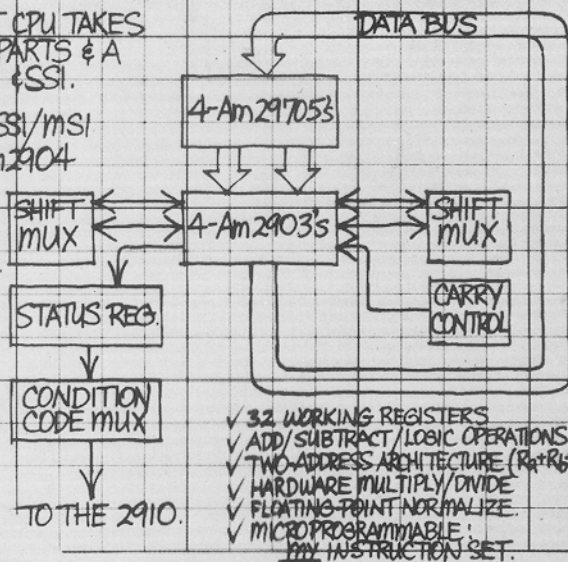


THE COMPLETE 16-BIT CPU TAKES
ONLY 8 Am 2900 PARTS & A
HANDFUL OF MSI & SSI.

REPLACE THE SSI/MSI
WITH THE Am2904
LATER.

MIL-STD-883
FOR FREE.



Build A Microcomputer

Chapter III The Data Path

Advanced Micro Devices



Copyright © 1978 by Advanced Micro Devices, Inc.

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

AM-PUB073-3

INTRODUCTION

The heart of most digital arithmetic processors is the arithmetic logic unit (ALU). The ALU can be thought of as a digital subsystem that performs various arithmetic and logic operations on two digital input variables. The Am2901A and Am2903 are Low Power Schottky TTL arithmetic logic unit/function generators that perform arithmetic/logic operations on two four-bit input variables. In most ALUs, speed is generally a key ingredient. Therefore, as much parallelism in the operation of the arithmetic logic unit as possible is desired.

The Am2901A and Am2903 ALUs are designed to operate with an Am2902A carry lookahead generator to perform multi-level full carry lookahead over any number of bits. Therefore, the devices have both the carry generate and carry propagate outputs required by the Am2902A carry lookahead generator. The devices also have the carry output (C_{n+4}) and a two's complement overflow detection signal (OVR) available at the output. The net result is that a very high-speed 16-bit arithmetic logic unit/function generator can be designed and assembled using four of these bit slice devices and one Am2902A (the Am2902A is a high-speed version of the '182 carry lookahead generator). In addition, the Am2901A and Am2903 provide a minimum of 16 working registers for providing source operands to the ALU.

UNDERSTANDING THE BASIC FULL ADDER

The results of an arithmetic operation in any position in a word depends not only on the two-input operand bits at that position, but also on all the lesser significant operand bits of the two input variables. The final result for any bit, therefore, is not available until the carries of all the previous bits have rippled through the logic array starting from the least significant bit and propagating through to the most significant bit. A full adder is a device that accepts two individual operand bits at the same binary weight, and also accepts a carry input bit from the next lesser significant weight full adder. The full adder then produces the sum bit for this bit position and also produces a carry bit to be used in the next more significant weight full adder carry input. The truth table for a full adder is shown in Figure 1. From this truth table, the equations for the full adder:

$$S = A \oplus B \oplus C$$

$$C_O = AB + BC + AC,$$

where A and B are the input operands to the full adder and C is the carry input into the adder.

Inputs			Outputs	
A	B	C	S	C ₀
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 1. Full Adder Truth Table.

The sum output, S, represents the sum of the A and B operand inputs and the carry input. The carry output, C_O, represents the carry out of this cell and can be used in the next more significant cell of the adder. Full adder cells can be cascaded as depicted in Figure 2 to form a four-bit ripple carry parallel adder.

Note that once we have cascaded devices as shown in Figure 2, we may wish to discuss the equations for the i-th bit of the adder. In so doing, we might describe the equations of the full adder as follows:

$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_i B_i + B_i C_i + A_i C_i$$

where the A_i and B_i are the input operands at the i-th bit, and the C_i is the carry input to the i-th bit. (Note that the equations for this adder are iterative in nature and each depends on the result of the previous lesser significant bits of the adder array.)

The connection scheme shown in Figure 2 requires a ripple propagation time through each full adder cell. If a 16-bit adder is to be assembled, the carry will have to propagate through all 16 full adder cells. What is desired is some technique for anticipating the carry such that we will not have to wait for a ripple carry to propagate through the entire network. By using some additional logic, such an adder array can be constructed. This type of adder is usually called a carry lookahead adder.

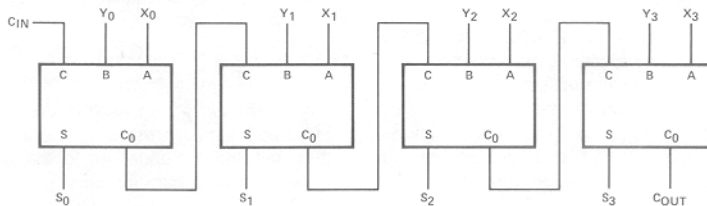


Figure 2. Cascaded Full Adder Cells Connected as a Four-Bit Ripple-Carry Full Adder.

A FOUR-BIT CARRY LOOKAHEAD ADDER

Looking back to the equations developed for i -th bit of an adder, let us now rewrite the carry equation in a slightly different form. When we factor the C_i in this equation, the new equation becomes:

$$C_{i+1} = A_i B_i + C_i (A_i + B_i)$$

From the above equation, let us now define two additional equations. These are:

$$G_i = A_i B_i$$

$$P_i = A_i + B_i$$

With these two new auxiliary equations, we can now rewrite the carry equation for the i -th bit as follows:

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

Note that we have now developed two terms: the P_i term is known as carry propagate and the G_i term is known as carry generate. An anticipated carry can be generated at any stage of the adder by implementing the above equations and using the auxiliary functions P_i and G_i as required.

It is interesting to note that the sum equation can also be written in terms of these two auxiliary equations, P_i and G_i . For this case, the equation is:

$$S_i = (A_i + B_i)(\bar{A}_i \bar{B}_i) \oplus C_i$$

The auxiliary function G_i is called carry generate, because if it is true, then a carry is immediately produced for the next adder stage. The function P_i is called carry propagate because it implies there will be a carry into the next stage of the adder if there is a carry into this stage of the adder. That is, G_i , causes a carry signal at the i -th stage of the adder to be generated and presented to the next stage of the adder while P_i causes an existing carry at the input to the i -th stage of the adder to propagate to the next stage of the adder.

Let us now write all of the sum and carry equations required for a full four-bit lookahead carry adder.

$$S_0 = A_0 \oplus B_0 \oplus C_0$$

$$S_1 = A_1 \oplus B_1 \oplus (G_0 + P_0 C_0)$$

$$S_2 = A_2 \oplus B_2 \oplus (G_1 + P_1 G_0 + P_1 P_0 C_0)$$

$$S_3 = A_3 \oplus B_3 \oplus (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0)$$

$$C_{i+4} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

An important point to note is that ALL of the sum equations and the final carry output equation, C_{i+4} , can be written in terms of the A_i , B_i , and C_0 inputs to the four-bit adder. The configuration as described above is shown in Figure 3. This figure is divided into two parts - the upper blocks show the auxiliary function generator circuitry required to implement the P_i and G_i equations while the lower block implements the logic required to generate the sum output at each bit position.

A serious drawback to the lookahead carry adder is that as the word length is increased, the carry functions become more and more complex, eventually becoming impractical due to the large number of interconnections and heavy loading of the G_i and P_i functions. The auxiliary function concept can be extended, however, by dividing the word length into fairly small increments and defining blocks of auxiliary functions G and P .

It is possible for a given block to define a function G as the carry out generated with the block; and P can be defined as the carry propagate over the block. If the block size is set at four bits, then the functions for G and P for this block can be defined as follows:

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

$$P = P_3 P_2 P_1 P_0$$

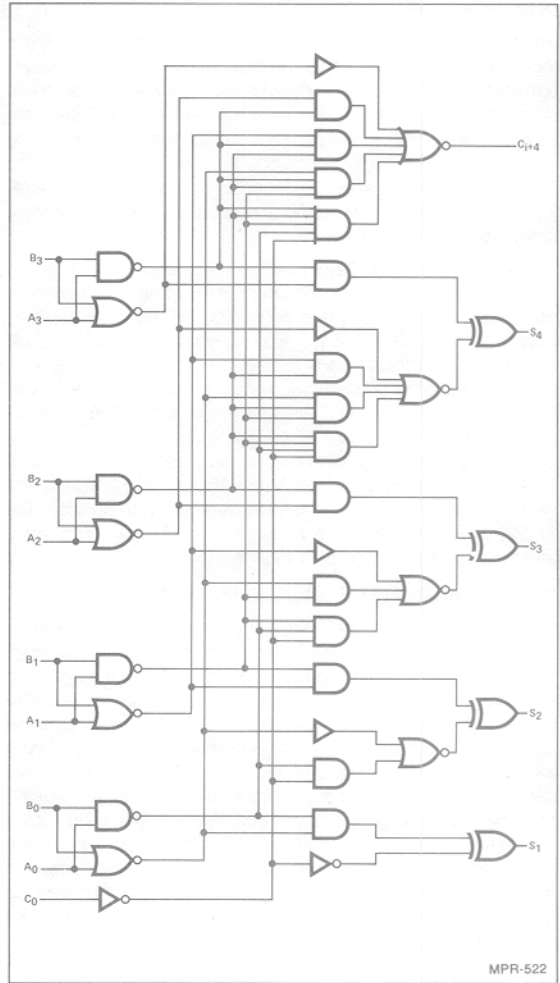


Figure 3. Full Four-Bit Carry-Lookahead Adder.

It is important to note that neither of these terms involves a carry-in (C_0) to the block, so no matter how many blocks are tied in an adder, all the blocks have stable G and P functions available in a minimum number of gate delays.

The G and P functions can be gated to produce a carry-in to each four-bit block, as a function of the lesser significant blocks. The carry-in to a block is therefore:

$$C_n = G_{n-1} + P_{n-1} G_{n-2} + P_{n-1} P_{n-2} G_{n-3} + \dots + P_{n-1} P_{n-2} P_{n-3} \dots P_2 P_1 P_0 C_0$$

Finally, the carry-in to each of the bits in a four-bit block must include a term for the actual least significant carry-in; note, therefore, that the equations for the four-bit full adder presented above include a term for carry-in at each bit position.

Figure 4 shows the technique for cascading typical bit slice ALUs such as the Am2901A or Am2903 and one Am2902A in a full 16-bit high-speed carry lookahead connection. Figure 5 shows a connection scheme using only four bit slices in a 16-bit arithmetic logic unit connection where the carries are rippled between the devices. Each bit slice does use internal carry lookahead over the four-bit block.

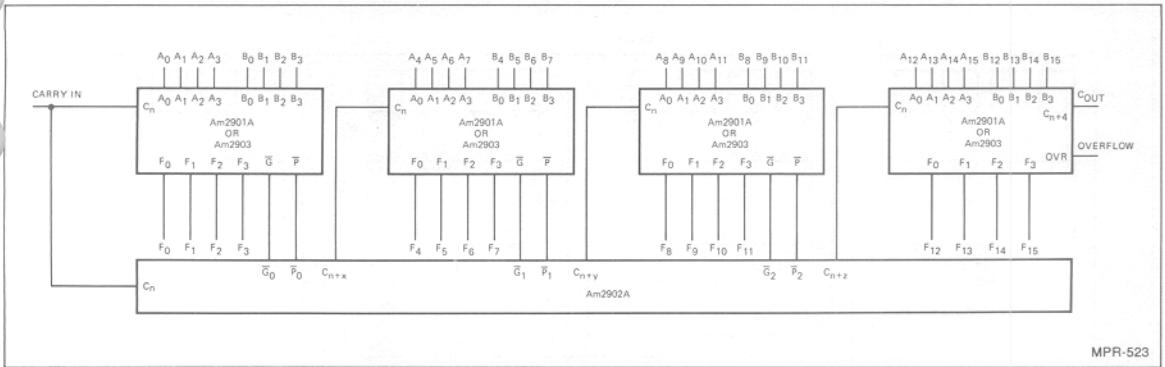


Figure 4. Full Lookahead Carry 16-Bit Adder.

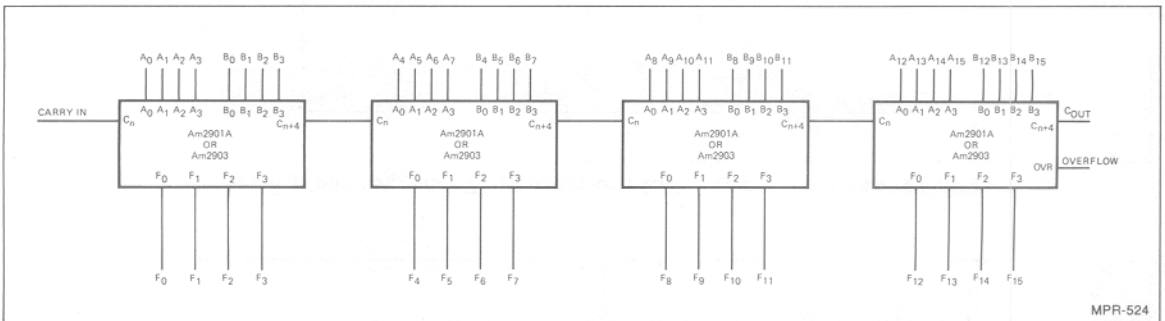


Figure 5. Connection of 16-Bit ALU Using Ripple Carry.

In summary, the ripple carry method can be used in conjunction with the lookahead technique in several ways.

1. Lookahead carry over sections of the adder and ripple carry between these sections of the adder can be used. This method is often the most efficient in terms of hardware for a given speed requirement. It does not require the use of a lookahead carry generator such as the Am2902A.
2. Lookahead carry across 16-bit blocks with a ripple carry between 16-bit blocks can be used. This technique is usually called two-level carry lookahead addition. This technique results in very high-speed arithmetic function generation and makes a reasonable tradeoff between the speed and hardware for word lengths greater than 16 bits.
3. Full lookahead carry across all levels and all block sizes can be used. This is the highest speed arithmetic logic unit connection scheme. For word sizes up to 64 bits, it is referred to as three-level lookahead carry addition. Such a 64-bit ALU requires the use of five Am2902A carry lookahead generator units in addition to the 16 bit slice ALU devices as shown in Figure 6.

OVERFLOW

When two's complement numbers are added or subtracted, the result must lie within the range of the numbers that can be handled by the operand word length. Numbers are normally represented either as fractions with a binary point between the sign bit and the rest of the word, or as integers where the binary point is after the least significant bit. The actual choice for the location of the binary point is really up to the design engineer, as

the hardware configuration required for either technique is identical. It is also possible to use number notations that include both integer and fractional representations in the same numbering scheme. Overflow is defined as the situation in which the result of an arithmetic operation lies outside of the number range that can be represented by the number of bits in the word. For example, if two eight-bit numbers are added and the result does not lie within the number range that can be represented by an eight-bit word, we say that an overflow has occurred. This can happen at either the positive end of the number range or at the negative end of the number range. The logic function that indicates that the result of an operation is outside of the representable number range is:

$$OVR = C_s \oplus C_{s+1}$$

where C_s is the carry-in to the sign bit and C_{s+1} is the carry-out of the sign bit.

Thus, for a four-bit ALU with the sign bit in the most significant bit position, the two's complement overflow can be defined as the C_{n+4} term exclusive OR'ed with the C_{n+3} term.

Putting the ALU in the Data Path of a Simple Computer

Once the Design Engineer understands the basic configuration and operation of a simple high speed carry lookahead adder, he can begin to understand the configuration required to implement the data handling section of a typical computing machine. The simplest architecture for the data handling path of a minicomputer is shown in Figure 7. Here, an accumulator is used in conjunction with an ALU to perform a basic arithmetic/storage capability for data handling. The computer control unit of Figure 7 can be a simple or sophisticated state machine as described in Chapter 2.

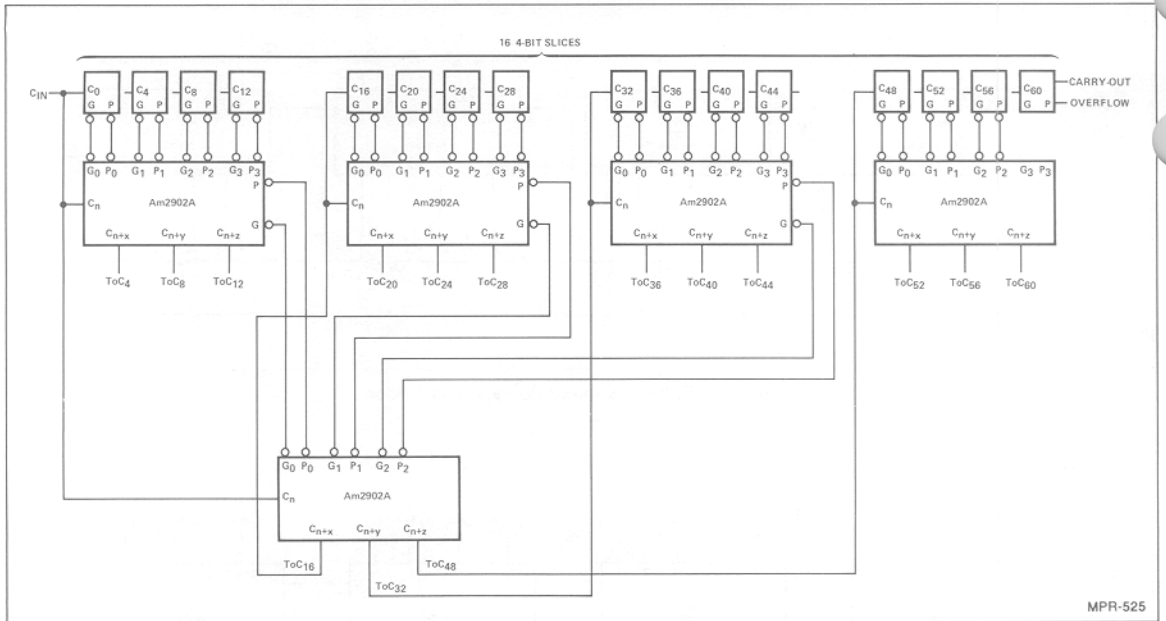


Figure 6. 64-Bit ALU with Full Carry Lookahead Using 5 Am2902s and 16 4-Bit Slices.

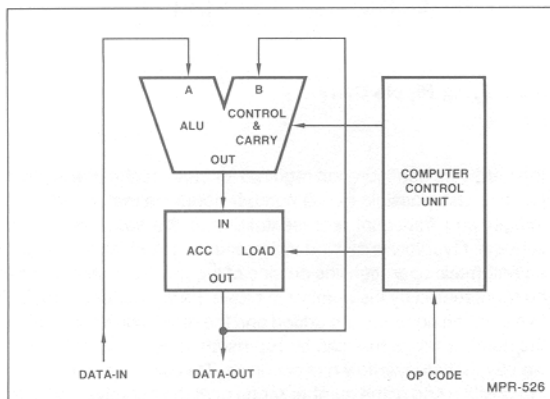


Figure 7. Basic Computer Data Path.

