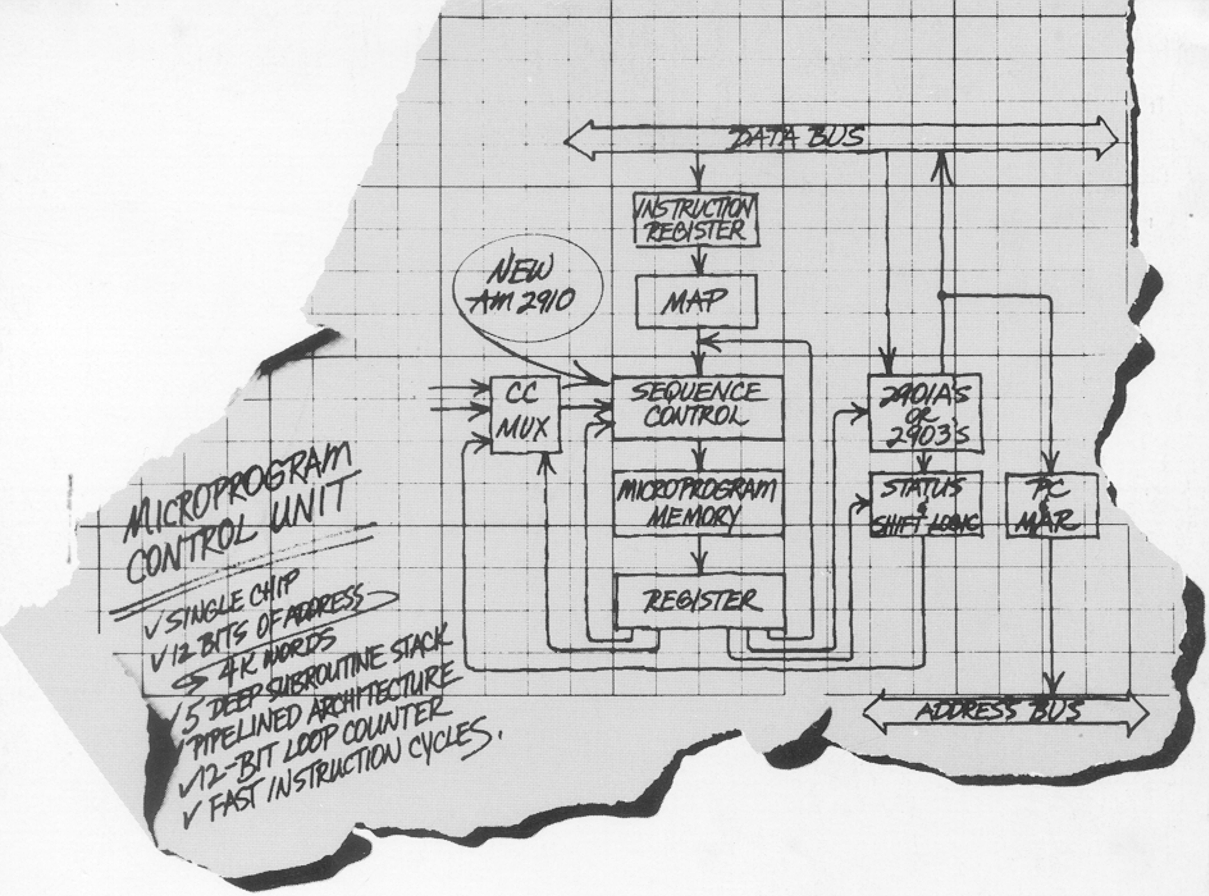


# Build a Microcomputer

## Chapter II
## Microprogrammed Design

Advanced
Micro Devices

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

AM-PUB073-2

# CHAPTER II
# MICROPROGRAMMED DESIGN

## INTRODUCTION

A microprogrammed machine is one in which a coherent sequence of microinstructions is used to execute various commands required by the machine. If the machine is a computer, each sequence of microinstructions can be made to execute a machine instruction. All of the little elemental tasks performed by the machine in executing the machine instruction are called microinstructions. The storage area for these microinstructions is usually called the microprogram memory. This technique was identified by Wilkes in the 1950's as a structured approach to the random control logic in a computer.

A microinstruction usually has two primary parts. These are: (1) the definition and control of all elemental micro-operations to be carried out and (2) the definition and control of the address of the next microinstruction to be executed.

The definition of the various micro-operations to be carried out usually includes such things as ALU source operand selection, ALU function, ALU destination, carry control, shift control, interrupt control, data-in and data-out control and so forth. The definition of the next microinstruction function usually includes identifying the source selection of the next microinstruction address, and in some cases, supplying the actual value of that microinstruction address.

Microprogrammed machines are usually distinguished from non-microprogrammed machines in the following manner. Older, non-microprogrammed machines implemented the control function by using combinations of gates and flip-flops connected in a somewhat random fashion in order to generate the required timing and control signals for the machine. Microprogrammed machines, on the other hand, are normally considered highly ordered and more organized with regard to the control function field. In its simplest definition, a microprogram control unit consists of the microprogram memory and the structure required to determine the address of the next microinstruction.

Microprogramming is normally selected by the design engineer as a control technique for finite state machines because it improves flexibility, performance, and LSI utilization. Several additional key features of microprogrammed designs are listed below:

- More structured organization
- Diagnostics can be implemented easily
- Design changes are simple
- Field updates are easy
- Adaptations are straightforward
- System definition can be expanded to include new features
- Documentation and Service are easier
- Design aids are available
- Cost and design time are reduced

## THE MICROPROGRAM MEMORY

The microprogram memory is simply an N word by M bit memory used to hold the various microinstructions. For an N word memory, the address locations are usually defined as location 0 through N−1. For example, a 256-word microprogram memory will have address locations 0 through 255. Each word of the microprogram memory consists of M bits. These M bits are usually broken into various field definitions and the fields can consist of various numbers of bits. It is the definition of the various fields of a microprogram word that is usually referred to as FORMATTING.

An example of how microinstruction fields are defined in a typical machine microprogram memory word is as follows:

Field 1 – General purpose
Field 2 – Branch address
Field 3 – Next microinstruction address control
Field 4 – Condition code multiplexer control
Field 5 – Interrupt control
Field 6 – Fast clock/slow clock select
Field 7 – Carry control
Field 8 – ALU source operand control
Field 9 – ALU function control
Field 10 – ALU destination control
Field 11 – Shift multiplexer control
Field 12 – etc.

## EXECUTING MICROINSTRUCTONS

Once the microprogram format has been defined, it is necessary to execute sequences of these microinstructions if the machine is to perform any real function. In its simplest form, all that is required to sequence through a series of microinstructions is a microprogram address counter. The microprogram address counter simply increments by one on each clock cycle to select the address of the next microinstruction. For example, if the microprogram address counter contains address 23, the next clock cycle will increment the counter and it will select address 24. The counter will continue to increment on each clock cycle thereby selecting address 25, address 26, address 27, and so forth. If this were the only control available, the machine would not be very flexible and it would be able to execute only a fixed pattern of microinstructions.

The technique of continuing from one microinstruction to the next sequential microinstruction is usually referred to as CONTINUE. Thus, in microprogram control definition, we will use the CONTINUE (CONT) statement to mean simply incrementing to the next microinstruction.

## MICROPROGRAM JUMPING

If the microprogram control unit is to have the ability to select other than the next microinstruction, the control unit must be able to load a JUMP address. The load control of a counter can be a single bit field within the microprogram word format. Let us call this one-bit field the microprogram address counter load enable bit. When this bit is at logic 0, a load will be inhibited and when this bit is a logic 1, a load will be enabled. If the load is enabled, the JUMP address contained within the microprogram memory will be parallel loaded into the microprogram address counter. This results in the ability to perform an N-way branch. For example, if the branch address field is eight bits wide, a JUMP to any address in the memory space from word 0 through word 255 can be performed.

This simple branching control feature allows a microprogram memory controller to execute sequential microinstructions or perform a JUMP (JMP) to any address either before or after the address currently contained in the microprogram address counter.

## CONDITIONAL JUMPING

While the JUMP instruction has added some flexibility to the sequencing of microprogram instructions, the controller still lacks any decision-making capability. This decision-making capability is provided by the CONDITIONAL JUMP (COND JMP) instruction. Figure 1 shows a functional block diagram of a microprogram memory/address controller providing the capability to jump on either of two different conditions. In this example, the load select control is a two-bit field used to control a

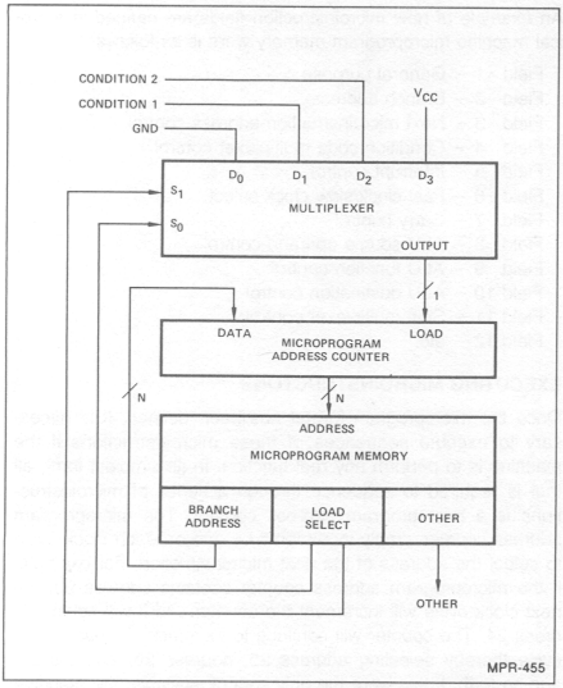


**Figure 1. A Two-Bit Control Field Can be Used to Select CONTINUE, BRANCH, or CONDITIONAL BRANCH.**

four-input multiplexer. When the two-bit field is equivalent to binary zero, the multiplexer selects the zero input which forces the load control inactive. Thus, the CONTINUE microprogram control instruction is executed. When the two-bit load select field contains binary one, the $D_1$ input of the multiplexer is selected. Now, the load control is a function of the Condition 1 input. If Condition 1 is logic 0, the microprogram address counter increments and if Condition 1 is logic 1, the jump address will be parallel loaded in the next clock cycle. This operation is defined as a CONDITIONAL JUMP. If the load select input contains binary 2, the $D_2$ input is selected and the same conditional function is performed with respect to the Condition 2 input. If the load select field contains binary 3, the $D_3$ input of the multiplexer is selected. Since the $D_3$ input is tied to logic HIGH, this forces the microprogram address counter to the load mode independent of anything else. Thus, the jump address is loaded into the microprogram address counter on the next clock cycle and an UNCONDITIONAL JUMP is executed. This load select control function definition is shown in Table 1.

**TABLE 1.**
**LOAD SELECT CONTROL FUNCTION.**

| $S_1$ $S_0$ | Function |
|---|---|
| 0 0 | Continue |
| 0 1 | Jump Condition 1 True |
| 1 0 | Jump Condition 2 True |
| 1 1 | Jump Unconditional |

## OVERLAPPING THE MICROPROGRAM INSTRUCTION FETCH

Now that a few basic microprogram address control instructions have been defined, let us examine the control instructions used in a microprogram control unit featuring the overlap fetching of the next microinstruction. This technique is also known as "pipelining". The block diagram for such a microprogram control unit is shown in Figure 2. The key difference when compared with previous microprogrammed architectures is the existence of the "pipeline register" at the output of the microprogram memory. By definition, the pipeline register (or microword register) contains the microinstruction currently being executed by the machine. Simultaneously, while this microinstruction is being executed, the address of the next microinstruction is applied to the microprogram memory and the contents of that memory word are being fetched and set-up at the inputs to the pipeline register. This technique of pipelining can be used to improve the performance of the microprogram control unit. This results because the contents of the microprogram memory word required for the next cycle are being fetched on an overlapping basis with the actual execution of the current microprogram word. It should be realized that when the pipeline approach is used, the design engineer must be aware of the fact that some registers contain the results of the previous microinstruction executed, some registers contain the current microinstruction being executed, and some registers contain data for the next microinstruction to be executed.

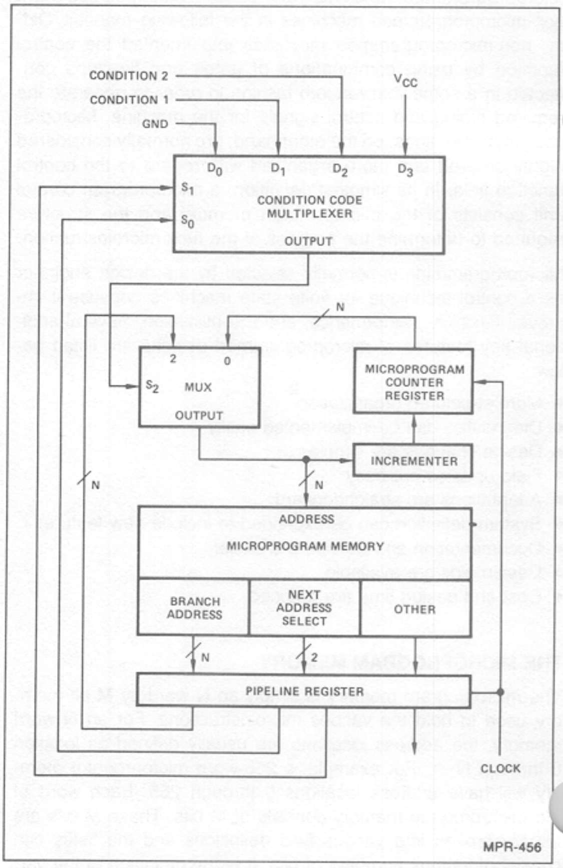


**Figure 2. Overlapping (or Pipelining) the Fetch of the Next Microinstruction.**

Let us now compare the block diagram of Figure 2 with that shown in Figure 1. The major difference, of course, is the addition of the pipeline register at the output of the microprogram control memory. Also, notice the addition of the address multiplexer at the source of the microprogram memory address. This address multiplexer is used to select the microprogram counter register or the pipeline register as the source of the next address for the microprogram memory. The condition code multiplexer is used to control the address multiplexer in this address selection. By placing an incrementer at the output of the address multiplexer, is is possible to always generate the current microprogram address "plus one" at the input of the microprogram counter register.

In Figure 1, the microprogram address counter was described as a counter and could be a device such as the Am25LS161 counter. In the implementation as shown in Figure 2, the Am25LS161 counter is not appropriate. Instead, an incrementer and register are used to give the equivalent effect of a counter.

The key difference between using a true binary counter and the incrementer register described here is as follows. When the jump address from the pipeline register is selected by the multiplexer, the incrementer will combinatorially prepare that address plus one for entry into the microprogram counter register. This entry will occur on the LOW-to-HIGH transition of the clock. Thus, the microprogram counter register can always be made to contain address plus one, independent of the selection of the next microinstruction address. When the address multiplexer is switched so that the microprogram counter register is selected as the source of the microprogram memory address, the incrementer will again set-up address plus one for entry into the microprogram counter register. Thus, when the address multiplexer selects the microprogram counter register, the address multiplexer, incrementer and microprogram counter register appear to operate as a normal binary counter.

The condition code multiplexer $S_0S_1$ operates in exactly the same fashion as described for the condition code multiplexer of Figure 1. That is, binary zero in the pipeline register (the current microinstruction being executed) forces an unconditional selection of the microprogram register via $D_0$. Binary one or binary two in the next address select control bits of the pipeline register cause a conditional selection at the address multiplexer via $D_1$ or $D_2$. Thus, a CONDITIONAL JUMP can be executed. Binary three in the next address select portion of the pipeline register causes an UNCONDITIONAL JUMP instruction to be executed via $D_3$.

When the overall machine timing is studied, it will be observed that the key difference between overlap fetching and non-overlap fetching involves the propagation delay of the microprogram memory. In the non-pipelined architecture, the microprogram memory propagation delay must be added to the propagation delay of all the other elements of the machine. In the overlap fetch architecture, the propagation delay associated with the next microprogram memory address fetch is a separate loop independent of the other portion of the machine.

## SUBROUTINING IN MICROPROGRAMMING CONTROL

Thus far, we have examined the CONTINUE instruction as well as the CONDITIONAL and UNCONDITIONAL JUMP instructions for overlap fetch. Just as in the programming of minicomputers and microcomputers, the advantages of SUBROUTINING can be realized in microprogramming. The idea here, of course, is that the same block of microcode (or even a single microinstruction) can be shared by several microinstruction sequences. This results in an overall reduction in the total number of microprogram memory words required by the design. If we are to jump to a subroutine, what is required is the ability to store an address to which the subroutine should return when it has completed its execution. Examining the block diagram of Figure 3, we see the addition of a subroutine and loop (push/pop) stack (also called the file) and its associated stack pointer. The control signals required by the stack are an enable stack signal (FILE ENABLE = FE) which will be used to tell the file whenever we wish to perform a push or a pop, and a push/pop control (PUP) used to control the direction of the stack pointer (push or pop).

In this architecture, the stack pointer always points to the address of the last microinstruction written on the stack. This allows the "next address multiplexer" to read the stack at any time via port F. When this selection is performed, the last word written on the stack will be the word applied to the microprogram memory. The condition code multiplexer of the previous example has also been replaced by a next address control unit. This next address control unit (Am29811A) can execute 16 different next address control functions where most of these functions are conditional. Thus, the device has four instruction inputs as well as one condition code test input which is connected to the condition code multiplexer. Note also that the next address control field of the microprogram word has been expanded to a four-bit field. Outputs from the Am29811A next address control block are used to control the stack pointer and the next address multiplexer of the Am2911. In addition, the device has outputs to control the three-state enable of the pipeline register and the three-state enable of the starting address decode PROM. Also, the architecture has a counter that can be used as a loop-counter or event counter.

The 16 instructions associated with the Am29811A are listed in Table 2. As is easily seen by referring to Table 2, three of the instructions in this set are associated with subroutining in microprogram memory. The first instruction of this set, is a simple conditional JUMP-TO-SUBROUTINE where the source of the subroutine address is in the pipeline register. The RETURN-FROM-SUBROUTINE instruction is also conditional and is used to return to the next microinstruction following the JUMP-TO-SUBROUTINE instruction. There is also a conditional JUMP-TO-ONE-OF-TWO-SUBROUTINES, where the subroutine address is either in the PIPELINE register or in the internal REGISTER in the Am2911. This instruction will be explained in more detail later.

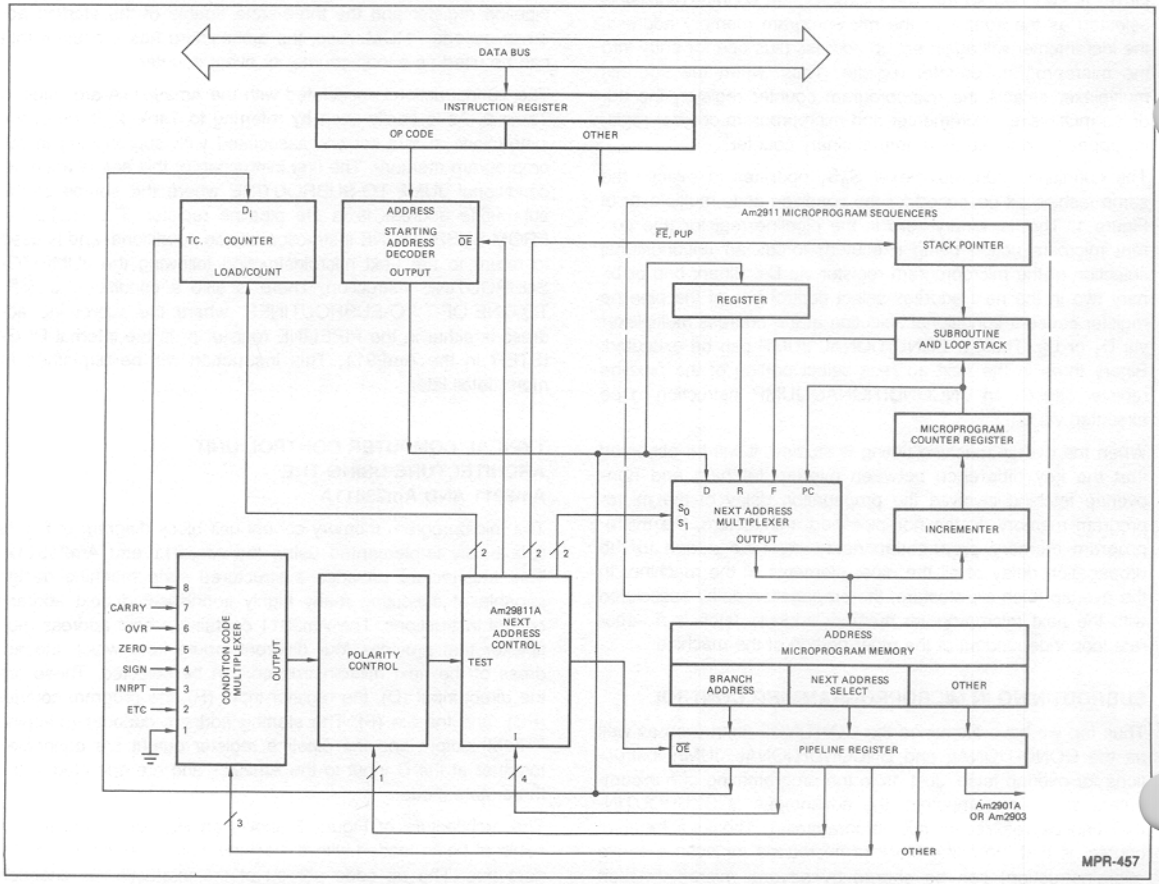
## TYPICAL COMPUTER CONTROL UNIT ARCHITECTURE USING THE Am2911 AND Am29811A

The microprogram memory control unit block diagram of Figure 3 is easily implemented using the Am2911 and Am29811A. This architecture provides a structured state machine design capable of executing many highly sophisticated next address control instructions. The Am2911 contains a next address multiplexer that provides four different inputs from which the address of the next microinstruction can be selected. These are the direct input (D), the register input (R), the program counter (PC), and the file (F). The starting address decoder (mapping PROM) output and the pipeline register output are connected together at the D input to the Am2911 and are operated in the three-state mode.

The architecture of Figure 3 shows an instruction register capable of being loaded with a machine instruction word from the data bus. The op code portion of the instruction is decoded using a mapping PROM to arrive at a starting address for the

## TABLE 2. FUNCTIONAL DESCRIPTION OF Am29811A INSTRUCTION SET.

| | INPUTS | | | OUTPUTS | | | | |
|---|---|---|---|---|---|---|---|---|
| MNEMONIC | INSTRUCTION $I_3$ $I_2$ $I_1$ $I_0$ | FUNCTION | TEST INPUT | NEXT ADDR SOURCE | FILE | COUNTER | MAP-E | PL-E |
| JZ | L L L L | JUMP ZERO | X | D | HOLD | L L | H | L |
| CJS | L L L H | COND JSB PL | L | PC | HOLD | HOLD | H | L |
| | | | H | D | PUSH | HOLD | H | L |
| JMAP | L L H L | JUMP MAP | X | D | HOLD | HOLD | L | H |
| CJP | L L H H | COND JUMP PL | L | PC | HOLD | HOLD | H | L |
| | | | H | D | HOLD | HOLD | H | L |
| PUSH | L H L L | PUSH/COND LD CNTR | L | PC | PUSH | HOLD | H | L |
| | | | H | PC | PUSH | LOAD | H | L |
| JSRP | L H L H | COND JSB R/PL | L | R | PUSH | HOLD | H | L |
| | | | H | D | PUSH | HOLD | H | L |
| CJV | L H H L | COND JUMP VECTOR | L | PC | HOLD | HOLD | H | H |
| | | | H | D | HOLD | HOLD | H | H |
| JRP | L H H H | COND JUMP R/PL | L | R | HOLD | HOLD | H | L |
| | | | H | D | HOLD | HOLD | H | L |
| RFCT | H L L L | REPEAT LOOP, CNTR ≠ 0 | L | F | HOLD | DEC | H | L |
| | | | H | PC | POP | HOLD | H | L |
| RPCT | H L L H | REPEAT PL, CNTR ≠ 0 | L | D | HOLD | DEC | H | L |
| | | | H | PC | HOLD | HOLD | H | L |
| CRTN | H L H L | COND RTN | L | PC | HOLD | HOLD | H | L |
| | | | H | F | POP | HOLD | H | L |
| CJPP | H L H H | COND JUMP PL & POP | L | PC | HOLD | HOLD | H | L |
| | | | H | D | POP | HOLD | H | L |
| LDCT | H H L L | LOAD CNTR & CONTINUE | X | PC | HOLD | LOAD | H | L |
| LOOP | H H L H | TEST END LOOP | L | F | HOLD | HOLD | H | L |
| | | | H | PC | POP | HOLD | H | L |
| CONT | H H H L | CONTINUE | X | PC | HOLD | HOLD | H | L |
| JP | H H H H | JUMP PL | X | D | HOLD | HOLD | H | L |



MPR-457

**Figure 3. A Typical Computer Control Unit Using the Am2911 and Am29811A.**

## TABLE 3. PIN FUNCTIONS.

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

microinstruction sequence required to execute the machine instruction. When the microprogram memory address is to be the first microinstruction of the machine instruction sequence, the Am29811A next address control unit selects the multiplexer D input and enables the three-state output from the mapping PROM. When the current microinstruction being executed is selecting the next microinstruction address as a JUMP function, the JUMP address will be available at the multiplexer D input. This is accomplished by having the Am29811A select the next address multiplexer D input and also enabling the three-state output of the pipeline register branch address field. The register enable input to the Am2911 is connected to ground so that this register will always load the value at the Am2911 D input. The value at D is clocked into the Am2911's register (R) at the end of the current microcycle, which makes the D value of *this* microcycle available as the R value of the *next* microcycle. Thus, by using the branch address field of two sequential microinstructions, a conditional JUMP-TO-ONE-OF-TWO-SUBROUTINES or a conditional JUMP-TO-ONE-OF-TWO-BRANCH-ADDRESSES can be executed by either selecting the D input or the R input of the next address multiplexer.

When sequencing through continuous microinstructions in microprogram memory, the program counter in the Am2911 is used. Here, the Am29811A simply selects the PC input of the next address multiplexer. In addition, most of these instructions enable the three-state outputs of the pipeline register associated with the branch address field, which allows the register within the Am2911 to be loaded.

The 4 x 4 stack in the Am2911 is used for looping and subroutining in microprogram operations. Up to four levels of subroutines or loops can be nested. Also, loops and subroutines can be intermixed as long as the four-word depth of the stack is not exceeded.

