Advanced Micro Devices

The Am2900 Family Data Book

Advanced Micro Devices

The Am2900 Family Data Book With Related Support Circuits

901 Thompson Place, P.O. Box 453, Sunnyvale, California 94086
(408) 732-2400 TWX: 910-339-9280 TELEX: 34-6306

AM-PUB003

Am2900 COMPONENTS CONTINUOUSLY BECOME FASTER AND FASTER

MORE SPEED: NO MORE POWER

There's a good old tried and proven way to make faster IC's – burn more power. (That's the only real difference between "LS" and "S" devices). But that solution isn't satisfactory for LSI devices like the Am2900 family. Power is constrained to existing levels for reliability reasons.

Am2900 parts are always designed to obtain the maximum speed at a power level which is safe for the package types and operating environment of the part. To increase speeds, new technologies must be used to build faster components at no increase in power.

NEW CIRCUIT DESIGN TECHNIQUES MAKE FASTER GATES

One way to make faster components is to use new circuit design techniques. The most obvious is internal ECL, which provides very fast gates at similar power levels to LS TTL. Other design techniques, such as low-level logic (with very small logic swings on-chip), can also provide higher speeds without introducing the time penalty of ECL to TTL conversion.

Finally, very low power gates used in non-critical speed paths make more power available for use in critical speed paths. As the 2900 family develops, all these technologies will be used within a single component to achieve the highest speeds without increasing power. Among the first products to take advantage of mixed-circuit technology will be the Am2903A.

IMPROVED PROCESS CONTROL ALLOWS TIGHTER SPECS

Today's 2900 parts are carefully characterized over a wide range of voltages, temperatures, and process parameters before an AC specification is published. As manufacturing technology improves, the process is subject to smaller run-to-run variations, so that all of the product is closer to design nominal. This makes it possible to specify parameters more closely to typical without incurring large yield losses. The first product reflecting this is the Am2903.

WHAT'S GOOD FOR THE GOOSE IS GOOD FOR THE GANDER

Many new tools in production technology are emerging, primarily spurred by the emphasis on high-speed MOS memories. The same tools, such as projection masking, also provide for smaller geometries in bipolar circuits. As MOS gets faster, so does bipolar. The Am2901B obtains its speed improvement over the Am2901A through these tools.

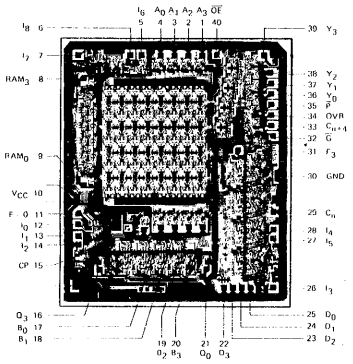
DESIGN FOR THE FUTURE

Every Am2900 part will undergo an evolution as new technologies become practical for production. Every part type will continuously become faster. Within a few short years, 2900-based designs will compete favorably with Schottky MSI on a speed basis at a fraction of the component count.

Most existing 2900 designs can be offered in higher performance versions simply by substitution of the 2901B for the 2901A, the 2909A for the 2909, the 2903A for the 2903, and so forth. Your 2900 design won't run out of speed in a few years. Advanced Micro Devices' 2900 Family will serve tomorrow's needs as well as today's.

SMALLER DIE SIZES MAKE FASTER PARTS

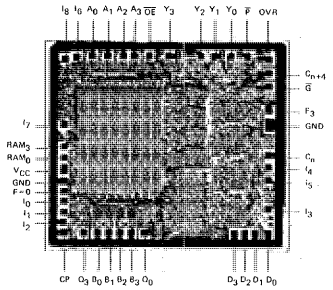
Am2901



DIE SIZE 0.167" X 0.198"

Speed A,B → $\overline{G}$,$\overline{P}$ 80ns

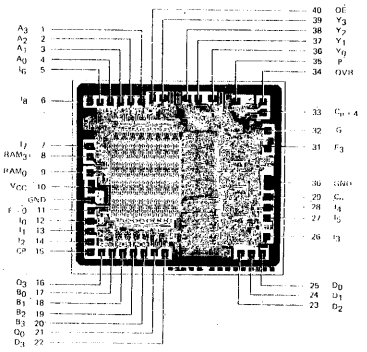
Am2901A



DIE SIZE 0.132" X 0.149"

Speed A,B → $\overline{G}$,$\overline{P}$ 65ns

Am2901B



DIE SIZE 0.117" X 0.128"

Speed A,B → $\overline{G}$,$\overline{P}$ 50ns

INTRODUCTION

THREE GENERATIONS OF TTL

Transistor-transistor logic has been the dominant technology for digital circuits since it was developed in the mid-1960's. It has proven itself to be manufacturable in high volume using an extremely reliable process technology. The processes used for TTL have evolved over the years, making components smaller, faster and less expensive. Relative to a TTL gate manufactured in 1966, a gate on a circuit manufactured today occupies 1/5 the area, consumes 1/10 the power, is twice as fast and costs less than 1/100 the price.

The circuits built using TTL technology have gone through two generations; the Am2900 Family represents the beginning of the third. Each generation consists of circuits which are fundamental building blocks of systems – circuits which can be interconnected in many different ways to build many different systems. Only by producing such universal circuits can manufacturing volumes be high enough to generate the rapid cost reductions characteristic of the integrated circuit industry.

The quality which distinguishes one generation from another is the level of integration used, and, because of the level of integration, the philosophy behind the circuit.

If one draws a curve plotting the cost of an individual gate against the number of gates on a chip, Figure 1 results.

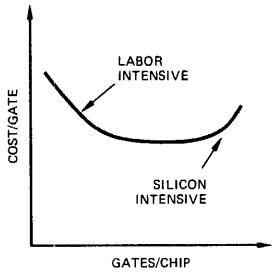


Figure 1. MPR-001

At the left, cost per gate is inversely proportional to the number of gates on the chip. The chip is small enough that it does not represent a significant portion of the cost of the product – it is virtually free. The cost of the product is composed of labor in assembly and test, the cost of processing an order, shipping and fixed overhead. Doubling the number of gates on the chip doesn't materially affect the cost so the cost per gate halves. As the number of gates per chip increases, the die begins to cost more, reversing the downward trend. As die cost dominates, the cost per gate remains relatively flat until the yield of the die begins to decline markedly. The cost per gate then begins to rise again. The lowest cost per gate is achieved at a level of integration corresponding to the flat region. This is the optimum level of integration.

As technology improves, costs are constantly reduced and the optimum level of integration occurs at more and more gates per chip.

The three curves of Figure 2 are the reason for the three generations of TTL. Each generation has consisted of fundamental system building blocks designed to take advantage of the optimum level of integration at the time.

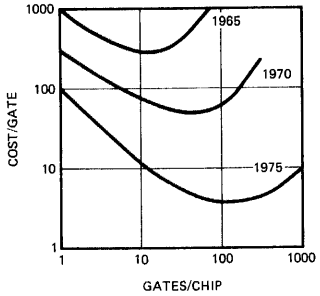


Figure 2. MPR-002

GENERATION I – SSI, 1965

In 1965, the optimum level of integration was three-to-six gates per chip. Users were delighted to buy such chips at $10-20 each. The circuits were useful in many systems. They consisted of gates – the 7400, 7410, 7420 – and, pressing the state of the art, some flip-flops. They were fundamental building blocks.

GENERATION II – MSI, 1970

Beginning around 1968, it became economical to put more gates on a chip and the industry was faced with a problem: How does one put 20 gates on a chip and build a universal building block? Clearly, one answer was to bring the inputs and outputs off chip as had been done before. But that was the wrong answer. The right answer was to redefine fundamental building blocks. The new building blocks fell into seven categories:

- Counters
- Decoders
- Multiplexers
- Operators (adders, comparators)
- Encoders
- Registers
- Latches

All systems could be defined in terms of these seven functions, and integrated circuits could be defined at the 20–50 gate/chip level which performed these functions efficiently. This, of course, is MSI. Over the last six or seven years, more and more circuits of this type have been introduced, utilizing standard gold-doped technology, low-power TTL, high-speed TTL, Schottky TTL, and now low-power Schottky TTL technology. Today, there are over 250 different MSI circuits and new ones appear every month. But in today's technology, many of these circuits are not particularly cost effective. They are too small for today's technology and their costs are labor intensive. (Labor costs do not follow traditional semiconductor pricing patterns.) In 1977, the optimum level of integration for bipolar logic is around 500 gates/chip.

GENERATION III – The Am2900 Family, 1976

At a 500-gate-per-chip level of integration, one does not build counters, decoders, and multiplexers. A new definition of fundamental system functions is needed. Advanced Micro Devices has defined these eight categories:

- Data Manipulation
- Microprogram Control
- Macroprogram Control
- Priority Interrupt
- Direct Memory Access
- I/O Control
- Memory Control
- Front Panel Control

The Am2900 Family consists of circuits designed to perform those functions efficiently. They are fundamental system building blocks; they contain hundreds of gates per chip; they are fast – utilizing Low-Power Schottky TTL technology; they are expandable; they are flexible – useful in emulation; and they are driven under microprogram control.

THE Am2900 FAMILY

The Am2900 Family consists of a series of LSI building blocks designed for use in microprogrammed computers and controllers. Each device is designed to be expandable and sufficiently flexible to be suitable for emulation of many existing machines. It is the wide variety of machine architectures possible with the Am2900 Family which sets it apart from the fixed-instruction microprocessors such as the Am9080A.

While an Am9080A can be used to build a microcomputer with only four or five packages, an Am2900 design will require 30 or 40 or more. The Am9080A design will, therefore, almost always be cheaper. But the Am9080A, or any other fixed-instruction processor, can execute only one instruction set, so it is not really suitable for emulation of another machine.

Moreover, a fixed-instruction processor operates only on words of a single length, usually eight bits. An Am2900 design, on the other hand, can be constructed for any word length which is a multiple of four bits.

Many applications require specialized operations to be performed at relatively high speed. Such functions as multiply and divide and special graphic control operations, can be done in microcode 10-100 times faster than in fixed-instruction MOS processors.

MICROPROGRAMMED ARCHITECTURE

Most small processors today are being designed using a technique called microprogramming. In microprogrammed systems, a large portion of the system's control is performed by a read only memory (usually PROM) rather than large arrays of gates and flip-flops. This technique frequently reduces the package count in the controller and provides a highly ordered structure in the controller, not present when random logic is used. Moreover, microprogramming makes changes in the machines' instruction set very simple to perform – reducing the post-production engineering costs for the system substantially.

The Am2900 Family of Bipolar LSI devices has been designed for use in microprogrammed systems. Each device performs a basic system function and is driven by a set of control lines from a microinstruction.

Figure 3 illustrates a typical system architecture. There are two "sides" to the system. At the left is the control circuitry and on the right is the data manipulation circuitry. The block labeled "2901 array" consists of the ALU, scratchpad registers, data steering logic (all internal to the Am2901's), plus left/right shift control and carry lookahead circuit. Data is processed by moving it from main memory (not shown) into the 2901 registers, performing the required operations on it and returning the result to main memory. Memory addresses may also be generated in the 2901's and sent out to the memory address register (MAR). The four status bits from the 2901's ALU are captured in the status register after each operation.

The logic on the left side is the control section of the computer. This is where the Am2909, 2910, or 2911 is used. The entire system is controlled by a memory, usually PROM, which contains long words called microinstructions. Each microinstruction contains bits to control each of the data manipulation elements in the system. There are, for example, nine bits for the 2901 instruction lines, eight bits for the A and B register addresses, two or three bits to control the shifting multiplexers at the ends of the 2901 array (Figure 19 on 2901 data sheet), and bits to control the register enables on the MAR, instruction register, and various bus transceivers. When the bits in a microinstruction are applied to all the data elements and everything is clocked, then one small operation (such as a data transfer or a register-to-register add) will occur.

A "machine instruction" (such as a minicomputer instruction or a 9080A instruction) is performed by executing several microinstructions in sequence. Each microinstruction therefore contains not only bits to control the data hardware, but also bits to define the location in PROM of the next microinstruction to be executed. The fields are labeled in Figure 3 as I, CC, and BA. The I field controls the sequencer. It indicates where the next address is located – the μPC, the stack, or the direct inputs – and whether the stack is to be pushed or popped.

The CC field contains bits indicating the conditions under which the I field applies. These are compared with the condition codes in the status register and may cause modification to the I field. The comparing and modification occurs in the block labeled "control logic". Frequently this is a PROM or PLA. In the case of the Am2910, it is built into the chip. The BA field is a branch address or the address of a subroutine.

PIPELINING

The address for the microinstructions is generated by the sequencer, starting from a clock edge. The address goes from the sequencer to the ROM and, an access time later, the microinstruction is at the ROM outputs.

A pipeline register is a register placed on the output of the microprogram memory to essentially split the system in two. The pipeline register contains the microinstruction currently being executed ①. (Refer to the circled numbers in Figure 3.) The data manipulation control bits go out to the system elements and a portion of the microinstruction is returned to the sequencer ② to determine the address of the next microinstruction to be executed. That address ③ is sent to the ROM and the next microinstruction ④ sits at the input of the pipeline register. So while the 2901's are executing one instruction, the next instruction is being fetched from ROM. Note that there is no sequential logic in the sequencer between the select lines and the output. This is important because the loop ① to ② to ③ to ④ must occur during a single clock cycle. During the same time, the loop from ① to ⑤ must occur in the 2901's. These two paths are roughly the same (around 200ns worst case for a 16-bit system). The presence of the pipeline register allows the microinstruction fetch to occur in parallel with the data operation rather than serially, allowing the clock frequency to be doubled.

Introduction

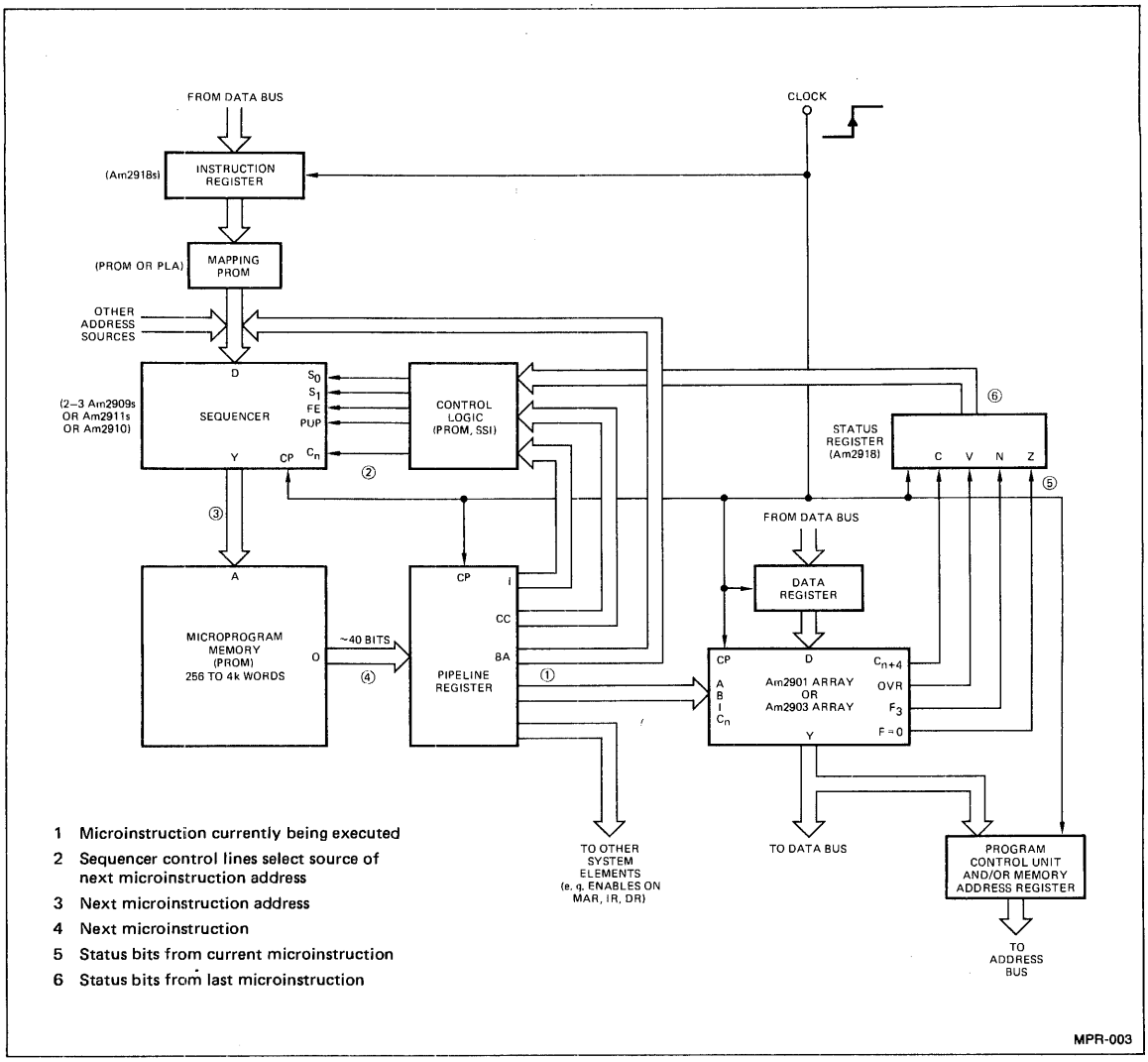


Figure 3.

The system shown in Figure 3 works as follows. A sequence of microinstructions in the PROM is executed to fetch an instruction from main memory. This requires that the program counter, often in a 2901 working register, be sent to the memory address register and incremented. The data returned from memory is loaded into the instruction register. The contents of the instruction register is passed through a PROM or PLA to generate the address of the first microinstruction which must be executed to perform the required function. A branch to this address occurs through the sequencer. Several microinstructions may be executed to fetch data from memory, perform ALU operations, test for overflow, and so forth. Then a branch will be made back to the instruction fetch cycle. At this point, there may be branches to other sections of microcode. For example, the machine might test for an interrupt here and obtain an interrupt service routine address from another mapping ROM rather than start on the next machine instruction. There are obviously many possibilities. Throughout this data book, in application notes, and within data sheets, some suggested techniques will be found.

Additional application notes are in preparation and are planned for publication. Advanced Micro Devices' Applications' staff is available to answer questions and provide technical assistance as well. They may be reached by calling (408) 732-2400, or, outside California (800) 538-8450. Ask for Am2900 Family Applications.

Am2901 • Am2901A • Am2901B

Four-Bit Bipolar Microprocessor Slice

2

DISTINCTIVE CHARACTERISTICS

- Two-address architecture –
  Independent simultaneous access to two working registers saves machine cycles.
- Eight-function ALU –
  Performs addition, two subtraction operations, and five logic functions on two source operands.
- Flexible data source selection –
  ALU data is selected from five source ports for a total of 203 source operand pairs for every ALU function.
- Left/right shift independent of ALU –
  Add and shift operations take only one cycle.
- Four status flags –
  Carry, overflow, zero, and negative.
- Expandable –
  Connect any number of Am2901's together for longer word lengths.
- Microprogrammable –
  Three groups of three bits each for source operand, ALU function, and destination control.
- Fast –
  Am2901B is up to 27% faster than Am2901A, up to 50% faster than Am2901. The Am2901B meets or exceeds all of the specifications for the Am2901 and Am2901A.

TABLE OF CONTENTS

For applications information see the last part of this data sheet and chapters III and IV of "Build a Microcomputer", AMD's application note series on the Am2900 Family.

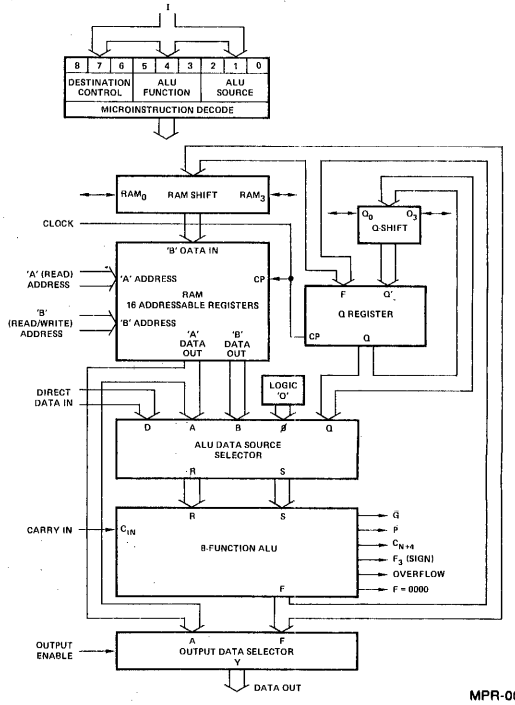
GENERAL DESCRIPTION

The four-bit bipolar microprocessor slice is designed as a high-speed cascadable element intended for use in CPU's, peripheral controllers, programmable microprocessors and numerous other applications. The microinstruction flexibility of the Am2901 will allow efficient emulation of almost any digital computing machine.

The device, as shown in the block diagram below, consists of a 16-word by 4-bit two-port RAM, a high-speed ALU, and the associated shifting, decoding and multiplexing circuitry. The nine-bit microinstruction word is organized into three groups of three bits each and selects the ALU source operands, the ALU function, and the ALU destination register. The microprocessor is cascadable with full look-ahead or with ripple carry, has three-state outputs, and provides various status flag outputs from the ALU. Advanced low-power Schottky processing is used to fabricate this 40-lead LSI chip.

The Am2901B is a plug-in replacement for the Am2901 or Am2901A, but is 25% faster than the Am2901A and 50% faster than the Am2901.

MICROPROCESSOR SLICE BLOCK DIAGRAM



MPR-004

ARCHITECTURE

A detailed block diagram of the bipolar microprogrammable microprocessor structure is shown in Figure 1. 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 1 block diagram are the 16-word by 4-bit 2-port RAM and the high-speed ALU.

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 1, 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 Am2901 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 definition of I_0, I_1, and I_2 for the eight source operand combinations are as shown in Figure 2. Also shown is the octal code for each selection.

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 inputs is shown in Figure 3. The octal code is also shown for reference. The normal technique for cascading the ALU of several devices is in a look-ahead carry mode. Carry generate, $\overline{G}$, and carry propagate, $\overline{P}$, are outputs of the device for use with a carry-look-ahead-generator such as the Am2902. 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.

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 4.

The four-bit data output field (Y) features three-state outputs and can be directly bus organized. An output control ($\overline{OE}$) is used to enable the three-state outputs. When $\overline{OE}$ 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. Refer to Figure 4 for the selected output for each microinstruction code combination.

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 (÷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 as defined in Figure 4.

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 as shown in Figure 4.

The clock input to the Am2901 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.

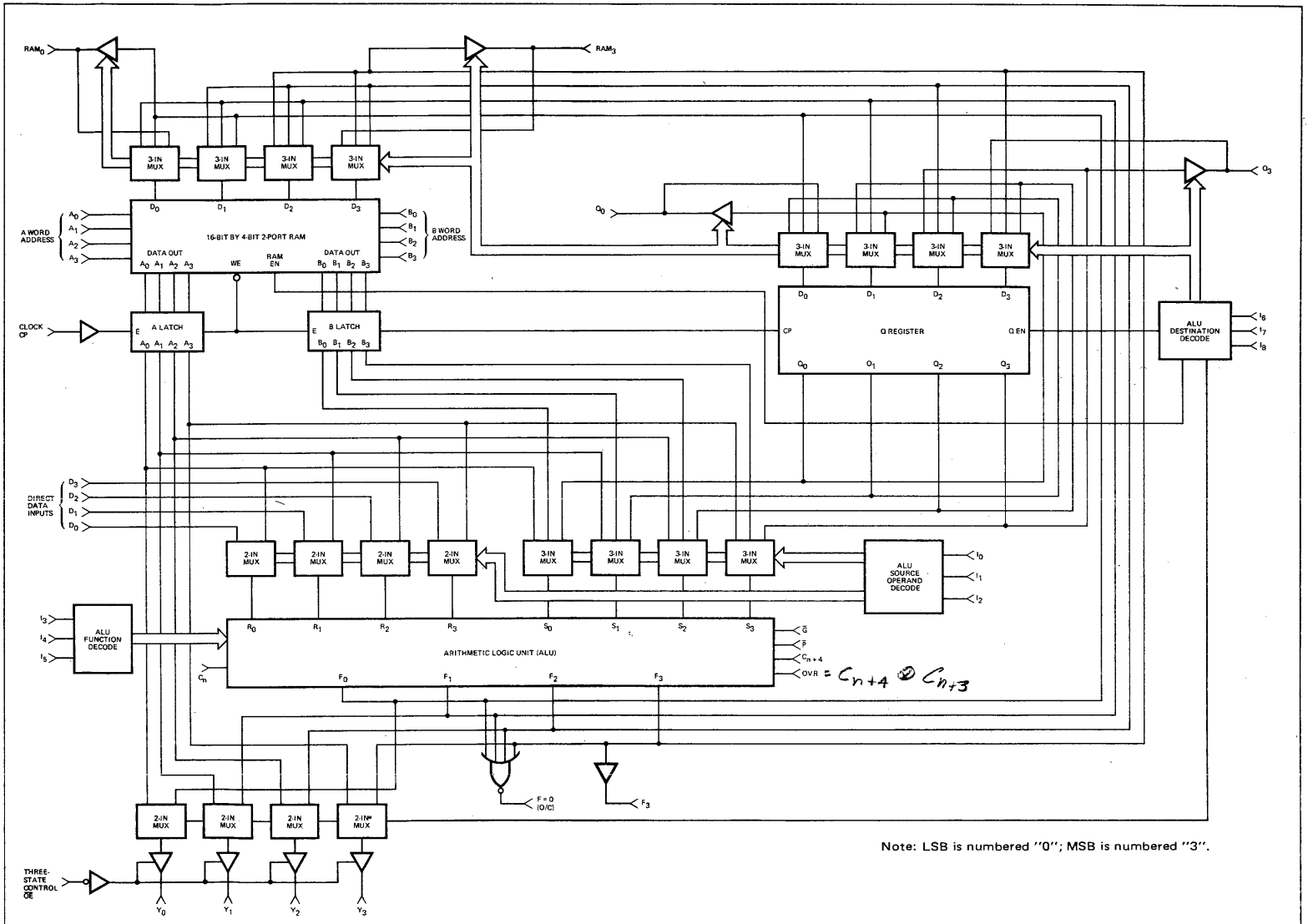


Figure 1. Detailed Am2901B Microprocessor Block Diagram.

MPR-005

	MICRO CODE				ALU SOURCE OPERANDS	
Mnemonic	I_2	I_1	I_0	Octal Code	R	S
AQ	L	L	L	0	A	Q
AB	L	L	H	1	A	B
ZQ	L	H	L	2	0	Q
ZB	L	H	H	3	0	B
ZA	H	L	L	4	0	A
DA	H	L	H	5	D	A
DQ	H	H	L	6	D	Q
DZ	H	H	H	7	D	0

Figure 2. ALU Source Operand Control.

	MICRO CODE				ALU	
Mnemonic	I_5	I_4	I_3	Octal Code	Function	SYMBOL
ADD	L	L	L	0	R Plus S	R + S
SUBR	L	L	H	1	S Minus R	S − R
SUBS	L	H	L	2	R Minus S	R − S
OR	L	H	H	3	R OR S	$R \vee S$
AND	H	L	L	4	R AND S	$R \wedge S$
NOTRS	H	L	H	5	$\overline{R}$ AND S	$\overline{R} \wedge S$
EXOR	H	H	L	6	R EX OR S	$R \forall S$
EXNOR	H	H	H	7	R EX NOR S	$\overline{R \forall S}$

Figure 3. ALU Function Control.

	MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y	RAM SHIFTER		Q SHIFTER	
Mnemonic	I_8	I_7	I_6	Octal Code	Shift	Load	Shift	Load	OUTPUT	RAM_0	RAM_3	Q_0	Q_3
QREG	L	L	L	0	X	NONE	NONE	F → Q	F	X	X	X	X
NOP	L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
RAMA	L	H	L	2	NONE	F → B	X	NONE	A	X	X	X	X
RAMF	L	H	H	3	NONE	F → B	X	NONE	F	X	X	X	X
RAMQD	H	L	L	4	DOWN	F/2 → B	DOWN	Q/2 → Q	F	F_0	IN_3	Q_0	IN_3
RAMD	H	L	H	5	DOWN	F/2 → B	X	NONE	F	F_0	IN_3	Q_0	X
RAMQU	H	H	L	6	UP	2F → B	UP	2Q → Q	F	IN_0	F_3	IN_0	Q_3
RAMU	H	H	H	7	UP	2F → B	X	NONE	F	IN_0	F_3	X	Q_3

✳ DON'T USE □ DISABLE $\overline{FEN}$ △ ENABLE $\overline{FEN}$

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state
B = Register Addressed by B inputs.
UP is toward MSB, DOWN is toward LSB.

Figure 4. ALU Destination Control.

OCTAL I_{210}	OCTAL I_{543} ALU Source / ALU Function	0	1	2	3	4	5	6	7
		A, Q	A, B	0, Q	0, B	0, A	D, A	D, Q	D, 0
0	C_n = L R Plus S	A+Q	A+B	Q	B	A	D+A	D+Q	D
	C_n = H	A+Q+1	A+B+1	Q+1	B+1	A+1	D+A+1	D+Q+1	D+1
1	C_n = L S Minus R	Q−A−1	B−A−1	Q−1	B−1	A−1	A−D−1	Q−D−1	−D−1
	C_n = H	Q−A	B−A	Q	B	A	A−D	Q−D	−D
2	C_n = L R Minus S	A−Q−1	A−B−1	−Q−1	−B−1	−A−1	D−A−1	D−Q−1	D−1
	C_n = H	A−Q	A−B	−Q	−B	−A	D−A	D−Q	D
3	R OR S	$A \vee Q$	$A \vee B$	Q	B	A	$D \vee A$	$D \vee Q$	D
4	R AND S	$A \wedge Q$	$A \wedge B$	0	0	0	$D \wedge A$	$D \wedge Q$	0
5	$\overline{R}$ AND S	$\overline{A} \wedge Q$	$\overline{A} \wedge B$	Q	B	A	$\overline{D} \wedge A$	$\overline{D} \wedge Q$	0
6	R EX-OR S	$A \forall Q$	$A \forall B$	Q	B	A	$D \forall A$	$D \forall Q$	D
7	R EX-NOR S	$\overline{A \forall Q}$	$\overline{A \forall B}$	$\overline{Q}$	$\overline{B}$	$\overline{A}$	$\overline{D \forall A}$	$\overline{D \forall Q}$	$\overline{D}$

+ = Plus; − = Minus; $\vee$ = OR; $\wedge$ = AND; $\forall$ = EX-OR

Figure 5. Source Operand and ALU Function Matrix.

SOURCE OPERANDS AND ALU FUNCTIONS

There are eight source operand pairs available to the ALU as selected by the I_0, I_1, and I_2 instruction inputs. The ALU can perform eight functions; five logic and three arithmetic. The I_3, I_4, and I_5 instruction inputs control this function selection. The carry input, C_n, also affects the ALU results when in the arithmetic mode. The C_n input has no effect in the logic mode. When I_0 through I_5 and C_n are viewed together, the matrix of Figure 5 results. This matrix fully defines the ALU/source operand function for each state.

The ALU functions can also be examined on a "task" basis, i.e., add, subtract, AND, OR, etc. In the arithmetic mode, the carry will affect the function performed while in the logic mode, the carry will have no bearing on the ALU output. Figure 6 defines the various logic operations that the Am2901 can perform and Figure 7 shows the arithmetic functions of the device. Both carry-in LOW (C_n = 0) and carry-in HIGH (C_n = 1) are defined in these operations.

2

Octal I_{543}, I_{210}	Group	Function
40	AND	A∧Q
41		A∧B
45		D∧A
46		D∧Q
30	OR	A∨Q
31		A∨B
35		D∨A
36		D∨Q
60	EX-OR	A∀Q
61		A∀B
65		D∀A
66		D∀Q
70	EX-NOR	$\overline{A \forall Q}$
71		$\overline{A \forall B}$
75		$\overline{D \forall A}$
76		$\overline{D \forall Q}$
72	INVERT	$\overline{Q}$
73		$\overline{B}$
74		$\overline{A}$
77		$\overline{D}$
62	PASS	Q
63		B
64		A
67		D
32	PASS	Q
33		B
34		A
37		D
42	"ZERO"	0
43		0
44		0
47		0
50	MASK	$\overline{A}$∧Q
51		$\overline{A}$∧B
55		$\overline{D}$∧A
56		$\overline{D}$∧Q

Figure 6. ALU Logic Mode Functions.

Octal	C_n = 0 (Low)		C_n = 1 (High)	
I_{543}, I_{210}	Group	Function	Group	Function
00	ADD	A+Q	ADD plus one	A+Q+1
01		A+B		A+B+1
05		D+A		D+A+1
06		D+Q		D+Q+1
02	PASS	Q	Increment	Q+1
03		B		B+1
04		A		A+1
07		D		D+1
12	Decrement	Q–1	PASS	Q
13		B–1		B
14		A–1		A
27		D–1		D
22	1's Comp.	–Q–1	2's Comp. (Negate)	–Q
23		–B–1		–B
24		–A–1		–A
17		–D–1		–D
10	Subtract (1's Comp)	Q–A–1	Subtract (2's Comp)	Q–A
11		B–A–1		B–A
15		A–D–1		A–D
16		Q–D–1		Q–D
20		A–Q–1		A–Q
21		A–B–1		A–B
25		D–A–1		D–A
26		D–Q–1		D–Q

Figure 7. ALU Arithmetic Mode Functions.

LOGIC FUNCTIONS FOR G, P, C_{n+4}, AND OVR

The four signals G, P, C_{n+4}, and OVR are designed to indicate carry and overflow conditions when the Am2901 is in the add or subtract mode. The table below indicates the logic equations for these four signals for each of the eight ALU functions. The R and S inputs are the two inputs selected according to Figure 2.

Definitions (+ = OR)

$P_0 = R_0 + S_0$ $\quad G_0 = R_0S_0$

$P_1 = R_1 + S_1$ $\quad G_1 = R_1S_1$

$P_2 = R_2 + S_2$ $\quad G_2 = R_2S_2$

$P_3 = R_3 + S_3$ $\quad G_3 = R_3S_3$

$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_n$

$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_n$

I_{543}	Function	$\overline{P}$	$\overline{G}$	C_{n+4}	OVR
0	R + S	$\overline{P_3P_2P_1P_0}$	$\overline{G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0}$	C_4	$C_3 \forall C_4$
1	S − R	← Same as R + S equations, but substitute $\overline{R_i}$ for R_i in definitions →			
2	R − S	← Same as R + S equations, but substitute $\overline{S_i}$ for S_i in definitions →			
3	R ∨ S	LOW	$P_3P_2P_1P_0$	$\overline{P_3P_2P_1P_0} + C_n$	$\overline{P_3P_2P_1P_0} + C_n$
4	R ∧ S	LOW	$\overline{G_3 + G_2 + G_1 + G_0}$	$G_3 + G_2 + G_1 + G_0 + C_n$	$G_3 + G_2 + G_1 + G_0 + C_n$
5	$\overline{R} \wedge S$	LOW	← Same as R ∧ S equations, but substitute $\overline{R_i}$ for R_i in definitions →		
6	R ∀ S	← Same as $\overline{R \forall S}$, but substitute $\overline{R_i}$ for R_i in definitions →			
7	$\overline{R \forall S}$	$G_3 + G_2 + G_1 + G_0$	$G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1P_0$	$\overline{G_3 + P_3G_2 + P_3P_2G_1} + P_3P_2P_1P_0 (G_0 + \overline{C_n})$	Seenote

Note: $[\overline{P}_2+\overline{G}_2\overline{P}_1+\overline{G}_2\overline{G}_1\overline{P}_0+\overline{G}_2\overline{G}_1\overline{G}_0C_n] \forall [\overline{P}_3+\overline{G}_3\overline{P}_2+\overline{G}_3\overline{G}_2\overline{P}_1+\overline{G}_3\overline{G}_2\overline{G}_1\overline{P}_0+\overline{G}_3\overline{G}_2\overline{G}_1\overline{G}_0C_n]$ + = OR

Figure 8.