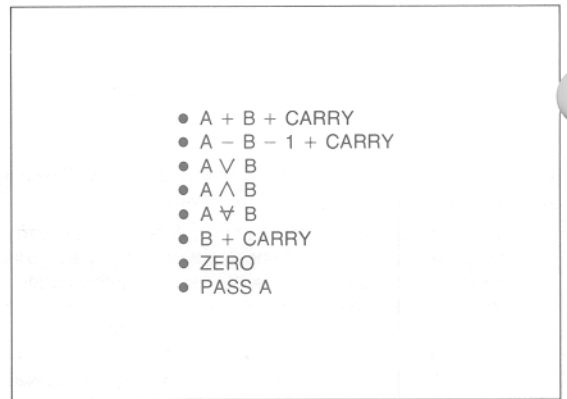


Figure 8. Basic ALU Instructions.

While the introductory material of this chapter concentrated on full adders, it should be understood that more ALU functions than addition are required if we are in to implement the data path of a typical minicomputer. Typically, some or all of the functions shown in Figure 8 are needed if we are to implement a powerful data handling capability.

The operation of the ALU/accumulator configuration shown in Figure 7 can be described as follows. The accumulator can be loaded by bringing data in from the data-in port through the A input of the ALU, passed through the ALU and loaded into the accumulator. A second word of data can be presented at the data-in port to the A input of the ALU and the ALU can be used to perform an operation such as $A + B$, $A \text{ OR } B$, $A \text{ AND } B$, $A - B$ and so forth. The results of this ALU operation can then be placed into the accumulator. The accumulator output is available at the data-out port for use elsewhere. Additional ALU functions such as

those shown in Figure 8 are easily implemented by adding some additional circuitry to the four-bit carry look ahead adder shown in Figure 3. If this circuitry is added, we will arrive at a logic diagram as shown in Figure 9. This diagram certainly is familiar to most CPU designers and is the well known Am74S181 four-bit arithmetic logic unit/function generator.

Once the operation of the simple computer data path as shown in Figure 7 is understood, the Design Engineer will soon recognize the need for additional registers if our machine is to be general purpose and execute instructions. Very rapidly the need arises for a register to hold a program counter (PC) and a memory address register (MAR). The purpose of the program counter is to point to the address of the next instruction in main memory. Typically it is loaded into the memory address register which actually provides the address on to the address bus of the machine. Then, the program counter is incremented through the ALU and stored until

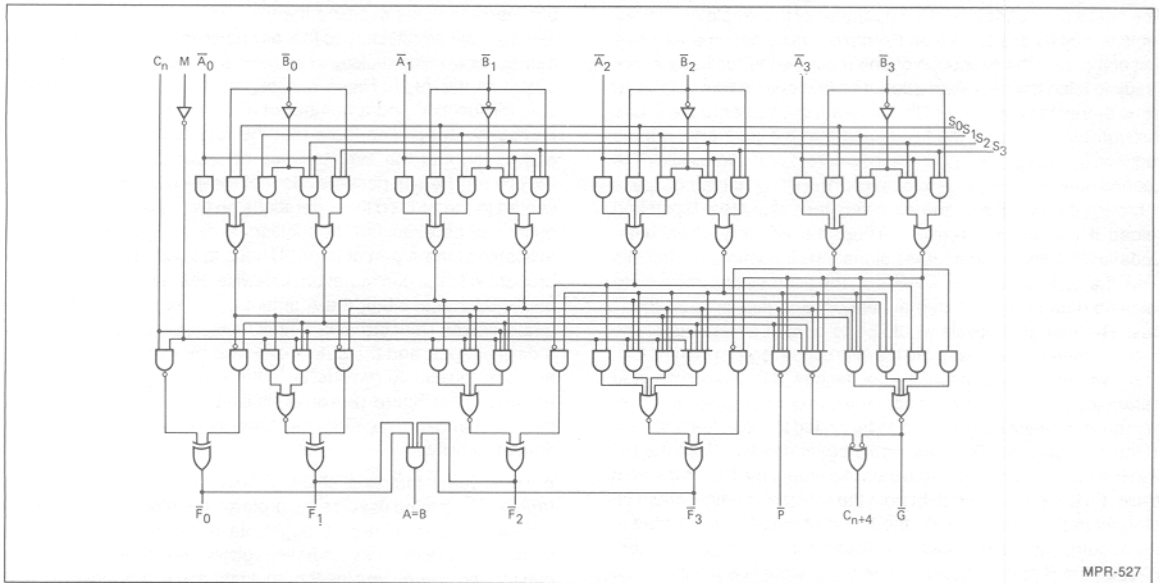


Figure 9. Logic Diagram for Am25LS181.

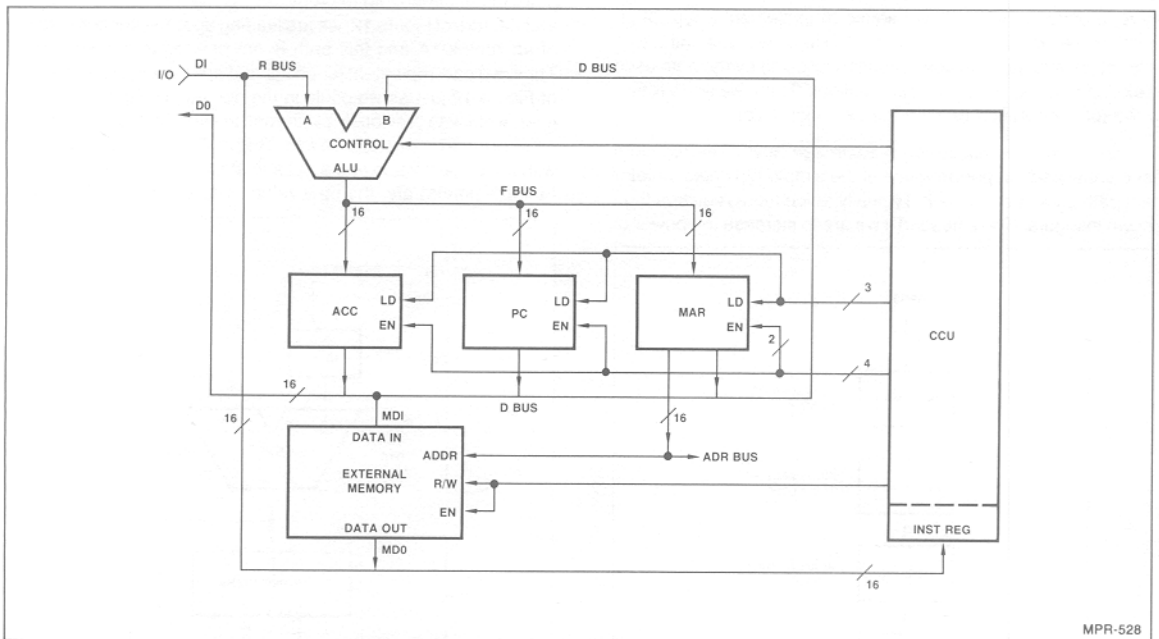


Figure 10. Three Register Computer Data Path.

it is needed again. The block diagram of Figure 10 shows these additional registers connected in parallel at the output of the ALU. This ALU output is called the F bus. Each of these registers (the accumulator, the PC, and the MAR) has an enable input from the CCU so that they can be selectively loaded with data from the ALU. In addition, each of these registers has an output enable such that they can be selectively enabled onto the D bus. The D bus represents the data output path from the basic computer data

path and also is used as one of the inputs to the actual ALU/function generator. The other input in this example is called the R bus and comes directly from the main memory data output as well as from the I/O data input. As shown in Figure 10, the memory address register (MAR) has a second output that is used to drive the address bus. In this example, this register always contains the address to be applied to the external memory whether it be the address of data or the address of an instruction.

The best way to understand the operation of this single ALU/three register machine is to take an example. Let us assume we have just completed the execution of one machine instruction and are ready to fetch the next instruction. The first operation would be to transfer the current value of the program counter onto the D bus through the ALU onto the F bus and into the memory address register. This might be accomplished during one microcycle. The second operation might be to again put the PC on the D bus, pass it through the ALU B port and increment the value at the B port and reload it into the PC register. Thus, the PC has again been updated to point to the address of the next instruction. During this time, the address from the MAR is on the address bus and we are fetching data from the external memory and placing it on the R bus. The third microcycle would be to bring the data out of the external memory and pass it to the instruction register in the CCU. The next microcycle might be to decode this instruction and determine that the next word after the current instruction in memory (an immediate operation) is to be added to the value currently in the accumulator. Thus, we would again need to place the PC into the MAR on one cycle and then increment the PC on the next cycle. Following this, the data from the external memory could be brought to the R bus through the A port of the ALU and added to the accumulator value which is placed on the D bus and brought through the B port of the ALU. The result would be placed in the accumulator. This operation would complete the example and we would be ready to fetch the next instruction. As can be seen, a number of microcycles are required to fetch the instruction, decode it, fetch the data and execute the instruction. One of the best ways to understand the flow needed to implement a typical instruction set is shown in Figure 11. Here, we see the basic instruction fetch and decode operation followed by the path used to execute each of the various instructions. Then, we see a return to the fetch operation to fetch the next instruction.

Certainly from this discussion we can see how three registers have enhanced the performance of the simple ALU/accumulator data path shown in Figure 7. Typically, even more registers than shown in Figure 10 are needed if we are to increase the power of

our machine. If we examine the block diagram of Figure 12, we see a similar architecture to that as shown in Figure 10. Here, the number of working registers has been expanded to sixteen at the output of the ALU. These can be used to provide a program counter function and a number of accumulator functions simultaneously. In addition, note that the registers have two output ports such that the simultaneous selection of any two of the sixteen registers is possible. Both of these registers can be presented to the ALU so that operations on two registers simultaneously can be executed. In addition, a data input multiplexer is available at the A port of the ALU such that external data can be brought in to the configuration. Likewise, there is an output multiplexer such that either the A output of the registers or the ALU output can be selected. This output multiplexer is used to provide a data out port and the output can also be loaded into memory address register to provide an address as required. Thus, the architecture of Figure 12 is quite similar to that of Figure 10 except that the number of registers has been increased to provide additional flexibility.

If we assume that one of the sixteen registers inside of this register file is to be used as the program counter, we see that the program counter can be brought out of the A output port and loaded into the memory address register and at the same time it can also be brought out the B output port and incremented in ALU and reloaded into the register file. In this architecture it appears the A output of the register stack can also be brought to the input multiplexer and the A port of the ALU and incremented via that path and reloaded into the registers. While this is possible in the architecture of Figure 12, we are leading up to the implementation of an Am2901A and this path is not needed in the Am2901A. Thus, we can implement functions and operations in the diagram of Figure 12 just as we could in the diagram of Figure 10. However, what was previously performed in two microcycles can now be performed in one microcycle. That is, the MAR can be loaded with the current value of the PC and at the same time the PC can be incremented and the new value restored in the PC register.

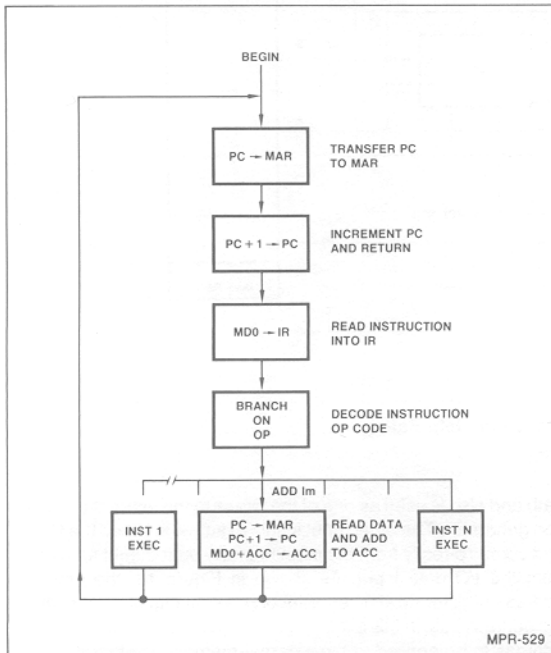


Figure 11. Steps for ADD Instruction.

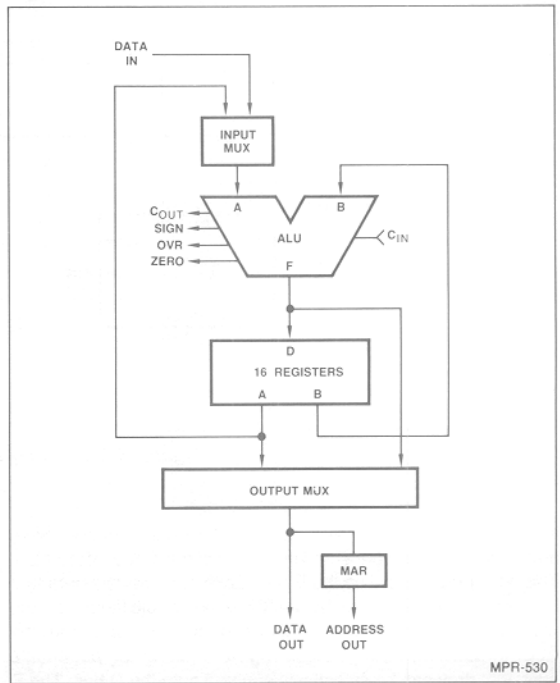


Figure 12. Multi-Register ALU.

Another feature of the block diagram of Figure 12 is the depiction of the carry in to the ALU and the four output flags associated with the ALU. Here, carry in is the normal carry in as needed in any adder such that the device is cascadable. In addition, certain kinds of arithmetic functions such two's complement arithmetic also need the ability to provide a carry in for certain operations. The most common is two's complement subtract which is usually performed by complementing the operand to be subtracted, adding and adding one at the carry in. Also, the ALU shows the four output flags usually associated with a typical minicomputer. These are the carry output, the sign bit, the overflow detect, and the zero detect. These four status flags are used to determine various things about the operation being performed. The carry out flag and overflow flag are as described in the previous sections of this chapter. They provide the carry and overflow information about the addition.

The sign bit is simply the most significant bit of the ALU and represents the sign of a two's complement number. That is, when the sign bit is LOW, we assume the two's complement number is positive and when the sign bit is HIGH, we assume the two's complement number is negative. Thus, the sign bit is active HIGH and carries negative weight as we assume in any standard two's complement number representation. If the reader is unfamiliar with two's complement number notations, a discussion of this topic can be found in an application note entitled "The Am25S05, Am2505 and Am25L05 Schottky, Standard and Low Power TTL Two's Complement Digital Multipliers" as found in Advanced Micro Devices' Schottky and Low Power Schottky Data Book dated 10/77. This application note begins on page 5-49 and fully details two's complement number notation and gives examples.

The fourth status flag is called the zero flag and again is just what the name implies. This flag represents the fact that all of the ALU outputs are at logic zero. In this design, a logic zero means that all of the ALU output bits are LOW.

If the architecture of Figure 12 is extended a little more, we will arrive at the Am2901A as depicted in Figure 13. Here, we have redrawn the structure so that the registers are placed above the ALU; however, the function is identical. Two new functions have been added to this block diagram that have not previously been discussed. These are the RAM shift matrix located directly above the sixteen registers now described as a 16 x 4 dual port RAM. The purpose of the RAM shift network is to allow the ability of shifting the data word to be written into the register either up one bit position or down one bit position. The second function added to the block diagram is that of the Q register and shift network. Here, the Q register is used as an auxiliary register such that double length operations can be performed and it is also used in the multiply and divide algorithms. In addition, the shift network allows the Q register contents to be shifted up one bit position or shifted down one bit position. In addition, it should be pointed out that the memory address register is not part of the Am2901A. This is because there were not enough pins on the package to implement the function and the additional power required by the output buffers would have reduced the performance of the ALU and register stack. Instead, this function is being designed into other 2900 family products.

Am2901A ARCHITECTURE

A detailed block diagram of the Am2901A bipolar microprogrammable microprocessor structure is shown in Figure 14. The circuit is a four-bit slice cascadable to any number of bits. Therefore, all data paths within the circuit are four bits wide. The two key elements in the Figure 14 block diagram are the 16-word by 4-bit 2-port RAM and the high-speed ALU.

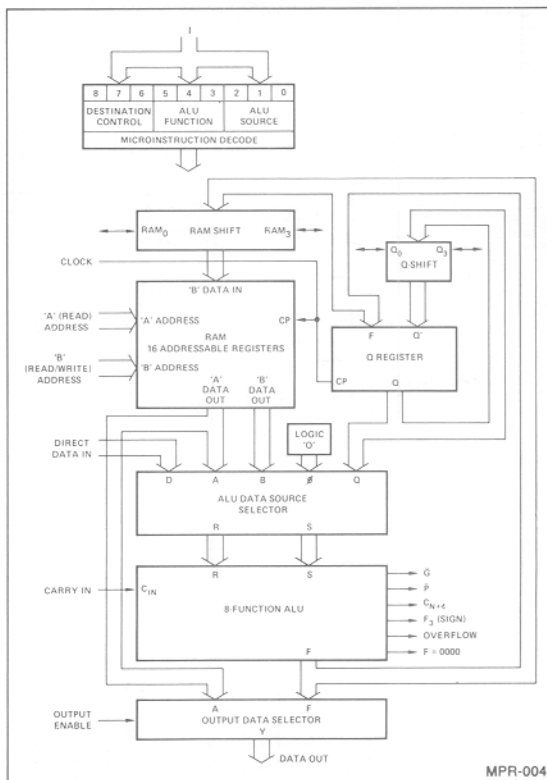


Figure 13. Am2901A Block Diagram.

Data in any of the 16 words of the Random Access Memory (RAM) can be read from the A-port of the RAM as controlled by the 4-bit A address field input. Likewise, data in any of the 16 words of the RAM as defined by the B address field input can be simultaneously read from the B-port of the RAM. The same code can be applied to the A select field and B select field in which case the identical file data will appear at both the RAM A-port and B-port outputs simultaneously.

When enabled by the RAM write enable (RAM EN), new data is always written into the file (word) defined by the B address field of the RAM. The RAM data input field is driven by a 3-input multiplexer. This configuration is used to shift the ALU output data (F) if desired. This three-input multiplexer scheme allows the data to be shifted up one bit position, shifted down one bit position, or not shifted in either direction.

The RAM A-port data outputs and RAM B-port data outputs drive separate 4-bit latches. These latches hold the RAM data while the clock input is LOW. This eliminates any possible race conditions that could occur while new data is being written into the RAM.

The high-speed Arithmetic Logic Unit (ALU) can perform three binary arithmetic and five logic operations on the two 4-bit input words R and S. The R input field is driven from a 2-input multiplexer, while the S input field is driven from a 3-input multiplexer. Both multiplexers also have an inhibit capability; that is, no data is passed. This is equivalent to a "zero" source operand.

Referring to Figure 14, the ALU R-input multiplexer has the RAM A-port and the direct data inputs (D) connected as inputs. Likewise, the ALU S-input multiplexer has the RAM A-port, the RAM B-port and the Q register connected as inputs.