### ARCHITECTURE OF THE Am2910

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

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

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

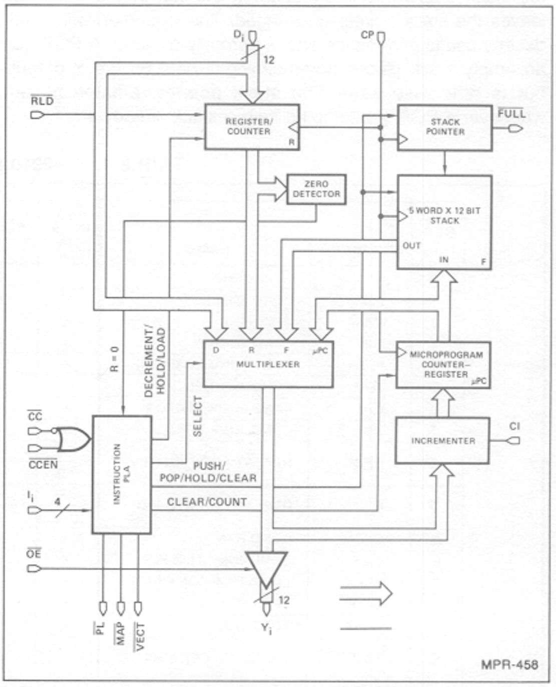


**Figure 4. Am2910 Block Diagram.**

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

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

The fourth source available at the multiplexer input is a 5-word by 12-bit stack (file). The stack is used to provide return address linkage when executing microsubroutines or loops. The stack contains a build-in stack pointer (SP) which always points to the last file word written. This allows stack reference operations (looping) to be performed without a pop. The stack pointer operates as an up/down counter. During microinstructions 2, 4 and 5, the PUSH operation is performed. This causes the stack pointer to increment and the file to be written with the required return linkage. On the cycle following the PUSH, the return data is at the new location pointed to by the stack pointer.

During six other microinstructions, a POP operation occurs. This places the information at the top of the stack onto the Y outputs. The stack pointer decrements at the next rising clock edge following a POP, effectively removing old information from the top of the stack.

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

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

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

## OPERATION

Table 4 shows the result of each instruction in controlling the multiplexer which determines the Y outputs, and in controlling the three enable signals $\overline{\text{PL}}$, $\overline{\text{MAP}}$ and $\overline{\text{VECT}}$. The effect on the $\mu$PC, the register/counter, and the stack after the next positive-going clock edge is also shown. The multiplexer determines which internal source drives the Y outputs. The value loaded into $\mu$PC is either identical to the Y output, or else one greater, as determined by CI. For each instruction, one and only one of the three outputs $\overline{\text{PL}}$, $\overline{\text{MAP}}$ and $\overline{\text{VECT}}$ is LOW. If these outputs control three-state enables for the primary source of microprogram jumps (usually part of a pipeline register), a PROM which maps the instruction to a microinstruction starting location, and an optional third source (often a vector from a DMA or interrupt source), respectively, the three-state sources can drive the D inputs without further logic.

Several inputs, as shown in Table 4 can modify instruction execution. The combination $\overline{\text{CC}}$ HIGH and $\overline{\text{CCEN}}$ LOW is used as a test in 10 of the 16 instructions. $\overline{\text{RLD}}$, when LOW, causes the D input to be loaded into the register/counter, overriding any HOLD or DEC operation specified in the instruction. $\overline{\text{OE}}$, normally LOW, may be forced HIGH to remove the Am2910 Y outputs from a three-state bus.

**TABLE 4. Am2910 MICROINSTRUCTION SET.**

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

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

The stack, a five-word last-in, first-out 12-bit memory, has a pointer which addresses the value presently on the top of the stack. Explicit control of the stack pointer occurs during instruction 0 (RESET), which makes the stack empty by resetting the SP to zero. After a RESET, and whenever else the stack is empty, the content of the top of stack is undefined until a PUSH occurs. Any POPs performed while the stack is empty put undefined data on the F outputs and leave the stack pointer at zero. Any time the stack is full (five more PUSHes than POPs have occurred since the stack was last empty), the $\overline{\text{FULL}}$ warning output occurs. No additional PUSH should be attempted onto a full stack; if tried, information at the top of the stack will be overwritten and lost.

## THE Am2910 INSTRUCTION SET

The Am2910 provides 16 instructions which select the address of the next microinstruction to be executed. Four of the instructions are unconditional – their effect depends only on the instruction. Ten of the instructions have an effect which is partially controlled by an external, data-dependent condition. Three of the instructions have an effect which is partially controlled by the contents of the internal register/counter. The instruction set is shown in Table 4. In this discussion it is assumed that CI is tied HIGH.

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

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

Perhaps the best technique for understanding the Am2910 is to simply take each instruction and review its operation. In order to provide some feel for the actual execution of these instructions, Figure 5 is included and depicts examples of all 16 instructions.

The examples given in Figure 5 should be interpreted in the following manner: The intent is to show microprogram flow as various microprogram memory words are executed. For example, the CONTINUE instruction, instruction number 14, as shown in Figure 5, simply means that the contents of microprogram memory word 50 is executed, then the contents of word 51 is executed. This is followed by the contents of microprogram memory word 52 and the contents of microprogram memory word 53. The microprogram addresses used in the examples were arbitrarily chosen and have no meaning other than to show instruction flow. The exception to this is the first example, JUMP ZERO, which forces the microprogram location counter to address ZERO. Each dot refers to the time that the contents of the microprogram memory word is in the pipeline register. While no special symbology is used for the conditional instructions, the text to follow will explain what the conditional choices are in each example.

It might be appropriate at this time to mention that AMD has a microprogram assembler called AMDASM, which has the capability of using the Am2910 instructions in symbolic representation. AMDASM's Am2910 instruction symbolics (or mnemonics) are given in Figure 5 for each instruction and are also shown in Table 4.

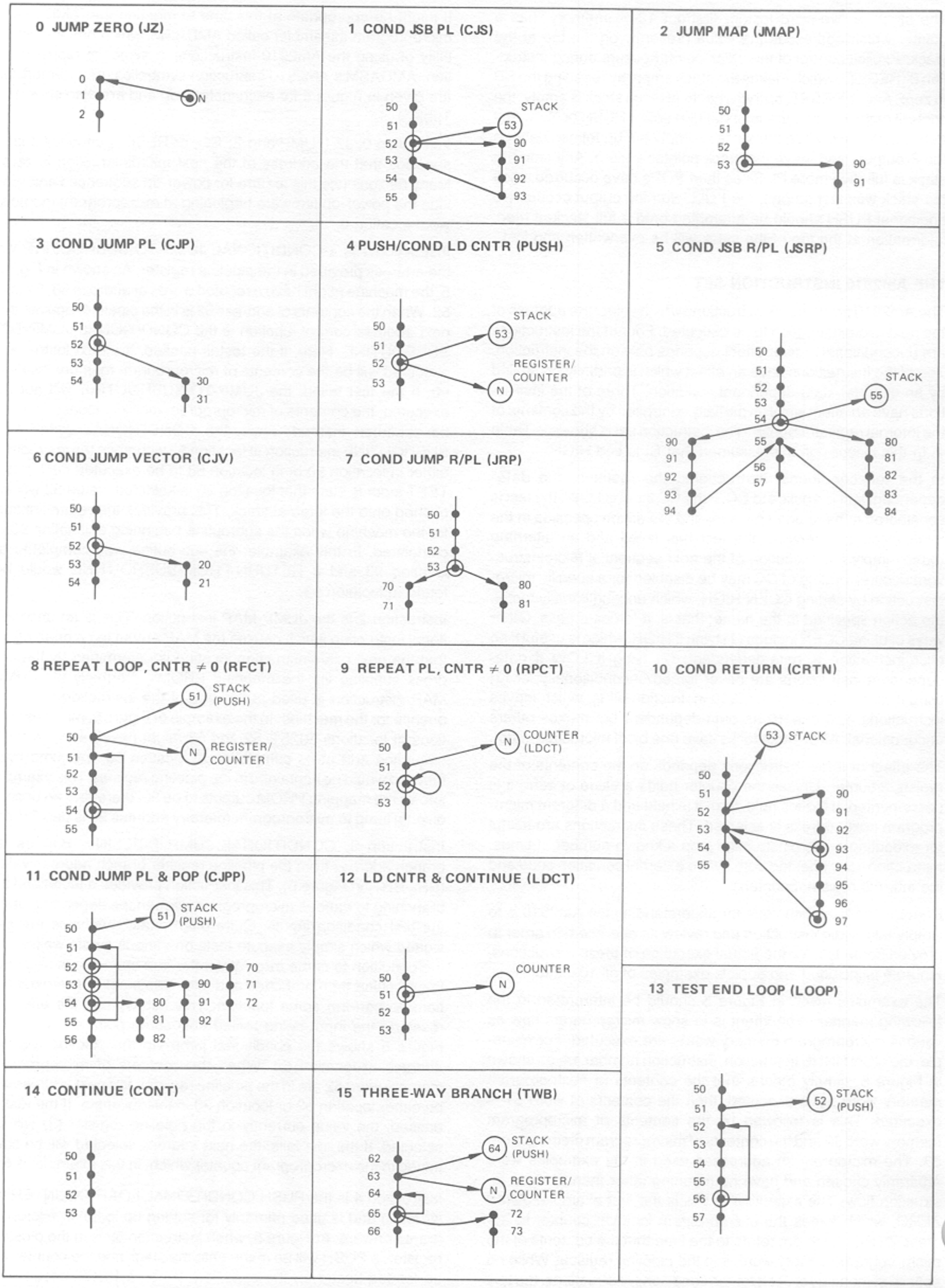
Instruction 0, JZ (JUMP and ZERO, or RESET) unconditionally specifies that the address of the next microinstruction is zero. Many designs use this feature for power-up sequences and provide the power-up firmware beginning at microprogram memory word location 0.

Instruction 1 is a CONDITIONAL JUMP-TO-SUBROUTINE via the address provided in the pipeline register. As shown in Figure 5, the machine might have executed words at address 50, 51 and 52. When the contents of address 52 is in the pipeline register, the next address control function is the CONDITIONAL JUMP-TO-SUBROUTINE. Here, if the test is passed, the next instruction executed will be the contents of microprogram memory location 90. If the test failed, the JUMP-TO-SUBROUTINE will not be executed; the contents of microprogram memory location 53 will be executed instead. Thus, the CONDITIONAL JUMP-TO-SUBROUTINE instruction at location 52 will cause the instruction either in location 90 or in location 53 to be executed next. If the TEST input is such that location 90 is selected, value 53 will be pushed onto the internal stack. This provides the return linkage for the machine when the subroutine beginning at location 90 is completed. In this example, the subroutine was completed at location 93 and a RETURN-FROM-SUBROUTINE would be found at location 93.

Instruction 2 is the JUMP MAP instruction. This is an unconditional instruction which causes the $\overline{\text{MAP}}$ output to be enabled so that the next microinstruction location is determined by the address supplied via the mapping PROMs. Normally the JUMP MAP instruction is used at the end of the instruction fetch sequence for the machine. In the example of Figure 5, microinstructions at locations 50, 51, 52 and 53 might have been the fetch sequence and at its completion at location 53, the jump map function would be contained in the pipeline register. This example shows the mapping PROM outputs to be 90; therefore, an unconditional jump to microprogram memory address 90 is performed.

Instruction 3, CONDITIONAL JUMP PIPELINE, derives its branch address from the pipeline register branch address value ($BR_0$-$BR_{11}$ in Figure 6). This instruction provides a technique for branching to various microprogram sequences depending upon the test condition inputs. Quite often, state machines are designed which simply execute tests on various inputs waiting for the condition to come true. When the true condition is reached, the machine then branches and executes a set of microinstructions to perform some function. This usually has the effect of resetting the input being tested until some point in the future. Figure 5 shows the conditional jump via the pipeline register address at location 52. When the contents of microprogram memory word 52 are in the pipeline register, the next address will be either location 53 or location 30 in this example. If the test is passed, the value currently in the pipeline register (3) will be selected. If the test fails, the next address selected will be contained in the microprogram counter which, in this example, is 53.

Instruction 4 is the PUSH/CONDITIONAL LOAD COUNTER instruction and is used primarily for setting up loops in microprogram firmware. In Figure 5, when instruction 52 is in the pipeline register, a PUSH will be made onto the stack and the counter will be loaded based on the condition. When a PUSH occurs, the value pushed is always the next sequential instruction address. In this case, the address is 53. If the test fails, the counter is not



MPR-111

Figure 5. Am2910 Execution Examples.

loaded; if it is passed, the counter is loaded with the value contained in the pipeline register branch address field. Thus, a single microinstruction can be used to set up a loop to be executed a specific number of times. Instruction 8 will describe how to use the pushed value and the register/counter for looping.

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

Instruction 6 is a CONDITIONAL JUMP VECTOR instruction which provides the capability to take the branch address from a third source heretofore not discussed. In order for this instruction to be useful, the Am2910 output, $\overline{\text{VECT}}$, is used to control a three-state control input of a register, buffer, or PROM containing the next microprogram address. This instruction provides one technique for performing interrupt type branching at the microprogram level. Since this instruction is conditional, a pass causes the next address to be taken from the vector source, while failure causes the next address to be taken from the microprogram counter. In the example of Figure 5, if the CONDITIONAL JUMP VECTOR instruction is contained at location 52, execution will continue at vector address 20 if the TEST input is HIGH and the microinstruction at address 53 will be executed if the TEST input is LOW.

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

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

An example of the REPEAT LOOP, COUNTER ≠ ZERO instruction is shown in Figure 5. In this example, location 50 most likely would contain a PUSH/CONDITIONAL LOAD COUNTER instruction which would have caused address 51 to be PUSHed on the stack and the counter to be loaded with the proper value for looping the desired number of times.

In this example, since the loop test is made at the end of the instructions to be repeated (microaddress 54), the proper value to be loaded by the instruction at address 50 is one less than the desired number of passes through the loop. This method allows a loop to be executed from 0 to 4095 times.

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

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

In the example of Figure 5, the REPEAT PIPELINE, COUNTER ≠ ZERO instruction is instruction 52 and is shown as a single microinstruction loop. The address in the pipeline register would be 52. Instruction 51 in this example could be the LOAD COUNTER AND CONTINUE instruction (number 12). While the example shows a single microinstruction loop, by simply changing the address in a pipeline register, multi-instruction loops can be performed in this manner for a fixed number of times as determined by the counter.

Instruction 10 is the conditional RETURN-FROM-SUBROUTINE instruction. As the name implies, this instruction is used to branch from the subroutine back to the next microinstruction address following the subroutine call. Since this instruction is conditional, the return is performed only if the test is passed. If the test is failed, the next sequential microinstruction is performed. The example in Figure 5 depicts the use of the conditional RETURN-FROM-SUBROUTINE instruction in both the conditional and the unconditional modes. This example first shows a jump-to-subroutine at instruction location 52 where control is transferred to location 90. At location 93, a conditional RETURN-FROM-SUBROUTINE instruction is performed. If the test is passed, the stack is accessed and the program will transfer to the next instruction at address 53. If the test is failed, the next microinstruction at address 94 will be executed. The program will continue to address 97 where the subroutine is complete. To perform an unconditional RETURN-FROM-SUBROUTINE, the conditional RETURN-FROM-SUBROUTINE instruction is executed unconditionally; the microinstruction at address 97 is programmed to force $\overline{\text{CCEN}}$ HIGH, disabling the test and the forced PASS causes an unconditional return.

Instruction 11 is the CONDITIONAL JUMP PIPELINE register address and POP stack instruction. This instruction provides another technique for loop termination and stack maintenance.

The example in Figure 5 shows a loop being performed from address 55 back to address 51. The instructions at locations 52, 53 and 54 are all conditional JUMP and POP instructions. At address 52, if the TEST input is passed, a branch will be made to address 70 and the stack will be properly maintained via a POP. Should the test fail, the instruction at location 53 (the next sequential instruction) will be executed. Likewise, at address 53, either the instruction at 90 or 54 will be subsequently executed, respective to the test being passed or failed. The instruction at 54 follows the same rules, going to either 80 or 55. An instruction sequence as described here, using the CONDITIONAL JUMP PIPELINE and POP instruction, is very useful when several inputs are being tested and the microprogram is looping waiting for any of the inputs being tested to occur before proceeding to another sequence of instructions. This provides the powerful jump-table programming technique at the firmware level.

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

Instruction 13 is the TEST END-OF-LOOP instruction, which provides the capability of conditionally exiting a loop at the bottom; that is, this is a conditional instruction that will cause the microprogram to loop, via the file, if the test is failed else to continue to the next sequential instruction. The example in Figure 5 shows the TEST END-OF-LOOP microinstruction at address 56. If the test fails, the microprogram will branch to address 52. Address 52 is on the stack because a PUSH instruction had been executed at address 51. If the test is passed at instruction 56, the loop is terminated and the next sequential microinstruction at address 57 is being executed, which also causes the stack to be POPd; thus, accomplishing the required stack maintenance.

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

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

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

As one example, consider the case of a memory search instruction. As shown in Figure 5, the instruction at microprogram address 63 can be Instruction 4 (PUSH), which will push the value 64 onto the microprogram stack and load the number N, which is one less than the number of memory locations to be searched before giving up. Location 64 contains a microinstruction which fetches the next operand from the memory area to be searched and compares it with the search key. Location 65 contains a microinstruction which tests the result of the comparison and also is a THREE-WAY BRANCH for microprogram control. If no match is found, the test fails and the microprogram goes back to location 64 for the next operand address. When the count becomes zero, the microprogram branches to location 72, which does whatever is necessary if no match is found. If a match occurs on any execution of the THREE-WAY BRANCH at location 65, control falls through to location 66 which handles this case. Whether the instruction ends by finding a match or not, the stack will have been POP'd once, removing the value 64 from the top of the stack.

### Am29811A Instruction Set Difference

The Am29811A instruction set is identical to the Am2910 except for instruction number 15. In the Am29811A, instruction number 15 is an unconditional JUMP PIPELINE REGISTER instruction. This provides the ability to unconditionally branch to any address contained in the branch address field of the microprogram. Thus, an unconditional N-way branch can be performed. Use of this instruction as opposed to a forced conditional jump pipeline instruction simply allows the condition code multiplexer select field to be shared (formatted) with other functions.

## TYPICAL COMPUTER CONTROL UNIT ARCHITECTURE USING THE Am2910

The microprogram memory control unit block diagram of Figure 6 is easily implemented using the Am2910. This architecture provides a structured state machine design capable of executing many highly sophisticated next address control instructions.

The architecture of Figure 6 shows an instruction register capable of being loaded with a machine instruction word from the data bus. The op code portion of the instruction is decoded using a mapping PROM to arrive at a starting address for the microinstruction sequence required to execute the machine instruction. When the microprogram memory address is to be the first microinstruction of the machine instruction sequence, the Am2910 next address control selects the multiplexer D input and enables the three-state output from the mapping PROM. When the current microinstruction being executed is selecting the next microinstruction address as a JUMP function, the JUMP address will be available at the multiplexer D input. This is accomplished by having the Am2910 select the next address multiplexer D input and also enabling the three-state output of the pipeline register branch address field. The register enable input to the Am2910 can be grounded so that this register will load the value at the

Am2910 D input. The value at D is clocked into the Am2910's register (R) at the end of the current microcycle, which makes the D value of this microcycle available as the R value of the next microcycle. Thus, by using the branch address field of two sequential microinstructions, a conditional JUMP-TO-ONE-OF-TWO-SUBROUTINES or a conditional JUMP-TO-ONE-OF-TWO-BRANCH-ADDRESSES can be executed by either selecting the D input or the R input of the next address multiplexer.

When sequencing through continuous microinstructions in microprogram memory, the program counter in the Am2910 is used. Here, the control logic simply selects the PC input of the next address multiplexer. In addition, most of these instructions enable the three-state outputs of the pipeline register associated with the branch address field, which allows the register within the Am2910 to be loaded. The 5 x 12 stack in the Am2910 is used for looping and subroutining in microprogram operations. Up to five levels of subroutines or loops can be nested. Also, loops and subroutines can be intermixed as long as the five word depth of the stack is not exceeded.

## CCU TIMING

The minimum clock cycle that can be used in a CCU design is usually determined by the component delays along the longest "pipeline-register-clock to logic to pipeline-register-clock" path. At the beginning of any given clock cycle, data available at the output of the microprogram memory, counter status, and any other data and/or status fields, are latched into their associated pipeline registers. At this point, all delay paths begin. Visual inspection will not always point out the longest signal delay path.

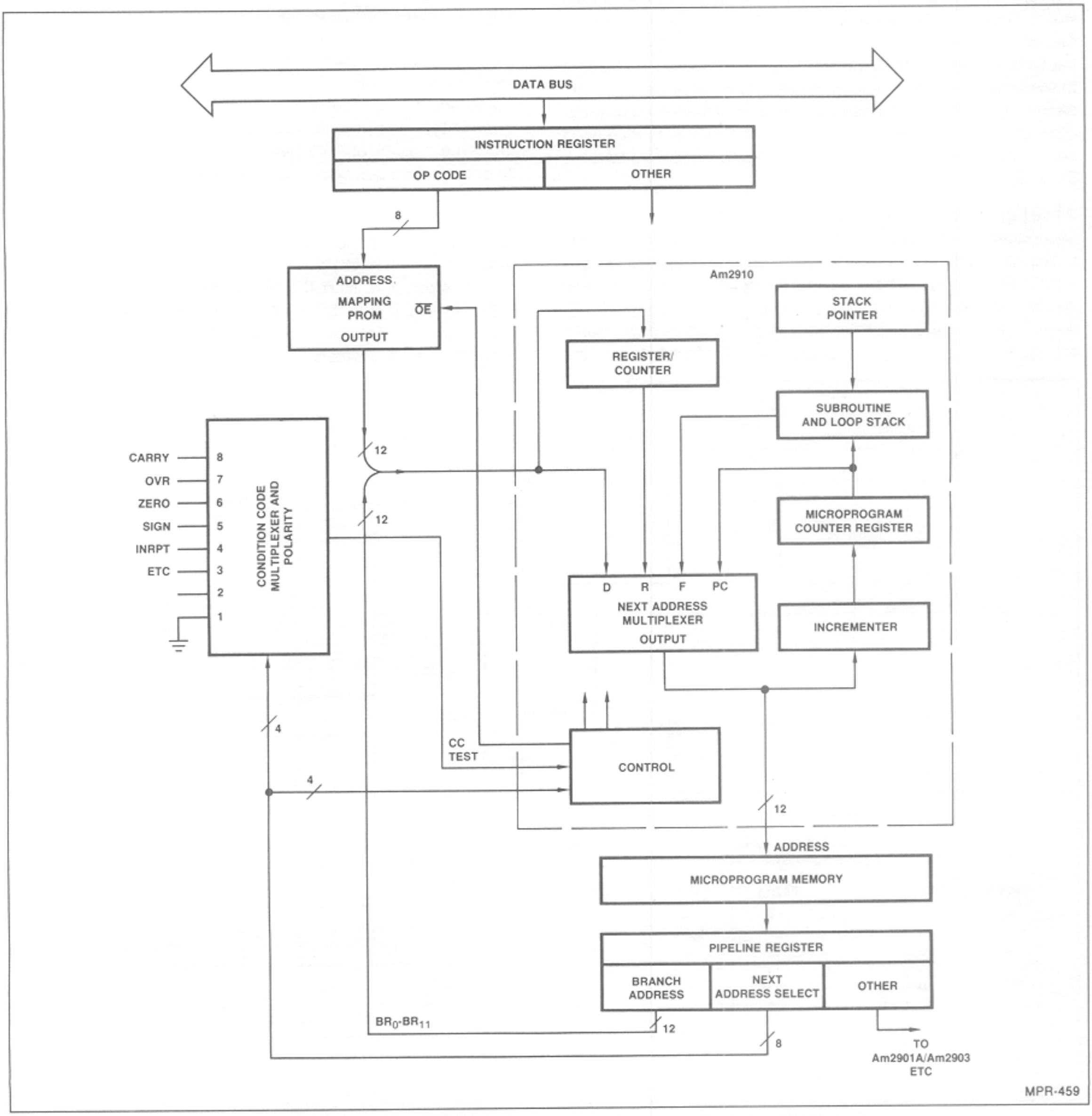


Figure 6. A Typical Computer Control Unit Using the Am2910.

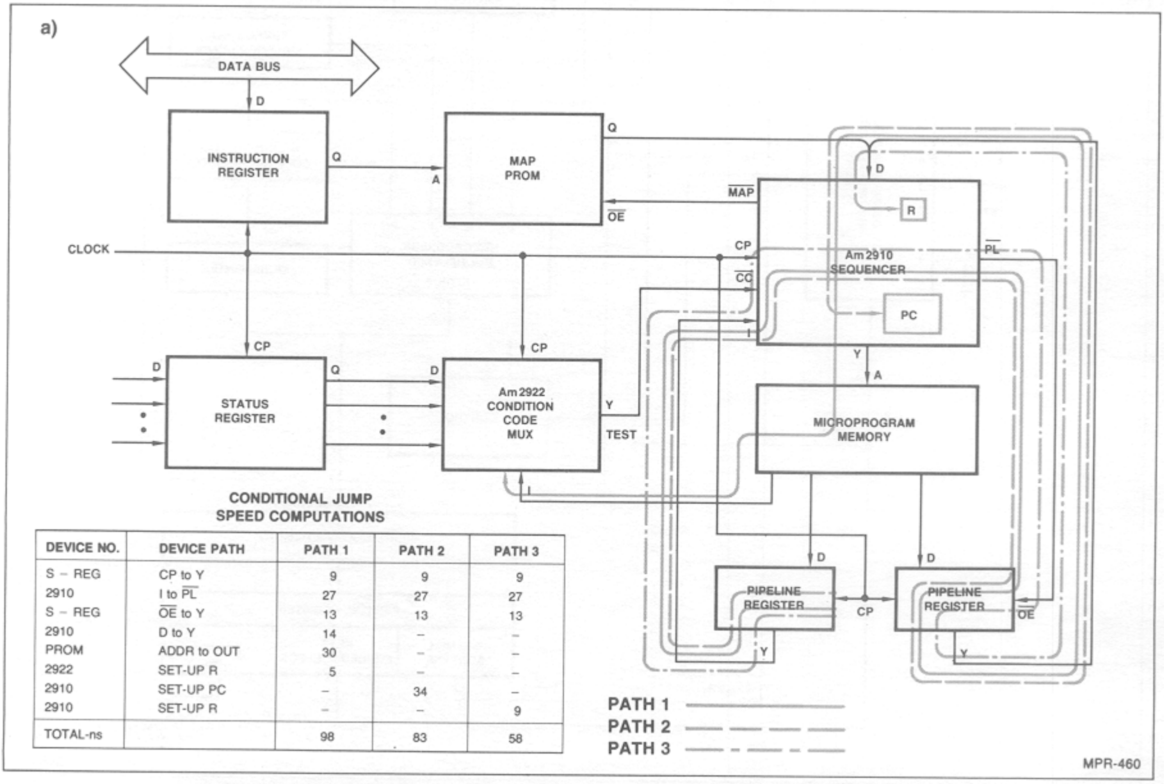
The obviously long paths are a good place to start, but each definable path should be calculated on a component by component basis until the truly longest logic signal path is found.

