



AMD CUSTOMER EDUCATION

ED2900A

**INTRODUCTION TO
DESIGNING WITH THE
Am2900 FAMILY OF
MICROPROGRAMMABLE
BIPOLAR DEVICES**

**LECTURE
VOLUME II**

ED2900A

**INTRODUCTION TO DESIGNING WITH THE Am2900 FAMILY
OF MICROPROGRAMMABLE BIPOLAR DEVICES**

VOLUME II

3rd Edition

**January 1985
Advanced Micro Devices, Inc.
Customer Education Center**

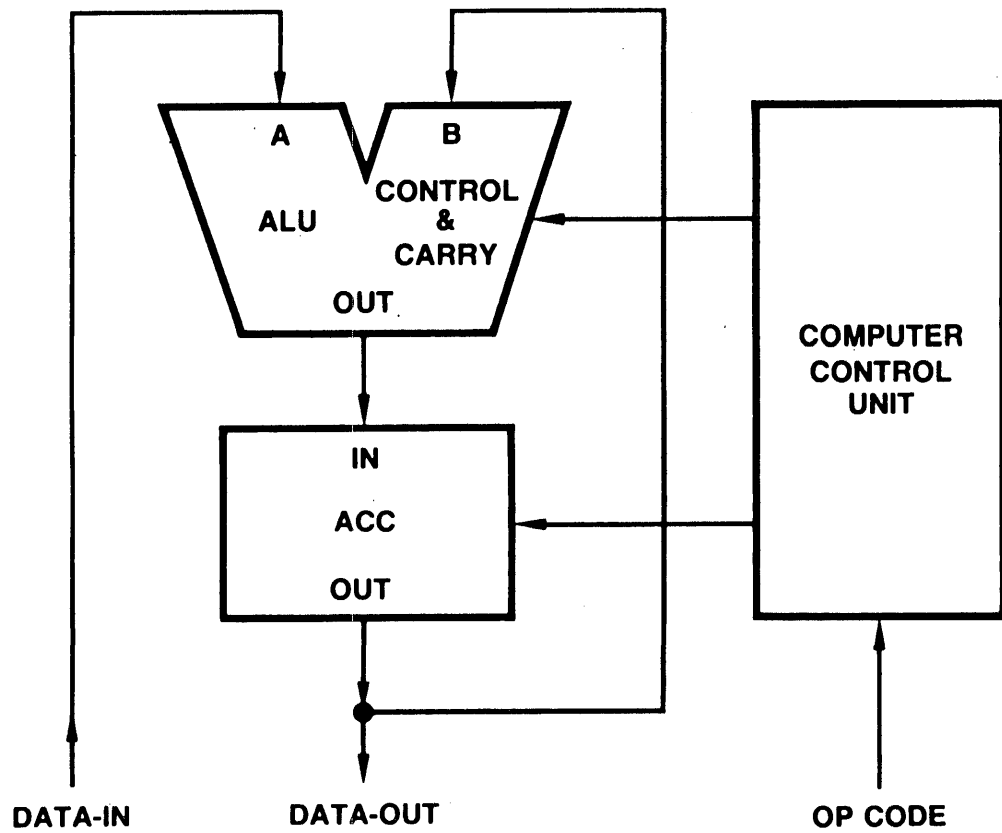
Volume II
Table of Contents

- Improving the ALU
 - Improving ALU Performance
 - Additional ALU Improvements
- Bit-slice ALU, Am2901
 - Interconnection of Slices, Am2901
 - Carry-Lookahead, Am2902
 - Sample Microcode, Am2901
- Introducing the Super Slice, Am2903/Am29203
 - Interconnecting Slices, Am2903/Am29203
 - Example Microcode, Am2903/Am29203
 - Special Functions of Am2903/Am29203
 - Multiplication Process
 - Single-Length Normalize Process
 - Am29203 Additional Special Functions
- Expanded Memory for ALUs
- Introduction to Interrupts
 - Implementation of Interrupt Control, the Am2913
 - A Complete Interrupt Controller, Am2914
- Am2900 Family Support Devices
- 16-bit ALU Controller, Am29116
- AMD Families
- Future Microprogrammable AMD Devices
- AMD Support Tools for Microprogram Development

IMPROVING THE ALU

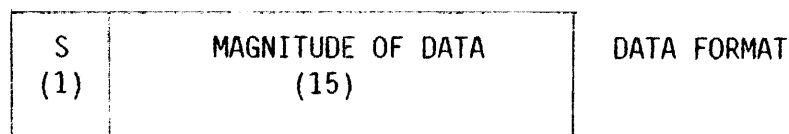
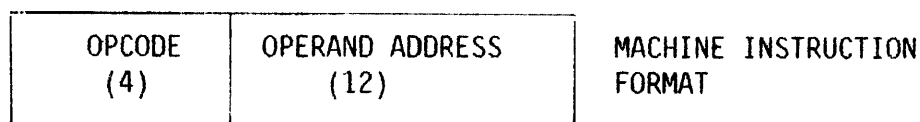
THE BASIC STRUCTURE

- The following page repeats the basic structure which is the initial configuration for a simple computer system.
- A highly capable computer control unit has evolved using the Am2910 as an example microinstruction sequencer.
- An arithmetic/logic unit (ALU) with more operational and storage capabilities will now be developed leading to a variety of commercial devices that can be selected for implementation.



ALU Development

- The system thus far can support basic machine instructions: add, subtract, OR, AND, exclusive OR, load accumulator, and store.
- The particular way in which the A and B ports of the ALU are connected to the outside world dictate that a single-address machine format be used for this architecture.
- Since only one address is supplied, the second operand address (for two operand functions) is assumed to be the accumulator.
- This addressing technique is also known as implied addressing and is often used to save program space (instructions) at the expense of instruction generality.
- **Single Address Format**



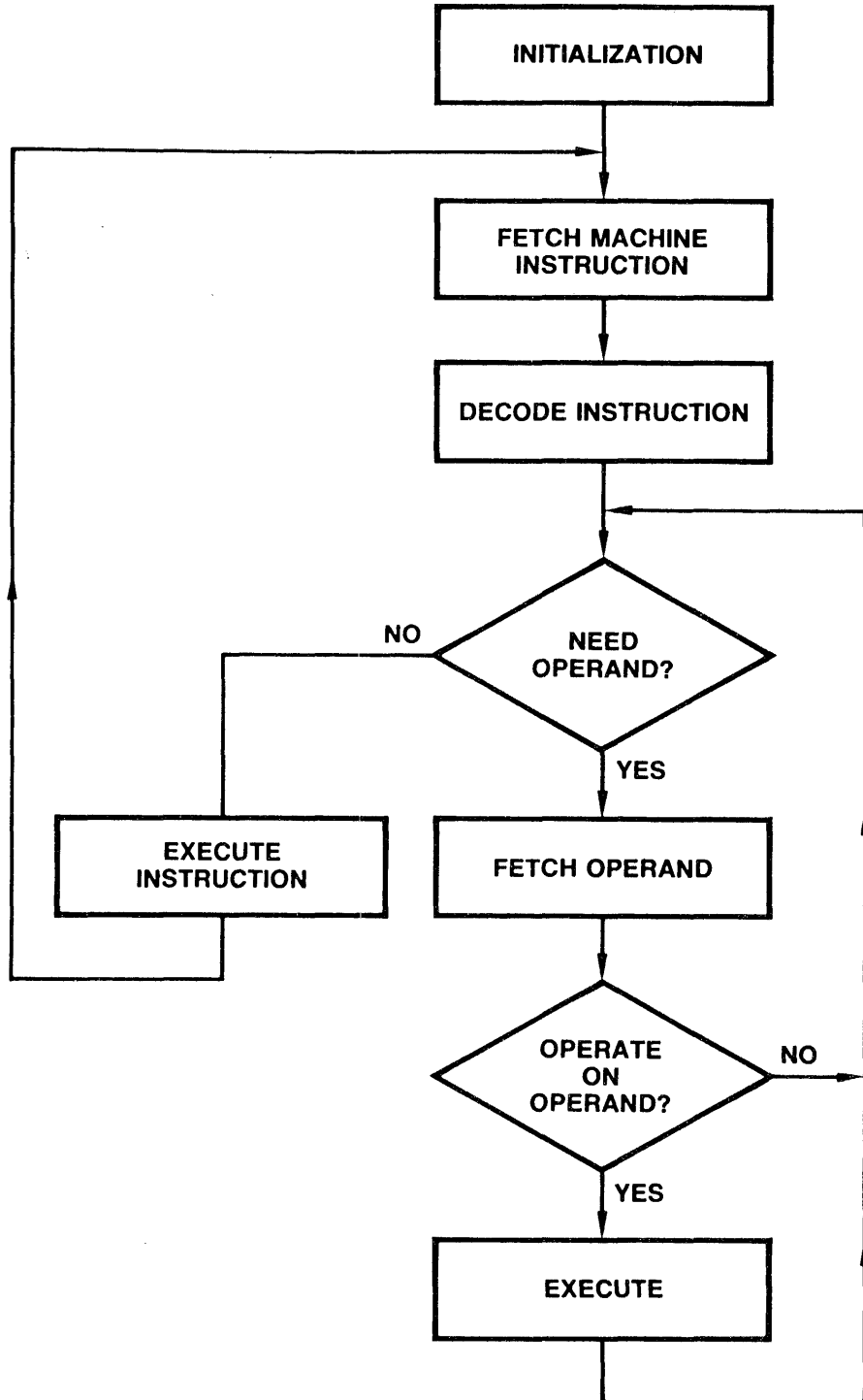
DEFINITIONS

- PC** Program counter (register); maintains the memory address of the next machine instruction to be fetched.
- MAR** Memory address register; contains the address of the item (instruction or operand) which is to be fetched from main memory.
- MAIN MEMORY** Read/write storage (CORE; RAM); contains the program under execution and the associated data; or contains part of the program and part of the data (that which is actively in use).
- ACC** Accumulator register (accumulates ALU results).
- ALU** Arithmetic/logic unit; operates on data according to the instruction operation code.

COMPUTER CONTROL UNIT OPERATION

Once the system is initialized and the uPC (microprogram counter) has been given an initial or first microprogram statement address the general cycles (phases) of "fetch instruction", "fetch operand" and "execute instruction" occur. The register transfer operations in these cycles are further defined below.

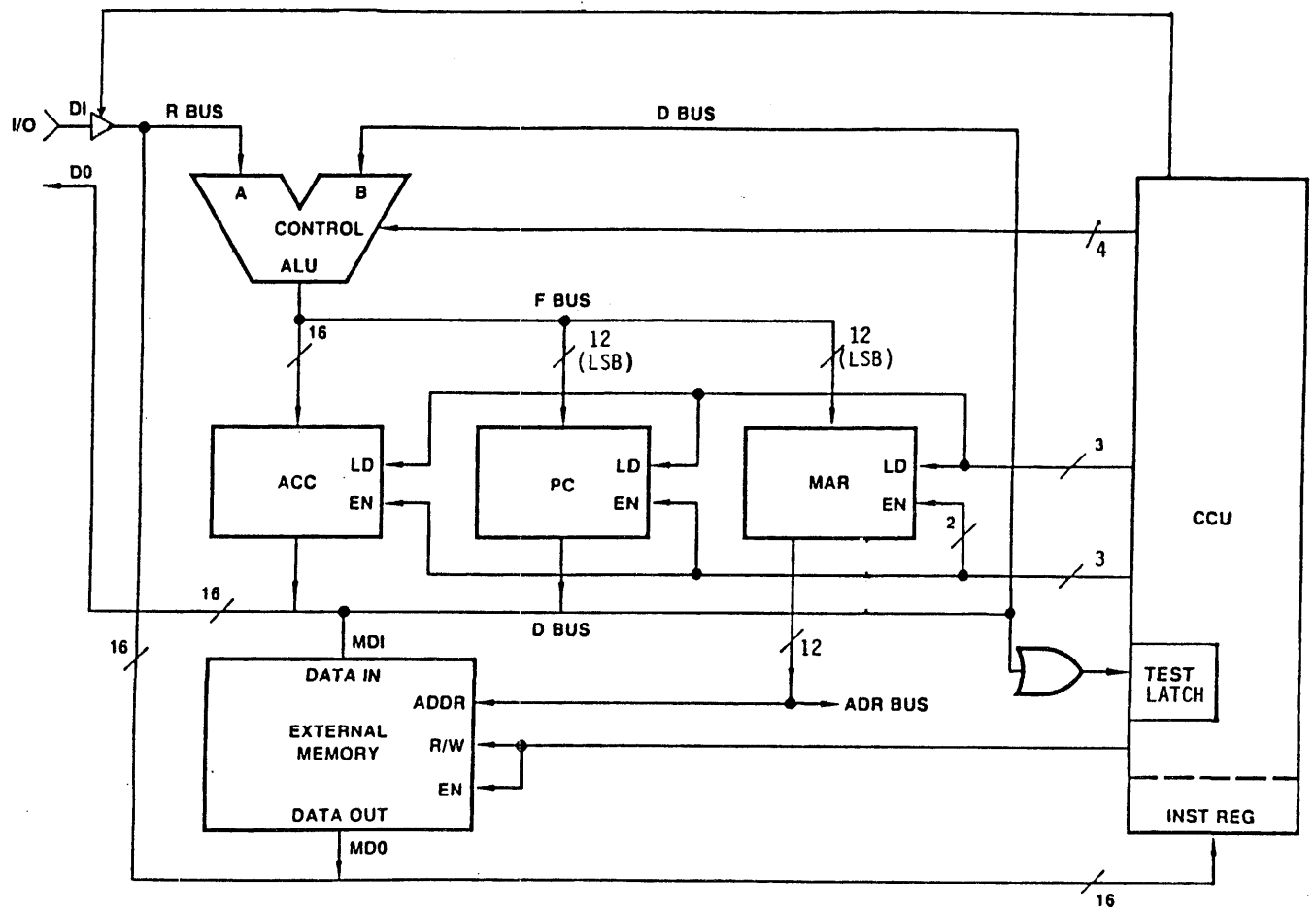
- Fetch the machine instruction (macro level) at the memory address defined by the PC register.
- Decode the opcode portion of the macro instruction (assuming the simplistic machine instruction format shown previously). The instruction decode determines if a data operand is needed.
- During the microinstruction decode (JMAP) step, increment the PC register. Note that the amount of the increment (1, 2, ...) depends on how many memory locations are taken up by the completed instruction (1 here). (This may be done later.)
- Fetch the operand if required. Determine if any operations are required before the operand is ready for use, such as complementing (none in the current architecture).
- If none - determine if any other operand is needed (none required in current architecture).
- If all operands are present, execute the instruction.
- When the execution steps are completed (and the result left in the accumulator), then increment PC, if not already done, and fetch the next instruction.



SIMPLE COMPUTER

The following page diagrams a simple general purpose computer capable of executing the basic machine instructions previously defined.

- The ALU output is connected to an F BUS, which in turn may be connected to provide input to the ACC, the PC or the MAR.
- The MAR is the only register which may address the main memory.
- The PC register is incremented using the ALU.
- When the MAR supplies an address to the external main memory, the value in the addressed location could be an instruction, data, or the location into which data is to be written. The associated register transfer activity is controlled by the CCU.
- The TEST input to CCU comes from the D BUS (the ACC). The test is for \emptyset . If any bit = 1, TEST = 1 (OR operation). Use for conditional jump.
e.g. Jump (CJP) if ACC = \emptyset



MORE DETAILED FETCH CYCLE

(Defined in pseudo register transfer language)

FIRST:

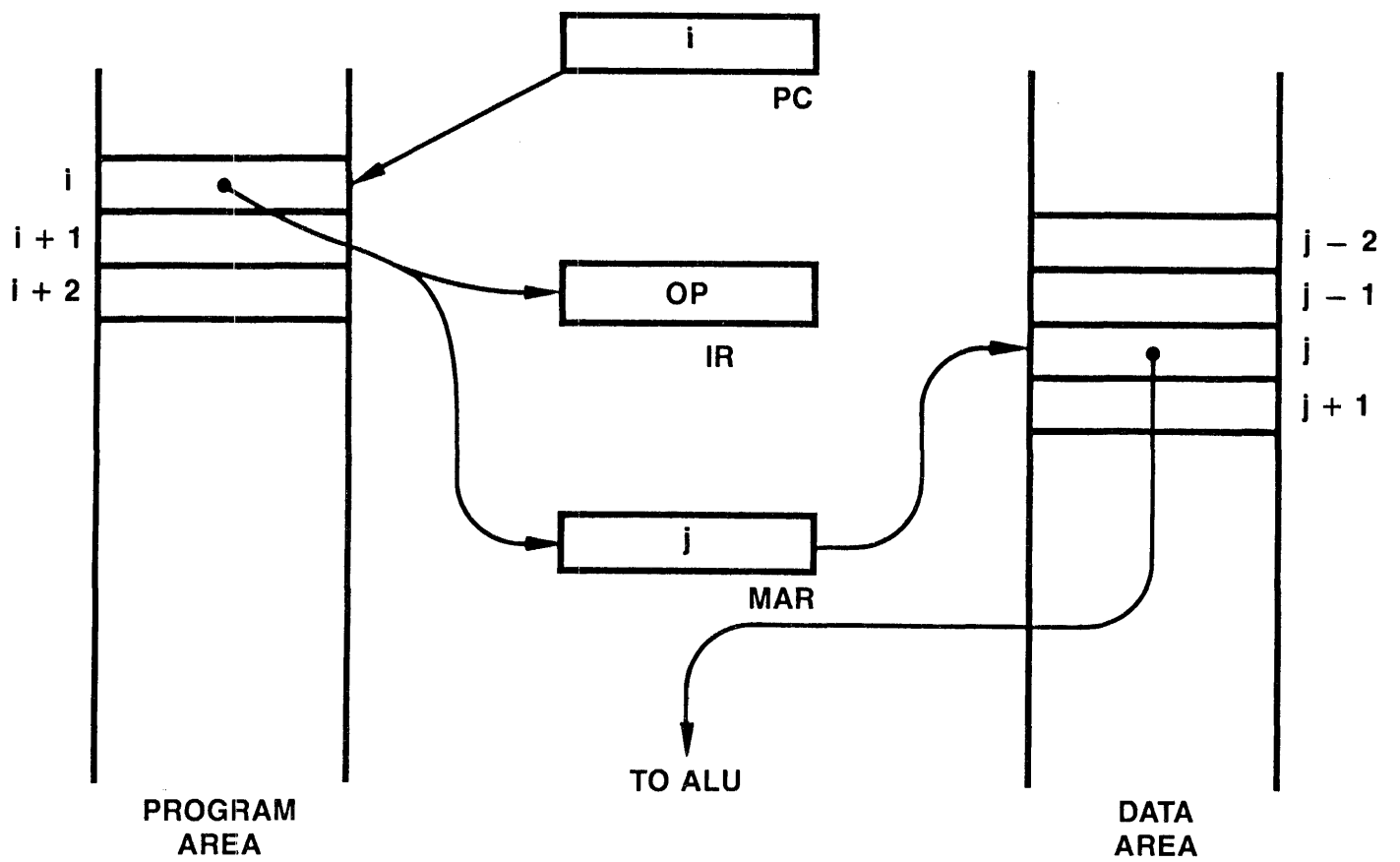
- The CCU controls the transfer of the <PC> (contents of the PC) to the B port of the ALU.
- The CCU causes the ALU to pass <PC> (no operation).
- The ALU output is written into the MAR register.

THEN:

- The MAR contents are enabled on the Address Bus.
- The CCU instructs the memory to perform a READ.
- The CCU instructs the ALU to pass the macroinstruction.
- The CCU instructs the instruction register to latch the upper 4 bits (opcode), with the lower 12 bits latched by the MAR (operand address) in anticipation of fetching an operand.

FINALLY:

- The CCU transfers <PC> through the ALU and increments it by 1 before storing the incremented value.



WHEN DATA IS TO BE FETCHED FROM MEMORY (Fetch Operand Phase):

- The CCU causes the MAR to output to the Address Bus.
- The CCU causes the memory to perform a READ.
- The DATA READ is gated to the A port of the ALU.

EXECUTE INSTRUCTION PHASE

- The CCU causes the data and, if appropriate, ACC data to be manipulated according to the macro-level instruction opcode.

WHEN DATA IS TO BE WRITTEN TO MEMORY

- The CCU causes the MAR to output to the Address Bus.
- The CCU causes the ACC to output to the MDI port of the memory.
- The CCU causes the memory to perform a write.

Note:

The PC contents must be transferred to the MAR before the instruction can be fetched. The MAR is used to address the memory for instruction fetches as well as operand fetches.

THE NEW BASIC COMPUTER INSTRUCTION SET

LDA,ADDR	Load accumulator with contents of address
ADD,ADDR	Add accumulator and contents of address
SUB,ADDR	Subtract accumulator from contents of address
OR,ADDR	OR accumulator with contents of address
AND,ADDR	AND accumulator with contents of address
XOR,ADDR	Exclusive-OR accumulator with contents of address
INA	Input to accumulator
OUT	Output from accumulator
JMP,ADDR	Jump to <Address>
JMZ,ADDR	Jump to <Address> if accumulator is 0
STO,ADDR	Store contents of accumulator at address

DESIGN PROBLEM: A VERY SIMPLE COMPUTER**HOMEWORK - CPU MICROPROGRAM**

- Turn to your ED2900A Exercise and Laboratory Manual. The basic hardware and macroinstruction set for the simple computer design problems are presented.

- Your assignment:

Write the microprogram to support the entire macroinstruction set (implement direct jump first, indirect jump if you have time).

- Limit: 16 microinstructions

- Do not look at the solution until you have tried the problem:

Learning by doing!

IMPROVING ALU PERFORMANCE

ALU SPEED OF EXECUTION

- Consider an ADD (assume $\langle \text{ACC} \rangle + \langle \text{MEM} \rangle \rightarrow \langle \text{ACC} \rangle$):

```

ADD, MEMADDR2    PC -> MAR
                  FETCH INSTR
                  DECODE, INCR PC
                  FETCH DATA, ADD TO  $\langle \text{ACC} \rangle$ 
                  -----
                  4 MICROCYCLES

```

- This is only valid for sequential operations on the accumulator. If the accumulator is needed to store other data, then the intermediate results must be stored in memory and refetched before each operation. In that case, the time needed becomes:

```

LDA, MEMADDR1    4 MICROCYCLES

ADD, MEMADDR2    4 MICROCYCLES

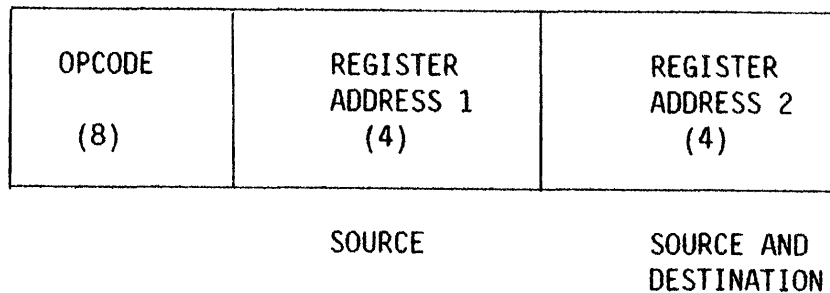
STO, MEMADDR2    4 MICROCYCLES
                  -----
TOTAL            12 MICROCYCLES

```

- This is more realistic, since the ALU accumulator is often needed for current storage and cannot be considered generally available for storage of intermediate results. Additional accumulators or general purpose ALU registers could thus reduce the number of microcycles required per macro level instruction execution.

GENERAL REGISTER ARCHITECTURE

To improve speed and flexibility, consider a different machine instruction format:



Now an ADD could consist of these steps:

ADD R1, R2	PC → MAR. FETCH INSTR DECODE, INCR PC ADD R1 + R2 → R2
------------	---

TOTAL MICROCYCLES	4
-------------------	---

This speed improvement for an "ADD" is valid ONLY

- If the data for the instruction is already in the register.
- If the result is to be used in a following instruction such that it remains in the registers.

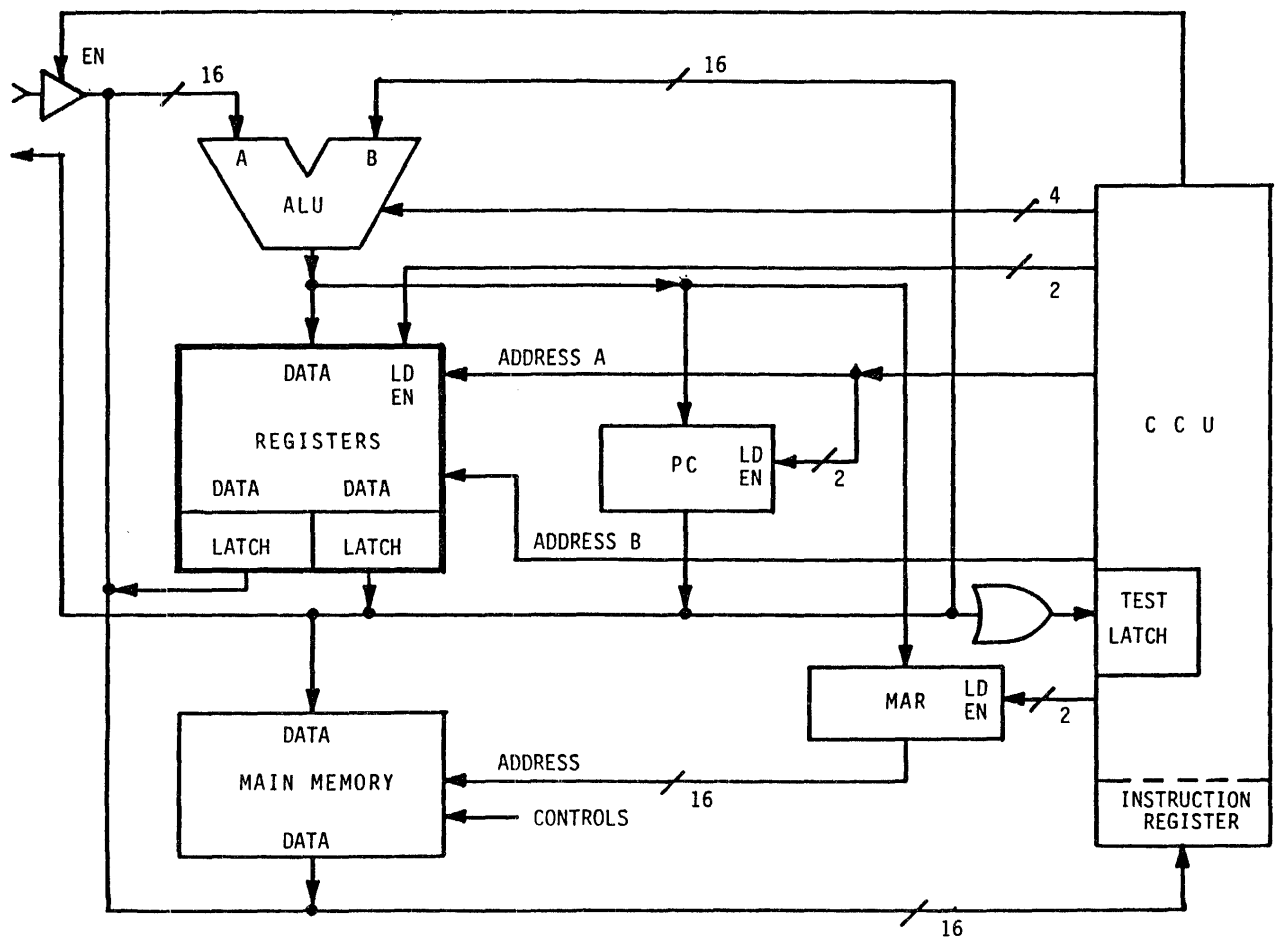
If "enough" ALU registers exist, these assumptions are valid because of the tendency of data to cluster (locality of data, locality of reference) within most computer programs. The number of required ALU registers depends upon the specific application. Bit-slice architecture using the Am2900 family permits devices with a choice.

FURTHER IMPROVEMENTS

- How many registers? Let's begin with 16 general purpose "scratchpad" registers.
 - These registers are multiport registers. Two may be accessed at a time in order to perform:

$R_A + R_B \rightarrow R_B$ in one microcycle

- The accumulator register is now any register which provides a more general system architecture.
- However, one must always specify two operand addresses per machine instruction -- the advantage of implied addressing no longer exists.
- Note that the one word register address machine instruction format is as compact as the one word single address instruction. However, the address space directly addressable is lower; 2^{12} versus 2^4 which is the current configuration.



A, B ALU ADDRESSES

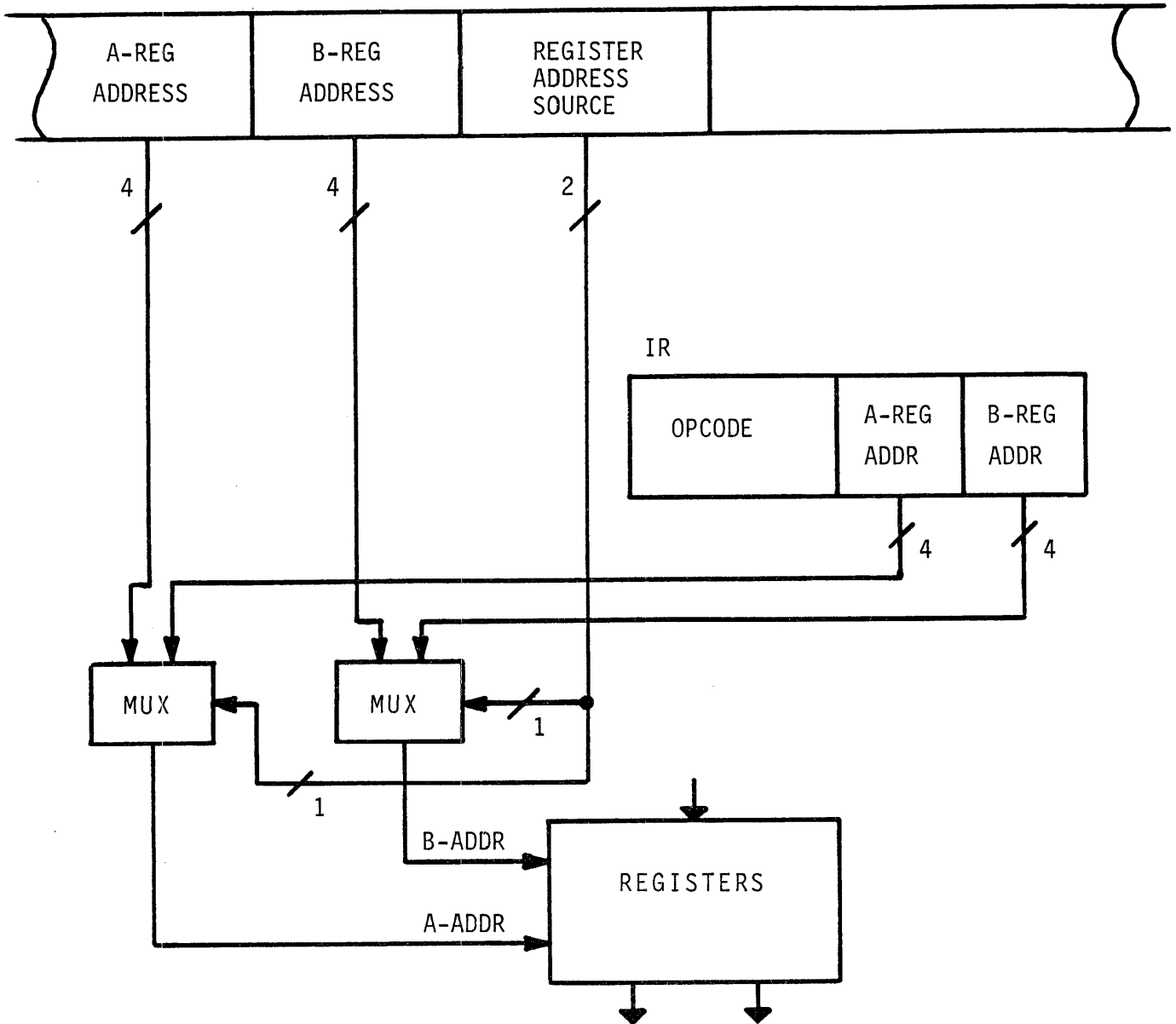
- The register addresses can be supplied from two sources:
 - from the instruction register (macroprogramming)
 - from the microword (microprogramming)

- This implies a multiplexer for selection of A, B addresses and the microword field to control the selection.

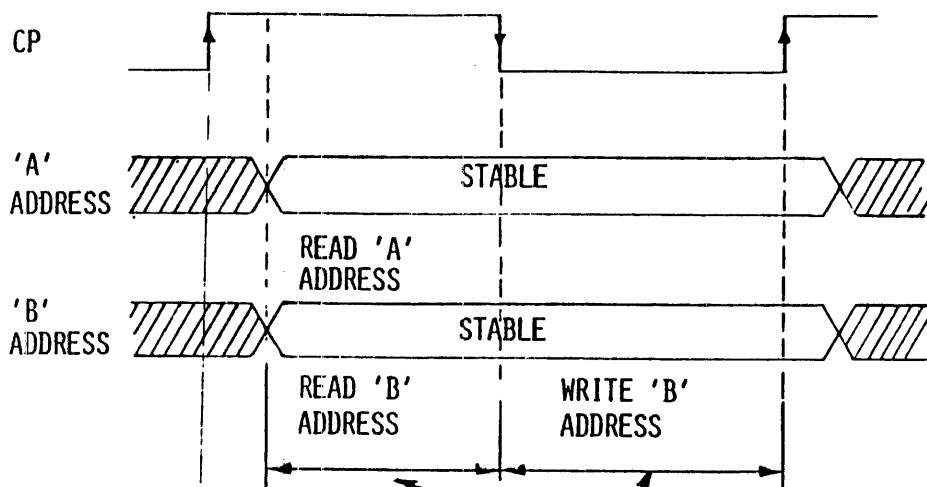
- A and B addresses would be taken from the macroinstruction register for most instructions.

- A and B addresses could also be supplied by the microword in special cases (long word arithmetic; floating point operations; etc.).

PIPELINE



MULTI-PORT MEMORY TIMING



FOR:
 $REG_B \leftarrow REG_A + REG_B$

CYCLE LONG ENOUGH FOR READ AND WRITE

ADDRESSES MUST STAY STABLE

MULTIPOINT READ/WRITE TIMING

- Clock rising edge
 - Load pipeline register
 - Load instruction register (if Fetch done previously)
 - Load RAM register (if Write done previously)

- Time delay into clock high
 - A, B addresses stable

- Clock high
 - READ <A>,

- Clock falling edge
 - Latch the RAM outputs

- Clock low
 - Perform ALU function
 - Set-up time for RAM

MACHINE INSTRUCTION SET**ONE WORD FORMAT**

ADD R1, R2	Addition
SUB R1, R2	Subtraction
OR R1, R2	Boolean OR
AND R1, R2	Boolean AND
XOR R1, R2	Boolean Exclusive OR
MOV R1, R2	MOVE <R1> TO <R2>
IN R2	Input <R2>
OUT R2	Output <R2>
JMP R2	JUMP TO <R2>
JMZ R1, R2	JUMP TO <R2> IF <R1> = 0

TWO WORD FORMAT

LDR R1, MEMADDR
STO R1, MEMADDR

THE MACRO PROGRAM COUNTER

- There is no need nor real advantage to maintaining a separate PC register.
- Use one of the scratchpad registers (general purpose) as the PC for increased flexibility (addressing).
- Any one of the registers could be used (R15 is usually selected).
- Sophisticated addressing schemes are possible:

ADDRESS = <PC>

ADDRESS = <PC> + <BASE REGISTER>

ADDRESS = <PC> + <OFFSET REGISTER>

ADDRESS = <PC> + <BASE> + <OFFSET>

ADDRESS = <R1>

THE "NEW" ARCHITECTURE

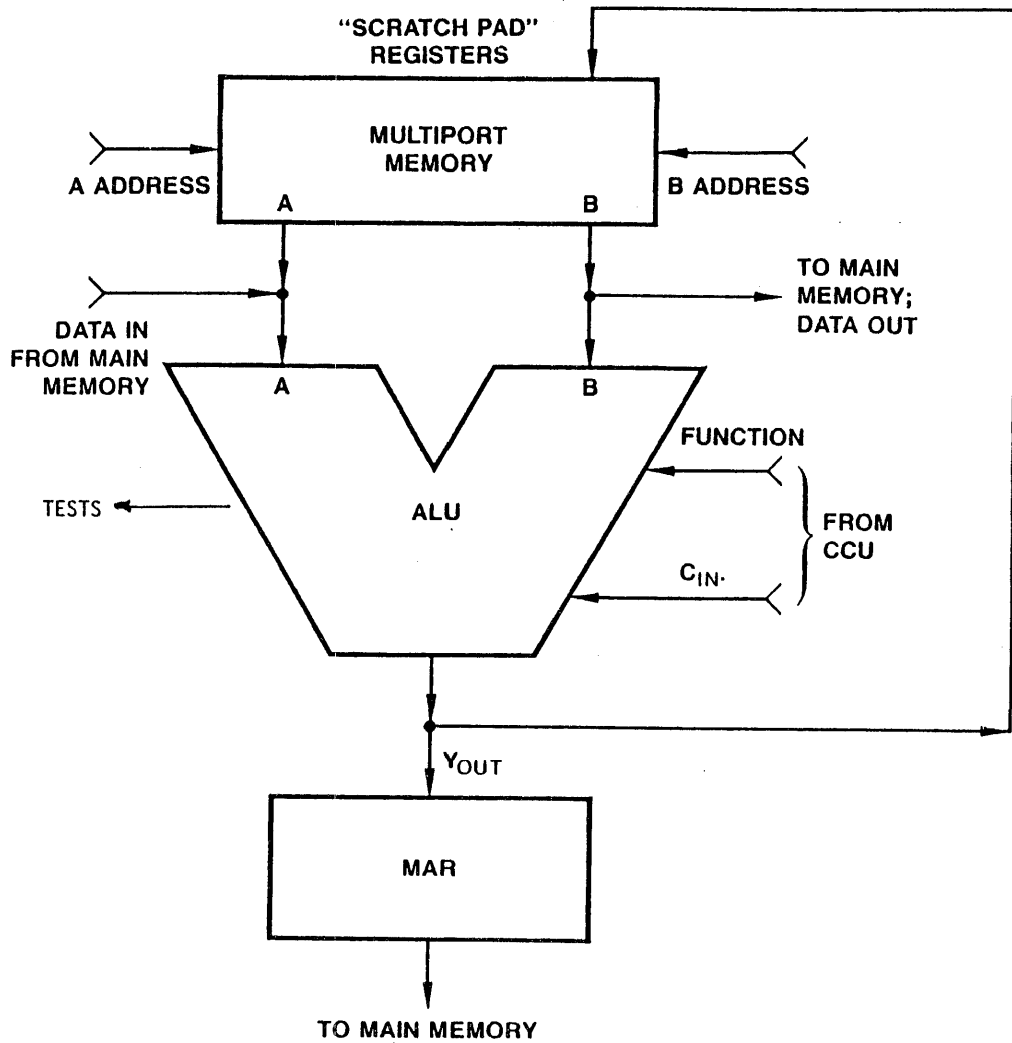
- A separate PC register is no longer included.

- The scratchpad registers are shown in a new position for consistency with typical AMD data sheets.

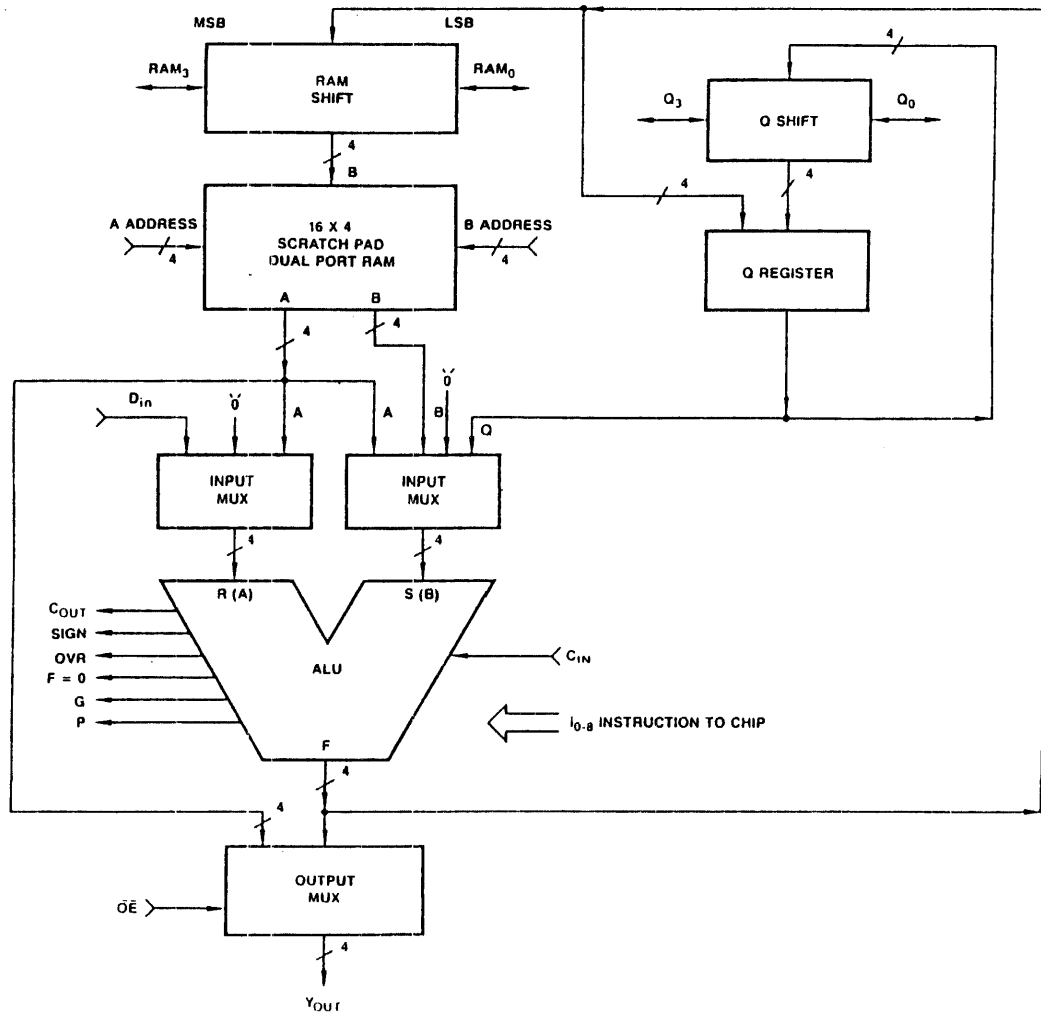
- Carry-in (C_{IN}) is available for arithmetic operations.

- TESTS (zero, overflow, negative, carry-out) are now made by the ALU.

- The MAR remains as a separate register. It will not be considered as part of the "ALU architecture", and will not be shown in subsequent drawings since it is an address buffer for main memory.



ADDITIONAL IMPROVEMENTS



ADDITIONAL ALU ARCHITECTURAL FEATURES

- To allow for more flexibility in selecting ALU sources, multiplexers are added to both ALU inputs.
- Three sources are available to the A-ALU Input (R-Port):
 - data in (D_{IN})
 - the RAM (register) A-Port Output
 - the value ' \emptyset '
- Four sources are available to the B-ALU Input (S-Port):
 - the RAM (Register) A-Port Output
 - the RAM (Register) B-Port Output
 - The value ' \emptyset '
 - The Q-register (to be defined)
- In order to allow register values to be output quickly without passing through the ALU, an output MUX is added to allow selection of the RAM (Register) A-Port output or the ALU output. An output enable control on the output MUX allows these outputs to be connected to a tri-state bus.
- As indicated earlier, the MAR is no longer shown as part of this architecture, although one would usually be connected to the tri-state output Y_{OUT} .
- Control signals for the multiplexers are included with the ALU function control as a set of instruction lines.

ADDITIONAL EXPANSION:

- Add a shifter at the ALU output - RAM input
 - Allow up/down 1-bit shift
 - With external support can perform 1-bit up/down rotate
 - Can choose not to shift/rotate

- A separate shifter allows a shift and an arithmetic operation (OP) to be performed in one microcycle, i.e.
 - $R_i = 2*(R_i \text{ .OP. } R_j)$
 - $R_i = 4*R_i = 2*(R_i + R_i)$
 - etc.

- This capability also allows less complex firmware routines for multiply, divide, and other functions.

ALU COMPLETION

- Multiplication of $N \times N$ bit numbers produces a $2N$ bit result. Thus, add an extension register - **the Q register** - to store the least significant part of the product. Note that the Q register output is connected to the S port selection MUX.

- Add a shifter for the Q register. With external connections this allows double precision up/down shift/rotate as well as supporting the multiply process.

- Expand ALU status lines. Add carry propagate and generate for high speed addition (e.g. carry-look-ahead operation which requires additional external logic discussed in detail later).