This multiplexer scheme gives the capability of selecting various pairs of the A, B, D, Q and "0" inputs as source operands to the ALU. These five inputs, when taken two at a time, result in ten possible combinations of source operand pairs. These combinations include AB, AD, AQ, A0, BD, BQ, B0, DQ, D0 and Q0. It is apparent that AD, AQ and A0 are somewhat redundant with BD, BQ and B0 in that if the A address and B address are the same, the identical function results. Thus, there are only seven completely non-redundant source operand pairs for the ALU. The Am2901A microprocessor implements eight of these pairs. The microinstruction inputs used to select the ALU source operands are the I_0 , I_1 and I_2 inputs.

The two source operands not fully described as yet are the D input and Q input. The D input is the four-bit wide direct data field input. This port is used to insert all data into the working registers inside the device. Likewise, this input can be used in the ALU to modify any of the internal data files. The Q register is a separate 4-bit file intended primarily for multiplication and division routines but it can also be used as an accumulator or holding register for some applications.

The ALU itself is a high-speed arithmetic/logic operator capable of performing three binary arithmetic and five logic functions. The I_3 , I_4 and I_5 microinstruction inputs are used to select the ALU function. The definition of these functions is shown in Figure 15. The normal technique for cascading the ALU of several devices is in a look-ahead carry mode. Carry generate, \bar{G} , and carry propagate, \bar{P} , are outputs of the device for use with a carry-look-ahead-generator such as the Am2902A ('182). A carry-out, C_{n+4} , is also generated and is available as an output for use as the carry flag in a status register. Both carry-in (C_n) and carry-out (C_{n+4}) are active HIGH.

SOURCE OPERANDS		DESTINATION		
A, B	B, 0	SHIFT	LOAD	Y-OUT
A, D	D, 0	UP	RAM	F
A, Q	Q, 0	UP	RAM & Q	F
A, 0	D, Q	DOWN	RAM	F
		DOWN	RAM & Q	F
		NONE	NONE	F
		NONE	Q	F
		NONE	RAM	F
		NONE	RAM	A
ALU FUNCTIONS				
R+S	R OR S			
R-S	R AND S			
S-R	R EXOR S			
	R EXNOR S			
	R AND S			

Figure 15. Am2901A Microinstruction Control.

The ALU has three other status-oriented outputs. These are F_3 , $F = 0$, and overflow (OVR). The F_3 output is the most significant (sign) bit of the ALU and can be used to determine positive or negative results without enabling the three-state data outputs. F_3 is non-inverted with respect to the sign bit output Y_3 . The $F = 0$ output is used for zero detect. It is an open-collector output and can be wire OR'ed between microprocessor slices. $F = 0$ is HIGH when all F outputs are LOW. The overflow output (OVR) is used to flag arithmetic operations that exceed the available two's complement number range. The overflow output (OVR) is HIGH when overflow exists; that is, when C_{n+3} and C_{n+4} are not the same polarity.

The ALU data output is routed to several destinations. It can be a data output of the device and it can also be stored in the RAM or the Q register. Eight possible combinations of ALU destination functions are available as defined by the I_6 , I_7 and I_8 microinstruction inputs. These combinations are shown in Figure 15.

The four-bit data output field (Y) features three-state outputs and can be directly bus organized. An output control (\bar{OE}) is used to enable the three-state outputs. When \bar{OE} is HIGH, the Y outputs are in the high-impedance state.

A two-input multiplexer is also used at the data output such that either the A-port of the RAM or the ALU outputs (F) are selected at the device Y outputs. This selection is controlled by the I_6 , I_7 and I_8 microinstruction inputs.

As was discussed previously, the RAM inputs are driven from a three-input multiplexer. This allows the ALU outputs to be entered non-shifted, shifted up one position (X2) or shifted down one position ($\div 2$). The shifter has two ports; one is labeled RAM_0 and the other is labeled RAM_3 . Both of these ports consist of a buffer-driver with a three-state output and an input to the multiplexer. Thus, in the shift up mode, the RAM_3 buffer is enabled and the RAM_0 multiplexer input is enabled. Likewise, in the shift down mode, the RAM_0 buffer and RAM_3 input are enabled. In the no-shift mode, both buffers are in the high-impedance state and the multiplexer inputs are not selected. This shifter is controlled from the I_6 , I_7 and I_8 microinstruction inputs.

Similarly, the Q register is driven from a 3-input multiplexer. In the no-shift mode, the multiplexer enters the ALU data into the Q register. In either the shift-up or shift-down mode, the multiplexer selects the Q register data appropriately shifted up or down. The Q shifter also has two ports; one is labeled Q_0 and the other is Q_3 . The operation of these two ports is similar to the RAM shifter and is also controlled from I_6 , I_7 and I_8 .

The clock input to the Am2901A controls the RAM, the Q register, and the A and B data latches. When enabled, data is clocked into the Q register on the LOW-to-HIGH transition of the clock. When the clock input is HIGH, the A and B latches are open and will pass whatever data is present at the RAM outputs. When the clock input is LOW, the latches are closed and will retain the last data entered. If the RAM-EN is enabled, new data will be written into the RAM file (word) defined by the B address field when the clock input is LOW.

Am2903 GENERAL DESCRIPTION

The Am2903 is a four-bit expandable bipolar microprocessor slice that performs all functions performed by the industry standard Am2901A. In addition, it provides a number of significant enhancements that are especially useful in arithmetic oriented processors. The Am2903 contains sixteen internal working registers arranged in a two address architecture and it also provides all of the necessary signals to expand the register file externally using the Am29705 register stack. Any number of registers can be cascaded to the Am2903 using this technique. In addition to its complete arithmetic and logic instruction set, the Am2903 provides a special set of instructions which facilitate the implementation of multiplication, division, normalization and other previously time consuming operations such as parity generation and sign extension. A block diagram of the Am2903 is shown in Figure 16.

ARCHITECTURE OF THE Am2903

The Am2903 is a high-performance, cascadable, four-bit bipolar microprocessor slice designed for use in CPU's, peripheral controllers, microprogrammable machines, and numerous other applications. The microinstruction flexibility of the Am2903 allows the efficient emulation of almost any digital computing machine.

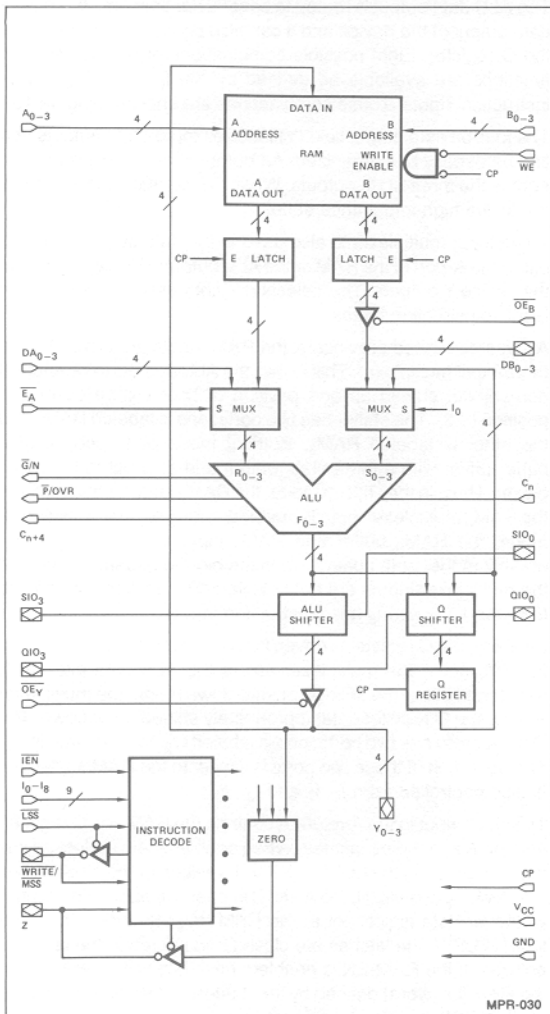


Figure 16. Basic Am2903 Block Diagram.

The nine-bit microinstruction selects the ALU sources, function, and destination. The Am2903 is cascadable with full lookahead or ripple carry, has three-state outputs, and provides various ALU status flag outputs. Advanced Low-Power Schottky processing is used to fabricate this 48-pin LSI circuit.

All data paths within the device are four bits wide. As shown in the block diagram of Figure 16, the device consists of a 16-word by 4-bit, two-port RAM with latches on both output ports, a high-performance ALU and shifter, a multi-purpose Q Register with shifter input, and a nine-bit instruction decoder.

Two-Port RAM

Any two RAM words addressed at the A and B address ports can be read simultaneously at the respective RAM A and B output ports. Identical data appear at the two output ports when the same address is applied to both address ports. The latches at the RAM output ports are transparent when the clock input, CP, is HIGH and they hold the RAM output data when CP is LOW. Under control of the \overline{OE}_B three-state output enable, RAM data can be read directly at the Am2903 DB I/O port.

External data at the Am2903 Y I/O port can be written directly into the RAM, or ALU shifter output data can be enabled onto the Y I/O port and entered into the RAM. Data is written into the RAM at the B address when the write enable input, \overline{WE} , is LOW and the clock input, CP, is LOW.

Arithmetic Logic Unit

The Am2903 high-performance ALU can perform seven arithmetic and nine logic operations on two 4-bit operands. Multiplexers at the ALU inputs provide the capability to select various pairs of ALU source operands. The \overline{E}_A input selects either the DA external data input or RAM output port A for use as one ALU operand and the \overline{OE}_B and I_0 inputs select RAM output port B, DB external data input, or the Q Register content for use as the second ALU operand. Also, during some ALU operations, zeros are forced at the ALU operand inputs. Thus, the Am2903 ALU can operate on data from two external sources, from an internal and external source, or from two internal sources.

When instruction bits I_4, I_3, I_2, I_1 and I_0 are LOW, the Am2903 executes special functions. Figure 17 defines these special functions and the operation which the ALU performs for each. When the Am2903 executes instructions other than the nine special functions, the ALU operation is determined by instruction bits I_4, I_3, I_2 and I_1 . Figure 18 defines the ALU operation as a function of these four instruction bits.

Am2903s may be cascaded in either a ripple carry or lookahead carry fashion. When a number of Am2903s are cascaded, each slice must be programmed to be a most significant slice (MSS), intermediate slice (IS), or least significant slice (LSS) of the array. The carry generate, \overline{G} , and carry propagate, \overline{P} , signals required for a lookahead carry scheme are generated by the Am2903 and are available as outputs of the least significant and intermediate slices.

The Am2903 also generates a carry-out signal, C_{n+4} , which is generally available as an output of each slice. Both the carry-in, C_n , and carry-out, C_{n+4} , signals are active HIGH. The ALU generates two other status outputs. These are negative, N, and overflow, OVR. The N output is generally the most significant (sign) bit of the ALU output and can be used to determine positive or negative results. The OVR output indicates that the arithmetic operation being performed exceeds the available two's complement number range. The N and OVR signals are available as outputs of the most significant slice. Thus, the multi-purpose \overline{G}/N and \overline{P}/OVR outputs indicate \overline{G} and \overline{P} at the least significant and intermediate slices, and sign and overflow at the most significant slice. To some extent, the meaning of the C_{n+4} , \overline{P}/OVR , and \overline{G}/N signals vary with the ALU function being performed.

ALU Shifter

Under instruction control, the ALU shifter passes the ALU output (F) non-shifted, shifts it up one bit position (2F), or shifts it down one bit position (F/2). Both arithmetic and logical shift operations are possible. An arithmetic shift operation shifts data around the most significant (sign) bit position of the most significant slice, and a logical shift operation shifts data through this bit position (see Figure 19). SIO_0 and SIO_3 are bidirectional serial shift inputs/outputs. During a shift-up operation, SIO_0 is generally a serial shift input and SIO_3 a serial shift output. During a shift-down operation, SIO_3 is generally a serial shift input and SIO_0 a serial shift output.

The ALU shifter also provides the capability to sign extend at slice boundaries. Under instruction control, the SIO_0 (sign) input can be extended through Y_0, Y_1, Y_2, Y_3 and propagated to the SIO_3 output.

I ₈	I ₇	I ₆	I ₅	Hex Code	Special Function	ALU Function	ALU Shifter Function	SIO ₃		SIO ₀	Q Reg & Shifter Function	QIO ₃	QIO ₀	WRITE
								Most Sig. Slice	Other Slices					
L	L	L	L	0	Unsigned Multiply	F = S + C _n if Z = L F = R + S + C _n if Z = H	Log. F/2 → Y (Note 1)	Hi-Z	Input	F ₀	Log. Q/2 → Q	Input	Q ₀	L
L	L	H	L	2	Two's Complement Multiply	F = S + C _n if Z = L F = R + S + C _n if Z = H	Log. F/2 → Y (Note 2)	Hi-Z	Input	F ₀	Log. Q/2 → Q	Input	Q ₀	L
L	H	L	L	4	Increment by One or Two	F = S + 1 + C _n	F → Y	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	L	H	5	Sign/Magnitude-Two's Complement	F = S + C _n if Z = L F = S + C _n if Z = H	F → Y (Note 3)	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L	H	H	L	6	Two's Complement Multiply, Last Cycle	F = S + C _n if Z = L F = S - R - 1 + C _n if Z = H	Log. F/2 → Y (Note 2)	Hi-Z	Input	F ₀	Log. Q/2 → Q	Input	Q ₀	L
H	L	L	L	8	Single Length Normalize	F = S + C _n	F → Y	F ₃	F ₃	Hi-Z	Log. 2Q → Q	Q ₃	Input	L
H	L	H	L	A	Double Length Normalize and First Divide Op.	F = S + C _n	Log 2F → Y	R ₃ ∨ F ₃	F ₃	Input	Log. 2Q → Q	Q ₃	Input	L
H	H	L	L	C	Two's Complement Divide	F = S + R + C _n if Z = L F = S - R - 1 + C _n if Z = H	Log. 2F → Y	R ₃ ∨ F ₃	F ₃	Input	Log. 2Q → Q	Q ₃	Input	L
H	H	H	L	E	Two's Complement Divide, Correction and Remainder	F = S + R + C _n if Z = L F = S - R - 1 + C _n if Z = H	F → Y	F ₃	F ₃	Hi-Z	Log. 2Q → Q	Q ₃	Input	L

NOTES: 1. At the most significant slice only, the C_{n+4} signal is internally gated to the Y₃ output.
 2. At the most significant slice only, F₃ ∨ OVR is internally gated to the Y₃ output.
 3. At the most significant slice only, S₃ ∨ F₃ is generated at the Y₃ output.
 4. Op codes 1, 3, 7, 9, B, D, and F are reserved for future use.

L = LOW
 H = HIGH
 X = Don't Care
 Hi-Z = High Impedance
 ∨ = Exclusive OR
 Parity = SIO₃ ∨ F₃ ∨ F₂ ∨ F₁ ∨ F₀

Figure 17. Special Functions: I₀ = I₁ = I₂ = I₃ = I₄ = LOW, IEN = LOW.

I ₄	I ₃	I ₂	I ₁	Hex Code	ALU Functions
L	L	L	L	0	I ₀ = L Special Functions I ₀ = H F _i = HIGH
L	L	L	H	1	F = S Minus R Minus 1 Plus C _n
L	L	H	L	2	F = R Minus S Minus 1 Plus C _n
L	L	H	H	3	F = R Plus S Plus C _n
L	H	L	L	4	F = S Plus C _n
L	H	L	H	5	F = S̄ Plus C _n
L	H	H	L	6	F = R Plus C _n
L	H	H	H	7	F = R̄ Plus C _n
H	L	L	L	8	F _i = LOW
H	L	L	H	9	F _i = R̄ _i AND S _i
H	L	H	L	A	F _i = R _i EXCLUSIVE NOR S _i
H	L	H	H	B	F _i = R _i EXCLUSIVE OR S _i
H	H	L	L	C	F _i = R _i AND S _i
H	H	L	H	D	F _i = R _i NOR S _i
H	H	H	L	E	F _i = R _i NAND S _i
H	H	H	H	F	F _i = R _i OR S _i

L = LOW H = HIGH i = 0 to 3

Figure 18. ALU Functions.

A cascaded, five-bit parity generator/checker is designed into the Am2903 ALU shifter and provides ALU error detection capability. Parity for the F₀, F₁, F₂, F₃ ALU outputs and SIO₃ input is generated and, under instruction control, is made available at the SIO₀ output.

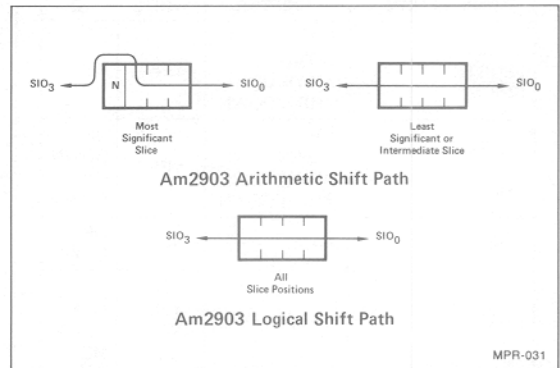


Figure 19.

The instruction inputs determine the ALU shifter operation. Figure 17 defines the special functions and the operation the ALU shifter performs for each. When the Am2903 executes instructions other than the nine special functions, the ALU shifter operation is determined by instruction bits I₈I₇I₆I₅. Figure 20 defines the ALU shifter operation as a function of these four bits.

Q Register

The Q Register is an auxiliary four-bit register which is clocked on the LOW-to-HIGH transition of the CP input. It is intended primarily for use in multiplication and division operations; however, it can also be used as an accumulator or holding register for some applications. The ALU output, F, can be loaded into the Q Register, and/or the Q Register can be selected as the source for the ALU S operand. The shifter at the input to the Q Register provides

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

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

L = LOW
 H = HIGH

Hi-Z = High Impedance

Figure 20a. ALU Destination Control for I₀ or I₁ or I₂ or I₃ or I₄ = HIGH, \overline{IEN} = LOW.

OPERATION		ALU SHIFTER	RAM WRITE	Q
SINGLE LENGTH SHIFT		UP DOWN ARITH UP ARITH DOWN	YES	NC
DOUBLE LENGTH SHIFT		UP DOWN ARITH UP ARITH DOWN	YES	UP DOWN UP DOWN
Q-SHIFT		PASS	NO	UP DOWN
LOAD	RAM	PASS	YES	NC
	RAM & Q		YES	LOAD
	Q		NO	LOAD
SIGN EXTEND		SIO ₀	YES	NC

NC = No Change

Figure 20b. Am2903 ALU Destination Control Summary.

the capability to shift the Q Register contents up one bit position (2Q) or down one bit position (Q/2). Only logical shifts are performed. QIO₀ and QIO₃ are bidirectional shift serial inputs/outputs. During a Q Register shift-up operation, QIO₀ is a serial shift input and QIO₃ is a serial shift output. During a shift-down operation, QIO₃ is a serial shift input and QIO₀ is a serial shift output.

Double-length arithmetic and logical shifting capability is provided by the Am2903. The double-length shift is performed by connecting QIO₃ of the most significant slice to SIO₀ of the least significant slice, and executing an instruction which shifts both the ALU output and the Q Register.