Referring to Figure 6, a number of potentially long paths can be identified. These include the instruction register to pipeline register time, the pipeline register to pipeline register time via the condition code multiplexer and the status to pipeline register time. In order to demonstrate the technique for calculating the AC performance of the Am2910 state machine design, the timing diagrams of Figure 7 are presented. Here, a number of propagation delay paths are evaluated such that the reader can learn the technique for performing these computations.

All of the propagation delays have been calculated using typical propagation delays because at the time of this writing, the characterization of the Am2910 has not been completed. When the final data sheet is published, the user need only select the appropriate worst case specifications and he can compute the desired maximum propagation delays for his design. Also, by looking at the typical propagation delay numbers, the designer will be able to evaluate the design margin in the system after he has completed all of the worst case calculations. These typical propagation delays represent the expected values if a system were set up on the bench and actual measurements would be taken at 5V and 25°C operating temperature.

While Figure 6 and Figure 7 deal with the Am2910 microprogram sequencer, it is also instructive to evaluate the AC performance of a typical computer control unit using the Am2911 and Am29811A. Figure 3 shows such a connection and will be used as the basis for performing the propagation delay path calculations. The calculations for the various propagation delay paths are demonstrated in Figure 8 and are intended to show the technique for computing these delays. As before, the typical propagation delays have been used in the computation for comparison purposes. The user can derive the maximum numbers at 25°C and 5V, commercial temperature range and power supply variations or military temperature range and power supply variations as required for his design.

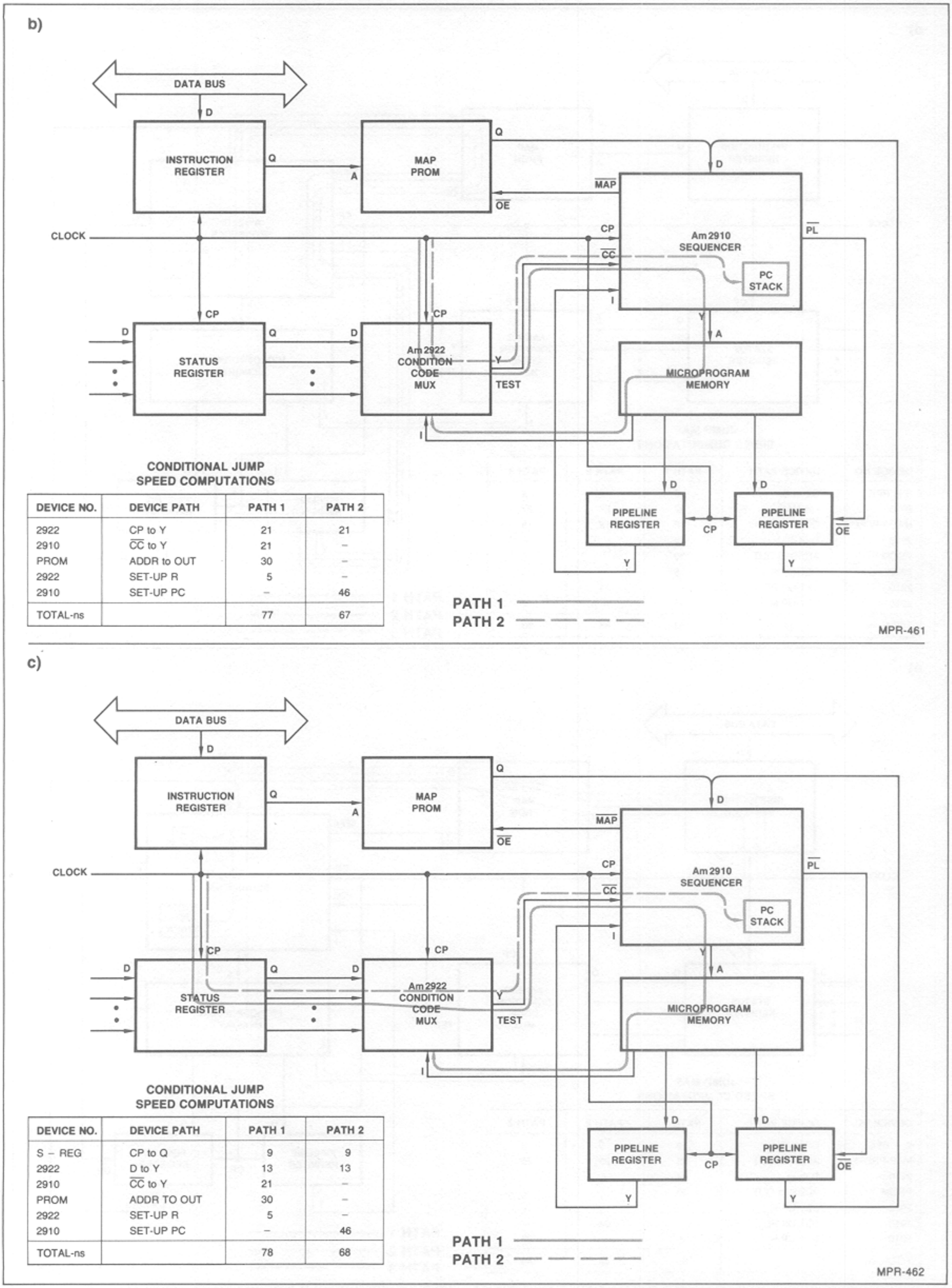
When Figure 7 and Figure 8 are reviewed in detail, the reader will recognize that the longest propagation delay paths in the case of the Am2910 as well as the Am2911 and Am29811A involve the three-state enables on the map PROM or the pipeline register for the branch address. If absolute maximum speed is desired, these paths can be eliminated by using one of several techniques. One technique is to simply allocate one or more bits in the pipeline register to control the three-state enables of the various devices connected to the D input of the Am2910. For the example of Figure 6, one bit would be sufficient and the pipeline register could be implemented using an Am74S175 register. This would allow the true and complement outputs to be used to drive the pipeline register branch address output enable and the mapping PROM output enable. Thus, these longest paths would be eliminated and an improvement of about 30ns would be achieved. A second technique for eliminating these propagation delay paths would be to use a four input NAND gate and a four input NOR gate to encode the equivalent function of the $\overline{\text{MAP}}$ enable and the $\overline{\text{PL}}$ enable. This technique is demonstrated in Figure 9. Again, an Am74S175 register would be used as the pipeline register to provide the instruction inputs to the Am2910 sequencer. This would allow instruction 2 to be decoded to provide the MAP enable signal and "NOT INSTRUCTION 2" to be decoded as the pipeline enable signal. This technique can be applied as well to the computer control unit of Figure 3 to accomplish the same longest path elimination.



CONDITIONAL JUMP SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| S – REG | CP to Y | 9 | 9 | 9 |
| 2910 | I to $\overline{\text{PL}}$ | 27 | 27 | 27 |
| S – REG | $\overline{\text{OE}}$ to Y | 13 | 13 | 13 |
| 2910 | D to Y | 14 | – | – |
| PROM | ADDR to OUT | 30 | – | – |
| 2922 | SET-UP R | 5 | – | – |
| 2910 | SET-UP PC | – | 34 | – |
| 2910 | SET-UP R | – | – | 9 |
| TOTAL-ns | | 98 | 83 | 58 |

**Figure 7. Propagation Delay Calculations on the Am2910 Microprogram Sequencer.**



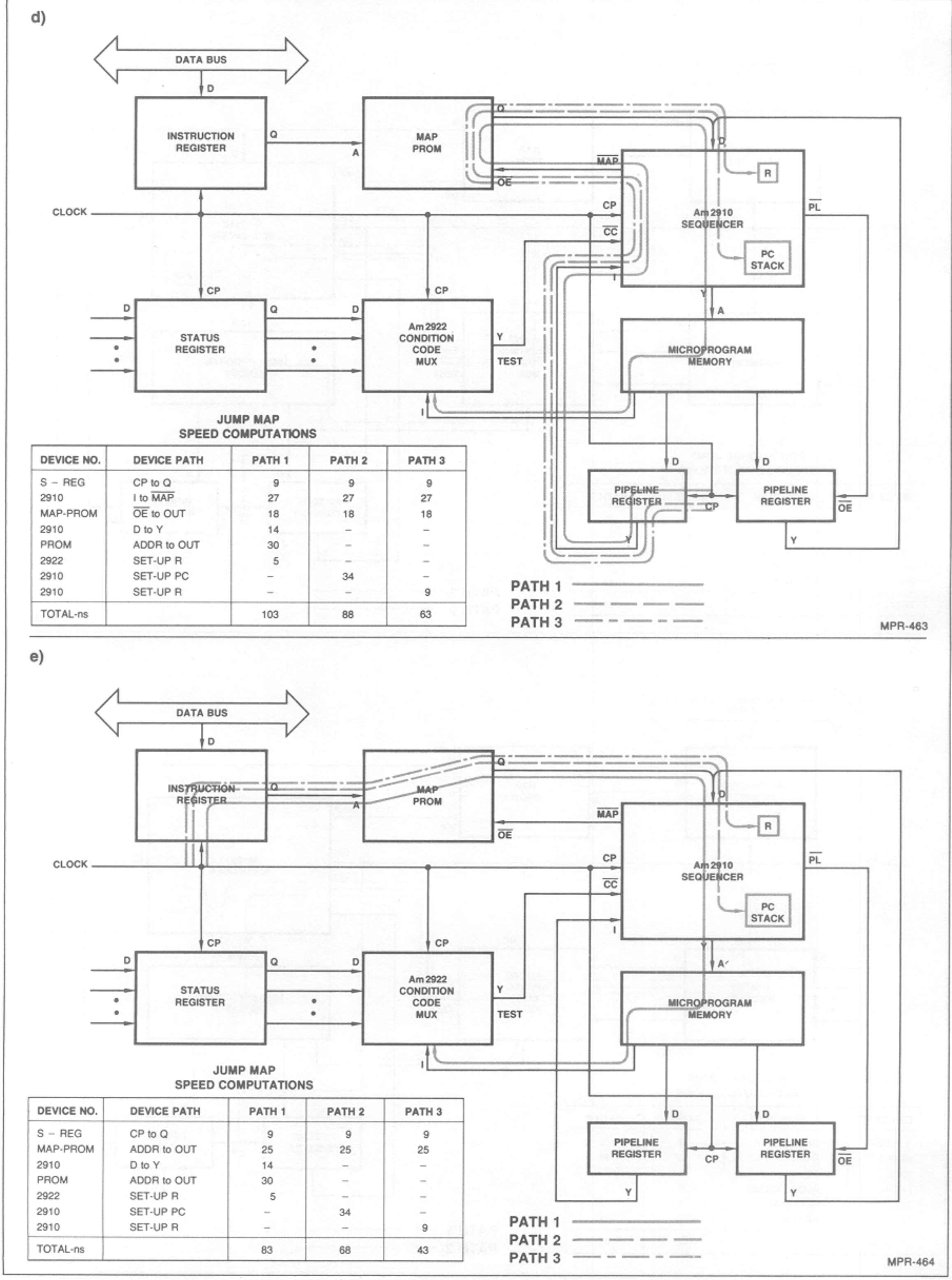
CONDITIONAL JUMP SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 2922 | CP to Y | 21 | 21 |
| 2910 | $\overline{CC}$ to Y | 21 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2910 | SET-UP PC | – | 46 |
| TOTAL-ns | | 77 | 67 |

CONDITIONAL JUMP SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| S – REG | CP to Q | 9 | 9 |
| 2922 | D to Y | 13 | 13 |
| 2910 | $\overline{CC}$ to Y | 21 | – |
| PROM | ADDR TO OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2910 | SET-UP PC | – | 46 |
| TOTAL-ns | | 78 | 68 |

Figure 7. Propagation Delay Calculations on the Am2910 Microprogram Sequencer (Cont.).



d) JUMP MAP SPEED COMPUTATIONS

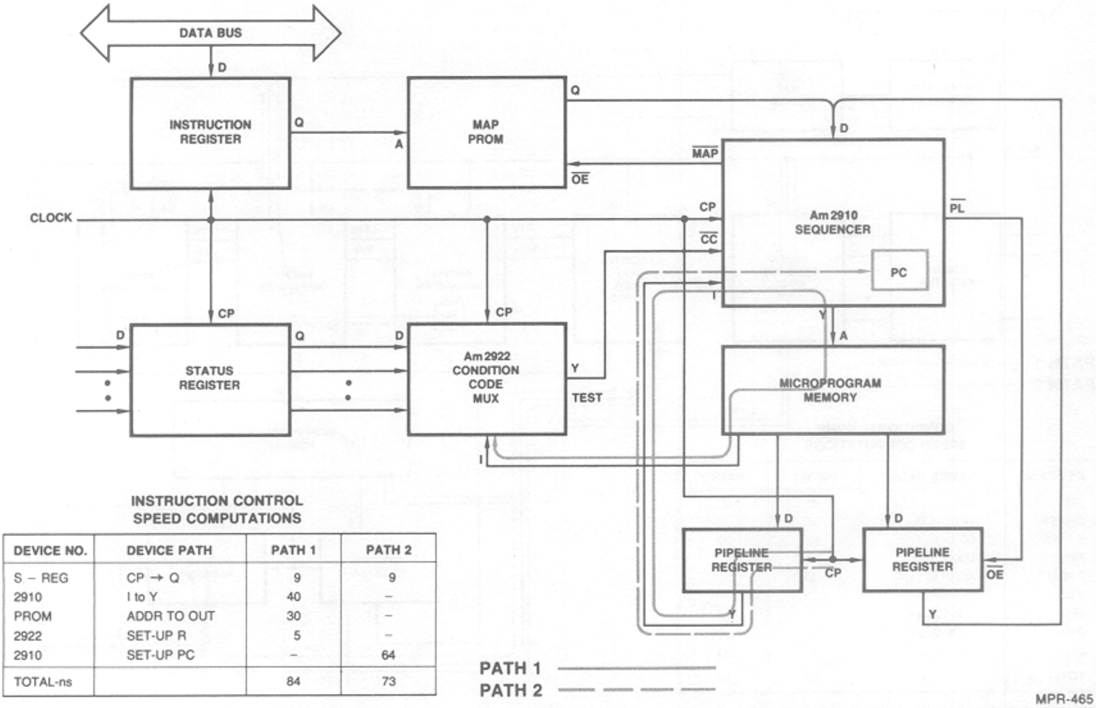
| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| S – REG | CP to Q | 9 | 9 | 9 |
| 2910 | I to $\overline{\text{MAP}}$ | 27 | 27 | 27 |
| MAP-PROM | $\overline{\text{OE}}$ to OUT | 18 | 18 | 18 |
| 2910 | D to Y | 14 | – | – |
| PROM | ADDR to OUT | 30 | – | – |
| 2922 | SET-UP R | 5 | – | – |
| 2910 | SET-UP PC | – | 34 | – |
| 2910 | SET-UP R | – | – | 9 |
| TOTAL-ns | | 103 | 88 | 63 |

e) JUMP MAP SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| S – REG | CP to Q | 9 | 9 | 9 |
| MAP-PROM | ADDR to OUT | 25 | 25 | 25 |
| 2910 | D to Y | 14 | – | – |
| PROM | ADDR to OUT | 30 | – | – |
| 2922 | SET-UP R | 5 | – | – |
| 2910 | SET-UP PC | – | 34 | – |
| 2910 | SET-UP R | – | – | 9 |
| TOTAL-ns | | 83 | 68 | 43 |

Figure 7. Propagation Delay Calculations on the Am2910 Microprogram Sequencer (Cont.).

f)

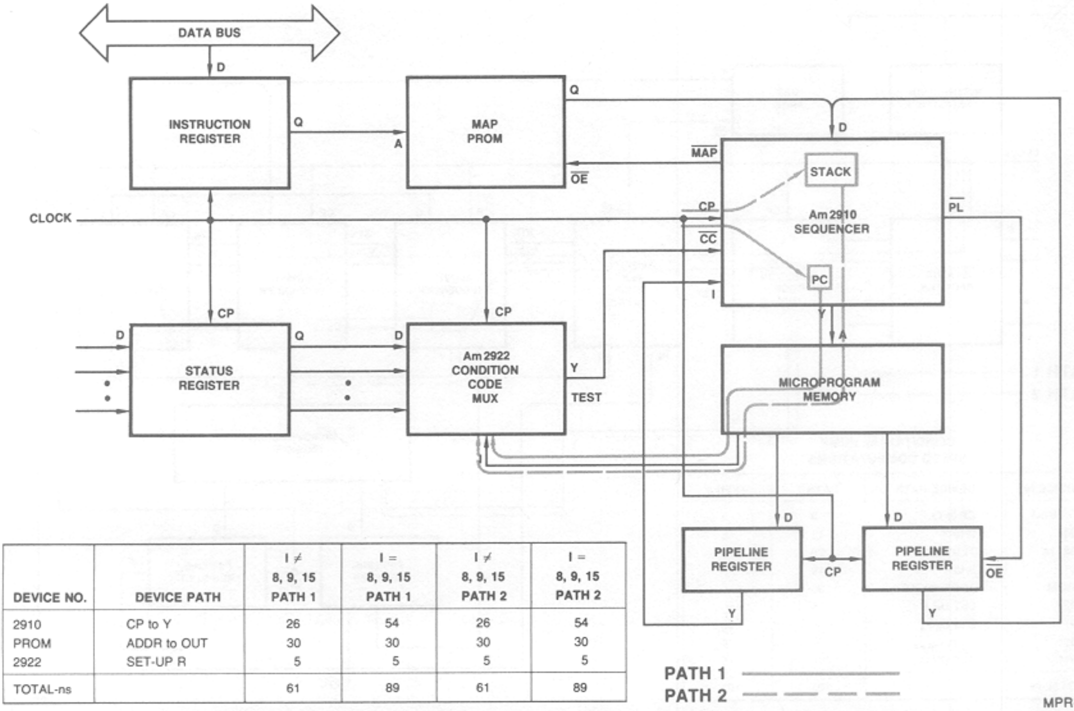


INSTRUCTION CONTROL SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| S – REG | CP → Q | 9 | 9 |
| 2910 | I to Y | 40 | – |
| PROM | ADDR TO OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2910 | SET-UP PC | – | 64 |
| TOTAL-ns | | 84 | 73 |

MPR-465

g)

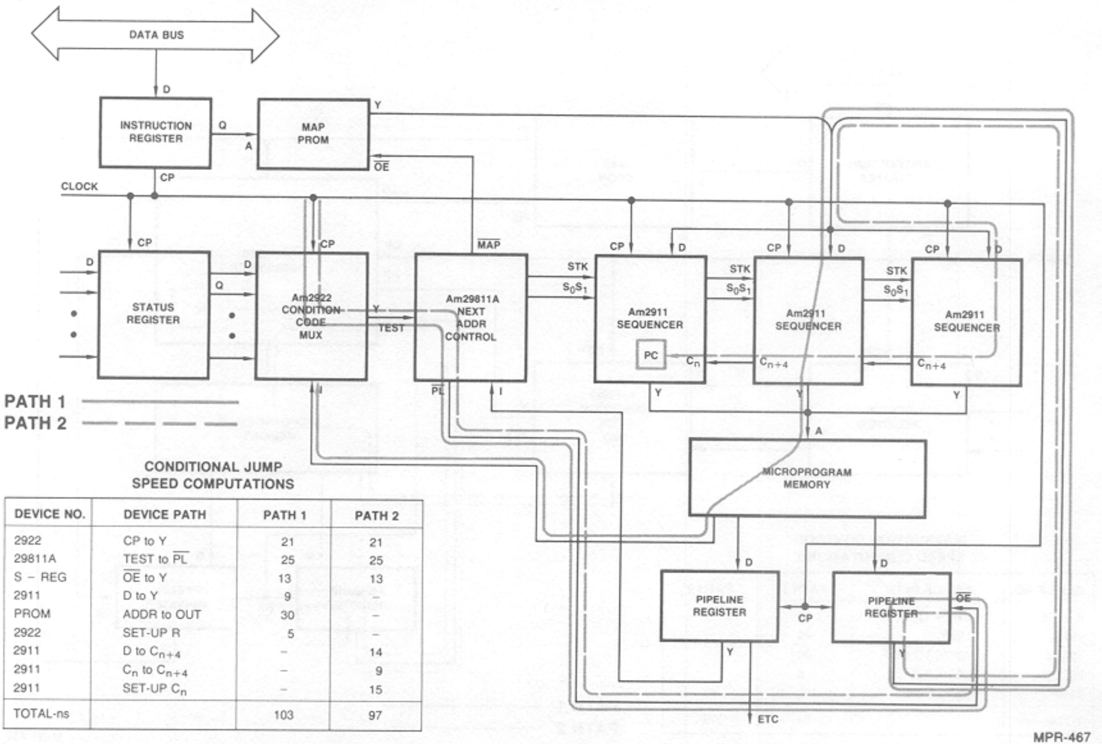


| DEVICE NO. | DEVICE PATH | I ≠ 8, 9, 15 PATH 1 | I = 8, 9, 15 PATH 1 | I ≠ 8, 9, 15 PATH 2 | I = 8, 9, 15 PATH 2 |
|---|---|---|---|---|---|
| 2910 | CP to Y | 26 | 54 | 26 | 54 |
| PROM | ADDR to OUT | 30 | 30 | 30 | 30 |
| 2922 | SET-UP R | 5 | 5 | 5 | 5 |
| TOTAL-ns | | 61 | 89 | 61 | 89 |

MPR-466

Figure 7. Propagation Delay Calculations on the Am2910 Microprogram Sequencer (Cont.).

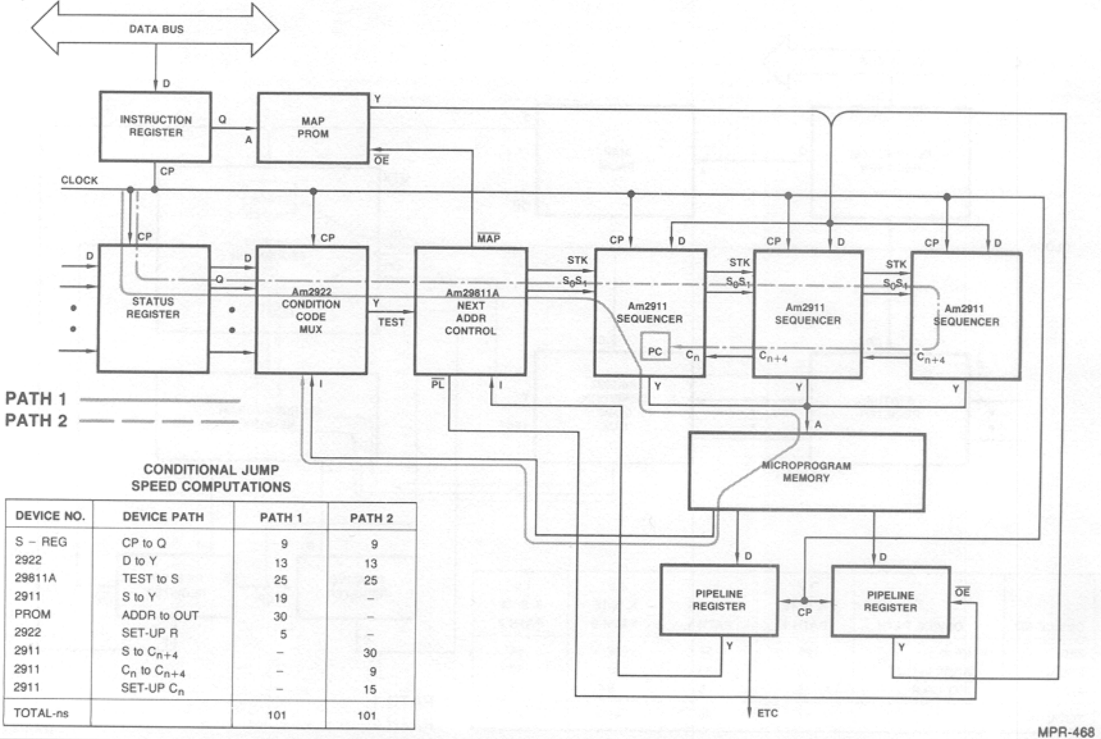
a)



CONDITIONAL JUMP
SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 2922 | CP to Y | 21 | 21 |
| 29811A | TEST to $\overline{PL}$ | 25 | 25 |
| S – REG | $\overline{OE}$ to Y | 13 | 13 |
| 2911 | D to Y | 9 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2911 | D to $C_{n+4}$ | – | 14 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 |
| 2911 | SET-UP $C_n$ | – | 15 |
| TOTAL-ns | | 103 | 97 |

b)

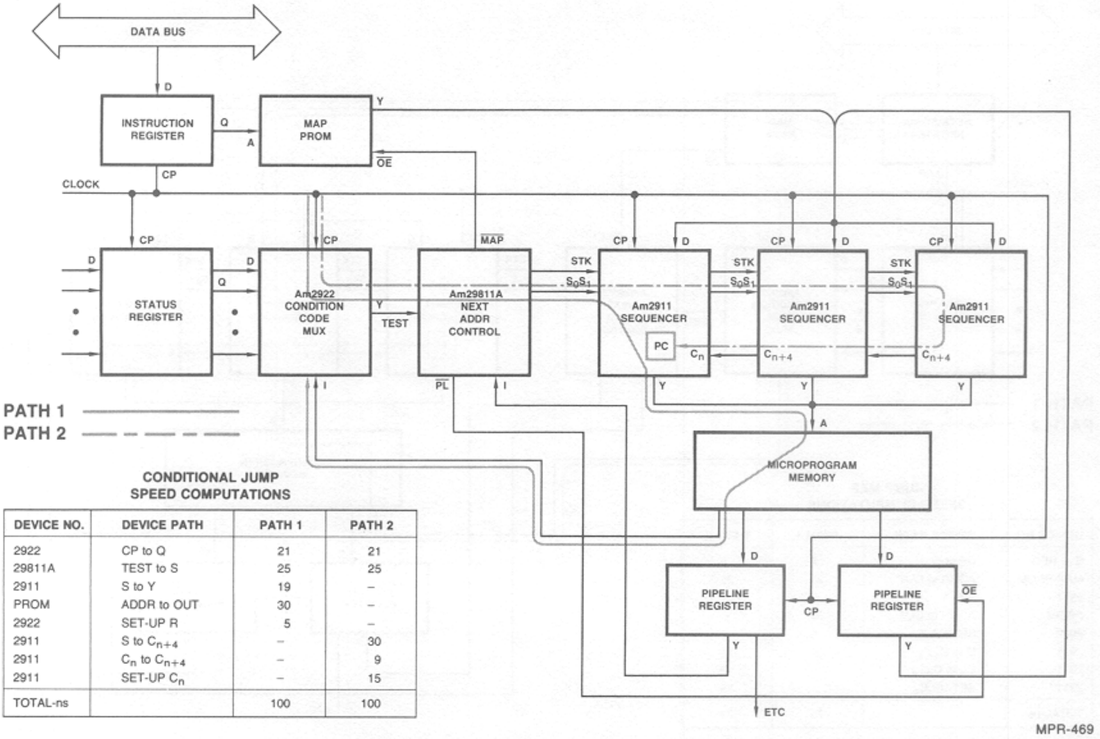


CONDITIONAL JUMP
SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| S – REG | CP to Q | 9 | 9 |
| 2922 | D to Y | 13 | 13 |
| 29811A | TEST to S | 25 | 25 |
| 2911 | S to Y | 19 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2911 | S to $C_{n+4}$ | – | 30 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 |
| 2911 | SET-UP $C_n$ | – | 15 |
| TOTAL-ns | | 101 | 101 |

Figure 8. Propagation Delay Calculations for the Am2911 and Am29811A Design.