ORDERING INFORMATION

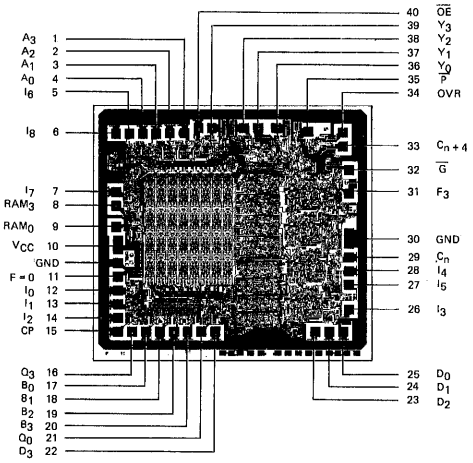
Order the part number according to the table below to obtain the desired package, temperature range, and screening level.

Am2901 Order Number	Am2901A Order Number	Am2901B Order Number	Package Type (Note 1)	Operating Range (Note 2)	Screening Level (Note 3)
AM2901PC	AM2901APC	AM2901BPC	P-40	C	C-1
AM2901DC	AM2901ADC	AM2901BDC	D-40	C	C-1
AM2901DC-B	AM2901ADC-B	AM2901BDC-B	D-40	C	B-2 (Note 4)
	AM2901ADM	AM2901BDM	D-40	M	C-3
	AM2901ADM-B	AM2901BDM-B	D-40	M	B-3
	AM2901AFM	AM2901BFM	F-42	M	C-3
	AM2901AFM-B	AM2901BFM-B	F-42	M	B-3
	AM2901AXC	AM2901BXC	Dice	C	Visual inspection to MIL-STD-883 Method 2010B.
	AM2901AXM	AM2901BXM	Dice	M	

Notes: 1. P = Molded DIP, D = Hermetic DIP, F = Flat Pak. Number following letter is number of leads. See Appendix B for detailed outline. Where Appendix B contains several dash numbers, any of the variations of the package may be used unless otherwise specified.
2. C = 0°C to +70°C, V_{CC} = 4.75V to 5.25V, M = − 55°C to +125°C, V_{CC} = 4.50V to 5.50V.
3. See Appendix A for details of screening. Levels C-1 and C-3 conform to MIL-STD-883, Class C. Level B-3 conforms to MIL-STD-883, Class B.
4. 96 hour burn-in.

Figure 9.

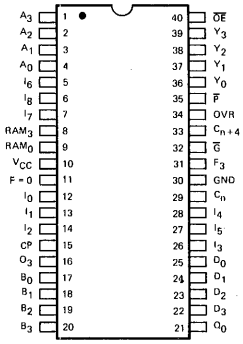
METALLIZATION AND PAD LAYOUT



Am2901B
DIE SIZE 0.117" X 0.128"

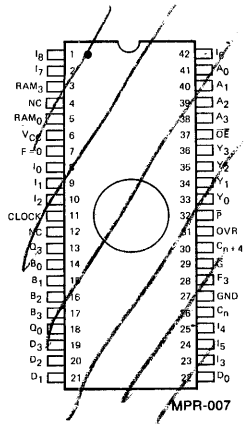
CONNECTION DIAGRAM
Top View

DIP



MPR-006

FLAT PACK CONNECTIONS
Top View



MPR-007

Note: Pin 1 is marked for orientation.

Figure 10.

2

PIN DEFINITIONS

A_{0-3} The four address inputs to the register stack used to select one register whose contents are displayed through the A-port.

B_{0-3} The four address inputs to the register stack used to select one register whose contents are displayed through the B-port and into which new data can be written when the clock goes LOW.

I_{0-8} The nine instruction control lines. Used to determine what data sources will be applied to the ALU (I_{012}), what function the ALU will perform (I_{345}), and what data is to be deposited in the Q-register or the register stack (I_{678}).

Q_3 RAM_3 A shift line at the MSB of the Q register (Q_3) and the register stack (RAM_3). Electrically these lines are three-state outputs connected to TTL inputs internal to the device. When the destination code on I_{678} indicates an up shift (octal 6 or 7) the three-state outputs are enabled and the MSB of the Q register is available on the Q_3 pin and the MSB of the ALU output is available on the RAM_3 pin. Otherwise, the three-state outputs are OFF (high-impedance) and the pins are electrically LS-TTL inputs. When the destination code calls for a down shift, the pins are used as the data inputs to the MSB of the Q register (octal 4) and RAM (octal 4 or 5).

Q_0 RAM_0 Shift lines like Q_3 and RAM_3, but at the LSB of the Q-register and RAM. These pins are tied to the Q_3 and RAM_3 pins of the adjacent device to transfer data between devices for up and down shifts of the Q register and ALU data.

D_{0-3} Direct data inputs. A four-bit data field which may be selected as one of the ALU data sources for entering data into the device. D_0 is the LSB.

Y_{0-3} The four data outputs. These are three-state output lines. When enabled, they display either the four outputs of the ALU or the data on the A-port of the register stack, as determined by the destination code I_{678}.

$\overline{OE}$ Output Enable. When $\overline{OE}$ is HIGH, the Y outputs are OFF; when $\overline{OE}$ is LOW, the Y outputs are active (HIGH or LOW).

$\overline{G}$, $\overline{P}$ The carry generate and propagate outputs of the internal ALU. These signals are used with the Am2902 for carry-lookahead.

OVR Overflow. This pin is logically the Exclusive-OR of the carry-in and carry-out of the MSB of the ALU. At the most significant end of the word, this pin indicates that the result of an arithmetic two's complement operation has overflowed into the sign-bit.

F = 0 This is an open collector output which goes HIGH (OFF) if the data on the four ALU outputs F_{0-3} are all LOW. In positive logic, it indicates the result of an ALU operation is zero.

F_3 The most significant ALU output bit.

C_n The carry-in to the internal ALU.

C_{n+4} The carry-out of the internal ALU.

CP The clock input. The Q register and register stack outputs change on the clock LOW-to-HIGH transition. The clock LOW time is internally the write enable to the 16 x 4 RAM which compromises the "master" latches of the register stack. While the clock is LOW, the "slave" latches on the RAM outputs are closed, storing the data previously on the RAM outputs. This allows synchronous master-slave operation of the register stack.

MAXIMUM RATINGS (Above which the useful life may be impaired)

Storage Temperature	−65°C to +150°C
Temperature (Ambient) Under Bias	−55°C to +125°C
Supply Voltage to Ground Potential	−0.5 V to +7.0 V
DC Voltage Applied to Outputs for HIGH Output State	−0.5 V to +V_{CC} max.
DC Input Voltage	−0.5 V to +5.5 V
DC Output Current, Into Outputs	30 mA
DC Input Current	−30 mA to +5.0 mA

OPERATING RANGE

Part Number Suffix	V_{CC}	Temperature
PC, PCB, DC, DCB XC	4.75V to 5.25V	T_A = 0°C to +70°C
DM, DMB FM, FMB XM	4.50V to 5.50V	T_C = −55°C to +125°C

Notes on Testing

Incoming test procedures on this device should be carefully planned, taking into account the high complexity and power levels of the part. The following notes may be useful.

1. Insure the part is adequately decoupled at the test head. Large changes in V_{CC} current as the device switches may cause erroneous function failures due to V_{CC} changes.
2. Do not leave inputs floating during any tests, as they may start to oscillate at high frequency.
3. Do not attempt to perform threshold tests at high speed. Following an input transition, ground current may change by as much as 400mA in 5-8ns. Inductance in the ground cable may allow the ground pin at the device to rise by 100's of millivolts momentarily.
4. Use extreme care in defining input levels for AC tests. Many inputs may be changed at once, so there will be significant noise at the device pins and they may not actually reach V_{IL} or V_{IH} until the noise has settled. AMD recommends using $V_{IL} \leq$ 0.4V and $V_{IH} \geq$ 2.4V for AC tests.
5. To simplify failure analysis, programs should be designed to perform DC, Function, and AC tests as three distinct groups of tests.
6. To assist in testing, AMD offers complete documentation on our test procedures and, in most cases, can provide Fairchild Sentry programs, under license.

USING THE Am2901

BASIC SYSTEM ARCHITECTURE

The Am2901 is designed to be used in microprogrammed systems. Figure 15 illustrates such an architecture. The nine instruction lines, the A and B addresses, and the D data inputs normally will all come from registers clocked at the same time as the Am2901. The register inputs come from a ROM or PROM – the "microprogram store". This memory contains sequences of microinstructions, typically 28 to 40 bits wide, which apply the proper control signals to the Am2901's and other circuits to execute the desired operation.

The address lines of the microprogram store are driven from the Am2910 microprogram sequencer. This device has facilities for storing an address, incrementing an address, jumping to any address, and linking subroutines. The Am2910 is controlled by some of the bits coming from the microprogram store. Essentially these bits are the "next instruction" control.

Note that with the microprogram register in-between the microprogram memory store and the Am2901's, an instruction accessed on one cycle is executed on the next cycle. As one instruction is executed, the next instruction is being read from microprogram memory. In this configuration, system speed is improved because the execution time in the Am2901's occurs in parallel with the access time of the microprogram store. Without the "pipeline register", these two functions must occur serially.

EXPANSION OF THE Am2901

The Am2901 is a four-bit CPU slice. Any number of Am2901's can be interconnected to form CPU's of 12, 16, 24, 36 or more bits, in four-bit increments. Figure 16 illustrates the interconnection of three Am2901's to form a 12-bit CPU, using ripple carry. Figure 17 illustrates a 16-bit CPU using carry lookahead, and Figure 18 is the general carry lookahead scheme for long words.

With the exception of the carry interconnection, all expansion schemes are the same. Refer to Figure 16. The Q_3 and RAM_3 pins are bidirectional left/right shift lines at the MSB of the device. For all devices except the most significant, these lines are connected to the Q_0 and RAM_0 pins of the adjacent more significant device. These connections allow the Q-registers of all Am2901's to be shifted left or right as a contiguous n-bit register, and also allow the ALU output data to be shifted left or right as a contiguous n-bit word prior to storage in the RAM. At the LSB and MSB of the CPU, the shift pins should be connected to three-state multiplexers which can be controlled by the microcode to select the appropriate input signals to the shift inputs. (See Figure 19)

The open collector F = 0 outputs of all the Am2901's are connected together and to a pull-up resistor. This line will go HIGH if and only if the output of the ALU contains all zeroes. Most systems will use this line as the Z (zero) bit of the processor status word.

The overflow and F_3 pins are generally used only at the most significant end of the array, and are meaningful only when two's complement signed arithmetic is used. The overflow pin is the Exclusive-OR of the carry-in and carry-out of the sign bit (MSB). It will go HIGH when the result of an arithmetic

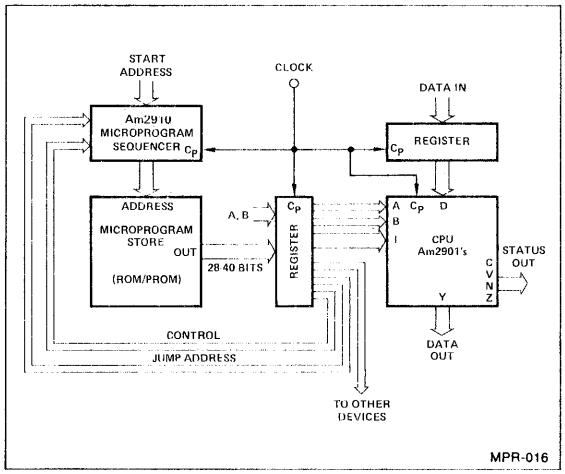


Figure 15. Microprogrammed Architecture Around Am2901's.

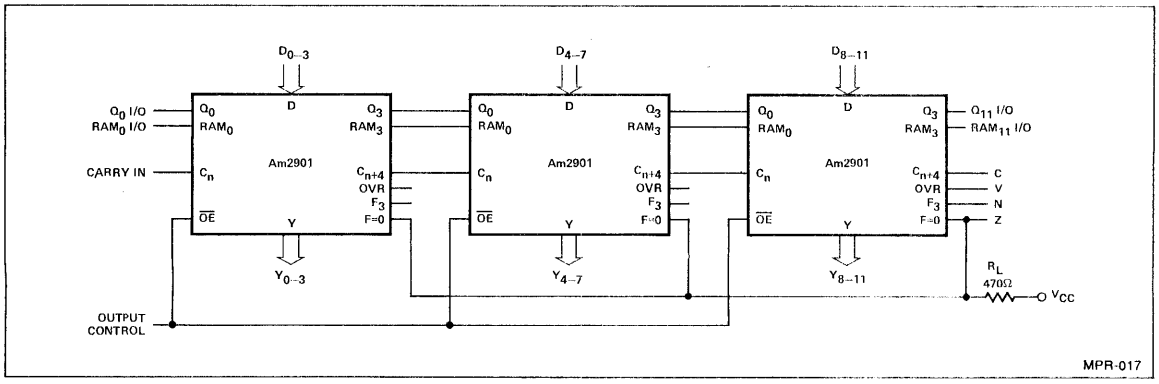


Figure 16. Three Am2901's used to Construct 12-Bit CPU with Ripple Carry. Corresponding A, B, and I Pins on all Devices are Connected Together.

operation is a number requiring more bits than are available, causing the sign bit to be erroneous. This is the overflow (V) bit of the processor status word. The F_3 pin is the MSB of the ALU output. It is the sign of the result in two's complement notation, and should be used as the Negative (N) bit of the processor status word.

The carry-out from the most significant Am2901 (C_{n+4} pin) is the carry-out from the array, and is used as the carry (C) bit of the processor status word.

Carry interconnections between devices may use either ripple carry or carry lookahead. For ripple carry, the carry-out (C_{n+4}) of each device is connected to the carry-in (C_n) of the next more significant device. Carry lookahead uses the Am2902 lookahead carry generator. The scheme is identical to that used with the 74181/74182. Users unfamiliar with this technique should refer to AMD's application note on Arithmetic Logic Units. Figures 17 and 18 illustrate single and multiple level lookahead.

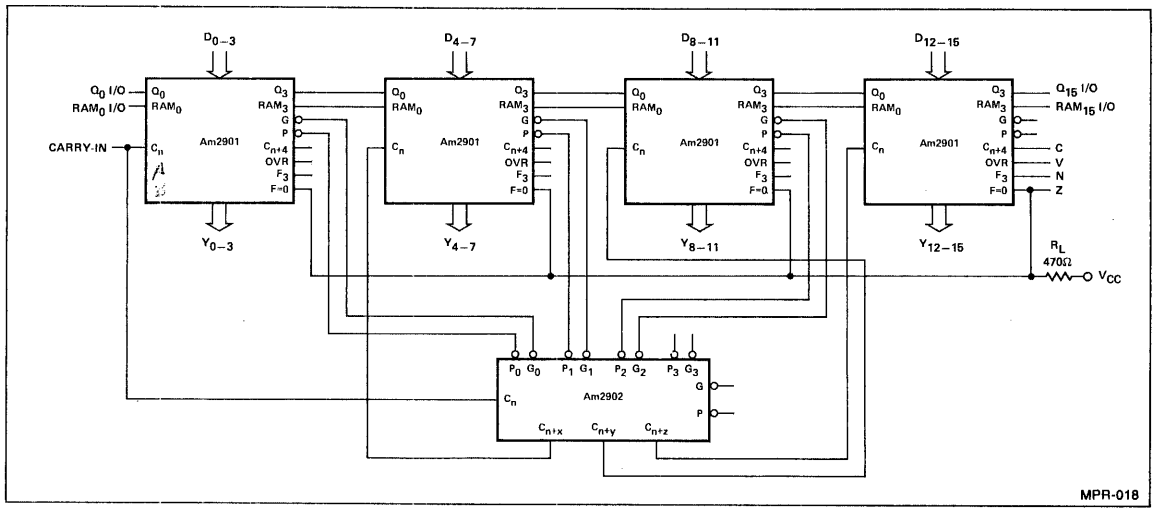


Figure 17. Four Am2901s in a 16-Bit CPU Using the Am2902 for Carry Lookahead.

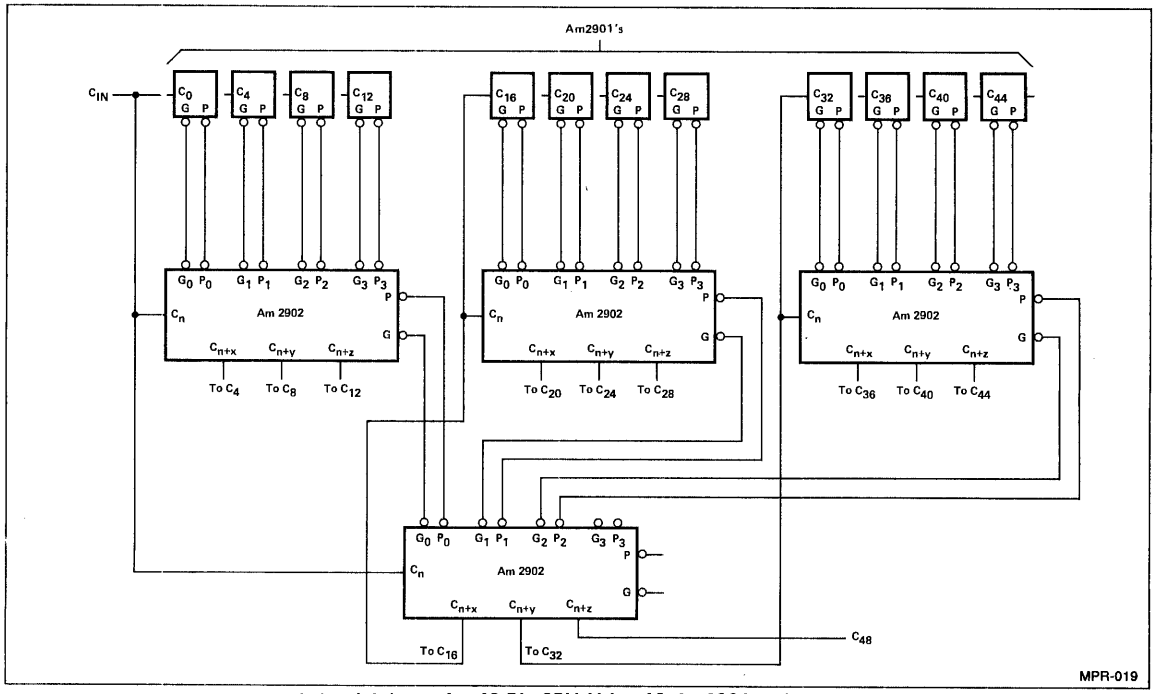


Figure 18. Carry Lookahead Scheme for 48-Bit CPU Using 12 Am2901s. The Carry-Out Flag (C_{48}) Should be Taken From the Lower Am2902 Rather Than the Right-Most Am2901 for Higher Speed.

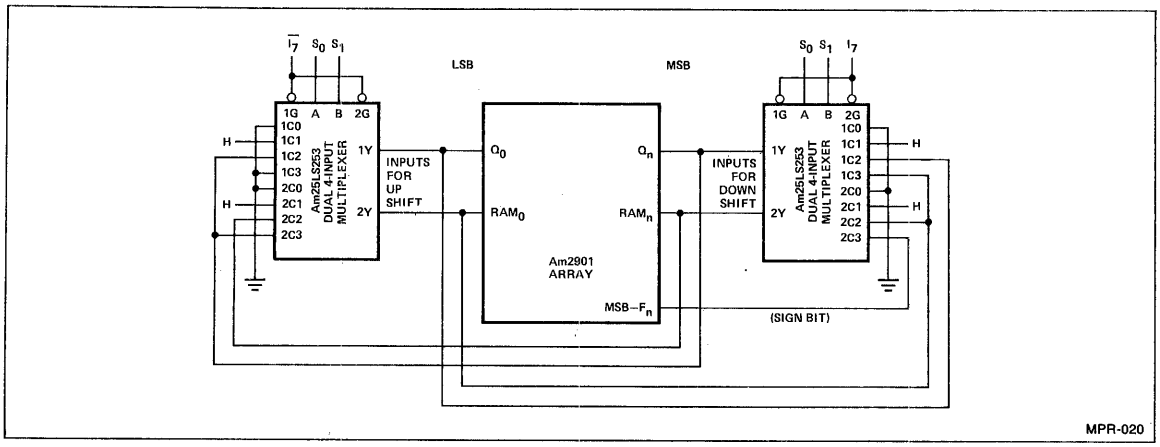


Figure 19. Three-State Multiplexers Used on Shift I/O Lines.

SHIFT I/O LINES AT THE END OF THE ARRAY

The Q-register and RAM left/right shift data transfers occur between devices over bidirectional lines. At the ends of the array, three-state multiplexers are used to select what the new inputs to the registers should be during shifting. The Am2904 includes these multiplexers in a single LSI chip. Figure 19 shows two Am25LS253 dual four-input multiplexers connected to provide four shift modes. Instruction bit I_7 (from the Am2901) is used to select whether the left-shift multiplexer or the right-shift multiplexer is active. The four shift modes in this example are:

Zero A LOW is shifted into the MSB of the RAM on a down shift. If the Q-register is also shifted, then a LOW is deposited in the Q-register MSB. If the RAM or both registers are shifted up, LOWs are placed in the LSBs.

One Same as zero, but a HIGH level is deposited in the LSB or MSB.

Rotate A single precision rotate. The RAM MSB shifts into the LSB on a right shift and the LSB shifts into the MSB on a left shift. The Q-register, if shifted, will rotate in the same manner.

Arithmetic A double-length Arithmetic Shift if Q is also shifted. On an up shift a zero is loaded into the Q-register LSB and the Q-register MSB is loaded into the RAM LSB. On a down shift, the RAM LSB is loaded into the Q-register MSB and the ALU output MSB (F_n, the sign bit) is loaded into the RAM MSB. (This same bit will also be in the next less significant RAM bit.)

Code			Source of New Data				Shift	Type
I_7	S_1	S_0	Q_0	Q_n	RAM_0	RAM_n		
H	L	L	0	Q_{n-1}	0	F_{n-1}	Up	Zero
H	L	H	1	Q_{n-1}	1	F_{n-1}		One
H	H	L	Q_n	Q_{n-1}	F_n	F_{n-1}		Rotate
H	H	H	0	Q_{n-1}	Q_n	F_{n-1}		Arithmetic
L	L	L	Q_1	0	F_1	0	Down	Zero
L	L	H	Q_1	1	F_1	1		One
L	H	L	Q_1	Q_0	F_1	F_0		Rotate
L	H	H	Q_1	F_0	F_1	$RAM_n = RAM_{n-1} = F_n$		Arithmetic

HARDWARE MULTIPLICATION

Figure 20 illustrates the interconnections for a hardware multiplication using the Am2901. The system shown uses two devices for 8 x 8 multiplication, but the expansion to more bits is simple — the significant connections are at the LSB and MSB only.

The basic technique used is the "add and shift" algorithm. One clock cycle is required for each bit of the multiplier. On each cycle, the LSB of the multiplier is examined; if it is a "1", then the multiplicand is added to the partial product to generate a new partial product. The partial product is then shifted one place toward the LSB, and the multiplier is also shifted one place toward the LSB. The old LSB of the multiplier is discarded. The cycle is then repeated on the new LSB of the multiplier available at Q_0.

The multiplier is in the Am2901 Q-register. The multiplicand is in one of the registers in the register stack, R_a. The product will be developed in another of the registers in the stack, R_b.

The A address inputs are used to address the multiplicand in R_a, and the B address inputs are used to address the partial product in R_b. On each cycle, R_a is conditionally added to R_b, depending on the LSB of Q as read from the Q_0 output, and both Q and the ALU output are shifted down one place. The instruction lines to the Am2901 on every cycle will be:

I_{876} = 4 (shift register stack input and Q register left)
I_{543} = 0 (Add)
I_{210} = 1 or 3 (select A, B or 0, B as ALU sources)

Figure 20 shows the connections for multiplication. The circled numbers refer to the paragraphs below.

1. The adjacent pins of the Q-register and RAM shifters are connected together so that the Q-registers of both (or all) Am2901s shift left or right as a unit. Similarly, the entire eight-bit (or more) ALU output can be shifted as a unit prior to storage in the register stack.

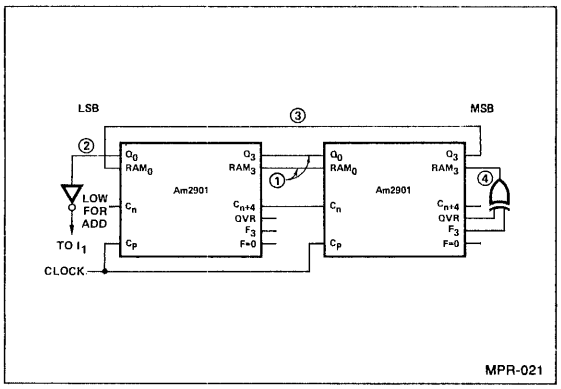


Figure 20. Interconnection for Dedicated Multiplication (8 by 8 Bit) (Corresponding A, B and I Connected Together).

2. The shift output at the LSB of the Q-register determines whether the ALU source operands will be A and B (add multiplicand to partial product) or 0 and B (add nothing to partial product. Instruction bit I_1 can select between A, B or 0, B as the source operands; it can be driven directly from the complement of the LSB of the multiplier.

3. As the new partial product appears at the input to the register stack, it is shifted left by the RAM shifter. The new LSB of the partial product, which is complete and will not be affected by future operations, is available on the RAM_0 pin. This signal is returned to the MSB of the Q-register. On each cycle then, the just-completed LSB of the product is deposited in the MSB of the Q-register; the Q-register fills with the least significant half of the product.

4. As the ALU output is shifted down on each cycle, the sign bit of the new partial product should be inserted in the RAM MSB shift input. The F_3 flag will be the correct sign of the partial product unless overflow has occurred. If overflow occurs during an addition or subtraction, the OVR flag will go HIGH and F_3 is not the sign of the result. The sign of the result must then be the complement of F_3. The correct sign bit to shift into the MSB of the partial product is therefore $F3 \oplus OVR$; that is, F_3 if overflow has not occurred and $\overline{F_3}$ if overflow has occurred. On the last cycle, when the MSB of the multiplier is examined, a conditional subtraction rather than addition should be performed, because the sign bit of the multiplier carries negative rather than positive arithmetic weight.

$$Y = -Y_i 2^i + Y_{i-1} 2^{i-1} + \ldots + Y_0 2^0$$

This scheme will produce a correct two's complement product for all multiplicands and multipliers in two's complement notation.

Figure 21 is a table showing the input states of the Am2901 for each step of a signed, two's complement multiplication. The Am2904 LSI chip conveniently implements the required shift linkages and the EX-OR function for this algorithm.

Initial Register States

R	
0	Multiplier
1	Multiplicand
2	X
3	X

Am2901 Microcode

Program 2's Comp. Multiply
Date 8/5/75 By J. S.

Final Register States

R	
0	Multiplier
1	Multiplicand
2	LSH Product
3	MSH Product

S, F →	D	Description	Repeat	Pin States (Octal)										Jump	
				A	B	I_{876}	I_{543}	I_{210}	C_n	Q_0	Q_3	RAM_0	RAM_3	To	If
O ∨ A	Q	Move Multiplier to Q	–	0	X	0	3	4	X	X	X	X	X		
O ∧ B	B	Clear R_3	–	X	3	2	4	3	X	X	X	X	X		
(O+B)/2 (A+B)/2	B	Cond. Add & Shift	n–1	1	3	4	0	1 or 3 $I_1 = Q_0$ LO	0	–	RAM_0	–	$F_3 \veebar OVR$		
(B–O)/2 (B–A)/2	B	Cond. Subt. & Shift	–	1	3	4	1	1 or 3 $I_1 = Q_0$ LO	1	–	RAM_0	–	$F_3 \veebar OVR$		
O ∨ Q	B	Move LSH Prod. to R_2	–	X	2	2	3	2	X	X	X	X	X		

X = Don't Care S = Source F = Function D = Destination

Figure 21.

Hardware Division

Division, unlike multiplication, is much more difficult to realize. One of these difficulties can be easily understood by visualizing a 2n-bit Dividend (X) and an n-bit Divisor (Y). The Quotient (Q) can range from 1 bit (when $X \leq Y$) to 2 n bits (when $Y = 1$), discarding the attempt to divide by 0. In most of the divide functions, the Remainder (R) is as important to find as is the Quotient – there is no equivalent to it in multiplication. Division becomes even more complicated when negative numbers are represented in the 2's complement notation. In the "everyday" decimal system, using Sign-and-Magnitude notation, dealing with negative numbers is relatively easy: The sign of the quotient is determined first and then a normal division is performed. Note that in this "normal" division we first "guess" the first digit of the quotient by comparing the most significant part of the dividend to the divisor. Then verify our guess by a multiplication (no "direct" division method is known), and continue to do so for all of the other digits, shifting the divisor to the right one place at a time.

The most straightforward division scheme (for unsigned numbers) is Subsequent Subtraction. The algorithm is as follows: Subtract divisor from dividend and increment a counter (initially reset to zero). Continue to do so as long as the Remainder is positive. When the Remainder becomes negative, cancel the last step; i.e., add back divisor and decrement counter. The counter will contain the Quotient and the Remainder will be correct. The main drawback of this scheme is, of course, the great number of arithmetic operations needed. Again, when dealing with signed numbers, the subtraction should be substituted by addition and vice versa.

A more rapid division can be realized by calculating the Quotient digits instead of counting them. In this algorithm, the divisor is first subtracted from the most significant part of the dividend. If the remainder is positive, the quotient digit is "1", otherwise the subtraction is cancelled (by adding the divisor to the remainder) and the quotient digit will be "0". Now shift the remainder one place to the right (much like you do in a "paper and pencil" division) and repeat until all the quotient digits have been calculated. This algorithm is called "Restoring Division". When signed numbers are involved, inversion of the operations and the quotient digits will be necessary and correction should be performed in some cases. Some time is wasted in the Restoring Division because for every "0" digit in the quotient, two arithmetic operations are needed. This can be saved in the "Non-Restoring Division".

The basis of Non-Restoring Division is the same as in Restoring Division. Consider first unsigned (positive) numbers only. At the beginning, the divisor is subtracted from the most significant part of the dividend. If the result (first remainder) is positive (or zero), the first quotient digit is "1". Otherwise, the quotient digit is "0", but do not restore! Shift divisor one place to the right (or remainder to the left) and add if last quotient digit was "0"; otherwise subtract. Determine quotient digit as before and continue until all quotient digits have been computed. The remainder will be correct if it is non-negative, otherwise correction is needed by a restoring operation (on the remainder only). Extreme care should be taken of the number of bits and the value of the divisor. Assuming the divisor has n bits and the dividend as 2n bits, the above process develops n+1 bits of the quotient. This will not be sufficient if the MSB of the divisor is "0" (which means that the divisor is a small number and more digits are needed in the quotient). Although this condition can be easily detected as overflow will occur in the first subtraction, it can be avoided by aligning the first "1" of the divisor to the MSB of the dividend (by shifting the divisor left until all leading zeros are discarded) before performing the first subtraction. Ample space should be provided for the additional bits of the quotient. Note that leading zeros in the dividend do not disturb the normal operation. The flow chart for unsigned non-restoring division is shown in Figure 22.

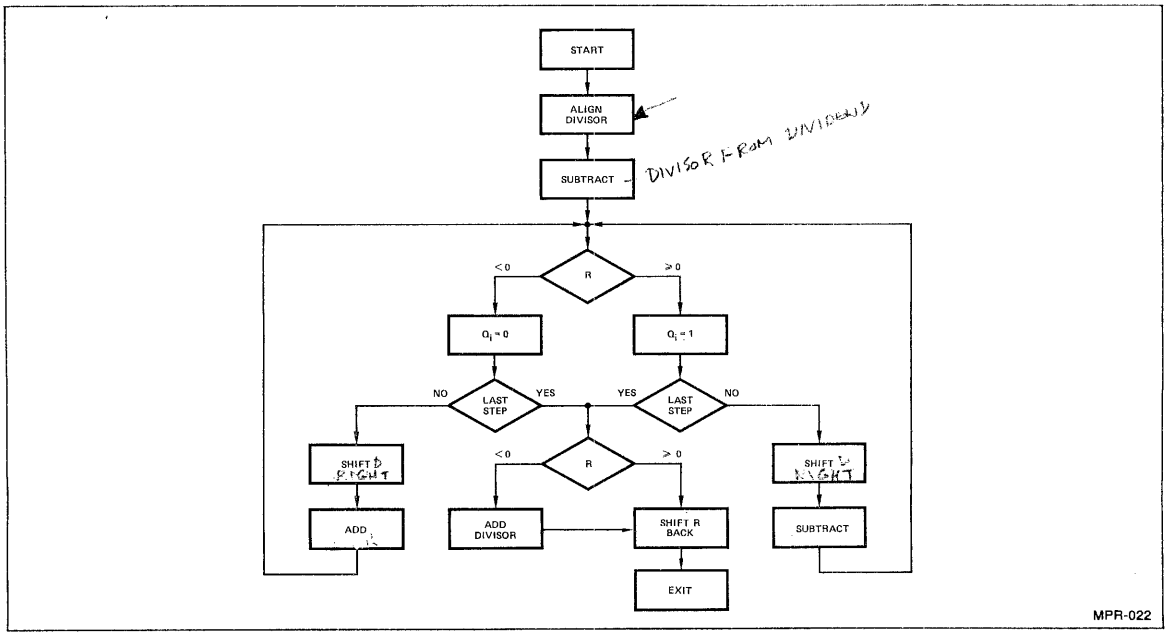


Figure 22. Flowchart for Non-Restoring Division (Unsigned Numbers).

The unsigned division scheme can be applied to signed positive numbers without any change. When negative numbers are encountered, however, changes in the algorithm are necessary. The straightforward method of signed division seems to be "Division in the first quadrant." In that scheme, negative numbers are 2's complemented to obtain positive numbers, remembering the changes done. The division is performed on positive numbers, and finally again 2's complementing occurs wherever necessary. Figure 23 is the flowchart for this algorithm.

Figure 24 is the Interconnection Diagram for both the alignment procedure and the Division Algorithm. It is assumed that the Dividend is in Register R_X (it will be lost during the division and replaced by the Remainder), the Divisor is in Register R_Y. The Quotient will be in the Q register, which should be cleared beforehand.

After checking the signs of the Dividend and Divisor, setting the flags and negating (using 23 or 24 octal as I_5 through I_0 ALU control bits) when necessary, the Divisor should be aligned. This can be done by ORing R_X with 0 (I_{5-0} = 33 octal). The most significant bit is deposited in the Status Register, and can be shifted out by setting $I_8 = I_6$ = HIGH and I_7 to the Exclusive NOR of the previous and present MSB of the Divisor. If these are both "0", I_7 will be HIGH, and an up shift will occur, filling in trailing zeros. When the checked bits are different, I_7 will go LOW, causing a down shift. At the same time the Y output of the Status Register is enabled the leftmost "0" (the sign bit) will be restored.

The first step in the Division routine is a subtract, then shift the R_X and Q registers up. I_{876} will be 6 in octal while I_{210} = 1 in octal and $I_5 = I_4$ = LOW. Pulling the CL bit in the microcode to HIGH, both I_3 and C_n will be HIGH and the ALU is performing a 2's complement subtract. The sign of the Remainder will be latched in the Status Register and the complement of it will be stored in the LSB of the Q register during the shift up operation, which also discards the sign bit of the Remainder.

Now repeating the same operation for all of the other bits of the Remainder with the CL bit in the microcode LOW will leave the control of I_3 to the (complemented) previous sign bit. If it was "0" ($R \geq 0$), I_3 and C_n will be HIGH and the ALU will subtract; if it was 1 ($R < 0$), I_3 and C_n will be LOW and the ALU will ADD, as required. In each up shift, the complement of the present sign bit will be placed at the right of the Quotient, again, as required.

At the end of the division, the sign of the Remainder should be examined and if it is HIGH, the Divisor should be added to it. This can be easily implemented (not depicted on Figure 24) by performing an unconditional ADD (with C_n LOW), letting I_2 LOW, I_0 HIGH and controlling I_1 by the complement of the sign of the Remainder, thus adding to RX either RY (if R_S = 1) or zero (if R_S = 0). If an alignment was performed, the remainder should be shifted down the same number of places.

Finally, the Quotient and/or the Remainder should be 2's complemented again according to the flags.

Figure 25 is a table showing the input states of the Am2901s for each phase of the Alignment and Division.

