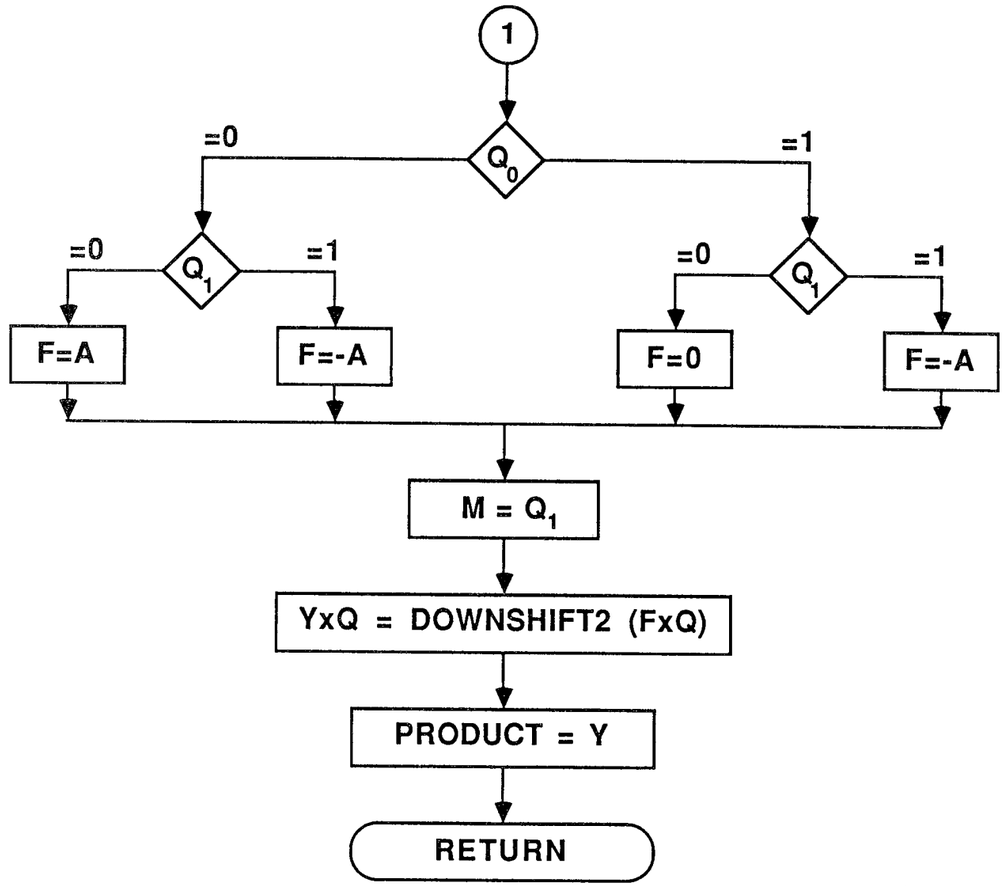
Input: MPR x MCND

Output: PROD

Modified Booth Algorithm

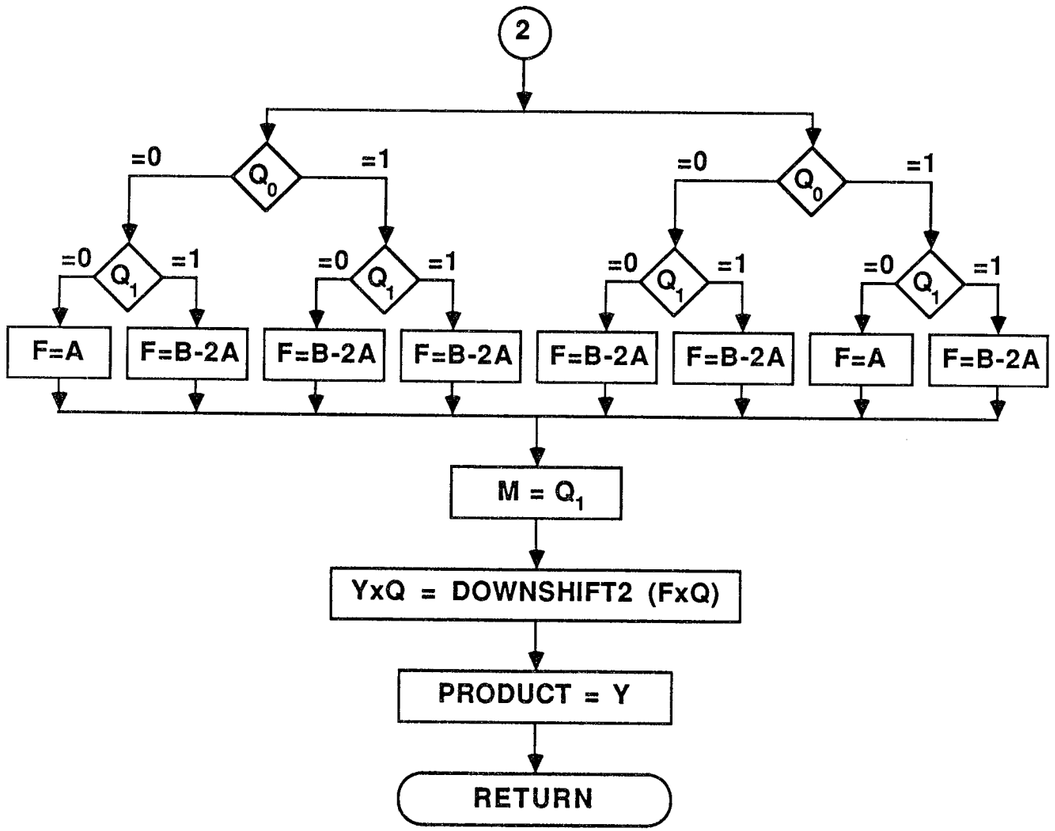
(F is an internal register in the Am29332)

Initialization and first iteration step.



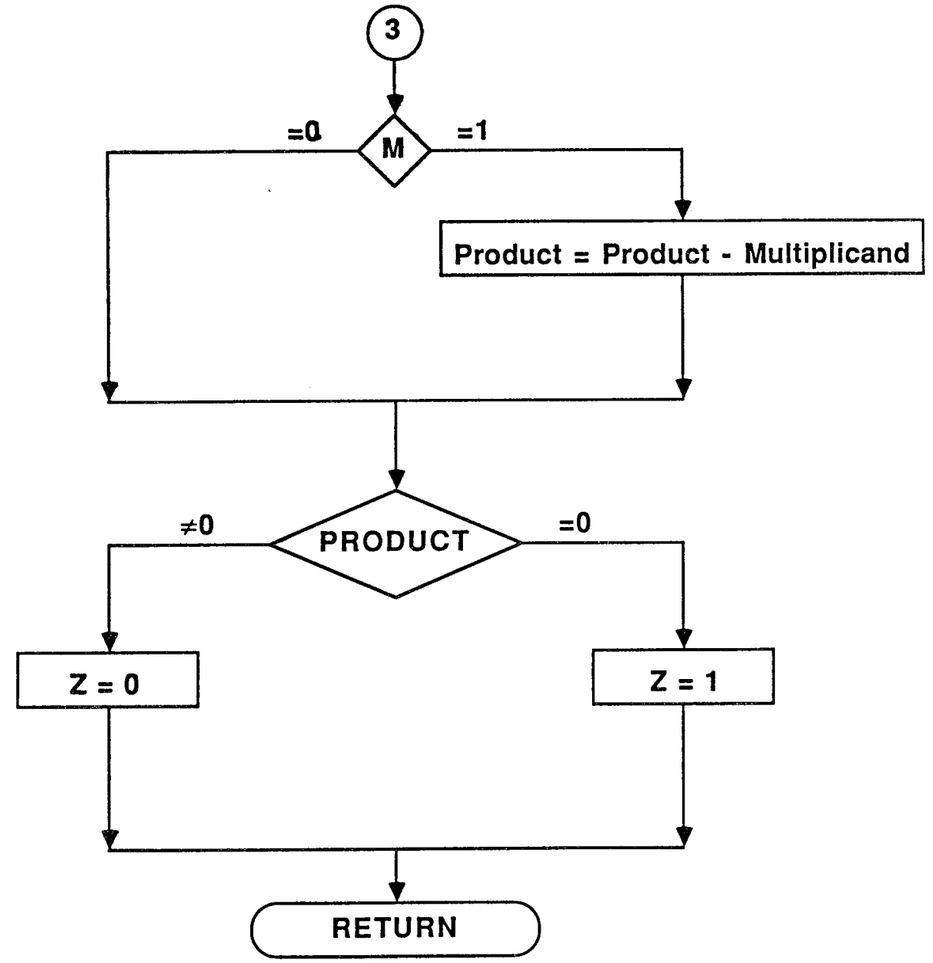
A is multiplicand.

Iteration Step



A is the multiplicand.
B is the product.

Final Correction Step for Unsigned Multiplication



7. Multiply Instructions

(Aligned Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
		Signed Multiply Steps									
SMULFIRST	5 F	first multiply instruction	B	Y†							
SMULSTEP	5 E	iterate step (#bits/2 - 1 steps)	B	Y†							
		Unsigned Multiply Steps									
UMULFIRST	5 B	first multiply instruction	B	Y†		*					
UMULSTEP	5 C	iterate step (#bits/2 - 1 steps)	B	Y†		*					
UMULLAST	5 D	last multiply instruction	B	Y†				*			

Note:

† Put ALU output in B

1) Multiplicand in A, multiplier in Q
   Product (high) in B, Product (low) in Q

2) Set product-high (PRODH) to zero at beginning of routine
   (see example)

Pg. 29 (ED2900C)-A

Signed Multiplication

Product formed in Q and $PROD_H$ register.

1. Zero out Q, Q <- MPR; use LOAD-Q
2. FOR (4*BW-1), $PROD_H$ <- SMULFIRST (MCND)
3. ENDFOR, $PROD_H$ <- SMULSTEP ($PROD_H$, MCND
4. $PROD_L$ <-Q:

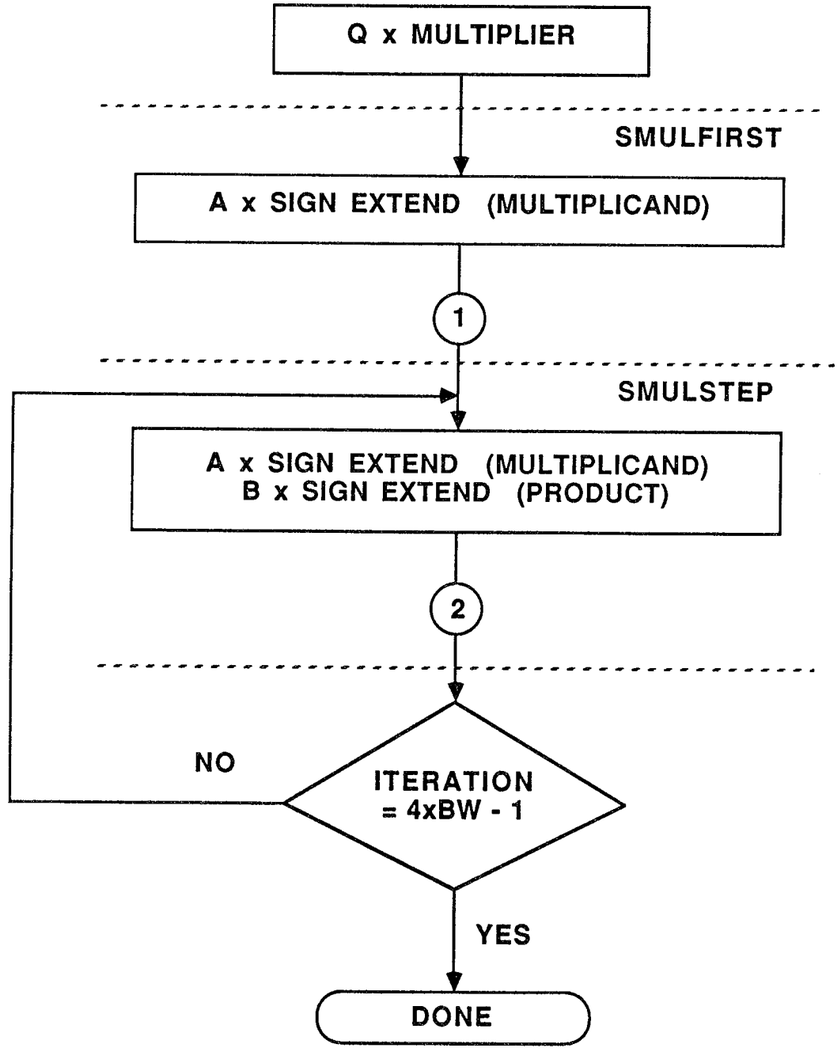
Notes:

1. $PROD_H$ does not need to be initialized; this is done by Step 2.
2. For integer multiplication where a single precision result is required, $PROD_H$ may be discarded if it is zero, or if it is -1 and $PROD_L$ is negative. Otherwise, overflow has occurred.
3. For multiplication of normalized mantissas, where a single precision result is required, $PROD_L$ may be discarded.
4. In either case where a double precision result is required and where the operands were smaller than a word, it is necessary to merge the least significant bytes of $PROD_H$ and $PROD_L$. For example, for three byte operands:

   a. $PROD_L$ <-PASSF-A ($PROD_H$, $PROD_L$, P=24, W=8)

   b. $PROD_H$ <-PASSF-A ($PROD_H$, $PROD_H$, P =-8, W=16)

Algorithm for Signed Multiplication



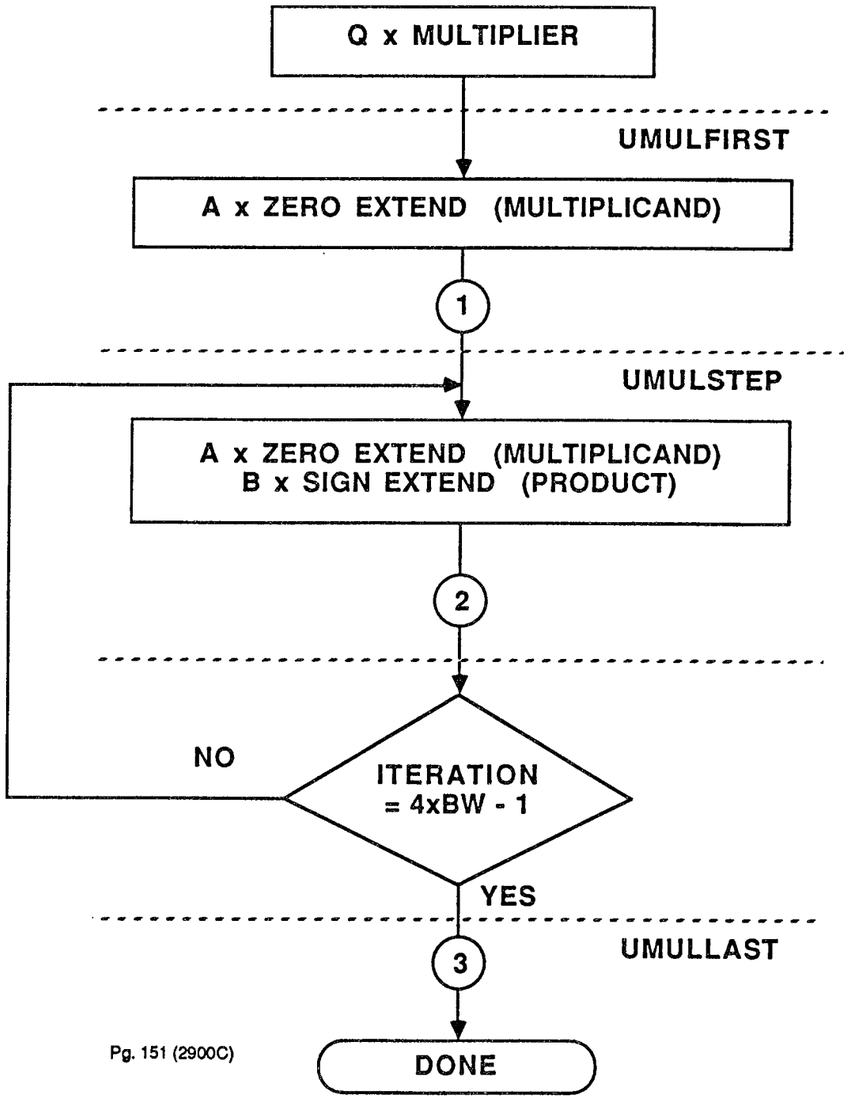
Unsigned Multiplication

1. Zero out Q register, Q <- MPR;
2. FOR (4*BW-1), $PROD_H$ <- UMULFIRST (MCND):
3. ENDFOR, $PROD_H$ <- UMUSTEP ($PROD_H$, MCND):
4. $PROD_H$ <- UMULLAST ($PROD_H$, MCND):
5. $PROD_L$ <- Q

Notes:

Notes (1) through (4) for signed multiplication apply here as well. Exception: for single precision unsigned multiplication, the overflow condition is simpler: $PROD_H$=0).

Algorithm for Unsigned Multiplication



Pg. 151 (2900C)

29300 Example Coding Form

Unsigned Multiply

R1 - Multiplier
R2 - Multiplicand
R3 - Product (high)
R4 - Product (low)

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multl Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	OE
CONT				3	ZERO				R3	R3	0
CONT				3	LOADQ-A			R1			1
FOR_D	11			3	ULMULFIRST			R2	R3	R3	0
DJMP_S				3	UMULSTEP			R2	R3	R3	0
CONT				3	UMULLAST			R2	R3	R3	0
CONT				3	PASS-Q					R4	0

Pg. 27 (2900C)

- For B/W =3, word width is 24 bits.
- Multiply is performed two bits at a time,
  requiring one first instruction,
  eleven step instructions, and one last instruction.

8. Shift/Rotate Instructions

(Field Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
NB-0F-SHA	6 2	A full-word shift with zero fill †	—	Y				*		*	
NB-SN-SHA	6 0	A full-word shift with sign fill †	—	Y				*		*	
NB-0F-SHB	6 3	B full-word shift with zero fill †	—	Y				*		*	
NB-SN-SHB	6 1	B full-word shift with sign fill †	—	Y				*		*	
NBROT-A	6 4	rotate full-word A up	—	Y				*		*	
NBROT-B	6 5	rotate full-word B up	—	Y				*		*	

Pg. 31 (ED2900C)-A

† positive position - up shift
negative position - down shift

Examples:

0, NB-0F-SHA,,4 Shift A up 4 bits and zero fill

0, NB-SN-SHB,,-17 Shift B down 17 bits and sign fill

Note:

1. Width field not used.
2. Range of shift amount is -32 (downshift) to +31 (upshift).
3. NB-SN-SHA and NB-SN-SHA fills with sign of A or B for downshift; 0 for upshift.
4. Rotate instructions ignore sign of position.
5. Position comes from STATUS bits <5:0> if POSITION_SEL pin is high. Otherwise, position comes from POSITION pins <5:0>.

SHIFT AND ROTATE INSTRUCTION EXERCISES

(All operands are 32-bits unless otherwise noted)

Using the Am29300 Example Coding Form, give the Am29332 and Am29334 field to perform the following functions. Move, logical, single-bit shift, priority, and arithemtic instructions may be also needed in some solutions.

Assume position and width is obtained from the input pins of the Am29332.

1. Shift R22 up by 17 with zero fill.

2. Rotate R1 down by 14.

3. Shift R22 down by 17 with zero fill.

4. Shift R22 down by 17 bits with one's fill.

29300 Example Coding Form

Am29331					Am29332			Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multl Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF

9. Bit Instructions

(Field Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
SETBIT-A	6 8	set bit in <position> of A	A	Y				0			
SETBIT-B	6 9	set bit in <position> of B	B	Y				0			
SETBIT-STAT	6 C	set bit in <position> of STATUS	STATUS	Y	1	1	1	1	1	1	1
RSTBIT-A	6 A	reset bit in <position> of A	A	Y				1			
RSTBIT-B	6 B	reset bit in <position> of B	B	Y				1			
RSTBIT-STAT	6 D	reset bit in <position> of STATUS	STATUS	Y	1	1	1	1	1	1	1
EXTBIT-A	6 6	extract bit in <position> of A †	—	Y			*	*			
EXTBIT-B	6 7	extract bit in <position> of B †	—	Y			*	*			
EXTBIT-STAT	7 E	extract bit in <position> of STATUS †	—	Y			*				

Pg. 32 (ED2900C)-A

If status selected, only bit set/reset is affected

Examples:

0, RESET-B,,3 3rd bit is set to 0 in B

0, EXTBIT-S,, 4 4th bit in status register is extracted and inverted.

† If sign of <position> is negative, then extracted bit is complemented on Y

Note:

1. For extract bit instructions, extracted bit appears on Y with the high order bits of Y set to zero.
2. For set and reset bit instructions, sign of position is not used.

BIT FIELD INSTRUCTION EXERCISES

(All operands are 32-bits unless otherwise noted)

Using the Am29300 Example Coding Form, give the Am29332 and Am29334 fields to perform the following functions. Move, logical, single-bit shift, priority, arithmetic, and shift and rotate instructions may be needed in some solutions.

1. Set bit 30 in R29.

2. Reset bit 21 in the Q register.

3. Set the zero status bit in the STATUS register.

4. Load 1 into R30 if STATUS zero bit is a 0. Otherwise load 0 into R30.

29300 Example Coding Form

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multl Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$

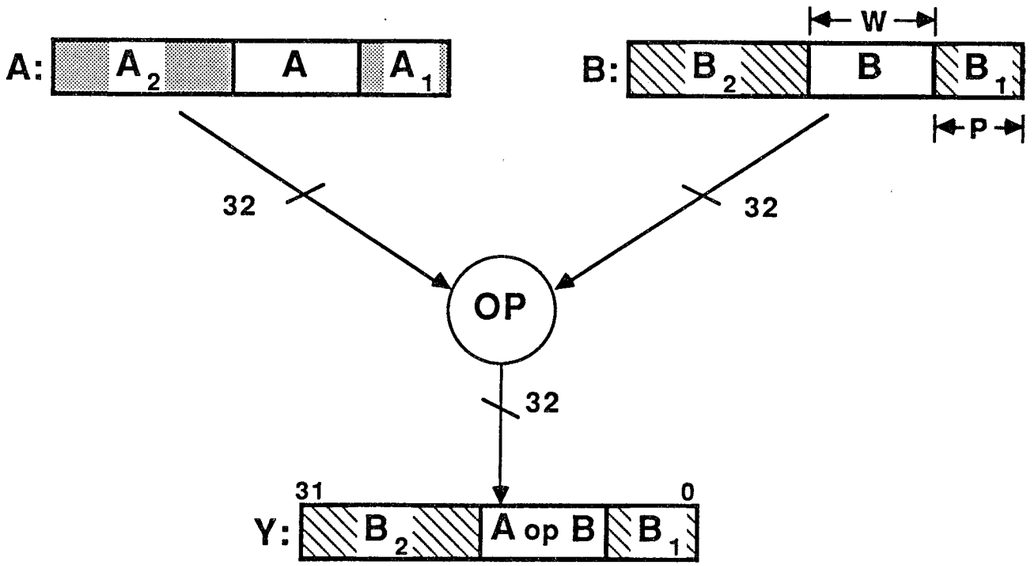
Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF

Am29332

Field Logical Operations

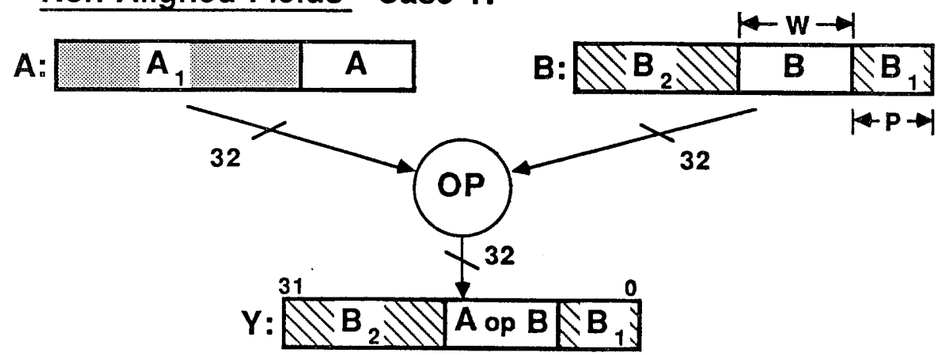
Aligned Fields:



Am29332

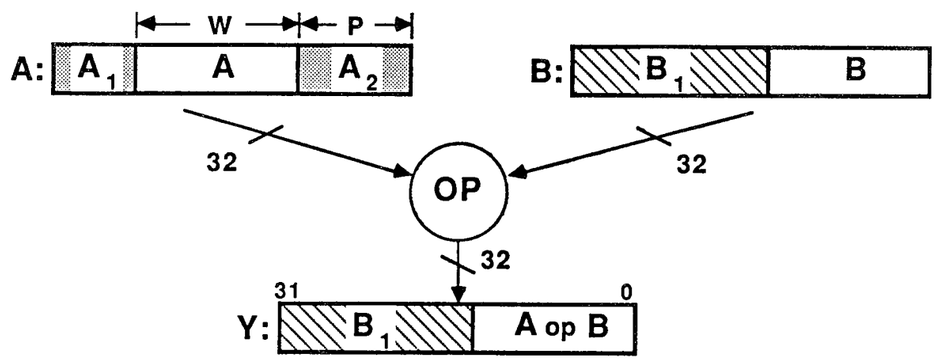
Field Logical Operations

Non-Aligned Fields Case 1:



If position (P5 - P0) ≥ 0, A is LSB aligned
width (W4 - W0) = 1 to 32

Non-Aligned Fields Case 2:



If position (P5 - P0) < 0, B is LSB aligned
width (W4 - W0) = 1 to 32

10. Field Logical Instructions

(Field Format)

	Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
†	PASSF-AL-A	7 3	pass field in A to Y merged with B	B	Y				*		*	
†	PASSF-AL-B	6 F	pass B to Y, set Z if field in B = 0	B	Y				*		*	
†	NOTF-AL-A	7 1	merge complement of field in A with B	B	Y				*		*	
†	NOTF-AL-B	6 E	pass complement of field in B to Y	B	Y				*		*	
†	ORF-AL-A	7 5	OR A and B - aligned field	B	Y				*		*	
†	XORF-AL-A	7 7	XOR A and B - aligned field	B	Y				*		*	
†	ANDF-AL-A	7 9	AND A and B - aligned field	B	Y				*		*	
††	ORF-A	7 4	OR A and B - unaligned field	B	Y				*		*	
††	XORF-A	7 6	XOR A and B - unaligned field	B	Y				*		*	
††	ANDF-A	7 8	AND A and B - unaligned field	B	Y				*		*	
††	NOTF-A	7 0	merge $\overline{\text{field(A)}}$ with B, unaligned	B	Y							
††	PASSF-A	7 2	pass field(A) merged with B, unaligned	B	Y							
††	EXTF-A	7 A	extract field in A	0	Y				*		*	
††	EXTF-B	7 B	extract field in B	0	Y				*		*	
†††	EXTF-AB	7 C	extract field from double word AB	0	Y				*		*	
†††	EXTF-BA	7 D	extract field from double word BA	0	Y				*		*	

Pg. 35 (ED2900C)

† Aligned Field: Fields in A, B both have same position and width.

†† Unaligned Field: + Position: LSB (A), Position (B), ⟹ Position (Y) [Case 1]
- Position: LSB (B), Position (A), ⟹ LSB (Y) [Case 2]

††† Output LSB aligned on Y
+ position : left shift
- position : right shift

Note:

1. Whenever position + width > 32, operation takes place only over the portion of the field up to the end of the word. No wraparound occurs.

Examples of Field Logical Instructions

For all examples, assume STATUS(7:0) is -7 and STATUS (12:8) is 3.

1. 0,PASSF-AL-B,11,20

Pass B to Y and test if B_{20} to B_{30} are all zero. Set Z status if so.

B: 1[00000000000]00000101011100110100

Z set to 1 in this case

2. 3,XORF-A,,

Exclusive-OR bits A_9 - A_7 with bits B_2 - B_0 and output to Y_2 - Y_0. Pass B_{31} - B_3 to Y_{31} - Y_3. Width and position values are obtained from STATUS(12:0).

A: 01101110001001000010111[001]101011
B: 00011100001010001100101001001[001]

$A_{9-7} + B_{2-0}$ = Y: 00011100001010001100101001001[101]

3. 2,NOTF-A,28,

Merge complement of A_{27-0} with B and pass to Y shifted up by seven bits

A: 0110100[11101001011110101101110 11]
B: 10000111001111000100110100[011111]

Y: 00010110100001010010001000011111

Complement first 25 bits of field of A ↖ from B input

FIELD LOGICAL INSTRUCTION EXERCISES

(All operands are 32-bits unless otherwise noted)

Using the Am29300 Example Coding Form, give the Am29332 and Am29334 fields to perform the following functions. Move, logical, single-bit shift, priority, arithmetic, shift and rotate, and bit field instructions may be needed in some solutions.

1. Complement the second byte in R27.

2. AND bits 27-30 of R14 with bits 17-20 of R15 and store the result in bits 7-10 of R16 with all other bits in R16 being the complement of R14.

3. Merge the third byte of R19 with the first byte of R20 and store the result in the first byte of R21.

4. Store 20 bits beginning at bit 20 of R1 in the first 20 bits of R10. Clear rest of R10 to zero.

5. Store 20 bits beginning at bit 20 of the double word R1,R2 in the first 20 bits of R10. Clear rest of R10 to zero.

29300 Example Coding Form

Am29331				Am29332				Am29334			Am29332 Y-Out
OP	Branch or Counter Value	Cond Select	Multl Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF

11. Mask Instruction

(Field Format)

Name	Code	Description	Source for Unselected Bytes	Dest.	Status S	M	L	Z	V	N	C
PASS-MASK	7 F	generate a field mask pattern	0 or 1	Y							

Pg. 37 (ED2900C)

Example:

0, MASK, 8, 10

Note:

1. Negative position inverts mask (i.e., field of 0's in word of 1's)

Am29332 ALU
Pin Description

- **Data Input**

DA_{31}-DA_0	Data input lines for operand A.
PA_3-PA_0	Parity input for operand A on DA-bus (one per byte).
DB_{31}-DB_0	Data input lines for operand B.
PB_3-PB_0	Parity input for operand B on DB-bus (one per byte).

- **Instruction Input**

I_6-I_0	Instruction Inputs
I_8-I_7	Byte width inputs for byte boundary aligned operand instructions. Selects the sources for width and position inputs for variable field bit operands. If I_7 is LOW it selects the width input from pins W_4-W_0. If I_7 is HIGH the width input is selected from the internal width register. Similarly if I_8 is LOW it selects the position inputs from pins P_5-P_0 and if HIGH it selects input from the internal position register.
W_4-W_0	Width input to select the width of a contiguous bit field.
P_5-P_0	Position input to select the position of the least significant bit of a field. Also indicates the amount by which data is to be shifted up (P_5 = LOW) or down (P_1 = HIGH) or rotated.

Am29332
Pin Descriptions
(continued)

- **Data Output**

Y_{31}-Y_0 Data Input/Output Lines.
When $\overline{OE\text{-}Y}$ is LOW and the ALU is in the Master mode, the ALU result is enabled on the Y-bus. When $\overline{OE\text{-}Y}$ is HIGH, the Y-bus is tristated. In Slave mode the Y-bus acts as external data input.

PY_3-PY_0 Parity output for data on Y-bus (one per byte).

- **Status Data**

C, Z, N, V, L When the Register Status pin is LOW, these pins give the carry, zero, negative, overflow and link outputs of the ALU where applicable being executed. When not applicable to the instruction being executed, or when the Register Status pin is HIGH, these pins give the outputs of the carry, zero, negative, overflow and link bits of the internal status register. In SLAVE mode, C, Z, N, V and L become inputs.

- **Status Control**

Register Status Register Status Mode Pin
Selects between ALU status (Register Status = LOW) or register status (Register Status = HIGH) on the C, Z, N, V and L outputs.

Hold When HIGH it inhibits the update of the status and Q registers.

Macro Carry Macro Status Carry Input

Macro Link Macro Status Link Input

Macro/ Micro SEL When HIGH selects macro carry and macro link pins as input instead of micro carry and micro link from the micro-status register.

Am29332
Pin Descriptions
(continued)

- **Control**

$\overline{OE\text{-}Y}$ Output Enable
When $\overline{OE\text{-}Y}$ is HIGH the Y-bus is disabled (tri-stated).

CP Clocks internal registers (status, Q) at the LOW to HIGH transition, provided HOLD input is LOW.

Borrow When HIGH the Carry In and Carry Out are borrows for subtract operations.

- **Error**

Parity-Error When HIGH indicates that a byte-parity error was detected on the DA or DB inputs.

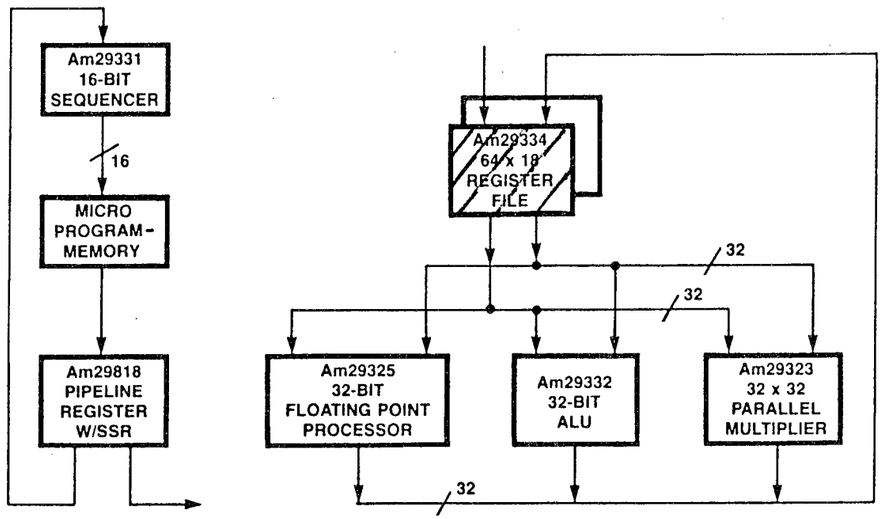
MS-Error Master-Slave Error
When HIGH this signal indicates either: a) two Am29332s are configured as master and slave so the slave can indicate a disagreement with the master or the master can indicate its own ouput driver failures; or b) an Am29332 operates alone (as a master) to indicate output failures.

Slave When HIGH this pin puts the ALU in the slave mode. All output pins (except errors) become input pins and signals on them are compared with the ALU's internally-generated results. When $\overline{OE\text{-}Y}$ is HIGH, the Y_0-Y_{31} and PY_0-PY_3 inputs are ignored. When the SLAVE pin is LOW, the ALU is put in master mode where outputs are generated as normal.

Am29334 Register File

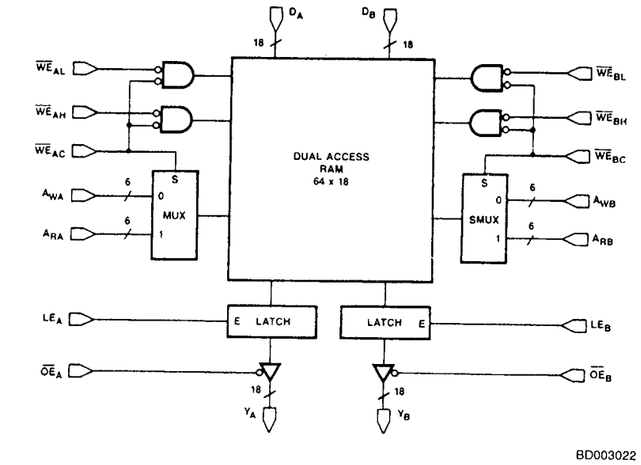
Am29334 Register File

- ECL with TTL I/O
- 24 nsec access (commercial)
- Four ports, dual access (3 or 4 address architecture)
- 64 x 18 organization
- Expandable in either direction
- 120-pin PGA package

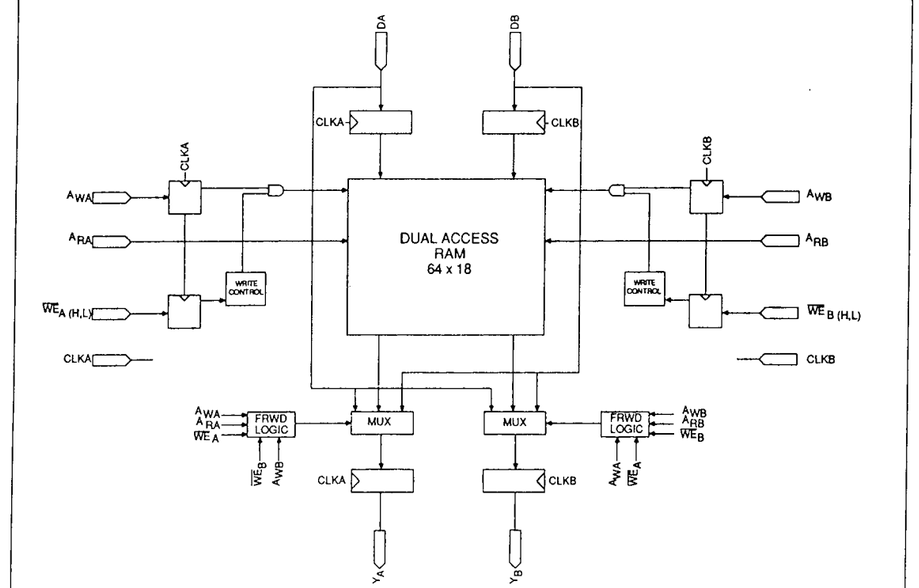


Am29300 Family High Performance System Block Diagram

BLOCK DIAGRAMS



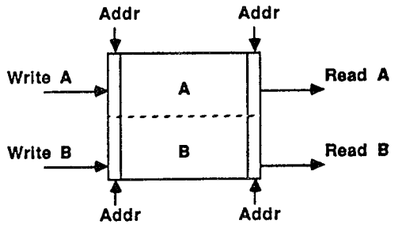
Non-Pipelined Mode



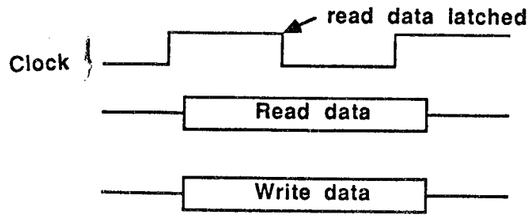
Pipelined Mode

Characteristics

- Each register is 18-bits wide. (Two data bytes plus a parity bit for each byte.)
- Two Read ports (Y_A, Y_B)
  - Each has an output latch enable
  - Each has an output enable
- Two Write ports (D_A, D_B)
  - Each port has three enables:
    - low byte
    - high byte
    - common
- Four Addresses (A_{RA}, A_{WA}, A_{RB}, A_{WB})
- Can independently read two registers and write two registers in same clock cycle.