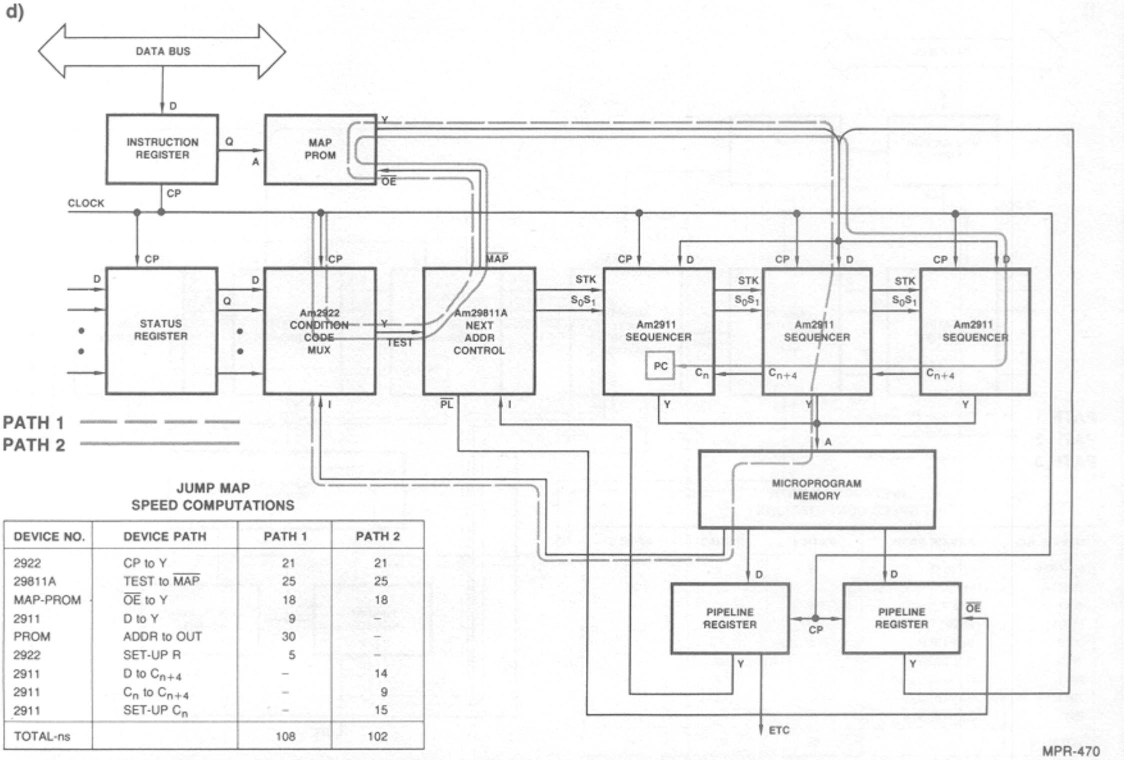
c)



**CONDITIONAL JUMP SPEED COMPUTATIONS**

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 2922 | CP to Q | 21 | 21 |
| 29811A | TEST to S | 25 | 25 |
| 2911 | S to Y | 19 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2911 | S to $C_{n+4}$ | – | 30 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 |
| 2911 | SET-UP $C_n$ | – | 15 |
| TOTAL-ns | | 100 | 100 |

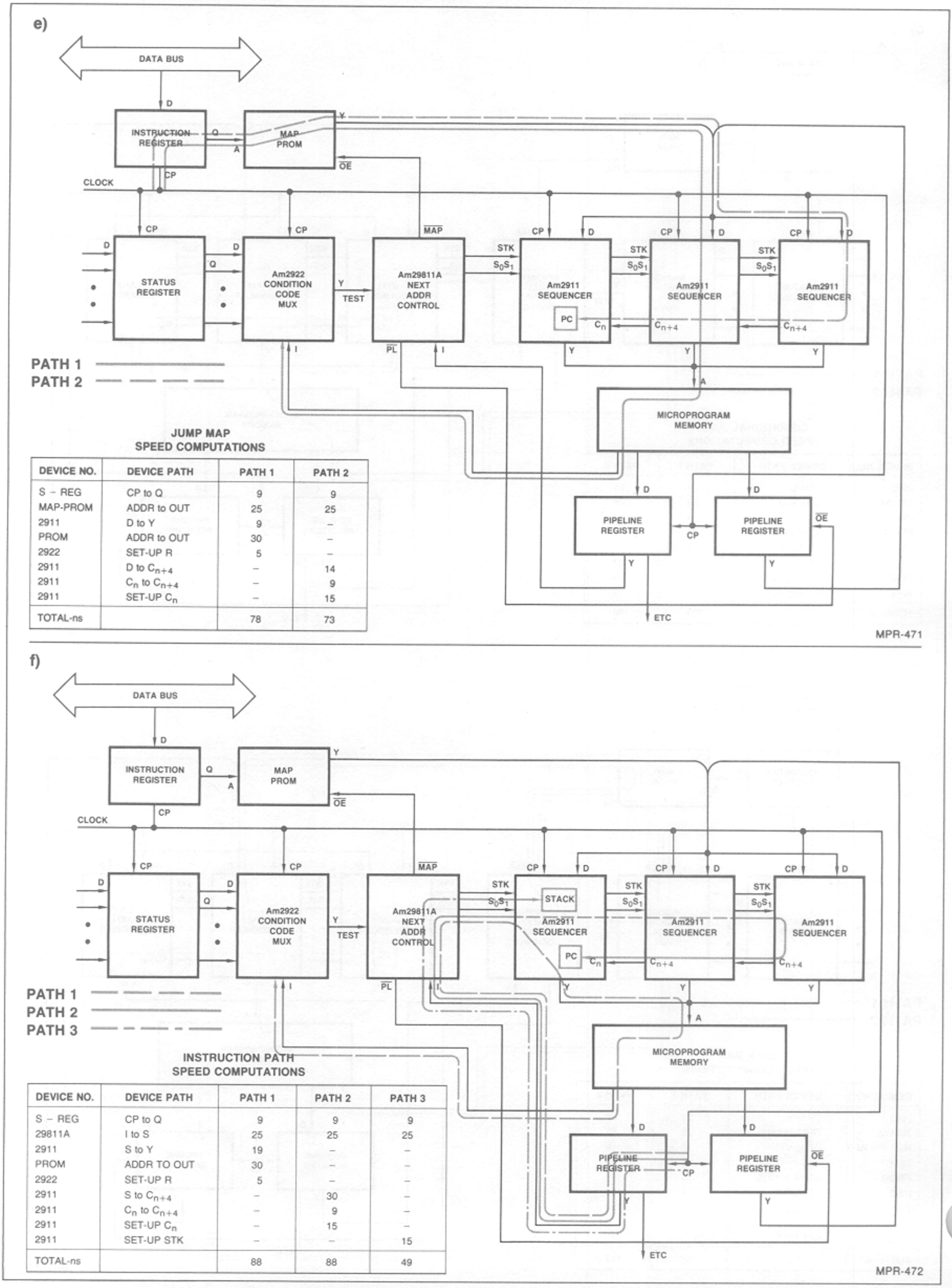
d)



**JUMP MAP SPEED COMPUTATIONS**

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 2922 | CP to Y | 21 | 21 |
| 29811A | TEST to $\overline{MAP}$ | 25 | 25 |
| MAP-PROM | $\overline{OE}$ to Y | 18 | 18 |
| 2911 | D to Y | 9 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2911 | D to $C_{n+4}$ | – | 14 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 |
| 2911 | SET-UP $C_n$ | – | 15 |
| TOTAL-ns | | 108 | 102 |

**Figure 8. Propagation Delay Calculations for the Am2911 and Am29811A Design (Cont.).**



JUMP MAP
SPEED COMPUTATIONS

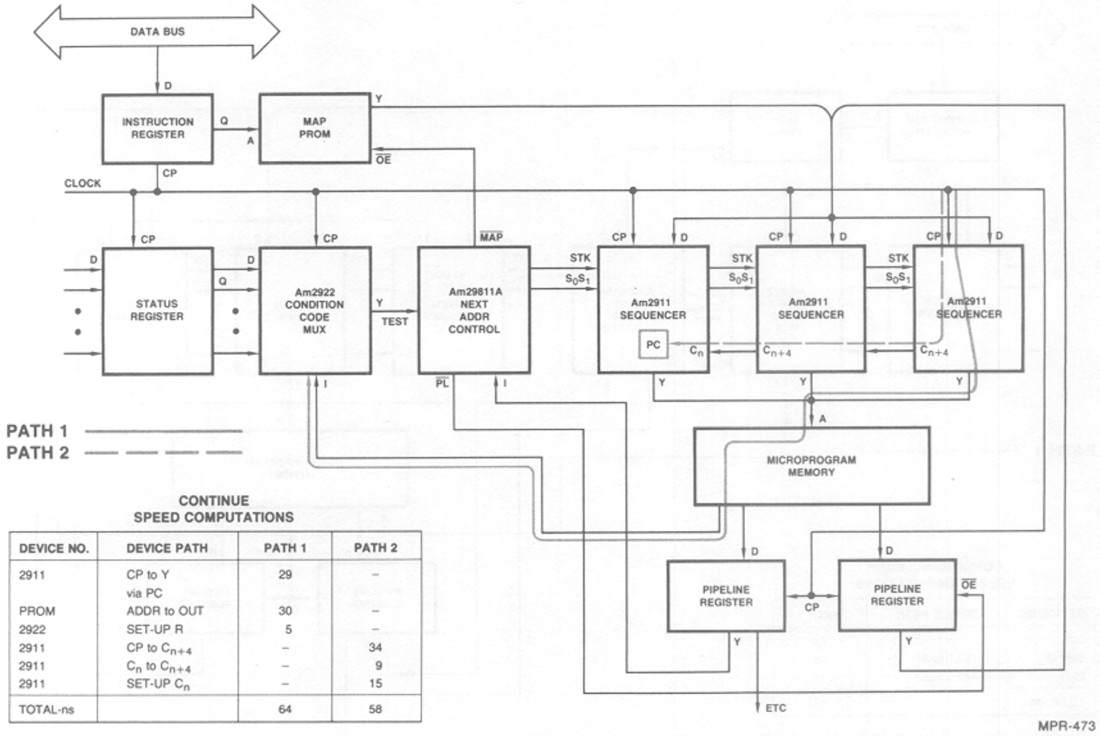
| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| S – REG | CP to Q | 9 | 9 |
| MAP-PROM | ADDR to OUT | 25 | 25 |
| 2911 | D to Y | 9 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2911 | D to $C_{n+4}$ | – | 14 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 |
| 2911 | SET-UP $C_n$ | – | 15 |
| TOTAL-ns | | 78 | 73 |

INSTRUCTION PATH
SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| S – REG | CP to Q | 9 | 9 | 9 |
| 29811A | I to S | 25 | 25 | 25 |
| 2911 | S to Y | 19 | – | – |
| PROM | ADDR TO OUT | 30 | – | – |
| 2922 | SET-UP R | 5 | – | – |
| 2911 | S to $C_{n+4}$ | – | 30 | – |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 | – |
| 2911 | SET-UP $C_n$ | – | 15 | – |
| 2911 | SET-UP STK | – | – | 15 |
| TOTAL-ns | | 88 | 88 | 49 |

Figure 8. Propagation Delay Calculations for the Am2911 and Am29811A Design (Cont.).

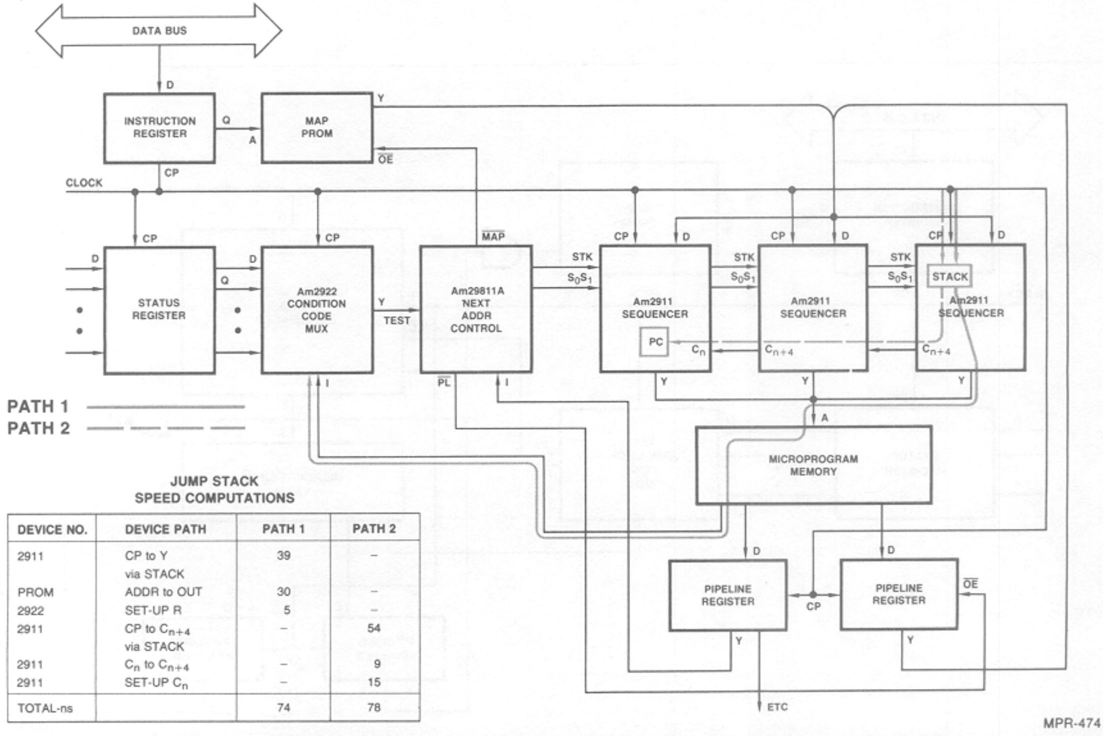
g)



CONTINUE
SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 2911 | CP to Y<br>via PC | 29 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2911 | CP to $C_{n+4}$ | – | 34 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 |
| 2911 | SET-UP $C_n$ | – | 15 |
| TOTAL-ns | | 64 | 58 |

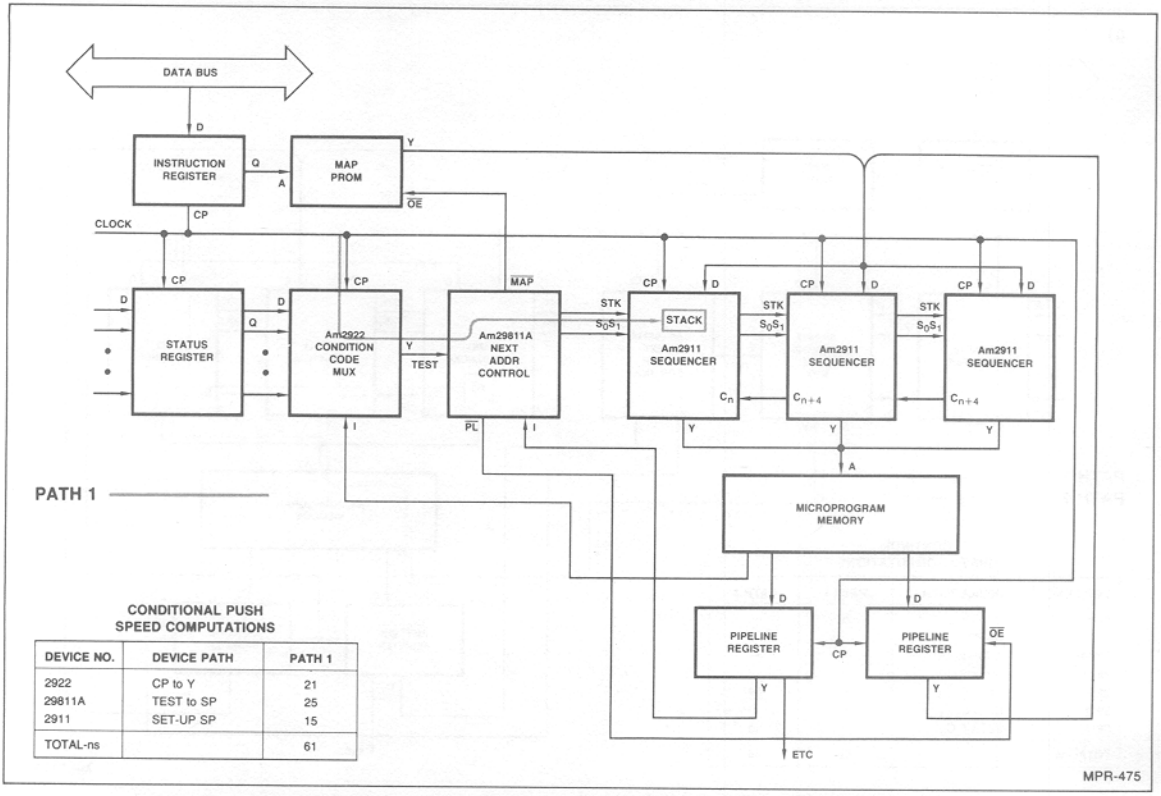
h)



JUMP STACK
SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 2911 | CP to Y<br>via STACK | 39 | – |
| PROM | ADDR to OUT | 30 | – |
| 2922 | SET-UP R | 5 | – |
| 2911 | CP to $C_{n+4}$<br>via STACK | – | 54 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 |
| 2911 | SET-UP $C_n$ | – | 15 |
| TOTAL-ns | | 74 | 78 |

**Figure 8. Propagation Delay Calculations for the Am2911 and Am29811A Design (Cont.).**



CONDITIONAL PUSH
SPEED COMPUTATIONS

| DEVICE NO. | DEVICE PATH | PATH 1 |
|---|---|---|
| 2922<br>29811A<br>2911 | CP to Y<br>TEST to SP<br>SET-UP SP | 21<br>25<br>15 |
| TOTAL-ns | | 61 |

Figure 8. Propagation Delay Calculations for the Am2911 and Am29811A Design (Cont.).

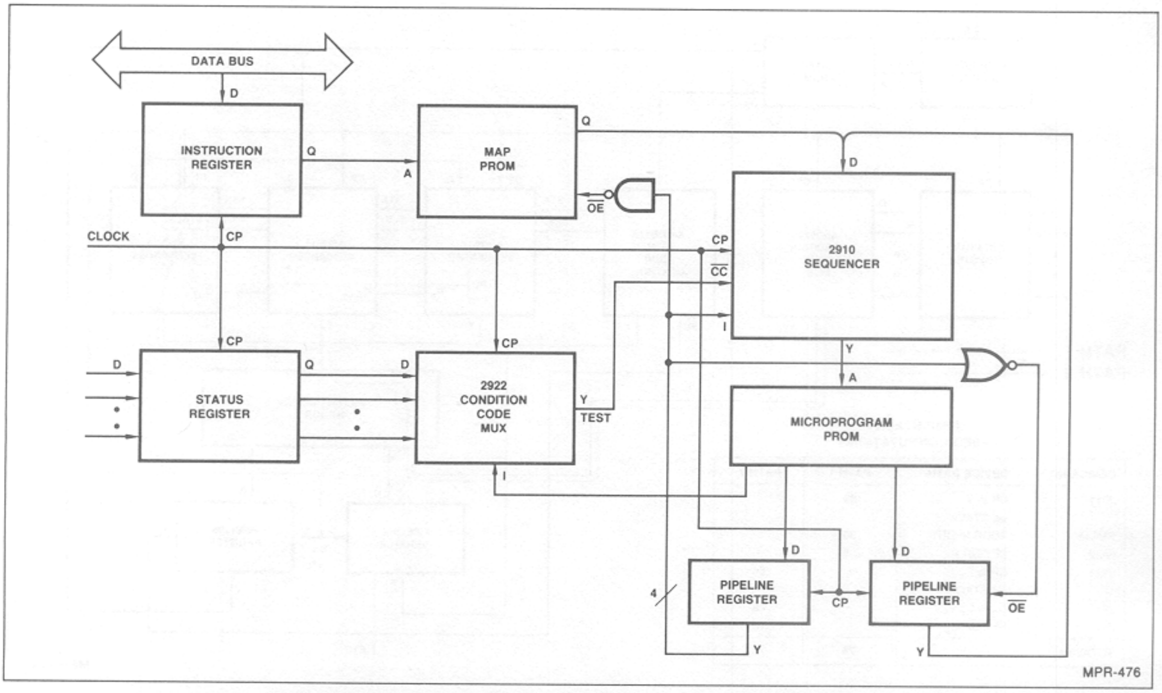


Figure 9. Using NAND and NOR Gates to Improve Am2910 Speed.

In order to compare the performance of the Am2910 with the Am2911 and Am29811A, Table 5 is presented. Here the propagation delays for the Am2911 and Am29811A are for a 12-bit wide microprogram sequencer configuration. If a wider configuration is used, only one additional carry input to carry output delay must be added to the appropriate paths of these calculations. A 12-bit wide Am2911/29811A configuration has been evaluated so that an "apples to apples" comparison can be made.

As is shown in Table 5, a number of combinations are possible for the longest AC propagation delay paths for these microprogram sequencers. First, the continue instruction can be executed the fastest of any of the microprogram instructions if the continues are sequential. That is, from the second continue on, the typical microcycle can be either 61 or 64ns respectively. To achieve this speed, it is required that various signals throughout the architecture be stable such that the only paths that enter into the propagation delay calculation are the clock-to-output of the microprogram counter, the microprogram memory and the pipeline register setup.

The second group of instructions shown in Table 5 show some examples of instruction execution and jumping. These examples assume that the $\overline{\text{MAP}}$ and $\overline{\text{OE}}$ outputs are not used as described earlier. These calculations apply to several of the instructions but not to all the instructions. For the Am2910 sequencer all of the propagation delays are around 80 to 85ns; while for the Am2911/Am29811A combination, the propagation delays range from about 80ns to 100ns, depending on the instruction. It should be noted that certain other instructions such as push and conditional load counter should be evaluated to determine the speed at which they can be executed.

The last two instructions shown in Table 5 are for jumps where the output enable of the field supplying the address to the D inputs of the microprogram sequencers are controlled by either the Am2910 or Am29811A. Notice that for Am2910 configuration, the jump map represents the longest propagation delay path and is 103ns typical. Also, for the Am2911/Am29811A combination, the jump map instruction also represents the longest propagation delay path and is 109ns typical.

It is not the purpose of this exercise to show every possible propagation delay path; but rather, to show the reader the technique for computing propagation delays such that any design can be evaluated and the worst case past derived. Even here, not all of the worst case numbers shown in Table 5 have been derived in Figures 7 and 8. This was done intentionally and is left as an exercise for the student.

If the Am2909 or Am2911 and the Am29811A are combined into microprogram sequencers of either 8 bits in width or 16 bits in width, the calculations need only be modified slightly to determine the microcycle times. Obviously, if two Am2911s are used, the worst case propagation delay paths do not change. However, if four Am2911s are used, the carry path will become the longer propagation delay path on several of the computations. This may be offset however since larger microprogram PROMs may be used if 64K of microcode is actually being addressed or high power buffers may be placed between the Am2911 outputs and the microprogram memory to provide sufficient drive for such a large microprogram store.

In addition, the Am2909 and Am2911 may be used without the Am29811A where the user wishes to generate a special purpose instruction set or very high speed control of the internal multiplexer and push pop stack. In some, designs as much as 25 to 30ns, typical, can be removed from the longest propagation delay paths of the design by using high speed Schottky SSI. While this has not been the typical case, some designers have used it to provide a performance improvement not achievable with a standard Schottky condition code multiplexer and the Am29811A next address control unit.

## APPLICATIONS

It should be understood that the microprogram state machine built using either the Am2910 or the Am2911/29811A represents a general purpose state machine controller. Applications for this type of microprogrammed control include uses in minicomputers, communications, instrumentation, controllers and peripherals as well as special purpose processors. Typically, the microprogrammed approach provides a more structured organization to the design and allows the design engineer the greatest flexibility in implementation.