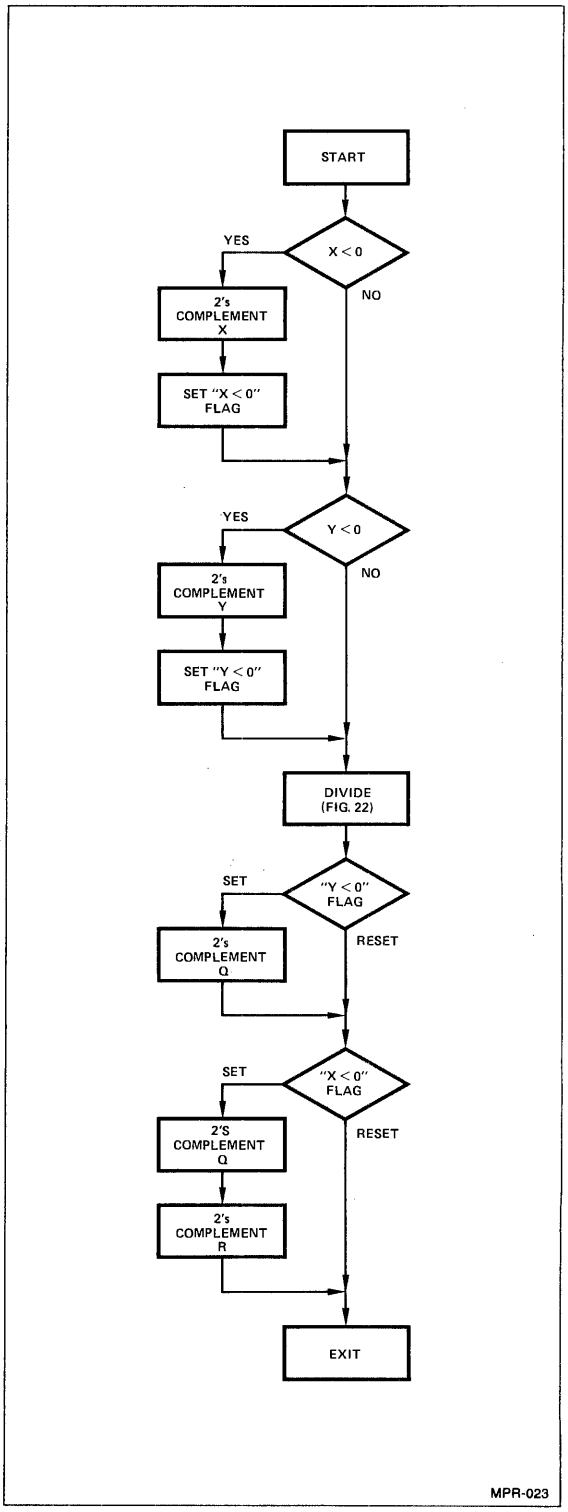


Figure 23. Flowchart for First Quadrant Division with Signed Numbers.

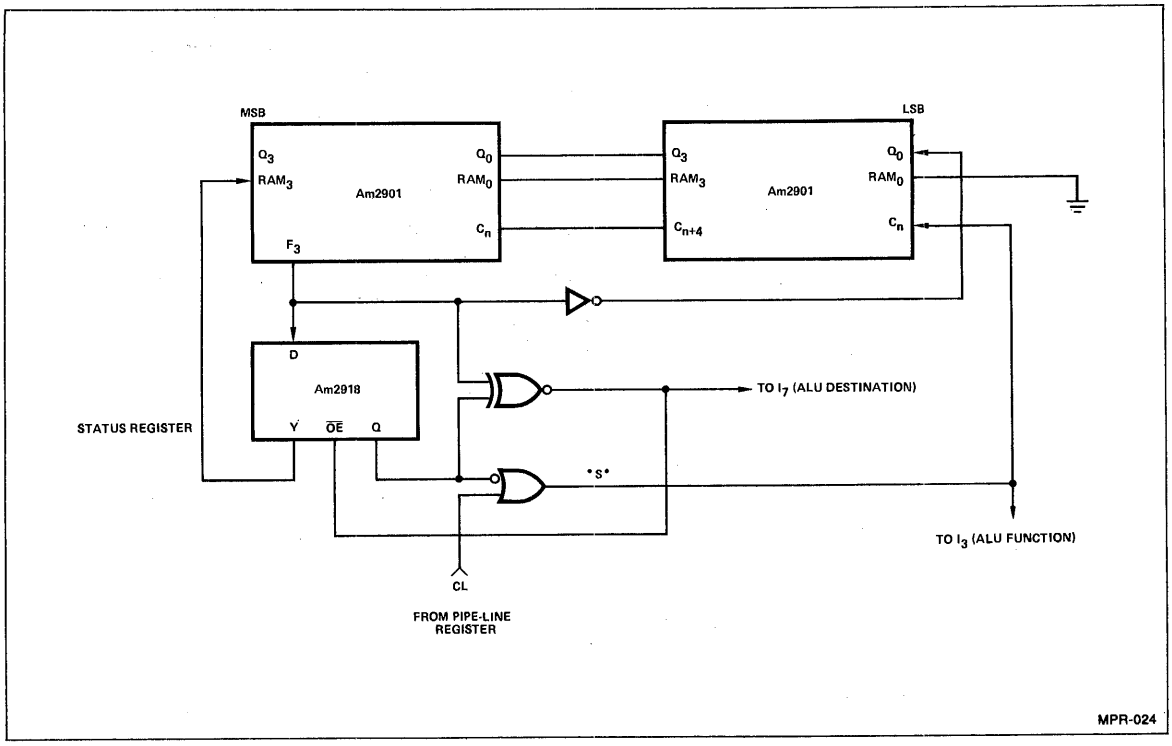


Figure 24. Interconnections for Dedicated Division.

Initial Register Status

R	
0	Dividend
1	Divisor
Q	(Cleared)

Am2901 Microcode

Program: 2's Complement Division
Date: 9/24/76 By: M.S.

Final Register Status

R	
0	Remainder
1	Divisor
Q	Quotient

S, F	D	Description	CL	Repeat	Pin Status (Octal) A	B	I_{876}	I_{543}	I_{210}	C_n	Q_0	Q_3	RAM_0	RAM_3	Jump to	Jump If
OVB	B	Align Divisor	X	k	X	1	5/7	3	3	X	X	X	0	X or Sign		
(B−A)*2	B	First Subtract & Shift	1	–	1	0	6	1	1	1	$\overline{F_3}$	X	0	X		
(B±A)*2	B	Loop Subtract/Add & Shift	0	k	1	0	6	1/0	1	1/0	$\overline{F_3}$	X	0	X		
B+A or	B	Correct Remainder	X	–	1	0	3	0	1/3	0	X	X	X	X		
B+0	B	Shift Remainder	1	k	X	1	5	1	3	1	X	X	X	Sign		

k = Number of leading zeros of the Divisor

Figure 25. Am2901 Microcode for Dedicated Division.

EXAMPLES OF SOME OTHER OPERATIONS

1. Byte Swapping

Occasionally the two halves of a 16-bit word must be swapped. D_{0-7} is interchanged with D_{8-15}. The quickest way to perform this operation is to rotate the word in RAM, shifting two bits at a time. Only four shift cycles are required. The same register is selected on both the A and B ports; the two are added together with carry-in connected to carry-out, producing a right shift of one place; then the ALU is shifted right one more place prior to storage.

Byte Swap of R_0

$A = B = 0$ $I = 701$ $RAM_0 = RAM_{15}$ $C_{IN} = C_{OUT}$

Repeat 4 times.

2. Instruction Fetch Cycle

Execution of a macroinstruction generally begins with an instruction fetch cycle. The current contents of the PC (in one of the registers) is the address of the macroinstruction to be fetched, and must be read out to the memory address register. Then the PC is incremented to point to the next macroinstruction. The macroinstruction obtained from memory is then loaded into the microprogram sequencer to cause a jump to the microcode for executing the instruction.

The PC can be read out and incremented in one cycle by using the Am2901 destination code 2, and addressing the PC with both the A and B addresses. The current value of PC will appear on the Y outputs, and PC+1 will be returned to the register. If the PC is in register 15, then:

$A = B = 15$, $I = 203$, Carry-in = 1

The PC will be on the Y outputs via the RAM A-port. On the clock LOW-to-HIGH transition, the program counter is incremented and the value on the Y outputs is loaded into the memory address register. During the following cycle, the memory is read and, on the next clock LOW-to-HIGH transition the instruction from the memory is dropped into the instruction register. The fetch operation requires only two microcycles.

ADDITIONAL READING

For more detailed information on applications of the Am2901, the following application notes are available from AMD.

Title	Publication Number
A 16-Bit Microprogrammed Computer	AM-PUB030
An Emulation of the Am9080A	AM-PUB064
A High Performance Disc Controller	AM-PUB065
Build a Microcomputer	
Chapter III	AM-PUB073-3
Chapter IV	AM-PUB073-4

Am2902A

High-Speed Look-Ahead Carry Generator

DISTINCTIVE CHARACTERISTICS

- Provides look-ahead carries across a group of four Am2901 microprocessor ALU's
- Capability of multi-level look-ahead for high-speed arithmetic operation over large word lengths
- Typical carry propagation delay of 4.5 ns
- 100% reliability assurance testing in compliance with MIL-STD-883

FUNCTIONAL DESCRIPTION

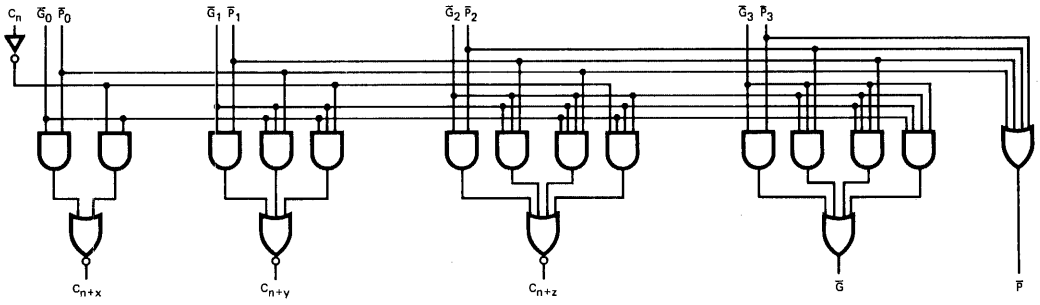
The Am2902A is a high-speed, look-ahead carry generator which accepts up to four pairs of carry propagate and carry generate signals and a carry input and provides anticipated carries across four groups of binary ALU's. The device also has carry propagate and carry generate outputs which may be used for further levels of look-ahead.

The Am2902A is generally used with the Am2901 bipolar microprocessor unit to provide look-ahead over word lengths of more than four bits. The look-ahead carry generator can be used with binary ALU's in an active LOW or active HIGH input operand mode by reinterpreting the carry functions. The connections to and from the ALU to the look-ahead carry generator are identical in both cases.

The logic equations provided at the outputs are:

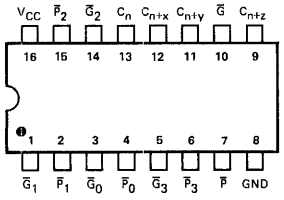
$$C_{n+x} = G_0 + P_0C_n$$
$$C_{n+y} = G_1 + P_1G_0 + P_1P_0C_n$$
$$C_{n+z} = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_n$$
$$G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$
$$P = P_3P_2P_1P_0$$

LOGIC DIAGRAM



MPR-026

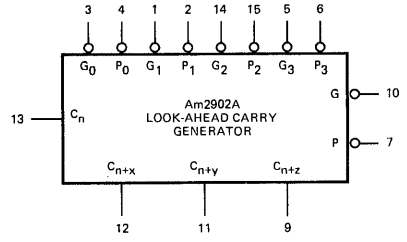
CONNECTION DIAGRAM
Top View



Note: Pin 1 is marked for orientation.

MPR-027

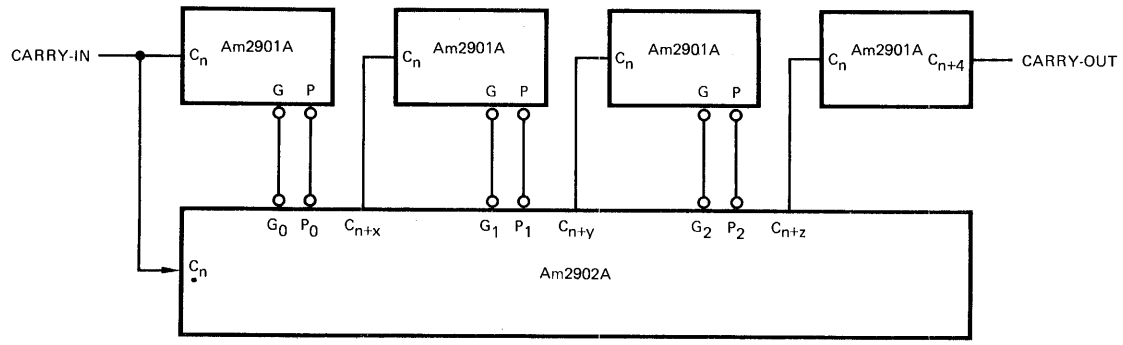
LOGIC SYMBOL



V_{CC} = Pin 16
GND = Pin 8

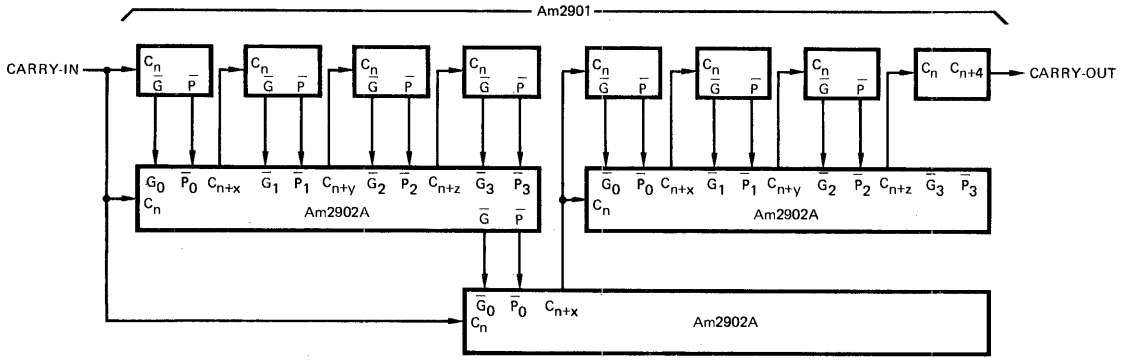
MPR-025

APPLICATIONS



16-BIT CARRY LOOK-AHEAD CONNECTION.

MPR-028



32-BIT ALU, THREE LEVEL CARRY LOOK-AHEAD.

MPR-029

ORDERING INFORMATION

Order the part number according to the table below to obtain the desired package, temperature range, and screening level.

Order Number	Package Type (Note 1)	Operating Range (Note 2)	Screening Level (Note 3)
AM2902APC	P-16	C	C-1
AM2902ADC	D-16	C	C-1
AM2902ADC-B	D-16	C	B-1
AM2902ADM	D-16	M	C-3
AM2902ADM-B	D-16	M	B-3
AM2902AFM	F-16	M	C-3
AM2902AFM-B	F-16	M	B-3
Am2902AXC	Dice	C	Visual inspection to MIL-STD-883 Method 2010B.
Am2902AXM	Dice	M	

Notes: 1. P = Molded DIP, D = Hermetic DIP, F = Flat Pak. Number following letter is number of leads. See Appendix B for detailed outline. Where Appendix B contains several dash numbers, any of the variations of the package may be used unless otherwise specified.
2. C = 0°C to +70°C, V_{CC} = 4.75V to 5.25V, M = −55°C to +125°C, V_{CC} = 4.50V to 5.50V.
3. See Appendix A for details of screening. Levels C-1 and C-3 conform to MIL-STD-883, Class C. Level B-3 conforms to MIL-STD-883, Class B.

Am2904

Status and Shift Control Unit

DISTINCTIVE CHARACTERISTICS

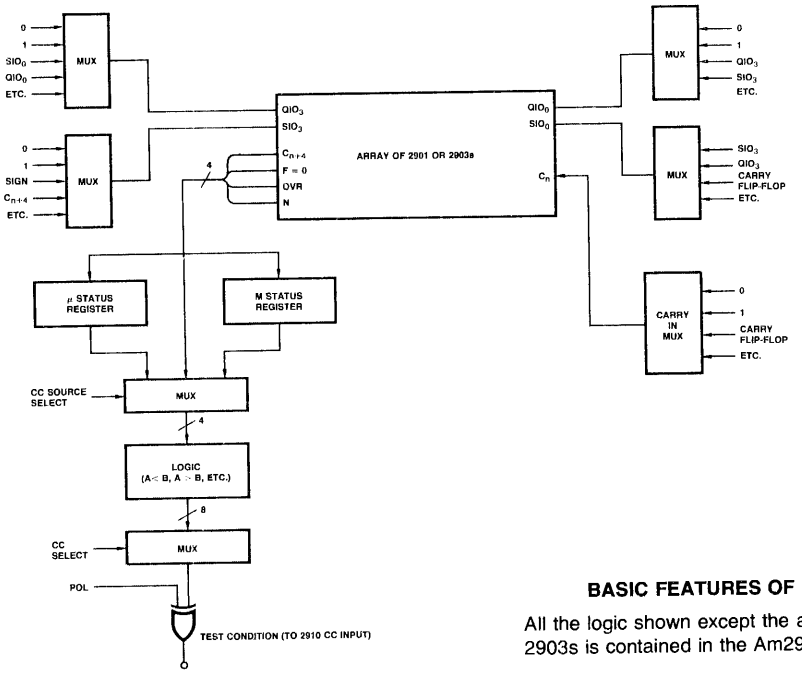
- **Replaces most MSI used around any ALU** including the Am2901, Am2903 and MSI ALUs.
- **Generates Carry-In to the ALU**
  Carry signal is selectable from 7 different sources.
- **Contains shift linkage multiplexers**
  Connects to shift lines at the ends of an Am2901 or Am2903 array to implement single and double length arithmetic and logical shifts and rotates – 32 different modes in all.
- **Contains two edge-triggered status registers**
  Use for foreground/background registers in controllers or as microlevel and machine level status registers. Bit manipulating instructions are provided.
- **Condition Code Multiplexer on chip**
  Single cycle tests for any of 16 different conditions. Tests can be performed on either of the two status registers or directly on the ALU output.

DESCRIPTION

The Am2904 is designed to perform all the miscellaneous functions which are usually performed in MSI around an ALU. These include the generation of the carry-in signal to the ALU and carry lookahead unit; the interconnection of the data path, auxiliary register, and carry flip-flop during shift operations; and the storage and testing of ALU status flags. These tasks are accomplished in the Am2904 by three nearly independent blocks of logic. The carry-in is generated by a multiplexer. The shift linkages are established by four three-state multiplexers. There are two registers for storing the carry, overflow, zero, and negative status flags. The condition code multiplexer on the Am2904 can look at true or complement of any of the four status bits and certain combinations of status bits from either of the storage registers or directly from the ALU.

For additional applications refer to Chapter 4 of "Build A Microcomputer," the AMD application book on bipolar microprocessors.

TABLE OF CONTENTS

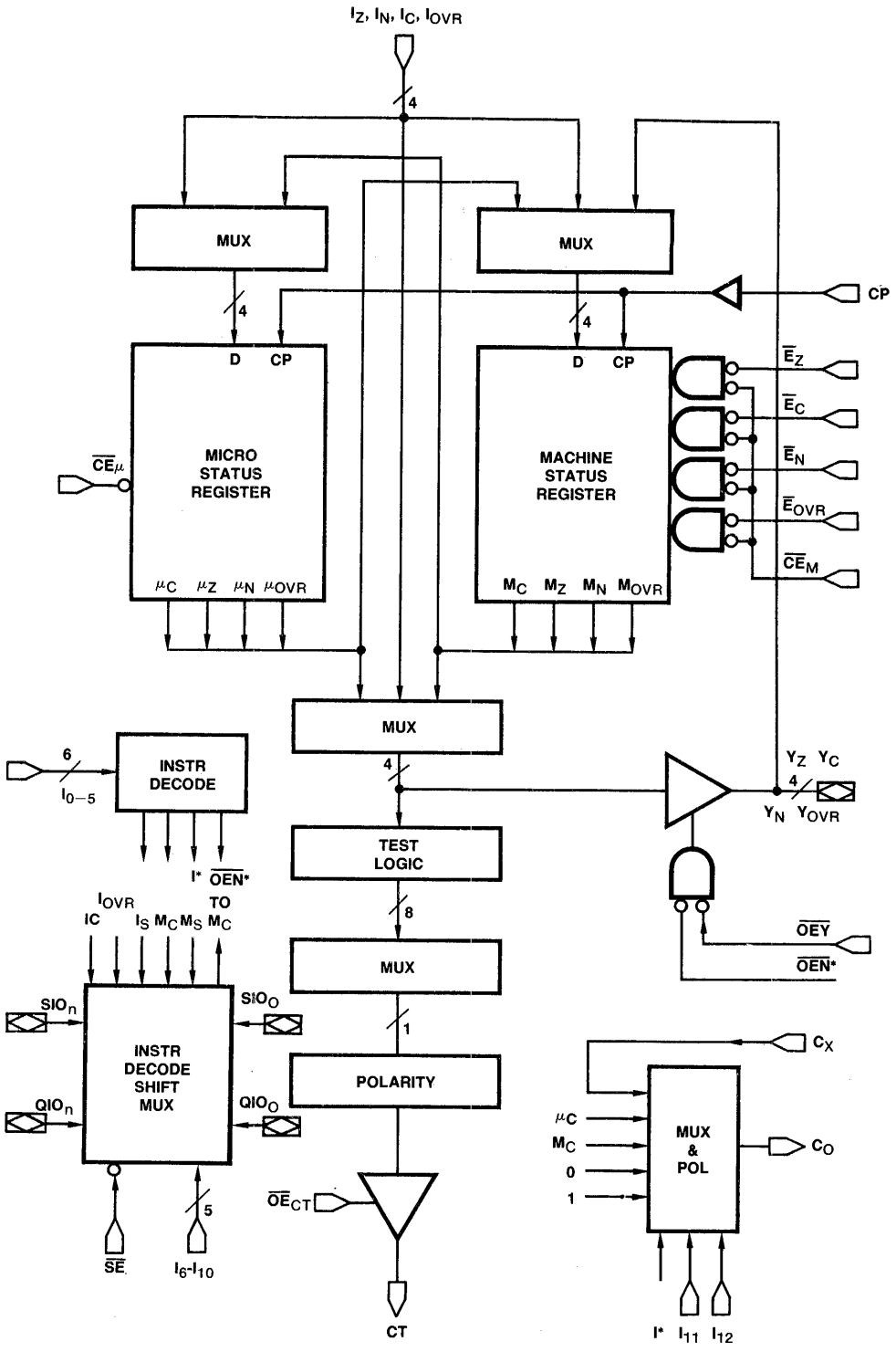


MPR-722

BASIC FEATURES OF Am2904

All the logic shown except the array of 2901s or 2903s is contained in the Am2904.

BLOCK DIAGRAM



*INTERNAL

MPR-060

Am2904 ARCHITECTURE

The Am2904 Status and Shift Control Unit provides four functions which are included in all processors. These are: a) Status Register, b) Condition Code Multiplexer, c) Shift Linkages and d) Carry-in Control. The architecture and instruction codes have been designed to complement the flexibility of the 2900 Family.

Status Register

The Am2904 contains two four-bit registers which can store the status outputs of an ALU: Carry (C), Negative (N), Zero (Z), and Overflow (OVR). They are designated Micro Status Register (μSR) and Machine Status Register (MSR). Each register can be independently controlled. The registers use edge-triggered D-type flip-flops which change state on the LOW to HIGH transition of the Clock Input.

The μSR can be loaded from the four status inputs (I_C, I_N, I_Z, I_{OVR}) or from the MSR under instruction control (I_{0-5}). The bits in the μSR can also be individually set or reset under instruction control (I_{0-5}). When the $\overline{CE\mu}$ input is HIGH, the μSR is inhibited from changing, independent of the I_{0-5} inputs.

The MSR can be loaded from the four status inputs (I_C, I_N, I_Z, I_{OVR}), from the μSR, and from the four parallel input/output pins (Y_C, Y_N, Y_Z, Y_{OVR}) under instruction control (I_{0-5}) .The MSR can also be set, reset or complemented under instruction control (I_{0-5}). The bits in the MSR can be selectively updated by controlling the four bit-enable inputs ($\overline{E_Z}$, $\overline{E_N}$, $\overline{E_C}$, $\overline{E_{OVR}}$) and the $\overline{CE_M}$ input. A LOW on both the $\overline{CE_M}$ input and the bit enable input for a specific bit enables updating that bit. A HIGH on a given bit enable input prevents the corresponding bit changing in the MSR. A HIGH on $\overline{CE_M}$ prevents any bits changing in the MSR.

The four parallel bidirectional input/output pins (Y_Z, Y_N, Y_C, Y_{OVR}) allow the contents of both the μSR and the MSR to be transferred to the system data bus and also allows the MSR to be loaded from the system data bus. This capability is used to save and restore the status registers during certain subroutines and when servicing interrupts.

Condition Code Multiplexer

The Condition Code Multiplexer output, CT, can be selected from 16 different functions. These include the true and complemented state of each of the status bits and combinations of these bits to detect such conditions as "greater than", "greater than or equal to", "less than" or "less than or equal to" for unsigned or two's complement numbers.

The Am2904 can perform these tests on the contents of the μSR, the MSR or the direct status inputs, (I_Z, I_N, I_C, I_{OVR}). The CT output is used as the test ($\overline{CC}$) input of the Am2910 and is provided with an output enable, $\overline{OE_{CT}}$ to make the addition of other condition inputs to this point easy.

Shift Linkage Multiplexer

The Shift Linkage Multiplexer generates the necessary linkages to allow the ALU to perform 32 different shift and rotate functions. Both single length and double length shifts and rotates, with and without carry (M_C), are provided. When the $\overline{SE}$ input is HIGH, the four input/output pins (SIO_0, SIO_n, QIO_0, QIO_n) are disabled. The SIO_0, SIO_n, QIO_0, QIO_n pins of the Am2904 are intended to be directly connected to the RAM_0, RAM_3, Q_0 and Q_3 pins of the Am2901 or the SIO_0, SIO_3, QIO_0, QIO_3 pins of the Am2903.

Carry-In Control Multiplexer

The Carry-In Control Multiplexer generates the C_0 output which can be selected from 7 functions (0, 1, C_X, μ_C, M_C, $\overline{\mu_C}$, $\overline{M_C}$). These functions allow easy implementation of both single length and double length addition and subtraction. The C_X input is intended to be connected to the Z output of the Am2903 to facilitate execution of some of the Am2903 special instructions. The C_0 pin is to be connected to the C_n pin of the least significant Am2901 or Am2903 and the C_n pin of the Am2902A.

Am2904 INSTRUCTION SET

The Am2904 is controlled by manipulating the 13 instruction lines, I_{0-12}, together with the nine enable lines, $\overline{CE_M}$, $\overline{CE\mu}$, $\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$, $\overline{OE_Y}$, $\overline{OE_{CT}}$, $\overline{SE}$. Most systems will save on microword bits by tying some of these lines to a fixed level or by connecting certain lines together, or by decoding microinstructions to generate appropriate Am2904 controls.

Status Registers

Instruction lines I_5, I_4, I_3, I_2, I_1, I_0 control the Status Registers. Below, these lines are referred to as two octal digits.

Micro Status Register (μSR)

The instruction codes for the Micro Status Register fall into three groups: Bit Operations, Register Operations and Load Operations (See Table 1 and Map 1). All operations require

TABLE 1. MICRO STATUS REGISTER INSTRUCTION CODES.

Bit Operations

I_{543210} Octal	μSR Operation	Comments
10	0 → μ_Z	RESET ZERO BIT
11	1 → μ_Z	SET ZERO BIT
12	0 → μ_C	RESET CARRY BIT
13	1 → μ_C	SET CARRY BIT
14	0 → μ_N	RESET SIGN BIT
15	1 → μ_N	SET SIGN BIT
16	0 → μ_{OVR}	RESET OVERFLOW BIT
17	1 → μ_{OVR}	SET OVERFLOW BIT

Register Operations

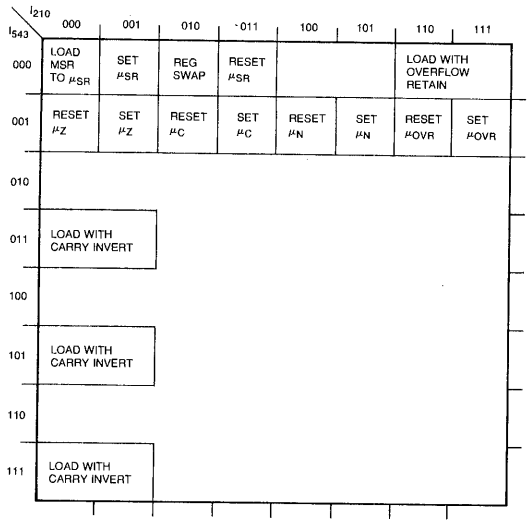
I_{543210} Octal	μSR Operation	Comments
00	M_X → μ_X	LOAD MSR TO μSR
01	1 → μ_X	SET μSR
02	M_X → μ_X	REGISTER SWAP
03	0 → μ_X	RESET μSR

Load Operations

I_{543210} Octal	μSR Operation	Comments
06, 07	I_Z → μ_Z I_C → μ_C I_N → μ_N I_{OVR} + μ_{OVR} → μ_{OVR}	LOAD WITH OVERFLOW RETAIN
30, 31 50, 51 70, 71	I_Z → μ_Z $\overline{I_C}$ → μ_C I_N → μ_N I_{OVR} → μ_{OVR}	LOAD WITH CARRY INVERT
04, 05 20-27 32-47 52-67 72-77	I_Z → μ_Z I_C → μ_C I_N → μ_N I_{OVR} → μ_{OVR}	LOAD DIRECTLY FROM I_Z, I_C, I_N, I_{OVR}

Note: The above tables assume $\overline{CE}$ μ is LOW.

MAP 1. MICRO STATUS REGISTER INSTRUCTION CODES.



Notes: 1. All unmarked locations are a load direct from I_Z, I_C, I_N, I_{OVR}.

that $\overline{CE}_\mu$ be LOW to operate.

Instruction Codes 10_8 to 17_8 are BIT operations. These operations set or reset the individual bits in the μSR.

Instruction Codes 00_8 to 03_8 are REGISTER operations. These operations affect all bits in the μSR.

00_8 This instruction loads the μSR with the contents of the MSR while loading the MSR from the Y inputs and is further explained under "INTERRUPTS".

01_8 This instruction SETS all μSR bits.

02_8 This instruction SWAPS the contents of the μSR and the MSR. It will also COPY one register to the other if the register to be copied is not enabled.

03_8 This instruction RESETS all μSR bits.

All instruction codes except those mentioned in the above two sections cause a LOAD operation from the I_Z, I_C, I_N, I_{OVR} inputs.

06_8, 07_8 When a series of arithmetic operations are being executed sometimes it is not necessary to test for an overflow condition after *each* operation, but rather it is sufficient simply to know that an overflow occured during any one of the operations. Use of these instructions captures the overflow condition by loading the μSR overflow bit with the LOGICAL OR of its present state and I_{OVR}. Thus, once an overflow occurs, μOVR will remain set throughout the remaining operations.

30_8, 31_8, 50_8, 51_8, 70_8, 71_8 These instructions cause a load from the I inputs, but invert the carry bit. The reason for this is explained more fully under the "BORROW SAVE" section.

All others The remaining instructions load the μSR directly from the I_Z, I_C, I_N, I_{OVR} inputs.

Machine Status Register (MSR)

The instruction codes for the MSR fall into two groups; REGISTER Operations and LOAD Operations. All operations require that CE_M be LOW to operate (See Table 2 and Map 2).

BIT operations are accomplished by the use of Register or Load Operations with the $\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$ inputs selectively set LOW.

Instruction codes 00_8-03_8 and 05_8 are REGISTER operations. They affect only those bits enabled by $\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$.

00_8 This instruction loads the MSR from the Y inputs while transferring the present contents to the μSR. The use of this instruction is further explained under "INTERRUPTS".

01_8 This instruction SETS all enabled MSR bits.

02_8 This instruction SWAPS the contents of the μSR and the MSR. It will also COPY one register to the other if the register to be copied is not enabled.

03_8 This instruction RESETS all enabled MSR bits.

05_8 This instruction COMPLEMENTS all enabled MSR bits.

All instruction codes except those mentioned in the above section cause a LOAD operation from the I_Z, I_C, I_N, I_{OVR} inputs.

04_8 The Am2904 Shift Linkage Multiplexer allows for shifts and rotates through the MSR CARRY bit. Some machines require a shift or rotate through the OVERFLOW bit. By using this code, which swaps the contents of the MSR CARRY bit (M_C) and OVERFLOW bit (M_{OVR}), the shift or rotate can be made to appear to take place through the OVERFLOW bit. The procedure is to swap the bits, shift or rotate (any number or positions) then swap the bits again.

TABLE 2. MACHINE STATUS REGISTER INSTRUCTION CODES.

Register Operations

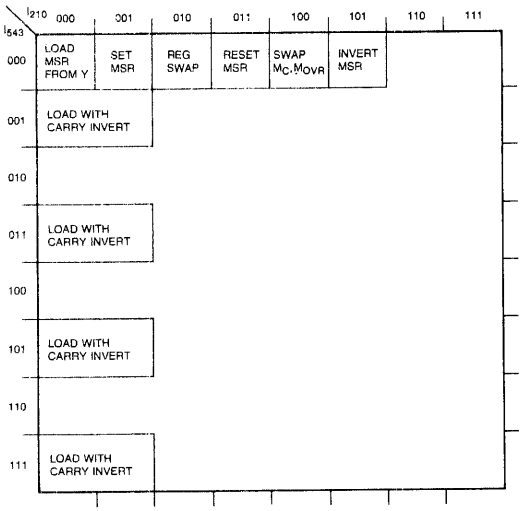
I_{543210} Octal	MSR Operation	Comments
00	$Y_X \rightarrow M_X$	LOAD Y_Z, Y_C, Y_N, Y_{OVR} TO MSR
01	$1 \rightarrow M_X$	SET MSR
02	$\mu_X \rightarrow M_X$	REGISTER SWAP
03	$0 \rightarrow M_X$	RESET MSR
05	$\overline{M_X} \rightarrow M_X$	INVERT MSR

Load Operations

I_{543210} Octal	MSR Operation	Comments
04	$I_Z \rightarrow M_Z$ $M_{OVR} \rightarrow M_C$ $I_N \rightarrow M_N$ $M_C \rightarrow M_{OVR}$	LOAD FOR SHIFT THROUGH OVERFLOW OPERATION
10, 11 30, 31 50, 51 70, 71	$I_Z \rightarrow M_Z$ $\overline{I_C} \rightarrow M_C$ $I_N \rightarrow M_N$ $I_{OVR} \rightarrow M_{OVR}$	LOAD WITH CARRY INVERT
06, 07 12-17 20-27 32-37 40-47 52-67 72-77	$I_Z \rightarrow M_Z$ $I_C \rightarrow M_C$ $I_N \rightarrow M_N$ $I_{OVR} \rightarrow M_{OVR}$	LOAD DIRECTLY FROM I_Z, I_C I_N, I_{OVR}

Notes: 1. The above tables assume $\overline{CE_M}$, $\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$ are LOW.
2. A shift-through-carry instruction loads M_C irrespective of I_5-I_0.

MAP 2. MACHINE STATUS REGISTER INSTRUCTION CODES.



Note 1. All unmarked locations are a load direct from I_Z, I_C, I_{OVR}, I_N.

06_8, 07_8
12_8-27_8
32_8-47_8
52_8-67_8
72_8-77_8 These instructions load the MSR directly from the I_Z, I_C, I_N, I_{OVR} inputs.

10_8, 11_8
30_8, 31_8
50_8, 51_8
70_8, 71_8 These instructions cause a load from the I inputs but invert the CARRY bit. The reason for this is explained more fully under the "BORROW SAVE" section

Condition Code Multiplexer

The four instruction lines I_3, I_2, I_1, I_0 will select one of 16 possible operations to be carried out on the input bits, the result being routed to the Conditional Test Output (CT). Eight of the operations supply an individual status bit or its complement to the CT output. Another four do more complex operations while the remaining four are the complemented results of these (See Table 4).

TABLE 3. Y OUTPUT INSTRUCTION CODES.

