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
  Multiplicaton 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

<table>
<thead>
<tr>
<th>OPCODE (4)</th>
<th>OPERAND ADDRESS (12)</th>
<th>MACHINE INSTRUCTION FORMAT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>S (1)</th>
<th>MAGNITUDE OF DATA (15)</th>
<th>DATA FORMAT</th>
</tr>
</thead>
</table>
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 0. If any bit = 1, TEST = 1 (OR operation). Use for conditional jump.

  e.g. Jump (CJP) if ACC = 0
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 Ø
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 ACC+MEM -> ACC):

  ADD,MEMADDR2  PC -> MAR
  FETCH INSTR
  DECODE, INCR PC
  FETCH DATA, ADD TO ACC

  -----------------------------
  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:

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>REGISTER ADDRESS 1</th>
<th>REGISTER ADDRESS 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(8)</td>
<td>(4)</td>
<td>(4)</td>
</tr>
</tbody>
</table>

SOURCE | SOURCE AND DESTINATION

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:

\[ R_A + R_B \rightarrow R_B \text{ in one microcycle} \]

- 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.).
MULTIPORT MEMORY TIMING

CP

'A' ADDRESS

STABLE

READ 'A' ADDRESS

'B' ADDRESS

STABLE

READ 'B' ADDRESS

WRITE 'B' ADDRESS

FOR:

\[ \text{REG}_B \leftarrow \text{REG}_A + \text{REG}_B \]

CYCLE LONG ENOUGH FOR READ AND WRITE

ADDRESSES MUST STAY STABLE
MULTIPORT 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>, <B>

- 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 = \langle PC \rangle
  
  ADDRESS = \langle PC \rangle + \langle BASE REGISTER \rangle
  
  ADDRESS = \langle PC \rangle + \langle OFFSET REGISTER \rangle
  
  ADDRESS = \langle PC \rangle + \langle BASE \rangle + \langle OFFSET \rangle
  
  ADDRESS = \langle R1 \rangle
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 '0'

- 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 '0'
  - 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 \textit{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"
BIT-SLICE ALU

The Am2901 Sequencer
### MICRO CODE

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>$i_2$</th>
<th>$i_1$</th>
<th>$i_0$</th>
<th>Octal Code</th>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>AQ</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>0</td>
<td>A</td>
<td>Q</td>
</tr>
<tr>
<td>AB</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>1</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>ZQ</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>2</td>
<td>O</td>
<td>Q</td>
</tr>
<tr>
<td>ZB</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>3</td>
<td>O</td>
<td>B</td>
</tr>
<tr>
<td>ZA</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>4</td>
<td>O</td>
<td>A</td>
</tr>
<tr>
<td>DA</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>5</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>DQ</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>6</td>
<td>D</td>
<td>Q</td>
</tr>
<tr>
<td>DZ</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>7</td>
<td>D</td>
<td>O</td>
</tr>
</tbody>
</table>

### SOURCE CONTROL

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>$i_5$</th>
<th>$i_4$</th>
<th>$i_3$</th>
<th>Octal Code</th>
<th>ALU Function</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>0</td>
<td>R Plus S</td>
<td>R + S</td>
</tr>
<tr>
<td>SUBR</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>1</td>
<td>S Minus R</td>
<td>S – R</td>
</tr>
<tr>
<td>SUBS</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>2</td>
<td>R Minus S</td>
<td>R – S</td>
</tr>
<tr>
<td>OR</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>3</td>
<td>R OR S</td>
<td>R ∨ S</td>
</tr>
<tr>
<td>AND</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>4</td>
<td>R AND S</td>
<td>R ∧ S</td>
</tr>
<tr>
<td>NOTRS</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>5</td>
<td>R̄ AND S</td>
<td>R̄ ∧ S</td>
</tr>
<tr>
<td>EXOR</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>6</td>
<td>R EX-OR S</td>
<td>R ⊕ S</td>
</tr>
<tr>
<td>EXNOR</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>7</td>
<td>R EX-NOR S</td>
<td>R ⊖ S</td>
</tr>
</tbody>
</table>

### FUNCTION CONTROL

Consult the AMD Data Book for discussion of tables. Note the effect of $C_{IN}$ in Am2901 function control.
<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>MICRO CODE</th>
<th>RAM FUNCTION</th>
<th>Q-REG. FUNCTION</th>
<th>OUTPUT</th>
<th>RAM SHIFTER</th>
<th>Q SHIFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I₈</td>
<td>I₇</td>
<td>I₆</td>
<td>Octal Code</td>
<td>Shift</td>
<td>Load</td>
</tr>
<tr>
<td>QREG</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>0</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>NOP</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>1</td>
<td>X</td>
<td>NONE</td>
</tr>
<tr>
<td>RAMA</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>2</td>
<td>NONE</td>
<td>F → B</td>
</tr>
<tr>
<td>RAMF</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>3</td>
<td>NONE</td>
<td>F → B</td>
</tr>
<tr>
<td>RAMQD</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>4</td>
<td>DOWN</td>
<td>F/2 → B</td>
</tr>
<tr>
<td>RAMD</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>5</td>
<td>DOWN</td>
<td>F/2 → B</td>
</tr>
<tr>
<td>RAMQU</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>6</td>
<td>UP</td>
<td>2F → B</td>
</tr>
<tr>
<td>RAMU</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>7</td>
<td>UP</td>
<td>2F → B</td>
</tr>
</tbody>
</table>

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
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.,
    
    $$G_i = A_i \cdot B_i \text{ (Boolean AND)}$$
    
    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.,
    
    $$P_i \cdot C_i \text{ where}$$
    
    $$P_i = A_i \cdot B_i + A_i \cdot B_i \text{ (Boolean Exclusive OR)}$$

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

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

\[ C_0 = C_{in} \text{ (carry-in)} \]
\[ C_1 = G_0 + P_0 C_{in} \]
\[ C_2 = G_1 + P_1 C_0 + P_0 C_{in} \]
\[ C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in} \]
\[ C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in} \]

- These carry bit equations are calculated and the "modulus two" sum (Exclusive OR) or 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):

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & 0 \\
\end{array}
\]

4-bit sum with \( C_4 = 1 \)

where \( G_0 = 1 \) \( P_0 = 0 \) \( C_1 = 1 \)
\( G_1 = 0 \) \( P_1 = 1 \) \( C_2 = 1 \)
\( G_2 = 0 \) \( P_2 = 1 \) \( C_3 = 1 \)
\( G_3 = 0 \) \( P_3 = 1 \) \( C_4 = 1 \)
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 $C_{IN}$. 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 \( G_0 = G_1 = 1 \):

  \[
  \begin{array}{cccc}
  0 & 0 & 0 & 0 \\
  0 & 1 & 1 & 1 \\
  0 & 1 & 1 & 1 \\
  \end{array}
  \quad \text{augend}
  \]

  \[
  \begin{array}{cccc}
  0 & 0 & 1 & 0 \\
  1 & 1 & 0 & 1 \\
  1 & 0 & 1 & 0 \\
  \end{array}
  \quad \text{addend}
  \]

  \[
  \begin{array}{cccc}
  0 & 1 & 0 & 0 \\
  1 & 1 & 0 & 0
  \end{array}
  \quad \text{sum}
  \]

- 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

Am2901

Am2901

Am2901
32-bit ALU

NOTE
CONNECTING THE SHIFTERS

- For rotates
  - connect RAM₀ - RAMₙ
  - connect Q₀ - 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 RAM₀ to Q₀
  - connect output F₀ to RAMₙ for down shift (sign)

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

- MUX
- MAP
- CC Am2910
  - I
  - A + 1
- MICROPROGRAM MEMORY
  - I (A + 1)
- PIPELINE REGISTER
  - I (A)
- Am2901A ALU
  - S (A)
- STATUS REGISTER
  - S (A - 1)
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.

- $I_{210}$ (source) is set to 3 to select source operands $\emptyset$ and B \quad (ZB)

- $I_{543}$ (function) is set to $\emptyset$ \quad (ADD)

- $I_{876}$ (destination) is set to 2 to select $F \rightarrow B$ and $A \rightarrow Y$ \quad (RAMA)

- $C_{in}$ is set to 1 \quad (ONE)
INCREMENT A REGISTER AND OUTPUT ITS ORIGINAL VALUE.
Am2901

BYTE SWAP:

to exchange two halves of a 16 bit word

CHANGE:

\[
\begin{array}{c|c}
B & C \\
15 & 87 \\
\end{array}
\quad \begin{array}{c|c}
F & 1 \\
0 & 87 \\
\end{array}
\]

INTO:

\[
\begin{array}{c|c}
F & 1 \\
15 & 87 \\
\end{array}
\quad \begin{array}{c|c}
B & C \\
0 & 87 \\
\end{array}
\]

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

\[ \text{RAM}_{15} \rightarrow \text{RAM}_0 \]

\[ \text{C}_{\text{out}} \rightarrow \text{C}_{\text{in}} \]

SHIFTING SEQUENCE:

<table>
<thead>
<tr>
<th>REG 15</th>
<th>1011</th>
<th>1100</th>
<th>1111</th>
<th>0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>START</td>
<td>0111</td>
<td>1001</td>
<td>1110</td>
<td>0011</td>
</tr>
<tr>
<td>2A</td>
<td>1111</td>
<td>0011</td>
<td>1100</td>
<td>0110</td>
</tr>
<tr>
<td>SHIFT</td>
<td>1110</td>
<td>0111</td>
<td>1000</td>
<td>1101</td>
</tr>
<tr>
<td>2A</td>
<td>1100</td>
<td>1111</td>
<td>0001</td>
<td>1011</td>
</tr>
<tr>
<td>SHIFT</td>
<td>1001</td>
<td>1110</td>
<td>0011</td>
<td>0111</td>
</tr>
<tr>
<td>2A</td>
<td>0011</td>
<td>1100</td>
<td>0110</td>
<td>1111</td>
</tr>
<tr>
<td>SHIFT</td>
<td>0111</td>
<td>1000</td>
<td>1100</td>
<td>1110</td>
</tr>
<tr>
<td>2A</td>
<td>1111</td>
<td>0001</td>
<td>1011</td>
<td>1100</td>
</tr>
</tbody>
</table>

ADD SHIFT REGISTER TO ITSELF
SHIFT LEFT (ROTATE)

\{ \text{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 $\rightarrow$ B
  RAM shift up (toward MSB) (RAMU)

- $C_{IN}$ is set to $C_{OUT}$

- Repeat 4 times (2 bit - rotate per cycle)

<table>
<thead>
<tr>
<th>$I_{876}$</th>
<th>$I_{543}$</th>
<th>$I_{210}$</th>
<th>$C_{IN}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>$C_{OUT}$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>FUNC</th>
<th>DEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DZ</td>
<td>ADD</td>
<td>RAMA</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"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!
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

<table>
<thead>
<tr>
<th>The Am2903/29203</th>
<th>The Am2901</th>
</tr>
</thead>
<tbody>
<tr>
<td>48 pins</td>
<td>40 pins</td>
</tr>
<tr>
<td>Higher throughput</td>
<td>Faster clock speed</td>
</tr>
<tr>
<td>Q register can shift on its own</td>
<td>Q register shifted only when RAM also shifted</td>
</tr>
<tr>
<td>Arithmetic &amp; logical shifts</td>
<td>Logical shifts only</td>
</tr>
<tr>
<td>Expandable RAM</td>
<td>RAM not designed for expansion</td>
</tr>
<tr>
<td>Two- or three-address operations</td>
<td>Two-address operation only (not designed for three)</td>
</tr>
<tr>
<td>DA$_0$-3 input</td>
<td>DA$_0$-3 input</td>
</tr>
<tr>
<td>DA$_0$-3 output (29203)</td>
<td>-</td>
</tr>
<tr>
<td>DA$_0$-3 input/output</td>
<td>-</td>
</tr>
<tr>
<td>Y$_0$-3 input/output</td>
<td>Y$_0$-3 output only</td>
</tr>
<tr>
<td>Arithmetic operation plus shift and output</td>
<td>Requires two microcycles (shift before RAM load)</td>
</tr>
<tr>
<td>Parity bit generation</td>
<td>-</td>
</tr>
<tr>
<td>Special functions</td>
<td>- (requires external assist logic)</td>
</tr>
<tr>
<td>Internal logic support</td>
<td>-</td>
</tr>
</tbody>
</table>
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
Am2903/29203 DIFFERENCES

- DA_{0-3} bidirectional on Am29203

- External $\bar{I}_E$N 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

- $\bar{O}_E$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

DATA IN
A
ADDRESS
RAM
B
ADDRESS
WRITE
ENABLE
A
DATA OUT
B
DATA OUT

E LATCH
CP
LATCH E
CP

MUX S

S MUX

ALU

ALU
SHIFTER

O
SHIFTER

O
REGISTER

ZERO
Y_0-3
Am2903 Operand Sources

<table>
<thead>
<tr>
<th>$E_A$</th>
<th>A-REGISTER</th>
<th>$O_E^B$</th>
<th>B-REGISTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>INTERNAL</td>
<td>L</td>
<td>INTERNAL</td>
</tr>
<tr>
<td>H</td>
<td>EXTERNAL</td>
<td>H</td>
<td>EXTERNAL</td>
</tr>
</tbody>
</table>

$I_0$ S-SOURCE

<table>
<thead>
<tr>
<th>$I_0$</th>
<th>S-SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>RAM/B-PORT</td>
</tr>
<tr>
<td>H</td>
<td>Q</td>
</tr>
</tbody>
</table>

ALU OPERAND SOURCES

<table>
<thead>
<tr>
<th>$E_A$</th>
<th>$I_0$</th>
<th>$O_E^B$</th>
<th>ALU Operand R</th>
<th>ALU Operand S</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>RAM Output A</td>
<td>RAM Output B</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>RAM Output A</td>
<td>DB0-3</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>X</td>
<td>RAM Output A</td>
<td>Q Register</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>DA0-3</td>
<td>RAM Output B</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>DA0-3</td>
<td>DB0-3</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>X</td>
<td>DA0-3</td>
<td>Q Register</td>
</tr>
</tbody>
</table>

$L$ = 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**

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S + R + C_N )</td>
<td>R AND S</td>
</tr>
<tr>
<td>( S - R + C_N - 1 )</td>
<td>R OR S</td>
</tr>
<tr>
<td>( R - S + C_N - 1 )</td>
<td>R NAND S</td>
</tr>
<tr>
<td>( S + C_N^* )</td>
<td>R NOR S</td>
</tr>
<tr>
<td>( \bar{S} + C_N^* )</td>
<td>R EXOR S</td>
</tr>
<tr>
<td>( R + C_N )</td>
<td>R EXNOR S</td>
</tr>
<tr>
<td>( \bar{R} + C_N )</td>
<td>( \bar{R} ) AND S</td>
</tr>
</tbody>
</table>

- For Am29203, \( I_O \) must be high, hence source must be RAMAQ or DAQ.
### Am2903 ALU Functions

<table>
<thead>
<tr>
<th>$I_4$</th>
<th>$I_3$</th>
<th>$I_2$</th>
<th>$I_1$</th>
<th>Hex Code</th>
<th>ALU Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>0</td>
<td>$I_0 = L$ Special Functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$I_0 = H$ $F_i = \text{HIGH}$</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>1</td>
<td>$F = S \text{ Minus } R \text{ Minus } 1 \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>2</td>
<td>$F = R \text{ Minus } S \text{ Minus } 1 \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>3</td>
<td>$F = R \text{ Plus } S \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>4</td>
<td>$F = S \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>5</td>
<td>$F = S' \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>6</td>
<td>$F = R \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>7</td>
<td>$F = R \text{ Plus } C_n$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>8</td>
<td>$F_i = \text{LOW}$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>9</td>
<td>$F_i = \overline{R_i} \text{ AND } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>A</td>
<td>$F_i = R_i \text{ EXCLUSIVE NOR } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>B</td>
<td>$F_i = R_i \text{ EXCLUSIVE OR } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>C</td>
<td>$F_i = R_i \text{ AND } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>D</td>
<td>$F_i = R_i \text{ NOR } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>E</td>
<td>$F_i = R_i \text{ NAND } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F</td>
<td>$F_i = R_i \text{ OR } S_i$</td>
</tr>
</tbody>
</table>

$L = \text{LOW} \quad H = \text{HIGH} \quad i = 0 \text{ to } 3$
### Am29203 ALU Functions

<table>
<thead>
<tr>
<th>$I_4$</th>
<th>$I_3$</th>
<th>$I_2$</th>
<th>$I_1$</th>
<th>$I_0$</th>
<th>ALU Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Special Functions</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$F_i = \text{HIGH}$</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>X</td>
<td>$F = S \text{ Minus } R \text{ Minus 1 Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>$F = R \text{ Minus } S \text{ Minus 1 Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>X</td>
<td>$F = R \text{ Plus } S \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>$F = S \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>X</td>
<td>$F = \overline{S} \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>Reserved Special Functions</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>$F = R \text{ Plus } C_n$</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>Reserved Special Functions</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>$F = \overline{R} \text{ Plus } C_n$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Reserved Special Functions</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>$F_i = \text{LOW}$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>X</td>
<td>$F_i = \overline{R}_i \text{ AND } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>$F_i = R_i \text{ EXCLUSIVE NOR } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>X</td>
<td>$F_i = R_i \text{ EXCLUSIVE OR } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>X</td>
<td>$F_i = R_i \text{ AND } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>X</td>
<td>$F_i = R_i \text{ NOR } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>X</td>
<td>$F_i = R_i \text{ NAND } S_i$</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>X</td>
<td>$F_i = R_i \text{ OR } S_i$</td>
</tr>
</tbody>
</table>