It is important to understand that microprogrammed machines need not be part of a typical minicomputer type structure. That is, a general purpose minicomputer usually has a machine instruction set that is totally different from its microprogram instruction control. As such, it is essential that the designer new to computer design and microprogram design understand the difference between a machine instruction and a microprogram instruction. This differentiation is shown in Figure 10 where a typical 16-bit machine level instruction is demonstrated as compared with a typical microprogram instruction. The machine level instruction usually consists of 16 bits and in this example, these bits are used to provide the op code, source register definition and destination register definition. The microprogram instruction on the other hand usually consists of anywhere from 32 to 128 bits in a typical minicomputer type design. Here, the bits are used to control the elemental functions of a machine such as the Am2910 instruction control and condition code multiplexer, the Am2903 source, ALU function and destination control and so forth. For purposes of this explanation, let us assume that the machine level instruction is available to the machine programmer while the microprogram

**TABLE 5. SUMMARY OF LONGEST AC PATHS FOR MICROPROGRAM SEQUENCERS.**

| Instruction | Am2910 | Am2911<br>Am29811A | Comments |
|---|---|---|---|
| Continue | 61 | 64 | The fastest instruction.<br>Assumes sequential continues! |
| Instruction Execute | 84 | 88 | If the $\overline{\text{MAP}}$ and $\overline{\text{PL}}$ outputs are not used. |
| Jump Map (no $\overline{\text{OE}}$) | 83 | 78 | |
| Jump PL (No $\overline{\text{OE}}$) | 78 | 101 | |
| Jump Map (via $\overline{\text{OE}}$) | 103 | 109 | If the $\overline{\text{MAP}}$ and $\overline{\text{PL}}$ outputs are used. |
| Jump PL (via $\overline{\text{OE}}$) | 98 | 104 | |

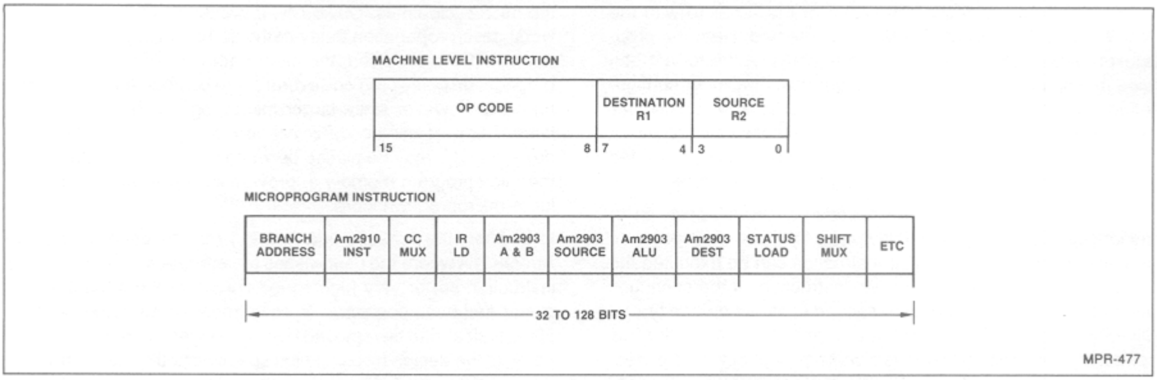


**Figure 10. Understanding Machine and Microprogram Instructions.**

instruction is not available to the machine programmer at the assembly language level. Let it suffice to say that this assumption is not necessarily valid in machines being designed today.

Perhaps one of the most typical applications of the microprogrammed computer control unit state machine design is as the controller for a minicomputer. Here, the function of the microprogrammed controller is to fetch and execute machine level instructions. The flow required to perform this function is depicted in Figure 11 which should be representative for all general purpose type machines. Figure 11 shows that after initialization, the computer control unit simply fetches machine instructions, decodes these instructions and then fetches the required operands such that the original instruction can be executed. This cycle of fetching and executing instructions is performed without end. Such things as hardware halts or resets are ignored and should be assumed to only cause re-initialization.

Once the flow of a typical computer control unit is understood, it is possible to evaluate a number of architectures using the Am2910 or Am2911/Am29811A such that the flow diagram of Figure 11 can be implemented.

## STATE MACHINE ARCHITECTURES

After a machine instruction is fetched from memory, it is normally placed in the machine instruction register as described in Figure 6. Then the op code portion of the instruction is decoded so that a sequence of microinstructions in the microprogram memory can be selected for execution. Each microinstruction is fetched and its contents placed in the pipeline register as shown in Figure 6 for execution.

While the architecture of Figure 6 is recommended and has been used throughout the preceding portion of this chapter, it should be understood that a number of architectures are possible using these microprogram sequencers. The normal flow in fetching microinstructions is to determine the address of the next microinstruction, fetch the contents at that address and set up this data at the input of the pipeline register such that it can be clocked into the pipeline register for execution. If we assume that a clock is being used to clock the pipeline register, the Am2910, the machine instruction register and the Am2903 microprocessor bit slices, it is possible to define a number of computer control unit designs where the relationship between the clock edges is different.

There seem to be a minimum of seven different architectures that can be defined based on placing registers in the appropriate signal paths and storing data on the low-to-high transition of the clock. For purposes of this discussion, we will assume that all clocked devices will operate using the same clock such that changes will occur on the LOW-to-HIGH transition of the clock. While it is possible to use multiphase clocks and tie different clock phases to different devices, that type of system operation will not be described here. In all cases, we will be talking about the flow of signals between LOW-to-HIGH transitions of the clock. Typically, a cycle is started by a clock edge at a device and the signals begin to flow from one device to the next until a set-up time to a clock edge results. Then, the next microinstruction is executed in

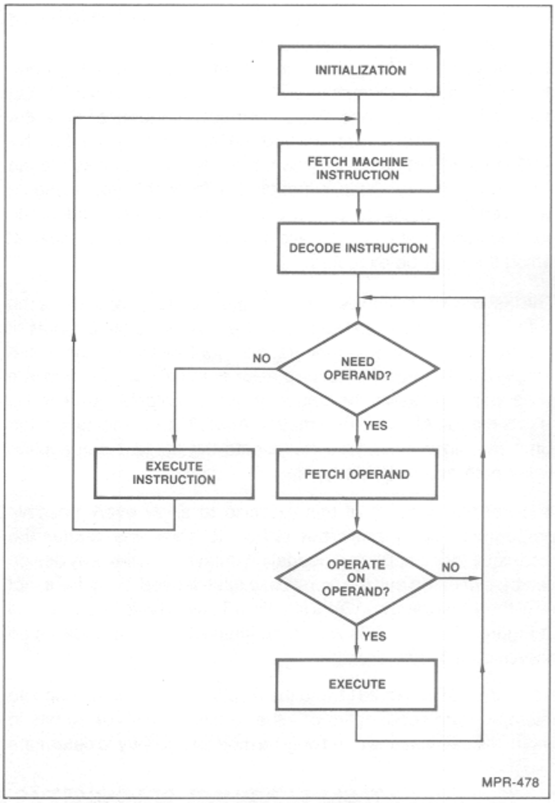
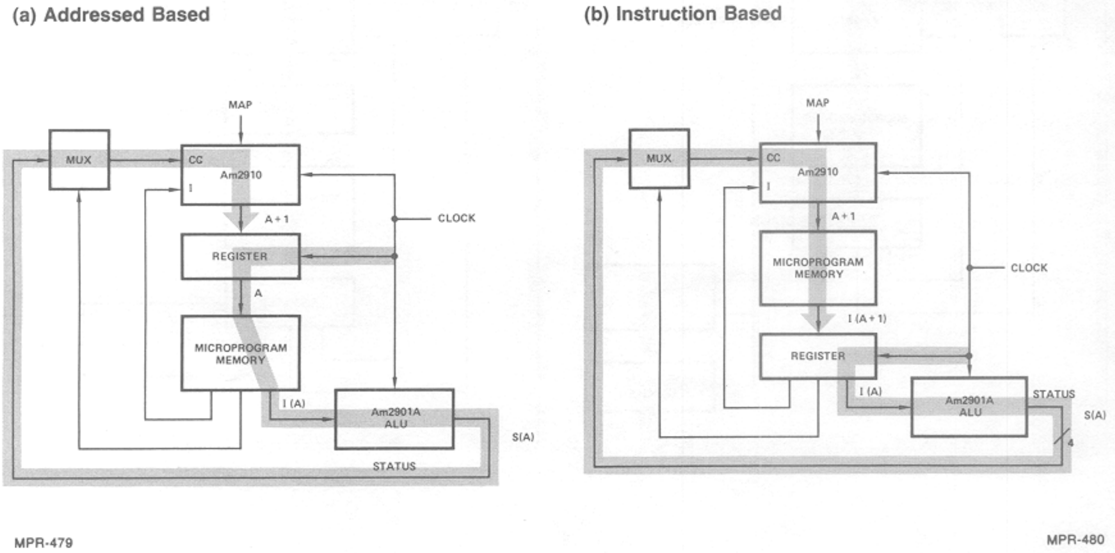


**Figure 11. Computer Control Flow Diagram.**

exactly the same manner. There are three different identifiable types of microinstruction sequences where only one register is in the signal flow loop. The first of these we shall call an Address-Based microinstruction cycle. It usually starts with the address of a microprogram memory word being stored in a register by the clock. This address has been determined by the previous microinstruction. This address then accesses the microprogram memory to fetch its contents which are presented at its outputs to control the Arithmetic Logic Unit and the results of the Arithmetic Logic Unit function may be used to determine the next address selected that will be stored in this microprogram address register. This is shown as Figure 12a. The second type of microprogram architecture is called Instruction-Based. Here, the register is placed at the output of the microprogram memory as shown in Figure 12b. Again, the cycle consists of executing the microinstruction in the ALU; perhaps using the results of the operation to determine the address of the next microinstruction and then fetching the contents of that microinstruction and setting this new data up at the input to the register. The third basic architecture for microprogram control is called Data-Based. Here, a register is used to hold the status data from the ALU and this is the determining clock point for the cycle. Here, the status register initiates the selection of the next address from which the microprogrammed data is fetched and this microprogram instruction is used to execute a new function in the ALU thereby setting up the results for the status register. This scheme is shown in Figure 12c. Note that this scheme requires an additional register at the output of the microprogram memory to hold a portion of the microprogram instruction for controlling the condition code multiplexer and Am2910 instruction set. These primitive architectures for microprogrammed control demonstrate the three points at which a register can be placed to provide a start and an end for the microcycle. In a general sense, each of these three architectures is one level pipelined. This, however, is not the definition normally associated with pipelining of microprogram control.

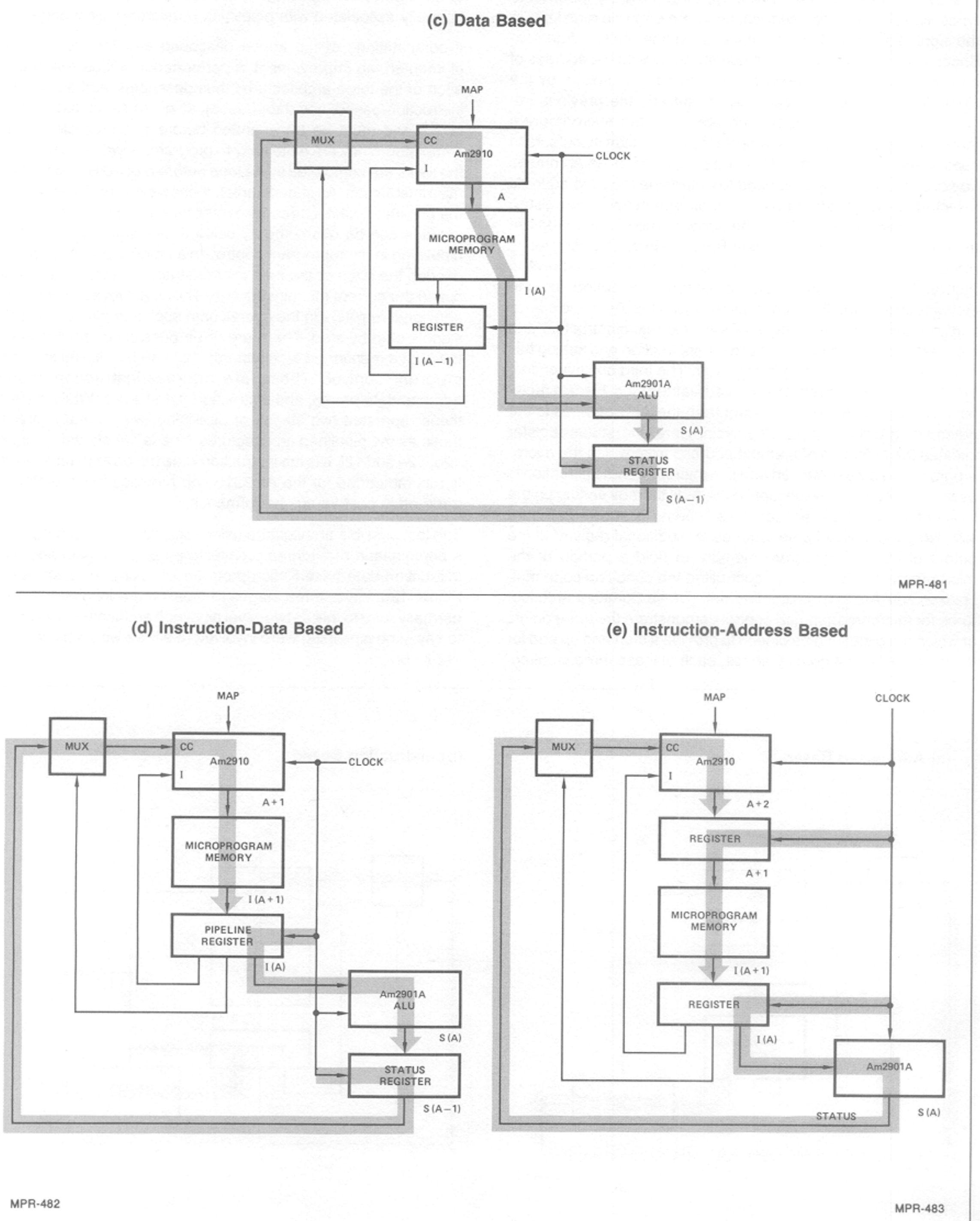
If combinations of the above described architectures are implemented, an improvement in performance will be realized. In each of the three architectures thus described (address-based, instruction-based, and data-based), all of the signal paths are in series and must be transcended before a microcycle can be completed. They are quite easy to program, however, since all of the tasks are completed in the loop before proceeding to the next microinstruction. As stated earlier, these tend to be the slowest of the possible architectures for microprogram control. This disadvantage can be overcome by using a technique referred to as pipelining in microprogram control. In a pipeline architecture, we overlap the fetch of the next microinstruction while we are executing the current microinstruction. This is achieved by inserting additional registers in the overall path such that we can hold the signals step-by-step. There are three possible combinations of the above mentioned architectures that can be utilized in microprogram control. These are address-instruction-based, address-data-based, and instruction-data-based. While each of these represent two stages of pipelining, we normally refer to these as the pipelined architectures. These are shown in Figure 12d, 12e and 12f. It is the instruction-data based architecture that is recommended for the Am2910 and provides the overall best trade-off in cost versus performance.

The last possible architecture using registers in the signal path is a combination of all three architectures and is called address-instruction-data-based microprogram control and is shown in Figure 12g. Here, three stages of pipeline are involved and we normally refer to this as two-level pipelined archiecture. Needless to say, if no pipelining were involved at all, we would have a ring oscillator.



Shaded Lines Show Required Signal Flow to Complete a Microcycle:
Determine Address, Fetch Instruction and Execute.

Figure 12. Standard Microprogram Control Architectures.



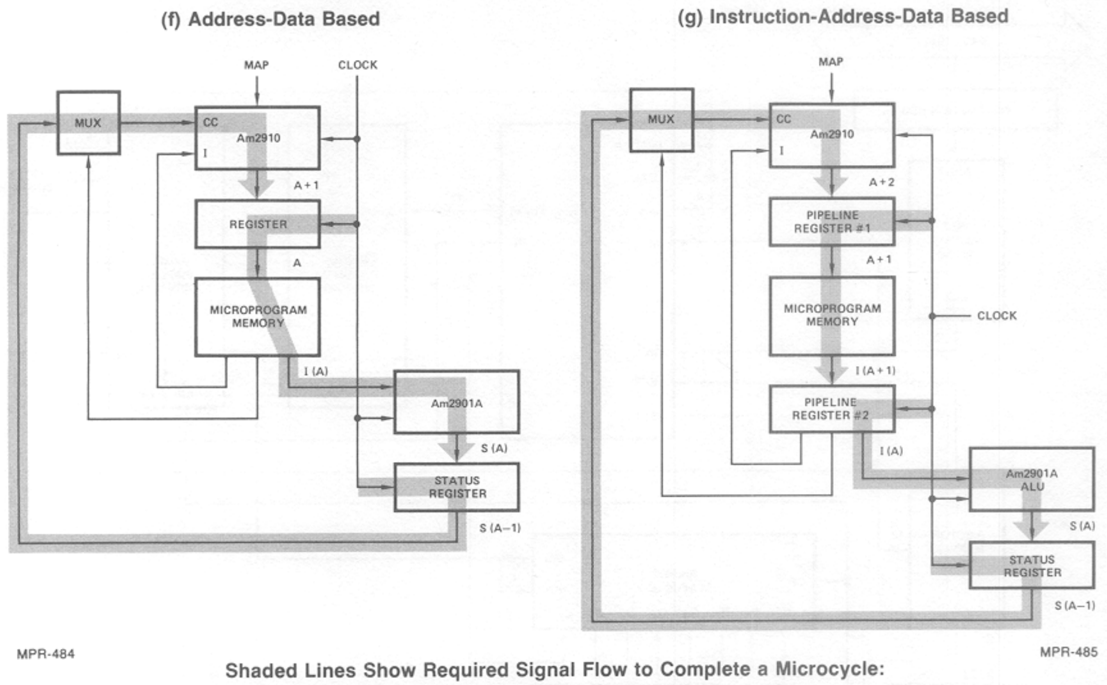
(c) Data Based

(d) Instruction-Data Based

(e) Instruction-Address Based

**Shaded Lines Show Required Signal Flow to Complete a Microcycle: Determine Address, Fetch Instruction and Execute.**

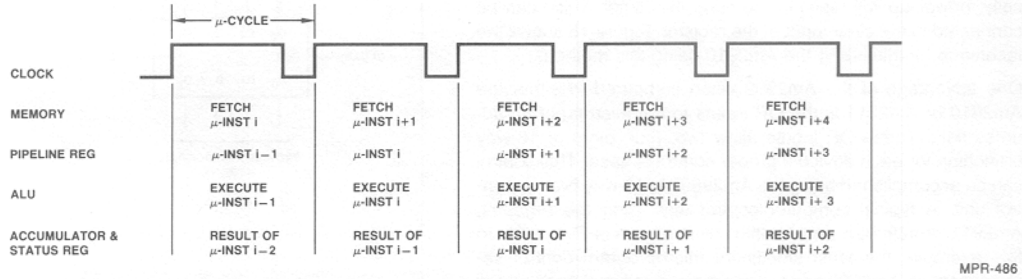
Figure 12. Standard Microprogram Control Architectures (Cont.).



**Shaded Lines Show Required Signal Flow to Complete a Microcycle: Determine Address, Fetch Instruction and Execute.**

Figure 12. Standard Microprogram Control Architectures (Cont.).

The advantage of the instruction-data-based architecture is that the address and contents of the next microinstruction are being fetched while the current microinstruction in the pipeline register (Figure 6) is being executed. This allows a shorter microcycle since the microprogram memory fetch and ALU execution can be operated in parallel. The results of this type operation are demonstrated in Figure 13 where we see a typical timing diagram of the microprogram execution of the address-data-based instruction architecture. It should be noted that when the computational aspects of a microinstruction are not completed in the same microcycle, they obviously cannot be used to determine the address of another microcycle until the computation has been completed and stored in the status register. Thus, this pipelined architecture offers significant speed improvement except in the case of certain conditional jumps. In other words, the conditional jump may not use the status register information of the immediately preceding microinstruction because the computation is just being performed. For this architecture, the conditional jump fetch must be executed on the cycle after the status register contains the proper execution results. This can be seen by studying Figure 13. In most microprogram designs this is not a disadvantage because other housekeeping and ALU operations can be performed while the address of the next microinstruction is being determined using the current contents of the status register. While it is not directly pertinent to the discussion at this time, let us point out that the Am2904 has been designed such that the machine architect can utilize both instruction-data-based architecture as well as instruction-based architecture if no housekeeping is required. Thus, the Am2910 and Am2904 can be used in a variable architecture cycle to achieve maximum performance for the machine.



| | | | | | |
|---|---|---|---|---|---|
| MEMORY | FETCH<br>μ-INST i | FETCH<br>μ-INST i+1 | FETCH<br>μ-INST i+2 | FETCH<br>μ-INST i+3 | FETCH<br>μ-INST i+4 |
| PIPELINE REG | μ-INST i−1 | μ-INST i | μ-INST i+1 | μ-INST i+2 | μ-INST i+3 |
| ALU | EXECUTE<br>μ-INST i−1 | EXECUTE<br>μ-INST i | EXECUTE<br>μ-INST i+1 | EXECUTE<br>μ-INST i+2 | EXECUTE<br>μ-INST i+ 3 |
| ACCUMULATOR & STATUS REG | RESULT OF<br>μ-INST i−2 | RESULT OF<br>μ-INST i−1 | RESULT OF<br>μ-INST i | RESULT OF<br>μ-INST i+ 1 | RESULT OF<br>μ-INST i+2 |

Figure 13. Timing Diagram of Microprogram Execution.

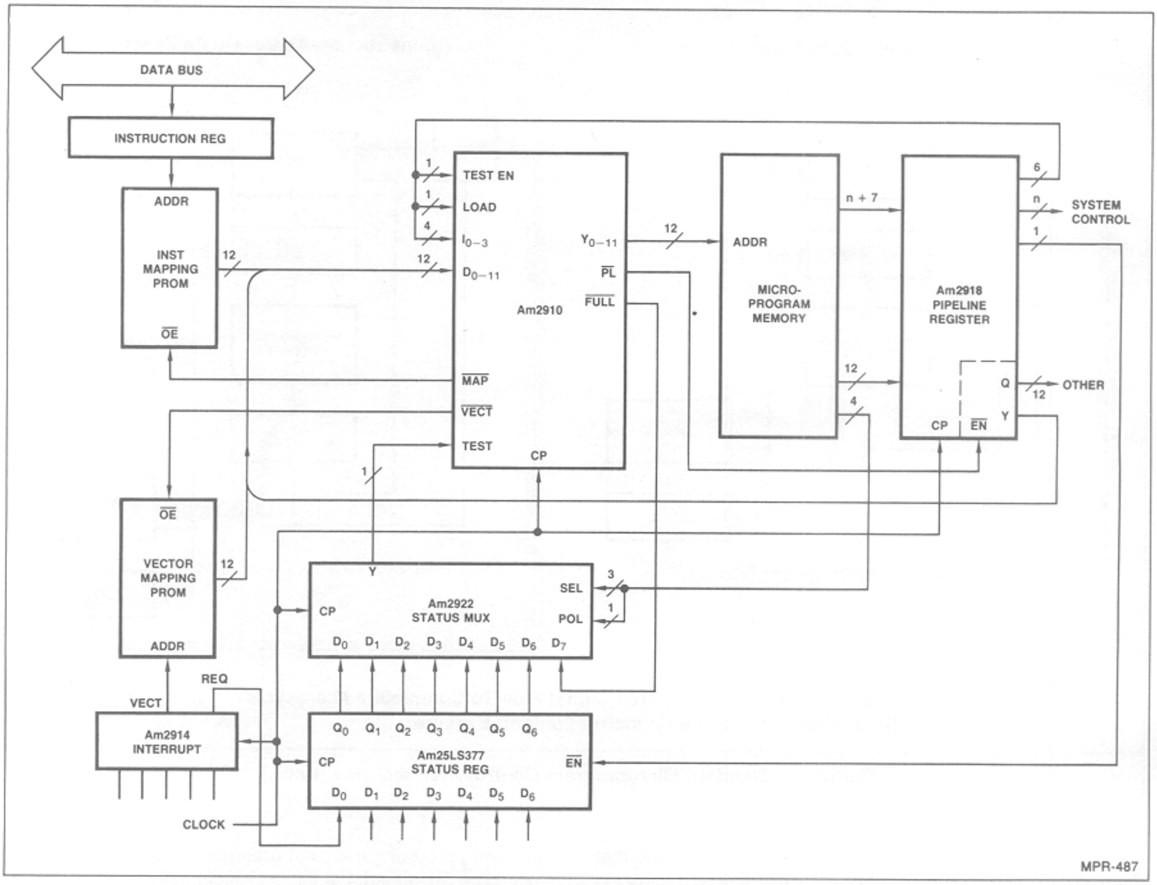


Figure 14. Typical Am2910 Microprogram Control Unit.

## The Am2910 in Computer Control

A general state machine design using the Am2910 is shown in Figure 14. Here, all three output enables are used to advantage in order to control the mapping PROM, pipeline register and vector PROM in this design. This design is very straightforward and in fact is identical to that shown earlier.

One area that should not be overlooked is that of initializing the Am2910 at power up. One technique for accomplishing this is to use a pipeline register with a clear input to provide all LOWs to the instruction inputs of the Am2910. This will cause a reset of the stack in the Am2910 and force the outputs to the zero word and microcode which can be used for the initialization routine. Typically, power up will result in the firing of a timer which can be connected to the clear input of the register. Figure 15 shows the technique for initializing the Am2910 using this method.

One advantage of the Am2909 when compared to either the Am2910 or Am2911 is the OR inputs to the microprogram address field. These OR inputs allow two, four, eight or 16-way branching for each device if proper control is used. This control can be accomplished using the Am29803A, 16-way branch control unit. A typical computer control unit using the Am2909, Am2911, Am29803A and Am29811A is shown in Figure 16. In this example, the least significant microprogram control sequencer is an Am2909 and the two more significant sequencers are Am2911s.

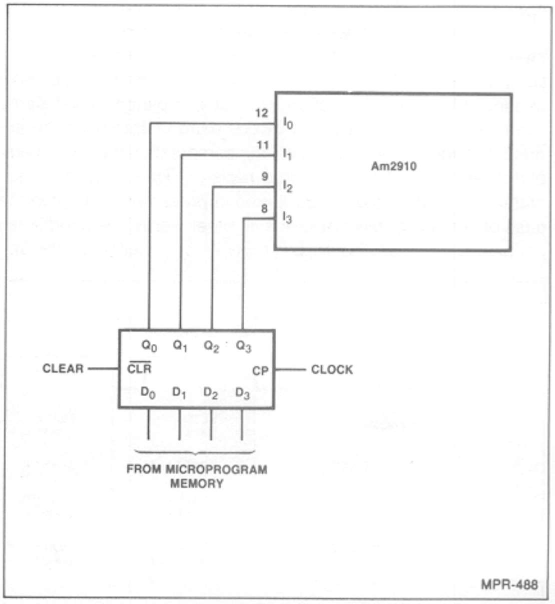


Figure 15. Initializing the Am2910.