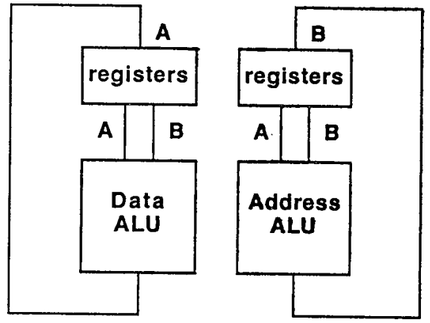


- Can write into one or two bytes of a word
  - Unselected byte in register one-byte writes is unmodified.
- Timing



Am29334 Applications

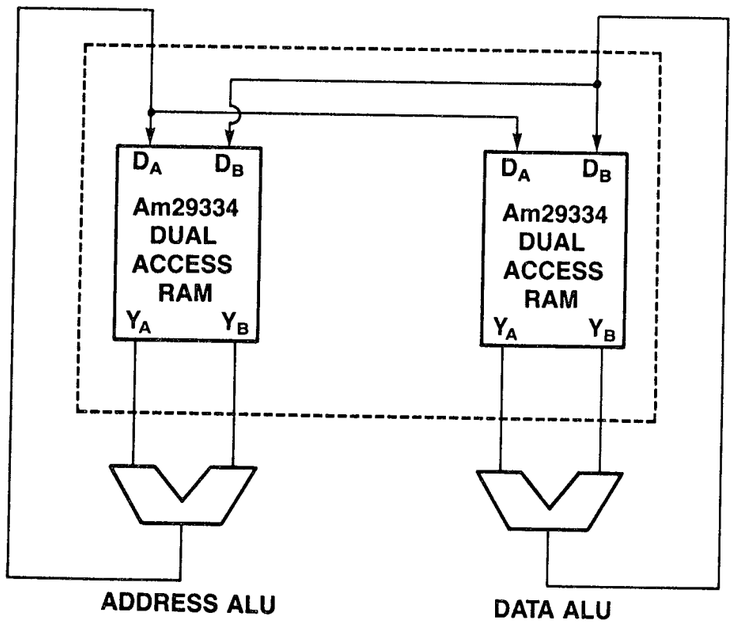
- Consider the need to perform address calculations concurrent with data calculations on 16-bit words.



- Since dynamic addressing is often desired, where a result from the data ALU affects the address of subsequent data, it would be helpful if the two ALUs operated on a common register file.
- Thus, there is a need for 4 simultaneous Read ports and two simultaneous Write ports. To obtain this, use two Am29334's (for 16-bit words) and correct them in parallel.

Note: Upon power-up, write to both Am29334's to initialize them to common values.

RAM WITH 4 READ PORTS AND 2 WRITE PORTS



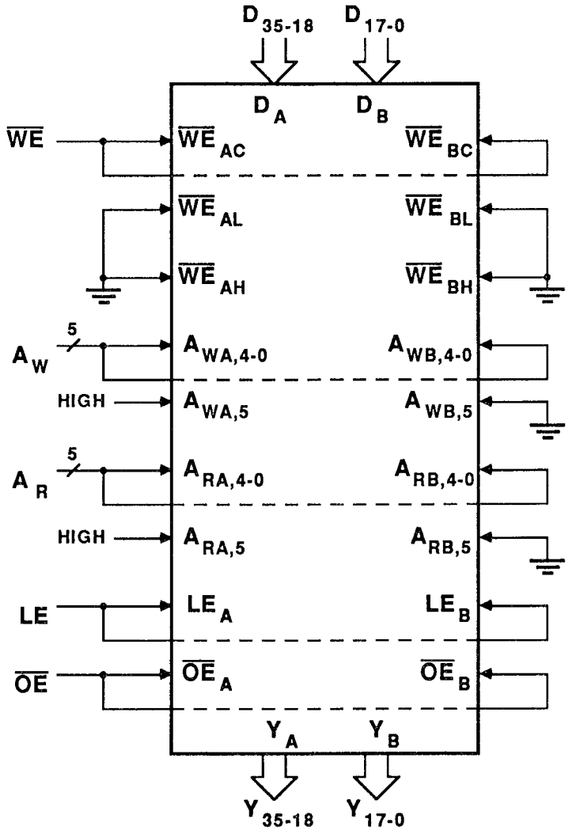
Am29334 Applications

As another example, consider the use of a single 18-bit x 64-word Am29334 as a 36-bit x 32-word register. To do this --

- Store upper half of the 36 bits in the upper half of the 64-words -- address these from the A side.
- Store lower half of the 36 bits in the lower half of the 64 words -- address these from the B side.

$A_B = A_A + 32$ <---------- Tie bit 5 of address for A_A low to address lower 32 words and bit 5 of address A_B high to address upper 32 words.

Note: Dual access is lost!



32 x 36 RAM (Single Access) Using Am29334 64 x 18 Dual Access RAM

Am29334 Pin Description

$A_{RA,0-5}$ **Input**
Read address for Y_A

$A_{RB,0-5}$ **Input**
Read address for Y_B

$Y_{A,0-17}$ **Three-State Output**
Data output A

$Y_{B,0-17}$ **Three-State Output**
Data output B

$A_{WA,0-5}$ **Input**
Write address for D_A

$A_{WB,0-5}$ **Input**
Write address for D_B

$D_{A,0-17}$ **Input**
Data input A

$D_{B,0-17}$ **Input**
Data input B

LE_A **Input**
Latch enable A

LE_B **Input**
Latch enable B

OE_A **Input**
Output enable for Y_A

OE_B **Input**
Output enable for Y_B

WE_{AC} **Input**
Common write enable A

WE_{AL} **Input**
Low byte write enable A (bits 0-8)

WE_{AH} **Input**
High byte write enable A (bits 9-17)

WE_{BC} **Input**
Common write enable B

WE_{BL} **Input**
Low byte write enable B (bits 0-8)

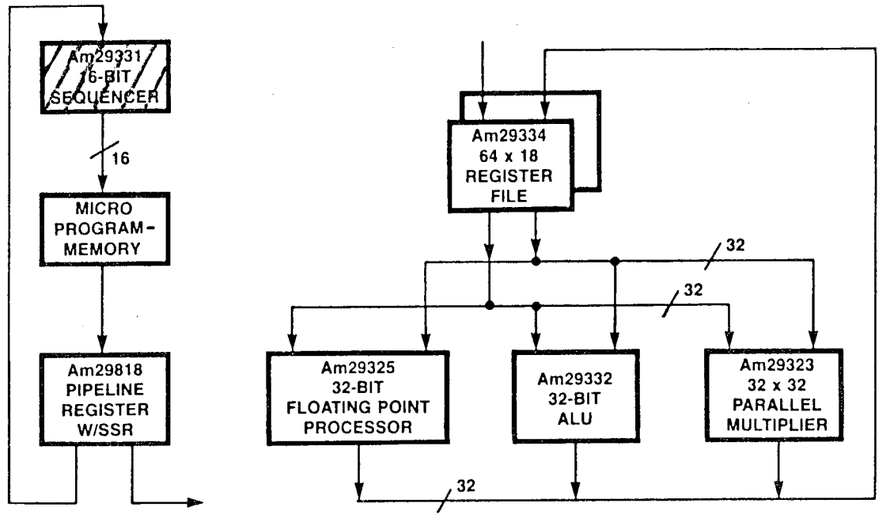
WE_{BH} **Input**
High byte write enable B (bits 9-17)

14 power pins

Am29331 16-bit Sequencer

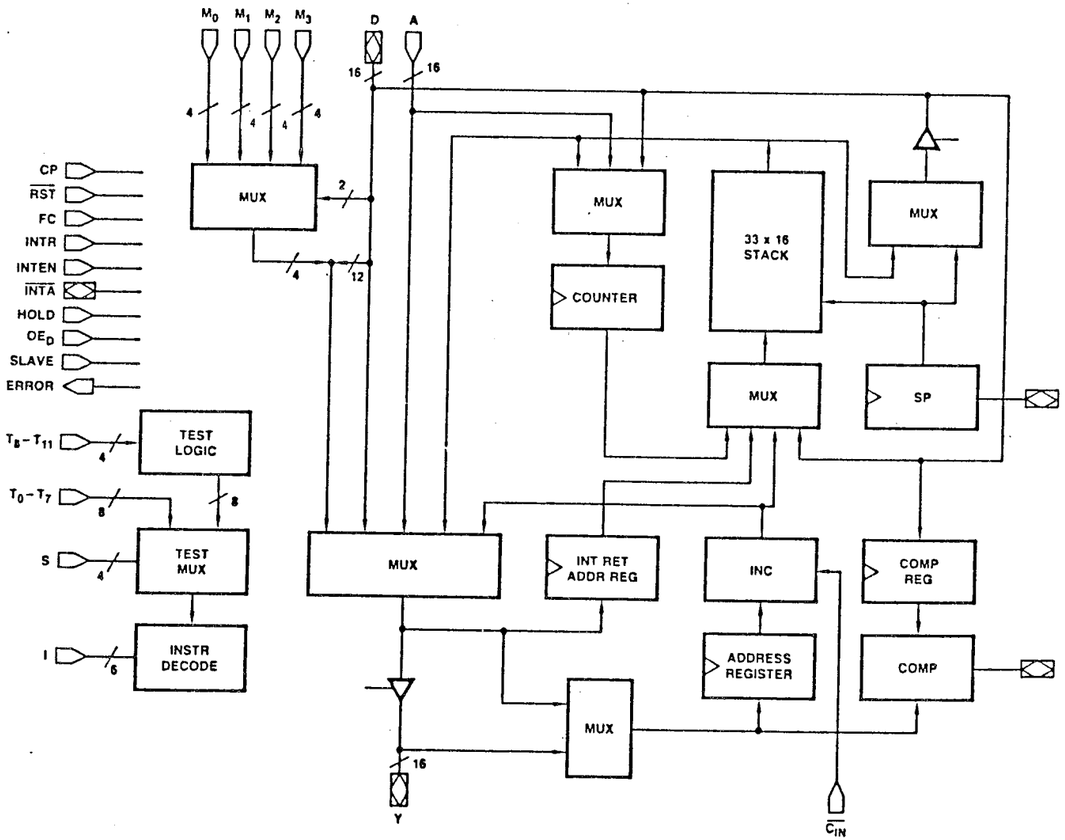
Am29331 16-bit Sequencer

- Addresses 64K words of microcode
- ECL with TTL I/O
- Breakpoint logic
- Interruptible at microinstruction boundary
- Micro-trap support (allows re-execution of prior instruction)
- 120-pin PGA package
- Not cascadable



Am29300 Family High Performance System Block Diagram

Am29331



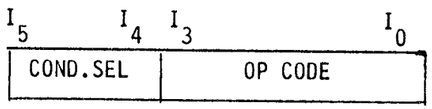
Am29331

16-bit Interruptible Sequencer

- Two separate 16-bit Branch Address Buses
  - Bus A often used for input addresses from mapping ROM
  - Bus D used for iteration count, branch address, or stack access
- Four sets of 4-bit Multiway Inputs
  - Useful for table lookup
- 33-level deep stack
  - Supports nested interrupts, loops, and subroutining
  - Supports diagnostics
- Master/Slave Error Checking

Am29331

Instruction Format



I_5	I_4	COND. SEL
0	0	conditional (normal) instructions
0	1	conditional inst. with complemented test
1	0	unconditional sequence control
1	1	special functions with implicit continue

$I_5 \equiv \overline{CCEN}$ (condition code override)

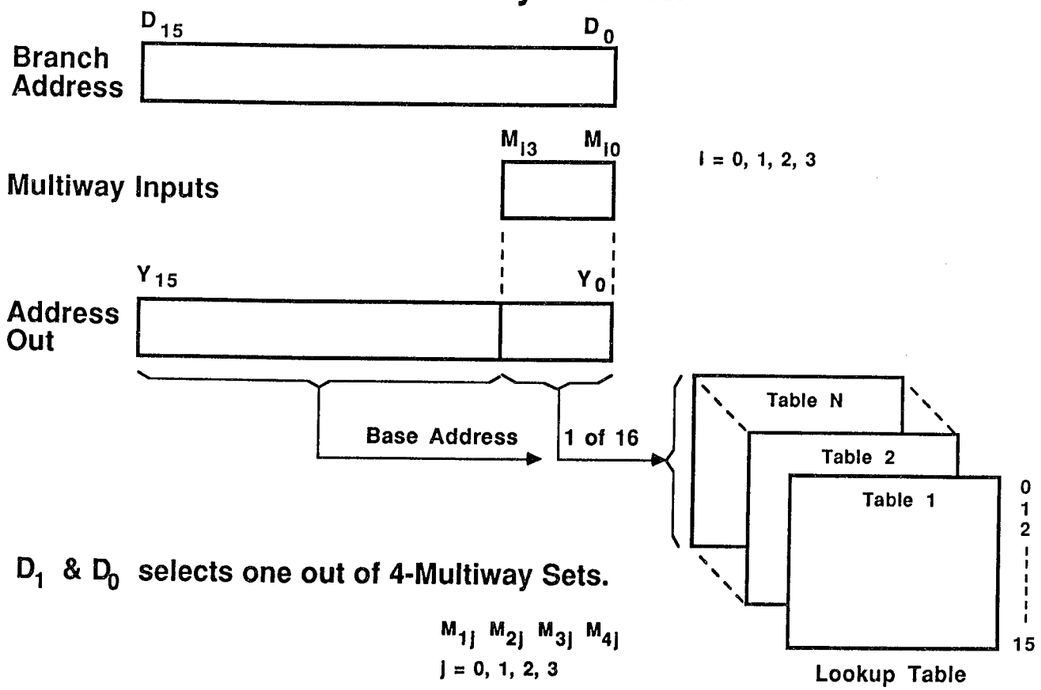
Am29331

Test Conditions

	$S_3 - S_0$	Test	Intended Use
Direct Input	0-7	$T_0 - T_7$	External test conditions
	8	T_8	Carry
	9	T_9	Negative
	10	T_{10}	Overflow
	11	T_{11}	Zero
Logically Generated	12	$T_8 + T_{11}$	C + Z (A≤B, Unsigned No., Borrow)
	13	$\overline{T_8} + T_{11}$	$\overline{C}$ + Z (A≤B, Unsigned No., Carry)
	14	$T_9 \oplus T_{10}$	N ⊕ V (A<B, 2's Compl. No.)
	15	$(T_9 \oplus T_{10}) + T_{11}$	(N ⊕ V)+Z (A≤B, 2's Compl. No.)

Am29331

Multiway Branch



D_1 & D_0 selects one out of 4-Multiway Sets.

M_{1j} M_{2j} M_{3j} M_{4j}
j = 0, 1, 2, 3

Pg. 55 (ED2900C)

Am29331

Interrupts and Traps

- An interrupt is the result of an external event.
  - Complete the current instruction in the pipeline register
  - Execute interrupt routine whose address is forced at Y input
  - Return to the main program
- A trap is caused by the current instruction which must be aborted.
  - Abort the current instruction in the pipeline register
  - Execute trap routine whose address is forced at Y input
  - Return to the aborted instruction and re-execute

Note: Interrupt or trap routine address is usually supplied by an Am29114 Interrupt Controller at the bidirectional Y input.

Am29331 - Interrupt Cycle 1

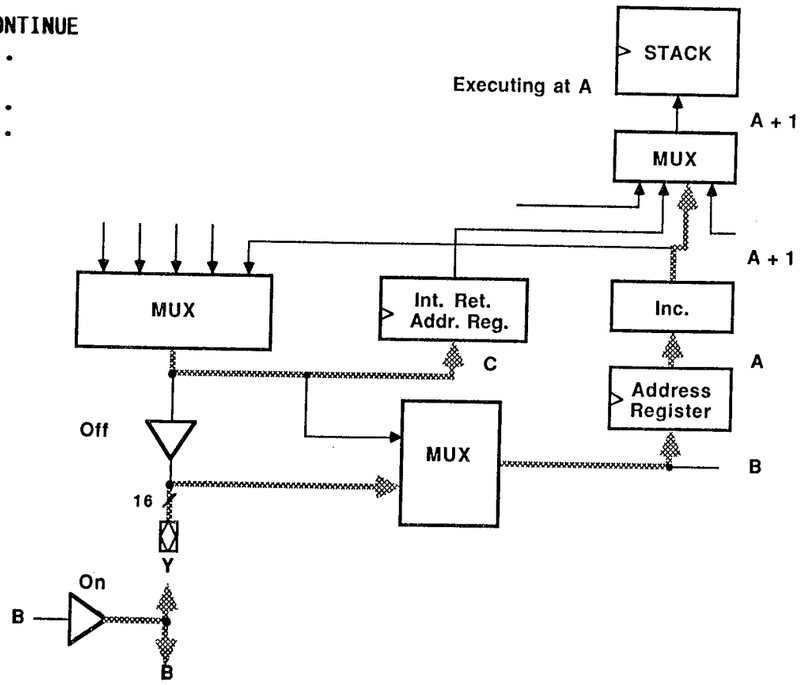
While executing the instruction at A,
the sequence is interrupted and
directed to B.

Executing at A

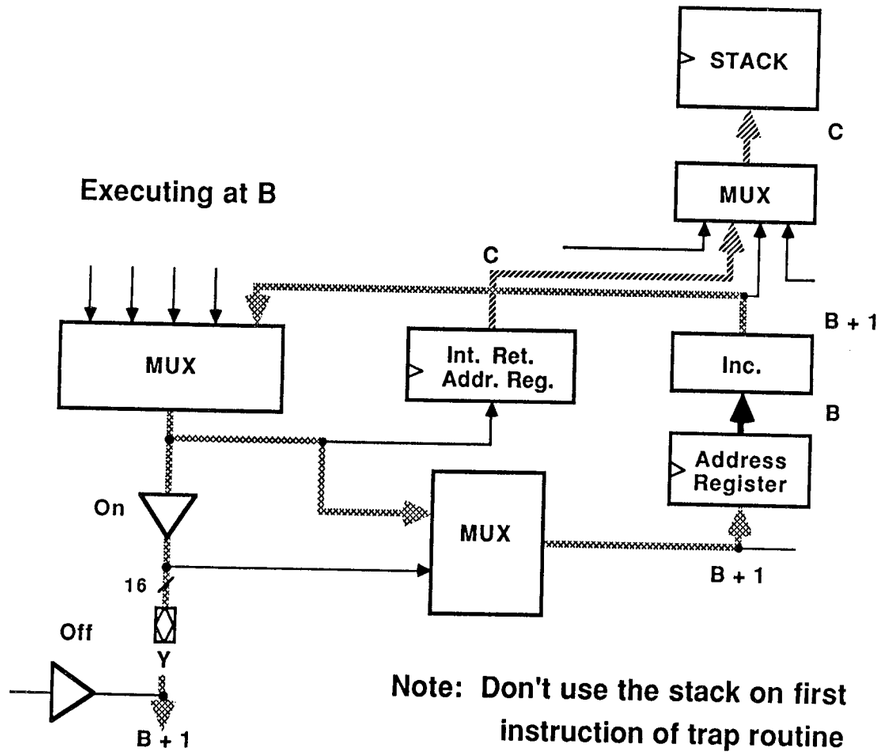
```
A  : CALL C
A+1: ...

B  : CONTINUE
B+1: ...

C  : ...
C+1: ...
```



Am29331 - Interrupt Cycle 2



Note: Don't use the stack on first instruction of trap routine

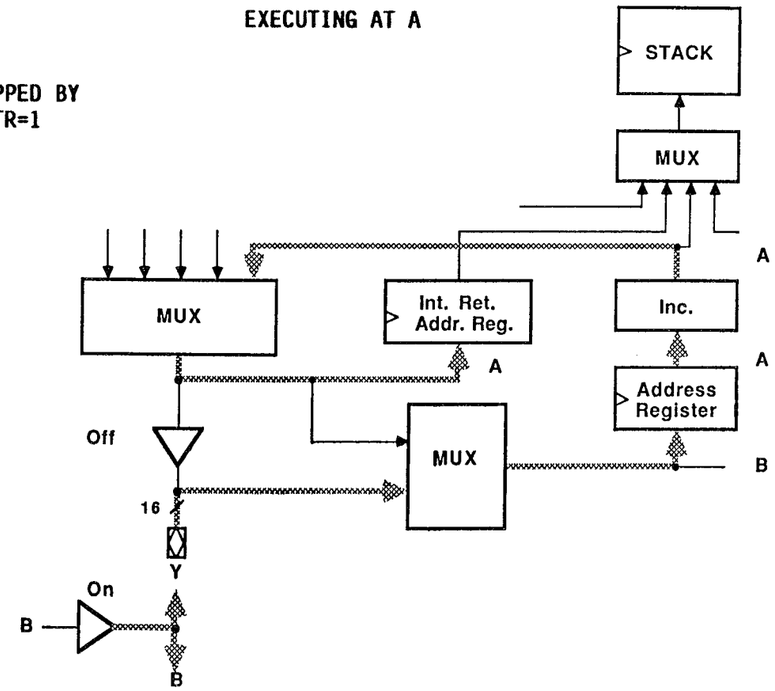
Pg. 61A (ED2900C)

Am29331 - Traps Cycle 1

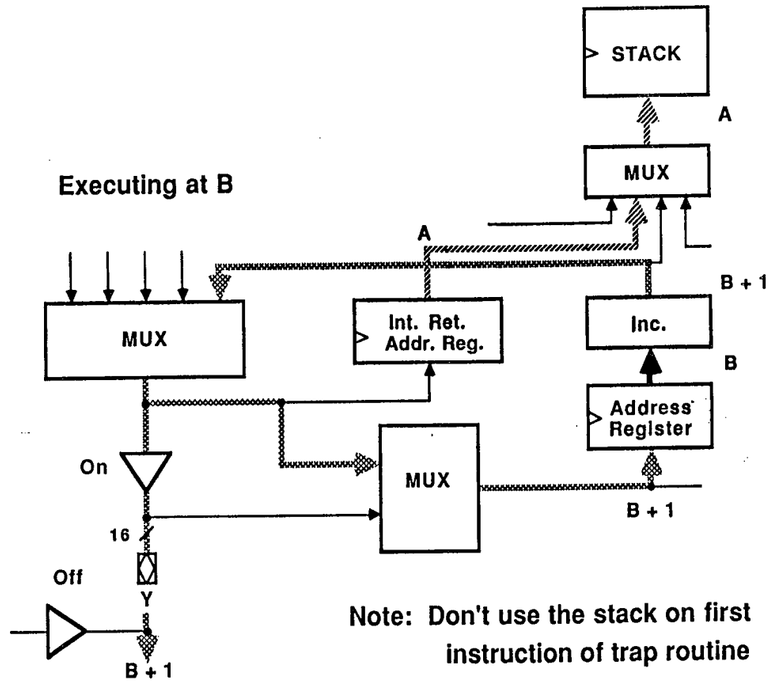
A trap occurs at the instruction A,
and the sequence is directed to B.

A : INSTRUCTION TRAPPED BY
$\overline{FC}$=1, $\overline{C_{IN}}$=1, INTR=1
A+1: ...

B : CONTINUE
B+1: ...



Am29331 - Traps Cycle 2



Note: Don't use the stack on first instruction of trap routine

Am29331

Structured Microprogramming Support

High Level Language	Am29331 Instructions
1. REPEAT : : UNTIL CC	LOOP : : END LOOP, NOT CC
2. WHILE CC DO : : : END WHILE L:	LOOP IF NOT CC, THEN EXIT L : : : END LOOP L:
3. LOOP : : IF CC, THEN EXIT : END LOOP L:	LOOP : : IF CC, THEN EXIT L : END LOOP L:

Am29331

Significant Features

- **Forced Continue**
  - An external input (FC) which overrides instruction inputs I_0-I_5 with a CONTINUE instruction
  - Useful for initializing writable control store
- **Reset**

  External input which resets controller
  - Overrides instruction input and forced continue
  - Outputs 0 at controller output Y
  - Forces EQUAL output to Low
  - Disables address comparison
  - Initializes stack pointer to 0
  - Disregards a potential interrupt request
- **Hold Mode**

  Sequencer suspends operation after current cycle when HOLD goes active
  - All outputs are disabled
  - Internal state remains unchanged
  - D-bus disabled

Am29331

Significant Features

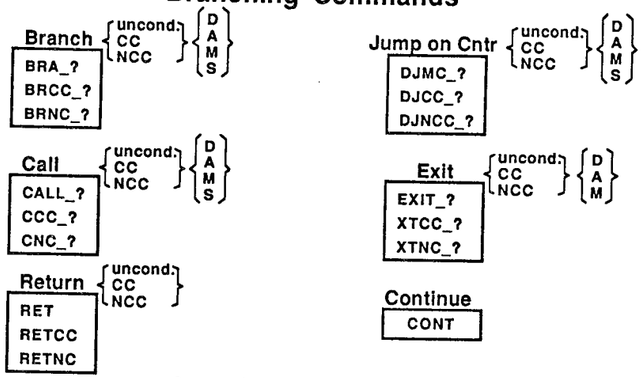
- **Stack**
  - 33-word deep by 16-bits wide
  - Holds retrun addresses, loop addresses, and counter values
  - Access to the stack via the D-bus may be used for context switching, stack extension, or diagnostics
  - Stack extension requires popping stack to external memory and loading stack with new data using PUSH operations
  - Stack pointer is a modulo-64 counter
    - -- Overflow occurs when pointer "wraps around" from 63 to 0. Bottom of stack is overwritten in this case.
    - -- Item is undefined on underflow
  - Stack pointer value available on D-bus for all instructions except POP_D
  - A-FULL HIGH when SP≥28
- **Nested Loop**
  - Push counter value to stack and load new counter value to enter nested loop
  - Pop counter value to counter restart outer loop

Am29331 16-bit Sequencer

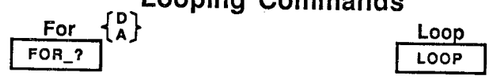
Commands

Am29331 Command Summary

Branching Commands

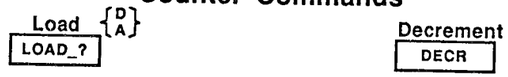


Looping Commands



Stack Commands

Counter Commands



Breakpoint Commands

D: D Input
A: A Input
M: D concatenated M (Multiway)
S: Stack (T05)

CC = CC Is High
NCC = CC Is Low

Pg. 63A (2900C)

Am29331 Commands

I_4 OR I_5 = Low

I_0-I_3	Instruction	Cond.: False Y	Cond.: False Stack	Cond.: True Y	Cond.: True Stack	Counter	Comp.	D-Mux
0	Goto D	PC	–	D	–	–	–	SP
1	Call D	PC	–	D	Push PC	–	–	SP
2	Exit D	PC	–	D	Pop	–	–	SP
3	End for D, C ≠ 1	PC	–	D	–	C←C – 1	–	SP
	End for D, C = 1	PC	–	PC	–	C←C – 1	–	SP
4	Goto A	PC	–	A	–	–	–	SP
5	Call A	PC	–	A	Push PC	–	–	SP
6	Exit A	PC	–	A	Pop	–	–	SP
7	End for A, C ≠ 1	PC	–	A	–	C←C – 1	–	SP
	End for A, C = 1	PC	–	PC	–	C←C – 1	–	SP
8	Goto M	PC	–	D:M	–	–	–	SP
9	Call M	PC	–	D:M	Push PC	–	–	SP
10	Exit M	PC	–	D:M	Pop	–	–	SP
11	End for M, C ≠ 1	PC	–	D:M	–	C←C – 1	–	SP
	End for M, C = 1	PC	–	PC	–	C←C – 1	–	SP
12	End Loop	PC	Pop	TOS	–	–	–	SP
13	Call Coroutine	PC	–	TOS	TOS←PC	–	–	SP
14	Return	PC	–	TOS	Pop	–	–	SP
15	End for, C ≠ 1	PC	Pop	TOS	–	C←C – 1	–	SP
	End for, C = 1	PC	Pop	PC	Pop	C←C – 1	–	SP

Cond. = (Test[S] or I_5) XOR I_4

: = Concatenation
C = Counter

I_4 = I_5 = High

I_0-I_3	Instruction	Y	Stack	Counter	Comp.	D-Mux
0	Continue	PC	–	–	–	SP
1	For D	PC	Push PC	C←D	–	SP
2	Decrement	PC	–	C←C – 1	–	SP
3	Loop	PC	Push PC	–	–	SP
4	Pop D	PC	Pop	–	–	TOS
5	Push D	PC	Push D	–	–	SP
6	Reset SP	PC	SP←0	–	–	SP
7	For A	PC	Push PC	C←A	–	SP
8	Pop C	PC	Pop	C←TOS	–	SP
9	Push C	PC	Push C	–	–	SP
10	Swap	PC	TOS←C	C←TOS	–	SP
11	Push C Load D	PC	Push C	C←D	–	SP
12	Load D	PC	–	C←D	–	SP
13	Load A	PC	–	C←A	–	SP
14	Set	PC	–	–	R←D, Enable	SP
15	Clear	PC	–	–	Disable	SP

I_4 = I_5 = HIGH; R = Comp. Register

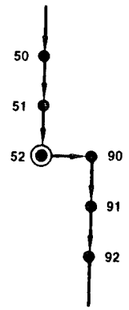
INSTRUCTION SET DEFINITION

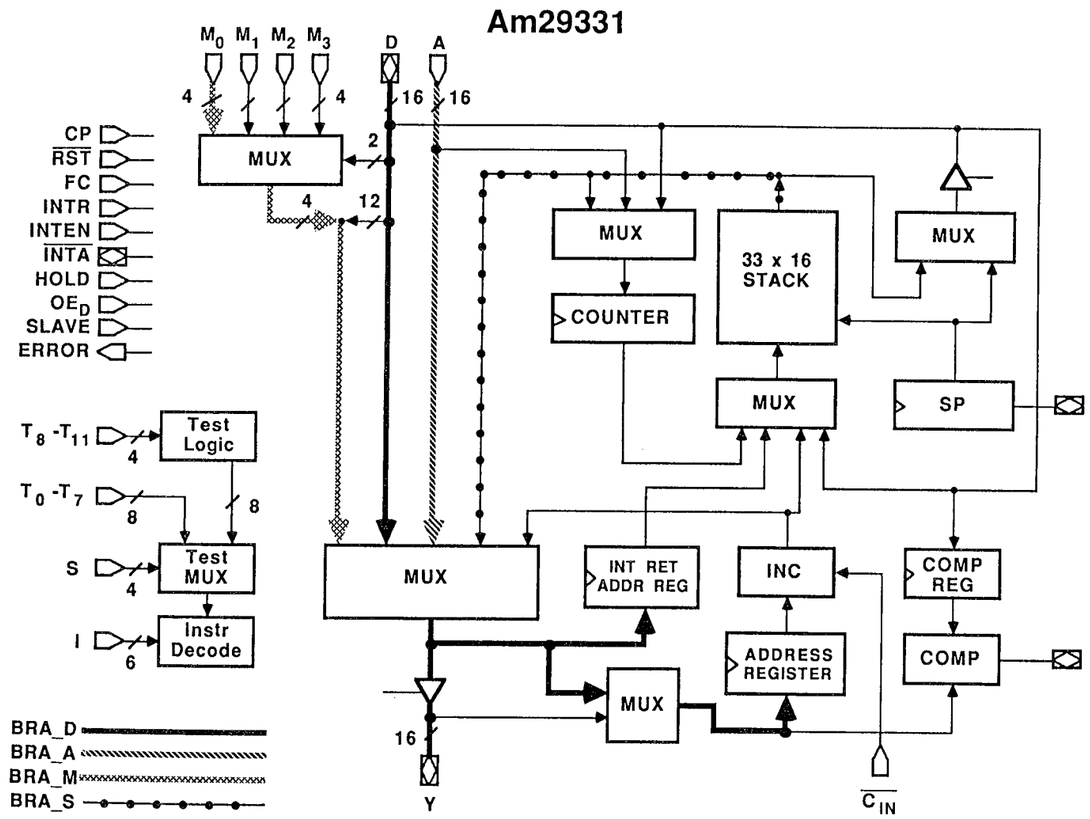
Legend: ● = Other instruction
◉ = Instruction being described
P = Test pass
F = Test fail
◯ = Register in part

UNCONDITIONAL BRANCH

Opcode (I_5-I_0)	Mnemonics	Description
32	BRA_D	Go to D. Unconditional branch to the address specified by the D inputs.
36	BRA_A	Go to A. Unconditional branch to the address specified by the A inputs.
40	BRA_M	Go to M. Unconditional branch to the address specified by the D inputs catenated with the multiway M inputs.
44	BRA_S	Go to TOS. Unconditional branch to the address on the top of the stack. Also End Loop when used to terminate WHILE... ENDWHILE loops.

GO TO { D, A, M, S }

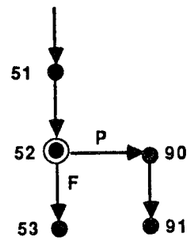


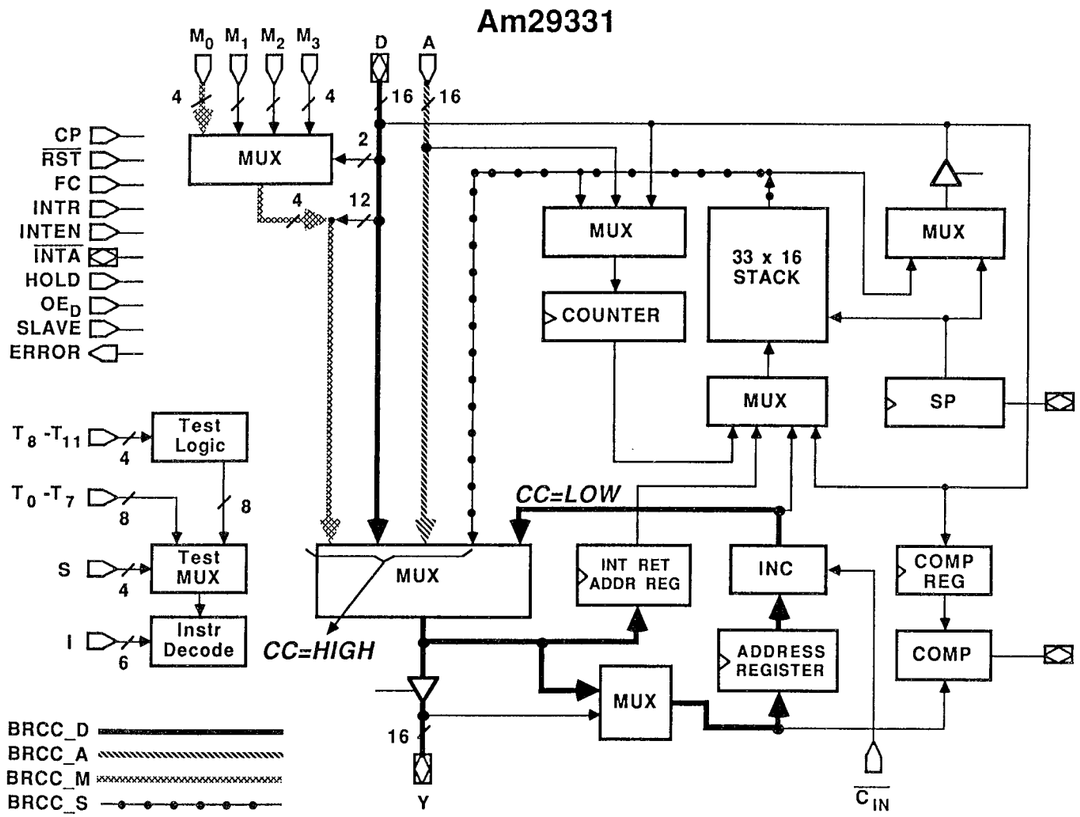


CONDITIONAL BRANCH (NORMAL)

0	BRCC_D	If CC is HIGH then branch to the address specified by the D inputs else continue.
4	BRCC_A	If CC is HIGH then branch to the address specified by the A inputs else continue.
8	BRCC_M	If CC is HIGH then branch to the address specified by the D inputs catenated with the multiway M inputs else continue.
12	BRCC_S	If CC is HIGH then branch to the address on the top of the stack else pop the stack and continue. Also End Loop when used to terminate REPEAT...UNTIL loops.

IF (CC = CONDITION) THEN GO TO { D, A, M, S }

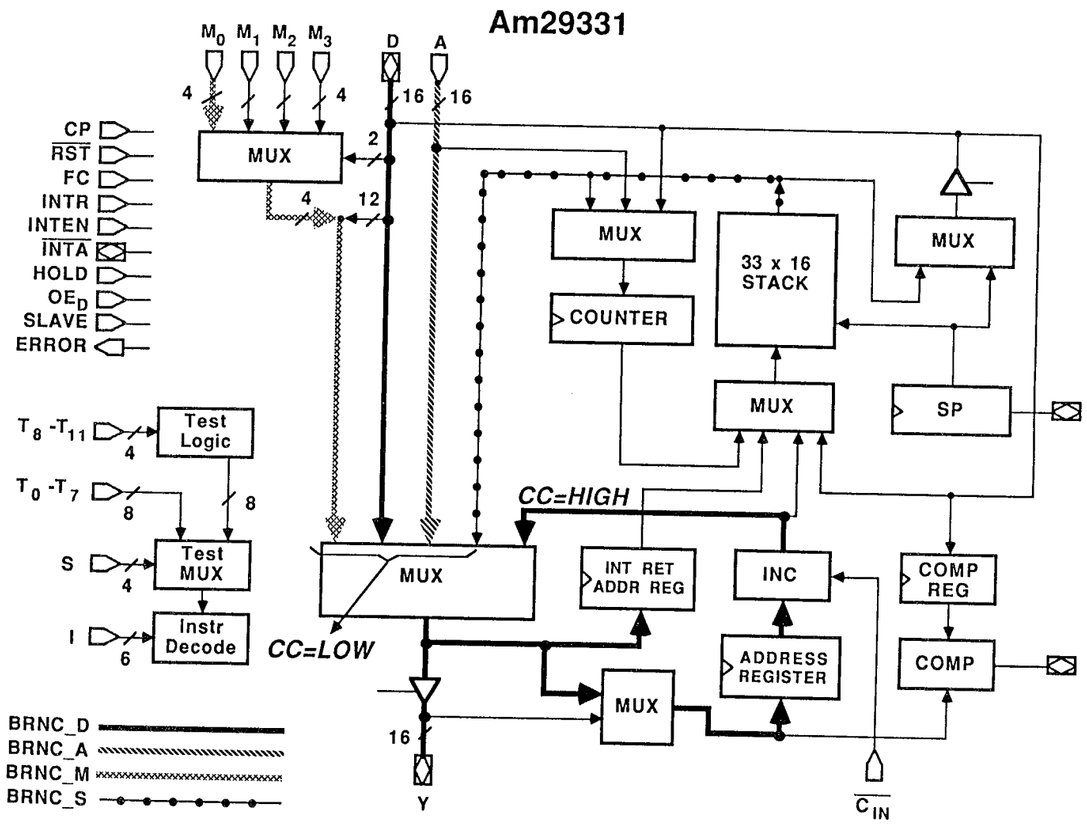




CONDITIONAL BRANCH (INVERSE)

16	BRNC_D	If CC is LOW then branch to the address specified by the D inputs else continue.
20	BRNC_A	If CC is LOW then branch to the address specified by the A inputs else continue.
24	BRNC_M	If CC is LOW then branch to the address specified by the D inputs catenated with the multiway M inputs else continue.
28	BRNC_S	If CC is LOW then branch to the address on the top of the stack else pop the stack and continue. Also End Loop when used to terminate REPEAT...UNTIL loops.

