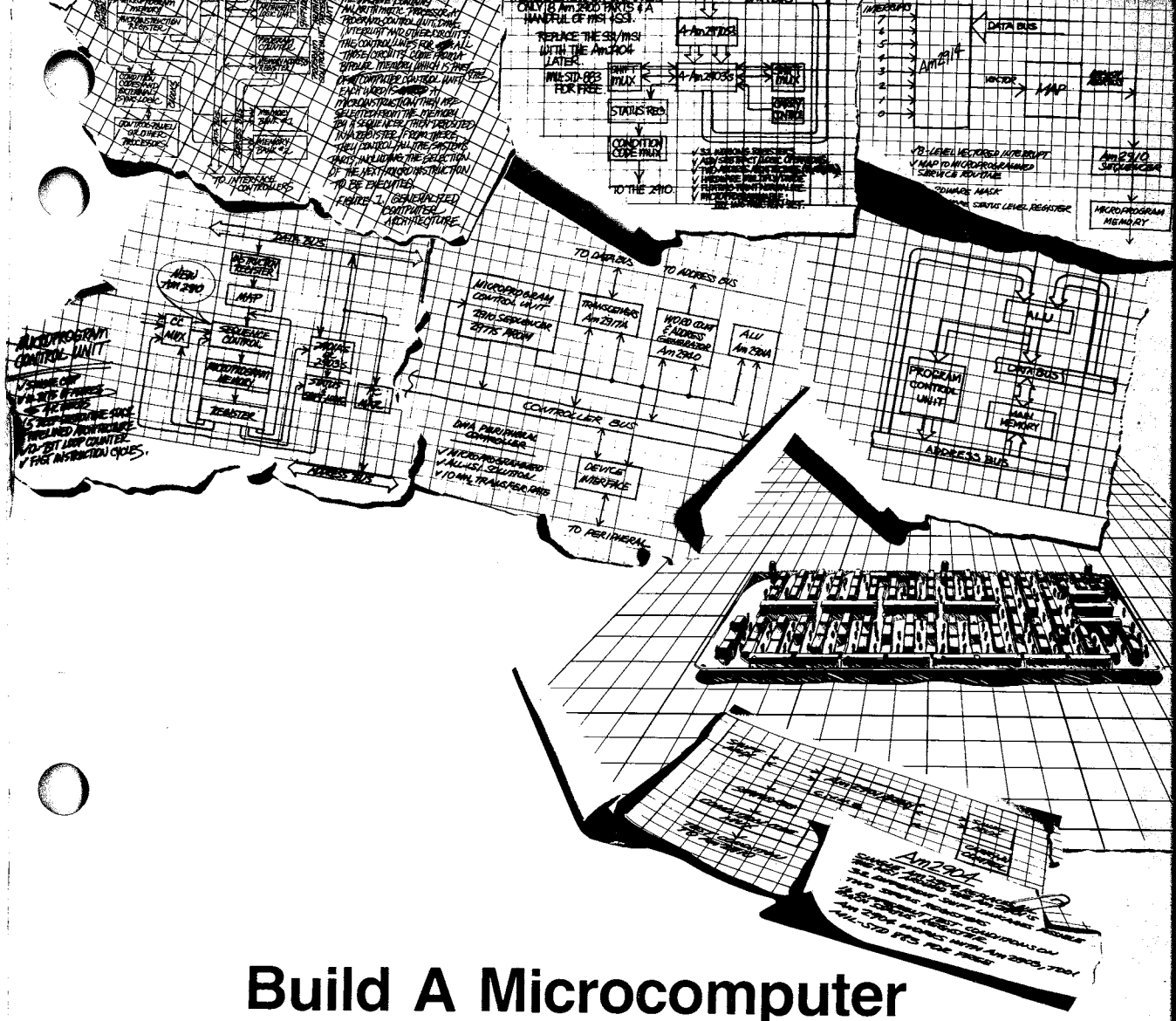


Build A Microcomputer

Chapter IX
Super Sixteen

Advanced Micro Devices

ADVANCED MICRO DEVICES, INC.
901 Thompson Place
Sunnyvale
California 94086
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL FREE
(800) 538-8450

10-79

AM-PUB073-9

INTRODUCTION

The AMD 16-Bit Computer design is an example of a high-speed microprocessor system which takes full advantage of AMD's Am2900 Family of Bipolar microprocessor circuits to provide an economical, high performance, self contained 16-bit computer. It was designed to demonstrate the principles of a microprogrammed system.

This design is intended to show some of the techniques used to achieve high performance. This includes pipelining at the microprogram level as well as pipelining at the macro or machine instruction program level. A powerful instruction set is demonstrated which allows the user to write efficient programs in a minimum amount of time.

One of the unique features of the design is that in addition to using the high performance Am2900 Bipolar microprocessor family, it takes advantage of the MOS peripherals normally associated with MOS microprocessors. These are used to perform the slower functions, particularly in the I/O interface area.

SYSTEM ORGANIZATION

The 16-Bit Computer is designed to perform in a system environment as shown in Figure 1. The system consists of a central processing unit (the 16-Bit Computer), memory units, I/O units (peripheral controllers), and a bus controller. These units communicate over the system bus consisting of a 16-bit wide address bus, 16-bit wide bi-directional data bus, and a control bus. The control bus is a collection of signals that include the memory and I/O interface controls and the interrupt request lines.

This organization allows systems to be configured with more than one CPU and multiple memory and I/O units. The bus controller arbitrates requests for bus use from the CPU's or I/O units that require DMA transfers.

This application note concentrates on the design of the CPU portion of the system.

INSTRUCTIONS

An instruction is either one or two 16-bit words in length and must be located in main memory on an integral word boundary. The left most eight bits of the instruction is always the operation code, followed by two, 4-bit register designation fields (Figure 2). The 16-bit (one word) instruction is always this format. The 32-bit (two words) instruction has the first (left most) word exactly like the 16-bit instruction. The second word of the 32-bit instruction is always full 16-bit value (d) which acts as a memory reference address or an immediate value (Figure 3). This architecturally simple instruction format becomes very powerful when implemented on a microprogrammed machine.

The 8-bit opcode provides for 256 primary instructions, which is usually more than enough for most general purpose computers. The 4-bit register fields (R_1 and R_2) each designate one of the sixteen, 16-bit registers (R_0-R_{15}). Depending upon the operation, each register can act as either an accumulator for arithmetic and logic operations, or an index register in modulo address arithmetic. On operations where the result is placed in a register, the R_1 field depicts the destination register and R_2 (or R_2+d) is, or points to the source field in main memory. On operations where the result is transferred from a register to memory, the R_1 field depicts the source register and R_2 (or R_2+d) points to the destination memory location. Memory to memory transfers will have R_2 as the source pointer and R_1 as the destination pointer. Even though the R_1 and R_2 fields are architecturally wired to the Am2903 register address inputs, variations of the source/destination assignment may be implemented via microcode.

The complete defined standard instruction set is given in Table 1. This is a typical "machine level" instruction set. It allows manipu-

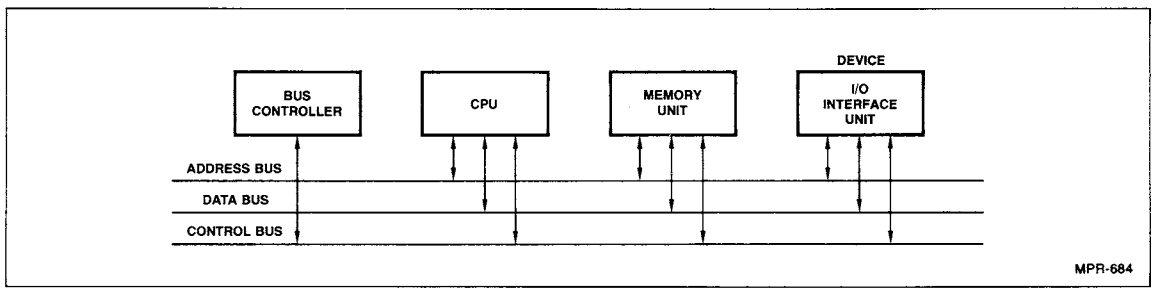


Figure 1. System Organization.

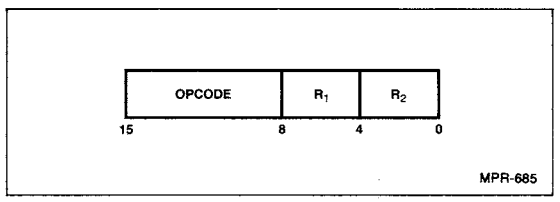


Figure 2. 16-Bit Instruction (RR, RS, SS).

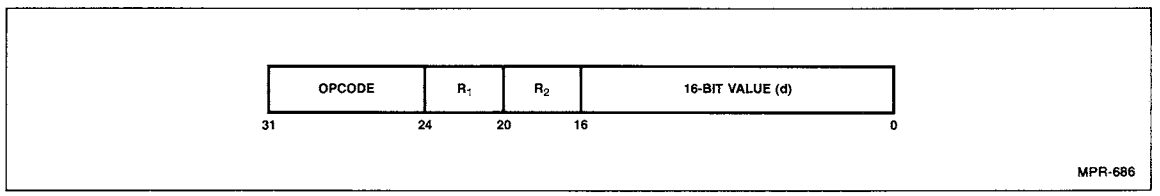


Figure 3. 32-Bit Instruction (RX, RSI).

Table 1. 16-Bit Computer Instruction Summary Mnemonic Instruction Format.

FIXED-POINT LOAD/STORE INSTRUCTIONS		
LD	LOAD	RR, RS, SS, RX, RSI
ST	STORE	RS, RX
FIXED-POINT ARITHMETIC INSTRUCTIONS		
ADD	ADD	RR, RS, SS, RX, RSI
ADC	ADD WITH CARRY	RR, RX
SUB	SUBTRACT	RR, RS, SS, RX, RSI
SBC	SUBTRACT WITH CARRY	RR, RX
AND	AND	RR, RS, SS, RX, RSI
OR	OR	RR, RS, SS, RX, RSI
XOR	XOR	RR, RS, SS, RX, RSI
TSTI	TEST IMMEDIATE	RSI
CMP	COMPARE	RR, RS, SS, RX, RSI
CMPL	COMPARE LOGICAL	RR, RS, SS, RX, RSI
MUL	MULTIPLY	RR, RX
MULU	MULTIPLY UNSIGNED	RR, RX
DIV	DIVIDE	RR, RX
COMP	ONES COMPLEMENT	RR, RS, SS, RX, RSI
BYTE INSTRUCTIONS		
LDB	LOAD BYTE	RR, RX, RSI
IC	INSERT CHARACTER	RR, RX, RSI
STC	STORE BYTE	RR, RX, RSI
XCHB	EXCHANGE	RR, RX, RSI
BS	BYTE SWAP	RR, RX
CLB	COMPARE LOGICAL BYTE	RR, RS, RX, RSI
ANDB	AND BYTE	RR, RS, RX, RSI
ORB	OR BYTE	RR, RS, RX, RSI
XORB	XOR BYTE	RR, RS, RX, RSI
SYSTEM INSTRUCTIONS		
LPSW	LOAD PROGRAM STATUS WORD	RX
SPSW	STORE PROGRAM STATUS WORD	RX
EPSW	EXCHANGE PROGRAM STATUS WORD	RR
SVC	SUPERVISOR CALL	RX
SETP	SET BIT PSW	RI
RSTP	RESET BIT PSW	RI
TSTP	TEST BIT PSW	RI
CMPP	COMPLEMENT BIT PSW	RI
STACK INSTRUCTIONS		
CALL	BRANCH AND STACK	RR, RX
RTN	RETURN	RR
PUSH	PUSH	RR
POP	POP	RR
PPUSH	PARTIAL PUSH	RR
PPOP	PARTIAL POP	RR
LDSP	LOAD STACK POINTER	RX
LDSLL	LOAD STACK LOWER LIMIT	RX
LDSUL	LOAD STACK UPPER LIMIT	RX
STSP	STORE STACK POINTER	RX
STSLL	STORE STACK LOWER LIMIT	RX
STSUL	STORE STACK UPPER LIMIT	RX
EXTENDED INSTRUCTIONS		
TR	TRANSLATE	RR
TRT	TRANSLATE AND TEST	RR
MVCL	MOVE LONG	RR
CLCL	COMPARE LONG	RR
EXEC	EXECUTE	RX
DA	DECIMAL ADD	RR, RX
DS	DECIMAL SUBTRACT	RR, RX
DI	DECREMENT INDEXES	RR
SHIFT/ROTATE		
SRL	SHIFT RIGHT LOGICAL	RX, RSI
SRA	SHIFT RIGHT ARITHMETIC	RX, RSI
RR	ROTATE RIGHT	RX, RSI
SLL	SHIFT LEFT LOGICAL	RX, RSI
RL	ROTATE LEFT	RX, RSI
SRDL	SHIFT RIGHT DOUBLE LOGICAL	RX, RSI
SRDA	SHIFT RIGHT DOUBLE ARITHMETIC	RX, RSI
SLDL	SHIFT LEFT DOUBLE LOGICAL	RX, RSI
SLDA	SHIFT LEFT DOUBLE ARITHMETIC	RX, RSI
RRD	ROTATE RIGHT DOUBLE	RX, RSI
RLD	ROTATE LEFT DOUBLE	RX, RSI
I/O INSTRUCTIONS		
IN	INPUT WORD	RR, RX
INB	INPUT BYTE	RR, RX
OUT	OUTPUT WORD	RR, RX
OUTB	OUTPUT BYTE	RR, RX
BRANCHES		
B	UNCONDITIONAL BRANCH	RX
BR	UNCONDITIONAL BRANCH REGISTER	RR
BC	BRANCH ON CONDITION TRUE	RX
BAL	BRANCH AND LINK	RX
BALR	BRANCH AND LINK REGISTER	RR
BXH	BRANCH ON INDEX HIGH	RX
BXLE	BRANCH ON INDEX LOW OR EQUAL	RX

lation of bit, byte, word and multibyte data; PUSH/POP single or multiple registers to/from stacks; maintain multiple stacks; decimal, binary and integer arithmetic; byte and word I/O; and maintain supervisory control over hardware and software generated interrupts.

Instruction Format

Many of the instructions have multiple formats. These formats depict addressing modes and determine where the source and destination fields are located. The defined instruction formats are shown in Figure 4.

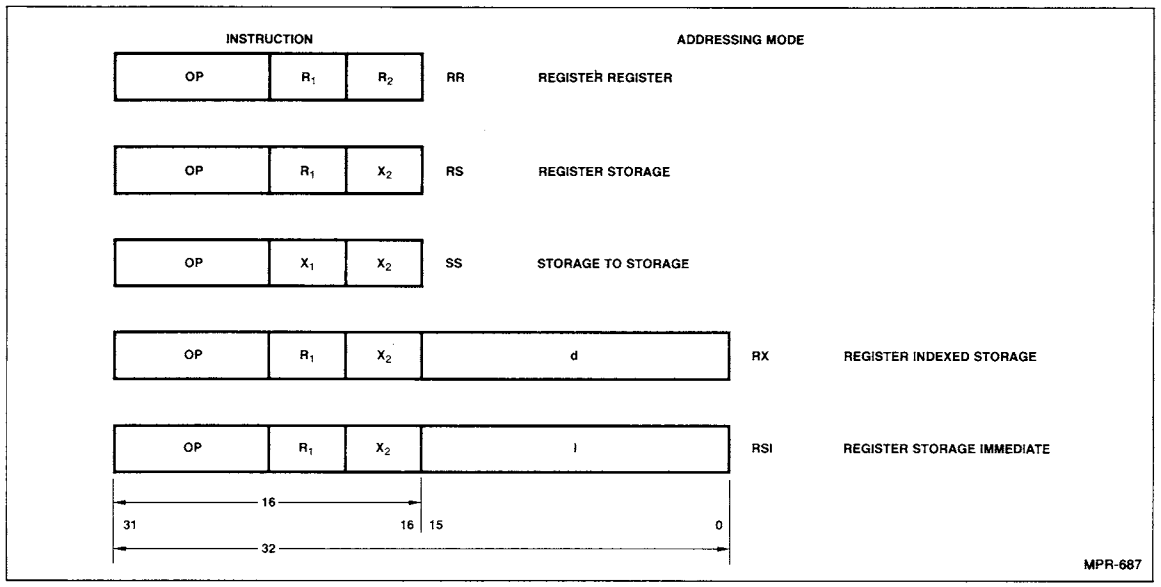


Figure 4. Instruction Formats.

The instructions set consists of nine instruction groups:

- Fixed-point load/store
- Fixed-point arithmetic
- Byte
- Shift/rotate
- Branch control
- I/O
- Stack
- Extended
- System

A complete description of each instruction is given in Appendix A.

CENTRAL PROCESSING UNIT ARCHITECTURE

Processor Organization

The organization of the computer is shown in Figure 5 (Computer Block Diagram). The computer is organized into several distinct sections, the Program Control Unit (PCU), the Arithmetic and Logic Unit (ALU), and the Computer Control Unit (CCU), the Data Path, the Memory Control and Clock Control, and Input/Output Interface and Interrupt Section. The logic diagrams for the CPU are located in Appendix F. Earlier chapters in the Build a Microcomputer series have described the principle sections of a computer and the Am2900 components used in these sections. This chapter describes how these components are used to implement a very high-speed low cost computer.

Note: Figure 5 is sheet 1 of the logic diagrams.

The Program Control Unit

The Program Control Unit (PCU) under control of the microprogram is used to update the Program Counter and load this value into the Memory Address Register (MAR) for reading instructions/data from main memory. The PCU is also used to update the stack pointer and compare this value to the stack limits during stack operations. As can be seen in Figure 5, the Computer Block Diagram, data can be sent to the PCU from the ALU via the Transfer Register. The PCU can also output data onto the PCU bus to the Y-bus of the ALU via the bi-directional PCU transfer drivers.

The PCU is organized around four Am2901's. The use of Am2901's allow the PCU to generate addresses with the flexibility of an ALU chip, to increment the Program Counter by two in one microcycle, and to provide the stack pointer registers for in main memory stack operations. The registers of these Am2901's are defined as shown in Figure 6. Register 0 holds the program counter and Registers 4 and 5 hold constants for incrementing. Byte addressing requires the address to be incremented by two every time 16 bits of instruction data are fetched.

The Arithmetic and Logic Unit (ALU)

The ALU shown in Figure 7 is organized around four Am2903's. The Am2903 performs all of the functions performed by the Am2901A but also provides the computer with separate DA bus and DB bus input ports as well as additional instructions to implement multiplication and division. Three major buses connect to the ALU: DA, DB and Y buses. The memory data from the Z_0 Register and microcode immediates are brought into the Am2903 through the DA port while Program Status Bits 16-23 enter via the DB port. The Am2903's output or receive data on the Y bus for loading into the RAM registers. The Am2903's zero decode logic detects zero on the Y port whether or not the Y port is receiving or sending data.

To implement the defined instruction set, the RAM register selection controls are sent from the Instruction (I) Register to the Am2903's. I_{0-3} (used with instructions with the R_2 or X_2 field) are

Register Number	Register Assignment
0	Program Counter
1	Stack Pointer
2	Stack Lower Limit
3	Stack Upper Limit
4	+ 2
5	+ 4
6	Not used – available
7	Not used – available
8-15	Not used (wired disable)

Figure 6. PCU Register Assignments.

connected to the A address inputs on the Am2903 while I_{4-7} are connected to the B address inputs. The ALU operations performed are controlled by microcode bits M_{78-86} which are connected to the Am2903 I_{0-8} inputs.

The Am2904 provides the microcode and machine status registers holding the carry, negative, zero and overflow status. The machine status bits C, N, Z and OVR are defined as PSW bits 16-23. Logic in the Am2904 includes a condition code multiplexer to select the true or complement of any of the four status bits and combinations of status bits from either the machine or microstatus registers or directly from the ALU. This condition code multiplexer is controlled by Instruction Register bits I_{4-7} which are gated to the Am2904 I_{0-3} inputs during the execution of a conditional branch. The output of the multiplexer, labeled TEST is routed to the test tree for input into the Am2910. The Am2904 also provides the shift linkages and shift linkage control and selection of the type of carry signal to the ALU and lookahead carry unit.

The ALU is designed to work with byte operations as well as 16-bit operations. Byte operations operate only on the lower 8 bits of register data without affecting the upper 8 bits of data. During byte operations the WORD signal (M_{90}) goes inactive disabling the Write Enable and Output Y Enable for ALU bit slices 3 and 4. The word/byte multiplexer circuit will select C, N and OVR status bits from ALU bit slice 2 and at the same time ALU bit slice 2 has its MSS input pulled LOW to indicate most significant slice. The zero status bit being OR tied to all of the ALU bit slices cannot be multiplexed. Instead the Y bus signals 8-15 are forced to zero by gating zeroes from the PCU resulting in the Z signal line state being a function of ALU bit slices 1 and 2 only.

The Computer Control Unit

The Computer Control Unit controls the sequence of execution of the microinstructions. The Am2910 Microprogram Controller provides the sequencer for the microprogram (see logic diagrams Sheet 5). Branch addresses and counter values loaded into the Am2910 D_{0-11} inputs, originate from the Pipeline Register (M_{0-11}), the interrupt vector decoder, and the machine instruction decoder. The instruction decoder, also called Mapping ROM, (a 512 x 8 PROM) uses the Instruction Register I_{8-15} as address bits with the PROM outputs being the starting address of the microcode sequence that executes each machine instruction. In this design the Am29775 Registered PROM's are used to provide both the microprogram memory (512 x 96 bits wide) and the Pipeline Register. The microcode bits M_{16-20} are output from Am29774 because these signals require open collector outputs rather than the standard tri-state outputs to allow the Am2910 inputs I_{0-3} to be pulled to zero.

The starting address generation for the interrupt service routine and initialization routine is accomplished with a minimum of extra logic. During the last microcode cycle of the previous machine instruction, the MAPEN signal is activated to enable the output of the Mapping ROM. However, if an interrupt request is pending, the Mapping ROM is disabled and the pull-up resistors force the eight least significant microprogram branch address lines to all ones, vectoring the microprogram to the interrupt service routine. After a reset, the microprogram should be vectored to address zero, the starting address of the initialization routine. This is accomplished by having the reset signal force zeroes into the Am2910 I_{0-3} inputs which causes the Am2910 to output address zero.

Clock and Memory Control

The architecture of this computer achieves its high throughput by being able to execute machine instructions in as little as one microcycle. This is accomplished by overlapping (also called pipelining) the fetch and decode with the execute microcycles. An essential part of this design is the memory control section. The clock and memory control circuits shown in Sheet 6 of the logic diagrams work together to provide a very efficient mechanism for integrating memory operations with the computer. The memory interface timing is a clocked handshaked protocol shown in Figure 8. Each memory transfer consists of a Bus Request, Bus Acknowledge response, Memory Request, Address Accept response, Data Request and a Data Sync response. At the maximum rate a memory interface response can occur 50ns after the computer activates a control line. This makes it possible to read from main memory once every microcycle (4 x 50ns = 200ns); however should a particular memory board require a longer cycle, it can delay sending Data Sync to the computer to extend the cycle.

The read and write timing are shown in more detail in Figures 9 and 10. Note that if a memory read is taking place during microcycle N, the Bus Request, Bus Acknowledge and the start of memory address are output from the computer in the previous N−1 cycle, and the data is sent to the computer during the first half of the following N+1 cycle. Now consider the case of back-to-back main memory read cycles. In this case, in the microcycle that the computer sends the address to the memory board, the memory board is sending data to the computer; but this is not the data associated with the address being received but the data associated with the address received during the previous microcycle.

A free running or uncontrolled 20MHz clock on the backplane is connected to all of the devices which effect memory transfers (CPU, bus controller, and memory modules). All of the signal handshaking that is required by the memory interface protocol is clocked with the same 20MHz clock to ensure no metastable conditions occur during memory transfer. Careful examination of this memory interface operation will reveal that not only does it solve the very serious metastable problem, but also that the clock synchronization and bus propagation delay occur during the memory read access time (or write time) and do not slow down the memory transfer rate.

The CPU clock generation is intimately related to the Memory Control Logic. The CPU clock signals Phase 1 (ϕ_1) and Phase 2 (ϕ_2) are shown along with the memory interface signals in Figure 8. Phase 1 is a square wave set high at the beginning of the microcycle and has a period of 200ns. Almost all operations of the computer are clocked with the leading edge of ϕ_1. The clock control logic will enable the next cycle only if a Bus Request has received a Bus Acknowledge and only if a Memory Request has

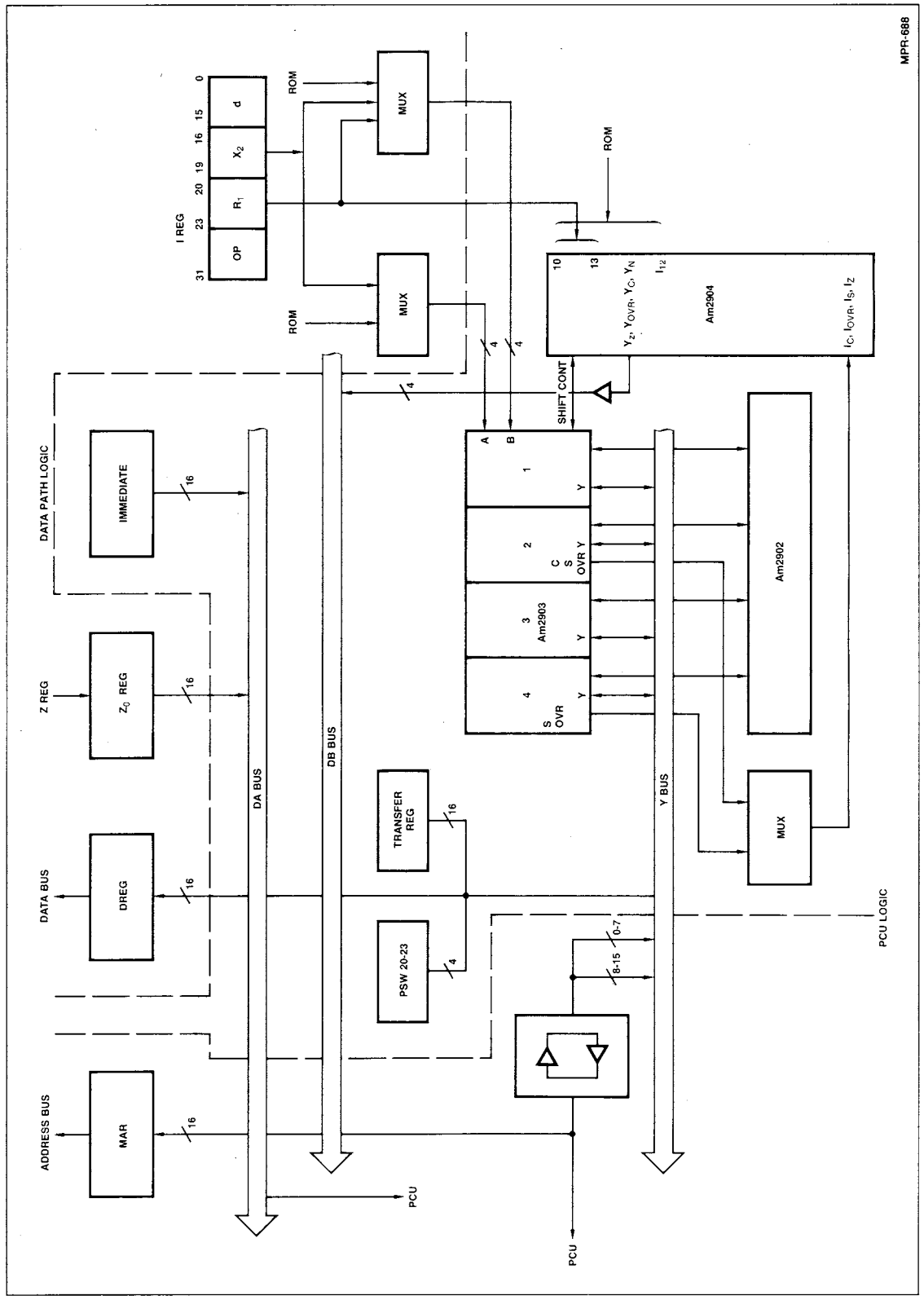


Figure 7. ALU Block Diagram.

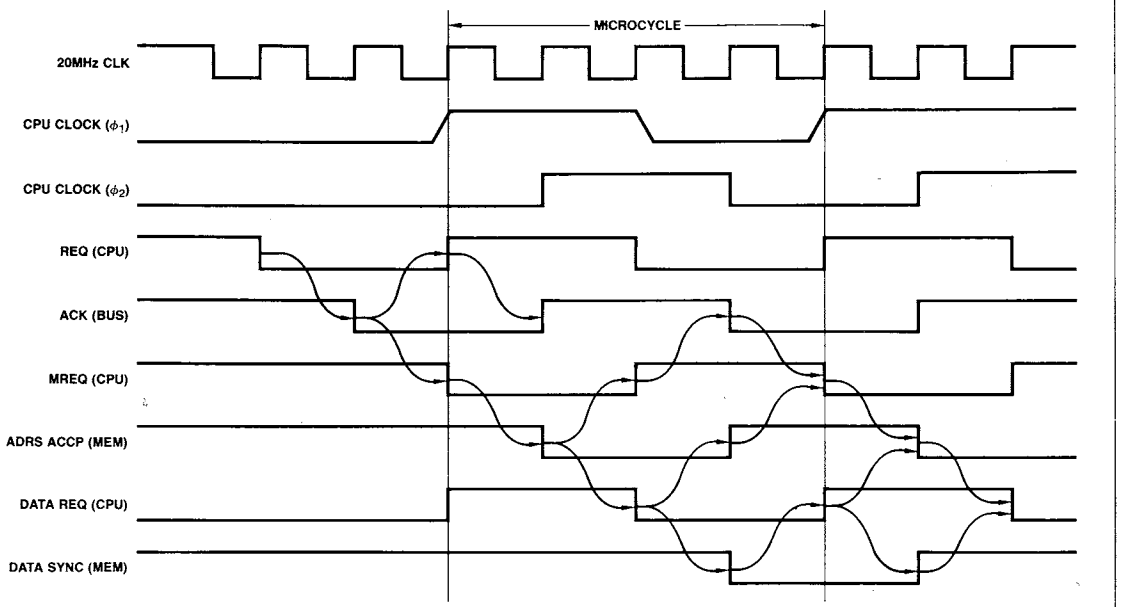


Figure 8. Clocked Handshaked Protocol.

eceived a Data Sync response. If the bus or memory resources f the system are temporarily being used by other processors, the omputer will stop the clock and wait.

Data Path

The Data Path logic incorporates 8-bit wide devices wherever possible. The D Register drives directly onto the external data bus. Both main memory and I/O data are received through the Z Registers. Registers Z, Z_0 and Z_1 are actually latches implemented with Am74S373's. The Z Register enable latch signal, LDZ is derived from the memory control logic and main memory board logic both of which are clocked with the uncontrolled 20MHz clock (20MHzUNC). Using the uncontrolled clock allows the memory operation to go to completion at memory speed even when single stepping the microcode. This allows the system to use dynamic RAM's in the main memory since stopping the handshaking circuits during single step would prevent refresh operations from taking place.

Data from the main memory passes through the Z Register to the Z_0 and Z_1 Registers. The Z_0 and Z_1 Registers are enabled transparent at the beginning of the microcycle following the read main memory microcycle. This allows memory data to flow through the Z and Z_0 Registers (actually latches) to the ALU or flow through the Z and Z_1 Registers to the Instruction Decoder (Mapping ROM). The Z_1 and Z_0 Registers are locked down halfway through the microcycle guaranteeing the computer solid data and making it possible to send data from the D-Register out to the external Data Bus during the second half of the same microcycle. This is another example of how this design tightly dovetails data transfers in order to gain very high execution rates.

Interrupt and Input/Output

The interrupt and I/O section is shown in Sheet 7 of the logic diagrams.