The Q Register and shifter operation is controlled by instruction bits I₈I₇I₆I₅. Figures 17 and 20 define the Q Register and shifter operation as a function of these four bits.

Output Buffers

The DB and Y ports are bidirectional I/O ports driven by three-state output buffers with external output enable controls. The Y output buffers are enabled when the \overline{OE}_Y input is LOW and are in the high-impedance state when \overline{OE}_Y is HIGH. Likewise, the DB output buffers are enabled when the \overline{OE}_B input is LOW and in the high-impedance state when \overline{OE}_B is HIGH.

The zero, Z, pin is an open collector input/output that can be wire-OR'ed between slices. As an output it can be used as a zero detect status flag and generally indicates that the Y₀₋₃ pins are all LOW, whether they are driven from the Y output buffers or from an external source connected to the Y₀₋₃ pins. To some extent the meaning of this signal varies with the instruction being performed.

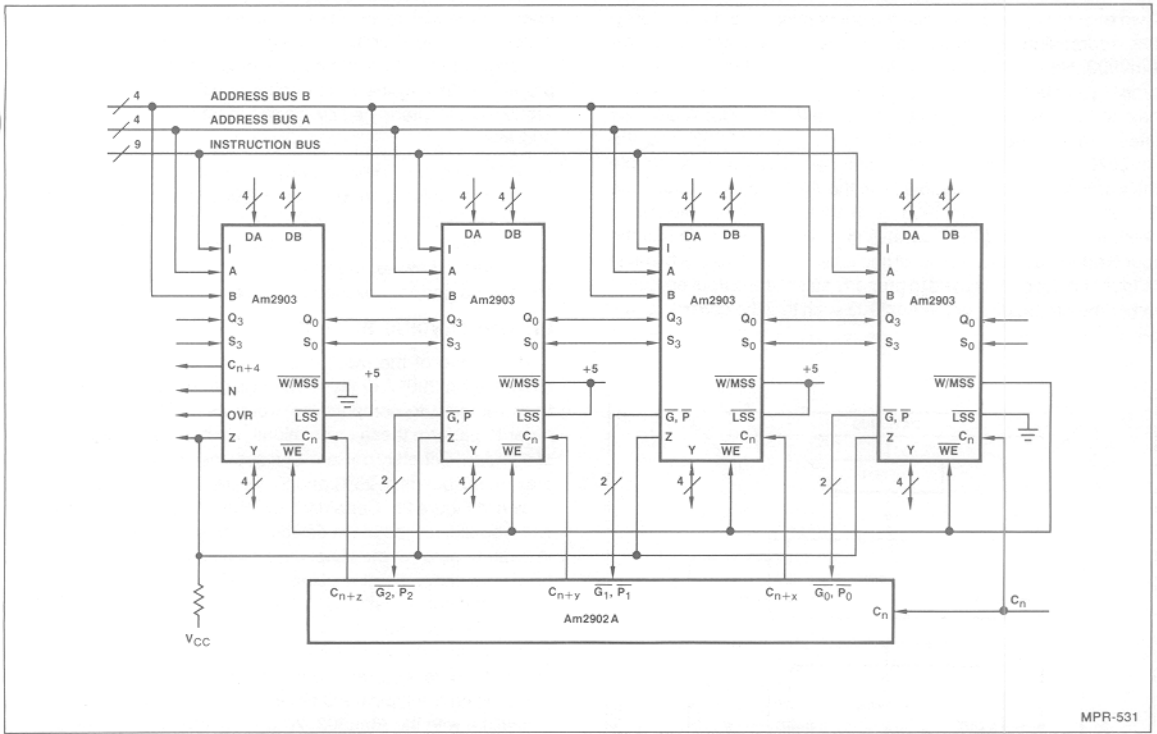
Instruction Decoder

The Instruction Decoder generates required internal control signals as a function of the nine Instruction inputs, I₀₋₈; the Instruction Enable input, \overline{IEN} ; the LSS input; and the WRITE/MS input/output. The \overline{WRITE} output is LOW when an instruction which writes data into the RAM is being executed.

When \overline{IEN} is LOW, the \overline{WRITE} output is enabled and the Q Register and Sign Compare Flip-Flop can be written according to the Am2903 instruction. The Sign Compare Flip-Flop is an on-chip flip-flop which is used during an Am2903 divide operation.

Programming the Am2903 Slice Position

Tying the LSS input LOW programs the slice to operate as a least significant slice (LSS) and enables the \overline{WRITE} output signal onto the WRITE/MSS bidirectional I/O pin. When LSS is tied HIGH, the WRITE/MSS pin becomes an input pin; tying the WRITE/MSS pin HIGH programs the slice to operate as an intermediate slice (IS) and tying it LOW programs the slice to operate as a most significant slice (MSS). This is shown in Figure 21.



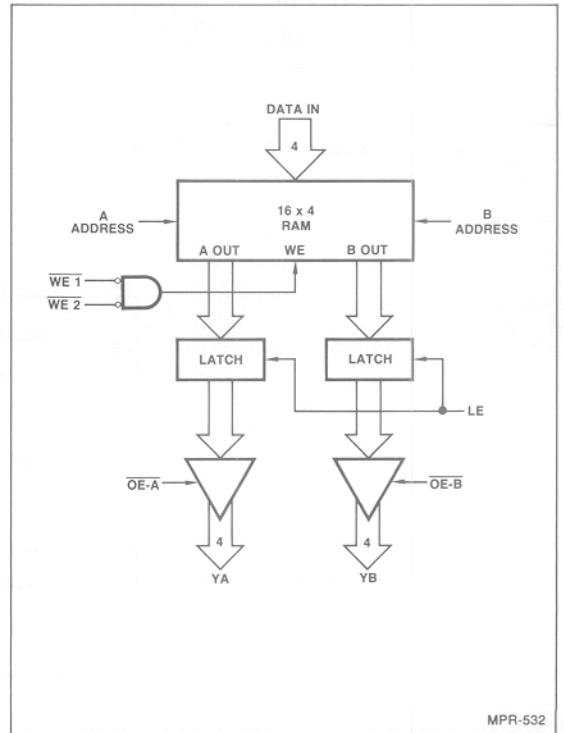
MPR-531

Figure 21. Am2903 - 16-Bit CPU with Carry Look Ahead.

EXPANDING THE NUMBER OF Am2903 REGISTERS

The Am2903 contains 16 internal working registers configured in a standard two port architecture. The number of working registers in the ALU configuration can be increased by utilizing the Am29705 16-word by 4-bit two-port RAM. Any number of Am29705's can be connected to the Am2903 to increase the number of working registers. Figure 22 shows a block diagram of the basic Am29705. As is seen, the device consists of a 16 word by 4 bit two port RAM with latches at the A and B outputs similar to the RAM contained within the Am2903. Each of the latch outputs has three state drivers capable of driving the DA and DB inputs of the Am2903. The Am29705 is a non-inverting device. That is, data presented at the inputs is stored in the RAM and when brought to the RAM outputs, it is non-inverted from when it was originally brought into the device.

The technique for using the Am29705 to expand the number of registers in the Am2903 can best be visualized by referring to Figures 23 and 24 simultaneously. In Figure 23, the data bus connections are shown such that the Am2903 Y output is used to drive the Am29705 inputs. Here, we also assume this bus may be tied to a data bus through a bi-directional buffer. In Figure 23, the A outputs of the Am29705 are connected together and also connected to the DA input of the Am2903. Likewise, the B outputs from the Am29705 are also shown connected to the DB inputs of the Am2903. In all cases, we are assuming 16-bit data busses. Thus, four Am2903's are assumed and eight Am29705's are assumed. As shown in Figure 23, one of the write enable inputs to the Am29705 is tied to the latch enable input of the Am2903. This allows the latches in the Am29705 to perform identically to those in the Am2903.



MPR-532

Figure 22. Am29705 Block Diagram.

If we refer to Figure 24, we see the connections required to set up the addressing for additional registers associated with the Am2903. Here, three two-line to four-line decoders are used to properly control the A address, B address and write enable signals to the devices. As shown in Figure 24, the four A address lines are all tied in parallel between the Am2903 and the Am29705's. The two-line to four-line decoder is used to enable the appropriate output enable from the Am29705's or switch the EA MUX inside the Am2903 such that the proper register is selected. The B address operates in a similar fashion in that the four B address lines are also all tied together. Likewise, a two-line to four-line decoder is used to properly select the output enable of either the Am29705's or the Am2903 such that the correct source

operand register is selected. In addition, a two-line to four-line decoder is used to control the write enable signal such that only one register is written into as a destination. This is controlled by properly selecting the write enable of either the Am2903 or the Am29705 as determined by the two most significant bits of the B address.

If this technique is used properly, any number of Am29705's can be used in conjunction with the Am2903. It may be necessary to use either a three-line to eight-line decoder or perhaps even a larger circuit to decode the more significant bits of the A and B addresses. Likewise, the write enable signal must be controlled so that the correct destination register will be written.

UNDERSTANDING BIT SLICE TIMING

Perhaps one of the most important aspects of designing with either the Am2901A or the Am2903 is understanding the calculations required to compute the worst case AC performance. In order to perform these calculations, we have selected a number of standard Schottky devices and assigned minimum, typical and maximum speeds at 25°C and 5V for use in these calculations as shown in Figure 25. Certainly the design engineer should use the exact specifications of the devices he has selected for his design in order to perform the worst case calculations. What is intended here is an understanding of the technique to perform these calculations and some method to allow a comparison of the Am2901A and Am2903 in terms of their AC performance. Since at the time of this writing the Am2903 is still being characterized, only the typical AC data is currently available. Thus, all calculations will be made using the typical AC times such that we can compare the Am2901A with the Am2903. When final characterization data on the Am2903 is available, the designer can then compute his performance by selecting the appropriate temperature range and power supply variations as required by his design.

Figure 26 shows the typical AC calculations for the functions usually considered in an Am2901A design. These functions are usually the speed for a logic operation, arithmetic operation, logic operation with shift and arithmetic operation with shift. In each case, we are computing speeds from the LOW-to-HIGH transition of a clock through an entire microcycle to the next LOW-to-HIGH transition of a clock.

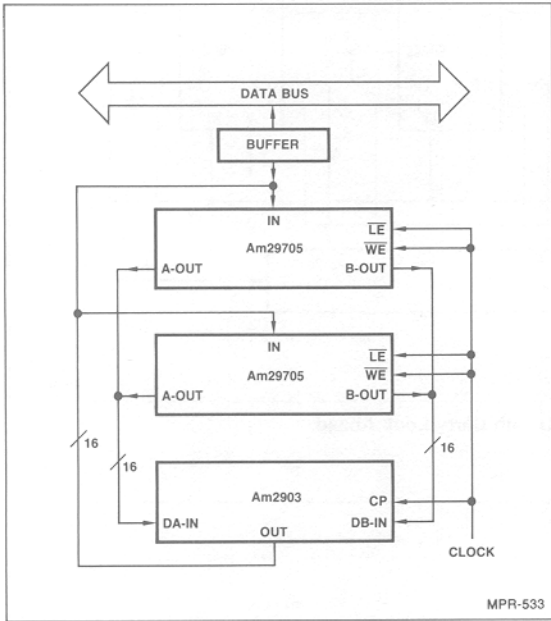


Figure 23. Am2903 - Data Bus Cascading.

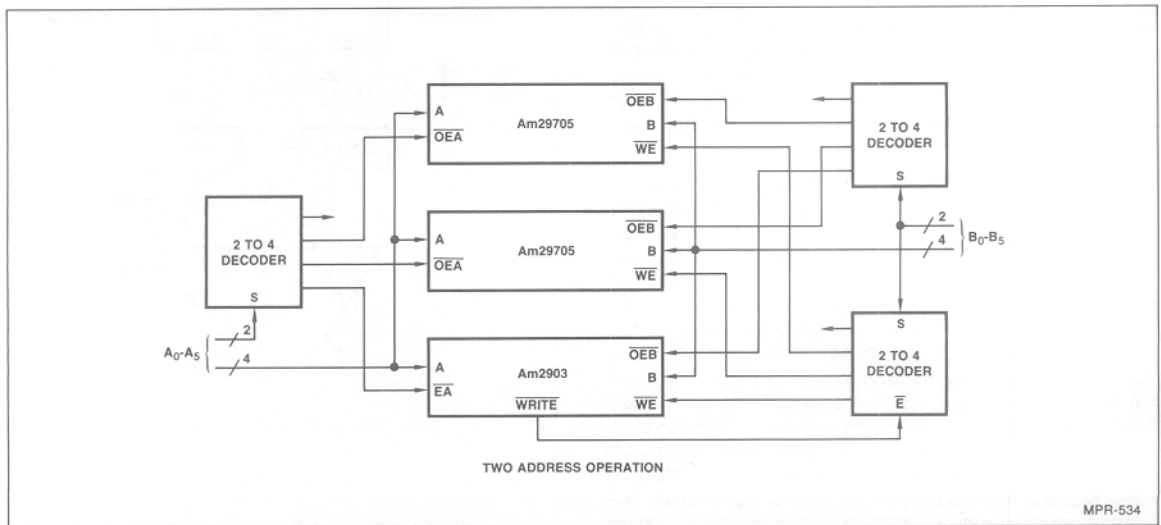


Figure 24. Am2903 - RAM Address Cascading.

DEVICE & PATH	MIN.	TYP.	MAX.
S Register			
Clock to Output		9	15
OE to Output		13	20
Set-Up	5	2	
S MUX			
Data to Output		5	8
Select to Output		12	18
OE to Output		13	20
Microprogram PROM			
Address to Output		30	50
OE to Output		18	25
Mapping PROM			
Address to Output		25	45
OE to Output		18	25
Decoder			
Select to Output		8	12
Counter			
Clock to Q		9	13
Clock to TC		12	18
CET to TC		8	12
Data Set-Up	8	4	
Load Set-Up	16	10	
CEP or CET Set-Up	12	7	
S-EXOR			
IN to OUT		7	11
Am2922			
Clock to Output		21	32
Data to Output		13	19
OE to Output		10	17
Data Set-Up	10	5	
Am29811A			
Input to Output		25	35
Am29803A			
Input to Output		25	35
Am2902A			
C_n to $C_{n+x,y,z}$		7	11
G, P to G, P		7	10
G, P to $C_{n+x,y,z}$		5	7

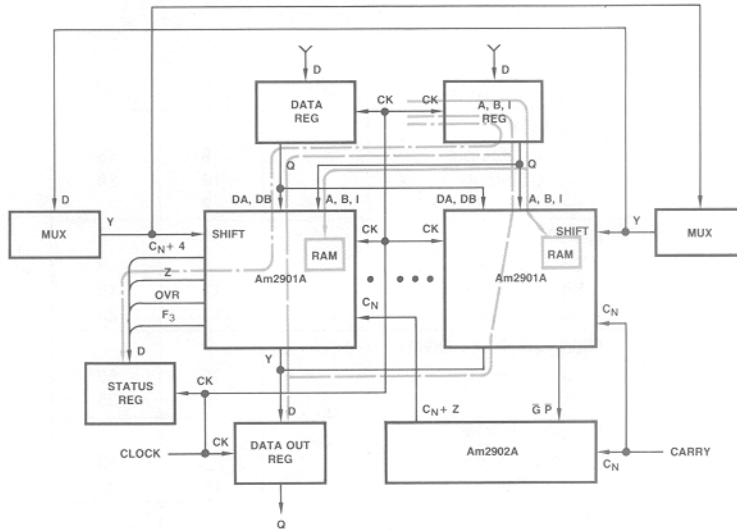
Figure 25. Standard Device Schottky Speeds.

Similarly, Figure 27 shows the same type of computations for an Am2903 system. There is one very important distinction that should be made in computing the timing of an Am2903 16-bit ALU when compared with an Am2901A ALU in that in the Am2903, the shifter is at the output of the ALU and is followed by the zero detector. Thus, in an Am2903 design, the flags are no longer

independent of the shift operation. This is easily seen in Figure 27.

By way of comparison, Figure 28 shows speeds for the four types of operations for the Am2901A 16-bit system as compared with the Am2903 16-bit system.

a)



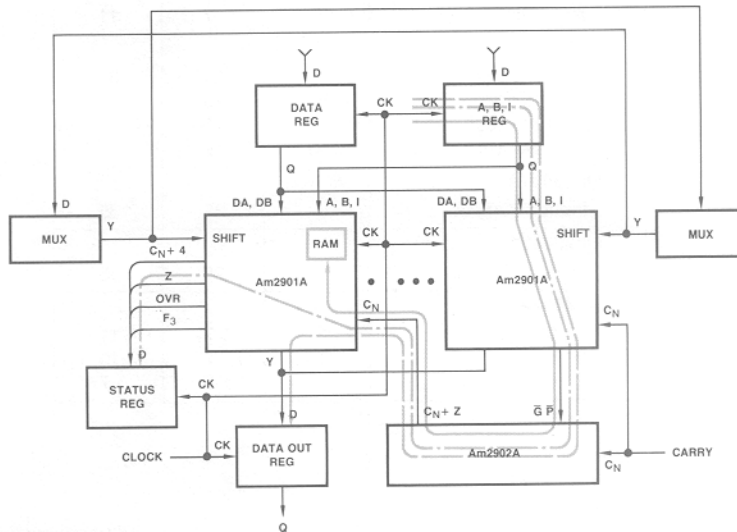
**LOGIC OPERATION
SPEED COMPUTATIONS**

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to Q	9	9	9
2901A	READ-MODIFY-WRITE	55	-	-
2901A	AB - Y	-	45	-
2901A	AB - Zero	-	-	65
S-REG	SET-UP D	-	2	2
TOTAL-ns		64	56	76

PATH 1 —————
 PATH 2 - - - - -
 PATH 3 - · - - -

MPR-535

b)



**ARITHMETIC OPERATION
SPEED COMPUTATIONS**

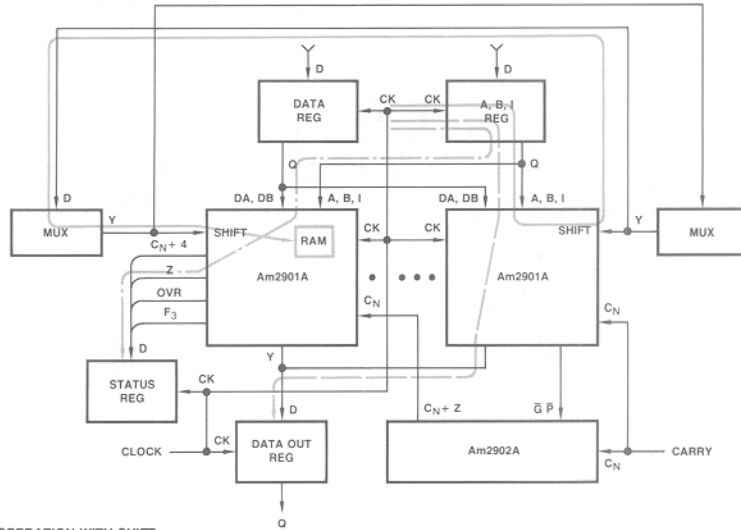
DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S-REG	CP to Q	9	9	9
2901A	AB to GP	40	40	40
2902A	GP to C _{N+xyz}	5	5	5
2901A	SET-UP C _n	40	-	-
2901A	C _n to Y	-	20	-
2901A	C _n to Zero	-	-	35
S-REG	SET-UP D	-	2	2
TOTAL-ns		94	76	91