$L = \text{LOW} \\
H = \text{HIGH} \\
i = 0 \text{ to } 3 \\
X = \text{LOW or HIGH}$
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
<table>
<thead>
<tr>
<th>I_8</th>
<th>I_7</th>
<th>I_6</th>
<th>I_5</th>
<th>Hex Code</th>
<th>ALU Shifter Function</th>
<th>SIO_3</th>
<th>Most Sig. Slice</th>
<th>Other Slices</th>
<th>Y_3</th>
<th>Most Sig. Slice</th>
<th>Other Slices</th>
<th>Y_2</th>
<th>Most Sig. Slice</th>
<th>Other Slices</th>
<th>Y_1</th>
<th>Y_0</th>
<th>SIO_0</th>
<th>Write</th>
<th>Q Reg &amp; Shifter Function</th>
<th>QIO_3</th>
<th>QIO_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>0</td>
<td>Add F:2→Y</td>
<td>Input</td>
<td>Input</td>
<td>F_3</td>
<td>SIO_3</td>
<td>SIO_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>L</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>1</td>
<td>Log F:2→Y</td>
<td>Input</td>
<td>Input</td>
<td>SIO_3</td>
<td>SIO_3</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>L</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>2</td>
<td>Add F:2→Y</td>
<td>Input</td>
<td>Input</td>
<td>SIO_3</td>
<td>SIO_3</td>
<td>SIO_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>L</td>
<td>Log Q:2→Q Input</td>
<td>QIO_3</td>
<td>QIO_0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>3</td>
<td>Log F:2→Y</td>
<td>Input</td>
<td>Input</td>
<td>SIO_3</td>
<td>SIO_3</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>L</td>
<td>Log Q:2→Q Input</td>
<td>QIO_3</td>
<td>QIO_0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>4</td>
<td>F→Y</td>
<td>Input</td>
<td>Input</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>Parity</td>
<td>L</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>5</td>
<td>F→Y</td>
<td>Input</td>
<td>Input</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>Parity</td>
<td>H</td>
<td>Log Q:2→Q Input</td>
<td>QIO_3</td>
<td>QIO_0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>6</td>
<td>F→Y</td>
<td>Input</td>
<td>Input</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>Parity</td>
<td>H</td>
<td>F→Q</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>7</td>
<td>F→Y</td>
<td>Input</td>
<td>Input</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_2</td>
<td>F_1</td>
<td>F_0</td>
<td>Parity</td>
<td>L</td>
<td>F→Q</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>8</td>
<td>Add 2F→Y</td>
<td>F_2</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_1</td>
<td>F_0</td>
<td>SIO_0 Input</td>
<td>L</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>9</td>
<td>Log 2F→Y</td>
<td>F_2</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_1</td>
<td>F_0</td>
<td>SIO_0 Input</td>
<td>L</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>A</td>
<td>Add 2F→Y</td>
<td>F_2</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_1</td>
<td>F_0</td>
<td>SIO_0 Input</td>
<td>L</td>
<td>Log 2Q→Q Q_3 Input</td>
<td>QIO_3</td>
<td>QIO_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>B</td>
<td>Log 2F→Y</td>
<td>F_2</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_1</td>
<td>F_0</td>
<td>SIO_0 Input</td>
<td>L</td>
<td>Log 2Q→Q Q_3 Input</td>
<td>QIO_3</td>
<td>QIO_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>C</td>
<td>F→Y</td>
<td>F_3</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_1</td>
<td>F_0</td>
<td>Hi-Z</td>
<td>H</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>D</td>
<td>F→Y</td>
<td>F_3</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_1</td>
<td>F_0</td>
<td>Hi-Z</td>
<td>H</td>
<td>Log 2Q→Q Q_3 Input</td>
<td>QIO_3</td>
<td>QIO_0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>E</td>
<td>SiO_0→Y_0, Y_1, Y_2, Y_3</td>
<td>SiO_0</td>
<td>SiO_0</td>
<td>SiO_0</td>
<td>SiO_0</td>
<td>SiO_0</td>
<td>SiO_0</td>
<td>SiO_0</td>
<td>SiO_0 Input</td>
<td>L</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>F</td>
<td>F→Y</td>
<td>F_3</td>
<td>F_3</td>
<td>F_3</td>
<td>F_2</td>
<td>F_1</td>
<td>F_1</td>
<td>F_0</td>
<td>Hi-Z</td>
<td>L</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Parity = F_3 ⊕ F_2 ⊕ F_1 ⊕ F_0 ⊕ SIO_3

L = LOW

Hi-Z = High Impedance

H = HIGH

Exclusive OR
LOGICAL VERSUS ARITHMETIC SHIFT:

Am2903 Arithmetic Shift Path

Am2903 Logical Shift Path

MOST SIGNIFICANT SLICE

LEAST SIGNIFICANT OR INTERMEDIATE SLICE

ALL SLICE POSITIONS
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

<table>
<thead>
<tr>
<th>(Hex) $I_{BH}/I_{BO}$</th>
<th>$I_{DH}/I_{DO}$</th>
<th>Special Function</th>
<th>ALU Function</th>
<th>ALU Shifter Function</th>
<th>$SIO_3$</th>
<th>Q Reg &amp; Shifter Function</th>
<th>$QIO_3$</th>
<th>$QIO_2$</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>F_0</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>1</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>Z</td>
<td>Input</td>
<td>F_0</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>F_0</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>3</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Parity</td>
<td>Z</td>
<td>Z</td>
<td>L</td>
</tr>
<tr>
<td>4</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>Parity</td>
<td>Z</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>Parity</td>
<td>Z</td>
<td>L</td>
</tr>
<tr>
<td>6</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>Q_0</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>Z</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>Z</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>L</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Z</td>
<td>Input</td>
<td>Z</td>
<td>Z</td>
<td>L</td>
</tr>
</tbody>
</table>

### Notes:
1. At the most significant slice only, the $C_9$ signal is internally gated to the $Y_3$ output.
2. At the most significant slice only, $F_3 \lor OVR$ is internally gated to the $Y_3$ output.
3. At the most significant slice only, $S_3 \oplus F_3$ is generated at the $Y_3$ output.
4. On each slice, $F = S$ if magnitude of $S_0 \cdot 3$ is less than 8 and $F = S$ plus 3 if magnitude of $S_0 \cdot 3$ is 8 or greater.
5. On each slice, $F = S$ if magnitude of $S_0 \cdot 3$ is less than 5 and $F = S$ plus 3 if magnitude of $S_0 \cdot 3$ 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.

- $L = \text{LOW}$
- $H = \text{HIGH}$
- $X = \text{Don't Care}$
- $Hi-Z = \text{High Impedance}$
- $\lor = \text{Exclusive OR}$
- $\equiv = \text{Parity}$

---

**ADVANCED MICRO DEVICES**
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 Control

(SOURCE) A → RAM → B

MUX

ALU

SEL

B (SOURCE)

C (DEST)

SWITCHED AT 50% OF CLOCK PERIOD
Am2903 Three-Address Operation

- CLOCK
- INST ENABLE
- A & B ADDRESS
- C ADDRESS

\( \mu \) CYCLE

25%
INTERCONNECTING THE SLICES

Am2903/Am29203
16-Bit CPU with Ripple Carry.

NOTE ISOLATING RESISTORS
16-BIT CPU WITH LOOKAHEAD 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 1015
- $\bar{E}_A\ I_0\ \bar{O}_B$ (Source) are set to $\bar{0}\bar{0}\bar{0}$ (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
- $\bar{O}_B = \bar{0}$ places the original value of $R_{15}$ at DBI/O
- $I_{8765}$ (Destination is set to F and $\bar{O}_Y$ to Low to select F--$\rightarrow$Y and to write the new value F into the RAM. (RAM or RAMEXT)

\[
\begin{array}{cccccc}
I_{8765} & I_{4321} & \bar{E}_A & I_0 & \bar{O}_B & C_{IN} & \bar{O}_Y \\
F & 6 & \bar{0} & H & L \\
\text{RAM} & \text{INCRR} & \text{RAMAB}
\end{array}
\]
Notes: 1. \(DA_0\ldots_3\) is input only on Am2903, but is I/O port on Am29203.
2. On Am29203, zero logic is connected to \(Y\), after the OE\(_y\) buffer.
Am2903

BYTE SWAP

To exchange two halves of a 16-bit word

Change:

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>F</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>87</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Into:

<table>
<thead>
<tr>
<th>F</th>
<th>1</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>87</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- Assume again Register 15.
  (Number chosen for no particular reason)
Am2903

- Address A and Address B are both set to 15
- \( E_A \bar{I} \bar{O}_B \) set to \( \bar{0} \) 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
- \( WE \) Low
- \( OE_Y \) Low

Repeat for total of four times, same as for Am2901

<table>
<thead>
<tr>
<th>SOURCE</th>
<th>FUNC</th>
<th>DEST</th>
<th>( C_{IN} )</th>
<th>2904</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMAB</td>
<td>ADD</td>
<td>RAMUPL</td>
<td>( C_{OUT} )</td>
<td>S10_{15} to S10_{0}</td>
</tr>
</tbody>
</table>
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
Byte Swap requires an enable for the tri-state buffer drivers

The code is:

<table>
<thead>
<tr>
<th>addr</th>
<th>2910</th>
<th>INST</th>
<th>COND</th>
<th>MUX</th>
<th>BRCH</th>
<th>CNTR</th>
<th>SRCE</th>
<th>FUNC</th>
<th>DEST</th>
<th>RA ADDR</th>
<th>RB ADDR</th>
<th>Cin</th>
<th>OEy</th>
<th>Ien</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>CONT</td>
<td>#</td>
<td>#</td>
<td>DARAMB</td>
<td>INCRR</td>
<td>RAM</td>
<td>#</td>
<td>Rb</td>
<td>LOW</td>
<td>EN</td>
<td>EN</td>
<td>EN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OR

<table>
<thead>
<tr>
<th>addr</th>
<th>2910</th>
<th>INST</th>
<th>COND</th>
<th>MUX</th>
<th>BRCH</th>
<th>CNTR</th>
<th>SRCE</th>
<th>FUNC</th>
<th>DEST</th>
<th>RA ADDR</th>
<th>RB ADDR</th>
<th>Cin</th>
<th>OEy</th>
<th>Ien</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>CONT</td>
<td>#</td>
<td>#</td>
<td>RAMAB</td>
<td>#</td>
<td>RAM</td>
<td>#</td>
<td>Rb</td>
<td>#</td>
<td>DIS</td>
<td>EN</td>
<td>EN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
THE SPECIAL FUNCTIONS