The basic interrupt handling is controlled by the Am2914. In this design the Am2914 is used to prioritize and enable interrupts, provide the mask register, generate an Interrupt Request and Interrupt Vector. Interrupt nesting is done in the machine software interrupt handler. The external interrupt request signals (INT_0-INT_7) are input into the Am2914 from the external Control Bus (C Bus). When a peripheral controller requests computer servicing, it activates its assigned interrupt line. If this interrupt level is unmasked and interrupts are enabled, the Am2914 activates the INTERRUPT REQ signal that goes to the Computer Control Unit which causes the microprogram to vector to the microcode interrupt service routine. This microcode routine pushes the PSW onto the main memory stack, then reads the interrupt vector from the Am2914 and uses this value to vector the computer to the machine software routine that services the interrupt.

The Am9519 MOS Universal Interrupt Controller is incorporated into the design and its Group Interrupt signal is connected to the least significant INT_0 input of the Am2914. The Am9519 handles an additional eight interrupt levels for low speed requesting devices. This MOS LSI component offers the computer comprehensive interrupt handling capabilities at low cost. One feature the Am9519 offers is the capability of software generated interrupts. The console function, single instruction stepping, is implemented using a microcode routine that uses the software generated interrupt capability.

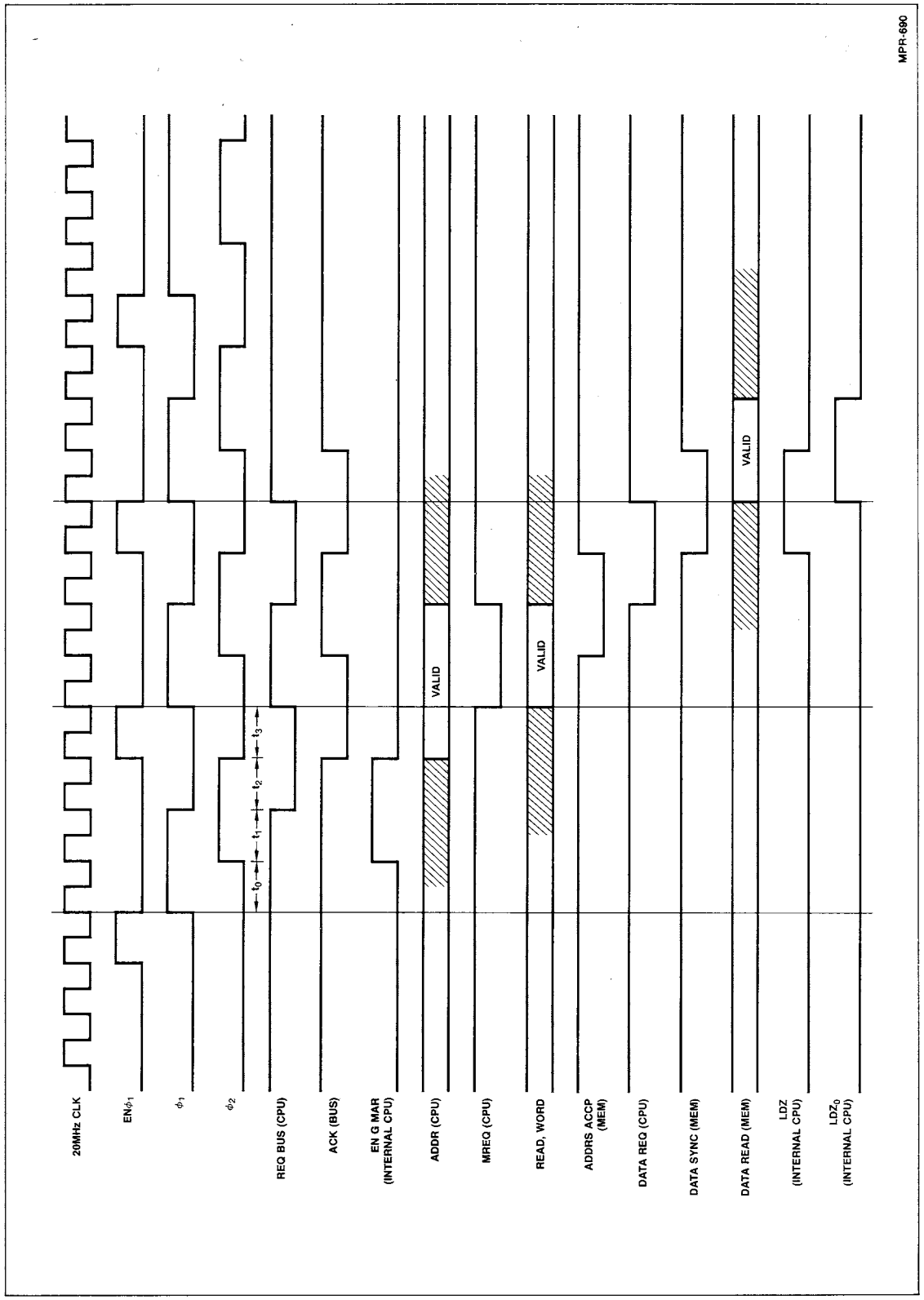


Figure 9. CPU Read Timing.

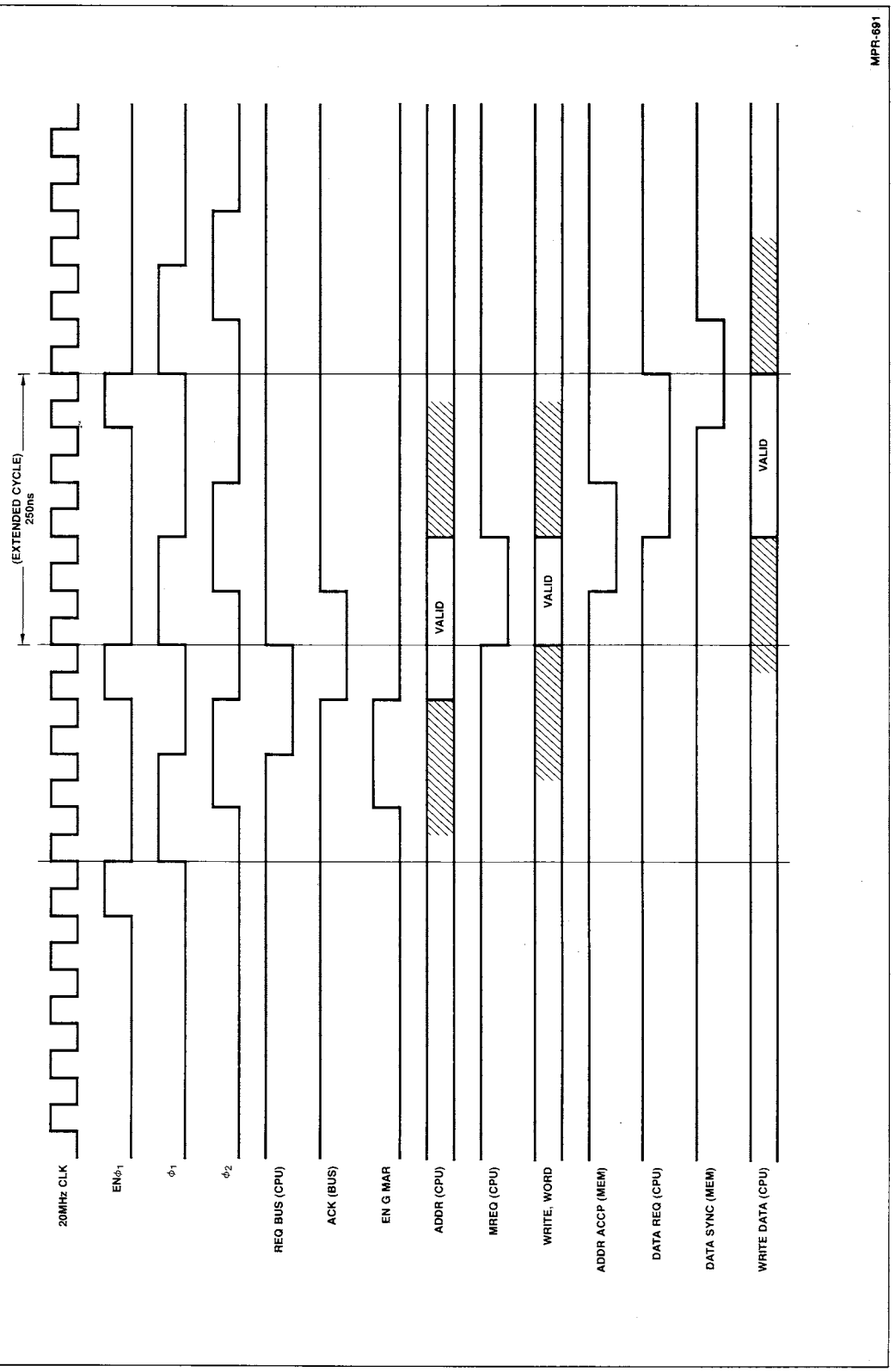


Figure 10. CPU Write Timing.

The I/O protocol for the AMD 16-Bit Computer is similar to that required to control Am8080/9080 peripheral circuits. As shown in Figures 11 and 12, the computer outputs the address over the system address bus, activates a control line (e.g., IORD) and holds these outputs until receiving a response, IOACK, from the peripheral controller. Execution of the I/O operation is done almost entirely in microcode with the I/O Control Register, a single Am2920, being the only additional hardware required. This is an example of a design precept followed in this computer which is to implement all features in microcode wherever possible. This results in a low cost computer, although sometimes slower, and a design that is flexible and easily modifiable to meet new requirements.

The I/O section has two Am8251/9551 Programmable Communication Interface components giving the computer two serial I/O Ports, one of which is reserved for the console. The console can be any standard RS-232 interface terminal.

Instruction Execution

To execute instructions, the main steps performed by the computer are: (1) form memory address, (2) instruction fetch, (3) decode, (4) displacement fetch, (5) form operand address, (6) operand fetch, and (7) execute. Every instruction type is made up of microinstructions that execute these basic steps, but most instructions require three steps or less. Instruction sequences for Register to Register (RR) and Register to Indexed Storage (RX) instructions are shown in Figures 13 and 14 to illustrate how the computer operates. These figures show the RR instruction requiring four microcycles and the typical RX instruction requiring seven microcycles. However, as will be explained later, in actual operation the effective time for an RR instruction is one microcycle and three for the RX.

Form Instruction Address

During this microcycle the instruction address is formed by having the Program Control Unit (PCU) under control of the microprogram increment the Program Counter by two. This address is then loaded into the MAR and back into the PC.

At the beginning of the cycle, Bus Request is activated causing the Bus Controller to respond with Bus Acknowledge. The address is then output from the MAR out on the Address Bus 50ns prior to the beginning of the next cycle.

Instruction Fetch

During this cycle, the main memory is fetching the contents of the address previously generated. The computer is designed to work with high-speed main memory capable of reading a memory location in one microcycle so that the instruction will be sent back to the computer at the beginning of the next cycle.

Decode Cycle

The instruction fetched from main memory during the previous cycle is sent to the computer at the beginning of the cycle. The instruction falls through the Z and Z_1 Registers (actually transparent latches) and is routed to the Instruction Decoder (Mapping PROM). The Instruction Decoder translates the 8-bit operation code of the instruction into an 8-bit address used as the starting address for the microprogram that will execute this instruction.

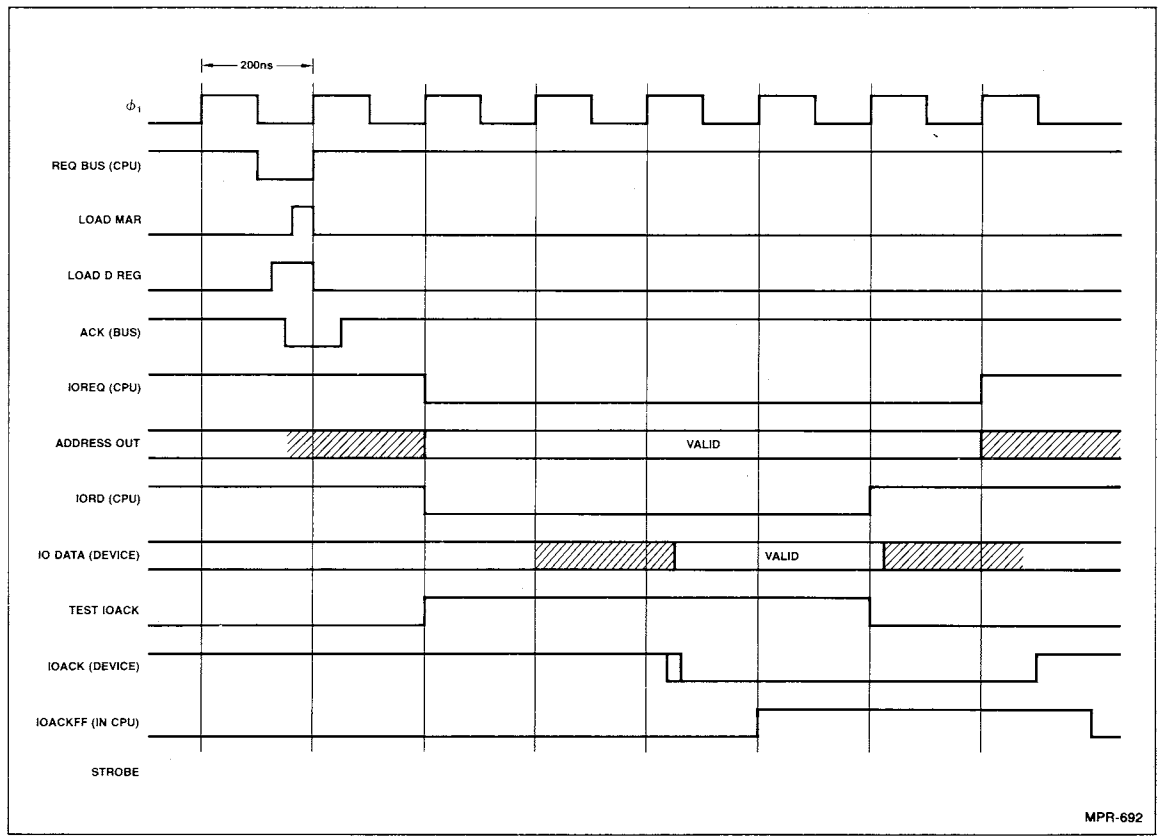


Figure 11. I/O Read Timing.

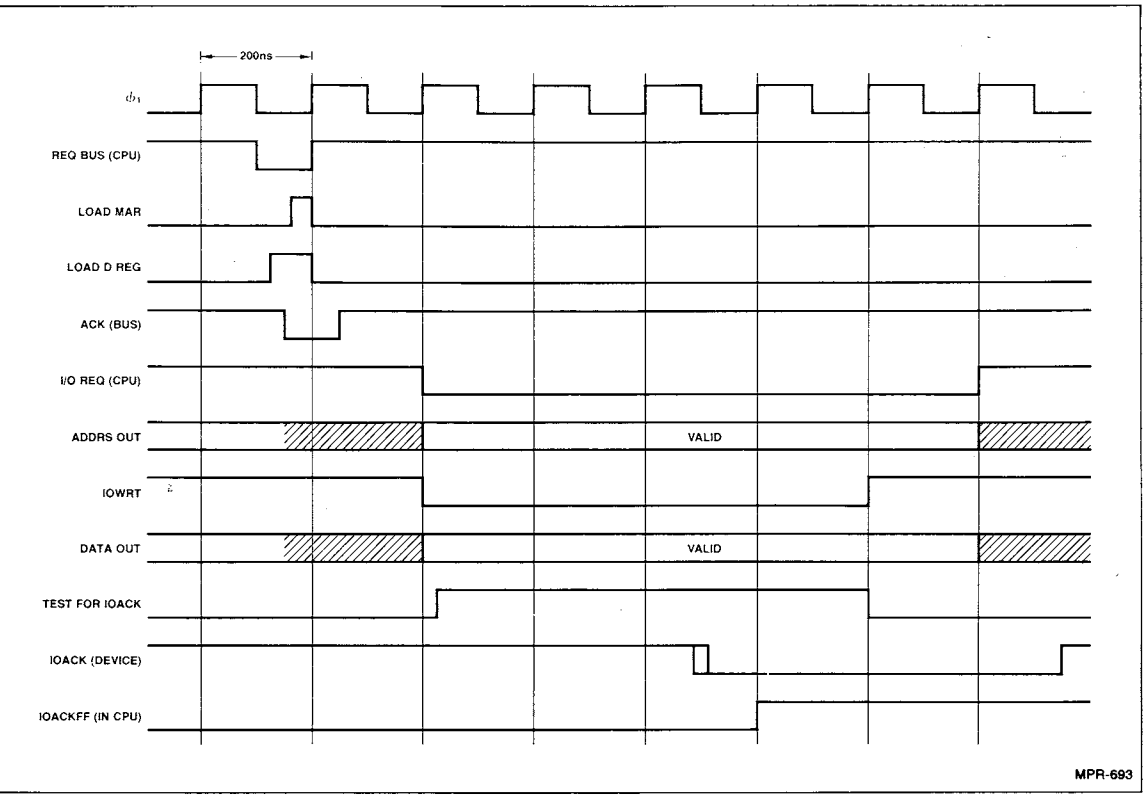


Figure 12. I/O Write Timing.

Microinstruction Operation	Microcycle Time			
	T_0	T_1	T_2	T_3
Form Instruction Address	A			
Instruction Fetch		A		
Decode			A	
Displacement Fetch				
Form Operand Address				
Operand Fetch				
Execute				A

Figure 13. RR Instruction Sequence.

Microinstruction Operation	Microcycle Time						
	T_0	T_1	T_2	T_3	T_4	T_5	T_6
Form Instruction Address	B						
Instruction Fetch		B					
Decode			B				
Displacement Fetch				B			
Form Operand Address					B		
Operand Fetch						B	
Execute							B

Figure 14. RX Instruction Sequence.

Displacement Fetch Cycle

After every instruction fetch another read cycle takes place. The second memory read will be another instruction fetch or an operand displacement fetch. The computer does not know what kind of a read out it is until the instruction decode is finished. For an RX instruction, after the memory read is completed, the computer identifies it as a displacement.

Form Operand Address Cycle

The memory word is sent from the main memory at the beginning of this cycle and then passes through the Z and Z_0 Register and goes to the ALU (Am2903's). The ALU adds the displacement and the contents of the register specified by X_2 field in the opcode and forms an operand address which is then loaded into the MAR. This has to be completed 50ns before the end of the cycle.

Operand Fetch Cycle

The memory read cycle is performed and the operand is sent to the computer at the beginning of the next cycle.

Execute Cycles

As the name implies, these are the microcycles that perform the task of the instruction but with the Am2903's normally only one execute cycle is required; however, some instructions (e.g., I/O instructions) take as many as seven execute cycles.

Simultaneously with the last execute cycle the Instruction Decoder is enabled.

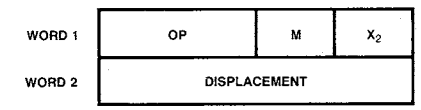
Pipelined Operations

If the architecture of the computer executed each of the instructions and each microstep sequentially, this computer would be just another computer relying on a high-speed clock to gain high throughput. However, the 16-Bit Computer becomes an exceptional machine by using pipelining techniques. In this approach, the instruction steps for the following instructions are done during the decode and execute steps of the current instruction. The pipelining operation for a Register to Register class of instructions is shown in Figure 15. With the pipeline full, note that when instruction A is being executed, instruction B is being decoded, instruction C is being fetched from Main Memory and the MAR is being loaded with the address for instruction D. In the following cycle, RR instruction B is executed and RR instructions C, D and E proceed through the pipeline. The pipelining technique results in an RR instruction effectively being executed in one microcycle. As illustrated in Figure 16, a new RX instruction can be executed every three microcycles.

Pipelining is great for throughput, but it is a bear to microcode especially the first time through since during any one cycle up to four instruction sequences have to be considered. It is not as bad as it first appears. Note that an instruction decode cannot take place until the last execute cycle of the current instruction. The major pipelining takes place during the first three steps: form memory address, instruction fetch, and decode. Execute and operand fetch steps allow full overlapped operation only during the last execute cycle. Instructions that require many execute microcycles (e.g., I/O instructions) cause the computer performance to drop down to nearly that of a non-pipelined machine.

Pipeline Operation with Regard to Branching and Interrupts

Pipeline operations greatly reduce instruction execution time if machine instructions are executed in sequential order; however, if a branch is taken this advantage is lost because the steps set up in preparation for a decode cycle become useless. The pipeline is said to be "flushed out" when a branch is taken. The RX Branch on Condition instruction has the form:



Where: M is a 4-bit field specifying the conditions for the jump.
(X_2) + displacement is the branch address

Figure 17 shows the sequence chart for a RX Branch on Condition instruction. During the microcycle A_1 the target address K for the branch is formed and loaded into the MAR and also the instruction B is fetched for the no branch case. By microcycle A_2, it has been determined to take or not take the branch. If the branch is not taken, the MAR is loaded with address B+2, while if the branch is taken, an instruction fetch is performed for K and the MAR is loaded with K+2. Finally in A_3 the next instruction is decoded. By proper microcoding, the conditional branch is executed in only three microsteps even though the pipeline was "flushed out".

Action	A, B, C, D are RR instructions												
Form Instruction Address	A	B	C	D									
Fetch Instruction		A	B	C	D								
Decode			A	B	C	D							
Fetch Displacement													
Form Operand Address													
Fetch Operand													
Execute				A	B	C	D						

Figure 15. Register-to-Register Pipeline Operation.

Action	A, B, C, D are RX instructions												
Form Instruction Address	A		B			C							
Fetch Instruction		A		B			C						
Decode			A			B			C				
Fetch Displacement			A			B			C				
Form Operand Address				A			B			C			
Fetch Operand					A			B			C		
Execute						A			B			C	

Figure 16. Register-to-Indexed Storage Pipeline Operation.

A = RX Branch Instruction
B = Next RX Instruction if branch is not taken
K = next RX Instruction if branch is taken

Action													
Form Instruction Address	A		B	K	B+2 K+2								
Fetch Instruction		A		B	K	B+2 K+2							
Decode			A			B K	etc.						
Fetch Displacement			A				B K						
Form Operand Address								B K					
Fetch Operand									B K				
Execute				A_1	A_2	A_3				B K			

Figure 17. Branch on Condition RX Pipeline Operation.

As with branching, an interrupt response alters the sequence of execution and "flushes" the pipeline. As was discussed previously in the Interrupt and Input/Output section, an interrupt request blocks the decoding of the next machine instruction and causes the Computer Control Unit to vector to the interrupt service routine. This microcode service routine pushes the PSW consisting of flags and Program Counter (PC) value onto the stack. The PC value is the current PC value minus 4. It is necessary to back the PC up to two instruction words (4 bytes), because the fetch instruction and form instruction address steps in the pipeline at the time of the jump to the interrupt microcode sequence have to be repeated when returning to the main machine program.

MICROINSTRUCTION FORMAT

All operations of the AMD 16-Bit Computer are under control of the microinstruction. Each microinstruction is 96 bits in length. The microinstruction format is summarized in Figure 18. The microinstruction definition is summarized in Figures 19a and 19b and is detailed in Table 2.

Figure 20 illustrates the AMDASM® Definition file for the 16-Bit Computer. AMDASM® is a meta-assembler developed by AMD for writing microprogams. The definition file defines microword length (WORD statement), formats (DEF statements) and constants (EQU statements) for the use of the actual microprogram (Figure 31).

The definition file is divided into 8 parts:

1. Am2910 sequencer opcode definitions
2. Am2903 ALU opcode definitions
3. Am2901A PCU opcode definitions
4. Am2904 shift mux and status control definitions
5. Datapath control bits definitions
6. Memory control bits definitions
7. Control strobe and control bits definitions
8. Immediate operand field definition

Am2910 Sequencer

Bit 91 of the microword is the input of CCEN of the Am2910. When bit 91 is a logical 1, the conditional operations are forced to unconditional operations. Bits 19-16 are the input to the instruction inputs to the Am2910. Bits 11-0 are the jump address field for instructions that need an address operand.

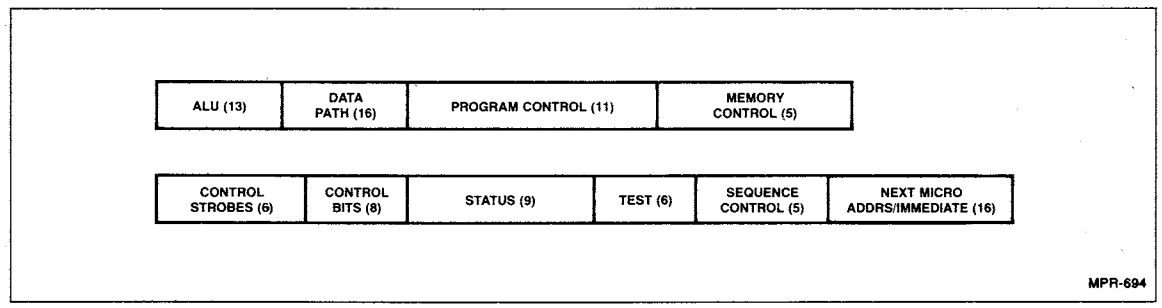


Figure 18. Summary of Microinstruction Word Fields.

				16-BIT COMPUTER
				MICRO CONTROL WORD BIT DEFINITIONS
MISC	95		ROUTE TO B	$\overline{RTB}$
	9291		TRANSFER Z TO ZI Am2910	(BP) Z → ZI $\overline{CCEN}$
ALU (13)	90898887868584838281807978		Am2903 IEU WORD/BYTE Am2903 Am2903 Am2903 Am2903 Am2903 Am2903 Am2903 Am2903 Am2903 Am2903	$\overline{WORD}$ $\overline{EA}$ $\overline{OEY}$ $\overline{OEB}$ I_8 I_7 I_6 I_5 I_4 I_3 I_2 I_1 I_0
Data Path (13)	77767574737271706968676665		ENABLE TRANSFER REG. LOAD TRANSFER REG. I-REG EN CTR I-REG INC/DEC PCU TRANS CHIP DISABLE PCU TRANSFER REG. LOAD MEMORY ADDR. REG. LOAD D-REG. LOAD ZI INTO I REG. ENABLE Z0 → DA ENABLE PSW SHIFT CNT Am2910 ADDR. BRANCH INSTR. EN	$\overline{ENTREG}$ $\overline{LDTREG}$ $\overline{ENCTR}$ INC PCUCD PCU → Y LDMAR LDD ZI → I $\overline{ENZ0}$ $\overline{PSW}$ $\overline{SHTCNTEN}$ $\overline{BRIEN}$
Program Control (11)	64636261595857565554		Am2901 F → B/$\overline{Q}$ Am2901 Am2901 Am2901 Am2901 Am2901 Am2901 Am2901 Am2901	$PCUI_7$ $PCUI_3$ $PCUI_2$ $PCUI_1$ $PCUI_0$ $PCUA_2$ $PCUA_1$ $PCUA_0$ $PCUB_2$ $PCUB_1$ $PCUB_0$
Memory Control (5)	53525150 49		BUS REQUEST MEMORY REQUEST HOLD REQUEST MEMORY WRITE/$\overline{READ}$ MEMORY WORD/$\overline{BYTE}$	REQB $\overline{MREQ}$ $\overline{HREQ}$ WRITE MWORD

Figure 19a. Micro Control Word Bit Definitions.

MICRO CONTROL WORD BIT DEFINITIONS

EN IMMEDIATE → DA BUS	$\overline{IMMD}$	X	Control Strobes (6)
ROM/IREGEN	ROM/I	X	
I/O CONTROL REG. EN	$\overline{IOEN}$	48	
Am2914 INTERRUPTS DISABLE	$\overline{INTDIS}$	47	
Am2914 ENI_0-ENI_3	$\overline{INTRIEN}$	46	
Am2904 SHIFT EN	$\overline{SHFTEN}$	45	
		44	
		43	
GENERAL USE CONTROL BITS	$CNTLB_7$	42	Control Bits (8)
	$CNTLB_6$	41	
	$CNTLB_5$	40	
	$CNTLB_4$	39	
	$CNTLB_3$	38	
	$CNTLB_2$	37	
	$CNTLB_1$	36	
	$CNTLB_0$	35	
Am2904 OUT EN CONDITIONAL TEST	$\overline{OECT}$	X	Status (9)
Am2904 EN ZERO	$\overline{EZ}$	X	
Am2904 EN CARRY	$\overline{EC}$	34	
Am2904 EN SIGN	$\overline{ES}$	33	
Am2904 EN OVERFLOW	$\overline{EOVR}$	32	
Am2904 EN MACHINE STATUS	$\overline{CEM}$	31	
Am2904 EN MICRO STATUS	$\overline{CE\mu}$	30	
Am2904 I_{12} CARRY OUT CNTL	I_{12}	29	
Am2904 I_{11} CARRY OUT CNTL	I_{11}	28	
		27	
		26	
Am2904	$TEST_5$	25	Test (6)
Am2904	$TEST_4$	24	
Am2904	$TEST_3$	23	
Am2904 & Am25LS251	$TEST_2$	22	
Am2904 & Am25LS251	$TEST_1$	21	
Am2904 & Am25LS251	$TEST_0$	20	
Am2910 I_3	NAC_3	19	Sequences CNTL (4)
Am2910 I_2	NAC_2	18	
Am2910 I_1	NAC_1	17	
Am2910 I_0	NAC_0	16	
	M_{15}	15	Next Micro Addr & Immed (16)
	M_{14}	14	
	M_{13}	13	
	M_{12}	12	
	M_{11}	11	
	M_{10}	10	
	M_9	9	
	M_8	8	
	M_7	7	
	M_6	6	
	M_5	5	
	M_4	4	
	M_3	3	
	M_2	2	
	M_1	1	
	M_0	0	

Figure 19a. Micro Control Word Bit Definitions (Cont.)

Control Strobes / Control Bits (35-42)	ROM/IREGEN Bit 47	I/O Control Register Bit 46	Am2914 I_0-I_3 Bit 44	Am2904 Shift Enable Bit 43
$CNTLB_7$	B_3	I/O7		
$CNTLB_6$	B_2	I/O6		
$CNTLB_5$	B_1	I/O5		
$CNTLB_4$	B_0	I/O4		I_{10}
$CNTLB_3$	A_3	I/O3	I_3	I_9
$CNTLB_2$	A_2	I/O2	I_2	I_8
$CNTLB_1$	A_1	I/O1	I_1	I_7
$CNTLB_0$	A_0	I/O0	I_0	I_6

Figure 19b. Detailed Description of Bits 34 through 47.

Table 2. Microinstruction Definition.

		Definition
95	$\overline{RTB}$	Routes second register field to B-RAM of Am2903.
92	$Z \rightarrow Z_1$	Loads the value in the Z register into the Z_1 Register at the beginning of the microcycle.
91	$\overline{CCEN}$	Enables the CC input of the Am2910.
ALU		
90	$\overline{WORD}$	These bits control the four Am2903's. The function of EA, OEY, OEB, and I_{8-0} is listed in Figure 20. WORD when enabled (LOW) causes the Am2903's to operate on words (16-bits). When disabled (HIGH) the ALU operates on bytes (the least significant byte). This bit disabled blocks WE to the upper two Am2903's and turns off their Y outputs.
89	$\overline{EA}$	
88	$\overline{OEY}$	
87	$\overline{OEB}$	
86	I_8	
85	I_7	
84	I_6	Zeroes should be forced to the upper 8 bits of the Y bus via the PCU to allow the zero status to operate correctly when the WORD bit is disabled. Also, when disabled the status (C, OVR, S) sent to the Am2904 is taken from the second Am2903 (numbering 0-3 least significant to most significant slice) instead of the most significant Am2903.
84	I_6	
83	I_5	
82	I_4	
81	I_3	
80	I_2	
79	I_1	
78	I_0	
77	$\overline{ENTREG}$	Enable Transfer Register – enables the Transfer Register onto the DA input bus of the Am2901A's and Am2903's.
76	LDTREG	Load Transfer Register – loads the Transfer Register from the Y bus.
75	$\overline{ENCTR}$	Enable I Register Counter – enables the I Register Counter (I_{7-14}) to count. This value is used to address the general registers during stack instructions and by incrementing or decrementing this value the microprogram can read or write successive registers.
74	INC	I Register INC/DEC – the value in I_{7-14} can be either incremented (if this bit is HIGH) or decremented.
73	PCUCD	PCU Transceiver Disable – when HIGH this bit disables the PCU Transceivers from receiving or transmitting data.
72	PCU → Y	PCU Transceiver Control – when HIGH this bit allows the PCU Transceivers to pass data from PCU to the Y bus. [WORD high (microbit 90) disables the least significant 8 bits of these transceivers.] When LOW data passes from the Y bus to the MAR.
71	LDMAR	Load Memory Address Register (MAR) – this bit loads the Memory Address Register.
70	LDD	Load D Register – this bit loads the D Register with data from the Y bus.
69	$Z_1 \rightarrow I$	Load Z_1 into I Register – this bit loads data from Z_1 into the I Register. The I Register holds only the upper 16 bits of the instruction.
68	$\overline{ENZ_0}$	Enable $Z_0 \rightarrow DA$ – this bit LOW enables the Z_0 Register onto the ALU DA.