PATH 1 —————
 PATH 2 - - - - -
 PATH 3 - · - - -

MPR-536

Figure 26. Typical AC Calculations for the Am2901A.

c)



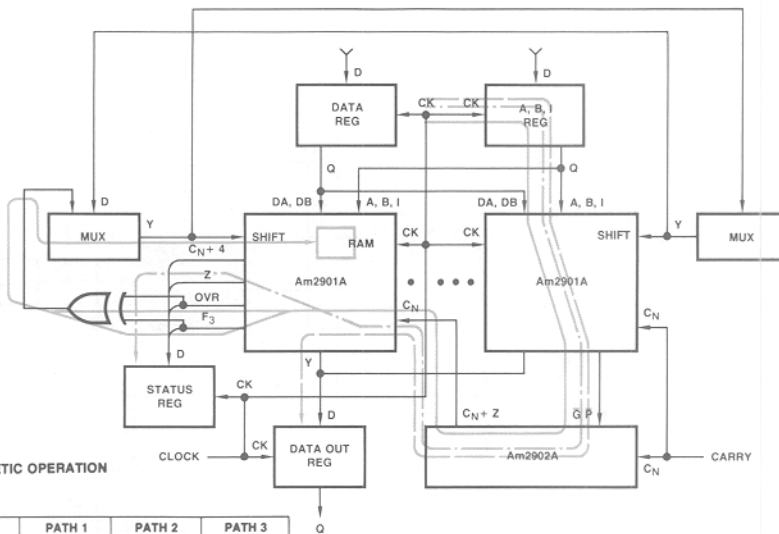
LOGIC OPERATION WITH SHIFT
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to Q	9	9	9
2901A	AB to RAM ₀₃	60	-	-
S-MUX	D to Y	5	-	-
2901A	SET-UP RAM ₀₃	15	-	-
2901A	AB to Y	-	45	-
2901A	AB to Z	-	-	65
S-REG	SET-UP D	-	2	2
TOTAL-ns		89	56	76

PATH 1 ———
PATH 2 - - -
PATH 3 - - -

MPR-537

d)



TWO'S COMPLEMENT ARITHMETIC OPERATION
WITH SHIFT DOWN
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to Q	9	9	9
2901A	AB to GP	40	40	40
2902A	GP to C _N +XYZ	5	5	5
2901A	C _N to F ₃ , OVR	20	-	-
S-EXOR	IN - OUT	7	-	-
S-MUX	D to Y	5	-	-
2901A	SET-UP RAM ₃	15	-	-
2901A	C _N to Y	-	20	-
2901A	C _N to Zero	-	-	35
S-REG	SET-UP D	-	2	2
TOTAL-ns		101	76	91

PATH 1 ———
PATH 2 - - -
PATH 3 - - -

MPR-538

Figure 26. (Cont.)

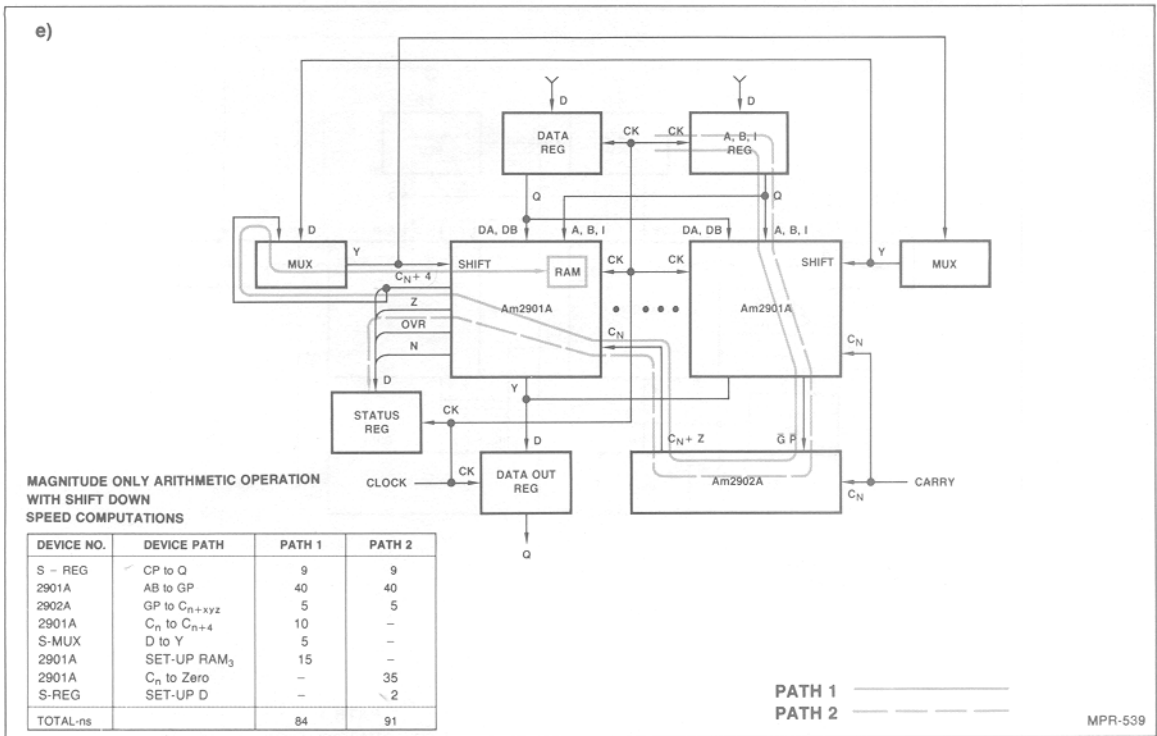


Figure 26. (Cont.)

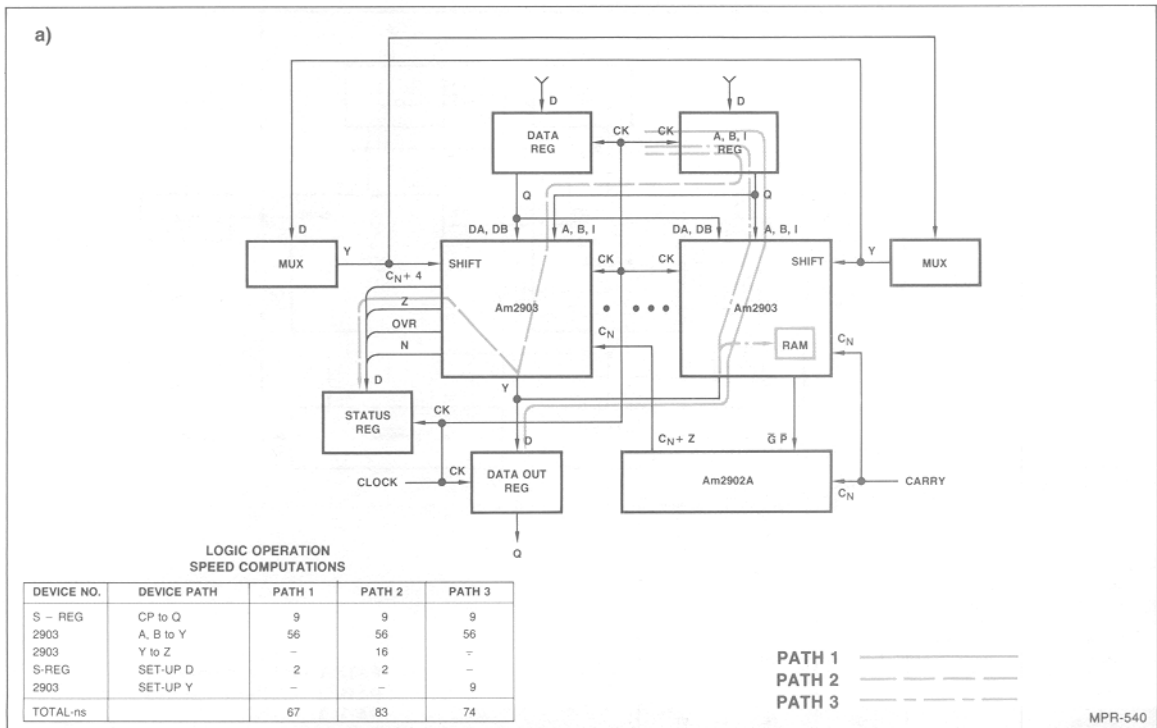
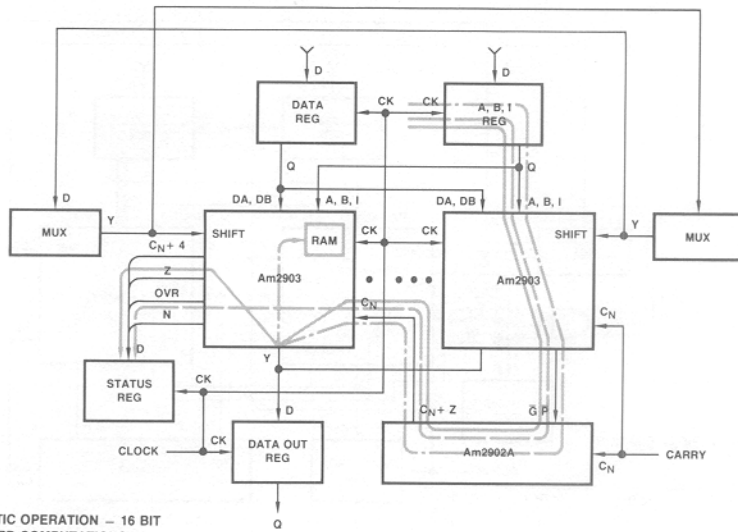


Figure 27. Typical AC Calculations for the Am2903.

b)



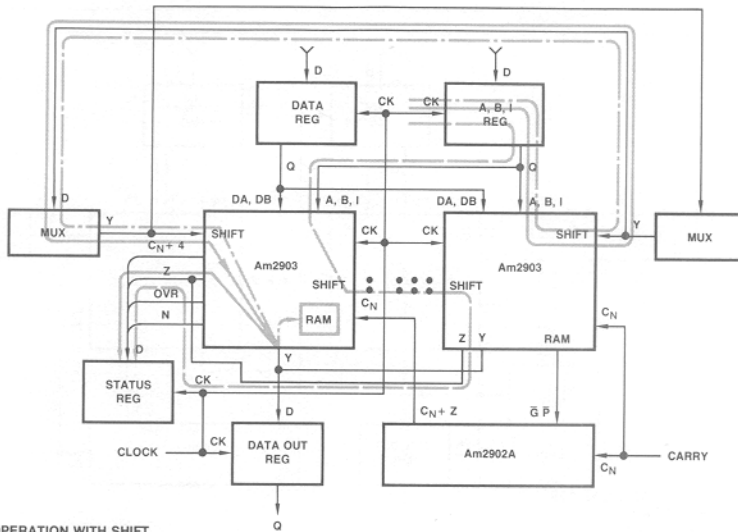
ARITHMETIC OPERATION - 16 BIT
SPEED COMPUTATIONS

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S-REG	CP to Q	9	9	9
2903	A, B to G, P	56	56	56
2902A	G, P to C_{n+xyz}	5	5	5
2903	C_n to Y	25	-	25
2903	C_n to FLAG	-	38	-
2903	Y to Z	16	-	-
S-REG	SET-UP D	2	2	-
2903	SET-UP Y	-	-	9
TOTAL-ns		113	110	104

PATH 1 ———
PATH 2 - - -
PATH 3 - - -

MPR-541

c)



LOGIC OPERATION WITH SHIFT
SPEED COMPUTATIONS

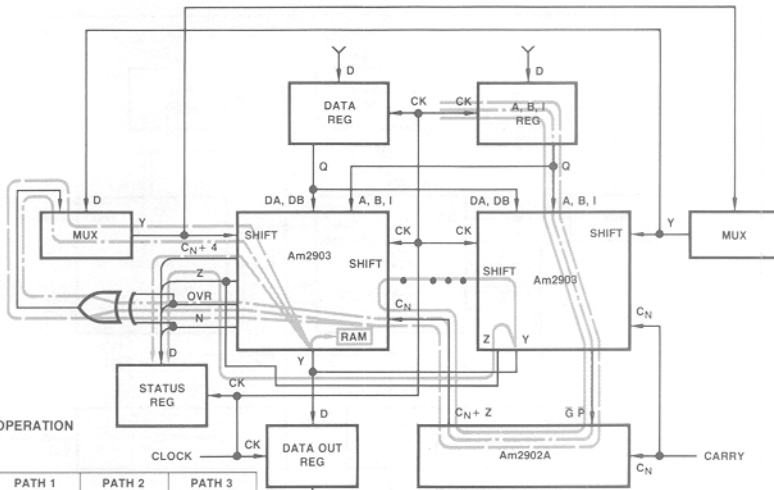
DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to Q	9	9	9
2903	A, B to S_0	64	64	64
MUX	D to Y	5	-	5
2903	S_3 to Y	13	13	13
2903	Y to Z	16	16	-
S-REG	SET-UP D	2	2	-
2903	SET-UP Y	-	-	9
TOTAL-ns		109	104	100

PATH 1 ———
PATH 2 - - -
PATH 3 - - -

MPR-542

Figure 27. (Cont.)

d)



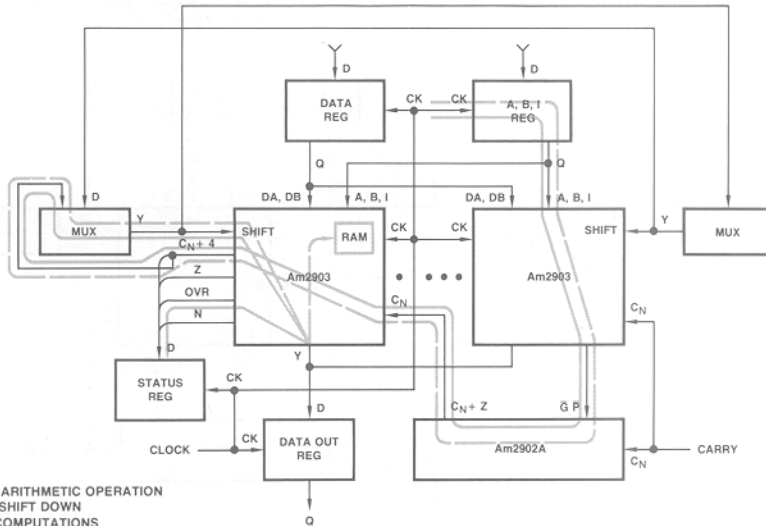
**TWO'S COMPLEMENT ARITHMETIC OPERATION
WITH SHIFT DOWN - 16 BIT
SPEED COMPUTATIONS**

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2	PATH 3
S - REG	CP to Q	9	9	9
2903	A, B to G, P	56	56	56
2902A	GP to C_{n+xyz}	5	5	5
2903	C_n to SiO_0	21	-	-
2903	SiO_3 to Y	13	-	-
2903	C_n to N, OVR	-	38	38
S-EXOR	IN to OUT	-	7	7
S-MUX	D to Y	-	5	5
2903	SiO_3 to Y	-	13	13
2903	Y to Z	16	16	-
2903	SET-UP Y	-	-	9
S-REG	SET-UP D	2	2	-
TOTAL-ns		122	151	142

PATH 1 ———
 PATH 2 - - -
 PATH 3 - - -

MPR-543

e)



**MAGNITUDE ONLY ARITHMETIC OPERATION
WITH SHIFT DOWN
SPEED COMPUTATIONS**

DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
S - REG	CP to Q	9	9
2903	A, B to G, P	56	56
2902A	GP to C_{n+xyz}	5	5
2903	C_n to C_{n+4}	21	21
S-MUX	D to Y	5	5
2903	SiO_3 to Y	13	13
2903	Y to Z	16	-
S-REG	SET-UP D	2	-
2903	SET-UP Y	-	9
TOTAL-ns		127	118

PATH 1 ———
 PATH 2 - - -

MPR-544

Figure 27. (Cont.)

Functional Operation	Am2901A	Am2903
Logic	76	83
Arithmetic	94	113
Logic with Shift	89	109
Two's Complement Arithmetic with Shift Down	101	151
Magnitude Only Arithmetic with Shift Down	91	127

Figure 28. Summary of Am2901A and Am2903 AC Performance in a 16-Bit Configuration.

USING THE Am2903 IN A 16-BIT DESIGN

Perhaps the best technique for understanding the design of the 16-bit ALU is to simply take an example. Figure 29 shows a block diagram overview of four Am2903's with the appropriate shift matrix control, status register, MAR and the usual interface to a CCU and main memory. This block diagram represents the normal data handling path associated with a simple 16-bit minicomputer. If we expand this block diagram to show what would normally be considered to be the complete 16-bit central processing unit, the block diagram of Figure 30 results. Here, we see the Am2903's surrounded by a typical set of MSI support chips. In addition, the block diagram shows a typical computer control unit as described in Chapter 2 of this series. Thus, all of the blocks are

now in place to show a simple 16-bit microcomputer built using the Am2900 family devices. The full design for such a machine is shown in Figure 31.

Figures 31A, Figure 31B and Figure 31C detail the connection of each IC used in this design. Quite simply, the design can be described as follows. Figure 31A represents the microprogram sequencer portion of the design. U1, U2 and U3 are the instruction register that receive a 16-bit instruction from main memory. U4, U5 and U6 are the mapping PROMs used to decode the OP code portion of the instruction to arrive at a starting address for the microprogram sequencer. The microprogram sequencer is the Am2910 and is shown as U7. The branch address pipeline register is U8, U9 and U10 and can be enabled to the D inputs of the Am2910 sequencer to provide the jump address from microcode. The pipeline register for the instruction inputs to the Am2910 is U14. This machine also has the ability to select the A and B addresses for the Am2903 devices from the microprogram as well as the instruction register and U11 and U12 provide this capability as a part of the pipeline register. U13 is a two line to four line decoder used as part of the control for the A and B address select for the Am2903's. U15 is part of the pipeline register and provides both true and complement outputs for bit 11. U16 and U17 represent a one of sixteen decoder whose output can be applied to the DA bus to allow the implementation of all the bit operations. These include bit set, bit clear, bit toggle and bit test. U18 and U19 are PROM's that provide the ability to enter one of thirty-two preprogrammed constants onto the DA bus.