$\overline{OE_Y}$	I_5	I_4	Y Output	Comment
1	X	X	Z	Output Off High Impedance
O	O	X	$\mu_i \rightarrow Y_i$	See Note 1
O	1	O	$M_i \rightarrow Y_i$	
O	1	1	$I_i \rightarrow Y_i$	

Notes: 1. For the conditions:
I_5, I_4, I_3, I_2, I_1, I_0 are LOW, Y is an input.
$\overline{OE_Y}$ is "Don't Care" for this condition.
2. X is "Don't Care" condition.

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

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

Notes: 1. $\oplus$ Represents EXCLUSIVE-OR
$\odot$ Represents EXCLUSIVE-NOR or coincidence.

TABLE 5. CRITERIA FOR COMPARING TWO NUMBERS FOLLOWING "A MINUS B" OPERATION.

	For Unsigned Numbers			For 2's Complement Numbers		
		I_{3-0}			I_{3-0}	
Relation	Status	CT = H	CT = L	Status	CT = H	CT = L
A = B	Z = 1	4	5	Z = 1	4	5
A ≠ B	Z = 0	5	4	Z = 0	5	4
A ≥ B	C = 1	A	B	N ⊙ OVR = 1	3	2
A < B	C = 0	B	A	N ⊕ OVR = 1	2	3
A > B	$C \cdot \overline{Z} = 1$	D	C	$(N \odot OVR) \cdot \overline{Z} = 1$	1	0
A ≤ B	$\overline{C} + Z = 1$	C	D	(N ⊕ OVR) + Z = 1	0	1

⊕ = Exclusive OR H = HIGH
⊙ = Exclusive NOR L = LOW

Note: For Am2910, the CC input is active LOW, so use I_{3-0} code to produce CT = L for the desired test.

The more complex operations are intended to follow the calculation A-B to give an indication of which is the larger (A, B unsigned) or more positive (A, B in 2's complement form). See Table 5.

The two instruction lines I_4, I_5 select whether the μSR, the MSR or the direct inputs I_Z, I_C, I_N, I_{OVR} are used as the inputs to the Y output buffer and the CT output (see Tables 3 and 4).

Instruction codes 16_8 and 17_8 form the EXCLUSIVE – OR and the EXCLUSIVE – NOR functions of M_N and I_N. The use of these instructions is explained under "NORMALIZING"

Shift Linkage Multiplexer

The five instruction lines I_{10}, I_9, I_8, I_7, I_6 control the SHIFT LINKAGE multiplexer. All instructions set up the linkages for both the ALU shifter (RAM shifter on the Am2901A) and the Q register.

UP and DOWN shifts are decided by I_{10} which should be connected to I_8 of the Am2903's instruction lines or I_7 of the Am2901's instruction lines. A wide range of input and output connections are provided, allowing for single or double length shifting or rotating with or without the use of the MSR CARRY or SIGN bits (See Table 7).

In the following discussion of some of the shifts the instruction codes are given as two octal digits AB; A represents I_{10}, I_9, B represents I_8, I_7, I_6.

When adding and down shifting on the same microcycle, (i.e. when doing multiplication or averaging) the shifter input must be the present CARRY, I_C, rather than the carry resulting from the last cycle (M_C). Instruction Code 13_8 accomplishes this for unsigned arithmetic. For 2's complement arithmetic, the required shifter input is: $I_N \oplus I_{OVR}$. This is provided by Instruction Code 16_8.

Instruction Codes 14_8, 15_8, 17_8 provide the RIGHT ROTATE THROUGH CARRY, ROTATE BRANCH CARRY and ROTATE WITHOUT CARRY functions respectively.

Instruction Codes 34_8, 35_8, 37_8 provide the LEFT ROTATE THROUGH CARRY, ROTATE BRANCH CARRY and ROTATE WITHOUT CARRY functions respectively.

The shift outputs are in the high impedance state unless $\overline{SE}$ is LOW.

Loading of the M_C bit by a shift operation overrides any loading or holding of the M_C bit by MSR Instructions (I_{0-5}, $\overline{CE_M}$ and $\overline{E_C}$).

"CARRY-IN" Control Multiplexer

The two instruction lines I_{12}, I_{11} control the source of the CARRY output (C_0).

When $I_{12} = 0$ $C_0 = I_{11}$

When $I_{12} = 1$ and $I_{11} = 0$, the external carry input C_X is presented to the carry output.

When $I_{12} = I_{11} = 1$ the carry output is selected from μ_C, $\overline{\mu_C}$, M_C or $\overline{M_C}$ as defined by I_5, I_3, I_2, I_1 (See Table 6).

TABLE 6. CARRY-IN CONTROL MULTIPLEXER INSTRUCTION CODES.

I_{12}	I_{11}	I_5	I_3	I_2	I_1	C_0
0	0	X	X	X	X	0
0	1	X	X	X	X	1
1	0	X	X	X	X	C_X
1	1	0	0	X	X	μ_C
1	1	0	X	1	X	μ_C
1	1	0	X	X	1	μ_C
1	1	0	1	0	0	$\overline{\mu_C}$
1	1	1	0	X	X	M_C
1	1	1	X	1	X	M_C
1	1	1	X	X	1	M_C
1	1	1	1	0	0	$\overline{M_C}$

APPLICATIONS INFORMATION

Borrow – Save

One of the capabilities of the Am2900 Family is the complete emulation of other processing machines. One requirement of an emulator is that, when a calculation is being performed, not only must the answer obtained from the Am2900 chips be the same as that from the machine being emulated, but after each machine level instruction, the status bits must be indentical.

There are alternative methods for subtracting in a digital machine and the state of the CARRY after the calculation depends on the method. For instance, the subtraction of 0100 from 1010 by the 2's complement add method generates a result of 0110 with a CARRY. Direct subtraction however, yields an answer of 0110 with no BORROW.

Many machines store the state of the CARRY for subtract operations, and this is the recommended method for maximum effective use of the Am2904, but, to allow those machines which store the BORROW to be efficiently emulated, the Am2904 has allocated special instructions. Using these codes causes the CARRY bit to be inverted before storage in the status registers and also re-inverts these status bits before using them as carry inputs. These codes are 10_8, 11_8, 30_8, 31_8, 50_8, 51_8, 70_8, 71_8 (I_{5-0}).

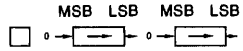
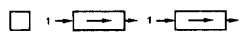
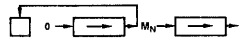
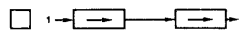
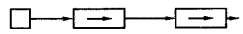
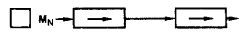
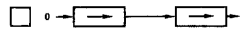
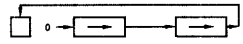
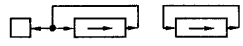
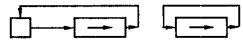
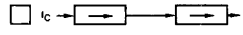
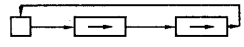
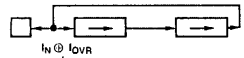
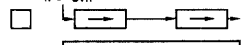
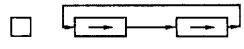
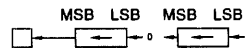
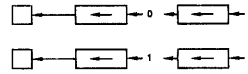
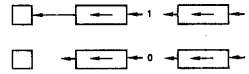
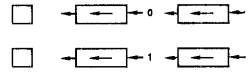
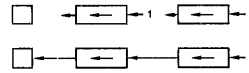
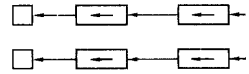
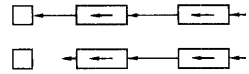
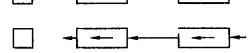
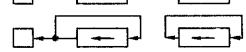
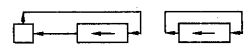
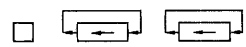
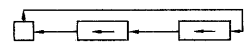
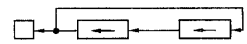
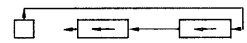
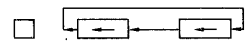
Notice that when these codes are used to load the inverted CARRY to either of the status registers, the CT output selected by the Condition Code Multiplexer assumes the CARRY is inverted and still defines whether A > B or A ≤ B (See Table 4).

Similarly, when doing a compare on a machine which saves the borrow, testing for A > B, A≤ B forces the complement of the CARRY to be stored in the status registers (See Tables 1 and 2).

Normalizing

Normalizing is the process of stripping off all leading sign bits until the two most significant bits are complementary. The Am2904 facilitates both single and double length normalization in the Am2901 and the Am2903. When using the NORMALIZE special instructions with the Am2903, the EXCLUSIVE – OR of the most significant two bits is generated at the C_{n+4} pin of the most significant Am2903. The EXCLUSIVE – OR of the two bits next to the most significant bit is also generated at the OVR pin. The procedure for normalizing then is to loop on the normalize instruction with a branch condition on the C_{n+4} state or the OVR state, depending on the architecture employed. The C_{n+4} or OVR output is routed to the Am2910 CC input through the Am2904 Condition Code multiplexer. As the contents of the status registers always refers to the last cycle, not the present one, the last operation in Normalizing is to downshift, bringing the sign bit (M_N) back into the most significant bit position. This is achieved using the shift operations 05_8 (I_{10-6}) for double length normalizing,

TABLE 7. SHIFT LINKAGE MULTIPLEXER INSTRUCTION CODES.

I_{10}	I_9	I_8	I_7	I_6	M_C RAM Q	SIO_0	SIO_n	QIO_0	QIO_n	Loaded Into M_C
					MSB LSB MSB LSB					
0	0	0	0	0		Z	0	Z	0	
0	0	0	0	1		Z	1	Z	1	
0	0	0	1	0		Z	0	Z	M_N	SIO_0
0	0	0	1	1		Z	1	Z	SIO_0	
0	0	1	0	0		Z	M_C	Z	SIO_0	
0	0	1	0	1		Z	M_N	Z	SIO_0	
0	0	1	1	0		Z	0	Z	SIO_0	
0	0	1	1	1		Z	0	Z	SIO_0	QIO_0
0	1	0	0	0		Z	SIO_0	Z	QIO_0	SIO_0
0	1	0	0	1		Z	M_C	Z	QIO_0	SIO_0
0	1	0	1	0		Z	SIO_0	Z	QIO_0	
0	1	0	1	1		Z	I_C	Z	SIO_0	
0	1	1	0	0		Z	M_C	Z	SIO_0	QIO_0
0	1	1	0	1		Z	QIO_0	Z	SIO_0	QIO_0
0	1	1	1	0		Z	$I_N \oplus I_{OVR}$	Z	SIO_0	
0	1	1	1	1		Z	QIO_0	Z	SIO_0	
					MSB LSB MSB LSB					
1	0	0	0	0		0	Z	0	Z	SIO_n
1	0	0	0	1		1	Z	1	Z	SIO_n
1	0	0	1	0		0	Z	0	Z	
1	0	0	1	1		1	Z	1	Z	
1	0	1	0	0		QIO_n	Z	0	Z	SIO_n
1	0	1	0	1		QIO_n	Z	1	Z	SIO_n
1	0	1	1	0		QIO_n	Z	0	Z	
1	0	1	1	1		QIO_n	Z	1	Z	
1	1	0	0	0		SIO_n	Z	QIO_n	Z	SIO_n
1	1	0	0	1		M_C	Z	QIO_n	Z	SIO_n
1	1	0	1	0		SIO_n	Z	QIO_n	Z	
1	1	0	1	1		M_C	Z	0	Z	
1	1	1	0	0		QIO_n	Z	M_C	Z	SIO_n
1	1	1	0	1		QIO_n	Z	SIO_n	Z	SIO_n
1	1	1	1	0		QIO_n	Z	M_C	Z	
1	1	1	1	1		QIO_n	Z	SIO_n	Z	

Notes: 1. Z = High impedance (outputs off) state.
2. Outputs enabled and M_C loaded only if $\overline{SE}$ is LOW.
3. Loading of M_C from I_{10-6} overrides control from I_{5-0}, $\overline{CE}_M$, $\overline{E}_C$.

and 02_8 for single length normalizing. For more details regarding normalizing with the Am2903 see the Am2903 data sheet.

The Am2901 does not have the EXCLUSIVE – OR gates to help with normalizing, so the Am2904 includes in the Condition Code multiplexer the EXCLUSIVE – OR and EXCLUSIVE – NOR functions of M_N (the sign bit resulting from the last operation) and I_N (the sign bit resulting from the present operation).

Interrupts

Some machines allow interrupts only at the machine instruction level while others allow them at the microinstruction level. The Am2904 is designed to handle both cases.

When the machine is interrupted, it is necessary to store the contents of either the MSR (machine instruction level interrupts) or both the status registers (micro instruction level interrupts) into an external store. This transfer is intended to take place over the Y input/output pins (See Table 3).

After the interrupt has been serviced the registers must be restored to their pre-interrupt state. This is accomplished by two operations of instruction 00_8 (I_{5-0}) which loads the MSR from the Y inputs while loading the μSR from the MSR. Thus, the pre-interrupt contents of the μSR are first loaded to the MSR (first instruction 00_8), then this data is transferred to the μSR while the MSR is restored to its pre-interrupt state (second instruction 00_8).

In controllers and some other microprogrammed machines the applications program itself is often in the microprogram memory; that is, there is no macroinstruction set. These machines require only a microstatus register since there is no separate machine status. The MSR in the Am2904 can be used as a one-level stack on the microstatus register. When an interrupt occurs, the μSR and the MSR are simply swapped ($I_{5-0} = 02_8$).

PIN DEFINITIONS

I_Z Zero status input pin, intended for connection to the Z outputs of the Am2903 or the F = 0 ouputs of the Am2901.

I_C Carry status input pin, intended for connection to the C_{n+4} output of the most significant ALU slice.

I_N Sign status input pin, intended for connection to the most significant ALU slice. The connection is to the N pin on the Am2903, and the F_3 pin on the Am2901.

I_{OVR} Overflow status input pin, intended for connection to the OVR pin on the most significant ALU slice.

I_{0-12} The thirteen instruction pins which select the operation the Am2904 is to perform.

$\overline{CE_M}$ This pin, used in conjunction with $\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$ acts as the overall enable for the machine status register. When the pin is LOW, MSR bits may be modified, according to the states of $\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$. When HIGH, the MSR will retain the present state, regardless of the state of $\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$.

$\overline{E_Z}$, $\overline{E_C}$, $\overline{E_N}$, $\overline{E_{OVR}}$ These pins, when LOW, enable the corresponding bits in the Machine Status Register. When HIGH, they will prevent the corresponding bits from changing state. By using these pins together with the $\overline{CE_M}$ pin, MSR bits can be selectively modified.

$\overline{CE_\mu}$ This pin, when LOW, enables all four bits of the Micro Status Register. When this pin is HIGH, the μSR will not change state.

Y_Z, Y_C, Y_N, Y_{OVR} These pins form a three-state bidirectional bus over which MSR and μSR status can be read out or the MSR can be loaded in parallel.

$\overline{OE_Y}$ When LOW, this pin enables the Y pins as outputs. When HIGH, the Y outputs are in the high impedance state.

CT The conditional test output. The output of the Condition Code multiplexer appears here.

$\overline{OE_{CT}}$ When this pin is LOW, the CT pin is active. When HIGH the CT pin is in the high impedance state.

SIO_0, SIO_n QIO_0 QIO_n These pins complete the linking for the various shift and rotate conditions. SIO_0 is intended for connection to the SIO_0 pin of the least significant Am2903 slice (RAM_0 for Am2901). SIO_n connects to the SIO_3 pin of the most significant Am2903 slice, (RAM_3 for Am2901). QIO_0 connects to the QIO_0 pin of the least significant Am2903 slice (QIO_0 for Am2901) and QIO_n connects to the QIO_3 pin of the most significant Am2903 slice (Q_3 for Am2901).

$\overline{SE}$ This pin controls the state of the shift outputs. When LOW, the shift outputs are enabled. When HIGH, the shift outputs are in the high impedance state.

C_0 This pin is the output of the Carry In control multiplexer. It connects to the C_n input of the least significant ALU slice, and the C_n input of the Am2902A.

C_X This pin is used as an input to the Carry In Control multiplexer which can route it to the C_0 pin. The C_X pin is intended for connection to the Z output of the Am2903 to facilitate some of the Am2903 special instructions.

CP The clock input to the device. The μSR and MSR are modified on the LOW to HIGH transition of the clock input. All other portions of the Am2904 are combinational and are unaffected by CP.

ORDERING INFORMATION

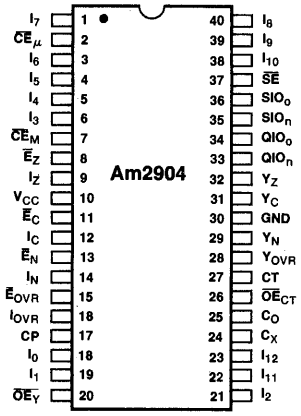
Order the part number according to the table below to obtain the desired package, temperature range, and screening level.

Order Number	Package Type (Note 1)	Operating Range (Note 2)	Screening Level (Note 3)
AM2904PC	P-40	C	C-1
AM2904DC	D-40	C	C-1
AM2904DC-B	D-40	C	B-2 (Note 4)
AM2904DM	D-40	M	C-3
AM2904DM-B	D-40	M	B-3
AM2904FM	F-42	M	C-3
AM2904FM-B	F-42	M	B-3
AM2904XC	Dice	C	Visual inspection to MIL-STD-883 Method 2010B.
AM2904XM	Dice	M	

Notes:
1. P = Molded DIP, D = Hermetic DIP, F = Flat Pak. Number following letter is number of leads. See Appendix B for detailed outline. Where Appendix B contains several dash numbers, any of the variations of the package may be used unless otherwise specified.
2. C = 0°C to +70°C, V_{CC} = 4.75V to 5.25V.
   M = −55°C to +125°C, V_{CC} = 4.50V to 5.50V.
3. See Appendix A for details of screening. Levels C-1 and C-3 conform to MIL-STD-883, Class C. Level B-3 conforms to MIL-STD-883, Class B.
4. 96 hour burn-in.

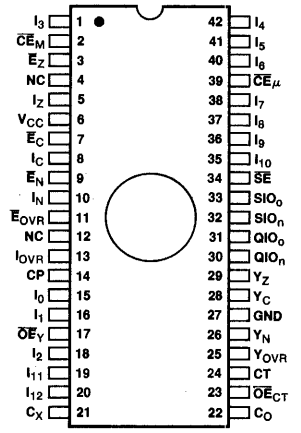
CONNECTION DIAGRAMS
Top Views

DIP



MPR-723

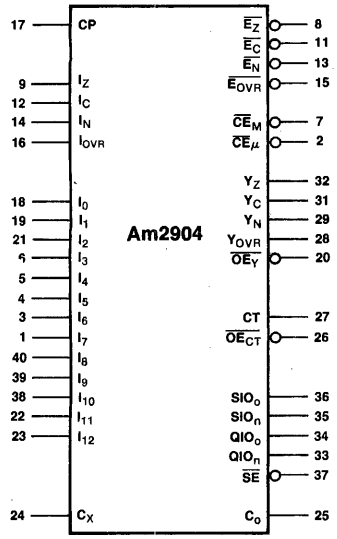
Flat Package



MPR-724

Note: Pin 1 is marked for orientation.
NC = No Connection.

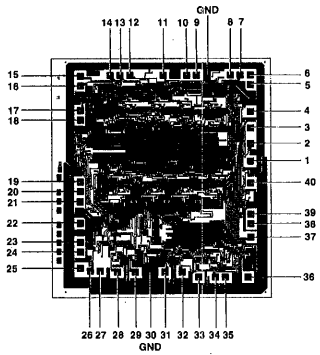
LOGIC SYMBOL
(DIP)



MPR-725

V_{CC} = Pin 10
GND = Pin 30

Am2904 Metallization and Pad Layout



DIE SIZE 0.140" X 0.161"
Pad Numbers correspond to DIP pinout

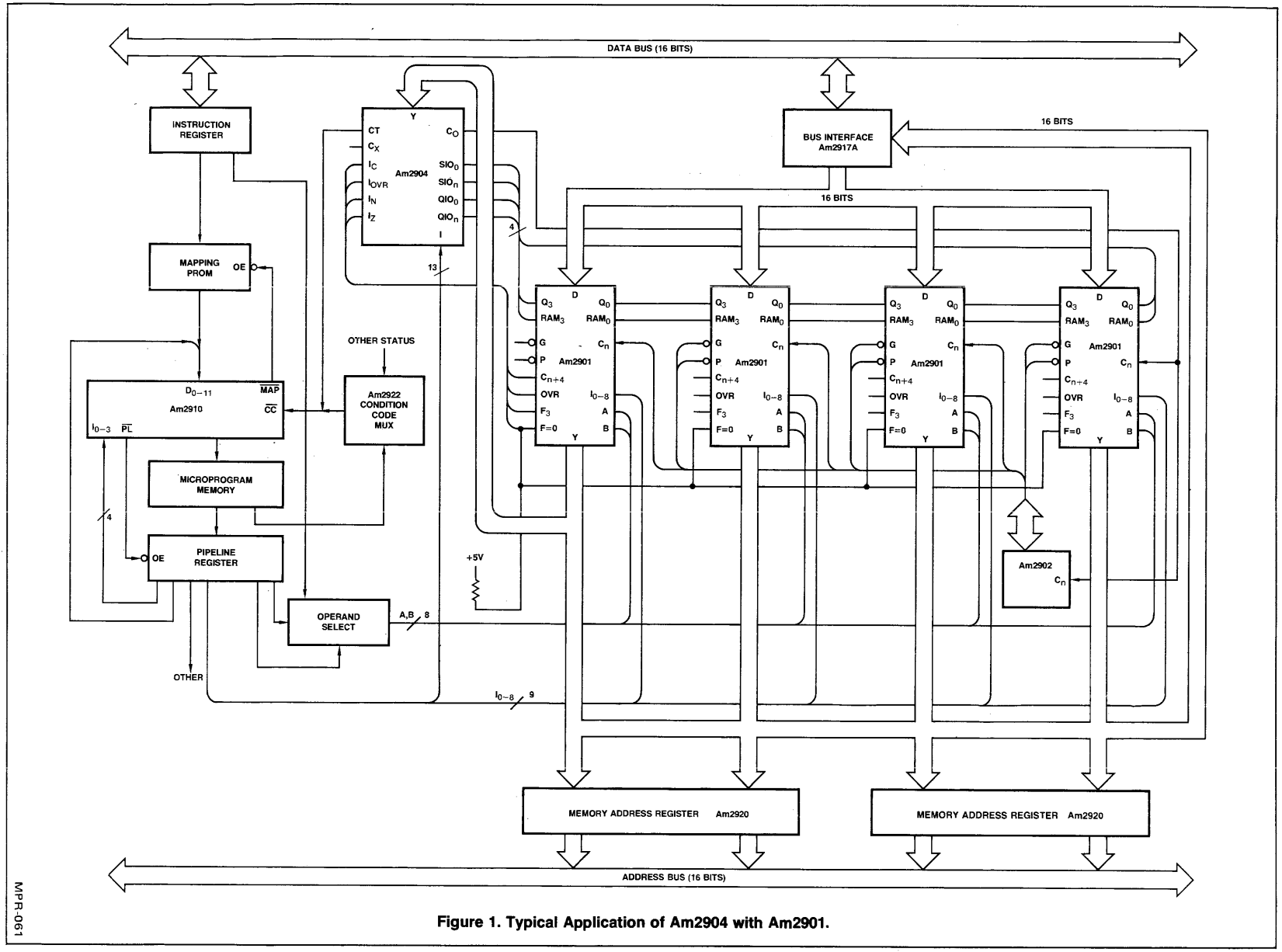


Figure 1. Typical Application of Am2904 with Am2901.

MPR-061

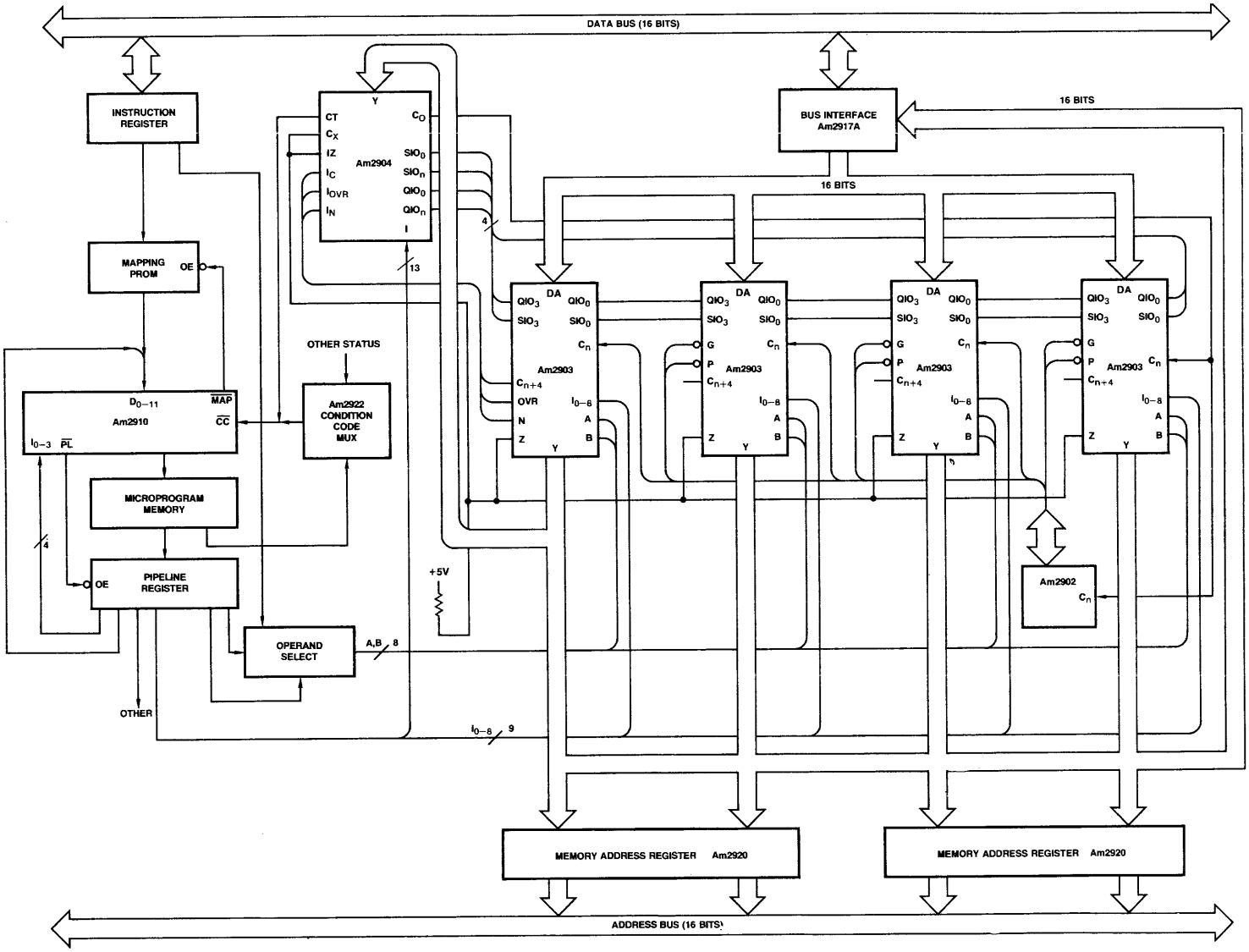


Figure 2. Typical Application of Am2904 with Am2903.

MPR-062

Am2910
Microprogram Controller

DISTINCTIVE CHARACTERISTICS

- Twelve Bits Wide
  Address up to 4096 words of microcode with one chip. All internal elements are a full 12 bits wide.
- Internal Loop Counter
  Pre-settable 12-bit down-counter for repeating instructions and counting loop iterations.
- Four Address Sources
  Microprogram Address may be selected from microprogram counter, branch address bus, 5-level push/pop stack, or internal holding register.
- Sixteen Powerful Microinstructions
  Executes 16 sequence control instructions, most of which are conditional on external condition input, state of internal loop counter, or both.
- Output Enable Controls for Three Branch Address Sources
  Built-in decoder function to enable external devices onto branch address bus. Eliminates external decoder.
- All Registers Positive Edge-triggered
  Simplifies timing problems. Eliminates long set-up times.
- Fast Control from Condition Input
  Delay from condition code input to address output only 21ns typical.

GENERAL DESCRIPTION

The Am2910 Microprogram controller is an address sequencer intended for controlling the sequence of execution of microinstructions stored in microprogram memory. Besides the capability of sequential access, it provides conditional branching to any microinstruction within its 4096-microword range. A last-in, first-out stack provides microsubroutine return linkage and looping capability; there are five levels of nesting of microsubroutines. Microinstruction loop count control is provided with a count capacity of 4096.

During each microinstruction, the Microprogram controller provides a 12-bit address from one of four sources: 1) the microprogram address register (μPC), which usually contains an address one greater than the previous address; 2) an external (direct) input (D); 3) a register/counter (R) retaining data loaded during a previous microinstruction; or 4) a five-deep last-in, first-out stack (F).

For a detailed discussion of this architectural approach to microprogram control units, refer to "The Microprogramming Handbook", an AMD applications publication.

Am2910 BLOCK DIAGRAM

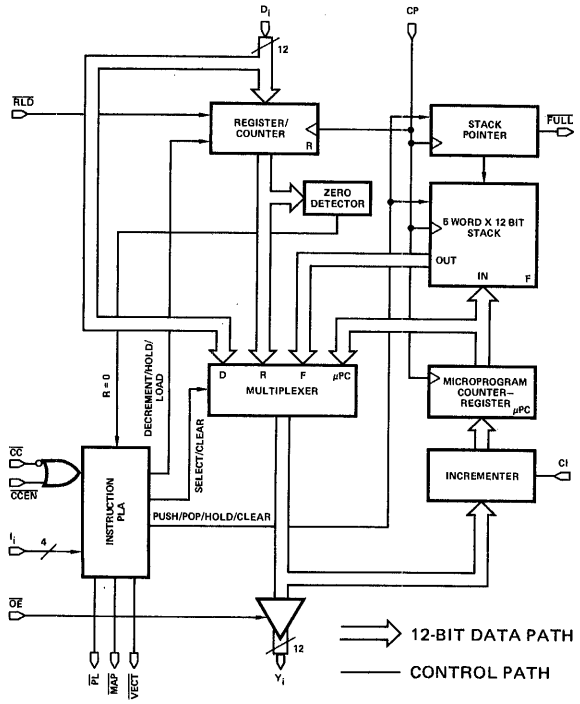


Figure 1.

MPR-106

TABLE OF CONTENTS

For applications information, see Chapter II of "Build a Microcomputer".

ARCHITECTURE OF THE Am2910

The Am2910 is a bipolar microprogram controller intended for use in high-speed microprocessor applications. It allows addressing of up to 4K words of microprogram. A block diagram is shown in Figure 1.

The controller contains a four-input multiplexer that is used to select either the register/counter, direct input, microprogram counter, or stack as the source of the next microinstruction address.

The register/counter consists of 12 D-type, edge-triggered flip-flops, with a common clock enable. When its load control, $\overline{RLD}$, is LOW, new data is loaded on a positive clock transition. A few instructions include load; in most systems, these instructions will be sufficient, simplifying the microcode. The output of the register/counter is available to the multiplexer as a source for the next microinstruction address. The direct input furnishes a source of data for loading the register/counter.

The Am2910 contains a microprogram counter (μPC) that is composed of a 12-bit incrementer followed by a 12-bit register. The μPC can be used in either of two ways: When the carry-in to the incrementer is HIGH, the microprogram register is loaded on the next clock cycle with the current Y output word plus one ($Y + 1 \rightarrow \mu PC$). Sequential microinstructions are thus executed. When the carry-in is LOW, the incrementer passes the Y output word unmodified so that μPC is reloaded with the same Y word on the next clock cycle ($Y \rightarrow \mu PC$). The same microinstruction is thus executed any number of times.

The third source for the multiplexer is the direct (D) input. This source is used for branching.

The fourth source available at the multiplexer input is a 5-word by 12-bit stack (file). The stack is used to provide return address linkage when executing microsubroutines or loops. The stack contains a built-in stack pointer (SP) which always points to the last file word written. This allows stack reference operations (looping) to be performed without a pop.

The stack pointer operates as an up/down counter. During microinstructions 1, 4, and 5, the PUSH operation may occur. This causes the stack pointer to increment and the file to be written with the required return linkage. On the cycle following the PUSH, the return data is at the new location pointed to by the stack pointer.

During five microinstructions, a POP operation may occur. The stack pointer decrements at the next rising clock edge following a POP, effectively removing old information from the top of the stack.

The stack pointer linkage is such that any sequence of pushes, pops, or stack references can be achieved. At RESET (Instruction 0), the depth of nesting becomes zero. For each PUSH, the nesting depth increases by one; for each POP, the depth decreases by one. The depth can grow to five. After a depth of five is reached, $\overline{FULL}$ goes LOW. Any further PUSHes onto a full stack overwrite information at the top of the stack, but leave the stack pointer unchanged. This operation will usually destroy useful information and is normally avoided. A POP from an empty stack may place non-meaningful data on the Y outputs, but is otherwise safe. The stack pointer remains at zero whenever a POP is attempted from a stack already empty.

The register/counter is operated during three microinstructions (8, 9, 15) as a 12-bit down counter, with result = zero available as a microinstruction branch test criterion. This provides efficient iteration of microinstructions. The register/counter is arranged such that if it is preloaded with a number N and then used as a loop termination counter, the sequence will be executed exactly N+1 times. During instruction 15, a three-way branch under combined control of the loop counter and the condition code is available.

The device provides three-state Y outputs. These can be particularly useful in designs requiring automatic checkout of the processor. The microprogram controller outputs can be forced into the high-impedance state, and pre-programmed sequences of microinstructions can be executed via external access to the address lines.

OPERATION

Table I shows the result of each instruction in controlling the multiplexer which determines the Y outputs, and in controlling the three enable signals $\overline{PL}$, $\overline{MAP}$, and $\overline{VECT}$. The effect on the register/counter and the stack after the next positive-going clock edge is also shown. The multiplexer determines which internal source drives the Y outputs. The value loaded into μPC is either identical to the Y output, or else one greater, as determined by CI. For each instruction, one and only one of the three outputs $\overline{PL}$, $\overline{MAP}$, and $\overline{VECT}$ is LOW. If these outputs control three-state enables for the primary source of microprogram jumps (usually part of a pipeline register), a PROM which maps the instruction to a microinstruction starting location, and an optional third source (often a vector from a DMA or interrupt source), respectively, the three-state sources can drive the D inputs without further logic

Several inputs, as shown in Table II, can modify instruction execution. The combination $\overline{CC}$ HIGH and $\overline{CCEN}$ LOW is used as a test in 9 of the 16 instructions. $\overline{RLD}$, when LOW, causes the D input to be loaded into the register/counter, overriding any HOLD or DEC operation specified in the instruction. $\overline{OE}$, normally LOW, may be forced HIGH to remove the Am2910 Y outputs from a three-state bus.

The stack, a five-word last-in, first-out 12-bit memory, has a pointer which addresses the value presently on the top of the stack. Explicit control of the stack pointer occurs during instruction 0 (RESET), which makes the stack empty by resetting the SP to zero. After a RESET, and whenever else the stack is empty, the contents of the top of stack is undefined until a PUSH occurs. Any POPs performed while the stack is empty put undefined data on the F outputs and leave the stack pointer at zero.

Any time the stack is full (five more PUSHes than POPs have occurred since the stack was last empty), the $\overline{FULL}$ warning output occurs. This signal first appears on the microcycle after a fifth PUSH. No additional PUSH should be attempted onto a full stack; if tried, information within the stack will be overwritten and lost.

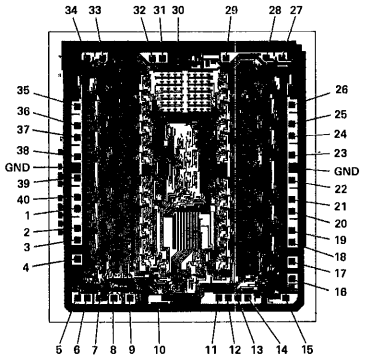
ORDERING INFORMATION

Order the part number according to the table below to obtain the desired package, temperature range, and screening level.