IF (CC ≠ CONDITION) THEN GO TO { D, A, M, S }



BRANCH ON COUNTER

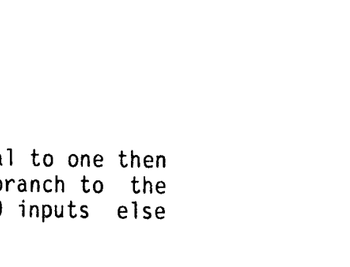
35 DJMP_D — If the counter is not equal to one then decrement the counter and branch to the address specified by the D inputs else continue.

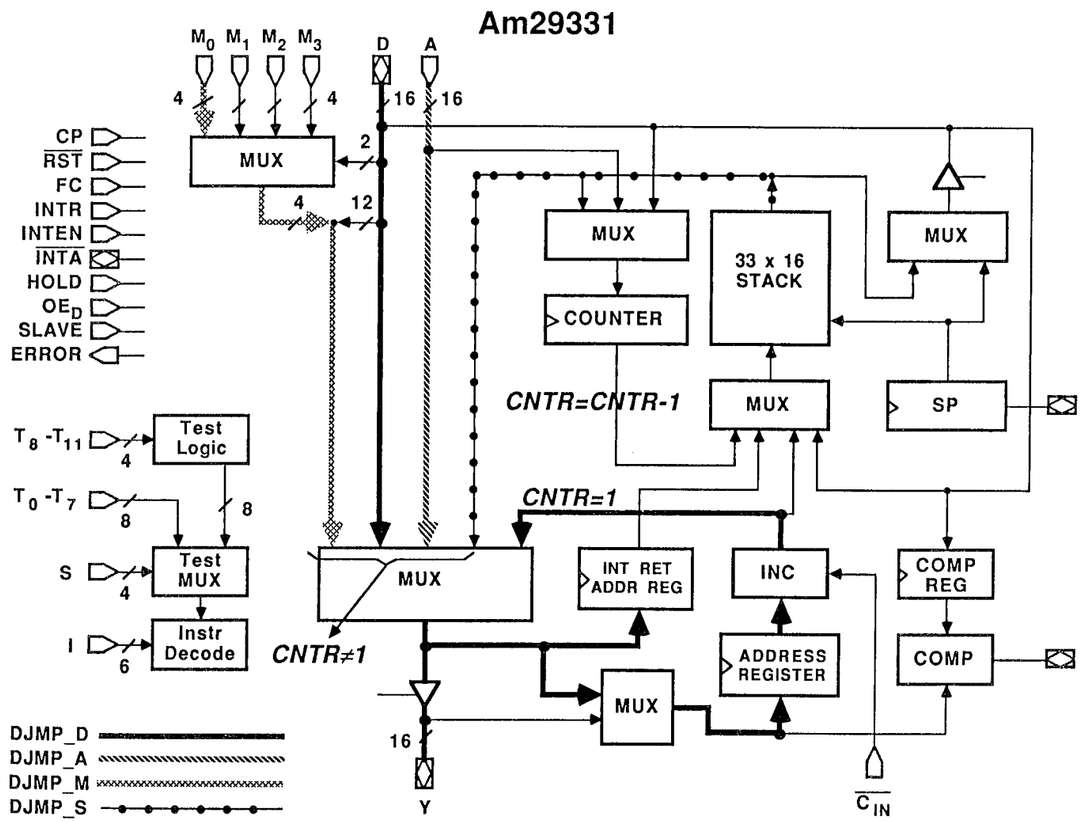
39 DJMP_A — If the counter is not equal to one then decrement the counter and branch to the address specified by the A inputs else continue.

43 DJMP_M — If the counter is not equal to one then decrement the counter and branch to the address specified by the D inputs catenated with the multiway M inputs else continue.

47 DJMP_S — If the counter is not equal to one then decrement the counter and branch to the address on the top of the stack else decrement the counter, pop the stack and continue.

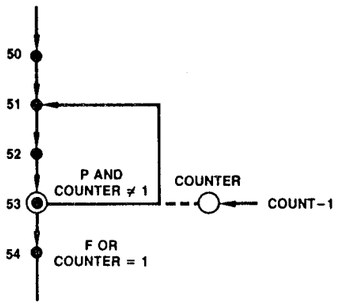
```
IF (COUNTER ≠ 1) THEN
        DECREMENT COUNTER;
        GO TO { D, A, M, S } ;
    ELSE CONTINUE;
```

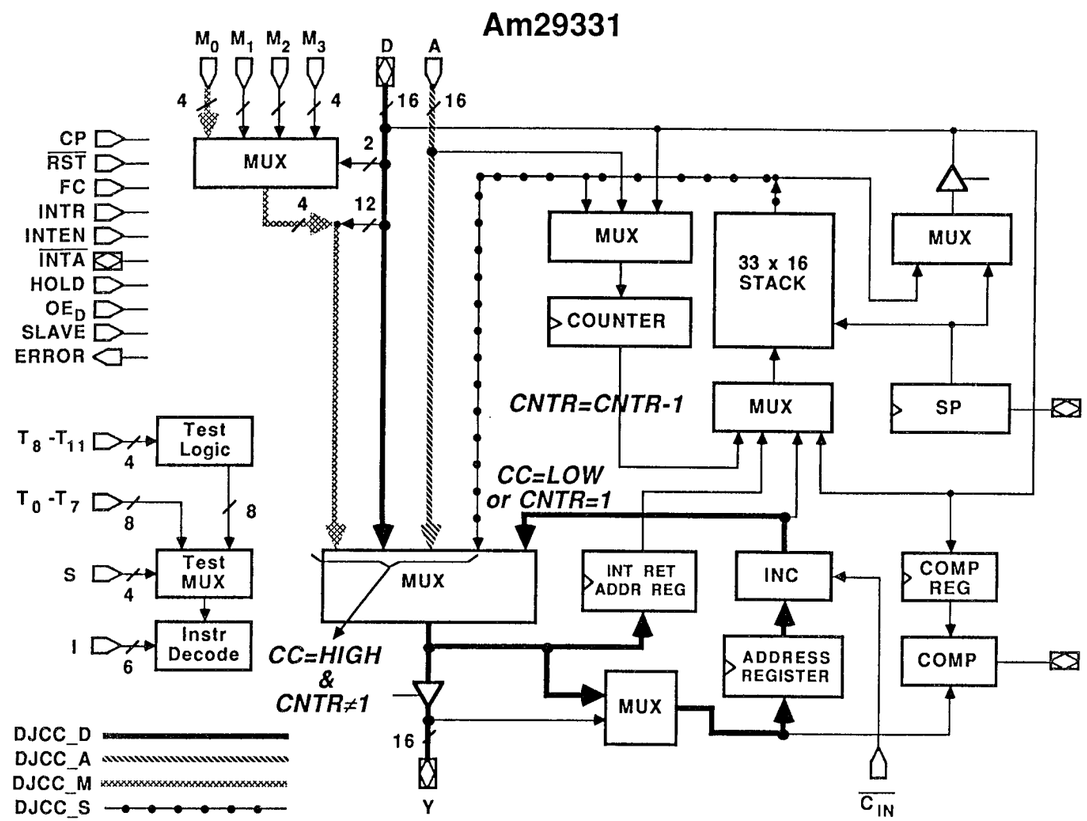




BRANCH ON NORMAL CONDITION AND COUNTER

3	DJCC_D	If CC is HIGH and the counter is not equal to one then decrement the counter and branch to the address specified by the D inputs else decrement the counter and continue.
7	DJCC_A	If CC is HIGH and the counter is not equal to one then decrement the counter and branch to the address specified by the A inputs else decrement the counter and continue.
11	DJCC_M	If CC is HIGH and the counter is not equal to one then decrement the counter and branch to the address specified by the D inputs catenated with the multiway M inputs else decrement the counter and continue.
15	DJCC_S	If CC is HIGH and the counter is not equal to one then decrement the counter and branch to the address on the top of the stack else decrement the counter, pop the stack and continue.

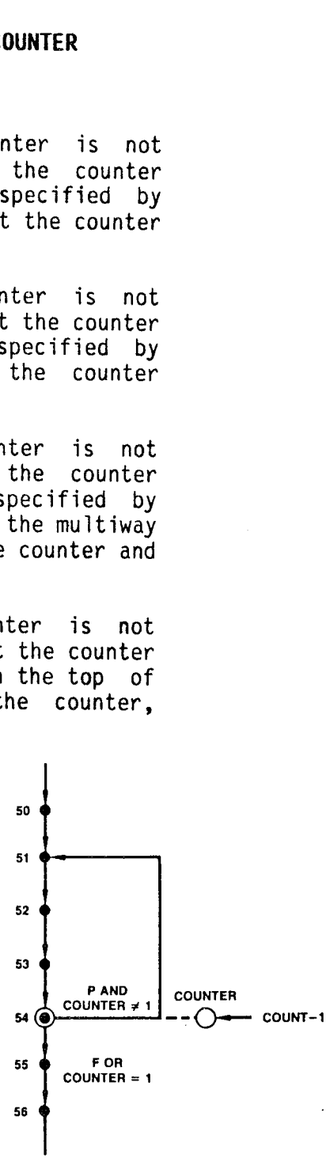


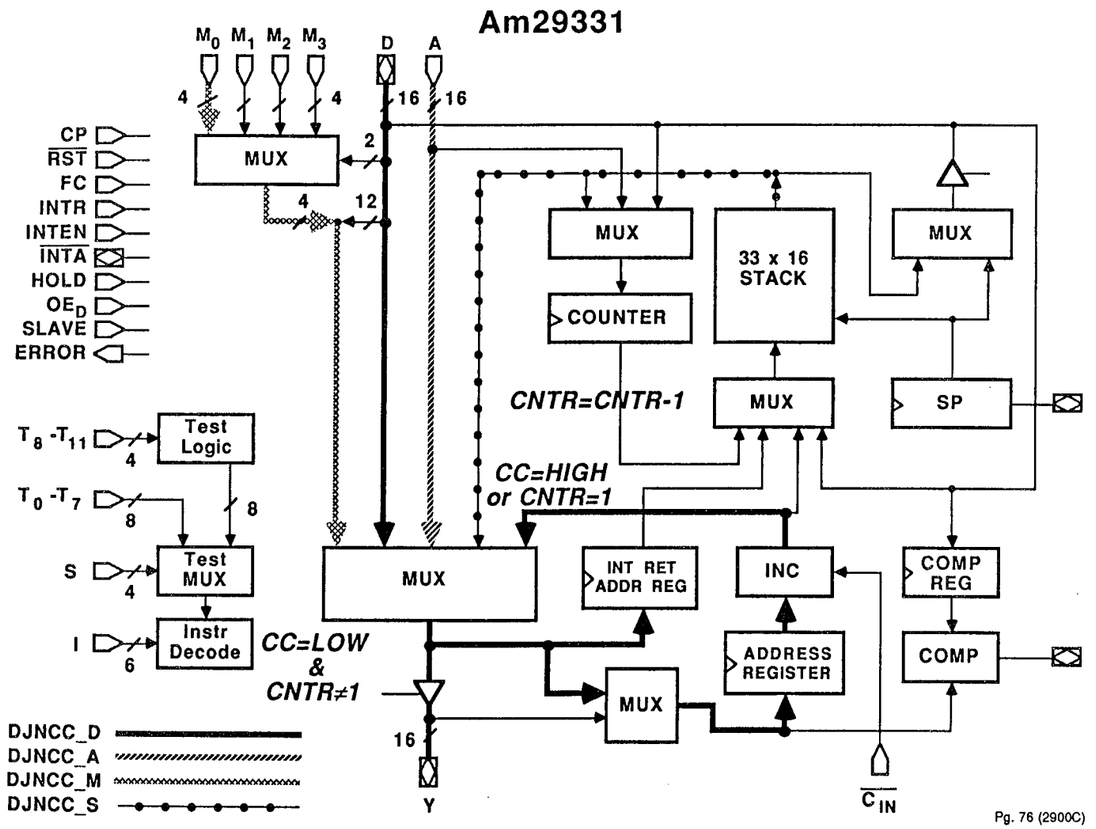


BRANCH ON INVERSE CONDITION AND COUNTER

19	DJNCC_D	If CC is LOW and the counter is not equal to one then decrement the counter and branch to the address specified by the D inputs else decrement the counter and continue.
23	DJNCC_A	If CC is LOW and the counter is not equal to one then decrement the counter and branch to the address specified by the A inputs else decrement the counter and continue.
27	DJNCC_M	If CC is LOW and the counter is not equal to one then decrement the counter and branch to the address specified by the D inputs catenated with the multiway M inputs else decrement the counter and continue.
31	DJNCC_S	If CC is LOW and the counter is not equal to one then decrement the counter and branch to the address on the top of the stack else decrement the counter, pop the stack and continue.

```
IF (CC ≠ CONDITION) AND (COUNTER ≠ 1)
   THEN DECREMENT COUNTER:
        GO TO { D }
              { A }   ;
              { M }
              { S }
   ELSE CONTINUE;
```

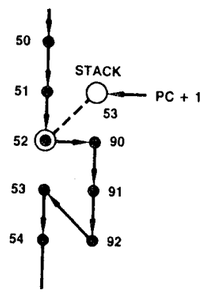


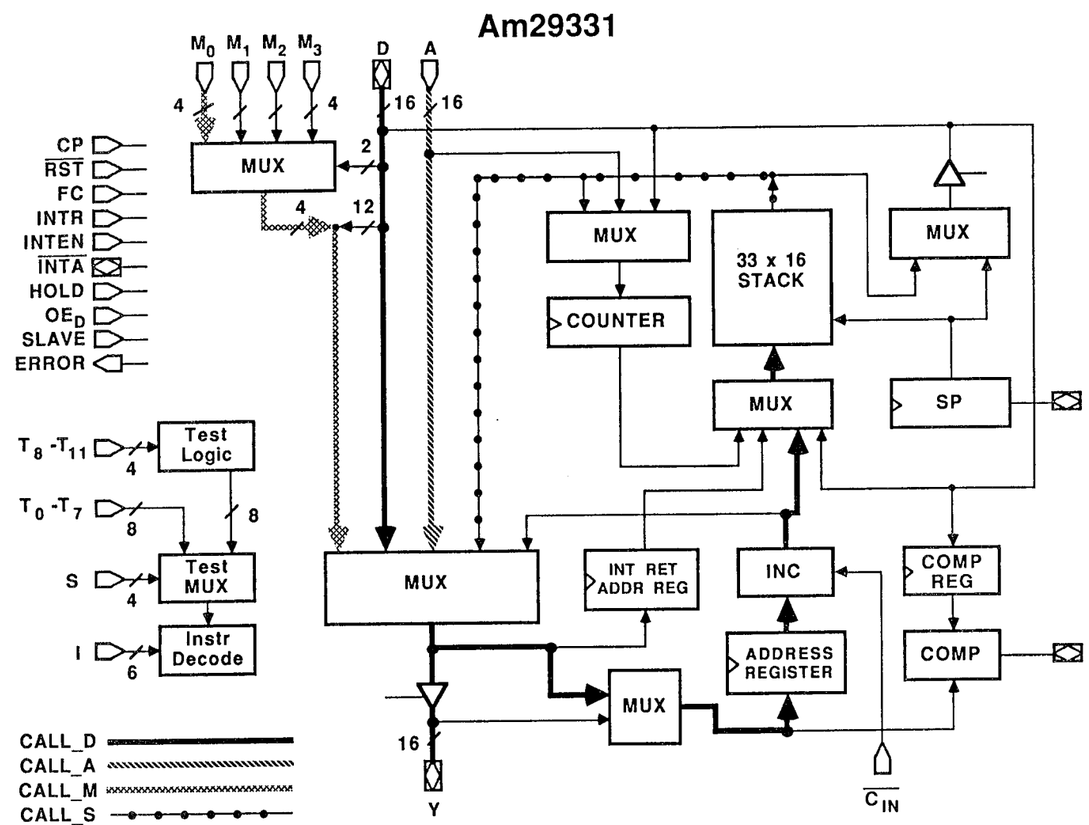


Pg. 76 (2900C)

CALL SUBROUTINE

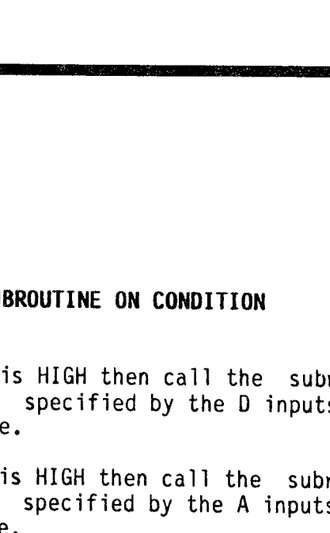
33	CALL_D	Call D. Unconditional branch to the subroutine address specified by the D inputs and push the PC on the stack.
37	CALL_A	Call A. Unconditional branch to the subroutine address specified by the A inputs and push the PC on the stack.
41	CALL_M	Call M. Unconditional branch to the subroutine address specified by the D inputs catenated with the multiway M inputs and push the PC on the stack.
45	CALL_S	Call TOS. Exchange PC and TOS. Also call coroutine.

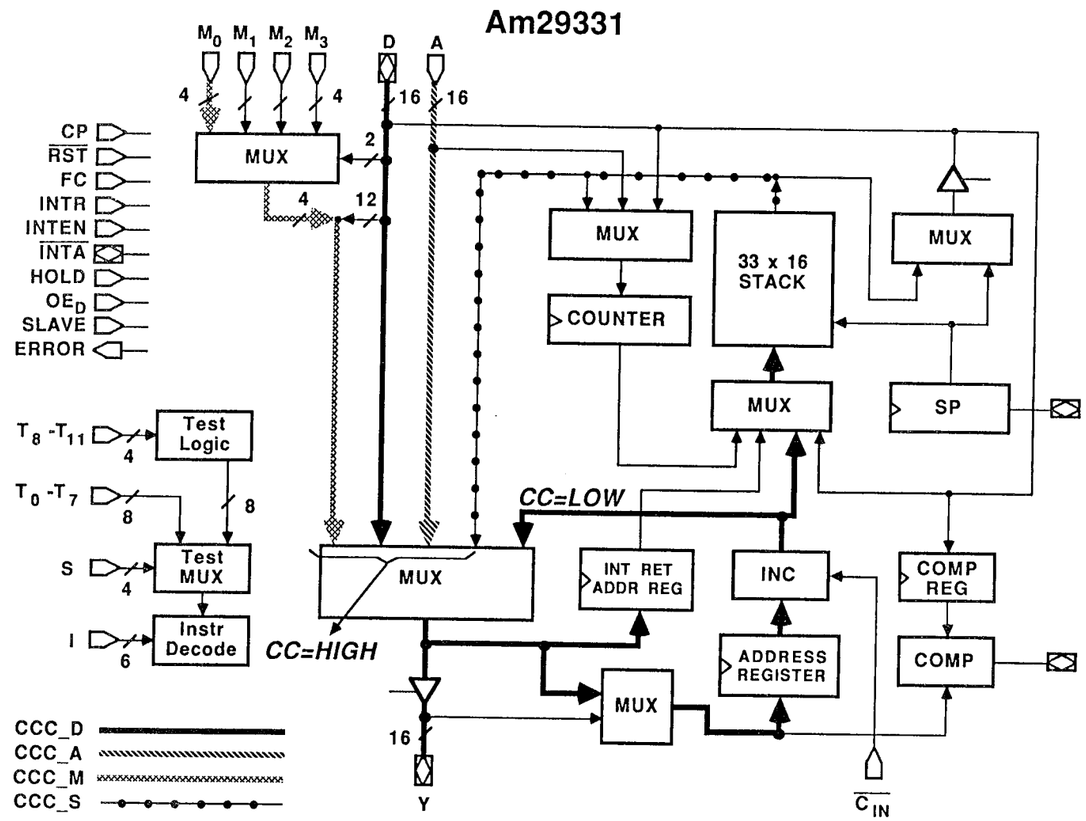




CALL SUBROUTINE ON CONDITION

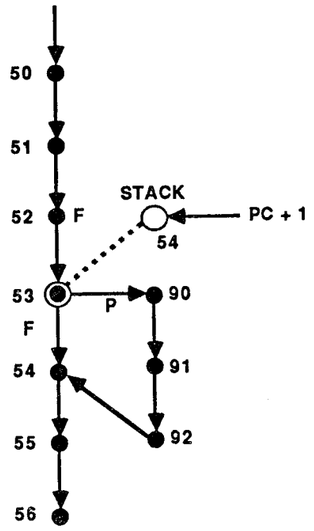
1	CCC_D	If CC is HIGH then call the subroutine address specified by the D inputs else continue.
5	CCC_A	If CC is HIGH then call the subroutine address specified by the A inputs else continue.
9	CCC_M	If CC is HIGH then call the subroutine address specified by the D inputs catenated with the multiway M inputs else continue.
13	CCC_S	If CC is HIGH then call the address on the top of the stack else continue. Also used for conditional coroutine calls.

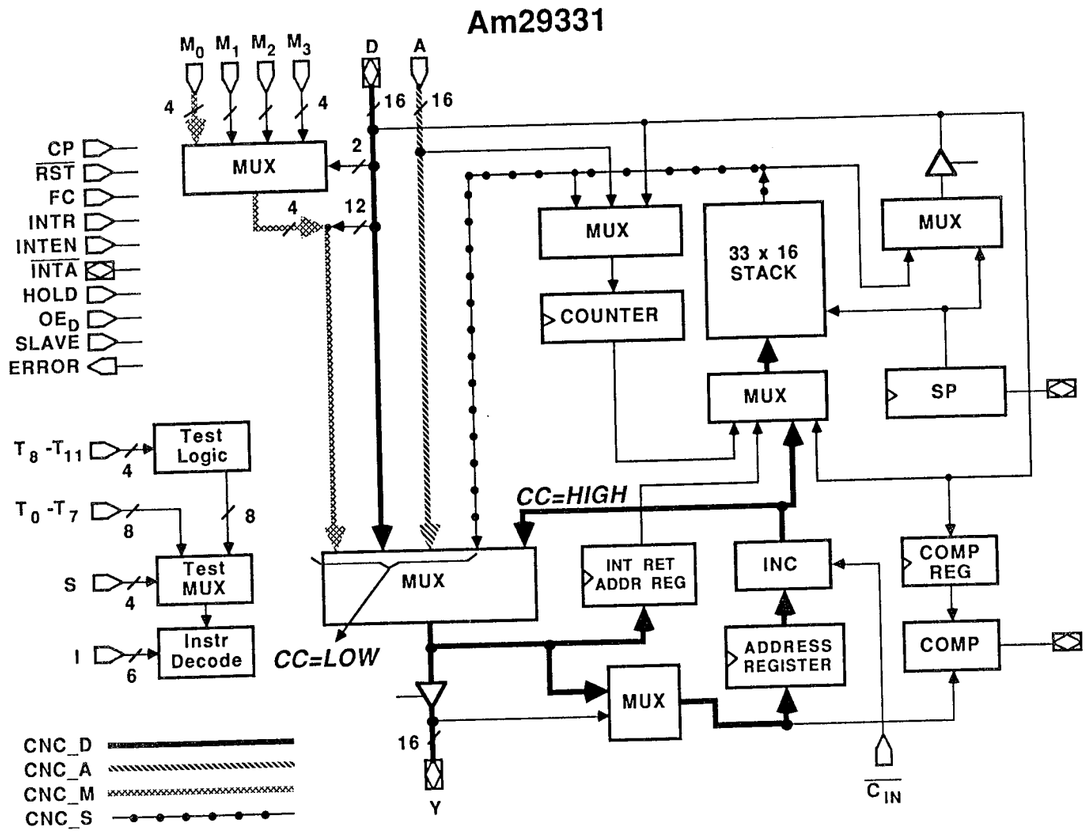




CALL SUBROUTINE ON INVERSE CONDITION

17	CNC_D	If CC is LOW then call the address specified by the D inputs else continue.
21	CNC_A	If CC is LOW then call the address specified by the A inputs else continue.
25	CNC_M	If CC is LOW then call the address specified by the D inputs catenated with the multiway M inputs else continue.
29	CNC_S	If CC is LOW then call the address on the top of the stack else continue. Also a conditional coroutine call.

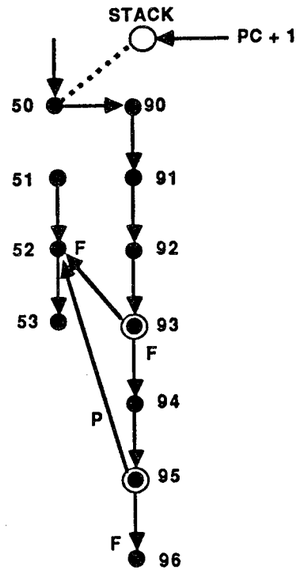


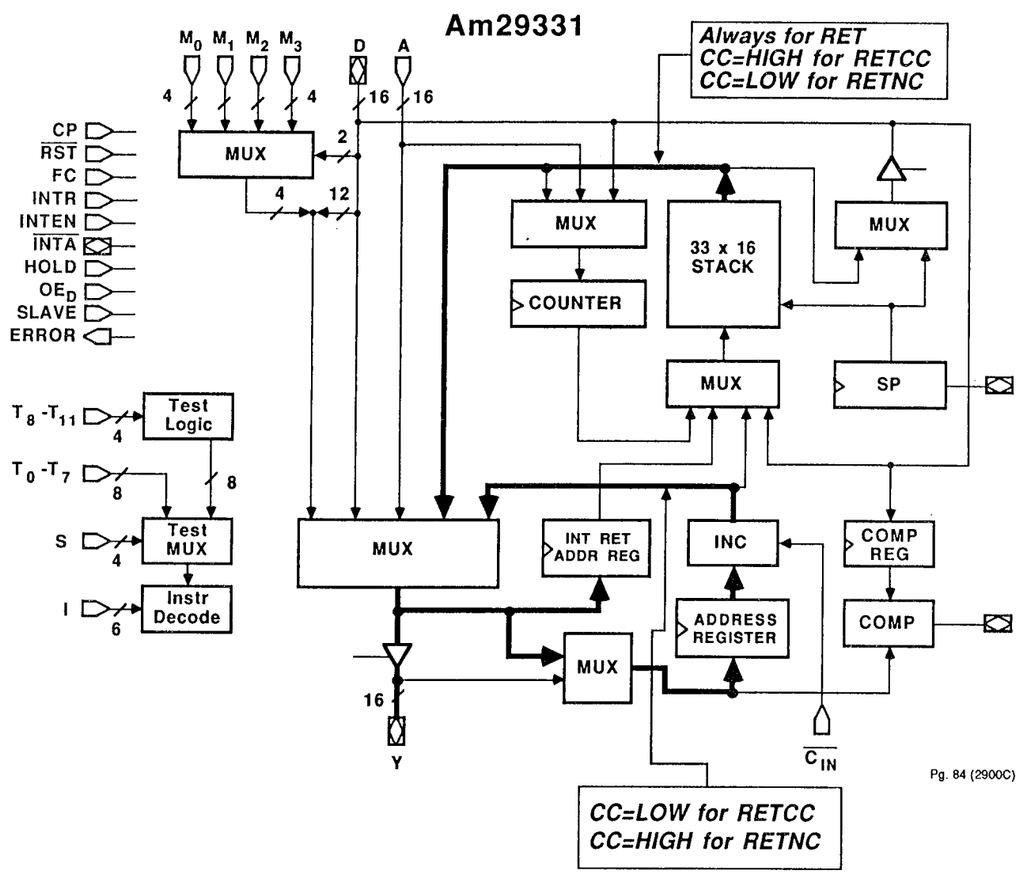


RETURN

46	RET	Unconditional return from subroutine.
14	RETCC	If CC is HIGH then return from subroutine else continue.
30	RETNC	If CC is LOW then return from subroutine else continue.

IF (CC = CONDITION) THEN RETURN;





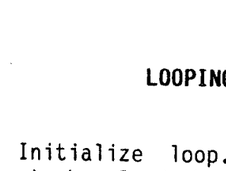
Pg. 84 (2900C)

LOOPING

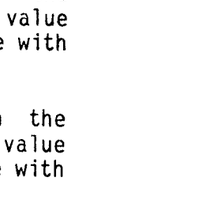
49	FOR_D	Initialize loop. Push the PC on the stack, load the counter with the value of the D inputs and continue. Use with DJMP_S for FOR...NEXT loops.
55	FOR_A	Initialize loop. Push the PC on the stack, load the counter with the value of the A inputs and continue. Use with DJMP_S for FOR...NEXT loops.
51	LOOP	Initialize loop. Push the PC and continue. Use with BRCC_S for REPEAT... UNTIL loops or with XTCC_D and BRA_S for WHILE...ENDWHILE loops.

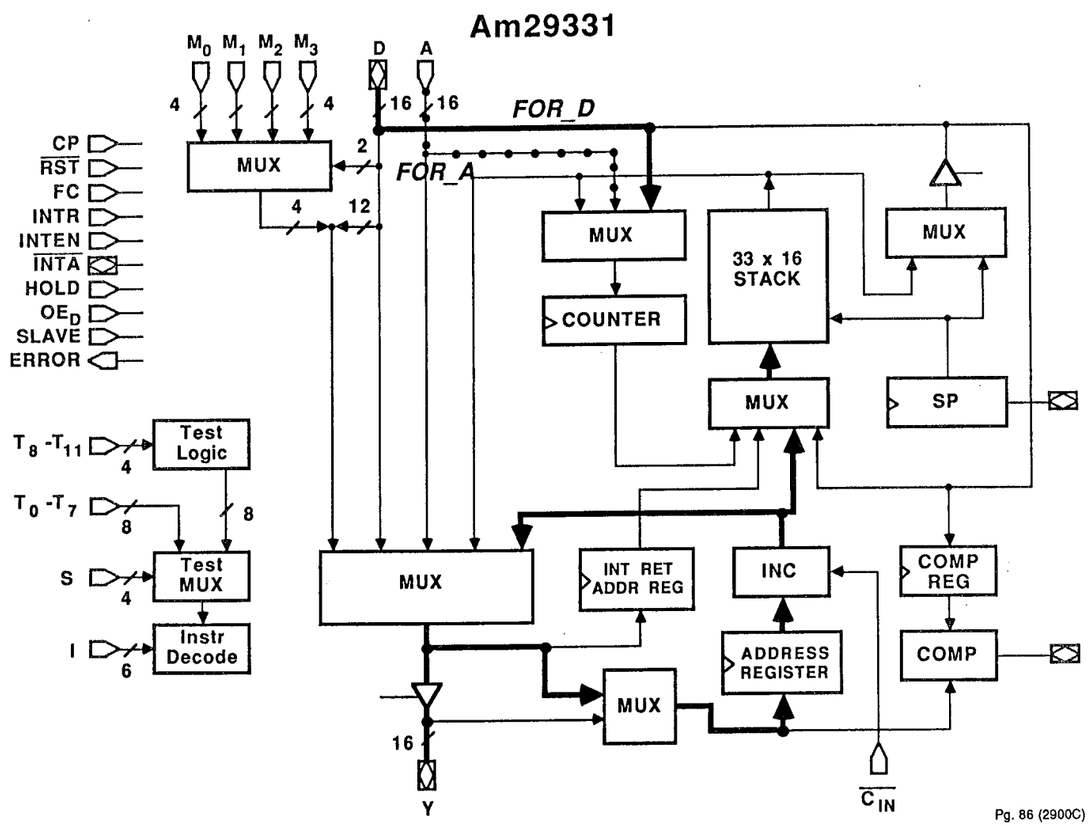
High-Level Language	Am29331 code
FOR COUNTER = {D / A} DOWNTO 1	{FOR_D / FOR_A}
. . .	. . .
IF (COUNTER ≠ 1) CONTINUE LOOP	DJMP_S

FOR_D / FOR_A:



LOOP:





Pg. 86 (2900C)

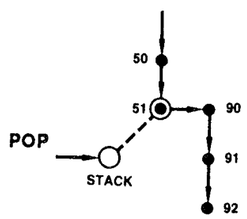
EXIT

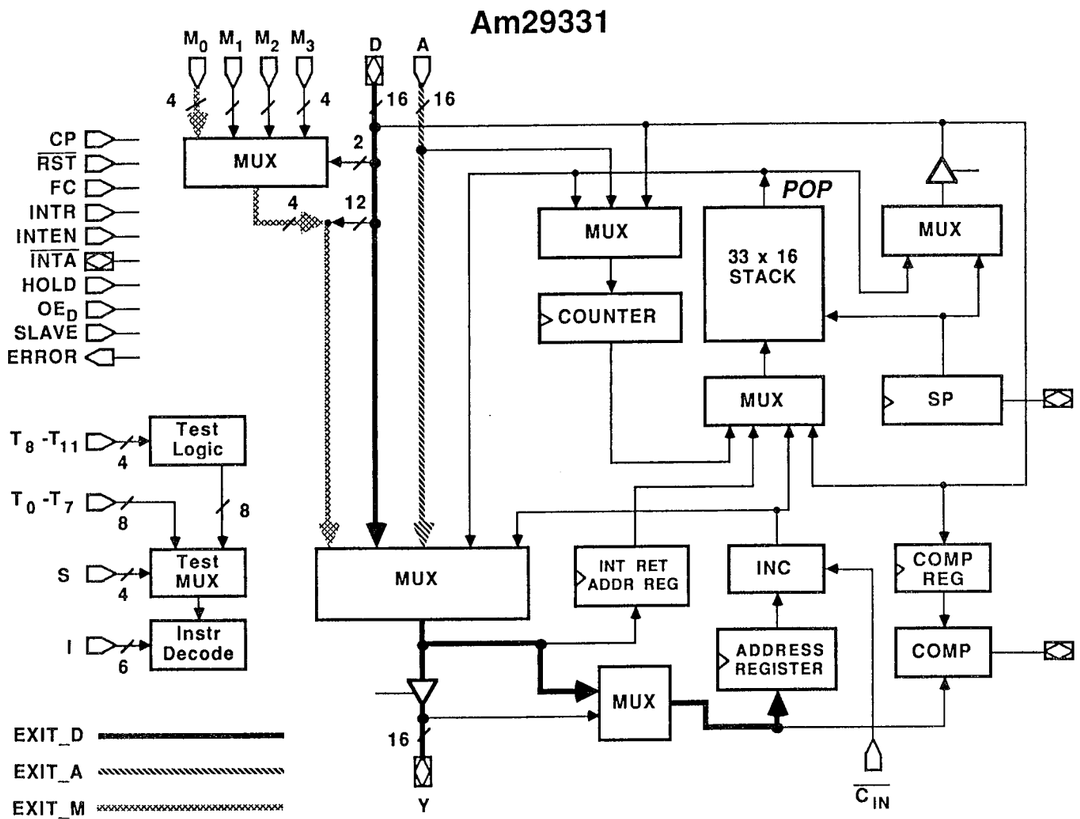
34 EXIT_D — Exit to D. Unconditional branch to the address specified by the D inputs and pop the stack.