Table 2. Microinstruction Definition. (Cont.)

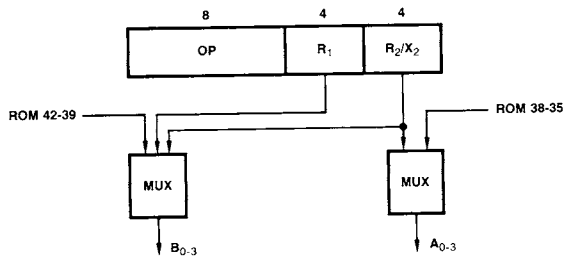
		Definition
67	$\overline{PSW}$	Enable PSW – this bit LOW enables the PSW onto the ALU DA.
66	$\overline{SHTCNTEN}$	Shift Count to Am2910 – this bit LOW enables the least significant four bits of the instruction (I_{0-3}) onto the D input to the Am2910 sequencer. This allows the value to be entered into the Am2910 internal counter to be used during shift instructions.
65	$\overline{BRIEN}$	Branch Instruction Enable – this bit LOW enables I_{4-7} of the Instruction Register onto the Am2904 I_{0-3} input. The I_{0-3} inputs control the tests of the status register.
PCU		
64	$PCUI_7$	These bits control the PCU which is designed around four Am2901's. The $PCUI_7$, $PCUI_3$, $PCUI_2$, $PCUI_1$ and $PCUI_0$ bits connect directly to the Am2901 I_7, I_3, I_2, I_1 and I_0 respectively. The $PCUA_2$-$PCUA_0$ and $PCUB_2$-$PCUB_0$ connect to the A and B Address inputs of the Am2901. I_4, I_5, I_8, A_3 and B_3 are tied to ground. I_6 is tied to I_7.
63	$PCUI_3$	
62	$PCUI_2$	
61	$PCUI_1$	
60	$PCUI_0$	
59	$PCUA_2$	
58	$PCUA_1$	
57	$PCUA_0$	
56	$PCUB_2$	
55	$PCUB_1$	
54	$PCUB_0$	
53	REQB	Request Bus – this bit requests use of the system bus. This request is made the microcycle preceding a Memory Request or use of the bus for an I/O transfer. If the request is not honored, the processing of the next microinstruction is halted until the acknowledge is issued.
52	MREQ	Memory Request – this bit requests the memory to do a read or write operation.
51	$\overline{HREQ}$	Hold Request – this bit LOW blocks the bus controller from releasing the system bus to another device. Normally a Bus Request is cleared as soon as the Bus Acknowledge is issued. HREQ holds Bus Request and prevents any other device from using the bus.
50	WRITE	Memory Write/READ – this bit indicates to the memory the MREQ is for a write operation (if HIGH) and a read operation (if LOW).
49	MWORD	Memory Word/BYTE – the Memory Word/BYTE microbit specifies whether the memory operation will be a word operation or a byte operation. If the operation specified is a byte operation the least significant address bit determines which byte of the two byte pair in memory is affected. If the LSBit is a zero, the most significant byte is read or written, and the LSBit is a one, the least significant byte is read or written.
48	$\overline{IMMD}$	EN Immediate DA Bus – this bit LOW enables the 16-bit immediate value (least significant 16 bits of the microinstruction) to the ALU DA bus.
47	ROM/I	ROM/I REG Enable – this bit enables either the ROM bits 42-35 or the I register bits I_{0-7} onto the A/B address inputs of the ALU according to the following: 
46	$\overline{IOEN}$	I/O Control Register Enable – this bit loads the I/O Control Register with microbits 42-35.
45	$\overline{INTDIS}$	Am2914 Interrupt Disable – this bit disables the Am2914 Interrupt Controller from recognizing interrupt requests.
44	$\overline{INTRIEN}$	Am2914 ENI_0-ENI_3 – this bit is the instruction enable for the Am2914. The instruction inputs I_{0-3} are connected to microbits 35-38 respectively.
43	$\overline{SHFTEN}$	Am2904 Shift Enable – this bit is connected to the shift enable of the Am2904. The shift controls I_{6-10} are connected to microbits 35-39 respectively.

Table 2. Microinstruction Definition. (Cont.)

		Definition	
42	$CNTLB_7$	This control field is used to provide several different functions as defined by the previously described control strobes (microbits 47-43).	
41	$CNTLB_6$		
40	$CNTLB_5$		
39	$CNTLB_4$		
38	$CNTLB_3$		
37	$CNTLB_2$		
36	$CNTLB_1$		
35	$CNTLB_0$		
34	$\overline{OECT}$	OUT EN CONDITIONAL TEST	These bits are used to control the Am2904. Their functions are defined in Figure 21. OECT is used to enable the test output of the Am2904 to the CC input of the Am2910.
33	$\overline{EZ}$	EN ZERO	
32	$\overline{EC}$	EN CARRY	
31	$\overline{ES}$	EN SIGN	
30	$\overline{EOVR}$	EN OVERFLOW	
29	$\overline{CEM}$	EN MACRO STATUS	
28	$\overline{CE}$	EN MICRO STATUS	
27	I_{12}	CARRY OUT CONTROL	
26	I_{11}	CARRY OUT CONTROL	
25	$TEST_5$	These bits determine which test is to be performed for the conditional branch and stack functions. The various tests are listed in Figure 25. The testing is done both in the Am2904 and an 8 to 1 multiplexer.	
24	$TEST_4$		
23	$TEST_3$		
22	$TEST_2$		
21	$TEST_1$		
20	$TEST_0$		
19	NAC_3	291013	These bits are connected to the I_{3-0} inputs of the Am2910 to control the sequencing of the microprogram. Their definitions are listed in Figure 26.
18	NAC_2	291012	
17	NAC_1	291011	
16	NAC_0	291010	
15	M_{15}	These bits provide the branch address for the Am2910 and the 16-bit immediate field.	
14	M_{14}		
13	M_{13}		
12	M_{12}		
11	M_{11}		
10	M_{10}		
9	M_9		
8	M_8		
7	M_7		
6	M_6		
5	M_5		
4	M_4		
3	M_3		
2	M_2		
1	M_1		
0	M_0		

Am2903 ALU

The first 16 equates assign mnemonics for the I8-I5 of the Am2903 which controls the destination of the ALU result. The next 16 equates assign mnemonics for I4-I1 of the Am2903 which control the operations of the ALU. The ALU definition indicates the default is the Y bus forced to zero with no operation on destination. The next group of definition selects the source operand, followed by the special function definitions of the Am2903.

Am2901A PCU

The PCU definitions include a group of often used PC instructions such as PCU. NEXT, PCU. JUMP etc. The PCU definition itself allows a not predefined instruction be accessible to the microprogrammer.

AM2904 Shift Linkage Multiplexer and Status Register

The group of equates control the updating of the status register and the TEST definition controls the shift linkage multiplexer. The carry control controls the carry into the least significant Am2903 slice.

Datapath Control

The data control equates assign mnemonics to different datapath control bits.

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
DEFINITION FILE FOR 16 BIT COMPUTER

 ;
 ;       AMDASM DEFINTION FILE FOR 16-BIT COMPUTER
 ;       USING AM2901A, AM2903, AM2904 & AM2910
 ;       FILE CREATED BY STEVE CHENG 8/25/78
 ;
 ;       REVISION 2.0    12/6/78
 ;
         WORD 96
 ;
 ;       DEFINITIONS FOR AM2910 SEQUENCER
 ;
 JZ:     DEF 4X,B#0,71X,H#0,16X          ;JUMP ZERO
 CJS:    DEF 4X,B#0,71X,H#1,4X,12V%      ;COND JSB PL
 JSB:    DEF 4X,B#1,71X,H#1,4X,12V%      ;UNCONDITIONAL JSR PL
 JMAP:   DEF 4X,B#0,71X,H#2,15X          ;JUMP MAP
 CJP:    DEF 4X,B#0,71X,H#3,4X,12V%      ;COND JUMP PL
 JMP:    DEF 4X,B#1,71X,H#3,4X,12V%      ;UNCONDITONAL JUMP PL
 PUSH:   DEF 4X,B#0,71X,H#4,4X,12V%      ;PUSH/COND LD CNTR
 PHLC:   DEF 4X,B#1,71X,H#4,4X,12V%      ;PUSH AND ID CNTR
 JSRP:   DEF 4X,B#0,71X,H#5,4X,12V%      ;COND JSR R/PL
 CJV:    DEF 4X,B#0,71X,H#6,16X          ;COND JUMP VECTOR
 JMPV:   DEF 4X,B#1,71X,H#6,16X          ;UNCONDTIONAL JUMP VECTOR
 JRP:    DEF 4X,B#0,71X,H#7,4X,12V%      ;CONI JUMP R/PL
 RFCT:   DEF 4X,B#0,71X,H#8,4X,12V%      ;REPEAT LOOP, CNTR <> 0
 RPCT:   DEF 4X,B#0,71X,H#9,4X,12V%      ;REPEAT PL, CNTR <> 0
 CRTN:   DEF 4X,B#0,71X,H#A,16X          ;COND RTN
 RTN:    DEF 4X,B#1,71X,H#A,16X          ;UNCONDTIONAL RETURN
 CJPP:   DEF 4X,B#0,71X,H#B,4X,12V%      ;COND JUMP PL & POP
 LDCT:   DEF 4X,B#0,71X,H#C,4X,12V%H#FFF ;ID CNTR & CONT
 LOOP:   DEF 4X,B#0,71X,H#D,16X          ;TEST END LOOP
 CONT:   DEF 4X,B#0,71X,H#E,16X          ;CONTINUE
 TWB:    DEF 4X,B#0,71X,H#F,4Y,12V%      ;THREE-WAY BRANCH
 ;
 ;       DEFINITIONS FOR AM2903 ALU
 ;
 ;       THE ALU DEFINTION IS OF THE FOLLOWING FORMAT
 ;       ALU DESTINATION CONTROL, FUNCTION
 ;
 ;       EQUATES FOR ALU DESTINATION CONTROI
 ;
 ADR:    EQU H#0         ;ARITHMETIC SHIFT DOWN, RESULTS INTC RAM
 LDR:    EQU H#1         ;LOGICAL SHIFT DOWN, RESULTS INTO RAM
 ADRQ:   EQU H#2         ;ARITH. SHIFT DOWN, RESULTS INTO RAM AND Q
 LDRQ:   EQU H#3         ;LOGICAL SHIFT DOWN, RESULTS INTO RAM AND Q
 RPT:    EQU H#4         ;RESULTS INTO RAM, GENERATE PARITY
 LDQP:   EQU H#5         ;LOGICAL SHIFT DOWN Q, GENERATE PARITY
 QPT:    EQU H#6         ;RESULTS INTO Q, ;GENERATE PARITY
 RQPT:   EQU H#7         ;RESULTS INTO RAM AND Q, GENERATE PARITY
 AUR:    EQU H#8         ;ARITH. SHIFT UP, RESULTS INTO RAM
 LUR:    EQU H#9         ;LOGICAL SHIFT UP, RESULTS INTO RAM
 AURQ:   EQU H#A         ;ARITH. SHIFT UP, RESULTS INTO RAM AND Q
 LURQ:   EQU H#B         ;LOGICAL SHIFT UP, RESULTS INTO RAM AND Q
 YBUS:   EQU H#C         ;RESULTS TO Y BUS ONLY
 LUQ:    EQU H#D         ;LOGICAL SHIFT UP Q
 SINEX:  EQU H#E         ;SIGN EXTEND
 REG:    EQU H#F         ;RESULTS TO RAM, SIGN EXTEND
 ;
 ;       EQUATES FOR ALJ FUCNTIONS
 ;
 HIGH:   EQU H#0         ;FI = 1
 SUBR:   EQU H#1         ;SUBTRACT R FROM S
 SUBS:   EQU H#2         ;SUBTRACT S FROM R
 ADD:    EQU H#3         ;ADD R AND S
 PASS:   EQU H#4         ;PASS S
 COMPLS: EQU H#5         ;2'S COMPLEMENT OF S
 PASSR:  EQU H#6         ;PASS R
 COMPLR: EQU H#7         ;2'S COMPLEMENT OF R
 LOW:    EQU H#8         ;FI = 0
 NOTRS:  EQU H#9         ;COMPLEMENT R AND WITH S
 EXNOR:  EQU H#A         ;EXCLUSIVE NOR R WITH S
 EXOR:   EQU H#B         ;EXCLUSIVE OR R WITH S
 AND:    EQU H#C         ;AND R WITH S
 NOR:    EQU H#D         ;NOR R WITH S
 NAND:   EQU H#E         ;NAND R WITH S
 OR:     EQU H#F         ;OR R WITH S
 ;
 ;       ALU DEFINTION
 ;
 ALU:    DEF 9X,4VH#C,4VH#8,79X
 ;
 ;       ALU OPERAND SOURCES
 ;
 AB:     DEF 6X,B#0,1X,B#0,6X,B#0,78X    ;R = RAM A, S = RAM B
 ADB:    DEF 6X,B#0,1X,B#1,6X,B#0,78X    ;R = RAM A, S = DB
 AQ:     DEF 6X,B#0,10X,B#1,78X          ;R = RAM A, S = Q
 DAB:    DEF 6X,B#1,1X,B#0,6X,B#0,78X    ;R = DA, S = RAM B
 DADB:   DEF 6X,B#1,1X,B#1,6X,B#0,78X    ;R = DA, S = DB
 DAQ:    DEF 6X,B#1,10X,B#1,78X          ;R = DA, S = Q
 ;
 ;       WORD/BYTE CONTROL
 ;
 WORD:   DEF 5X,B#0,90X
 ;
 ;       OUTPUT Y ENABLE
 ;
 OEY:    DEF 7X,B#0,88X
 ;
 ;       SPECIAL FUNCTIONS FOR AM2903
 ;
 ;       TO USE THE SPECIAL FUNCTIONS, THE DESTINATION
 ;       CONTROL MUST NOT BE AQ OR DAQ
 ;
 ;       SPECIAL FUNCTION EQUATES
 ;
 USMUL:  EQU H#00        ;UNSIGNED MULTIPLY
 TCMUL:  EQU H#20        ;TWO'S COMPLEMENT MULTIPLY
 INCTWO: EQU H#40        ;INCREMENT BY ONE OR TWO
 SMTC:   EQU H#50        ;SIGN-MAGNITUDE/TWO'S COMPLEMENT
 TCMLS:  EQU H#60        ;TWO'S COMPLEMENT MULT. LAST STEP
 SLN:    EQU H#80        ;SINGLE LENGTH NORMALIZE
 DLN:    EQU H#A0        ;DOUBLE LENGTH NORMALIZE AND 1ST DIVIDE OP.
 TCDIV:  EQU H#C0        ;TWO'S COMPLEMENT DIVIDE
 TCDC:   EQU H#E0        ;TWO'S COMPLEMENT DIVISION CORRECTION
 ;
 ;       SPECIAL FUNCTION DEFINITION
 ;
 SPF14:  DEF 9X,8VH#,79X
 ;
 ;       DEFINITION FOR AM2901 PROGRAM CONTROL UNIT (PCU)
 ;
 ;       PCU REGISTER DEFINITIONS:
 ;       R0 = PC         PROGRAM COUNTER
 ;       R1 = SP         STACK POINTER
 ;       R2 = SPLL       STACK POINTER LOWER LIMIT
 ;       R3 = SPUL       STACK POINTER UPPER LIMIT
 ;       R4 = 2          CONSTANT 0
 ;       R5 = 4          CONSTANT 2
 ;
 ;       EQUATES FOR PCU DEFINITIONS
 ;
 QEU:    EQU B#0         ;Q REG = ZERO, B-RAM = ONE DEFAULT
 ;
 ;       EQUATE FOR PCU FUNCTIONS
 ;
 SUB:    EQU B#1         ;SUB = ONE, ADD = ZERO DEFAULT
 ;
 ;       EQUATES FOR SOURCE CONTROL
 ;
 PCUAQ:  EQU Q#0
 PCUAB:  EQU Q#1
 PCUZQ:  EQU Q#2
 PCUZB:  EQU Q#3
 PCUZA:  EQU Q#4
 PCUDA:  EQU Q#5
 PCUDQ:  EQU Q#6
 PCUDZ:  EQU Q#7
 ;
 ;       EQUATES FOR PCU A-RAM
 ;
 A0:     EQU Q#0
 A1:     EQU Q#1
 A2:     EQU Q#2
 A3:     EQU Q#3
 A4:     EQU Q#4
 A5:     EQU Q#5
 A6:     EQU Q#6
 A7:     EQU Q#7
 ;
 ;       EQUATES FOR PCU B-RAM
 ;
 B0:     EQU Q#0
 B1:     EQU Q#1
 B2:     EQU Q#2
 B3:     EQU Q#3
 B4:     EQU Q#4
 B5:     EQU Q#5
 B6:     EQU Q#6
 B7:     EQU Q#7
 ;
 ;       PCU DEFINITION
 ;
 PCU:    DEF 31X,1VB#1,1VB#0,3VQ#1,3VQ#,54X
 ;
 PCU.NEXT:       DEF 31X,B#1000110000001,54X     ;PC = PC + 2
 PCU.PUSH:       DEF 31X,B#1100110000001,54X     ;SP = SP - 2
 PCU.POP:        DEF 31X,B#1000110000001,54X     ;SP = SP + 2
 PCU.JUMP:       DEF 31X,B#1011100000000,54X     ;PC = D
 PCU.TR2:        DEF 31X,B#1010110000000,54X     ;PC = TREG + 2
 PCU.NOP:        DEF 31X,B#1001100000000,54X     ;PC TO OUTPUT
 PCU.SP:         DEF 31X,B#1001100001001,54X     ;SP TO OUTPUT
 PCU.DEC4:       DEF 31X,B#1100110100000,54X     ;PC = PC - 4
 ;
 ;       DEFINTIONS FOR AM2904 RELATED CONTROL BITS
 ;
 ;       AM2904 BIT DEFINITIONS ARE AS FOLLOWS:
 ;       BITS 95-44 = DON'T CARES
 ;       BIT 43 = SHIFT ENABLE
 ;       BITS 42-35 = GENERAL PURPOSE CONTROL BITS
 ;       BIT 34 = OUT EN CONDITIONAL TEST
 ;       BIT 33 = ENABLE ZERO
 ;       BIT 32 = ENABLE CARRY
 ;       BIT 31 = ENABLE SIGN
 ;       BIT 30 = ENABLE OVERFLOW
 ;       BIT 29 = ENABLE MACHINE STATUS
 ;       BIT 28 = ENABLE MICRO STATUS
 ;       BITS 27-26 = CARRY OUT CONROL
 ;       BITS 25-20= CONDTIONAL BRANCH TEST
 ;
 SHIFTEN:        EQU B#0         ;SHIFT ENABLE
 OECT:           EQU B#0         ;OUT EN CONDITIONAL TEST
 EZ:             EQU B#0         ;ENABLE ZERO
 EC:             EQU B#0         ;ENABLE CARRY
 ES:             EQU B#0         ;ENABLE SIGN
 EOVR:           EQU B#0         ;ENABLE OVERFLOW
 CEM:            EQU B#0         ;ENABLE MACHINE STATUS
 CEU:            EQU B#0         ;ENABLE MICRO STATUS
 ;
 AM2904: DEF 52X,1VB#1,8X,1VB#1,1VB#1,1VB#1,1VB#1,1VB#1,1VB#1,1VB#1,28X
 ;
 ;       TEST BITS DEFINTION
 ;
 TEST:   DEF 70X,6VQ#,20X
 ;
 ;       EQUATES FOR AM2904 CARRY-OUT CONTROL
 ;
 COEQ0:  EQU B#00        ;CARRY-OUT = 0
 COEQ1:  EQU B#01        ;CARRY-OUT = 1
 COEQCI: EQU B#10        ;CARRY-OUT = CARRY-IN
 COEQST: EQU B#11        ;CARRY-OUT = CARRY OF STATUS REGISTER
 ;
 ;       CARRY-OUT CONTROL DEFINITION
 ;
 CARRYCTL:       DEF 68X,2VB#00,26X
 ;
 ;       REGISTER MUX SELECT
 ;
```

Figure 20. Definition File for 16-Bit Computer.


```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
DEFINITION FILE FOR 16 BIT COMPUTER

 RTB:    DEF B#0,95X     ;ROUTE R1 TO RAM B
 ;
 ;       EQUATES FOR DATAPATH DEFINITION
 ;
 ZZI:    EQU B#1         ;Z REG TO ZI REG
 ENTREG: EQU B#0         ;ENABLE TRANSFER REGISTER
 LDTREG: EQU B#1         ;LOAD TRANSFER REGISTER
 ENCTR:  EQU B#0         ;I-REG EN CTR
 INC:    EQU B#1         ;I-REG INC/DEC*
 PCUT:   EQU B#01        ;PCU TRANSCEIVER TO Y-BUS
 YMAR:   EQU B#00        ;PCU TRANSCEIVER TO MAR BUS
 PCUMAR: EQU B#11        ;PCU TRANSCEIVER CHIP DISABLE
 LDMAR:  EQU B#1         ;LOAD MAR
 LDD:    EQU B#1         ;LOAD D REG
 ZII:    EQU B#1         ;LOAD ZI INTO I-REG
 ENZ0:   EQU B#0         ;ENABLE Z0 TO DA
 PSW:    EQU B#0         ;ENABLE PSW
 SHTCNTEN:EQU B#0        ;SHIFT CNT 2910 ADDR
 BRIEN:  EQU B#0         ;BRANCH INSTRUCTION ENABLE
 ;
 ;               DATAPATH DEFINITION
 ;
 DATAPATH:       DEF 3X,1VB#0,14X,1VB#1,1VB#0,1VB#1,1VB#0,2VB#11,1VB#0,
 /                   1VB#0,1VB#0,1VB#1,1VB#1,1VB#1,1VB#1,55X
 ;
 ;       EQUATES FOR MEMORY CONTROL
 ;
 REQB:   EQU B#1         ;BUS REQUEST
 MREQ:   EQU B#1         ;MEMORY REQUEST
 HREQ:   EQU B#0         ;HOLD REQUEST
 WRITE:  EQU B#1         ;MEMORY WRITE
 MWORD:  EQU B#1         ;MEMORY WORD/BYTE*
 ;
 ;       DEFINTION FOR MEMORY CONTROL
 ;
 MEM.CONT:DEF 42X,1VB#0,1VB#0,1VB#1,1VB#1,1VB#0,49X
 ;
 ;       EQUATES FOR CONTROL STROBES
 ;
 ROM:    EQU B#1         ;ROM/INEGEN*
 IOEN:   EQU B#0         ;I/O CONTROL REG. ENABLE
 INTDIS: EQU B#0         ;INTERRUPT DISABLE
 INTRIEN:EQU B#0         ;ENABLE I0-I3 ON AM2914
 ;
 ;       CONTROL STROBE DEFINTION
 ;
 CONTROL:        DEF 48X,1VB#0,1VB#1,1VB#1,1VB#1,44X
 ;
 ;       CONROL BITS DEFINITION
 ;
 CNTLB:  DEF 53X,8VH#,35X
 ;
 ;       IMMEDIATE ROM DEFINITION
 ;
 IMMD:   DEF 47X,B#0,32X,16VH#   ;ENABLE IMMEDIATE OPERAND
 ;
         END