Figure 31B is predominately the data handling portion of the design. Here, U20 and U21 represent a data register that receives data from the data bus. U26, U27, U28 and U29 are the four Am2903's that form a 16-bit register/ALU combination. U30 is the carry look ahead generator for the ALU section. U22, U23

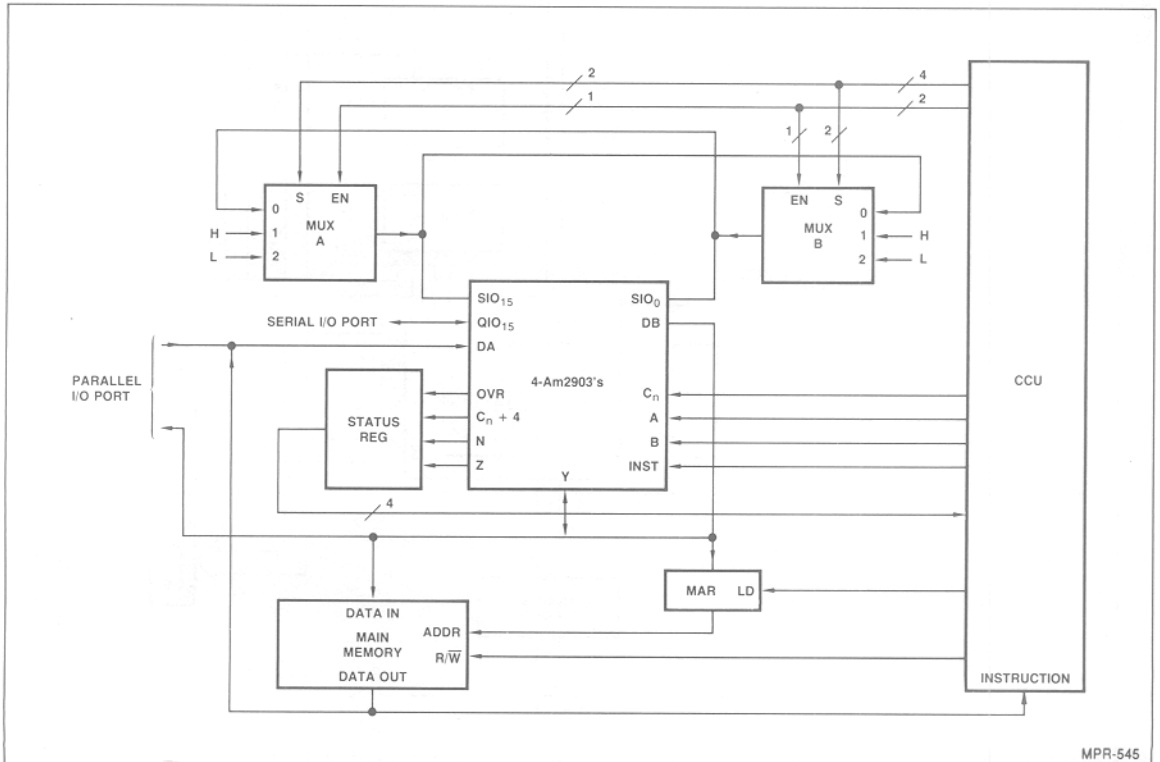
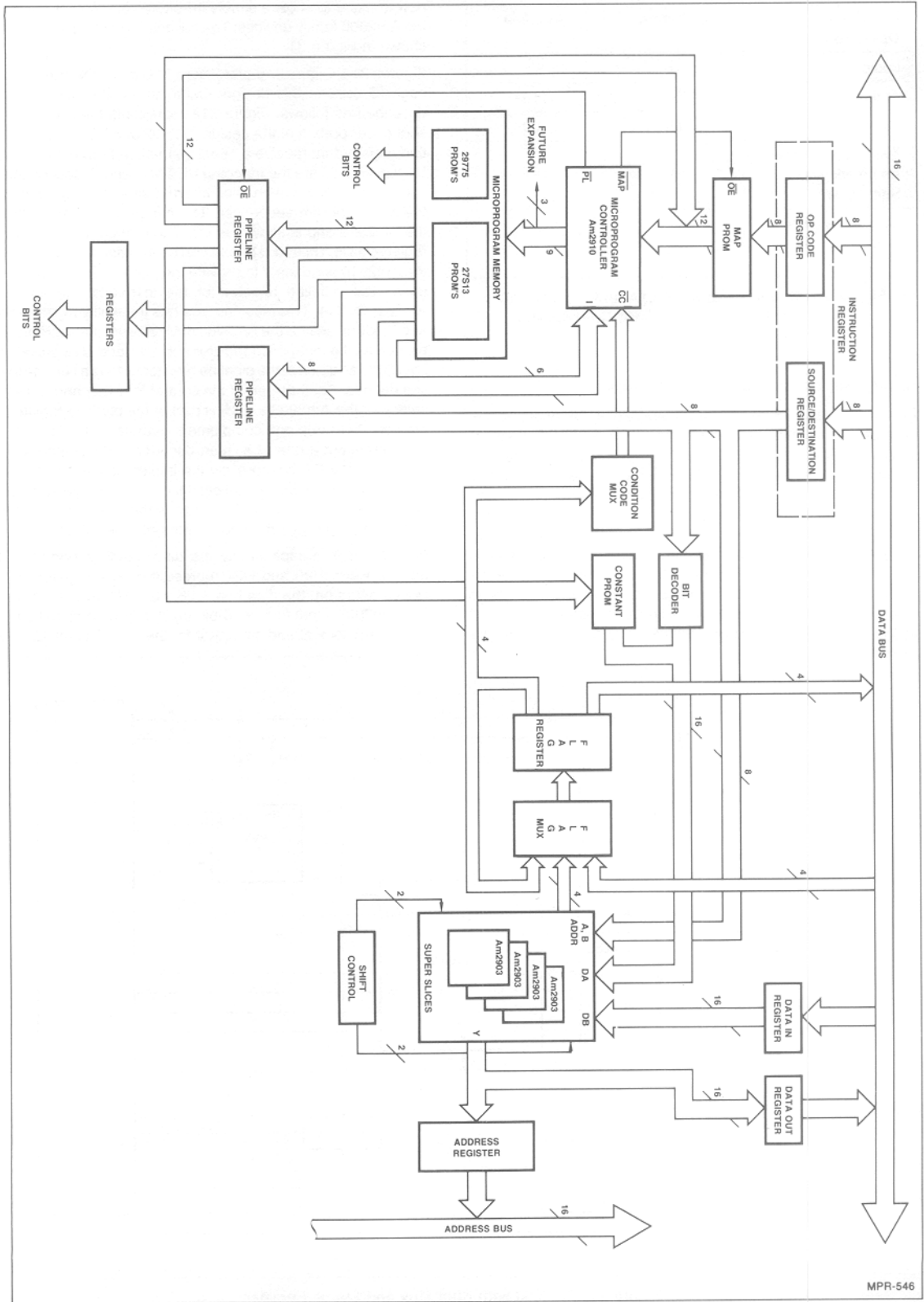


Figure 29. Am2903 with Shift Mux and Status Register.



MPR-546

Figure 30.

and U24 represent the status register with the ability to save and restore the flags in main memory. U25 is the condition code multiplexer for the microprogram sequencer. U33, U34, U35 and U36 represent the shift linkage multiplexers that tie together the internal shifters within the Am2903's. U37 is part of the pipeline register and provides both true and complement outputs of a number of the microprogram bits. U38 is part of the carry in logic control such that double length arithmetic operations can be performed. U31 and U32 are the data out register that can be used to accept data from the Am2903s and enable this data onto the data bus. U39 and U40 represent the memory address register and are used to hold the address provided from the CPU to main memory.

The microprogram store is shown in Figure 31C. Here, we have used both the 512 x 8 registered PROM's and 512 x 4 non-registered PROM's in this design. A total of 68 microprogram bits have been depicted in this design. These are shown so that maximum flexibility is achieved. In most typical designs some 10 to 20 of these bits would not be used. Figure 31C shows four 512-word by 8-bit registered PROM's (U41, U42, U43 and U44). It also shows nine 512-word by 4-bit PROM's represented as U45 through U53.

Perhaps the best way to review the design is to simply understand the function of each of the microprogram control bits. If the purpose of each of these bits is well understood, the design engineer will be well along in understanding the design of the simple minicomputer CPU presented here.

The Microprogram Structure

The microprogram for the design shown in Figure 31 is 68 bits wide. The functions of the microprogram control bits are as follows:

- Bits PL0 through PL8 The 9 instruction bits of the Am2903 superslices.
- Bits PL9, PL10, PL11 The \overline{IEN} , \overline{EA} , \overline{OEB} control inputs of the Am2903 superslices, respectively. PL11 is also connected to the data-in registers (U20 and U21) output-enable. This connection assures that there will be no conflict on the DB pins.
- Bits PL12 through PL14 ($\mu 12$ through $\mu 14$) Select the source for SIO of the Am2903, both for shift-up and for shift-down operations. The following table summarizes the functions of these bits.

Microprogram Bits			SIO _n	SIO _o
14	13	12	(Shift-down)	(Shift-up)
L	L	L	0	0
L	L	H	SIO _o	SIO _n
L	H	L	QIO _o	QIO _n
L	H	H	Carry	Carry
H	L	L	Zero	Zero
H	L	H	Sign	Sign
H	H	L	Not allocated	Not allocated
H	H	H	1	1

- Bits PL15 through PL17 ($\mu 15$ through $\mu 17$) Select the source for QIO of the Am2903, both for shift-up and shift-down operations. The following table summarizes the functions of these bits.

Microprogram Bits			QIO _n	QIO _o
17	16	15	(Shift-down)	(Shift-up)
L	L	L	0	0
L	L	H	SIO _o	SIO _n
L	H	L	QIO _o	QIO _n
L	H	H	Carry	Carry
H	L	L	Zero	Zero
H	L	H	Sign	Sign
H	H	L	Not allocated	Not allocated
H	H	H	1	1

- Bit PL18 When LOW, enables the MAR clock input, i.e. the data appearing on the Y output pins of the Am2903 Superslices™ will be clocked into the MAR at the LOW-to-HIGH transition of the clock pulse.
- Bit PL19 When LOW, enables the MAR output onto the Memory Address Bus.
- Bit PL20 When LOW, enables the data output register clock, i.e. the data appearing in the Y output pins of the Am2903 Superslices™ will be clocked into the data output registers (U31 and U32) at the LOW-to-HIGH transition of the clock pulse.
- Bit PL21 When LOW, enables the data output registers onto the Data Bus.
- Bit PL22 When LOW, enables the data-in register clock, i.e. the data appearing in the Data-Bus will be clocked into the data-in registers at the LOW-to-HIGH transition of the clock pulse.
- Bit PL23 This is the CI input of the Am2910 microprogram sequencer.
- Bits PL24 through PL27 This is a 4-bit wide field which can be used either for the A-address, for the B-address or for both A and B addresses of the Am2903 superslices.
- Bits PL28 through PL31 This is a 4-bit wide field, which can be used for either the A-address of the Am2903 superslice or to designate one of sixteen bits to the DA inputs of the Am2903 superslice via the Am2921's ($\mu 16$ and $\mu 17$).
- Bits PL32 and PL33 Select the source for the Am2903 A-address, according to the table below:

Bits		A-Address Source
33	32	
L	L	Data Bus bits 0 through 3
L	H	Microprogram bits 28 through 31
H	L	Data Bus bits 4 through 7
H	H	Microprogram bits 24 through 27

- Bit PL34 Selects the source of the Am2903 B-address, according to the table below:

Bit	B-Address Source
34	
L	Data Bus bits 4 through 7
H	Microprogram bits 24 through 27

Bit PL35 Is the C_n input of the least significant Am2903 via an Am74S157 mux (μ 38).

Bits PL36 and PL37 Affect the status register input signals, according to the table below:

Bits		Next Carry	Next Zero, Sign, Overflow
37	36		
L	L	Previous Carry	Previous Zero, Sign, Overflow
L	H	Previous SIO ₁₅	Previous Zero, Sign, Overflow
H	L	Am2903 superslices' Output	
H	H	Data Bus bits 0 through 3	

Bit PL38 Selects either the carry flip-flop or the PL35 bit for carry in.

Bit PL39 When LOW, enables the status register output to the data bus bits 0 through 3.

Bit PL40 Controls the output polarity of the one-of-sixteen bit select logic.

Bit PL41 When LOW, enables the Instruction register (U1, U2, U3) clock. The data present at bits 0 through 15 of the Data-Bus will be latched into the Instruction register at the next LOW-to-HIGH transition of the clock pulse.

Bit PL42 This is an output signal. When HIGH, it signals the main memory that a memory read is requested.

Bit PL43 This is an output signal. When HIGH, it signals to the main memory that a memory write is requested.

Bit PL44 Selects the source of the one of sixteen bit decoders (U16 and U17). When LOW, the output of the Am2919 register (U12) containing the previously latched microprogram bits 28 through 31 will be applied to the decoders. When HIGH, the output of the Am2919 register (U3) containing the previously latched Data-Bus bits 0 through 3 will be applied to the decoders.

Bit PL45 Selects the Am2903 Superslices™ DA port source. When LOW, the output of the one of sixteen bit decoder (U16 and U17) will be applied to that port. When HIGH, the output of the Am29771 PROM's (U18 and U19) will be applied to the Am2903 DA ports.

Bit PL46 and PL47 These are the \overline{RLD} and \overline{CCEN} control inputs of the Am2910 sequencer, respectively.

Bits PL48 through PL50 These select the condition code according to the following table:

Bits			Condition Code Selected
50	49	48	
L	L	L	Carry Sign Zero Overflow
L	L	H	
L	H	L	
L	H	H	
H	L	L	Not Allocated
H	L	H	
H	H	L	
H	H	L	
H	H	H	

Bit PL51 Is the condition code polarity control. When HIGH, the condition code selected will pass non-inverted. When LOW, the selected condition code will be complemented.

Bits PL52 through PL55 Are the I inputs of the Am2910 sequencer.

Bits PL56 through PL67 This is a 12-bit wide field and it serves, usually as the next microprogram address. However, the 5 least significant bits of this field (bits 56-60) serve also as an address field of the Am29771 "constant" PROM's (U18 and U19).

Some Sample Microroutines

Figure 32 shows the microprogram code for a few sample microroutines. Different addressing schemes are demonstrated with the "ADD" operation. All the other arithmetic or logic operations can be easily programmed by substituting the I₁-I₄ field of the Am2903 with the appropriate function. Since the main memory address is generated by the Am2903 superslices, the internal register No. 15 serves as the program counter.

The following is a description of some sample microroutines. The reader should refer to the description of the microprogram bits given earlier in this chapter and to the data sheets of the Am2910 sequencer and of the Am2903 superslice.

Microword INIT.

This microword should be at address 0 and when the machine is reset, the Am2910 will start executing from here. The purpose of this location is to reset the machine program counter (Register 15) to zero. Ultimately more microinstructions can be added, should the necessity of other reset functions arise.

Bits 1-4 (Am2903 I₁-I₄) being 8_H will cause the superslices to generate all zeroes at the F-points (internal). Bits 5-8 (Am2903 I₅-I₈) being F_H will cause this data (all zeroes) to appear on the Y outputs. Bit 9 (\overline{IEN}) is LOW and therefore, WRITE will be LOW and this data will be written into the internal register selected by the B-address inputs. Bit 34 is HIGH; therefore, microprogram bits 24-27 will be selected as B address source. Since F_H is in these bits, all zeroes will be written into the program counter (Register 15). Bit 18 is LOW; therefore, the data at the Y outputs (all zeroes) will be latched into the MAR at the next clock pulse. Bits 36 and 37 are set such that the flags will be updated, namely CY=N=OVF=0, Z=1.

Bits 42, 43 are both LOW so no memory reference signal is sent to the main memory (the MAR is still in an undetermined state). Bits 52-55 (Am2910 I) are set to E_H which will force the sequencer to continue to the next sequential address (1) as the CI (bit 23) is HIGH.

Bits 21 and 39 are both HIGH to ensure that there is no conflict on the data bus though in this case one of them could be a DON'T-CARE. Bit 38 could also be a DON'T-CARE as the carry is zeroed by the ALU. Making a HIGH in bit 46 enables executing this microstep without disturbing the Am2910 sequencer's internal register which at power-up has no significance but may be important, should a software restart be issued.

All the other bits are DON'T-CARES.

Microword FETCH

This is the first step in the machine instruction fetch routine. In this step, the main memory is addressed by the MAR, a read signal is issued (bit 42 = HIGH), and the machine instruction (macroinstruction) is placed on the data bus by the memory. It is

	PL	2910					DA		MMW		POL		FDOE CY=0		Flags
		I	CCP	CC	CLEN	RLD	CONS	BIT	MMW	MMR	IRE	POL	FDOE	CY=0	
Number of Bits	12	4	1	3	1	1	1	1	1	1	1	1	1	2	
Bit No.	56-67	52-55	51	48-50	47	46	45	44	43	42	41	40	39	38	36-37
INIT	X	E	X	X	X	1	X	X	0	0	X	X	1	0	2
FETCH	X	E	X	X	X	1	X	X	0	1	0	X	1	0	0
FETCH + 1	X	2	X	X	X	1	X	X	0	0	1	X	1	0	0
ADD	FETCH + 1	7	X	X	1	1	X	X	0	1	0	X	1	0	2
ADDIMM	X	E	X	X	X	1	X	X	0	1	1	X	1	0	0
ADDIMM + 1	FETCH + 1	7	X	X	1	1	X	X	0	1	0	X	1	0	2
ADD DIR	X	E	X	X	X	1	X	X	0	1	1	X	1	0	0
ADD DIR + 1	X	E	X	X	X	1	X	X	0	0	1	X	1	0	0
ADD DIR + 2	ADDIMM + 1	7	X	X	1	1	X	X	0	1	1	X	1	0	0
ADD RR1	X	E	X	X	X	1	X	X	0	0	1	X	1	0	0
ADD RR1 + 1	X	E	X	X	X	1	X	X	0	1	1	X	1	0	0
ADD RR1 + 2	FETCH + 1	7	X	X	1	1	X	X	0	1	0	X	1	0	2

	2903					2910	Y-D			MAR		2903								
	C _n	B	A	R ₂	R ₁		CI	DDBE	OE	E	OE	E	Q	S	OEB	EA	IEN	I ₅₋₈	I ₁₋₄	I ₀
Number of Bits	1	1	2	4	4	1	1	1	1	1	1	3	3	1	1	1	4	4	1	
Bit No.	35	34	32-33	28-31	24-27	23	22	21	20	19	18	15-17	12-14	11	10	9	5-8	1-4	0	
INIT	X	1	X	X	F	1	X	1	X	X	0	X	X	X	X	0	F	8	X	
FETCH	X	X	X	X	X	1	1	1	1	0	1	X	X	0	X	1	X	X	X	
FETCH + 1	1	1	X	X	F	1	1	1	1	0	0	X	X	0	X	0	F	4	0	
ADD	0	0	0	X	X	1	1	1	1	0	1	X	X	0	0	0	F	3	0	
ADDIMM	1	1	X	X	F	1	0	1	1	0	0	X	X	0	X	0	F	4	0	
ADDIMM + 1	0	0	0	X	X	1	1	1	1	0	1	X	X	1	0	0	F	3	0	
ADD DIR	1	1	X	X	F	1	0	1	1	0	X	X	X	0	X	0	F	4	0	
ADD DIR + 1	0	X	X	X	X	1	1	1	1	X	0	X	X	1	X	1	X	4	0	
ADD DIR + 2	0	X	3	X	F	1	0	1	1	0	0	X	X	X	0	1	F	6	X	
ADD RR1	0	X	0	X	X	1	X	1	1	X	0	X	X	X	0	1	F	6	X	
ADD RR1 + 1	0	X	3	X	F	1	0	1	1	0	0	X	X	X	0	1	F	6	X	
ADD RR1 + 2	0	0	2	X	X	1	1	1	1	0	1	X	X	1	0	0	F	3	0	

1. 4-bit fields in hex, others in octal.
2. X = Don't Care.