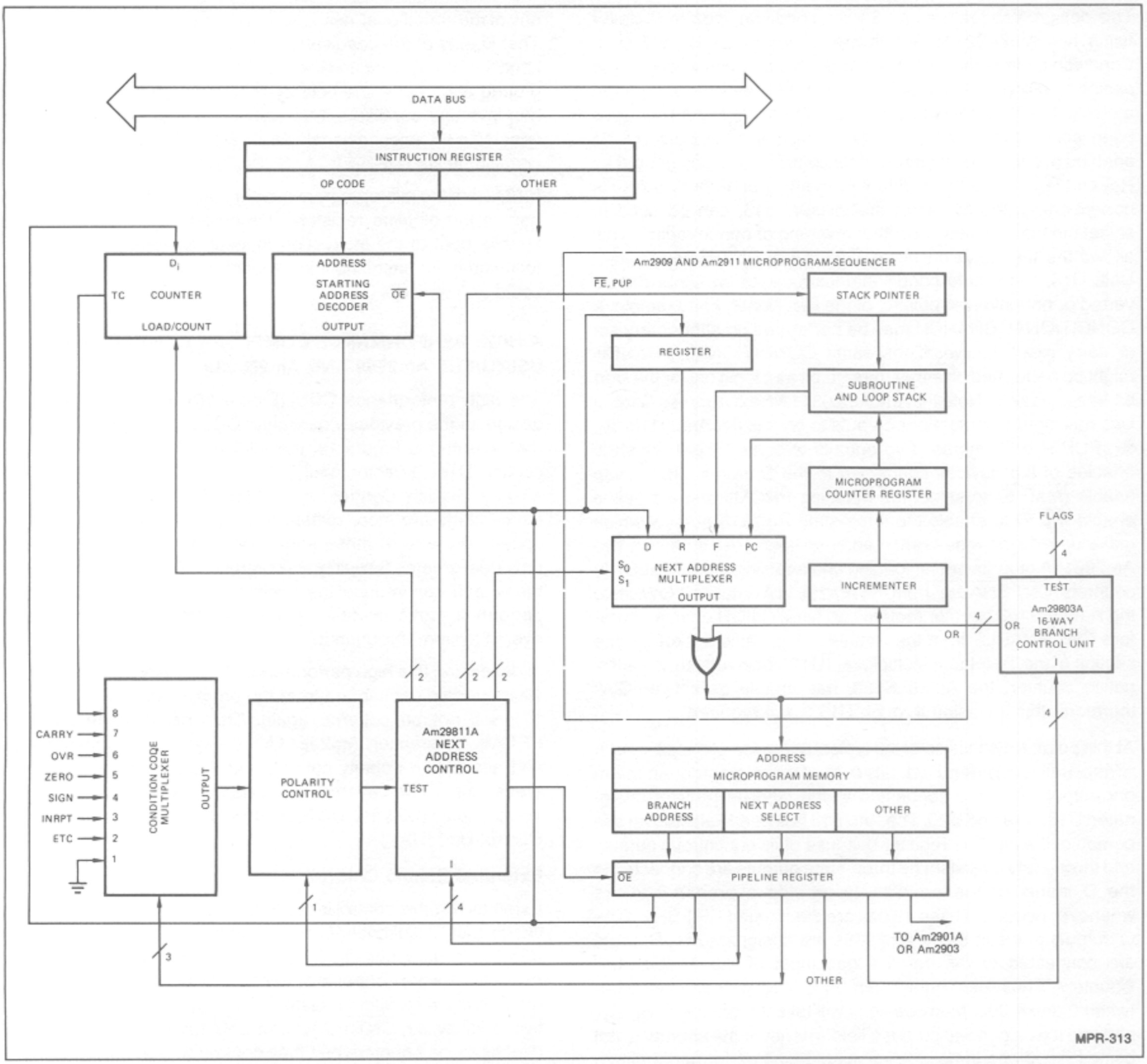


**Figure 16. A High Performance Microprogram Controller Using the Am2909, Am29811A and Am29803A.**

## DETAILED DESCRIPTION OF THE Am2911 AND Am29811A IN A COMPUTER CONTROL UNIT

The detailed connection diagram of a straight-forward computer control unit is shown in Figure 17. This design features all of the next address control functions described previously and a few features have also been added.

Referring to Figure 17, the instruction register consists of two Am25LS377 Eight-Bit Registers with Clock Enable. These registers are designated as U1 and U2 and provide ability to selectively load a 16-bit instruction. This particular design assumes that the instruction word consists of an eight-bit op code as well as eight bits of other data. Therefore, the op code is decoded using three 256-word by 4-bit PROMs. The Am29761 has been selected for this function and is shown in Figure 17 as U3, U4 and U5.

The basic control function for the microprogram memory is provided by the Am2911s. In this design, three Am2911s (U6, U7, and U8) are used so that up to 4K words of microprogram memory can be addressed. The microprogram memory can consist of PROMs, ROMs, or RAMs, depending on the particular design and the point of its development. This particular design shows the capability of a 64-bit microword; however, the actual number of bits used will vary from design to design.

The pipeline register associated with the computer control unit consists of five integrated circuits designated U16, U17, U18, U19 and U20.

One of the features of the architecture depicted in Figure 17 is the event counter shown as U9, U10 and U11. This event counter consists of three Am25LS163s connected as a 12-bit counter. The counter can be parallel loaded with a 12-bit word from pipeline registers U18, U19 and U20. The multiplexer and D-type flip-flop (U21 and U22) at the counter overflow output (U9) is present to improve system cycle time and will be described in detail later.

Note: Figures 17, 18, 20, and 24 are at back of the book.

This design also features a 16-input condition code multiplexer using two Am74S251s, which are designated U12 and U14. Condition code polarity control capability has been added to the design by using an Am74S158 Two-Input Multiplexer designated as U13. The W outputs and Y outputs from U12 and U14 have been connected together but only one set of outputs will be enabled at a time via the three-state control signal designated as $R_{20}$ and $\overline{R_{20}}$. Since the Y output is inverting and the W output is non-inverting, the two-input multiplexer, U13, can be used to select the test condition as either inverting or non-inverting. This allows the test input on the Am29811A Next Address Control Unit, U15, to execute conditional instructions on either the inverted or non-inverted polarity of the test signal. For example, a CONDITIONAL BRANCH may be performed on either carry set or carry reset. Likewise, the same CONDITIONAL BRANCH might be performed on either the *sign* bit as a logic one or the *sign* bit as a logic zero. Note that the Am29811A Next Address Control Unit has eight outputs. Four outputs to control the Am2911's $S_0$, $S_1$, PUP and $\overline{\text{FE}}$ inputs. Two outputs to control the three-state enables of the devices connected to the D inputs, i.e., a map enable ($\overline{\text{MAP E}}$) to select the mapping PROMs and a pipeline enable ($\overline{\text{PL E}}$) to enable the three-state Am2918 outputs which make up a 12-bit wide branch address field. The remaining two Am29811A outputs are for loading and enabling the Am25LS163 counters. $\overline{\text{CNT ENABLE}}$ from the Am29811A is active-LOW while the Am25LS163 counter requires an active-HIGH enable, therefore $\overline{\text{CNT ENABLE}}$ from the Am29811A is passed through one section of the Two-Input Multiplexer (U13) for inversion. An alternative counter, the Am25LS169, has enable as active-LOW; therefore, this inversion through U13 is not required.

At this point, a discussion of the typical operation of this computer control unit is in order. First, bits 0-11 of the microprogram memory output word, are connected to the pipeline register designated U18, U19 and U20. The Am2918 has been selected for this portion of the pipeline register because of its continuous outputs and three-state outputs. The three-state outputs are connected to the D inputs of the Am2911 to provide a branch address whenever needed. These 12 bits are designated $BR_0$-$BR_{11}$. The Q outputs of these same Am2918s are designated $R_0$-$R_{11}$ and are connected to the parallel load input of the Am25LS163 Counters. Thus, the counter can be loaded with any value between 0 and 4,095. Many designs will take advantage of $R_0$-$R_{11}$ and use it as a general purpose field whenever the counter is not being loaded or a jump pipeline is not being performed. Using a microprogram memory field for more than one function (branch address and counter load value in this example) is called FORMATTING and will be covered in greater detail later. The other two devices in the pipeline register shown on the architecture of Figure 17 are U16 and U17. First, U17 receives four bits (12, 13, 14 and 15) from the microprogram memory to provide four-bit instruction field to the Am29811A. This four-bit field, designated $R_{12}$-$R_{15}$, provides the actual next address control instruction for the computer control unit. $R_{16}$ is the polarity control bit for the test input and is connected to the select input of the Am74S158 Two-Input Multiplexer. When $R_{16}$ is LOW, the signal at the Am29811A test input will be inverted, but when $R_{16}$ is HIGH, the test input will be non-inverted.

The Am74S175 has been used as part of the pipeline register (U16) because it has both inverting and non-inverting outputs. Signals $R_{17}$, $R_{18}$ and $R_{19}$ are used to control the One-of-Eight Multiplexer (U12 and U14) A, B and C inputs. Pipeline register output $R_{20}$ and $\overline{R_{20}}$ are used to enable either the U12 outputs or the U14 outputs such that a one-of-sixteen multiplexer function is implemented. In this design, the TEST 0 input of U14 is connected to ground. This provides a convenient path for converting any of the conditional instructions to non-conditional instructions. That is, any of the conditional instructions can be executed unconditionally by selecting the TEST 0 input which is connected to ground and forcing the polarity control to either the inverting or non-inverting condition. This allows the execution of unconditional JUMP, unconditional JUMP-TO-SUBROUTINE, and unconditional RETURN-FROM-SUBROUTINE instructions.

Bit 21 from the microprogram memory utilizes a flip-flop in U17 as part of the pipeline register. This output, $R_{21}$, is used as the enable input to the instruction register. Needless to say, other techniques for encoding this enable function in a formatted field could be provided.

## A HIGH PERFORMANCE COMPUTER CONTROL UNIT USING THE Am2909 AND Am29803A

The high performance CCU (Figure 18) is of a similar basic design as the previously described CCU. The major differences are, referring to Figure 18, the addition of an extended enable control (U16), a vector input (U24 and U25), and an Am29803A 16-way Branch Control Unit (U23). These performance enhancements are more related to function than to actual circuit speed. The use of these enhancements by the microprogram provides greater flexibility in controlling a machine's environment, and can reduce the microinstruction count required to perform a particular task, which has the effect of increasing overall system throughput.

In describing this high performance CCU design, those sections which remain unchanged from the previous description (Figure 17), will not be covered again. This includes the mapping PROMs, sequencer, Am29811A, counter, condition test inputs and associated polarity control, and the pipeline register. The areas that will be covered are: extended enable control (U16), Vector inputs (U24 and U25), and the Am29803A 16-way Branch Control Unit (U23).

### Extended Enable Control

Extended enable control is accomplished via an Am74S139 dual two-to-four line decoder in conjunction with the Am29811A next address control unit. In Figure 17, PL E and MAP E of the Am29811A were connected directly to the components that they are to control (pipeline registers and mapping PROMs, respectively). Likewise, $\overline{\text{CNT LOAD}}$ and $\overline{\text{CNT ENABLE}}$ are connected directly to the counters that they control (with the exception that $\overline{\text{CNT ENABLE}}$ requires inversion when using Am25LS163 counters). In Figure 18, $\overline{\text{PL E}}$, $\overline{\text{MAP E}}$, $\overline{\text{CNT LOAD}}$ and $\overline{\text{CNT ENABLE}}$ go to the inputs of the Am74S139 two-to-four line decoder (U16). When either $\overline{\text{PL E}}$ or $\overline{\text{MAP E}}$ is LOW, then either $2Y_1$ or $2Y_2$ of U16 is LOW and either the pipeline branch address registers or mapping PROMs are enabled. If both PL E and MAP E are HIGH, then output $2Y_3$ of U16 is LOW enabling the three-state outputs of U24 and U25 which are alternate microprogram starting address decoders (alternate mapping PROMs), and called VECTOR INPUT in this design. Likewise, $\overline{\text{CNT LOAD}}$ and $\overline{\text{CNT ENABLE}}$ follow the same rules, enabling the counter to load or count via $1Y_1$ and $1Y_2$ of U16.

### Vector Input

The "Vector Input" provides the system designer with a powerful next starting address control. For example, one possible use might be as an interrupt vector. For instance, use the "Interrupt Request" output of an Am2914 Vectored Priority Interrupt Controller (or group of Am2914s) as an input to one of the conditional test inputs of multiplexers (U12 or U14). Then connect the Am2914 Vector Out lines to the vector mapping PROMs (Vector input U24 and U25). The microprogram then could, at the appro-

Note: Figures 17, 18, 20, and 24 are at back of the book.

priate time, test for a pending interrupt and if present, jump in microprogram memory directly to the routine which handles the specific interrupt as requested via the Am2914 Vector Output lines. This routine will take the proper steps to preserve the status of the interrupt system, and then will service the interrupt. This is one of many possible uses for the Vector Input. Other possible uses include both hardware and software "TRAP" routines and so forth. As can be seen, the design presented here uses the Vector Enable line (output $2Y_3$ or U16) to enable an alternate starting address input at the Am2911. This, however, does not preclude the use of other devices in place of mapping PROMs as the D-input vector source.

It should be understood that this does not accomplish a "micro-interrupt" function in that it is not a random possibility. Instead a microprogrammed test is made and an alternate microroutine is performed. A true "microprogram interrupt" is one that could occur at any microinstruction. The Am2910 does not handle this case internally.

### Am29803A 16-Way Branch Control Unit

The Am29803A provides 16-way branch control when used in conjunction with the Am2909 bipolar microprocessor sequencer, and is shown as U23 in Figure 18 with its pipeline register U22. The Am29803A has four TEST-inputs, four INSTRUCTION-inputs, four OR-outputs, and an enable control. The four OR-outputs connect directly to the Am2909 OR-inputs (U8 in Figure 18). The four INSTRUCTION-inputs to the Am29803A provide control over the TEST-inputs and OR-outputs, and are provided by the microprogram via the pipeline register U22 (Figure 18).

Basically, the INSTRUCTION-inputs ($I_0$-$I_3$) provide sixteen instructions (0-$F_{16}$) which can select sixteen possible combinations of the TEST-inputs and provide a specific output on the OR-outputs depending upon the state of the inputs being tested. (The subscript 16 refers to basic 16.) All possible combinations of instruction-inputs, TEST-inputs and OR-outputs are shown in Figure 19.

Note that instruction zero does not test any inputs (a disable instruction). Instructions 1, 2, 4 and 8 test one input and can cause a branch to one of two words. Instructions 3, 5, 6, 9, 10 and 12 test two inputs and can jump to one of four words (a 4-word page). Instructions 7, 11, 13 and 14 test three inputs and can jump on an eight word page. Instruction number 15 tests all four inputs and the result can jump to any word on a sixteen word page.

### USING THE Am29803A

In the architecture of Figure 18, the Am29803A allows 2-way, 4-way, 8-way or 16-way branching as determined by selectable combinations of the TEST-inputs. Referring to Figure 19, the ZERO instruction (all instruction bits LOW) inhibits the testing of any TEST-inputs, thus providing LOW OR-outputs. Any single TEST-input selected ($T_0$, $T_1$, $T_2$ or $T_3$) will result in $OR_0$ being HIGH or LOW in correspondence with the polarity of the selected TEST-input. Selecting any combination of two TEST inputs results in the outputs $OR_0$ and/or $OR_1$ being HIGH or LOW, following a mapped one-to-one relationship, i.e., $OR_0$ and $OR_1$ will follow the TEST-inputs, but no matter which pair of TEST-inputs are selected, their HIGH/LOW condition is mapped to the $OR_0$ and $OR_1$ outputs. Likewise, selecting any three TEST inputs, will map their HIGH/LOW condition to the $OR_0$, $OR_1$ and $OR_2$ outputs. Selecting all four TEST-inputs, of course, causes a one-to-one relationship to exist between the HIGH/LOW conditions of the TEST-inputs and the corresponding OR-outputs. Refer to Figure 19 to verify the relationships between INSTRUCTION-inputs, TEST-input, and OR-output. It is very important that the mapping relationship between these signals be completely understood. When using the Am29803A TEST-OR capability as shown in Figure 18, the microprogrammer must position the applicable microcode within microprogram memory so that the low-order address bits are available for ORing. Sequencer instructions using the Am2909/2911 D-inputs (JRP, JSRP, JP and CJS in particular) are ideally suited for the Am29803A TEST-OR capability. The jump-to-location, available via pipeline $BR_0$-$BR_{11}$ or the Am2909/2911 register, can contain the address of a branch table. A branch table is merely a sequential series of unconditional jump instructions. The particular jump instruction executed is determined by the low-order address bits; that is, the first jump instruction in a branch table must start at a location in microprogram memory whose low-order address bit (or bits) is zero. If a single Am29803A TEST-input is selected (2-way branching) then only the least significant bit in the beginning branch table address needs to be zero. Two Am29803A TEST-inputs selected (4-way branching) requires that the branch table start on an address with the low-order two bits equal to zero; 8-way branching requires three low-order zero bits, and 16-way branching requires four low-order zero address bits. Understanding this branch control concept is really quite simple. The branch table is located in microprogram memory beginning at a location whose address has sufficient low-order zero bits to accommodate the number of selected Am29803A TEST-inputs. If, for instance, three TEST-inputs were selected, the first jump instruction in the branch table must be at an address whose low-order three bits are zero, such as address $0F8_{16}$. The second jump instruction in the branch table would begin in microprogram memory address $0F9_{16}$. The third jump at location $0FA_{16}$, the fourth at $0FB_{16}$, etc. Through all eight locations ($0F8_{16}$-$0FF_{16}$). Assume the following pipeline instruction (referring to Figure 18): (1) U22 selects three Am29803A TEST-inputs, (2) U18 instructs the Am29811A Next Address Controller to select the Am2909/2911 D-inputs, (3) U16 enables the pipeline branch address as the D source, and (4) U19, U20 and U21 supplies the address $0F8_{16}$ as the branch address. The Am29803A TEST-inputs will be ORed into the low-order three bit positions, thus providing a jump entry into the branch table *indexed* by the value of the OR bits. Each instruction in the branch table is usually a jump instruction, which allows the selection of a particular microcode routine determined by the value presented at the Am29803A TEST-inputs. These jump instructions are the first instruction of the particular sequence. There are, of course, many other ways to use the Am29803A 16-way Branch Control Unit.

The microprogram memory address supplied via an Am2909 sequencer can be modified by the Am29803A 16-way Branch Control Unit. Remember, however, that the microcode associated with this address modification relies on certain address bits being zero, therefore this microcode is not arbitrarily relocatable. The above discussion describes using the D-input and branching to provide low-order zeroes to use the OR inputs. Through proper design, the Register, PC Counter, or File can be used equally well.

### THE COMPLETE COMPUTER CONTROL UNIT USING THE Am2910

A detailed connection diagram for a straightforward computer control unit using the Am2910 is shown in Figure 20. This design utilizes the Am25LS377 as U1 and U2 to implement a 16-bit instruction register. The op code outputs from the instruction register drive three Am29761 PROMs to perform the op code decoding function. These are shown in the diagram of Figure 20 as U3, U4 and U5. The Am2910 sequencer (U6) is used to perform the basic microprogram sequencing function.

Note: Figures 17, 18, 20, and 24 are at back of the book.

| Function | $I_3$ | $I_2$ | $I_1$ | $I_0$ | $T_3$ | $T_2$ | $T_1$ | $T_0$ | $OR_3$ | $OR_2$ | $OR_1$ | $OR_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| No Test | L | L | L | L | X | X | X | X | L | L | L | L |
| Test $T_0$ | L | L | L | H | X | X | X | L | L | L | L | L |
| | | | | | X | X | X | H | L | L | L | H |
| Test $T_1$ | L | L | H | L | X | X | L | X | L | L | L | L |
| | | | | | X | X | H | X | L | L | L | H |
| Test $T_0$ & $T_1$ | L | L | H | H | X | X | L | L | L | L | L | L |
| | | | | | X | X | L | H | L | L | L | H |
| | | | | | X | X | H | L | L | L | H | L |
| | | | | | X | X | H | H | L | L | H | H |
| Test $T_2$ | L | H | L | L | X | L | X | X | L | L | L | L |
| | | | | | X | H | X | X | L | L | L | H |
| Test $T_0$ & $T_2$ | L | H | L | H | X | L | X | L | L | L | L | L |
| | | | | | X | L | X | H | L | L | L | H |
| | | | | | X | H | X | L | L | L | H | L |
| | | | | | X | H | X | H | L | L | H | H |
| Test $T_1$ & $T_2$ | L | H | H | L | X | L | L | X | L | L | L | L |
| | | | | | X | L | H | X | L | L | L | H |
| | | | | | X | H | L | X | L | L | H | L |
| | | | | | X | H | H | X | L | L | H | H |
| Test $T_0$, $T_1$ & $T_2$ | L | H | H | H | X | L | L | L | L | L | L | L |
| | | | | | X | L | L | H | L | L | L | H |
| | | | | | X | L | H | L | L | L | H | L |
| | | | | | X | L | H | H | L | L | H | H |
| | | | | | X | H | L | L | L | H | L | L |
| | | | | | X | H | L | H | L | H | L | H |
| | | | | | X | H | H | L | L | H | H | L |
| | | | | | X | H | H | H | L | H | H | H |
| Test $T_3$ | H | L | L | L | L | X | X | X | L | L | L | L |
| | | | | | H | X | X | X | L | L | L | H |
| Test $T_0$ & $T_3$ | H | L | L | H | L | X | X | L | L | L | L | L |
| | | | | | L | X | X | H | L | L | L | H |
| | | | | | H | X | X | L | L | L | H | L |
| | | | | | H | X | X | H | L | L | H | H |
| Test $T_1$ & $T_3$ | H | L | H | L | L | X | L | X | L | L | L | L |
| | | | | | L | X | H | X | L | L | L | H |
| | | | | | H | X | L | X | L | L | H | L |
| | | | | | H | X | H | X | L | L | H | H |
| Test $T_0$, $T_1$ & $T_3$ | H | L | H | H | L | X | L | L | L | L | L | L |
| | | | | | L | X | L | H | L | L | L | H |
| | | | | | L | X | H | L | L | L | H | L |
| | | | | | L | X | H | H | L | L | H | H |
| | | | | | H | X | L | L | L | H | L | L |
| | | | | | H | X | L | H | L | H | L | H |
| | | | | | H | X | H | L | L | H | H | L |
| | | | | | H | X | H | H | L | H | H | H |
| Test $T_2$ & $T_3$ | H | H | L | L | L | L | X | X | L | L | L | L |
| | | | | | L | H | X | X | L | L | L | H |
| | | | | | H | L | X | X | L | L | H | L |
| | | | | | H | H | X | X | L | L | H | H |
| Test $T_0$, $T_2$ & $T_3$ | H | H | L | H | L | L | X | L | L | L | L | L |
| | | | | | L | L | X | H | L | L | L | H |
| | | | | | L | H | X | L | L | L | H | L |
| | | | | | L | H | X | H | L | L | H | H |
| | | | | | H | L | X | L | L | H | L | L |
| | | | | | H | L | X | H | L | H | L | H |
| | | | | | H | H | X | L | L | H | H | L |
| | | | | | H | H | X | H | L | H | H | H |
| Test $T_1$, $T_2$ & $T_3$ | H | H | H | L | L | L | L | X | L | L | L | L |
| | | | | | L | L | H | X | L | L | L | H |
| | | | | | L | H | L | X | L | L | H | L |
| | | | | | L | H | H | X | L | L | H | H |
| | | | | | H | L | L | X | L | H | L | L |
| | | | | | H | L | H | X | L | H | L | H |
| | | | | | H | H | L | X | L | H | H | L |
| | | | | | H | H | H | X | L | H | H | H |
| Test $T_0$, $T_1$, $T_2$ & $T_3$ | H | H | H | H | L | L | L | L | L | L | L | L |
| | | | | | L | L | L | H | L | L | L | H |
| | | | | | L | L | H | L | L | L | H | L |
| | | | | | L | L | H | H | L | L | H | H |
| | | | | | L | H | L | L | L | H | L | L |
| | | | | | L | H | L | H | L | H | L | H |
| | | | | | L | H | H | L | L | H | H | L |
| | | | | | L | H | H | H | L | H | H | H |
| | | | | | H | L | L | L | H | L | L | L |
| | | | | | H | L | L | H | H | L | L | H |
| | | | | | H | L | H | L | H | L | H | L |
| | | | | | H | L | H | H | H | L | H | H |
| | | | | | H | H | L | L | H | H | L | L |
| | | | | | H | H | L | H | H | H | L | H |
| | | | | | H | H | H | L | H | H | H | L |
| | | | | | H | H | H | H | H | H | H | H |

L = LOW, H = HIGH, X = Don't care

Figure 19. Function Table.

A 16 input condition code multiplexer function is provided by using two Am2922s as U7 and U8. These devices allow one of sixteen inputs to be tested and the polarity of the test can also be determined. The pipeline register consists of U9, U10, U11, U12 and U13. These devices are edge triggered D type registers and have been selected to provide unique functions as required depending on their bit positions in the pipeline register. An Am74S175 was selected for U9 because both a true and complement output were desired to provide control to the condition code multiplexer three state enables. An Am74S174 register was selected as U10 because it provides a clear input for initializing the Am2910 microprogram sequencer. Three Am2918s were selected for U11, U12 and U13 because they have a three state output that can be used to provide the branch address field to the D inputs of the Am2910 and they also have a set of outputs that can be used to provide other control signals via this field when it does not contain a branch address. No specific devices are shown for the microprogram memory as the user should select the desired width and depth depending on his design.

## ANOTHER DESIGN EXAMPLE

The Am2909, Am2910, Am2911, Am29811A and Am29803A have been designed to operate in the microprogram sequencing section of any digital state machine. Typically, the examples shown are for performing the computer control unit function of a typical minicomputer class machine. The design engineer should not limit his thinking for the use of these devices simply to that of microprogram sequencing in a computer control unit. These devices can be successfully used in other areas of designing such as memory control, DMA control, interrupt control and special purpose microprogrammed machine architectures. In order to provide an example of a design using these devices in something other than a typical computer control unit, a microprogrammed CRT controller is described in the following.

In order to provide some basis for the design of a CRT controller, the requirements of this controller must be spelled out. These are given as follows:

A) Character size: 5 x 7 dot matrix. The character field will be 7 dots by 10 horizontal lines thereby providing ample space for the 5 x 7 character and the intervening space between characters and lines of characters.

B) 80 characters per line. A standard 80 character per line display will be utilized and there will be 18 character periods allowed for horizontal retrace time.

C) 24 lines of characters per frame. This provides a total of 240 visible lines per frame (24 lines of characters by 10 horizontal lines per character). There are a total of 24 lines provided for vertical retrace bringing the total number of lines per frame to 264.

D) Refresh rate: 60 frames per second. Therefore, the horizontal line rate will be 264 x 60 = 15,840Hz. As there are a total of 80 + 18 = 98 character periods in a line, the character rate will be 98 x 15.84 = 1,552.32KHz, and the dot rate will be 7 x 1.5288 = 10.86624MHz. (Note: No interlace is used.)

E) It is assumed that there is a 2K word deep x 8-bit wide character RAM available to the host computer in which it can write the ASCII equivalent of the characters to be displayed. If scrolling is to be used, the host computer must also write the first visible character's address divided by $16_{10}$ into the Am25LS374 "First Address Register".