TOTAL PHASE 1 ERRORS =   0
```

Figure 20. Definition File for 16-Bit Computer (Cont.).

Memory Control

The memory control equates assign mnemonics to different memory control bits.

Control Strobe and Control Bits

The control strobe equates assign mnemonics to the control bit strobe signals. The control bit definition defines a hexadecimal bit pattern for the 8 control bits.

Immediate Operand

When the Am2910 sequencer is executing an instruction which does not require an address operand, bits 15-0 in the microword can be used as a 16-bit constant to load ALU, PCU etc. This is accomplished by putting the constant in bits 15-0 and force bit 48 to logic 0.

MICROCODE

Flowcharts

The flowcharts of the major instruction types are shown in the following figures.

Figure 21 illustrates the basic microprogram flowchart and demonstrates how the pipelining is done in microcode. This figure illustrates the sequencing of the computer starting with no instructions in the pipeline. By the fourth microinstruction, the pipeline is full and the CPU can execute for example a macroinstruction every microcycle.

Figure 22 illustrates the execution of an RR instruction. During an RR instruction, PC+6 is loaded into the MAR and a bus request is issured for the content of PC+6. The contents of PC+4 are read into the Z register. The Z_1 and I Registers are loaded with the contents of PC+2. The instruction at PC is executed. The input to the mapping PROM is loaded with the contents of PC+2. Thus in a stream of RR instructions, four instructions are in progress concurrently.

Figure 23 illustrates the execution of an RX instruction. In this figure the decode operation takes the microprogram to the microstep where the form address operation is done. Since the decode of the instruction has been completed in the previous step, the form address microinstructions are unique to each RX instruction in spite of the fact the operation performed is identical.

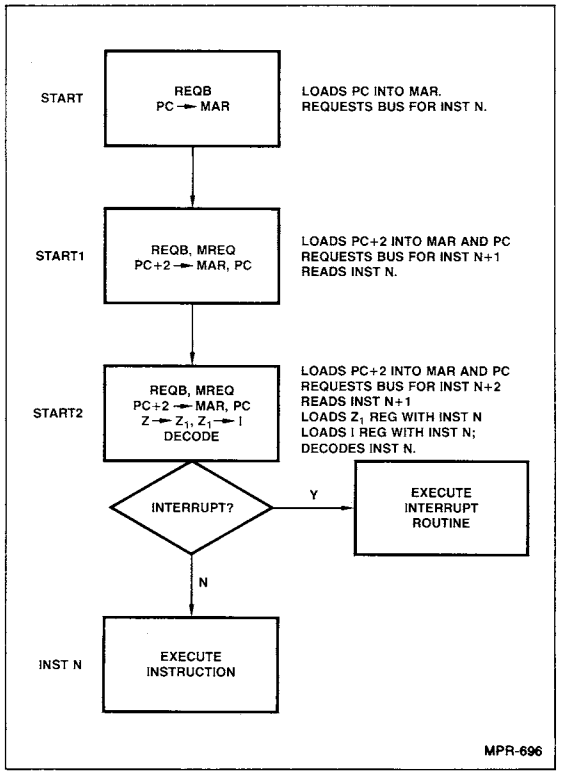
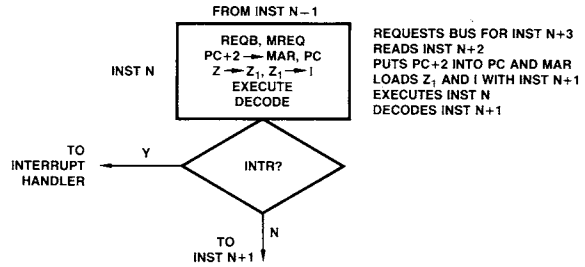


Figure 21. Microprogram Start Up Flow Chart.

From the form address step, the microprogram jumps to FETCHOP where the operand is fetched. This step returns to where the instruction is actually executed.

Figure 24 illustrates the execution of an RSI instruction. At the first microstep, the immediate operand is already in the Z_0 register. So the instruction is executed in the first step. The microprogram is then jumped to START2 to refill the pipeline.

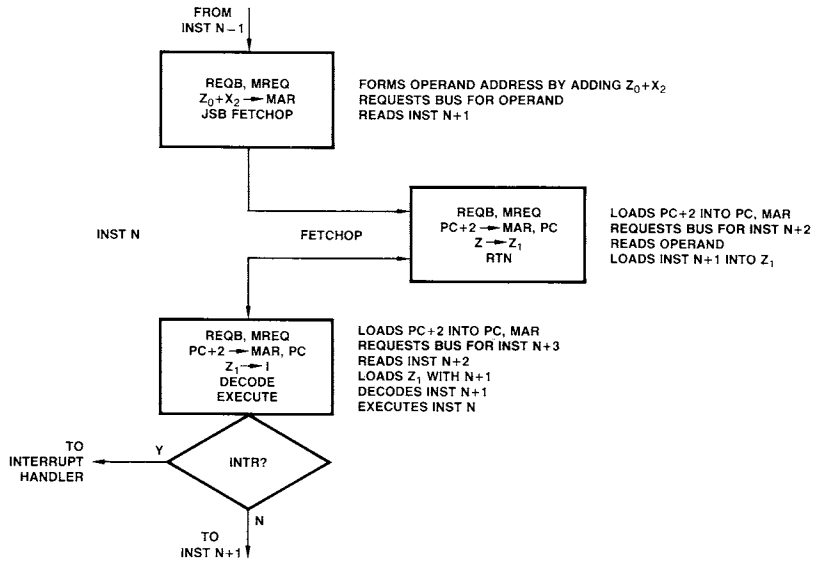


RR INSTRUCTIONS IMPLEMENTED

OPCODE	R1, R2	
LR	Load Register	R1 = (R2)
AR	Add Register	R1 = (R1) + (R2), Set CC
SR	Subtract Register	R1 = (R1) – (R2), Set CC
NR	AND Register	R1 = (R1) AND (R2), Set CC
ORR	OR Register	R1 = (R1) OR (R2), Set CC
CLR	Compare Logical Register	Set CC with (R1) – (R2)
XORR	Exclusive OR Register	R1 = (R1) XOR (R2), Set CC

MPR-697

Figure 22. RR Instruction Flow Chart.

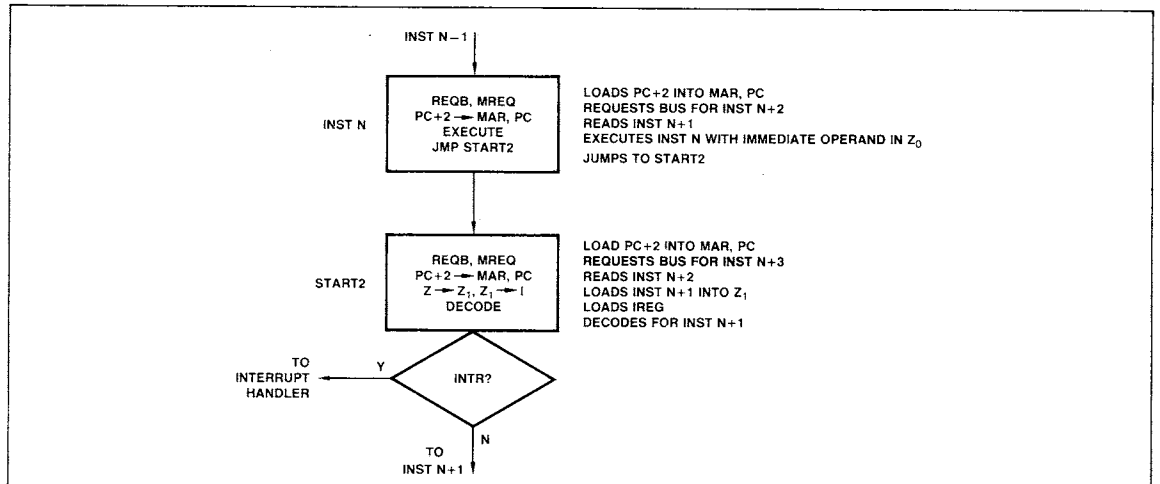


RX INSTRUCTIONS IMPLEMENTED

OPCODE	R1, X2 (DISP)	
LD	R1, X2 (D)	R1 = (X2) + D
ST	R1, X2 (D)	(X2) + D = (R1)
ADD	R1, X2 (D)	R1 = (R1) + [(X2) + D], Set CC
SUB	R1, X2 (D)	R1 = (R1) – [(X2) + D], Set CC
N	R1, X2 (D)	R1 – (R1) AND [(X2) + D], Set CC
O	R1, X2 (D)	R1 = (R1) OR [(X2) + D], Set CC
CMP	R1, X2 (D)	Set CC FOR (R1) – [(X2) + D]

MPR-698

Figure 23. RX Type Instruction.



IMMEDIATE INSTRUCTIONS IMPLEMENTED

OP CODE	R1, DATA	
LI	Load Immediate	R1 = DATA
NI	AND Immediate	R1 = (R1) AND DATA, Set CC
OI	OR Immediate	R1 = (R1) OR DATA, Set CC
XI	Exclusive or Immediate	R1 = (R1) XOR DATA, Set CC
AI	Add Immediate	R1 = (R1) + DATA, Set CC
SI	Subtract Immediate	R1 = (R1) – DATA, Set CC
CI	Compare Immediate	Set CC with (R1) – DATA

MPR-699

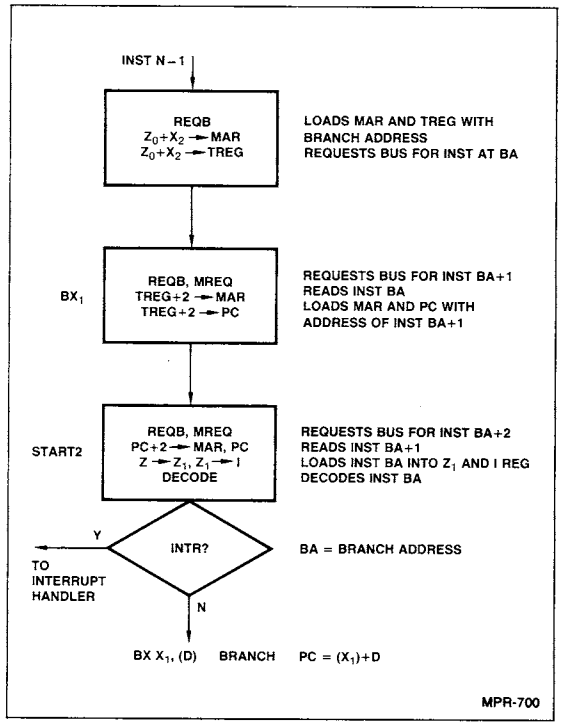
Figure 24. Immediate Instructions.

Figure 25 illustrates the execution of an unconditional branch instruction. At the first microstep the displacement is already in the Z_0 register. The branch address is formed by adding the contents of the Z_0 register to the contents of the index register X_1. The MAR is loaded with the branch address and a bus request is issued for the contents of the branch address. The branch address is also loaded into the transfer register for subsequent loading of PC. In the next step, the contents of the transfer register+2 is loaded into the PC and MAR. A bus request is issued to BA+2. The content of BA is read. The microprogram is then transferred to START2 to fill up the pipeline.

Figure 26 illustrates the Conditional Branch instruction. In step 1, unlike the Unconditional Branch instruction, the contents of the memory (instruction N+1) is read, in case the test condition fails and the macro program falls through. The condition test is enabled in this step. If the test passes, the microprogram transfers to Unconditional Branch routine. If the test fails, the microprogram proceeds to fill the pipeline and continue.

Figure 27 illustrates the branch and link instruction. The flowchart is similar to Unconditional Branch except an extra step (STEP 2) is inserted. This step saves PC in R_1.

Figure 28 illustrates a shift or rotate instruction. In STEP 1 the opcode of the next instruction is loaded into Z_1 registers and the shift count of the shift instruction is loaded into the loop counter of Am2910. STEP 2 executes the shift instruction N+1 times, where N is the shift count in the instruction. It should be noted that since Am2910 detects – 1 as the stop condition, the shift count loaded should be one less than the desired count. Step 3 is the same as the RNI (request next instruction). It is duplicated because the fail condition of RPCT in Am2910 can only fall through.



MPR-700

Figure 25. Unconditional Branch.

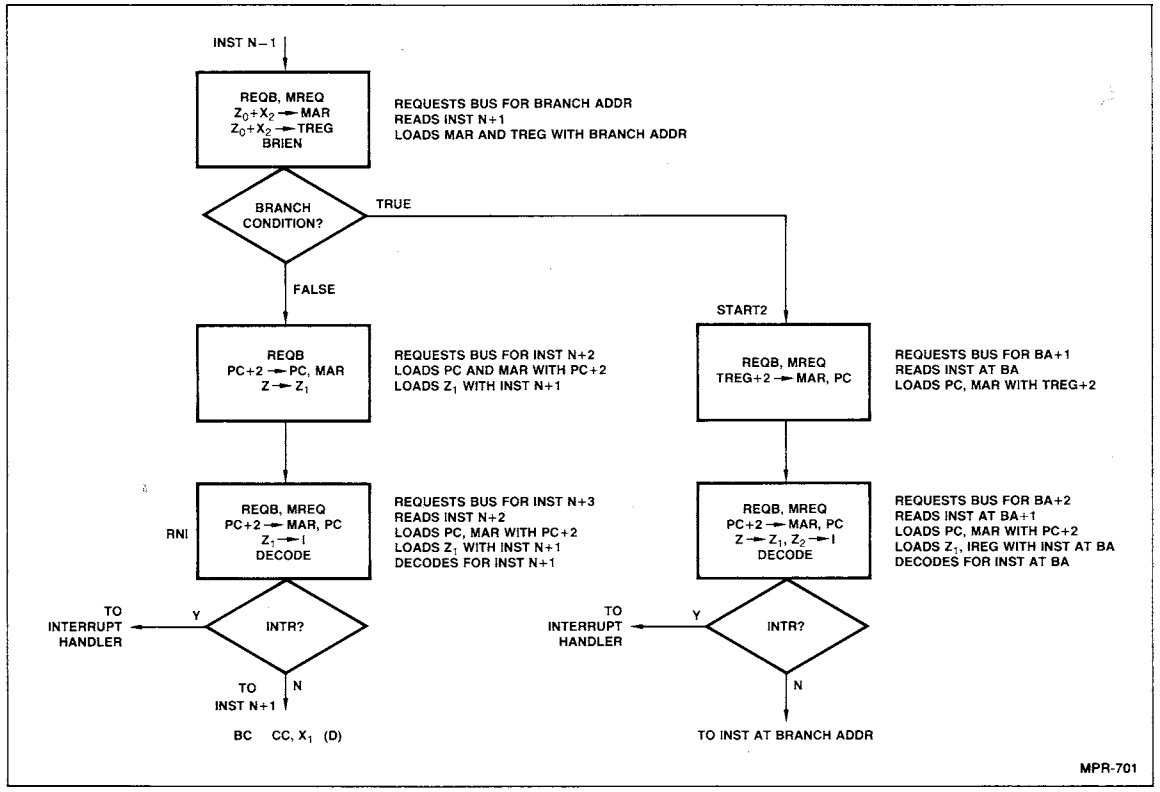


Figure 26. Conditional Branch.

Figure 29 illustrates the input instruction. In STEP 1, the I/O Port Address is formed by adding Z_0 and X_2. Bus request is issued for the I/O Port. The desired width of the I/O read pulse is loaded into the Am2910 Loop Counter. The width of the I/O read pulse is (N+2) X cycle time where N is the number loaded. The I/O read signal is turned on. In STEP 2, the bus is held for the I/O address and the loop counter is decremented until it becomes −1. In STEP 3, I/O read pulse is turned off but I/O address is held for possible address hold time requirement of the I/O device. On the trailing edge of the I/O read pulse, the content of the I/O Port is strobed into the Z_0 register. In STEP 4, the content of Z_0 register is loaded into R_1, thus completing the I/O read. Bus request is issued for the next instruction and microprogram jumps to START1 to refill the pipeline.

Figure 30 illustrates the output instruction. In STEP 1, bus request is ussued for the I/O Port Address. In STEP 1, the content of R_1 is transferred to the D register for outputting to the data bus. The I/O write pulse is set and the width of the write pulse is loaded into the Am2910 Loop Counter as in the input instruction. In STEP 3, the I/O address is held until loop counter becomes −1. In STEP 4, the content of the D register is strobed into the I/O Port by turning off the I/O Write Pulse. The microprogram jumps to START to refill the pipeline.

The Figures 21-30 illustrate the major instruction types implemented. These are by no means the only possible instructions for the 16-bit computer described. Some other instructions such as stack instructions are shown in the microcode but not in the figures and should be easily understood with the above examples as a guide.

Figure 31 illustrates the implementation of some typical instructions. Instruction 0 is the restart instruction. It jumps to INIT which is located in location H#180 because the mapping PROM maps only into the first 256 locations. So it is desirable to preserve these locations for Macro instructions. The initialization routine does the following:

1. Turn on I/O reset signal and jump (Inst H#0)
2. Set R_0 in ALU to 0 (Inst H#180)
3. Set R_0 in PCU (PC) to 0 (Inst H#181)
4. Set R_1 in PCU (SP) to H#4000 (Inst H#182)
5. Set R_4 in PCU to 2 (Inst H#183)
6. Set R_5 in PCU to 4 (Inst H#184)
7. Turn off I/O reset signal (Inst H#185)
8. Initialize console USART (Inst H#186-H#190)

The microinstruction that executes macroinstructions are grouped as follows:

Type	Figure	Microinst # (Hex)
RR Instructions	22	005-00B
RX Instructions	23	00C-01B
RSI Instructions	24	01C-022
Branch Instructions	25-27	023-02A
Shift Instructions	28	02B-042
Input Instruction	29	043-046
Output Instruction	30	047-04A
Stack Instructions	–	04B-059
Interrupt Instructions	–	05A-061

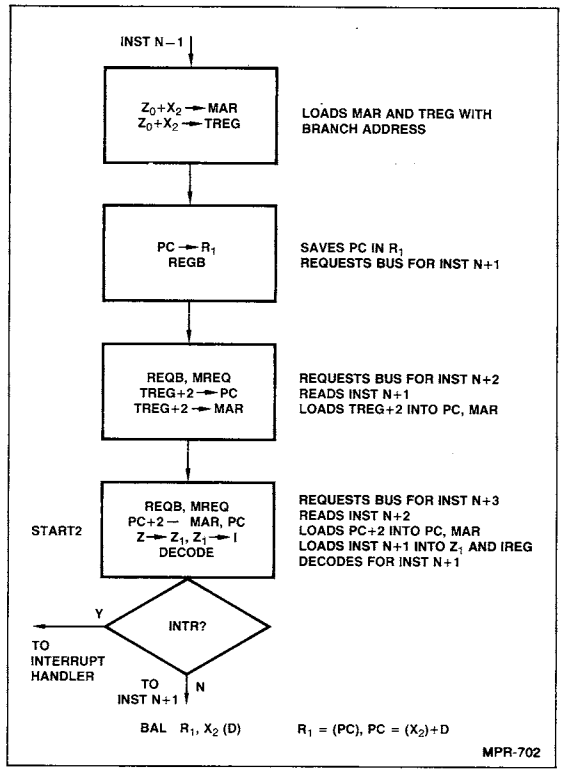


Figure 27. Branch and Link.

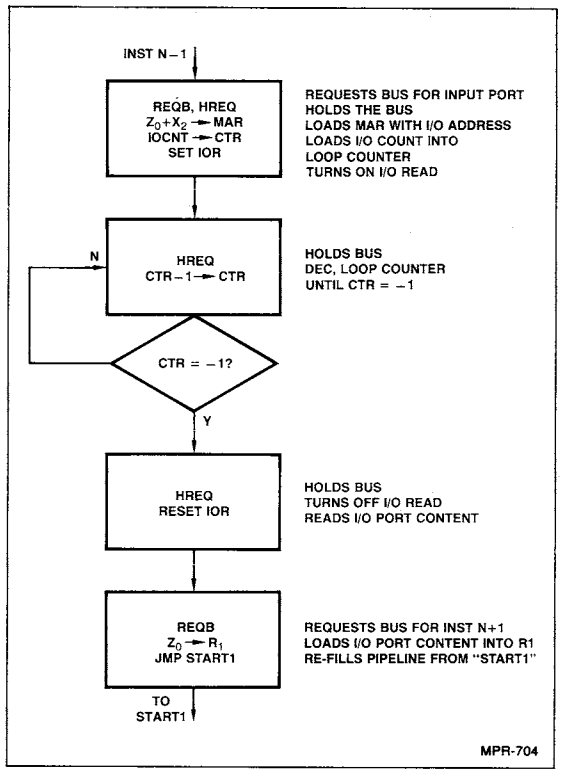


Figure 29. Input Instruction.

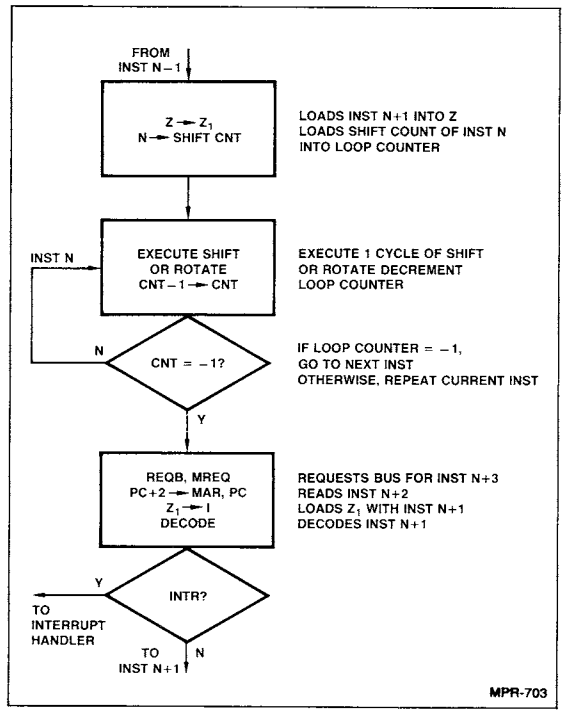


Figure 28. Shift and Rotate Instructions.

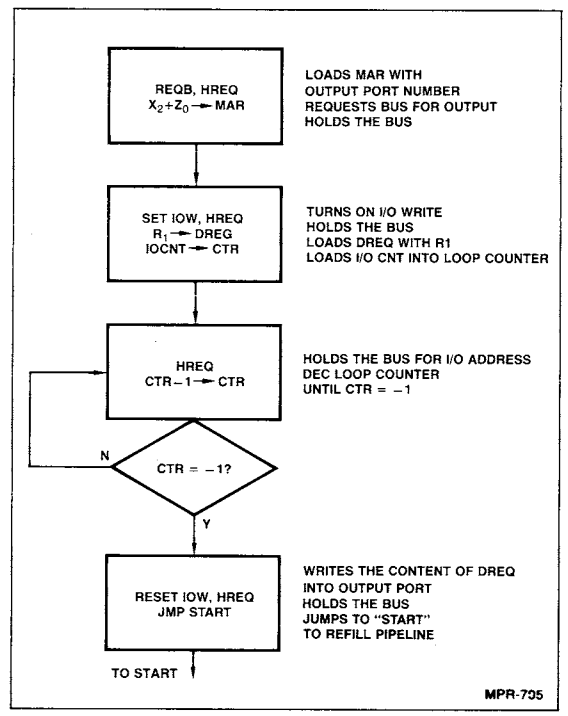


Figure 30. Output Instruction.

Upon an interrupt, the 16-Bit Computer finishes its current instruction and jumps to microinstruction H#1FF. The interrupt handler works as follows:

1. Current PSW is stored in DREG and SP = SP−2 (Inst H#1FF).
2. The content of PSW is written onto the stack in memory. PC = PC−4 to flush out the pipeline (Inst H#1F0).
3. SP = SP−2 (Inst H#1F1).
4. The content of the adjusted PC is written to the DREG (Inst H#1F2).
5. The content of the PC is written onto the stack in memory and the vector in the Am2914 is output to the interrupt vector PROM. A vector jump is made following this instruction depending on the interrupt number (Inst H#1F3).
6. The vector jump directs to 1 of 8 locations labelled INT_0-INT_7. For INT_1-INT_7, the first instruction disables interrupt in the Am2914 and forces new PC value into PC. INT_0 requires an extra instruction to clear the Am9519. The interrupt vector in the Am9519 is to be determined by the macro interrupt handler.
7. This next instruction is the same as the START instruction. The previous instruction cannot jump to START directly because the immediate operand uses the jump address field. The macroprogram resumes at the new PC value.

The instructions implemented cover only a small portion of all possible instructions. Only 137 or 512 microinstructions are used. The rest of the instruction space could be used to vastly enhance the instruction set such as byte operations, storage to storage instructions, etc.

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
MICROPROGRAM FOR 16 BIT COMPUTER

        ;       ************************************
        ;
        ;       MICROPROGRAM FOR AMD 16-BIT COMPUTER
        ;         WRITTEN BY STEVE CHENG  9/78
        ;            REVISION 1.1  12/15/78
        ;
        ;       ************************************
        ;
        ;       RESET SEQUENCE STARTS HERE
        ;
0000 RESET:  ALU & WORD & CONTROL ,IOEN,INTDIS, & CNTLB H#7F & DATAPATH &
      /      MEM.CONT REQB,,HREQ,,MWORD & AM2904 & PCU.NOP & JMP INIT
        ;
        ;       REQUEST BUS FOR INSTRUCTION N
        ;
0001 START:  ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,MWORD &
      /      AM2904 & PCU.NOP & CONT
        ;
        ;       REQUEST BUS FOR INSTRUCTION N+1. READ INSTRUCTION N
        ;
0002 START1: ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,,,,, & MEM.CONT REQB,MREQ,,MWORD &
      /      AM2904 & PCU.NEXT & CONT
        ;
        ;       REQUEST BUS FOR INSTRUCTION N+2. READ INTRUCTION N+1.
        ;       LOAD ZI REGISTER WITH INSTRUCTION N.
        ;       DECODE FOR INSTRUCTION N
        ;
0003 START2: ALU YBUS,PASS & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       REQUEST NEXT INSTRUCTION
        ;
0004 RNI:    ALU YBUS,PASS & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;       ========================================
        ;
        ;              RR TYPE INSTRUCTIONS
        ;
        ;       ========================================
        ;
        ;       LOAD REGISTER                   18      RR      CC: NONE
        ;       LR R1,R2        R1 = (R2)
        ;
0005 LR:     ALU REG,PASSR & AB & CARRYCTL & OEY & WORD &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & CONTROL & PCU.NEXT & JMAP
        ;
        ;       ADD REGISTERS                   1A      RR      CC: CSVZ
        ;       AR R1,R2        R1 = (R1) + (R2)
        ;
0006 AR:     ALU REG,ADD & AB & CARRYCTL & OEY & WORD &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & CONTROL & PCU.NEXT & JMAP
        ;
        ;       SUBTRACT REGISTERS              1B      RR      CC: CSVZ
        ;       SR R1,R2        R1 = (R1) - (R2)
        ;
0007 SR:     ALU REG,SUBR & AB & CARRYCTL COEQ1 & OEY & WORD &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & CONTROL & PCU.NEXT & JMAP
        ;
        ;       AND REGISTERS                   14      RR      CC: CSVZ
        ;       NR R1,R2        R1 = (R1) AND (R2)
        ;
0008 NR:     ALU REG,AND & AB & CARRYCTL & OEY & WORD &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & CONTROL & PCU.NEXT & JMAP
        ;
        ;       OR REGISTERS                    16      RR      CC: CSVZ
        ;       OR R1,R2        R1 = (R1) OR (R2)
        ;
0009 ORR:    ALU REG,OR & AB & CARRYCTL & OEY & WORD &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & CONTROL & PCU.NEXT & JMAP
        ;
        ;       COMPARE LOGICAL REGISTERS       15      RR      CC: CSVZ
        ;       CLR R1,R2               CC = RESULT OF (R1) - (R2)
        ;                               CONTENTS OF R1 AND R2 ARE NOT AFFECTED
        ;
000A CLR:    ALU YBUS,SUBR & AB & CARRYCTL COEQ1 & OEY & WORD &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & CONTROL & PCU.NEXT & JMAP
        ;
        ;       EXCLUSIVE OR REGISTERS          17      RR      CC: CSVZ
        ;       XOR R1,R2       R1 = (R1) XOR (R2)
        ;
000B XORR:   ALU REG,EXOR & AB & CARRYCTL & OEY & WORD &
      /      DATAPATH ZZI,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & CONTROL & PCU.NEXT & JMAP
        ;       ========================================
        ;
        ;              RX TYPE INSTRUCTIONS
        ;
        ;       ========================================
        ;
        ;       LOAD                            58      RX      CC: NONE
        ;       LD R1,X2(D)     R1 = [(X2) + D]
        ;
000C LD:     ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NOP & JSB FETCHOP
        ;
000D         ALU REG,PASSR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       STORE                           59      RX      CC: NONE
        ;       ST R1,X2(D)     (X2) + D = (R1)
        ;
000E ST:     ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NOP & CONT
        ;
000F         ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ZZI,,,,,,LDD,,,, & MEM.CONT REQB,,,,MWORD &
      /      AM2904 & PCU.NOP & CONT
        ;
0010         ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,,,,, & MEM.CONT REQB,MREQ,,WRITE,MWORD &
      /      AM2904 & PCU.NEXT & JMP RNI
        ;
        ;       ADD                             5A      RX      CC: CSVZ
        ;       ADD R1,X2(D)    R1 = (R1) + [(X2) + D]
        ;
0011 ADDX:   ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NOP & JSB FETCHOP
        ;
0012         ALU REG,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMAP
        ;
        ;       SUBTRACT                        5B      RX      CC: CSVZ
        ;       SUB R1,X2(D)    R1 = (R1) - [(X2) + D]
        ;
0013 SUBX:   ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NOP & JSB FETCHOP
        ;
0014         ALU REG,SUBR & DAB & CARRYCTL COEQ1 & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMAP
        ;
        ;       AND                             54      RX      CC: CSVZ
        ;       N R1,X2(D)      R1 = (R1) AND [(X2) + D]
        ;
0015 N:      ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NOP & JSB FETCHOP
        ;
0016         ALU REG,AND & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMAP
        ;
        ;       OR                              56      RX      CC: CSVZ
        ;       O R1,X2(D)      R1 = (R1) OR [(X2) + D]
        ;
0017 O:      ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NOP & JSB FETCHOP
        ;
0018         ALU REG,OR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMAP
        ;
        ;       COMPARE                         55      RX      CC: CSVZ
        ;       CMP R1,X2(D)    CC = RESULT OF (R1) - [(X2) + D]
        ;                       THE CONTENTS OF R1 AND (X2) + D ARE NOT AFFECTED
        ;
0019 CMP:    ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NOP & JSB FETCHOP
```

Figure 31. Microprogram for 16-Bit Computer.


```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
MICROPROGRAM FOR 16 BIT COMPUTER

        ;
001A         ALU YBUS,SUBR & DAB & CARRYCTL COEQ1 & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMAP
        ;
        ;       ########################################
        ;
        ;       SUBROUTINE TO FETCH OPERAND FROM MEMORY
        ;
        ;       ########################################
        ;
001B FETCHOP:ALU YBUS,PASSR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ZZI,,,,,LDMAR,,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & RTN
        ;       ========================================
        ;
        ;              IMMEDIATE INSTRUCTIONS
        ;
        ;       ========================================
        ;
        ;       LOAD IMMEDIATE                  41      RSI     CC: NONE
        ;       LI R1,DI        R1 = DI
        ;
001C LI:     ALU REG,PASSR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMP START2
        ;
        ;       AND IMMEDIATE                   94      RSI     CC: CSVZ
        ;       NI R1,DI        R1 = R1 AND DI
        ;
001D NI:     ALU REG,AND & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMP START2
        ;
        ;       OR IMMEDIATE                    96      RSI     CC: CSVZ
        ;       OI R1,DI        R1 = R1 OR DI
        ;
001E OI:     ALU REG,OR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMP START2
        ;
        ;       EXCLUSIVE OR IMMEDIATE          97      RSI     CC: CSVZ
        ;       XI R1,DI        R1 = R1 XOR DI
        ;
001F XI:     ALU REG,EXOR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMP START2
        ;
        ;       ADD IMMEDIATE                   9A      RSI     CC: CSVZ
        ;       AI R1,DI        R1 = R1 + DI
        ;
0020 AI:     ALU REG,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMP START2
        ;
        ;       SUBTRACT IMMEDIATE              9B      RSI     CC: CSVZ
        ;       SI R1,DI        R1 = R1 - DI
        ;
0021 SI:     ALU REG,SUBR & DAB & CARRYCTL COEQ1 & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMP START2
        ;
        ;       COMPARE IMMEDIATE               95      RSI     CC: CSVZ
        ;       CI R1,DI        CC = RESULT OF R1 - DI
        ;                       THE CONTENT OF R1 IS NOT AFFECTED
        ;
0022 CI:     ALU YBUS,SUBR & DAB & CARRYCTL COEQ1 & OEY & WORD & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.NEXT & JMP START2
        ;
        ;       ++++++++++++++++++++++++++++++++++++
        ;
        ;              BRANCH INSTRUCTIONS
        ;
        ;       ++++++++++++++++++++++++++++++++++++
        ;
        ;       BRANCH UNCONDITIONAL            74      RX      CC: NONE
        ;       B X2(D)         PC = (X2) + D
        ;
0023 BX:     ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,ENTREG,,,,,LDMAR,,,,,, &
      /      MEM.CONT REQB,MREQ,,,MWORD & AM2904 & PCU.TR2 & JMP START2
        ;
0024 BX1:    ALU YBUS,PASS & AB & WORD & OEY & CONTROL &
      /      DATAPATH ,ENTREG,,,,,LDMAR,,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.TR2 & JMP START2
        ;
        ;       BRANCH ON CONDITION             47      RX      CC: NONE
        ;       BC C,X2(D)              IF CC = 1, PC = (X2) + D
        ;                               ELSE PC = (PC) + 2
        ;
0025 BC:     ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,LDTREG,,,YMAR,LDMAR,,,ENZ0,,,BRIEN & TEST 57 &
      /      MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 ,OECT,,,,,, & PCU.NOP & CJP BX1
        ;
        ;       BRANCH NOT NEEDED
        ;
0026         ALU YBUS,PASS & AB & WORD & OEY & CONTROL &
      /      DATAPATH ZZI,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,,MWORD &
      /      AM2904 & PCU.NEXT & JMP RNI
        ;
        ;       BRANCH AND LINK                 45      RX      CC: NONE
        ;       BAL R1,X2(D)    R1 = PC + 2, PC = [(X2) + D]
        ;
0027 BAL:    ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /      DATAPATH ,,LDTREG,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT ,,,,MWORD &
      /      AM2904 & PCU.NOP & CONT
        ;
0028 BAL1:   ALU REG,PASS & DAB & CARRYCTL & WORD & CONTROL &
      /      DATAPATH ,,,,,PCUY,,,,,, & MEM.CONT REQB,,,MWORD &
      /      AM2904 & PCU.NOP & JMP BX1
        ;
        ;       BRANCH AND LINK REGISTER        85      RR      CC: NONE
        ;       BALR R1,R2      R1 = (PC), PC = (R2)
        ;
0029 BALR:   ALU YBUS,PASSR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,LDTREG,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT ,,,,MWORD &
      /      AM2904 & PCU.NOP & JMP BAL1
        ;
        ;       BRANCH REGISTER ALWAYS          86      RR      CC: NONE
        ;       BRA R1                  PC = (R1)
        ;
002A BR:     ALU YBUS,PASS & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /      DATAPATH ,,LDTREG,,,YMAR,LDMAR,,,,, & MEM.CONT REQB,,,MWORD &
      /      AM2904 & PCU.NOP & JMP BX1
        ;       ========================================
        ;
        ;              SHIFT INSTRUCTIONS
        ;
        ;       ========================================
        ;
        ;       SHIFT LEFT ARITHMETIC   8B      RSI     CC: CSVZ
        ;       SLA R1,CNT      R1 = SHIFT (R1) ARITHMETIC LEFT CNT PLACES
        ;
002B SLA:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
      /      AM2904 & PCU.NOP & LDCT
        ;
002C         ALU AUR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#F8 &
      /      DATAPATH & MEM.CONT ,,,MWORD &
      /      AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
        ;
002D         ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       SHIFT LEFT LOGICAL              89      RSI     CC: CSVZ
        ;       SLL R1,CNT      R1 = SHIFT (R1) LEFT LOGICAL CNT PLACES
        ;
002E SLL:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
      /      AM2904 & PCU.NOP & LDCT
        ;
002F         ALU LUR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#F8 &
      /      DATAPATH & MEM.CONT ,,,MWORD &
      /      AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
        ;
0030         ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       SHIFT RIGHT ARITHMETIC          8A      RSI     CC: CSVZ
        ;       SRA R1,CNT      R1 = SHIFT (R1) RIGHT ARITHMETIC CNT PLACES
        ;
0031 SRA:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
      /      AM2904 & PCU.NOP & LDCT
        ;
0032         ALU ADR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#E8 &
      /      DATAPATH & MEM.CONT ,,,MWORD &
      /      AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
        ;
0033         ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       SHIFT RIGHT LOGICAL             88      RSI     CC: CSVZ
        ;       SRL R1,CNT      R1 = SHIFT (R1) RIGHT LOGICAL CNT PLACES
        ;
0034 SRL:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
      /      AM2904 & PCU.NOP & LDCT
        ;
0035         ALU LDR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#E8 &
      /      DATAPATH & MEM.CONT ,,,MWORD &
      /      AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
        ;
0036         ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       ROTATE RIGHT                    A8      RSI     CC: CSVZ
        ;       RR R1,CNT       R1 = ROTATE (R1) RIGHT CNT PLACES
        ;
0037 RRL:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
      /      AM2904 & PCU.NOP & LDCT
        ;
0038         ALU LDR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#F8 &
      /      DATAPATH & MEM.CONT ,,,MWORD &
      /      AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
        ;
0039         ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       ROTATE LEFT                     A9      RSI     CC: CSVZ
        ;       RL R1,CNT       R1 = ROTATE (R1) LEFT CNT PLACES
        ;
003A RLL:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
      /      AM2904 & PCU.NOP & LDCT
        ;
003B         ALU LUR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#F8 &
      /      DATAPATH & MEM.CONT ,,,MWORD &
      /      AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
        ;
003C         ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       ROTATE RIGHT THROUGH CARRY      A9      RSI     CC: CSVZ
        ;       RRC R1,CNT
        ;
003D RRC:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
      /      AM2904 & PCU.NOP & IDCT
        ;
003E         ALU LDR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#E9 &
      /      DATAPATH & MEM.CONT ,,,MWORD &
      /      AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
        ;
003F         ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /      AM2904 & PCU.NEXT & JMAP
        ;
        ;       ROTATE LEFT THROUGH CARRY       AB      RSI     CC: CSVZ
        ;       RLC R1,CNT      ROTATE (R1) CNT TIME LEFT THROUGH CARRY
        ;
0040 RLC:    ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /      DATAPATH ZZI,,,,,,,,,,SHTCNTEN, & MEM.CONT ,,,MWORD &
```

Figure 31. Microprogram for 16-Bit Computer (Cont.)