OF

THE Am2903/Am29203
Am2903/Am29203 SPECIAL FUNCTIONS AND FEATURES

- Parity
- Sign extension
- Sign magnitude \(\leftrightarrow\) 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: \( l_0 = l_1 = l_2 = l_3 = l_4 = \text{LOW}, \text{IEN} = \text{LOW} \)

<table>
<thead>
<tr>
<th>( l_8 \ l_7 \ l_6 \ l_5 )</th>
<th>Hex Code</th>
<th>Available On</th>
<th>Special Function</th>
<th>ALU Function</th>
<th>ALU Shifter Function</th>
<th>Si03</th>
<th>Si02</th>
<th>Si01</th>
<th>Si00</th>
<th>Si00</th>
<th>Si01</th>
<th>Si02</th>
<th>Si03</th>
<th>Q Reg &amp; Shifter Function</th>
<th>QIO3</th>
<th>QIO0</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>L L L L 0</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Unsigned Multiply</td>
<td>( F = S + C_n ) ( Z = 1 ) ( F = R + S + C_p ) ( Z = 1 )</td>
<td>Log F2 ( = Y ) (Note 1)</td>
<td>Hi-Z</td>
<td>Input</td>
<td>( F_0 )</td>
<td>Log Q2 ( = Q )</td>
<td>Input</td>
<td>( Q_0 )</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L L L H 1</td>
<td>Am29203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L L H L 2</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Two's Complement Multiply</td>
<td>( F = S + C_n ) ( Z = 1 ) ( F = R + S + C_p ) ( Z = 1 )</td>
<td>Log F2 ( = Y ) (Note 2)</td>
<td>Hi-Z</td>
<td>Input</td>
<td>( F_0 )</td>
<td>Log Q2 ( = Q )</td>
<td>Input</td>
<td>( Q_0 )</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L L H H 3</td>
<td>Am29203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L H L L 4</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Increment by One or Two</td>
<td>( F = S + 1 + C_n )</td>
<td>( F = Y )</td>
<td>Input</td>
<td>Input</td>
<td>Party</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L H L H 5</td>
<td>Am29203</td>
<td></td>
<td>Sign Magnitude Two's Complement</td>
<td>( F = S + C_n ) ( Z = 1 ) ( F = R + S + C_p ) ( Z = 1 )</td>
<td>( F = Y ) (Note 3)</td>
<td>Input</td>
<td>Input</td>
<td>Party</td>
<td>Hold</td>
<td>Hi-Z</td>
<td>Hi-Z</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L H H L 6</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Two's Complement Multiply, Last Cycle</td>
<td>( F = S + C_n ) ( Z = 1 ) ( F = R - 1 + C_n ) ( Z = 1 )</td>
<td>Log F2 ( = Y ) (Note 2)</td>
<td>Hi-Z</td>
<td>Input</td>
<td>( F_0 )</td>
<td>Log Q2 ( = Q )</td>
<td>Input</td>
<td>( Q_0 )</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L H H H 7</td>
<td>Am29203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H L L L 8</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Single Length Normalize</td>
<td>( F = S + C_n )</td>
<td>( F = Y )</td>
<td>( F_3 )</td>
<td>( F_3 )</td>
<td>Hi-Z</td>
<td>Log 2Q ( = Q )</td>
<td>( Q_3 )</td>
<td>Input</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H L L H 9</td>
<td>Am29203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H L L A</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Double Length Normalize and First Divide Op</td>
<td>( F = S + C_n )</td>
<td>( \log 2F = Y ) ( F_3 ) ( F_3 ) ( F_3 ) ( F_3 )</td>
<td>Input</td>
<td>Log 2Q ( = Q )</td>
<td>( Q_3 )</td>
<td>Input</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H L H H B</td>
<td>Am29203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H L L C</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Two's Complement Divide</td>
<td>( F = S + R + C_n ) ( Z = 1 ) ( F = R - 1 + C_n ) ( Z = 1 )</td>
<td>Log ( 2F = Y ) ( R_3 ) ( V_3 ) ( F_3 )</td>
<td>Input</td>
<td>Log 2Q ( = Q )</td>
<td>( Q_3 )</td>
<td>Input</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H L H H D</td>
<td>Am29203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H H L L E</td>
<td>Am2903 Am29203</td>
<td></td>
<td>Two's Complement Divide, Correction and Remainder</td>
<td>( F = S + C_n ) ( Z = 1 ) ( F = R - 1 + C_n ) ( Z = 1 )</td>
<td>( F = Y ) ( F_3 ) ( F_3 ) ( Hi-Z )</td>
<td>Log 2Q ( = Q )</td>
<td>( Q_3 )</td>
<td>Input</td>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H H H H F</td>
<td>Am29203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**
1. At the most significant slice only, the \( C_{m+4} \) signal is internally gated to the \( Y_3 \) output.
2. At the most significant slice only, \( F_3 \) \( V \) OVR is internally gated to the \( Y_3 \) output.
3. At the most significant slice only, \( S_3 \) \( V \) \( F_3 \) is generated at the \( Y_3 \) output.

- \( L = \text{LOW} \)
- \( H = \text{HIGH} \)
- \( V = \text{Exclusive OR} \)
- \( X = \text{Don't Care} \)
- \( \text{Parity} = \text{SiO}_{3} \lor \text{F}_3 \lor \text{F}_2 \lor \text{F}_1 \lor \text{F}_0 \)
- \( \text{Hi-Z} = \text{High Impedance} \)
PARITY

- Parity is computed and available at $\text{SIO}_0$
  when the destination field $I_{8-5}$ is either \{4, 5, 6, 7\}

- This corresponds to:
  
  4  RAM  F $\rightarrow$ Y, F $\rightarrow$ RAM, No Q activity
  5  QD   F $\rightarrow$ Y, Z/2 $\rightarrow$ Q, no write to RAM
  6  LOADQ F $\rightarrow$ Y, F $\rightarrow$ Q, no write to RAM
  7  RAMQ F $\rightarrow$ Y, F $\rightarrow$ Q, Y $\rightarrow$ RAM

- The computed equation is:

$$
\text{SIO}_0 = F_0 \lor F_1 \lor F_2 \lor F_3 \lor F_4 \lor \ldots \lor F_n \lor \text{SIO}_n
$$
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:

<table>
<thead>
<tr>
<th>I8-16</th>
<th>I5</th>
<th>HEX</th>
<th>MNEMONIC</th>
<th>DEVICE ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>Ø</td>
<td>E</td>
<td>SGNEXT</td>
<td>S10Ø -&gt; Y, Y -&gt; RAM, S10Ø -&gt; S103</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
<td>F</td>
<td>RAMEXT</td>
<td>F -&gt; Y, Y -&gt; RAM, F3 -&gt; S103</td>
</tr>
</tbody>
</table>

- Thus by controlling I5 separately to each slice when I8-16 = 111:

If I5 = 1, slice behaves "normally"

If I5 = Ø, whatever is input on S10Ø will appear on all Yi and S103
- Add a second 15-bit position to the microword
- Change .DEF file to a 5-bit destination field
- Example:

<table>
<thead>
<tr>
<th>INST</th>
<th>SRCX</th>
<th>FUNCT</th>
<th>DEST</th>
<th>RA</th>
<th>RB</th>
<th>Cin</th>
<th>OEX</th>
<th>TEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONT</td>
<td>RMAQ</td>
<td>INCRRR</td>
<td>SINGEXT</td>
<td>Ra</td>
<td>#</td>
<td>LOW</td>
<td>EN</td>
<td>EN</td>
</tr>
</tbody>
</table>