F) This CRT controller must generate an 11-bit character address that is used by the 2K word deep character RAM. It must also generate the required video enable signals and the horizontal and vertical blanking signals.

**Principle of Operation**

A detailed block diagram of the CRT controller is shown in Figure 21. The block diagram shows an interface to an SBC-80/10 data bus, address bus and control bus. The outputs of the CRT controller are connected to a CRT monitor on the block diagram. Otherwise the block diagram shows a straightforward use of the Am2910 and three Am2911s to implement the CRT control function using microprogrammed techniques. The SBC-80/10 was selected for this example since it is well known.

A logic diagram of the CRT controller is shown in Figure 22. Three Am29775 512-word x 8-bit registered PROMs are used to contain the 23-bit wide microprogram. While only a minimum number of words are used in the design as shown, many additional words can be used to add various options (as described later). The address for these Am29775 registered PROMs is provided by an Am2910 microprogram sequencer. Three Am2911 sequencers are used to generate the character address for the character RAM. The least significant Am2911 sequencer is connected as a divide by 16 counter. This RAM address is compared with the desired last character address (80 x 24 = 1920) value using an Am25LS2521 8-bit equal to detector. When the last address is detected, it can be sensed at the condition code multiplexer (Am25LS153) that is used to select the condition code for the Am2910 sequencer.

The data derived from the 2K word character RAM is decoded by a character generator (6061) in this design and the character output is parallel loaded into an Am25LS23 shift register. This shift register is used to provide the video signal from its $Q_0$ output to eventually drive the display via an Am74S240 buffer. The diagram of Figure 22 depicts an oscillator input source to supply the dot frequency. In this design, a 10.86624MHz oscillator should be connected to this oscillator input point. This oscillator input signal is used to clock the shift register containing the individual dot bits (dot-on or dot-off) and also drives an Am25LS169 counter which divides this frequency by 7 to generate the character rate clock. This character rate clock is used throughout the controller to provide a timing signal for the state machine design.

An Am25LS168 decade counter is used to generate the line inputs for the character generator and to count 10 horizontal lines per character space. This counter is clocked by the horizontal blanking signal (HB) and its $\overline{\text{RCO}}$ output is used as one of the condition code multiplexer inputs. The $\overline{\text{RCO}}$ output can be tested to determine when 10 counts have been executed by the counter and it is also used to enable the last address comparator during the 10th horizontal line time.

When the host computer accesses the character RAM, the $\overline{\text{HOST-ACCESS}}$ line is pulled LOW. This removes the Am2911 outputs from the character RAM address bus. When this access occurs, improper data may be present at the shift register inputs. Thus, the character generator PROM output is disabled by the $\overline{\text{HOST-ACCESS}}$ signal during this time.

When power is applied to this CRT controller or whenever it is reset, the RESET line is driven LOW. This signal is inverted through an Am25LS240 and then disables a part of the pipeline register outputs as well as enabling one half of an Am25LS241. This Am25LS241 inserts LOWs onto the instruction (I) inputs of the Am2910 sequencer. Then, the next character rate clock will force the microprogram address outputs to zero and the microprogram for the CRT controller as shown in Figure 23 will be executed starting at address zero.

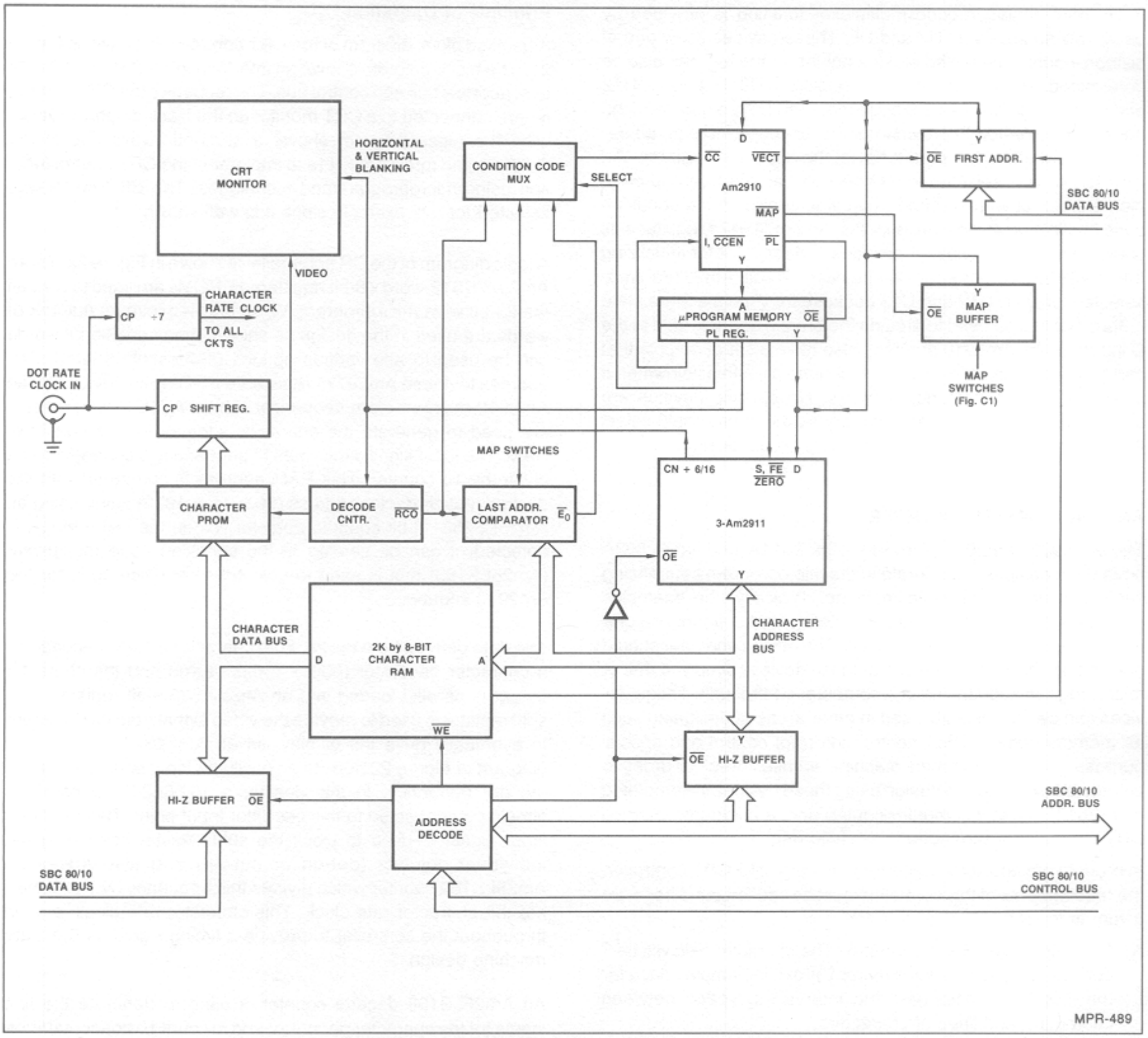


**Figure 21. CRT Controller Block Diagram.**

### The Microprogram for the CRT Controller

Table 6 shows a complete description of the microprogrammed CRT controller microcode. Execution of these microinstructions is controlled by the Am2910 sequencer.

As can be seen in Table 6, several techniques were used in this short microprogram to provide the different counting requirements of this CRT controller. Although only one format (80 characters per line, 24 lines per frame) was shown here, the designer can easily configure his own format by simply changing some constants in the microprogram. As an exercise, the reader is encouraged to find a means to program the CRT controller for different formats. The host computer software could configure the controller format by using an additional register similar to the "First Address Register". This will be discussed in an appendix at the end of this chapter.

A complete wiring diagram for the microprogrammed CRT controller is shown in Figure 24. This can be used directly with the interface shown in Appendix A such that the CRT controller can be connected directly to an Am9080A based microprocessor system. Appendix A also depicts the use of a 2K word x 8 bit character RAM as described previously.

### CRT Controller Timing Considerations

As was discussed earlier, the character clock frequency for the CRT controller is 1,552.32KHz. Thus, it is desirable to calculate the longest path of the design to ensure that none exceed this clock period of 644.1ns. The timing diagrams of Figure 25 depict a number of different paths with the associated propagation delay calculations.

When all of the timing diagrams of Figure 25 are examined, it will be found that only three show propagation delay times of over 200ns typical. Of these, the worst case is 318ns as shown in Figure 25(i). Since the requirement of the design is to insure that none exceed 644.1ns, we have more than a 2 to 1 margin in the design based on the typicals. Thus, we can see that the design will operate properly even over the full military temperature range and power supply variations based on this analysis.

Note: Figure 24 is at back of the book.

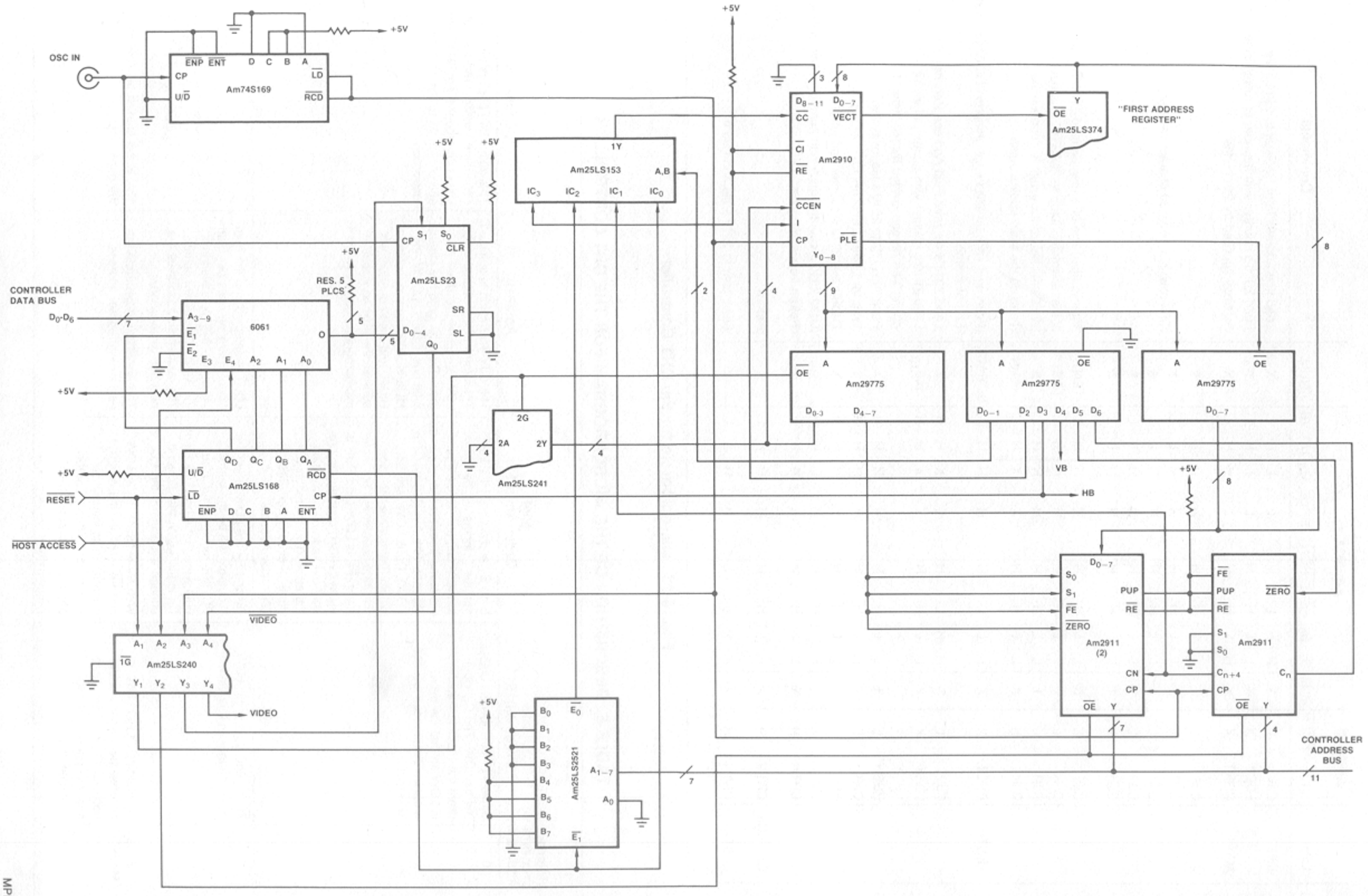


Figure 22. CRT Controller.

| ADDR | | Am2910 | | | Am2911 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (Hex) | Label | I | $\overline{\text{CCEN}}$ | MUX | $S_1$ | $S_0$ | $\overline{\text{FE}}$ | $\overline{\text{ZEROH}}$ | $\overline{\text{ZEROL}}$ | $C_n$ | HB | VB | NUM | Comments |
| 0 | INIT | CJV | L | 3 | H | H | L | H | L | L | H | L | X | ;Load first address from Register to 2911's file |
| 1 | | LDCT | X | X | L | L | H | H | L | L | H | L | $23_{10}$ | ;Load 2910's counter with member of rows/frame − 1 |
| 2 | MAIN | CONT | X | X | H | L | H | H | L | H | H | L | X | ;Address supplied by 2911's file |
| 3 | | CJP | L | 1 | L | L | H | H | H | H | L | L | $ | ;One row: 5 x 16 = 80 characters |
| 4 | | CJP | L | 1 | L | L | H | H | H | H | L | L | $ | |
| 5 | | CJP | L | 1 | L | L | H | H | H | H | L | L | $ | |
| 6 | | CJP | L | 1 | L | L | H | H | H | H | L | L | $ | |
| 7 | | CJP | L | 1 | L | L | H | H | H | H | L | L | $ | |
| 8 | | CJS | L | 0 | L | L | H | H | H | H | H | L | TENTH | ;If tenth (last) line of a row: jump to "TENTH" subroutine |
| 9 | | CJS | L | 2 | L | L | H | H | H | H | H | L | LASTA | ;If last character: jump to "LASTA" subroutine |
| A | | CJP | L | 1 | L | L | H | H | H | H | H | L | $ | ;Wait, until horizontal invisible counts done |
| B | | CJP | H | X | L | L | H | H | X | X | H | L | MAIN | ;Then do the Main routine again |
| C | TENTH | RPCT | X | X | L | L | L | H | H | H | H | L | GOBACK | ;Push next addr on 2911's file: jump to "GOBACK" if not End of Frame |
| D | | CJV | L | 3 | H | H | L | H | L | X | H | H | X | ;Load 2911's file from First Address Register |
| E | | LDCT | X | Y | L | L | H | H | X | X | H | H | $146_{10}$ | ;Load 2910's counter with number of invisible characters during Vert retrace divided by 16, minus 1 |
| F | | PUSH | L | 3 | L | L | H | H | H | H | H | H | X | ;Push next PC to 2910's file for double |
| 10 | | CJP | L | 1 | L | L | H | H | H | H | H | H | $ | ;Wait for LS2911 to count 16 |
| 11 | | RFCT | X | X | L | L | H | H | H | H | H | H | X | ;Decrement 2910's counter and jump one line back if = 0 |
| 12 | | LDCT | X | X | L | L | H | H | H | H | H | H | $23_{10}$ | ;Load 2910's counter again with number of rows/frame − 1 |
| 13 | | CRTN | H | X | L | L | H | H | H | H | H | H | X | ;Return from subroutine |
| 14 | GOBACK | CRTN | H | X | L | L | H | H | H | H | H | L | X | ;Return |
| 15 | LASTA | CRTN | H | X | X | X | L | L | H | H | H | L | X | ;Load zero to 2911's file and return. |

**Figure 23. Microprogram for the CRT Controller.**

### TABLE 6. DESCRIPTION OF THE MICROPROGRAM FOR THE CRT CONTROLLER.

| Micro-program Address | Low Order Am2911 | High Order Am2911s | Am2910 | Comments |
|---|---|---|---|---|
| 0 | Since $\overline{\text{ZERO}}$ is low, its output will be LOW. The $C_n$ input (from the Pipeline Register) is LOW so that the microprogram incrementer will not increment. | Both $S_1$ and $S_0$ are HIGH so that the D inputs will be routed to the Y outputs. These inputs will come from the First Address Register (the Am2910 $\overline{\text{VECT}}$ is LOW). $C_n$ is LOW (see left column); therefore the microprogram counter will not increment. $\overline{\text{FE}}$ is LOW (and PUP is always HIGH) causing the present output to be pushed on the stack. The character address is already the "First Character Address". | The CJV instruction is selected. Therefore, $\overline{\text{VECT}}$ output will be LOW, enabling the "First Address Register onto the internal 8-bit bus. $\overline{\text{CCEN}}$ is LOW; the MUX is selecting a constant HIGH, and the sequencer will address the next consecutive microprogram address (word 1). | This instruction pushes the "First Character Address" more significant bits onto the Am2911's file, and continues to the next microinstruction. |
| 1 | $\overline{\text{ZERO}}$ and $C_n$ are still LOW, so no change in this device. | $S_1$ and $S_2$ are LOW; thus, the Y outputs will be the current PC, (the same as the Y outputs were in the previous step). $C_n$ is still LOW, therefore no change will occur in the PC. | LDCT is selected and the number of character-rows per frame minus 1 ($23_{10}$) is loaded into the Am2910 register/counter. The sequencer addresses the next microinstruction. | |
| 2<br>"MAIN" | Maintaining $\overline{\text{ZERO}}$ LOW assures the proper starting address. $C_n$ is HIGH; therefore, the internal PC will be incremented. | With $\underline{S_1}$ = HIGH, $S_0$ = LOW and $\overline{\text{FE}}$ = HIGH, the Am2911 will refer to its internal file (the starting address of this particular character-row) without popping. | The Am2910 will generate the next microprogram address. | This is the starting location for the main loop. |

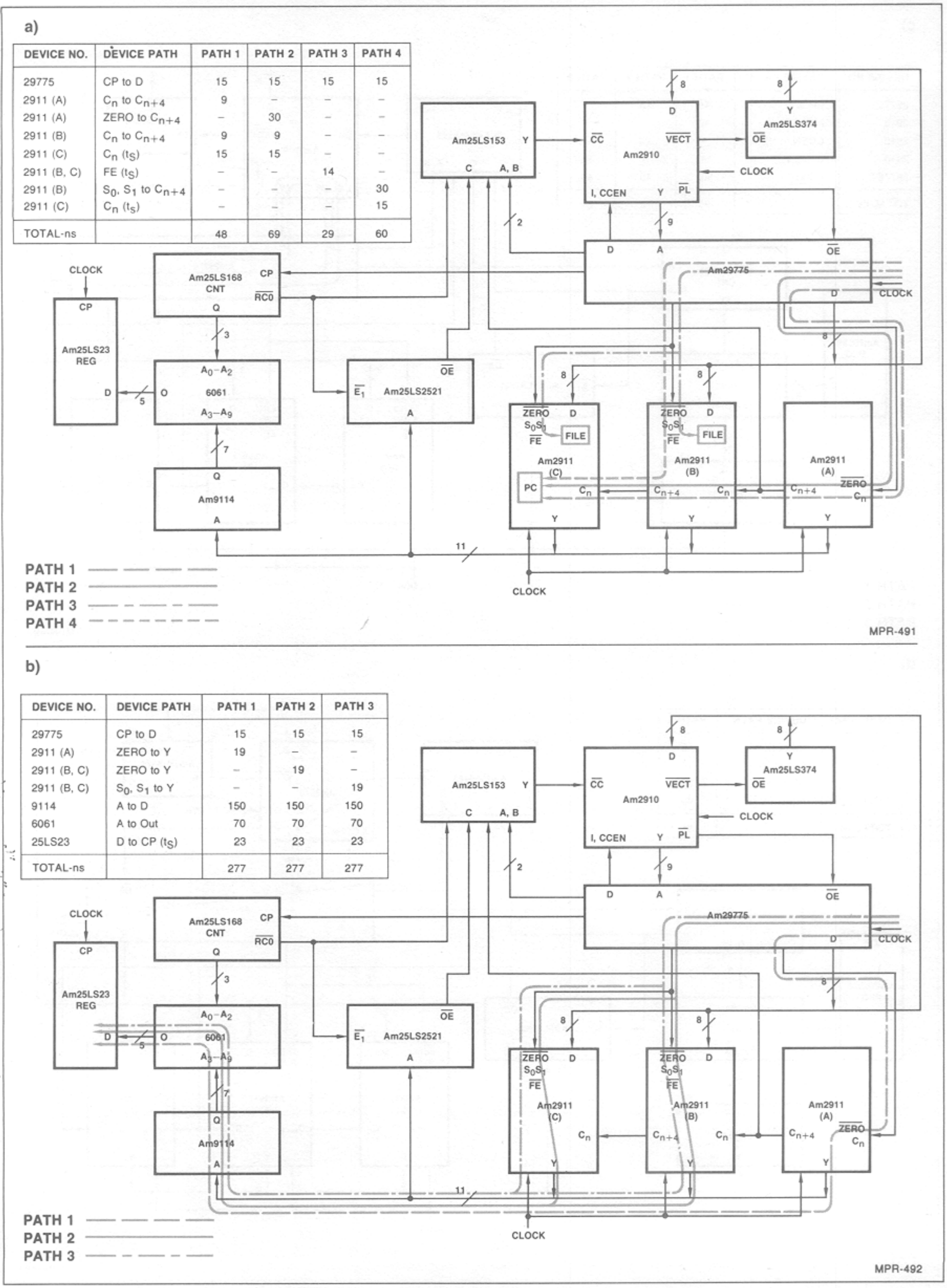
Note: Figure 24 is at back of the book.

## TABLE 6. DESCRIPTION OF THE MICROPROGRAM FOR THE CRT CONTROLLER (Cont.).

| Micro-program Address | Low Order Am2911 | High Order Am2911s | Am2910 | Comments |
|---|---|---|---|---|
| 3 | This Am2911 now counts up using its PC incrementer. At the final count (moving from $F_{16}$ to 0) its $C_{n+4}$ output will be HIGH. | Initially these two Am2911s will not change their Y outputs since their $C_n$ input is LOW. However, when the $C_n$ input goes HIGH, the internal PC will increment | With the MUX selecting the $C_{n+4}$ output from the least significant Am2911 slice, the CC input to the Am2910 sequencer will be LOW until the Am2911 counts 16. $\overline{CC}$ = LOW will cause the next microprogram address to be the pipeline register contents; this is also the current microprogram address (word 3). When $C_{n+4}$ goes HIGH, $\overline{CC}$ will go HIGH and together with $\overline{CCEN}$ = LOW, will force the Am2910 to address the next consecutive microprogram address (4). | This microstep will be executed 16 times. (Note that 80 = 5 x 16.) |
| 4 through 7 | Same as 3. | Same as 3. | Same as 3, except that at each address, the current microprogram address is written. | The microprogram itself is used as a counter in this application since the count is only 5, the microprogram is relatively short versus the memory's depth and this is a convenient means to economize on chip count. |
| 8 | Continues to count (note that it enters this line with an output of zero). | Since $C_n$ is LOW (see left column) no change occurs in these devices. Note that the Y outputs contain the more significant bits of the address of the first character of the *next* character row. | The MUX selects the Am25LS168 ten-line-counters $\overline{RCO}$ as the condition code input to the Am2910 (CC). If the line count is less than 10, $\overline{CC}$ will be HIGH and the next microinstruction will be addressed. If the tenth line of a character row is executed, $\overline{CC}$ will be LOW and a JUMP-TO-SUBROUTINE to an address, supplied by the pipeline register ("TENTH") will be executed. | We are now at the end of a TV line. Therefore, the Horizontal Blanking Signal (HB) is HIGH. The least significant Am2911 slice now counts the invisible characters during the horizontal retrace. |
| 9 | Continues to count through the internal PC incrementer. | No change. | The MUX now selects the Last Address Comparator output for $\overline{CC}$. If the current more significant bits of the character-address coincide with the last address + 1 ($1920_{10}$/16) a subroutine call will be performed to "LASTA". Otherwise, the microprogram will continue consecutively. | Note that 80 characters/row and 24 rows/frame requires a $1920_{10}$ word memory. When the last memory location ($1920_{10}$) is read out, the scan will begin at 0. |
| A | Continues to count. At count 15, $C_{n+4}$ goes HIGH. | No change until $C_n$ goes HIGH, then count. | Same as at address 3. | Waiting for the least significant Am2911 to count to 15. This microstep will be executed as many times as necessary to accomplish this. |
| B | It doesn't matter what this device does at this microstep because at the next microstep it will receive LOW on its $\overline{ZERO}$ input. | No change. | Unconditionally ($\overline{CCEN}$ = HIGH) steers the microprogram to the address supplied by the pipeline register ("MAIN" = 2). | Performing a JUMP to the beginning of the main-loop (address 2). |
| C "TENTH" | Continues to count. | No change. | If internal counter is equal to zero, it means that 24 character rows were already displayed and we are at the bottom of the CRT display. A vertical retrace period is needed and the microprogram will continue sequentially. If the counter is not yet zero, we do not need to execute the vertical retrace routine and the next address will be supplied by the pipe-register ("GOBACK" = $14_{16}$) while the internal counter is decremented. | The decision whether the bottom of the CRT (End of Frame) is reached or not is made internally in the Am2910, using its counter. |

## TABLE 6. DESCRIPTION OF THE MICROPROGRAM FOR THE CRT CONTROLLER (Cont.).

| Micro-program Address | Low Order Am2911 | High Order Am2911s | Am2910 | Comments |
|---|---|---|---|---|
| D | $\overline{ZERO}$ = LOW, therefore, output Y = 0. This is necessary to assure that $C_{n+4}$ is LOW. | Same as at address 0. | Same as at address 0. | As we are at the End of Frame, the "First-Address-Register" contents (enabled by the Am2910's $\overline{VECT}$ output) is pushed onto the Am2911's file. Note that the Vertical Blanking Signal (VB) goes HIGH. |
| E | Same as at address B. | No change. | The internal counter is loaded with $146_{10}$, supplied by the pipeline register. The next consecutive microstep is addressed. | $(146_{10} + 1) \times 16_{10} = 2352_{10}$ equals the number of character-periods during vertical retrace. Loading $2352_{10}$ directly into the Am2910's counter would require 7 bits. Usingthis scheme we reduce the microprogram width. |
| F | Counts. | No change. | With $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH (supplied from a constant HIGH by the MUX), the next address ($10_{16}$) will be pushed onto the Am2910 file, the counter will not be affected and the next consecutive microstep will be addressed. | This is a preparatory step for the 2 step "Vertical Retrace" double-nested loop. |
| $10_H$ | Counts. When final count is reached, $C_{n+4}$ = HIGH. | No change with $C_n$ = LOW; increments with $C_n$ = HIGH. This has no practical affect as the HB signal is HIGH, and at the beginning of the next visible line, the correct address will be fetched from the file (address 2). | The MUX supplies the $C_{n+4}$ output of the less significant Am2911 slice to the Am2910 $\overline{CC}$ input. While this signal is low, the Am2910 will select the pipeline register as the source of the next microinstruction address. The current address ($10_H$) being written there, this instruction will be executed until $\overline{CC}$ goes HIGH. Then the next consecutive instruction will be selected through the Am2910 internal PC. | Again, this is a possible way to dwell on a certain microstep waiting a condition to change its status (like address 3 through 7). This is the internal loop of a double-nested loop system. |
| $11_H$ | Counts. | No change. | If the final count has been reached, the next microinstruction will be addressed and the internal stack will be popped (adjusted). Otherwise, the next microinstruction address will be the one residing on the top of the stack (which is $10_{16}$). | This is the external loop of the double-nested loop system, which counts the vertical retrace interval. By adding a single microinstruction the chip count was reduced. |
| $12_H$ | Counts. | No change. | Same as at address 1. | Reinitializes the Am2910 internal counter with the number of character rows per frame. |
| $13_H$ | Counts. | No change. | Unconditional return from subroutine. ($\overline{CCEN}$ = HIGH). | End of "TENTH" subroutine at End of Frame (with vertical retrace). |
| $14_H$ "GOBACK" | Counts. | No change. | Unconditional return from subroutine. | End of "TENTH" subroutine without vertical retrace. |
| $15_H$ "LASTA" | Counts. | Pushes zero into file. | Unconditional return from subroutine. | A one-line subroutine to reinitialize character address to zero. |



a)