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
MICROPROGRAM FOR 16 BIT COMPUTER

      /       AM2904 & PCU.NOP & LDCT
      ;
0041  /       ALU LUR,PASS & AB & WORD & OEY & CARRYCTL & CONTROL & CNTLB H#F9 &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 SHIFTEN,,EZ,EC,ES,EOVR,CEM, & PCU.NOP & RPCT $
      ;
0042  /       ALU YBUS,PASS & AB & WORD & OEY & CARRYCTL & CONTROL &
      /       DATAPATH ,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,MWORD &
      /       AM2904 & PCU.NEXT & JMAP
      ;
      ;       ====================================

      ;               I/O INSTRUCTIONS

      ;       ====================================

      ;       INPUT                   A0      RX      CC: NONE
      ;       IN R1,X2(D)     R1 = PORT (X2) + D
      ;
0043 IN:      ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /       DATAPATH ,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,,EREQ,,MWORD &
      /       AM2904 & PCU.NOP & LDCT H#001
      ;
0044  /       ALU & WORD & OEY & CONTROL ,IOEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,HREQ,,MWORD &
      /       AM2904 & PCU.NOP & RPCT $
      ;
0045  /       ALU & WORD & OEY & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH & MEM.CONT ,,HREQ,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
0046  /       ALU REG,PASSR & DAB & WORD & OEY & CARRYCTL & CONTROL &
      /       DATAPATH ,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;
      ;       OUTPUT L                A2      RX      CC: NONE
      ;       OUT R1,X2(D)    PORT (X2) + D = (R1)
      ;
0047 OUT:     ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /       DATAPATH ,,,,YMAR,LDMAR,,,ENZ0,,, & MEM.CONT REQB,,HREQ,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
0048  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD &
      /       CONTROL ,IOEN,, & CNTLB H#FB &
      /       DATAPATH ,,,,,,,LDD,,,, & MEM.CONT ,,HREQ,,MWORD &
      /       AM2904 & PCU.NOP & LDCT H#001
      ;
0049  /       ALU & WORD & CONTROL & OEY &
      /       DATAPATH & MEM.CONT ,,HREQ,,MWORD &
      /       AM2904 & PCU.NOP & RPCT $
      ;
004A  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF & OEY &
      /       DATAPATH & MEM.CONT ,,HREQ,,MWORD &
      /       AM2904 & PCU.NOP & JMP START
      ;
      ;       ====================================

      ;               STACK OPERATIONS

      ;       ====================================

      ;       PUSH REGISTERS                  C0      RR      CC: NONE
      ;       PUSH R1,RN      (SP - 2) = R1
      ;                       (SP - 4) = R2
      ;                       (SP - 2*N) = RN
      ;                       SP = SP - 2*N
      ;
004B PUSH:    ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ZZI,,,,,,,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
004C  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,LDMAR,LDD,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.PUSH & CONT
      ;
004D  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,,,INCTR,INC,,,,,,,, & MEM.CONT REQB,MREQ,,WRITE,MWORD &
      /       TEST Q#70 & AM2904 & PCU.NOP & CJP PUSH+1
      ;
004E  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,LDMAR,,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NOP & JMP RN1
      ;
      ;       POP REGISTERS                   C1      RR      CC: NONE
      ;       POP R2,R1       R2 = (SP)
      ;                       R1 = (SP + 2)
      ;                       RN = (SP + 2*N)
      ;                       SP = SP + 2*N
      ;
004F POP:     ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ZZI,,,,,,,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
0050  /       ALU REG,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,,,MWORD &
      /       AM2904 & PCU.SP & CONT
      ;
0051  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,ENCTR,,,,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       TEST Q#78 & AM2904 & PCU.POP & CJP POP+1
      ;
0052  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,LDMAR,,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NOP & JMP RN1
      ;
      ;       SUBROUTINE CALL         C2      RX      CC: NONE
      ;       CALL X1(D)      SP = SP - 2
      ;                       (SP - 2) = (PC)
      ;                       PC = [(X1) + D]
      ;
0053 CALL:    ALU YBUS,ADD & DAB & CARRYCTL & OEY & WORD & CONTROL & RTB &
      /       DATAPATE ,,LDTREG,,,,LDMAR,,,ENZ0,,, &
      /       MEM.CONT REQB,MREQ,,,MWORD & AM2904 & PCU.PUSH & CONT
      ;
0054  /       ALU YBUS,PASS & AB & CARRYCTL & WORD & CONTROL &
      /       DATAPATH ,,,,,PCUY,,LDD,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
0055  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,ENTREG,,,,,LDMAR,,,,, &
      /       MEM.CONT REQB,MREQ,,WRITE,MWORD & AM2904 & PCU.JUMP & JMP START1
      ;
      ;       RETURN FROM SUBROUTINE          C3              CC: NONE
      ;       RET             PC = (SP)
      ;                       SP = SP + 2
      ;
0056 RETURN:  ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,,LDMAR,,,,, & MEM.CONT REQB,,,MWORD &
      /       AM2904 & PCU.SP & CONT
      ;
0057  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,,LDMAR,,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.SP & CONT
      ;
0058  /       ALU YBUS,PASSR & DAB & CARRYCTL & OEY & WORD &'CONTROL &
      /       DATAPATH ,,LDTREG,,,YMAR,LDMAR,,,ENZ0,,, &
      /       MEM.CONT REQB,MREQ,,,MWORD & AM2904 & PCU.POP & CONT
      ;
0059  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,ENTREG,,,,,,,,,, & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU.JUMP & JMP START1
      ;
      ;       ====================================

      ;               INTERRUPT INSTRUCTIONS

      ;       ====================================

      ;       LOAD INTERRUPT MASK             CA      RI      CC: NONE
      ;       LIM DI          LOAD LOWER BYTE OF DI INTO MASK REGISTER
      ;
005A LIM:     ALU YBUS,PASSR & DAB & CARRYCTL & OEY & WORD &
      /       CONTROL ,,,INTRIEN & CNTLB H#FE &
      /       DATAPATH ,,,,,,LDMAR,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NEXT & JMP START2
      ;
      ;       ENABLE INTERRUPT                C8      CTL     CC: NONE
      ;       EI              ENABLE INTERRUPT SYSTEM
      ;
005B EI:      ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD &
      /       CONTROL ,,,INTRIEN & CNTLB H#FF &
      /       DATAPATH ZZI,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NEXT & JMAP
      ;
      ;       DISPABLE INTERRUPT              C9      CTL     CC:NONE
005C DI:      DI              DISABLE INTERRUPT SYSTEM
      /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD &
      /       CONTROL ,,,INTRIEN & CNTLB H#FD &
      /       DATAPATH ZZI,,,,,,LDMAR,,ZII,,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NEXT & JMAP
      ;
      ;       RETURN FROM INTERRUPT           CB      CTL     CC: (SP+2)
      ;       RTI             PC = (SP) ,PSW = (SP+2)
      ;                       SP = SP + 4, INTERRUPT ENABLED
      ;
005D RTI:     ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,,,,LDMAR,,,,, & MEM.CONT REQB,,,MWORD &
      /       AM2904 & PCU.SP & CONT
      ;
005E  /       ALU YBUS,PASSR & DAB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,,LDTREG,,,,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.POP & CONT
      ;
005F  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD & CONTROL &
      /       DATAPATH ,ENTREG,,,,,,,,,, & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU.JUMP & CONT
      ;
0060  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD &
      /       CONTROL ,,,INTRIEN & CNTLB H#F9 &
      /       DATAPATH ,,,,,, LDMAR,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.SP & CONT
      ;
0061  /       ALU YBUS,PASSR & DAB & CARRYCTL & OEY & WORD &
      /       CONTROL ,,,INTRIEN & CNTLB H#FF & TEST Q#00 &
      /       DATAPATH ,,,,,,,,ENZ0,,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 ,,EZ,EC,ES,EOVR,CEM, & PCU.POP & JMP START

      ;       ++++++++++++++++++++++++++++++++++++

      ;               INITIALIZATION ROUTINES

      ;       ++++++++++++++++++++++++++++++++++++

0180  ORG H#180

0180 INIT:    ALU REG,PASSR & DAB & WORD & OEY & CARRYCTL &
      /       DATAPATH & MEM.CONT ,,HREQ,,MWORD & CONTROL ROM,,, & CNTLB 00 &
      /       IMMD H#0000 & CONT
      ;       INITIALIZE REGISTERS IN AM29818
      ;       R0 = 0, R1 = 4000H, R4 = 2, AND R5 = 4
      ;
0181  /       ALU & WORD & CONTROL ,,,INTRIEN & CNTLB H#F0 &
      /       DATAPATH & MEM.CONT ,,HREQ,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#0000 & CONT
      ;
0182  /       ALU & WORD & CONTROL ,,,INTRIEN & CNTLB H#F8 &
      /       DATAPATH & MEM.CONT ,,HREQ,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A1,B1 & IMMD H#4000 & CONT
      ;
0183  /       ALU & WORD & PCU ,,PCUDZ,A4,B4 & CARRYCTL & DATAPATH &
      /       MEM.CONT ,,HREQ,,MWORD & IMMD H#0002 & CONT
      ;
0184  /       ALU & WORD & PCU ,,PCUDZ,A5,B5 & CARRYCTL & DATAPATH &
      /       MEM.CONT ,,HREQ,,MWORD & IMMD H#0004 & CONT
      ;
0185  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF & DATAPATH &
      /       MEM.CONT ,,HREQ,,MWORD & AM2904 & PCU.NOP & CONT
      ;
      ;       INITIALIZE CONSOLE AM9551
      ;
0186  /       ALU REG,PASSR & DAB & WORD & OEY & CARRYCTL &
      /       DATAPATH & MEM.CONT & CONTROL ROM,,, & CNTLB 10 &
      /       IMMD H#FFFB & CONT
      ;
0187  /       ALU REG,PASSR & DAB & WORD & OEY & CARRYCTL &
      /       DATAPATH & MEM.CONT & CONTROL ROM,,, & CNTLB 20 &
      /       IMMD H#00CE & CONT
      ;
0188  /       ALU & WORD & CONTROL & DATAPATH & MEM.CONT & JSB IOW
      ;
0189  /       ALU REG,PASSR & DAB & WORD & OEY & CARRYCTL &
      /       DATAPATH & MEM.CONT & CONTROL ROM,,, & CNTLB 20 &
      /       IMMD H#0035 & CONT
      ;
```

Figure 31. Microprogram for 16-Bit Computer (Cont.)


```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
MICROPROGRAM FOR 16 BIT COMPUTER

018A  /       ALU & WORD & CONTROL & DATAPATH & MEM.CONT & JSB IOW
      ;
018B  /       ALU & PCU.NOP & DATAPATH & MEM.CONT & JMP START
      ;
      ;       ################################

      ;               I/O WRITE SUBROUTINE
      ;       THE ADDRESS OF I/O PORT IS IN R1
      ;       THE DATA TO BE WRITTEN IS IN R2

      ;       ################################

018C IOW:     ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD &
      /       CONTROL ROM,,, & CNTLB 10 &
      /       DATAPATH ,,,,,YMAR,LDMAR,,,,,, & MEM.CONT REQB,,EREQ,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
018D  /       ALU YBUS,PASS & AB & CARRYCTL & OEY & WORD &
      /       CONTROL ROM,,, & CNTLB 20 &
      /       DATAPATH ,,,,,,,LDD,,,,, & MEM.CONT REQB,,HREQ,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
018E  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FB & DATAPATH &
      /       MEM.CONT REQB,,HREQ,,MWORD & AM2904 & PCU.NOP & LDCT H#001
      ;
018F  /       ALU & WORD & CONTROL & DATAPATH &
      /       MEM.CONT ,,HREQ,,MWORD & AM2904 & PCU.NOP & RPCT $
      ;
0190  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF & DATAPATH &
      /       MEM.CONT REQB,,HREQ,,MWORD & AM2904 & PCU.NOP & RTN
      ;
      ;       ++++++++++++++++++++++++++++++++++

      ;               VECTOR JUMP ENTRY POINTS

      ;       ++++++++++++++++++++++++++++++++++
      ;
01D0  ORG H#1D0
      ;
      ;       INTERRUPT 0, PC = 10H
      ;
01D0 INT0:    ALU & WORD & CONTROL ,IOEN,INTDIS, & CNTLB H#F7 &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#0010 & CONT
      ;
01D1  /       ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT & AM2904 & PCU.NOP & CONT
      ;
01D2  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;       INTERRUPT 1, PC = 14H
      ;
01D3 INT1:    ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#0014 & CONT
      ;
01D4  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,, & MEM.CONT REQB,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;       INTERRUPT 2, PC = 18H
      ;
01D5 INT2:    ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#0018 & CONT
      ;
01D6  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,, & MEM.CONT REQB,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;       INTERRUPT 3, PC = 1CH
      ;
01D7 INT3:    ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#001C & CONT
      ;
01D8  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;       INTERRUPT 4, PC = 20H
      ;
01D9 INT4:    ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#0020 & CONT
      ;
01DA  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;       INTERRUPT 5, PC = 24H
      ;
01DB INT5:    ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#0024 & CONT
      ;
01DC  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;       INTERRUPT 6, PC = 28H
      ;
01DD INT6:    ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#0028 & CONT
      ;
01DE  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;       INTERRUPT 7, PC = 2CH
      ;
01DF INT7:    ALU & WORD & CONTROL ,,INTDIS,INTRIEN & CNTLB H#FD &
      /       DATAPATH & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU ,,PCUDZ,A0,B0 & IMMD H#002C & CONT
      ;
01E0  /       ALU & WORD & CONTROL ,IOEN,, & CNTLB H#FF &
      /       DATAPATH ,,,,,,LDMAR,,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.NOP & JMP START1
      ;
      ;       ====================================

      ;               INTERRUPT HANDLER

      ;       ====================================

01F0  ORG H#1F0
      ;
01F0 INTR:    ALU & WORD & CONTROL &
      /       DATAPATH & MEM.CONT REQB,MREQ,,WRITE,MWORD &
      /       AM2904 & PCU.DEC4 & CONT
      ;
01F1  /       ALU & WORD & CONTROL &
      /       DATAPATH ,,,,,,LDMAR,,,,,, & MEM.CONT ,,,,MWORD &
      /       AM2904 & PCU.PUSH & CONT
      ;
01F2  /       ALU & WORD & CONTROL &
      /       DATAPATH ,,,,PCUY,,LDD,,,,, & MEM.CONT REQB,,,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
01F3  /       ALU & WORD & CONTROL ,,,INTRIEN & CNTLB H#F5  &
      /       DATAPATH & MEM.CONT REQB,MREQ,,WRITE,MWORD &
      /       AM2904 & PCU.NOP & JMPV
      ;
      ;       ************************************

      ;               INTERRUPT ENTRY POINT

      ;       ************************************
      ;
01FF  ORG H#1FF
      ;
01FF  /       ALU & WORD & CONTROL &
      /       DATAPATH ,,,,,,LDMAR,LDD,,,PSW,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.NOP & CONT
      ;
01FF  ORG H#1FF
      ;
01FF  /       ALU & WORD & CONTROL &
      /       DATAPATH ,,,,,,LDMAR,LDD,,,PSW,, & MEM.CONT REQB,MREQ,,,MWORD &
      /       AM2904 & PCU.PUSH & JMP INTR
      ;
              END
```

Figure 31. Microprogram for 16-Bit Computer (Cont.)

MICROCODE TRANSLATION

It is often convenient for the microprogrammer to assign microword fields such that they occupy positions that differ from those in the actual hardware implementation. This is often the case when the microprogrammer, for convenience, allocates bits according to the functions to be performed and then needs to translate the object code produced by AMDASM® to be consistent with the hardware microprogram memory design.

There is another instance where the ability to shift bit assignment is important to the engineer. As a given product evolves, bits may be added or deleted from the original microword format. When this occurs, a mapping function is desired to minimize hardware changes.

The program in SYSTEM/29® that performs such a mapping function is called AMSCRM. The AMSCRM maps the output of AMDASM (logical bit pattern) into the bit pattern that is consistent with the 16-bit computer hardware. A table of the logical to physical mapping is shown in Table 3.

ENGINEERING MODEL AND MACROCODE

With the proper tools – designing, microprogramming, prototyping, and checking out a new computer design is not overly difficult. The major tools used for the high-speed 16-bit design described in this application note was System 29[1]. System 29 is a software driven hardware prototyping system which allows microprogramming, hardware design/checkout, and macroprogramming (programming in the language of the target machine) to occur simultaneously. At the point where the design is reasonably rigid, and the hardware is mostly fabricated, System 29 allows the engineer to create "instant" microprograms to check out the new computers' internal data paths. Microprogram software support features of System 29 also allow the engineer to single cycle, single instruction step, instruction trace, and trap on pre-specified events coming true. Simultaneously with this initial internal check-out, the microcode for some very simple machine instruction should be written (i.e., load register, add register, or register, etc.). The next step is to check out the main memory paths with load and store instructions. At this point, a reasonable

Table 3.

```
                ***********************************

                BIT ASSIGNMENT FOR 16-BIT COMPUTER

                ***********************************

BIT POSITION    MNEMONIC  *              DESCRIPTION
LOG     PHY

 95     95      RTB       *     REG. FIELD 2 TO B PORT OF AM2903
 94             SPARE
 93             SPARE

 92     54      ZZI             LOAD Z REG. INTO ZI REG.
 91     94      CCEN      *     AM2910 CONDITON CODE ENABLE

                AM2903 ALU CONTROL BITS

 90     93      WORD      *     WORD MODE = 0, BYTE MODE = 1
 89     92      EA        *     ENABEL A LATCH ON AM2903
 88     91      OEY       *     ENABLE Y OUTPUT ON AM2903
 87     90      OEB       *     ENABLE B LATCH ON AM2903
86-78   89-81   I8-I0           INSTRUCTION LINES FOR AM2903

                DATAPATH BITS

 77     80      ENTREG    *     ENABLE TRANSFER REG
 76     79      LDTREG          LOAD TRANSFER REG.
 75     78      ENCTR     *     I-REG ENABLE COUNTER
 74     77      INC             I-REG INC=1/DEC=0
 73     76      PCUCD           PCU TRANSFER CHIP DISABLE
 72     75      PCUY            PCU TRANSFER TO Y-BUS
 71     74      LDMAR           LOAD MEMORY ADDRESS REGISTER
 70     73      LDD             LOAD D-REGISTER
 69     72      ZII             LOAD ZI INTO I REGISTER
 68     71      ENZ0      *     ENABLE Z0 REGISTER TO DA BUS
 67     70      PSW       *     ENABLE PSW REGISTER TO DA BUS
 66     69      SHTCNEN   *     SHIFT COUNT AM2910 ADDRESS
 65     68      BRIEN     *     BRANCH INSTRUCTION ENABLE

                AM2901A PROGRAM CONTROL UNIT

 64     67      PCUI7           F TO B-RAM = 1 (DEFAULT), F TO Q-REG = 0
 63     66      PCUI3           ADD = 1 (DEFAULT), SUB = 0
62-60   65-63   PCUI2-0         PCU SOURCE CONTROL
59-57   62-60   PCUA2-0         PCU A-RAM SELECT
56-54   59-57   PCUB2-0         PCU B-RAM SELECT

                MEMORY CONTROL

 53     52      REQB            BUS REQUEST
 52     51      MREQ            MEMORY REQUEST
 51     50      HREQ      *     HOLD REQUEST
 50     49      WRITE           MEMORY READ = 0 (DEFAULT), MEMORY WRITE = 1
 49     48      MWORD           MEMORY BYTE OP = 0 (DEFAULT), MEMORY WORD OP = 1

                CONTROL BIT STROBES

 48     56      IMMD      *     ENABLE IMMEDIATE FIELD TO DA BUS
 47     47      ROM             I-REG ENABLE = 0 (DEFAULT), ROM ENABLE = 1
 46     46      IOEN      *     I/O CONTROL REGISTER ENABLE
 45     45      INTDIS    *     AM2914 INTERRUPTS DISABLE
 44     44      INTRIEN   *     AM2914 INSTRUCTION ENABLE
 43     43      SHFTEN    *     AM2904 SHIFT ENABLE

                GENERAL PURPOSE CONTROL BITS

42-35   42-35   CNTLB7-0        BITS TO BE STROBED BY CONTROL STROBES

                AM2904 STATUS REGISTER CONTROL BITS

 34     34      OECT      *     OUTPUT ENABLE OF CONDTIONAL TEST
 33     33      EZ        *     ENABLE ZERO FLAG UPDATE
 32     32      EC        *     ENABLE CARRY FLAG UPDATE
 31     31      ES        *     ENABLE SIGN FLAG UPDATE
 30     30      EOVR      *     ENABLE OVERFLOW FLAG UPDATE
 29     29      CEM       *     ENABLE MACHINE STATUS REGISTER
 28     28      CEU       *     ENABLE MICROPROGARM STATUS REGISTER
 27     27      I12             AM2904 I12 CARRY OUT CONTROL
 26     26      I11             AM2904 CARRY OUT CONTROL

                TEST BITS

25-23   25-23   TEST5-3         AM2904 TEST BITS
22-20   22-20   TEST2-0         AM2904 & AM26LS251 TEST BITS

                AM2910 SEQUENCE CONTROL

19-16   19-16   NAC3-0          AM2910 NEXT ADDRESS CONTROL

                NEXT MICRO ADDRESS OR IMMEDIATE FIELD

15-0    15-0    M15-0           SHARED FIELD FOR NEXT ADDRESS OR IMMD

                END
```


instruction sub-set should be microprogrammed (a phase 1 instruction set) that will allow a simple monitor to be written in the target machines's language. This monitor should run on the target machine and provide commands for: memory display, memory store and jump to memory location. The phase 1 instruction set and simple monitor now provides the basic foundation for completing the full computer design.

The standard System 29 configuration provides automatically for microcode and hardware development. In order to efficiently develop and implement the target machine's software, a target machine assembler and a mechanism for loading the machine's main memory must be provided. System 29 uses an Am9080A microprocessor, dual floppy disks, and a full function disk operating system to support microprogrammed hardware and firmware development. The Am9080A microprocessor can address 64k bytes of memory. The disk operating system uses only the first 32k bytes and the remaining 32k is used to memory map (page) functions from the hardware development side. Through this mechanism, the designer has the ability to directly load and manipulate microprograms, monitor hardware functions, etc. There are extra enable lines from the page register which allow the System 29 user to map other functions into the support processor's upper 32k of memory.

The main memory of this 16-bit high-speed computer design was mapped into the support processors upper 32k via one of the unused page register enable lines. Besides the normal 16-bit interface, a simple 8-bit interface was added to the main memory thus making it a simple two port memory. When the 16-bit computer is halted (via a System 29 command) location 0 of 16-bit main memory would be addressed as location 8000 hex of System 29 support processor memory. Location 1 would be 8001, 2 would be 8002, etc. This affected a mechanical link between the 16-bit prototype design and System 29.

In order to efficiently write a reasonably complex piece of software (such as a simple monitor), an assembler for the target instruction set is needed. Since this 16-bit computer design is not exactly like any other 16-bit computer, ready to run software tools are not available. A macro assembler is available as an optional enhancement to the System 29 software base. Even though this macro assembler is for programming in Am9080A assembly language, there is a user installable patch which will disable all of the Am9080A operation codes (Figure 32). With this patch installed, the user may now write a macro library defining the target machine's instruction set. It is not necessary to code the entire instruction set, as the first level of programming for the new machine (simple monitor, etc.) will be using only the phase 1 instruction set. A complete macro library of the AMD high-speed 16-bit computer phase 1 instruction set is contained in Appendix B.

Now that the tools are in place, it is relatively simple to code and implement a simple monitor for the target machine. Appendix C contains the complete simple monitor listing for the AMD high-speed 16-bit computer. Only the phase 1 instruction set was used which does not include byte instruction, call and return instructions, stack instructions, any special instructions, etc. This simple monitor understands three commands: Display (D), Store (S), and Jump (J). Typing D followed by an address value will display 256 bytes of main memory beginning on the address given (rounded back to the nearest eight word boundary). Typing an S followed by an address, followed by data, will store the data consecutively, on a nibble basis beginning at the given address. Typing in J followed by an address will cause the processor to begin execution at the main memory location given by the address. Commands, addresses, and data must be separated by at least one delimiter (space, comma, or period).

The change file shown below can be integrated into MAC to produce a new program, which we will call MAC29. The MAC29 program will not recognize 8080 mnemonics, but will recognize all the MAC pseudo operators and arithmetic functions.