Where in the .DEF file

```
SINGEXT 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 \( Z = \emptyset \)
- \( F = \langle B \rangle + C_{in} \) if \( Z = 1 \)
- \( Y_{3MSS} = (S_3 \lor F_3)MSS \)
EXAMPLES

1. \( B = 10001010 \) sign magnitude for \(-10\)

   \( Z = 1 \) therefore

   \( B = 01110101 \) except \( F3 \lor S3 \rightarrow Y3 \) of MSS

   \( 11110101 \)

   \( \emptyset \lor 1 \rightarrow 1 \)

   \( +C_n = 1 \) 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

   \( 0001001 \)

   \( +C_n = 1 \) add 1

   \( 10001010 \) Voila!
EXAMPLE MICROCODE

2910 SOURCE FUNCT DEST RA RB C\textsubscript{in}

| CONT | RAMAB | SPECL | SGNTWO | # | RØ | Z |

TWO'S COMPLEMENT

F = [B] + C\textsubscript{n} if Z = 0
F = [B] + C\textsubscript{n} if Z = 1
F = Y, B
Y\textsubscript{2} MSS = (S\textsubscript{2} + F\textsubscript{2}) MSS

DEVICE 1

DEVICE 2

DEVICE 3

DEVICE 4

F\textsubscript{n+4}
OVR
N
Am2903/29203
Am2903/29203
Am2903/29203
Am2903/29203

SIO\textsubscript{3}
SIO\textsubscript{0}
SIO\textsubscript{3}
SIO\textsubscript{0}
SIO\textsubscript{3}
SIO\textsubscript{0}
SIO\textsubscript{3}
SIO\textsubscript{0}

\textsuperscript{5}S\textsubscript{MSS} +S
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: \(<R\_0> \leftarrow <R\_0> + 1\)
  
  word addressing: \(<P\_C> \leftarrow <P\_C> + 2\)

- The special function "INCRMNT" provides this capability.

  4 INCRMNT \( F = S + 1 - Cin \)

<table>
<thead>
<tr>
<th>2910</th>
<th>SOURCE</th>
<th>FUNCT</th>
<th>DEST</th>
<th>RA</th>
<th>RB</th>
<th>Cin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CONT</td>
<td>RAMAB</td>
<td>SPECL</td>
<td>INCRMNT</td>
<td></td>
<td>R1</td>
</tr>
</tbody>
</table>


INCR 1 or 2

F = [B] + 1 + C_n
F = Y, B
HOLD Q

ALU

B + 1 + C_n

DESTINATION

LOAD RAM
MULTIPLICATION
MULTIPLICATION

B X A

UNSIGNED BINARY

77
X 11
847

\[ \begin{array}{c}
\text{1001101} \\
\times \text{0001011} \\
\hline
\text{1001101} \\
\text{1001101} \\
\text{0000000} \\
\text{1001101} \\
\text{0000000} \\
\text{0000000} \\
\end{array} \]

\[ \begin{array}{c}
\text{IN FIRST POSITION} \\
\text{IN SECOND POSITION} \\
\text{IN FOURTH POSITION} \\
\end{array} \]

\[ \text{0001101001111} \]

\[ \begin{array}{c}
\text{WEIGHT} \\
\text{\rightarrow} \\
\end{array} \]

\[ \begin{array}{cccc}
\text{512} & \text{256} & \text{64} & \text{8} \\
\text{256} & \text{64} & \text{8} & \text{1} \\
\end{array} \]

\[ \text{847} \]
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 x N multiply is 2N bits long.
MULTIPLICATION

TWO'S COMPLEMENT - "METHOD OF FLORES"

Case #1

\[ B \times A \quad B \text{ positive} \quad A \text{ positive} \]

- 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 \( C_{\text{out}} \)

- 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))

B X A   B NEGATIVE   A POSITIVE

0.01101
1.10101
-------

0.00000001101
0.0000000000
0.00001110100
0.0000000000
0.00111010000
-------
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)

B X A  B NEGATIVE  A NEGATIVE

1.10011
1.10101
--------
1.1111110011
0.0000000000
1.1111001100
0.0000000000
1.1100110000
----------
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 Ø 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 Ø) 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

- LSB of $R_B$ input to MSB of $Q$

- $F_3 \uparrow OVR$ 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
**Specific Interconnections for 16 Bit Multiply - Ripple Carry**

---

<table>
<thead>
<tr>
<th>$i_0$</th>
<th>$i_1$</th>
<th>ADD</th>
<th>$i_2$</th>
<th>$i_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>$B + 0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$B + A$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
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 $R_2$ $\longrightarrow$ 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)

F = [B] + Cn if Z = 0
F = [B] + [A] + Cn if Z = 1
Log F/2 -> Y, B
Q/2 -> Q

**Device 4**

<table>
<thead>
<tr>
<th>Q0</th>
<th>Q10</th>
</tr>
</thead>
</table>

**Device 3**

<table>
<thead>
<tr>
<th>Q0</th>
<th>Q10</th>
</tr>
</thead>
</table>

**Device 2**

<table>
<thead>
<tr>
<th>Q0</th>
<th>Q10</th>
</tr>
</thead>
</table>

**Device 1**

<table>
<thead>
<tr>
<th>Q0</th>
<th>Q10</th>
<th>F0 LSS</th>
</tr>
</thead>
</table>

Note: For unsigned multiply, Cn + 4 MSS is internally shifted into position Y3 MSS; 2's complement multiply N Y OVR is internally shifted into position Y3 MSS.

Q0 LSS internally routed to Z lines

<table>
<thead>
<tr>
<th>Z</th>
<th>ALU</th>
<th>DESTINATION</th>
<th>S3 in MSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>PASS</td>
<td>F/2 -&gt; RAM; Q/2 -&gt; Q</td>
<td>INTERNAL OPERATION</td>
</tr>
<tr>
<td>H</td>
<td>A + B</td>
<td></td>
<td>Cn+4</td>
</tr>
</tbody>
</table>
MULTIPLY

START
0 in R0
Multiplicand in R1
Multiplier in R2

START

R2 → Q
15 → CTR

UNSIGNED MULTIPLY DECREMENT CTR

N

COUNT = 0?

N

END
Product (MS) in R0
Product (LS) in Q
Multiplicand in R1
Multiplier in R2

Y

END
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:

```
add 2910 COND BRCH SRCE FUNC DEST RA RB Cin OEy Ten 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 MULTR Ra Rb LOW EN EN SIOQ TO QI03
```

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

START
0 in R₀
Multiplicand in R₁
Multiplier in R₂

START

R₂ ← Q
14 ← CTR

2's COMPLEMENT MULTIPLY
DECREMENT CTR

COUNT = 0?

Y

2's COMPLEMENT MULTIPLY
LAST CYCLE

END
Product (MS) in R₀
Product (LS) in Q
Multiplicand in R₁
Multiplier in R₂

END
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 $N \lor OVR$ is internally shifted into Y3 of the MSS.

- The code is:

<table>
<thead>
<tr>
<th>addr</th>
<th>2910</th>
<th>COND</th>
<th>BRCH</th>
<th>SRCE</th>
<th>FUNC</th>
<th>DEST</th>
<th>RA</th>
<th>RB</th>
<th>Cin</th>
<th>OE</th>
<th>Ien</th>
<th>ROTATE</th>
<th>INST</th>
<th>MUX</th>
<th>CNTR</th>
<th>A</th>
<th>L</th>
<th>U</th>
<th>ADDR</th>
<th>ADDR</th>
<th>CONNECTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>LDCT # 14</td>
<td>RAMAB</td>
<td>INCRR</td>
<td>LOADQ</td>
<td>R1 #</td>
<td>LOW #</td>
<td>EN</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+1</td>
<td>RPCT # n+1</td>
<td>LOW</td>
<td>SPECL</td>
<td>TWOMULT</td>
<td>Ra</td>
<td>Rb</td>
<td>LOW</td>
<td>EN</td>
<td>EN</td>
<td>S100 TO Q103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n+1</td>
<td>CONT # #</td>
<td>LOW</td>
<td>SPECL</td>
<td>TWOLAST</td>
<td>Ra</td>
<td>Rb</td>
<td>Z</td>
<td>EN</td>
<td>EN</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TWO'S COMPLEMENT MULTIPLY (Cont'd)

- Algorithm

<table>
<thead>
<tr>
<th>FUNCTION</th>
<th>CN</th>
<th>Z</th>
<th>ALU</th>
<th>DESTINATION</th>
<th>S3 in MSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLY</td>
<td>L</td>
<td></td>
<td>PASS</td>
<td>DOWN SHIFTS</td>
<td>INTERNAL OPERATION</td>
</tr>
<tr>
<td></td>
<td>H</td>
<td></td>
<td>A + B</td>
<td></td>
<td>SIGN + OVR</td>
</tr>
<tr>
<td>LAST STEP</td>
<td>L</td>
<td></td>
<td>PASS</td>
<td>F/2 -&gt; RAM; Z/2 -&gt; Q</td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td></td>
<td>B-A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Am2903 CLASS EXERCISE

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

<table>
<thead>
<tr>
<th>2910 CND</th>
<th>BRCH</th>
<th>ADDR</th>
<th>INST</th>
<th>MUX</th>
<th>CNTR</th>
<th>SRC</th>
<th>L</th>
<th>U</th>
<th>RA</th>
<th>ADDR</th>
<th>RB</th>
<th>ADDR</th>
<th>CIN</th>
<th>ØY</th>
<th>ÏEN</th>
<th>ÏEN</th>
<th>ROT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CONT</td>
<td>#</td>
<td>#</td>
<td>RAMAB</td>
<td>LOW</td>
<td>RAM</td>
<td>#</td>
<td>R3</td>
<td>LOW</td>
<td>EN</td>
<td>EN</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 LDCT</td>
<td>#</td>
<td>14</td>
<td>RAMAQ</td>
<td>INCRR</td>
<td>LOADQ</td>
<td>R2</td>
<td>#</td>
<td>LOW</td>
<td>#</td>
<td>EN</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 RPCT</td>
<td>#</td>
<td>3</td>
<td>LOW</td>
<td>SPECL</td>
<td>TWOMULT</td>
<td>R1</td>
<td>R3</td>
<td>LOW</td>
<td>EN</td>
<td>EN</td>
<td>SIO10-Q103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 CONT</td>
<td>#</td>
<td>#</td>
<td>LOW</td>
<td>SPECL</td>
<td>TWOLAST</td>
<td>R1</td>
<td>R3</td>
<td>Z</td>
<td>EN</td>
<td>EN</td>
<td>SIO00-Q103</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 CONT</td>
<td>#</td>
<td>#</td>
<td>RAMAQ</td>
<td>INCRS</td>
<td>RAM</td>
<td>#</td>
<td>R4</td>
<td>LOW</td>
<td>EN</td>
<td>EN</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 Q₀
  - \( C_{n+4} = Q_3 \lor Q_2 \) (MSS)
  - OVR = Q₂ \lor Q₁ (MSS)
  - Z = 1 if Q = Ø (cannot normalize)
  - \( \langle B \rangle + C_N \rightarrow \langle B \rangle \) (use for exponent)
\[ F = (B) + C_n \]

![Diagram of electronic circuit with labels QIO, QIO0, SI03, SI0, and connections between blocks.](image)
SINGLE LENGTH NORMALIZE

GENERAL CASE

1. NUMBER TO NORMALIZE → Q

2. EXEC S.L. NORMALIZE
   INST EN = H

3. ZERO?
   YES → ABORT
   NO → CLR EXPONENT REG
   YES → Q_3 ▷ Q_2 = 1
   NO → \( \text{RESET X}_{EN} = \text{LOW} \)

5. EXEC S.L. NORMALIZE
   EXP + EXP + 1
   \( Q_2 ▷ Q_1 = 1 \)
   YES → END
   NO

6. EXEC S.L. NORMALIZE
   EXP + EXP + 1
   \( Q_2 ▷ Q_1 = 1 \)

7. \( Q + Q/2 \quad Q_3 = M_S \)
   \( \text{EXP} + \text{EXP} - 1 \)
   END

\( M_S = \text{MACRO SIGN BIT STATUS REGISTER} \)
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

Q:Q₃  Q₂  Q₁  Q₀  Z  Cₙ+4  Q₃⊕Q₂  OVR  Q₂⊕Q₁

STATUS:
STZ  X
STCₙ+4  X
STOVR  X

LOAD/CONT  SLN/CONT  NOP/CJP ON  CLEAR/CJP ON  SLN/CJP ON
N→Q  DIS  Z  /Cₙ+4  EN / OVR

ALU/2910

Z ON CURRENT
Cₙ₊₄ Q not
= OVR NEW ONE
TIMING FLOW (CONT.)

SHIFT OCCURS ON CLOCK PULSE

STATUS:

0
0
0*
1

SLN:CJP ON
EN OVR
NORMALIZED BUT SYSTEM DOESN'T KNOW IT YET
THIS SAYS STOP BUT OVERSHIFT ALREADY IN PROCESS

SLN/CJP ON OVR

Q/2 \rightarrow Q
Sign \rightarrow Q_3
(There during SLN execute)
/STOP

DEC R EXP

BRANCH TO 'BACK UP' TO GET RESULT
### Single Length Normalize Microcode

<table>
<thead>
<tr>
<th>INST</th>
<th>MUX</th>
<th>CNTR</th>
<th>SRCE</th>
<th>FUNCT</th>
<th>DEST</th>
<th>RA</th>
<th>RB</th>
<th>C1n</th>
<th>OEQ</th>
<th>TEN</th>
<th>ROT</th>
<th>STATUS</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2910</td>
<td>CND</td>
<td>BRCH</td>
<td>A</td>
<td>L</td>
<td>U</td>
<td>STORE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 CONT # # RAMAQ INCRR 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 DECRMNTH # Rb ONE EN EN # NO DECR EXPONENT USING SPECIAL FUNCTION
Am2903 DOUBLE LENGTH NORMALIZE (DLN)

Set-up:

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

1. NUMBER TO NORMALIZE + Q

2. EXEC S.L. NORMALIZE
   INST EN = H

3. ZERO?
   YES
   ABORT
   NO
   RESET EN = LOW

4. CLR EXPONENT REG
   YES
   Q_3 = Q_2
   NO
   END

5. EXEC S.L. NORMALIZE
   EXP + EXP + 1

6. EXEC S.L. NORMALIZE
   EXP + EXP + 1
   Q_2 = Q_1

7. Q + Q/2
   Q_3 = M_S
   EXP + EXP - 1

END

M_S = MACRO SIGN BIT STATUS REGISTER

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)
### SPECIAL FUNCTIONS (Note 7)

<table>
<thead>
<tr>
<th>(Hex) \ l_3/b_2/l_1</th>
<th>l_4 \ l_3/b_2/l_1</th>
<th>Special Function</th>
<th>ALU Function</th>
<th>ALU Shifter Function</th>
<th>Most Sig Slice</th>
<th>Other Slices</th>
<th>SiO_3</th>
<th>Q Reg &amp; Shifter Function</th>
<th>QIO_3</th>
<th>QIO_2</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>L 0</td>
<td>Unsigned Multiply</td>
<td>F = S \cdot C_n \text{ if } Z = L &lt; L \text{ and } C_n \text{ if } Z = H &lt; H</td>
<td>Log F/2 \to Y (Note 1)</td>
<td>Z</td>
<td>Input</td>
<td>F_0</td>
<td>Log Q/2 \to Q</td>
<td>Input</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>1</td>
<td>L 0</td>
<td>BCD to Binary Conversion</td>
<td>(Note 4)</td>
<td>Log F/2 \to Y</td>
<td>Input</td>
<td>Input</td>
<td>F_0</td>
<td>Log Q/2 \to Q</td>
<td>Input</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>1</td>
<td>H 0</td>
<td>Multiplication BCD to Binary</td>
<td>(Note 4)</td>
<td>Log F/2 \to Y</td>
<td>Input</td>
<td>Input</td>
<td>F_0</td>
<td>Hold</td>
<td>Z</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>L 0</td>
<td>Two's Complement Multiply</td>
<td>F = S + C_n \text{ if } Z = L \text{ and } F = R + S + C_n \text{ if } Z = H \text{ if } Z = H</td>
<td>Log F/2 \to Y (Note 2)</td>
<td>Z</td>
<td>Input</td>
<td>F_0</td>
<td>Log Q/2 \to Q</td>
<td>Input</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>3</td>
<td>L 0</td>
<td>Decrement by One or Two</td>
<td>F = S \cdot 2 + C_n</td>
<td>F = Y</td>
<td>Z</td>
<td>Z</td>
<td>Parity</td>
<td>Hold</td>
<td>Z</td>
<td>Z</td>
<td>L</td>
</tr>
<tr>
<td>4</td>
<td>L 0</td>
<td>Increment by One or Two</td>
<td>F = S + 1 + C_n</td>
<td>F = Y</td>
<td>Input</td>
<td>Input</td>
<td>Parity</td>
<td>Hold</td>
<td>Z</td>
<td>Z</td>
<td>L</td>
</tr>
<tr>
<td>5</td>
<td>L 0</td>
<td>Sign/ Magnitude Two's Complement</td>
<td>F = S + C_n \text{ if } Z = L \text{ and } F = S \cdot 2 + 1 + C_n \text{ if } Z = Z</td>
<td>Log F/2 \to Y (Note 3)</td>
<td>Z</td>
<td>Input</td>
<td>F_0</td>
<td>Log Q/2 \to Q</td>
<td>Input</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>6</td>
<td>L 0</td>
<td>Two's Complement Multiply, Last Cycle</td>
<td>F = S + C_n \text{ if } Z = L \text{ and } F = S \cdot 2 + 1 + C_n \text{ if } Z = H \text{ if } Z = H</td>
<td>Log F/2 \to Y (Note 2)</td>
<td>Z</td>
<td>Input</td>
<td>F_0</td>
<td>Log Q/2 \to Q</td>
<td>Input</td>
<td>Q_0</td>
<td>L</td>
</tr>
<tr>
<td>7</td>
<td>L 0</td>
<td>BCD Divide by Two</td>
<td>(Note 4)</td>
<td>F = Y</td>
<td>Z</td>
<td>Z</td>
<td>Parity</td>
<td>Hold</td>
<td>Z</td>
<td>Z</td>
<td>L</td>
</tr>
<tr>
<td>8</td>
<td>L 0</td>
<td>Single Length Normalize</td>
<td>F = S + C_n</td>
<td>F = Y</td>
<td>F_3</td>
<td>F_3</td>
<td>Input</td>
<td>Log Q/2 \to Q</td>
<td>Q_3</td>
<td>Input</td>
<td>L</td>
</tr>
<tr>
<td>9</td>
<td>L 0</td>
<td>Binary to BCD Conversion</td>
<td>(Note 5)</td>
<td>Log 2F \to Y</td>
<td>F_3</td>
<td>F_3</td>
<td>Input</td>
<td>Log Q/2 \to Q</td>
<td>Q_3</td>
<td>Input</td>
<td>L</td>
</tr>
<tr>
<td>9</td>
<td>H 0</td>
<td>Multiplication Binary to BCD</td>
<td>(Note 5)</td>
<td>Log 2F \to Y</td>
<td>F_3</td>
<td>F_3</td>
<td>Input</td>
<td>Hold</td>
<td>Z</td>
<td>Input</td>
<td>L</td>
</tr>
<tr>
<td>A</td>
<td>L 0</td>