"A BIT-SLICE ALU HAS BEEN DEVELOPED"

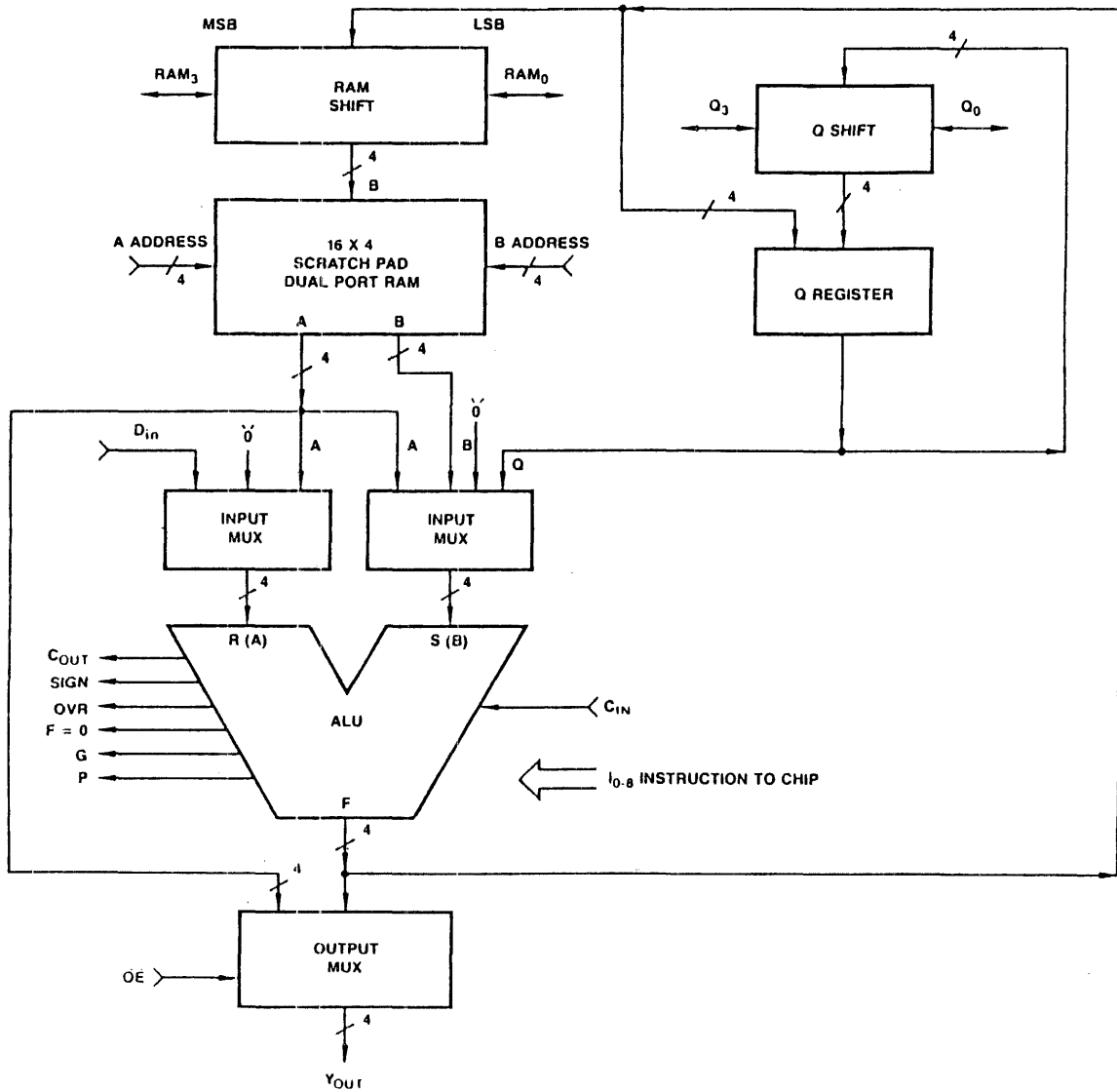
2-340

ED2900A

2-340

BIT-SLICE ALU

The Am2901 Sequencer



Mnemonic	MICRO CODE				ALU SOURCE OPERANDS	
	I ₂	I ₁	I ₀	Octal Code	R	S
AQ	L	L	L	0	A	Q
AB	L	L	H	1	A	B
ZQ	L	H	L	2	O	Q
ZB	L	H	H	3	O	B
ZA	H	L	L	4	O	A
DA	H	L	H	5	D	A
DQ	H	H	L	6	D	Q
DZ	H	H	H	7	D	O

SOURCE CONTROL

Mnemonic	MICRO CODE				ALU Function	SYMBOL
	I ₅	I ₄	I ₃	Octal Code		
ADD	L	L	L	0	R Plus S	R + S
SUBR	L	L	H	1	S Minus R	S - R
SUBS	L	H	L	2	R Minus S	R - S
OR	L	H	H	3	R OR S	R ∨ S
AND	H	L	L	4	R AND S	R ∧ S
NOTRS	H	L	H	5	\bar{R} AND S	$\bar{R} \wedge S$
EXOR	H	H	L	6	R EX-OR S	R ⊕ S
EXNOR	H	H	H	7	R EX-NOR S	$\overline{R \oplus S}$

FUNCTION CONTROL

Consult the AMD Data Book for discussion of tables. Note the effect of C_{IN} in Am2901 function control.

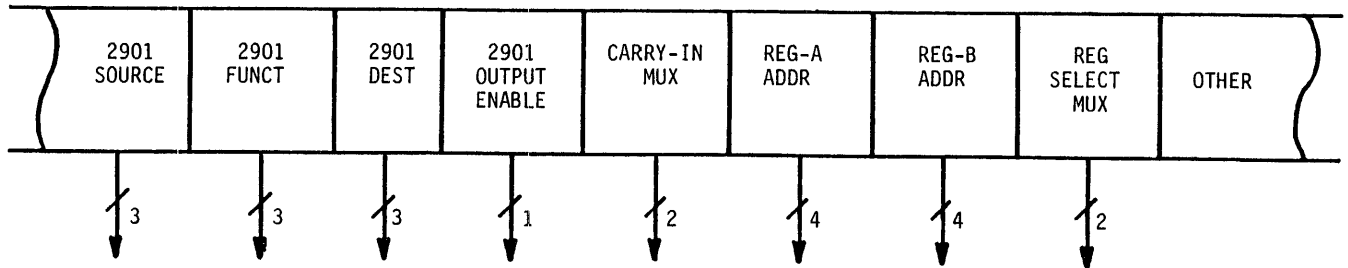
DESTINATION CONTROL

Mnemonic	MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
	I ₈	I ₇	I ₆	Octal Code	Shift	Load	Shift	Load		RAM ₀	RAM ₃	Q ₀	Q ₃
QREG	L	L	L	0	X	NONE	NONE	F → Q	F	X	X	X	X
NOP	L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
RAMA	L	H	L	2	NONE	F → B	X	NONE	A	X	X	X	X
RAMF	L	H	H	3	NONE	F → B	X	NONE	F	X	X	X	X
RAMQD	H	L	L	4	DOWN	F/2 → B	DOWN	Q/2 → Q	F	F ₀	IN ₃	Q ₀	IN ₃
RAMD	H	L	H	5	DOWN	F/2 → B	X	NONE	F	F ₀	IN ₃	Q ₀	X
RAMQU	H	H	L	6	UP	2F → B	UP	2Q → Q	F	IN ₀	F ₃	IN ₀	Q ₃
RAMU	H	H	H	7	UP	2F → B	X	NONE	F	IN ₀	F ₃	X	Q ₃

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state

B = Register Addressed by B inputs.

UP is toward MSB, DOWN is toward LSB.



Am2901 PIPELINE REQUIREMENTS

2-400

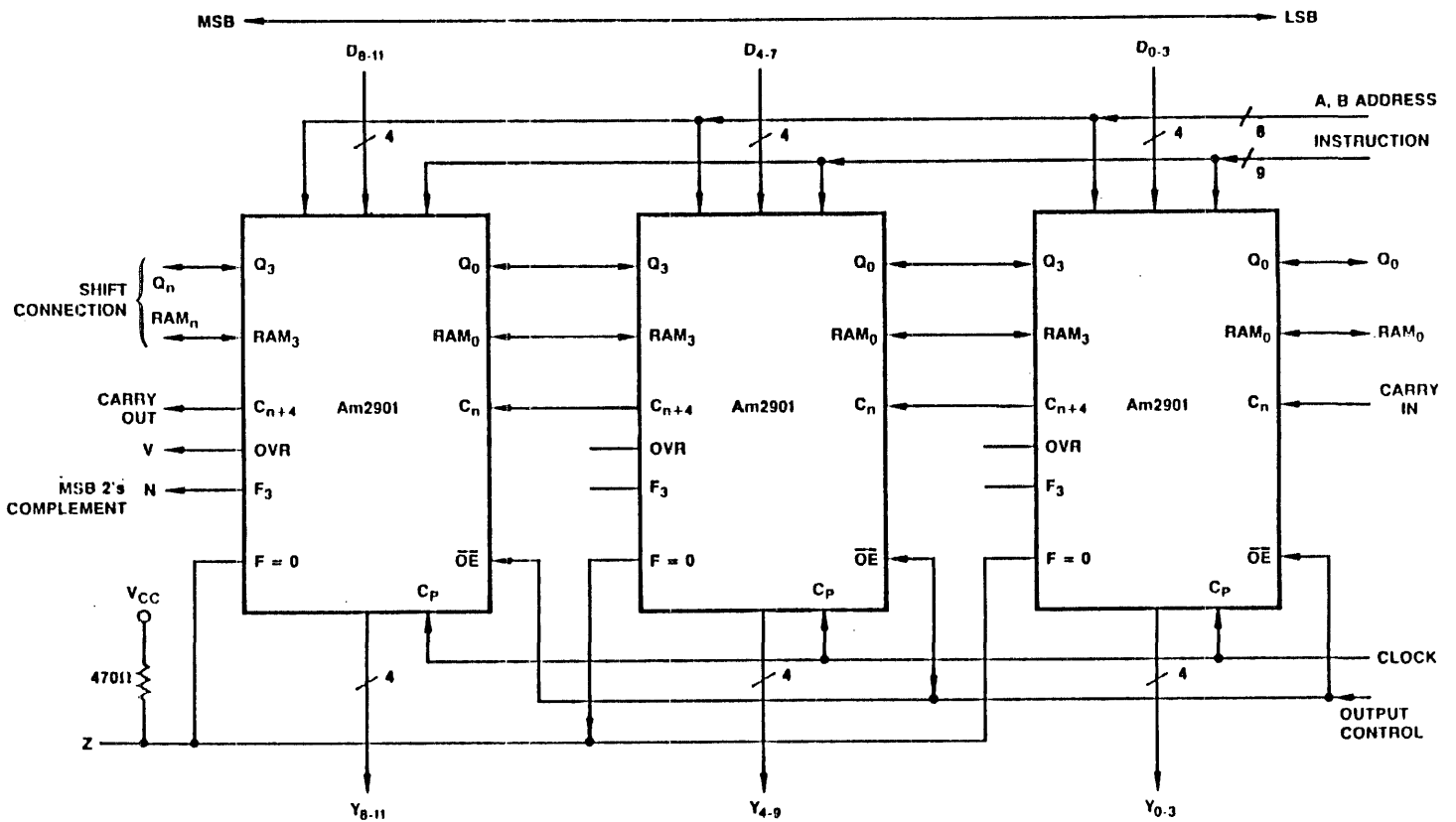
ED2900A

2-400

INTERCONNECTION OF SLICES

Am2901

12-bit ALU, Ripple Carry



Am2902A**CARRY LOOKAHEAD DEVICE**

To develop the implemented equations for the carry lookahead device, a single bit addition is considered first, then a single ALU device is considered next and finally, a combination of slices.

- A carry, C_{i+1} , from the i 'th bit location is either

- generated (G_i) at the i 'th position, i.e.,

$$G_i = A_i B_i \text{ (Boolean AND)}$$

where A_i = i 'th bit of augend
 B_i = i 'th bit of addend

- or propagated (P_i) across the i 'th bit position if $C_i = 1$, i.e.,

$P_i C_i$ where

$$P_i = A_i B_i + A_i \bar{B}_i \text{ (Boolean Exclusive OR)}$$

Carry-out, C_{i+1} , of the i 'th bit position is then

$$C_{i+1} = G_i + P_i C_i$$

CARRY LOOKAHEAD (CONT'D)

- Internal to each 4 bit slice, the carry value is developed using the i 'th bit equations for $i = 1, 2, 3, 4$. Note that each bit can calculate P_i and G_i without the i 'th bit carry-in value. The resulting carry equations for each bit are

$$C_0 = C_{in} \quad (\text{carry-in})$$

$$C_1 = G_0 + P_0 C_{in} \quad + = \text{Boolean OR}$$

$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$

$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$

$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

- These carry bit equations are calculated and the "modulus two" sum (Exclusive OR) of each augend bit, addend bit and carry bit generates the 4 bit sum. For example, consider the addition of two 4-bit binary numbers (augend = 0111, addend = 1001):

$$\begin{array}{r} 1 \ 1110 \ \text{carry } (C_{in} = 0) \\ 0111 \ \text{augend} \\ 1001 \ \text{addend} \end{array}$$

$$\overline{0000} \quad \text{4-bit sum with } C_4 = 1$$

$$\text{where } G_0 = 1 \quad P_0 = 0 \quad C_1 = 1$$

$$G_1 = 0 \quad P_1 = 1 \quad C_2 = 1$$

$$G_2 = 0 \quad P_2 = 1 \quad C_3 = 1$$

$$G_3 = 0 \quad P_3 = 1 \quad C_4 = 1$$

CARRY LOOKAHEAD (CONT'D)

- Now, since it is desired to connect ALU slices together why not consider the use of the carry generate and propagate equations for a combination of slices. To accomplish this task the generated and propagated values must be developed at the slice level. The associated equations for the j'th slice are:

$$G_j = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0$$

$$P_j = P_3 P_2 P_1 P_0$$

- Thus, each slice can develop G_j and P_j internally and provide the values on the device pin connections. Then each slice can receive its carry-in, C_j , as soon as the associated lower significant slice produces P_j and G_j .
- Therefore, Carry-out of the j'th slice (which is Carry-in to the [j + 1]st slice) can be calculated from P and G of the lesser slices plus C_{jn} . Using an external carry-look-ahead device, then the Carry-out of each slice can be generated faster than that using a ripple carry connection.

CARRY LOOKAHEAD (CONT'D)

- Considering a 16-bit ALU, the external Carry-in equations for the three most significant slices using the equations similar to those internal to the slice are given by:

$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_1 = G_0 + P_0C_0$$

$$C_0 = C_{in} \text{ (carry-in)}$$

where P_j and G_j are generated by j 'th ALU slice ($j=0,1,2,3$) and C_j is the Carry-in for each ALU slice generated by the Am2902A. For example, with $G_2 = P_0 = P_1 = P_2 = 0$ and $G_0 = G_1 = 1$:

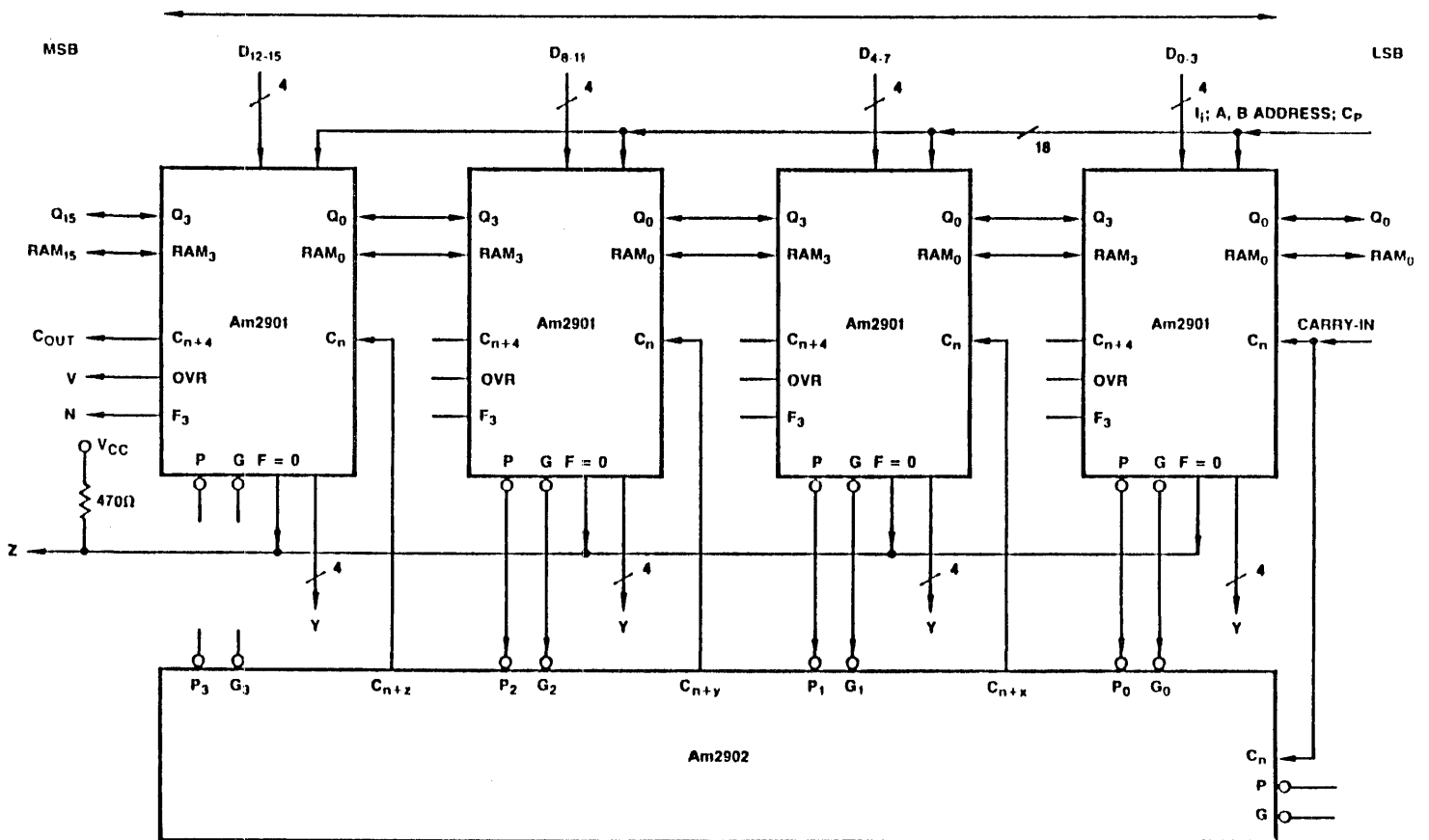
```

0 ---0 ---1 ---1 ---0 carry from Am2902
0011 0111 0111 0111 augend
0001 0101 1010 1001 addend
-----
0100 1101 0010 0000 sum

```

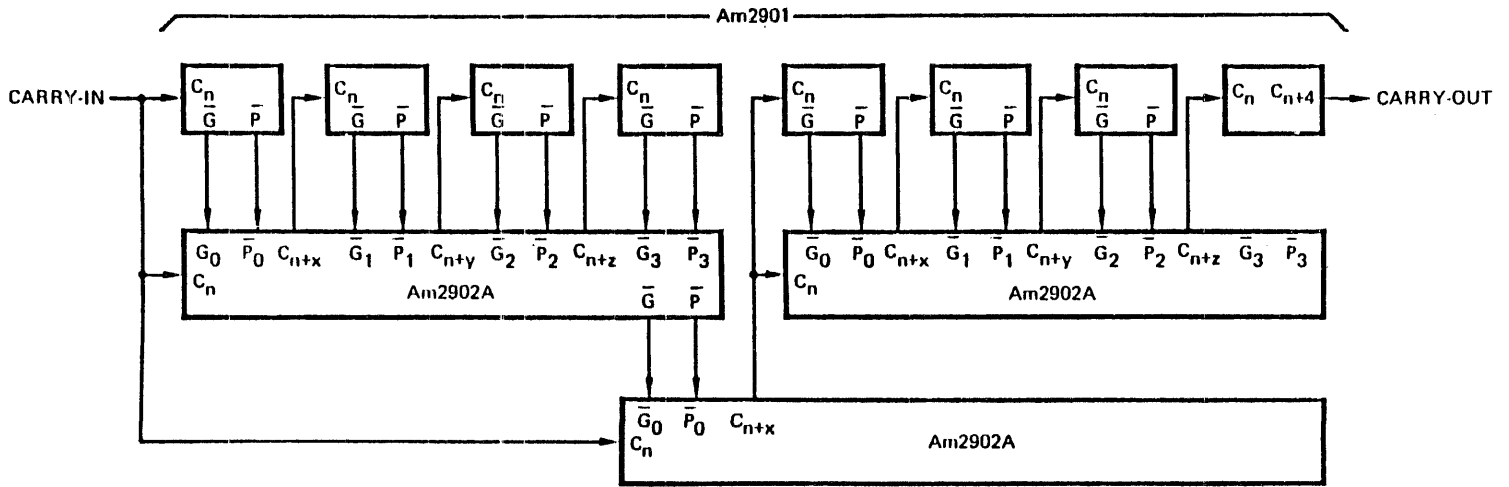
- The Am2902A look-ahead carry generator performs these calculations for four slices (see figures). More Am2902As can be ganged for longer word size; a hierarchy of carry lookahead operations.
- The result is that the Carry-in time to any slice is equal to the time (t_{pg}) to generate P_j and G_j for all slices in parallel plus the delay (t_{02}) through the Am2902 (see the AMD Data Book). The total add time is this value plus the Am2901 add time (t_{add}).

16-bit ALU, Lookahead Carry



32-bit ALU

NOTE ← *msb*
lsb →



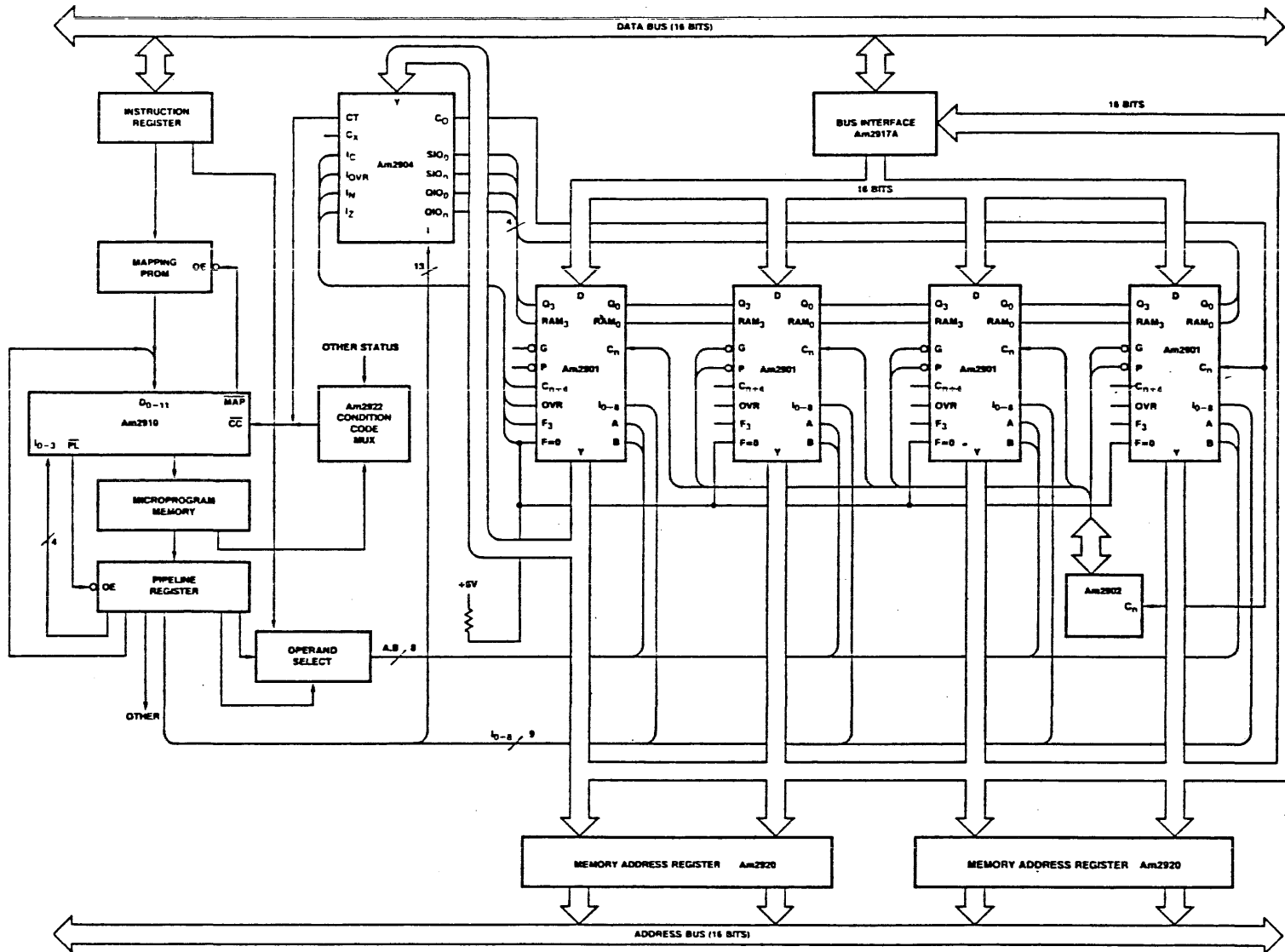
CONNECTING THE SHIFTERS

- For rotates
 - connect RAM \emptyset - RAMn
 - connect Q \emptyset - Qn

- For shifts
 - connect '0' or '1' to be shifted in as required
 - could shift into MSB or LSB
 - needed for both Q and RAM

- For double-length arithmetic shift
 - connect RAM \emptyset to Qn
 - connect output Fn to RAMn for down shift (sign)

- To connect the various Am2901 pins use SSI logic or use the Am2904, Status and Shift Control Logic.



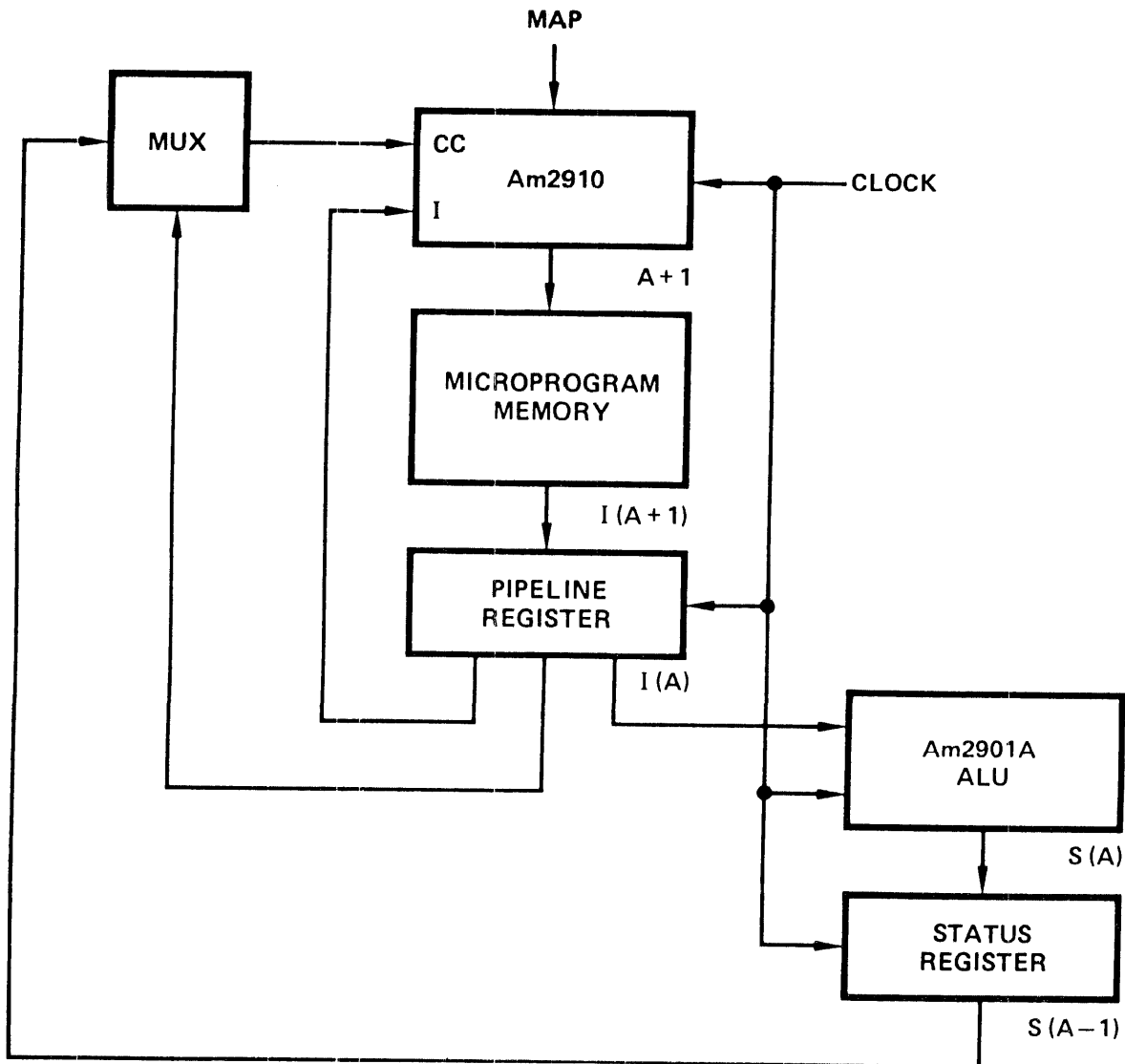
Typical Application of Am2904 with Am2901.

RECOMMENDED MICROPROGRAMMED SYSTEM ARCHITECTURE

- This is the architecture that has been developed

- Single pipeline register allows two parallel operations:
 - fetch next microinstruction (sequencer operations)
 - execute current microinstruction (ALU operations)

Recommended Architecture



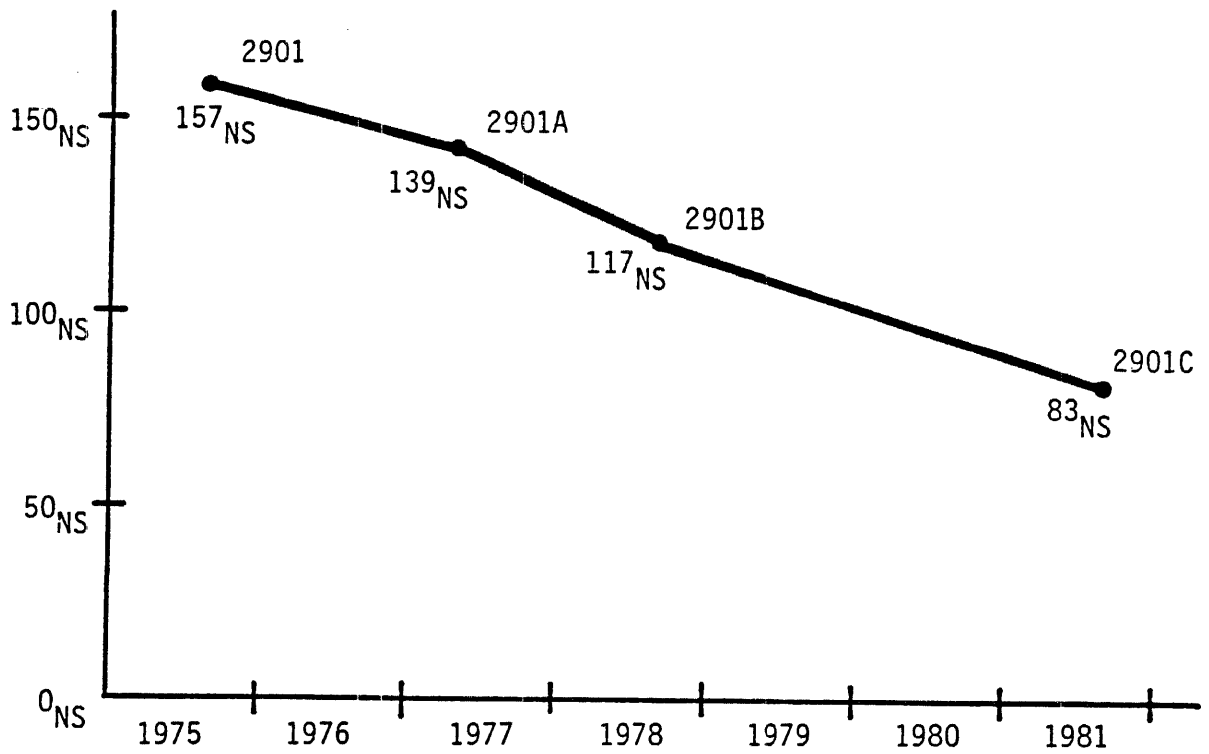
2-530

ED2900A

2-530

IMPROVING SPEED

Am2901C



**Worst Case 16 to 16-bit Registers
Add Time with Am2901C**

SAMPLE MICROCODE - AM2901

- Various ALU processes are described on the following pages in terms of Am2901 micro-operations:
 - Increment a register and output original value
 - Byte swap

Am2901

INCREMENT A REGISTER

AND

OUTPUT ITS ORIGINAL VALUE

- This operation is required in the macroinstruction fetch cycle for the PC (macro program counter).
- Assume the register is loaded with macroinstruction address in main memory.
- Assume that Reg 15 is the PC.

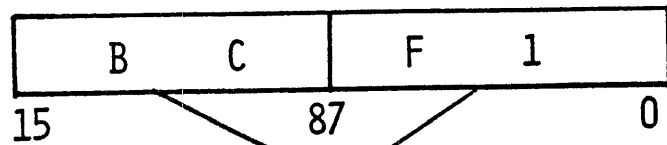
Am2901 SOLUTION

- Address A and Address B are both set to 15 from pipeline
- I₂₁₀ (source) is set to 3 to select source operands \emptyset and B (ZB)
- I₅₄₃ (function) is set to \emptyset (ADD)
- I₈₇₆ (destination) is set to 2 to select F \rightarrow B and A \rightarrow Y (RAMA)
- C_{in} is set to 1 (ONE)

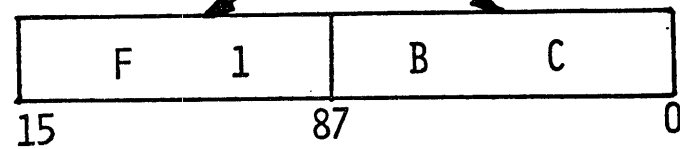
Am2901**BYTE SWAP:**

to exchange two halves of a 16 bit word

CHANGE:

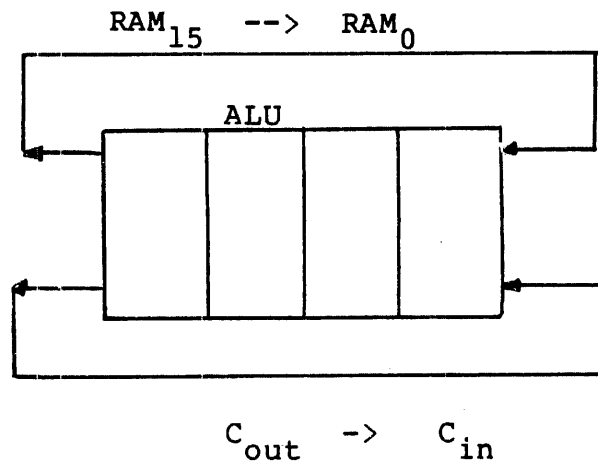


INTO:



- Assume again register 15.
(Number chosen for no particular reason.)

CONNECTIONS:



SHIFTING SEQUENCE:

REG 15				
START	1011	1100	1111	0001
2A	0111	1001	1110	0011
SHIFT	1111	0011	1100	0110
2A	1110	0111	1000	1101
SHIFT	1100	1111	0001	1011
2A	1001	1110	0011	0111
SHIFT	0011	1100	0110	1111
2A	0111	1000	1100	1110
SHIFT	1111	0001	1011	1100

ADD SHIFT REGISTER TO ITSELF
SHIFT LEFT (ROTATE)

ROTATES 2 BITS IN
ONE MICROCYCLE

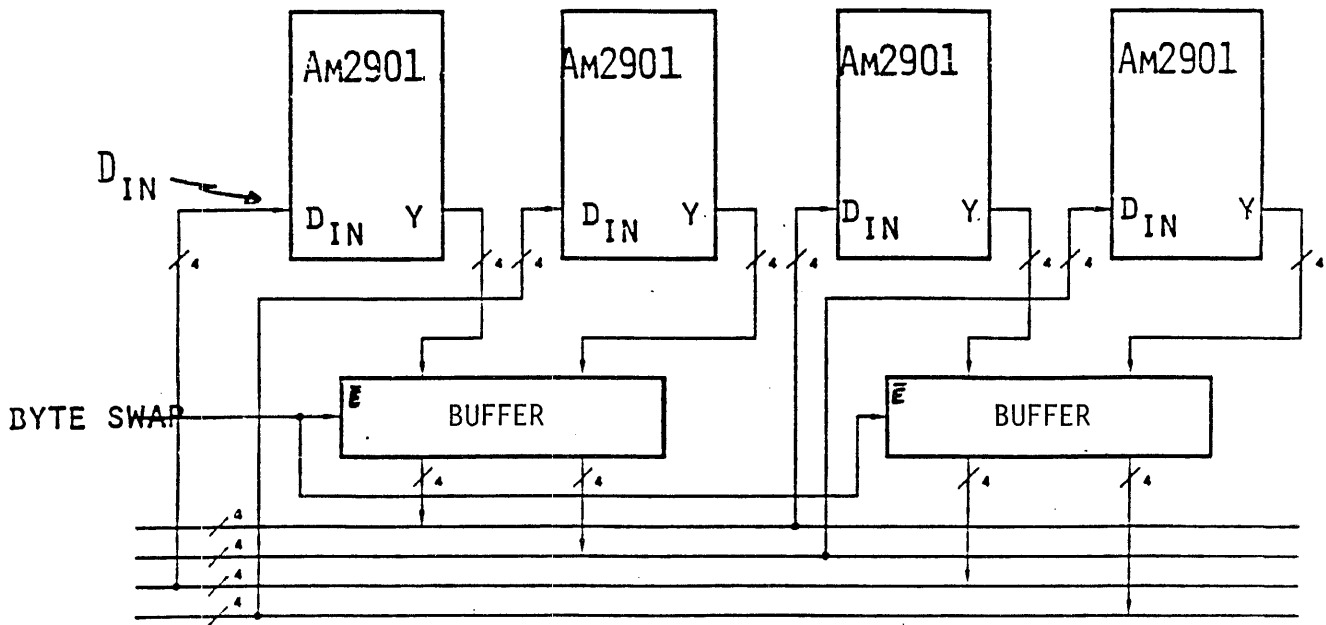
Am2901 SOLUTION

- Address A and address B are both set to 15 (F)
- I_{210} is set to 1 to select source operands A and B (AB)
- I_{543} is set to 0 (ADD)
- I_{876} is set to 7 to select 2F → B RAM shift up (toward MSB) (RAMU)
- C_{in} is set to C_{out}
- Repeat 4 times (2 bit - rotate per cycle)

I_{876}	I_{543}	I_{210}	C_{IN}
7	0	1	C_{OUT}
RAMU	ADD	AB	

HARDWARE BYTE SWAP IMPLEMENTATION

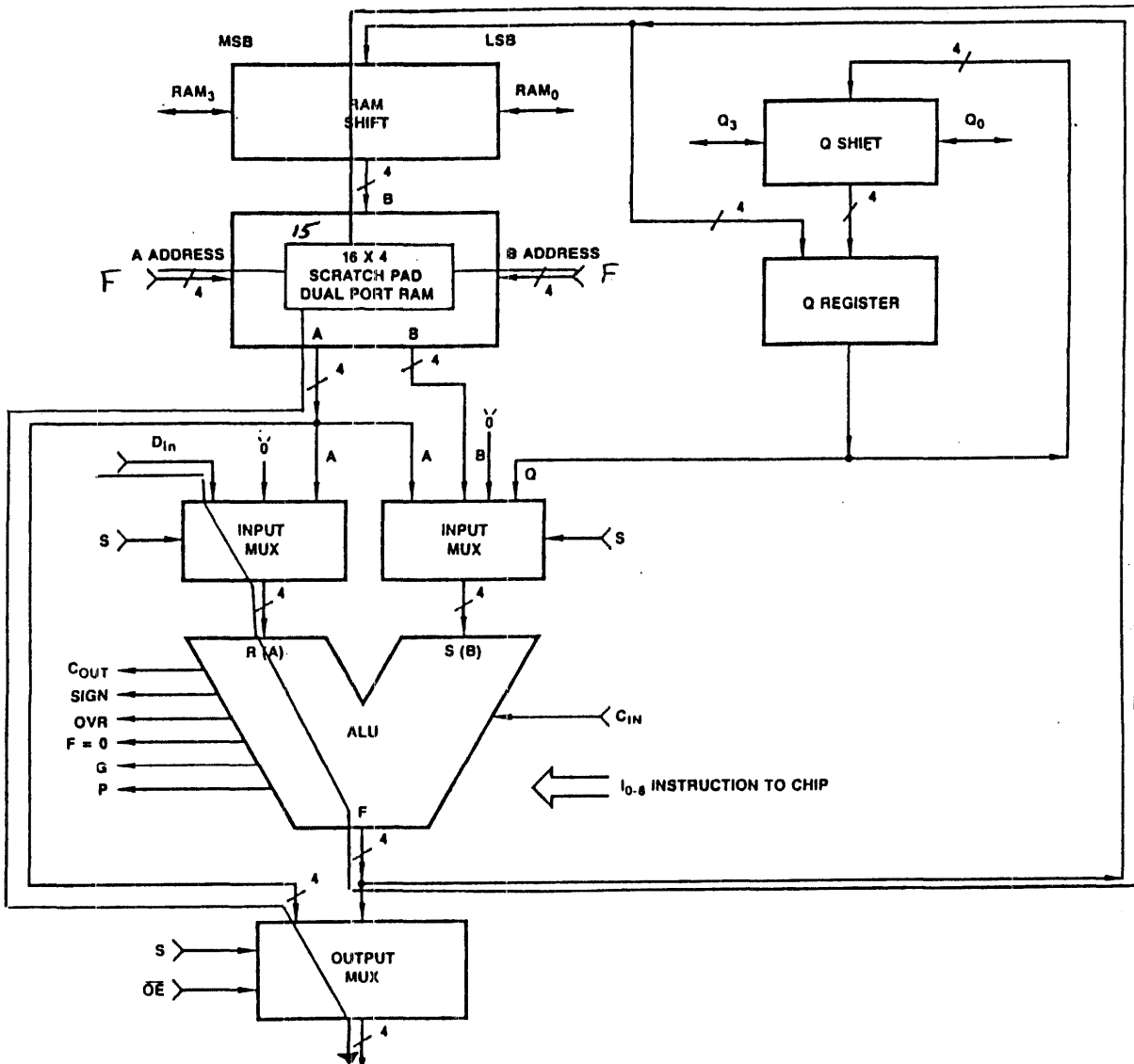
- Trade-off added logic hardware for less micromemory (ROM) and faster execution time.
- Output register to be manipulated.
- Use Am25LS240/244 tri-state buffers with permuted outputs to input ports.



BYTE SWAP HARDWARE WITH THE Am2901

SOURCE	FUNC	DEST
DZ	ADD OR	RAMA

"OR" is faster than "ADD"



CLASS EXERCISE - Am2901

- Turn to the Am2901 exercises in the ED2900A Exercise and Laboratory Manual and work numbers 1 through 14.

- A coding sheet is provided.

Use mnemonics!

2-660

ED2900A

2-660

INTRODUCING THE SUPER SLICE™

Am2903/ Am29203

IMPROVEMENTS BEYOND THE Am2901

- Add ability to expand the multiport "scratchpad" register memory (unlimited expansion).
 - Simplify multiplication
 - Simplify division
- Add normalization for floating point numbers for use in arithmetic operations.