Figure 32. Example Microcode for Figure 31 Design.

latched into the instruction register (U1, U2, and U3) at the next clock LOW-to-HIGH transition (bit 41 = LOW). It is assumed that if a relatively slow main memory is used, the clock is halted until the data is stable on the data bus and the register set up times are met. We will see in a later chapter how easy it is to implement this requirement using the Am2925 clock generator. The same assumption will also be made in a memory write cycle.

Bit 9 (Am2903 \overline{IEN}) is HIGH; thus, we don't care what the ALU does during this microstep. We prevent the flags from changing by setting bits 36-38 LOW. Also, the registers at the Y output have the \overline{E} input HIGH (bits 18, 20). Bits 21 and 39 are both HIGH; thus, the data bus is free to accept data from the main memory (bit 42 is HIGH, signaling memory read request). The MAR is enabled to the address bus (bit 19 = LOW) and at the next clock, the macroinstruction will be latched into the instruction registers (bit 41 = LOW). The Am2910 sequencer will continue to the next instruction (bits 52-55 = E_H).

Microword FETCH + 1

This is the second step in the macroinstruction fetch routine. The instruction already resides in the instruction registers U1, U2 and U3).

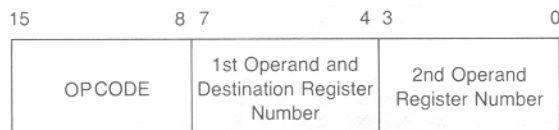
The Am2910 sequencer receives a JUMP MAP instruction (bits 52 though 55 = 2). The next microinstruction will begin to execute the present macroinstruction – according to the mapping PROM.

We use this microstep to update (increment) the program counter (Register 15). Bit 34 being HIGH, microprogram bits 24-27 (= F_H) will be the B address. The Am2903 OEB and I_0 are LOW, therefore, the contents of Register 15 will serve as the S operand for the ALU. C_n being HIGH, a 4 in the I_1 - I_4 field will increment this value. \overline{IEN} = LOW with I_5 - I_8 = F will write this (incremented) value into the same register (R15). At the same time, the MAR is also updated (bit 18 = LOW).

We could update the program counter and the MAR in the previous microstep (location FETCH), but then we had to leave the ALU idle during this microcycle. By adopting the present scheme, we can overlap the first step of the macroinstruction fetch routine (the memory-read cycle) with the execution of the last step of the previous macroinstruction – provided the memory and the data bus are free to perform it. The JUMP MAP cycle is always necessary – and that is why we prefer to update the PC at this step.

Microword ADD

This is a sample register-to-register operation. The two operands reside in the internal registers pointed to by the two 4-bit fields of the macroinstruction:



Bits 32-33 are set LOW, instruction register bits 0-3 are selected as A address. Bit 34 = LOW selects instruction register bits 4-7 as B address (see Fig. above). Bit 1 (I_0), bit 10 (\overline{EA}) and bit 11 (\overline{OEB}) are also LOW; therefore, the contents of the selected registers will be presented to the ALU's R and S inputs. Bits 1-4 (I_1 - I_4) = 3, the ALU will perform:

$$F = R \text{ plus } S \text{ plus } C_n.$$

Note that bit 35 and 38 are LOW. With I_5 - I_8 (bits 5-8) = F_H and \overline{IEN} (bit 0) = LOW, the result will be written into the internal register pointed at by the B address lines.

Bits 18 and 20 are HIGH and inhibit the MAR and the data out registers from being affected, while bits 36, 37 (=2) allow the flags to assume values according to the result of the operation.

During the execution of the function required (ADD in this example) we fetch the next OP CODE from the main memory. The MAR is enabled to the address bus (bit 19 = LOW) and a memory read is requested (bit 42 = HIGH). At the end of this microstep the next macroinstruction will be latched into the instruction registers (bit 41 = LOW).

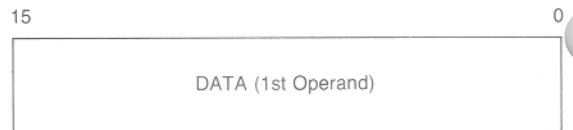
The Am2910 sequencer is instructed to select the pipeline register bits 56-67 as the next microprogram address (bits 52-57 = 7, bit 47 = HIGH) where the location of FETCH + 1 (2 in this example) is written. The next step will be JUMP MAP and update PC.

Microword ADD IMMEDIATE

This 2 step microroutine adds the contents of an internal register, pointed at by bits 0-3 of the macroinstruction with its second word, placing the result into the internal register pointed at by bits 4-7 of the OPCODE.



First word of the macroinstruction



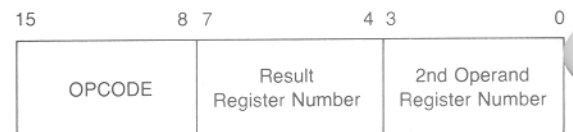
Second (next consecutive) word of the macroinstruction

The first step is to read the first operand from the memory (bit 19 = LOW, bit 42 = HIGH) and to latch it into the data-in register (U20 and U21) (bit 22 = LOW). At the same time the ALU updates (increments) the program counter (register 15) and the MAR (bit 18 = LOW). (Compare the location FETCH + 1). The Am2910 sequencer will continue to the next microprogram address (compare to location FETCH).

Location ADDIMM + 1 is the second step of this macroinstruction. It is very similar to location ADD, the only difference is that bit 11 (\overline{OEB}) is HIGH, selecting the Data-in register as source for the ALU's S operand. The same macroinstruction fetch overlap technique is used again.

Microword ADD DIrect

This is the starting location to execute a macroinstruction where the second word is the address of the operand:



First word of the macroinstruction



Second (next consecutive) word of the macroinstruction

The first step is to read the second word of the macroinstruction into the Data-in register. This microword is identical to the one written at location ADDIMM.

Microword ADD DIR + 1

The Data-in register now contains the address of the operand. We have to transfer it into the MAR.

With I_0 (bit 0) LOW and \overline{OEB} (bit 11) HIGH, the ALU's operand will be the DB bus, i.e., the Data-in register. I_1 - I_4 (bits 1-4) = 4 will pass this input to its output, as C_n (bit 3) is LOW. With \overline{IEN} (bit 9) = HIGH, the WRITE line will be HIGH too, assuring that the internal registers maintain their contents. Since I_5 - I_8 (bits 5-8) = F_H , the ALU output will appear on the Am2903 Y pins. This data which is actually the operand address and will be transferred into the MAR at the next clock cycle. The Am2910 sequencer continues to the next consecutive microstep.

Microword ADD DIR + 2

Now we read in the operand from the main memory. The MAR is enabled to address bus (bit 19 = LOW), a memory read signal is issued (bit 42 = HIGH) and the data-in register's clock is enabled (bit 22 = LOW). At the next LOW-to-HIGH transition of the clock, the operand will be placed in the data-in register.

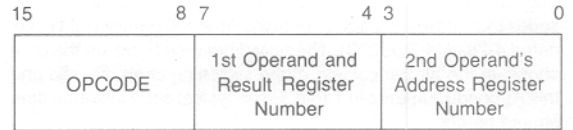
Meanwhile, we need to restore the address of the next macroinstruction in the MAR. Bits 32-33 = 3 select microprogram bits 24-27 as the A address (an F_H is written there); therefore, the internal program counter will be addressed, as \overline{EA} (bit 10) = LOW. The ALU performs an $F = R + C_n$ with C_n (bit 35) LOW, thus passing the program counter contents to the output. \overline{IEN} (bit 9) = HIGH prevents disturbance of internal Am2903 registers and bit 18 will enable the MAR to receive the next macroinstruction address.

Note that the situation now is exactly the same as after the first step of ADD IMMEDIATE. The operand is in the data register and the MAR points to the next macroinstruction. Therefore, the Am2910 sequencer will address, as the next microstep, location ADDIMM + 1. The step after this will, of course, be FETCH + 1. A total of 5 microsteps were needed to execute this macroinstruction but it occupies only 3 microprogram locations.

It is worthwhile to note here that by adding two more Am2920 registers between the Data-bus and the Address-bus and a couple of control-bits in the microprogram, we could shorten the microprogram by one step. In this design we chose not to do so in order to demonstrate the Data-bus to Address-bus path through the ALU.

Microword ADD RR1

The macroinstruction to be executed here points to the register in which the first operand is written, and also into which the result should be written. The second 4-bit field of the OP-CODE (bits 0-3) points to the register in which the address of the second operand is stored.



Bits 32 and 33 are LOW. Therefore, instruction register bits 0-3 will form the A-address. Now we take the contents of this register and place it in the MAR exactly the same way as we did in location ADD DIR + 2 with the program counter. The Am2910 continues.

Microword ADD RR1 + 1

Here we fetch the operand and place it in the Data-in register. At the same time, we restore the program counter into the MAR.

Microword ADD RR1 + 2

Bits 32, 33 = 2 and instruction register bits 4-7 serve as the A-address. Bit 34 = LOW; the same instruction register bits serve as B-address, too. Note, that \overline{OEB} (bit 11) is HIGH; therefore, the ALU R-source will be the Data-in register and the S-source will be the register addressed by A-address. The result (sum), however, will be written to the correct register, as \overline{IEN} (bit 9) is LOW.

At the same time, the next macroinstruction is fetched in the usual overlapping way and the next microinstruction to be executed will be at location FETCH + 1.

Summary

In this design shown in Figure 31, we have demonstrated some of the addressing schemes mentioned in Chapter 1. We used the ADD instruction throughout these examples, but any other arithmetic or logic instruction can be executed, in *exactly* the same manner by changing the microcode bits 1-4 to the appropriate ALU code.

The reader is encouraged to write several microcode-lines to execute the other addressing modes mentioned in Chapter 1. He will discover that when the result of the macroinstruction is to be written into main memory, the overlapping instruction-fetch is not feasible. In some cases, when the MAR no longer contains the Program Counter value, an additional microstep is needed in order to restore the Program Counter into the MAR. The reader is again encouraged to modify location FETCH in order to save this additional microstep.

Appendix

Throughout Chapter 3, a number of AC calculations have been made to show typical speeds for an Am2901A and Am2903 16-bit ALU configuration. This Appendix shows the latest SWITCHING CHARACTERISTICS for the Am2901A and Am2903.

The typical data on the Am2901A shown in this Appendix supersedes that shown on page 2-12 of the Am2900 Family Data Book dated 4-78 (AM-PUB003). The only difference between the data shown in the typical column of the switching characteristic and this Appendix appears in Table 3. The typical carry in set-up time should be 40ns.

The typical switching characteristic data for the Am2903 as shown in this Appendix supersedes the data presented in the Am2903 Bipolar Microprocessor Slice/Am2910 Microprogram Controller Data Booklet dated 3-78. Here, a number changes have been made to the table for both the combinatorial propagation delays and the set-up and hold times.

Should any questions arise concerning the switching characteristics for either the Am2901A or Am2903, please do not hesitate to contact the AMD factory and ask for Bipolar Microprocessor Marketing or Bipolar Microprocessor Applications.

ROOM TEMPERATURE SWITCHING CHARACTERISTICS

(See next page for AC Characteristics over operating range.)

Tables I, II, and III below define the timing characteristics of the Am2901A at 25°C. The tables are divided into three types of parameters; clock characteristics, combinational delays from inputs to outputs, and set-up and hold time requirements. The latter table defines the time prior to the end of the cycle (i.e., clock LOW-to-HIGH transition) that each input must be stable to guarantee that the correct data is written into one of the internal registers.

All values are at 25°C and 5.0V. Measurements are made at 1.5V with $V_{IL} = 0V$ and $V_{IH} = 3.0V$. For three-state disable tests, $C_L = 5.0pF$ and measurement is to 0.5V change on output voltage level. All outputs fully loaded.


TABLE I

CYCLE TIME AND CLOCK CHARACTERISTICS

TIME	TYPICAL	GUARANTEED
Read-Modify-Write Cycle (time from selection of A, B registers to end of cycle)	55ns	93ns
Maximum Clock Frequency to Shift Q Register (50% duty cycle)	40MHz	20MHz
Minimum Clock LOW Time	30ns	30ns
Minimum Clock HIGH Time	30ns	30ns
Minimum Clock Period	75ns	93ns

TABLE II

COMBINATIONAL PROPAGATION DELAYS (all in ns, $C_L = 50pF$ (except output disable tests))

From Input \ To Output	TYPICAL 25°C, 5.0V								GUARANTEED 25°C, 5.0V							
	Y	F ₃	C _{n+4}	$\overline{G}, \overline{P}$	F=0 R _L = 270	OVR	Shift Outputs		Y	F ₃	C _{n+4}	$\overline{G}, \overline{P}$	F=0 R _L = 270	OVR	Shift Outputs	
							RAM ₀	Q ₀							RAM ₀	Q ₀
A, B	45	45	45	40	65	50	60	—	75	75	70	59	85	76	90	—
D (arithmetic mode)	30	30	30	25	45	30	40	—	39	37	41	31	55	45	59	—
D (I = X37) (Note 5)	30	30	—	—	45	—	40	—	36	34	—	—	51	—	53	—
C _n	20	20	10	—	35	20	30	—	27	24	20	—	46	26	45	—
I ₀₁₂	35	35	35	25	50	40	45	—	50	50	46	41	65	57	70	—
I ₃₄₅	35	35	35	25	45	35	45	—	50	50	50	42	65	59	70	—
I ₆₇₈	15	—	—	—	—	—	20	20	26	—	—	—	—	—	26	26
OE Enable/Disable	20/20	—	—	—	—	—	—	—	30/33	—	—	—	—	—	—	—
A bypassing ALU (I = 2xx)	30	—	—	—	—	—	—	—	35	—	—	—	—	—	—	—
Clock  (Note 6)	40	40	40	30	55	40	55	20	52	52	52	41	70	57	71	30

SET-UP AND HOLD TIMES (all in ns) (Note 1)

TABLE III

From Input	Notes	TYPICAL 25°C, 5.0V		GUARANTEED 25°C, 5.0V	
		Set-Up Time	Hold Time	Set-Up Time	Hold Time
A, B Source	2, 4 3, 5	40 $t_{pwL} + 15$	0	93 $t_{pwL} + 25$	0
B Dest.	2, 4	$t_{pwL} + 15$	0	$t_{pwL} + 15$	0
D (arithmetic mode)		25	0	70	0
D (I = X37) (Note 5)		25	0	60	0
C _n		40	0	55	0
I ₀₁₂		30	0	64	0
I ₃₄₅		30	0	70	0
I ₆₇₈	4	$t_{pwL} + 15$	0	$t_{pwL} + 25$	0
RAM _{0, 3} , Q _{0, 3}		15	0	20	0

Notes: 1. See next page.

2. If the B address is used as a source operand, allow for the "A, B source" set-up time; if it is used only for the destination address, use the "B dest." set-up time.

3. Where two numbers are shown, both must be met.

4. " t_{pwL} " is the clock LOW time.

5. DV0 is the fastest way to load the RAM from the D inputs. This function is obtained with I = 337.

6. Using Q register as source operand in arithmetic mode. Clock is not normally in critical speed path when Q is not a source.

A. Am2903 SWITCHING CHARACTERISTICS (TYPICAL ROOM TEMPERATURE PERFORMANCE) – (MAY 18, 1978)

Tables IA, IIA, and IIIA define the nominal timing characteristics of the Am2903 at 25°C and 5.0V. The Tables divide the parameters into three types: pulse characteristics for the clock and write enable, combinational delays from input to output, and set-up and hold times relative to the clock and write pulse.

Measurements are made at 1.5V with $V_{IL} = 0V$ and $V_{IH} = 3.0V$. For three-state disable tests, $C_L = 5.0pF$ and measurement is to 0.5V change on output voltage level.

TABLE IA – Write Pulse and Clock Characteristics

Time	
Minimum Time CP and \overline{WE} both LOW to write	15ns
Minimum Clock LOW Time	15ns
Minimum Clock HIGH Time	35ns

**TABLE IIA – Combinational Propagation Delays (All in ns)
Outputs Fully Loaded. $C_L = 50pF$ (except output disable tests)**

To Output From Input	Y	C_{n+4}	$\overline{G}, \overline{P}$	(S) Z	N	OVR	DB	\overline{WRITE}	QIO_0, QIO_3	SIO_0	SIO_3	SIO_0 (Parity)
A, B Addresses (Arith. Mode)	65	60	56	–	64	70	33	–	–	65	69	87
A, B Addresses (Logic Mode)	56	–	46	–	56	–	33	–	–	55	64	81
DA, DB Inputs	39	38	30	–	40	56	–	–	–	39	47	60
\overline{EA}	38	33	26	–	36	41	–	–	–	36	41	58
C_n	25	21	–	–	20	38	–	–	–	21	25	48
I_0	40	31	24	–	37	42	–	15(1)	–	41	39	63
I_{4321}	45	45	32	–	44	52	–	17(1)	–	45	51	68
I_{8765}	25	–	–	–	–	–	–	21	22/29(2)	24/17(2)	27/17(2)	24/17(2)
\overline{IEN}	–	–	–	–	–	–	–	10	–	–	–	–
\overline{OEB} Enable/Disable	–	–	–	–	–	–	12/15(2)	–	–	–	–	–
\overline{OEY} Enable/Disable	14/14(2)	–	–	–	–	–	–	–	–	–	–	–
SIO_0, SIO_3	13	–	–	–	–	–	–	–	–	–	19	20
Clock	58	57	40	–	56	72	24	–	28	56	63	76
Y	–	–	–	16	–	–	–	–	–	–	–	–
\overline{MSS}	25	–	25	–	25	25	–	–	–	24	27	24

- Notes: 1. Applies only when leaving special functions.
 2. Enable/Disable. Enable is defined as output active and correct. Disable is a three-state output turning off.
 3. For delay from any input to Z, use input to Y plus Y to Z.