Order Number	Package Type (Note 1)	Operating Range (Note 2)	Screening Level (Note 3)
AM2910PC	P-40	C	C-1
AM2910DC	D-40	C	C-1
AM2910DC-B	D-40	C	B-2 (Note 4)
AM2910DM	D-40	M	C-3
AM2910DM-B	D-40	M	B-3
AM2910FM	F-42	M	C-3
AM2910FM-B	F-42	M	B-3
AM2910XC	Dice	C	Visual inspection to MIL-STD-883 Method 2010B.
AM2910XM	Dice	M	

Notes: 1. P = Molded DIP, D = Hermetic DIP, F = Flat Pak. Number following letter is number of leads. See Appendix B for detailed outline. Where Appendix B contains several dash numbers, any of the variations of the package may be used unless otherwise specified.
2. C = 0°C to +70°C, V_{CC} = 4.75V to 5.25V, M = − 55°C to +125°C, V_{CC} = 4.50V to 5.50V.
3. See Appendix A for details of screening. Levels C-1 and C-3 conform to MIL-STD-883, Class C. Level B-3 conforms to MIL-STD-883, Class B.
4. 96 hour burn-in.

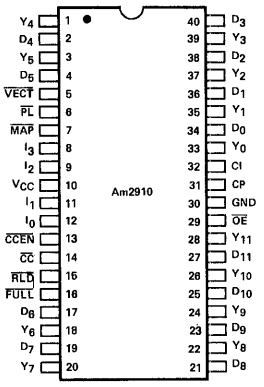
Metallization and Pad Layout



Die Size 0.170" x 0.194"
(Note: Numbers refer to DIP connections)

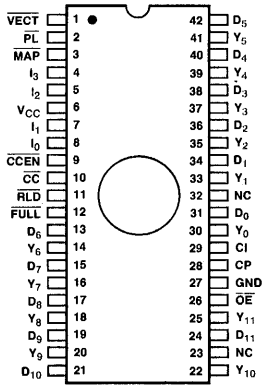
CONNECTION DIAGRAMS – Top Views

DIP



MPR-107

Flat Package



MPR-108

Pin 1 is marked for orientation.

TABLE I. INSTRUCTIONS

I_3-I_0	MNEMONIC	NAME	REG/ CNTR CONTENTS	FAIL $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH		PASS $\overline{CCEN}$ = HIGH or $\overline{CC}$ = LOW		REG/ CNTR	ENABLE
				Y	STACK	Y	STACK		
0	JZ	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	PL
1	CJS	COND JSB PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP VECTOR	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
8	RFCT	REPEAT LOOP, CNTR ≠ 0	≠0	F	HOLD	F	HOLD	DEC	PL
			=0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL, CNTR ≠ 0	≠0	D	HOLD	D	HOLD	DEC	PL
			=0	PC	HOLD	PC	HOLD	HOLD	PL
10	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
11	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
12	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
13	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
14	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
15	TWB	THREE-WAY BRANCH	≠0	F	HOLD	PC	POP	DEC	PL
			=0	D	POP	PC	POP	HOLD	PL

Note 1: If $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH, hold; else load. X = Don't Care

TABLE II. PIN FUNCTIONS

Abbreviation	Name	Function
D_i	Direct Input Bit i	Direct input to register/counter and multiplexer. D_0 is LSB
I_i	Instruction Bit i	Selects one-of-sixteen instructions for the Am2910
$\overline{CC}$	Condition Code	Used as test criterion. Pass test is a LOW on $\overline{CC}$.
$\overline{CCEN}$	Condition Code Enable	Whenever the signal is HIGH, $\overline{CC}$ is ignored and the part operates as though $\overline{CC}$ were true (LOW).
CI	Carry-In	Low order carry input to incrementer for microprogram counter
$\overline{RLD}$	Register Load	When LOW forces loading of register/counter regardless of instruction or condition
$\overline{OE}$	Output Enable	Three-state control of Y_i outputs
CP	Clock Pulse	Triggers all internal state changes at LOW-to-HIGH edge
V_{CC}	+5 Volts	
GND	Ground	
Y_i	Microprogram Address Bit i	Address to microprogram memory. Y_0 is LSB, Y_{11} is MSB
$\overline{FULL}$	Full	Indicates that five items are on the stack
$\overline{PL}$	Pipeline Address Enable	Can select #1 source (usually Pipeline Register) as direct input source
$\overline{MAP}$	Map Address Enable	Can select #2 source (usually Mapping PROM or PLA) as direct input source
$\overline{VECT}$	Vector Address Enable	Can select #3 source (for example, Interrupt Starting Address) as direct input source

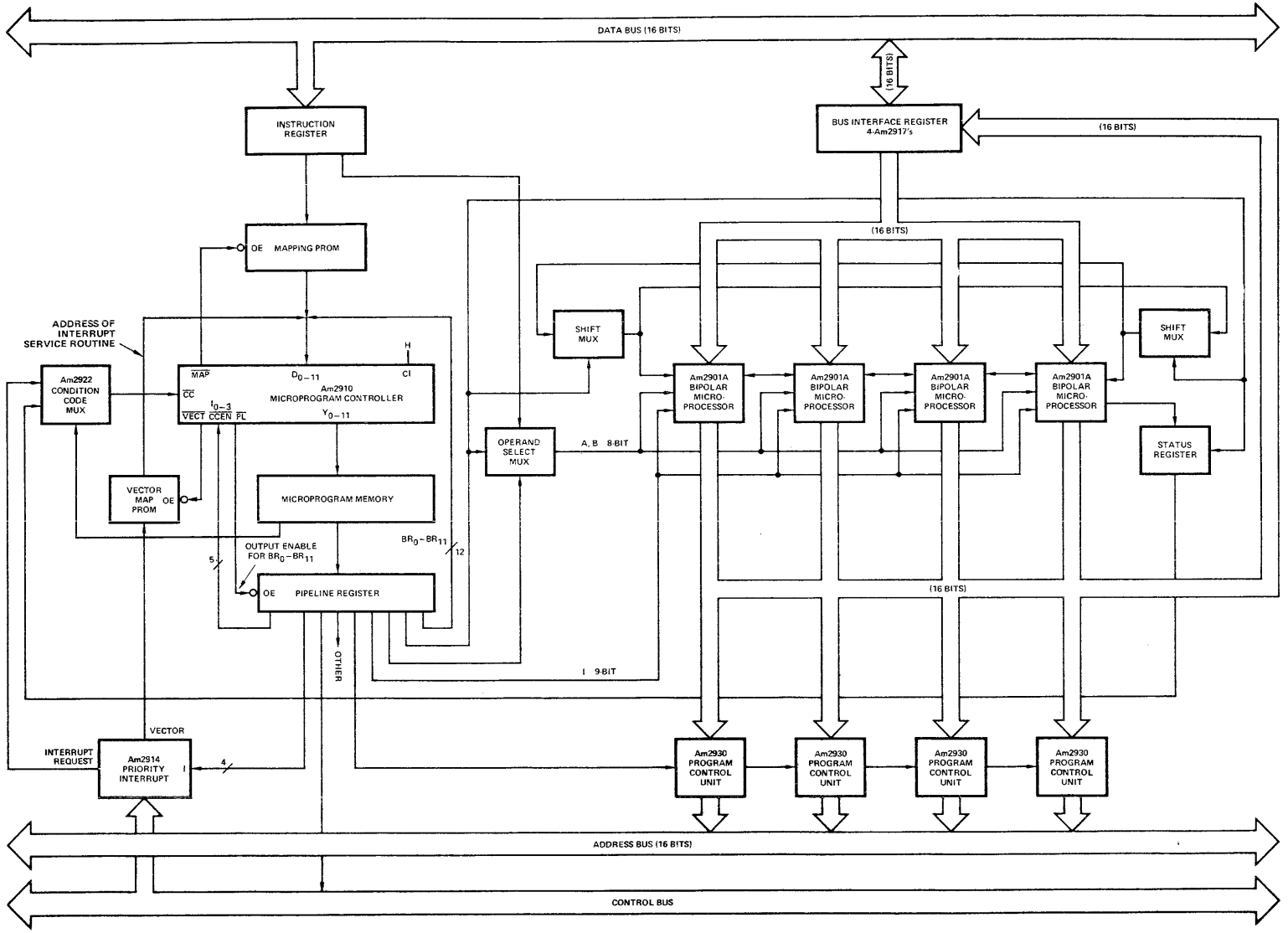
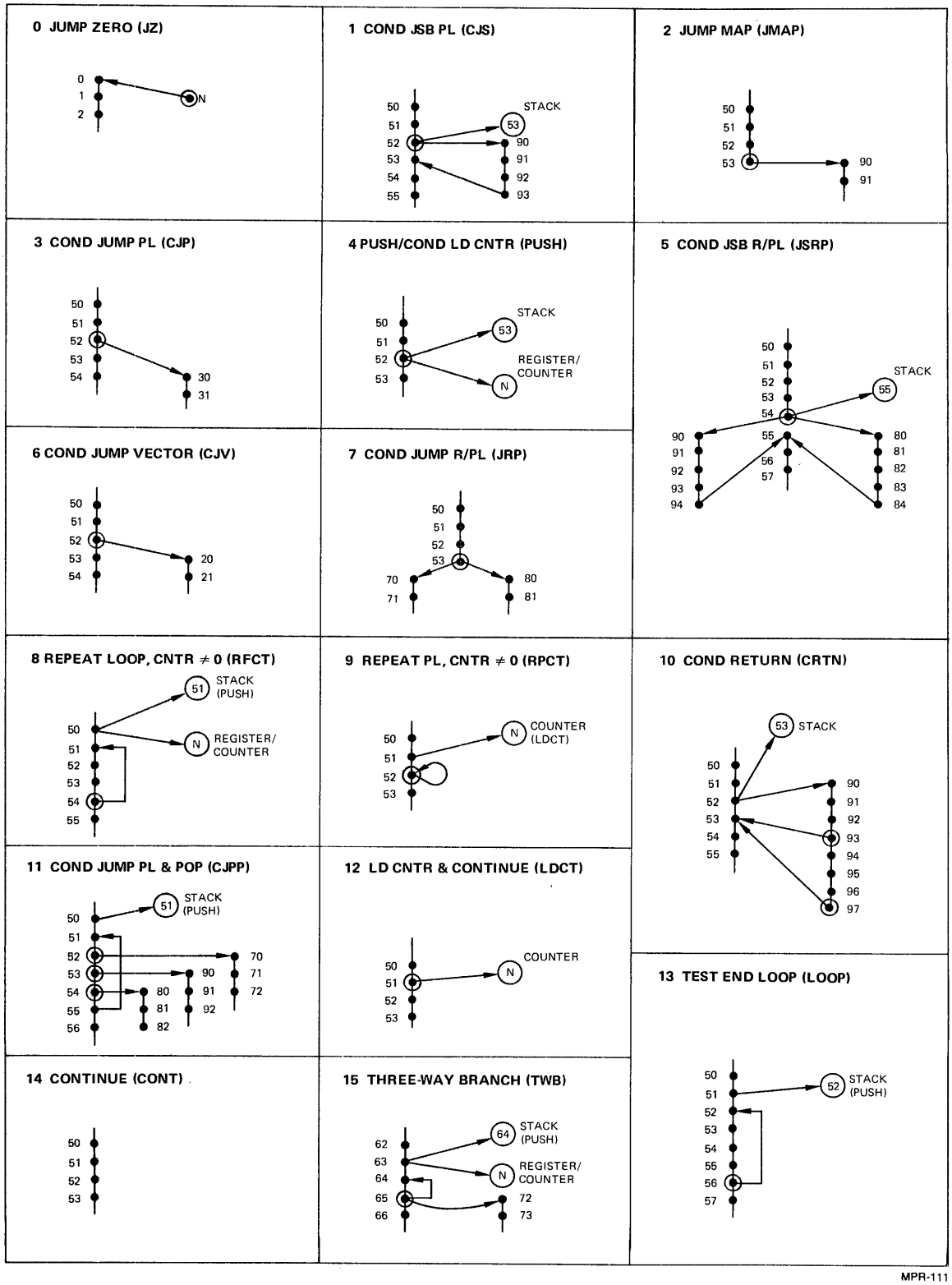


Figure 3. Typical Bipolar Microcomputer Using Am2910.

MPR-110



MPR-111

Figure 4. Am2910 Execution Examples.

THE Am2910 INSTRUCTION SET

The Am2910 provides 16 instructions which select the address of the next microinstruction to be executed. Four of the instructions are unconditional – their effect depends only on the instruction. Ten of the instructions have an effect which is partially controlled by an external, data-dependent condition. Three of the instructions have an effect which is partially controlled by the contents of the internal register/counter. The instruction set is shown in Table I. In this discussion it is assumed that C_n is tied HIGH.

In the ten conditional instructions, the result of the data-dependent test is applied to $\overline{CC}$. If the $\overline{CC}$ input is LOW, the test is considered to have been passed, and the action specified in the name occurs; otherwise, the test has failed and an alternate (often simply the execution of the next sequential microinstruction) occurs. Testing of $\overline{CC}$ may be disabled for a specific microinstruction by setting $\overline{CCEN}$ HIGH, which unconditionally forces the action specified in the name; that is, it forces a pass. Other ways of using $\overline{CCEN}$ include (1) tying it HIGH, which is useful if no microinstruction is data-dependent; (2) tying it LOW if data-dependent instructions are never forced unconditionally; or (3) tying it to the source of Am2910 instruction bit I_0, which leaves instructions 4, 6, and 10 as data-dependent but makes others unconditional. All of these tricks save one bit of microcode width.

The effect of three instructions depends on the contents of the register/counter. Unless the counter holds a value of zero, it is decremented; if it does hold zero, it is held and a different microprogram next address is selected. These instructions are useful for executing a microinstruction loop a known number of times. Instruction 15 is affected both by the external condition code and the internal register/counter.

Perhaps the best technique for understanding the Am2910 is to simply take each instruction and review its operation. In order to provide some feel for the actual execution of these instructions, Figure 4 is included and depicts examples of all 16 instructions.

The examples given in Figure 4 should be interpreted in the following manner: The intent is to show microprogram flow as various microprogram memory words are executed. For example, the CONTINUE instruction, instruction number 14, as shown in Figure 4, simply means that the contents of microprogram memory word 50 is executed, then the contents of word 51 is executed. This is followed by the contents of microprogram memory word 52 and the contents of microprogram memory word 53. The microprogram addresses used in the examples were arbitrarily chosen and have no meaning other than to show instruction flow. The exception to this is the first example, JUMP ZERO, which forces the microprogram location counter to address ZERO. Each dot refers to the time that the contents of the microprogram memory word is in the pipeline register. While no special symbology is used for the conditional instructions, the text to follow will explain what the conditional choices are in each example.

It might be appropriate at this time to mention that AMD has a microprogram assembler called AMDASM, which has the capability of using the Am2910 instructions in symbolic representation. AMDASM's Am2910 instruction symbolics (or mnemonics) are given in Figure 4 for each instruction and are also shown in Table I.

Instruction 0, JZ (JUMP and ZERO, or RESET) unconditionally specifies that the address of the next microinstruction is zero. Many designs use this feature for power-up sequences and provide the power-up firmware beginning at microprogram memory word location 0.

Instruction 1 is a CONDITIONAL JUMP-TO-SUBROUTINE via the address provided in the pipeline register. As shown in Figure 4, the machine might have executed words at address 50, 51, and 52. When the contents of address 52 is in the pipeline register, the next address control function is the CONDITIONAL JUMP-TO-SUBROUTINE. Here, if the test is passed, the next instruction executed will be the contents of microprogram memory location 90. If the test has failed, the JUMP-TO-SUBROUTINE will not be executed; the contents of microprogram memory location 53 will be executed instead. Thus, the CONDITIONAL JUMP-TO-SUBROUTINE instruction at location 52 will cause the instruction either in location 90 or in location 53 to be executed next. If the TEST input is such that location 90 is selected, value 53 will be pushed onto the internal stack. This provides the return linkage for the machine when the subroutine beginning at location 90 is completed. In this example, the subroutine was completed at location 93 and a RETURN-FROM-SUBROUTINE would be found at location 93.

Instruction 2 is the JUMP MAP instruction. This is an unconditional instruction which causes the $\overline{MAP}$ output to be enabled so that the next microinstruction location is determined by the address supplied via the mapping PROMs. Normally, the JUMP MAP instruction is used at the end of the instruction fetch sequence for the machine. In the example of Figure 4, microinstructions at locations 50, 51, 52, and 53 might have been the fetch sequence and at its completion at location 53, the jump map function would be contained in the pipeline register. This example shows the mapping PROM outputs to be 90; therefore, an unconditional jump to microprogram memory address 90 is performed.

Instruction 3, CONDITIONAL JUMP PIPELINE, derives its branch address from the pipeline register branch address value (BR_0 – BR_{11} in Figure 2). This instruction provides a technique for branching to various microprogram sequences depending upon the test condition inputs. Quite often, state machines are designed which simply execute tests on various inputs waiting for the condition to come true. When the true condition is reached, the machine then branches and executes a set of microinstructions to perform some function. This usually has the effect of resetting the input being tested until some point in the future. Figure 4 shows the conditional jump via the pipeline register address at location 52. When the contents of microprogram memory word 52 are in the pipeline register, the next address will be either location 53 or location 30 in this example. If the test is passed, the value currently in the pipeline register (3) will be selected. If the test fails, the next address selected will be contained in the microprogram counter which, in this example, is 53.

Instruction 4 is the PUSH/CONDITIONAL LOAD COUNTER instruction and is used primarily for setting up loops in microprogram firmware. In Figure 4, when instruction 52 is in the pipeline register, a PUSH will be made onto the stack and the counter will be loaded based on the condition. When a PUSH occurs, the value pushed is always the next sequential instruction address. In this case, the address is 53. If the test fails, the counter is not loaded; if it is passed, the counter is loaded with the value contained in the pipeline register branch address field. Thus, a single microinstruction can be used to set up a loop to be executed a specific number of times. Instruction 8 will

THE Am2910 INSTRUCTION SET (Cont.)

describe how to use the pushed value and the register/counter for looping.

Instruction 5 is a CONDITIONAL JUMP-TO-SUBROUTINE via the register/counter or the contents of the PIPELINE register. As shown in Figure 4, a PUSH is always performed and one of two subroutines executed. In this example, either the subroutine beginning at address 80 or the subroutine beginning at address 90 will be performed. A return-from-subroutine (instruction number 10) returns the microprogram flow to address 55. In order for this microinstruction control sequence to operate correctly, both the next address fields of instruction 53 and the next address fields of instruction 54 would have to contain the proper value. Let's assume that the branch address fields of instruction 53 contain the value 90 so that it will be in the Am2910 register/counter when the contents of address 54 are in the pipeline register. This requires that the instruction at address 53 load the register/counter. Now, during the execution of instruction 5 (at address 54), if the test failed, the contents of the register (value = 90) will select the address of the next microinstruction. If the test input passes, the pipeline register contents (value = 80) will determine the address of the next microinstruction. Therefore, this instruction provides the ability to select one of two subroutines to be executed based on a test condition.

Instruction 6 is a CONDITIONAL JUMP VECTOR instruction which provides the capability to take the branch address from a third source heretofore not discussed. In order for this instruction to be useful, the Am2910 output, $\overline{VECT}$ is used to control a three-state control input of a register, buffer, or PROM containing the next microprogram address. This instruction provides one technique for performing interrupt type branching at the microprogram level. Since this instruction is conditional, a pass causes the next address to be taken from the vector source, while failure causes the next address to be taken from the microprogram counter. In the example of Figure 4, if the CONDITIONAL JUMP VECTOR instruction is contained at location 52, execution will continue at vector address 20 if the $\overline{CC}$ input is LOW and the microinstruction at address 53 will be executed if the $\overline{CC}$ input is HIGH.

Instruction 7 is a CONDITIONAL JUMP via the contents of the Am2910 REGISTER/COUNTER or the contents of the PIPELINE register. This instruction is very similar to instruction 5; the conditional jump-to-subroutine via R or PL. The major difference between instruction 5 and instruction 7 is that no push onto the stack is performed with 7. Figure 4 depicts this instruction as a branch to one of two locations depending on the test condition. The example assumes the pipeline register contains the value 70 when the contents of address 52 is being executed. As the contents of address 53 is clocked into the pipeline register, the value 70 is loaded into the register/counter in the Am2910. The value 80 is available when the contents of address 53 is in the pipeline register. Thus, control is transferred to either address 70 or address 80 depending on the test condition.

Instruction 8 is the REPEAT LOOP, COUNTER $\neq$ ZERO instruction. This microinstruction makes use of the decrementing capability of the register/counter. To be useful, some previous instruction, such as 4, must have loaded a count value into the register/counter. This instruction checks to see whether the register/counter contains a non-zero value. If so, the register/counter is decremented, and the address of the next microinstruction is taken from the top of the stack. If the register counter contains zero, the loop exit condition is occuring; control falls through to the next sequential microinstruction by selecting μPC; the stack is POP'd by decrementing the stack pointer, but the contents of the top of the stack are thrown away.

An example of the REPEAT LOOP, COUNTER $\neq$ ZERO instruction is shown in Figure 4. In this example, location 50 most likely would contain a PUSH/CONDITIONAL LOAD COUNTER instruction which would have caused address 51 to be PUSHed on the stack and the counter to be loaded with the proper value for looping the desired number of times.

In this example, since the loop test is made at the end of the instructions to be repeated (microaddress 54), the proper value to be loaded by the instructions at address 50 is one less than the desired number of passes through the loop. This method allows a loop to be executed 1 to 4096 times. If it is desired to execute the loop from 0 to 4095 times, the firmware should be written to make the loop exit test immediately after loop entry.

Single-microinstruction loops provide a highly efficient capability for executing a specific microinstruction a fixed number of times. Examples include fixed rotates, byte swap, fixed point multiply, and fixed point divide.

Instruction 9 is the REPEAT PIPELINE REGISTER, COUNTER $\neq$ ZERO instruction. This instruction is similar to instruction 8 except that the branch address now comes from the pipeline register rather than the file. In some cases, this instruction may be thought of as a one-word file extension; that is, by using this instruction, a loop with the counter can still be performed when subroutines are nested five deep. This instruction's operation is very similar to that of instruction 8. The differences are that on this instruction, a failed test condition causes the source of the next microinstruction address to be the D inputs; and, when the test condition is passed, this instruction does not perform a POP because the stack is not being used.

In the example of Figure 4, the REPEAT PIPELINE, COUNTER $\neq$ ZERO instruction is instruction 52 and is shown as a single microinstruction loop. The address in the pipeline register would be 52. Instruction 51 in this example could be the LOAD COUNTER AND CONTINUE instruction (number 12). While the example shows a single microinstruction loop, by simply changing the address in a pipeline register, multi-instruction loops can be performed in this manner for a fixed number of times as determined by the counter.

Instruction 10 is the conditional RETURN-FROM-SUBROUTINE instruction. As the name implies, this instruction is used to branch from the subroutine back to the next microinstruction address following the subroutine call. Since this instruction is conditional, the return is performed only if the test is passed. If the test is failed, the next sequential microinstruction is performed. The example in Figure 4 depicts the use of the conditional RETURN-FROM-SUBROUTINE instruction in both the conditional and the unconditional modes. This example first shows a jump-to-subroutine at instruction location 52 where control is transferred to location 90. At location 93, a conditional RETURN-FROM-SUBROUTINE instruction is performed. If the test is passed, the stack is accessed and the program will transfer to the next instruction at address 53. If the test is failed, the next microinstruction at address 94 will be executed. The program will continue to address 97 where the subroutine is complete. To perform an unconditional RETURN-FROM-SUBROUTINE, the conditional RETURN-FROM-SUBROUTINE instruction is executed unconditionally; the microinstruction at address 97 is programmed to force

THE Am2910 INSTRUCTION SET (Cont.)

$\overline{CCEN}$ HIGH, disabling the test and the forced PASS causes an unconditional return.

Instruction 11 is the CONDITIONAL JUMP PIPELINE register address and POP stack instruction. This instruction provides another technique for loop termination and stack maintenance. The example in Figure 4 shows a loop being performed from address 55 back to address 51. The instructions at locations 52, 53, and 54 are all conditional JUMP and POP instructions. At address 52, if the $\overline{CC}$ input is LOW, a branch will be made to address 70 and the stack will be properly maintained via a POP. Should the test fail, the instruction at location 53 (the next sequential instruction) will be executed. Likewise, at address 53, either the instruction at 90 or 54 will be subsequently executed, respective to the test being passed or failed. The instruction at 54 follows the same rules, going to either 80 or 55. An instruction sequence as described here, using the CONDITIONAL JUMP PIPELINE and POP instruction, is very useful when several inputs are being tested and the microprogram is looping waiting for any of the inputs being tested to occur before proceeding to another sequence of instructions. This provides the powerful jump-table programming technique at the firmware level.

Instruction 12 is the LOAD COUNTER AND CONTINUE instruction, which simply enables the counter to be loaded with the value at its parallel inputs. These inputs are normally connected to the pipeline branch address field which (in the architecture being described here) serves to supply either a branch address or a counter value depending upon the microinstruction being executed. There are altogether three ways of loading the counter – the explicit load by this instruction 12; the conditional load included as part of instruction 4; and the use of the $\overline{RLD}$ input along with any instruction. The use of $\overline{RLD}$ with any instruction overrides any counting or decrementation specified in the instruction, calling for a load instead. Its use provides additional microinstruction power, at the expense of one bit of microinstruction width. This instruction 12 is exactly equivalent to the combination of instruction 14 and $\overline{RLD}$ LOW. Its purpose is to provide a simple capability to load the register/counter in those implementations which do not provide microprogrammed control for $\overline{RLD}$.

Instruction 13 is the TEST END-OF-LOOP instruction, which provides the capability of conditionally exiting a loop at the bottom; that is this is a conditional instruction that will cause the microprogram to loop, via the file, if the test is failed else to continue to the next sequential instruction. The example in Figure 4 shows the TEST END-OF-LOOP microinstruction at address 56. If the test fails, the microprogram will branch to address 52. Address 52 is on the stack because a PUSH instruction had been executed at address 51. If the test is passed at instruction 56, the loop is terminated and the next sequential microinstruction at address 57 is executed, which also causes the stack to be POP'd; thus, accomplishing the required stack maintenance.

Instruction 14 is the CONTINUE instruction, which simply causes the microprogram counter to increment so that the next sequential microinstruction is executed. This is the simplest microinstruction of all and should be the default instruction which the firmware requests whenever there is nothing better to do.

Instruction 15, THREE-WAY BRANCH, is the most complex. It provides for testing of both a data-dependent condition and the counter during one microinstruction and provides for selecting among one of three microinstruction addresses as the next microinstruction to be performed. Like instruction 8, a previous instruction will have loaded a count into the register/counter while pushing a microbranch address onto the stack. Instruction 15 performs a decrement-and-branch-until-zero function similar to instruction 8. The next address is taken from the top of the stack until the count reaches zero; then the next address comes from the pipeline register. The above action continues as long as the test condition fails. If at any execution of instruction 15 the test condition is passed, no branch is taken; the microprogram counter register furnishes the next address. When the loop is ended, either by the count becoming zero, or by passing the conditional test, the stack is POP'd by decrementing the stack pointer, since interest in the value contained at the top of the stack is then complete.

The application of instruction 15 can enhance performance of a variety of machine-level instructions. For instance, (1) a memory search instruction to be terminated either by finding a desired memory content or by reaching the search limit; (2) variable-field-length arithmetic terminated early upon finding that the content of the portion of the field still unprocessed is all zeroes; (3) key search in a disc controller processing variable length records; (4) normalization of a floating point number.

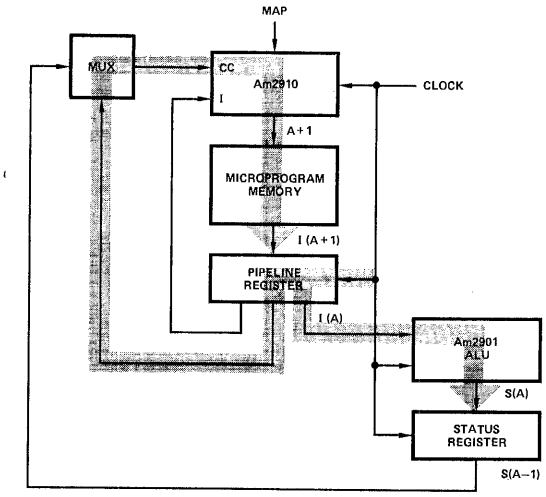
As one example, consider the case of a memory search instruction. As shown in Figure 4, the instruction at microprogram address 63 can be Instruction 4 (PUSH), which will push the value 64 onto the microprogram stack and load the number N, which is one less than the number of memory locations to be searched before giving up. Location 64 contains a microinstruction which fetches the next operand from the memory area to be searched and compares it with the search key. Location 65 contains a microinstruction which tests the result of the comparison and also is a THREE-WAY BRANCH for microprogram control. If no match is found, the test fails and the microprogram goes back to location 64 for the next operand address. When the count becomes zero, the microprogram branches to location 72, which does whatever is necessary if no match is found. If a match occurs on any execution of the THREE-WAY BRANCH at location 65, control falls through to location 66 which handles this case. Whether the instruction ends by finding a match or not, the stack will have been POP'd once, removing the value 64 from the top of the stack.

ARCHITECTURES USING THE Am2910

(Shading shows path(s) which usually limit speed)

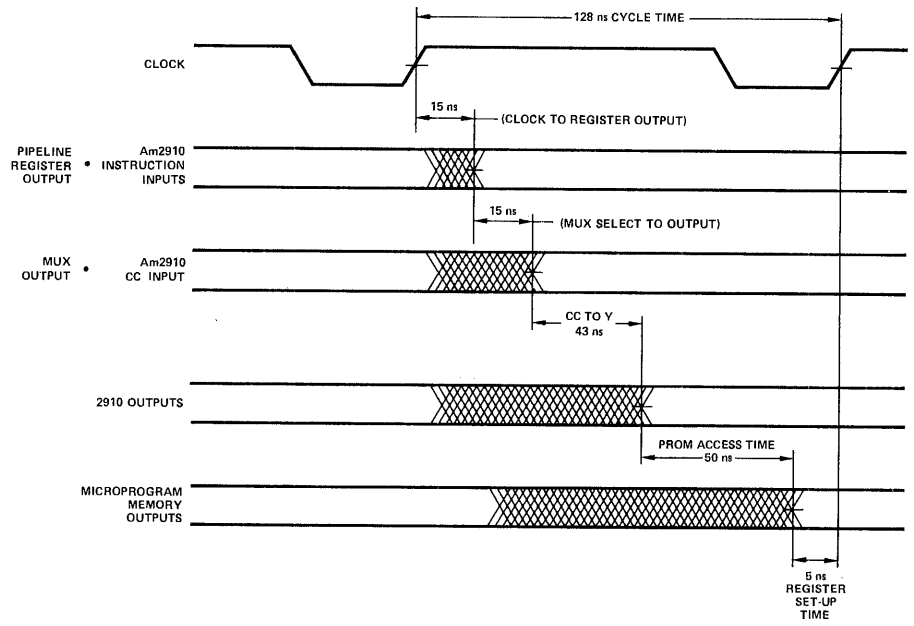
Figure 5.

One Level Pipeline Based
(Recommended)



One level pipeline provides better speed than most other architectures. The μProgram Memory and the Am2901 array are in parallel speed paths instead of in series. This is the recommended architecture for Am2900 designs.

MPR-112



Typical CCU Cycle Timing Waveforms.

This drawing shows the timing relationships in the CCU illustrated above.

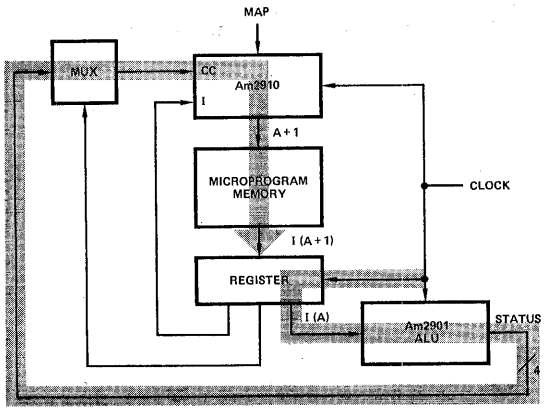
MPR-113

OTHER ARCHITECTURES USING THE Am2910

(Shading shows path(s) which usually limit speed)

Figure 6.

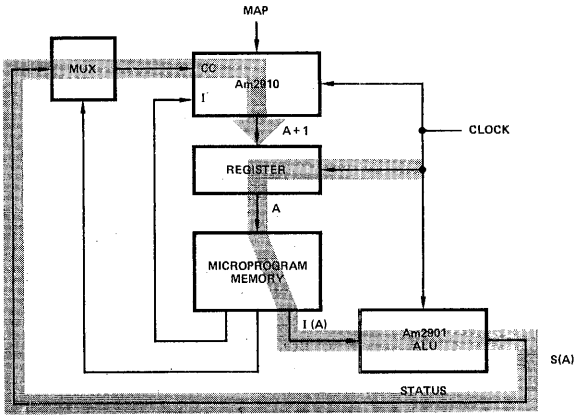
A. Instruction Based



A Register at the Microprogram Memory output contains the microinstruction being executed. The microprogram memory and Am2901 delay are in series. Conditional branches are executed on same cycle as the ALU operation generating the condition.

MPR-114

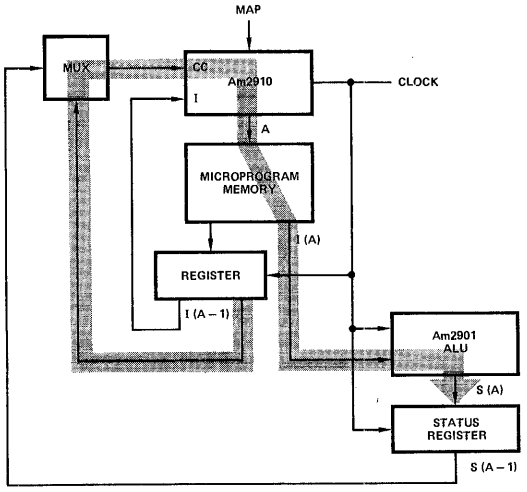
B. Addressed Based



The Register at the Am2910 output contains the address of the microinstruction being executed. The Microprogram Memory and Am2901 are in series in the critical path. This architecture provides about the same speed as the Instruction based architecture, but requires fewer register bits, since only the address (typically 10-12 bits) is stored instead of the instruction (typically 40-60 bits).

MPR-115

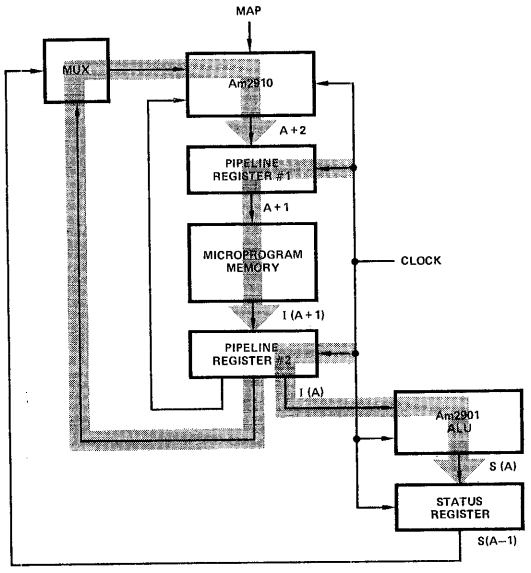
C. Data Based



The Status Register provides conditional Branch control based on results of previous ALU cycle. The Microprogram Memory and Am2901 are in series in the critical paths.

MPR-116

D. Two Level Pipeline Based



Two level pipeline provides highest possible speed. It is more difficult to program because the selection of a microinstruction occurs two instructions ahead of its execution.

MPR-117

Am2913

Priority Interrupt Expander

Distinctive Characteristics

- Encodes eight lines to three-line binary
- Expands use of Am2914
- Cascadable
- Similar in function to Am54LS/74LS/25LS148/2513
- Gated three-state output
- Advanced Low-Power Schottky processing
- 100% reliability assurance testing in compliance with MIL-STD-883

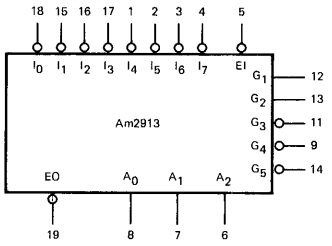
2

FUNCTIONAL DESCRIPTION

The Low-Power Schottky Priority Interrupt Expander is an extention of the Am2900 series of Bipolar Processor family and is used to expand and prioritize the output of the Am2914 Priority Interrupt circuit. Affording an increase of vectored priority interrupt in groups of eight, this unit accepts active LOW inputs and produces a three-state active HIGH output prioritized from active $\overline{I}_7$ to $\overline{I}_0$. The output is gated by five control signals, three active LOW and two active HIGH. Also provided is a cascade input ($\overline{EI}$) and Enable Output ($\overline{EO}$).