1. XXXX E ± XX

 Mantissa Exponent

- Add fault tolerance/fault detection features: parity of ALU output.
- Add two's complement sign extend for byte manipulation.
- Add three address instruction for faster operation.

ie: C = A + B

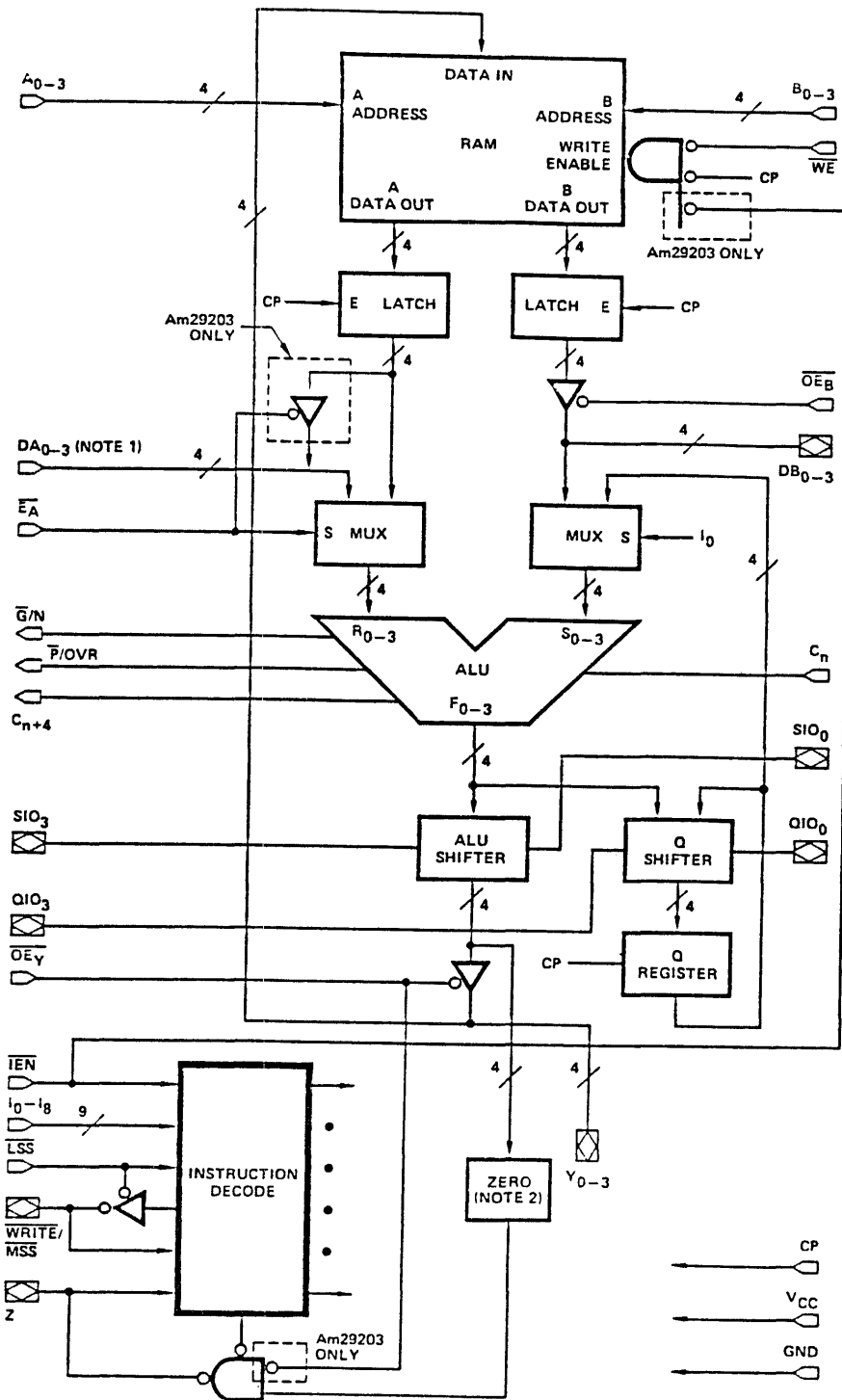
COMPARISONS

<u>The Am2903/29203</u>	<u>The Am2901</u>
48 pins	40 pins
Higher throughput	Faster clock speed
Q register can shift on its own	Q register shifted only when RAM also shifted
Arithmetic & logical shifts	Logical shifts only
Expandable RAM	RAM not designed for expansion
Two- or three-address operations	Two-address operation only (not designed for three)
DA ₀₋₃ input	DA ₀₋₃ input
DA ₀₋₃ output (29203)	-
DA ₀₋₃ input/output	-
Y ₀₋₃ input/output	Y ₀₋₃ output only
Arithmetic operation plus shift and output	Requires two microcycles (shift before RAM load)
Parity bit generation	-
Special functions	-
Internal logic support	(requires external <u>assist</u> logic)

Am2903/29203 DISTINCTIVE CHARACTERISTICS

- Three port RAM - same as Am2901
- 16 ALU functions
 - Am2901 functions are a proper subset
- 9 (Am2903) or 16 (Am29203) special functions
- Expandable registers
- Microprogrammable
 - 9 bits of instruction
 - 4 enables
 - 2 position selects:
 - Least Significant Slice - LSS
 - Most Significant Slice - MSS
- Four status flags
 - similar to 2901
- Q register
 - capable of independent operation
- Two shifts
 - arithmetic
 - logical
- Uses shared pins - some lines multifunctional

Am2903 / Am29203



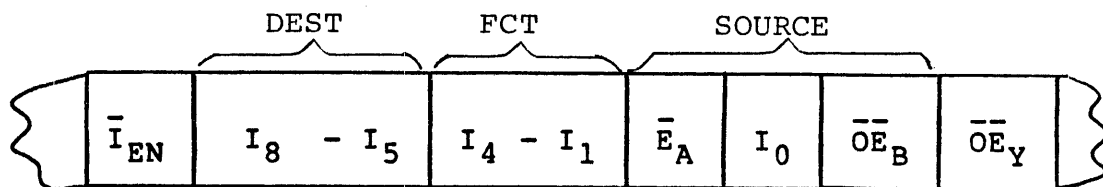
Am2903/29203 DIFFERENCES

- DA_{0-3} bidirectional on Am29203
- External \bar{I}_{EN} internally connected to write enable on Am29203 for use with ALU RAM write operation
- Zero detect on ALU shifter output on Am29203 on output of buffer on Am2903
- \bar{OE}_y connected to Z pin on Am29203

Am29203

- Faster than Am2903, but not Am2903A
- Can handle byte operations better
- Has 16 special functions

Am2903/29203 Microinstruction format (ALU only)



- Under normal operation

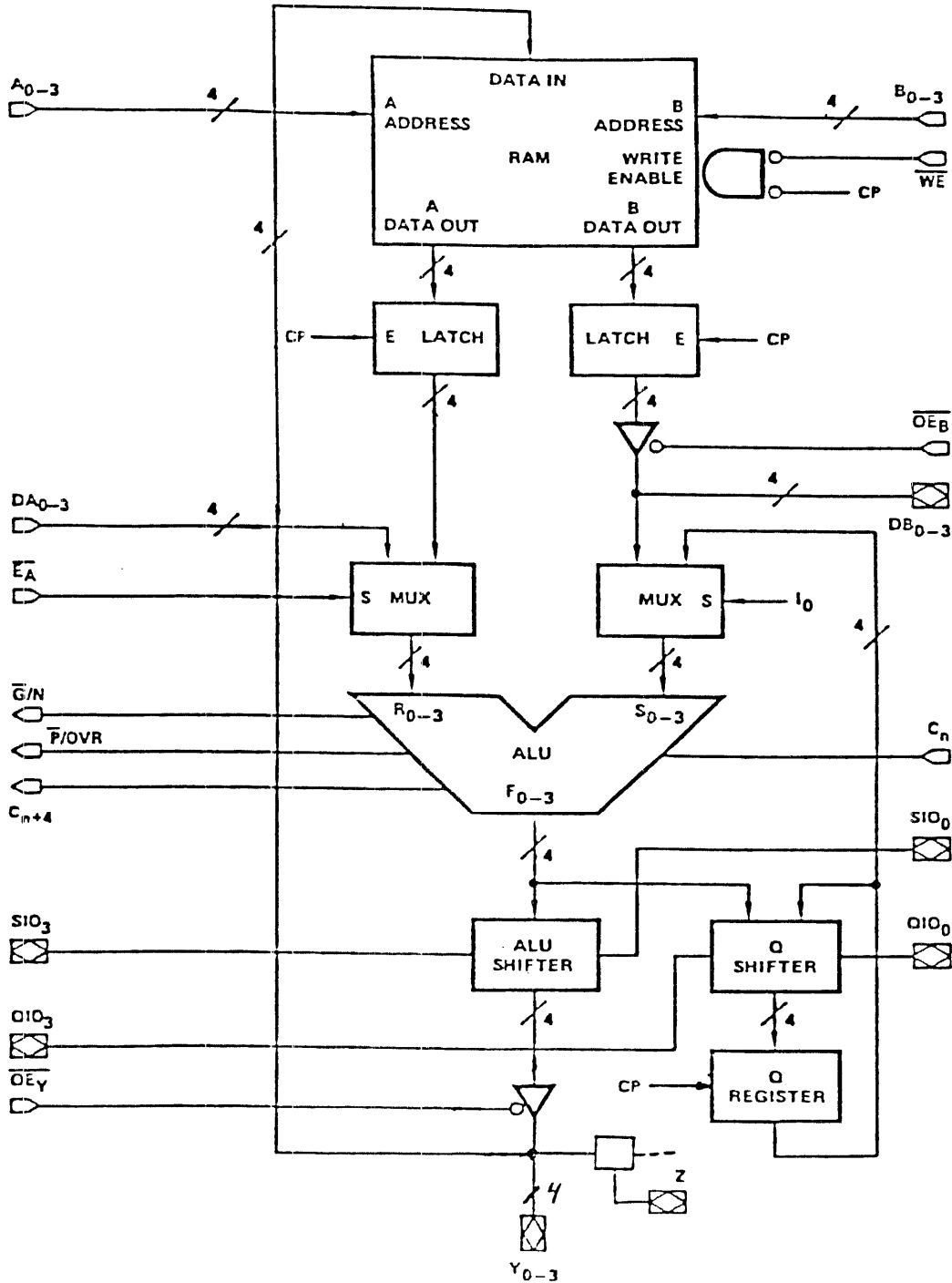
Destination is controlled by $I_8 - I_5$

Function is controlled by $I_4 - I_1$

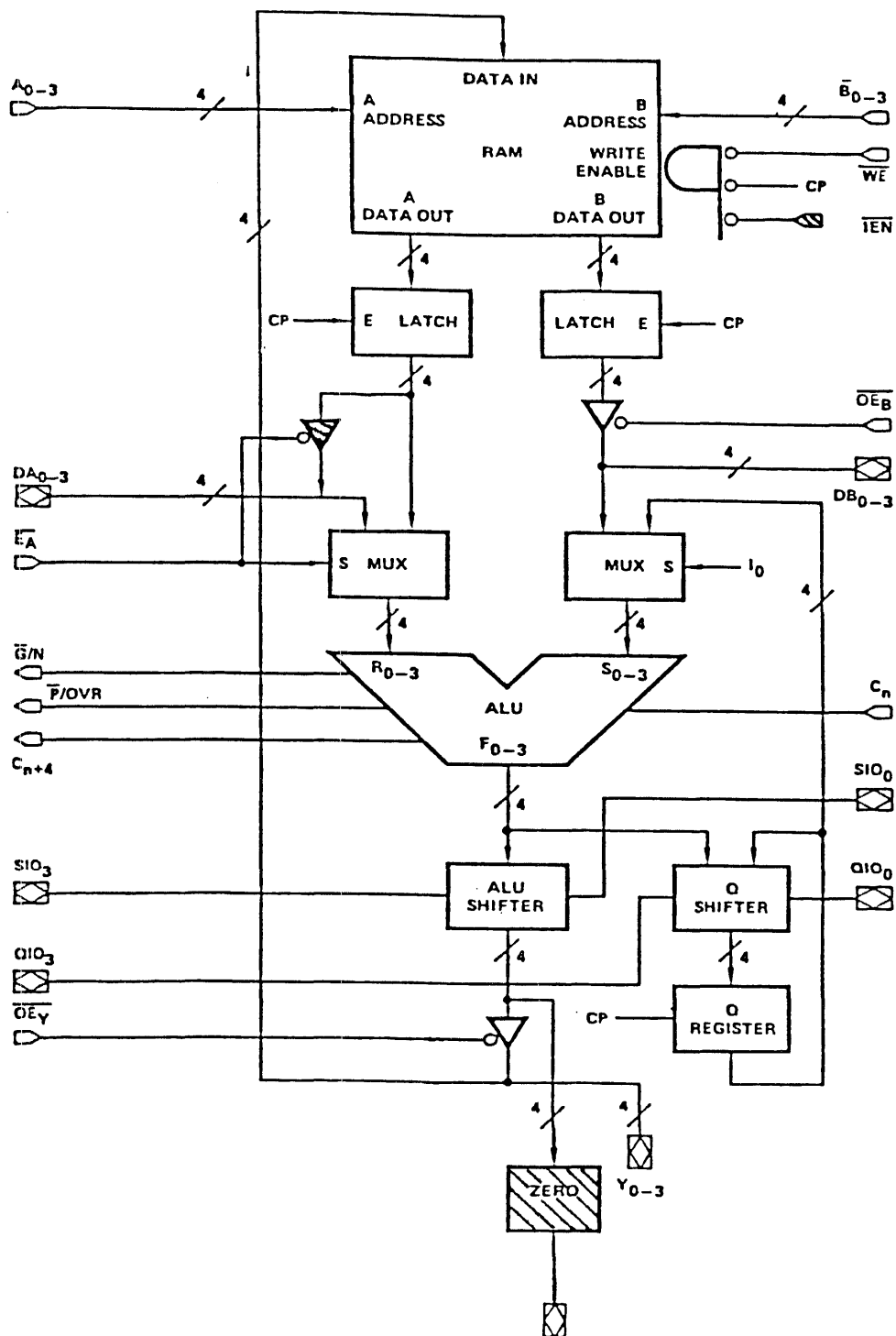
- Under special function operation (when $I_4 - I_0$ are all low)

Destination and function are controlled by $I_8 - I_5$

Am2903



Am29203



Am2903 Operand Sources

E_A	A-REGISTER	\overline{OE}_B	B-REGISTER
L	INTERNAL	L	INTERNAL
H	EXTERNAL	H	EXTERNAL

I_0 S-SOURCE

L RAM/B-PORT

H Q

ALU OPERAND SOURCES

\overline{E}_A	I_0	\overline{OE}_B	ALU Operand R	ALU Operand S
L	L	L	RAM Output A	RAM Output B
L	L	H	RAM Output A	DB ₀₋₃
L	H	X	RAM Output A	Q Register
H	L	L	DA ₀₋₃	RAM Output B
H	L	H	DA ₀₋₃	DB ₀₋₃
H	H	X	DA ₀₋₃	Q Register

L = LOW

H = HIGH

X = Don't Care

Note: All 8 input codes are valid, but only 6 combinations are possible. (Note Don't Cares)

Am2903 / Am29203 ALU Functions

Arithmetic	Logic
$S + R + C_N$	R AND S
$S - R + C_N - 1$	R OR S
$R - S + C_N - 1$	R NAND S
$S + C_N^*$	R NOR S
$\bar{S} + C_N^*$	R EXOR S
$R + C_N$	R EXNOR S
$\bar{R} + C_N$	\bar{R} AND S
	HIGH'S*
	LOW'S*

- For Am29203, I_0 must be high, hence source must be RAMAQ or DAQ.

Am2903 ALU Functions

I_4	I_3	I_2	I_1	Hex Code	ALU Functions
L	L	L	L	0	$I_0 = L$ Special Functions
					$I_0 = H$ $F_i = \text{HIGH}$
L	L	L	H	1	$F = S \text{ Minus } R \text{ Minus } 1 \text{ Plus } C_n$
L	L	H	L	2	$F = R \text{ Minus } S \text{ Minus } 1 \text{ Plus } C_n$
L	L	H	H	3	$F = R \text{ Plus } S \text{ Plus } C_n$
L	H	L	L	4	$F = S \text{ Plus } C_n$
L	H	L	H	5	$F = \bar{S} \text{ Plus } C_n$
L	H	H	L	6	$F = R \text{ Plus } C_n$
L	H	H	H	7	$F = \bar{R} \text{ Plus } C_n$
H	L	L	L	8	$F_i = \text{LOW}$
H	L	L	H	9	$F_i = \bar{R}_i \text{ AND } S_i$
H	L	H	L	A	$F_i = R_i \text{ EXCLUSIVE NOR } S_i$
H	L	H	H	B	$F_i = R_i \text{ EXCLUSIVE OR } S_i$
H	H	L	L	C	$F_i = R_i \text{ AND } S_i$
H	H	L	H	D	$F_i = R_i \text{ NOR } S_i$
H	H	H	L	E	$F_i = R_i \text{ NAND } S_i$
H	H	H	H	F	$F_i = R_i \text{ OR } S_i$

L = LOW

H = HIGH

 $i = 0 \text{ to } 3$

Am29203 ALU Functions

I ₄	I ₃	I ₂	I ₁	I ₀	ALU Functions
L	L	L	L	L	Special Functions
L	L	L	L	H	$F_i = \text{HIGH}$
L	L	L	H	X	$F = S \text{ Minus } R \text{ Minus } 1 \text{ Plus } C_n$
L	L	H	L	X	$F = R \text{ Minus } S \text{ Minus } 1 \text{ Plus } C_n$
L	L	H	H	X	$F = R \text{ Plus } S \text{ Plus } C_n$
L	H	L	L	X	$F = S \text{ Plus } C_n$
L	H	L	H	X	$F = \bar{S} \text{ Plus } C_n$
L	H	H	L	L	Reserved Special Functions
L	H	H	L	H	$F = R \text{ Plus } C_n$
L	H	H	H	L	Reserved Special Functions
L	H	H	H	H	$F = \bar{R} \text{ Plus } C_n$
H	L	L	L	L	Reserved Special Functions
H	L	L	L	H	$F_i = \text{LOW}$
H	L	L	H	X	$F_i = \bar{R}_i \text{ AND } S_i$
H	L	H	L	X	$F_i = R_i \text{ EXCLUSIVE NOR } S_i$
H	L	H	H	X	$F_i = R_i \text{ EXCLUSIVE OR } S_i$
H	H	L	L	X	$F_i = R_i \text{ AND } S_i$
H	H	L	H	X	$F_i = R_i \text{ NOR } S_i$
H	H	H	L	X	$F_i = R_i \text{ NAND } S_i$
H	H	H	H	X	$F_i = R_i \text{ OR } S_i$

L = LOW

H = HIGH

i = 0 to 3

X = LOW or HIGH

ALU DESTINATION CONTROL

- Destination is controlled by I8-I7-I6-I5
- Includes choice of down-, up-, or no-shift
- Allows choice of logical or arithmetic shift (RAM only)
- Controls whether data is written to RAM registers

ALU DESTINATION CONTROL FOR I₀ OR I₁ OR I₂ OR I₃ = HIGH, IEN = LOW

I ₈	I ₇	I ₆	I ₅	Hex Code	ALU Shifter Function	SIO ₃		Y ₃		Y ₂		Y ₁	Y ₀	SIO ₀	Write	Q Reg & Shifter Function	QIO ₃	QIO ₀
						Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices							
L	L	L	L	0	Arith. F2→Y	Input	Input	F ₃	SIO ₃	SIO ₃	F ₃	F ₂	F ₁	F ₀	Hold	Hi-Z	Hi-Z	
L	L	L	H	1	Log. F2→Y	Input	Input	SIO ₃	SIO ₃	SIO ₃	F ₃	F ₂	F ₁	F ₀	Hold	Hi-Z	Hi-Z	
L	L	H	L	2	Arith. F2→Y	Input	Input	F ₃	SIO ₃	SIO ₃	F ₃	F ₂	F ₁	F ₀	Log. Q2→Q	Input	Q ₀	
L	L	H	H	3	Log. F2→Y	Input	Input	SIO ₃	SIO ₃	SIO ₃	F ₃	F ₂	F ₁	F ₀	Log. Q2→Q	Input	Q ₀	
L	H	L	L	4	F→Y	Input	Input	F ₃	F ₃	F ₂	F ₂	F ₁	F ₀	Parity	Hold	Hi-Z	Hi-Z	
L	H	L	H	5	F→Y	Input	Input	F ₃	F ₃	F ₂	F ₂	F ₁	F ₀	Parity	Log. Q2→Q	Input	Q ₀	
L	H	H	L	6	F→Y	Input	Input	F ₃	F ₃	F ₂	F ₂	F ₁	F ₀	Parity	F→Q	Hi-Z	Hi-Z	
L	H	H	H	7	F→Y	Input	Input	F ₃	F ₃	F ₂	F ₂	F ₁	F ₀	Parity	F→Q	Hi-Z	Hi-Z	
H	L	L	L	8	Arith. 2F→Y	F ₂	F ₃	F ₃	F ₃	F ₁	F ₁	F ₀	SIO ₀	Input	Hold	Hi-Z	Hi-Z	
H	L	L	H	9	Log. 2F→Y	F ₃	F ₃	F ₂	F ₂	F ₁	F ₁	F ₀	SIO ₀	Input	Hold	Hi-Z	Hi-Z	
H	L	H	L	A	Arith. 2F→Y	F ₂	F ₃	F ₃	F ₂	F ₁	F ₁	F ₀	SIO ₀	Input	Log. 2Q→Q	Q ₃	Input	
H	L	H	H	B	Log. 2F→Y	F ₃	F ₃	F ₂	F ₂	F ₁	F ₁	F ₀	SIO ₀	Input	Log. 2Q→Q	Q ₃	Input	
H	H	L	L	C	F→Y	F ₃	F ₃	F ₃	F ₃	F ₂	F ₂	F ₁	F ₀	Hi-Z	Hold	Hi-Z	Hi-Z	
H	H	L	H	D	F→Y	F ₃	F ₃	F ₃	F ₃	F ₂	F ₂	F ₁	F ₀	Hi-Z	Log. 2Q→Q	Q ₃	Input	
H	H	H	L	E	SIO ₀ →Y ₀ , Y ₁ , Y ₂ , Y ₃	SIO ₀	SIO ₀	SIO ₀	SIO ₀	SIO ₀	SIO ₀	SIO ₀	SIO ₀	SIO ₀	Input	Hold	Hi-Z	Hi-Z
H	H	H	H	F	F→Y	F ₃	F ₃	F ₃	F ₃	F ₂	F ₂	F ₁	F ₀	Hi-Z	Hold	Hi-Z	Hi-Z	

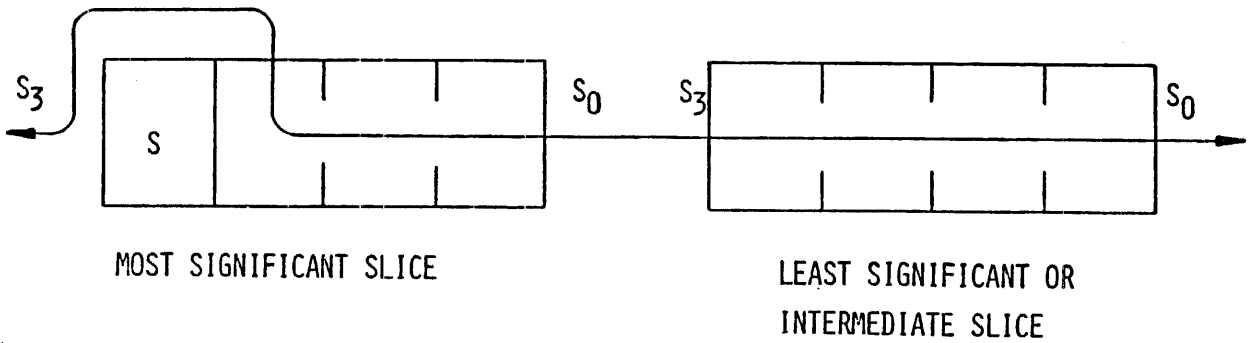
Parity = F₃ ∨ F₂ ∨ F₁ ∨ F₀ ∨ SIO₃
 ∨ = Exclusive OR

L = LOW
 H = HIGH

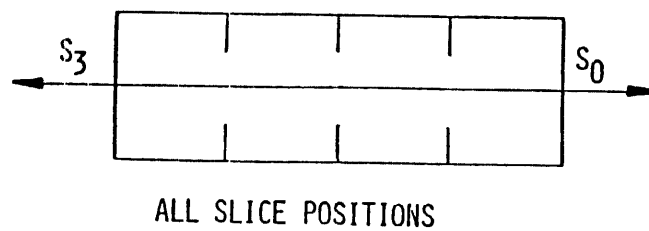
Hi-Z = High Impedance

LOGICAL VERSUS ARITHMETIC SHIFT:

Am2903 Arithmetic Shift Path



Am2903 Logical Shift Path



Am2903/29203 SPECIAL FUNCTIONS**Am2903 and Am29203:**

- Parity
- Sign extension
- Sign magnitude/two's complement conversion
- Unsigned multiply
- Two's complement multiply
- Increment by 1 or 2
- Single length normalize
- Double length normalize
- Two's complement divide

Am29203 only:

- BCD/binary conversion
- Decrement by 1 or 2
- BCD divide by 2
- BCD add and subtract

Am29203 Special Functions

(Hex) 167165	I ₄	(Hex) 1312110	Special Function	ALU Function	ALU Shifter Function	SIO ₃		SIO ₀	Q Reg & Shifter Function	QIO ₃	QIO ₀	WRITE
						Most Sig Slice	Other Slices					
0	L	0	Unsigned Multiply	$F = S + C_n$ if Z = L $F = R + S + C_n$ if Z = H	Log F/2 → Y (Note 1)	Z	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
1	L	0	BCD to Binary Conversion	(Note 4)	Log F/2 → Y	Input	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
1	H	0	Multiprecision BCD to Binary	(Note 4)	Log F/2 → Y	Input	Input	F ₀	Hold	Z	Q ₀	L
2	L	0	Two's Complement Multiply	$F = S + C_n$ if Z = L $F = R + S + C_n$ if Z = H	Log F/2 → Y (Note 2)	Z	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
3	L	0	Decrement by One or Two	$F = S - 2 + C_n$	F → Y	Z	Z	Parity	Hold	Z	Z	L
4	L	0	Increment by One or Two	$F = S + 1 + C_n$	F → Y	Input	Input	Parity	Hold	Z	Z	L
5	L	0	Sign/Magnitude Two's Complement	$F = S + C_n$ if Z = L $F = \bar{S} + C_n$ if Z = H	F → Y (Note 3)	Input	Input	Parity	Hold	Z	Z	L
6	L	0	Two's Complement Multiply, Last Cycle	$F = S + C_n$ if Z = L $F = S - R - 1 + C_n$ if Z = H	Log F/2 → Y (Note 2)	Z	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
7	L	0	BCD Divide by Two	(Note 4)	F → Y	Z	Z	Parity	Hold	Z	Z	L
8	L	0	Single Length Normalize	$F = S + C_n$	F → Y	F ₃	F ₃	Z	Log 2Q → Q	Q ₃	Input	L
9	L	0	Binary to BCD Conversion	(Note 5)	Log 2F → Y	F ₃	F ₃	Input	Log 2Q → Q	Q ₃	Input	L
9	H	0	Multiprecision Binary to BCD	(Note 5)	Log 2F → Y	F ₃	F ₃	Input	Hold	Z	Input	L
A	L	0	Double Length Normalize and First Divide Op	$F = S + C_n$	Log 2F → Y	$R_3 \nabla F_3$	F ₃	Input	Log 2Q → Q	Q ₃	Input	L
B	L	0	BCD Add	$F = R + S + C_n$ BCD (Note 6)	F → Y	0	0	Z	Hold	Z	Z	L
C	L	0	Two's Complement Divide	$F = S + R + C_n$ if Z = L $F = S - R - 1 + C_n$ if Z = H	Log 2F → Y	$R_3 \nabla F$	F ₃	Input	Log 2Q → Q	Q ₃	Input	L
D	L	0	BCD Subtract	$F = R - S - 1 + C_n$ BCD (Note 6)	F → Y	0	0	Z	Hold	Z	Z	L
E	L	0	Two's Complement Divide Correction and Remainder	$F = S + R + C_n$ if Z = L $F = S - R - 1 + C_n$ if Z = H	F → Y	F ₃	F ₃	Z	Log 2Q → Q	Q ₃	Input	L
F	L	0	BCD Subtract	$F = S - R - 1 + C_n$ BCD (Note 6)	F → Y	0	0	Z	Hold	Z	Z	L

- Notes: 1. At the most significant slice only, the C_{n+4} signal is internally gated to the Y₃ output.
2. At the most significant slice only, $F_3 \nabla OVR$ is internally gated to the Y₃ output.
3. At the most significant slice only, $S_3 \nabla F_3$ is generated at the Y₃ output.
4. On each slice, $F = S$ if magnitude of S_{0-3} is less than 8 and $F = S$ minus 3 if magnitude of S_{0-3} is 8 or greater.
5. On each slice, $F = S$ if magnitude of S_{0-3} is less than 5 and $F = S$ plus 3 if magnitude of S_{0-3} is 5 or greater. Addition is module 16.
6. Additions and subtractions are BCD adds and subtracts. Results are undefined if R or S are not in valid BCD format.
7. The Q Register cannot be used explicitly as an operand for any Special Functions. It is defined implicitly within the functions.

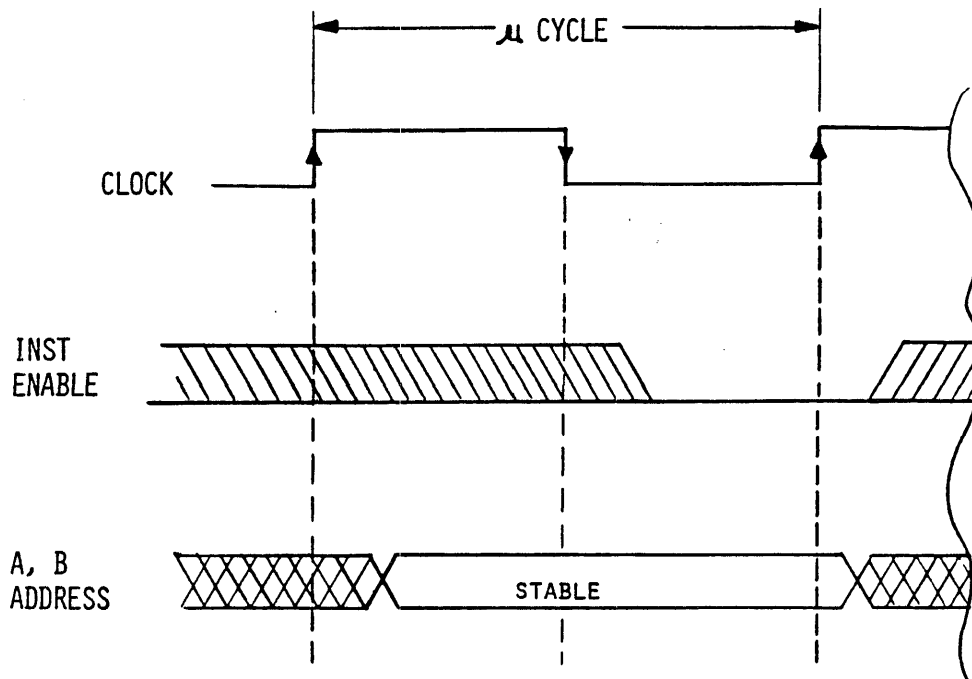
L = LOW
H = HIGH
X = Don't Care
Hi-Z = High Impedance
 ∇ = Exclusive OR
Parity = $SIO_3 \nabla F_3 \nabla F_2 \nabla F_1 \nabla F_0$

Am2903 TWO ADDRESS OPERATION**SOURCES:**

- Registers selected by A-address and B-address

DESTINATION:

- Register selected by B-address



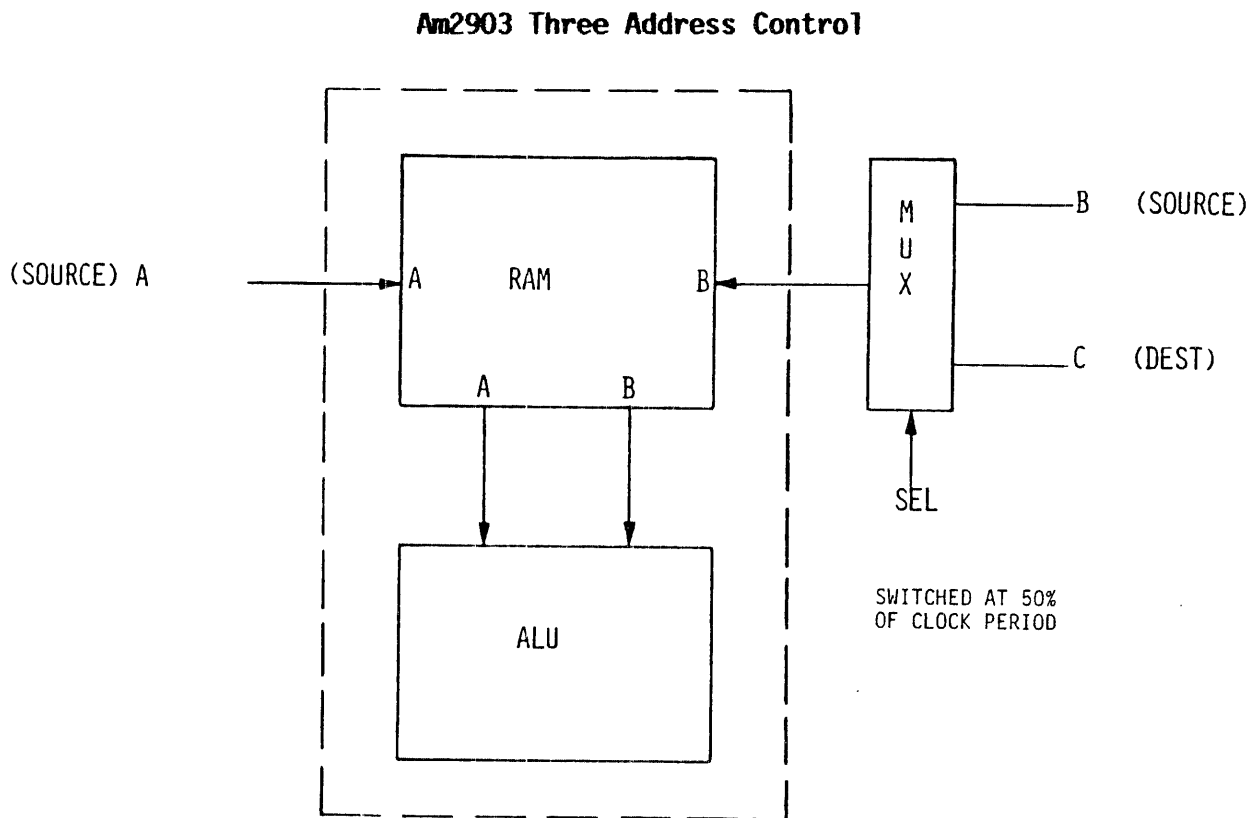
Am2903 THREE ADDRESS OPERATION

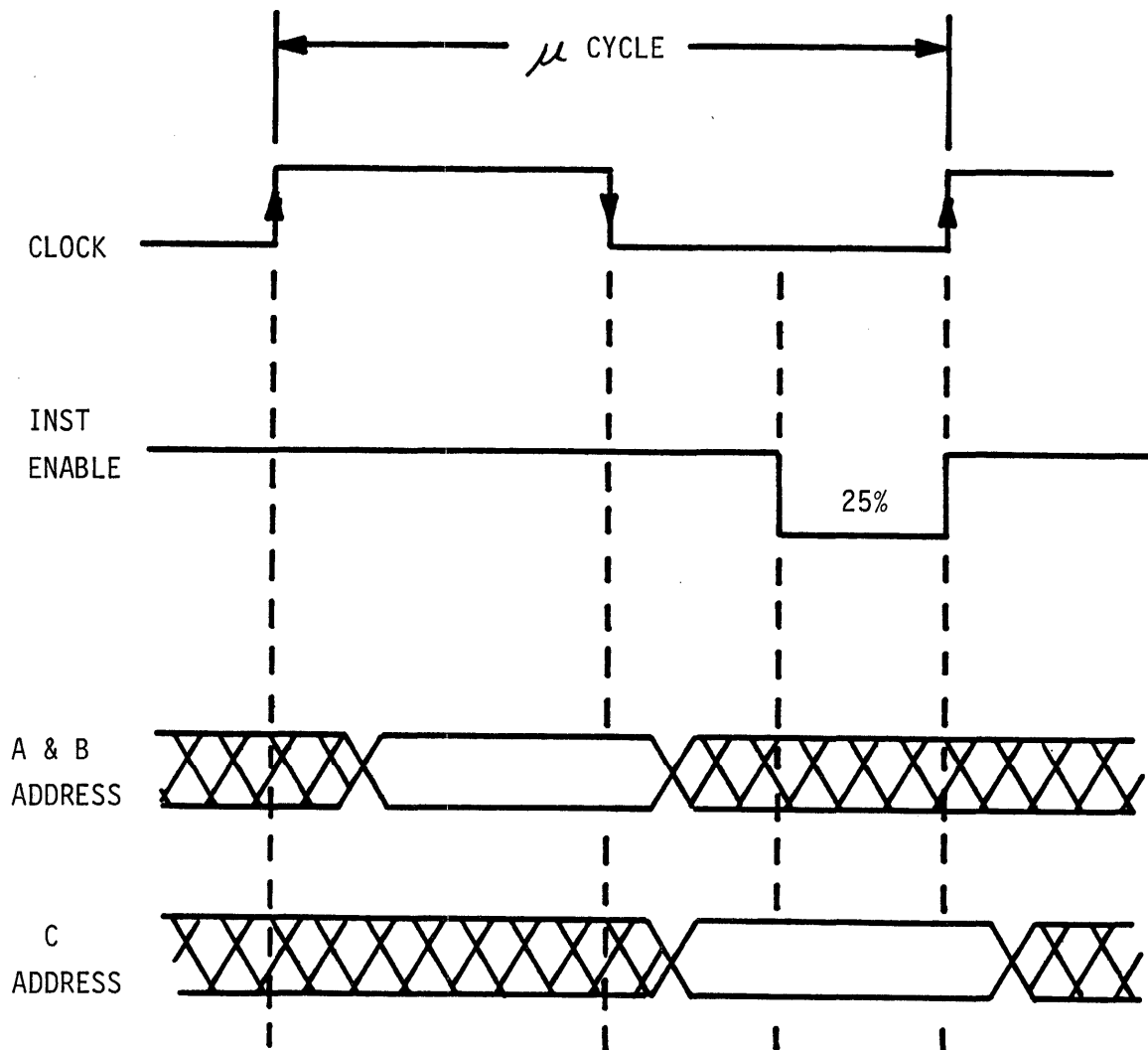
SOURCES:

- Register selected by A-address and B-address

DESTINATION:

- Register selected by C-address through use of RAM B-address MUX which is controlled by the clock



Am2903 Three-Address Operation

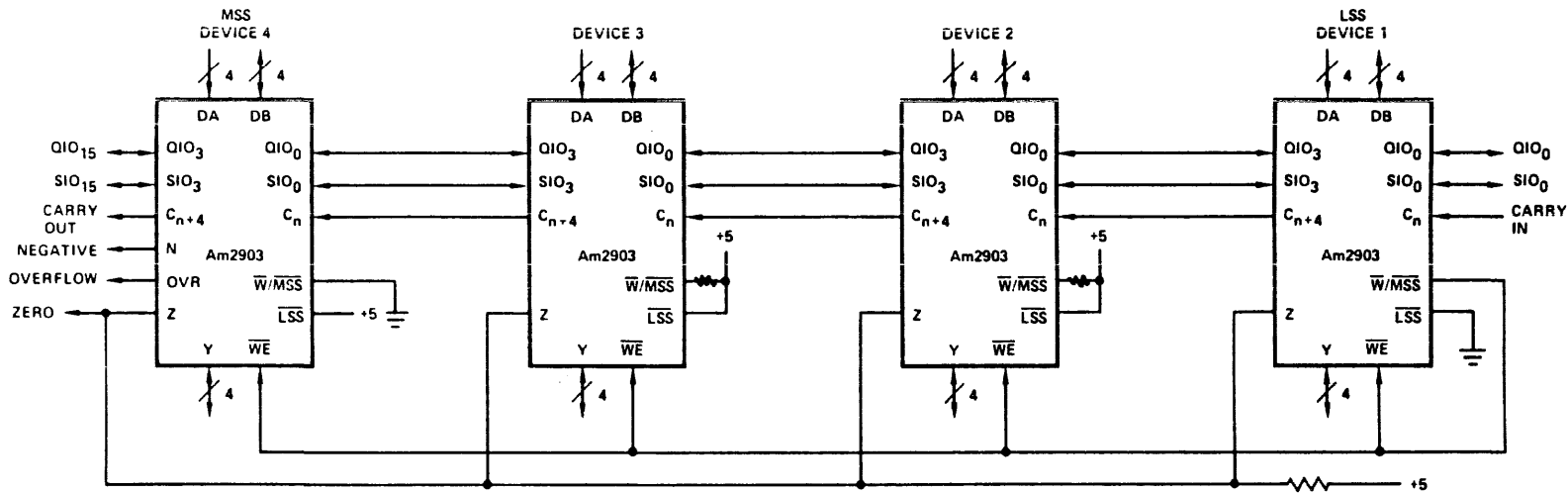
2-880

ED2900A

2-880

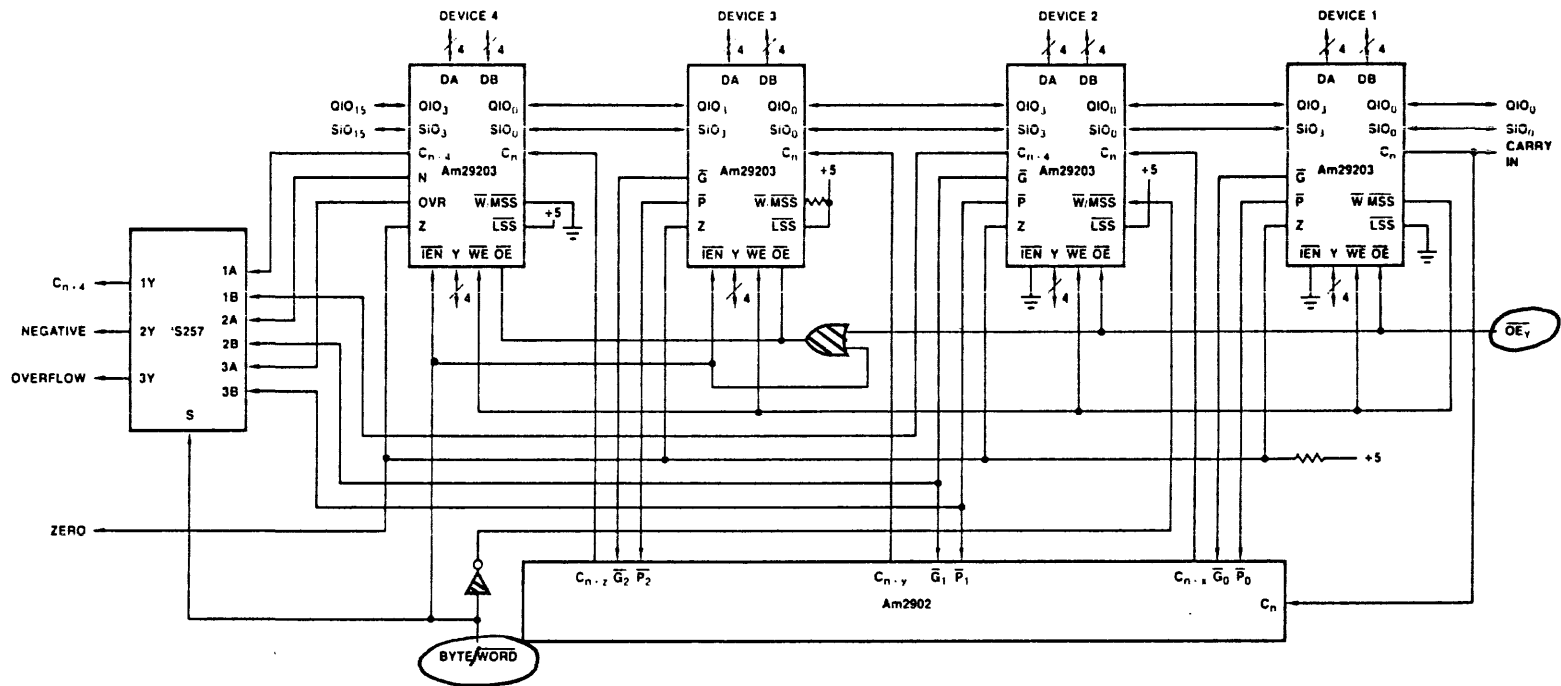
INTERCONNECTING THE SLICES

Am2903/Am29203



16-Bit CPU with Ripple Carry.

NOTE ISOLATING RESISTORS



Connections for Word/Byte Operations (Am29203 Only).

EXAMPLE**Am2903/29203 MICROCODE**

- Increment a register and output original value
- Byte swap

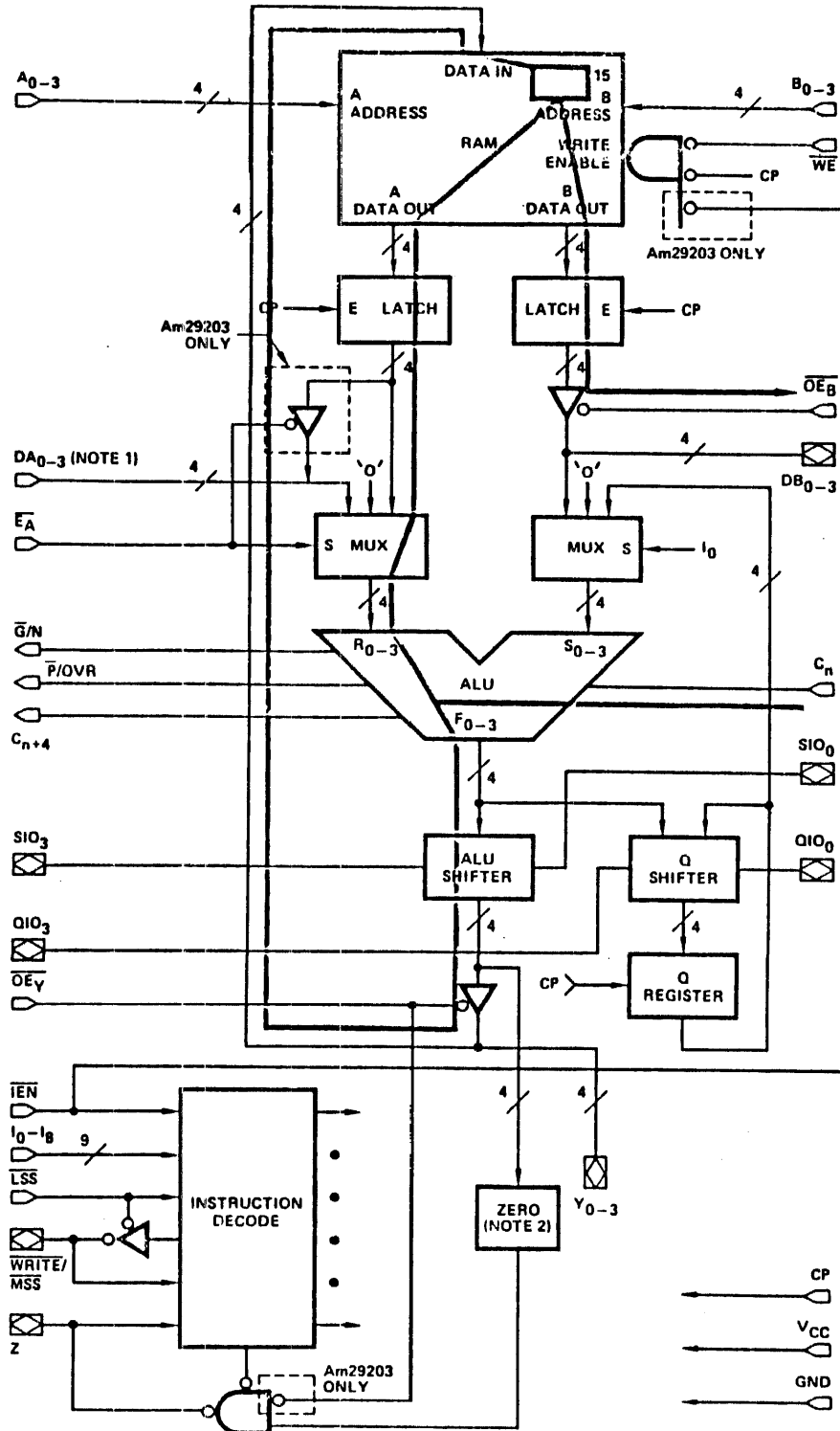
Am2903**INCREMENT A REGISTER****AND****OUTPUT ITS ORIGINAL VALUE**

- This operation can be used in the macroinstruction fetch cycle for the PC (macro program counter).
- Assume the register is loaded with the macroinstruction address in main memory.
- Assume that Reg 15 is the PC.

Am2903 Solution

- Address A and Address B are both set 10 15
- \bar{E}_A I_0 \bar{OE}_B (Source) are set to 000 (RAMAB)
to select as source operands RAM output A,B
- I_{4321} (Function) is set to 6 (increment) (INCRR)
- C_{IN} is set to 1
- $\bar{OE}_B = 0$ places the original value of R_{15} at $DB_{I/O}$
- I_{8765} (Destination is set to F and \bar{OE}_Y to Low
to select F-->Y and to write the new value F
into the RAM. (RAM or RAMEXT)

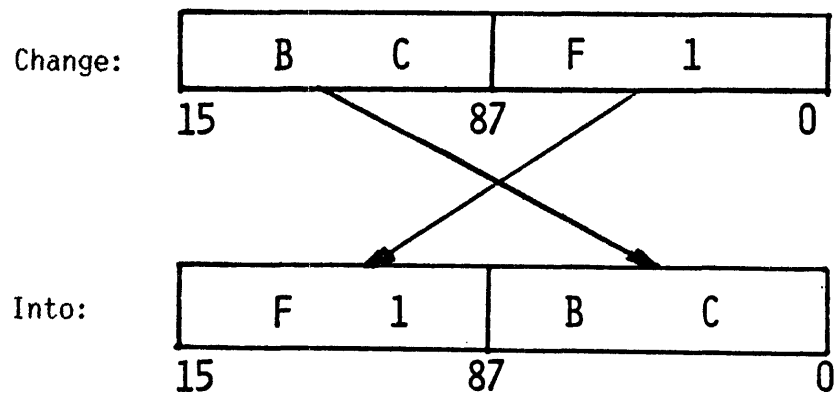
I_{8765}	I_{4321}	$\bar{E}_A I_0 \bar{OE}_B$	C_{IN}	\bar{OE}_Y
F	6	0	H	L
RAM	INCRR	RAMAB		



- Notes: 1. DA_{0-3} is input only on Am2903, but is I/O port on Am29203.
 2. On Am29203, zero logic is connected to Y, after the OE_V buffer.