<td>Double Length Normalize and First Division</td>
<td>F = S + C_n</td>
<td>Log 2F \to Y</td>
<td>R_3 \lor F_3</td>
<td>F_3</td>
<td>Input</td>
<td>Log Q/2 \to Q</td>
<td>Q_3</td>
<td>Input</td>
<td>L</td>
</tr>
<tr>
<td>B</td>
<td>L 0</td>
<td>BCD Add</td>
<td>F = R + S + C_n \text{ if } Z = L \text{ and } C_n \text{ if } Z = H \text{ if } Z = H</td>
<td>Log 2F \to Y</td>
<td>R_3 \lor F</td>
<td>F_3</td>
<td>Input</td>
<td>Log Q/2 \to Q</td>
<td>Q_3</td>
<td>Input</td>
<td>L</td>
</tr>
<tr>
<td>C</td>
<td>L 0</td>
<td>Two's Complement Divisor</td>
<td>F = S + R + C_n \text{ if } Z = L \text{ and } F = S \cdot 2 + 1 + C_n \text{ if } Z = Z \text{ if } Z = H</td>
<td>Log 2F \to Y</td>
<td>R_3 \lor F</td>
<td>F_3</td>
<td>Input</td>
<td>Log Q/2 \to Q</td>
<td>Q_3</td>
<td>Input</td>
<td>L</td>
</tr>
<tr>
<td>D</td>
<td>L 0</td>
<td>BCD Subtract</td>
<td>F = R \cdot 2 + 1 + C_n BCD</td>
<td>(Note 6)</td>
<td>F = Y</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>Hold</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>E</td>
<td>L 0</td>
<td>Two's Complement Divide Correction and Remainder</td>
<td>F = S + R + C_n \text{ if } Z = L \text{ and } F = S \cdot 2 + 1 + C_n \text{ if } Z = Z \text{ if } Z = H</td>
<td>Log 2F \to Y</td>
<td>F_3</td>
<td>F_3</td>
<td>Input</td>
<td>Log Q/2 \to Q</td>
<td>Q_3</td>
<td>Input</td>
<td>L</td>
</tr>
<tr>
<td>F</td>
<td>L 0</td>
<td>BCD Subtract</td>
<td>F = S \cdot 2 + 1 + C_n BCD</td>
<td>(Note 6)</td>
<td>F = Y</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>Hold</td>
<td>Z</td>
<td>Z</td>
</tr>
</tbody>
</table>

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 \lor OVR is internally gated to the Y_3 output.
3. At the most significant slice only, S_9 \lor F_3 is generated at the Y_3 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 modulo 16.
6. Additions and subtractions are BCD adds and subtractions. 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 \lor F_3 \lor F_2 \lor F_1 \lor 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:  \( <R0> \leftarrow <R0> - 1 \)

- Word addressing:  \( <SP> \leftarrow <SP> - 2 \)

- The special function "DECRMNT" provides this capability.

- 4  DECRMNT  \( F = S - 2 + \text{Cin} \)

<table>
<thead>
<tr>
<th>2910</th>
<th>SOURCE</th>
<th>FUNCT</th>
<th>DEST</th>
<th>RA</th>
<th>RB</th>
<th>Cin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CONT</td>
<td>RAMAB</td>
<td>SPECL</td>
<td>DECRMNT</td>
<td>#</td>
<td>R2</td>
</tr>
</tbody>
</table>
**BINARY/BCD CONVERSION**

- BCD number is always in a specified RAM register
- Binary number is always in the Q-register
- S1Q0 is connected to Q10n
- S1Q0 is connected to Q10n
- **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

<table>
<thead>
<tr>
<th>2910</th>
<th>CND BRCH A L U</th>
</tr>
</thead>
<tbody>
<tr>
<td>INST</td>
<td>MUX CNTR SRCE FUNCTION DEST RA RB Cin ROT COMMENT</td>
</tr>
</tbody>
</table>

1. LDCT # 15 RAMAQ INCRR 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-SO PERFORM CONVERSION
BINAY/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 C1n ROT COMMENT |
------------------------------------------------------
1 CONT # # RAMAQ LOW LOADQ # # LOW # CLEAR Q
2 LDCT # 14 RAMAQ INCRR RAMQDL Ra # LOW # INITIAL DOWNSHIFT
3 RPCT # 3 RAMAB SPECL BCD.BIN # Ra LOW SO-Q3 PERFORM CONVERSION
Microcode Flow Chart of BCD Conversion

**Binary to BCD**

1. **Start**
   - Registers loaded
2. **Bin to BCD**
3. **Is iter = N?**
   - If NO, go back to Bin to BCD.
   - If YES, go to Finish.

**Do N times**

**BCD to Binary**

1. **Start**
   - Registers loaded
2. **F/2 + F Q/2 + Q**
3. **BCD to Bin**
4. **Is iter = N - 1?**
   - If NO, go back to BCD to Bin.
   - If YES, go to Finish.

**Do N-1 times**
BCD CONVERSIONS

RAM REGISTER

BCD NUMBER

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Q REGISTER

Q REGISTER

BINARY NUMBER

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
**Binary to BCD**

- **QN+4**
- **C**
- **S0**
- **Z** (OVR)

**RAM = ZERO**

**ALU**

<table>
<thead>
<tr>
<th>BIN TO BCD CONVERSION</th>
<th>DESTINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2F</strong></td>
<td>RAM; <strong>2Q</strong></td>
</tr>
</tbody>
</table>
BCD TO BINARY

MSS

Q₁

S₀

Q₀

Z

Q = ZERO

S₃

S₀

Q₀

Z

S₃

S₀

Q₀

Z

LSS

S₃

S₀

Q₀

Z

V₅

ALU | DESTINATION
---|------------------
BCD TO BIN CONVERSION | F/2 → RAM; Q/2 → Q
**SIGNIFICANT SPEED IMPROVEMENT ON Am2903**

<table>
<thead>
<tr>
<th></th>
<th>Am2903</th>
<th>Am2903A</th>
<th>%FASTER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(MIL MAX)</td>
<td>(PROJECTED MIL MAX)</td>
<td></td>
</tr>
<tr>
<td>ADDRESS TO G,P</td>
<td>84</td>
<td>57</td>
<td>32%</td>
</tr>
<tr>
<td>CN TO Z</td>
<td>65</td>
<td>40</td>
<td>38%</td>
</tr>
<tr>
<td>ADDRESS TO Z</td>
<td>126</td>
<td>75</td>
<td>41%</td>
</tr>
<tr>
<td>ADD CYCLE TIME</td>
<td>181</td>
<td>129</td>
<td>29%</td>
</tr>
<tr>
<td>LOGIC CYCLE TIME</td>
<td>152</td>
<td>100</td>
<td>34%</td>
</tr>
</tbody>
</table>
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
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
Am2903 - Data Bus Cascading
Am2903 - RAM Address Cascading

Two Address Operation
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

<table>
<thead>
<tr>
<th></th>
<th>Am29705</th>
<th>Am29705A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial maximum:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>access time</td>
<td>53</td>
<td>30</td>
</tr>
<tr>
<td>LE to YA/YB</td>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>A-latch reset</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td>address set up before latch closes</td>
<td>45</td>
<td>15</td>
</tr>
</tbody>
</table>
Am29707 -- 28 Pin

16 Word by 4 Bit Two Port RAM

A - PORT  WE  B - PORT

A3 A2 A1 A0
ADDRESS DECODER

B3 B2 B1 B0
ADDRESS DECODER

WE1 WE2 WE/BLE

ALE CE/A

LE
A - DATA 4-BIT LATCH

LE
B - DATA 4-BIT LATCH

YA3 YA2 YA1 YA0

YB3 YB2 YB1 YB0

CE-B
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

- Do you need: Greater than 16-bit word length OR special or predefined instruction set OR high speed instruction time (<1 µsec)?
  - YES: Use an Am2900 Microprocessor
  - NO: Use the Am2901

  - Do you need: Advanced arithmetic capability OR arithmetic and logical shifting OR two data input ports OR parity generation OR larger register file?
    - YES: Use the Am2903
    - NO: Use the Am2901

  - Do you need: BCD Arithmetic OR exceptional byte handling capability OR exceptional I/O architecture?
    - YES: Use the Am29203, the premier Microprocessor Slice.
    - NO: Use the Am2901
<table>
<thead>
<tr>
<th>ADDRESS TO:</th>
<th>2901B</th>
<th>2901C</th>
<th>2903</th>
<th>2903A/29203</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>60</td>
<td>40</td>
<td>99</td>
<td>68</td>
</tr>
<tr>
<td>GP</td>
<td>50</td>
<td>37</td>
<td>81</td>
<td>52</td>
</tr>
<tr>
<td>F=Ø</td>
<td>70</td>
<td>40</td>
<td>123</td>
<td>72</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D TO:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Cn+4</td>
</tr>
<tr>
<td>F=Ø</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I TO:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
</tr>
<tr>
<td>F=Ø</td>
</tr>
</tbody>
</table>
SIMPLE COMPUTER SOLUTION
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 REQUESTS

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.
INTERRUPT REQUEST (ACTIVE LOW) FROM SOMEWHERE

LATCH

LATCH BYPASS

INTERRUPT REQUEST TO SYSTEM

LATCH BYPASS = 0, PULSE CATCHER MODE
= 1, LEVEL FOLLOWER MODE
Multiple (4) Interrupt Storage

~ Circuit Diagram ~

CLOCK

Interrupt Request 0

Interrupt Request 1

Interrupt Request 2

Interrupt Request 3
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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Next Address</th>
<th>Test Select</th>
<th>Branch Address</th>
<th>CCEN</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER:</td>
<td>CJS</td>
<td>INT1</td>
<td>Routine1</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CJS</td>
<td>INT2</td>
<td>Routine2</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CJS</td>
<td>INT3</td>
<td>Routine3</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CJS</td>
<td>INT4</td>
<td>Routine4</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRTN</td>
<td>##</td>
<td>##</td>
<td>Disable</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Next Address</th>
<th>Test Select</th>
<th>Any Select</th>
<th>Branch Address</th>
<th>CCEN</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER:</td>
<td>CJS</td>
<td>##</td>
<td>Any</td>
<td>Inter1</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(continue with normal cycle)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTER1</td>
<td>CJS</td>
<td>INT1</td>
<td>All</td>
<td>Routine1</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CJS</td>
<td>INT2</td>
<td>All</td>
<td>Routine2</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CJS</td>
<td>INT3</td>
<td>All</td>
<td>Routine3</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CJS</td>
<td>INT4</td>
<td>All</td>
<td>Routine4</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRTN</td>
<td>##</td>