38 EXIT_A — Exit to A. Unconditional branch to the address specified by the A inputs and pop the stack.

42 EXIT_M — Exit to M. Unconditional branch to the address specified by the D inputs catenated with the multiway M inputs and pop the stack.

```
LOOP;
    .
    .
    .
  EXIT;
    .
    .
    .
UNTIL CONDITION;
```

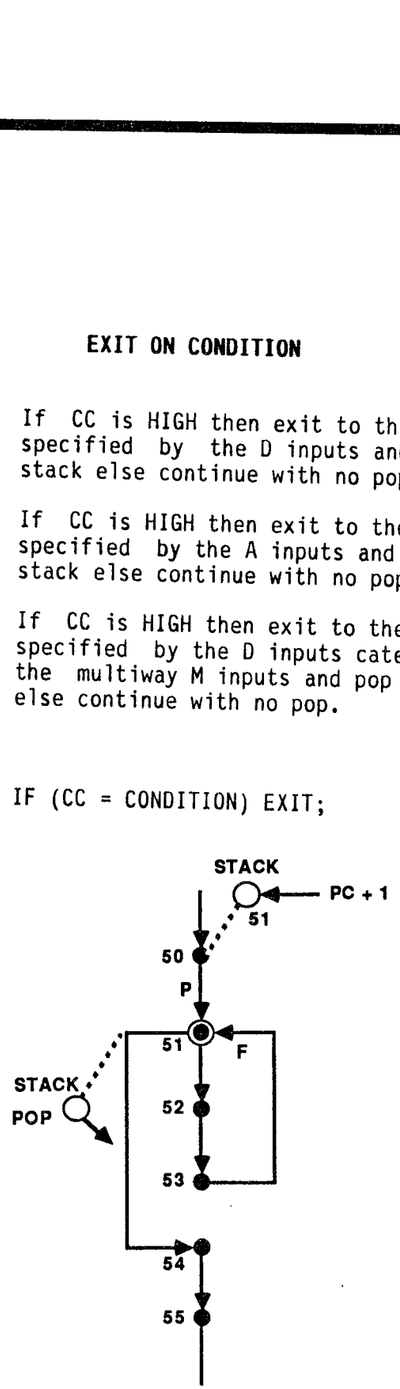


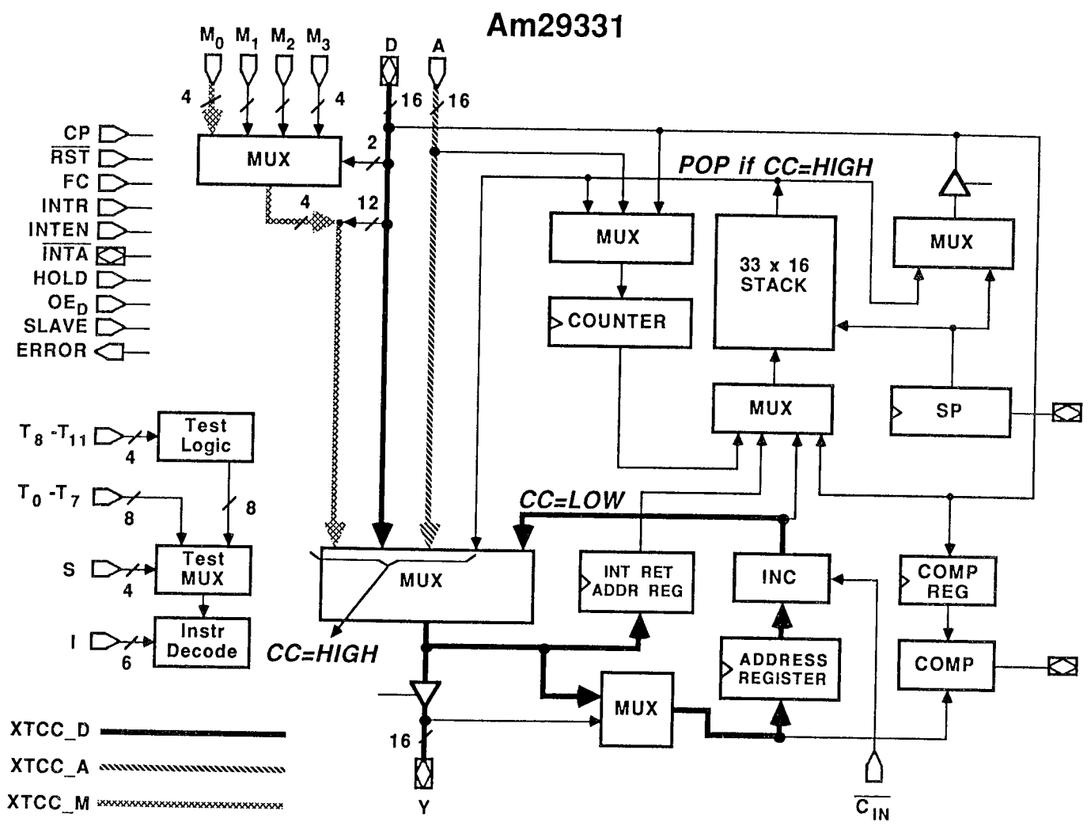


EXIT ON CONDITION

2	XTCC_D	If CC is HIGH then exit to the address specified by the D inputs and pop the stack else continue with no pop.
6	XTCC_A	If CC is HIGH then exit to the address specified by the A inputs and pop the stack else continue with no pop.
10	XTCC_M	If CC is HIGH then exit to the address specified by the D inputs catenated to the multiway M inputs and pop the stack else continue with no pop.

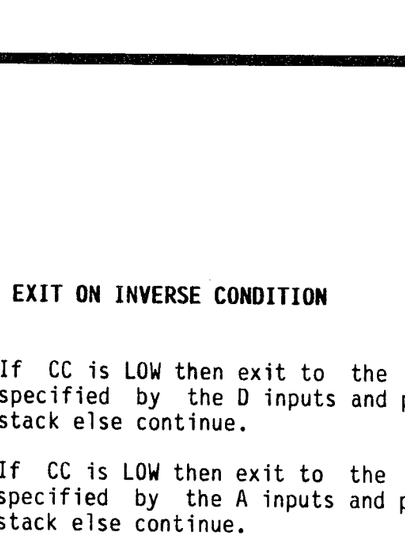
```
IF (CC = CONDITION) EXIT;
```

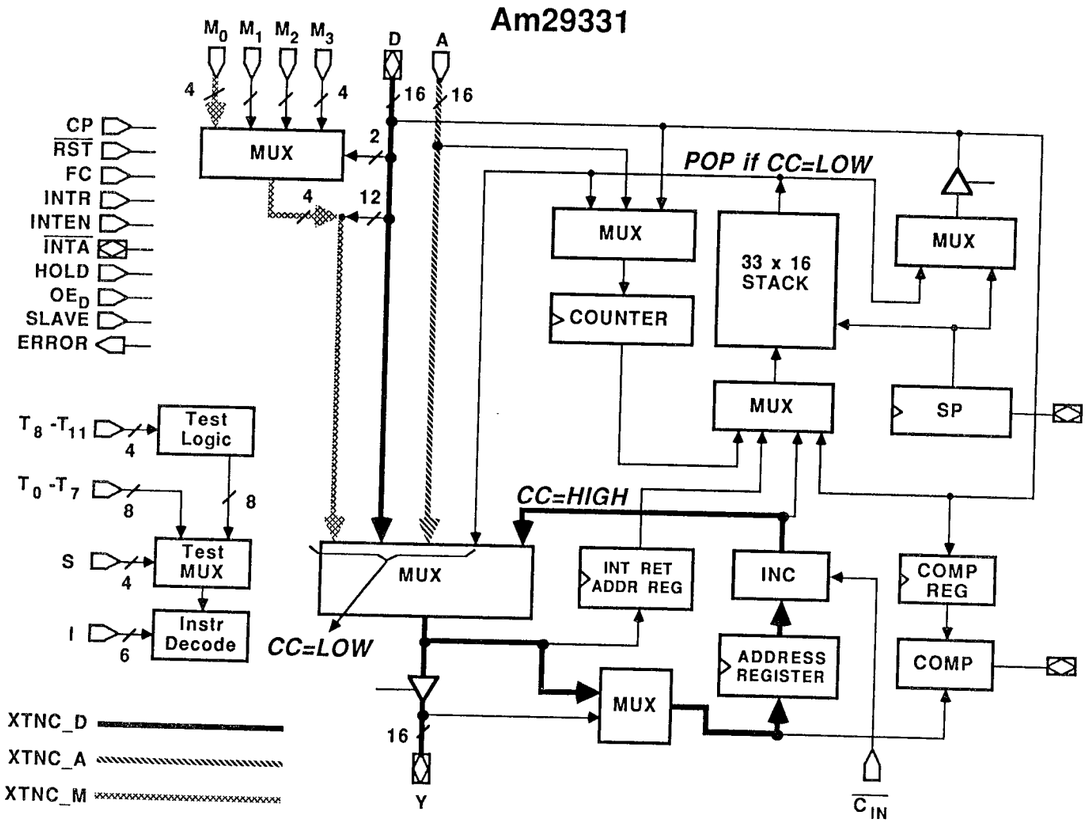




EXIT ON INVERSE CONDITION

18	XTNC_D	If CC is LOW then exit to the address specified by the D inputs and pop the stack else continue.
22	XTNC_A	If CC is LOW then exit to the address specified by the A inputs and pop the stack else continue.
26	XTNC_M	If CC is LOW then exit to the address specified by the D inputs catenated with the multiway M inputs and pop the stack else continue.



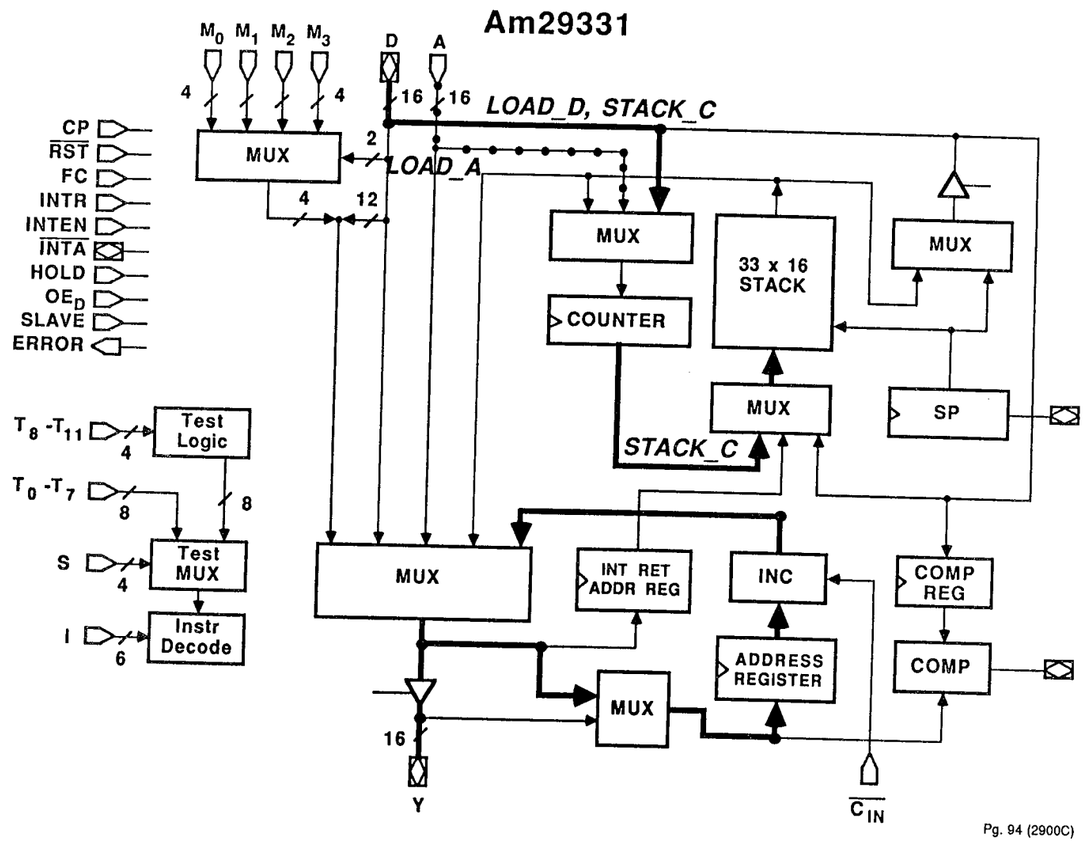


LOAD STACK/COUNTER

59	STACK_C	Push the counter on the stack, load the counter with the value of the D inputs and continue.
60	LOAD_D	Load the counter with the value of the D inputs and continue.
61	LOAD_A	Load the counter with the value of the A inputs and continue.

STACK_C

LOAD_D
LOAD_A

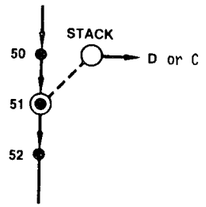


Pg. 94 (2900C)

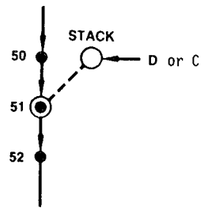
STACK

52	POP_D	Pop the stack, output the value on the D outputs and continue.
56	POP_C	Pop the stack, place the value in the counter and continue.
53	PUSH_D	Push the D inputs on the stack and continue.
57	PUSH_C	Push the counter on the stack and continue.
58	SWAP	Exchange the counter and the top of stack and continue.

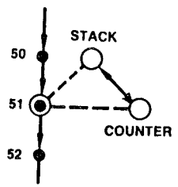
POP_D
POP_C



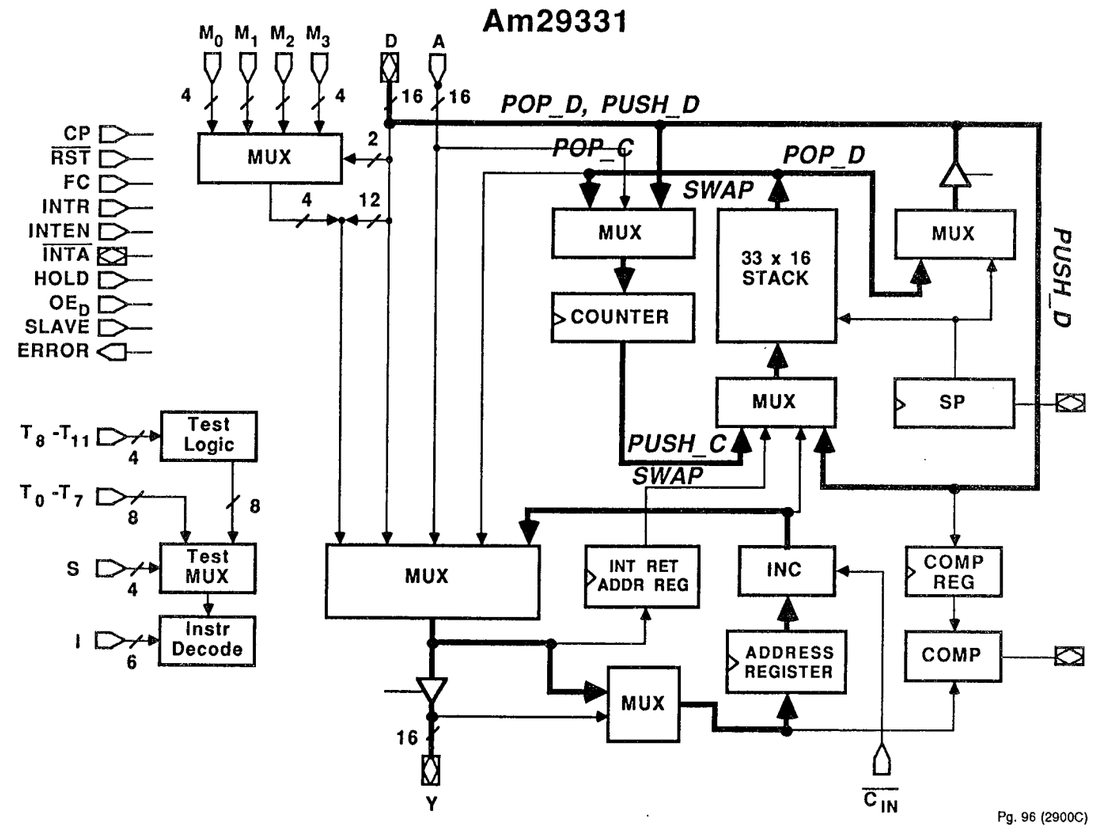
PUSH_D
PUSH_C



SWAP



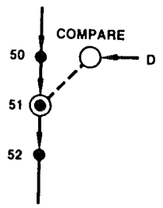
Am29331

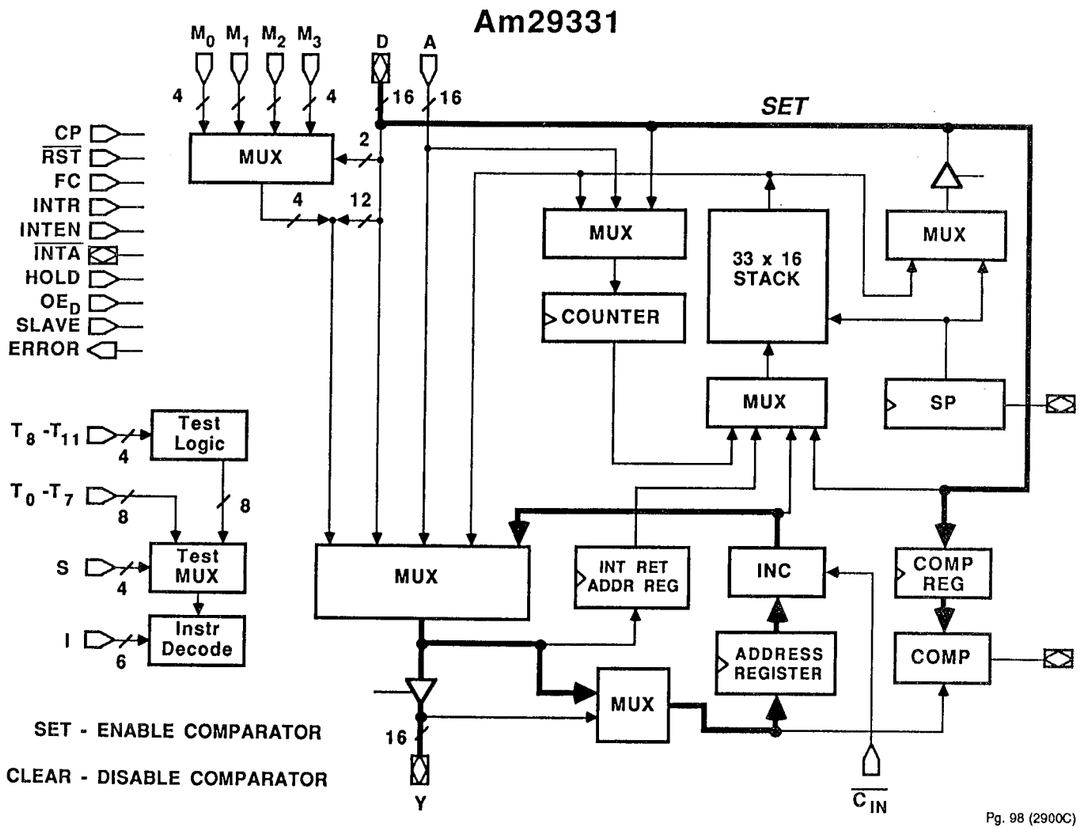


Pg. 96 (2900C)

COMPARISON REGISTER

62	SET	Load the comparison register with the value of the D inputs, enable the comparator and continue.
63	CLEAR	Disable the comparator and continue.

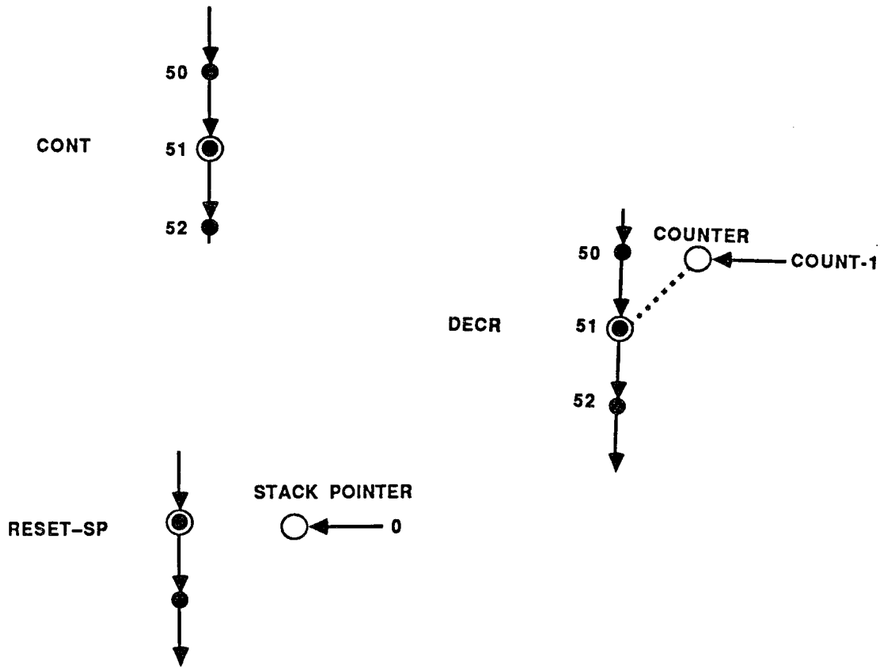


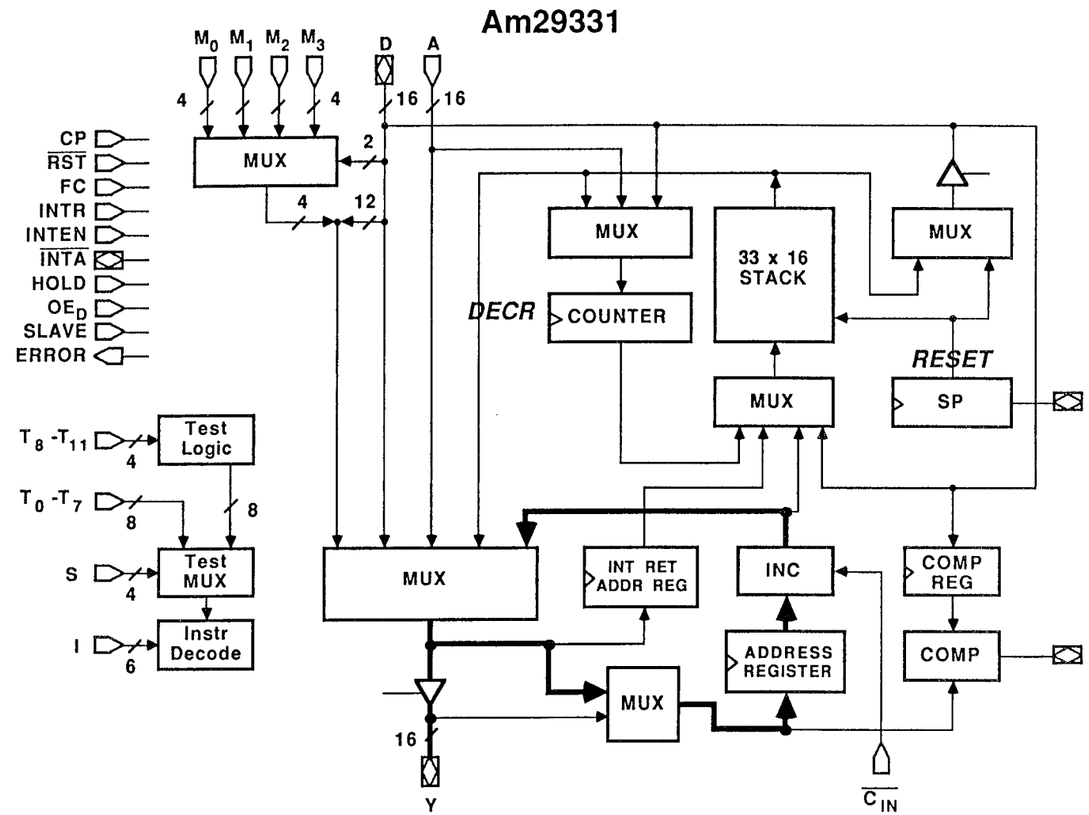


Pg. 98 (2900C)

CONTINUE

48	CONT	Continue.
50	DECR	Decrement the counter and continue.
54	RESET_SP	Reset the stack pointer and continue.





Am29331 Sequencer

Pin Descriptions

- **Address Inputs**

  D_0-D_{15} **Data, Bidirectional, Three-State**
  Input to address multiplexer, counter, stack and comparator register. Output for stack and stack pointer.

  A_0-A_{15} **Alternate Data Input**
  Input to address multiplexer and counter.

  $M_{0-3,0-3}$ **Multiway, Input**
  Four sets of multiway inputs providing 16-way branches. The first index refers to the set number.

- **Microcode Address**

  Y_0-Y_{15} **Address, Bidirectional, Three-State**
  Output of microcode address. Input for interrupt address.

- **Command**

  I_0-I_5 **Instruction, Input**
  Selects one of 64 instructions.

- **Test Inputs**

  T_0-T_{11} **Test, Input**
  Provides external test inputs.

  S_0-S_3 **Select, Input**
  Selects one of 16 test conditions.

Am29331 Sequencer

Pin Descriptions

- **Interrupt Control**

  INTR **Interrupt Request, Input**
  Requests the sequencer to interrupt execution.

  INTEN **Interrupt Enable, Input**
  Enables interrupts.

  $\overline{INTA}$ **Interrupt Acknowledge, Bidirectional, three-state, Active Low**
  Indicates that an interrupt is accepted.

- **Control**

  CP **Clock Pulse, Input**
  Clocks sequencer at the low to high transition.

  $\overline{RST}$ **Reset, Input**
  Resets the sequencer. If $\overline{C_{IN}}$ is high and CP is toggled, 0 will be loaded into the address register.

  FC **Force Continue, Input**
  Overrides instruction with CONTINUE.

  $\overline{C_{IN}}$ **Input, Active Low**
  Carry-in to incrementer.

  A-FULL **Almost Full, Bidirectional, Three-State**
  Indicates that SP $\geq$ 28.

  EQUAL **Bidirectional, Three-State**
  Indicates that the address comparator is enabled and has found a match.

  HOLD **Input**
  Stops the sequencer and three-states the outputs. $\overline{C_{IN}}$ is High and CP is toggled, 0 will be loaded into the address register.

  OE_D **Output Enable D-Bus, Input**
  Enables the D-bus driver provided the sequencer is not in the hold or slave mode.

Am29331 Sequencer

Pin Descriptions

- **Diagnostic**

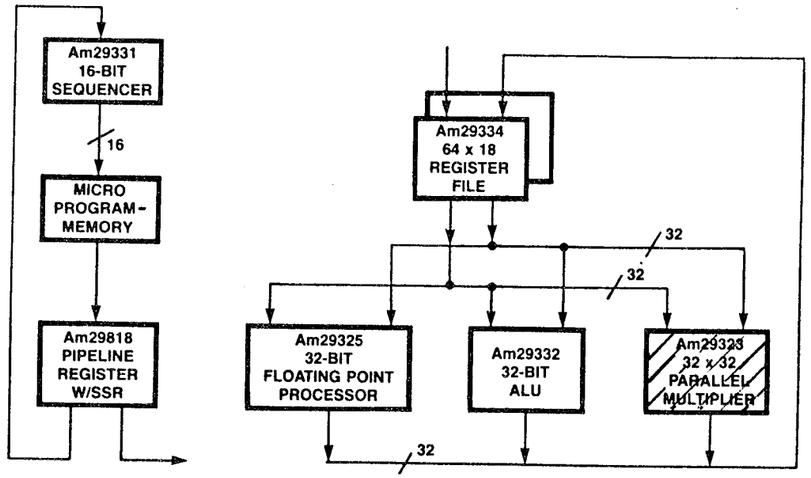
SLAVE **Input**
Makes the sequencer a slave.

ERROR **Output, Three-State**
Indicates a Master/Slave disagreement in slave mode, or a malfunctioning bus driver or bus contention in master mode.

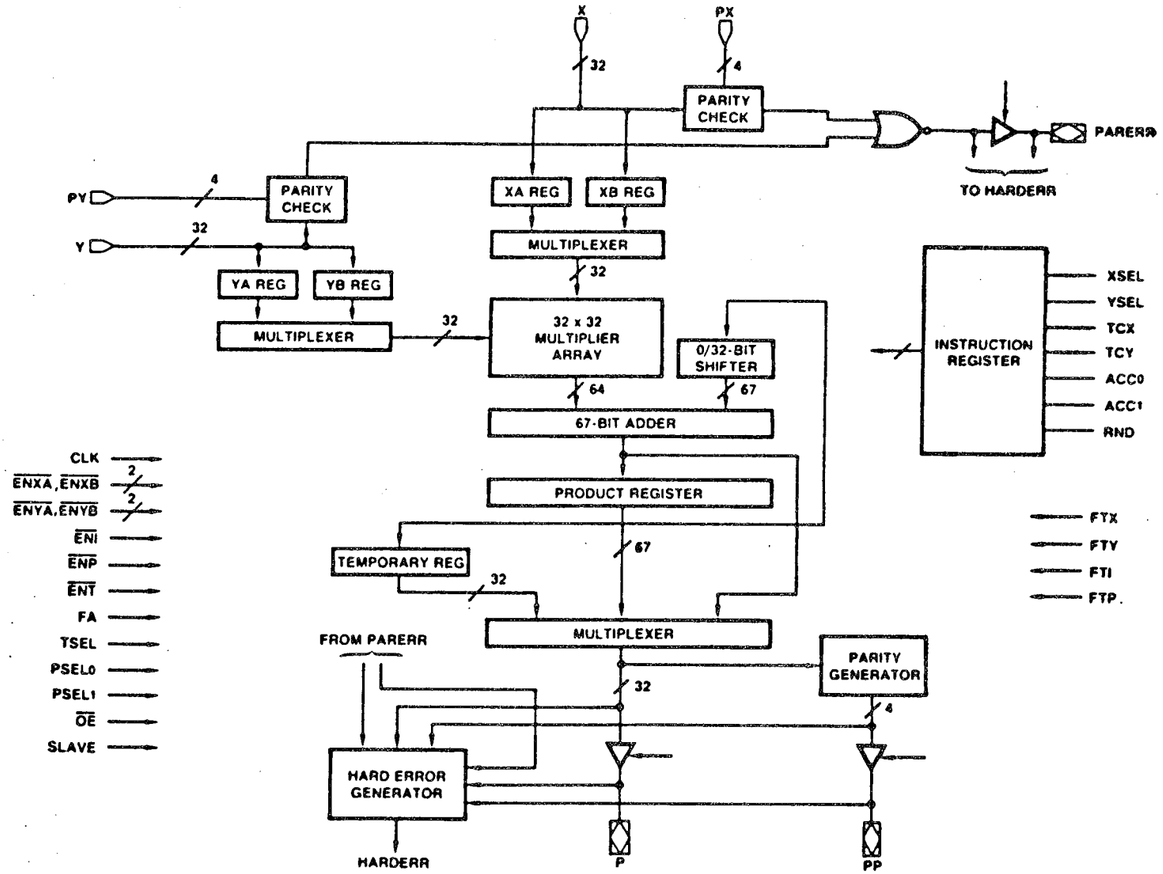
Am29323 Parallel Multiplier

Am29323 32 x 32 Parallel Multiplier

- Multiplies 32 bit x 32 bit operands in one 80-nsec cycle
- 32-bit, 3-bus flow-through architecture
- Operands can be two's complement or unsigned
- Includes 67-bit accumulator to support multiprecision multiplication
- Single 5 volt supply
- 169-pin (168 + index) PGA package
- Not cascadable



BLOCK DIAGRAM



Am29323 Architecture

Multiplier Array

- 32 x 32-bit array which produces a 64-bit product

Accumulator (also called Product Register)

- 67-bit wide for performing accumulation for sum-of-products operations and multiprecision multiplication.
- Operations:
  - Store product with accumulator
  - Accumulate product
  - Shift accumulator value and accumulate with product
- Accumulator operations selected by ACC1 and ACC0:

ACC1	ACC0	Accumulator Operation
0	0	PASS
0	1	ACCUMULATE
1	0	INVALID
1	1	SHIFT AND ACCUMULATE

Am29323 Architecture

Input Data Path

- Input operands enter Am29323 via two 32-bit buses: X, Y. Four formats can be handled.
- Input operands can be stored in one of two registers (XA or XB, or YA or YB), or fed directly to the multiplier array.
- Input parity checking performed as data input to Am29323.
- TCX and TCY select whether an input operand is signed two's complement or unsigned.

Fractional Two's Complement

TCX, TCY = 1

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
-2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}						2^{-28}	2^{-29}	2^{-30}	2^{-31}

Integer Two's Complement

TCX, TCY = 1

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
-2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^3	2^2	2^1	2^0

Unsigned Fractional

TCX, TCY = 0

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}						2^{-29}	2^{-30}	2^{-31}	2^{-32}

Unsigned Integer

TCX, TCY = 0

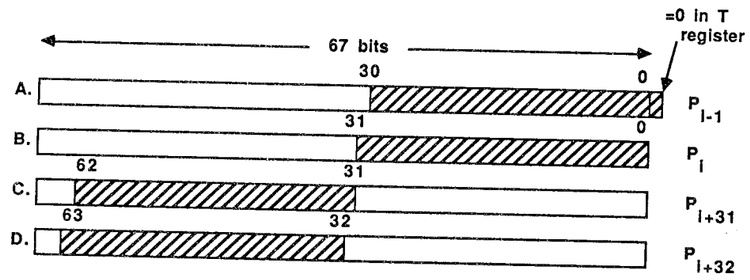
31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^3	2^2	2^1	2^0

Am29323 Architecture

Output Data Path

- Output is unsigned if both inputs are unsigned; otherwise output is signed two's complement.
- A temporary T register exists to hold results as required by some algorithms (e.g. multiprecision multiply).

The following 32-bit values (taken from the 67-bit accumulator) can be stored:



TSELect	To do
A	Obtain least significant 32 bits of number where sign is not desired, Note this gives one extra bit of precision.
B	Obtain least significant 32 bits of 64-bit number.
C	Obtain most significant 32 bits of number ignoring sign. This option is related to A.
D	Obtain most significant 32 bits of number including sign. This option is related to B.

TSEL and FA are used to select options A-D. FA, called Format Adjust, is used to determine whether 32 bits stored in T is right shifted by one bit or not. TSEL determines whether MSB or LSB 32 bits are selected from the accumulator.

TSEL	FA	Temp Reg Input
0	0	P_{i-1}
0	1	P_i
1	0	P_{i+31}
1	1	P_{i+32}

- An output multiplexer exists to select the output from:
  1. 67-bit adder
  2. Accumulator
  3. Temporary register

Note that FA is used to select whether a right-shifted 32-bits are output from the adder or accumulator.

PSEL1	PSEL0	FA	P Port Output
0	0	X	TEMP REGISTER
0	1	0	P_{i-1}
0	1	1	P_i
1	0	0	P_{i+31}
1	0	1	P_{i+32}
1	1	X	DISABLE

Output Formats

- Output from the Am29323 can be in several forms:
  1. Fractional Two's Complement
  2. Integer Two's Complement
  3. Unsigned Fractional
  4. Unsigned Integer
- The Fractional Two's Complement and Integer Two's Complelment formats can be output right-shifted one bit to increase the precision by one bit of the sign if the sign of the number is unimportant (e.g., least significant 32-bit segments of a multiprecision multiply).
- The formats are:

Fractional Two's Complement (Shifted)*

FA = 0, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
-2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}						2^{-28}	2^{-29}	2^{-30}	2^{-31}

FA = 0, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}	2^{-37}						2^{-60}	2^{-61}	2^{-62}	2^{-63}**

Fractional Two's Complement

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
-2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}						2^{-27}	2^{-28}	2^{-29}	2^{-30}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{-31}	2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}						2^{-59}	2^{-60}	2^{-61}	2^{-62}

Integer Two's Complement

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
-2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}						2^{35}	2^{34}	2^{33}	2^{32}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^{3}	2^{2}	2^{1}	2^{0}

Unsigned Fractional

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}						2^{-29}	2^{-30}	2^{-31}	2^{-32}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{-33}	2^{-34}	2^{-35}	2^{-36}	2^{-37}	2^{-38}						2^{-61}	2^{-62}	2^{-63}	2^{-64}

Unsigned Integer

FA = 1, PSEL1 = 1, PSEL0 = 0

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}						2^{35}	2^{34}	2^{33}	2^{32}

FA = 1, PSEL1 = 0, PSEL0 = 1

31	30	29	28	27	26	–	–	–	–	–	3	2	1	0
2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}						2^{3}	2^{2}	2^{1}	2^{0}

Example of Operation of Am29323

64 x 64 bit Signed Multiplication

- Each operand X and Y is split into two 32-bit inputs, XW1, XW0 and YW1, YW0.
- The signed 64-bit result is obtained by:

```
                    XW1    YW0
                    YW1    YW0
                  ------------
                    XW0  * YW0      <--- multiply only
              XW1  * YW0            <--- multiply and shift/acc
              XW0  * YW1            <--- multiply and accumulate
        XY1  * YW1                  <--- multiply and shift/acc
  ------------------------------
  P3    P2      P1      P0
```

- Sequence of events for performing the 64-bit multiply:

BUS AND REGISTER CONTENTS FOR A 64 x 64-BIT SIGNED MULTIPLICATION WITH ONE COMPLETE EXTENDED MULTIPLICATION SHOWN IN THE UNSHADED CYCLES

Cycle	0	1	2	3	4	5	6
X BUS	XW0	XW1			XW0	XW1	
XA BUS	XW0	XW0	XW0	XW0	XW0	XW0	XW0
XB REG	XW1	XW1	XW1	XW1	XW1	XW1	XW1
Y BUS	YW0	YW1			YW0	YW1	
YA REG	YW0	YW0	YW0	YW0	YW0	YW0	YW0
YB REG	YW1	YW1	YW1	YW1	YW1	YW1	YW1
MPY OP	XW1*YW1	XW0*YW0	XW1*YW0	XW0*YW1	XW1*YW1	XW0*YW0	XW1*YW0
ACC OP	S/A	PASS	S/A	ACC	S/A	PASS	S/A
T REG		PW3	PW0			PW3	
P BUS	PW1	PW2	PW3	PW0	PW1	PW2	PW3

Note: MPY OP = Operation of multiplier array (X*Y)
ACC OP = Operation of internal accumulator
PASS = pass through multiplier product
ACC = Add previous result to current product
S/A = Shift previous result then add to current product

Am29323 Pin List

X_{31}-X_0	Multiplicand data input port.
Y_{31}-Y_0	Multiplier data input port.
P_{31}-P_0	Product output port.
TCX,TCY	Mode control inputs for each input data word; LOW for unsigned data and HIGH for two's complement format.
ACC1,ACC0	Accumulator control lines used to determine accumulator function; PASS, ACCUMULATE, SHIFT/ACCUMULATE.
RND	Round control for rounding the most significant product.
CLK	Clock; all registers.
$\overline{\text{ENXA}}$,$\overline{\text{ENXB}}$	Register enables for multiplier data input registers (XA and XB).
$\overline{\text{ENYA}}$,$\overline{\text{ENYB}}$	Register enables for multiplier data input registers (YA and YB).
$\overline{\text{ENP}}$	Register enable for accumulator product register (P).
$\overline{\text{ENI}}$	Register enable for instruction register (I).
$\overline{\text{ENT}}$	Register enable for temporary register (T).
XSEL	Control line used to route the contents of either the XA register (HIGH) or XB register (LOW) into the multiplier array.
YSEL	Control line used to route the contents of either the YA register (HIGH) or YB register (LOW) into the multiplier array.