Am2903**BYTE SWAP**

To exchange two halves of a 16-bit word



- Assume again Register 15.
(Number chosen for no particular reason)

Am2903

- Address A and Address B are both set to 15
- $\overline{E}_A \overline{I}_0 \overline{OE}_B$ set to 0 for A and B ports as operands
- $C_{IN} = C_{OUT}$
- I_{4321} is set to 3 (Add)
- I_{8765} is set to 9 for 2F-->Y (Shift) and write to RAM
- \overline{WE} Low
- \overline{OE}_Y Low

Repeat for total of four times, same as for Am2901

SOURCE	FUNC	DEST	C_{in}	2904

RAMAB	ADD	RAMUPL	C_{out}	SIO_{15} to SIO_0

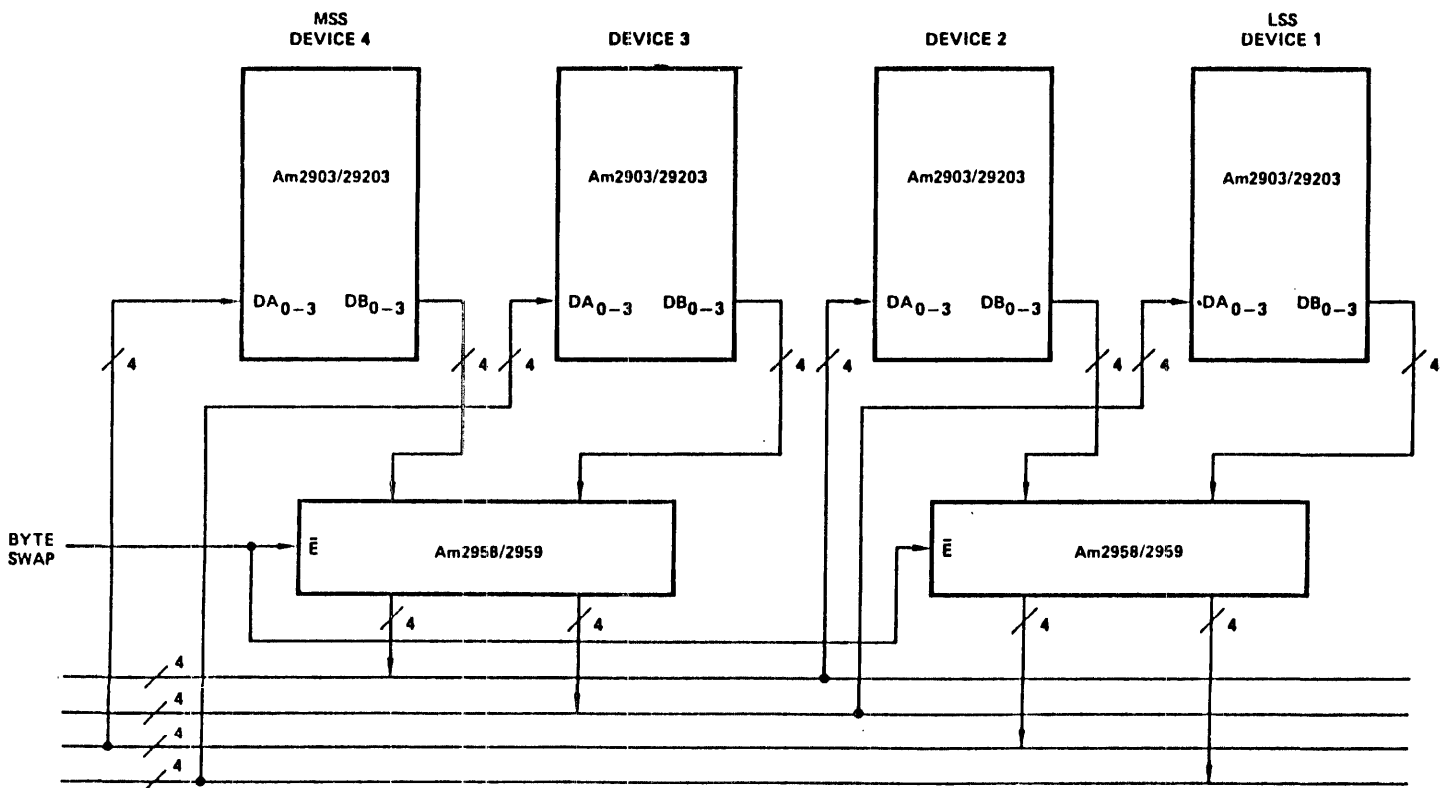
BYTE SWAP - HARDWARE ASSIST

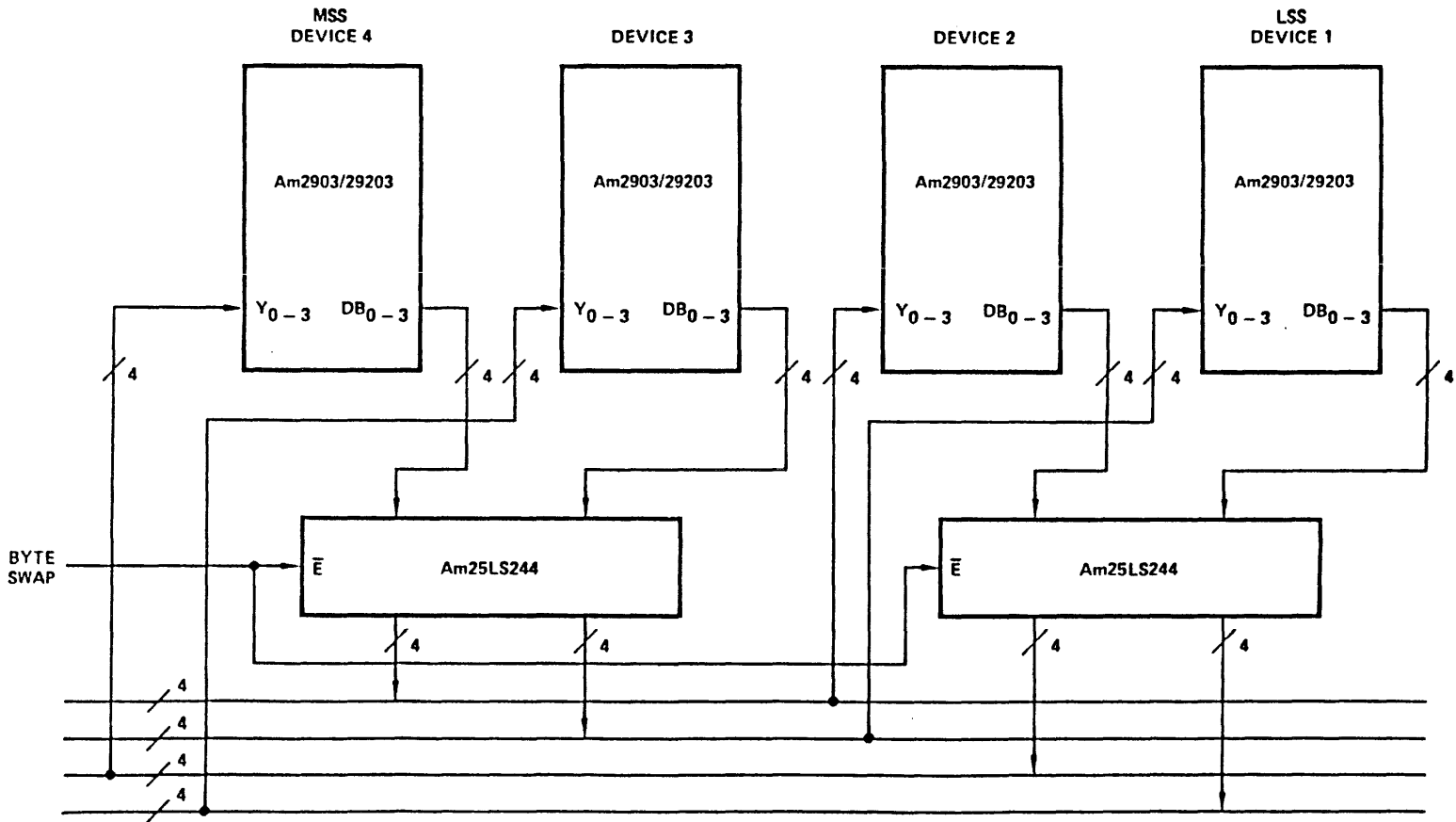
- There are several ways to handle a one-microcycle, hardware assist byte swap with the Am2903/Am29203.

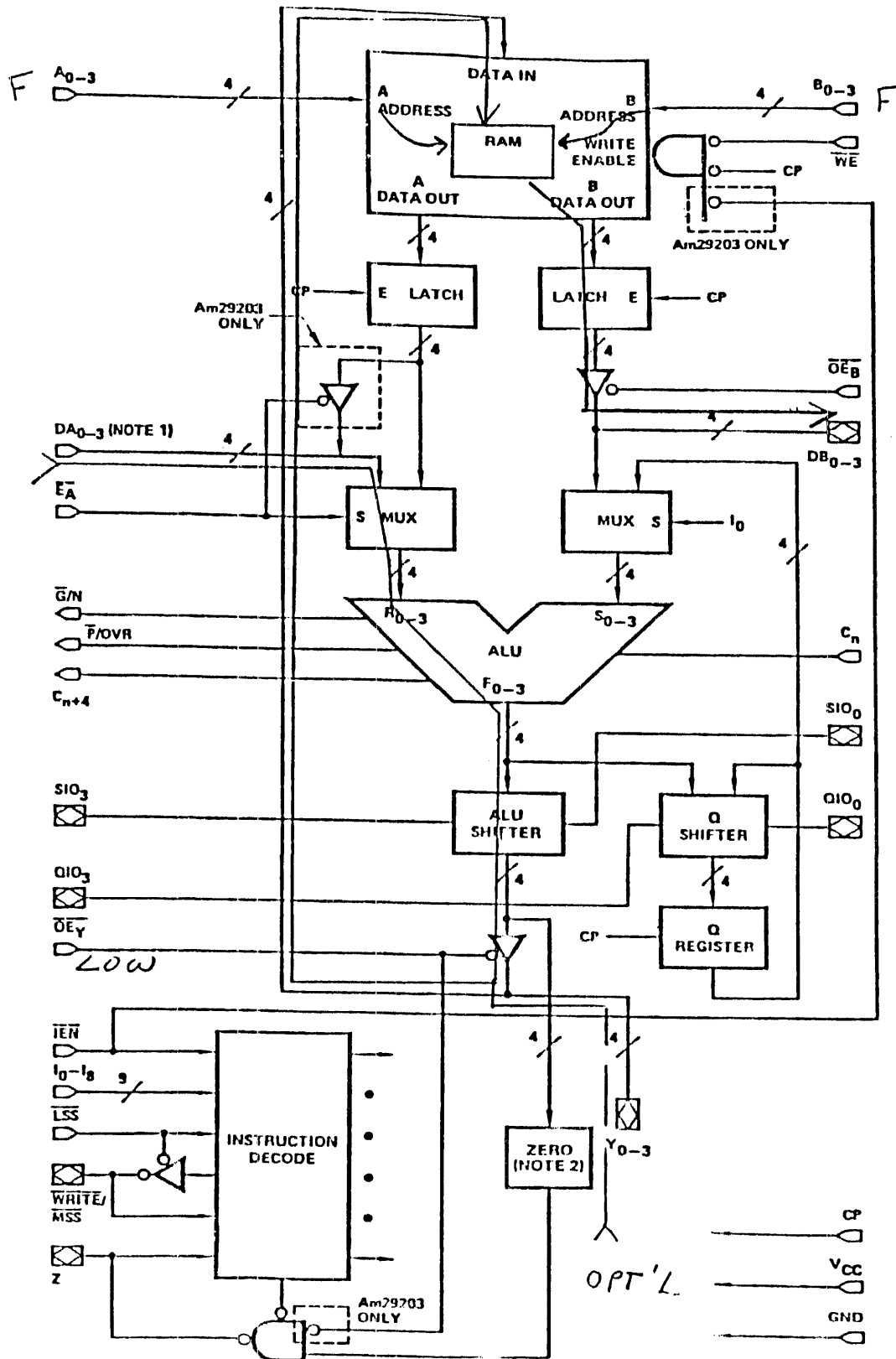
Basically,

- Bring the data (already in a register) out RAMB to DB
- Pass it through buffers (inverting or noninverting)
- Bring it back in either DA or Y
- The interconnections permute the data
- DA passes data through the ALU
 - medium speed version
 - use Am2958/59 (invert/true)
 - octal buffer/driver/receiver
 - tri-state output
- Y passes data directly to the RAM registers
 - high speed version
 - use Am25LS244

SWAP







Byte Swap requires an enable for the tri-state buffer drivers

The code is:

addr	2910 INST	COND MUX	BRCH CNTR	SRCE	FUNC	DEST	RA ADDR	RB ADDR	Cin	\overline{OE}_y	\overline{Ien}	\overline{E}
n	CONT	#	#	DARAMB	INCR	RAM	#	Rb	LOW	EN	EN	EN
OR												
n	CONT	#	#	RAMAB	#	RAM	#	Rb	#	DIS	EN	EN

2-1040

ED2900A

2-1040

THE SPECIAL FUNCTIONS
OF
THE Am2903/Am29203

Am2903/Am29203 SPECIAL FUNCTIONS AND FEATURES

- Parity
- Sign extension
- Sign magnitude <--> two's complement conversion
- Increment by 1 or 2
- Unsigned multiply
- Two's complement multiply
- Two's complement divide
- Single length normalize
- Double length normalize

SPECIAL FUNCTIONS: $I_0 = I_1 = I_2 = I_3 = I_4 = \text{LOW}$, $\overline{IEN} = \text{LOW}$

I_8	I_7	I_6	I_5	Hex Code	Available On	Special Function	ALU Function	ALU Shifter Function	SIO ₃		SIO ₀	Q Reg & Shifter Function	QIO ₃	QIO ₀	$\overline{\text{WRITE}}$
									Most Sig. Slice	Other Slices					
L	L	L	L	0	Am2903 Am29203	Unsigned Multiply	$F = S + C_n$ if Z=L $F = R + S + C_n$ if Z=H	Log. F/2→Y (Note 1)	Hi-Z	Input	F ₀	Log. Q/2→Q	Input	Q ₀	L
L	L	L	H	1	Am29203										
L	L	H	L	2	Am2903 Am29203	Two's Complement Multiply	$F = S + C_n$ if Z=L $F = R + S + C_n$ if Z=H	Log. F/2→Y (Note 2)	Hi-Z	Input	F ₀	Log. Q/2→Q	Input	Q ₀	L
L	L	H	H	3	Am29203										
L	H	L	L	4	Am2903 Am29203	Increment by One or Two	$F = S + 1 + C_n$	F→Y	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	L	H	5	Am2903 Am29203	Sign/Magnitude- Two's Complement	$F = S + C_n$ if Z=L $F = \overline{S} + C_n$ if Z=H	F→Y (Note 3)	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	H	L	6	Am2903 Am29203	Two's Complement Multiply, Last Cycle	$F = S + C_n$ if Z=L $F = S - R - 1 + C_n$ if Z=H	Log. F/2→Y (Note 2)	Hi-Z	Input	F ₀	Log. Q/2→Q	Input	Q ₀	L
L	H	H	H	7	Am29203										
H	L	L	L	8	Am2903 Am29203	Single Length Normalize	$F = S + C_n$	F→Y	F ₃	F ₃	Hi-Z	Log. 2Q→Q	Q ₃	Input	L
H	L	L	H	9	Am29203										
H	L	H	L	A	Am2903 Am29203	Double Length Normalize and First Divide Op.	$F = S + C_n$	Log. 2F→Y	R ₃ ∨ F ₃	F ₃	Input	Log. 2Q→Q	Q ₃	Input	L
H	L	H	H	B	Am29203										
H	H	L	L	C	Am2903 Am29203	Two's Complement Divide	$F = S + R + C_n$ if Z=L $F = S - R - 1 + C_n$ if Z=H	Log. 2F→Y	R ₃ ∨ F ₃	F ₃	Input	Log. 2Q→Q	Q ₃	Input	L
H	H	L	H	D	Am29203										
H	H	H	L	E	Am2903 Am29203	Two's Complement Divide, Correction and Remainder	$F = S + R + C_n$ if Z=L $F = S - R - 1 + C_n$ if Z=H	F→Y	F ₃	F ₃	Hi-Z	Log. 2Q→Q	Q ₃	Input	L
H	H	H	H	F	Am29203										

- NOTES: 1. At the most significant slice only, the C_{n+4} signal is internally gated to the Y_3 output.
 2. At the most significant slice only, $F_3 \vee \text{OVR}$ is internally gated to the Y_3 output.
 3. At the most significant slice only, $S_3 \vee F_3$ is generated at the Y_3 output.

L = LOW
 H = HIGH
 X = Don't Care
 Hi-Z = High Impedance
 \vee = Exclusive OR
 Parity = $\text{SIO}_3 \vee F_3 \vee F_2 \vee F_1 \vee F_0$

PARITY

- Parity is computed and available at SIO_0 when the destination field I_{8-5} is either {4, 5, 6, 7}
- This corresponds to:
 - 4 RAM F → Y, F → RAM, No Q activity
 - 5 QD F → Y, Z/2 → Q, no write to RAM
 - 6 LOADQ F → Y, F → Q, no write to RAM
 - 7 RAMQ F → Y, F → Q, Y → RAM
- The computed equation is:

$$SIO_0 = F_0 \vee F_1 \vee F_2 \vee F_3 \vee F_4 \vee \dots \vee F_n \vee SIO_n$$

SIGN EXTEND

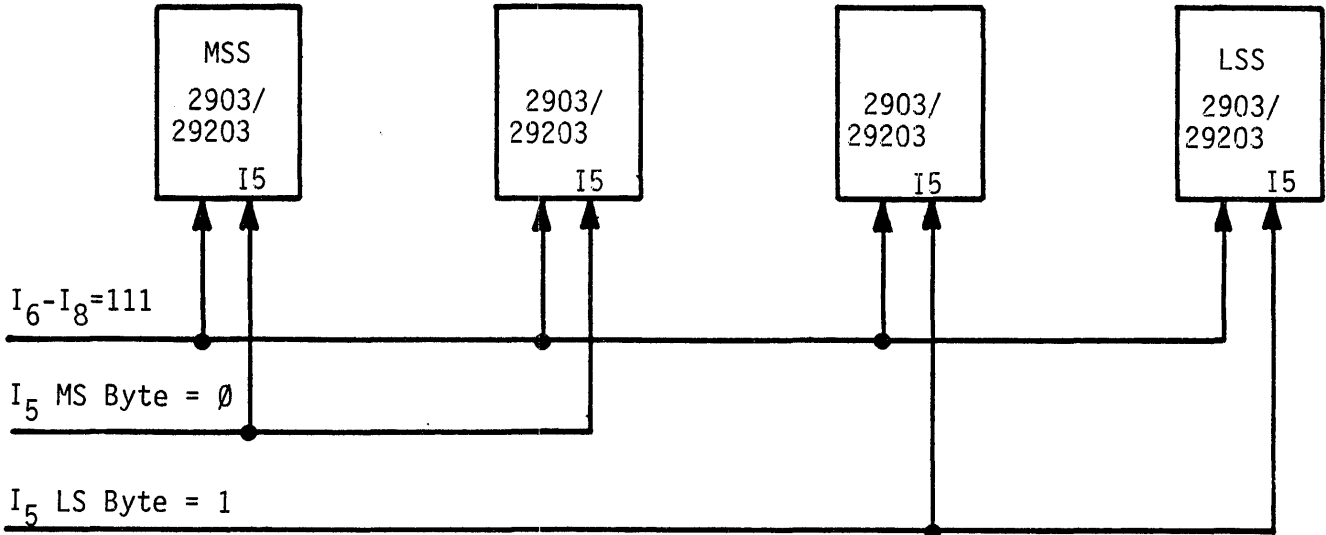
- By varying the destination control field into different ALU slices, specifically by varying I5, sign extension can be done across any number of devices in one microcycle.
- This corresponds to:

I8-I6	I5	HEX	MNEMONIC	DEVICE ACTIONS
111	0	E	SGNEXT	SI00 -> Y, Y -> RAM, SI00 -> SI03
111	1	F	RAMEXT	F -> Y, Y -> RAM, F3 -> SI03

- Thus by controlling I5 separately to each slice when I8-I6 = 111:

If I5 = 1, slice behaves "normally"

If I5 = 0, whatever is input on SI00 will appear on all Yi and SI03



- Add a second 15-bit position to the microword
- Change .DEF file to a 5-bit destination field
- Example:

2910	A	L	U					
INST	SRCE	FUNCT	DEST	RA	RB	Cin	OEY	\overline{IEN}
CONT	RAMAQ	INCRRR	SIGNEXT	Ra	#	LOW	EN	EN

Where in the .DEF file

```
SIGNEXT EQU B#11101
```

For all other destination codes, the last two bits are identical.

BINARY NUMBER REPRESENTATION

- There are three ways to represent binary numbers in a computer. They are:
 - sign plus magnitude
 - sign plus two's complement
 - sign plus one's complement

- The sign plus magnitude is the general way in which humans represent numbers in any base. The sign or sign bit is treated as a separate piece of information and must be manipulated with a different algorithm similar to the human operations for a base ten arithmetic operation (addition, subtraction, multiplication and division).

- Using sign plus two's complement or sign plus one's complement, the same binary arithmetic operations can be applied to the sign bit that are used for the other bits in the number. Thus, increasing operational speed and minimizing specialized hardware.

- Sign plus magnitude and sign plus two's complement representations are briefly introduced for use in ALU bit-slice manipulations.

NUMBER REPRESENTATIONS (CONT'D)

- Sign bit coding for all representations is:
 - sign bit is 0 if number is positive
 - sign bit is 1 if number is negative

- **Sign plus magnitude representation**
 - The value to the right of the sign bit is the absolute magnitude value of the number

Examples using an eight-bit register:

$$+6_{10} = 00000110_2$$

$$-6_{10} = 10000110_2$$

$$+13_{10} = 00001101_2$$

$$-11_{10} = 10001011_2$$

- The range of sign plus magnitude represented numbers is
 - largest positive number - 011...111
 - largest negative number - 111...111
 - zero (double representation) - 0000...000
1000...000

NUMBER REPRESENTATION (CONT'D)

- **Sign plus two's complement representation**

For positive numbers, this representation is identical to that for sign plus magnitude, i.e.

$$+7_{10} = 00000111_2$$

For negative numbers, the bits following the sign bit are the two's complement of the magnitude of the number. The two's complement of a number is found by reversing (toggle) the 1's and 0's of the magnitude (one's complement) and adding 1 in the LSB location. For example, using an eight-bit register:

-6_{10} :

magnitude	$6_{10} = 0000110_2$
one's complement	$= 1111001_2$
two's complement	$= 1111010_2$

$$-6_{10} = 1111010_2$$

The range of (sign plus) two's complement represented numbers is:

- largest positive number - 0111...111
- largest negative number - 1000...000
- zero (single) - 0000...000

- All AMD ALU's currently use sign plus two's complement notation as their primary representation.

SIGN MAGNITUDE TO/FROM TWO'S COMPLEMENT CONVERSION

- Word to be converted placed on S-Port of ALU
 - RAM B
 - DB input
- Carry-in = Z by connecting Z pin to carry-in
- Z is sign bit - indicates positive or negative number
- Overflow if number is largest negative number
- $F = \langle B \rangle + C_{in}$ if $Z = 0$
- $F = \langle B \rangle + C_{in}$ if $Z = 1$
- $Y_3^{MSS} = (S_3 \vee F_3)^{MSS}$

EXAMPLES

1. B = 10001010 sign magnitude for -10

Z = 1 therefore

B = 01110101 except F3 ∇ S3 \rightarrow Y₃ of MSS

11110101 $\emptyset \nabla 1 \rightarrow 1$

+C_n = 1 add 1 since Z = 1

11110110

check:

$$-128 + 64 + 32 + 16 + 0 + 4 + 2 + 0 = 10$$

2. 11110110 two's complement for -10

Z = 1 therefore

B = 00001001 except for Y3 of MSS

10001001

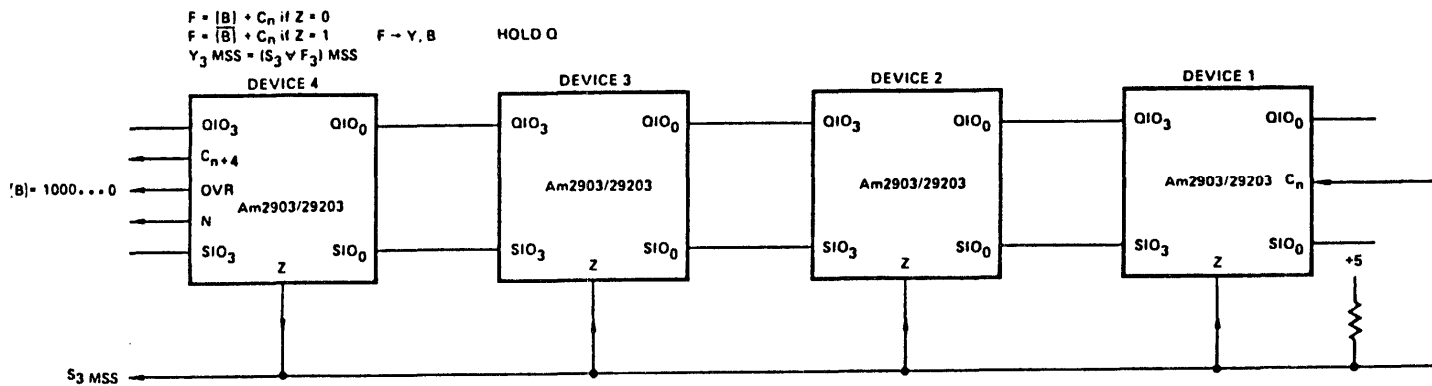
+C_n = 1 add 1

10001010 Voila!

EXAMPLE MICROCODE

2910	SOURCE	FUNCT	DEST	RA	RB	Cin
CONT	RAMAB	SPECL	SGNTWO	#	RØ	Z

TWO'S COMPLEMENT



INCREMENT BY 1 OR 2

- Although it is possible to increment by 1 without going to a special function, it is not possible to increment by 2.
- In a byte-addressable memory both byte addressing and word addressing capability may be desirable.

byte addressing: $\langle R0 \rangle \leftarrow \langle R0 \rangle + 1$

word addressing: $\langle PC \rangle \leftarrow \langle PC \rangle + 2$

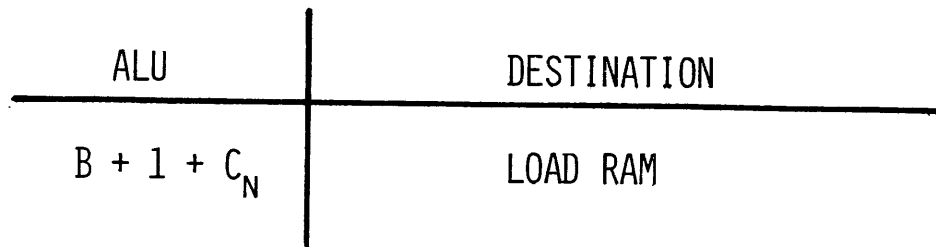
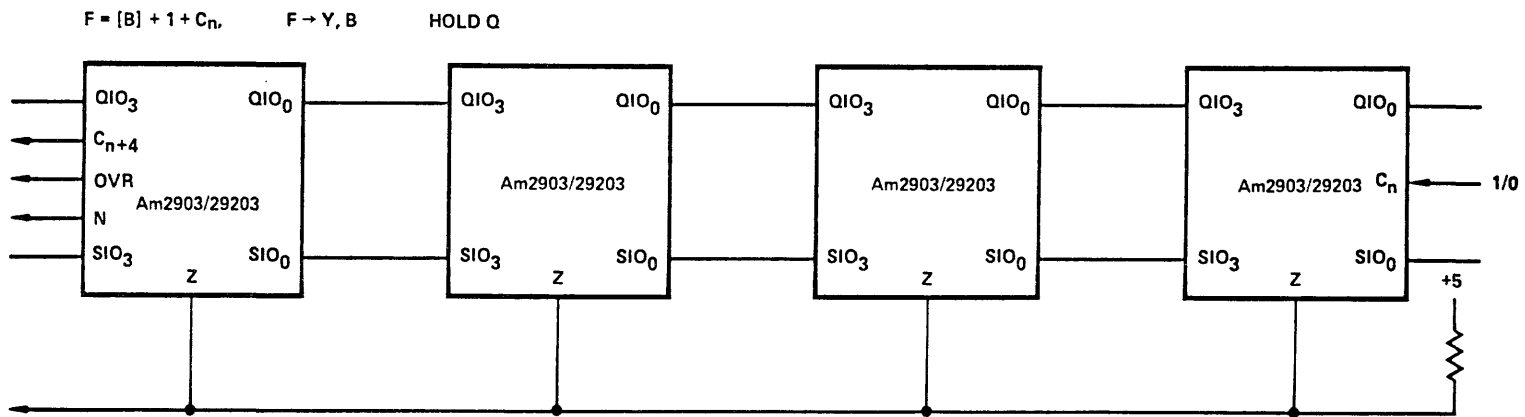
- The special function "INCRMNT" provides this capability.

4 INCRMNT $F = S + 1 - C_{in}$

2910	SOURCE	FUNCT	DEST	RA	RB	C _{in}

CONT	RAMAB	SPECL	INCRMNT	#	R1	one

INCR 1 or 2



2-1200

ED2900A

2-1200

MULTIPLICATION

MULTIPLICATION

B X A

UNSIGNED BINARY

77
X 11
847

```

1001101
*0001011
-----
1001101 ← IN FIRST POSITION
1001101 ← IN SECOND POSITION
0000000
1001101 ← IN FOURTH POSITION
0000000
0000000
0000000
-----
0001101001111

```

WEIGHT →

```

0001101001111
512 1 | | | |
256  | | | |
64  8 4 2 1
-----
512
256
64
8
4
2
1
-----
847

```


RULES (MANUAL OPERATION):

- For each 1 in multiplier B , add the multiplicand A shifted to align its LSB with the 1 of B .
- For each 0 in multiplier B , add zeros, also aligned.
- The result of an $N \times N$ multiply is $2N$ bits long.

MULTIPLICATION**TWO'S COMPLEMENT - "METHOD OF FLORES"**

Case #1

B X A B positive A positive

- The same algorithm as unsigned multiply

TWO'S COMPLEMENT - METHOD OF FLORES (Cont'd)

Case #2

B X A B positive A negative

```

S
1.10011
*0.01011
-----
1.111110011
1.1111100110
0.0000000000
1.1110011000
0.0000000000
1.1101110001

```

ignore C_{out}

- Rules

- expand A to 2N in length:

A = 1.1111110011

- proceed as for unsigned multiply

- use sign bit of A as MSB for first partial product

(MULTIPLICATION (CONT'D))

<u>B</u> X <u>A</u>	<u>B</u> NEGATIVE	<u>A</u> POSITIVE	
		0.01101	
		1.10101	

		0.0000001101	
		0.0000000000	
		0.0000110100	
		0.0000000000	
		0.0011010000	

		0.0100010001	
correction		1.10011	

		1.1101110001	result

- Rules:

- Multiply directly as for unsigned.

- Form correction at end by adding two's complement of A to the result.

MULTIPLICATION (CONT'D)

<u>B</u> X <u>A</u>	<u>B</u> NEGATIVE	<u>A</u> NEGATIVE	
		1.10011	
		1.10101	

		1.11111 <u>10011</u>	
		0.0000000000	
		1.1111 <u>001100</u>	
		0.0000000000	
		1.1 <u>100110000</u>	

		101.1011101111	
correction	0.01101		

	0.0010001111		result

● Rules:

- Expand A to 2N in length.
- Proceed as for B negative and A positive.

**EXAMINATION OF THE FOUR CASES SHOWS THE COMMON PATTERN FOR THE
MULTIPLY ALGORITHM:**

- Expand A, left bits depend upon sign bit of A.
- Multiply by adding A or \emptyset to running sum (partial product) depending upon LSB in B.
- "Correct" result using two's complement of A depending upon sign bit of B.

Therefore:

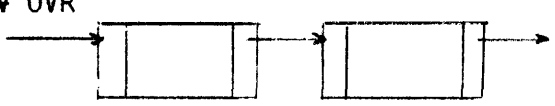
- The bits of B must be accessible.
- A method of conditional insert for left-most bits is needed.
- A source for conditional operand (A or \emptyset) is needed.

Am2901 MULTIPLY ALGORITHM

- Clear R_B
- Load multiplicand into R_A
- Load multiplier into Q
- LSB shifted out of Q

Q_{OUT} selects $R_B \leftarrow R_B + 0$
 or $R_{KB} \leftarrow R_B + R_A$

- Shift R_B, Q down

$F_3 \nabla OVR$

- LSB of R_B input to MSB of Q
- $F_3 \nabla OVR$ into MSB of R_B
 to generate the proper conditional insert for the left-most bits

Am2901 MULTIPLY ALGORITHM (CONT'D)

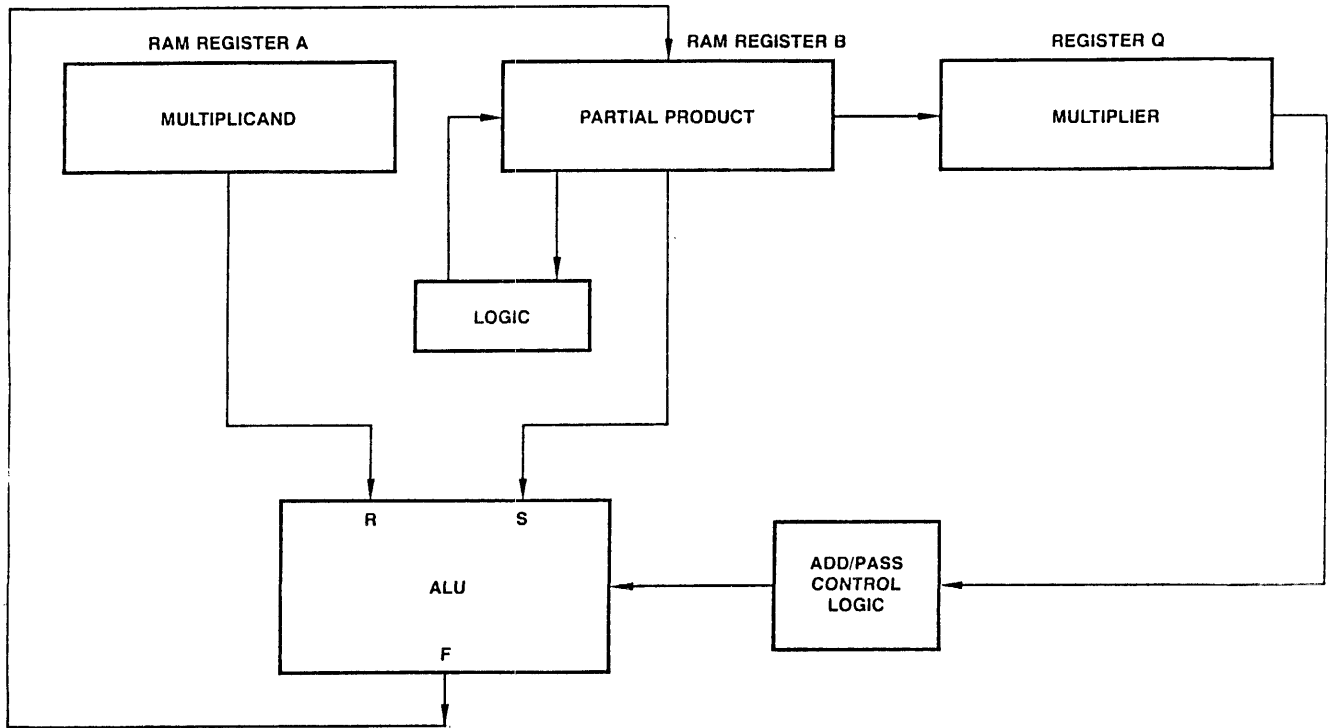
- LAST Q_{out} is sign of multiplier

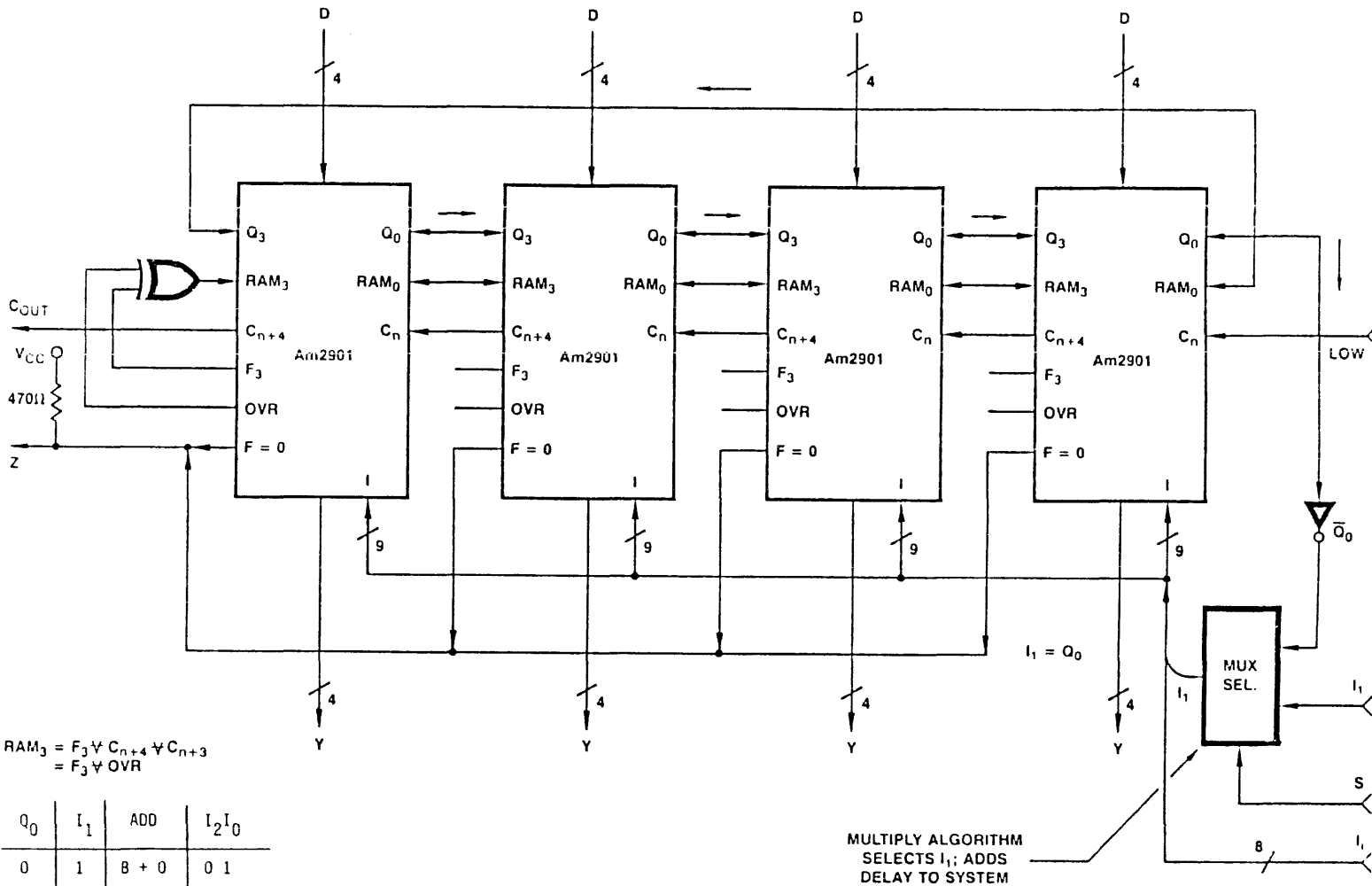
Q_{out} selects $R_B \leftarrow R_B$

or $R_B \leftarrow R_B - R_A$

- Shift R_B down
- The result is in R_B and Q

Am2901
Multiply Functional Diagram





$$RAM_3 = F_3 \vee C_{n+4} \vee C_{n+3}$$

$$= F_3 \vee OVR$$

Q_0	I_1	ADD	$I_2 I_0$
0	1	$B + 0$	0 1
1	0	$B + A$	0 1

SPECIFIC INTERCONNECTIONS FOR
16 BIT MULTIPLY - RIPPLE CARRY

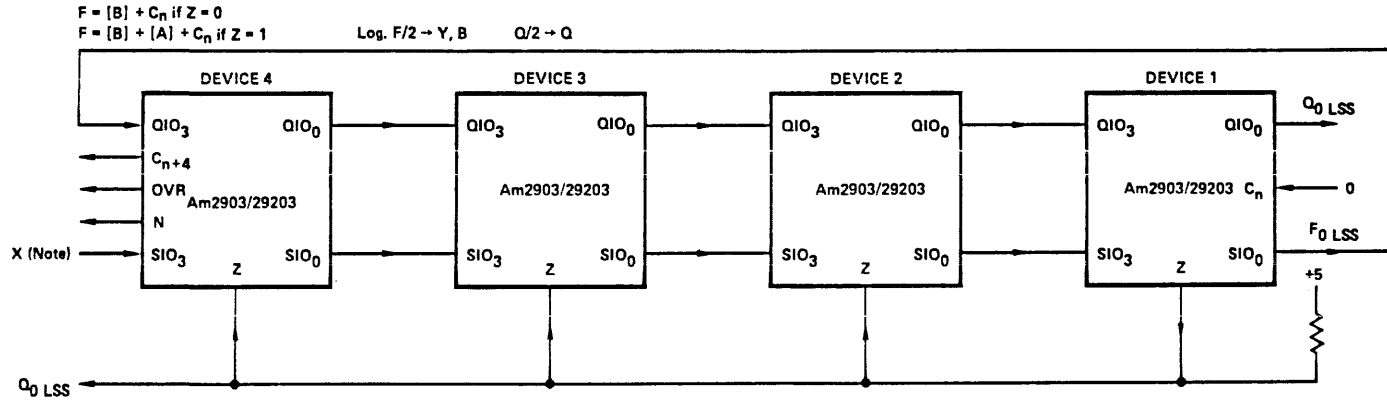
Am2903/Am29203 MULTIPLY ALGORITHM

- The Am2903/Am29203 improves this operation by providing internally what had to be added externally to the Am2901.

UNSIGNED MULTIPLY:

- Initially $R_0 = 0$ (B ADR, and R_b)
- Multiplicand in R_1 (A ADR, any R_a)
- Multiplier in R_2 (any R_i)
- Transfer R_2 -----> Q
- Execute unsigned multiply 16 times (counter = 15)
- 17 microcycles as shown (assuming all registers except Q initialized and result left in R_b and Q)

Am29203/2903 UNSIGNED MULTIPLY (16x16)

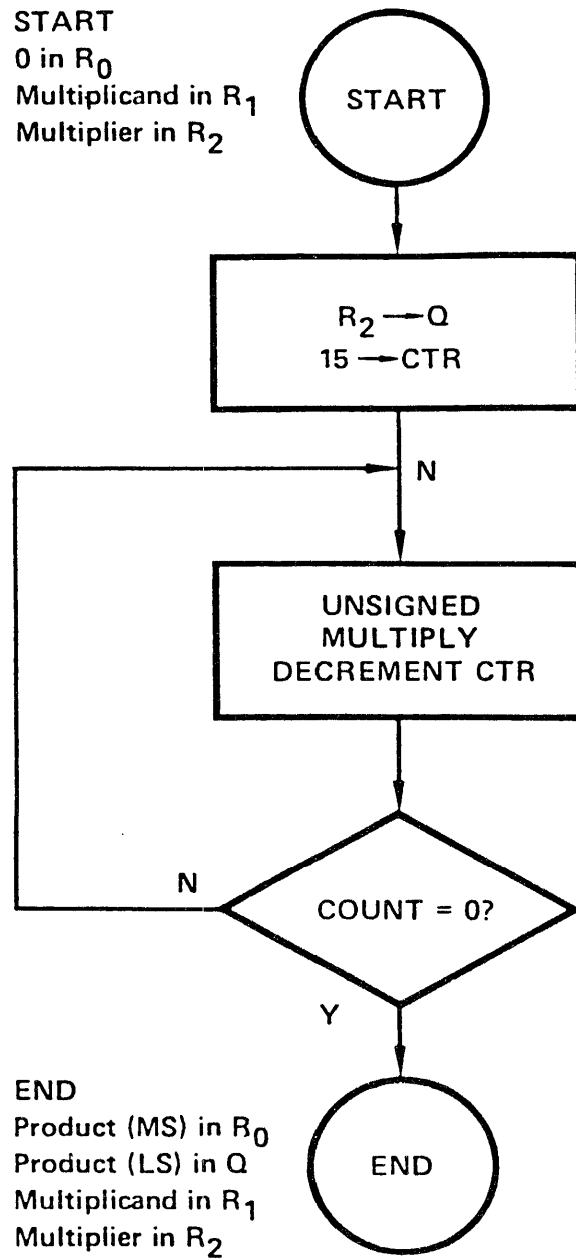


Note: For unsigned multiply, C_{n+4} MSS is internally shifted into position Y_3 MSS; 2's complement multiply $N \neq OVR$ is internally shifted into position Y_3 MSS.