```
                ;
                ;       MACRO ASSEMBLER "MAC" CHANGES TO DISABLE 8080 OPCODES.
                ;
                ;
0019 =          RT      EQU     25      ;8080 REGISTER NAME
001A =          PT      EQU     26      ;PSEUDO OPCODE TYPE
2561 =          TAREA   EQU     2561H   ;FREE AREA IN TOKEN MODULE
                ;
2444                    ORG     2444H   ;OVERLAY INX H  MOV B,M  RET
2444 C36125             JMP     TAREA
                ;
2561                    ORG     TAREA
                ;       TYPE IS IN THE ACCUMULATOR
2561 FE19               CPI     RT      ;BELOW RT IF ARITH OP
2563 DA6925             JC      TYPEOK
2566 FE1A               CPI     PT      ;PSEUDO OP?
2568 C0                 RNZ             ;RETURN WITH NON-ZERO FLAG
                ;       OTHERWISE, PSEUDO OP OR ARITH OP
2569 23         TYPEOK:         INX     H
256A 46                 MOV     B,M
256B BF                 CMP     A       ;SET ZERO FLAG
256C C9                 RET
                ;
256D                    END
```

Figure 32. Macro Assembler Disable Opcode Patch.

After writing the monitor, and putting it onto floppy disks via the System 29 editor, it must be assembled using the modified macro assember (described earlier). The result of the assembly is a hex file which is suitable for loading into the 16-bit computer's main memory. This hex file is now loaded into support processor memory beginning at location 8000 hex. As discussed previously, this is mapped at location zero in the 16-bit computer's main memory. Assuming the microcode is loaded and a terminal is connected to the 16-bit computer, the monitor in 16-bit main memory may now be executed. The complete System 29 session from editing and assembling the monitor to loading and executing it is given in Appendix D.

SUMMARY

As can be seen throughout these application notes, designing a high performance Bipolar microprocessor system is a straightforward task. The Am2900 Family is ideally suited to provide building blocks for the various elements of the computer. These include the Computer Control Unit, the Central Processing Unit, the Program Control Unit, the Interrupt Structure and the various bus controls. Together, these elements allow the designer to build computers using the current state-of-the-art architecture with LSI building blocks.

As technology improves, Advanced Micro Devices has been able to redesign these building blocks to offer increased performance. Thus, the Am2901 has evolved through an Am2901A, then an Am2901B and now an Am2901C is in the planning. In addition, the Am2903 offers additional architectural advantages and soon an Am29103 will provide additional speed and performance features. Similarly, the microprogram sequencer area began with the Am2909 and Am2911; then was followed by the larger Am2910. Soon, the Am2909A and Am2911A will provide higher speed in the microprogram sequencer area and will be followed by an Am2910A.

Thus, the future for Bipolar LSI building blocks includes not only more advanced product designs offering higher levels of integration and new functions for new architectures, but also offers higher performance versions of the already existing products. Advanced Micro Devices is committed to providing high performance Bipolar LSI circuits utilizing proven technology designed to operate over the full military operating range as well as the commercial operating range. As always, these products continue to meet the performance requirements of MIL-STD-883.

APPENDIX A
Complete Description of Instructions

LOAD

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The second operand is loaded into the general register specified by R_1.

STORE

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The first operand specified by R_1 is stored at the location specified by the second operand.

ADD

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The first operand is added to the second operand and replaces the first operand.

ADD WITH CARRY

RR

OP	R_1	R_2

RX

OP	R_1	X_2	d

The first operand (16 bits) with carry is added to the second operand and replaces the first operand.

SUBTRACT

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The second operand is subtracted from the first operand and replaces the first operand.

SUBTRACT WITH CARRY

RR

OP	R_1	R_2

RX

OP	R_1	R_2	d

The second operand (16 bits) with carry is subtracted from the first operand and replaces the first operand.

AND

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The AND of the first operand and the second operand replaces the first operand.

OR

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The OR of the first operand and the second operand replaces the first operand.

XOR

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The logical difference of the first operand and the second operand replaces the first operand.

TEST IMMEDIATE

RX, RSI

OP	R_1	X_2	d

The first operand and the second operand are logically ANDed. The contents of R_1 and X_2 are unchanged.

COMPARE

RR, RS, SS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The first operand is algebraically compared with the second operand. The result is indicated by the condition code.

COMPARE LOGICAL

RR, RS, SS

OP	R_1	R_2

The first operand is compared logically to the second operand. The result is indicated by the condition code.

MULTIPLY

RR

OP	R_1	R_2

RX

OP	R_1	X_2	d

The first operand (R_1 + 1) is multiplied by the second operand and the 32-bit product is contained in R_1 and R_1 + 1 registers. R_1 must be an even address. The sign of the product is determined by the rules of algebra.

MULTIPLY UNSIGNED

RR

OP	R_1	R_2

RX

OP	R_1	X_2	d

The first operand (R_1 + 1) is multiplied by the second operand and the 32-bit product is contained in R_1 and (R_1 + 1). R_1 must be even.

LOAD BYTE

RR

OP	R_1	R_2

RX, RXI

OP	R_1	X_2	d

The 8-bit byte stored in the low order byte of the second operand location is stored in the low order byte of R_1. The high order byte of the R_1 is set to zero.

INSERT CHARACTER

RR, RS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The byte at the second operand location is loaded into the low order byte of R_1 without changing the contents of the high order byte of R_1.

STORE CHARACTER
STORE BYTE

RR, RS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The least significant byte of the first operand is stored in the location specified by the second operand. The other byte of the second location is unchanged.

EXCHANGE BYTE

RR, RS

OP	R_1	R_2

RX

OP	R_1	X_2	d

The bytes specified by the first and second operands are exchanged. When the operand specifies a register (i.e. R_1, R_2) only the low order byte is exchanged.

BYTE SWAP

RR, RS

OP	R_1	R_2

RX

OP	R_1	R_2	d

The two bytes of the second operand are swapped and loaded into the register specified by the first operand.

COMPARE LOGICAL BYTE

RR, RS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The low order byte of the first and second operands are compared. The result is indicated in the condition code.

AND BYTE

RR, RS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The AND of the low order bytes specified by the first second operands replace the first operand low order byte. The high order byte of R_1 is set to zeros.

OR BYTE

RR, RS

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The OR of the low order bytes specified by the first and second operands replace the first operand low order byte. The high order byte of R_1 is set to zero.

XOR BYTE

RR, RX

OP	R_1	R_2

RX, RSI

OP	R_1	X_2	d

The XOR of the low order bytes specified by the first and second operands replace the first operand low order byte. The high order byte of R_1 is set to zero.

LOAD PROGRAM STATUS WORD

RX

	R_2	d

A 32-bit new PSW is loaded from the memory location specified by the second operand as the current PSW.

EXCHANGE PROGRAM STATUS

RR

	R_1	R_2

PSW (0:15) → (R_1)
R2 → PSW (0:15)

STORE PROGRAM STATUS WORD

RX

OP	R_2	d

The 32-bit PSW is stored at the location specified by the second operand.

SUPERVISOR CALL

RX

OP	R_1	X_2	d

OLD PSW → $[(X_2) + d]$
$[(X_2) + d] + 4$ → NEW PSW

SET, CLR, COMPLEMENT, TEST BIT PSW

OP	N

The condition flags in the current PSW are set, cleared, complemented, or tested. N defines the bit(s) to be affected or tested.

CALL

RX

OP	R_1	R_2	d

Jump to the memory location specified by the second operand and push PSW (16:31) onto stack.

RETURN

OP	X	X

POP STACK
STACK → PSW (16:31)

PUSH

OP	X	X

PSW, R_0-R_{15} → STACK

POP

OP	X	X

STACK → PSW, R_0-R_{15}

P/PUSH

RR

OP	R_1	R_2

R_1 THRU R_2 → STACK

P/POP

RR

OP	R_1	R_2

STACK → R_1 THRU R_2

LOAD STACK POINTER
LOAD STACK LIMIT LOWER
LOAD STACK LIMIT UPPER

RX

OP	R_2	d

STORE STACK POINTER
STORE STACK LIMIT LOWER
STORE STACK LIMIT UPPER

RX

OP	R_2	d

The stack point, stack limit lower or upper is read from or written into the address defined by the second operand.

TRANSLATE

RR

OP	R_1	R_2

R_1

LENGTH
ADDRESS

R_2

ADDRESS

The addresses specified by $R_1 + 1$ and R_2 define two tables, $R_1 + 1$ address is the top location of a table to be translated, R_2 address the first location of the translation table. The value (one byte) pointed to be the $R_1 + 1$ address is indexed by (added to) the address value of R_2 to find the translation code. This translation code replaces the value pointed to by the $R_1 + 1$ address. After one byte is translated, the length is decremented and the address of $R_1 + 1$ incremented and the instruction repeated, until the length equals zero. This instruction is interruptable. If this instruction is interrupted, the PC is left pointing to this instruction so that this instruction can be resumed after the interrupt service is complete.

TRANSLATE AND TEST

RR

OP	R_1	R_2

R_1

LENGTH
ADDRESS

R_2

ADDRESS

This instruction proceeds like translate except that the bytes of the first operand (defined by R_1) are not changed in storage. When the bytes of the translate table (R_2) the instruction proceeds to the next byte of the first operand. If the byte of the translate table is not zero, the instruction is halted with the address pointed to last in the translate table held in register 1.

MOVE LONG

RR

OP	R_1	R_2

R_1

LENGTH
ADDRESS

R_2

ADDRESS

Moves bytes defined by R_1 to R_2. Both adresses incremented after each transfer. This instruction is interruptable.

COMPARE LONG

RR

OP	R_1	R_2

R_1

LENGTH
ADDRESS

R_2

ADDRESS

Compares the first operand against the second operand. The length is decremented and the address incremented after each compare. When length = zero of the bytes compared are not equal, the instruction is halted.

EXECUTE

RX

OP	R_1	X_2	d

The upper 16 bits of the instruction at the second operand is 'OR'ed with R_1 and executed.

DECIMAL ADD

OP	R_1	R_2

OP	R_1	X_2	d

Nibbles in operand 1 and operand 2 are added. The result is placed in operand one.

DECIMAL SUBTRACT

	R_1	R_2

	R_1	X_2	d

Nibbles in operand 2 are subtracted from nibbles in operand 1 and the result is placed in operand 1.

DECREMENT INDEXES

RR

OP	R_1	R_2

$R_1 - 1 \rightarrow R_1$
$R_2 - 1 \rightarrow R_2$

One is subtracted from R_1 and the result placed back into R_1. One is subtracted from R_2 and the result placed back into R_2. R_1 and R_2 may specify the same register with will effectively subtract two from that register.

SHIFT RIGHT ARITHMETIC
SHIFT RIGHT DOUBLE ARITHMETIC

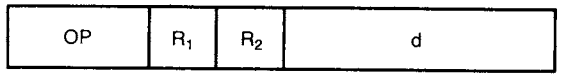
RX, RSI

OP	R_1	R_2	d

The contents of R_1 for single shifts and R_1, $R_1 + 1$ for double shifts are shifted the number of places specified by the second operand. The sign bit is unchanged. Bits shifted in are set equal to the sign bit. Bits shifted out are shifted through the carry bit.

ROTATE RIGHT
ROTATE RIGHT DOUBLE

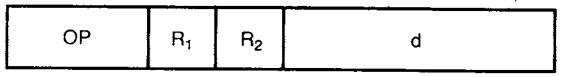
RX, RSI



The contents of R_1 for single shifts and R_1, $R_1 + 1$ for double shifts are rotated right the number of places specified by the second operand.

SHIFT LEFT ARITHMETIC
SHIFT LEFT DOUBLE ARITHMETIC

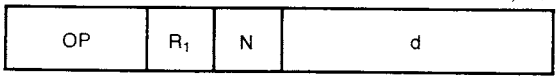
RX, RSI



The contents of R_1 for single shifts and $R_1 + 1$ for double shifts are shifted left the number of places specified by the second operand. The high order bit (sign bit) of the register a register pair is unaffected by the shift. Low order bits are filled with zeros. If a bit unlike the sign bit is shifted out of the position adjacent to the sign bit, the overflow flag is set.

ROTATE LEFT

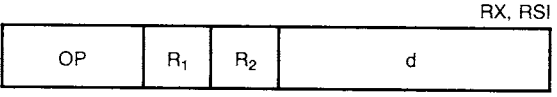
RX, RSI



The contents of R_1 for single shifts and R_1, $R_1 + 1$ for double shifts are rotated left, the number of places specified by the second operand.

SHIFT RIGHT LOGICAL
SHIFT RIGHT DOUBLE LOGICAL

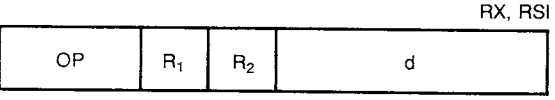
RX, RSI



The contents of R_1 for single shifts and $R_1 + 1$ for double shifts are shifted right the number of places specified by the second operand. High order bits shifted in are zeros, low order bits shifted out are shifted through the carry bit.

SHIFT LEFT LOGICAL
SHIFT LEFT DOUBLE LOGICAL

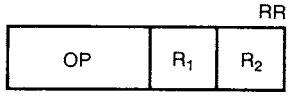
RX, RSI



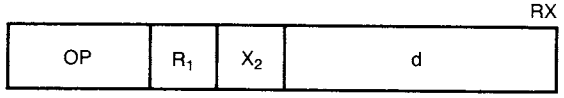
The contents of R_1 for single shifts and R_1, $R_1 + 1$ for double shifts are shifted left the number of positions specified by the second operand. High order bits shifted out are shifted through the carry bit. Zeros are shifted in. R_1 for double shifts must be even.

INPUT WORD

RR



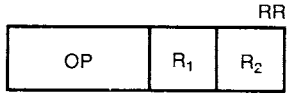
RX



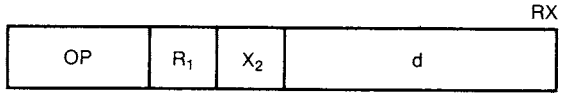
One 16-bit word of data is read into the first operand from the device which is addressed by the contents of the second operand.

INPUT BYTE

RR



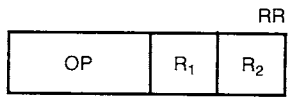
RX



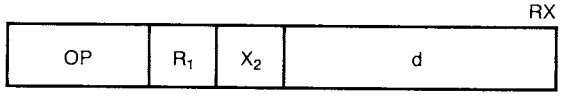
One byte of data is read into the low order 8 bits of the first operand from the device which is addressed by the contents of the second operand.

OUTPUT WORD

RR



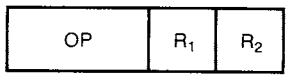
RX



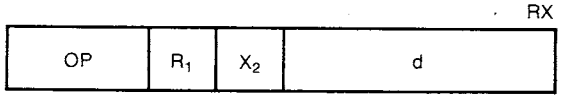
The 16 bits of R_1 is sent to the device which is addressed by the contents of the second operand.

OUTPUT BYTE

RR



RX

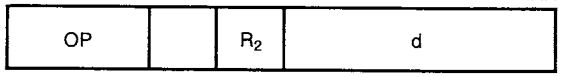


The low order 8 bits of R_1 is sent to the device which is addressed by the contents of the second operand.

BRANCH

RS

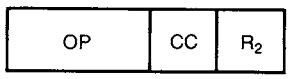
RX



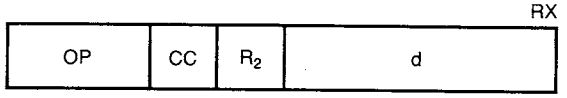
Unconditionally branch to the location specified by the second operand. The first operand is not used.

BRANCH ON CONDITION

RS



RX



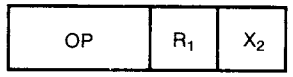
Branch to the location specified by the second operand if the condition code specified in the first operand postion is equal to the current PSW status bits.

Condition codes are:

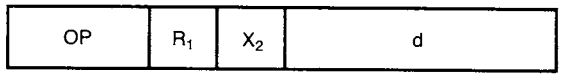
Carry	= B	(Sign=0)	
No Carry	= A	Minus	= F
Zero	= 5	(Sign=1)	
Not Zero	= 4	1's Comp>	= 9
2's Comp>	= 0	1's Comp<	= 8
2's Comp<	= 3	1's Comp>	= C
2's Comp>	= 2	1's Comp<	= D
2's Comp<	= 1	Overflow	= 7
Plus	= E	Not Overflow	= 6

BRANCH AND LINK

BALR RS



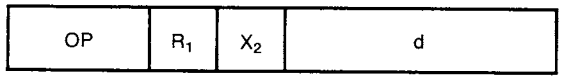
BAL RX



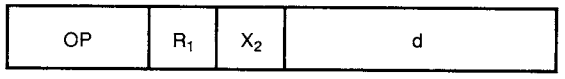
The address of the next sequential instruction is saved in R_1, and an unconditional branch to the jump address is taken.

BRANCH ON INDEX

BXH HIGH RX



BXLE LOW OR EQUAL RX



R_1 is incremented by the value in $R_1 + 1$, and logically compared to the index limit held in R_1+2.

INDEX HIGH

$(R_1) + (R_1 + 1) \rightarrow (R_1)$
$(R_1) : (R_1 + 2)$
IF $(R_1) > (R_1 + 2)$ THEN $d + (X_2) \rightarrow$ PSW (16:31)
IF $(R_1) \leq (R_1 + 2)$ THEN PSW (16:31) + 2 → PSW (16:31)

INDEX LOW OR EQUAL

$(R_1) + (R_1 + 1) \rightarrow (R_1)$
$(R_1) : (R_1 + 2)$
$(R_1) \leq (R_1 + 2)$ THEN $d + (X_2) \rightarrow$ PSW (16:31)
IF $(R_1) > (R_1 + 2)$ THEN PSW (16:31) + 2 → PSW (16:31)

APPENDIX B

```
;       ********************************
;
;        MACRO DEFINITIONS FOR MICRO/29
;
;       ********************************
;
;       DEFINITIONS FOR CPU REGISTERS
;
R0      SET 0
R1      SET 1
R2      SET 2
R3      SET 3
R4      SET 4
R5      SET 5
R6      SET 6
R7      SET 7
R8      SET 8
R9      SET 9
R10     SET 10
R11     SET 11
R12     SET 12
R13     SET 13
R14     SET 14
R15     SET 15
X0      SET 0
X1      SET 1
X2      SET 2
X3      SET 3
X4      SET 4
X5      SET 5
X6      SET 6
X7      SET 7
X8      SET 8
X9      SET 9
X10     SET 10
X11     SET 11
X12     SET 12
X13     SET 13
X14     SET 14
X15     SET 15
;
;       PRESET CONDITION CODES
;
CY?     SET 0BH         ;CARRY
NC?     SET 0AH         ;NO CARRY
Z?      SET 05H         ;ZERO
NZ?     SET 04H         ;NOT ZERO
GT?     SET 00H         ;2'S COMP GREATER THAN
LT?     SET 03H         ;2'S COMP. LESS THAN
GE?     SET 02H         ;2'S COMP. GREATER THAN OR EQUAL TO
LE?     SET 01H         ;2'S COMP. LESS THAN OR EQUAL TO
PL?     SET 0EH         ;PLUS, SIGN = 0
MI?     SET 0FH         ;MINUS, SIGN = 1
HI?     SET 09H         ;1'S COMP. HIGHER
LS?     SET 08H         ;1'S COMP. LOWER OR SAME
HS?     SET 0CH         ;1'S COMP. HIGHTER OR SAME
LO?     SET 0DH         ;1'S COMP. LOWER
OV?     SET 07H         ;OVERFLOW
NV?     SET 06H         ;NOT OVERFLOW
;
;       ================================
;
;            RR TYPE INSTRUCTIONS
;
;       ================================
;
;       LR      LOAD REGISTER                   18
;
LR      MACRO R1,R2
        DB 18H,R1*10H+R2
        ENDM
;
;       AR      ADD REGISTER                    1A
;
AR      MACRO R1,R2
        DB 1AH,R1*10H+R2
        ENDM
;
;       SR      SUBTRACT REGISTER               1B
;
SR      MACRO R1,R2
        DB 1BH,R1*10H+R2
        ENDM
;
;       NR      AND REGISTERS                   14
;
NR      MACRO R1,R2
        DB 14H,R1*10H+R2
        ENDM
;
;       ORR     OR REGISTERS                    16
;
ORR     MACRO R1,R2
        DB 16H,R1*10H+R2
        ENDM
;
;       CLR     COMPARE LOGICAL REGISTERS       15
;
CLR     MACRO R1,R2
        DB 15H,R1*10H+R2
        ENDM
;
;       XR      EXCLUSIVE OR REGISTERS          17
;
XR      MACRO R1,R2
        DB 17H,R1*10H+R2
        ENDM


        ================================

              RX TYPE INSTRUCTIONS

        ================================

;       LD      LOAD MEMORY                     58
;
LD      MACRO R1,X2,DI
        DB 58H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       ST      STORE IN MEMORY                 50
;
ST      MACRO R1,X2,DI
        DB 50H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       ADD     ADD FROM MEMORY                 5A
;
ADD     MACRO R1,X2,DI
        DB 5AH,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       SUB     SUBTRACT FROM MEMORY            5B
;
SUB     MACRO R1,X2,DI
        DB 5BH,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       N       AND WITH MEMORY                 54
;
N       MACRO R1,X2,DI
        DB 54H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       O       OR WITH MEMORY                  56
;
O       MACRO R1,X2,DI
        DB 56H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       CMP     COMPARE WITH MEMORY             55
;
CMP     MACRO R1,X2,DI
        DB 55H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM


;       ========================================
;
;               IMMEDIATE INSTRUCTIONS
;
;       ========================================
;
;       LI      LOAD IMMEDIATE                  41
;
LI      MACRO R1,I2
        DB 41H,R1*10H,(I2) SHR 8,(I2) AND 0FFH
        ENDM
;
;       NI      AND IMMEDIATE                   94
;
NI      MACRO R1,I2
        DB 94H,R1*10H,(I2) SHR 8,(I2) AND 0FFH
        ENDM
;
;       OI      OR IMMEDIATE                    96
;
OI      MACRO R1,I2
        DB 96H,R1*10H,(I2) SHR 8,(I2) AND 0FFH
        ENDM
;
;       XI      EXCLUSIVE OR IMMEDIATE          97
;
XI      MACRO R1,I2
        DB 97H,R1*10H,(I2) SHR 8,(I2) AND 0FFH
        ENDM
;
;       AI      ADD IMMEDIATE                   9A
;
AI      MACRO R1,I2
        DB 9AH,R1*10H,(I2) SHR 8,(I2) AND 0FFH
        ENDM
;
;       SI      SUBTRACT IMMEDIATE              9B
;
SI      MACRO R1,I2
        DB 9BH,R1*10H,(I2) SHR 8,(I2) AND 0FFH
        ENDM
;
;       CI      COMPARE IMMEDIATE               95
;
CI      MACRO R1,I2
        DB 95H,R1*10H,(I2) SHR 8,(I2) AND 0FFH
        ENDM

;       ========================================
;
;       BRANCH AND CONITIONAL BRANCH INSTRUCTIONS
;
;       ========================================
;
;       BX      UNCONDITIONAL BRANCH            74
;
BX      MACRO X1,DI
        DB 74H,X1*10H,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       BC      CONDTIONAL BRANCH               47
;
BC      MACRO CC,X2,DI
        DB 47H,CC*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       BAL     BRANCH AND LINK                 45
;
BAL     MACRO R1,X2,DI
        DB 45H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       BALR    BRANCH AND LINK REGISTER        05
;
BALR    MACRO R1,R2
        DB 05H,R1*10H+R2
        ENDM
;
;       BR      BRANCH REGISTER UNCONDITONAL    04
;
BR      MACRO R1
        DB 04H,R1*10H
        ENDM
;
;       ========================================
;
;           SHIFT AND ROTATE INSTRUCTIONS
;
;       ========================================
;
;       SLA     SHIFT LEFT ARITHMETIC           8B
;
SLA     MACRO R1,CT
        DB 8BH,R1*10H+(CT-1)
        ENDM
;
;       SRL     SHIFT RIGHT LOGICAL             88
;
SRL     MACRO R1,CT
        DB 88H,R1*10H+(CT-1)
        ENDM
;
;       SLL     SHIFT LEFT LOGICAL              89
;
SLL     MACRO R1,CT
        DB 89H,R1*10H+(CT-1)
        ENDM
;
;       SRA     SHIFT RIGHT ARITHMETIC          8A
;
SRA     MACRO R1,CT
        DB 8AH,R1*10H+(CT-1)
        ENDM
;
;       RRL     ROTATE RIGHT                    A8
;
RRL     MACRO R1,CT
        DB 0A8H,R1*10H+(CT-1)
        ENDM
;
;       RLL     ROTATE LEFT                     AA
;
RLL     MACRO R1,CT
        DB 0AAH,R1*10H+(CT-1)
        ENDM
;
;       ================================
;
;               I/O INSTRUCTIONS
;
;       ================================
;
;       IN      INPUT                           A0
;
IN      MACRO R1,X2,DI
        DB 0A0H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
;       OUT     OUTPUT                          A2
;
OUT     MACRO R1,X2,DI
        DB 0A2H,R1*10H+X2,(DI) SHR 8,(DI) AND 0FFH
        ENDM
;
```

APPENDIX C

```
                        $*MACRO
                        ;
                        ; SIMPLE MONITOR FOR THE AMD HIGH-SPEED 16-BIT COMPUTER
                        ;
                        ;       BY:     JIM BRICK
                        ;
                        ;
                        MACLIB  MICRO29
                        ;
FFFA =          DATA:   EQU     0FFFAH  ;USART DATA PORT ADDRESS
FFFB =          STATUS: EQU     0FFFBH  ;USART CONTROL PORT ADDRESS
0A0D =          CRLF:   EQU     0A0DH   ;LINE-FEED, CARRIAGE RETURN
                        ;
0000                    ORG     0
                        ;
0000+1700       BEGIN:  XR      R0,R0           ;CLEAR R0
                        BAL     R14,R0,DOCRLF   ;NEW LINE ON CONSOLE
0002+45E000028  MONLP:  BAL     R14,R0,PROMPT   ;I/P PROMPT
0006+45E00016           BAL     R14,R0,GETIP    ;GET USERS I/P
000A+45E00040           BAL     R14,R0,SCANER   ;DECODE & EXECUTE COMMAND
000E+45E0008C           BX      R0,MONLP        ;REPEAT LOOP FOREVER
                        ;
0012+74000006   PROMPT: ST      R14,R0,@PROMPT  ;SAVE RET
0016+50E003EA           LI      R2,'>'          ;PROMPT CHARACTER '>'
001A+4120003E           BAL     R14,R0,CRTOUT   ;PROMPT TO CRT
001E+45E003D8           LD      R14,R0,@PROMPT  ;RESTORE RET
0022+58E003EA           BR      R14             ;
0026+04E0               ;
0028+50E003EC   DOCRLF: ST      R14,R0,@DOCRLF  ;SAVE RET
002C+41200A0D           LI      R2,CRLF         ;CR/LF CODES
0030+45E003D8           BAL     R14,R0,CRTOUT   ;O/P LF
0034+8827               SRL     R2,8            ;GET CR CODE
0036+45E003D8           BAL     R14,R0,CRTOUT   ;O/P CR
003A+58E003EC           LD      R14,R0,@DOCRLF  ;RESTORE RET
003E+04E0               BR      R14             ;
                        ;
0040+50E003EE   GETIP:  ST      R14,R0,@GETIP   ;SAVE RET
0044+4130040E           LI      R3,BUFFER       ;A(I/P BUFFER)
0048+41400080           LI      R4,0080H        ;MAX I/P COUNT
004C+45E00398   IPLP:   BAL     R14,R0,GETCHR   ;GET NEXT I/P CHARACTER
0050+8917               SLL     R1,8            ;POSITION I/P CHAR TO HI BYTE
0052+1851               LR      R5,R1           ;SAVE HI BYTE
0054+16FF               ORR     R15,R15         ;TEST RET CODE
0056+47500082           BC      Z?,R0,DOEOF     ;TO EOF IF RC = ZERO
005A+45E00398           BAL     R14,R0,GETCHR   ;NEXT CHARACTER
005E+941000FF           NI      R1,00FFH        ;SAVE ONLY I/P CHARACTER IN LC
0062+1651               ORR     R5,R1           ;COMBINE TWO BYTES FOR ONE WORD
0064+16FF               ORR     R15,R15         ;TEST RET CODE
0066+47500082           BC      Z?,R0,DOEOF     ;TO EOF IF RC = ZERO
006A+58530000           ST      R5,R3,0         ;DATA TO I/P BUFFER
006E+9A300002           AI      R3,0002         ;TO NEXT BUFER SLOT
0072+9B400002           SI      R4,0002         ;COUNT-2
0076+4750007E           BC      Z?,R0,DOEOF2    ;STOP IF MAX I/P
007A+74000040           BX      R0,IPLP         ;CONTINUE GETTING I/P
007E+41500D00   DOEOF2: LI      R5,0D00H        ;EOF AFTER MAX LINE
0082+50530000   DOEOF:  ST      R5,R3,0         ;DATA/EOF TO BUFFER
0086+58E003EE           LD      R14,R0,@GETIP   ;RESTORE RET
008A+04E0               BR      R14             ;
                        ;
008C+50E003F0   SCANER: ST      R14,R0,@SCANER  ;SAVE RET
0090+4140040E           LI      R4,BUFFER       ;A(I/P BUFFER)
0094+58540000           LD      R5,R4,0         ;GET COMMAND (FIXED FORMAT)
0098+55500408           CMP     R5,R0,DMPCMD    ; D  FOR DUMP?
009C+475000C2           BC      Z?,R0,DUMP      ;GO IF TRUE
00A0+5550040A           CMP     R5,R0,STRCMD    ; S  FOR STORE?
00A4+475001B0           BC      Z?,R0,STORE     ;GO STORE IF TRUE
00A8+5550040C           CMP     R5,R0,JMPCMD    ; J  FOR JUMP?
00AC+475002F2           BC      Z?,R0,JUMP      ;GO JUMP IF TRUE
00B0+45E00028           BAL     R14,R0,DOCRLF   ;NEW LINE ON CRT
00B4+4120003F           LI      R2,'?'  ; ?
00B8+45E003D8           BAL     R14,R0,CRTOUT   ;UNKNOWN COMMAND
00BC+58E003F0           LD      R14,R0,@SCANER  ;RESTORE RET
                        BR      R14             ;
```



```
00C0+04E0               ;
                        ;
DUMP:   LI      R4,BUFOP1       ;A(ADDRESS PORTION OF BUFFER)
00C2+41400410           BAL     R14,R0,CVADDR   ;ASCII ADDRESS TO BINARY IN R6
00C6+45E00304           NI      R6,0FFF0H       ;BEGIN ON EVEN WORD BOUNDRY
00CA+9460FFF0           LI      R12,16          ;O/P LINE COUNT
00CE+41C00010   DMPLP:  ST      R6,R0,DMPAD     ;SAVE CURRENT O/P ADDRESS
00D2+50600404           BAL     R14,R0,TYPAD    ;TYPE CURRENT CONTENTS OF R6
00D6+45E00110           LI      R2,'  '         ;SPACE
00DA+41200020           BAL     R14,R0,CRTOUT   ;TO CRT
00DE+45E003D8           BAL     R14,R0,CRTOUT   ;2 SPACES
00E2+45E003D8           BAL     R14,R0,DMPOUT   ;PUT OUT ONE LINE OF DUMP DATA
00E6+45E00126           LI      R2,'  '         ;SPACE
00EA+41200020           BAL     R14,R0,CRTOUT   ;TO CRT
00EE+45E003D8           BAL     R14,R0,TYPLIT   ;O/P LITERAL DATA
00F2+45E0015C           BAL     R14,R0,DOCRLF   ;NEW LINE ON CRT
00F6+45E00028           LD      R6,R0,DMPAD     ;CURRENT DUMPOUT ADDRESS
00FA+58600404           AI      R6,16           ;ADDRESS NEXT LINE
00FE+9A600010           SI      R12,1           ;LINE COUNT -1
0102+98C00001           BC      NZ?,R0,DMPLP    ;LOOP THRU O/P DATA
0106+474000D2           LD      R14,R0,@SCANER  ;RESTORE RET
010A+58E003F0           BR      R14             ;
010E+04E0               ;
                        ;
0110+50E003F2   TYPAD:  ST      R14,R0,@TYPAD   ;SAVE RET
0114+A867               RRL     R6,8            ;HI ADDRESS BYTE
0116+45E0035E           BAL     R14,R0,BINOUT   ;O/P
011A+8867               SRL     R6,8            ;LO ADDRESS BYTE
011C+45E0035E           BAL     R14,R0,BINOUT   ;O/P
0120+58E003F2           LD      R14,R0,@TYPAD   ;RSTORE RET
0124+04E0               BR      R14             ;
                        ;
0126+50E003F4   DMPOUT: ST      R14,R0,@DMPOUT  ;SAVE RET
012A+58700424           LD      R7,R0,DMPAD     ;GET O/P DATA ADDRESS
012E+41D00008           LI      R13,8           ;O/P WORD COUNT
0132+58670000   DMPLPP: LD      R6,R7,0         ;GET NEXT WORD
0136+A867               RRL     R6,8            ;HI BYTE FIRST
0138+45E0035E           BAL     R14,R0,BINOUT   ;O/P
013C+8867               SRL     R6,8            ;LO BYTE
013E+45E0035E           BAL     R14,R0,BINOUT   ;O/P
0142+41200020           LI      R2,'  '         ;SPACE
0146+45E003D8           BAL     R14,R0,CRTOUT   ;TO CRT
014A+9A700002           AI      R7,0002         ;BUMP I/P DATA ADDRESS
014E+9BD00001           SI      R13,0001        ;WORD COUNT -1
0152+47400132           BC      NZ?,R0,DMPLPP   ;LOOP THRU LINE
0156+58E003F4           LD      R14,R0,@DMPOUT  ;RESTORE RET
015A+04E0               BR      R14             ;
                        ;
015C+50E003F6   TYPLIT: ST      R14,R0,@TYPLIT  ;SAVE RET
0160+58700404           LD      R7,R0,DMPAD     ;GET O/P DATA ADDRESS
0164+41D00008           LI      R13,8           ;WORD COUNT
0168+58670000   TYPLLP: LD      R6,R7,0         ;NEXT O/P WORD
016C+A867               RRL     R6,8            ;HI BYTE FIRST
016E+1826               LR      R2,R6           ;TO O/P REG
0170+45E00196           BAL     R14,R0,DOCIT    ;CHECK FOR PRINTABLE CHARACTER
0174+45E003D8           BAL     R14,R0,CRTOUT   ;TO CRT
0176+8867               SRL     R6,8            ;GET LO BYTE
017A+1826               LR      R2,R6           ;TO O/P REG
017C+45E00196           BAL     R14,R0,DOCIT    ;CHECK FOR PRINTABLE CHARACTER
0180+45E003D8           BAL     R14,R0,CRTOUT   ;TO CRT
0184+9A700002           AI      R7,0002         ;TO NEXT WORD
0188+9BD00001           SI      R13,1           ;WORD COUNT -1
018C+47400166           BC      NZ?,R0,TYPLLP   ;LOOP THRU O/P LINE
0190+58E003F6           LD      R14,R0,@TYPLIT  ;RESTORE RET
0194+04E0               BR      R14             ;
                        ;
0196+942000FF   DOCIT:  NI      R2,00FFH        ;GET LOW BYTE
                        CI      R2,'  '         ;BELOW BLANK?
```