**TABLE IIIA – Set-Up and Hold Times (All in ns)
CAUTION: READ NOTES TO TABLE III. NA = Note Applicable; no timing constraint.**

Input	With Respect to this Signal	HIGH-to-LOW		LOW-to-HIGH		Comment
		Set-up	Hold	Set-up	Hold	
Y	Clock	NA	NA	9	–3	To store Y in RAM or Q
\overline{WE} HIGH	Clock	5	Note 2	Note 2	0	To Prevent Writing
\overline{WE} LOW	Clock	NA	NA	15	0	To Write into RAM
A,B as Sources	Clock	19	–3	NA	NA	See Note 3
B as a Destination	Clock and \overline{WE} both LOW	–4	Note 4	Note 4	–3	To Write Data only into the Correct B Address
QIO_0, QIO_3	Clock	NA	NA	10	–4	To Shift Q
I_{8765}	Clock	2	Note 5	Note 5	–18	
\overline{IEN} HIGH	Clock	10	Note 2	Note 2	0	To Prevent Writing into Q
\overline{IEN} LOW	Clock	NA	NA	10	–5	To Write into Q

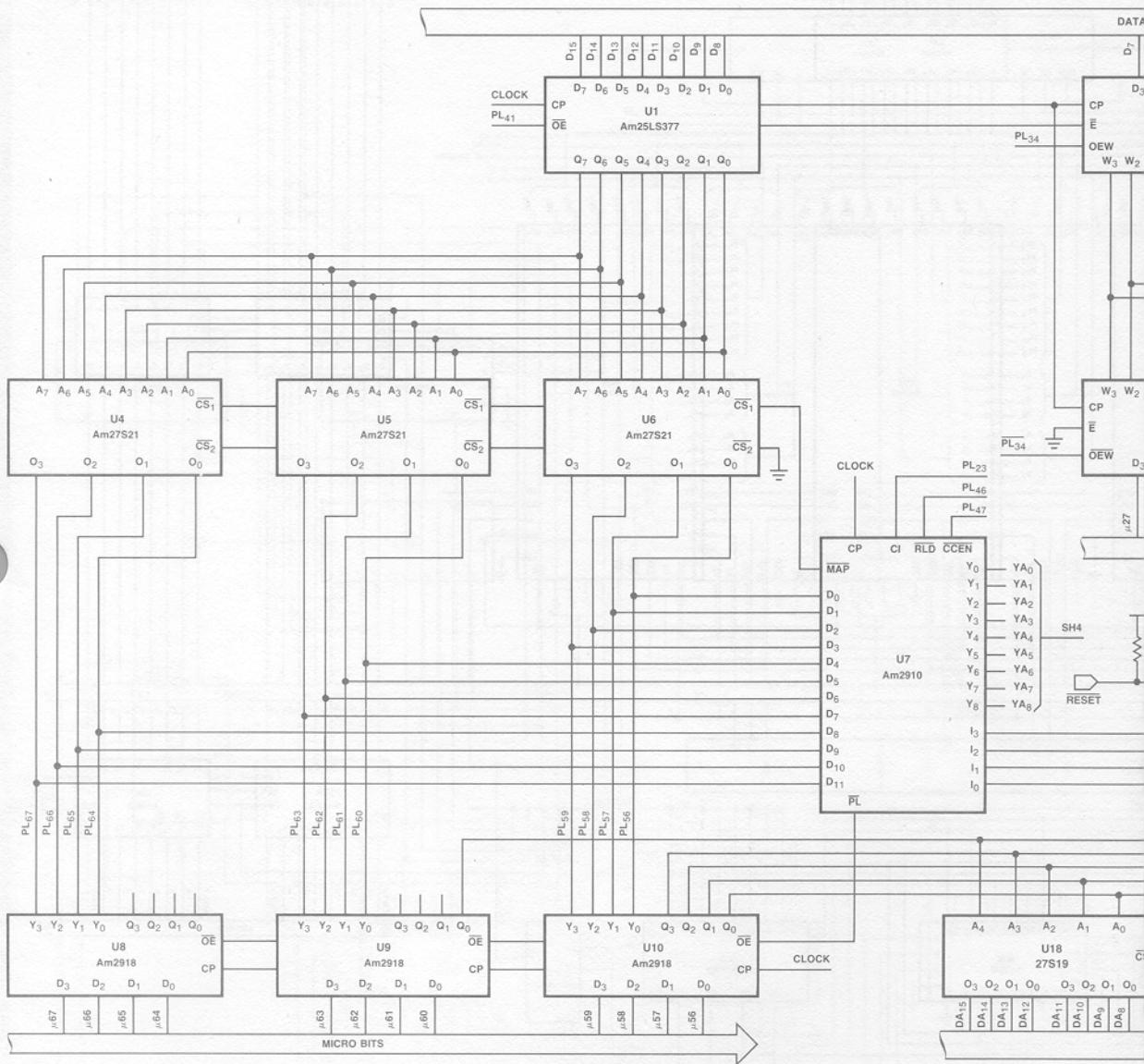
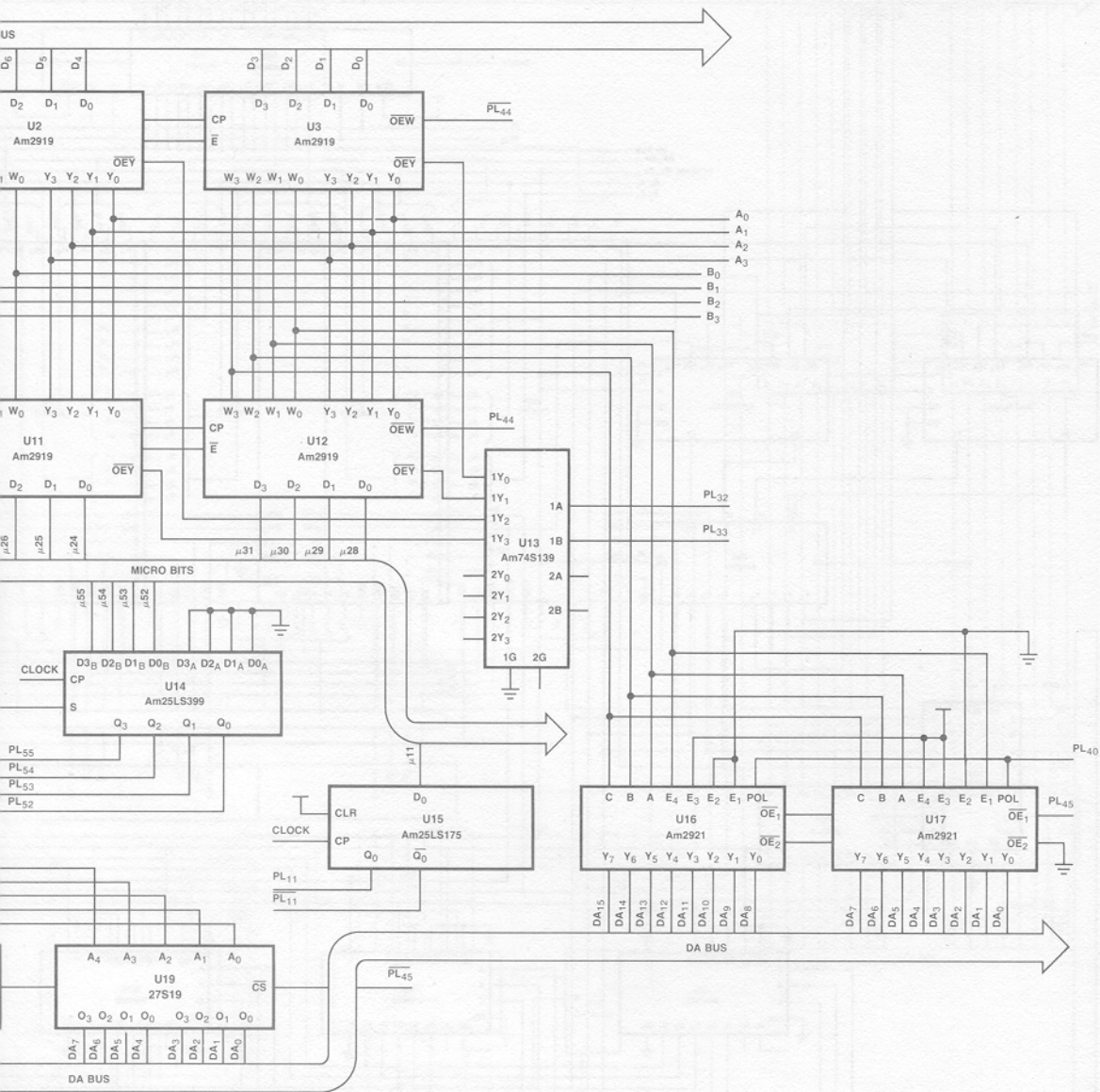


Figure 31



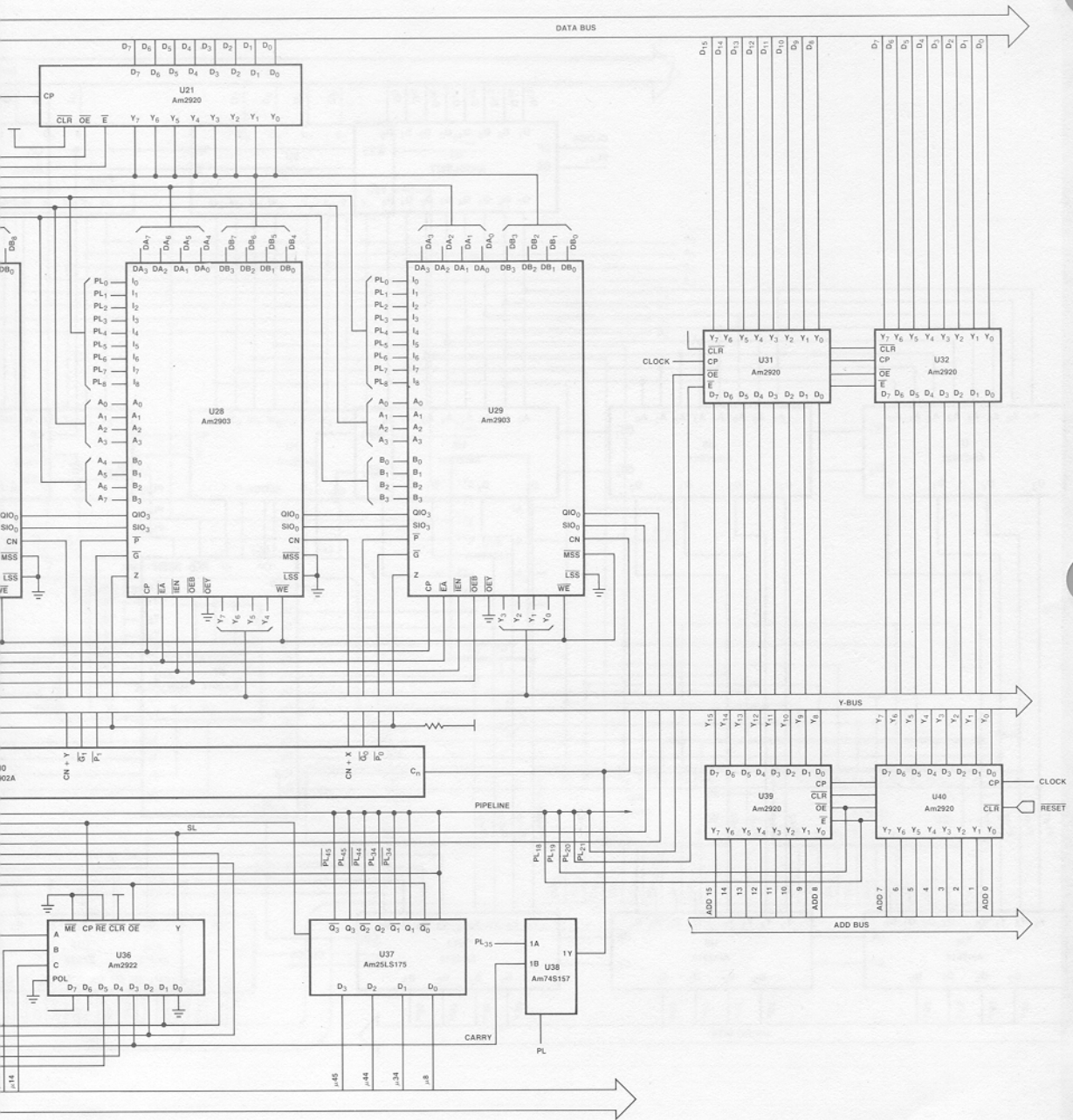
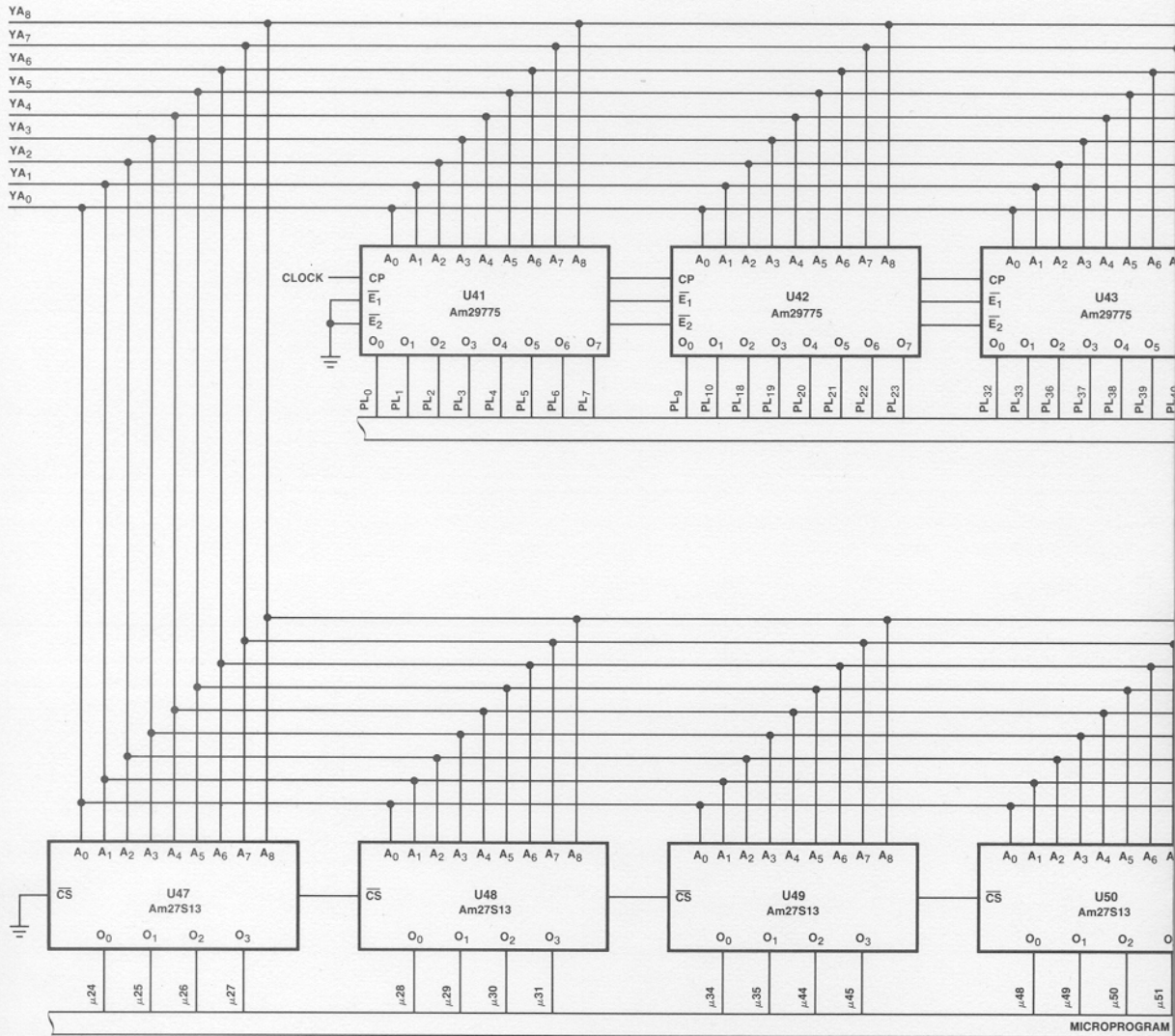
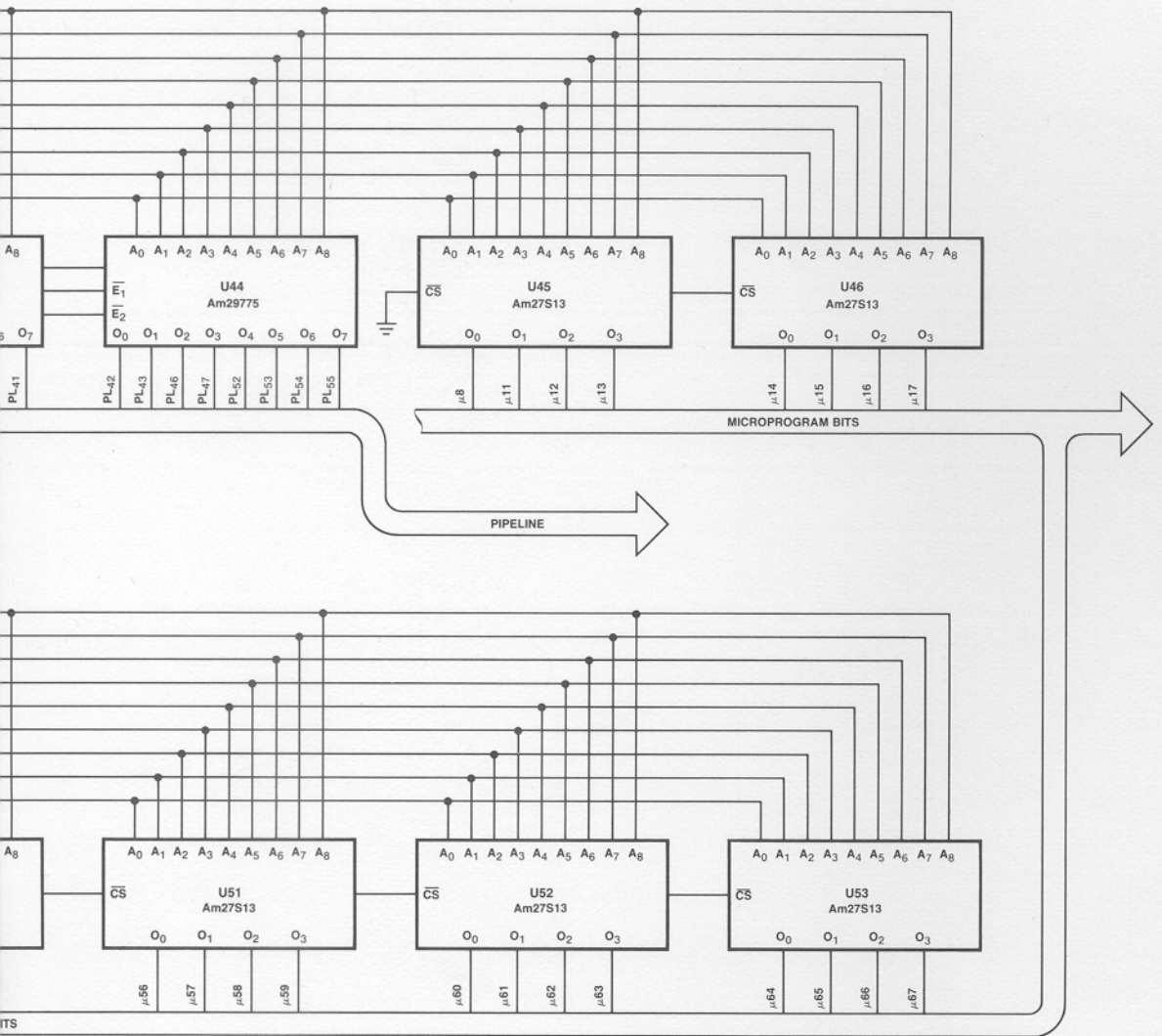


Figure 31b.



MICROPROGRAM

Figure 31c





**ADVANCED
MICRO
DEVICES, INC.**

901 Thompson Place
Sunnyvale

California 94086

(408) 732-2400

TWX: 910-339-9280

TELEX: 34-6306

TOLL FREE

(800) 538-8450

8-78