Q_0 LSS internally routed to Z lines

Z	ALU	DESTINATION	S_3 in MSS
L	PASS	F/2 → RAM; Q/2 → Q	INTERNAL OPERATION
H	A + B		C_{N+4}

MULTIPLY



Am2903/Am29203**UNSIGNED MULTIPLY**

- The flow chart for unsigned multiply is given on the facing page. The code (using Am2910 and Am2903/Am29203) would be:

addr	2910	COND	BRCH	SRCE	FUNC	DEST	RA	RB	Cin	\overline{OE}_y	Ien	ROTATE
	INST	MUX	CNTR	A	L	U	ADDR	ADDR				CONNECTIONS

n	LDCT	#	15	RAMAB	INCRR	LOADQ	R1	#	LOW	#	EN	#
N+1	RPCT	#	n+1	LOW	SPECL	MULT	Ra	Rb	LOW	EN	EN	SIO \emptyset TO QIO3

This equates to:

n Load Q register with multiplier
 Load 2910 counter with 15
 (One less than actual count)

n+1 Perform special function 15 times
 Result is in Rb and Q

Am2903/29203 TWO'S COMPLEMENT MULTIPLY ALGORITHM:

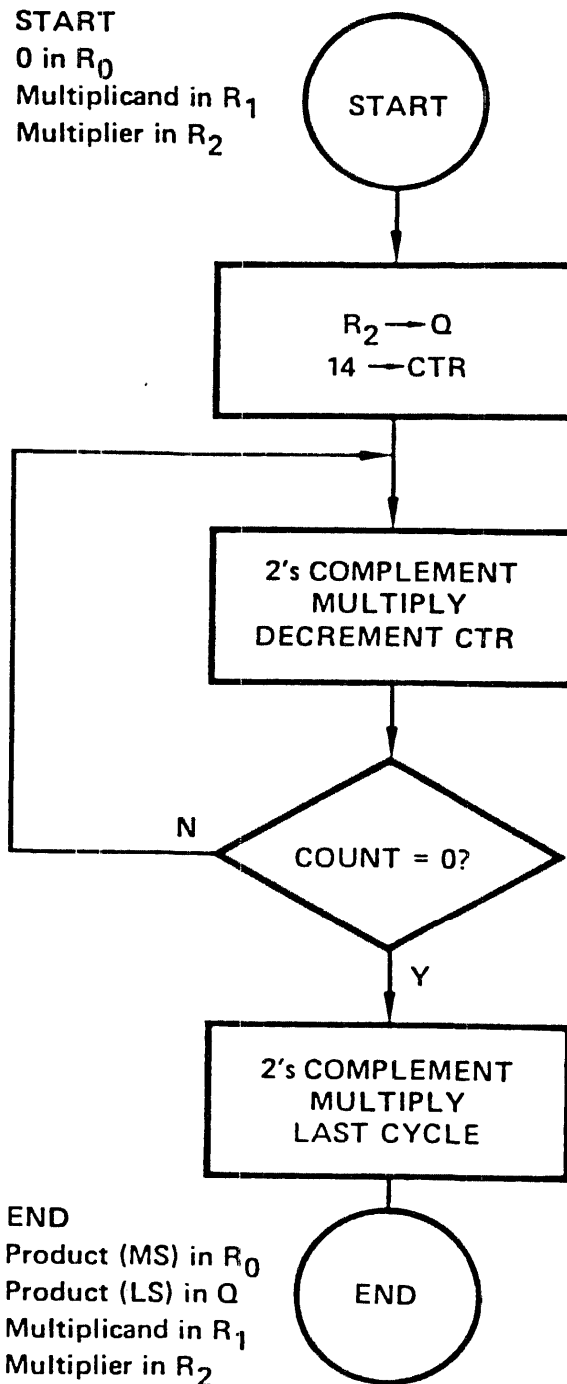
The flowchart for a 16-bit two's complement multiply is on the following page.

- The multiplier must be loaded into the Q register.
 - The flowchart shows register R2 -> Q
 - The code shows any register R1
 - Your code would match your application!

- The multiplicand must be in another register.
 - The flowchart shows register R1
 - The code shows any register Ra

- The result ends up in a RAM register (MSH) and the Q register (LSH).
 - The flowchart shows register R0
 - The code shows any register Rb

TWO'S COMPLEMENT MULTIPLY



TWO'S COMPLEMENT MULTIPLY (CONT'D)

- The interconnection for two's complement multiply is the same as for unsigned multiply, except for the last step (the "correction" step).
- For unsigned multiply C_{n+4} is internally shifted into Y3 of the MSS.
- For two's complement multiply $N \vee OVR$ is internally shifted into Y3 of the MSS.
- The code is:

addr	2910	COND	BRCH	SRCE	FUNC	DEST	RA	RB	Cin	\overline{OE}	\overline{E}	\overline{I}	ROTATE
	INST	MUX	CNTR	A	L	U	ADDR	ADDR					CONNECTIONS

n	LDCT	#	14	RAMAB	INCRR	LOADQ	Ri	#	LOW	#	EN	#	
n+1	RPCT	#	n+1	LOW	SPECL	TWOMULT	Ra	Rb	LOW	EN	EN	SI00 TO QI03	
n+1	CONT	#	#	LOW	SPECL	TWOLAST	Ra	Rb	Z	EN	EN	"	

TWO'S COMPLEMENT MULTIPLY (Cont'd)

- Algorithm

FUNCTION	C _N	Z	ALU	DESTINATION	S ₃ in MSS
MULTIPLY	L	L	PASS	DOWN SHIFTS	INTERNAL OPERATION
		H	A + B		
LAST STEP Z		L	PASS	F/2 → RAM; Z/2 → Q	SIGN + OVR
		H	B-A		

Am2903 CLASS EXERCISE

- Multiply R1 by R2
- Put result in R3 and R4
- Use two's complement multiply with Am2903

SOLUTION

2910 ADDR	CND INST	BRCH MUX	BRCH CNTR	A SRCE	L FUNCT	U DEST	RA ADDR	RB ADDR	CIN	\overline{OE}	\overline{EY}	\overline{TEN}	ROT
1	CONT	#	#	RAMAB	LOW	RAM	#	R3	LOW	EN	EN	#	
2	LDCT	#	14	RAMAQ	INCRR	LOADQ	R2	#	LOW	#	EN	#	
3	RPCT	#	3	LOW	SPECL	TWOMULT	R1	R3	LOW	EN	EN	SI010- Q103	
4	CONT	#	#	LOW	SPECL	TWOLAST	R1	R3	Z	EN	EN	SI00- Q103	
5	CONT	#	#	RAMAQ	INCRS	RAM	#	R4	LOW	EN	EN	#	

SINGLE LENGTH NORMALIZE

SINGLE LENGTH/DOUBLE LENGTH NORMALIZE

- Normalization is a technique for referencing a floating point number to a fixed radix (binary) point.
- Used in fixed to floating point conversion.
- Double length normalize requires an extra microcycle per loop or an external counter to shift exponent count.
- **Example:**

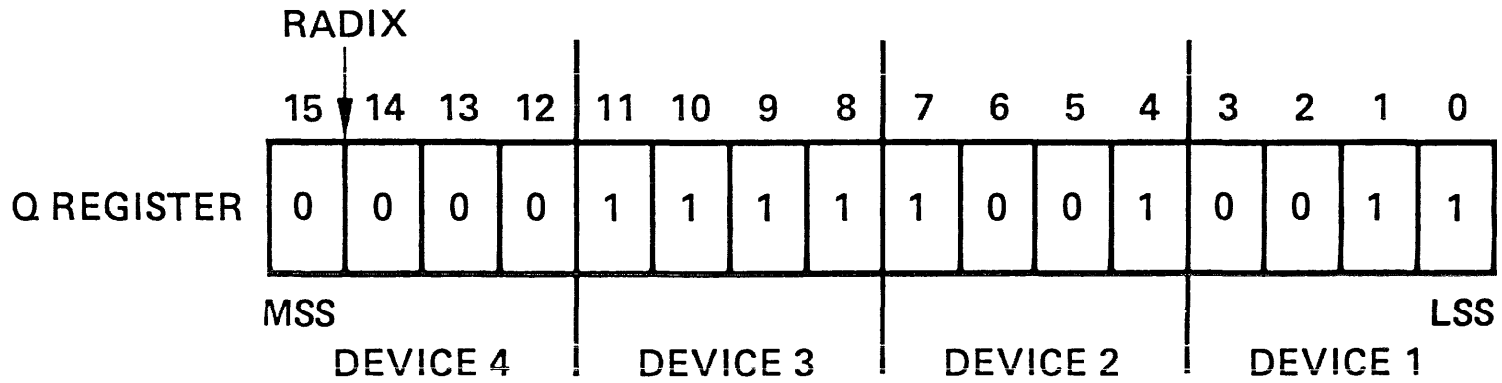
$$0.0031 \times 10^5 = 0.3100 \times 10^3 \quad (\text{base } 10)$$

$$0.0011 \times 2^7 = 0.1100 \times 2^5 \quad (\text{base } 2)$$

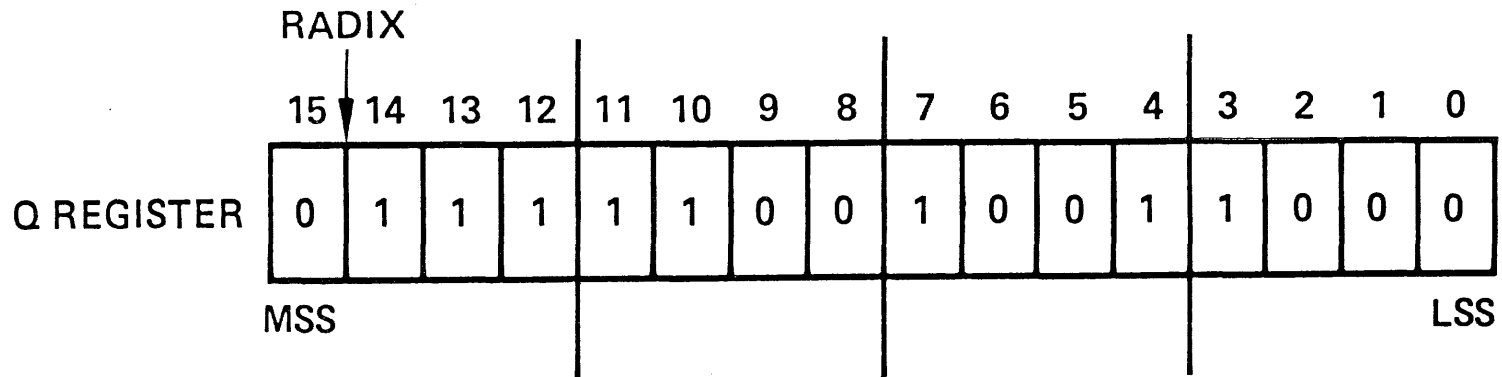
- For binary numbers, the normalization technique consists of shifting the mantissa left until the two bits immediately adjacent to the binary point are of opposite polarity.

NORMALIZE ALGORITHM

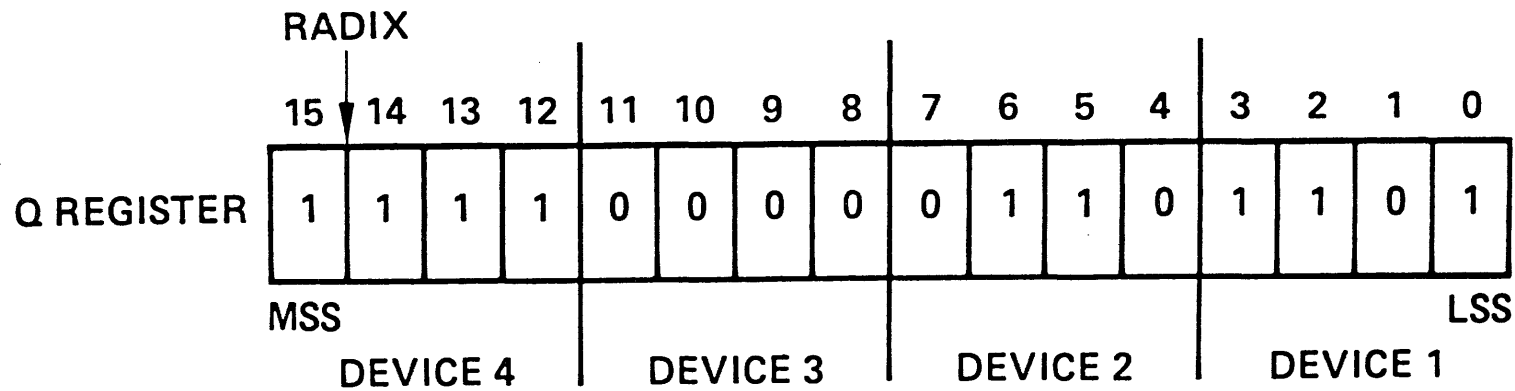
- Shift left (zero fill) until Q_n (MSB) $\neq Q_{n-1}$.
- Increment exponent register for each shift. This approach counts shifts. This value must be subtracted from the actual exponent value for the proper representation.



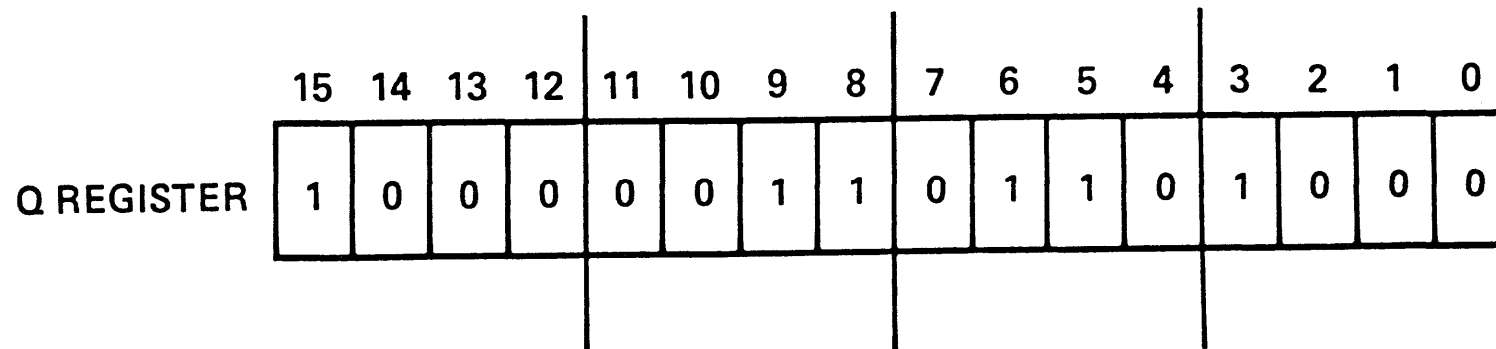
a) Unnormalized Positive Number.



b) Normalized Positive Number.



a) Unnormalized Negative Single Length Number.



b) Normalized Negative Single Length Number.

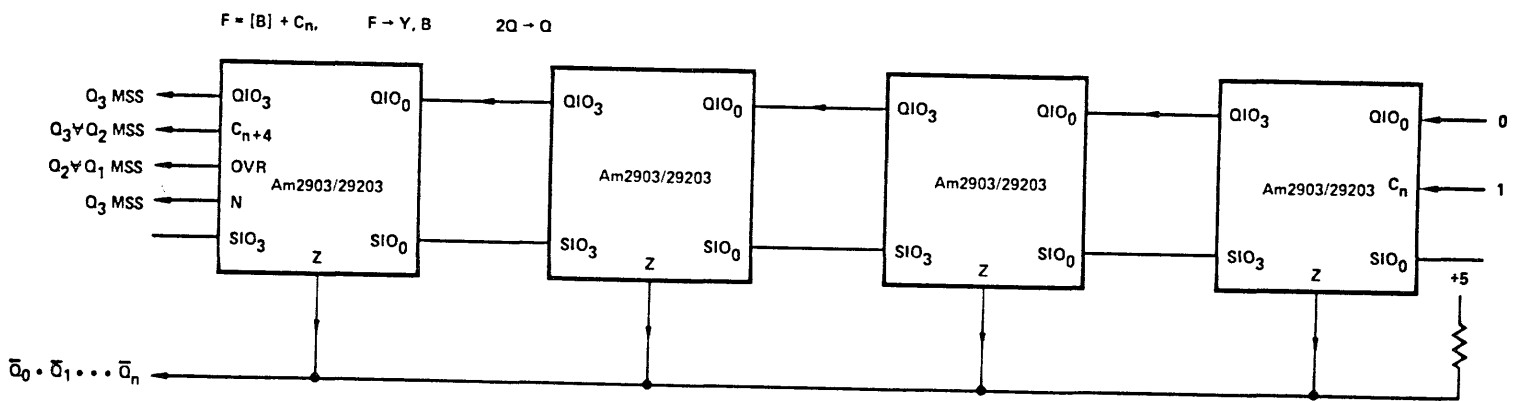
Am2903 SINGLE LENGTH NORMALIZE (SLN)

Set up:

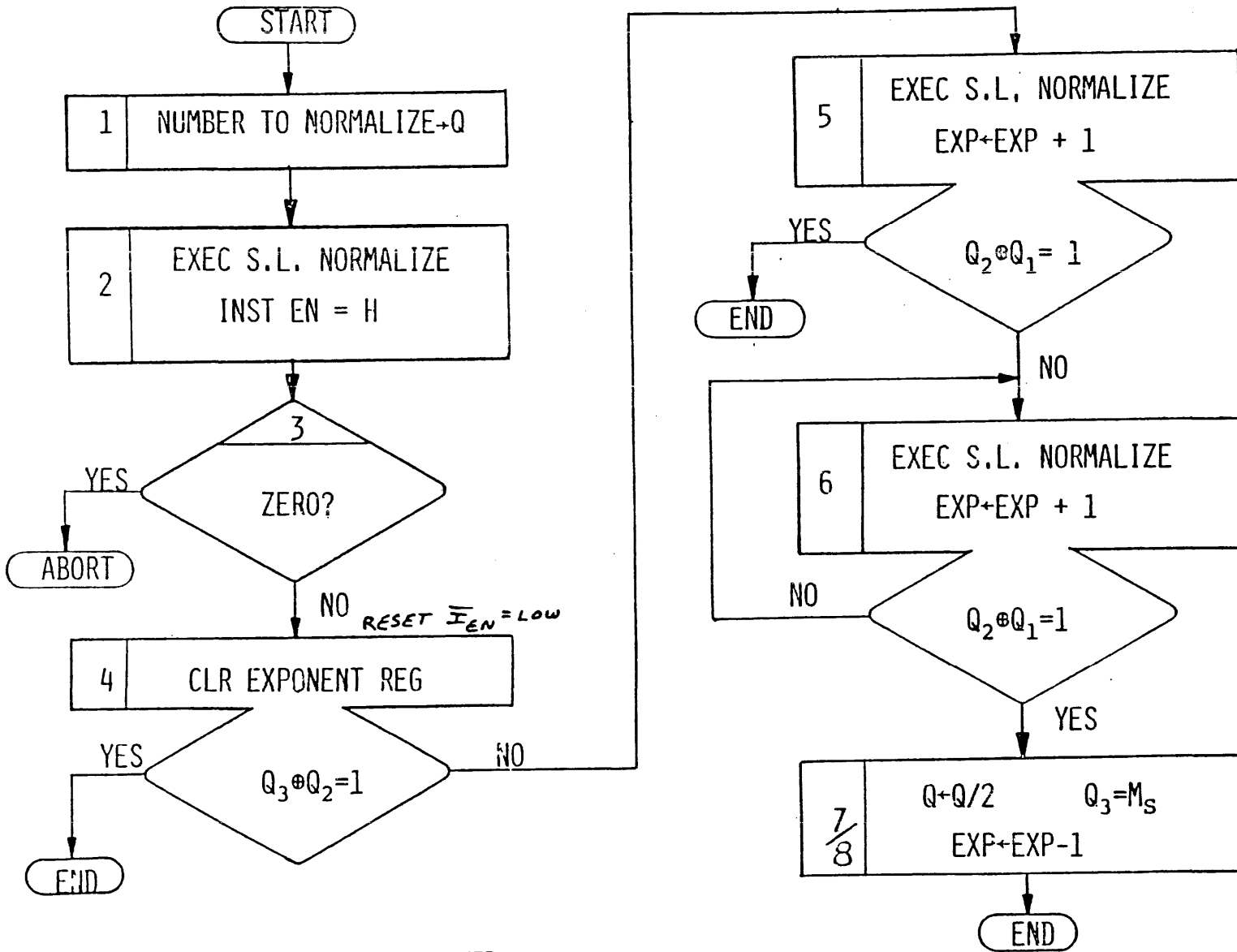
- Define exponent register.
- Put unnormalized number into Q.

Operation:

- When SLN is executed
 - Q is shifted left one bit
 - Zero is loaded into Q_0
 - $C_{n+4} = Q_3 \vee Q_2$ (MSS)
 - $OVR = Q_2 \vee Q_1$ (MSS)
- $Z = 1$ if $Q = 0$ (cannot normalize)
- $\langle B \rangle + C_N \rightarrow \langle B \rangle$ (use for exponent)



SINGLE LENGTH NORMALIZE GENERAL CASE

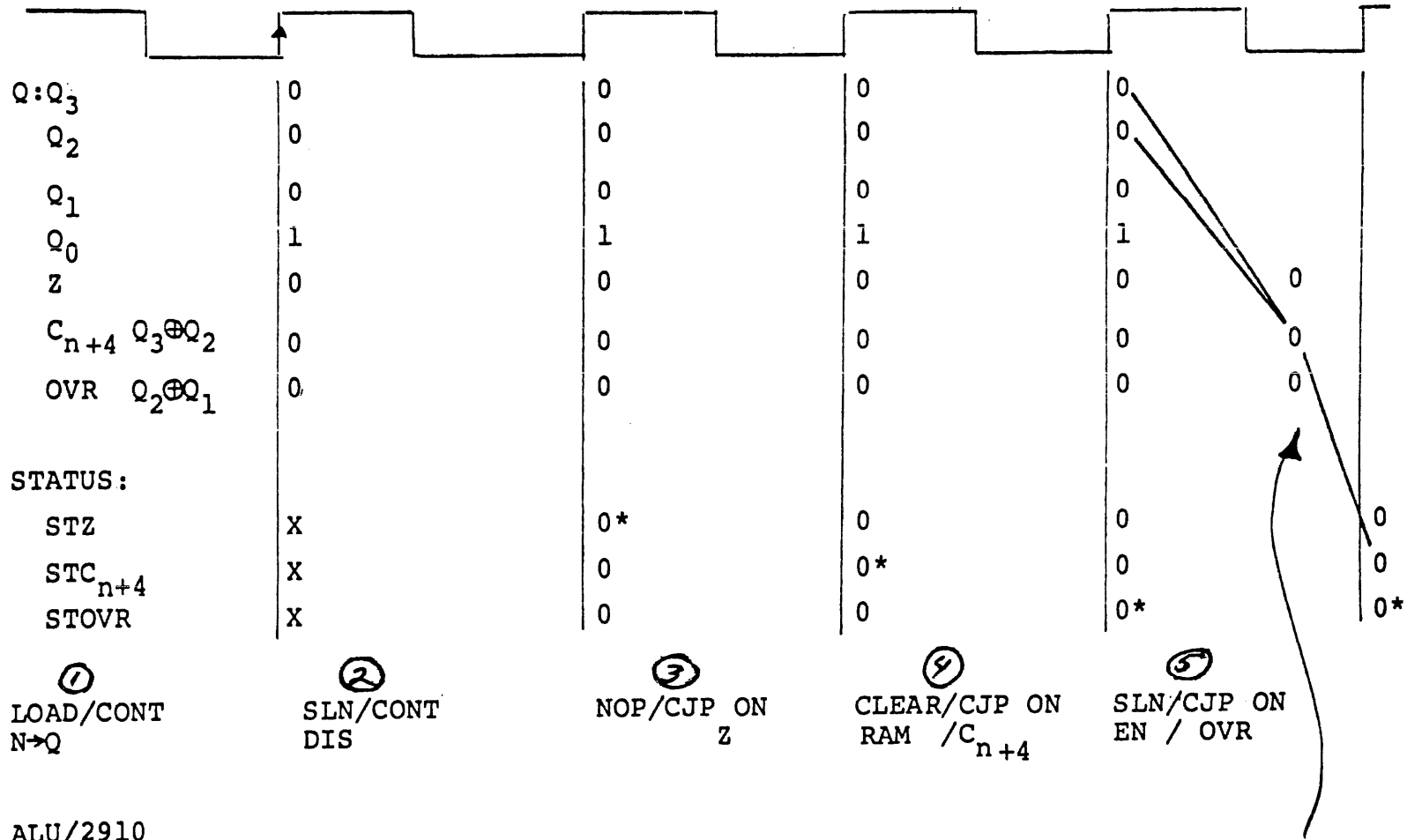


M_s = MACRO SIGN BIT STATUS REGISTER

ALGORITHM EXPLANATION

1. Number must be in Q for Am2903/29203.
2. Cannot normalize a zero. With IEN = high, execute SLN and store condition codes on this step only.
3. If result of 2 is zero, abort.
4. If already normalized, quit. Test result of step 2. Don't latch codes while clearing exponent register.
5. Again test result of Step 2 to see if one shift will normalize the number. Simultaneously shift with SLN. If TEST = TRUE, we are done.
6. Continue to test and shift until done. Because we execute the shift and test simultaneously, we in fact execute one more shift than we require.
7. Correction step. Downshift and restore sign bit from condition code register. Decrement exponent. This requires two microcycles.

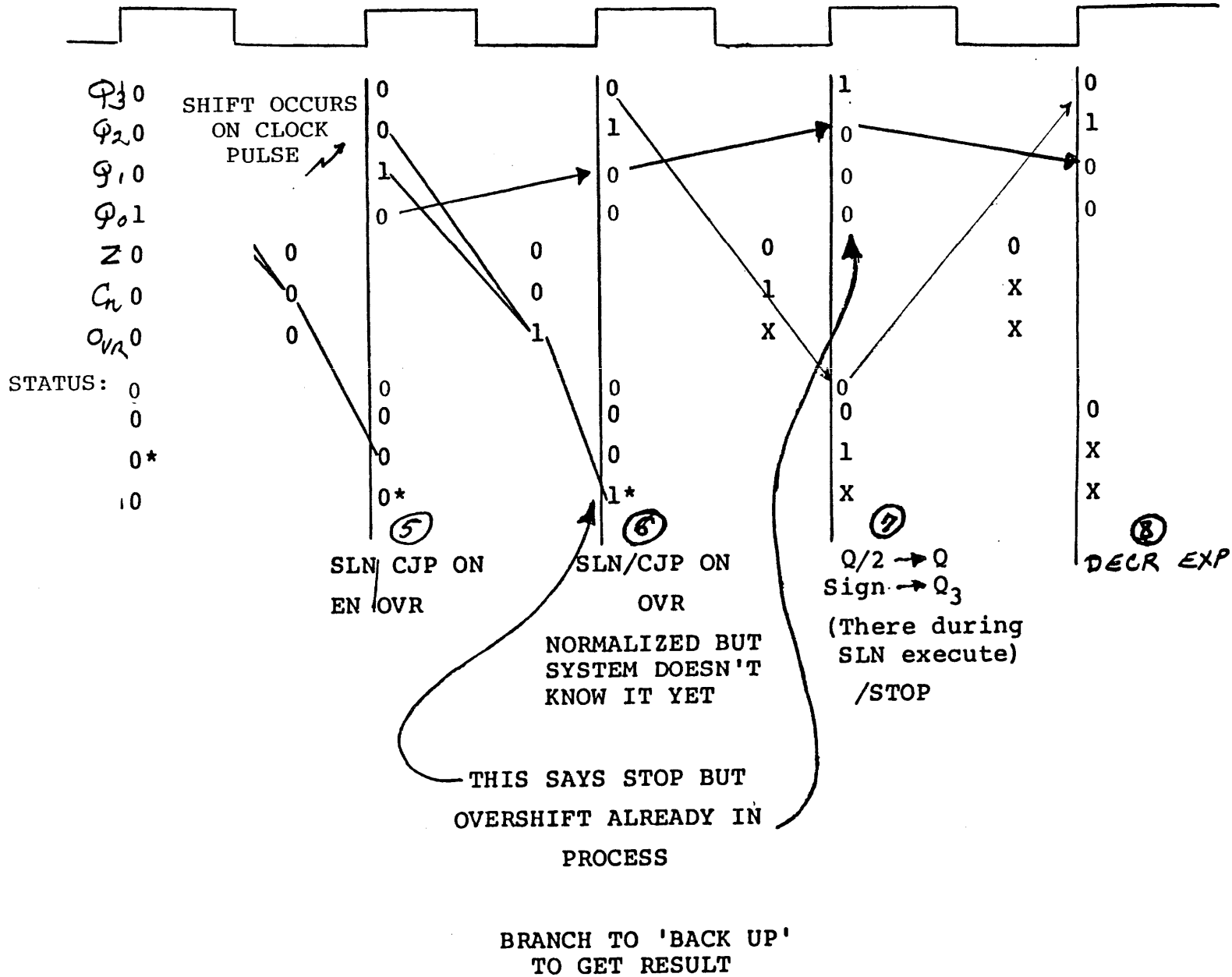
SINGLE LENGTH NORMALIZE TIMING FLOW



ALU/2910

Z ON CURRENT
 C_{n+4} Q not
 =
 OVR NEW ONE

TIMING FLOW (CONT.)



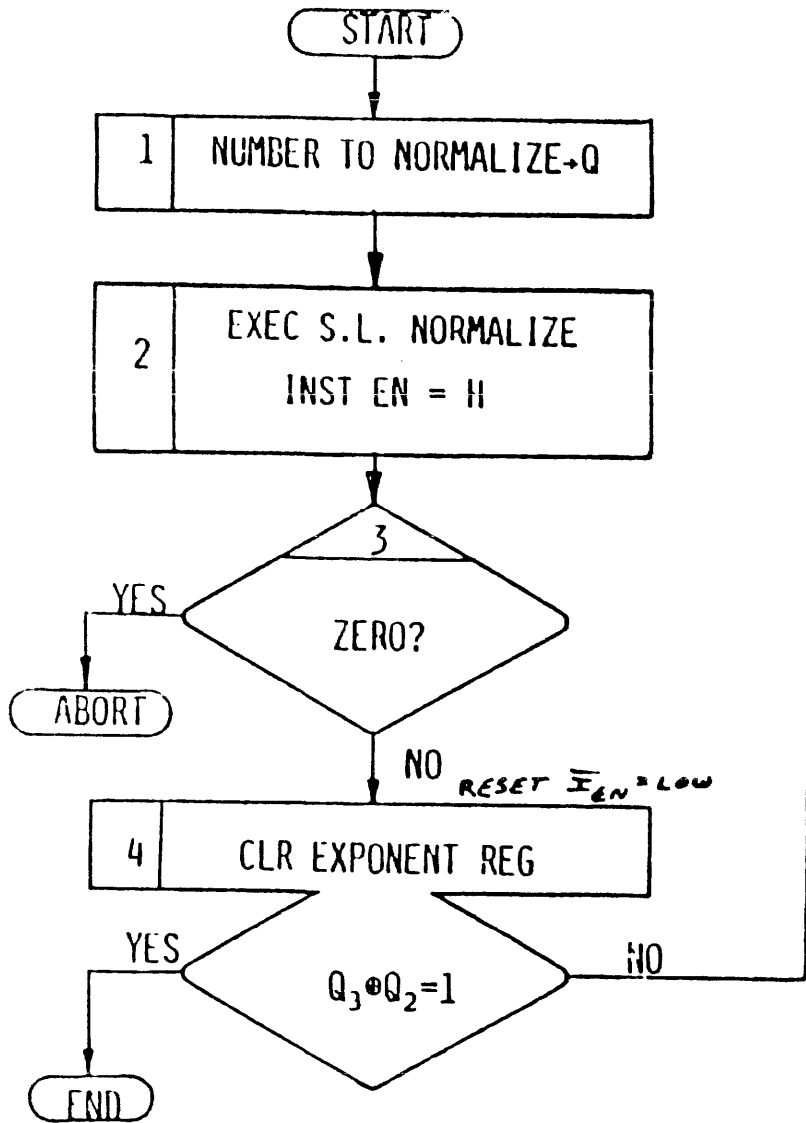
SINGLE LENGTH NORMALIZE MICROCODE

2910	CND	BRCH	A	L	U							STORE			
INST	MUX	CNTR	SRCE	FUNCT	DEST	RA	RB	C1n	OEY	IEN	ROT	STATUS	COMMENT		
1	CONT	#	#	RAMAQ	INCR	LOADQ	Ra	#	LOW	OFF	EN	#	NO	LOAD	Q
2	CONT	#	#	RAMAB	SPECL	SLN	#	#	LOW	OFF	OFF	#	YES	DO	SLN
3	CJP	ZERO	ABORT	#	#	#	#	#	OFF	OFF	#	#	NO	ZERO	?
4	CJP	CARY	DONE	RAMAQ	LOW	RAM	#	Rb	LOW	EN	EN	#	NO	CLR	REG/DONE?
5	CJP	OVR	DONE	RAMAB	SPECL	SLN	#	Rb	ONE	EN	EN	0-Q0	YES	DO	SLN
6	CJP	OVR	6	RAMAB	SPECL	SLN	#	Rb	ONE	EN	EN	0-Q0	YES	DO	SLN
7	CONT	#	#	#	#	QD	#	#	LOW	OFF	EN	SIGN-	NO	DOWN	SHIFT
												TO-Q3			
8	CONT	#	#	RAMAB	SPECL	DECRMNT	#	Rb	ONE	EN	EN	#	NO	DECR	EX- PONENT USING SPECIAL FUNCTION

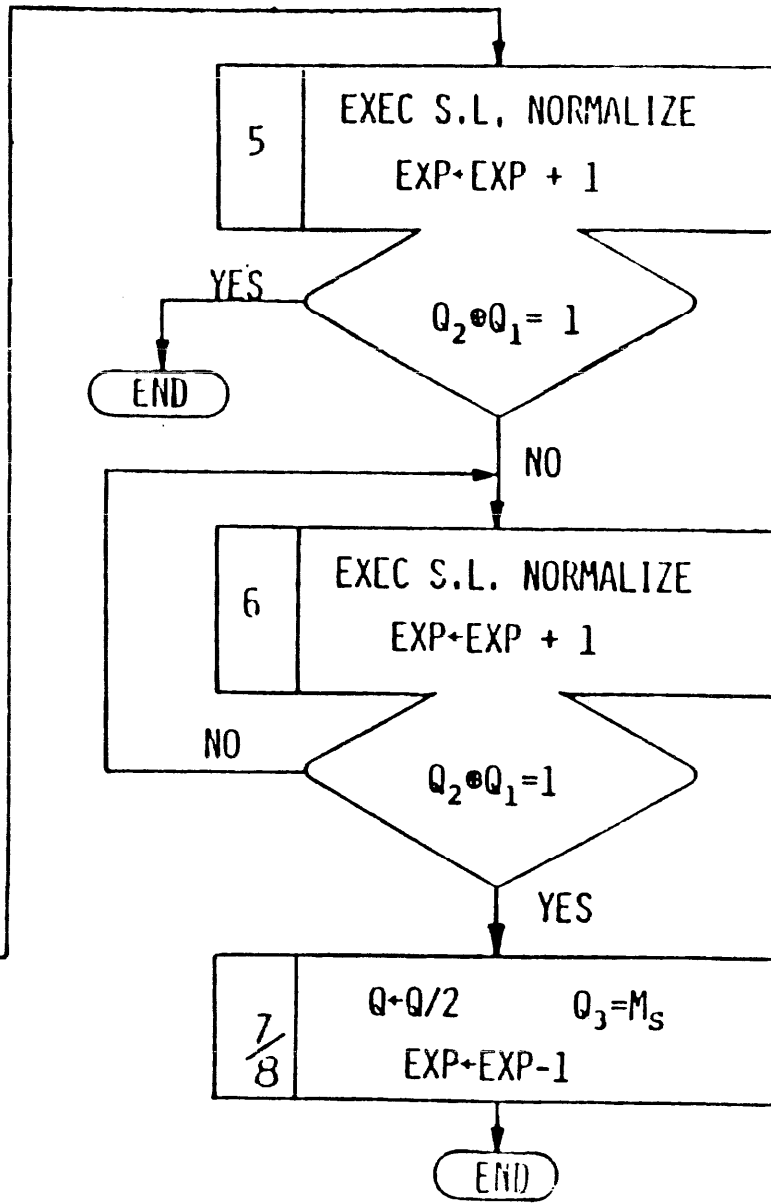
Am2903 DOUBLE LENGTH NORMALIZE (DLN)

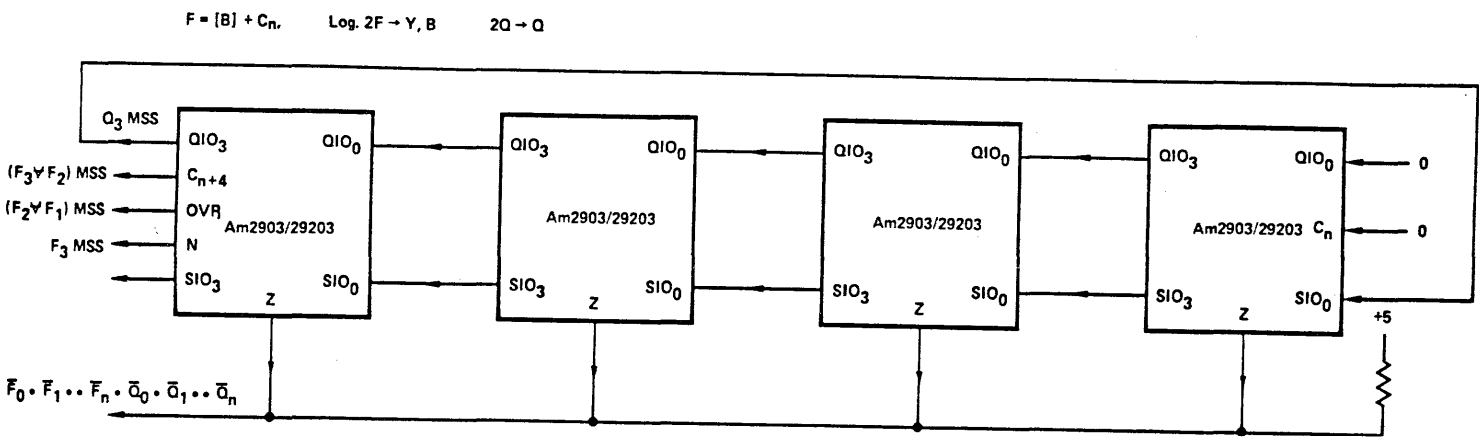
Set-up:

- MSH in RAM for B data out
- LSH in Q
- Define exponent register

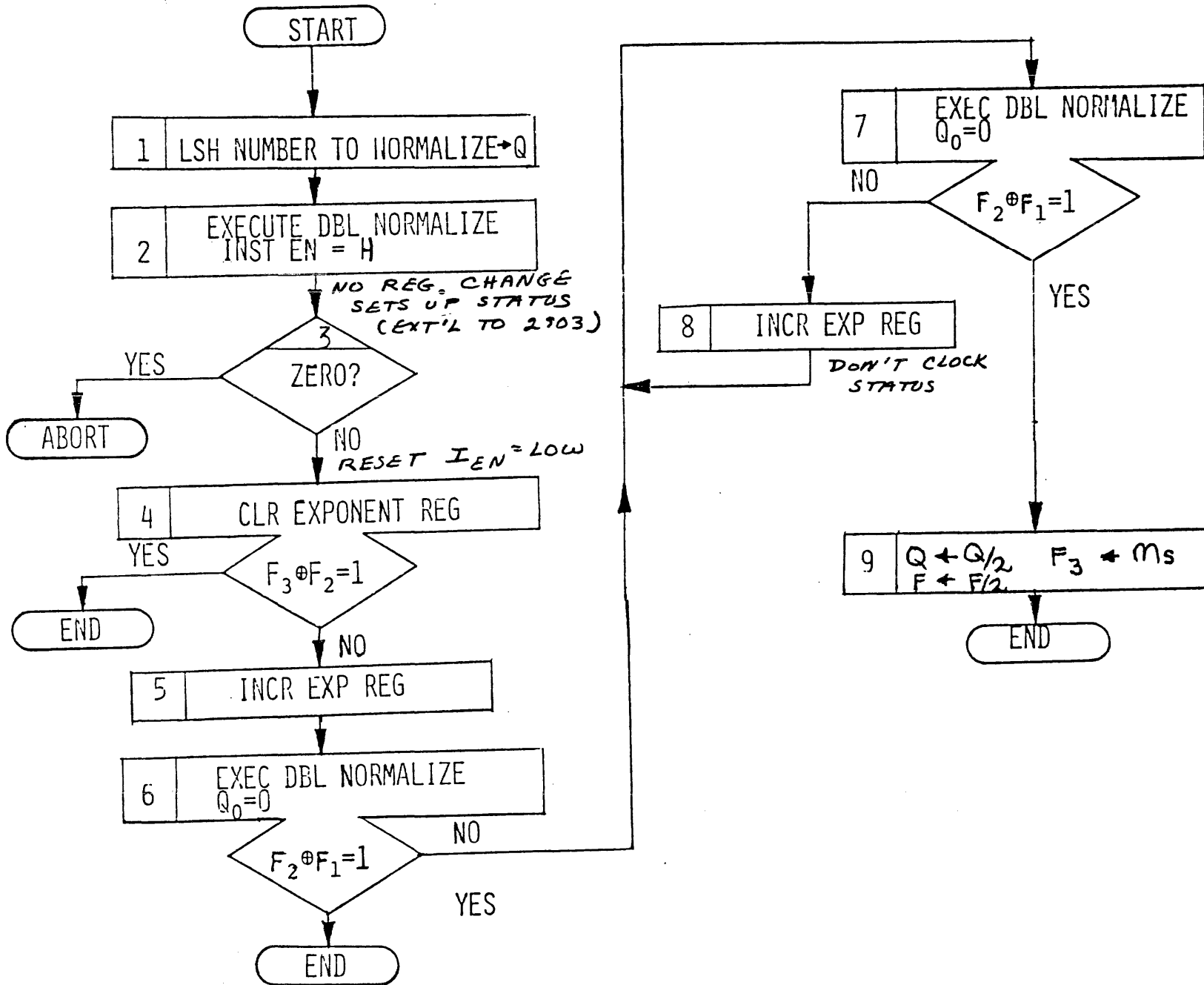


M_s = MACRO SIGN BIT STATUS REGISTER





DOUBLE LENGTH NORMALIZE - GENERAL CASE



Am29203

SPECIAL FUNCTIONS

Am29203 SPECIAL FUNCTIONS

The Am29203 will include all of the special functions on the Am2903 plus:

- Decrement by 1 or 2

- Single cycle BCD add

- Single cycle BCD subtracts
 - R - S

 - S - R

- BCD --> binary conversion

- Binary --> BCD conversion

- BCD divide by two adjust (performed after a downshift)

Am29203

SPECIAL FUNCTIONS (Note 7)

(Hex) I ₈ I ₇ I ₆ I ₅	I ₄	(Hex) I ₃ I ₂ I ₁ I ₀	Special Function	ALU Function	ALU Shifter Function	SIO ₃		SIO ₀	Q Reg & Shifter Function	QIO ₃	QIO ₀	WRITE
						Most Sig Slice	Other Slices					
0	L	0	Unsigned Multiply	$F = S + C_n$ if Z = L $F = R + S + C_n$ if Z = H	Log F/2 → Y (Note 1)	Z	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
1	L	0	BCD to Binary Conversion	(Note 4)	Log F/2 → Y	Input	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
1	H	0	Multiprecision BCD to Binary	(Note 4)	Log F/2 → Y	Input	Input	F ₀	Hold	Z	Q ₀	L
2	L	0	Two's Complement Multiply	$F = S + C_n$ if Z = L $F = R + S + C_n$ if Z = H	Log F/2 → Y (Note 2)	Z	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
3	L	0	Decrement by One or Two	$F = S - 2 + C_n$	F → Y	Z	Z	Parity	Hold	Z	Z	L
4	L	0	Increment by One or Two	$F = S + 1 + C_n$	F → Y	Input	Input	Parity	Hold	Z	Z	L
5	L	0	Sign/Magnitude Two's Complement	$F = S + C_n$ if Z = L $F = \bar{S} + C_n$ if Z = H	F → Y (Note 3)	Input	Input	Parity	Hold	Z	Z	L
6	L	0	Two's Complement Multiply, Last Cycle	$F = S + C_n$ if Z = L $F = S - R - 1 + C_n$ if Z = H	Log F/2 → Y (Note 2)	Z	Input	F ₀	Log Q/2 → Q	Input	Q ₀	L
7	L	0	BCD Divide by Two	(Note 4)	F → Y	Z	Z	Parity	Hold	Z	Z	L
8	L	0	Single Length Normalize	$F = S + C_n$	F → Y	F ₃	F ₃	Z	Log 2Q → Q	Q ₃	Input	L
9	L	0	Binary to BCD Conversion	(Note 5)	Log 2F → Y	F ₃	F ₃	Input	Log 2Q → Q	Q ₃	Input	L
9	H	0	Multiprecision Binary to BCD	(Note 5)	Log 2F → Y	F ₃	F ₃	Input	Hold	Z	Input	L
A	L	0	Double Length Normalize and First Divide Op	$F = S + C_n$	Log 2F → Y	$R_3 \vee F_3$	F ₃	Input	Log 2Q → Q	Q ₃	Input	L
B	L	0	BCD Add	$F = R + S + C_n$ BCD (Note 6)	F → Y	0	0	Z	Hold	Z	Z	L
C	L	0	Two's Complement Divide	$F = S + R + C_n$ if Z = L $F = S - R - 1 + C_n$ if Z = H	Log 2F → Y	$\overline{R_3 \vee F}$	F ₃	Input	Log 2Q → Q	Q ₃	Input	L
D	L	0	BCD Subtract	$F = R - S - 1 + C_n$ BCD (Note 6)	F → Y	0	0	Z	Hold	Z	Z	L
E	L	0	Two's Complement Divide Correction and Remainder	$F = S + R + C_n$ if Z = L $F = S - R - 1 + C_n$ if Z = H	F → Y	F ₃	F ₃	Z	Log 2Q → Q	Q ₃	Input	L
F	L	0	BCD Subtract	$F = S - R - 1 + C_n$ BCD (Note 6)	F → Y	0	0	Z	Hold	Z	Z	L

- Notes: 1. At the most significant slice only, the C_{n+4} signal is internally gated to the Y₃ output.
2. At the most significant slice only, $F_3 \vee \text{OVR}$ is internally gated to the Y₃ output.
3. At the most significant slice only, $S_3 \vee F_3$ is generated at the Y₃ output.
4. On each slice, $F = S$ if magnitude of S_{0-3} is less than 8 and $F = S$ minus 3 if magnitude of S_{0-3} is 8 or greater.
5. On each slice, $F = S$ if magnitude of S_{0-3} is less than 5 and $F = S$ plus 3 if magnitude of S_{0-3} is 5 or greater. Addition is module 16.
6. Additions and subtractions are BCD adds and subtracts. Results are undefined if R or S are not in valid BCD format.
7. The Q Register cannot be used explicitly as an operand for any Special Functions. It is defined implicitly within the functions.