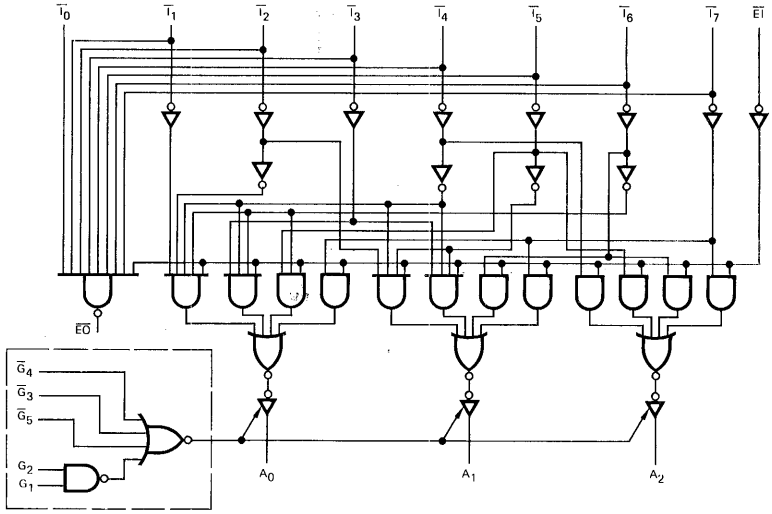
One Am2913 will accept and encode group signal lines from up to 8 Am2914's (64 levels of interrupt). Additional Am2913's may be used to encode more interrupt levels.

LOGIC SYMBOL



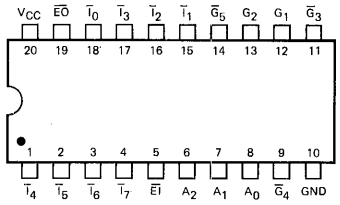
MPR-118

LOGIC DIAGRAM



MPR-119

CONNECTION DIAGRAM
Top View



Note: Pin 1 is marked for orientation.

MPR-120

DEFINITIONS OF FUNCTIONAL TERMS

A0, A1, A2	Three-state, active high encoder outputs
$\overline{EI}$	Enable input provided to allow cascaded operation
$\overline{EO}$	Enable output provided to enable the next lower order priority chip
G_1, G_2	Active high three-state output controls
$\overline{G}_3$, $\overline{G}_4$, $\overline{G}_5$	Active low three-state output controls
$\overline{I}_{0-7}$	Active low encoder inputs

TRUTH TABLE

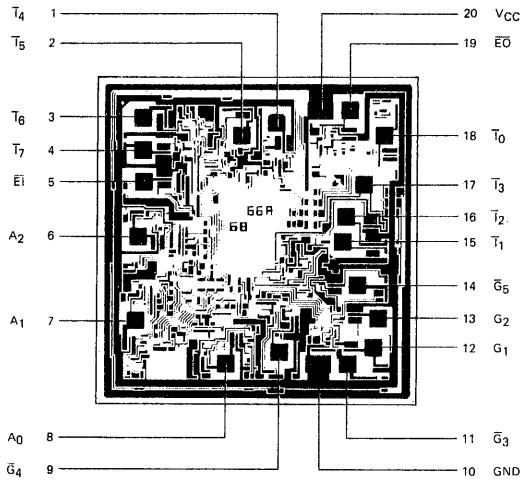
Inputs									Outputs			
$\overline{EI}$	$\overline{I}_0$	$\overline{I}_1$	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	A_0	A_1	A_2	$\overline{EO}$
H	X	X	X	X	X	X	X	X	L	L	L	H
L	H	H	H	H	H	H	H	H	L	L	L	L
L	X	X	X	X	X	X	X	L	H	H	H	H
L	X	X	X	X	X	X	L	H	L	H	H	H
L	X	X	X	X	X	L	H	H	H	L	H	H
L	X	X	X	X	L	H	H	H	L	L	H	H
L	X	X	X	L	H	H	H	H	H	H	L	H
L	X	X	L	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	L	L	L	H

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care
For G_1 = H, G_2 = H, G_3 = L, G_4 = L, G_5 = L

G1	G2	$\overline{G3}$	$\overline{G4}$	$\overline{G5}$	A_0	A_1	A_2
H	H	L	L	L	Enabled		
L	X	X	X	X	Z	Z	Z
X	L	X	X	X	Z	Z	Z
X	X	H	X	X	Z	Z	Z
X	X	X	H	X	Z	Z	Z
X	X	X	X	H	Z	Z	Z

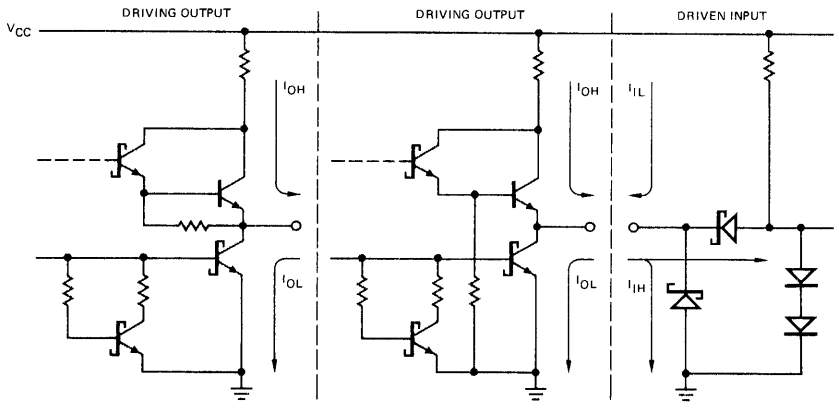
Z = HIGH Impedance

Metallization and Pad Layout



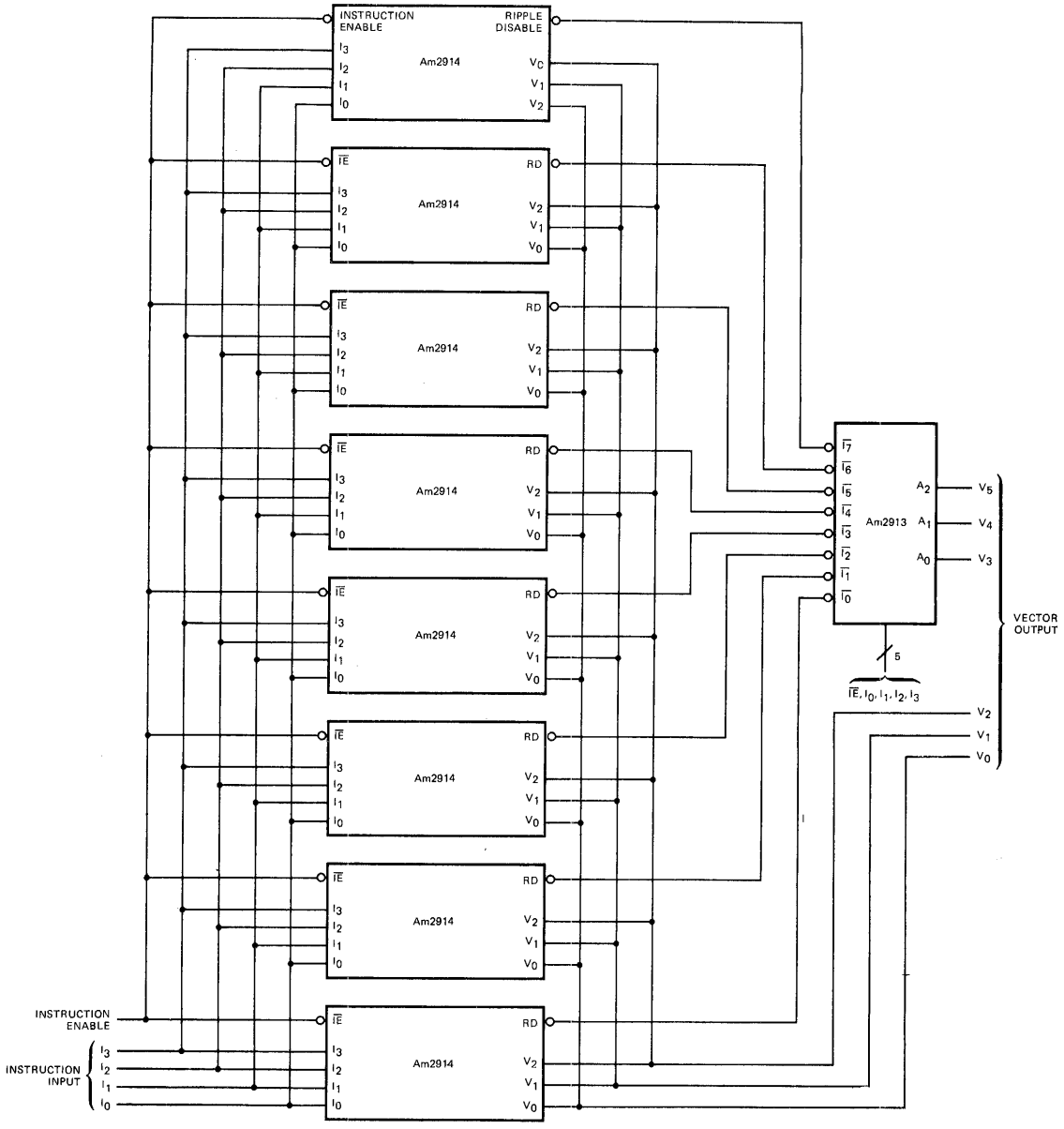
DIE SIZE 0.082" X 0.085"

LOW-POWER SCHOTTKY INPUT/OUTPUT CURRENT INTERFACE CONDITIONS



Note: Actual current flow direction shown.

MPR-121



Shown above is the connection of the instruction lines and vector output lines in a 64-input priority interrupt system. The Am2913 is used to encode the most significant bits associated with the vector output.

MPR-122

Am2914

Vectored Priority Interrupt Controller

DISTINCTIVE CHARACTERISTICS

- Accepts 8 interrupt inputs
  Interrupts may be pulses or levels and are stored internally
- Built-in mask register
  Six different operations can be performed on mask register
- Built-in status register
  Status register holds code for lowest allowed interrupt
- Vectored output
  Output is binary code for highest priority un-masked interrupt
- Expandable
  Any number of Am2914's may be stacked for large interrupt systems
- Microprogrammable
  Executes 16 different microinstructions
  Instruction enable pin aids in vertical microprogramming
- High-speed operation
  Delay from an interrupt clocked into the interrupt register to interrupt request output is typically 60 ns

TABLE OF CONTENTS

For applications information,
see Chapter VI of "Build a Microcomputer".

FUNCTIONAL DESCRIPTION

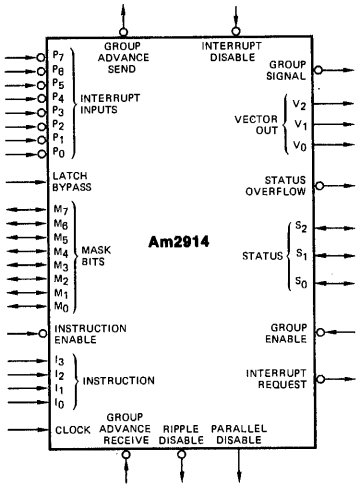
The Am2914 is a high-speed, eight-bit priority interrupt unit that is cascadable to handle any number of priority interrupt request levels. The high-speed of the Am2914 makes it ideal for use in Am2900 family microcomputer designs, but it can also be used with the Am9080A MOS microprocessor.

The Am2914 receives interrupt requests on 8 interrupt input lines (P_0-P_7). A LOW level is a request. An internal latch may be used to catch pulses on these lines, or the latch may be bypassed so the request lines drive the edge-triggered interrupt register directly. An 8-bit mask register is used to mask individual interrupts. Considerable flexibility is provided for controlling the mask register. Requests in the interrupt register are ANDed with the corresponding bits in the mask register and the results are sent to an 8-input priority encoder, which produces a three bit encoded vector representing the highest numbered input which is not masked.

An internal status register is used to point to the lowest priority at which an interrupt will be accepted. The contents of the status register are compared with the output of the priority encoder, and an interrupt request output will occur if the vector is greater than or equal to status. Whenever a vector is read from the Am2914 the status register is automatically updated to point to one level higher than the vector read. (The status register can be loaded externally or read out at any time using the S pins.) Signals are provided for moving the status upward across devices (Group Advance Send and Group Advance Receive) and for inhibiting lower priorities from higher order devices (Ripple Disable, Parallel Disable, and Interrupt Disable). A status overflow output indicates that an interrupt has been read at the highest priority.

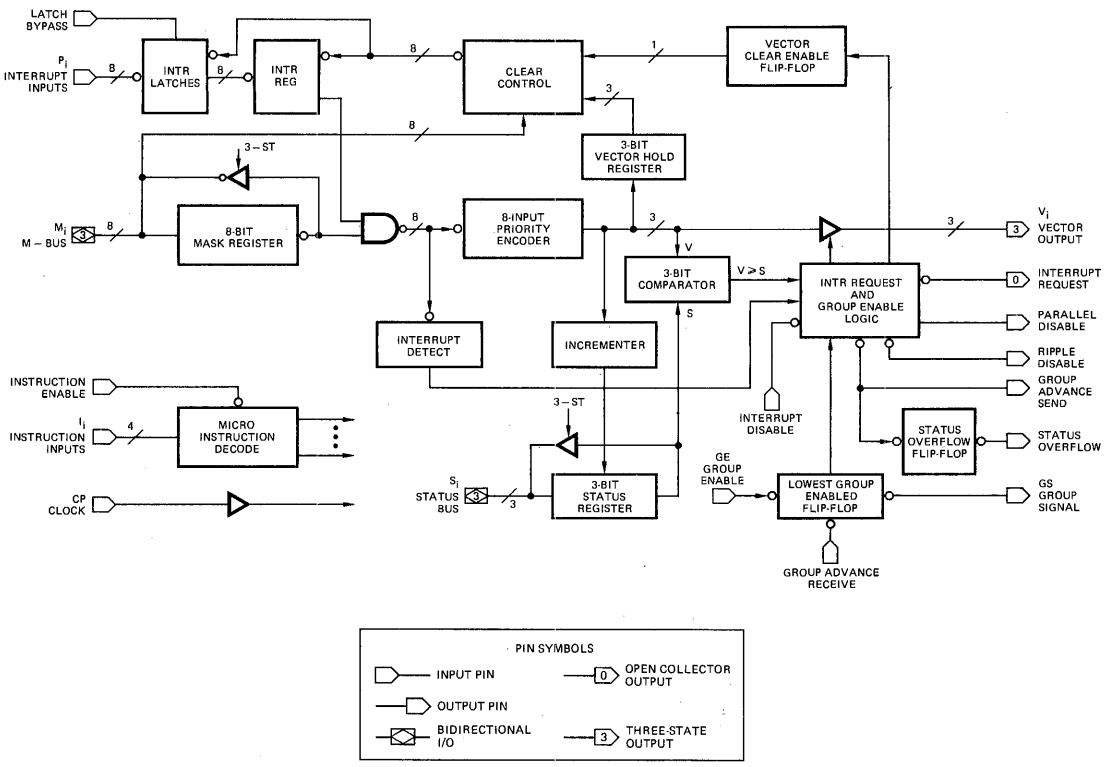
The Am2914 is controlled by a 4-bit instruction field I_0-I_3. The command on the instruction lines is executed if IE is LOW and is ignored if IE is HIGH, allowing the 4 I bits to be shared with other devices.

LOGIC SYMBOL



MPR-123

BLOCK DIAGRAM



MPR-124

BLOCK DIAGRAM DESCRIPTION

The Microinstruction Decode circuitry decodes the Interrupt Microinstructions and generates required control signals for the chip.

The Interrupt Register holds the Interrupt Inputs and is an eight-bit, edge-triggered register which is set on the rising edge of the CP Clock signal.

The Interrupt latches are set/reset-type latches. When the Latch Bypass signal is LOW, the latches are enabled and act as negative pulse catchers on the inputs to the Interrupt Register. When the Latch Bypass signal is HIGH, the Interrupt latches are transparent.

The Mask Register holds the eight mask bits associated with the eight interrupt levels. The register may be loaded from or read to the M Bus. Also, the entire register or individual mask bits may be set or cleared.

The Interrupt Detect circuitry detects the presence of any unmasked Interrupt Input. The eight-input Priority Encoder determines the highest priority, non-masked Interrupt Input and forms a binary coded interrupt vector. Following a Vector Read, the three-bit Vector Hold Register holds the binary coded interrupt vector. This stored vector is used for clearing interrupts.

The three-bit Status Register holds the status bits and may be loaded from or read to the S Bus. During a Vector Read, the Incrementer increments the interrupt vector by one, and the result is clocked into the Status Register. Thus the Status Register always points to the lowest level at which an interrupt will be accepted.

The three-bit Comparator compares the Interrupt Vector with the contents of the Status Register and indicates if the Interrupt Vector is greater than or equal to the contents of the Status Register.

The Lowest Group Enabled Flip-Flop is used when a number of 2914's are cascaded. In a cascaded system, only one Lowest Group Enabled Flip-Flop is LOW at a time. It indicates the eight interrupt group, which contains the lowest priority interrupt level which will be accepted and is used to form the higher order status bits.

The Interrupt Request and Group Enable logic contain various gating to generate the Interrupt Request, Parallel Disable, Ripple Disable, and Group Advance Send signals.

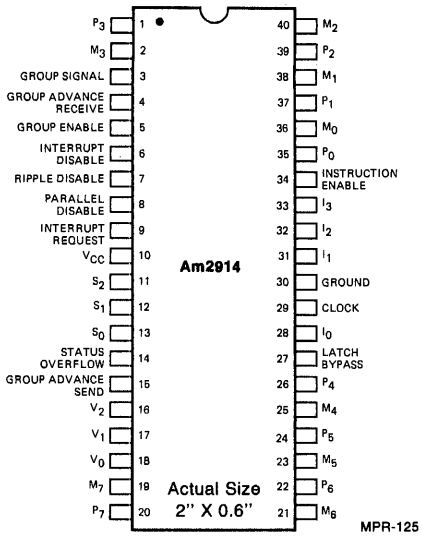
The Status Overflow signal is used to disable all interrupts. It indicates the highest priority interrupt vector has been read and the Status Register has overflowed.

The Clear Control logic generates the eight individual clear signals for the bits in the Interrupt Latches and Register. The Vector Clear Enable Flip-Flop indicates if the last vector read was from this group. When it is set, it enables the Clear Control Logic.

The CP clock signal is used to clock the Interrupt Register, Mask Register, Status Register, Vector Hold Register, and the Lowest Group Enabled, Vector Clear Enable and Status Overflow Flip-Flops, all on the clock LOW-to-HIGH transition.

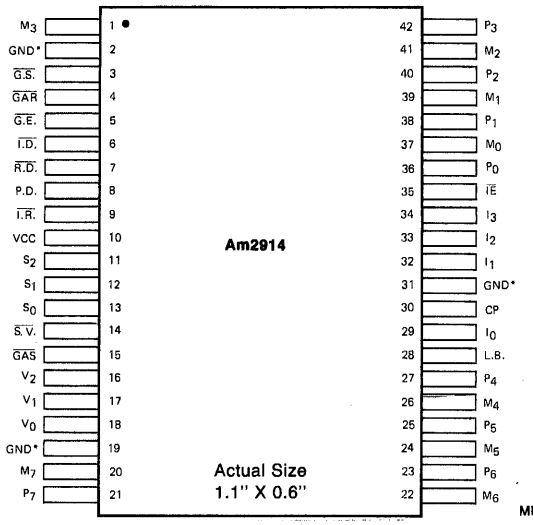
CONNECTION DIAGRAMS – Top Views

DIP



MPR-125

Flat Package

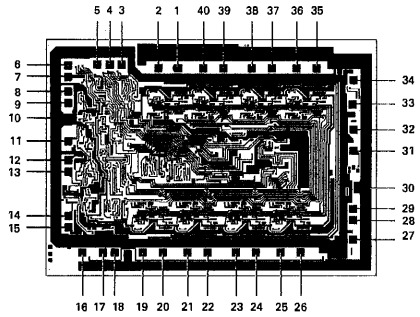


MPR-126

*Note: GND's and pins 2, 19, 31 must all be tied together externally.

Note: Pin 1 is marked for orientation.

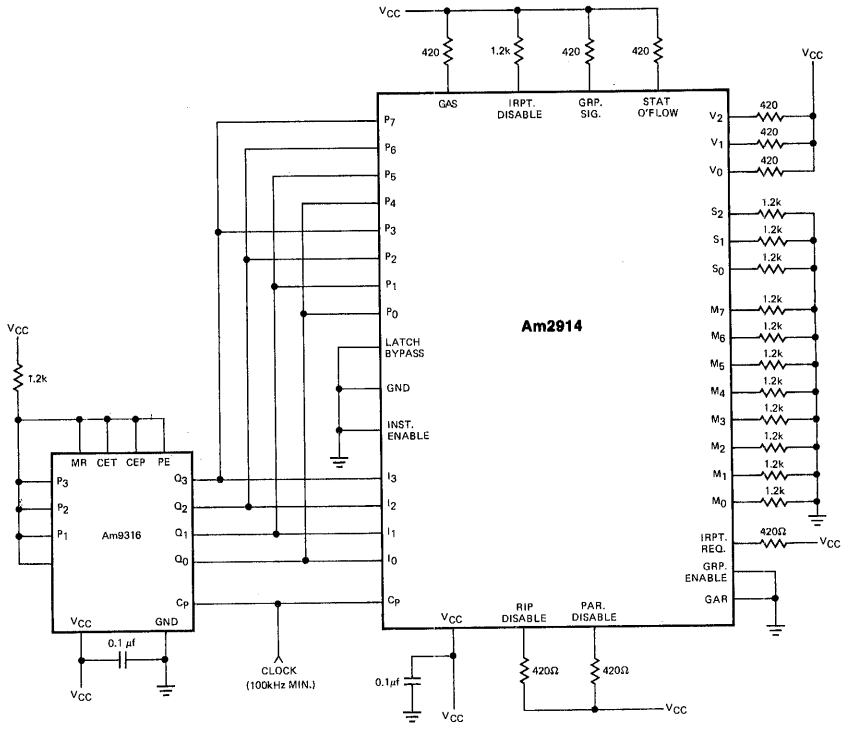
Metallization and Pad Layout



DIE SIZE
0.133" X 0.187"
Numbers correspond to DIP pin-out.

TABLE I
MICROINSTRUCTION SET FOR Am2914 PRIORITY INTERRUPT CIRCUIT

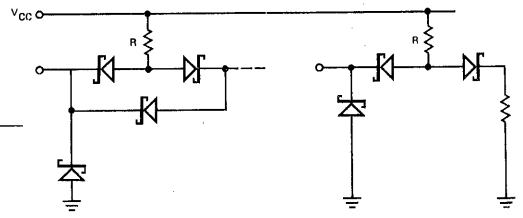
Decimal $I_3I_2I_1I_0$	Mnemonic	Instruction
		Mask Register Functions
14	LDM	Load mask register from M bus
7	RDM	Read mask register to M bus
12	CLRM	Clear mask register (enables all priorities)
8	SETM	Set mask register (inhibits all interrupts)
10	BCLRM	Bit clear mask register from M bus
11	BSETM	Bit set mask register from M bus
		Status Register Functions
9	LDSTA	Load status register from S bus and LGE flip-flop from GE input
6	RDSTA	Read status register to S bus
		Interrupt Request Control
15	ENIN	Enable interrupt request
13	DISIN	Disable interrupt request
		Vectored Output
5	RDVC	Read vector output to V outputs, load V+1 into status register, load V into vector hold register and set vector clear enable flip-flop.
		Priority Interrupt Register Clear
1	CLRIN	Clear all interrupts
3	CLRMR	Clear interrupts from mask register data (uses the M bus)
2	CLRMB	Clear interrupts from M bus data
4	CLRVC	Clear the individual interrupt associated with the last vector read
		Master Clear
0	MCLR	Clear all interrupts, clear mask register, clear status register, clear LGE flip-flop, enable interrupt request.



Am2914 Burn-in Circuit

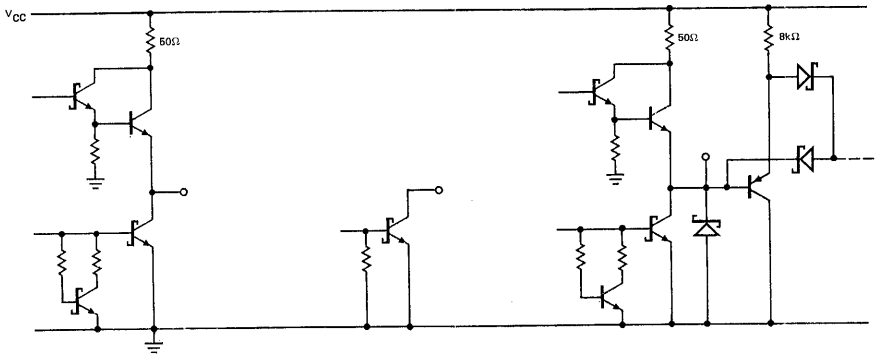
MPR-127

INPUT/OUTPUT CIRCUITS



INPUT	R
I_0-I_3, CP, $\overline{GE}$, $\overline{GAR}$	10K
LB	20K
$\overline{IE}$	6.7K
$\overline{ID}$	4K

INPUT	R
$\overline{P_7}$-$\overline{P_0}$	10K



OUTPUTS	
V_0-V_3	$\overline{SO}$
$\overline{GAS}$	PD
$\overline{GS}$	$\overline{RD}$

OUTPUT
$\overline{IR}$

OUTPUTS
M_0-M_7
S_0-S_3

MPR-128

A MICROPROGRAMMABLE, BIPOLAR, LSI INTERRUPT STRUCTURE USING THE Am2914

INTRODUCTION

Advanced Micro Devices' introduction of the Am2914 Vectored Priority Interrupt Controller now makes possible the structuring of a microprogrammable bipolar LSI interrupt system. The design engineer may use the Am2914 to simplify his design process, dramatically reduce the system cost, size and package count, and increase the speed, capability and reliability of his interrupt system.

The Am2914 is a modular, low cost, standard LSI component that may be microprogrammed to meet the requirements of specific applications. Today's engineer may utilize the Am2914 microprogrammability to provide functional flexibility and ease of engineering change, while taking advantage of its modularity to provide hardware regularity and future expansion capability.

THE INTERRUPT CONCEPT

In any state machine, a requirement exists for the efficient synchronization and response to asynchronous events such as power failure, machine malfunctions, control panel service requests, external timer signals, supervisory calls, program errors, and input/output device service requests. The merit of such an "asynchronous event handler" may be measured in terms of response time, system throughput, real time overhead, hardware cost and memory space required.

The simplest approach to asynchronous event handling is the poll approach. A status indicator is associated with each possible asynchronous event. The processor tests each indicator in sequence and, in effect, "asks" if service is required. This program-driven method is inefficient for a number of reasons. Much time is consumed polling when no service is required; programs must have frequent test points to poll indicators, and since indicators are polled in sequence, considerable time may elapse before the processor responds to an event. Thus, system throughput is low; real time overhead and response time are high, and a large memory space is required.

The interrupt method is a much more efficient way of servicing asynchronous requests. An asynchronous event requiring service generates an interrupt request signal to the processor. When the processor receives the interrupt request, it may suspend the program it is currently executing, execute an interrupt service routine which services the asynchronous request, then resume the execution of the suspended program. In this system, the execution of the service routine is initiated by an interrupt request; thus, the system is interrupt driven and service routines are executed only when service is requested. Although hardware cost may be higher in this type of system, it is more efficient since system throughput is higher, response time is faster, real time overhead is lower and less memory space is required.

INTERRUPT SYSTEM FUNCTIONAL DEFINITION

A complete and clear functional definition is key to the design of a good interrupt system. The following features are useful.

Multiple Interrupt Request Handling: Since interrupt requests are generated from a number of different sources, the interrupt system's ability to handle interrupt requests from several sources is important.

Interrupt Request Prioritization: Since the processor can service only one interrupt request at a time, it is important that the interrupt system has the ability to prioritize the requests and determine which has the highest priority.

Interrupt Service Routine "Nesting": This feature allows an interrupt service routine for a given priority request to be interrupted in turn, but only by a higher priority interrupt request. The service routine for the higher priority request is executed, then the execution of the interrupted service routine is resumed. If there are "n" interrupt requests, an "n" deep "nest" is possible.

Dynamic Interrupt Enabling/Disabling: The ability to enable/disable all interrupts "on the fly" under microprogram control can be used to prevent interruption of certain processes.

Dynamic Interrupt Request Masking: The ability to selectively inhibit or "mask" individual interrupt requests under microprogram control is useful.

Interrupt Request Vectoring: Many times, a particular interrupt request requires the execution of a unique interrupt service routine. For this reason, the generation of a unique binary coded vector for each interrupt request is very helpful. This vector can be used as a pointer to the start of a unique service routine.

Interrupt Request Priority Threshold: The ability to establish a priority threshold is valuable. In this type of operation, only those interrupt requests which have higher priority than a specified threshold priority are accepted. The threshold priority can be defined by microprogram or can be automatically established by hardware at the interrupt currently being serviced plus one. This automatic threshold prevents multiple interrupts from the same source. Also useful is the ability to read the threshold priority under microprogram control. Thus, the interrupt request being serviced may be determined by the microprogram.

Interrupt Request Clearing Flexibility: Flexibility in the method of clearing interrupt requests allows different modes of interrupt system operation. Of particular value are the abilities to clear the interrupt currently being serviced, clear all interrupts, or clear interrupts via a programmable mask register or bus.

Microprogrammability: Microprogrammability permits the construction of a general purpose or "universal" interrupt structure which can be microprogrammed to meet a specific application's requirements. The universality of the structure allows standardization of the hardware and amortization of the hardware development costs across a much broader user base. The end result is a flexible, low cost interrupt structure.

Hardware Modularity: Modular interrupt system hardware is beneficial in two ways. First, hardware modularity provides expansion capability. Additional modules may be added as the need to service additional requests arises. Secondly, hardware modularity provides a structural regularity which simplifies the system structure and also reduces the number of hardware part numbers.

Fast Interrupt System Response Time: Quick interrupt system response provides more efficient system operation. Fast response reduces real time overhead and increases overall system throughput.

INTERRUPT SYSTEM IMPLEMENTATION USING THE Am2914

The Am2914 provides all of the foregoing features on a single LSI chip. The Am2914 is a high-speed, eight-bit priority interrupt unit that is cascadable to handle any number of priority interrupt request levels. The Am2914's high speed is ideal for use in Am2900 Family microcomputer designs, but it can also be used with the Am9080A MOS microprocessor.

The Am2914 receives interrupt requests on eight Interrupt Input lines (P_0-P_7). A LOW level is a request. An internal latch may be used to catch pulses (HIGH-LOW-HIGH) on these lines, or the latch may be bypassed so that the request lines drive the D-inputs to the edge-triggered Interrupt Register directly. An eight-bit Mask Register is used to mask individual interrupts. Considerable flexibility is provided for controlling the Mask Register. Requests in the Interrupt Register (P_0-P_7) are ANDed with the corresponding bits in the mask register (M_0-M_7) and the results are sent to an eight-input priority encoder, which produces a three-bit encoded vector representing the highest priority input which is not masked.

An internal Status Register is used to point to the lowest priority at which an interrupt will be accepted. The contents of the Status Register are compared with the output of the priority encoder, and an Interrupt Request output will occur if the vector is greater than or equal to the contents of the Status Register. Whenever a vector is read from the Am2914, the Status Register is automatically updated to point to one level higher than the vector read. (The Status Register can be loaded externally or read out at any time using the S-Bus.) Signals are provided for moving the status upward across devices (Group Advance Send and Group Advance Receive) and for inhibiting lower priorities from higher order devices (Ripple Disable, Parallel Disable, and Interrupt Disable). A Status Overflow output indicates that an interrupt has been read at the highest priority.

The Am2914 is controlled by a four-bit microinstruction field I_0-I_3. The microinstruction is executed if $\overline{IE}$ (Instruction Enable) is LOW and is ignored if $\overline{IE}$ is HIGH, allowing the four I bits to be shared with other functions. Sixteen different microinstructions are executed. Figure 2 shows the microinstructions and the microinstruction codes.

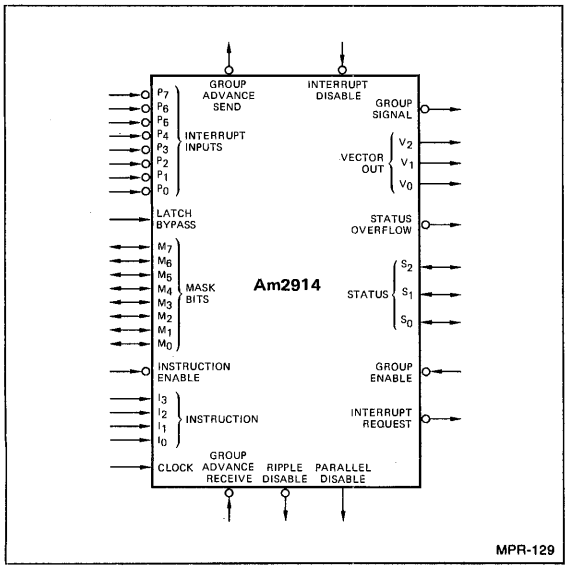


Figure 1. Am2914 Logic Symbol.

MICROINSTRUCTION DESCRIPTION	MICROINSTRUCTION CODE $I_3I_2I_1I_0$
MASTER CLEAR	0000
CLEAR ALL INTERRUPTS	0001
CLEAR INTERRUPTS FROM M-BUS	0010
CLEAR INTERRUPTS FROM MASK REGISTER	0011
CLEAR INTERRUPT, LAST VECTOR READ	0100
READ VECTOR	0101
READ STATUS REGISTER	0110
READ MASK REGISTER	0111
SET MASK REGISTER	1000
LOAD STATUS REGISTER	1001
BIT CLEAR MASK REGISTER	1010
BIT SET MASK REGISTER	1011
CLEAR MASK REGISTER	1100
DISABLE INTERRUPT REQUEST	1101
LOAD MASK REGISTER	1110
ENABLE INTERRUPT REQUEST	1111

Figure 2. Am2914 Microinstruction Set.

In this microinstruction set, the *Master Clear* microinstruction is selected as binary zero so that during a power up sequence, the microinstruction register in the microprogram control unit of the central processor can be cleared to all zeros. Thus, on the next clock cycle, the Am2914 will execute the *Master Clear* function. This includes clearing the Interrupt Latches and Register as well as the Mask Register and Status Register. The LGE flip-flop of the least significant group is set LOW because the Group Advance Receive input is tied LOW. All other Group Advance Receive inputs are tied to Group Advance Send outputs and these are forced HIGH during this instruction. This clear instruction also sets the Interrupt Request Enable flip-flop so that a fully interrupt driven system can be easily initiated from any interrupt.

The *Clear All Interrupts* microinstruction clears the Interrupt Latches and Register.

The *Clear Interrupts from Mask Register* microinstruction clears those Interrupt Latches and Register bits which have corresponding Mask Register bits set equal to one. The M-Bus is used by the Am2914 during the execution of this microinstruction and must be floating.

The *Clear Interrupts from M-Bus* microinstruction clears those Interrupt Latches and Register bits which have corresponding M-Bus bits set equal to one.

The *Clear Interrupt, Last Vector Read* microinstruction clears the Interrupt Latch and Register bit associated with the last vector read.

The *Read Vector* microinstruction is used to read the vector value of the highest priority request causing the interrupt. The vector outputs are three-state drivers that are enabled onto the $V_0V_1V_2$ bus during this instruction. This microinstruction also automatically loads the value "vector plus one" into the Status Register. In addition, this instruction sets the Vector Clear Enable flip-flop and loads the current vector value into the Vector Hold Register so that this value can be used by the *Clear Interrupt, Last Vector Read* microinstruction. This allows the user to read the vector associated with the interrupt, and at some later time clear the Interrupt Latch and Register bit associated with the vector read.

The *Load Status Register* microinstruction loads S-Bus data into the Status Register and also loads the LGE flip-flop from the Group Enable input.

During the *Read Status Register* microinstruction, the Status Register outputs are enabled onto the Status Bus (S_0-S_2). The Status Bus is a three-bit, bi-directional, three-state bus.

The *Load Mask Register* microinstruction loads data from the three-state, bi-directional M-Bus into the Mask Register.

The *Read Mask Register* microinstruction enables the Mask Register outputs onto the bi-directional, three-state M-Bus.