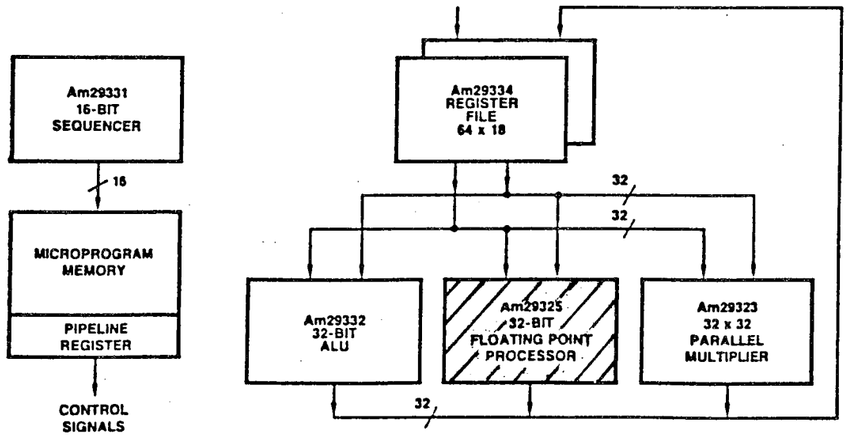
Am29323 Pin List (cont)

FA	Format adjust select either a full 64-bit product (HIGH) or a left-shifted 63-bit product suitable for fractional two's complement arithemtic (LOW).
TSEL	Select control line used to route the most significant product register (HIGH) or the least significant product register (LOW) into the temporary register.
FTX,FTY FTI	Feedthrough control lines for X, Y, and I registers.
FTP	Bypass control for output multiplexer.
PSEL1,PSEL0	Product control lines used to select desired output including disabling P output port.
PX_3-PX_0	Byte parity inputs on X input port.
PY_3-PY_0	Byte parity inputs on Y input port.
PP_3-PP_0	Byte parity outputs on P output port.
PARERR	Indicates a byte-parity error on an input bus.
$\overline{OE}$	Output enable control line used to disable the P output port.
SLAVE	$\overline{Master}$/Slave input control line used to determine mode of operation.
HARDERR	Hard error flag is used either: a) when two Am29323s are configured as master and slave so the slave card indicate a disagrrement with the master or the master can indicate its own output driver failures; b) when an Am29323 operates alone (as a master) to indicate output failures. In any case, HARDERR indicates a possible bus short or other hardware-related error in the device's proximity.

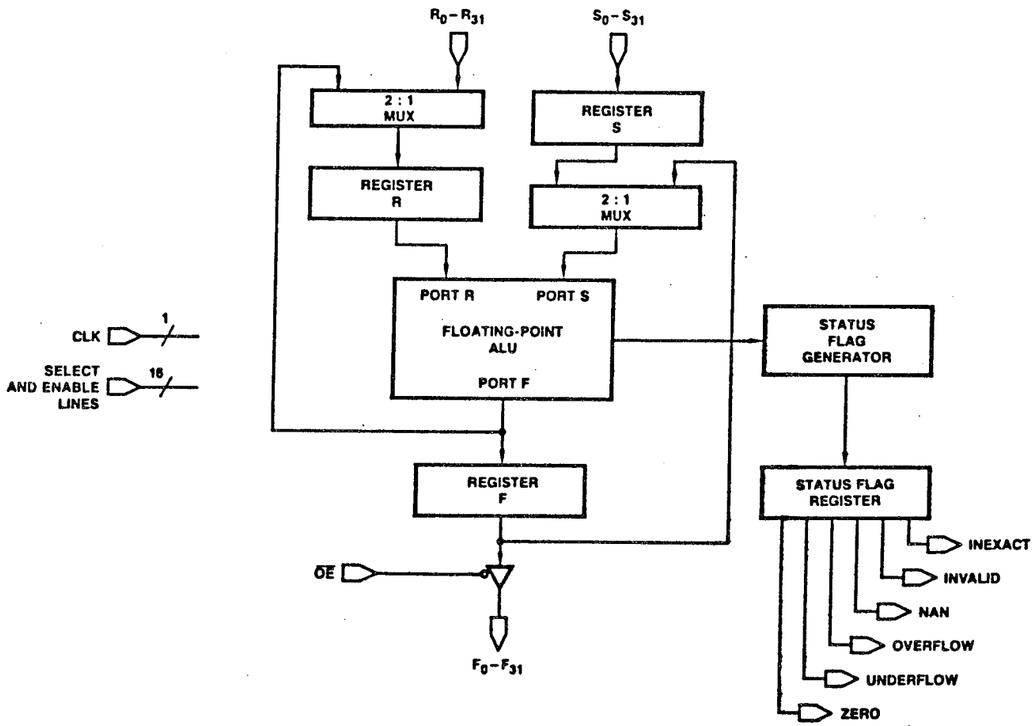
Am29325 Floating Point ALU

Am29325 Floating Point Processor

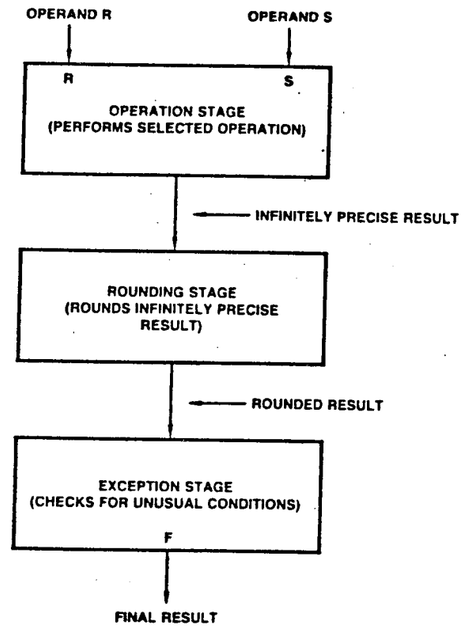
- All instructions performed in single 80 ns cycle.
- 32-bit, 3-bus flow-through architecture.
- Either 16-bit or 32-bit I/O interface.
- Supports both DEC and IEEE formats.
- Performs floating point addition, subtraction, multiply, format conversions, and operations to perform multi-cycle Newton-Raphson division.
- Independent control of input and output registers.
- Single 5-volt supply.
- 145-pin (144 + index) PGA package.
- Not cascadable.



BLOCK DIAGRAM
Am29325



Conceptual ALU Process



ALU Operations

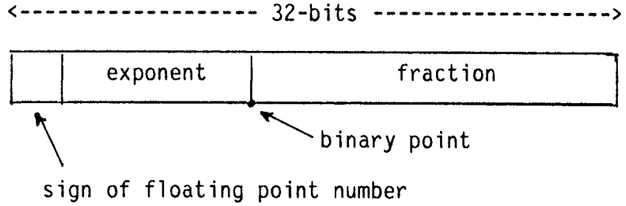
I_2	I_1	I_0	Operation	Output Equation
0	0	0	Floating-point addition (R PLUS S)	F = R + S
0	0	1	Floating-point subtraction (R MINUS S)	F = R − S
0	1	0	Floating-point multiplication (R TIMES S)	F = R * S
0	1	1	Floating-point constant subtraction (2 MINUS S)	F = 2 − S *
1	0	0	Integer-to-floating-point conversion (INT-TO-FP)	F (floating-point) = R (integer)
1	0	1	Floating-point-to-integer conversion (FP-TO-INT)	F (integer) = R (floating-point)
1	1	0	IEEE-TO-DEC format conversion (IEEE-TO-DEC)	F (DEC format) = R (IEEE format)
1	1	1	DEC-TO-IEEE format conversion (DEC-TO-IEEE)	F (IEEE format) = R (DEC format)

*This operation is used in the Newton-Raphson division algorithm.

Review of Floating Point Numbers and Arithmetic

- A floating point number consists of an **exponent** and a **mantissa.** Usually the exponent is represented as a **biased** value, while the mantissa is represented as a **signed** fractional number.

  Example:



- Consider the binary floating point number

  1.101×2^{-3}

  The mantissa is 1.101 (Note that the "1" to the left of the decimal point is not directly represented in the format; it is implied.)

  The **true** exponent (or, just "exponent") is -3

  The **biased** exponent is 127-3 = 124, if the bias is 127.

  Note that the exponent has no sign. The bias performs the function of a sign.

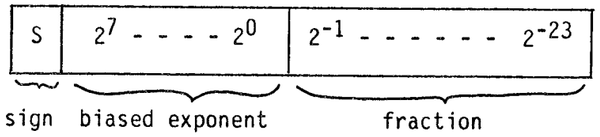
  The 32-bit representation of 1.101×2^{-3} is thus

sign	exponent	fraction
0	0 1 1 1 1 1 0 0	1 0 1 0 - - - - - - - 0

IEEE and DEC Formats

- **IEEE**

  Exponent bias = 127



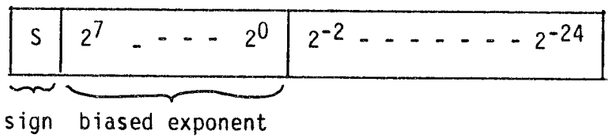
  Thus, 1.101×2^{-3} would be represented as

0	0 1 1 1 1 1 0 0	1 0 1 0 - - - - - - 0

(leading 1 in mantissa is implied)

- **DEC**

  Exponent bias = 128



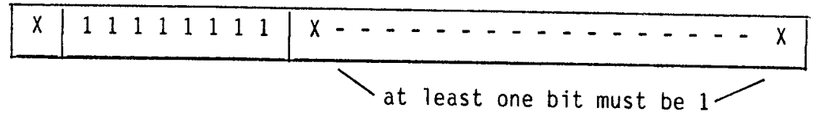
  Thus, 1.101×2^{-3} would be represented as 0.1101×2^{-2}

0	0 1 1 1 1 1 0 0	1 0 1 0 - - - - - - 0

(leading 1 of fraction is implied)

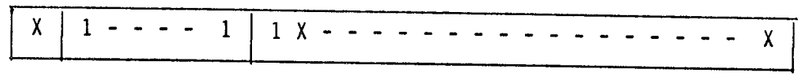
Special Numbers ((IEEE) bias = 127)

- **NAN - Not-a-number**
  - An IEEE floating-point number that is interpreted as a **symbol**
  - Has no numeric value
  - Biased exponent of 255, non-zero fraction
  - Fractional part used to represent flags or symbolic information



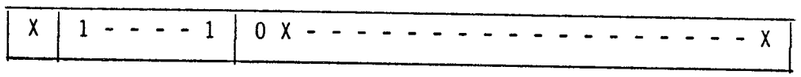
Signaling NAN

- Set invalid flag when used as input
- Useful for indicating uninitialized variables
- Am29325 never creates a signaling NAN
- Format:

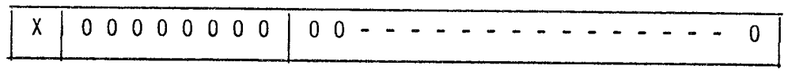


Quiet NAN

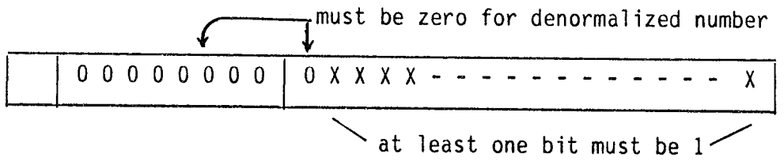
- Generated by invalid operations
- Pass through Am29325 without setting invalid flag if used as input (except for float-to-integer conversion)
- Format:



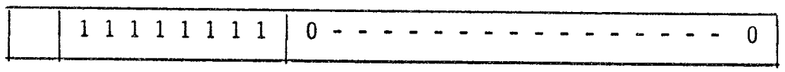
- **Zero**
  - Both the biased exponent and fractional part has value 0
  - Zero can be either positive or negative



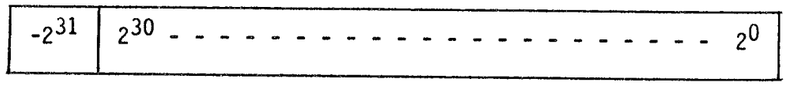
- **Denormalized Number**
  - Has magnitude >0 and $<2^{-126}$
  - Represented by 0 biased exponent and non-zero fraction with msb of fraction = 0



- **Infinity**
  - Has biased exponent = 255 and fractional part = 0

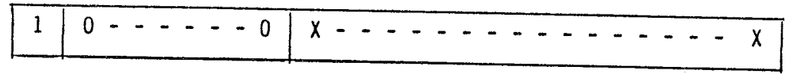


- **Integer**
  - 32-bit, two's complement

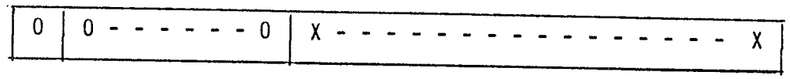


Special Numbers ((DEC) - bias = 128)

- **DEC reserved operand**
  - Biased exponent = 0 and sign = 1
  - Not a numeric value
  - Symbolic information and/or flags represented in fractional part of number



- **Zero**
  - Both exponent and sign are 0

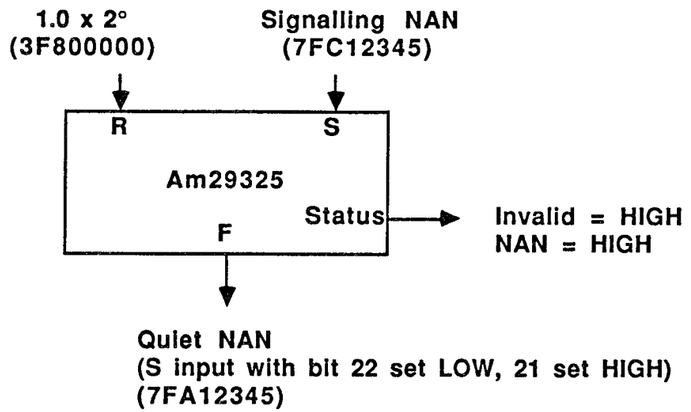


- Infinity not directly supported in DEC format
- DEC format is always considered normalized

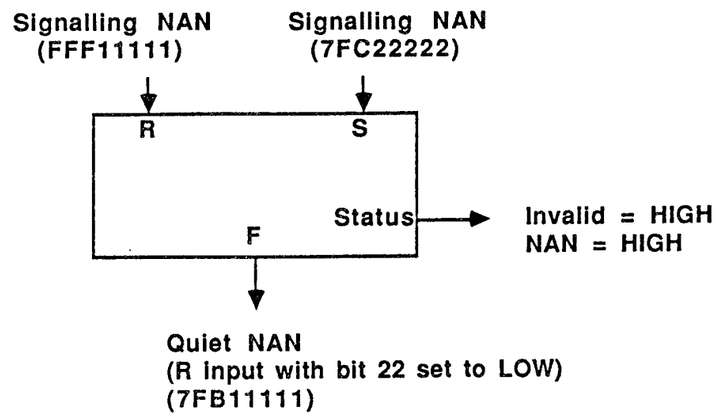
Am29325 Operation in IEEE Mode

- **Operations with NAN's**

Example: Floating point add with a signalling NAN input

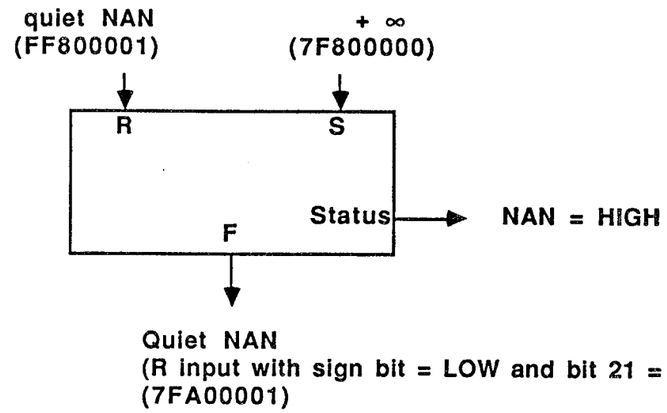


Example: Floating point multiply with a quiet NAN and a signalling NAN input



Operations with NAN's (cont)

Example: Floating point subtraction with quiet NAN and + ∞ input



- **Operations with Infinities**

  - Infinity is a valid input for the following operations:

    R+S
    R-S
    R*S
    2-S

  - Invalid operations are:

IEEE MODE INVALID OPERATIONS TABLE

Operation	Input Operand	Final Result
R PLUS S	$(+\infty) + (-\infty)$ or $(-\infty) + (+\infty)$	$7FA00000_{16}$ (quiet NAN)
R PLUS S	$(+\infty) + (+\infty)$ or $(-\infty) + (-\infty)$ (Note 1)	$7FA00000_{16}$ (quiet NAN)
R MINUS S	$(+\infty) - (+\infty)$ or $(-\infty) - (-\infty)$	$7FA00000_{16}$ (quiet NAN)
R MINUS S	$(+\infty) - (-\infty)$ or $(-\infty) - (+\infty)$ (Note 1)	$7FA00000_{16}$ (quiet NAN)
R TIMES S	$(+0) \cdot (+\infty)$ or $(+0) \cdot (-\infty)$ or $(-0) \cdot (+\infty)$ or $(-0) \cdot (-\infty)$	$7FA00000_{16}$ (quiet NAN)
R PLUS S R MINUS S R TIMES S	R or S is a signalling NAN	(Note 2)
2 MINUS S	S is a signalling NAN	(Note 2)
FP-TO-INT	R is a signalling or quiet NAN	(Note 2)
FP-TO-INT	$R > 2^{31}-1$ or $R < -(2^{31})$	$7FA00000_{16}$ (quiet NAN)

Notes: 1. These cases are invalid in projective mode only.
2. Results for these operations are described in the **Operations with NANs** section.

  - There are two ways to handle operations where <u>both</u> inputs are infinite.

    (1) Projective

    (2) Affine

  - The only differences between the modes in the Am29325 are during addition and subtraction:

Operation	Affine Mode	Projective Mode
$(+\infty)+(+\infty)$	Output $+\infty$	Output $7FA00000_{16}$ (quiet NAN), set invalid and NAN flags
$(-\infty)+(-\infty)$	Output $-\infty$	Output $7FA00000_{16}$ (quiet NAN), set invalid and NAN flags
$(+\infty)-(-\infty)$	Output $+\infty$	Output $7FA00000_{16}$ (quiet NAN), set invalid and NAN flags
$(-\infty)-(+\infty)$	Output $-\infty$	Output $7FA00000_{16}$ (quiet NAN), set invalid and NAN flags

- **Rounding**

  - The Am29325 supports four rounding modes:

RND_1	RND_0	Rounding Mode
0	0	Round to nearest
0	1	Round toward $-\infty$
1	0	Round toward $+\infty$
1	1	Round toward 0

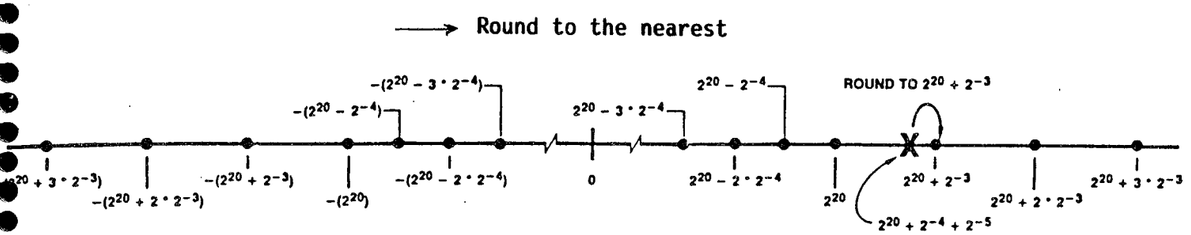
  - To see the effects of rounding mode on typical problems, consider the following examples (infinitely precise results shown):

1) $2^{20} + 2^{-4} + 2^{-5} = 1.00000000000000000000011 \times 2^{20}$

2) $2^{20} - 2^{-4} + 2^{-8} = 1.11111111111111111111110001 \times 2^{19}$

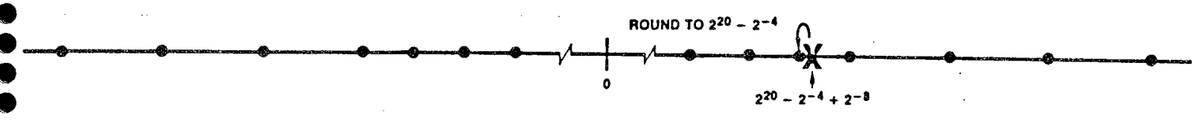
3) $-(2^{20} + 2^{-3} + 2^{-4}) = -1.00000000000000000000011 \times 2^{20}$

4) $2^{20} + 3 \cdot 2^{-3} = 1.00000000000000000000110 \times 2^{20}$



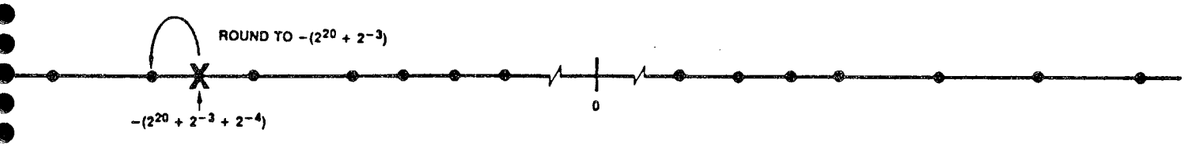
1) $2^{20} + 2^{-4} + 2^{-5} = 1.00000000000000000000011 \times 2^{20}$ infinite precision

   $2^{20} + 2^{-3} = 1.00000000000000000000001 \times 2^{20}$ rounded to nearest



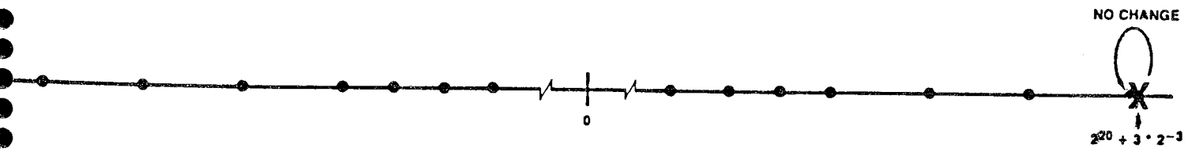
2) $2^{20} - 2^{-4} + 2^{-8} = 1.11111111111111111111110001 \times 2^{19}$ infinite precision

   $2^{20} - 2^{-4} = 1.11111111111111111111111 \times 2^{19}$ rounded to nearest



3) $-(2^{20} + 2^{-3} + 2^{-4}) = -1.00000000000000000000011 \times 2^{20}$ infinite precision

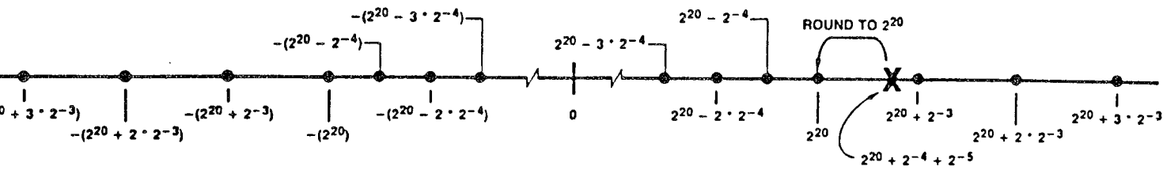
   $-(2^{20} + 2 * 2^{-3}) = -1.00000000000000000000010 \times 2^{20}$ rounded to nearest



4) $2^{20} + 3 \cdot 2^{-3} = 1.00000000000000000000110 \times 2^{20}$ infinite precision

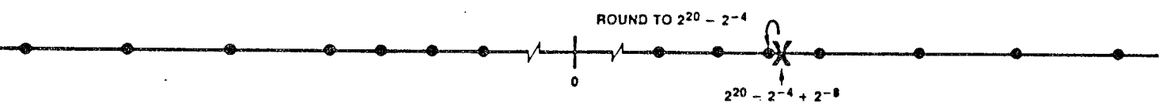
   $= 1.00000000000000000000011 \times 2^{20}$ rounded to nearest

Round toward - ∞



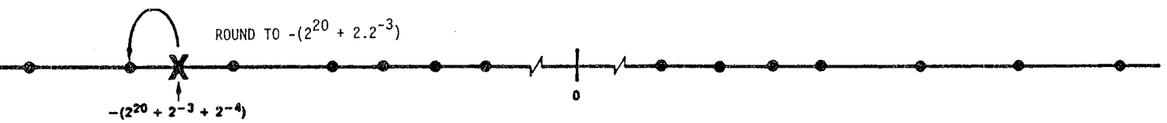
1) $2^{20} + 2^{-4} + 2^{-5}$ = $1.00000000000000000000011 \times 2^{20}$ infinite precision

2^{20} = $1.00000000000000000000000 \times 2^{20}$ rounded toward - ∞



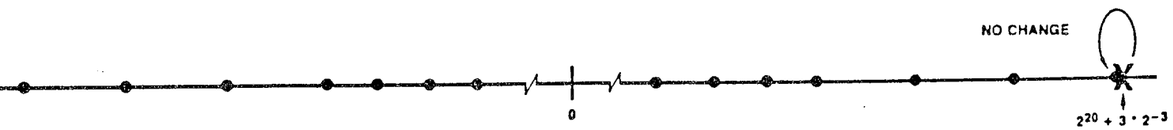
2) $2^{20} - 2^{-4} + 2^{-8}$ = $1.11111111111111111111110001 \times 2^{19}$ infinite precision

$2^{20} - 2^{-4}$ = $1.11111111111111111111111 \times 2^{19}$ rounded toward - ∞



3) $-(2^{20} + 2^{-3} + 2^{-4})$ = $-1.00000000000000000000011 \times 2^{20}$ infinite precision

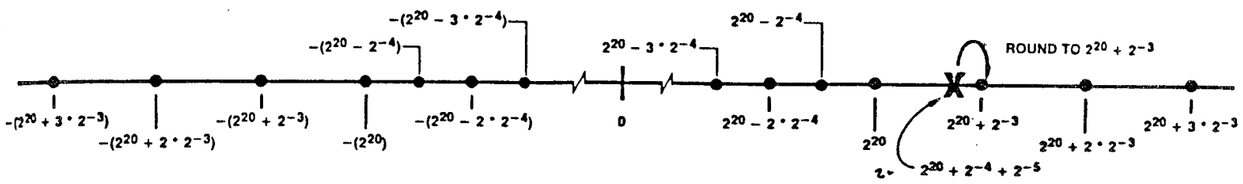
$-(2^{20} + 2 * 2^{-3})$ = $-1.00000000000000000000010 \times 2^{20}$ rounded toward - ∞



4) $2^{20} + 3 \cdot 2^{-3}$ = $1.00000000000000000000110 \times 2^{20}$ infinite precision

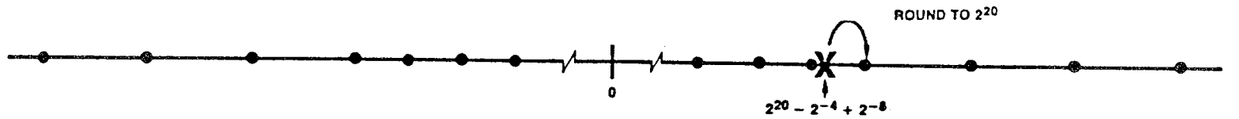
= $1.00000000000000000000011 \times 2^{20}$ rounded to nearest - ∞ (no change)

Round toward + ∞



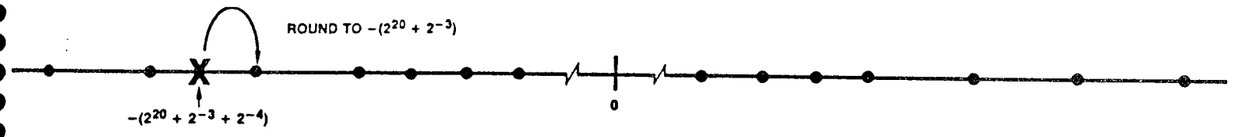
1) $2^{20} + 2^{-4} + 2^{-5}$ = $1.00000000000000000000011 \times 2^{20}$ infinite precision

$2^{20} + 2^{-3}$ = $1.00000000000000000000001 \times 2^{20}$ rounded toward + ∞



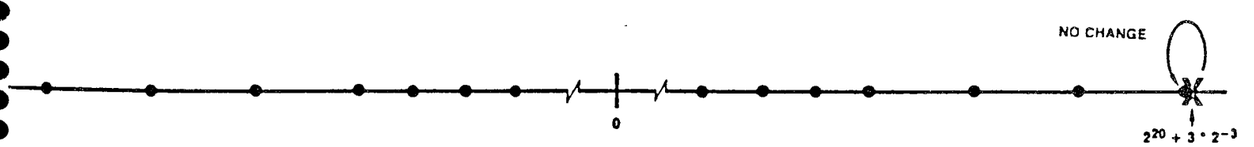
2) $2^{20} - 2^{-4} + 2^{-8}$ = $1.11111111111111111111110001 \times 2^{19}$ infinite precision

2^{20} = $1.00000000000000000000000 \times 2^{20}$ rounded toward + ∞



3) $-(2^{20} + 2^{-3} + 2^{-4})$ = $-1.00000000000000000000011 \times 2^{20}$ infinite precision

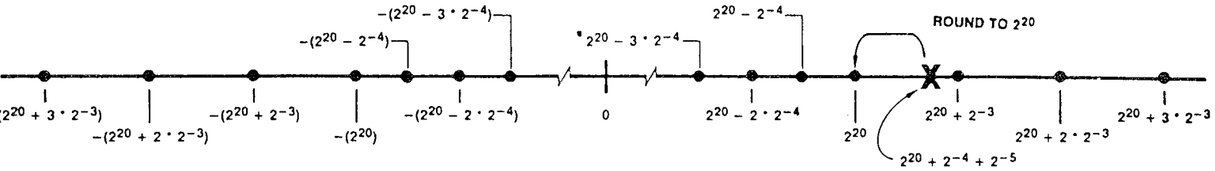
$-(2^{20} + 2^{-3})$ = $-1.00000000000000000000001 \times 2^{20}$ rounded toward + ∞



4) $2^{20} + 3 \cdot 2^{-3}$ = $1.00000000000000000000110 \times 2^{20}$ infinite precision

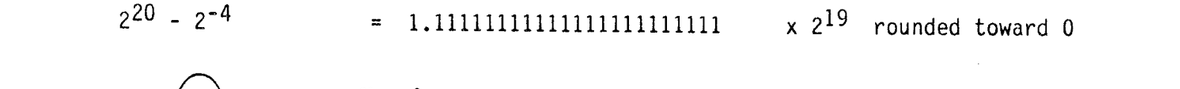
= $1.00000000000000000000011 \times 2^{20}$ rounded toward + ∞ (no change)

Round toward 0



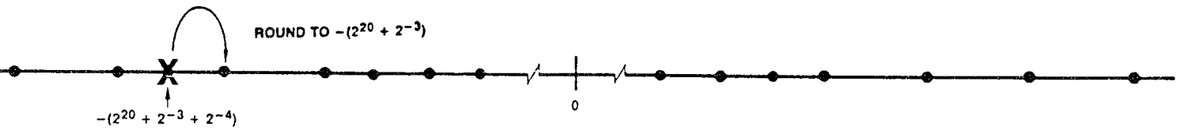
1) $2^{20} + 2^{-4} + 2^{-5}$ = $1.00000000000000000000011 \times 2^{20}$ infinite precision

2^{20} = $1.00000000000000000000000 \times 2^{20}$ rounded toward 0



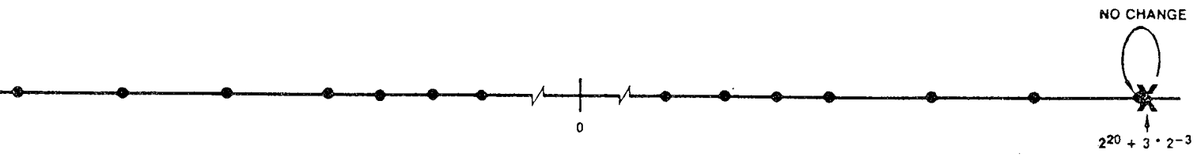
2) $2^{20} - 2^{-4} + 2^{-8}$ = $1.11111111111111111111110001 \times 2^{19}$ infinite precision

$2^{20} - 2^{-4}$ = $1.11111111111111111111111 \times 2^{19}$ rounded toward 0



3) $-(2^{20} + 2^{-3} + 2^{-4})$ = $-1.00000000000000000000011 \times 2^{20}$ infinite precision

$-(2^{20} + 2^{-3})$ = $-1.00000000000000000000001 \times 2^{20}$ rounded toward 0



4) $2^{20} + 3 \cdot 2^{-3}$ = $1.00000000000000000000110 \times 2^{20}$ infinite precision

= $1.00000000000000000000011 \times 2^{20}$ no change

Sign of 0 results

R PLUS S

+X + (-X)
-X + (+X)

+0 + (+0) = +0

-0 + (-0) = -0

Rounding Mode	Sign of Result
Round to nearest	0
Round toward $-\infty$	1
Round toward $+\infty$	0
Round toward 0	0

R MINUS S

+0 - (-0) = +0

-0 - (+0) = -0

Rounding Mode	Sign of Final Result
Round to nearest	0
Round toward $-\infty$	1
Round toward $+\infty$	0
Round toward 0	0

R TIMES S

O	X	Result
+	+	+0
+	-	-0
-	+	-0
-	-	+0

Result = Sign multiplier ⊕ Sign multiplicand

2 MINUS S

If S = 2, result is +0 round to nearest

-0 round to nearest -

+0 round to nearest +

+0 round to nearest 0

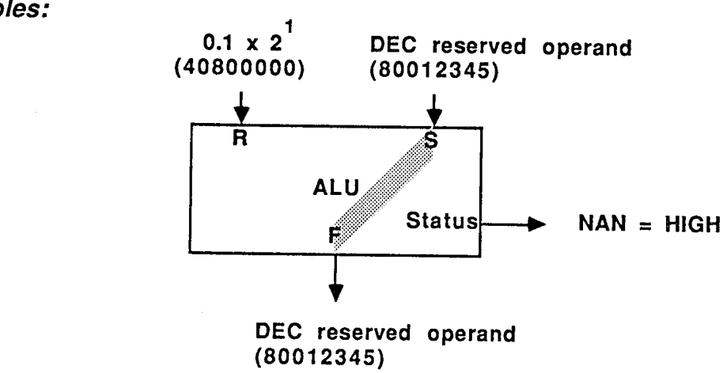
Am29325 Operations in DEC Mode

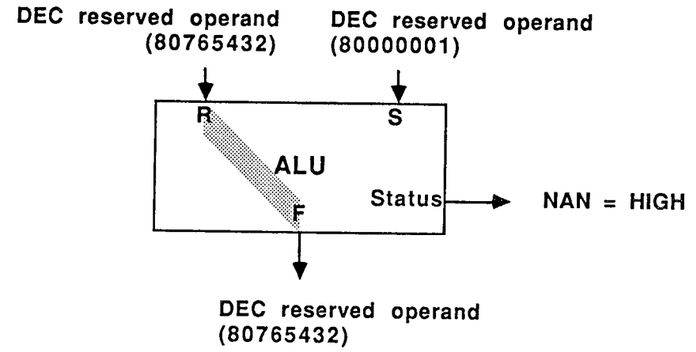
- **Operations with DEC Reserved Operands**

  DEC Reserved Operand as input

  - A single input reserved operand is passed directly to the output
  - Two input reserved operands pass the R input directly to the output
  - The NAN flag is raised whenever a DEC reserved operand is produced as a final result

Examples:





- **Operations producing overflow**
  - **Floating point**
    - -- Overflow can only occur for the following operations:

      R+S
      R-S
      R*S
      2-S

    - -- Overflow occurs when result, after rounding is $\geq 2^{127}$
    - -- DEC reserved operand, 800000000, output when overflow occurs
  - **Integer**
    - -- Overflow can only occur when floating-point-to-integer conversion takes place.
    - -- Overflow occurs when floating-point number, after rounding, is $> 2^{31}-1$ or $< -2^{31}$.
    - -- DEC reserved operand, 800000000, output when integer overflow occurs.

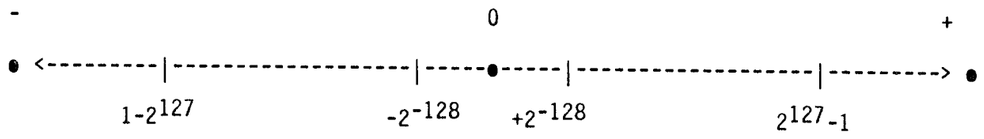
- **Operations producing underflow**
  - **Floating point**
    - -- Underflow can only occur for the following operations:

      R+S
      R-S
      R*S
      2-S

    - -- Underflow occurs when result, after rounding, has magnitude $0 < \text{magnitude} < 2^{-128}$
    - -- Zero output if underflow occurs. Underflow, inexact, and zero flags set to High
  - **Integer**
    - -- Underflow never occurs

- **Rounding**
  - DEC mode handles rounding the same as for IEEE mode.
  - Exception: For round-to-nearest mode, if the infinitely precise result is exactly half-way between two representable values, the result is rounded to the larger magnitude. (IEEE rounds to the value whose LSB is 0.)

- Representable floating-point numbers:



Am29325 Status Flags

1. **INEXACT** — Indicates that the result of the last operation was not infinitely precise.

2. **INVALID** — Indicates that the result of the last operation was invalid (e.g., ∞ times 0).

3. **NAN** — Indicates that the result of the last operation should not be interpreted as a number.

4. **OVERFLOW** — Indicates that the result of the last operation overflowed the floating point format.

5. **UNDERFLOW** — Indicates that the result of the last operation produced a rounded result which underflowed the floating point value.

6. **ZERO** — Indicates that the result of the last operation was zero.