L = LOW Hi-Z = High Impedance
H = HIGH = Exclusive OR
X = Don't Care Parity = $SIO_3 \vee F_3 \vee F_2 \vee F_1 \vee F_0$

DECREMENT by 1 or 2

- It is not possible to decrement by 1 or 2 without going to a special function except by storing "1" or "2" in a register.
- In a byte-addressable memory both byte addressing and word addressing capability is desirable. For address decrement (e.g. stack operations).
- Byte addressing: $\langle R\theta \rangle \leftarrow \langle R\theta \rangle - 1$
- Word addressing: $\langle SP \rangle \leftarrow \langle SP \rangle - 2$
- The special function "DECRMNT" provides this capability.
- 4 DECRMNT $F = S - 2 + C_{in}$

2910	SOURCE	FUNCT	DEST	RA	RB	C _{in}
CONT	RAMAB	SPECL	DECRMNT	#	R2	ONE

BINARY/BCD CONVERSION

- BCD number is always in a specified RAM register
- Binary number is always in the Q-register
- SIO_{\emptyset} is connected to QIO_n
- SIO_{\emptyset} is connected to QIO_n
- **For binary to BCD:**
 - Binary number must not exceed BCD value
 - Binary number is loaded into Q and Ra is cleared
 - BIN-to-BCD is executed N times for an N-bit number

	2910	CND	BRCH	A	L	U												
	INST	MUX	CNTR	SRCE	FUNCT	DEST	RA	RB	Cin	ROT	COMMENT							

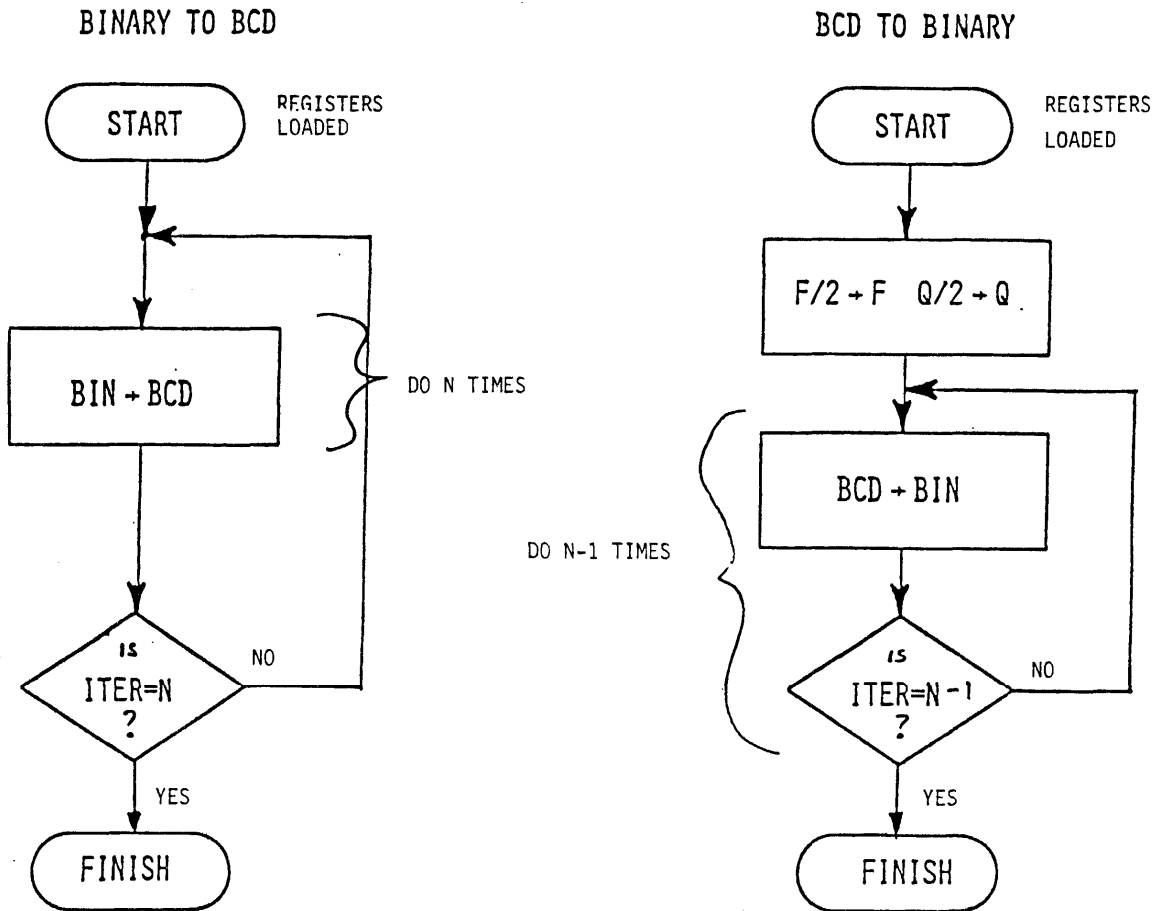
1	LDCT	#	15	RAMAQ	INCR	LOADQ	RA	#	LOW	#	PUT BINARY IN Q							
2	CONT	#	#	RAMAQ	LOW	RAM	#	Rb	LOW	#	CLEAR RAM Rb							
3	RPCT	#	3	RAMAB	SPECL	BIN.BCD	#	Rb	LOW	Q3-S0	PERFORM CONVERSION							

BINARY/BCD CONVERSION (cont)

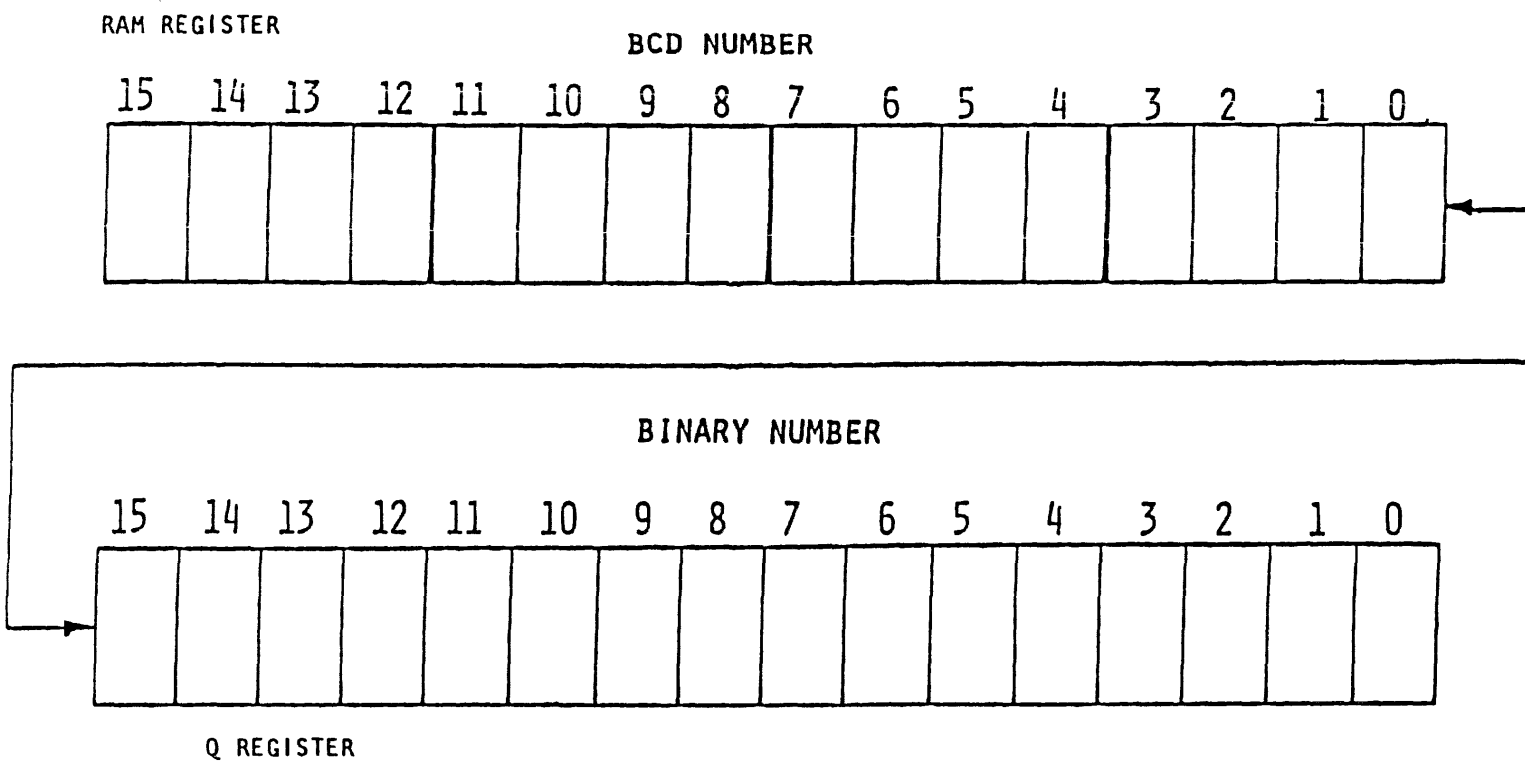
- **For BCD to binary:**
 - BCD number is loaded into Ra, Q is cleared
 - Ra and Q are downshifted one bit (a precorrection step)
 - BCD-to-BIN is executed N-1 times for an N-bit number

2910	CND	BRCH	A	L	U							
INST	MUX	CNTR	SRCE	FUNCT	DEST	RA	RB	Cin	ROT	COMMENT		
1	CONT	#	#	RAMAQ	LOW	LOADQ	#	#	LOW	#	CLEAR Q	
2	LDCT	#	14	RAMAQ	INCR	RAMQDL	Ra	#	LOW	#	INITIAL DOWNSHIFT	
3	RPCT	#	3	RAMAB	SPECL	BCD.BIN	#	Ra	LOW	SO-Q3	PERFORM CONVERSION	

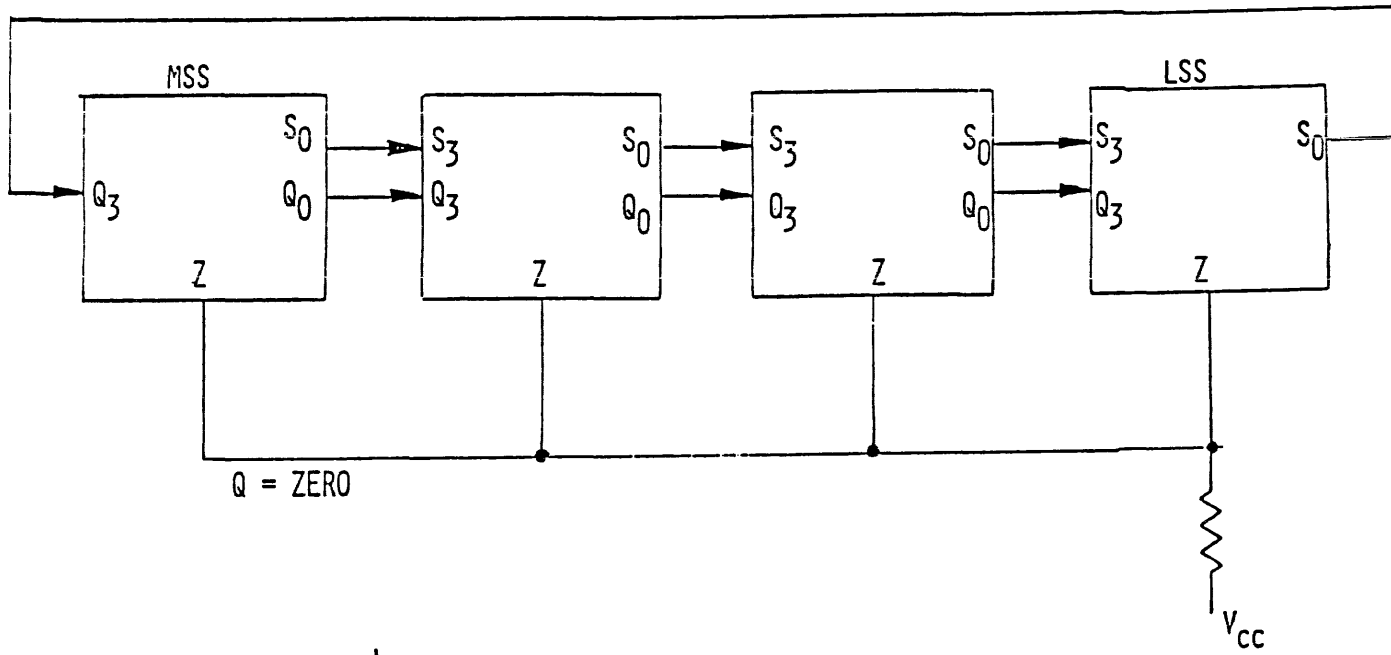
Microcode Flow Chart of BCD Conversion



BCD CONVERSIONS



BCD TO BINARY



ALU	DESTINATION
BCD TO BIN CONVERSION	F/2 → RAM; Q/2 → Q

SIGNIFICANT SPEED IMPROVEMENT ON Am2903

	Am2903 (MIL MAX)	Am2903A (PROJECTED MIL MAX)	%FASTER
	-----	-----	-----
ADDRESS TO G,P	84	57	32%
CN TO Z	65	40	38%
ADDRESS TO Z	126	75	41%
ADD CYCLE TIME	181	129	29%
LOGIC CYCLE TIME	152	100	34%

2-1700

ED2900A

2-1700

**EXPANDED MEMORY
FOR
ALU REGISTER EXPANSION**

EXPANDED MEMORY

- The Am2901, Am2903 and Am29203 each contains only 16 scratchpad registers plus the Q register.

- Some applications require more than 17 registers.

- The Am2903 and the Am29203 register set can easily be expanded.
 - Use the Am29705 RAM with the Am2903

 - Use the Am29707 RAM with the Am29203

2-1730

ED2900A

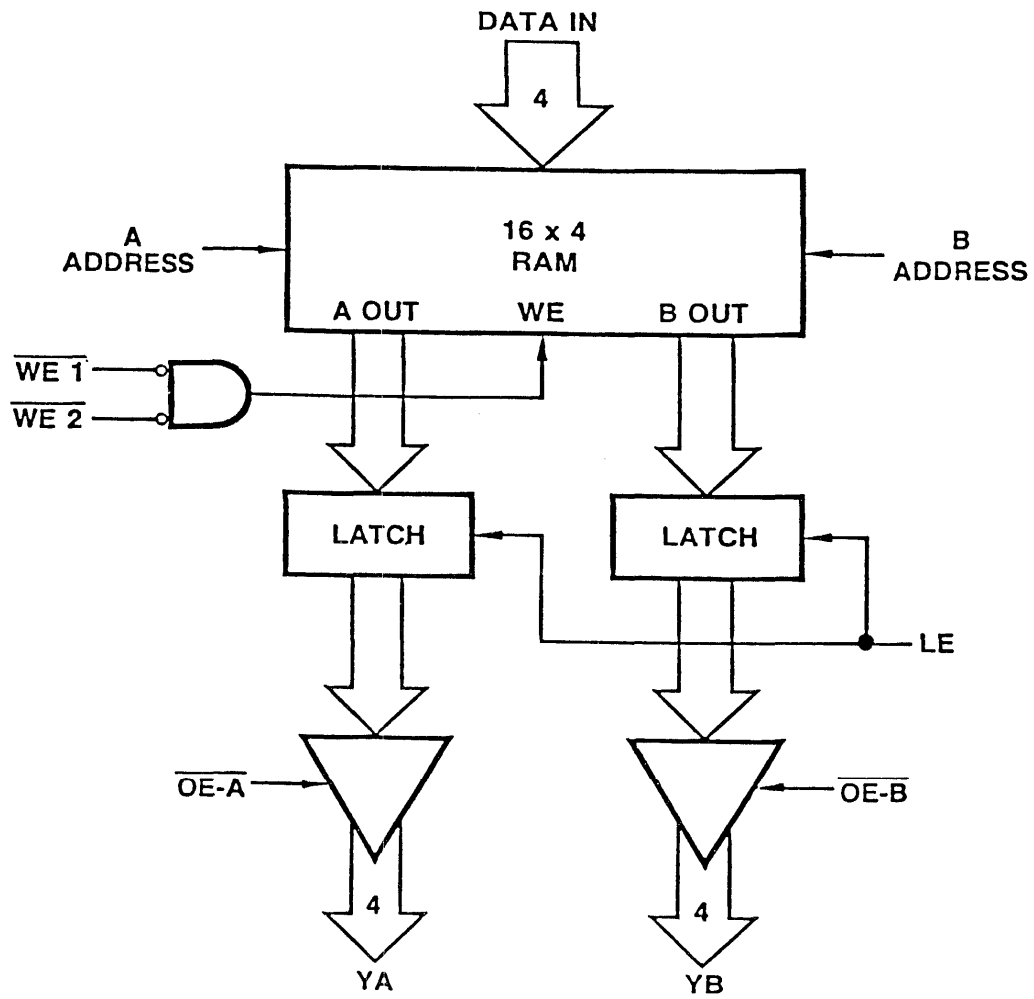
2-1730

Am29705

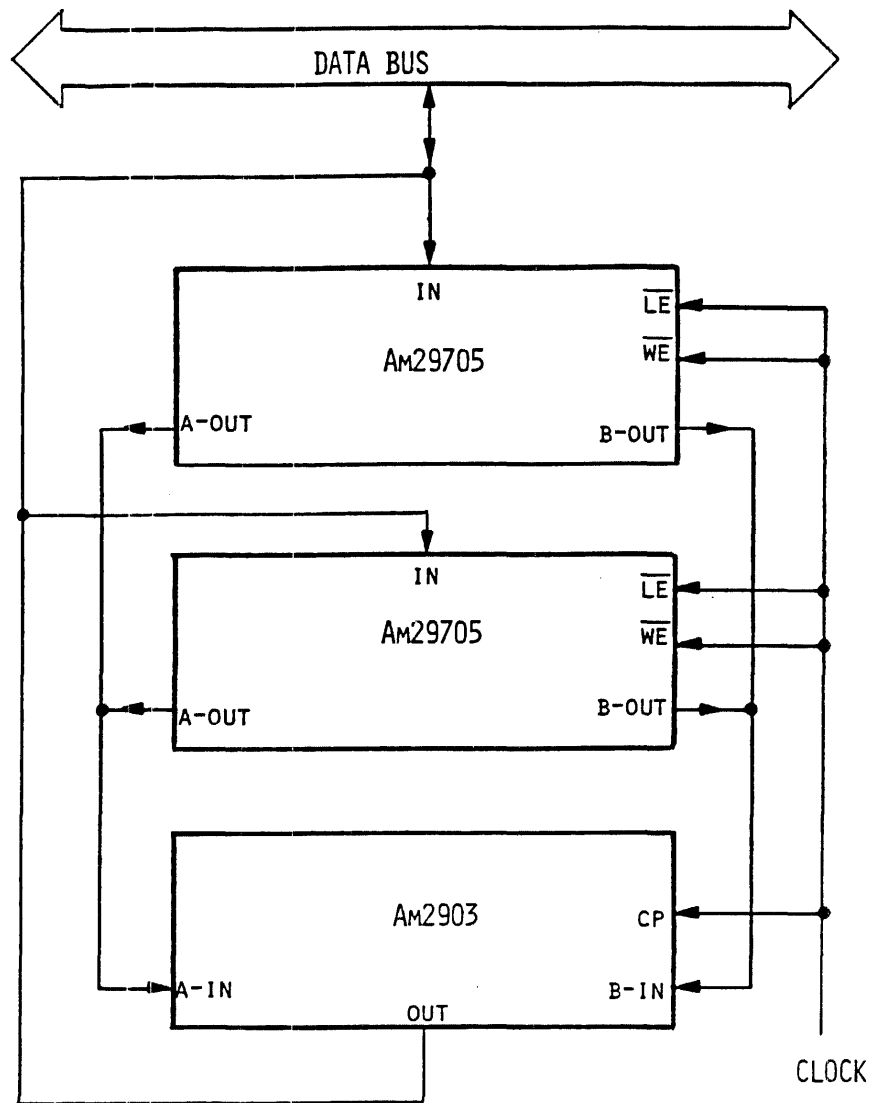
Am29705**16-WORD BY 4-BIT, 2-PORT RAM**

- Distinctive characteristics
 - Two output ports with latches (Buffers)
 - Separate data input port
 - Non-inverting data
 - Independent three-state outputs
 - Configurable in either "transparent" or "edge triggered" mode.
 - Designed for Am2903 register expansion

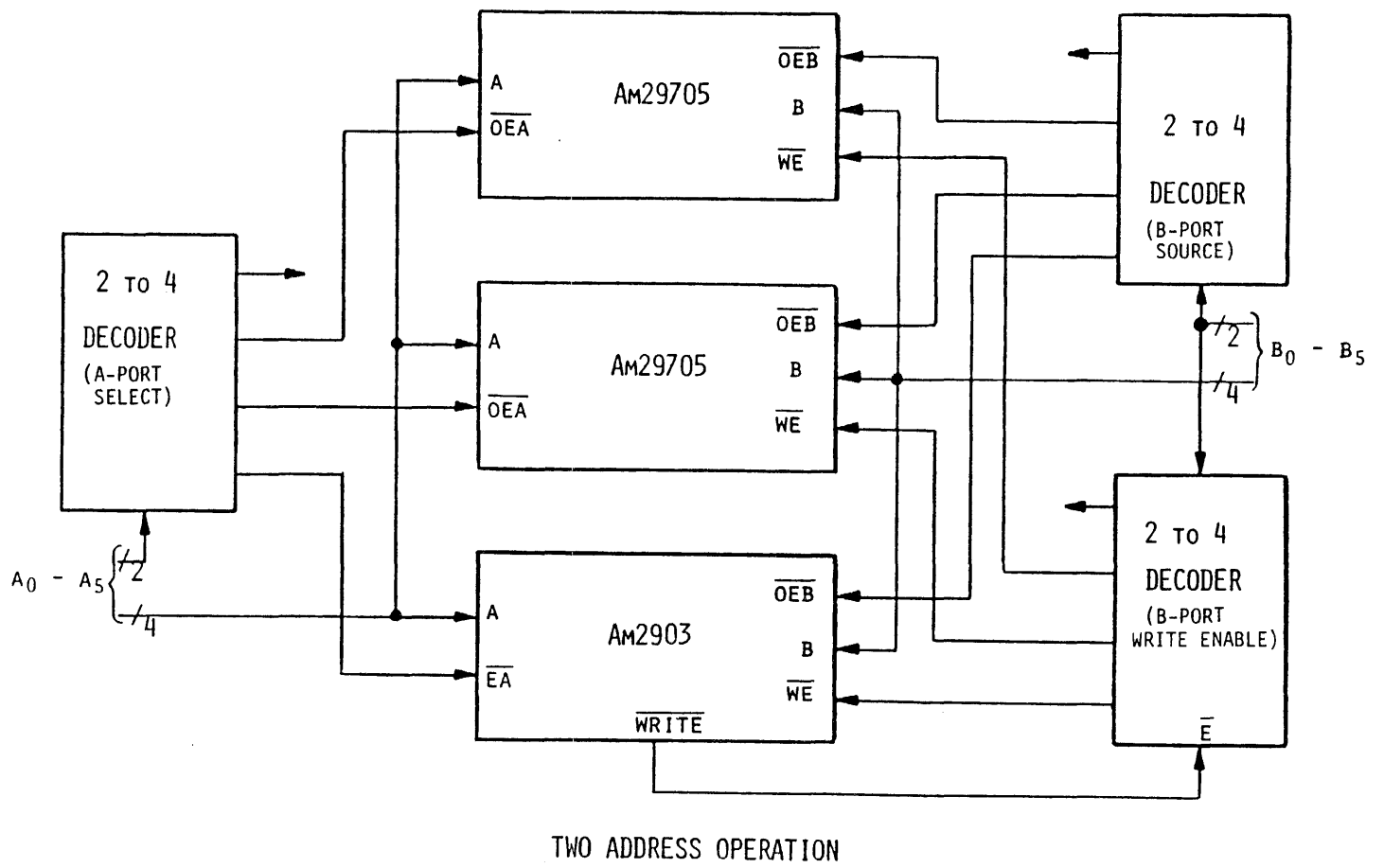
Am29705



Am2903 - Data Bus Cascading



Am2903 - RAM Address Cascading



Am2903 SCRATCHPAD EXPANSION

- Am29705 functionally identical to Am2903 registers
- Am29715A PROM stores constants, masks
- Five data busses shown
- Three-address architecture shown

Am29705A 16-WORD BY 4-BIT TWO PORT RAM

	<u>Am29705</u>	<u>Am29705A</u>
Commercial maximum:		
access time	53	30
LE to YA/YB	32	20
A-latch reset	35	20
address set up before latch closes	45	15

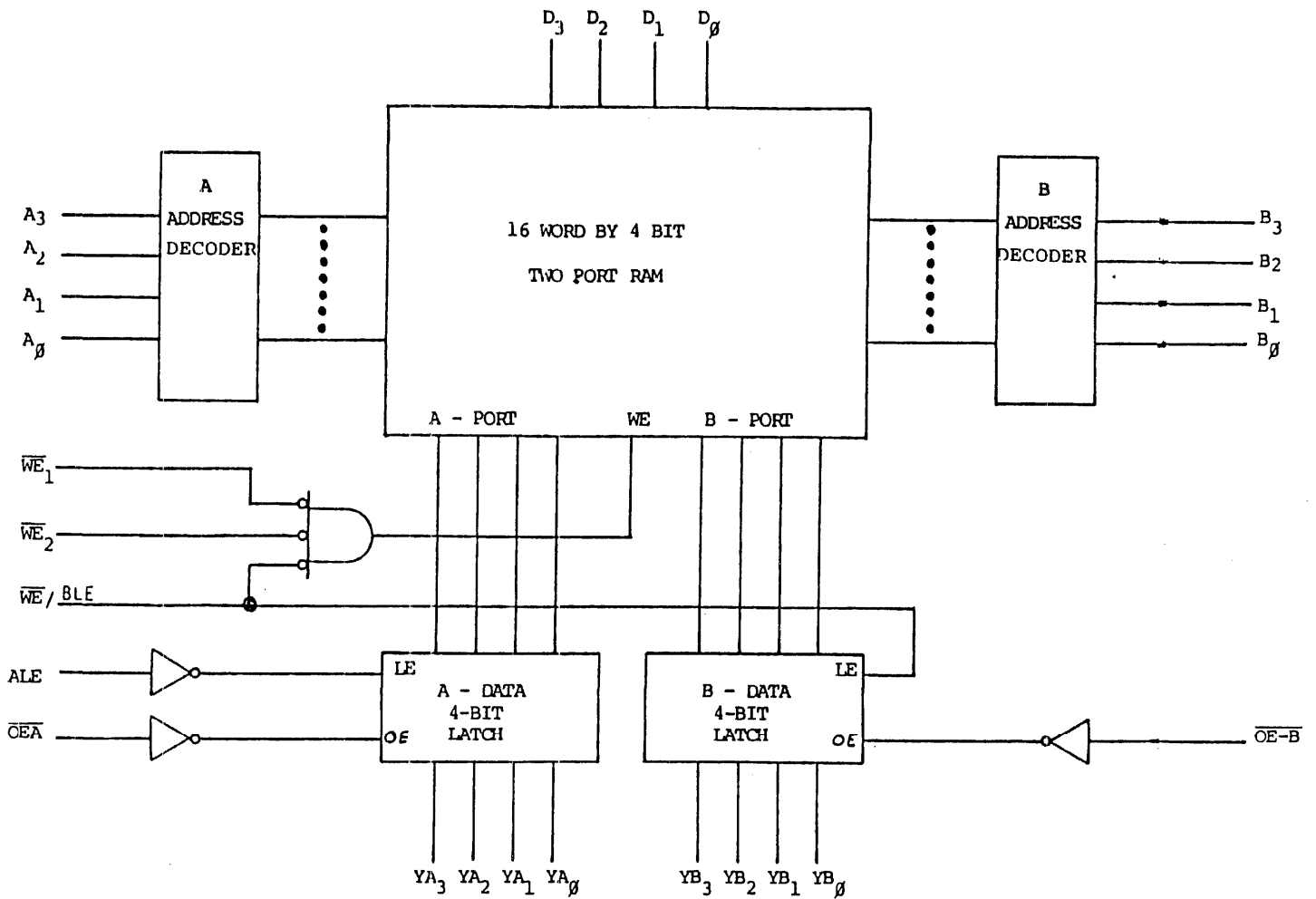
2-1810

ED2900A

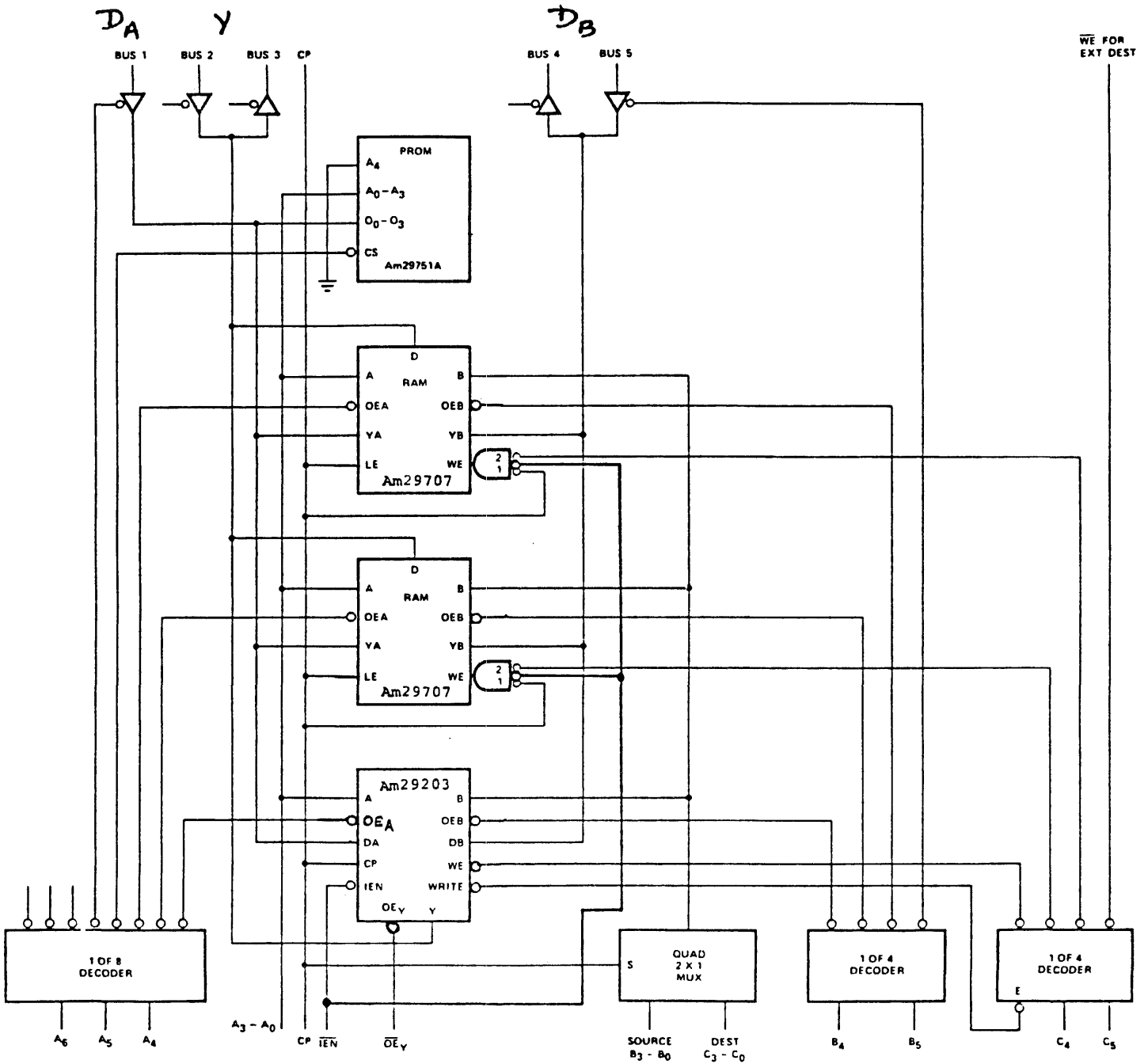
2-1810

Am29707

Am29707 -- 28 Pin



Am29203 - Am29707



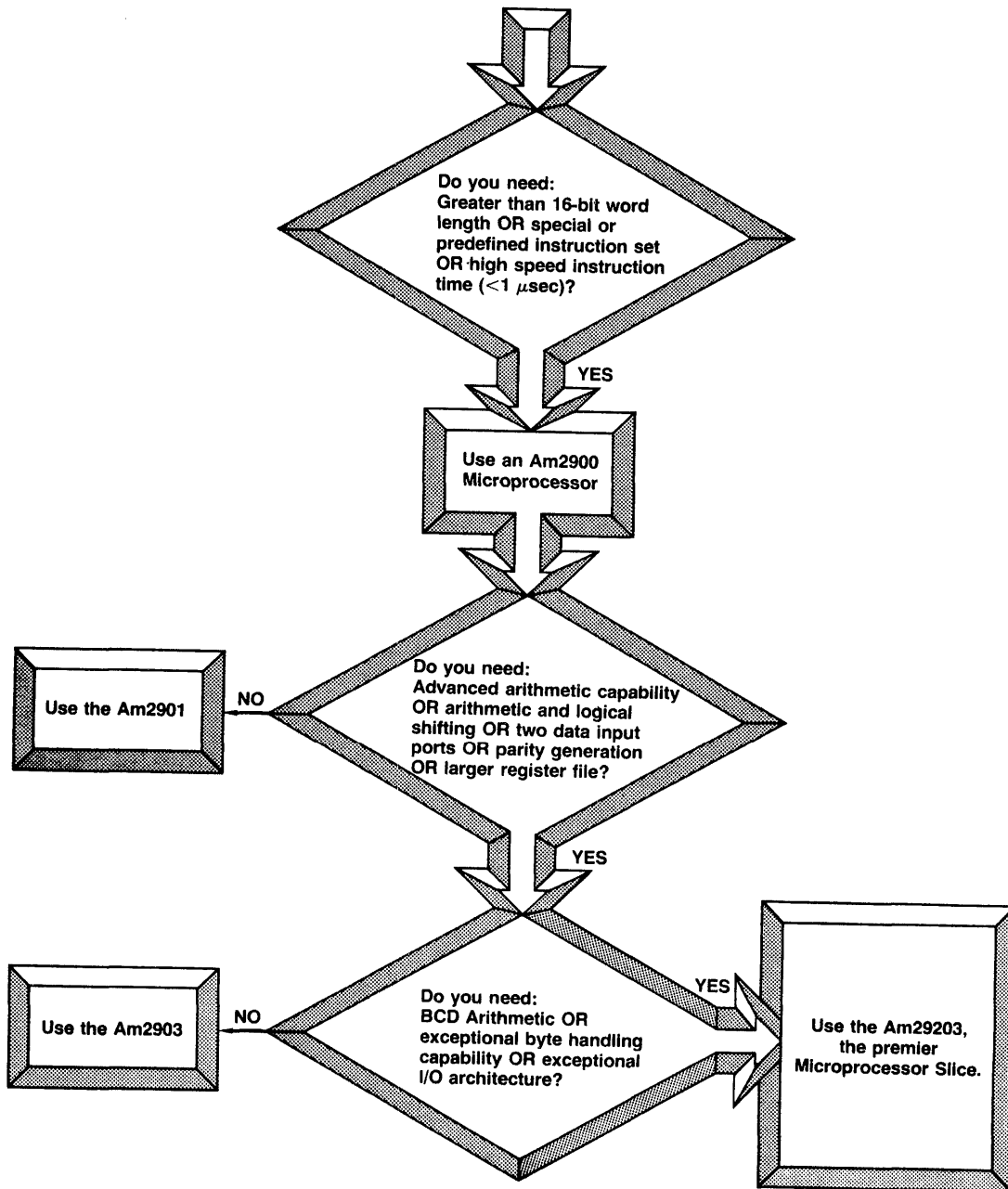
CLASS EXERCISE

- Turn to the Am2903/Am29203 exercises in the ED2900A Exercise and Laboratory Manual and do numbers 1 through 24.

Evaluation Board Experiments

- Do Am29203 laboratory exercises in Manual.

Selecting the #1 Am2900 Microprocessor Slice



TIMING COMPARISON (GUARANTEED COMMERCIAL)				
ADDRESS TO:	2901B	2901C	2903	2903A/29203
Y	60	40	99	68
GP	50	37	81	52
F=∅	70	40	123	72
D TO:				
Y	38	30	87	59
Cn+4	40	30	60	49
F=∅	48	38	111	65
I TO:				
Y	51	35	71	64
F=∅	60	38	95	72

2-1870

ED2900A

2-1870

SIMPLE COMPUTER SOLUTION

2-1880

ED2900A

2-1880

INTRODUCTION TO INTERRUPTS

INTERRUPTS

- An interrupt is a request for service by some device or process, external to the CPU.

- An interrupt usually occurs asynchronously with respect to the processor fetch-execute clock cycle (even though it may be checked on a synchronous basis).

- An interrupt request is often identified by a one-bit signal, similar to the ALU status lines. We will be primarily concerned with this type of interrupt.

- An interrupt may also be caused by particular processor instructions (such as invalid opcodes, privileged instructions, or system service calls) which decode to special microroutines that behave like interrupt routines.

- For interrupt support two specific AMD products are available:
 - Am2913 Priority Interrupt Expander
 - Am2914 Vectored Priority Interrupt Controller

- Only the Am2913 will be considered in this course.

TYPES OF INTERRUPTS

- Intraprocessors

- within the processor

- asynchronous

- zero divide

- ALU overflow

- invalid memory access

- invalid instruction

- privileged instruction

- other status testing

- Intrasystem

- within the system (outside the processor)

- I/O request CRT printer tape disk

- memory parity error

- DMA request

- peripheral failure

- power failure

INTERRUPT TYPES (CONT'D)

- Executive (traps)
 - task request
 - hardware allocation
 - interprogram communication
 - supervisory program call

- Interprocessor - between two processors
 - data transfer
 - status transfer

INTERRUPT LEVELS

- In any computer or controller there are three levels at which interrupts may be handled:
 - Software level (also machine level or macro level)
visible and handled at the machine language level
 - Firmware level (also microprogram level)
handled by microcode routines
 - Hardware level
handled by special purpose hardware

- The same general interrupt-handling algorithm (process) is used at all three levels.
 - Acknowledge interrupt
 - Save current "state" of system
 - Service interrupt
 - Reinstall "state" of system prior to interrupt

- **Software Level**

- Interrupts handled at a fixed place in the machine level instruction cycle.
- Interrupts are usually handled during the main memory fetch portion of the cycle (minimize storage of "state").
- On detection of an interrupt a machine level interrupt routine is activated and executed.
- On completion of the interrupt routine the original machine program continues.
- Where nested interrupts are allowed, an interrupt routine itself may be interrupted.

- **Advantages**

- Can be altered as needed via programming. Does not require space in the microprogram memory.

- **Disadvantages**

- Slow
- Does require space in the program memory

- **Firmware Level**

- Interrupts are handled at fixed places in the microprogram, generally at the end of a microroutine or at a quiescent point in the microroutine if it is very long.
- On detection of an interrupt, a microroutine is activated with no alteration of the macroinstruction register.
- Nested interrupts may or may not be allowed depending upon the microroutine.
- Where interrupts are handled and whether or not they are allowed to be nested is under the control of the microprogrammer.

- **Advantages**

- Faster than software routine
- Does not interrupt or alter machine program flow

- **Disadvantages**

- Requires space in the microprogram memory

- **Hardware Level**

- Hardware-level interrupts are necessary when the response must be in "fast" relative to the other implementation levels.
- The system can be forced to immediately handle the unit causing the interrupt request immediately.

- **Advantages**

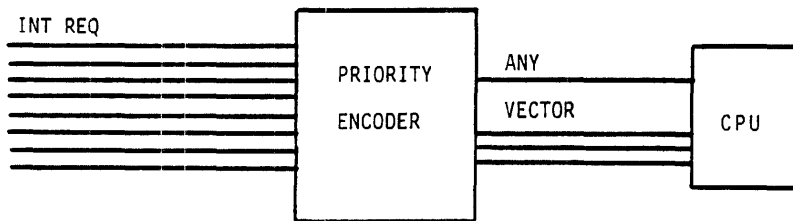
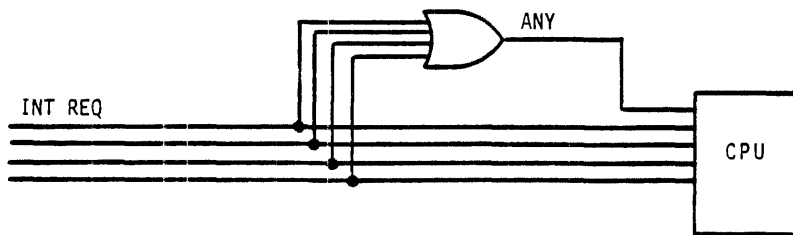
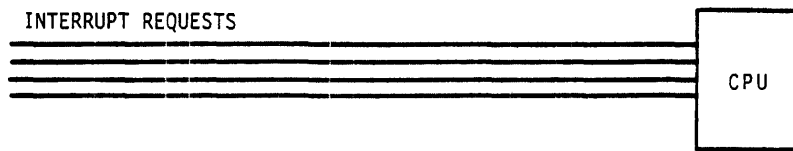
- Speed - a key feature of this approach

- **Disadvantages**

- More complex hardware
- More complex firmware

INTERRUPT ARCHITECTURES

- There are many ways in which the interrupt request lines from peripheral devices may be connected to the CPU. In order to present the basic concepts of interrupts and to examine the Am2913 and Am2914, only three basic forms need to be considered.
 1. A request line from each device is connected to the CPU.
 2. A request line from each device is connected to the CPU, and all lines are "ORed" together to form an "any device" signal to indicate that one or more of the devices requires service.
 3. A request line from each device is connected to an interrupt priority encoder. From the priority encoder a single interrupt request line (indicating that one or more devices need service) and an identifying code, called a vector (indicating the highest priority device requesting service), are connected to the CPU.



INTERRUPT ARCHITECTURES

INTERRUPT HANDLING STEPS (Algorithm)

- Recognize interrupt
 - Determine that an interrupt is pending
 - Halt the currently running process
 - Determine which device or process needs service

- Save status
 - Save the state of the system
 - Usually includes registers that will be used (overwritten) by the service routine software

- Service the interrupt
 - Perform the service needed by the requesting device

- Restore and return
 - Restore the system state saved earlier
 - Continue the running process from where it was interrupted

GENERAL INTERRUPT HANDLING TECHNIQUES

- Periodically (usually just prior to instruction fetch) the microprogram tests for interrupts by one of the following techniques.

- **Polled interrupts**
 - Check each device interrupt line to see if it needs service.
 - The order in which devices are checked determines the devices' priorities.
 - If available, check the ORed "any device" input first, poll only if it is active.
 - Wastes time and micromemory.

- **Vectored interrupts**
 - Check the "any device" signal first.
 - If any request is pending, read the coded vector to determine which device is waiting.
 - Faster, but requires priority encoder.