The *Set Mask Register* microinstruction sets all the bits in the Mask Register to one. This results in all interrupts being inhibited.

The entire Mask Register is cleared by the *Clear Mask Register* microinstruction. This enables all interrupts subject to the Interrupt Enable flip-flop and the Status Register.

The *Bit Clear Mask Register* microinstruction may be used to selectively clear individual Mask Register bits. This microinstruction clears those Mask Register bits which have corresponding M-Bus bits equal to one. Mask Register bits with corresponding M-Bus bits equal to zero are not affected.

The *Bit Set Mask Register* microinstruction sets those Mask Register bits which have corresponding M-Bus bits equal to one. Other Mask Register bits are not affected.

All Interrupt Requests may be disabled by execution of the *Disable Interrupt Request* microinstruction. This microinstruction resets an Interrupt Request Enable flip-flop on the chip.

The *Enable Interrupt Request* microinstruction sets the Interrupt Enable flip-flop. Thus, Interrupt Requests are enabled subject to the contents of the Mask and Status Registers.

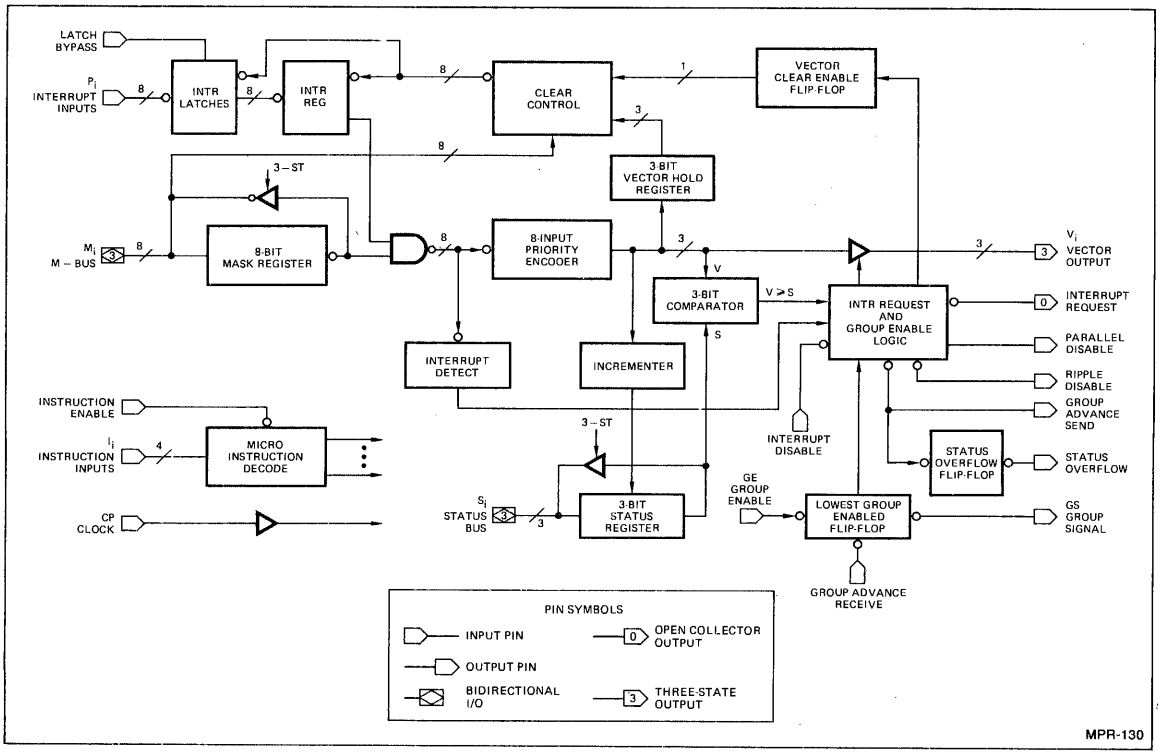


Figure 3. Am2914 Block Diagram.

Am2914 BLOCK DIAGRAM DESCRIPTION

The Am2914 block diagram is shown in Figure 3. The Micro-instruction Decode circuitry decodes the Interrupt Micro-instructions and generates required control signals for the chip.

The Interrupt Register holds the Interrupt Inputs and is an eight-bit, edge-triggered register which is set on the rising edge of the CP Clock signal if the Interrupt Input is LOW.

The Interrupt latches are set/reset latches. When the Latch Bypass signal is LOW, the latches are enabled and act as negative pulse catchers on the inputs to the Interrupt Register. When the Latch Bypass signal is HIGH, the Interrupt latches are transparent.

The Mask Register holds the eight mask bits associated with the eight interrupt levels. The register may be loaded from or read to the M-Bus. Also, the entire register or individual mask bits may be set or cleared.

The Interrupt Detect circuitry detects the presence of any unmasked Interrupt Input. The eight-input Priority Encoder determines the highest priority, non-masked Interrupt Input and forms a binary coded interrupt vector. Following a Vector Read, the three-bit Vector Hold Register holds the binary coded interrupt vector. This stored vector can be used later for clearing interrupts.

The three-bit Status Register holds the status bits and may be loaded from or read to the S-Bus. During a Vector Read, the Incrementer increments the interrupt vector by one, and the result is clocked into the Status Register. Thus, the Status Register points to a level one greater than the vector just read.

The three-bit Comparator compares the Interrupt Vector with the contents of the Status Register and indicates if the Interrupt Vector is greater than or equal to the contents of the Status Register.

The Lowest Group Enabled Flip-Flop is used when a number of Am2914's are cascaded. In a cascaded system, only one Lowest Group Enabled Flip-Flop is LOW at a time. It indicates the eight interrupt group, which contains the lowest priority interrupt level which will be accepted and is used to form the higher order status bits.

The Interrupt Request and Group Enable logic contain various gating to generate the Interrupt Request, Parallel Disable, Ripple Disable, and Group Advance Send signals.

The Status Overflow signal is used to disable all interrupts. It indicates the highest priority interrupt vector has been read and the Status Register has overflowed.

The Clear Control logic generates the eight individual clear signals for the bits in the Interrupt Latches and Register. The Vector Clear Enable Flip-Flop indicates if the last vector read was from this chip. When it is set it enables the Clear Control Logic.

The CP clock signal is used to clock the Interrupt Register, Mask Register, Status Register, Vector Hold Register, and the Lowest Group Enabled, Vector Clear Enable and Status Overflow Flip-Flops, all on the clock LOW-to-HIGH transition.

The Am2914 can be microprogrammed in many different ways. Figure 4 shows an example interrupt sequence. The *Read Vector* microinstruction is necessary in order to read the interrupt priority level. Since vector plus one is automatically loaded into the Status Register when a *Read Vector* microinstruction is executed, the Status Register possibly will overflow and disable all interrupts. For this reason, the Status Register must be reloaded periodically. The other Am2914 microinstructions are optional.

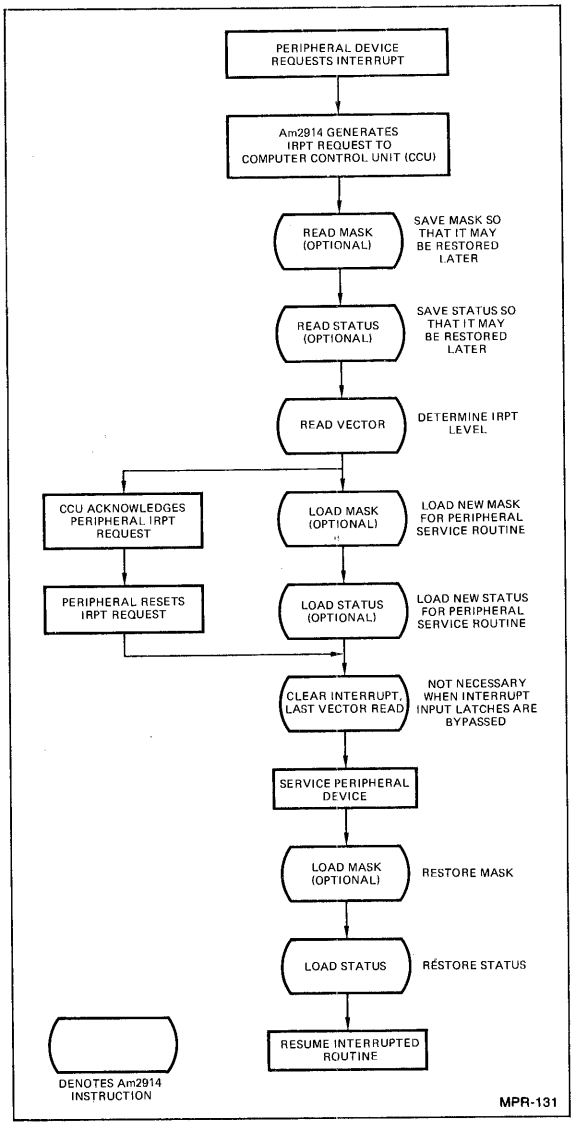


Figure 4. Example Interrupt Sequence.

CASCADING THE Am2914

A number of input/output signals are provided for cascading the Am2914 Vectored Priority Interrupt Encoder. A definition of these I/O signals and their required connections follows:

Group Signal ($\overline{GS}$) – This signal is the output of the Lowest Group Enabled flip-flop and during a Read Status microinstruction is used to generate the high order bits of the Status word.

Group Enable ($\overline{GE}$) – This signal is one of the inputs to the Lowest Group Enable flip-flop and is used to load the flip-flop during the Load Status microinstruction.

Group Advance Send ($\overline{GAS}$) – During a Read Vector microinstruction, this output signal is LOW when the highest priority vector (vector seven) of the group is being read. In a cascaded system Group Advance Send must be tied to the Group Advance Receive input of the next higher group in order to transfer status information.

Group Advance Receive ($\overline{GAR}$) – During a Master Clear or Read Vector microinstruction, this input signal is used with other internal signals to load the Lowest Group Enabled flip-flop. The Group Advance Receive input of the lowest priority group must be tied to ground.

Status Overflow ($\overline{SV}$) – This output signal becomes LOW after the highest priority vector (vector seven) of the group has been read and indicates the Status Register has overflowed. It stays LOW until a Master Clear or Load Status microinstruction is executed. The Status Overflow output of the highest priority group should be connected to the Interrupt Disable input of the same group and serves to disable all interrupts until new status is loaded or the system is master cleared. The Status Overflow outputs of lower priority groups should be left open (see Figure 7).

Interrupt Disable ($\overline{ID}$) – When LOW, this input signal inhibits the Interrupt Request output from the chip and also generates a Ripple Disable output.

Ripple Disable ($\overline{RD}$) – This output signal is used only in the Ripple Cascade Mode (see below). The Ripple Disable output is LOW when the Interrupt Disable input is LOW, the Lowest Group Enabled flip-flop is LOW, or an Interrupt Request is generated in the group. In the ripple cascade mode, the

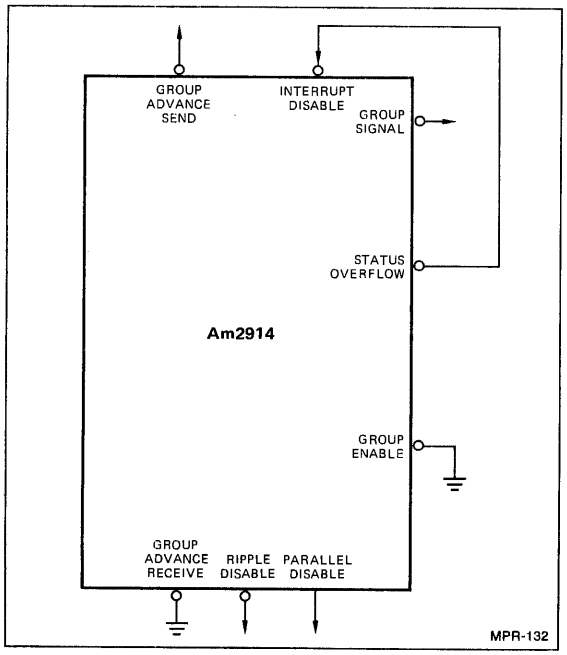


Figure 5. Cascade Lines Connection for Single Chip System.

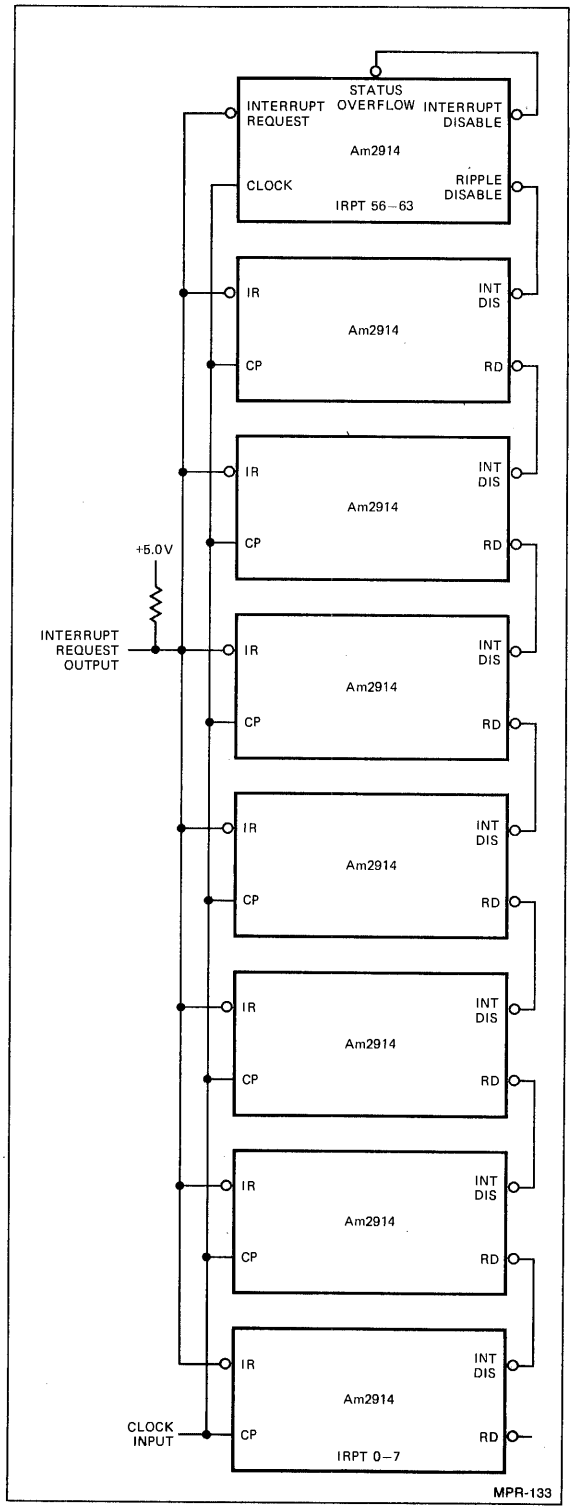


Figure 6. Interrupt Disable Connections for Ripple Cascade Mode.

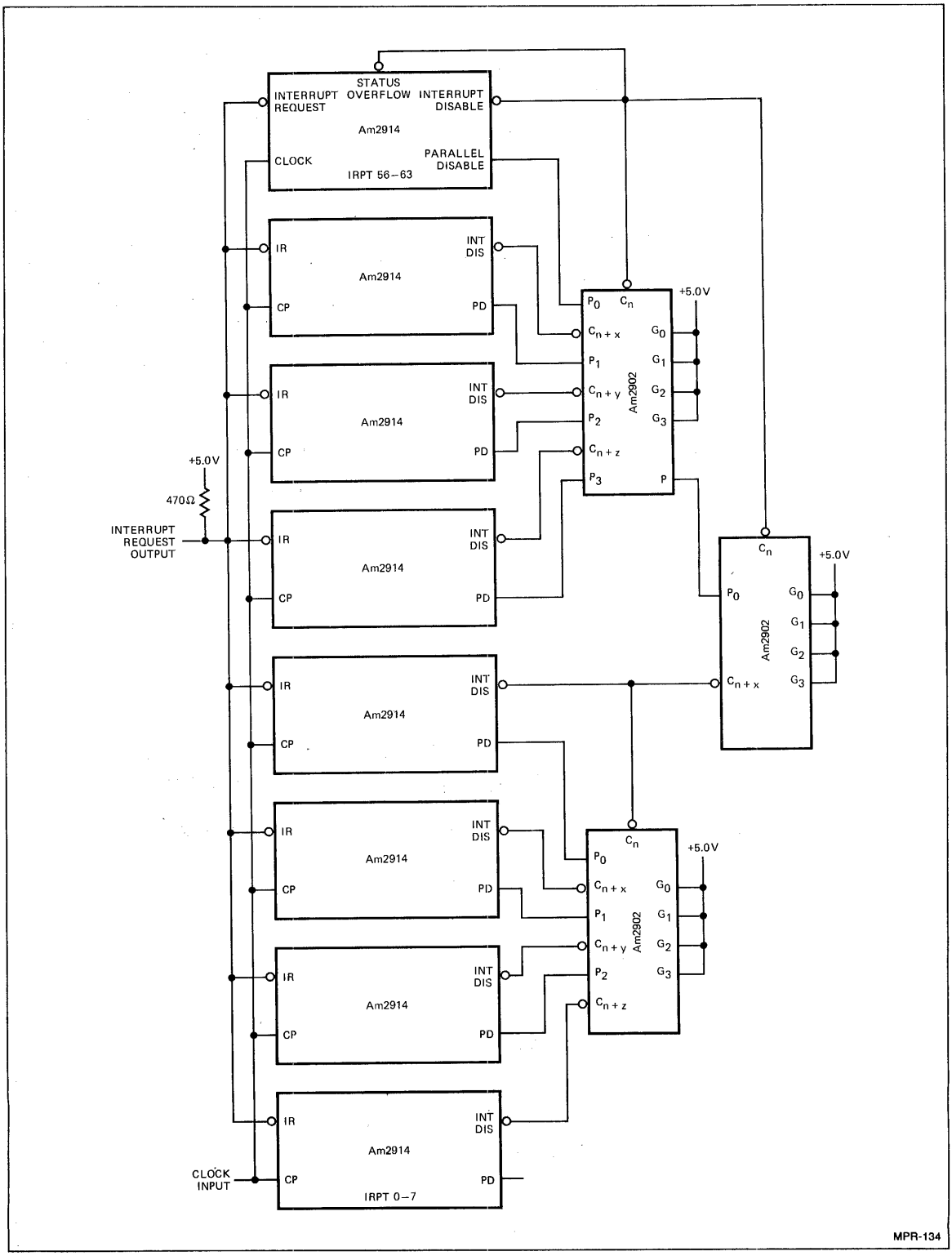


Figure 7. Interrupt Disable Connections for Parallel Cascade Mode.

Ripple Disable output is tied to the Interrupt Disable input of the next lower priority group (see Figure 6).

Parallel Disable (PD) – This output is used only in the parallel cascade mode (see below). It is HIGH when the Lowest Group Enabled flip-flop is LOW or an Interrupt Request is generated in the group. It is not affected by the Interrupt Disable input.

A single Am2914 chip may be used to prioritize and encode up to eight interrupt inputs. Figure 5 shows how the above cascade lines should be connected in such a single chip system.

The Group Advance Receive and Group Enable inputs should be connected to ground so that the Lowest Group Enabled flip-flop is forced LOW during a *Master Clear* or *Load Status* microinstruction. Status Overflow should be connected to Interrupt Disable in order to disable interrupts when vector seven is read. The Group Advance Send, Ripple Disable, Group Signal and Parallel Disable pins should be left open.

The Am2914 may be cascaded in either a Ripple Cascade Mode or a Parallel Cascade Mode. In the Ripple Cascade Mode, the Interrupt Disable signal, which disables lower priority interrupts, is allowed to ripple through lower priority groups. Figures 6, 9 and 11 show the cascade connections required for a ripple cascade 64 input interrupt system.

In the parallel cascade mode, a parallel lookahead scheme is employed using the high-speed Am2902 Lookahead Carry Generator. Figures 7, 9 and 10 show the cascade connections required for a parallel cascade 64-input interrupt system. For this application, the Am2902 is used as a lookahead interrupt disable generator. A Parallel Disable output from any group results in the disabling of all lower priority groups in parallel. Figure 8 shows the Am2902 logic diagram and equations.

In Figures 9 and 10, the Am2913 Priority Interrupt Expander is shown forming the high order bits of the vector and status, respectively. The Am2913 is an eight-line to three-line priority encoder with three-state outputs which are enabled by the five output control signals G1, G2, $\overline{G3}$, $\overline{G4}$, and $\overline{G5}$. In Figure 9, the Am2913 is connected so that its outputs are enabled during a Read Vector instruction, and in Figure 10 the Am2913 is connected so that its outputs are enabled during a Read Status instruction. The Am2913 logic diagram and truth table are shown in Figure 11.

The Am25LS138 three-line to eight-line Decoder also is shown in Figure 10. It is used to decode the three high order status bits during a Load Status instruction. The Am25LS138 logic diagram and truth table are shown in Figure 12.

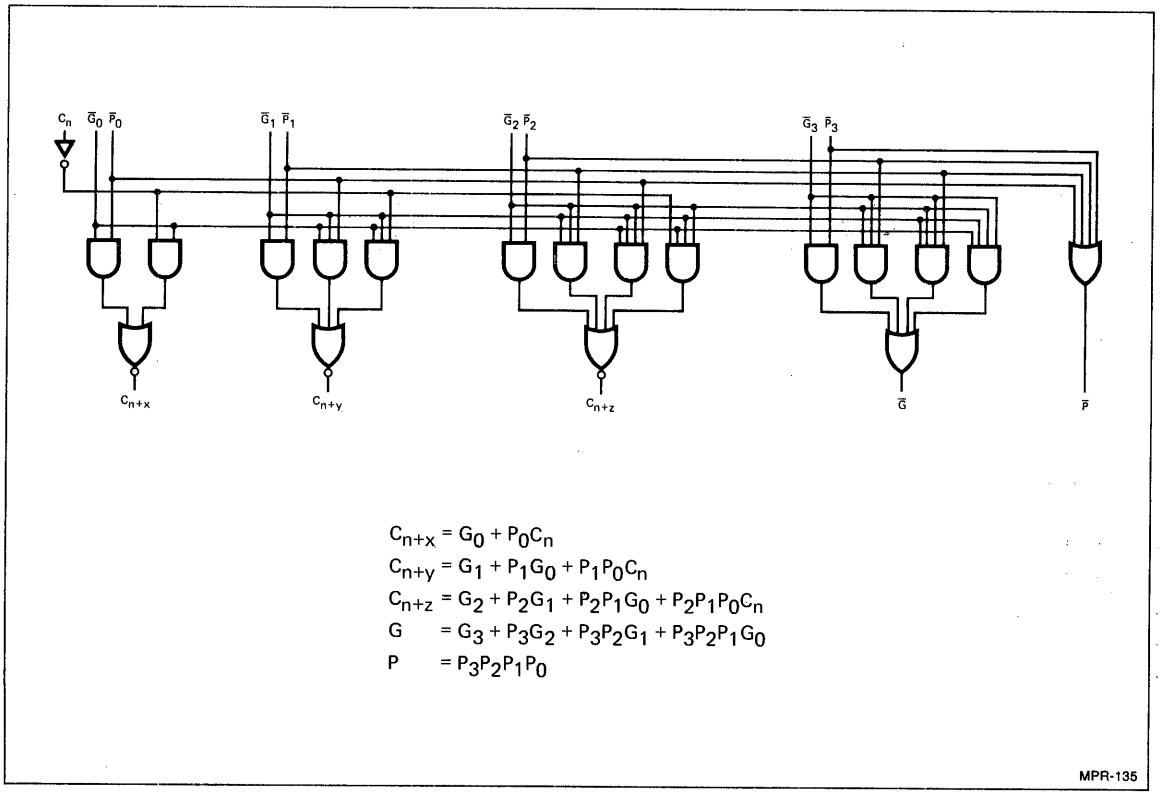


Figure 8. Am2902 Carry Look-Ahead Generator Logic Diagram and Equations.

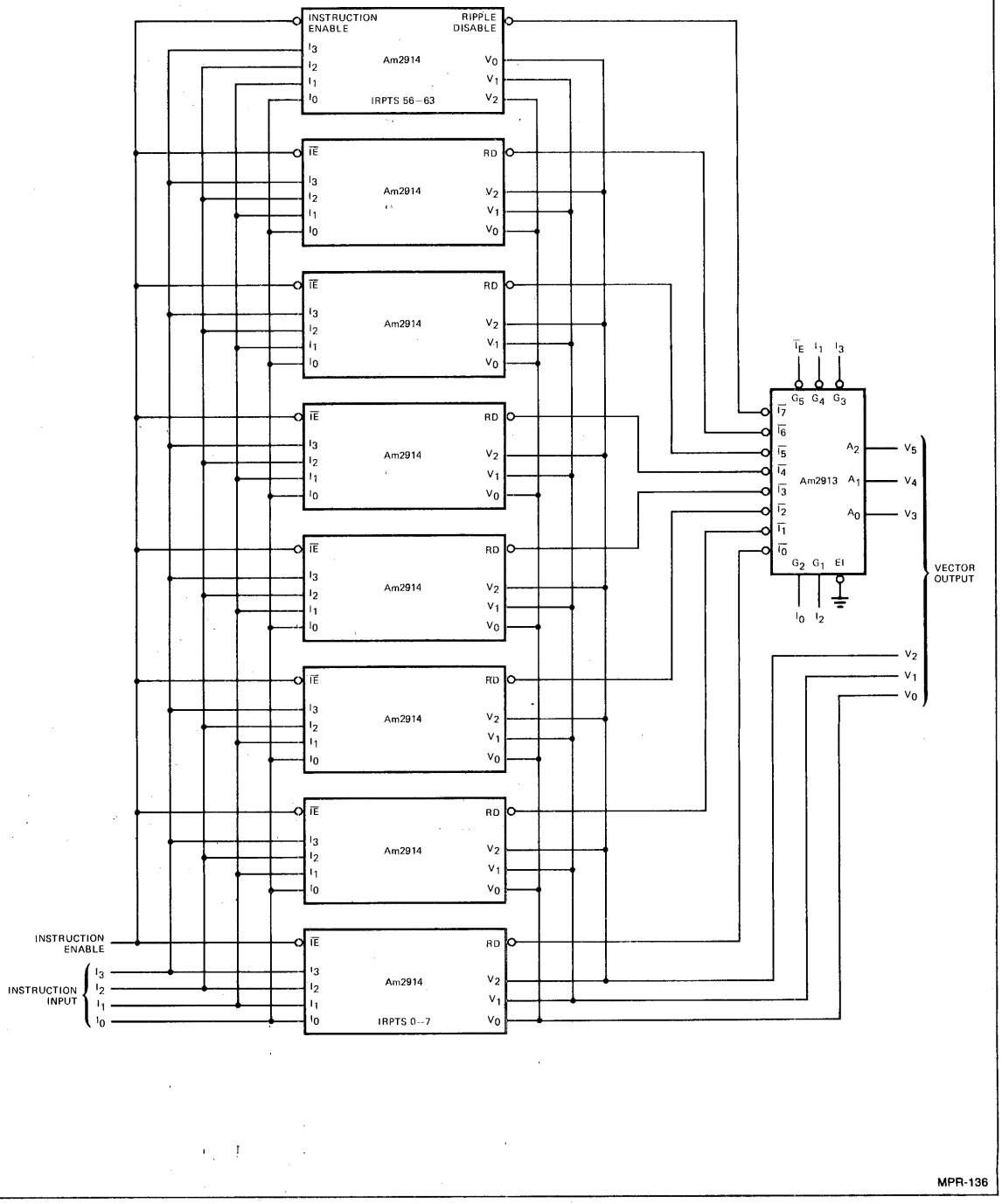


Figure 9. Vector Connections for both the Parallel and Ripple Cascade Modes.

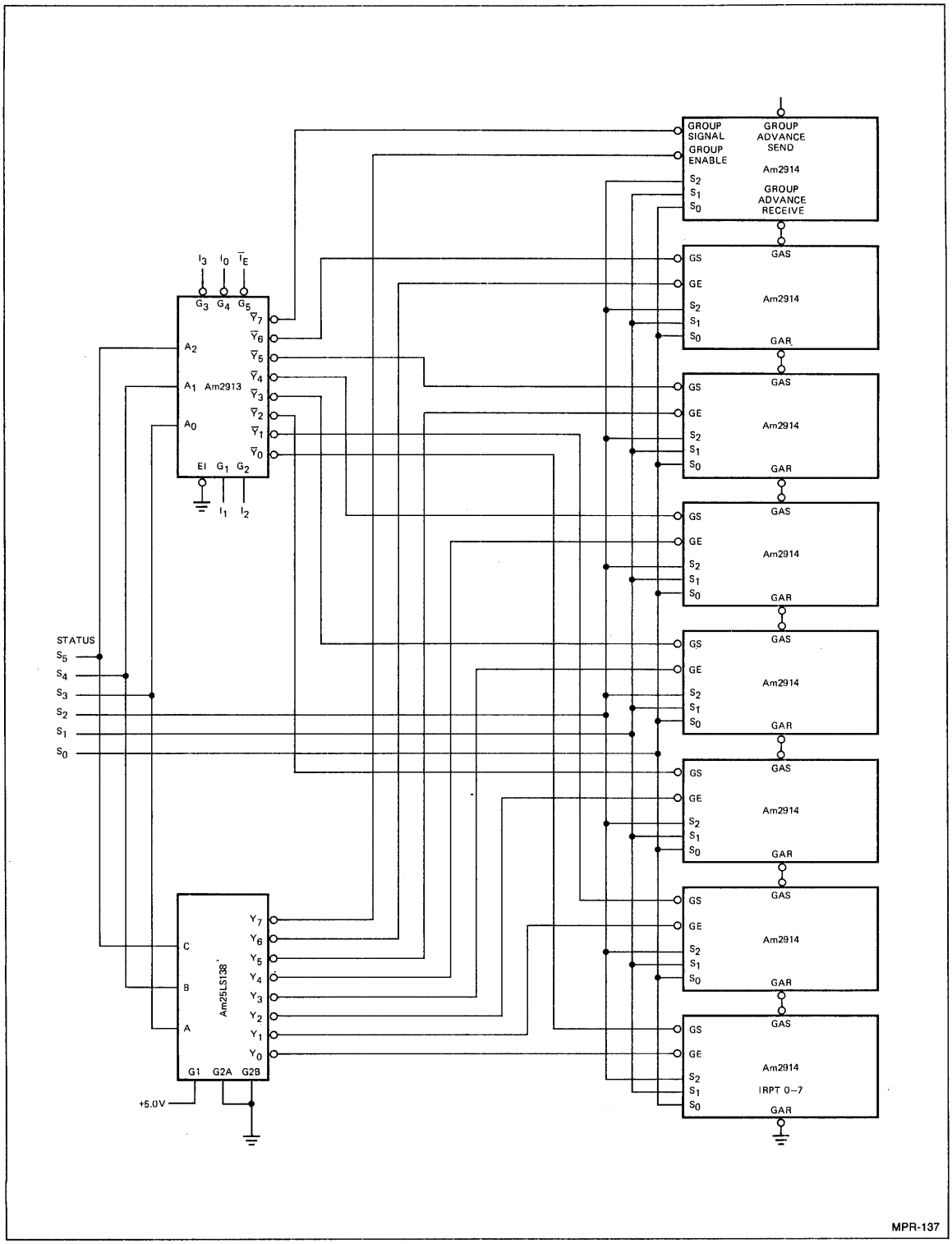
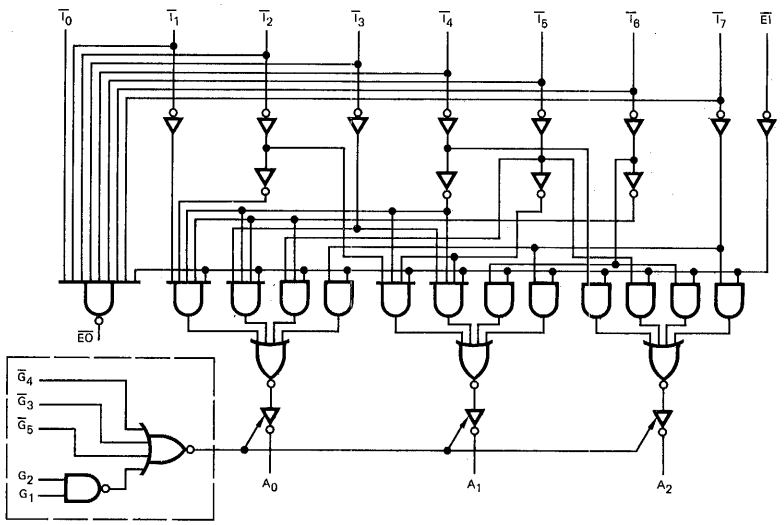


Figure 10. Group Signal, Group Enable, Group Advance Send, Group Advance Receive and Status Connections for Both the Parallel and Ripple Cascade Modes.



Inputs								Outputs			
$\overline{EI}$	$\overline{I}_0$	$\overline{I}_1$	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	A_0	A_1	$\overline{A}_2$	$\overline{EO}$
H	X	X	X	X	X	X	X	X	L	L	L	H
L	H	H	H	H	H	H	H	H	L	L	L	L
L	X	X	X	X	X	X	X	L	H	H	H	H
L	X	X	X	X	X	X	L	H	L	H	H	H
L	X	X	X	X	X	L	H	H	H	L	H	H
L	X	X	X	X	L	H	H	H	L	L	H	H
L	X	X	X	L	H	H	H	H	H	H	L	H
L	X	X	L	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	L	L	L	H

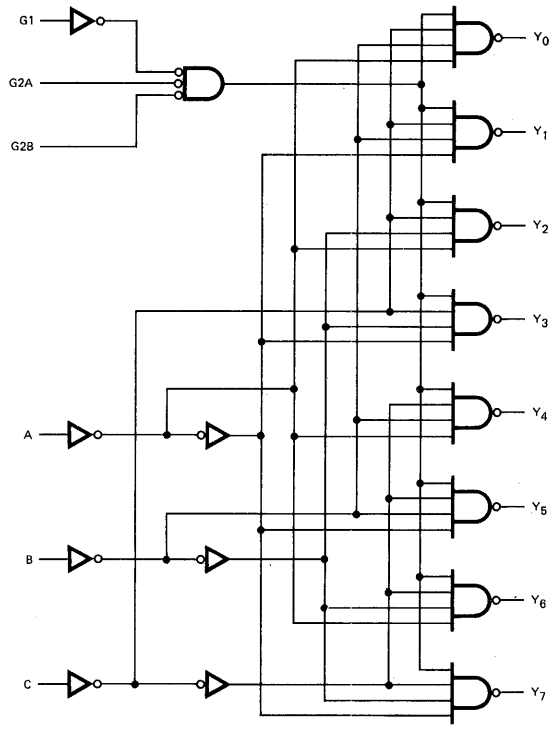
H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care
For G1 = H, G2 = H,
G3 = L, G4 = L, G5 = L

G1	G2	$\overline{G3}$	$\overline{G4}$	$\overline{G5}$	A_0	A_1	A_2
H	H	L	L	L	Enabled		
$\overline{L}$	X	X	X	X	Z	Z	Z
X	L	X	X	X	Z	Z	Z
X	X	H	X	X	Z	Z	Z
X	X	X	H	X	Z	Z	Z
X	X	X	X	H	Z	Z	Z

Z = HIGH Impedance

MPR-138

Figure 11. Am2913 Priority Interrupt Expander Logic Diagram and Truth Table.



Inputs						Outputs							
Enable			Select										
G1	G2A	G2B	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
L	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

H = HIGH
L = LOW
X = Don't Care

MPR-139

Figure 12. Am25LS138 3 to 8 Line Decoder Logic Diagram and Truth Table.

EXAMPLE INTERRUPT SYSTEMS DESIGNS FOR AN Am2900 SYSTEM

A classical computer architecture is shown in Figure 13. The Computer Control Unit controls the internal busses and subsystems of the processor, synchronizes internal and external events and grants or denies permission to external systems. The data bus is commonly used by all of the subsystems in the computer. Information, instructions, address operands, data and sometimes control signals are transmitted down the data bus under control of a microprogram. The microprogram selects the source of the data as well as the destination(s) of the data. The Address Bus is typically used to select a word in memory for an internal computer function or to select an input/output port for an external subsystem or peripheral function. The source of the data for the address bus, also selected by microprogram commands, may be the program counter, the memory address register, a direct memory address controller, an interface controller, etc.