Am29325 Data Path

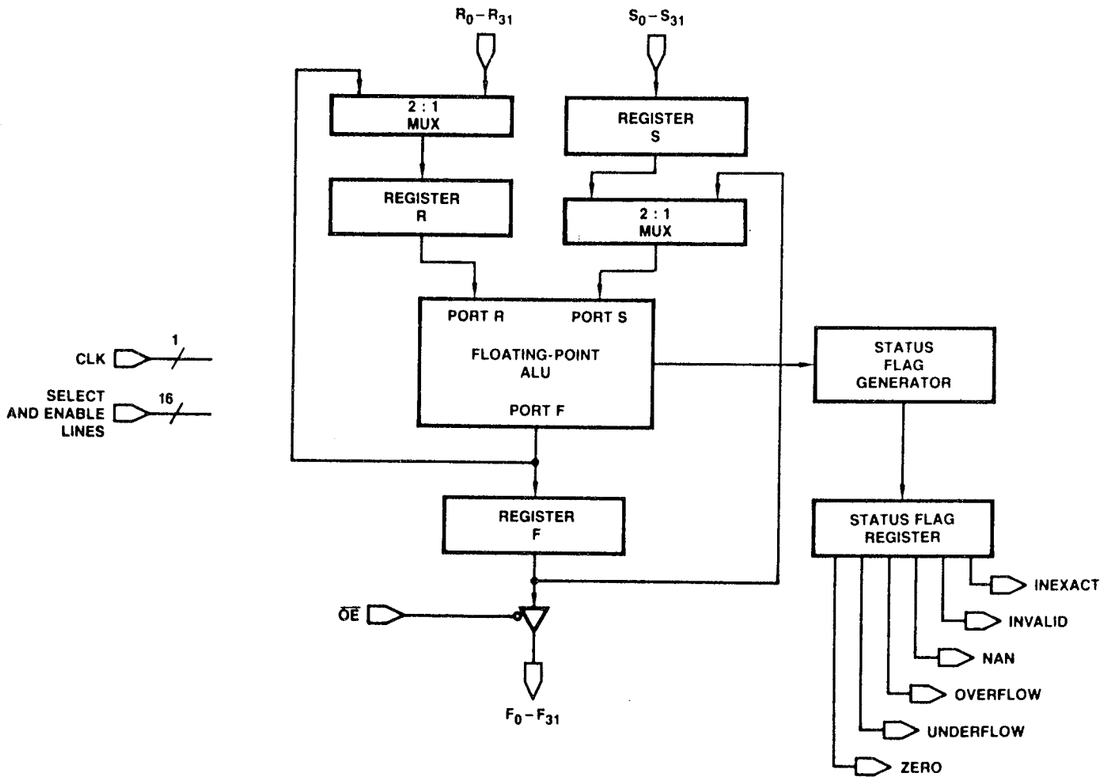
- R and S registers store input operands.
- F register stores the final result.
- Each register has a separate clock enable ($\overline{ENR}$, $\overline{ENS}$, $\overline{ENF}$)
- All data registers edge triggered.
- All data registers can be made transparent (FT_0, FT_1)

MUX SELECT

I_3	Data selected for floating-point ALU S port
0	Register S
1	Register F

I_4	Data selected for register R input
0	R bus
1	Floating-point ALU port F

Am29325 Block Diagram

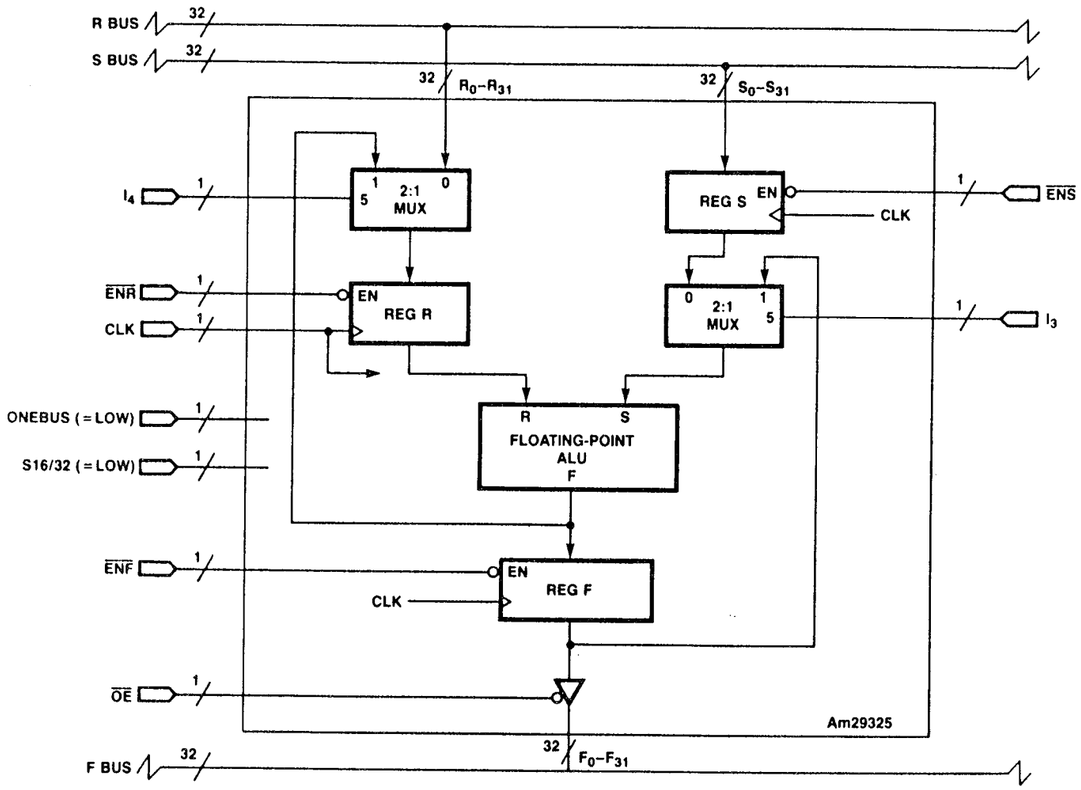


32/16-bit I/O

- Inputs and outputs to the Am29325 can be separately configurable to 32 or 16 bits.

S16/32	ONEBUS	I/O MODE
0	0	32-bit, two-input-bus mode
0	1	32-bit, single-input-bus mode
1	0	16-bit, two-input-bus mode
1	1	illegal

Functional Diagram for the 32-Bit, Two-Input-Bus Mode

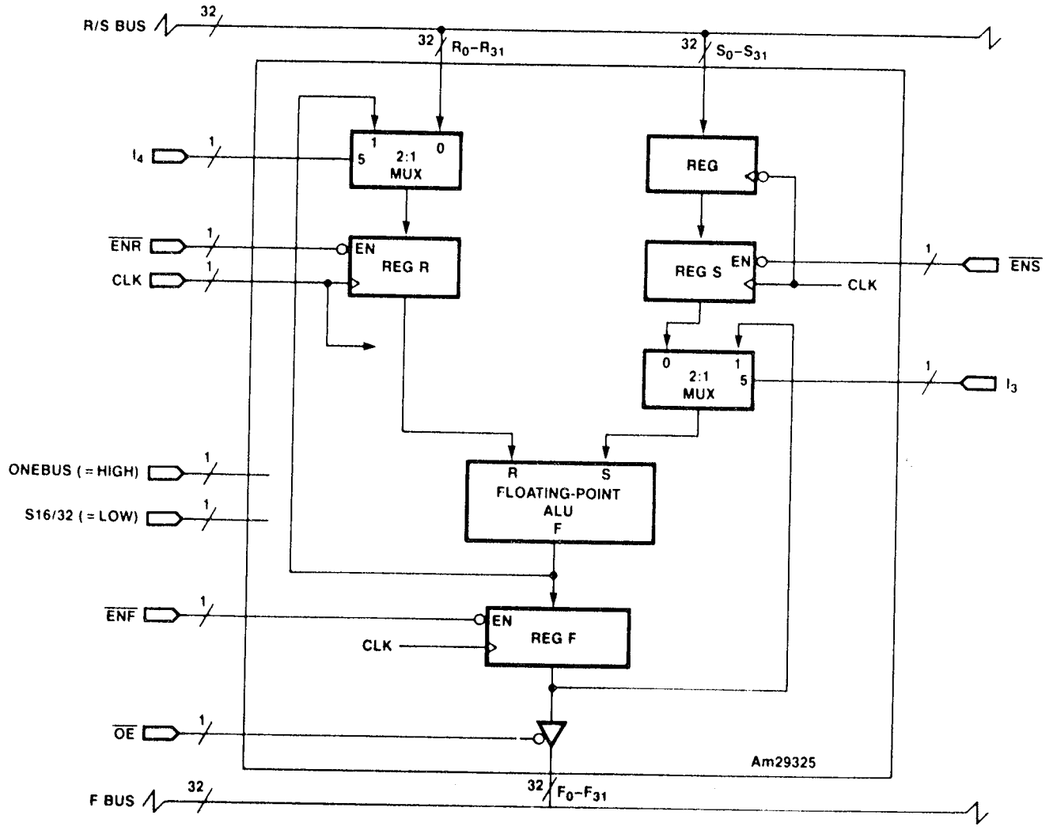


$S16/\overline{32}$ = ONEBUS = 0

Timing - 32-Bit, Two-Input-Bus Mode

Typical Bus Timing for the I/O Modes, with FT_0 = LOW, FT_1 = LOW

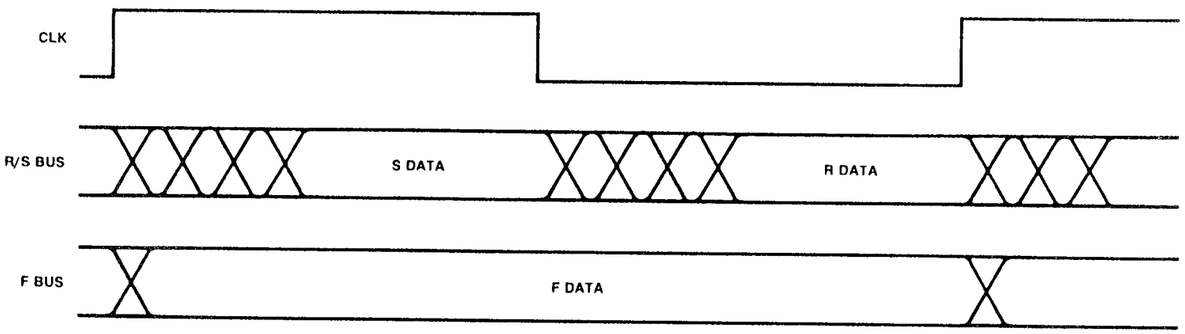
Functional Diagram for the 32-Bit, Single-Input-Bus Mode



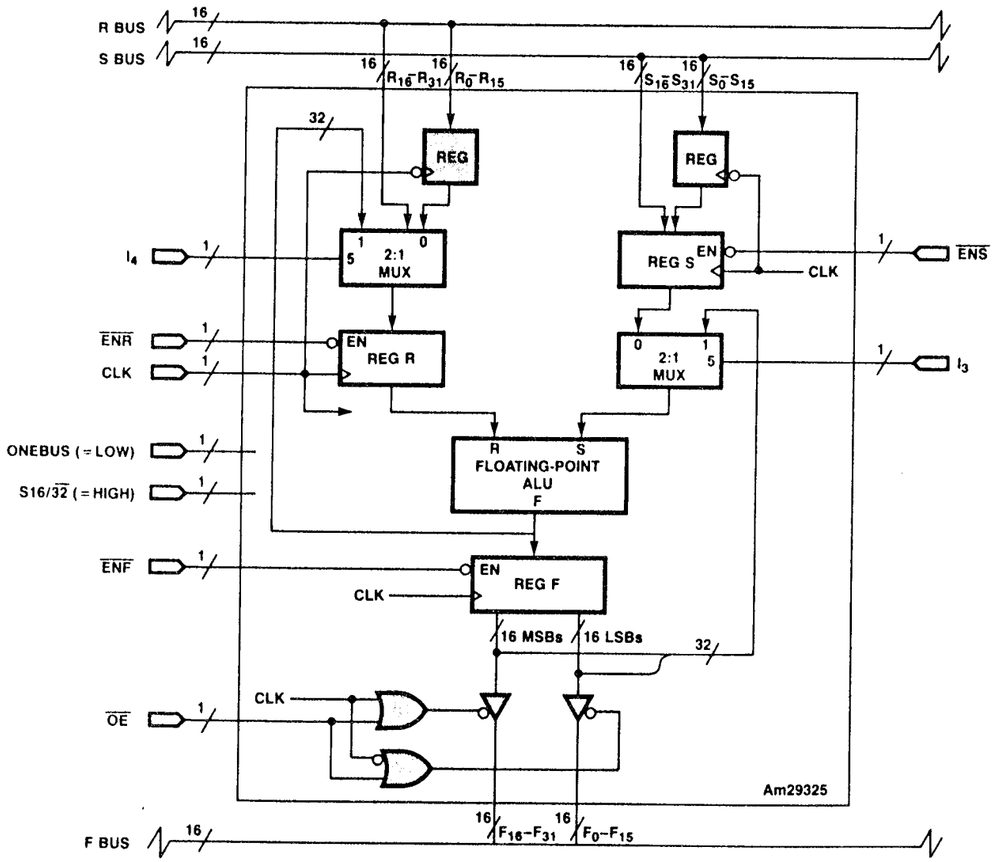
S16/$\overline{32}$ = 0
ONEBUS = 1

Timing - 32-Bit, Single-Input-Bus Mode

R and S inputs wired to single input bus

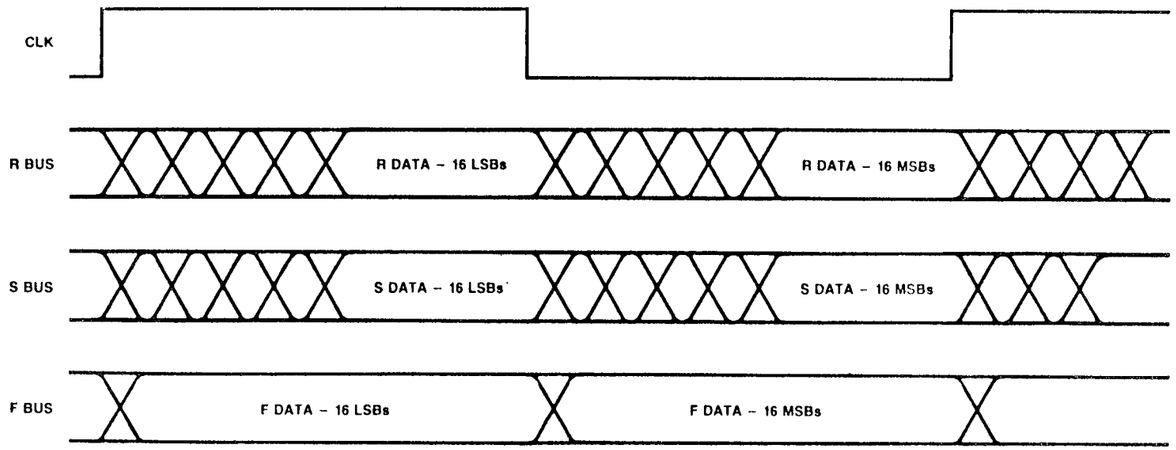


Functional Diagram for 16-Bit, Two-Input-Bus Mode



S16/$\overline{32}$ = 1
ONEBUS = 0

Timing - 16-Bit, Two-Input-Bus Mode



Floating Point Division on the Am29325

- Am29325 does not have a division operation.
- Evaluate-reciprocal-and-multiply algorithm is supported.

-- Am29325 Division Algorithm --

Let C = A/B, where C, A, and B are floating point numbers.

Then we can evaluate $C = A * \frac{1}{B}$ to get the same results.

Thus, we need a method for rapidly evaluating $\frac{1}{B}$.

A well-known technique is based on the Newton-Raphson method for finding the roots of an equation:

$$F(x) = 0$$

The roots of F(x) = 0 are found by the iterative equation:

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)}$$

If $F(x) = (\frac{1}{x} - B) = 0$, the the root of F(x) is $\frac{1}{B}$. In this case:

$$x_{i+1} = x_i - \frac{\frac{1}{x_i} - B}{-(x_i)^{-2}} = x_i (2 - B*x_i) = x_{i+1}$$

For this equation to converge the initial starting value for x_i (also called the seed) must fall in the following range:

$$0 < x_0 < \frac{2}{B} \quad \text{for } B > 0$$

$$\frac{2}{B} < x_0 < 0 \quad \text{for } B < 0$$

Note that the error $| x_i - x_{i+1} |$ reduces quadratically. Thus, the bits of accuracy in the result approximately double every iteration.

Example: Find the reciprocal of 7.25 (answer: 0.1378310345)

- First, find a seed (generally done by table look-up)

$$0 < x_0 < \frac{2}{7.25}$$

or, $0 < x_0 < 0.275862$

Suppose we choose $x_0 = 0.1$

- Next, iterate on x

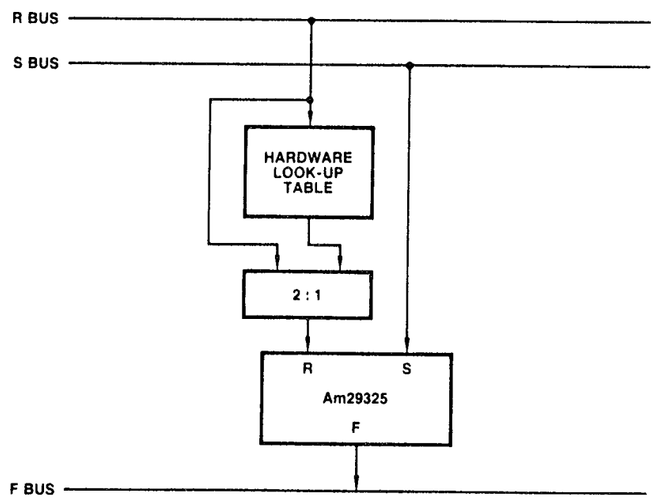
Iteration 1: $x_1 = x_0(2 - B*x_0)$

$= 0.1\ (2 - 7.25 \cdot 0.1)$

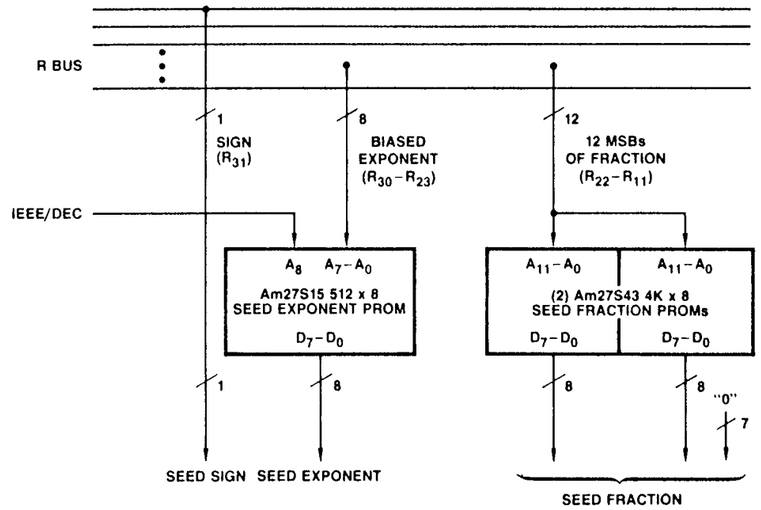
$= 0.1275$

Continuing for another two iterations, we find that the error after the third iteration is 4.5115×10^{-6}.

An approach to generating the seed value is to place a ROM look-up table between the R-Bus and the R input:



Both an exponent seed PROM and fraction seed PROM are needed:



Both the DEC format and IEEE format can be stored in the same PROMs.

On the next few slides we go step-by-step through the sequence of steps to calculate 1/B using the Am29325.

1. Place B on both the R and S buses. The 2:1 multiplexer at the output of the hardware look-up table should select the output of the look-up table.

2. Load the seed value x_0 into register R and load B into register S. Select the R*S operation.

3. Load product $B*x_0$ into register F. Select the 2-S operation, and select register F as the input to the ALU S port.

4. Load $2\text{-}Bx_0$ into register F. Select the R*S operation and select register F as the input to the ALU S port.

5. Load the value x_1 ($x_1 = x_0(2\text{-}Bx_0)$) into registers R and F. Select the R*S operation.

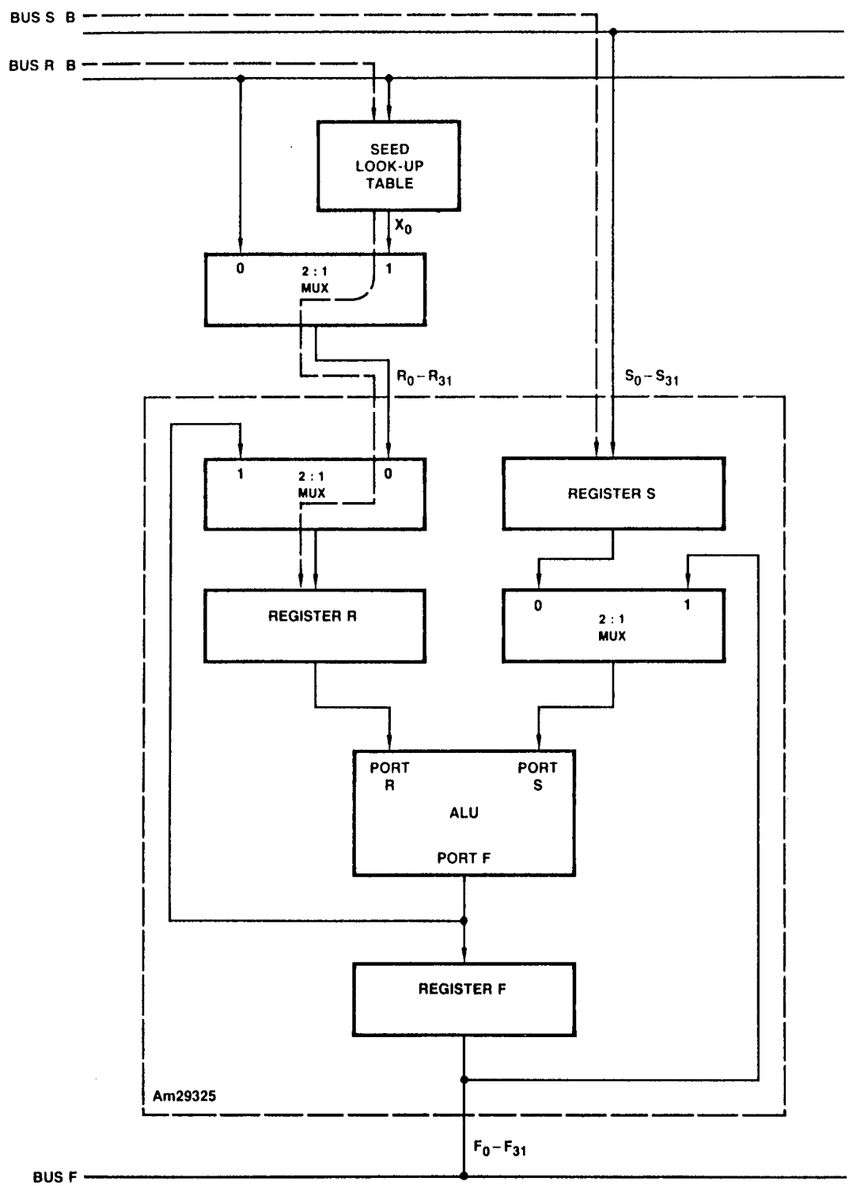
6. Repeat steps 3 through 5 until the result has the accuracy desired.

Sequence of Events for Evaluating Reciprocals

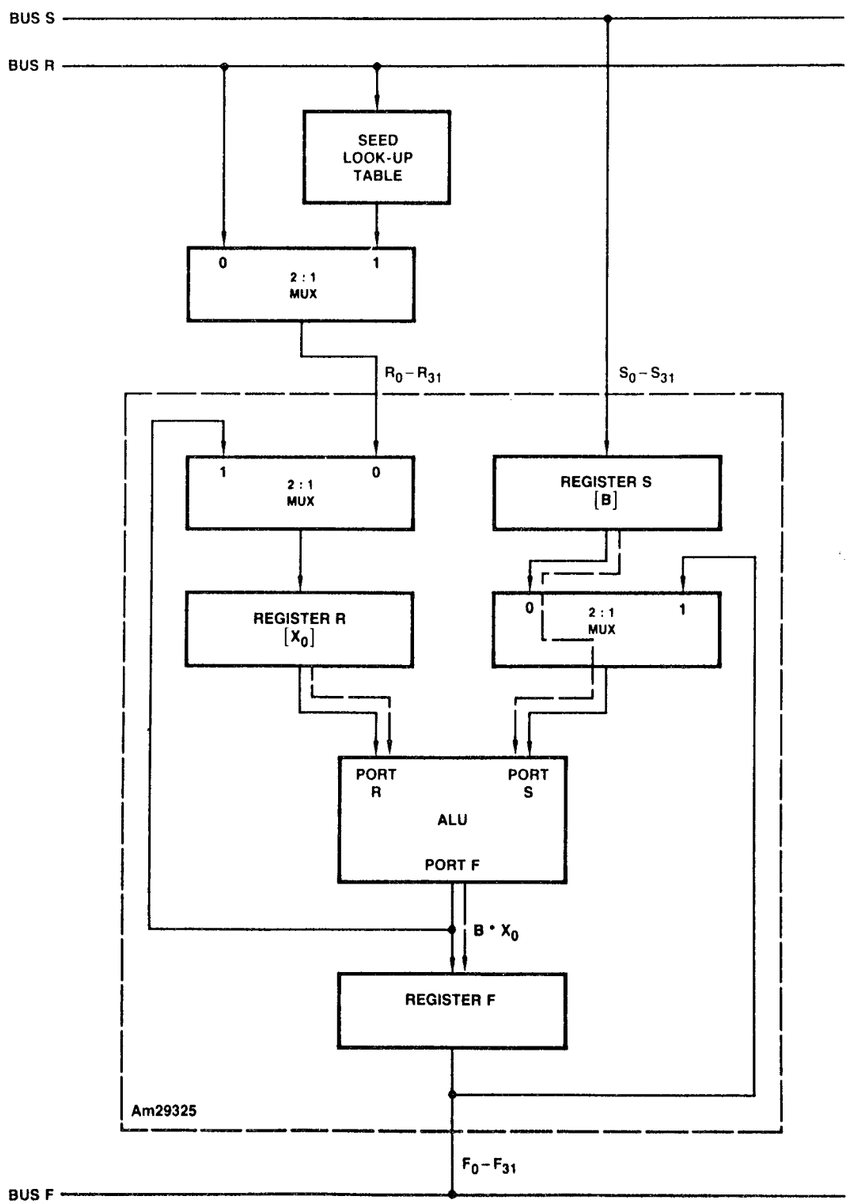
Clock Cycle	I_0-I_2	I_3	I_4	$\overline{ENR}$	$\overline{ENS}$	$\overline{ENF}$	Register R	Register S	Register F	
1	Y	X	0	0	0	X	–	–	–	
2	R TIMES S	0	X	1	1	0	X_0	B	–	
3	2 MINUS S	1	X	1	1	0	X_0	B	$B{\cdot}X_0$	First iteration
4	R TIMES S	1	1	0	1	0	X_0	B	$2-B{\cdot}X_0$	First iteration
5	R TIMES S	0	X	1	1	0	$X_1(=X_0(2-B{\cdot}X_0))$	B	$X_1(=X_0(2-B{\cdot}X_0))$	First iteration
6	2 MINUS S	1	X	1	1	0	X_1	B	$B{\cdot}X_1$	Second iteration
7	R TIMES S	1	1	0	1	0	X_1	B	$2-B{\cdot}X_1$	Second iteration
8	R TIMES S	0	X	1	1	0	$X_2(=X_1(2-B{\cdot}X_1))$	B	$X_2(=X_1(2-B{\cdot}X_1))$	Second iteration

X = DON'T CARE

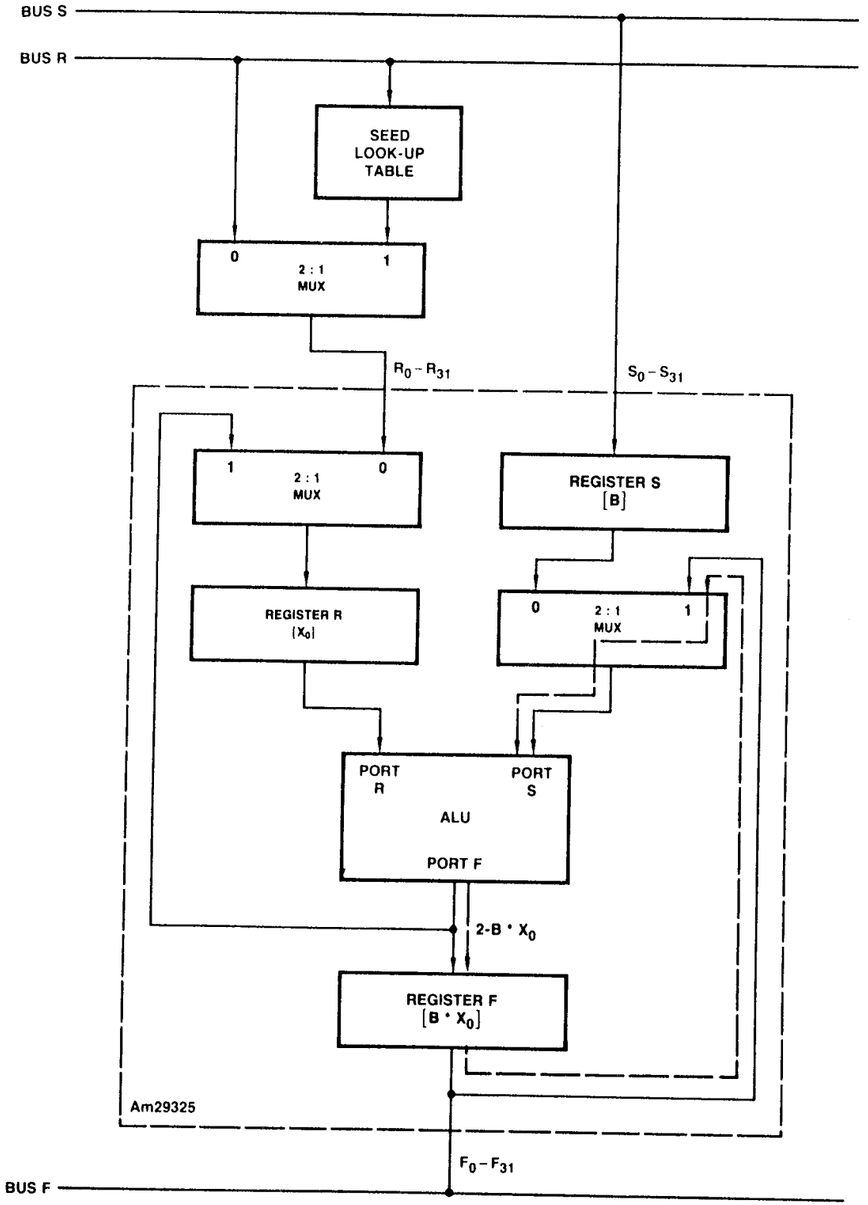
Data Flow for Step 1 of the Reciprocal Procedure



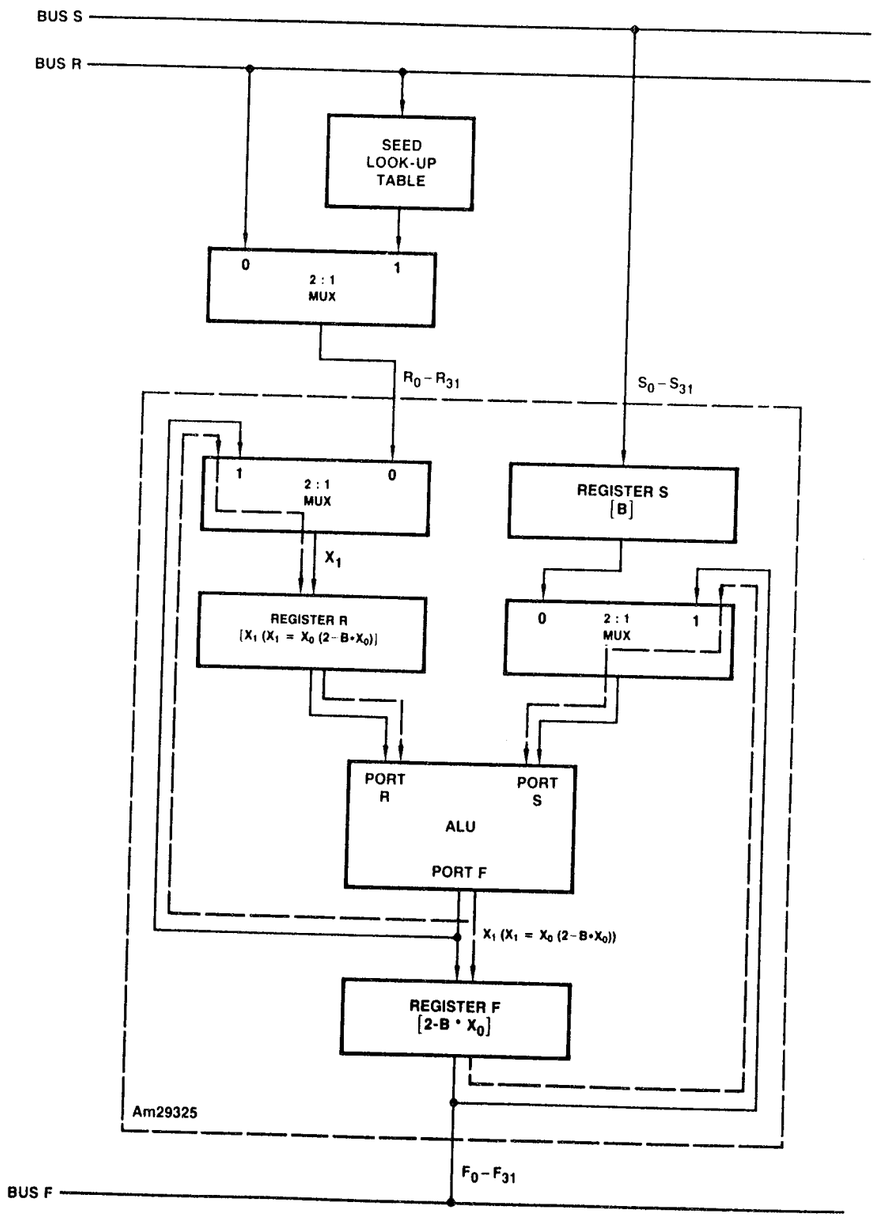
Data Flow for Step 2 of the Reciprocal Procedure



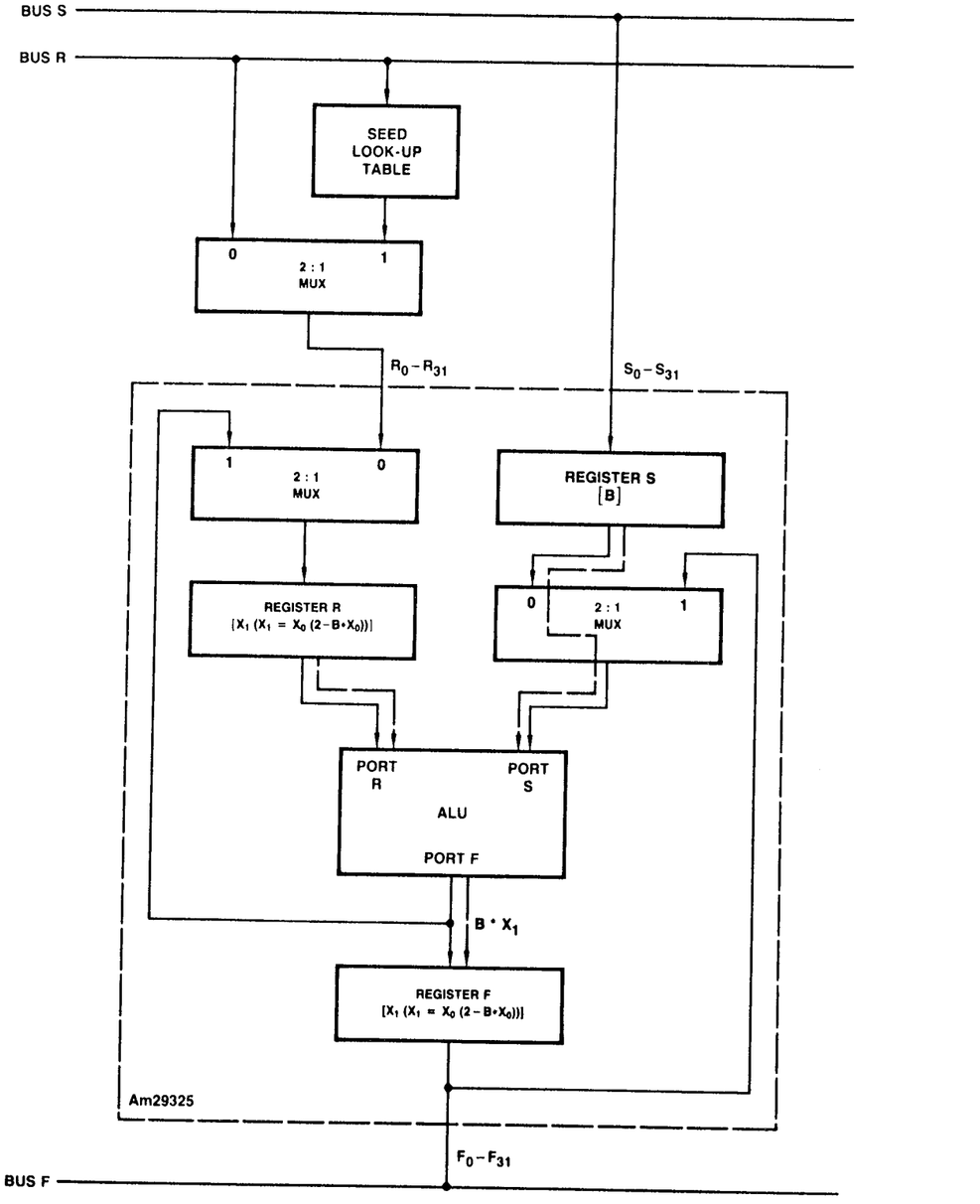
Data Flow for Step 3 of the Reciprocal Procedure



Data Flow for Step 4 of the Reciprocal Procedure



Data Flow for Step 5 of the Reciprocal Procedure



Example: Find the reciprocal of 25.3

$$\frac{1}{25.3} = 0.03952569_{10} = 3D21E5B1_{16} \quad \text{(IEEE)}$$

Clock Cycle	R Input	S Input	Register R	Register S	Register F	
1	3D21E800 (.03952789)	$41CA6666_{16}$ (25.3)	–	–	–	
2	–	–	$3D21E800_{16}$ (.03952789)	$41CA6666_{16}$ (25.3)	–	
3	–	–	$3D21E800_{16}$ (.03952789)	$41CA6666_{16}$ (25.3)	$3F8001D3_{16}$ (1.0000556)	
4	–	–	$3D21E800_{16}$ (.03952789)	$41CA6666_{16}$ (25.3)	$3F7FFC5A_{16}$ (.99984419)	
5	–	–	$3D21E5B1_{16}$ (.03952569)	$41CA6666_{16}$ (25.3)	$3D21E5B1_{16}$ (.03952569)	← Result of first iteration
6	–	–	$3D21E5B1_{16}$ (.03952569)	$41CA6666_{16}$ (25.3)	$3F7FFFFF_{16}$ (.99999994)	
7	–	–	$3D21E5B1_{16}$ (.03952569)	$41CA6666_{16}$ (25.3)	$3F800000_{16}$ (1.0)	
8	–	–	$3D21E5B1_{16}$ (.03952569)	$41CA6666_{16}$ (25.3)	$3D21E5B1_{16}$ (.03952569)	← Result of second iteration

Am29325 Pin Description

R_0-R_{31}	R operand bus, input. R_0 is the least-significant bit.
S_0-S_{31}	S operand bus, input. S_0 is the least-significant bit.
F_0-F_{31}	F operand bus, output, F_0 is the least-significant bit.
CLK	Clock input for the internal registers.
$\overline{ENR}$	Register R clock enable, input. When $\overline{ENR}$ is LOW, register R is clocked on the LOW-to-HIGH transition of CLK. When $\overline{ENR}$ is HIGH, register R retains the previous contents.
$\overline{ENS}$	Register F clock enable, input. When $\overline{ENS}$ is LOW, register S is clocked on the LOW-to-HIGH transition of CLK. When $\overline{ENS}$ is HIGH, register S retains the previous contents.
$\overline{ENF}$	Register F clock enable, input. When $\overline{ENF}$ is LOW, register F is clocked on the LOW-to-HIGH transition of CLK. When $\overline{ENF}$ is HIGH, register F retains the previous contents.
FT_0	Input register feedthrough control, input. When FT_0 is HIGH, registers R and S are transparent.
FT_1	Output register feedthrough control, input. When FT_1 is HIGH, register F and the status flag register are transparent.
I_0-I_2	Operation select lines, input. Used to select the operation to be performed by the ALU. See the ALU Operation Select Table for a list of operations and the corresponding codes.
I_3	ALU S port input select, input. A LOW on I_3 selects register S as the input to the ALU S port. A HIGH on I_3 selects register F as the input to the ALU S port.