```
019A+95200020           BC      LT?,R0,SETPER   ;SET PERIOD IF TRUE
019E+473001AA           CI      R2,007FH        ;BELOW DEL?
01A2+9520007F           BC      LT?,R14,0       ;RET IF TRUE (CHAR PRINTABLE)
01A6+473E0000   SETPER: LI      R2,'.'          ;SET PERIOD AS CHARACTER TO PRI
01AA+4120002E           BR      R14             ;
01AE+04E0               ;
01B0+41400410   STORE:  LI      R4,BUFOP1       ;A(ADDRESS FIELD)
01B4+45E00304           BAL     R14,R0,CVADDR   ;ASCII ADDRESS TO BINARY (IN R6
01B8+58400406           LD      R4,R0,DATAD     ;GET CURRENT I/P DATA ADDRESS
01BC+17DD               XR      R13,R13         ;CLEAR NIBBLE COUNT REG
01BE+45E001E6   STLP:   BAL     R14,R0,UPSTOR   ;UPPER BYTE FIRST
01C2+58E003F0           LD      R14,R0,@SCANER  ;GET RET
01C6+9550000D           CI      R5,000DH        ;END? (CR = END)
01CA+475E0000           BC      Z?,R14,0        ;RET IF TRUE
01CE+45E001FA           BAL     R14,R0,LOSTOR   ;LOWER BYTE
01D2+58E003F0           LD      R14,R0,@SCANER  ;GET RET
01D6+9550000D           CI      R5,000DH        ;END?
01DA+475E0000           BC      Z?,R14,0        ;RET IF TRUE
01DE+94400002           AI      R4,0002         ;TO NEXT WORD
01E2+740001BE           BX      R0,STLP         ;CONTINUE STORING DATA
                        ;
01E6+50E003FA   UPSTOR: ST      R14,R0,@UPSTOR  ;SAVE RET
01EA+58540000           LD      R5,R4,0         ;GET NEXT DATA
01EE+8857               SRL     R5,8            ;GET HI BYTE
01F0+45E00210           BAL     R14,R0,STDATA   ;GO STORE BYTE
01F4+58E003FA           LD      R14,R0,@UPSTOR  ;RESTORE RET
01F8+04E0               BR      R14             ;
                        ;
01FA+50E003FA   LOSTOR: ST      R14,R0,@UPSTOR  ;SAVE RET
01FE+58540000           LD      R5,R4,0         ;GET DATA
0202+945200FF           NI      R5,00FFH        ;KEEP LOW BYTE
0206+45E00210           BAL     R14,R0,STDATA   ;GO STORE BYTE
020A+58E003FA           LD      R14,R0,@UPSTOR  ;RESTORE RET
020E+04E0               BR      R14             ;
                        ;
0210+50E003FC   STDATA: ST      R14,R0,@STDATA  ;SAVE RET
0214+45E0023C           BAL     R14,R0,CKDEL    ;CHECK FOR DELIMITER
0218+58E003FC           LD      R14,R0,@STDATA  ;GET RET
021C+475E0000           BC      Z?,R14,0        ;RET IF RC = 0
0220+45E0025A           BAL     R14,R0,ASCHEX   ;ASCII BYTE TO HEX NIBBLE
0224+47300232           BC      LT?,R0,SETND    ;.NZ. RC = END
0228+45E00290           BAL     R14,R0,NIBBLE   ;STORE THIS NIBBLE
022C+58E003FC           LD      R14,R0,@STDATA  ;RESTORE RET
0230+04E0               BR      R14             ;
0232+4150000D   SETND:  LI      R5,000DH        ;FAKE EOF
0236+58E003FC           LD      R14,R0,@STDATA  ;RESTORE RET
023A+04E0               BR      R14             ;
023C+95500020   CKDEL:  CI      R5,'  '         ;SPACE?
0240+475E0000           BC      Z?,R14,0        ;RET IF TRUE
0244+9550002E           CI      R5,'.'          ;PERIOD?
0248+475E0000           BC      Z?,R14,0        ;RET IF TRUE
024C+9550002C           CI      R5,','          ;COMMA?
0250+475E0000           BC      Z?,R14,0        ;RET IF TRUE
0254+9550000D           CI      R5,000DH        ;CARRIAGE RET?
0258+04E0               BR      R14             ;LET CALLER DECIDE
                        ;
025A+945000FF   ASCHEX: NI      R5,00FFH        ;LOW BYTE ONLY
025E+95500030           CI      R5,'0'          ;LOWER THAN '0' ?
0262+473E0000           BC      LT?,R14,0       ;RET IF TRUE
0266+9550003A           CI      R5,':'          ;0-9 ?
026A+47300268           BC      LT?,R0,VNUM     ;NUMERICAL IF TRUE
026E+95500041           CI      R5,'A'          ;LOWER THAN 'A' ?
0272+473E0000           BC      LT?,R14,0       ;RET IF TRUE
0276+95500047           CI      R5,0047H        ;HEX ALPHA?
027A+47300284           BC      LT?,R0,VALPH    ;HEX ALPHA IF TRUE
027E+9550FFFF           CI      R5,0FFFFH       ;SET .LT. CC
0282+04E0               BR      R14             ;
0284+95500007   VALPH:   SI      R5,0007H        ;ASCII ADJUST
0288+9450000F           NI      R5,000FH        ;LOW NIBBLE ONLY
028C+1555               CLR     R5,R5           ;RC = ZERO
028E+04E0               BR      R14             ;
                        ;
0290+58760000   NIBBLE: LD      R7,R6,0         ;GET OLD DATA
0294+16DD               ORR     R13,R13         ;R13 = ZERO ?
0296+474002AC           BC      NZ?,R0,NXNIB1   ;TEST FOR ONE IF NOT TRUE
029A+9AD00001           AI      R13,0001        ;BUMP NIBBLE COUNTER
029E+895B               SLL     R5,12           ;POSITION THIS NIBBLE
02A0+94700FFF           NI      R7,0FFFH        ;PREPARE OLD DATA FOR NEW NIBEL
02A4+1675               ORR     R7,R5           ;INSERT NEW NIBBLE
02A6+50760000           ST      R7,R6,0         ;DATA BACK TO MEMORY
02AA+04E0               BR      R14             ;
02AC+95D00001   NXNIB1: CI      R13,0001        ;NEXT NIBBLE?
02B0+474002C6           BC      NZ?,R0,NXNIB2   ;TO NEXT IF NOT THIS
02B4+9AD00001           AI      R13,0001        ;BUMP NIBBLE COUNTER
02B8+8957               SLL     R5,8            ;POSITION THIS NIBBLE
02BA+9470F0FF           NI      R7,0F0FFH       ;PREPARE OLD DATA FOR NEW NIBBL
02BE+1675               ORR     R7,R5           ;INSERT NEW NIBBLE
02C0+50760000           ST      R7,R6,0         ;DATA BACK TO MEMORY
02C4+04E0               BR      14              ;
02C6+95D00002   NXNIB2: CI      R13,0002        ;NEXT NIBBLE?
02CA+474002E0           BC      NZ?,R0,NXNIB3   ;TO NEXT IF NOT THIS
02CE+9AD00001           AI      R13,0001        ;BUMP NIBBLE COUNT
02D2+8953               SLL     R5,4            ;POSITION THIS NIBBLE
02D4+9470FF0F           NI      R7,0FF0FH       ;PREPARE OLD DATA FOR NEW NIBBL
02D8+1675               ORR     R7,R5           ;INSERT NEW NIBBLE
02DA+50760000           ST      R7,R6,0         ;DATA BACK TO MEMORY
02DE+04E0               BR      R14             ;
02E0+17DD       NXNIB3: XR      R13,R13         ;LAST NIBBLE (LSN)
02E2+9470FFF0           NI      R7,0FFF0H       ;PREPARE OLD DATA FOR NEW NIBBL
02E6+1675               ORR     R7,R5           ;INSERT NEW NIBBLE
02E8+50760000           ST      R7,R6,0         ;DATA BACK TO MEMORY
02EC+9A600002           AI      R6,0002         ;BUMP MEM POINTER
02F0+04E0               BR      R14             ;
                        ;
02F2+41400410   JUMP:   LI      R4,BUFOP1       ;A(ADDRESS)
02F6+45E00304           BAL     R14,R0,CVADDR   ;ASCII ADDRESS TO BINARY ADDRES
02FA+18F6               LR      R15,R6          ;ADDRESS TO R15
02FC+45EF0000           BAL     R14,R15,0       ;JUMP...
0300+74000000           BX      R0,BEGIN        ;BACK TO MONITOR IF CALLEE RETU
                        ;
0304+50E00402   CVADDR: ST      R14,R0,@CVADDR  ;SAVE RET
0308+1766               XR      R6,R6           ;CLEAR R6
030A+58540000           LD      R5,R4,0         ;GET TWO ADDRESS BYTES
030E+8857               SRL     R5,8            ;UPPER BYTE FIRST
0310+45E0025A           BAL     R14,R0,ASCHEX   ;ASCII BYTE TO HEX NIBBLE
0314+47400350           BC      NZ?,R0,CVHOUT   ;STOP IF NOT HEX DATA
0318+1665               ORR     R6,R5           ;FIRST ADDRESS NIBBLE TO R6
031A+58540000           LD      R5,R4,0         ;GET ADDRESS BYTES AGAIN
031E+45E0025A           BAL     R14,R0,ASCHEX   ;ASCII BYTE TO HEX NIBBLE
0322+47400350           BC      NZ?,R0,CVHOUT   ;STOP IF NOT HEX DATA
0326+8963               SLL     R6,4            ;POSITION ADDRESS FOR NEXT NIBB
0328+1665               ORR     R6,R5           ;INSERT NEXT ADDRESS NIBBLE
032A+9A400002           AI      R4,0002         ;BUMP MEMORY PTR TO NEXT WORD
032E+58540000           LD      R5,R4,0         ;NEXT ASCII ADDRESS DATA
0332+8857               SRL     R5,8            ;HIGH BYTE FIRST
0334+45E0025A           BAL     R14,R0,ASCHEX   ;ASCII BYTE TO HEX NIBBLE
0338+47400354           BC      NZ?,R0,CVHOT1   ;STOP IF NOT HEX DATA
033C+8963               SLL     R6,4            ;POSITION ADDRESS FOR NEXT NIBB
033E+1665               ORR     R6,R5           ;INSERT NEXT NIBBLE
0340+58540000           LD      R5,R4,0         ;GET ADDRESS DATA AGAIN
0344+45E0025A           BAL     R14,R0,ASCHEX   ;ASCII BYTE TO HEX NIBBLE
0348+47400350           BC      NZ?,R0,CVHOUT   ;STOP IF NOT HEX DATA
034C+8963               SLL     R6,4            ;POSITION ADDRESS FOR NEXT NIBB
                        ORR     R6,R5           ;INSERT NEXT NIBBLE
```

```
034E+1665
0350+9A400002   CVHOUT: AI      R4,0002         ;TO NEXT MEMORY WORD
0354+50400406   CVHOT1: ST      R4,R0,DATAD     ;SAVE AS DATA ADDRESS
0358+58E00402           LD      R14,R0,@CVADDR  ;RESTORE RET
035C+04E0               BR      R14             ;
                ;
035E+50E003FE   BINOUT: ST      R14,R2,@BINOUT  ;SAVE RET
0362+1826               LR      R2,R6           ;O/P BYTE TO R2
0364+8823               SRL     R2,4            ;UPPER NIBBLE FIRST
0366+9420000F           NI      R2,000FH        ;KEEP ONLY GOOD DATA
036A+45E00386           BAL     R14,R0,HEXEX    ;BINARY NIBBLE TO ASCII BYTE
036E+45E003D8           BAL     R14,R0,CRTOUT   ;NIBBLE (BYTE) OUT TO CRT
0372+1826               LR      R2,R6           ;O/P DATA TO R2
0374+9420000F           NI      R2,000FH        ;KEEP ONLY LOW NIBBLE
0378+45E0038E           BAL     R14,R0,HEXEX    ;BINARY NIBBLE TO ASCII BYTE
037C+45E003D8           BAL     R14,R0,CRTOUT   ;AND OUT TO CRT
0380+58E003FE           LD      R14,R0,@BINOUT  ;RESTORE RET
0384+04E0               BR      R14             ;
                ;
0386+9520000A   HEXEX:  CI      R2,000AH        ;A-F ?
038A+47F00392           BC      MI?,R0,CON      ;BR IF NOT TRUE
038E+9A200007           AI      R2,0007H        ;ADJUST FOR A-F
0392+9A200030   CON:    AI      R2,0030H        ;MAKE ASCII
0396+04E0               BR      R14             ;
                ;
0398+50E00400   GETCHR: ST      R14,R0,@GETCHR  ;SAVE RET
039C+A010FFFB   RDCHR:  IN      R1,R0,STATUS    ;STRIP PARITY
03A0+94100002           NI      R1,0002         ;I/P READY?
03A4+4750039C           BC      Z?,R0,RDCHR     ;LOOP UNTIL CHARACTER READY
03A8+A010FFFA           IN      R1,R0,DATA      ;READ DATA
03AC+9410007F           NI      R1,007FH        ;KEEP ONLY DATA BYTE
03B0+1821               LR      R2,R1           ;DATA TO R2
03B2+45E003D8           BAL     R14,R0,CRTOUT   ;ECHO I/P
03B6+1812               LR      R1,R2           ;DATA BACK TO R1
03B8+58E00400           LD      R14,R0,@GETCHR  ;GET RET
03BC+41F0FFFF           LI      R15,-1          ;SET R15 .NZ.
                        LI      R2,000AH        ;LF CODE IN CASE OF CR
03C0+4120000A           CI      R1,000DH        ;DATA = CR ?
03C4+9510000D           BC      NZ?,R14,0       ;RET IF NO
03C8+474E0000           BAL     R14,R0,CRTOUT   ;DO LF IF PREVOIUS WAS CR
03CC+45E003D8           XR      R15,R15         ;SET RC = ZERO FOR CR
03D0+17FF               LD      R14,R0,@GETCER  ;RESTORE RETURN
03D2+58E00400           BR      R14             ;
03D6+04E0       ;
                CRTOUT: IN      R1,R0,STATUS    ;GET STATUS BYTE
03D8+A010FFFB           NI      R1,0001         ;XMITTER EMPTY?
03DC+94100001           BC      Z?,R0,CRTOUT    ;WAIT FOR XMITTER TO EMPTY
03E0+475003D8           OUT     R2,R0,DATA      ;O/P DATA TO CRT
03E4+A220FFFA           BR      R14             ;
03E8+04E0       ;
                ;
                ;
03EA            @PROMPT DS      2
03EC            @DOCRLF DS      2
03EE            @GETIP  DS      2
03F0            @SCANER DS      2
03F2            @TYPAD  DS      2
03F4            @DMPOUT DS      2
03F6            @TYPLIT DS      2
03F8            @STORE  DS      2
03FA            @UPSTOR DS      2
03FC            @STDATA DS      2
03FE            @BINOUT DS      2
0400            @GETCHR DS      2
0402            @CVADDR DS      2
                ;
0404            DMPAD:  DS      2
0406            DATAD:  DS      2
                ;
0408 4420       DMPCMD: DB      'D '
040A 5320       STRCMD: DB      'S '
040C 4A20       JMPCMD: DB      'J '
                ;
                ;
040E            BUFFER: DS      2
0410            BUFOP1: DS      128
                ;
                ;
0490                    END

025A ASCHEX     0000 BEGIN      035E BINOUT     040E BUFFER     0410 BUFOP1
023C CKDEL      0392 CON        0A3D CRLF       03D8 CRTOUT     0404 CVADDR
0354 CVHOT1     0350 CVHOUT     FFFA DATA       0406 DATAD      0404 DMPAD
0408 DMPCMD     00D2 DMPLP      0132 DMPLPP     0126 DMPOUT     0196 DOCIP
0028 DOCRLF     0082 DOEOF      007E DOEOF2     00C2 DUMP       0398 GETCHR
0040 GETIP      0386 HEXEX      004C IPLP       040C JMPCMD     02F2 JUMP
01FA LOSTOR     0006 MONLP      0290 NIBBLE     02AC NXNIB1     02C6 NXNIB2
02E0 NXNIB3     0016 PROMPT     039C RDCHR      00BC SCANER     0232 SETND
01AA SETPER     FFFB STATUS     0210 STDATA     01BE STLP       01B0 STORE
040A STRCMD     0110 TYPAD      015C TYPLIT     0168 TYPLLP     01E6 UPSTOR
0284 VALPH      0288 VNUM       03FE @BINOUT    0402 @CVADDR    03FA @DMPOUT
03EC @DOCRLF    0400 @GETCHR    03EE @GETIP     03EA @PROMPT    03F0 @SCANER
03FC @STDATA    03F8 @STORE     03F2 @TYPAD     03F6 @TYPLIT    03FA @UPSTOR
```


APPENDIX D

The System 29 operating system manages two floppy disk drives, A and B. The system will prompt with a A> or B> depending upon which disk the operator selects as the default. Generally, most system programs (editors, debuggers, compilers, etc.) are on the A disk and most user generated programs (source programs, user libraries, special assemblers, etc.) are on the B disk. In the following session, lower case letters are what the user typed-in, upper case letters are what System 29 responded, and comments (added as a tutorial) are in curly brackets.

```
A>ed b: amd16bit.asm        {call the editor to edit AMD16BIT.ASM from the B disk}
*                           {any program additions, changes, and/or deletions go here}
*e                          {exit the editor and save the new AMD16BIT.ASM on the B disk}
A>b:                        {switch to the B disk as default}
B>mac29 amd16bit $ab hb pb sb   {use the modified macro assembler (MAC29) to assemble AMD16BIT.ASM and put the
                                 HEX, PRINT and SYMBOL files back on to the B disk}