IMPLEMENTATION OF INTERRUPT CONTROL

SENSING THE REQUEST

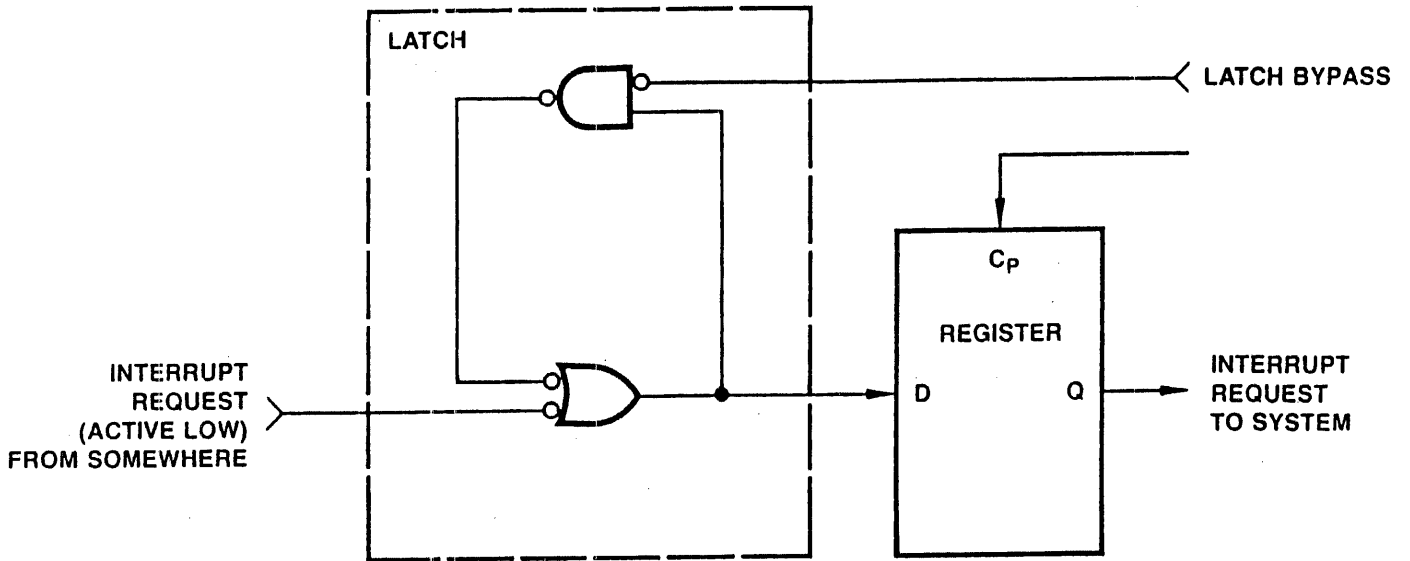
- Two types of interrupts need to be considered
 - Level signals
 - Pulse signals

- **Level signals (buffered/hold):**
 - Device request
 - Status register output

- **Pulse signals:**
 - CRT retrace

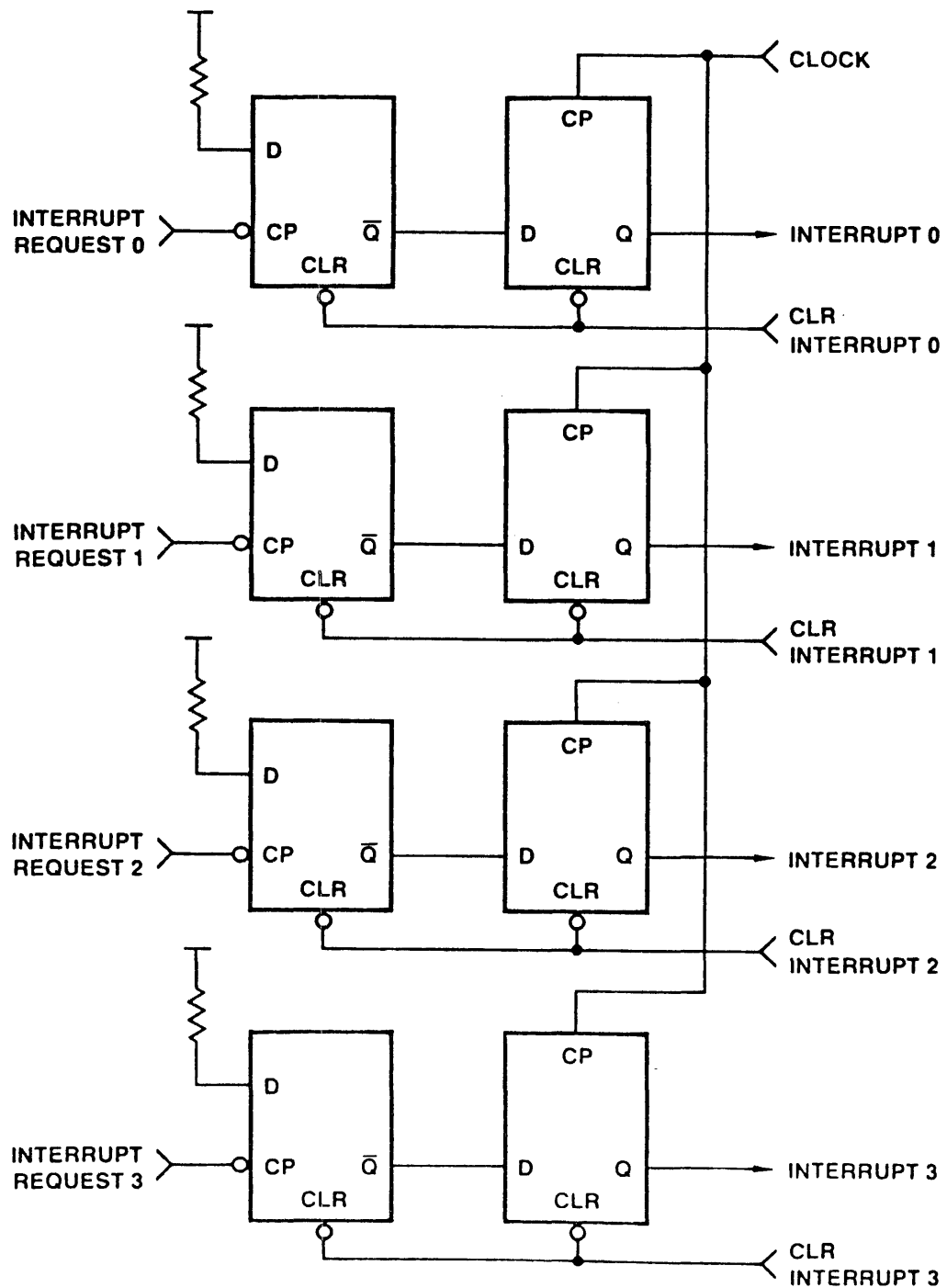
PULSE LATCHES AND SYNCHRONIZATION

- Signals are synchronized with the system clock by using them to set a clocked D flip-flop.
- Pulse signals might be gone before the next clock pulse. Thus pulse latching circuits must be used to hold the pulse.
- The circuit on the next page shows a pulse latch with a latch bypass for use with level signals.
- The following page illustrates the fact that such a circuit is required for each interrupt signal. Note that a different pulse catching circuit is used.



LATCH BYPASS = 0, PULSE CATCHER MODE = 1, LEVEL FOLLOWER MODE

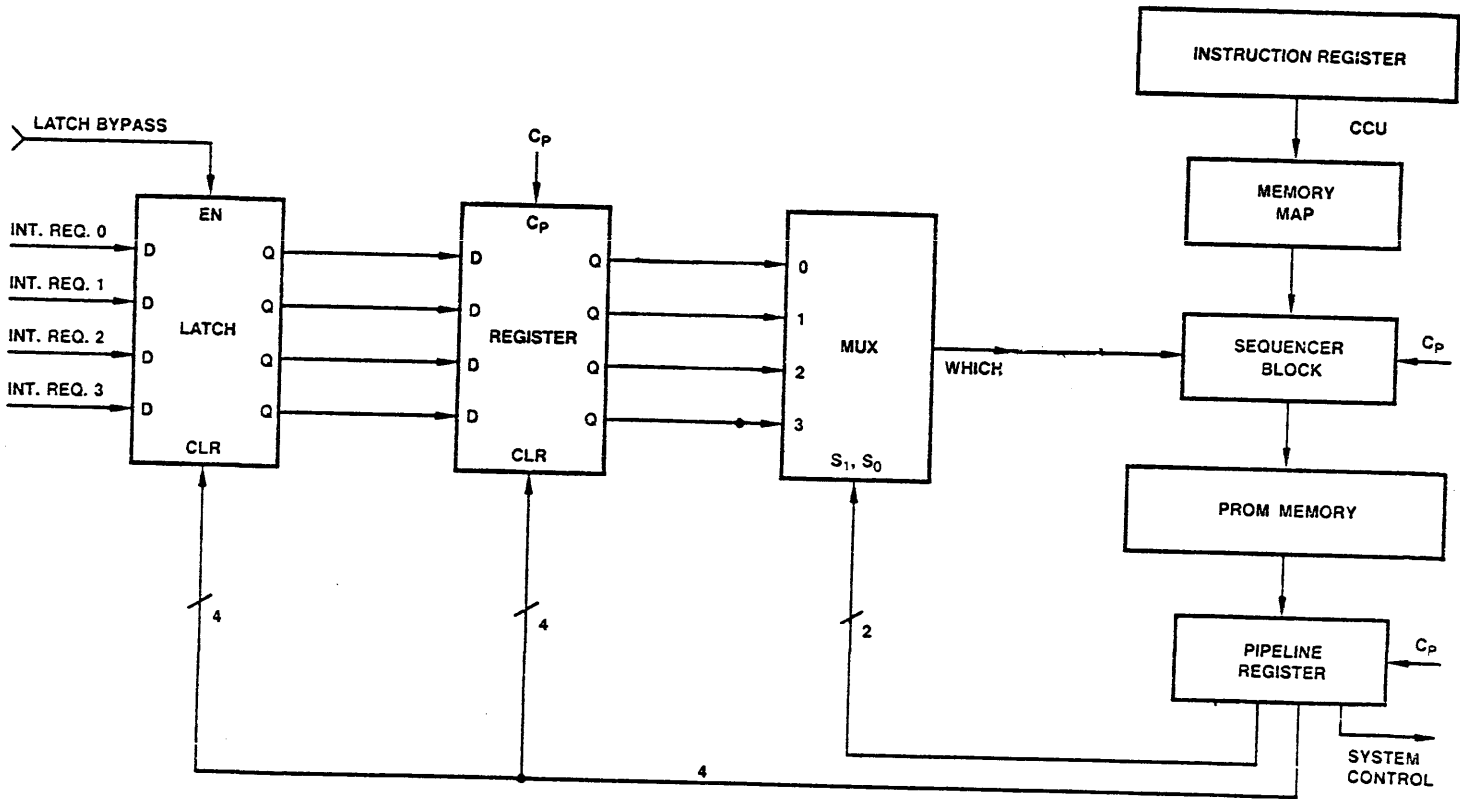
Multiple (4) Interrupt Storage



POLLED INTERRUPT IMPLEMENTATION

- Interrupt lines feed the condition code multiplexer.
- The microroutine selects each interrupt line via the test select field of the microword.
- This approach is slow, since only one interrupt can be tested per microword.
- The following microsubroutine, called at some regular point in the machine cycle, illustrates this approach.

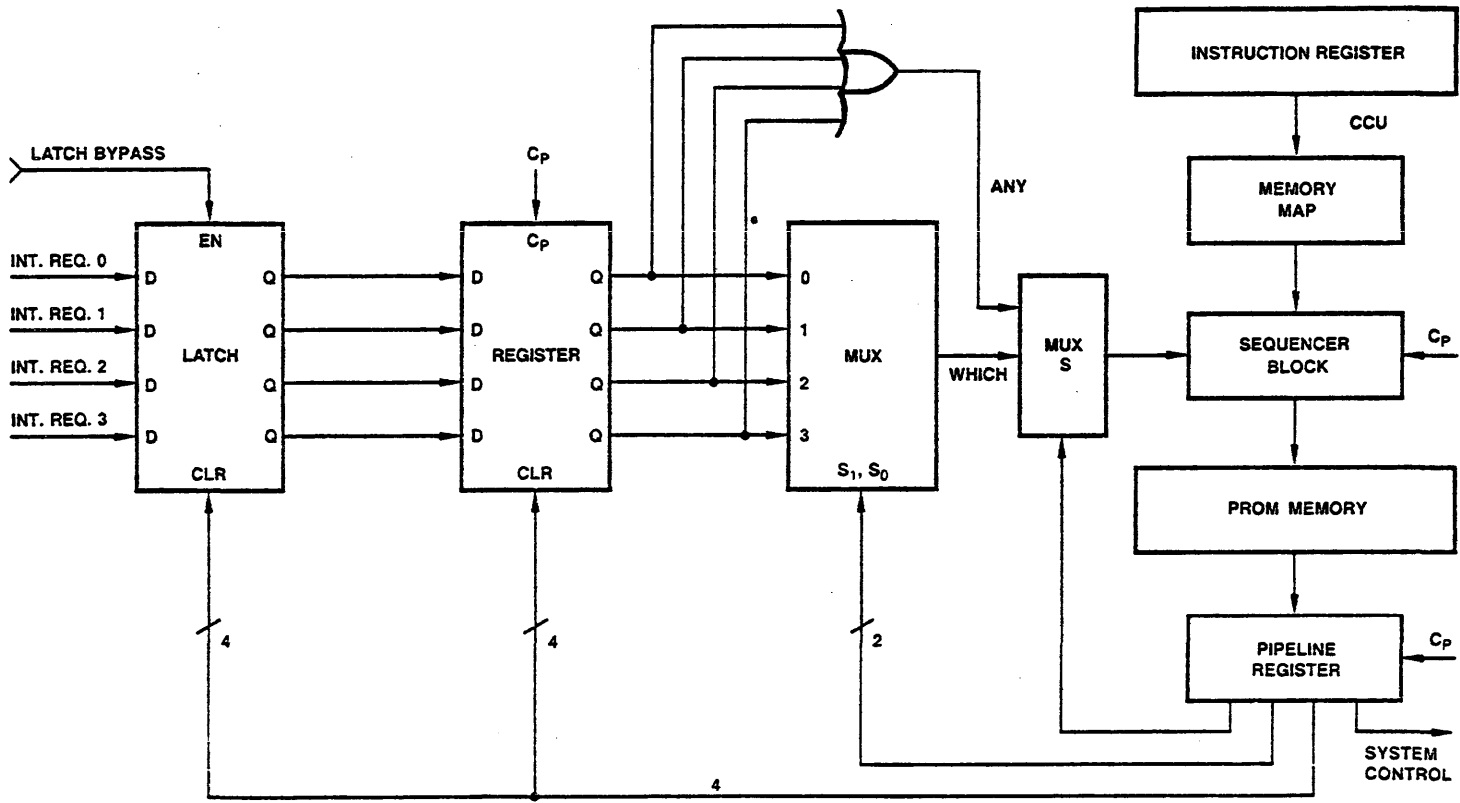
Address	Next Address	Test Select	Branch Address	CCEN	Other
INTER:	CJS	INT1	Routine1	Enable	
	CJS	INT2	Routine2	Enable	
	CJS	INT3	Routine3	Enable	
	CJS	INT4	Routine4	Enable	
	CRTN	##	##	Disable	



POLLED INTERRUPT WITH "ANY" REQUEST LINE IMPLEMENTATION

- Interrupt lines feed the condition code multiplexer.
- The microroutine can select any interrupt line via the test select field of the microword.
- An additional MUX is used to select the "any" signal or the individually selected signal. (An 8-input MUX could have handled both in this case.)
- The following microsubroutine, called at some regular point in the machine cycle, illustrates this approach.
- Speed is improved only when no interrupts are pending.

Address	Next Address	Test Select	Any Select	Branch Address	CCEN	Other
INTER:	CJS	##	Any	InterI	Enable	
	(continue with normal cycle)					
INTER1	CJS	INT1	A11	Routine1	Enable	
	CJS	INT2	A11	Routine2	Enable	
	CJS	INT3	A11	Routine3	Enable	
	CJS	INT4	A11	Routine4	Enable	
	CRTN	##	##	##	Disable	



VECTORED INTERRUPT IMPLEMENTATION

- Add a priority encoder to identify which interrupt caused the request.

- The highest priority interrupt is encoded in binary.

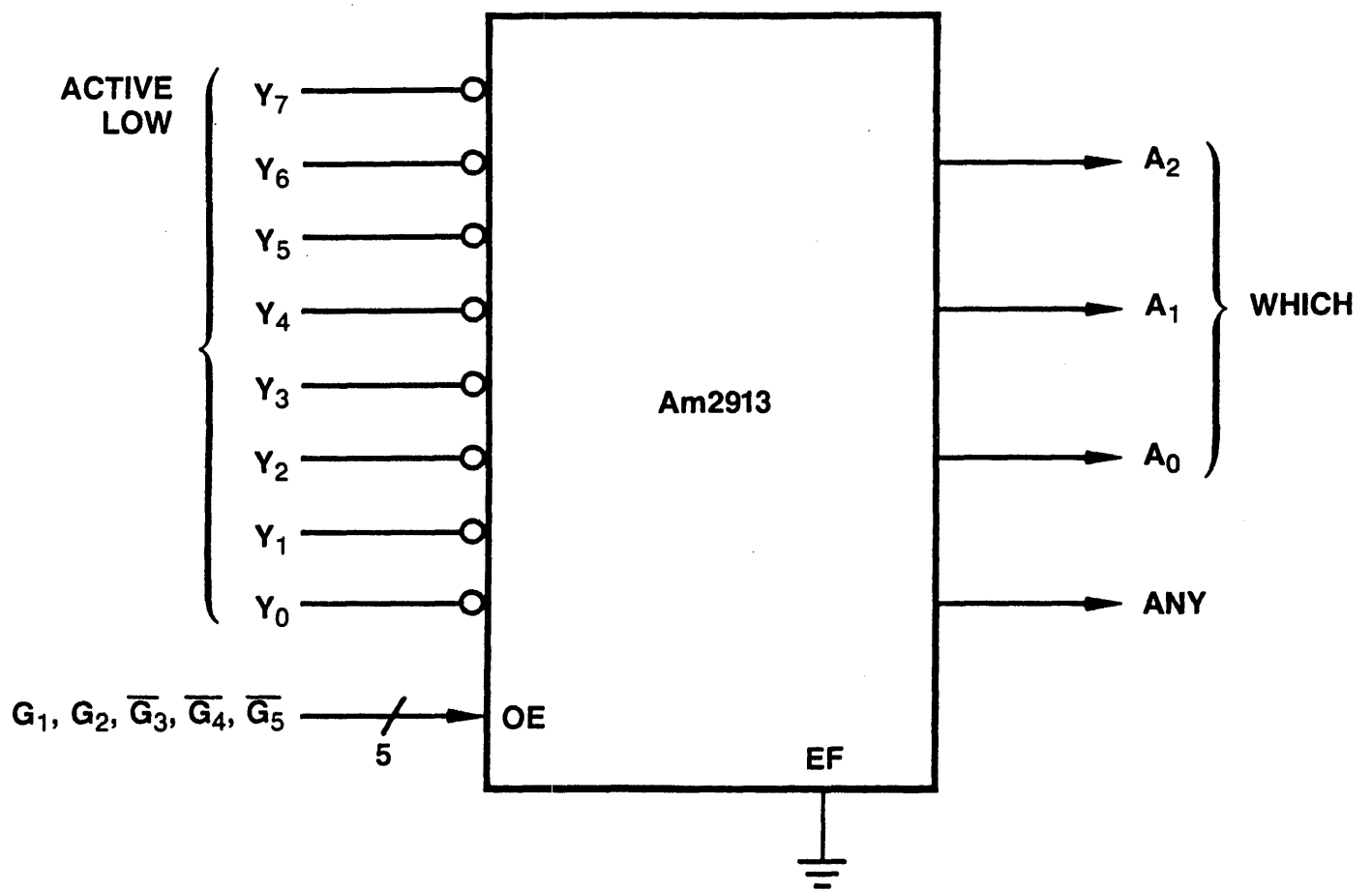
- For example:

The Am2913 provides

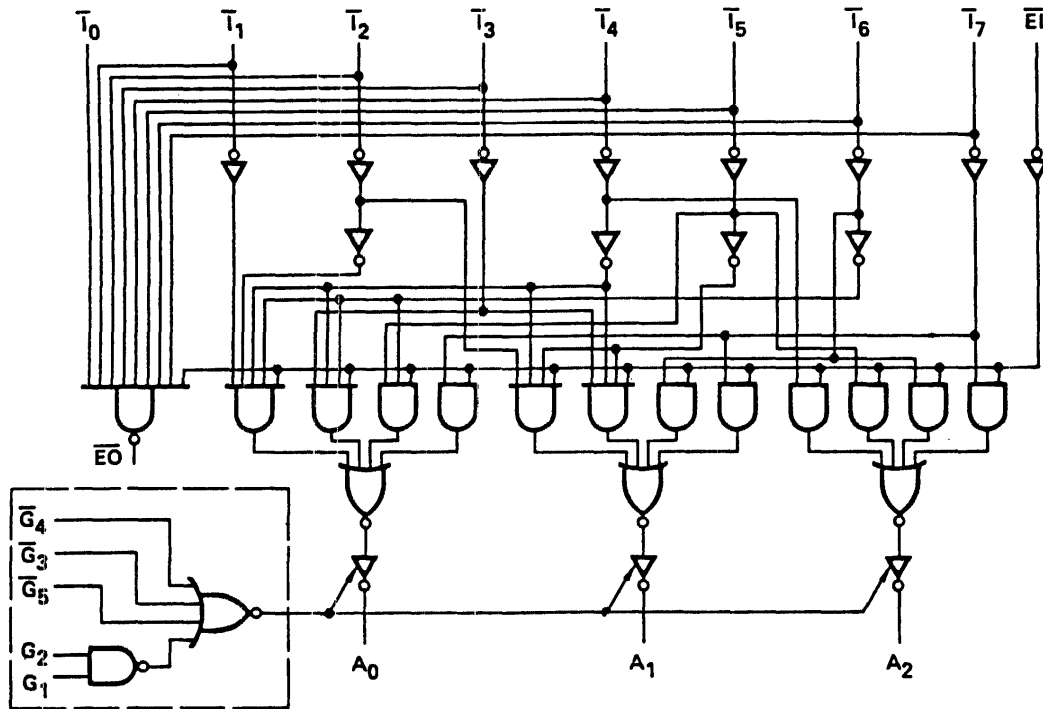
- Active low input for 8 device request lines

- Three-bit encoded output identifying highest priority request received

- "Any" request line output



Am2913 Logic Diagram



Inputs									Outputs			
$\bar{E}I$	\bar{T}_0	\bar{T}_1	\bar{T}_2	\bar{T}_3	\bar{T}_4	\bar{T}_5	\bar{T}_6	\bar{T}_7	A_0	A_1	A_2	$\bar{E}O$
H	X	X	X	X	X	X	X	X	L	L	L	H
L	H	H	H	H	H	H	H	H	L	L	L	L
L	X	X	X	X	X	X	X	L	H	H	H	H
L	X	X	X	X	X	X	L	H	L	H	H	H
L	X	X	X	X	L	H	H	H	L	L	H	H
L	X	X	L	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	L	L	L	H

G1	G2	\bar{G}_3	\bar{G}_4	\bar{G}_5	A_0	A_1	A_2
H	H	L	L	L	Enabled		
L	X	X	X	X	Z	Z	Z
X	L	X	X	X	Z	Z	Z
X	X	H	X	X	Z	Z	Z
X	X	X	H	X	Z	Z	Z
X	X	X	X	H	Z	Z	Z

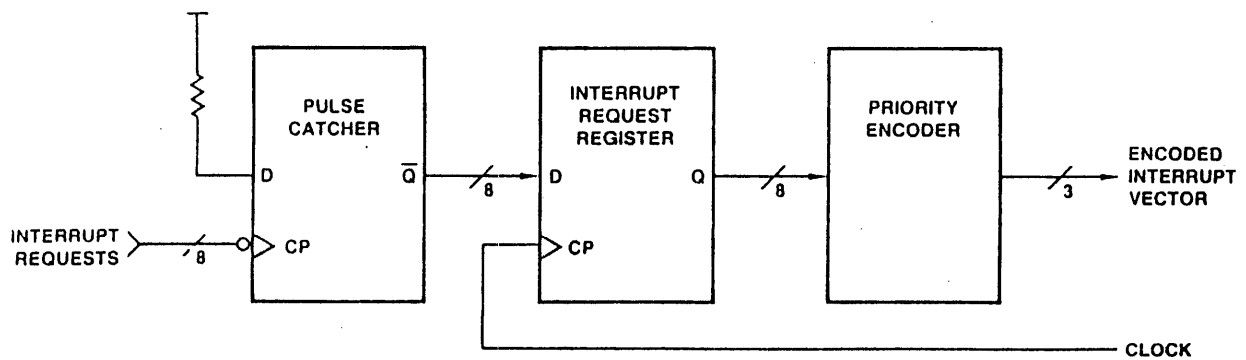
H = HIGH Voltage Level

L = LOW Voltage Level

X = Don't Care

For $G_1 = H, G_2 = H, G_3 = L, G_4 = L, G_5 = L$

Z = HIGH Impedance

POSITIONING THE PRIORITY ENCODER

VECTOR MAPPING PROM

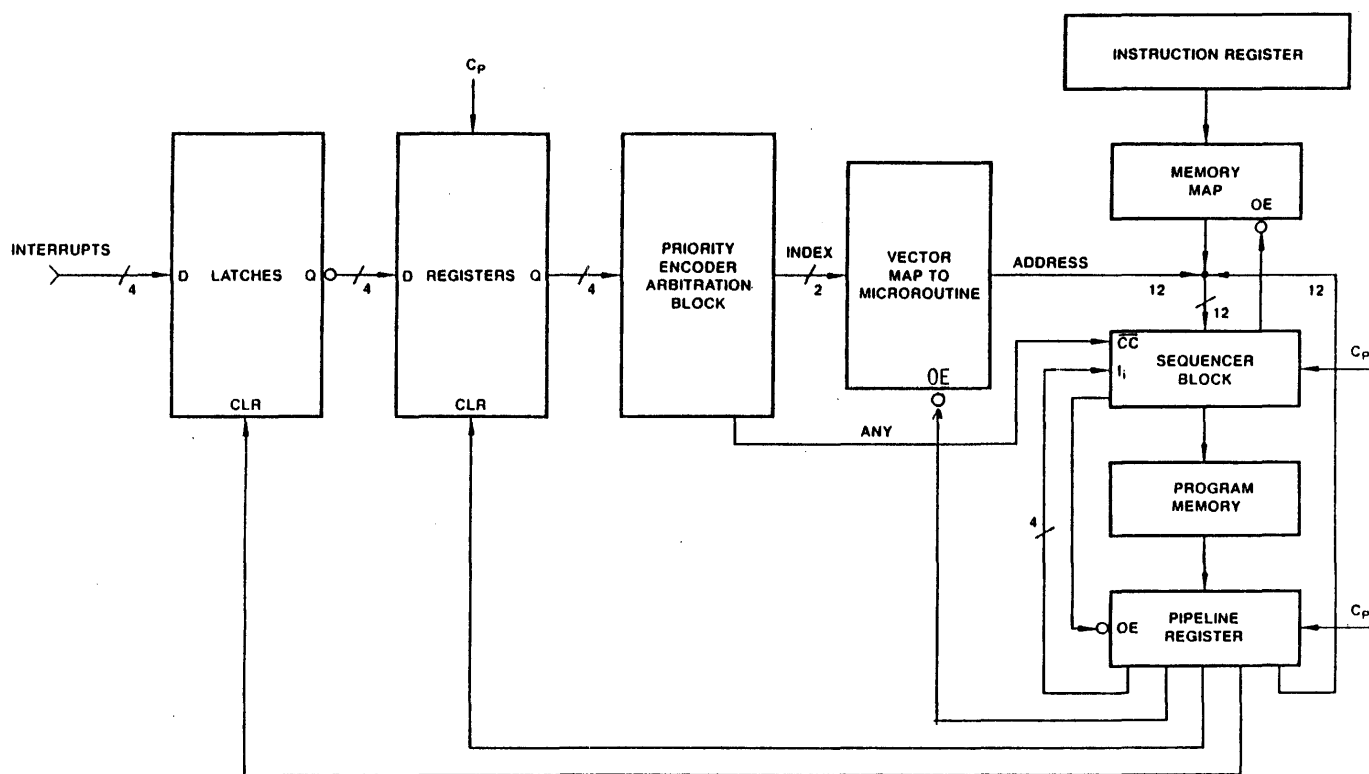
- On the detection of the "any" signal, branch to the microroutine designated by the vector.

- The same problem exists as existed with opcode decoding -- fewer bits in the vector than in the microword address.
 - Could use the lower micromemory addresses (000-111) for an interrupt jump table.

 - Could use the vector as high order bits and scatter the interrupt routines throughout micromemory.

- However, a vector mapping PROM is used with the opcode mapping PROM.

- Now on the detection of the "any" signal the microroutine can branch to any address in micromemory via the vector map.



USING PRIORITY INTERRUPTS

- Several design alternatives are available at this point.

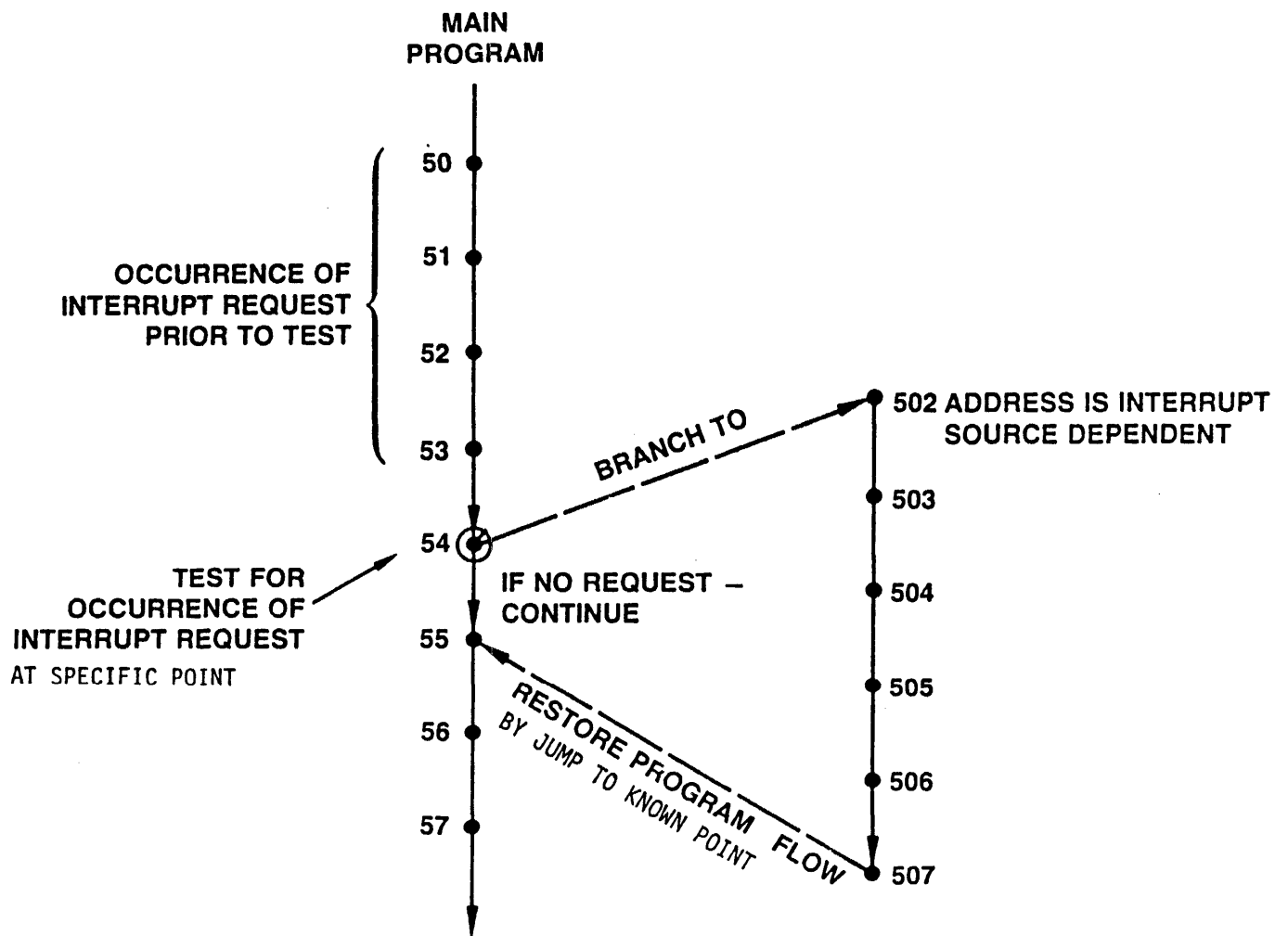
- The tri-state output enable signal for the vector mapping PROM can come from either of two places.
 1. From the OE-VECT output on the Am2910 (or from an Am29811 augmented by a decoder.)

 2. From a bit in the microword. (May be faster, but requires a wider microword.)

- The instruction for testing can be either CJS or CJV.
 1. CJS (Conditional Jump Subroutine) could perform the actual jump to the particular routine only if the vector map enable comes from the pipeline and also disabled the OE-pipeline from the Am2910.

 2. CJV (Conditional Jump Vector) could perform the actual jump to the particular routine for either source of the vector map enable signal.

CJV



VECTORED INTERRUPT IMPLEMENTATION

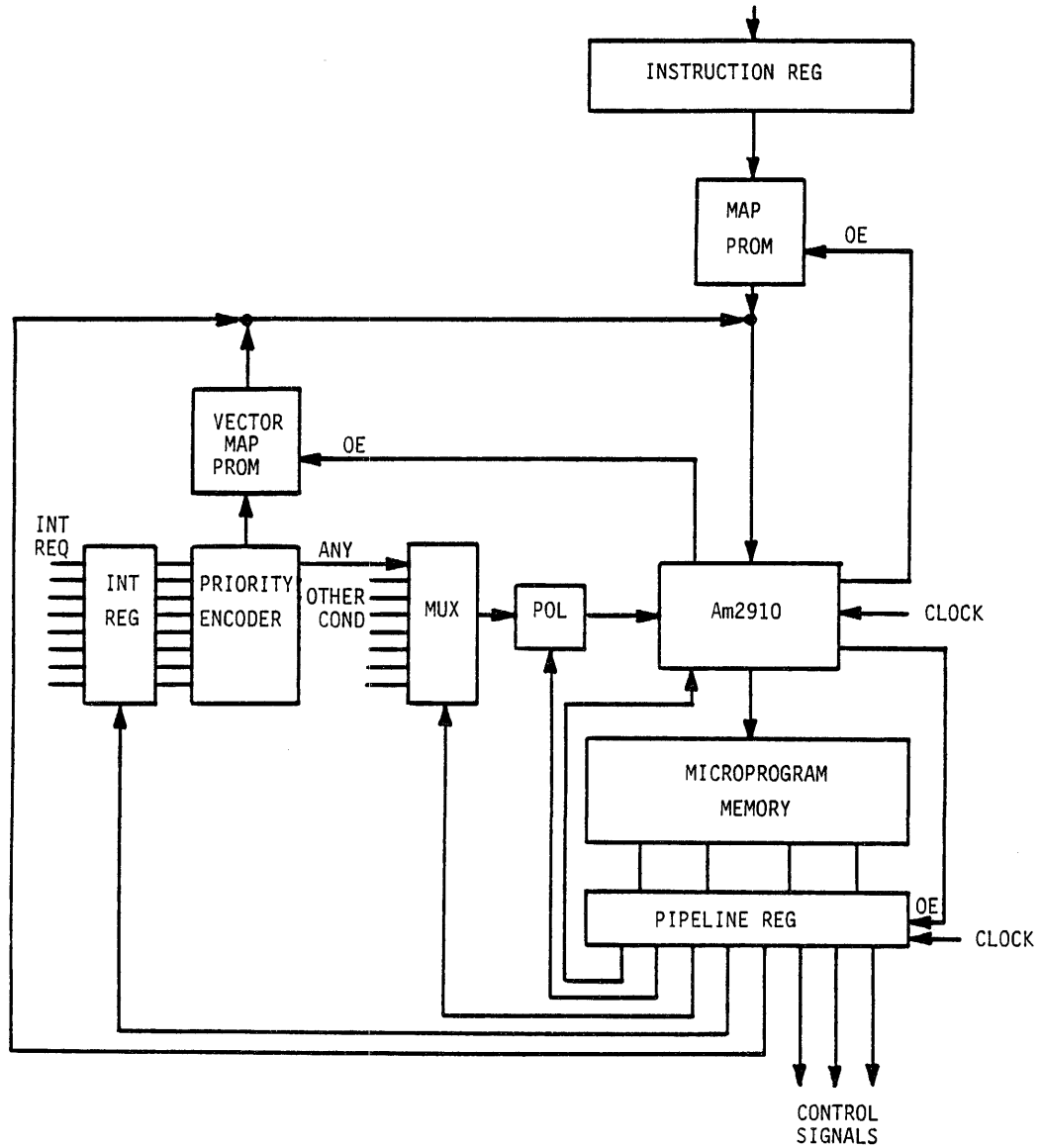
- Interrupt lines connected to priority encoder.
- The "any" output of the priority encoder connected to one input of the normal condition code MUX.
- Let the Am2910 provide the vector map enable signal.
- The following microsubroutines, called at some regular point in the machine cycle, illustrate this approach.
 1. In the first routine, CJS is used to test for "any" and jump to a common handling routine first.
 2. In the second routine, CJV is used to directly jump to the particular routine (which must save state).

EXAMPLE #1

	Next Address	Test Address Select	Branch Address	Other
INTER:	CJS	ANY	PL	
n+1		R4<--R4-R5-1		PL
n+2		R5<--R5+1		PL
n+3		R5<--R5+Q		State
	CJV	PASS	##	
RETURN:	CONT			Restore
	CONT			State
	CRTN	PASS	##	

EXAMPLE #2

	Next Address	Test Address Select	Branch Address	Other
INTER:	CJV	ANY	#	
RETURN:	(continue with normal cycle)			



SAMPLE SYSTEM FOR FIRMWARE LEVEL INTERRUPT HANDLING

2-2200

ED2900A

2-2200

The Am2914

A COMPLETE INTERRUPT CONTROLLER

REQUIREMENTS FOR INTERRUPT HANDLING

- Local storage save (save state)

- Clear interrupt
 - Clear one - last one read
 - Clear some - related to program, device
 - Clear all - warmstart

- Dynamic masking of interrupts
 - Block selected interrupts

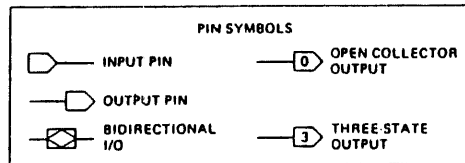
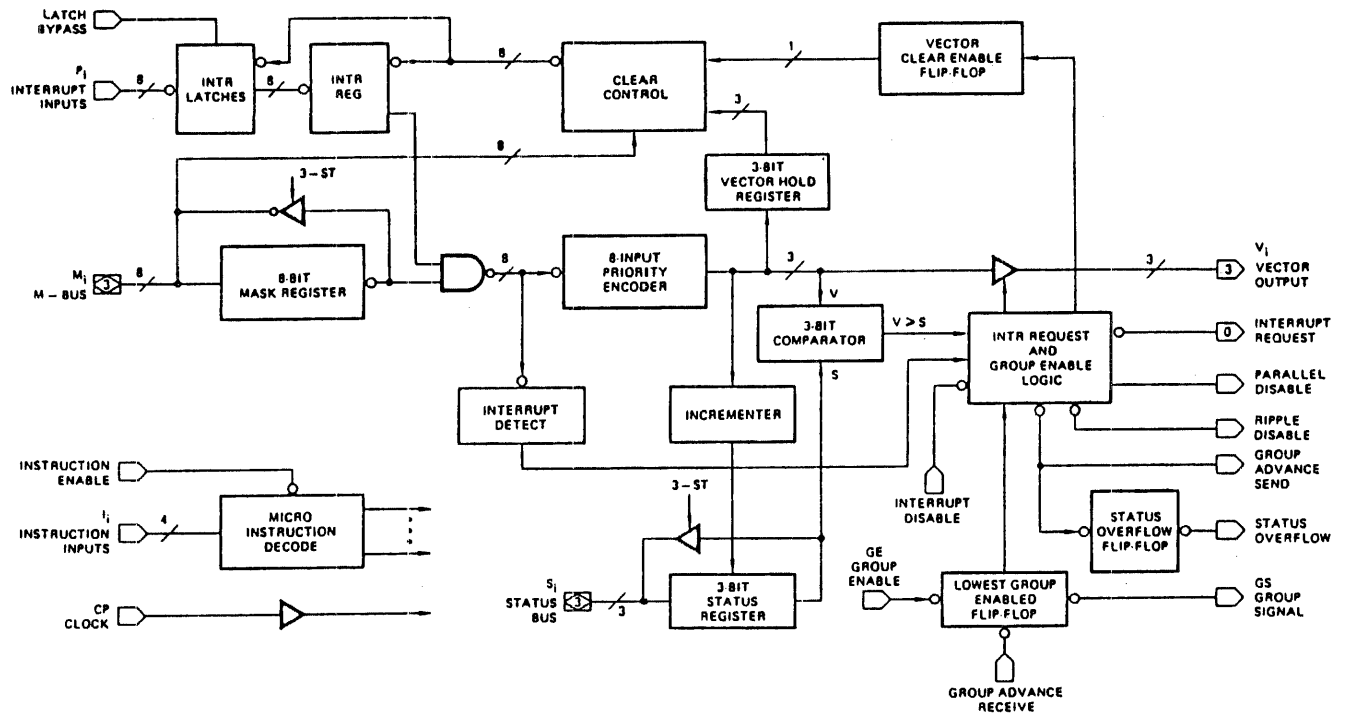
- Nesting interrupts
 - A higher priority interrupt can be acknowledged when a lower priority interrupt routine is executing.

- Status fence
 - To keep lower priority interrupt from interrupting a higher priority routine

The Am2914

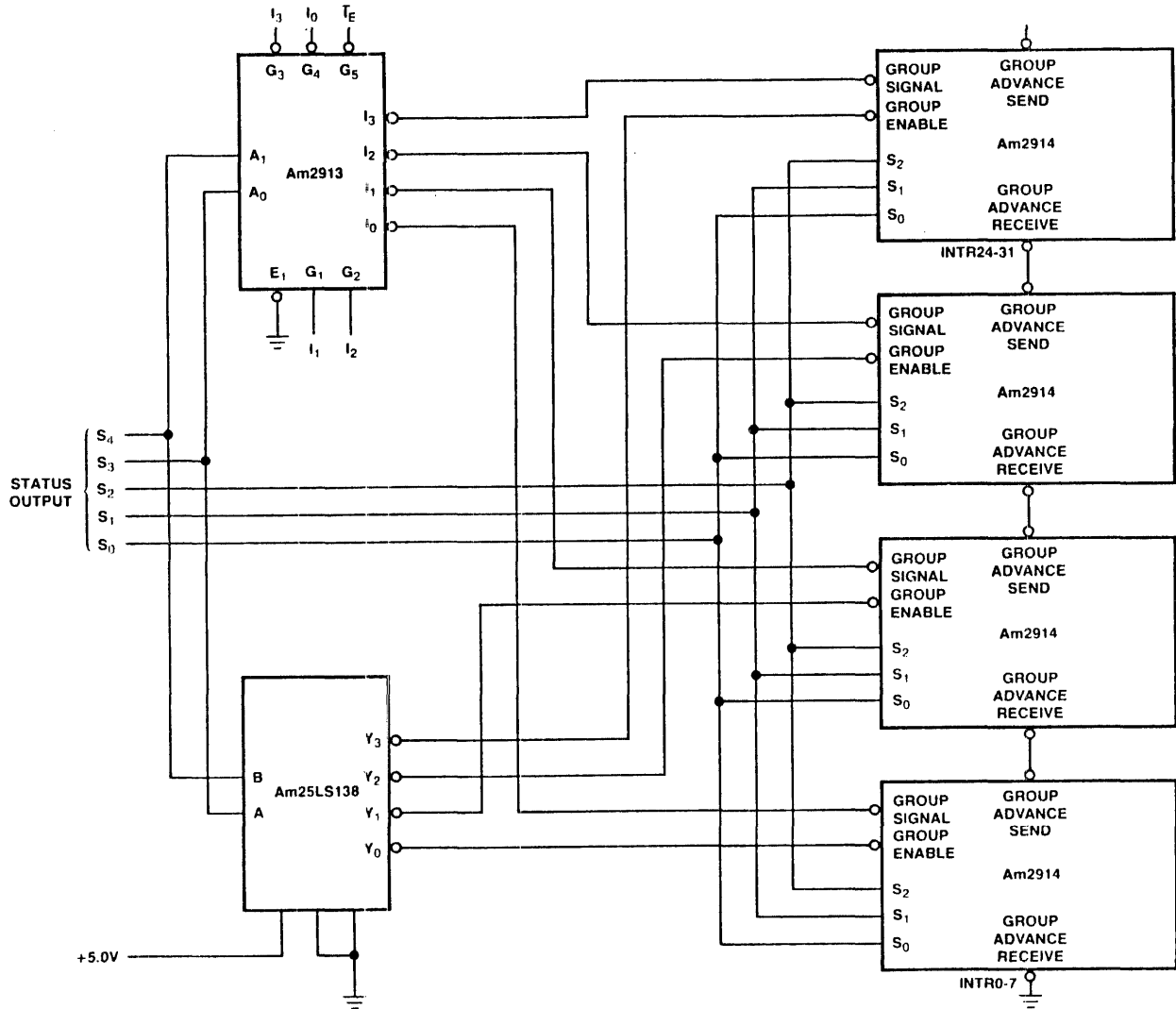
The Am2914 vectored priority interrupt controller performs the previous functions.

- Up to 8 interrupt inputs
- All 8 may be pulse or level inputs
- Produces a 3-bit vector output to address a vector map
- Contains a 3-bit fence register (status register)
- Contains an 8-bit mask register
- Both mask and status can be read from or written to
- Expandable
- Microprogrammable
- Four instruction lines plus enable



INTERCONNECTING Am2914s and Am2913s

- Multiple Am2914s can be interconnected to provide 16-, 32-, or 64-level architectures.
- Additional bits (A3, A4, ...) of vector address beyond A0, A1, and A2 must be provided using the ripple disable or the parallel disable signals to indicate the active Am2914.
- Group control pins must be connected appropriately to enable only highest priority device.
- See AMD Data Book for detailed information on interconnecting Am2914s for multi-level systems.
- The E2900B course develops the Am2914 in detail.