I_4 Register R input select, input. A LOW on I_4 selects R_0-R_{31} as the input to register R. A HIGH selects the ALU F port as the input to register R.

IEEE/$\overline{\text{DEC}}$ IEEE/$\overline{\text{DEC}}$ mode select, input. When IEEE/$\overline{\text{DEC}}$ is HIGH, IEEE mode is selected. When IEEE/$\overline{\text{DEC}}$ is LOW, DEC mode is selected.

INEXACT Inexact result flag, output. A HIGH indicates that the final result of the last operation was not infinitely precise, due to rounding.

INVALID Invalid operation flag, output. A HIGH indicates that the last operation performed was invalid, e.g., ∞ times 0.

NAN Not-a-number flag, output. A HIGH indicates that the final result produced by the last operation is not to be interpreted as a number. The output in such cases is either an IEEE Not-a-Number (NAN) or a DEC reserved operand.

$\overline{\text{OE}}$ Output enable, input. When $\overline{\text{OE}}$ is LOW, the contents of register F are placed on F_0-F_{31}. When $\overline{\text{OE}}$ is HIGH, F_0-F_{31} assume a high-impedance state.

ONEBUS Input bus configuration control, input. A LOW on ONEBUS configures the input bus circuitry for two-input bus operation. A HIGH on ONEBUS configures the input bus circuitry for single-input bus operation.

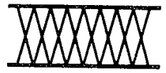
OVERFLOW Overflow flag, output. A HIGH indicates that the last operation produced a final result that overflowed the floating-point format.

PROJ/$\overline{\text{AFF}}$ Projective/affine mode select, input. Choice of projective or affine mode determines the way in which infinties are handled in IEEE mode. A LOW on PROJ/$\overline{\text{AFF}}$ selects affine mode; a HIGH selects projective mode.

-- Systems Issues --

Am29300 Timing

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 Change From L to H
	Don't Care: Any Change Permitted	Changing: State Unknown
	Does Not Apply	Center Line is High Impedance "Off" State

Am29332 TIMING (Commercial) in ns. Guaranteed over the operating range.

A. Combinational Propagation Delays

From Input	To Output: P-Error	PY(3–0)	Y(31–0)	Status(C,Z,V,N,L)	Status(Reg)	MS-Error
PA(3–0)	19					—
PB(3–0)						
DA(31–0)	28	42	35	43		49
DA(31–0)						
I(0–8)		53	47	48		55
W(4–0)		40	34	35		41
P(0–5)		40	42	43		45
CP		47	41	42	20	—
Reg Status				16		—
Macro Carry			31	34		37
Macro Link			33	37		38
Macro/$\overline{\text{Micro}}$ SEL			33	37		38
Borrow			33	37		38
PY(3–0)						20
Y(31–0)						19
Status Output						21
Parity Error						20
HOLD				22		29

B. Setup and Hold Times with Respect to CP Rising Edge

Input	CP: Setup Time	CP: Hold Time
DA(31–0)	31	1
DB(31–0)		
I(6–0)	37	2
W(4–0)	28	0
P(5–0)	28	0
Borrow	22	1
Macro Carry In	21	0
Macro Link	22	1
Macro/$\overline{\text{Micro}}$ SEL	22	1
HOLD	11	1
I(8–7)	30	1

C. Clock Pulse Characteristics

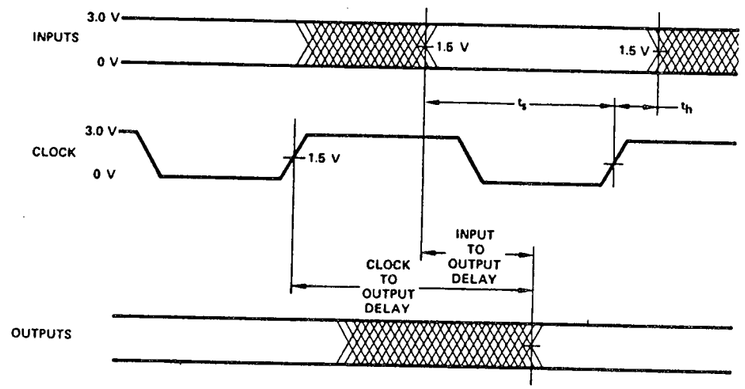
Min. LOW Time	Min. HIGH Time
20	20

Am29332 Timing

D. Enable/Disable Times (nsec)
(C_L = 5 pF for Disable Only)

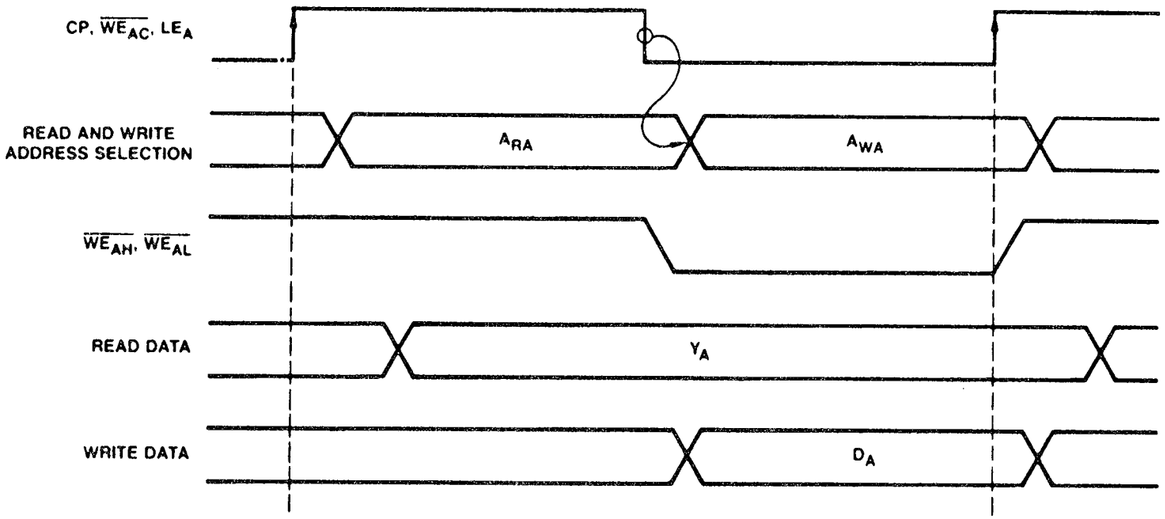
From Input	To Output	Enable: t_{PZH}	Enable: t_{PZL}	Disable: t_{PHZ}	Disable: t_{PLZ}
$\overline{\text{OE}}$-Y	Y_{31-0}, PY_{3-0}	25	25	25	25
SLAVE	Y_{31-0}, PY_{3-0}, Status, P-Error	25	25	25	25

SWITCHING WAVEFORMS



WFR02990

Am29334 Timing



Read through Y_A and Write through D_A in a Single Cycle (Two Bytes)

Am29334 Timing

SWITCHING CHARACTERISTICS (Commercial Only)

Parameters	From	To	Test Conditions	Delay (ns)
Access Time	A_{RA} or A_{RB}	Y_A or Y_B	LE_A or LE_B = H	24
Turn-On Time	$\overline{OE}_A$ or $\overline{OE}_B$ = L	Y_A or Y_B		20
Turn-Off Time**	$\overline{OE}_A$ or $\overline{OE}_B$ = H	Y_A or Y_B = Z	C_L = 5 pF load	16
Enable Time	LE_A or LE_B = H	Y_A or Y_B		16
Transparency	$\overline{WE}_A$ or $\overline{WE}_B$ = L	Y_A or Y_B	LE_A or LE_B = H	32
Transparency	D_A or D_B	Y_A or Y_B	LE_A or LE_B = H $\overline{WE}_A$ or $\overline{WE}_B$ = L	33

Minimum Setup and Hold Time

Parameters	For	WRT	Delay (ns)
Data Setup	D_A or D_B	$\overline{WE}_A$ or $\overline{WE}_B$ (L TO H)	9
Data Hold	D_A OR D_B	$\overline{WE}_A$ or $\overline{WE}_B$ (L TO H)	2
Address Setup	A_{WA} or A_{WB}	$\overline{WE}_A$ or $\overline{WE}_B$ (H TO L)	0
Address Hold	A_{WA} or A_{WB}	$\overline{WE}_A$ or $\overline{WE}_B$ (L TO H)	3
Address Setup	A_{RA} or A_{RB}	LE_A or LE_B (H TO L)	7
Address Hold	A_{RA} or A_{RB}	LE_A or LE_B (H TO L)	4
Latch close before Write	LE_A or LE_B (H TO L)	$\overline{WE}_A$ or $\overline{WE}_B$ (H TO L)	0

Minimum Pulse Widths

Parameters	Input	Pulse	Delay (ns)
Write Pulse	$\overline{WE}_A$ or $\overline{WE}_B$	HIGH – LOW – HIGH	18
Latch Data Capture	LE_A or LE_B	LOW – HIGH – LOW	10

$WE_A = WE_{AC} \bullet (WE_{AL} + WE_{AH})$
$WE_B = WE_{BC} \bullet (WE_{BL} + WE_{BH})$

** Y_A and Y_B Are Tested Independently

Am29331 Timing Preliminary

A. Combinational Propagation Delays

From Input	To Output (Note 2) Y	D	$\overline{INTA}$	A-FULL	EQUAL	ERROR
D	20*	–	–	–	26	28
A	19	–	–	–	26	28
M	19	–	–	–	26	28
Y	–	–	–	–	25	27
I	28	31	–	–	29	29
T	30	–	–	–	30	32
S	30	–	–	–	30	32
CP	24	20/Z	–	20	31	33
$\overline{RST}$	27	Z	16	–	27	29
FC	21	23	–	–	28	30
INTR	Z	–	12	–	35	31
INTEN	Z	–	12	–	35	31
HOLD	Z	–	Z	Z	25/Z	29
OED	–	Z	–	–	–	19
$\overline{INTA}$	–	–	–	–	–	21
A-FULL	–	–	–	–	–	21
EQUAL	–	–	–	–	–	21
$\overline{C_{in}}$	21	–	–	–	28	30
SLAVE	Z	Z	Z	Z	Z	–

Notes: 1. C_L = 50 pF; C_L = 5 pF for Disable Time only.
2. Z = Three-state output path, use table B.

*This includes using D as Select lines for Multiway sets.

B. Output Enable/Disable Times

Inputs	Outputs	Enable	Disable
$\overline{RST}$	Y	30	30
INTR			
INTEN			
HOLD			
SLAVE			
OED	D	30	30
$\overline{RST}$			
SLAVE			
CP	D	30	30
HOLD	$\overline{INTA}$	30	30
	A-FULL		
SLAVE	EQUAL		

C. Minimum Clock Requirement

Min. LOW Time	Min. HIGH Time
20	20

Am29331 Timing Preliminary

D. Setup and Hold Times

Input	CP: Setup Time Before H → L	Hold Time After H → L	Setup Time Before L → H	Hold Time After L → H
D	–	–	15	4
A	–	–	14	4
M	–	–	14	3
Y	–	–	10	4
I	15	Do Not Change		2
FC	15	Do Not Change		1
T	–	–	20	1
S	–	–	20	1
$\overline{RST}$	–	–	15	2
INTR	–	–	12	3
INTEN	–	–	*2	3
HOLD	–	–	12	4
$\overline{C_{in}}$	–	–	*16	3

Am29323 Timing

Am29323
Preliminary Switching Characteristics
(Guaranteed Over Temperature and Voltage Range)

		COMMERCIAL		MILITARY		
Parameter	**Description**	**Min**	**Max**	**Min**	**Max**	**Units**
Tmc	**Clocked Multiply Time**		**40**		**50**	**ns**
Tmuc	**Unclocked Multiply Time**		**50**		**56**	**ns**
Ts	**X, Y, INSTR, EN- Setup Time**		**12**		**12**	**ns**
Th	**X, Y, INSTR, EN- Hold Time**		**0**		**0**	**ns**
Tpwh	**Clock Pulse Width High**	**10**		**10**		**ns**
Tpwl	**Clock Pulse Width Low**	**10**		**10**		**ns**
Tpdp	**Clock to Product**		**15**		**17**	**ns**
Tpz	**OE-, SLAVE, PSEL0, PSEL1 Disable Time**		**13**		**15**	**ns**
Tzp	**OE-, SLAVE, PSEL0, PSEL1 Enable Time**		**13**		**15**	**ns**
Tdpe	**Data to PARERR**		**17**		**18**	**ns**
Tdhe	**Data to HARDERR**		**16**		**17**	**ns**

Am29325/15 Switching Characteristics Over Operating Range

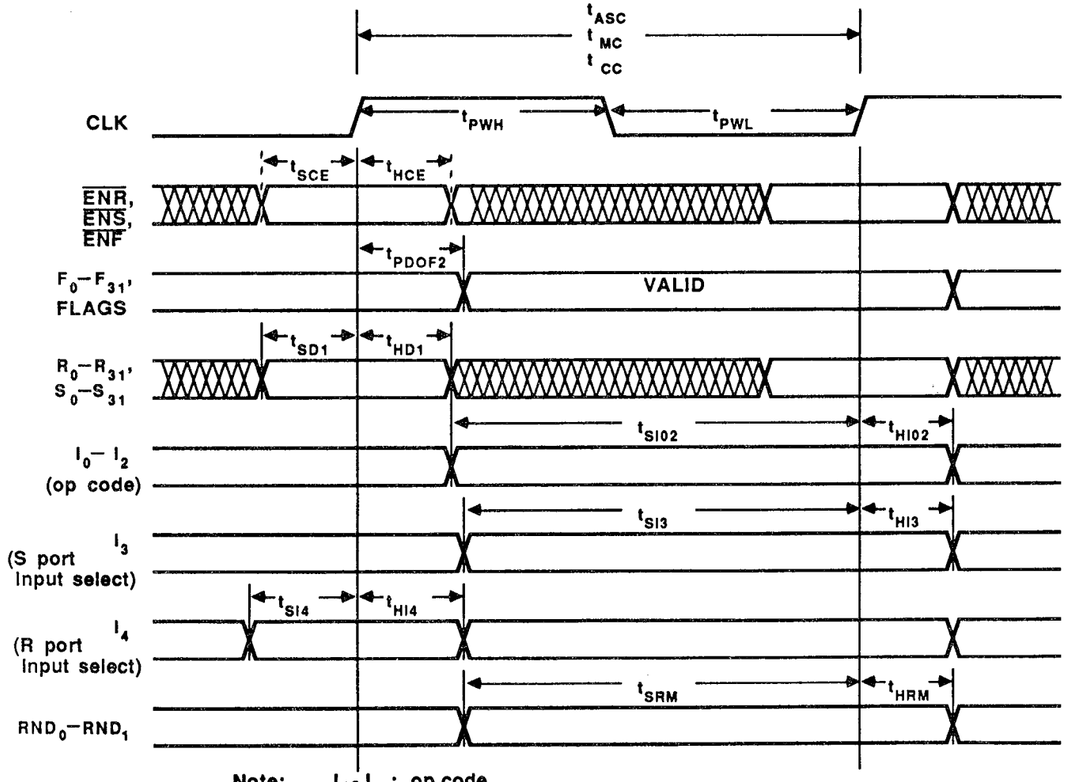
Preliminary

	Parameters	Description		Test Conditions	COM'L (Note 2) T_A = 0 to +70°C V_{CC} = +5V ±5% Min	Max	Units
1	t_{ASC}	Clocked Add, Subtract Time (R PLUS S, R MINUS S, 2 MINUS S)					ns
2	t_{MC}	Clocked Multiply Time (R TIMES S)					ns
3	t_{CC}	Clocked Conversion Time (INT-TO-FP, FP-TO-INT, IEEE-TO-DEC, DEC-TO-IEEE)					ns
4	t_{ASUC}	Unclocked Add, Subtract Time (R, S to F, Flags) for R PLUS S, R MINUS S, and 2 MINUS S instructions				150	ns
5	t_{MUC}	Unclocked Multiply Time (R, S to F, Flags) for R TIMES S Instruction		FT_0 = HIGH FT_1 = HIGH		150	ns
6	t_{CUC}	Unclocked Conversion Time (R, S to F, Flags) for INT-TO-FP, FP-TO-INT, IEEE-TO-DEC and DEC-TO-IEEE Instructions				150	ns
7	t_{PWH}	Clock Pulse Width HIGH					ns
8	t_{PWL}	Clock Pulse Width LOW					ns
9	t_{PDOF1}	Clock to F_0-F_{31} and Flag Outputs		FT_0 = LOW FT_1 = HIGH			ns
10	t_{PDOF2}			FT_1 = LOW			ns
11	t_{PZL}	$\overline{OE}$ Enable Time	Z to LOW			35	ns
12	t_{PZH}		Z to HIGH			35	ns
13	t_{PLZ}	$\overline{OE}$ Disable Time	LOW to Z			35	ns
14	t_{PHZ}		HIGH to Z			35	ns
15	t_{PZL16}	Clock ↑ to F_0-F_{15} Enable, 16-Bit I/O Mode	Z to LOW	S16/$\overline{32}$ = HIGH ONEBUS = LOW			ns
16	t_{PZH16}		Z to HIGH				ns
17	t_{PLZ16}	Clock ↓ to F_0-F_{15} Disable, 16-Bit I/O Mode	LOW to Z				ns
18	t_{PHZ16}		HIGH to Z				ns
19	t_{PZL16}	Clock ↓ to F_{16}-F_{31} Enable, 16-Bit I/O Mode	Z to LOW	S16/$\overline{32}$ = HIGH ONEBUS = LOW			ns
20	t_{PZH16}		Z to HIGH				ns
21	t_{PLZ16}	Clock ↑ to F_{16}-F_{31} Disable, 16-Bit I/O Mode	LOW to Z				ns
22	t_{PHZ16}		HIGH to Z				ns
23	t_{SCE}	Register Clock Enable Setup Time		FT_0 = LOW FT_1 = LOW			ns
24	t_{HCE}	Register Clock Enable Hold Time		FT_0 = LOW FT_1 = LOW			ns
25	t_{SD1}	R_0-R_{31}, S_0-S_{31} Setup Time (Note 1)		FT_0 = LOW			ns
26	t_{HD1}	R_0-R_{31}, S_0-S_{31} Hold Time (Note 1)					ns
27	t_{SD2}	R_0-R_{31}, S_0-S_{31} Setup Time (Note 1)		FT_0 = HIGH FT_1 = LOW			ns
28	t_{HD2}	R_0-R_{31}, S_0-S_{31} Hold Time (Note 1)					ns
29	t_{SI02}	I_0-I_2 Instruction Select Setup Time		FT for Destination Register = LOW			ns
30	t_{HI02}	I_0-I_2 Instruction Select Hold Time					ns
31	t_{PDI02}	I_0-I_2 Instruction Select to F_0-F_{31}, Flags		FT_1 = HIGH		145	ns
32	t_{SI3}	I_3 Port S Input Select Setup Time		FT_1 = LOW			ns
33	t_{HI3}	I_3 Port S Input Select Hold Time					ns
34	t_{SI4}	I_4 Register R Input Select Setup Time (Note 1)		FT_0 = LOW			ns
35	t_{HI4}	I_4 Register R Input Select Hold Time (Note 1)					ns
36	t_{SRM}	Round Mode Select Setup Time		FT for Destination Register = LOW			ns
37	t_{HRM}	Round Mode Select Hold Time					ns
38	t_{PRF}	Round Mode Select to F_0-F_{31}, Flags		FT_1 = HIGH		40	ns

Notes: 1. See timing diagram for desired mode of operation to determine clock edge to which these setup and hold times apply.
2. At air velocity of 200 linear feet per minute.

Am29325 Timing

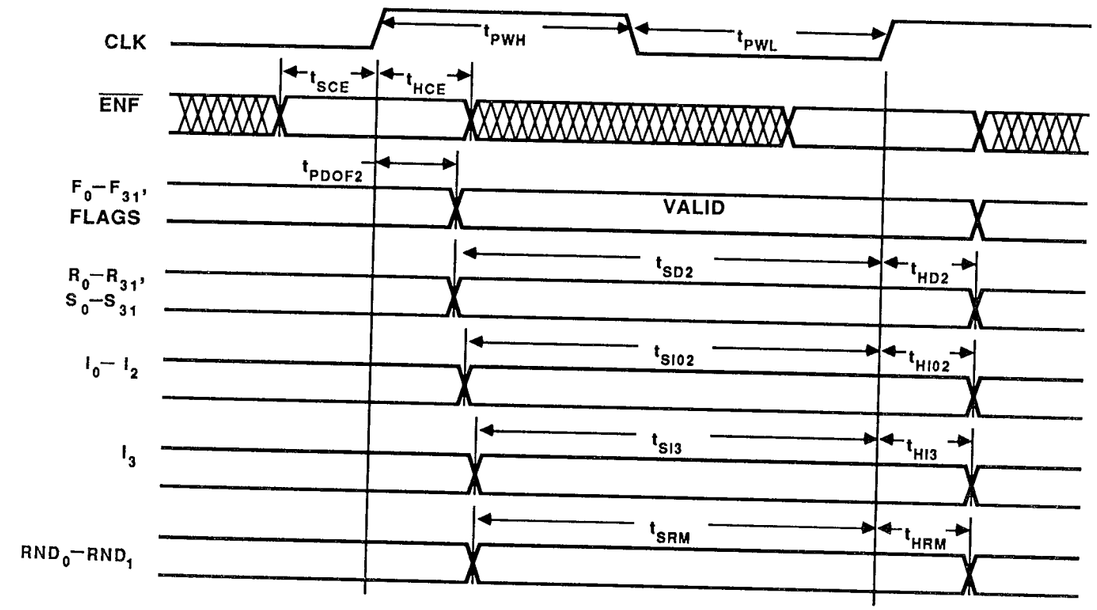
Clocked Operation: FT0 = LOW (R,S registers selected)
FT1 = LOW (F, status register selected)



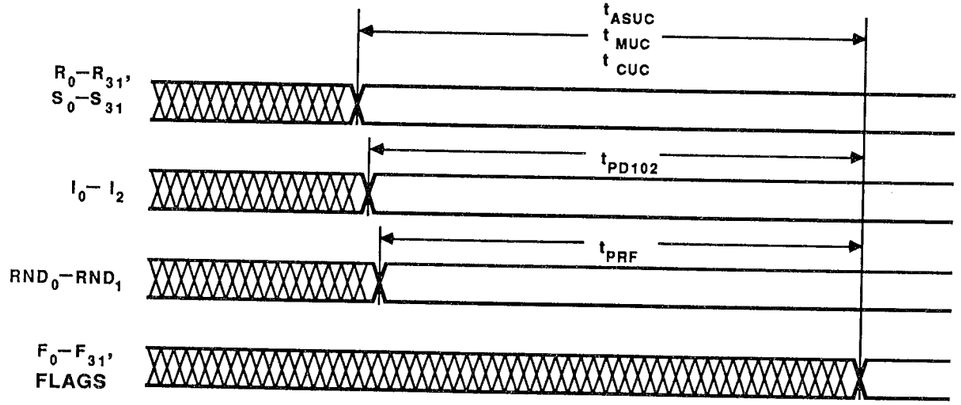
Note: I_0-I_2 : op code
I_3 : S port Input select
I_4 : R port Input select

Am29325 Timing

Clocked Operation: FT0 = HIGH (R,S transparent)
FT1 = LOW (F, status selected)

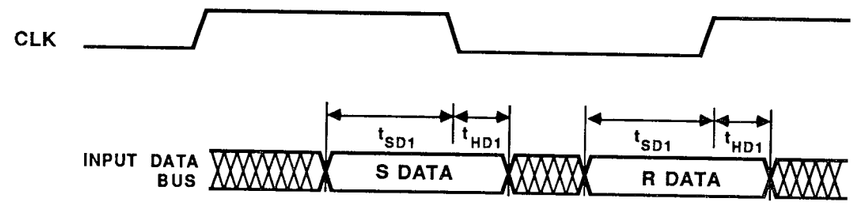


Flow-Through Operation (FT0 = HIGH, FT1 = HIGH)
(R,S transparent)
(F, status register selected)



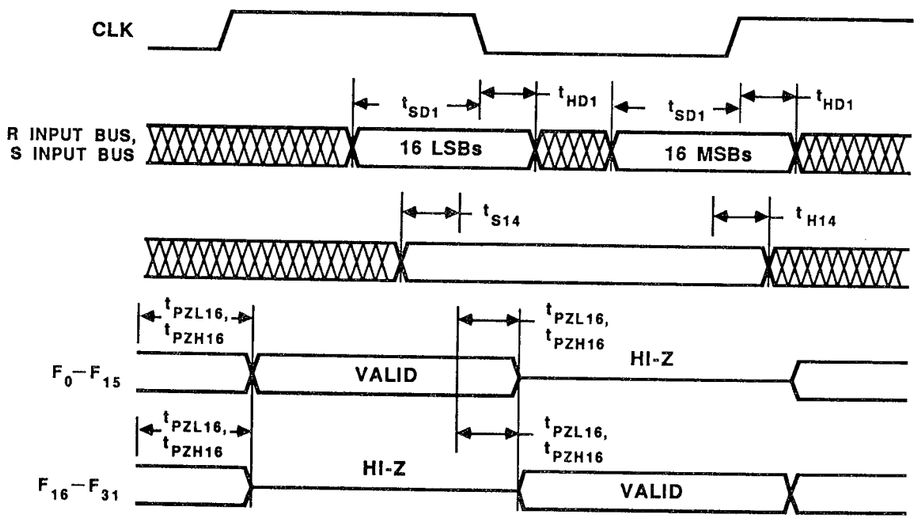
- **Bus Mode Timing**

1) 32-bit, Single-Input-Bus Mode



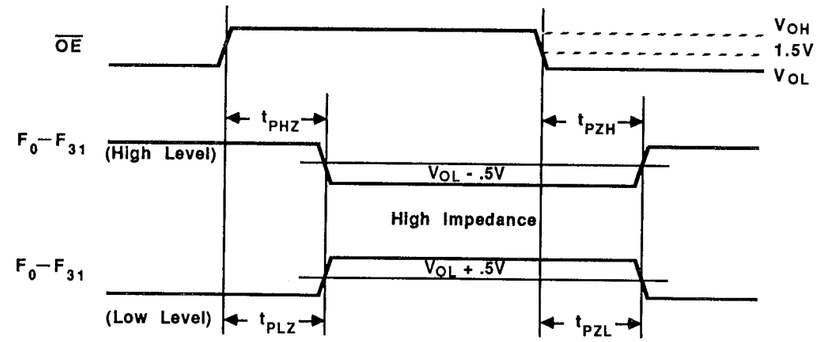
Am29325 Timing

2) 16-bit, Two-Input-Bus Mode



Am29325 Timing

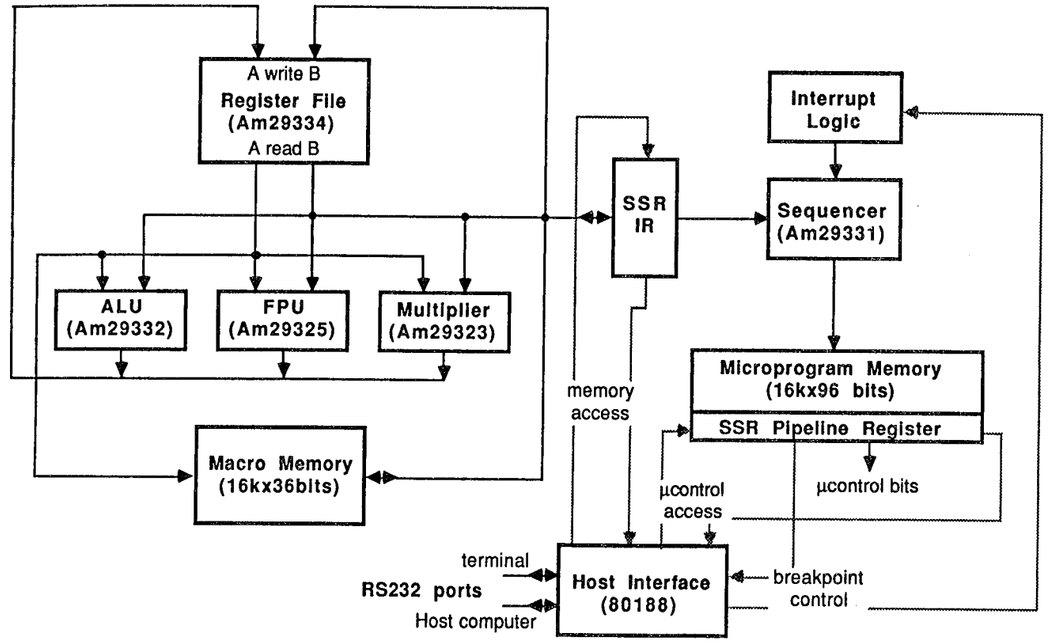
3) Output Enable/Disable Timing



-- Systems Issues --

Am29300 Family Evaluation Board

Am29300 Family Evaluation Board

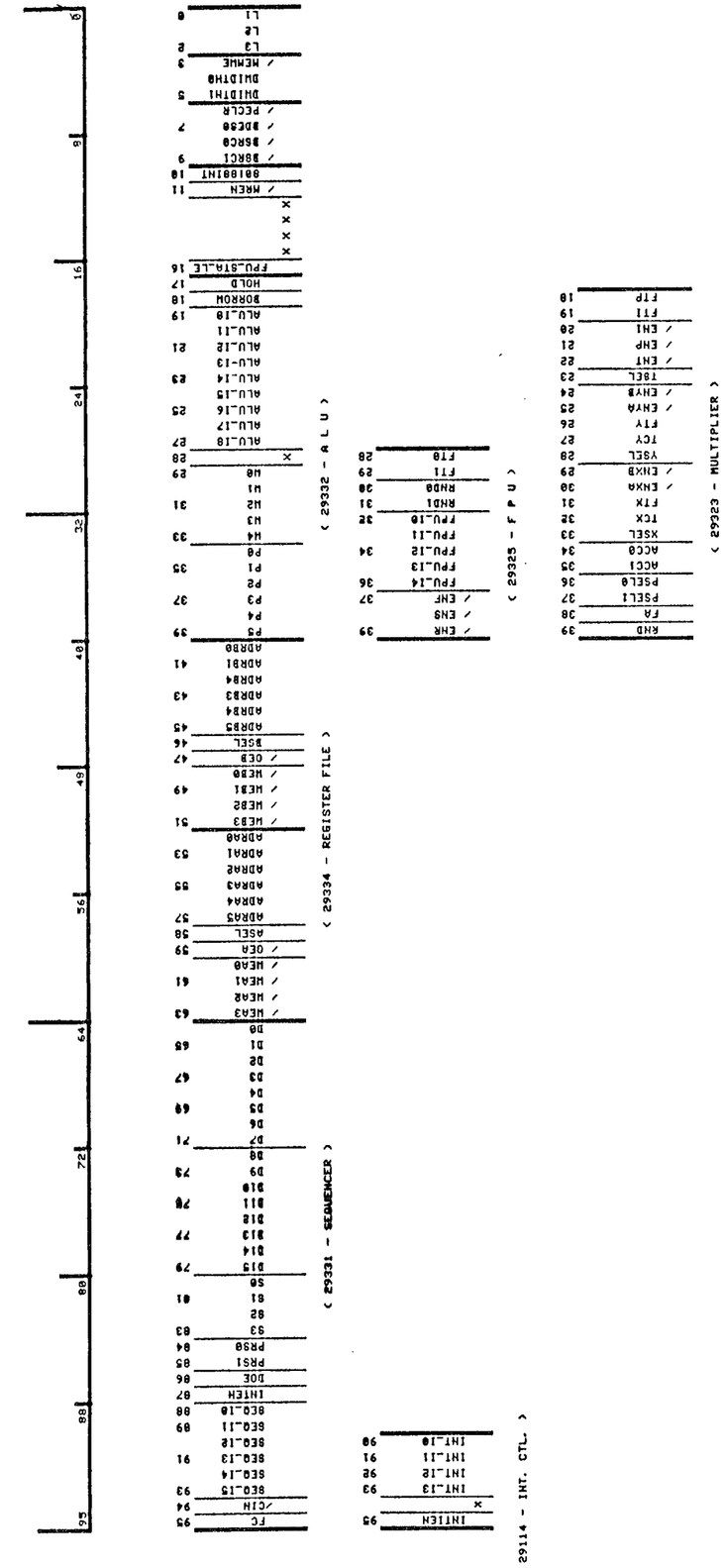


Am29300 Evaluation Board

Microword Structure Example

- To illustrate a typical application of the Am29300 family, the microword used to control the Am29300 Evaluation Board is described in the following pages.
- Bit fields are shown to control the following:
  - Sequencer (Am29331)
  - Interrupt Controller (Am29114)
  - Register File (Am29334)
  - ALU (Am29332)
  - Floating-Point Unit (Am29325)
  - Multiplier (Am29323)
- Note that some fields are overlayed (bits 96-90 and 39-18) to economize on microword width and thus on memory.
  - Am29331 and Am29114 share bits 95-90, while the Am29332, Am29325 and Am29323 share parts or all of bits 39-18.
  - The latter field can also be omitted if an instruction doesn't need outside computations.
- The tables show all control bits for the devices.
  - The mnemonics for pins are not repeated exactly in the micrword's notation (e.g. BS0 matches BSRC0, but the connections are there.
  - An important feature of the tables is that they not only show all control lines for a device, but all the instructions it can execute as well.