ASM29 VER. 1.0
0490
03BH USE FACTOR
END OF ASSEMBLY
B>a:                        {switch back to the A disk}
A>ddt29 h e                 {run DDT29, Halt the 16-bit computer's clock and Exit DDT29}
A>set pa 3d                 {set the page register bit to enable the 16-bit computer main memory as 9080 upper 32k}
A>ddt                       {load 9080 DDT}
DDT VERS 1.4
#ib:amd16bit.hex            {reference the simple monitor's HEX file on the B disk}
#r8000                      {read AMD16BIT.HEX into 9080 memory beginning at location 8000 HEX (upper 32k)}
NEXT PC END
840E 0100 577F
#↑C                         {exit DDT via control-C}
A>lbpm m29 wcs cl ul dc 1   {load the 16-bit computer's microcode (phase-1 instruction set)}
LOADING: M29.OBJ
TITLE: MICROPROGRAM FOR 16-BIT COMPUTER
VERIFYING: M29.OBJ
TITLE: MICROPROGRAM FOR 16-BIT COMPUTER
VERIFY COMPLETE
A>ddt29 ir 0 j r             {run DDT29, set the instruction address register to zero (IR 0), jam the address on to the
                              microprogram address bus (J), and run the 16-bit computer's clock (R)}
```

At this point, the AMD 16-bit high speed computer is running phase 1 instruction set in microcode and the simple monitor in target machine language in 16-bit main memory. A CRT terminal set to 9600 baud and connected to console USART can now exercise the simple monitor.

APPENDIX E

Memory Board

The 16-Bit Computer Main Memory board was organized with 8k by 16-bit RAM section and a 2k by 16-bit ROM section. The RAM section occupies address 0-8k while the ROMs are assigned addresses 8k through 10k. The memory word consists of two bytes. The least significant address line specified whether high or low byte but is not used in the word mode. The address value from the computer is captured in a register at the beginning of the cycle; however, the most significant address lines are routed straight from the bus to the clock decode logic to make an early decision as to whether the memory board has been selected.

In the word mode, the read and write transfers are straight forward. For the byte read mode, data is output on bus bits BD_{0-7} while BD_{8-15} are forced to zero. During byte write mode bus bits BD_{0-7} are duplicated internally on lines D_{0-7} and lines D_{8-15}. The signals WRHIGH or WRLOW select which byte in the RAM memory is effected.

The control logic generates the bus control line sequencing required by the 16-Bit Computer. The memory read and write timing is shown in Figures E1 and E2. The bus controller function is simulated for the purposes of the prototype. Bus Request is clocked into a flip-flop and Bus Acknowledge is returned to the computer. The Memory Request signal from the computer initiates a memory cycle. Fifty nanoseconds later the memory board responds with Address Accept. The computer then follows this with Data Request. The memory board responds with Data Sync and 50 nanoseconds later the data read out of the memory is clocked into the output registers and output on the data bus. Looking at the memory read timing diagram, it is seen that a read cycle is initiated with Memory Request but the data is not sent back to the computer until the beginning of the next microcycle.

The write cycle is extended one oscillator cycle. This is necessary with the Am9124 RAMs because the data are not sent to the memory board until Data Request goes active (see Figure E2), which is 100 nanoseconds into the write cycle. With the clocked handshaked memory protocol of the 16-Bit Computer, this is easily done by delaying Data Sync one oscillator cycle. Since normally a computer performs many more read than writes, this impacts throughput only slightly.

Additional logic was appended to allow the memory to be accessed by the System 29 microprogramming development system. The Map Page (MAPP) of System 29 was used to specify the memory. The logic interfaces the control signals required by System 29 and the 16-Bit Computer Memory board. With this logic, the System 29 user can readily read or write into the memory.

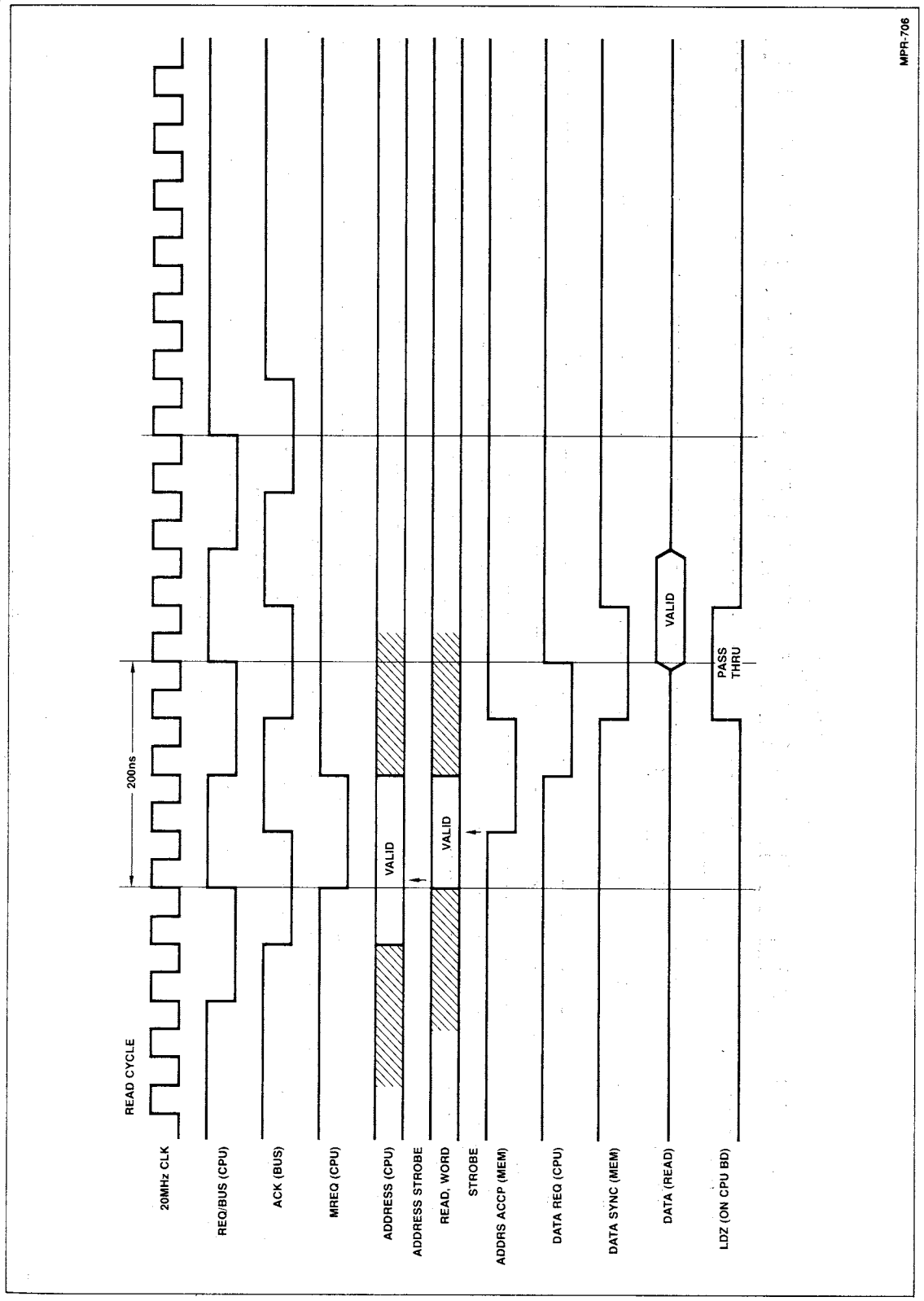


Figure E1. Memory Read Timing.

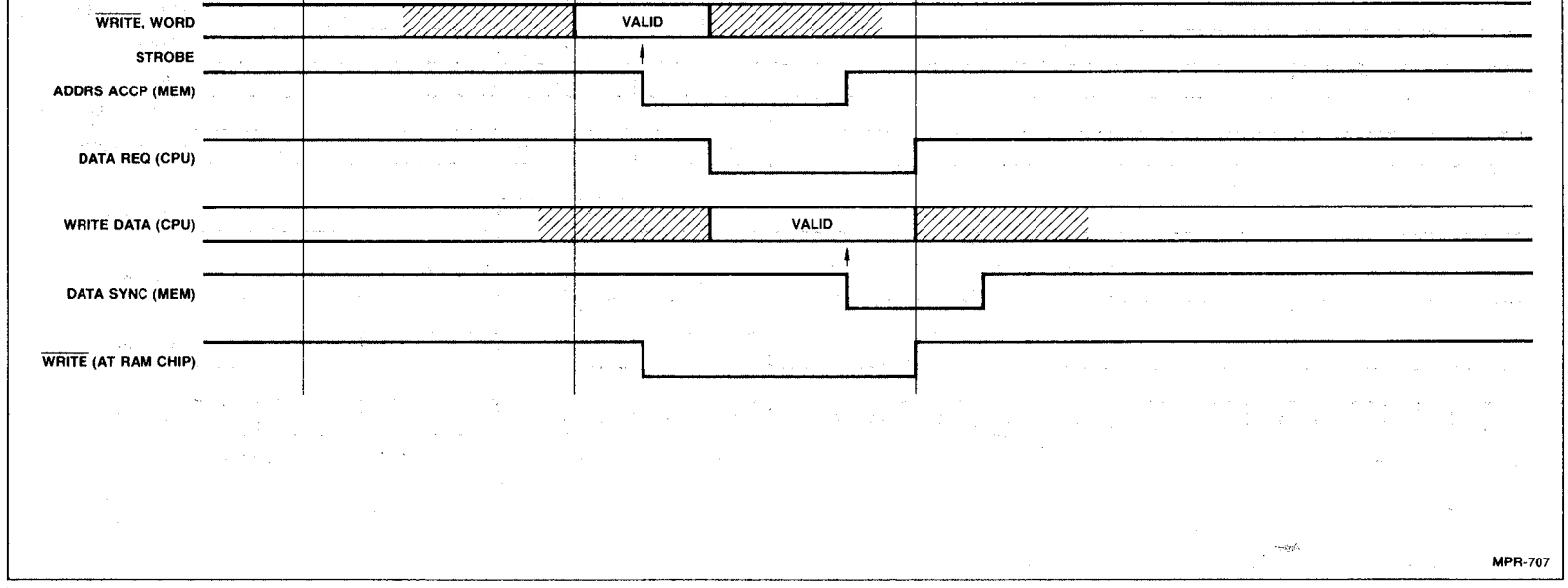
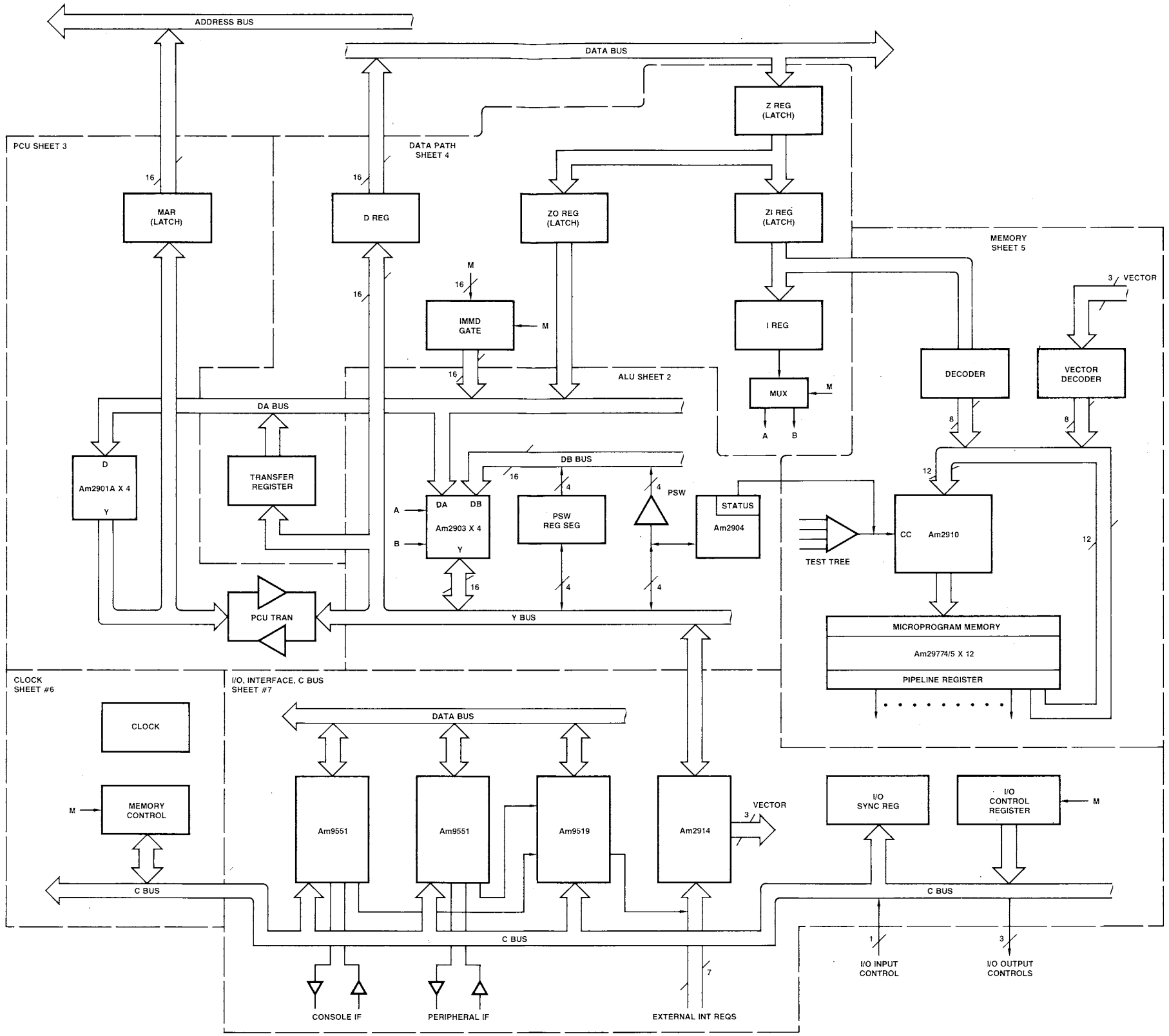


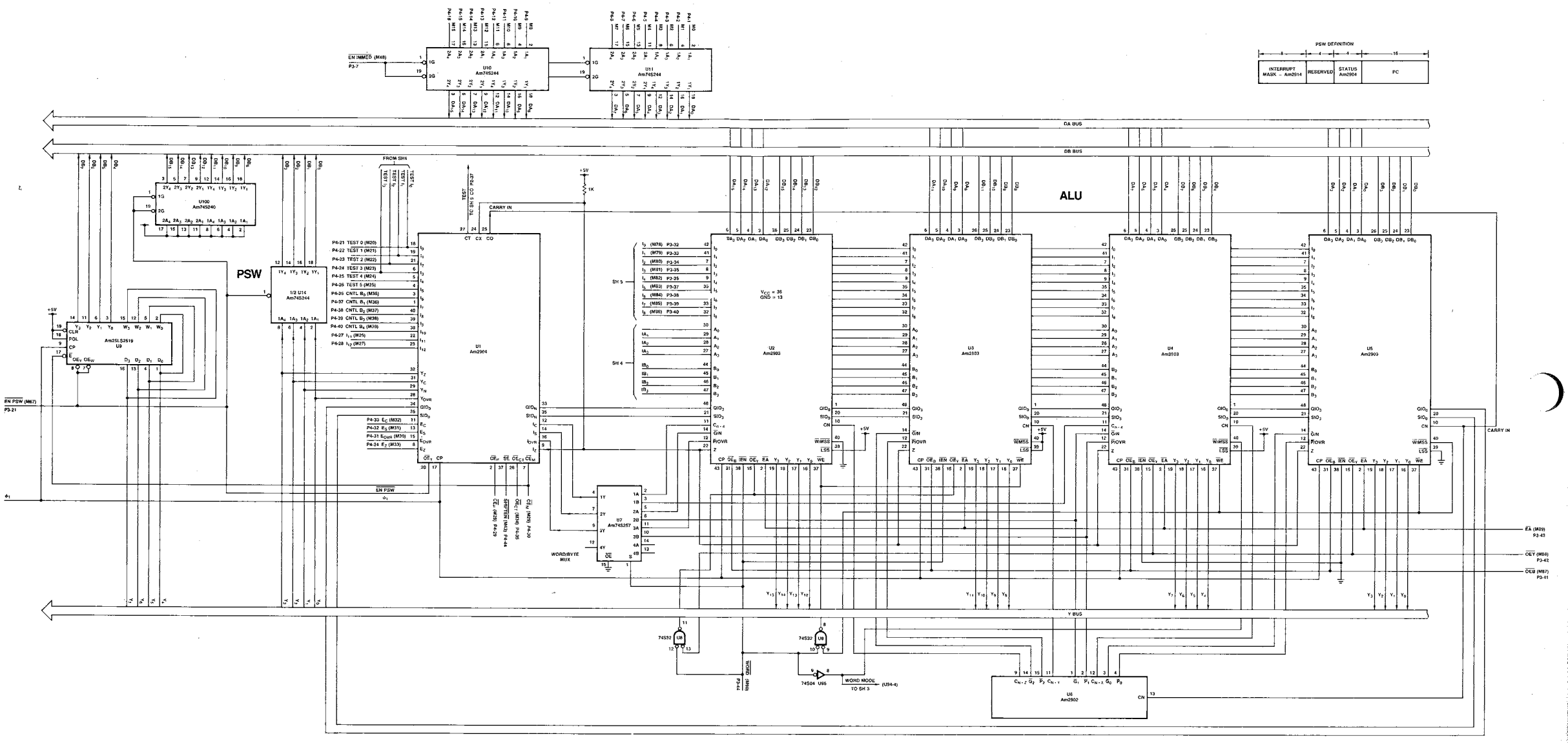
Figure E2. Memory Write Timing.



Block Diagram 16-Bit Computer.

CPU
Sheet 1

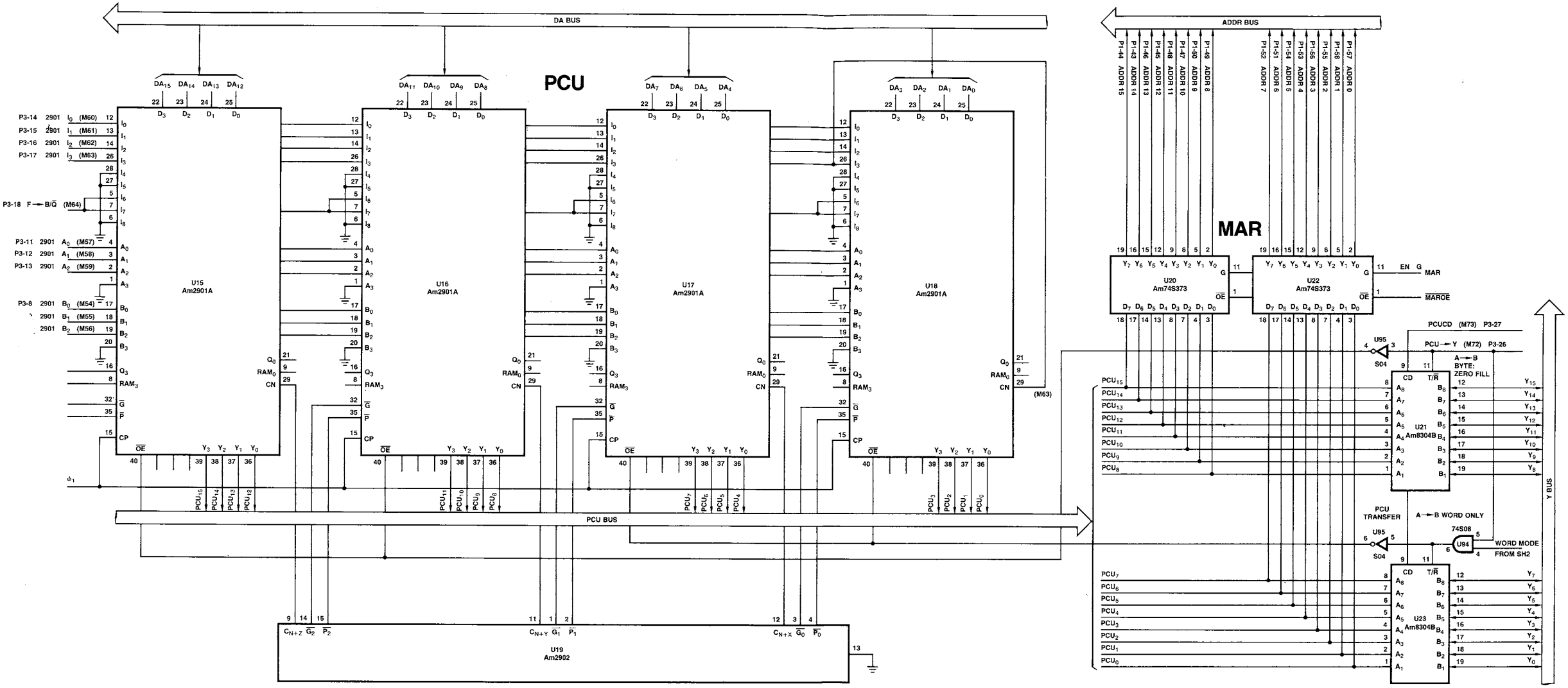
MPR-708



CPU
Sheet 2

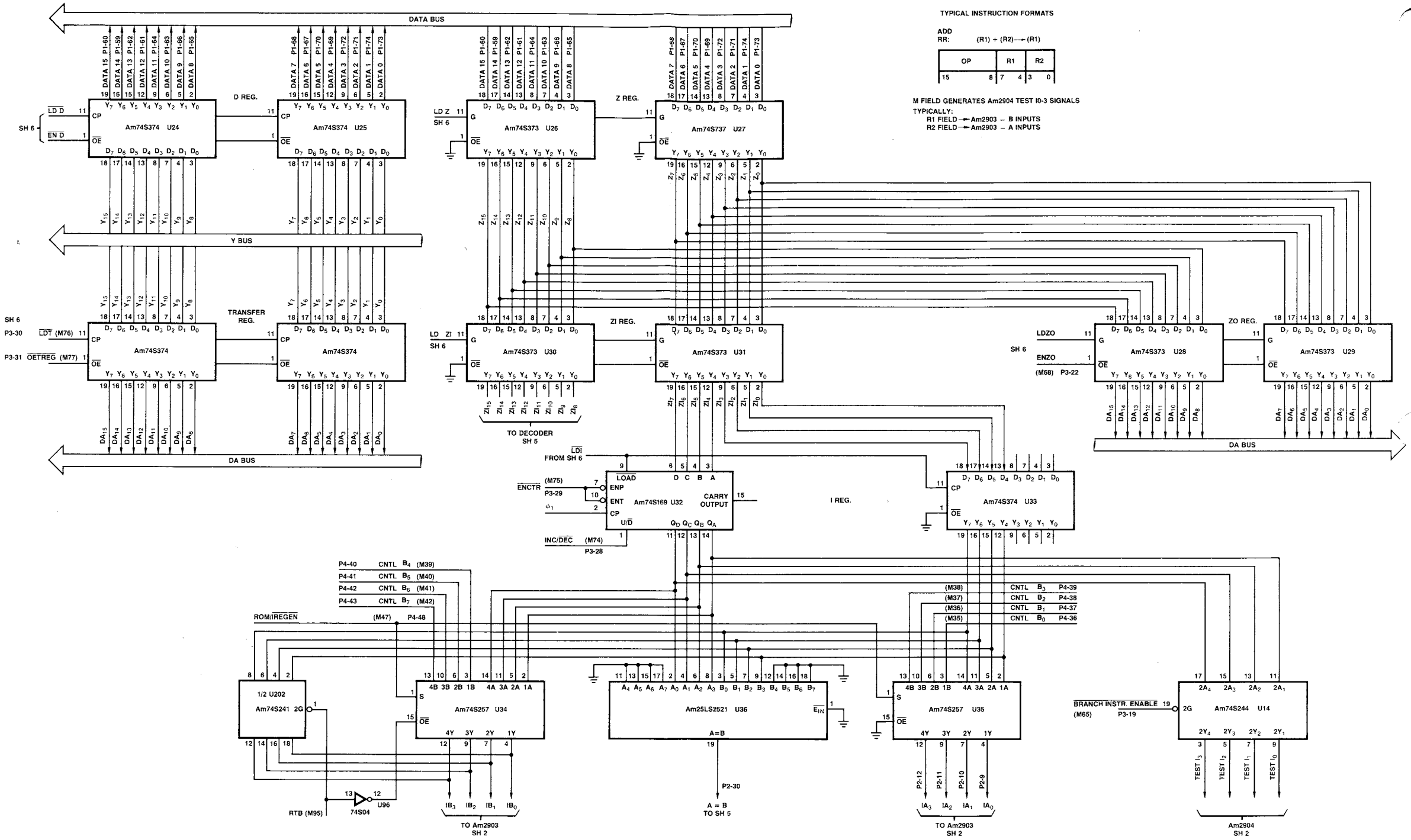
MPR-709

16-Bit Computer ALU.



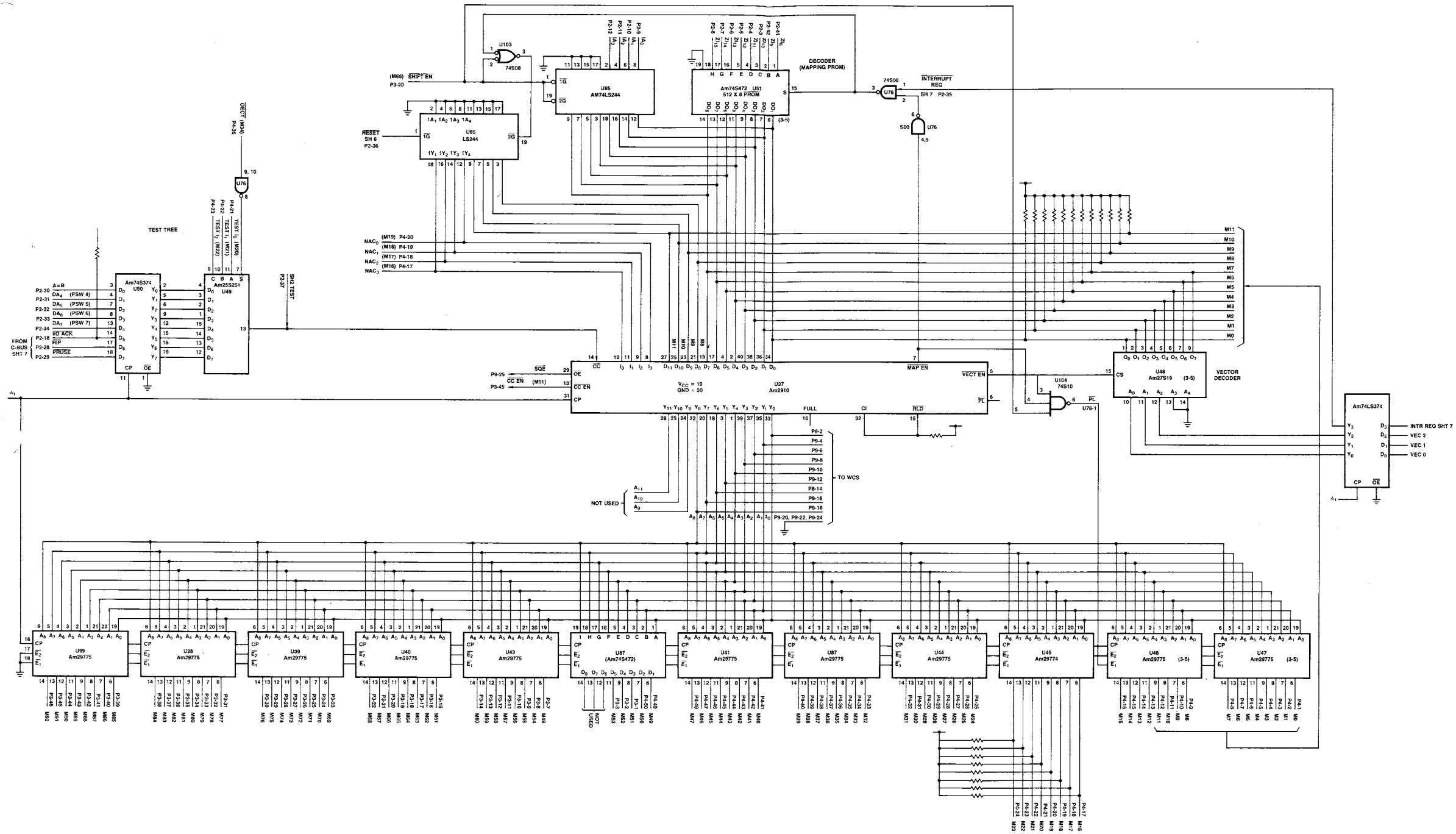
16-Bit Computer PCU Memory Address Register.

CPU
Sheet 3
MPR-710



CPU
Sheet 4
MPR-711

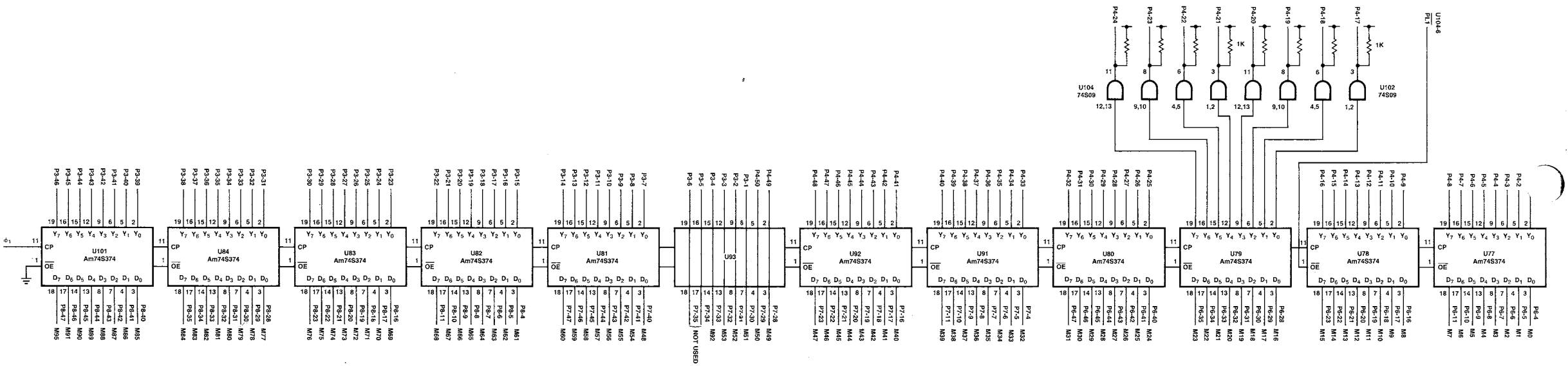
16-Bit Computer Data Path.



16-Bit Computer Microprogram Memory.

CPU
Sheet 5

MPR-712

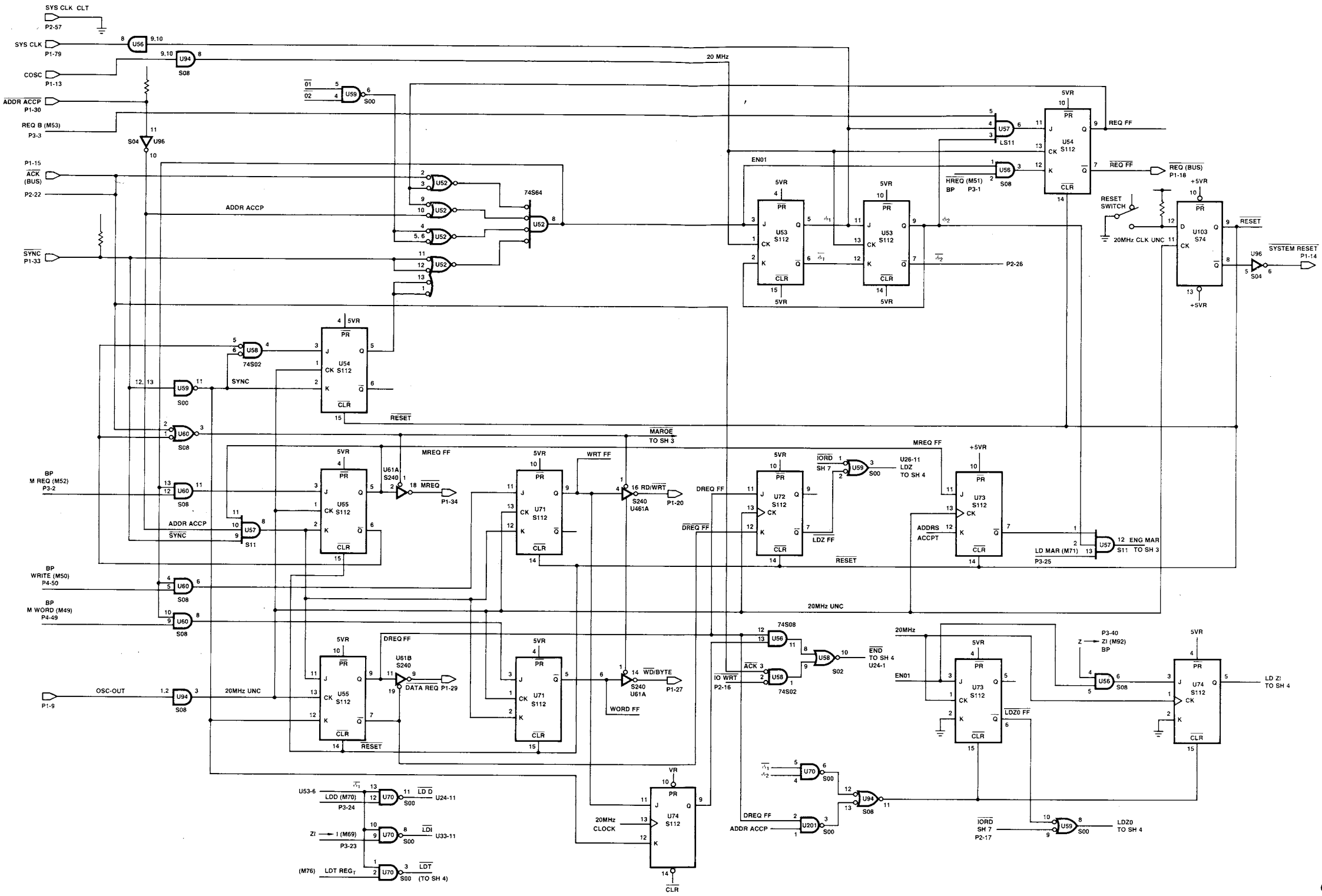


THE COMPONENTS ON THIS PAGE ARE USED TO INTERFACE TO THE WRITEABLE CONTROL STORE OF THE PROTOTYPING SYSTEM (S/29) AND ARE NOT PART OF THE FINAL COMPUTER DESIGN. DELETE THESE COMPONENTS FROM THE COMPONENT COUNT.

CPU
Sheet 5A

MPR-717

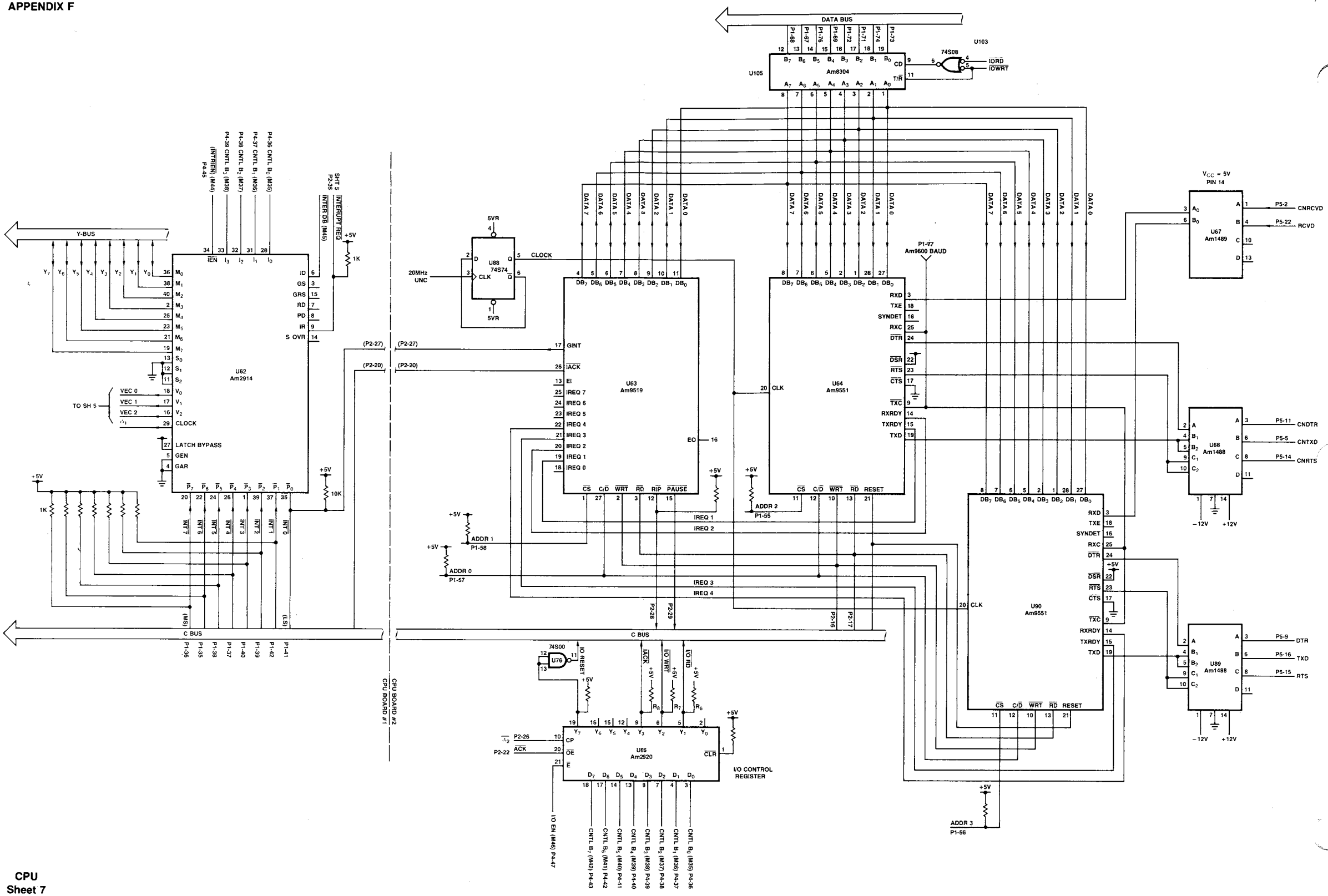
16-Bit Computer S/29 WCS Interface.



16-Bit Computer Memory and Clock Control.

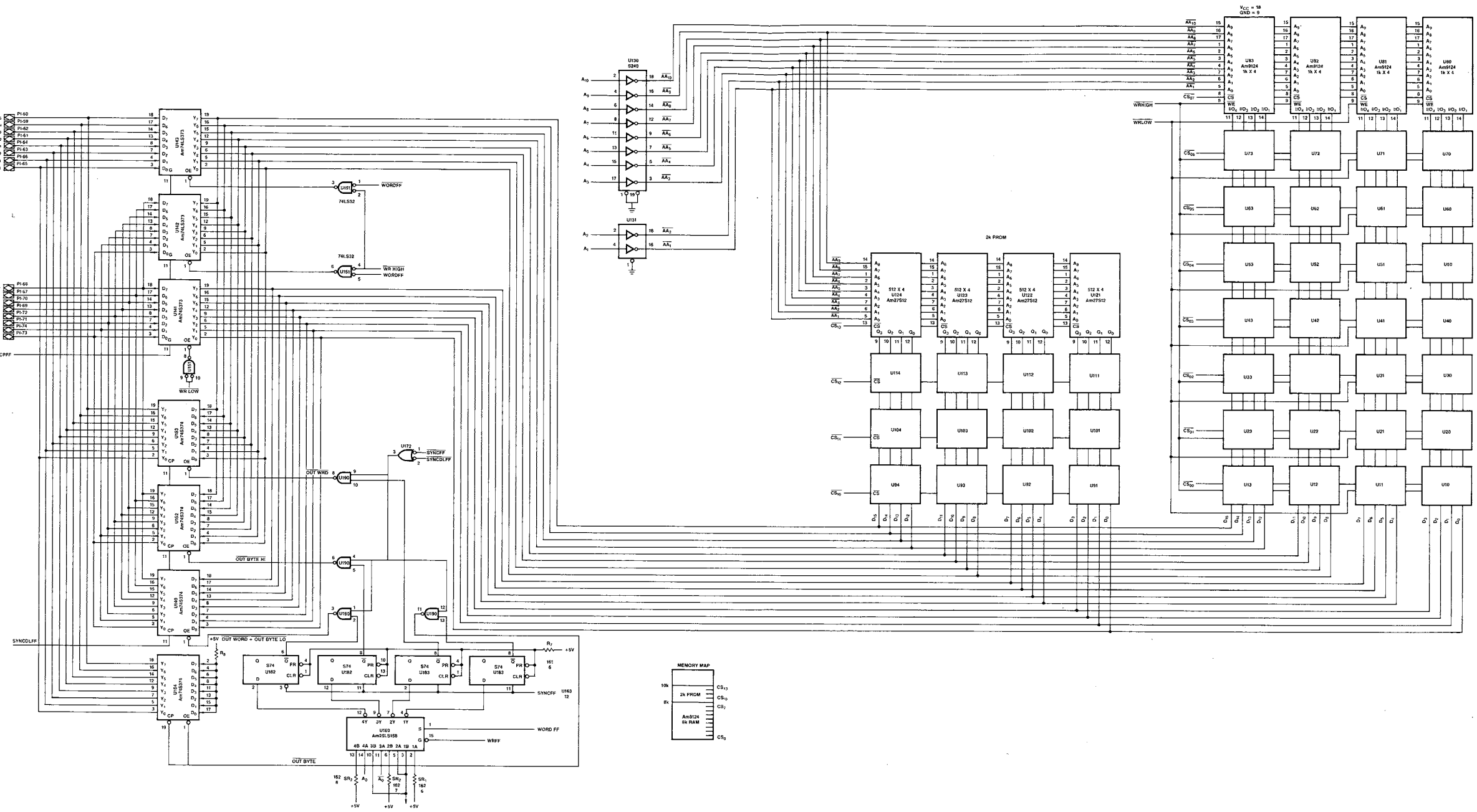
CPU Sheet 6

MPR-713



CPU
Sheet 7
MPR-714

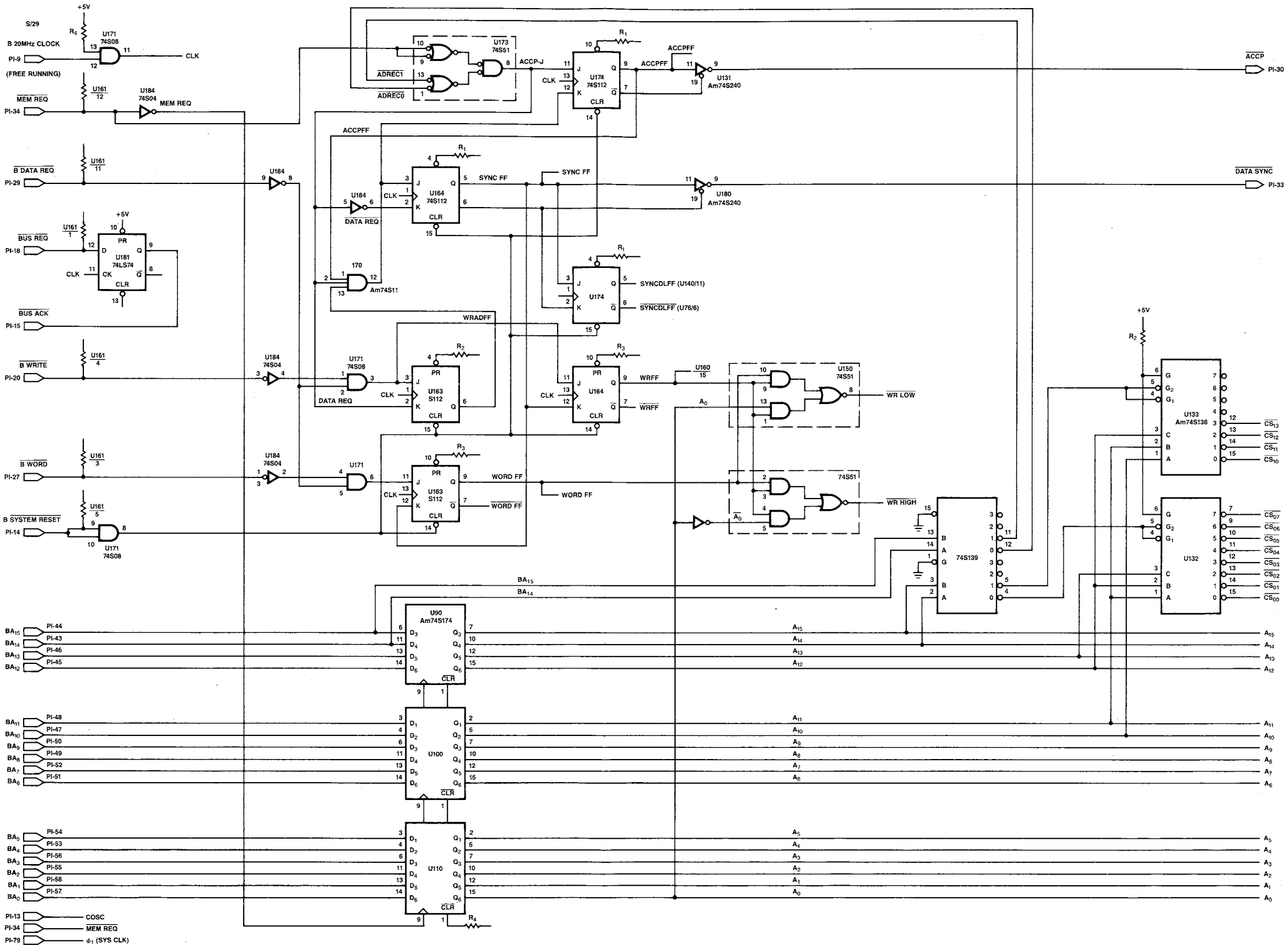
16-Bit Computer I/O, Bus Interface, Interrupt.



16-Bit Computer Memory Board.

Memory Sheet 1

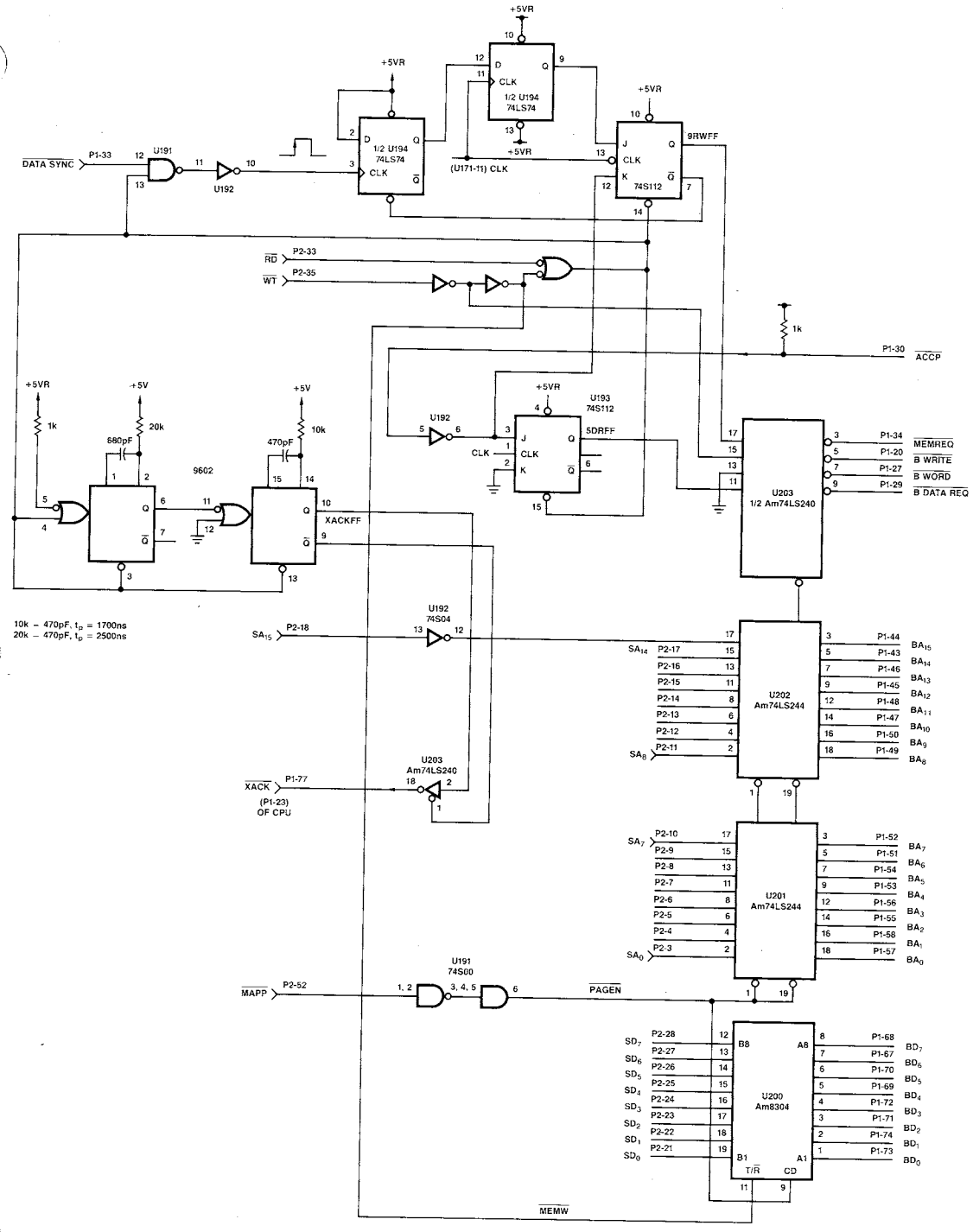
MPR-715



Memory Sheet 2

MPR-716

16-Bit Machine Memory Board.



Memory Sheet 2A

16-Bit Computer Memory Board (S/29 Interface).

MPR-718

AM-PUB073-9