<td>##</td>
<td>##</td>
<td>Disable</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>EI T0 T1 T2 T3 T4 T5 T6 T7</td>
<td>A0 A1 A2 EO</td>
</tr>
<tr>
<td>H X X X X X X X X</td>
<td>L L L H</td>
</tr>
<tr>
<td>L H H H H H H H</td>
<td>L L L L</td>
</tr>
<tr>
<td>L X X X X X X X L</td>
<td>H H H H</td>
</tr>
<tr>
<td>L X X X X X X L H</td>
<td>L H H H</td>
</tr>
<tr>
<td>L X X X X X L H H</td>
<td>H L H H</td>
</tr>
<tr>
<td>L X X X L H H H H</td>
<td>H H L H</td>
</tr>
<tr>
<td>L X X L H H H H H</td>
<td>L H L H</td>
</tr>
<tr>
<td>L X L H H H H H H</td>
<td>H L L L</td>
</tr>
<tr>
<td>L L H H H H H H</td>
<td>L L L H</td>
</tr>
</tbody>
</table>

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care
For G1 = H, G2 = H, G3 = L, G4 = L, G5 = L

Z = HIGH Impedance

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Z Z Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>L</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Z Z Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>Z Z Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>H</td>
<td>X</td>
<td>Z Z Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>H</td>
<td>H</td>
<td>Z Z Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADVANCED MICRO DEVICES
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

MAIN PROGRAM

50
51
52
53
54
55
56
57

502 ADDRESS IS INTERRUPT SOURCE DEPENDENT

BRANCH TO

IF NO REQUEST – CONTINUE

TEST FOR OCCURRENCE OF INTERRUPT REQUEST AT SPECIFIC POINT

RESTORE PROGRAM FLOW BY JUMP TO KNOWN POINT
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**

<table>
<thead>
<tr>
<th>Next Address</th>
<th>Test Address</th>
<th>Branch Address</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER: CJS</td>
<td>ANY</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>n+1</td>
<td>R4&lt;--R4-R5-1</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>n+2</td>
<td>R5&lt;--R5+1</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>n+3</td>
<td>R5&lt;--R5+Q</td>
<td>State</td>
<td></td>
</tr>
<tr>
<td>CJV</td>
<td>PASS</td>
<td>##</td>
<td></td>
</tr>
<tr>
<td>RETURN: CONT</td>
<td></td>
<td>Restore State</td>
<td></td>
</tr>
<tr>
<td>CONT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRTN</td>
<td>PASS</td>
<td>##</td>
<td></td>
</tr>
</tbody>
</table>

**EXAMPLE #2**

<table>
<thead>
<tr>
<th>Next Address</th>
<th>Test Address</th>
<th>Branch Address</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTER: CJV</td>
<td>ANY</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>RETURN: (continue with normal cycle)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SAMPLE SYSTEM FOR FIRMWARE LEVEL INTERRUPT HANDLING
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 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
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
512 X 8
Am 27S26/27

9 BIT ADDRESS

EDGE TRIGGER REGISTER

WORST CASE:

ADDRESS
55 NS SETUP

25 NS \( \overline{E}_S \) SETUP

35 NS OUTPUT
clk-Q

40 NS \( \overline{E} \) ENABLE-Q

ADDRESS AND \( \overline{E}_S \) LOW - FETCH

CLK AND \( \overline{E} \) LOW - DATA OUTPUT

REGISTERED PROM
$2K \times 8$

Am 25LS139

$A_{10}$ $A_9$ $A_8 - A_0$

Am27S27

$E_S$ LOW - SELECT
$E$ LOW - OUTPUT ENABLE

55 ns → 35 ns
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.
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.**

<table>
<thead>
<tr>
<th>$I_{10}$</th>
<th>$I_9$</th>
<th>$I_8$</th>
<th>$I_7$</th>
<th>$I_6$</th>
<th>$M_C$</th>
<th>RAM</th>
<th>Q</th>
<th>$SIO_{0}$</th>
<th>$SIO_{n}$</th>
<th>$QIO_{0}$</th>
<th>$QIO_{n}$</th>
<th>Loaded into $M_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>0</td>
<td>Z</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>1</td>
<td>Z</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>0</td>
<td>Z</td>
<td>$M_N$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>1</td>
<td>Z</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$M_C$</td>
<td>$Z$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$M_N$</td>
<td>$Z$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>0</td>
<td>Z</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>0</td>
<td>Z</td>
<td>$SIO_{0}$</td>
<td>$QIO_{0}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$SIO_{0}$</td>
<td>$Z$</td>
<td>$QIO_{0}$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$M_C$</td>
<td>$Z$</td>
<td>$QIO_{0}$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$SIO_{0}$</td>
<td>$Z$</td>
<td>$QIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$I_C$</td>
<td>$Z$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$M_C$</td>
<td>$Z$</td>
<td>$SIO_{0}$</td>
<td>$QIO_{0}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$QIO_{0}$</td>
<td>$Z$</td>
<td>$SIO_{0}$</td>
<td>$QIO_{0}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$I_N \oplus IOVR$</td>
<td>$Z$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Z</td>
<td>$QIO_{0}$</td>
<td>$Z$</td>
<td>$SIO_{0}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: 1. $Z = \text{High impedance (outputs off) state}$
2. Outputs enabled and $M_C$ loaded only if $SE$ is LOW.
3. Loading of $M_C$ from $I_{10}$ overrides control from $I_{6-0}$, $CE_M$, $OE_C$.
### TABLE 4. CONDITION CODE OUTPUT (CT) INSTRUCTION CODES.

<table>
<thead>
<tr>
<th>HEX</th>
<th>l₃</th>
<th>l₂</th>
<th>l₁</th>
<th>l₀</th>
<th>l₅ = l₄ = 0</th>
<th>l₅ = 0, l₄ = 1</th>
<th>l₅ = 1, l₄ = 0</th>
<th>l₅ = l₄ = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(μ₂ ⊕ μ₀VR) + μ₂</td>
<td>(μ₂ ⊕ μ₀VR) + μ₂</td>
<td>(M₀ ⊕ M₀VR) + M₀</td>
<td>(l₀ ⊕ l₀VR) + l₀</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(μ₂ ⊕ μ₀VR) ⋅ μ₀VR</td>
<td>(μ₂ ⊕ μ₀VR) ⋅ μ₀VR</td>
<td>(M₀ ⊕ M₀VR) ⋅ M₀</td>
<td>(l₀ ⊕ l₀VR) ⋅ l₀</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>μ₂ ⊕ μ₀VR</td>
<td>μ₂ ⊕ μ₀VR</td>
<td>M₀ ⊕ M₀VR</td>
<td>l₀ ⊕ l₀VR</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>μ₂ ⊕ μ₀VR</td>
<td>μ₂ ⊕ μ₀VR</td>
<td>M₀ ⊕ M₀VR</td>
<td>l₀ ⊕ l₀VR</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>μ₂</td>
<td>μ₂</td>
<td>M₀</td>
<td>l₀</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>μ₀VR</td>
<td>μ₀VR</td>
<td>M₀VR</td>
<td>T₀</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>μ₀VR</td>
<td>μ₀VR</td>
<td>M₀VR</td>
<td>l₀</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>μ₀VR</td>
<td>μ₀VR</td>
<td>M₀VR</td>
<td>T₀</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>μ₂ + μ₂</td>
<td>μ₂ + μ₂</td>
<td>M₀ + M₀</td>
<td>l₀ + l₀</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>μ₂ + μ₂</td>
<td>μ₂ + μ₂</td>
<td>M₀ + M₀</td>
<td>l₀ + l₀</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>μ₂</td>
<td>μ₂</td>
<td>M₀</td>
<td>l₀</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>μ₂</td>
<td>μ₂</td>
<td>M₀</td>
<td>l₀</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>μ₂ + μ₂</td>
<td>μ₂ + μ₂</td>
<td>M₀ + M₀</td>
<td>l₀ + l₀</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>μ₂ + μ₂</td>
<td>μ₂ + μ₂</td>
<td>M₀ + M₀</td>
<td>l₀ + l₀</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>l₀ ⊕ l₀VR</td>
<td>l₀ ⊕ l₀VR</td>
<td>l₀ ⊕ l₀VR</td>
<td>l₀ ⊕ l₀VR</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>l₀ ⊕ l₀VR</td>
<td>l₀ ⊕ l₀VR</td>
<td>l₀ ⊕ l₀VR</td>
<td>l₀ ⊕ l₀VR</td>
</tr>
</tbody>
</table>

**Notes:**
1. ☐ Represents EXCLUSIVE-OR
2. Correct code as stated.
3. ☐ Represents EXCLUSIVE-NOR or coincidence.

- u 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**

<table>
<thead>
<tr>
<th>PATTERN</th>
<th>WAVEFORMS AND TIMING</th>
<th>PATTERN</th>
<th>WAVEFORMS AND TIMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLL</td>
<td></td>
<td>LHM</td>
<td></td>
</tr>
<tr>
<td>F_3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F_4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F_5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F_6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F_7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRYSTAL FREQ &quot;f&quot;</td>
<td>1/f ns</td>
<td>000</td>
<td>001</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>16.6 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 &quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 &quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.3 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CYCLE LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/f ns</td>
</tr>
<tr>
<td>000</td>
</tr>
<tr>
<td>001</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>011</td>
</tr>
<tr>
<td>010</td>
</tr>
<tr>
<td>110</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Am 2925</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3L2L1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>400</td>
</tr>
<tr>
<td>300</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>33.3</td>
</tr>
<tr>
<td>333.3</td>
</tr>
</tbody>
</table>
Am29116

A High-Performance 16-bit Bipolar Microprocessor
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) | I_E (1) | O_E (1) | DLE (1) | SRE (1) | O_E (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 overlayed 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 prerequisite.

- **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