The arithmetic/logic unit (ALU) is that portion of the processor that computes. Under control of the microprogram, the ALU performs a number of different arithmetic and logic functions on data in the working registers or from the data bus. The ALU also provides a set of condition codes as a result of the current arithmetic or logic operation. The condition codes, along with other computer status information, are stored in a register for later use by the programmer or computer control unit.

The program counter and the memory address register are the two main sources of memory word and I/O address select data on the address bus. The program counter contains the address of the next instruction or instruction operand that is to be fetched from main memory, and the memory address register contains instruction address operands which are necessary to fetch the data required for the execution of the current instruction.

A subroutine address stack is provided to allow the return address linkage to be handled easily when exiting a subroutine. The address stack is a last-in, first-out stack that is controlled by a jump-to-subroutine, PUSH, or a return-from-subroutine, POP, instruction from the CCU microinstruction word.

The next microprogram address control (NMAC) circuitry controls the generation of microinstruction addresses. Based on microprogram control, interrupt requests, test conditions and commands from a control panel or other processor, the NMAC determines the address of the next microinstruction to be executed.

For a more detailed description of the above portions of the computer, refer to Advanced Micro Devices' Application Note *A Microprogrammed 16 Bit Computer* by James R.W. Clymer.

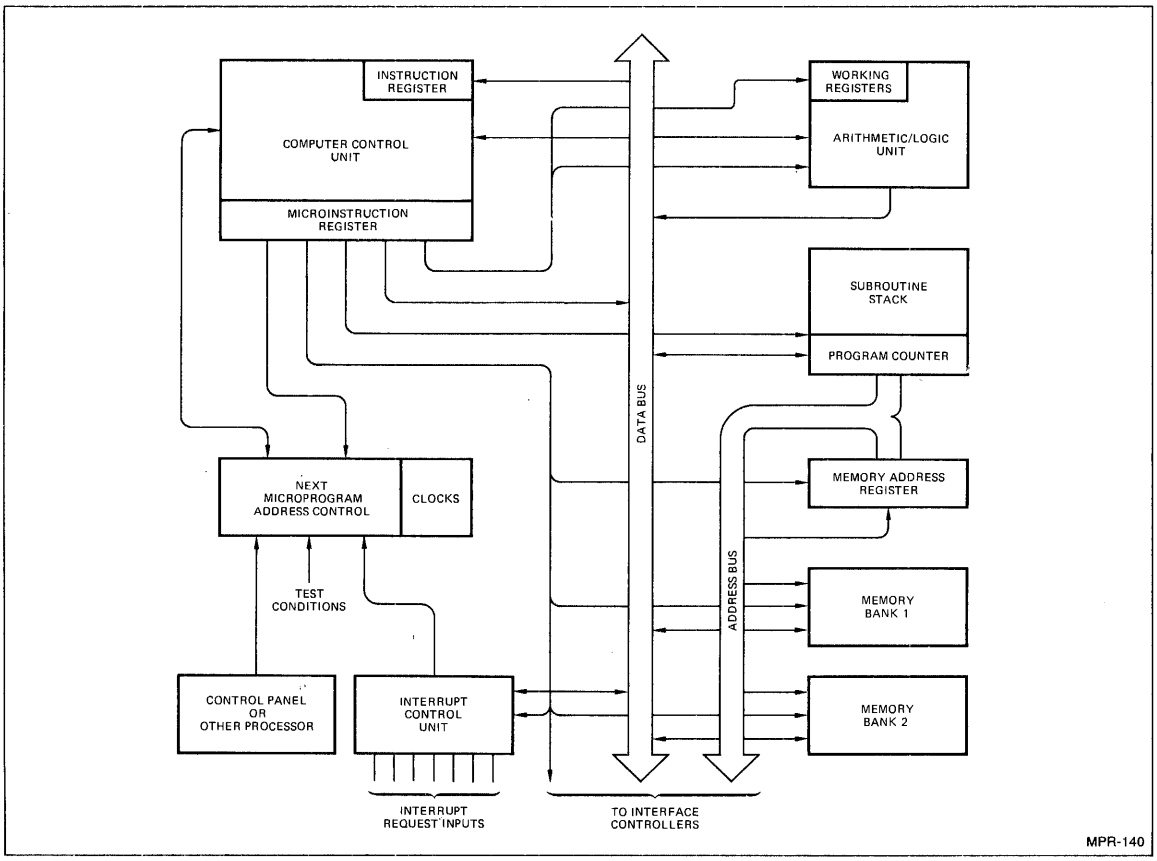


Figure 13. Generalized Computer Architecture.

Am2914 PRIORITY INTERRUPT ENCODER DETAILED LOGIC DESCRIPTION

INTRODUCTION

A clear understanding of the Am2914 Priority Interrupt controller's operation facilitates its efficient use. With that idea in mind, a detailed logic description of the Am2914 is presented here. A detailed logic diagram and control signal truth table are shown, and significant aspects of the Am2914 design are described verbally.

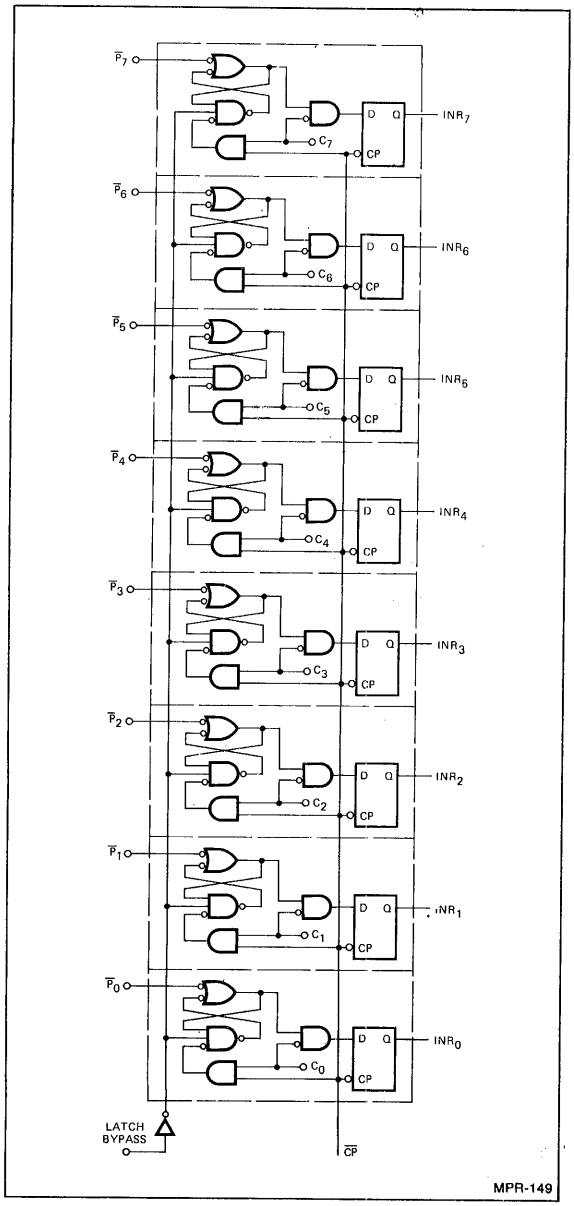


Figure 1. Interrupt Latches and Register.

LOGIC DIAGRAM DESCRIPTION

The Interrupt Latches and Register are shown in Figure 1. The Interrupt latches are set/reset-type latches. When the Latch Bypass signal is LOW, the latches are enabled and act as negative pulse catchers on the inputs to the Interrupt Register. When the Latch Bypass signal is HIGH, the Interrupt latches are transparent. The Interrupt Register holds the Interrupt Inputs and is an eight-bit, edge-triggered register. It is updated on the LOW-to-HIGH transition of the clock pulse (HIGH-to-LOW transition of the $\overline{CP}$ signal) as are all of the flip-flops on the chip.

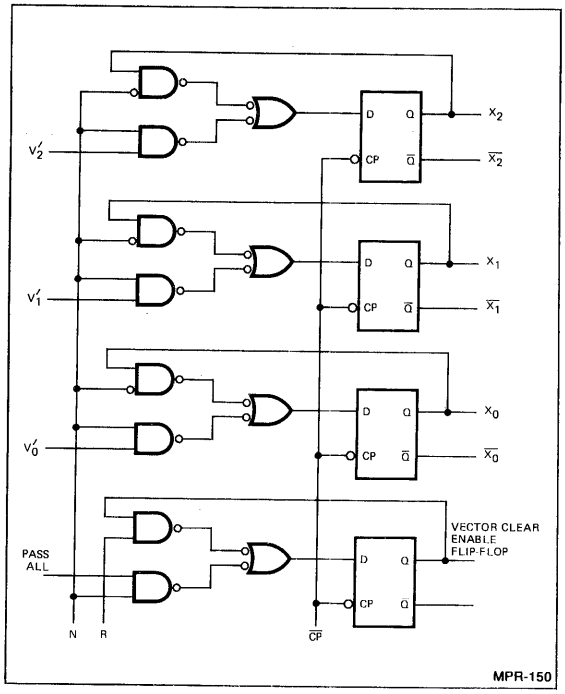


Figure 2. Vector Hold Register

When a Read Vector instruction is executed, the binary coded vector is loaded into the Vector Hold Register of Figure 2. This stored vector can be used later for clearing the interrupt associated with the last vector that was read. The Vector Clear Enable Flip-Flop of Figure 2 is set when a Read Vector instruction is executed and the PASS ALL signal is HIGH. A HIGH PASS ALL signal level indicates that this group is enabled and that an interrupt request in this group was detected and passed priority. The Vector Hold Register and the Vector Clear Enable Flip-Flop are cleared when a Master Clear, Clear All Interrupts, or Clear Interrupt Last Vector Read is executed. Table 1 shows the generation of the "N and R" control signals for each of these operations.

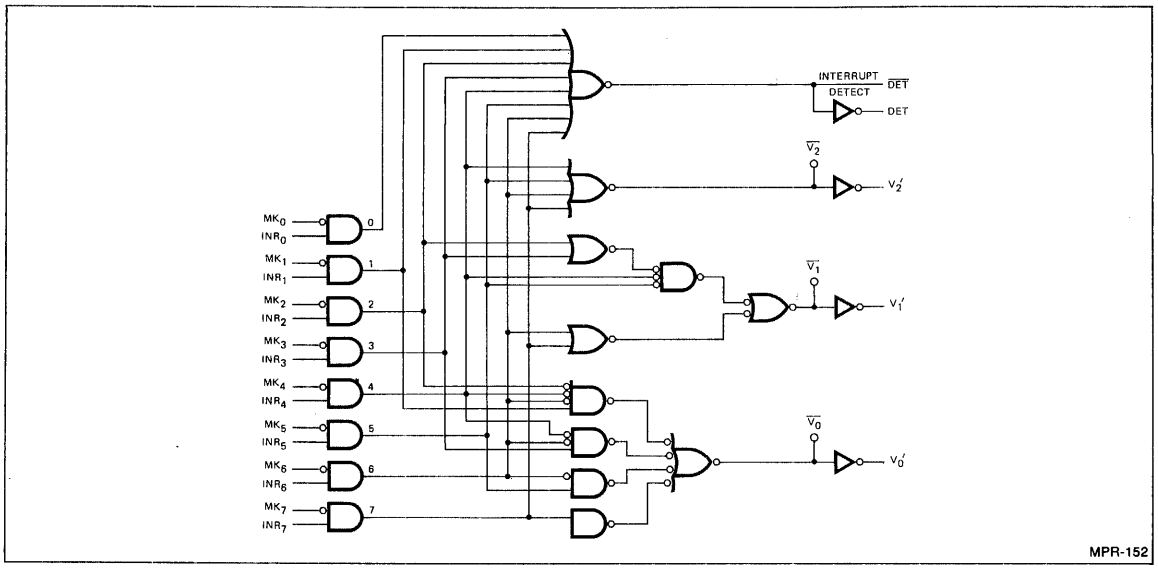


Figure 4. Interrupt Request Detect and Priority Decoder.

The Interrupt Request Detect and Priority Encode circuitry are shown in Figure 4. The Interrupt Detect circuitry detects the presence of any unmasked Interrupt Input. The eight-input Priority Encoder determines the highest priority, non-masked Interrupt Input and forms a binary coded interrupt vector, V_0-V_2.

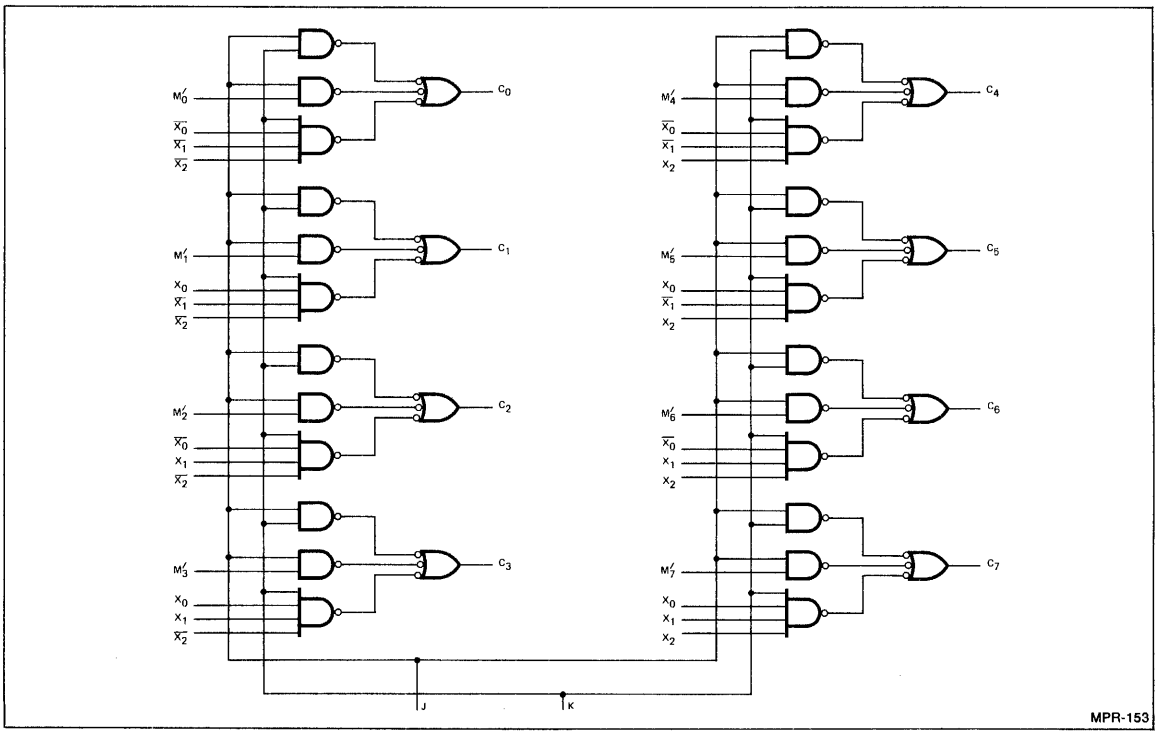


Figure 5. Clear Control.

The Clear Control logic of Figure 5 generates the eight individual clear signals for the eight Interrupt Register bits. Under microinstruction control, all interrupts, interrupts with corresponding mask register bits set, interrupts with corresponding mask bus bits equal to one, or the interrupt associated with the last vector read may be cleared. Table 1 shows the generation of the "J" and "K" control signals for each of these operations.

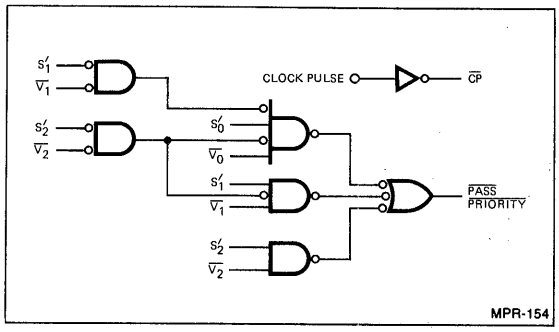


Figure 6. Three-Bit comparator.

The three-bit Comparator of Figure 6 compares the interrupt vector with the contents of the Status Register. A LOW signal level at the PASS PRIORITY output indicates that the interrupt vector is greater than or equal to the contents of the Status Register.

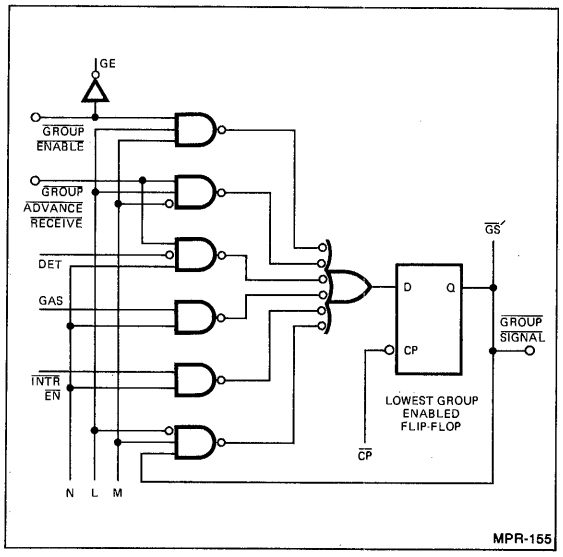


Figure 7. Group Enable Logic.

The Lowest Group Enabled Flip-Flop, Figure 7, is used when a number of Am2914's are cascaded. In a cascaded system, only one Lowest Group Enabled Flip-Flop is LOW at a time. It indicates the group which contains the lowest priority interrupt which will be accepted and is used to form the high order status bits. When a Load Status instruction is executed, the flip-flop is loaded from the GROUP ENABLE input. When a Master Clear instruction is executed, it is loaded from the GROUP ADVANCE RECEIVE input. The flip-flop is set HIGH when a Read Vector instruction is executed if a Group Advance is not received and no interrupt in this group is detected, if a Group Advance is sent from this group, or if interrupts from this group are disabled. For all other instructions, the flip-flop remains the same. Table 1 shows the generation of the "N", "L" and "M" control signals for these operations.

The Status Register holds the status bits and may be loaded from or read to the "S" bus as shown in Figure 8. Note that when a Load Status instruction is executed, status from the "S" bus is loaded into the Status Register only if the GROUP ENABLE input is LOW; if the GROUP ENABLE input is HIGH, the Status Register is cleared. Also note that during a Read Status instruction, the Status Register outputs are enabled onto the "S" bus only if the Lowest Group Enabled Flip-Flop of this group is LOW. When a Read Vector instruction is executed, the incrementer increases the vector by one and the result is loaded into the Status Register. Thus, the Status Register always points to the lowest level at which an interrupt will be accepted. Table 1 shows the generation of the "F", "G" and "OE-S" control signals for Status Register operations.

The Interrupt Request Logic, shown in Figure 9, generates the RIPPLE DISABLE, PARALLEL DISABLE, INTERRUPT REQUEST, GROUP ADVANCE SEND, and STATUS OVERFLOW output signals. The PARALLEL DISABLE signal is generated when the Lowest Group Enabled signal is LOW or an interrupt request in this group is detected and passes priority. The RIPPLE DISABLE signal is generated when the PARALLEL DISABLE signal is generated and also when the INTERRUPT DISABLE input signal is LOW. The INTERRUPT REQUEST output signal is generated when interrupt requests in this group are enabled and a request is detected and passes priority. The GROUP ADVANCE SEND output signal is generated when a vector of value seven is being read. The Status Overflow Flip-Flop is set LOW when a vector of value seven is read and indicates the Status Register has overflowed. The Interrupt Request Enable Flip-Flop is either set or reset by the Enable Request or Disable Request microinstructions respectively, and is used to enable or disable the INTERRUPT REQUEST output. Table 1 shows the generation of control signals "D", "E", "S" and "H".

Note that the vector outputs are enabled only when a Read Vector is being executed. Also note that when a Read Vector instruction is executed, the vector outputs will be disabled after the execution of the instruction since the Status Register is loaded with V+1, and the INTERRUPT REQUEST will no longer be generated.

The Microinstruction Decode circuitry, Figure 10, decodes the Am2914 microinstructions and generates the required internal control signals. Table 1 shows the truth table for these functions and Figure 11 shows the function tables.

Table 1. Am2914 Control Signal Truth Table.
0 = LOW, 1 = HIGH

Microinstruction						Function	Mask Register				Status Register			Group Enable		Clear Control		Irpt Request Enable		Vector Hold Register		Other	
Decimal	$\overline{I_E}$	I_3	I_2	I_1	I_0	Description	A	B	C	OE-M	F	G	$\overline{OE\text{-}S}$	L	M	J	K	D	E	N	R	S	H
0	0	0	0	0	0	Master Clear	0	0	1	0	0	0	1	1	0	1	1	0	1	0	0	1	1
1	0	0	0	0	1	Clear All Interrupts	1	0	1	0	0	1	1	0	1	1	1	1	X	0	0	1	0
2	0	0	0	1	0	Clear Intr Via M Bus	1	0	1	0	0	1	1	0	1	1	0	1	X	0	1	1	0
3	0	0	0	1	1	Clear Intr Via M Reg	1	0	1	1	0	1	1	0	1	1	0	1	X	0	1	1	0
4	0	0	1	0	0	Clear Intr, Last Vector	1	0	1	0	0	1	1	0	1	0	1/0	1	X	0	0	1	0
5	0	0	1	0	1	Read Vector	1	0	1	0	0/1	0	1	0	0	0	0	1	X	1	0	0	1
6	0	0	1	1	0	Read Status Reg	1	0	1	0	0	1	0	0	1	0	0	1	X	0	1	1	0
7	0	0	1	1	1	Read Mask Reg	1	0	1	1	0	1	1	0	1	0	0	1	X	0	1	1	0
8	0	1	0	0	0	Set Mask Reg	0	0	0	0	0	1	1	0	1	0	0	1	X	0	1	1	0
9	0	1	0	0	1	Load Status Reg	1	0	1	0	1	1	1	1	1	0	0	1	X	0	1	1	1
10	0	1	0	1	0	Bit Clear Mask Reg	0	1	0	0	0	1	1	0	1	0	0	1	X	0	1	1	0
11	0	1	0	1	1	Bit Set Mask Reg	1	1	1	0	0	1	1	0	1	0	0	1	X	0	1	1	0
12	0	1	1	0	0	Clear Mask Reg	0	0	1	0	0	1	1	0	1	0	0	1	X	0	1	1	0
13	0	1	1	0	1	Disable Request	1	0	1	0	0	1	1	0	1	0	0	0	0	0	1	1	0
14	0	1	1	1	0	Load Mask Reg	0	1	1	0	0	1	1	0	1	0	0	1	X	0	1	1	0
15	0	1	1	1	1	Enable Request	1	0	1	0	0	1	1	0	1	0	0	0	1	0	1	1	0
X	1	X	X	X	X	Instruction Disable	1	0	1	0	0	1	1	0	1	0	0	1	X	0	1	1	0

Notes: 1. Control line "F" during "READ VECTOR" instruction is 0 when "PASS ALL" is LOW and 1 when "PASS ALL" is HIGH.
2. Control line "K" during "Clear Intr, Last Vector" instruction is 0 when "Vector Clear Enable" is LOW and 1 when "Vector Clear Enable" is HIGH.

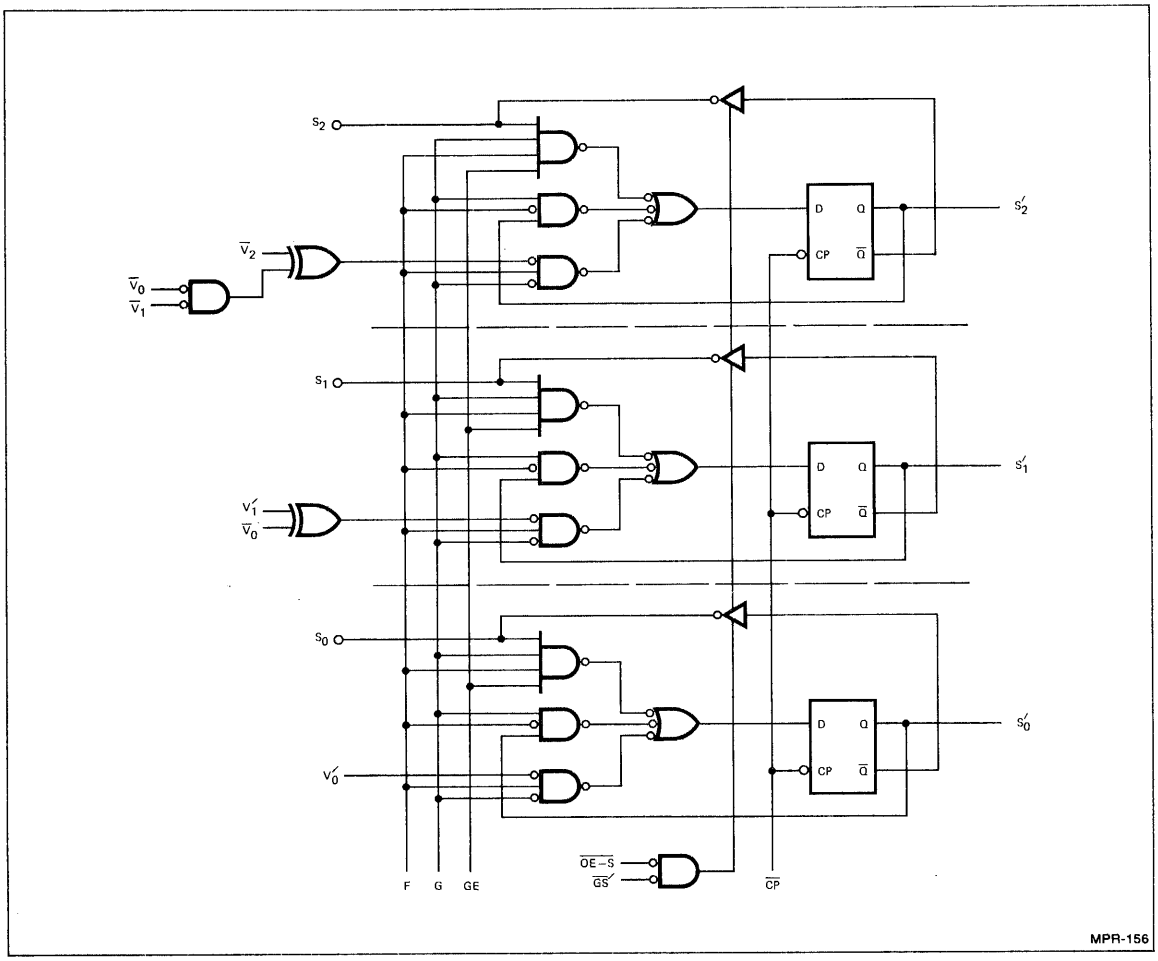


Figure 8. Incrementer and Status Register.

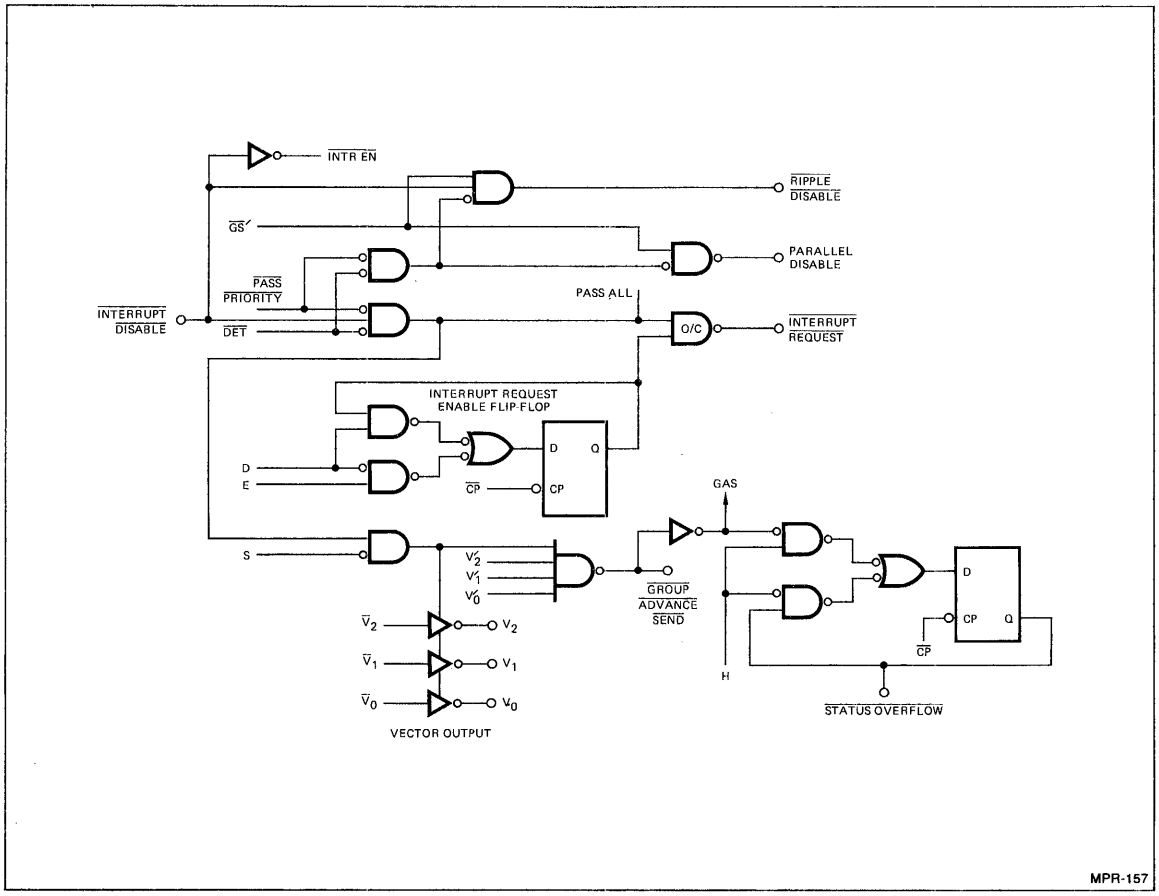


Figure 9. Interrupt Request Logic.

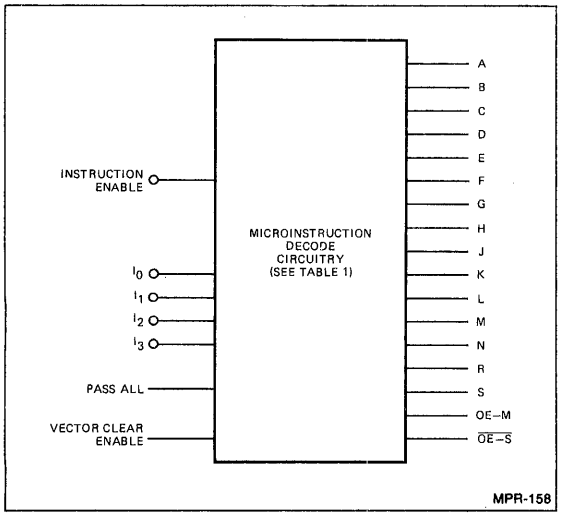


Figure 10.

MASK REGISTER

A	B	C	FUNCTION
0	0	0	SET
0	0	1	CLEAR
0	1	0	BIT CLEAR
0	1	1	LOAD
1	0	1	HOLD
1	1	1	BIT SET

CLEAR CONTROL

J	K	FUNCTION
0	0	NO CLEAR
0	1	CLEAR IRPT, VECTOR
1	0	CLEAR IRPTS VIA M
1	1	CLEAR ALL IRPTS

VECTOR HOLD REGISTER

N	FUNCTION
0	HOLD
1	LOAD

LOWEST GROUP ENABLED FLIP-FLOP

L	M	FUNCTION
0	0	UPDATE
0	1	HOLD
1	0	LOAD VIA GROUP ADVANCE RECEIVE
1	1	LOAD VIA GROUP ENABLE

STATUS REGISTER

F	G	FUNCTION
0	0	CLEAR
0	1	HOLD
1	0	LOAD VECTOR + 1
1	1	LOAD VIA "S" BUS

INTERRUPT REQUEST ENABLE FLIP-FLOP

D	E	FUNCTION
0	0	DISABLE IRPTS
0	1	ENABLE IRPTS
1	X	HOLD

VECTOR CLEAR ENABLE FLIP-FLOP

N	R	FUNCTION
0	0	CLEAR
0	1	HOLD
1	0	LOAD

STATUS OVERFLOW FLIP-FLOP

H	FUNCTION
0	HOLD
1	LOAD

Figure 11. Control Function Tables.

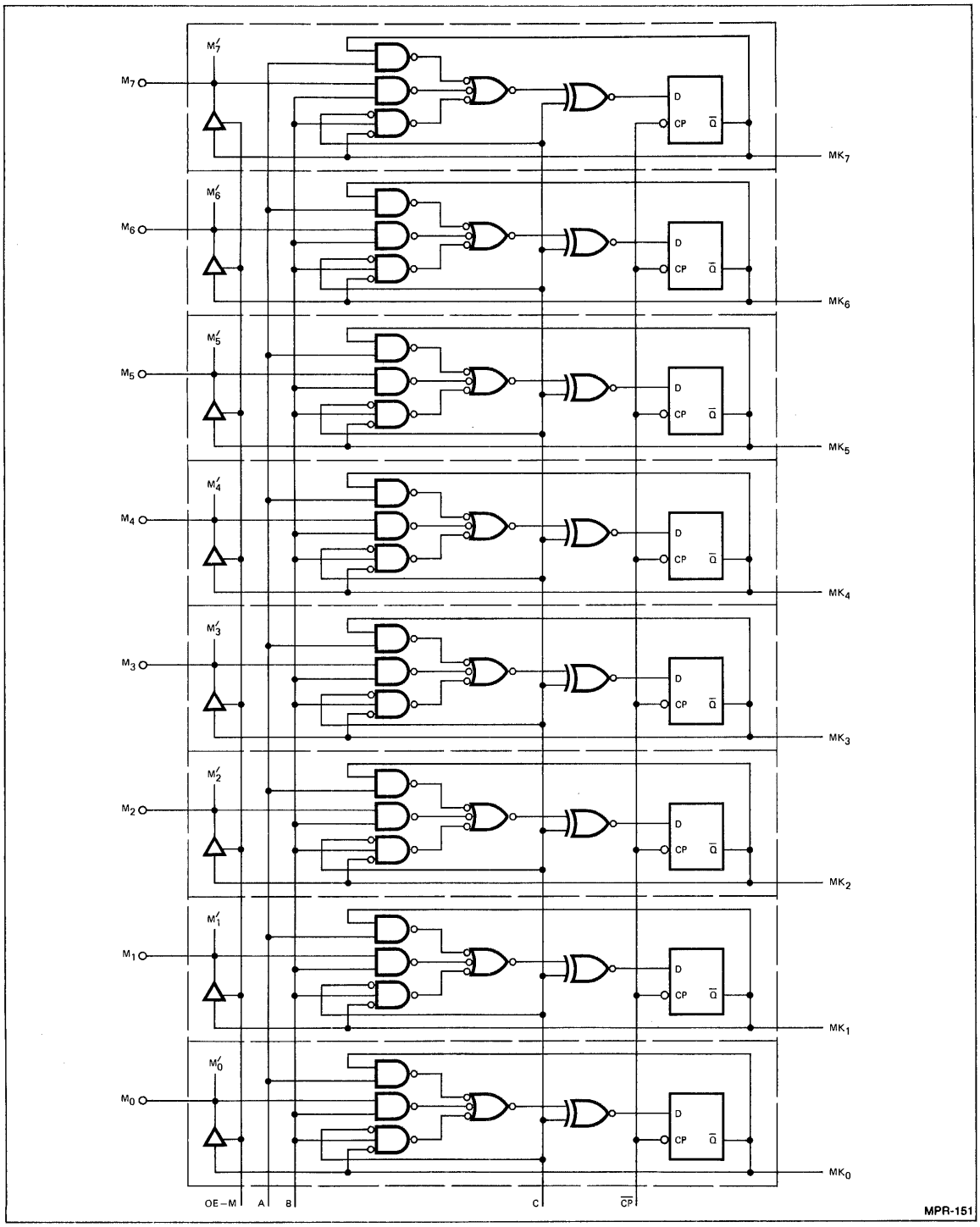


Figure 3. Mask Register.

The Mask Register shown in Figure 3 holds the eight mask bits associated with the eight interrupt levels. The register may be set or cleared, bit set or bit cleared from the "M" bus, or loaded or read to the "M" bus. Table 1 shows the generation of the "A", "B", "C" and "OE-M" control signals for each of these operations.