Am29300 Example Microword Fields



Am29332 Control-Field Structure

CTL-1

FC	1	Force Continue : 1 - 29114 0 - 29331
CIN/	1	Carry Input : 1 - No increment 0 - Increment
31I	6	Sequencer Instructions (Am29331)
		00 BRCC_D 10 BRNC_D 20 BRA_D 30 CONT 01 CCC_D 11 CNC_D 21 CALL_D 31 FOR_D 02 XTCC_D 12 XTNC_D 22 EXIT_D 32 DECR 03 DJCC_D 13 DJNCC_D 23 DJMP_D 33 LOOP 04 BRCC_A 14 BRNC_A 24 BRA_A 34 POP_D 05 CCC_A 15 CNC_A 25 CALL_A 35 PUSH_D 06 XTCC_A 16 XTNC_A 26 EXIT_A 36 RESET_SP 07 DJCC_A 17 DJNCC_A 27 DJMP_A 37 FOR_A 08 BRCC_M 18 BRNC_M 28 BRA_M 38 POP_C 09 CCC_M 19 CNC_M 29 CALL_M 39 PUSH_C 0A XTCC_M 1A XTNC_M 2A EXIT_M 3A SWAP 0B DJCC_M 1B DJNCC_M 2B DJMP_M 3B STACK_C 0C BRCC_S 1C BRNC_S 2C BRA_S 3C LOAD_D 0D CCC_S 1D CNC_S 2D CALL_S 3D LOAD_A 0E XTCC_S 1E XTNC_S 2E EXIT_S 3E SET 0F DJCC_S 1F DJNCC_S 2F DJSP_S 3F CLEAR
14I	4	Interrupt Controller Instructions (Am29114)
		0-MCLR 1-CHSR 2-CCIR 3-NOOP 4-BSMK 5-BCMK 6-LDMK 7-RDMK 8-BSSR 9-BCSR A-LDSR B-RDSR C-BSIR D-BCIR E-LDIR F-RDIR
ITE	1	InTerrupt Enable : 1 - Enable 0 - Disable
DOE	1	D-bus Output Enable : 1 - Enable 0 - Disable
YSE	2	Y-Bus Selection 0 - ALU 1 - MUL 2 - FPU 3 - DISABLE
S	4	Test Input Selection
		0 - FPU (325) INEXECT 8 - ALU (332) C Carry 1 - " IMVALID 9 - " N Negative 2 - " NAN A - " V Overflow 3 - " OVERFLOW B - " Z Zero 4 - " UNDERFLOW C - " C + Z 5 - " ZERO D - " C~ + Z 6 - From 80188's PIO E - " N EXOR V 7 - ALU (332) Z Zero F - " (N EXOR V) + Z
D	16	D_Bus

Am29332 Control-Field Structure

REG-A

AW3~	1	Write Enable 3 1 - Disable 0 - Enable
AW2~	1	Write Enable 2 1 - Disable 0 - Enable
AW1~	1	Write Enable 1 1 - Disable 0 - Enable
AW0~	1	Write Enable 0 1 - Disable 0 - Enable
AOE~	1	Output Enabke 1 - Disable 0 - Enable
ASE	1	Address Sel. 1 - Microcode 0 - Macro Inst.
AAR	6	Address 0 to 3F

REG-B

AW3~	1	Write Enable 3 1 - Disable 0 - Enable
AW2~	1	Write Enable 2 1 - Disable 0 - Enable
AW1~	1	Write Enable 1 1 - Disable 0 - Enable
AW0~	1	Write Enable 0 1 - Disable 0 - Enable
AOE~	1	Output Enabke 1 - Disable 0 - Enable
ASE	1	Address Sel. 1 - Microcode 0 - Macro Inst.
AAR	6	Address 0 to 3F

CTL-2

FSL	1	FPU Sta. Le 1 - Transpant 0 - Latch
MRE	1	Macro Reg. En. 1 - Enable 0 - Disable
INT	1	Interrupt 80188 1 - Interrupt 0 - No Interrypt
BS1~	1	Enable MIB79-64 1 - Disable 0 - Enable
BS0~	1	R to T Control 1 - Disable 0 - Enable
BDE~	1	T to R Control 1 - Disable 0 - Enable
PEC~	1	Parity Err. Clr. 1 - No Change 0 - Clear
DW	2	Data Width 0 - (4) 1 - (1) 2 - (2) 3 - (1)
MWE~	1	Write Enable 1 - Read 0 - Write
LEN	3	Clock Length 0 - (3) 1 - (4) 2 - (8) 3 - (7) 4 - (10) 5 - (5) 6 - (9) 7 - (6)

Am29332 Control-Field Structure

ALU

P	6	Position bits (2'S COMPLEMENT) 20 -> 3F -> 0 -> 1F
W	5	Width bits 1F - 0 (31 - 0)
BW	2	Byte Width 0 - (4) 1 - (1) 2 - (2) 3 - (3) P & W Sel: 0 - P (Pins), W (Pins) 1 - P (Pins), W (Reg.) 2 - P (Reg.), W (Pins) 3 - P (Reg.), W (Reg.)
32I	7	ALU Instructions (Am29332)

00 ZERO-EXTA	10 DECR-A	20 DN1-0F-A	30 UP1-0F-A
01 ZERO-EXTB	11 DECR-B	21 DN1-0F-B	31 UP1-0F-B
02 SIGN-EXTA	12 INCR-A	22 DN1-0F-AQ	32 UP1-0F-AQ
03 SIGN-EXTB	13 INCR-B	23 DN1-0F-BQ	33 UP1-0F-BQ
04 PASS-STAT	14 DECR2-A	24 DN1-1F-A	34 UP1-1F-A
05 PASS-Q	15 DECR2-B	25 DN1-1F-B	35 UP1-1F-B
06 LOADQ-A	16 INCR2-A	26 DN1-1F-AQ	36 UP1-1F-AQ
07 LOADQ-B	17 INCR2-B	27 DN1-1F-BQ	37 UP1-1F-BQ
08 NOT-A	18 DECR4-A	28 DN1-LF-A	38 UP1-LF-A
09 NOT-B	19 DECR4-B	29 DN1-LF-B	39 UP1-LF-B
0A NEG-A	1A INCR4-A	2A DN1-LF-AQ	3A UP1-LF-AQ
0B NEG-B	1B INCR4-B	2B DN1-LF-BQ	3B UP1-LF-BQ
0C PRIOR-A	1C LDSTAT-A	2C DN1-AR-A	3C ZERO
0D PRIOR-B	1D LDSTAT-B	2D DN1-AR-B	3D SIGN
0E MERGEA-B	1E -	2E DN1-AR-AQ	3E OR
0F MERGEB-A	1F -	2F DN1-AR-BQ	3F XOR
40 AND	50 SDIVSTEP	60 NB-SN-SHA	70 NOTF-A
41 XNOR	51 SDIVLAST1	61 NB-SN-SHB	71 NOTF-AL-A
42 ADD	52 MPDIVSTEP1	62 NB-0F-SHA	72 PASSF-A
43 ADDC	53 MPSDIVSTEP3	63 NB-0F-SHB	73 PASSF-AL-A
44 SUB	54 UDIVSTEP	64 NBROT-A	74 ORF-A
45 SUBC	55 UDIVLAST	65 NBROT-B	75 ORF-AL-A
46 SUBR	56 MPDIVSTEP2	66 EXTBIT-A	76 XORF-A
47 SUBRC	57 MPUDIVSTP3	67 EXTBIT-B	77 XORF-AL-A
48 SUM-CORR-A	58 REMCORR	68 SETBIT-A	78 ANDF-A
49 SUM-CORR-B	59 QUOCORR	69 SEIBIT-B	79 ANDF-AL-A
4A DIFF-CORR-A	5A SDIVLAST2	6A RSTBIT-A	7A EXTF-A
4B DIFF-CORR-B	5B UMULFIRST	6B RSTBIT-B	7B EXTF-B
4C -	5C UMULSTEP	6C SETBIT-STAT	7C EXTF-AB
4D -	5D UMULLAST	6D RSTBIT-STAT	7D EXTF-BA
4E SDIVFIRST	5E SMULSTEP	6E NOTF-AL-B	7E EXTBIT-STAT
4F UDIVFIRST	5F SMULFIRST	6F PASSF-AL-B	7F PASS-MASK

BOW	1	1 - BORROW MODE 0 - NORMAL MODE
HLD	1	1 - HOLD MODE 0 - NORMAL MODE

Am29332 Control-Field Structure

MULTIPLIER

RND	1	Round Ctl.	1 - Rounding 0 - No Rounding
FA	1	Format Adj.	1 - 64 bits 0 - Left Shift 63 bits
PSL	2	O/P Sel.	0 - Temp. reg. 2 - MS 32 bits 1 - LS 32 bits 3 - Disable
ACC	2	ACC Sel.	0 - Pass 2 - Invalid 1 - Accumulate 3 - Shift & accumulate
XSL	1	X Sel.	1 - XA reg. 0 - XB reg.
TCX	1	data format	1 - Unsigned 0 - 2's complement
FTX	1	Feedthrough	1 - FT mode 0 - Register mode
ENA~	1	XA register	1 - Disable 0 - Enable
ENB~	1	XB register	1 - Disable 0 - Enable
YSL	1	Y Sel.	1 - YA reg. 0 - YB reg.
TCY	1	data format	1 - Unsigned 0 - 2's complement
FTY	1	Feedthrough	1 - FT mode 0 - Register mode
ENA~	1	YA register	1 - Disable 0 - Enable
ENB~	1	YB register	1 - Disable 0 - Enable
TSL	1	Temp. reg.	1 - MS P_Reg. 0 - LS P_Reg.
ENT	1	Temp. Reg.	1 - Disable 0 - Enable
ENP	1	Pro. Reg.	1 - Disable 0 - Enable
ENI	1	Inst. Reg.	1 - Disable 0 - Enable
FTI	1	Feedthrough	1 - FT mode 0 - Register mode
FTP	1	Feedthrough	1 - FT mode 0 - Register mode

FPU

ENR~	1	Enable R_REG.	1 - Disable 0 - Enable
ENS~	1	Enable S_REG.	1 - Disable 0 - Enable
ENF~	1	Enable F_REG.	1 - Disable 0 - Enable
RSL	1	R-MUX Select	1 - Port F 0 - R_Bus
SSL	1	S-MUX Select	1 - Reg. F 0 - Reg. S
25I	3	FPU Instructions (Am29325)	
		0 : F = R + S	4 : F(FP) = R(INTEGER)
		1 : F = R - S	5 : F(INTEGER) = R(FP)
		2 : F = R * S	6 : F(DEC) = R(IEEE)
		3 : F = 2 - S	7 : F(IEEE) = R(DEC)
RND	2	Rounding toward	0 - nearest 2 - Plus Infinity 1 - minus infinity 3 - 0
FT1	1	F-Reg. F.T. Ctl.	1 - Trans. 0 - Reg. Mode
FT0	1	R,S_REG. F.T. CTL.	1 - Trans. 0 - Reg. Mode

-- Systems Issues --

Am29300 Power Supply and PC Board Routing Considerations

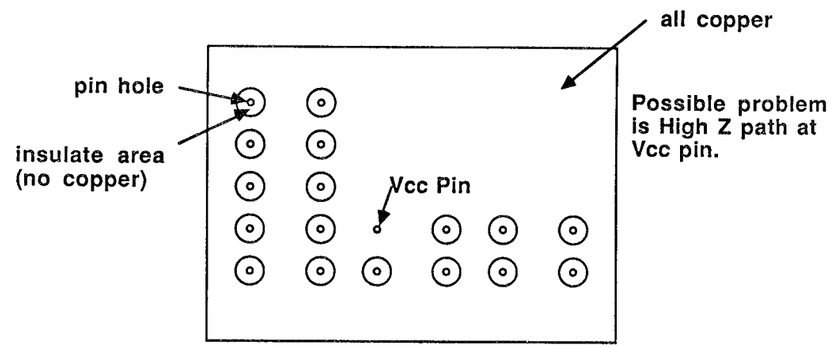
Power Supply Considerations

- Since Am29300 chips are ECL internally with TTL compatible I/O, two power supplies are needed, one for ECL and the other for TTL -- both are +5V, no negative supply is needed. The relationship of noise to these power sources is:

	Noise Generation	Noise Sensitivity
ECL	LOW	HIGH
TTL	VERY HIGH	LOW

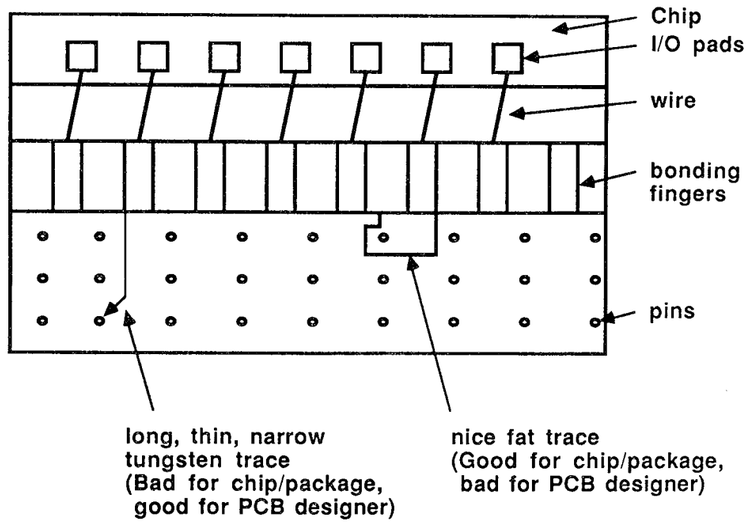
- Large voltage spikes can be created due to as much as 1.5 amp power used per Am29300 chip.

- Consider Vcc pins located on inner ring. What impact is there on the Vcc printed circuit board plane?



Power Supply Considerations

- Now consider Vcc pin on outer ring. OK for PC board designer, but bad for PC package.



- Just how much noise can be generated?
  - Up to one volt over one inch.
  - Cannot be seen with scope (1 nsec wide spikes).
  - Thick copper doesn't help (skin penetration at 16 MHz is only 8 microns).
  - Cannot use power planes to connect common Vcc pins across chip.

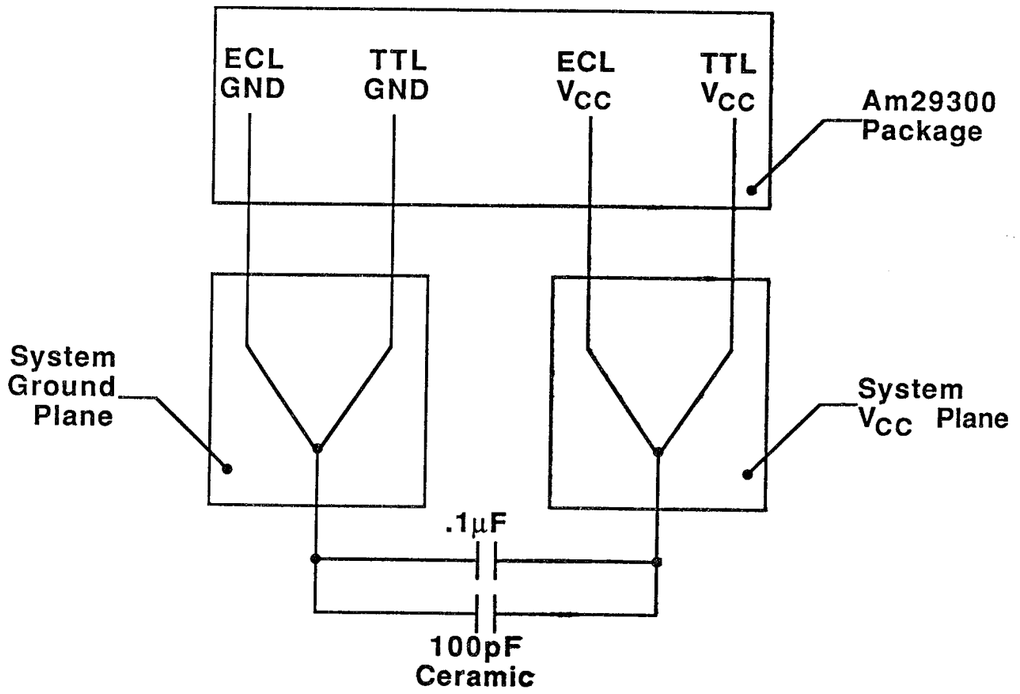
Power Supply Considerations

- A solution
  1. Separate ECL and TTL supplies at package interface.
  2. Connect noise sensitive ECL on inner ring
     - Interconnect on a quiet signal plane
     - Bring trace to outside of outer pins
     - Connect to plane at decouple point
  3. Connect noisy TTL supplies directly to power planes from outer ring

- Am29300 parts generally follow this convention. The exception is the Am29325 Floating Point ALU.

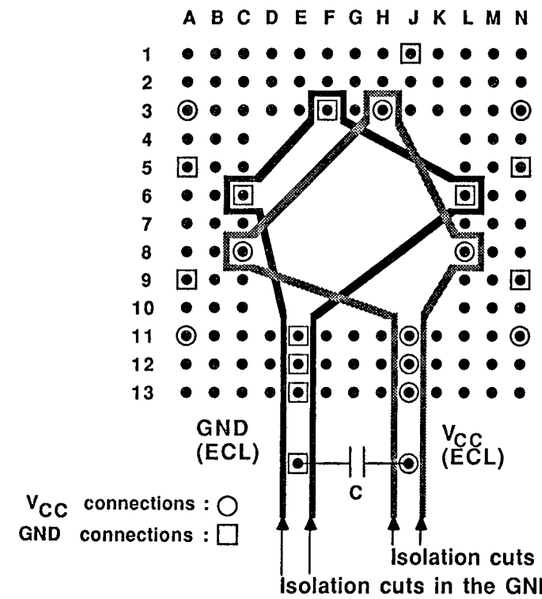
Am29300 Family

Power Supply Wiring Considerations



Am29331 Suggested Printed-Circuit Board Layout

Bottom View

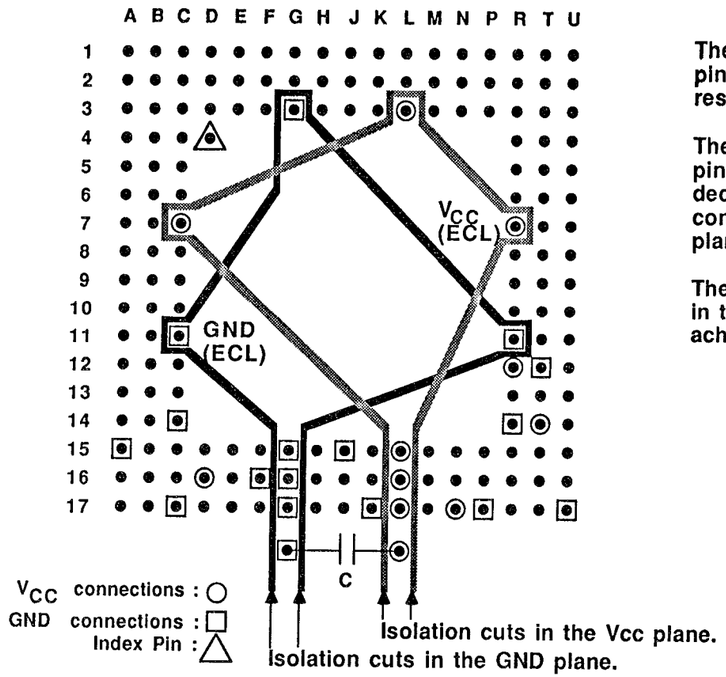


The noise-generating TTL supply pins are connected directly to their respective power plane.

The noise sensitive ECL supply pins are interconnected and decoupled as shown, then each connected to the respective power plane in one point only.

The heavy lines indicate cuts in the two supply planes that achieve this isolation.

Am29332 Suggested Printed Circuit Board Layout



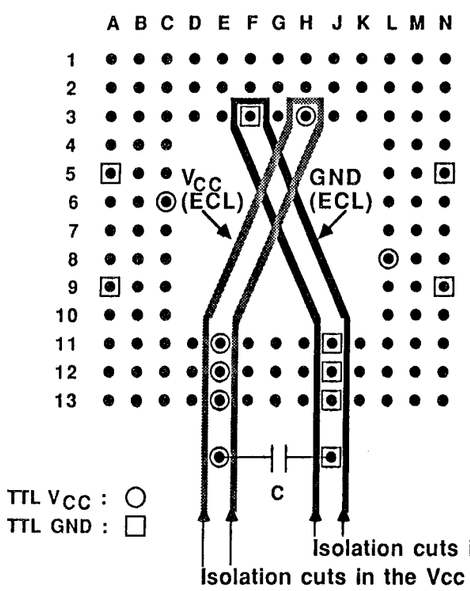
The noise-generating TTL supply pins are connected directly to their respective power plane.

The noise sensitive ECL supply pins are interconnected and decoupled as shown, then each connected to the respective power plane at one point only.

The heavy lines indicate cuts in the two supply planes that achieve this isolation.

Am29334 Suggested Printed Circuit Board Layout

Bottom View



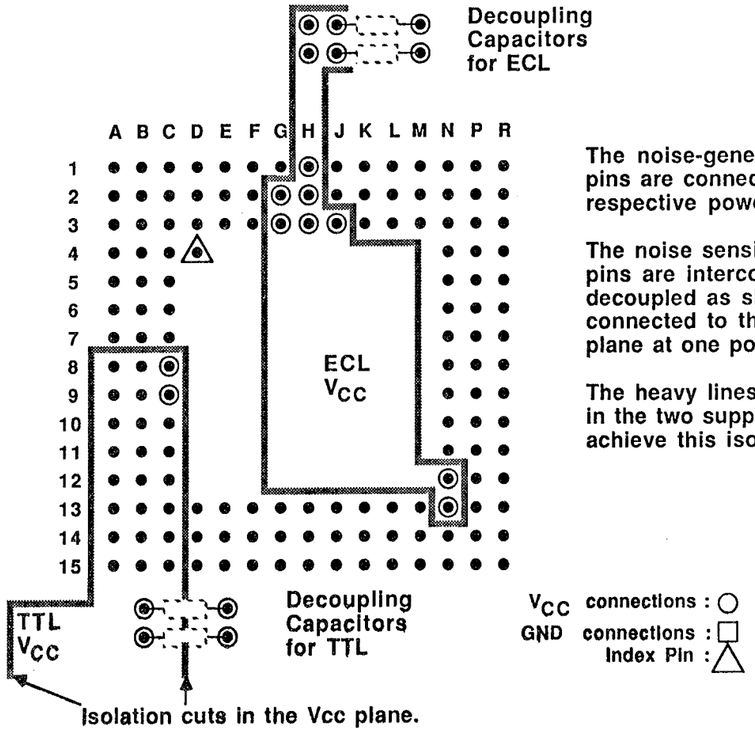
The noise-generating TTL supply pins are connected directly to their respective power plane.

The noise sensitive ECL supply pins are interconnected and decoupled as shown, then each connected to the respective power plane at one point only.

The heavy lines indicate cuts in the two supply planes that achieve this isolation.

Am29325 Suggested Printed Circuit Board Layout

V_{CC} Connections Bottom View



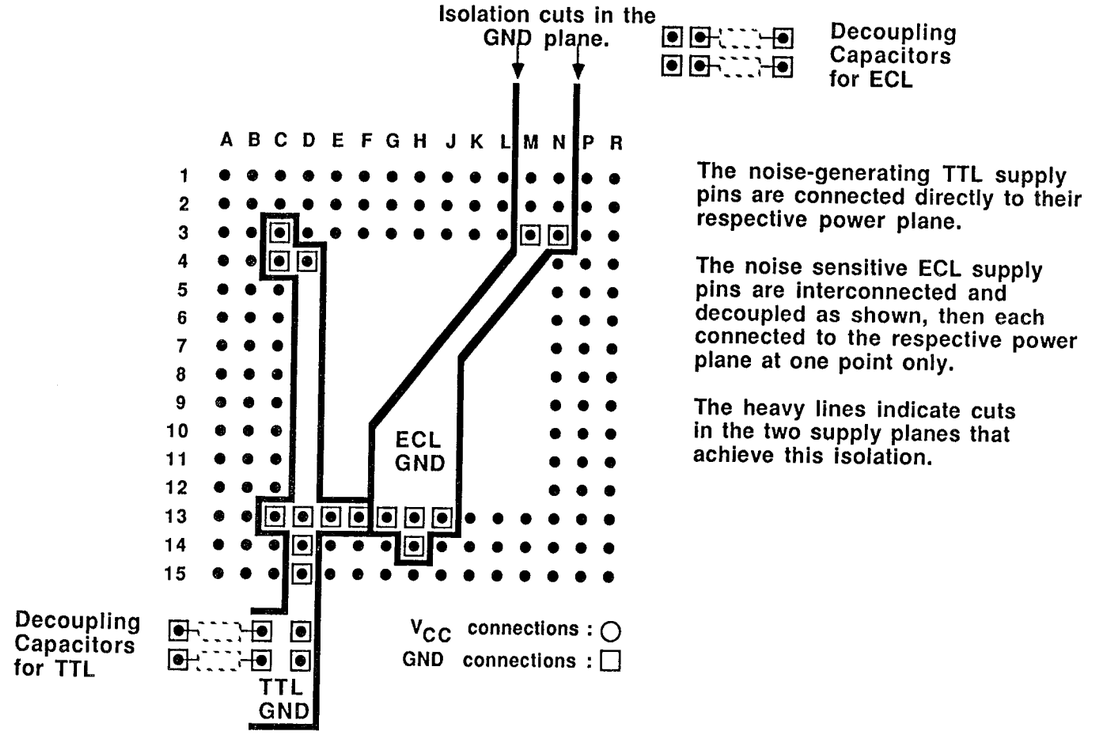
The noise-generating TTL supply pins are connected directly to their respective power plane.

The noise sensitive ECL supply pins are interconnected and decoupled as shown, then each connected to the respective power plane at one point only.

The heavy lines indicate cuts in the two supply planes that achieve this isolation.

Am29325 Suggested Printed Circuit Board Layout

GND Connections Bottom View



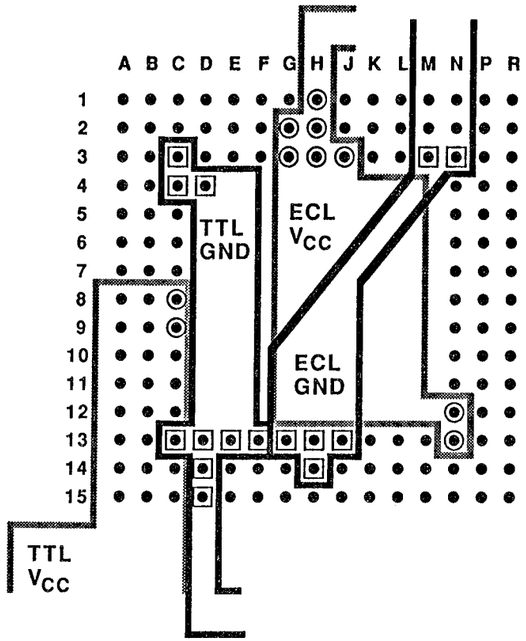
The noise-generating TTL supply pins are connected directly to their respective power plane.

The noise sensitive ECL supply pins are interconnected and decoupled as shown, then each connected to the respective power plane at one point only.

The heavy lines indicate cuts in the two supply planes that achieve this isolation.

Am29325 Suggested Printed Circuit Board Layout

Vcc &GND Connections **Bottom View**



The noise-generating TTL supply pins are connected directly to their respective power plane.

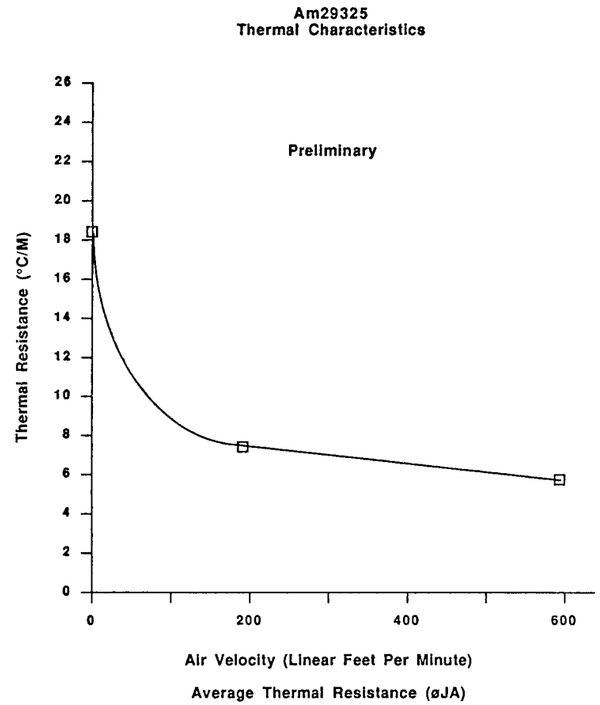
The noise sensitive ECL supply pins are interconnected and decoupled as shown, then each connected to the respective power plane at one point only.

The heavy lines indicate cuts in the two supply planes that achieve this isolation.

V_{CC} connections : ○
GND connections : □

Note: 1. D4 (index pin) is not connected internally – may be wired to TTL ground or left unconnected.

Am29300 Family Cooling Considerations



Recommendations

- 400 feet per minute (fpm) air flow across each Am29300 device
- Vertical, rather than horizontal, board mounting
- Locate hot parts at top of board
- Use baffles (Venturi effect)
- Add or increase effectiveness of heatsinks
- Mount component on cabinet wall

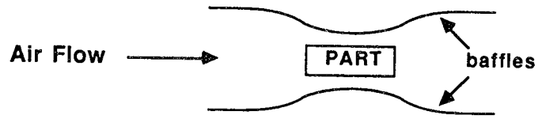


FIG. 1

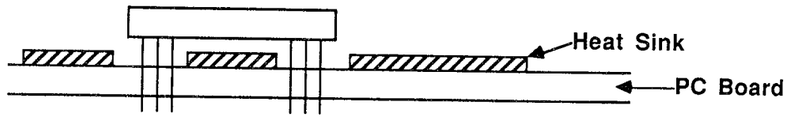


FIG. 2A

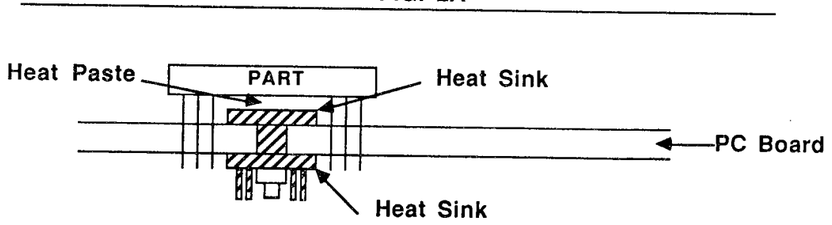


FIG. 2B

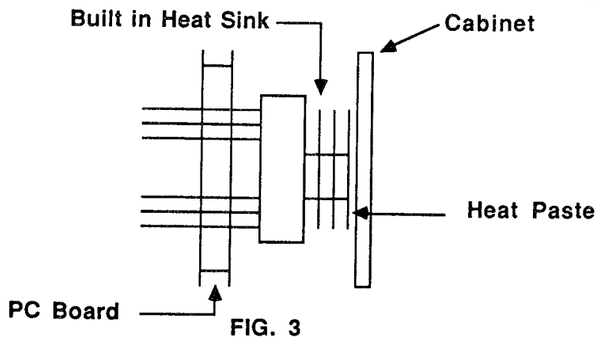


FIG. 3

Sockets and Insertion/Extraction Tools

The following manufacturers should be contacted for information on PGA sockets for the Am29300 family:

Advanced Interconnections Corp.
Warwick, RI (401)855-0485

AMP, Inc.
Harrisburg, PA (717)564-0100

Augat, Inc.
Attleboro, MA (617)222-2202

Azimuth Electronics, Inc.
San Clemente, CA (714)492-6481

Electronic Molding Corp.
Woonsocket, RI (401)769-3800

Methode Electronics, Inc.
Rolling Meadows, IL (312)392-3500

Midland-Ross Corp., Connector Div.
Cambridge, MA (617)491-5400

Mupac Corp.
Brockton, MA (617)588-6110

Precicontact, Inc.
Langhorne, PA (215)588-6110

Robinson-Nugent, Inc.
New Albany, IN (812)945-0211

Thomas & Betts Corp., Ansley Electronic Div.
Raritan, NJ (201)568-3838

3M Co. (Textool), Electronic Products Div.
St. Paul, MN (612)733-1378

Wells Electronics, Inc.
South Bend, IN (219)287-5941

Yamaichi, Mepenthe Distribution Inc.
Palo Alto, CA (415)856-9332

Sockets and Insertion/Extraction Tools

For insertion/extraction tools, contact:

Advanced Interconnections Corp.
Warwick, RI (401)885-0485

McKenzie Engineering
Fremont, CA (415)651-2700

Be sure to check that tools are applicable to parts with heat sinks, if not using CMOS versions.

-- System Issues --

Fault Detection

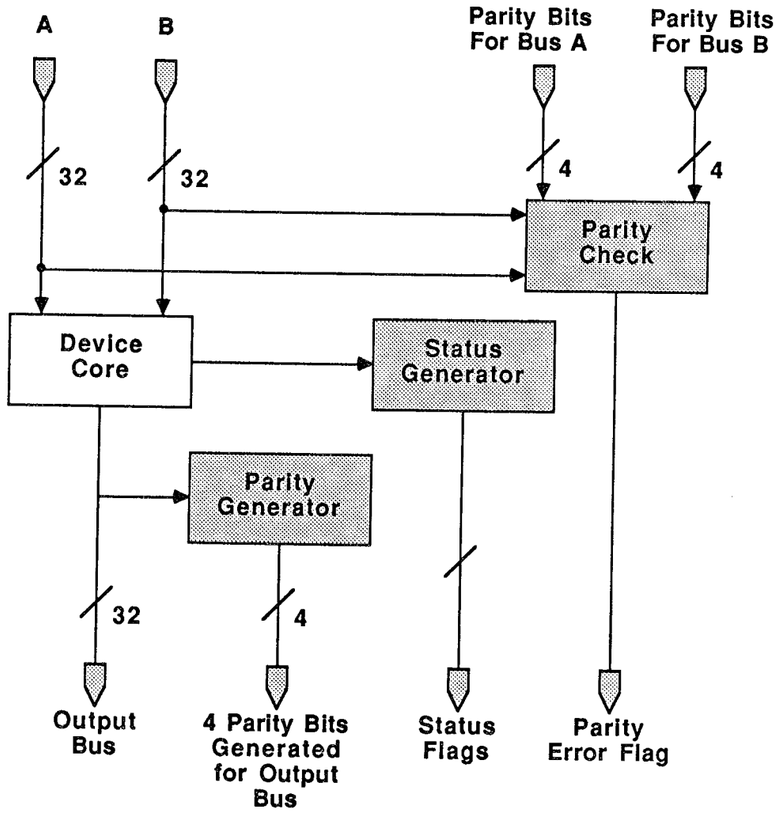
Am29300 Fault Detection

- Two schemes used
  1. Byte parity
  2. Master/slave checking

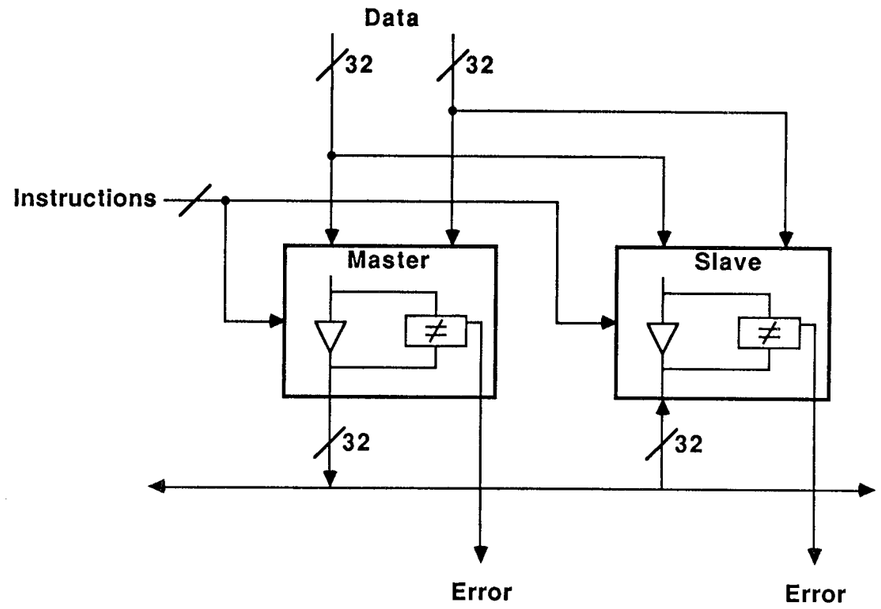
- Byte Parity
  1. Parity checked on each data byte input to chip.
  2. Parity generated on each data byte generated by chip.
  3. Error signal generated by chip if parity error.
  4. Exceptions:

     Am29331 Sequencer and Am29325 Floating Point do no parity checking/ generation.

- Master/Slave Checking
  1. Detects failures at the device level.
  2. Requires redundant device to check master device.
  3. Both incorrect results from master checked by comparing outputs from two identical devices both with identical input.



The three-bus, flow-through architecture affords the greatest access to the device cores. Byte-parity checking detects connection failures between devices.



Master/slave checking provides device failure detection using redundant parallel devices. This happens without incurring the delay of a "voting" scheme often used in high reliability designs.

Answers to Exercises

29300 Example Coding Form

Solutions to Move Instruction Excercises

	Am29331			Am29332				Am29334			Am29332 Y-Out	
	OP	Branch or Counter Value	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$	
1.					2	LOADQ-A			R7			1	*
2.					0	MERGEA-B			R12		R24	0	**
3.					2	MERGEA-B			R17	R1	R1	0	
4.					3	SIGN-EXTA			R3			1	***
					3	SIGN					R4	0	
5.					2	LDSTAT-A			R4			1	
6.					0	ZERO					R0	0	****
					0	LOADQ-A			R0			1	

Note: Assume Y-OUT feeds the B-Input to the Am29334.

* Input could just as well come from the B-IN port.
** Could use a number of other move instructions to accomplish a PASS (e.g., ZERO-EXTA, SIGN-EXTA).
*** SIGN-EXTA used solely to set N flag. Could have used MERGEA-B also.
**** R0 used solely as a temporary register.

29300 CF-1

29300 Example Coding Form

Solutions to Logical Instruction Exercises

	Am29331			Am29332				Am29334			Am29332 Y-Out	
	OP	Branch or Counter Value	Cond Select	Mult Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
1.					3	NOT-A			R14		R14	0
2.					0	XOR			R10	R11	R11	0
3.					0	AND			R21	R30	R25	0
4.					0	MERGEA-B			R9	R16	R0	0
5.					0	AND			R1	R31	R31	0 *

Note: Assume Y-OUT feeds the B-input to the Am29334.

* Assume R1 contains $FF00FFFF_{16}$

29300 CF-2

29300 Example Coding Form

Solutions to Single-bit Shift Instruction Exercises

	Am29331				Am29332				Am29334			Am29332 Y-Out	
	OP	Branch or Counter Value	Cond Select	Mult Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$	
1.					2	DN1-0F-A			R4		R4	0	
2.					0	UP1-0F-A			R9		R9	0	
3.					3	DN1-AF-A			R21		R0	0	
4.					0	DN1-0F-AQ			R2		R2	0	
5.					0	DN1-AF-A			R2		R2	0	*
					0	DN1-LF-A			R3		R3	0	

Note: Assume Y-OUT feeds the B-input to the Am29334.

* Sets link bit with $R2_0$ to fill MSB of R3 on next instructions.

29300 CF-3

29300 Example Coding Form

Solutions to Priority and Arithmetic Instruction Exercises

	Am29331				Am29332				Am29334			Am29332 Y-Out
	OP	Branch or Counter Value	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
1.		Priority is 6										
2.		Priority is 0										
3.					0	NEG-A			R17		R0	0
4.					0	INCR2-A			R2		R2	0
					0	INCR2-A			R2		R2	0
5.					0	ADD			R30	R31	R3	0
6.					0	SUB			R31	R30	R31	0
7.					0	SUB			R10	R8	R12	0
					0	SUBC			R9	R7	R11	0
8.					0	SUB			R31	R30	R31	0
					0	DIFF-CORR-A			R31		R31	0

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF-4

29300 Example Coding Form

Solutions to Shift and Rotate Instruction Exercises

	Am29331				Am29332				Am29334			Am29332 Y-Out	
	OP	Branch or Counter Value	Cond Select	Multi Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$	
1.					3	NB-0F-SHA		17	R22		R22	0	
2.					0	NBROT-A		-14	R1		R1	0	
3.					0	NB-0F-SHA		-17	R22		R22	0	
4.					0	NB-SN-SHA		-17	R22		R22	0	
					0	OR			R1		R22	0	*

Note: Assume Y-OUT feeds the B-input to the Am29334. * Assume R1 contains $FFFFFC00_{16}$

29300 CF-5

29300 Example Coding Form

Solutions to Bit-Field Instruction Exercises

	Am29331			Am29332				Am29334			Am29332 Y-Out
	OP	Branch or Counter Value	Cond Select	Mul-tl Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
1.					0	SETBIT		30	R29		R29	0
2.					0	PASS-Q					R0	0
					0	RSTBIT-A		21	R0		R0	0
					0	LOADQ-A			R0			1
3.					0	SETBIT-STAT		19				1
4.					0	EXTBIT-STAT		-30			R30	0

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF-6

29300 Example Coding Form

Solutions to Field Logic Instruction Exercises

	Am29331				Am29332				Am29334			Am29332 Y-Out
	OP	Branch or Counter Value	Cond Select	Mul-tl Sel	B/W	OP	Width	Position	A-IN	B-IN	Y-OUT	$\overline{OE}$
1.					0	NOTF-AL-A	8	8	R27	R27	R27	0
2.					0	NBROT-A		15	R15		R15	0
					0	ANDF-A	4	-27	R14	R15	R16	0
					0	NOTF-AL-A	4	0	R16		R16	0
					0	NBROTA		7	R16		R16	0
3.					0	PASSF-A	8	-16	R19	R20	R21	0
4.					0	ZERO					R2	0
					0	PASSF-A	20	-20	R1	R2	R10	0
5.					0	EXTF	20	20	R1	R2	R10	0

Note: Assume Y-OUT feeds the B-input to the Am29334.

29300 CF-7