Am2900 FAMILY

SUPPORT CHIPS

Am2900 SUPPORT CHIPS

- Am27S26/27 registered PROMS
- Am2904 status and shift control
- Am2925 clock generator

2-2290

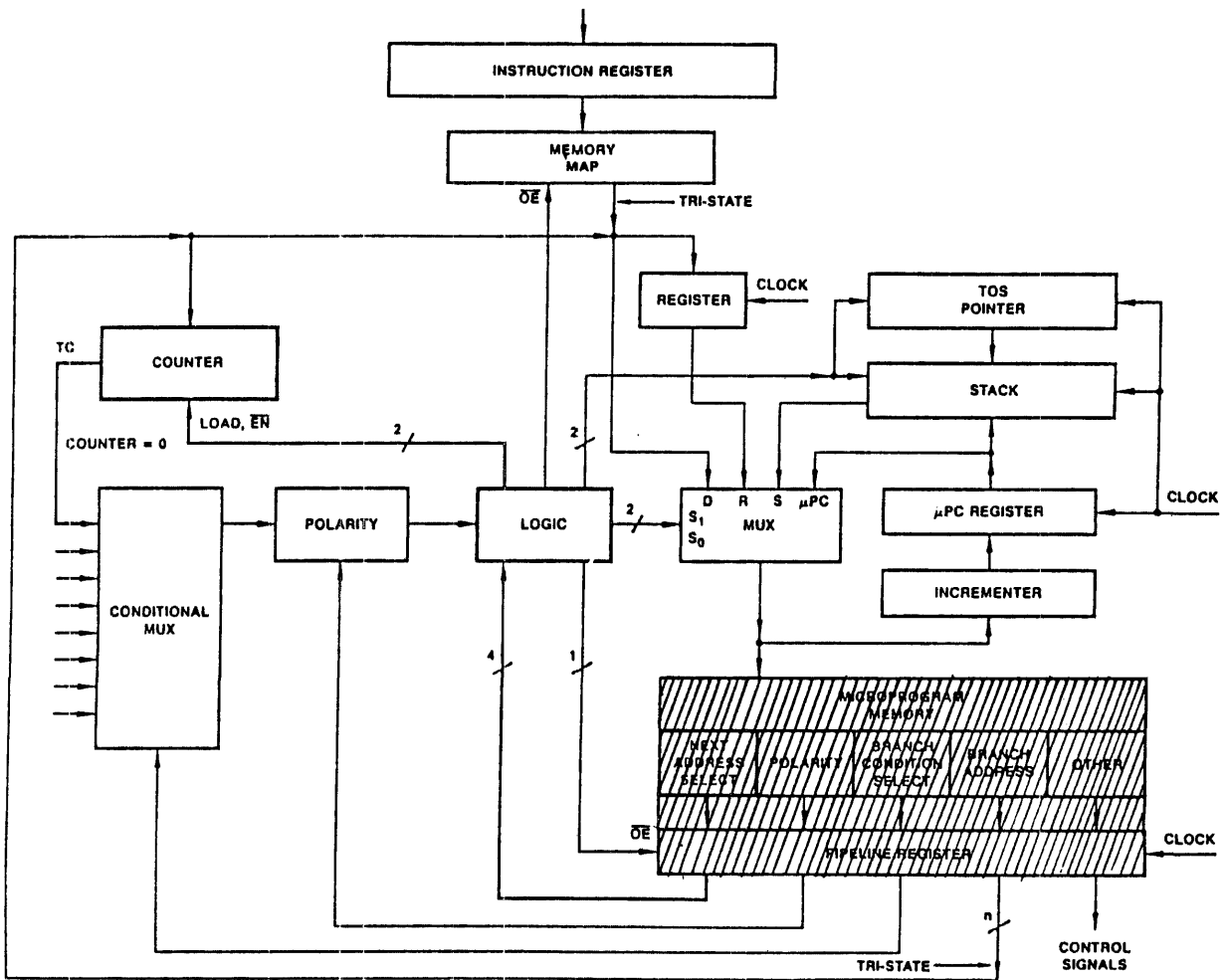
ED2900A

2-2290

REGISTERED PROMS

Am27S26/27

Am27S27 -- Am29774/Am29775



REGISTERED PROMS

- 512 X 8

- 9 address lines, 8 data lines

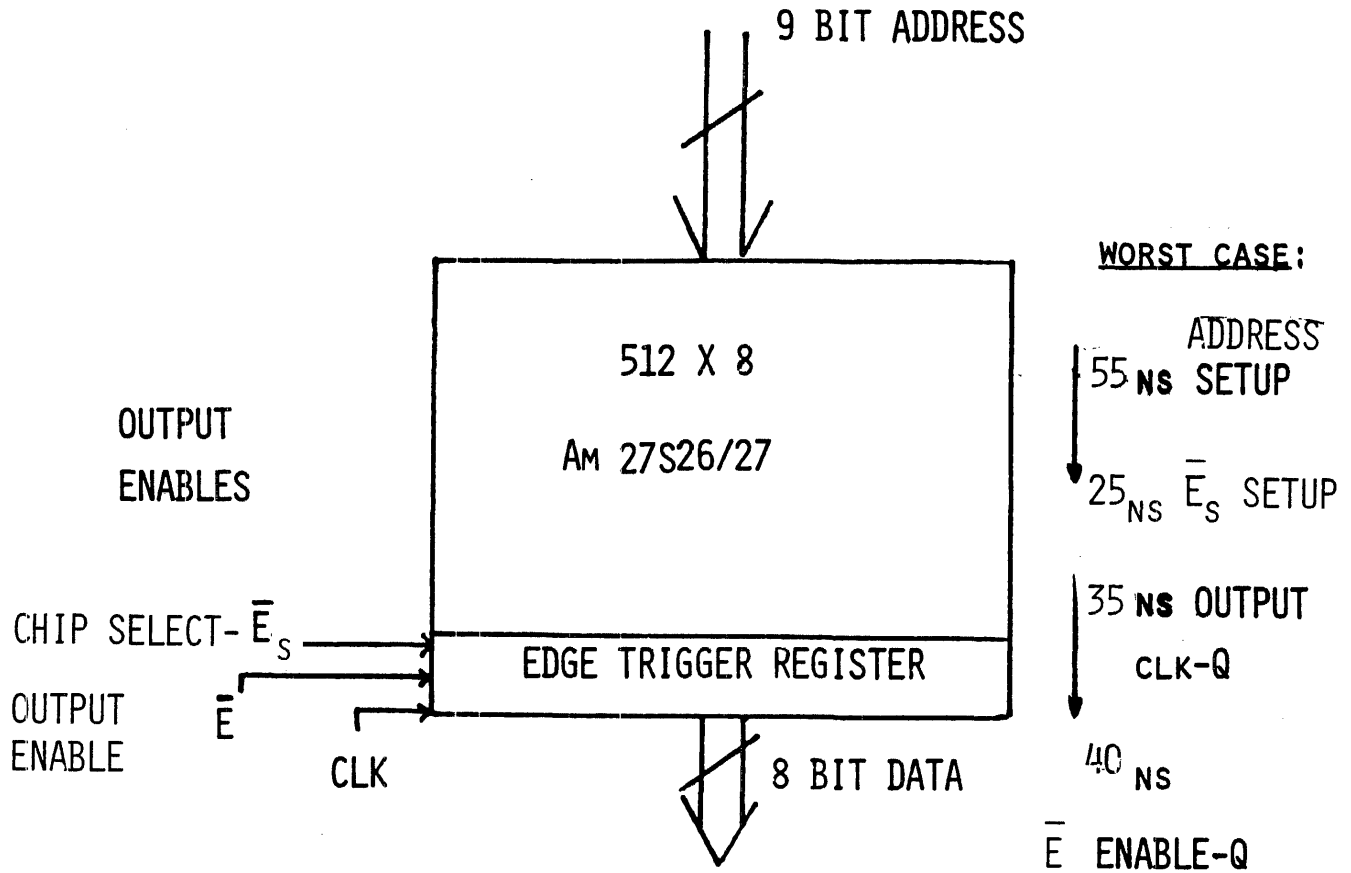
- Two enable controls

- Am27S26 - open collector output

- Am27S27 - tri-state output

- Am27S35/Am27S35A/Am27S37/Am27S37A 1K x 8

- Am27S45/Am27S45A/Am27S47/Am27S47A 2K x 8

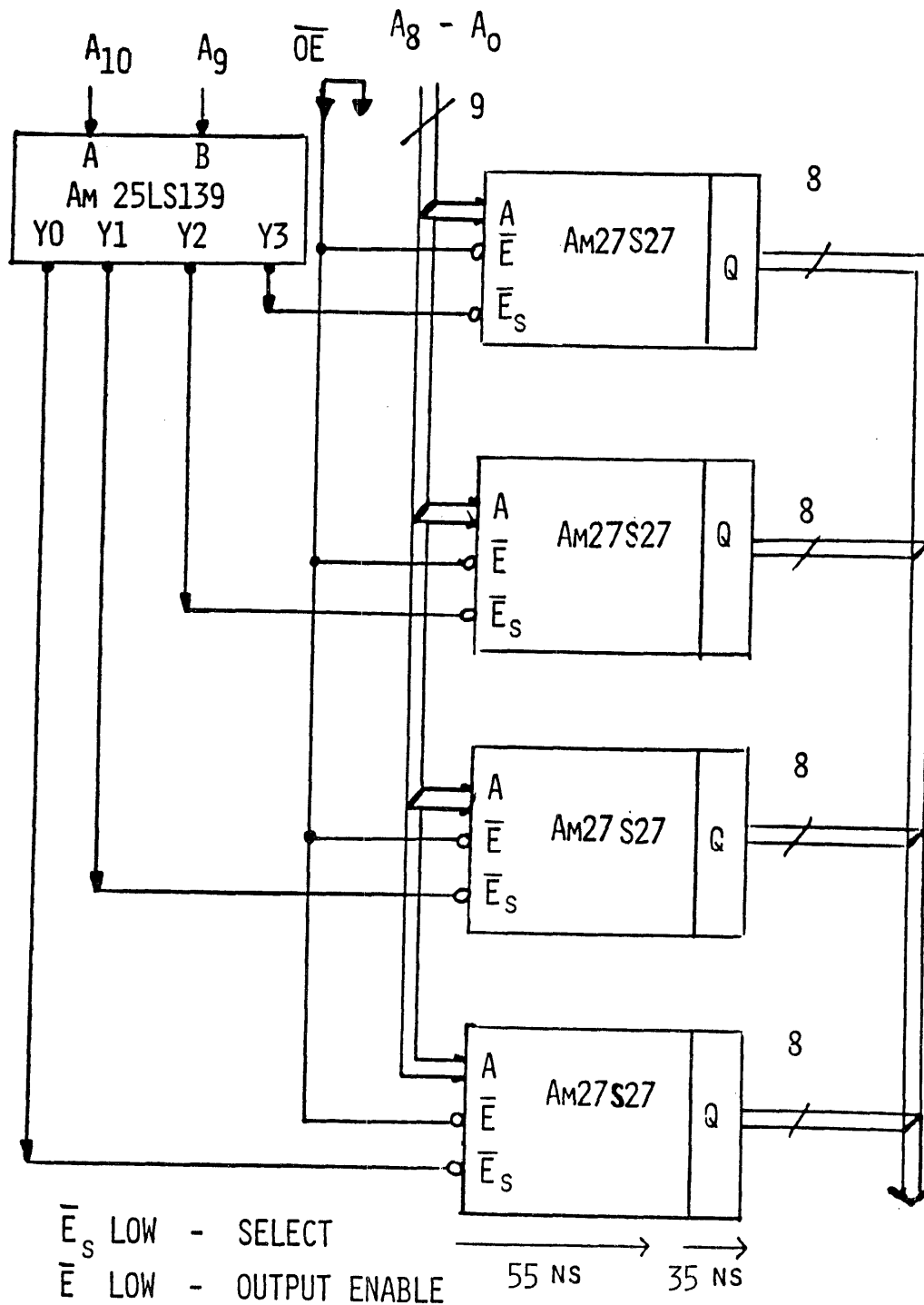


ADDRESS AND \bar{E}_S LOW - FETCH

CLK AND \bar{E} LOW - DATA OUTPUT

REGISTERED PROM

2K x 8



Optional Am27S27 Exercises

(See ED2900A Exercise and Laboratory Manual)

STATUS AND SHIFT CONTROL UNIT

Am2904

OVERVIEW

Am2904 STATUS AND SHIFT CONTROL

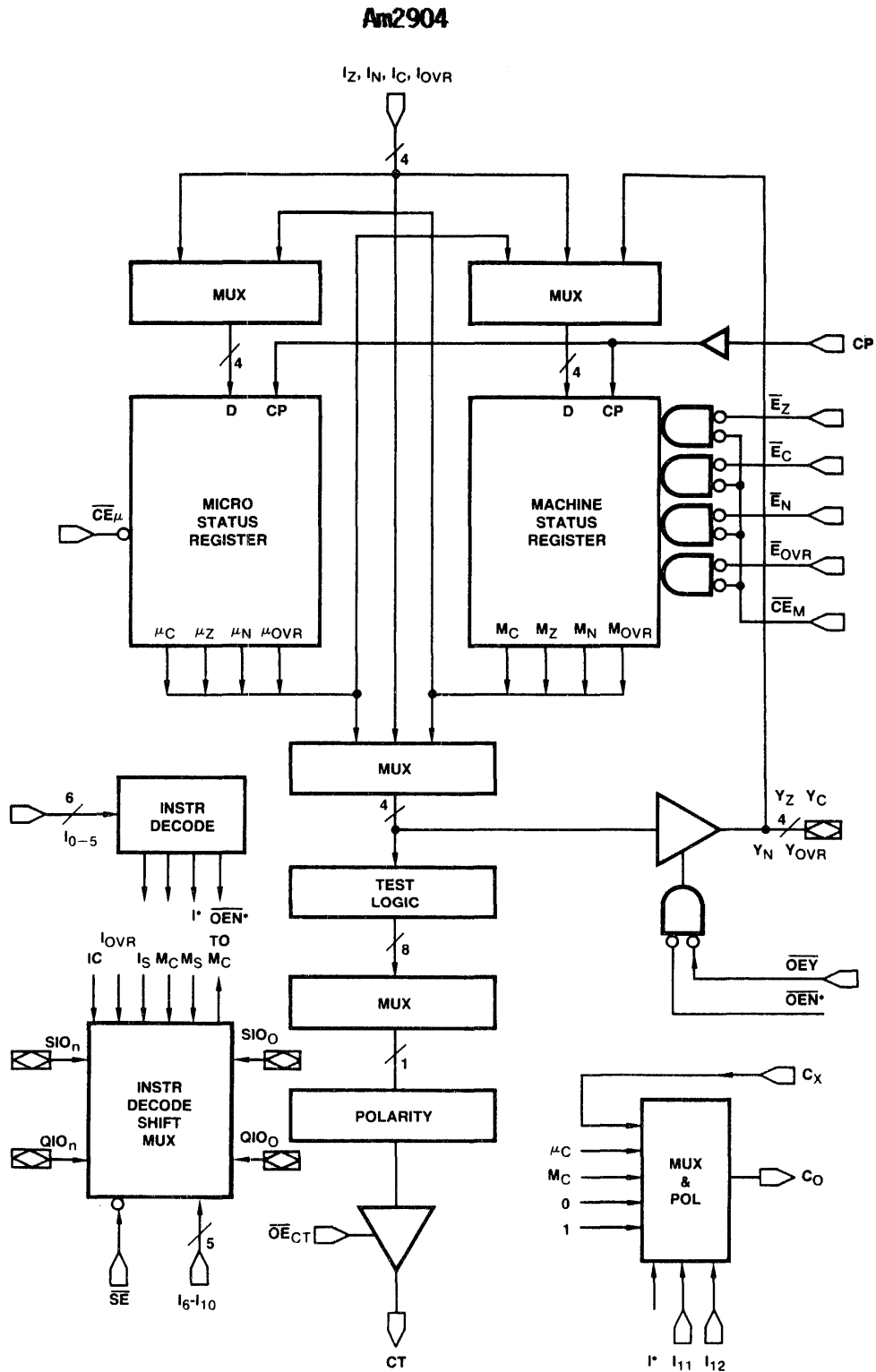
- The Am2904 was designed to replace much of the SSI/MSI which is used around the Am2901 or Am2903/29203.

- The Am2904 provides the following support on one chip:
 - Micro and macro status registers (carry, zero, sign, overflow) with ability to read and load registers.

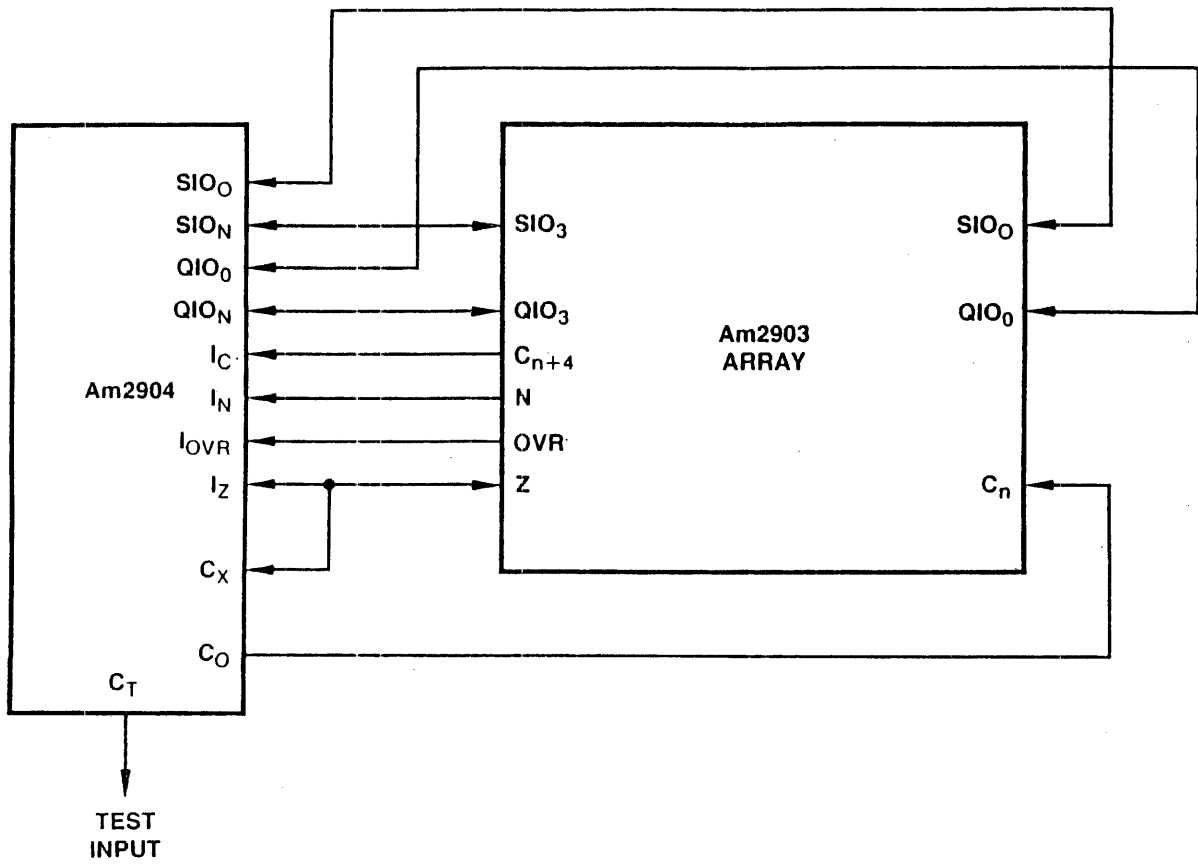
 - Shift linkage for the RAM and Q registers with 32 shift/rotate functions.

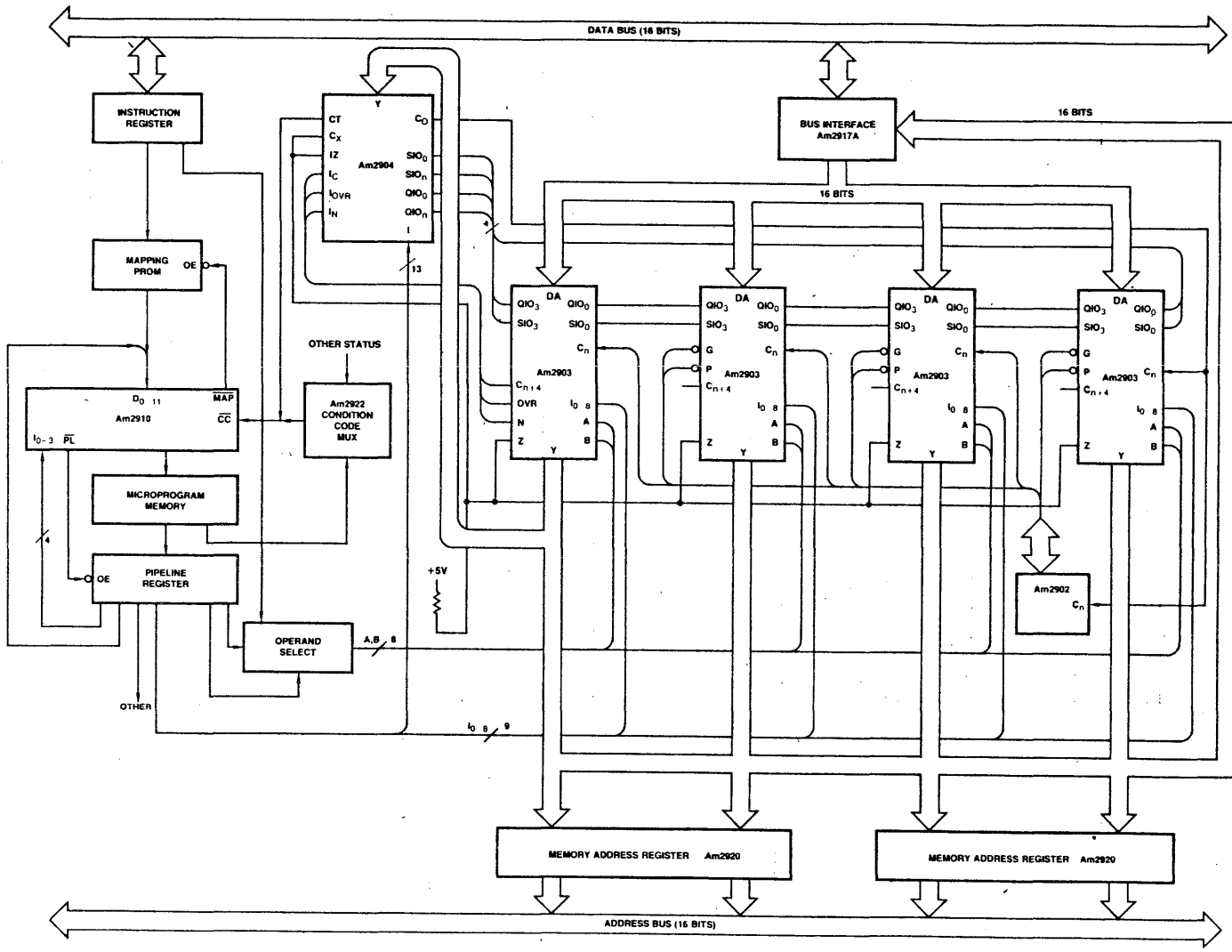
 - Carry-in select for the ALU with 7 possible sources.

 - Conditional test multiplexer to drive CC on Am2910 with 16 possible tests from 3 sources.



*INTERNAL





Typical Application of Am2904 with Am2903

MICROPROGRAM CONTROL OF Am2904

- Controlled by thirteen instruction lines, shift enable, six status enable pins, and two output enable pins (22 bits total).

- To save pins, some pins perform multiple functions.

- Status register functions use I5-I0 plus six enable pins.

- Condition code select uses I5-I0 plus one enable pin.

- Status output select uses I5-I4 plus one enable pin.

- Carry-in uses I12-I11 and I5-I1.

- Shift linkage uses I10-I6 and one enable.

- Depending on the state of the various enable pins, I5-I0 can cause up to four different functions to occur simultaneously.

- Programming is not for the faint-hearted. More detail provided in the ED2900B course.

Am2904

TABLE 7. SHIFT LINKAGE MULTIPLEXER INSTRUCTION CODES.

I ₁₀	I ₉	I ₈	I ₇	I ₆	M _C	RAM	Q	SIO ₀	SIO _n	QIO ₀	QIO _n	Loaded into M _C
0	0	0	0	0		MSB LSB	MSB LSB	Z	0	Z	0	
0	0	0	0	1		MSB LSB	MSB LSB	Z	1	Z	1	
0	0	0	1	0		MSB LSB	MSB LSB	Z	0	Z	M _N	SIO ₀
0	0	0	1	1		MSB LSB	MSB LSB	Z	1	Z	SIO ₀	
0	0	1	0	0		MSB LSB	MSB LSB	Z	M _C	Z	SIO ₀	
0	0	1	0	1		MSB LSB	MSB LSB	Z	M _N	Z	SIO ₀	
0	0	1	1	0		MSB LSB	MSB LSB	Z	0	Z	SIO ₀	
0	0	1	1	1		MSB LSB	MSB LSB	Z	0	Z	SIO ₀	QIO ₀
0	1	0	0	0		MSB LSB	MSB LSB	Z	SIO ₀	Z	QIO ₀	SIO ₀
0	1	0	0	1		MSB LSB	MSB LSB	Z	M _C	Z	QIO ₀	SIO ₀
0	1	0	1	0		MSB LSB	MSB LSB	Z	SIO ₀	Z	QIO ₀	
0	1	0	1	1		MSB LSB	MSB LSB	Z	I _C	Z	SIO ₀	
0	1	1	0	0		MSB LSB	MSB LSB	Z	M _C	Z	SIO ₀	QIO ₀
0	1	1	0	1		MSB LSB	MSB LSB	Z	QIO ₀	Z	SIO ₀	QIO ₀
0	1	1	1	0		MSB LSB	MSB LSB	Z	I _N ⊕ I _{OVR}	Z	SIO ₀	
0	1	1	1	1		MSB LSB	MSB LSB	Z	QIO ₀	Z	SIO ₀	
1	0	0	0	0		MSB LSB	MSB LSB	0	Z	0	Z	SIO _n
1	0	0	0	1		MSB LSB	MSB LSB	1	Z	1	Z	SIO _n
1	0	0	1	0		MSB LSB	MSB LSB	0	Z	0	Z	
1	0	0	1	1		MSB LSB	MSB LSB	1	Z	1	Z	
1	0	1	0	0		MSB LSB	MSB LSB	QIO _n	Z	0	Z	SIO _n
1	0	1	0	1		MSB LSB	MSB LSB	QIO _n	Z	1	Z	SIO _n
1	0	1	1	0		MSB LSB	MSB LSB	QIO _n	Z	0	Z	
1	0	1	1	1		MSB LSB	MSB LSB	QIO _n	Z	1	Z	
1	1	0	0	0		MSB LSB	MSB LSB	SIO _n	Z	QIO _n	Z	SIO _n
1	1	0	0	1		MSB LSB	MSB LSB	M _C	Z	QIO _n	Z	SIO _n
1	1	0	1	0		MSB LSB	MSB LSB	SIO _n	Z	QIO _n	Z	
1	1	0	1	1		MSB LSB	MSB LSB	M _C	Z	0	Z	
1	1	1	0	0		MSB LSB	MSB LSB	QIO _n	Z	M _C	Z	SIO _n
1	1	1	0	1		MSB LSB	MSB LSB	QIO _n	Z	SIO _n	Z	SIO _n
1	1	1	1	0		MSB LSB	MSB LSB	QIO _n	Z	M _C	Z	
1	1	1	1	1		MSB LSB	MSB LSB	QIO _n	Z	SIO _n	Z	

Notes: 1. Z = High impedance (outputs off) state.
 2. Outputs enabled and M_C loaded only if \overline{SE} is LOW.
 3. Loading of M_C from I₁₀₋₆ overrides control from I₅₋₀, \overline{CE}_M , \overline{E}_C .

TABLE 4. CONDITION CODE OUTPUT (CT) INSTRUCTION CODES.

$I_3 - 0$ HEX	I_3	I_2	I_1	I_0	$I_5 = I_4 = 0$	$I_5 = 0, I_4 = 1$	$I_5 = 1, I_4 = 0$	$I_5 = I_4 = 1$
0	0	0	0	0	$(\mu_N \oplus \mu_{OVR}) + \mu_Z$	$(\mu_N \oplus \mu_{OVR}) + \mu_Z$	$(M_N \oplus M_{OVR}) + M_Z$	$(I_N \oplus I_{OVR}) + I_Z$
1	0	0	0	1	$(\mu_N \odot \mu_{OVR}) \cdot \bar{\mu}_Z$	$(\mu_N \odot \mu_{OVR}) \cdot \bar{\mu}_Z$	$(M_N \odot M_{OVR}) \cdot \bar{M}_Z$	$(I_N \odot I_{OVR}) \cdot \bar{I}_Z$
2	0	0	1	0	$\mu_N \oplus \mu_{OVR}$	$\mu_N \oplus \mu_{OVR}$	$M_N \oplus M_{OVR}$	$I_N \oplus I_{OVR}$
3	0	0	1	1	$\mu_N \odot \mu_{OVR}$	$\mu_N \odot \mu_{OVR}$	$M_N \odot M_{OVR}$	$I_N \odot I_{OVR}$
4	0	1	0	0	μ_Z	μ_Z	M_Z	I_Z
5	0	1	0	1	$\bar{\mu}_Z$	$\bar{\mu}_Z$	\bar{M}_Z	\bar{I}_Z
6	0	1	1	0	μ_{OVR}	μ_{OVR}	M_{OVR}	I_{OVR}
7	0	1	1	1	$\bar{\mu}_{OVR}$	$\bar{\mu}_{OVR}$	\bar{M}_{OVR}	\bar{I}_{OVR}
8	1	0	0	0	$\mu_C + \mu_Z$	$\mu_C + \mu_Z$	$M_C + M_Z$	$I_C + I_Z$
9	1	0	0	1	$\bar{\mu}_C \cdot \bar{\mu}_Z$	$\bar{\mu}_C \cdot \bar{\mu}_Z$	$\bar{M}_C \cdot \bar{M}_Z$	$\bar{I}_C \cdot \bar{I}_Z$
A	1	0	1	0	μ_C	μ_C	M_C	I_C
B	1	0	1	1	$\bar{\mu}_C$	$\bar{\mu}_C$	\bar{M}_C	\bar{I}_C
C	1	1	0	0	$\bar{\mu}_C + \mu_Z$	$\bar{\mu}_C + \mu_Z$	$\bar{M}_C + M_Z$	$\bar{I}_C + I_Z$
D	1	1	0	1	$\mu_C \cdot \bar{\mu}_Z$	$\mu_C \cdot \bar{\mu}_Z$	$M_C \cdot \bar{M}_Z$	$I_C \cdot \bar{I}_Z$
E	1	1	1	0	$I_N \oplus M_N$	μ_N	M_N	I_N
F	1	1	1	1	$I_N \odot M_N$	$\bar{\mu}_N$	\bar{M}_N	\bar{I}_N

Notes: 1. \oplus Represents EXCLUSIVE-OR

2. Correct code as stated.

 \odot Represents EXCLUSIVE-NOR or coincidence.

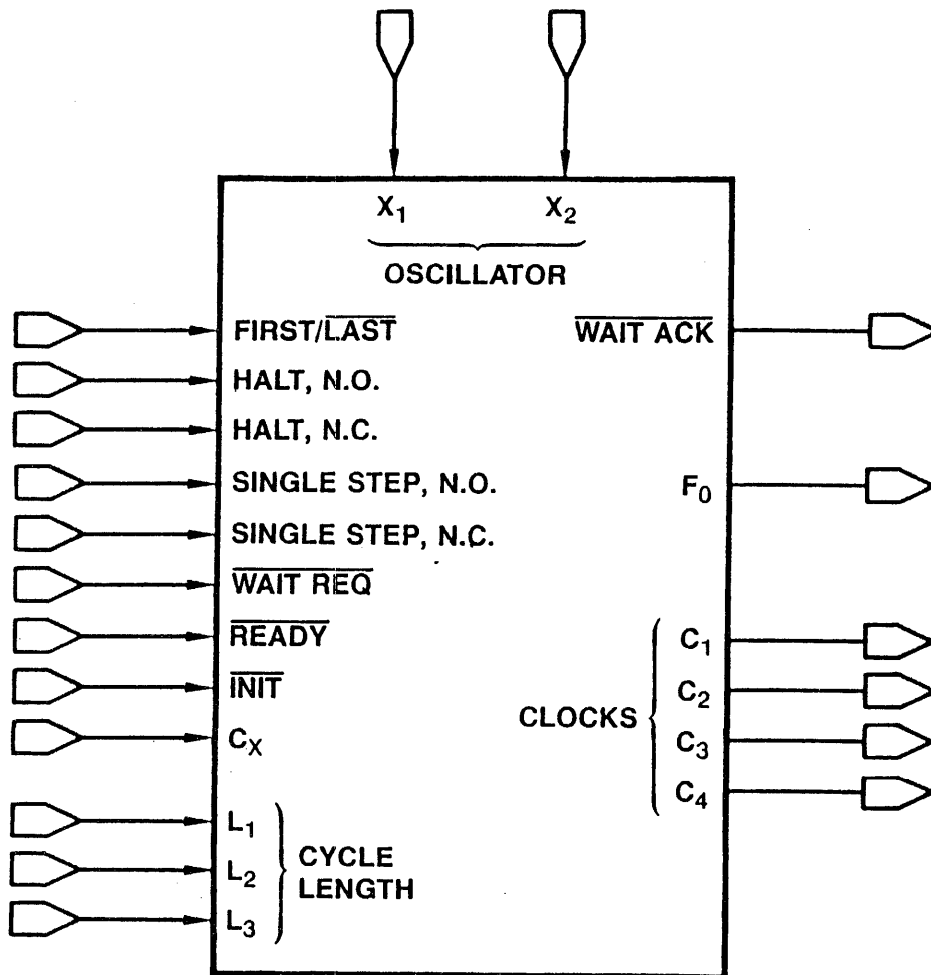
μ means micro status register
 M means macro status register
 C means carry bit
 Z means zero bit
 N means sign bit
 OVR means overflow bit

CLOCK GENERATOR

Am2925

Am2925 CLOCK GENERATOR & MICROCYCLE LENGTH CONTROLLER

- Single chip clock generator and driver
- Crystal controlled to maximum of 31 MHz
- Fundamental oscillator output available
- Four different clock output waveforms available on separate pins
- One of eight different cycle lengths may be selected by micro program control
- Clock halt, single-step, and wait controls



MICROPROGRAM CONTROL OF Am2925

- Cycle length controlled by three instruction lines.
- Other inputs would normally be provided from hardware connections instead of the microword since they concern stopping the clock and wait states.
- For three-address instructions (Am2903/29203) using output C3 as the clock and C2 for IEN, an additional field or steering bit would be needed to provide the third register address.
- The cycle-length control bits (L3, L2, L1) are latched internal to the Am2925 at the end of the microcycle. Thus no pipeline register is needed for this field.
- The cycle length value is specified in the same microword as the instruction with which it is associated. That is, the cycle length as specified stretches the current microcycle.

Am2925 Clock Waveforms

PATTERN		WAVEFORMS AND TIMING	PATTERN		WAVEFORMS AND TIMING
INPUT CODE L ₃ L ₂ L ₁			INPUT CODE L ₃ L ₂ L ₁		
F ₃ LLL			F ₇ LHH		
F ₄ LLH			F ₈ LHL		
F ₅ HLH			F ₉ HHL		
F ₆ HHH			F ₁₀ HLL		

CRYSTAL FREQ "f"	1/f ns	CYCLE LENGTH							
		000	001	101	111	011	010	110	100
16.6 MHz	60	180	240	300	360	420	480	540	600
20 "	50	150	200	250	300	350	400	450	500
25 "	40	120	160	200	240	280	320	360	400
<u>31 "</u>	32.4	<u>97</u>	<u>129</u>	<u>161</u>	<u>194</u>	<u>226</u>	<u>258</u>	<u>290</u>	<u>322</u>
30 "	33.3	100	133.3	166.7	200	233.3	266.7	300	333.3 ns

2-2490

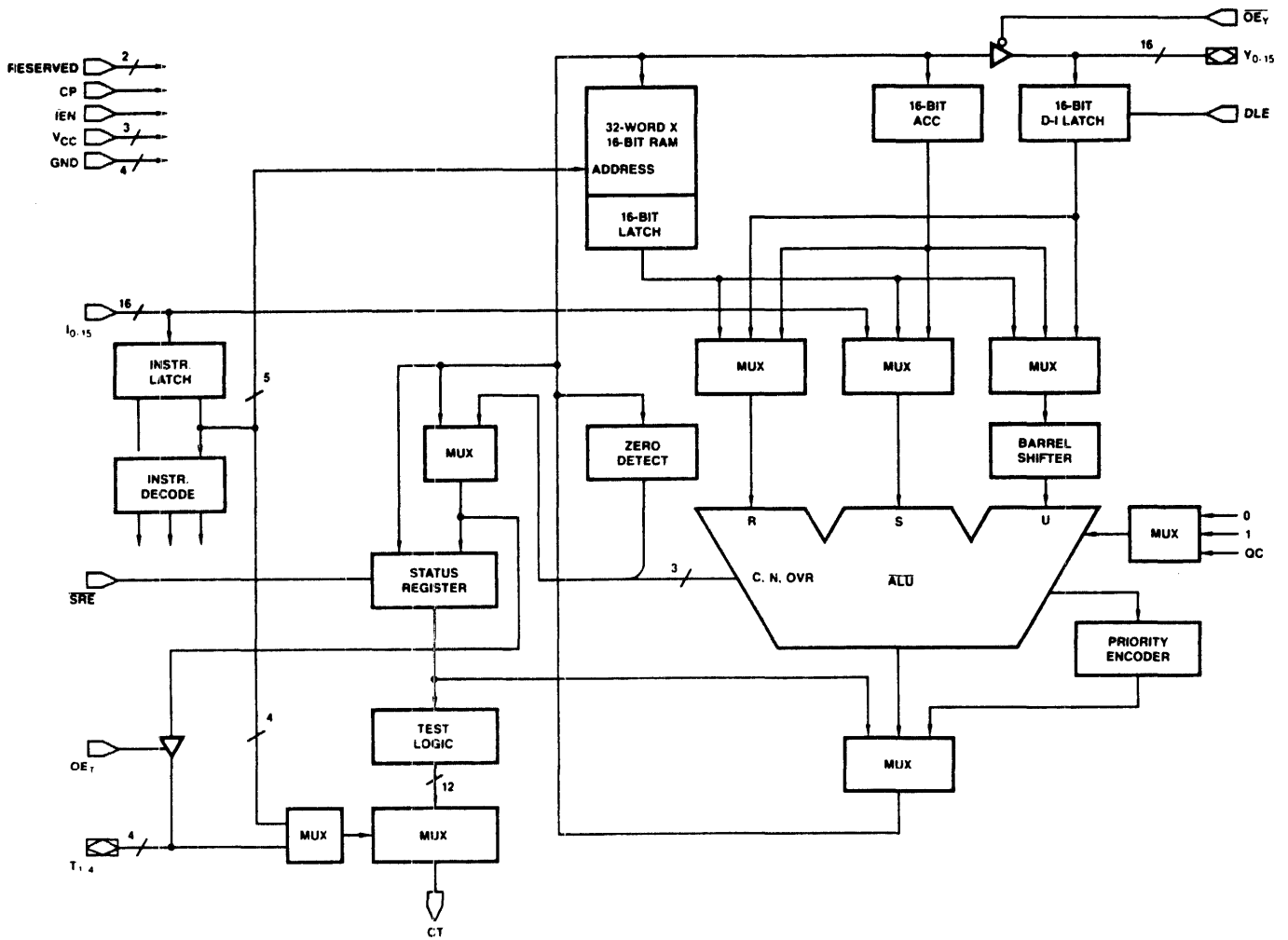
ED2900A

2-2490

Am29116

A High-Performance 16-bit Bipolar Microprocessor

Am29116 Block Diagram



Am29116, 16-bit ALU**Outstanding Features**

- 16-bit data path
 - 16-bit ALU
 - Full carry look-ahead
 - Can operate in 16-bit word mode
 - Can operate in 8-bit byte mode

- 32x16-bit RAM scratchpad on-board
 - Single port
 - With external multiplexer added may select different source and destination address for same instruction (requires timing adjust)

- 16-bit ACC

- 16-bit data latch

- 16-bit barrel shifter
 - Byte or word mode
 - Rotates 1 to 15 bits up in one cycle

Am29116 - Features (Cont'd)

- 8-bit status register

- Condition code generator/multiplexer
 - 12 different test conditions

- Immediate instruction capability
 - First microcycle - instruction latched
 - Second microcycle - immediate data available
 - Both the instruction and the immediate data are fetched via the 16 instruction lines.

- Cyclic Redundancy Check generation
 - Any CRC polynomial of 16-bits or less
 - 80% of CRC applications use 16-bit polynomials

- Powerful instruction set (see Data Book)

- Not expandable
 - Fixed data width of 16-bits
 - Fixed set of 32 16-bit registers

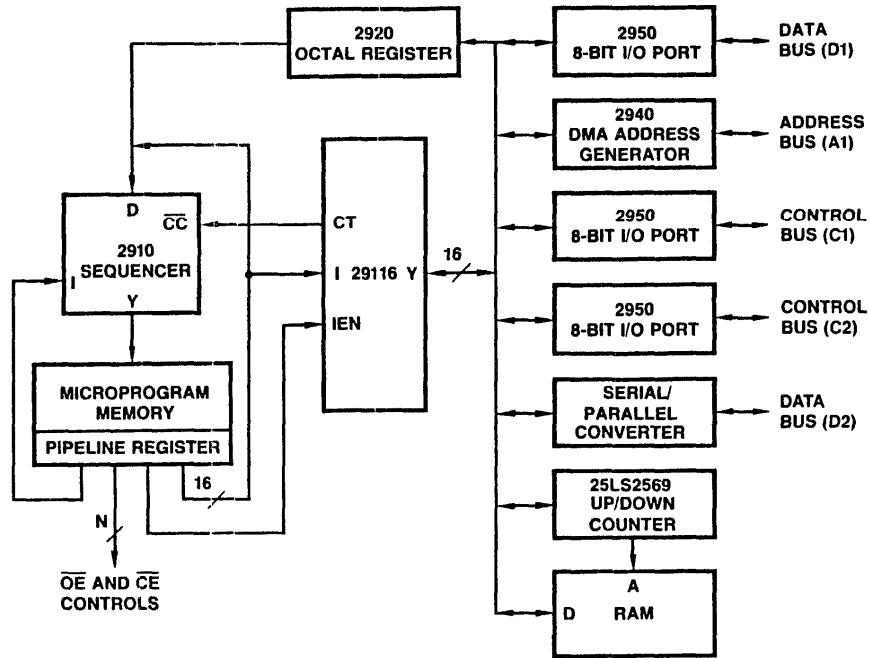
Am29116 MICROWORD FORMAT

- If it is assumed that the Am2910 is used as the sequencer for the Am29116, then the microword might look as follows:

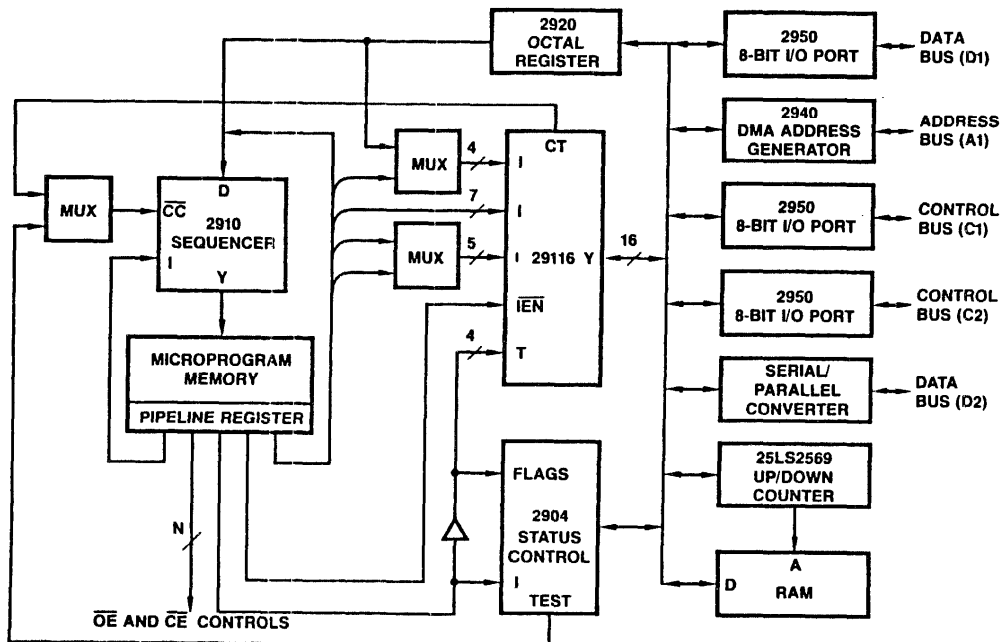
2910 INST (4)	COND MUX (3)	BRCH ADDR /COUNTER (12)	29116 INSTRUCTION/ IMMEDIATE DATA (16)	\bar{I}_{EN} (1)	\bar{OE}_Y (1)	DLE (1)	\bar{SRE} (1)	OE_T (1)	TEST INST (4)	...
---------------------	--------------------	-------------------------------	--	-----------------------	---------------------	------------	--------------------	---------------	---------------------	-----

- The conditional multiplexer may very well be replaced by an Am2904.
- The test instruction field is optional (only used when a test is to be performed during another instruction's execution).
- The Am29116 instruction field can be overlaid by the immediate data for the Am29116. (The instruction is latched on-board by the Am29116.)

**PERIPHERAL CONTROLLER,
MINIMUM PARTS CONFIGURATION**



**PERIPHERAL CONTROLLER,
MAXIMUM PERFORMANCE CONFIGURATION**



AMD FAMILIES

(see associated Data Books)

- **Analog and communications products**
- **Bipolar microprocessor logic and interface devices**
- **Bipolar/MOS memories data book**
- **MOS microprocessors and peripherals**
- **Programmable Array Logic**

THE FUTURE**In Bipolar Microprogrammable Microprocessor Logic**

- **ALU's**
32 bit

- **Sequencers**
16 bit

- **Support Devices**

VLSI

AMD SUPPORT TOOLS
FOR
Am2900 SYSTEM DEVELOPMENT

AMD SUPPORT

- META Assemblers

- AMD Customer Education courses

- Am29203 Evaluation Board

META ASSEMBLERS

MICROTEC:

- **Meta Assembler**
- Macro facility on **Macro Meta Assembler**
- PROM formatting
- Organization
 - Definition program
 - Assembler program
 - PROM formatter program
- Assembler
 - Two pass
 - Conditional assembly
- Written in ANSI FORTRAN
 - FORTRAN IV
 - 16 bits minimum
 - Disk or magnetic tape required
 - 20K macro memory minimum

AMD: (for SYS29 users)

- M29 Meta Retargetable Microcode Assembler
- Extensive microprogramming tools
- Organization
 - M29DEF, Microinstruction Definition
 - M29ASM, Microprogram Assembler
 - M29LINK, Relocating Linker
 - M29LIB, Microcode Library Manager

CUSTOMER EDUCATION COURSES

- **ED2900A "Introduction to Designing with the Am2900 Family".**
A three-day seminar on the design of microprogrammed systems using AMD's 2900 series devices, including outlines of newer, related parts, such as the Am29112 and Am29300. The Am29203 evaluation board is used for laboratory exercises.

- **ED2900B, Advanced Design with the Am2900 Family**
A four-day seminar that completes ED2900A's training on the 2900 family. A variety of detailed laboratory work with the Am29203 board provides the student with a thorough background in microcoded system design and debugging. ED2900A is a pre-requisite.

- **ED29116, Designing with the Am29116, 16-bit Microprocessor**
A two-day seminar on the design of microprogrammed systems using AMD's 16-bit, Am29116 bipolar microprocessor. Various types of design examples are discussed. Completion of ED2900A is suggested.

- **ED29500, Designing Digital Signal/Array Processors with the Am29500 Family**
A three-day seminar on the theory and design of hardware and software for microprogrammed digital signal processors using the Am29500 family. Digital filtering, array and FFT processing are discussed. The relationships among various commercially available device families are described. Completion of ED2900A is suggested.

Am29203 EVALUATION BOARD

- 16-bit computer using Am29203, Am2910, Am2904, Am2925

- Extensive monitor to allow
 - Loading microcode
 - Examining microcode
 - Executing microcode
 - Set breakpoints
 - Single-stepping microcode
 - Load and examine ALU registers
 - Load and examine the pipeline
 - Load and examine macro memory