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 | PATH 4 |
|---|---|---|---|---|---|
| 29775 | CP to D | 15 | 15 | 15 | 15 |
| 2911 (A) | $C_n$ to $C_{n+4}$ | 9 | – | – | – |
| 2911 (A) | ZERO to $C_{n+4}$ | – | 30 | – | – |
| 2911 (B) | $C_n$ to $C_{n+4}$ | 9 | 9 | – | – |
| 2911 (C) | $C_n$ ($t_S$) | 15 | 15 | – | – |
| 2911 (B, C) | FE ($t_S$) | – | – | 14 | – |
| 2911 (B) | $S_0$, $S_1$ to $C_{n+4}$ | – | – | – | 30 |
| 2911 (C) | $C_n$ ($t_S$) | – | – | – | 15 |
| TOTAL-ns | | 48 | 69 | 29 | 60 |

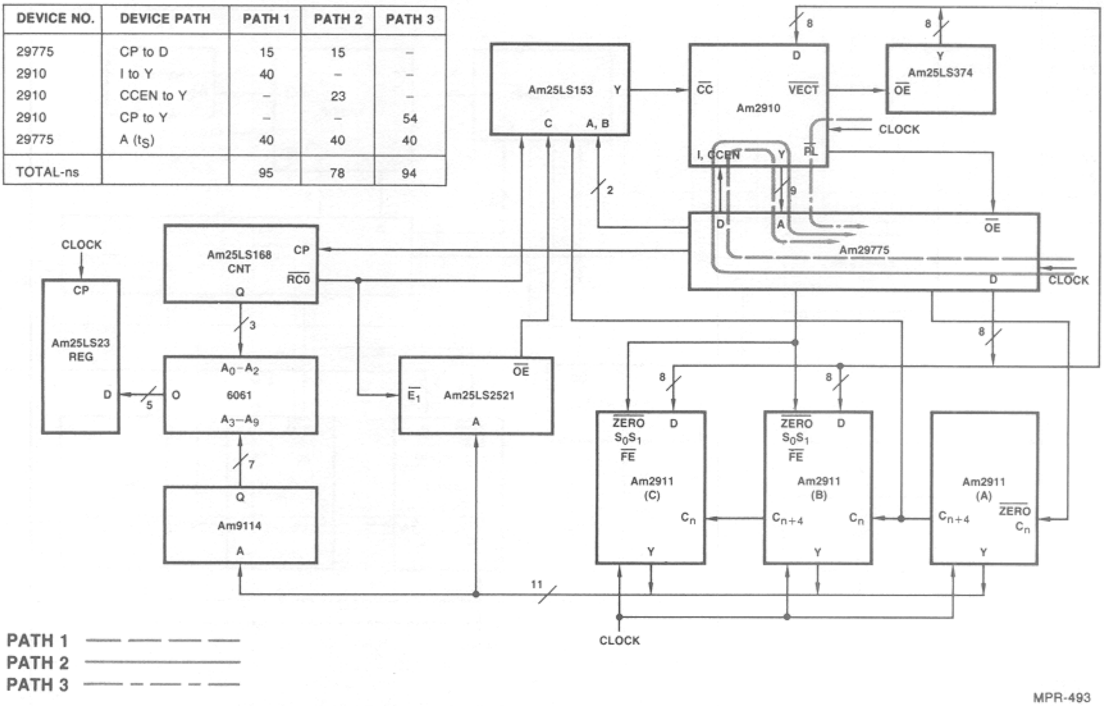
b)

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| 29775 | CP to D | 15 | 15 | 15 |
| 2911 (A) | ZERO to Y | 19 | – | – |
| 2911 (B, C) | ZERO to Y | – | 19 | – |
| 2911 (B, C) | $S_0$, $S_1$ to Y | – | – | 19 |
| 9114 | A to D | 150 | 150 | 150 |
| 6061 | A to Out | 70 | 70 | 70 |
| 25LS23 | D to CP ($t_S$) | 23 | 23 | 23 |
| TOTAL-ns | | 277 | 277 | 277 |

Figure 25.

c)

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| 29775 | CP to D | 15 | 15 | – |
| 2910 | I to Y | 40 | – | – |
| 2910 | CCEN to Y | – | 23 | – |
| 2910 | CP to Y | – | – | 54 |
| 29775 | A ($t_S$) | 40 | 40 | 40 |
| TOTAL-ns | | 95 | 78 | 94 |

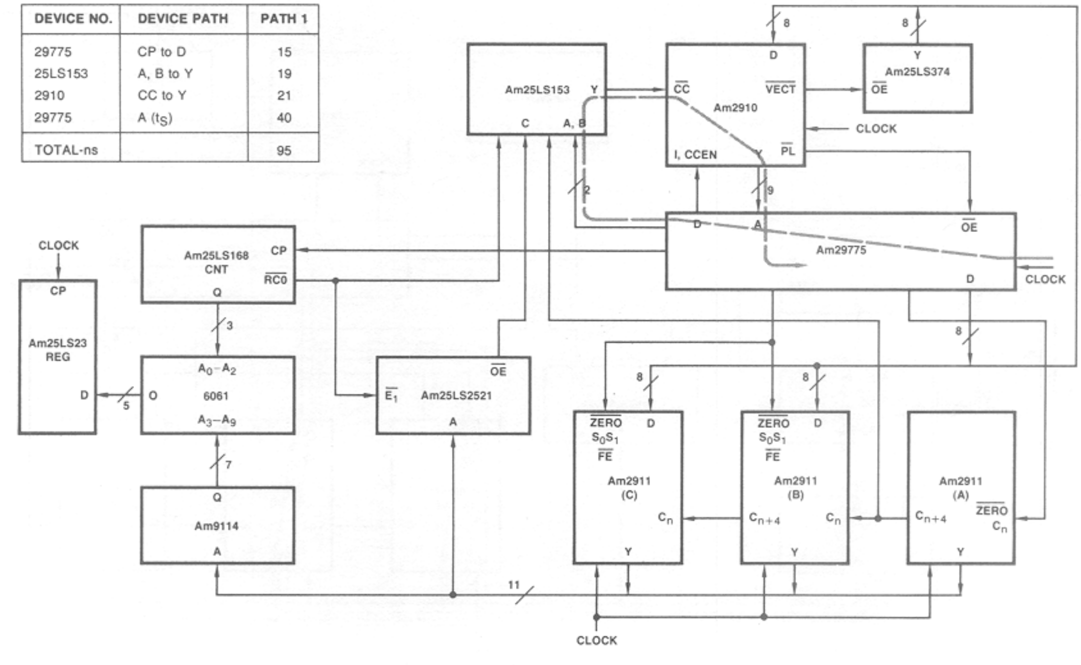


PATH 1 — — — —
PATH 2 ————————
PATH 3 —— – —— – ——

MPR-493

d)

| DEVICE NO. | DEVICE PATH | PATH 1 |
|---|---|---|
| 29775 | CP to D | 15 |
| 25LS153 | A, B to Y | 19 |
| 2910 | CC to Y | 21 |
| 29775 | A ($t_S$) | 40 |
| TOTAL-ns | | 95 |



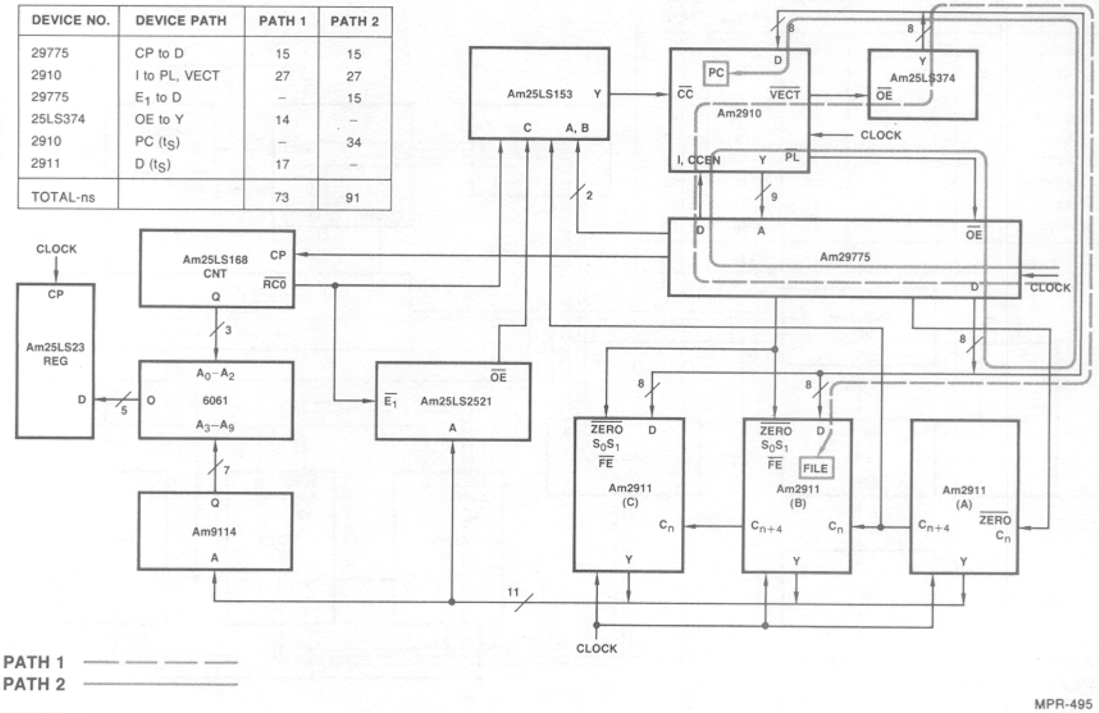
PATH 1 — — — —

MPR-494

Figure 25. (Cont.)

e)

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 29775 | CP to D | 15 | 15 |
| 2910 | I to PL, VECT | 27 | 27 |
| 29775 | $E_1$ to D | – | 15 |
| 25LS374 | OE to Y | 14 | – |
| 2910 | PC ($t_S$) | – | 34 |
| 2911 | D ($t_S$) | 17 | – |
| TOTAL-ns | | 73 | 91 |



f)

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| 29775 | CP to D | 15 | 15 | 15 |
| 2911 | ZERO to $C_{n+4}$ | – | – | 30 |
| 2911 | $C_n$ to $C_{n+4}$ | – | 9 | – |
| 25LS168 | CP to $\overline{RCO}$ | 19 | – | – |
| 25LS153 | D to Y | 20 | 20 | 20 |
| 2910 | CC to Y | 21 | 21 | 21 |
| 29775 | A ($t_S$) | 40 | 40 | 40 |
| TOTAL-ns | | 115 | 105 | 126 |

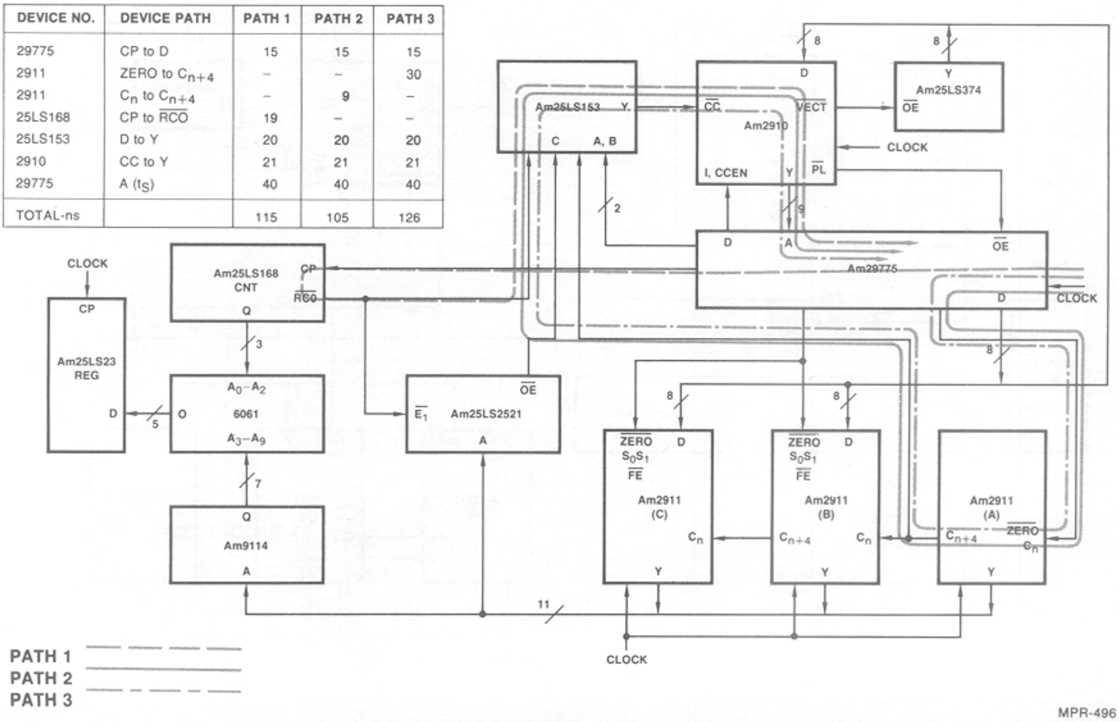
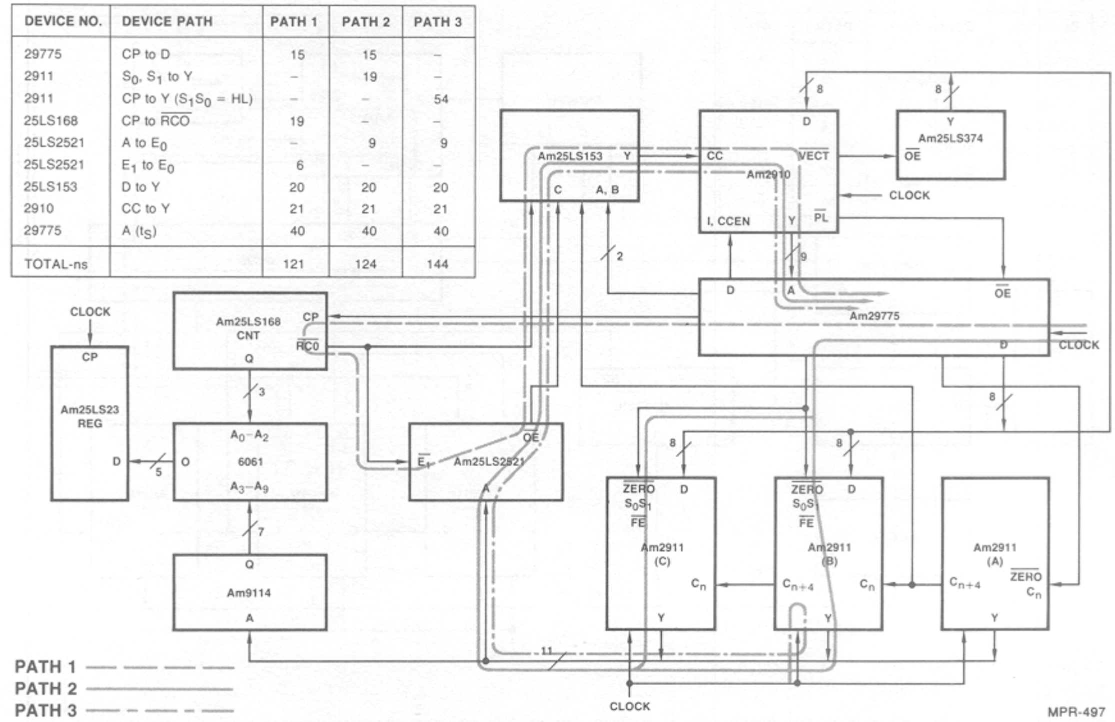


Figure 25. (Cont.)

g)

| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| 29775 | CP to D | 15 | 15 | – |
| 2911 | $S_0$, $S_1$ to Y | – | 19 | – |
| 2911 | CP to Y ($S_1S_0$ = HL) | – | – | 54 |
| 25LS168 | CP to $\overline{RCO}$ | 19 | – | – |
| 25LS2521 | A to $E_0$ | – | 9 | 9 |
| 25LS2521 | $E_1$ to $E_0$ | 6 | – | – |
| 25LS153 | D to Y | 20 | 20 | 20 |
| 2910 | CC to Y | 21 | 21 | 21 |
| 29775 | A ($t_S$) | 40 | 40 | 40 |
| TOTAL-ns | | 121 | 124 | 144 |

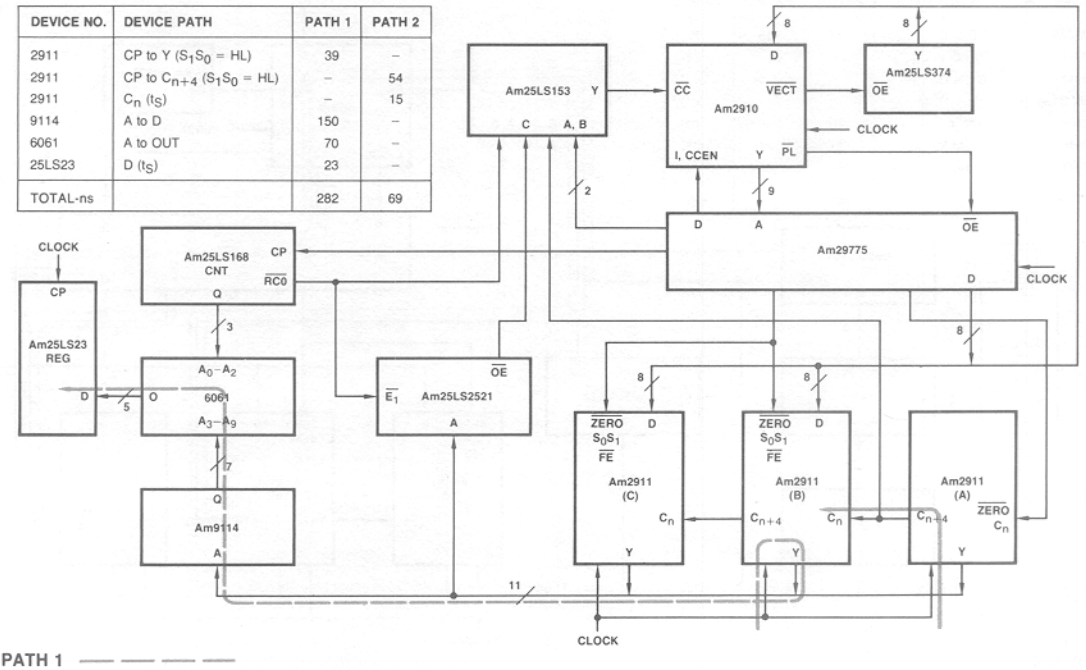


PATH 1 — — — —
PATH 2 ————
PATH 3 — - — - —

MPR-497

h)

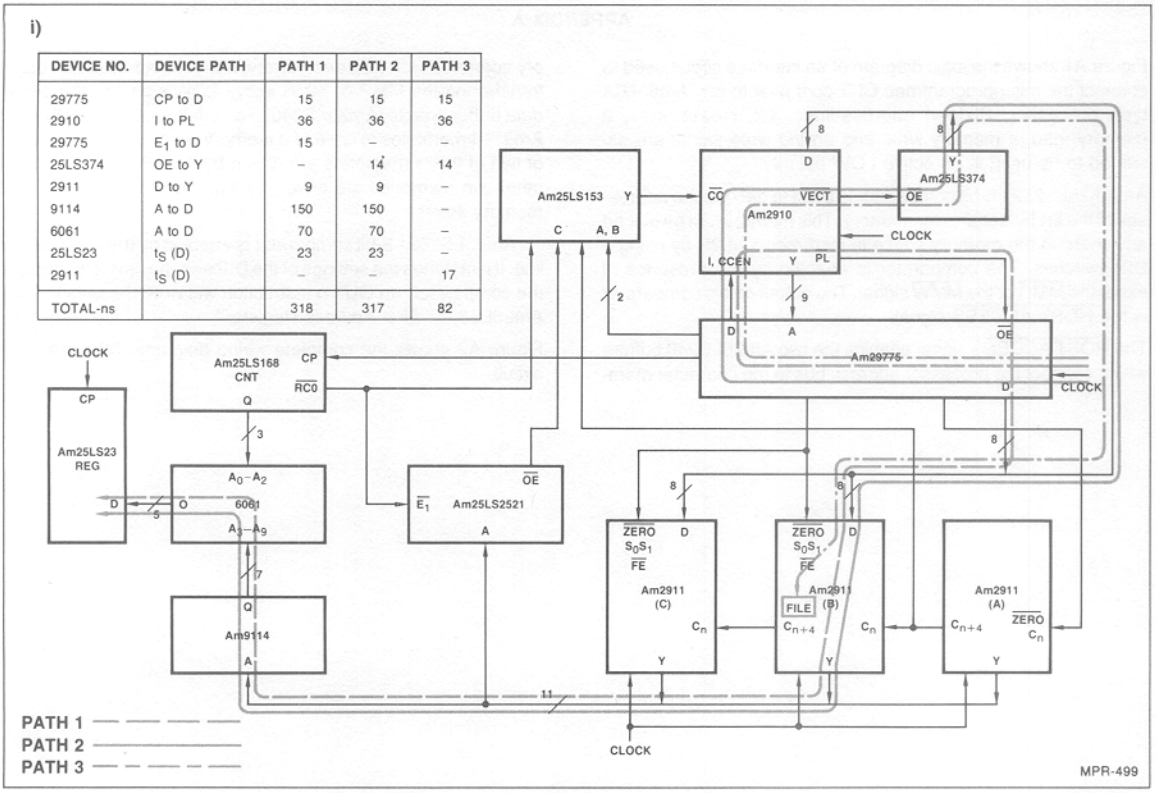
| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 |
|---|---|---|---|
| 2911 | CP to Y ($S_1S_0$ = HL) | 39 | – |
| 2911 | CP to $C_{n+4}$ ($S_1S_0$ = HL) | – | 54 |
| 2911 | $C_n$ ($t_S$) | – | 15 |
| 9114 | A to D | 150 | – |
| 6061 | A to OUT | 70 | – |
| 25LS23 | D ($t_S$) | 23 | – |
| TOTAL-ns | | 282 | 69 |



PATH 1 — — — —
PATH 2 ————

MPR-498

Figure 25. (Cont.)



| DEVICE NO. | DEVICE PATH | PATH 1 | PATH 2 | PATH 3 |
|---|---|---|---|---|
| 29775 | CP to D | 15 | 15 | 15 |
| 2910 | I to PL | 36 | 36 | 36 |
| 29775 | $E_1$ to D | 15 | – | – |
| 25LS374 | OE to Y | – | 14 | 14 |
| 2911 | D to Y | 9 | 9 | – |
| 9114 | A to D | 150 | 150 | – |
| 6061 | A to D | 70 | 70 | – |
| 25LS23 | $t_S$ (D) | 23 | 23 | – |
| 2911 | $t_S$ (D) | – | – | 17 |
| TOTAL-ns | | 318 | 317 | 82 |

Figure 25. (Cont.)

## SUMMARY

The Am2910 provides a powerful solution to the microprogram memory sequence control problem. The Am2910 is a fixed instruction set, 12-bit wide microprogram sequencer. In addition, the Am2909, Am2911, Am29811A and Am29803A provide another solution to the microprogram sequencing problem. These devices are bit slice oriented and provide more potential flexibility to the microprogram sequencing solution. All of these devices are particularly well suited for the high performance computer control unit and structured state machine designs using overlap fetch of the next microinstruction – also referred to as instruction-data-based microprogram architecture.

These Am2900 family microprogram control devices offer the highest performance LSI solution to the problem of microprogram control. They provide a number of conditional-branch source addresses as well as conditional jump-to-subroutine and conditional-return instructions. In addition, several techniques for timed and untimed looping are provided such that loops from one to several microinstructions can be executed. All of the devices described in this chapter are competitively priced and currently available. In addition, all of these devices are available with specifications guaranteed over the full commercial temperature range and power supply tolerance as well as the full military temperature range and power supply tolerance. All of these devices undergo 100% reliability assurance testing in compliance with MIL-STD-883.

## APPENDIX A

Figure A1 shows the logic diagram of an interface circuit used to connect the microprogrammed CRT controller to any Am9080A type processor. Sixteen address-lines, eight data lines, a memory-read, a memory write and an I/O write signal are assumed to be used in an active LOW polarity.

An Am25LS2521 8-bit comparator is used to decode the addresses of the 2K by 8 character memory. This memory can be placed anywhere in the memory space in increments of 2K by using 5 DIP-switches. The comparator is enabled by the presence of either the $\overline{\text{MMR}}$ or the $\overline{\text{MMW}}$ signal. The output of this comparator is the $\overline{\text{HOST ACCESS}}$ signal.

The $\overline{\text{HOST ACCESS}}$ signal enables the two Am25LS240 buffers which connect the processor address bus to the character memory address bus. It also enables one half of an Am25LS241 buffer transferring the $\overline{\text{MMR}}$ or $\overline{\text{MMR}}$ active LOW signal to the proper data buffer enable (Am25LS240's) and to the $\overline{\text{WE}}$ pins of the four Am9114 memories in case of a memory write operation. The $\overline{\text{CS}}$ of two of these memories are driven by $A_{10}$ while the $\overline{\text{CS}}$ of the other two memories are driven by $A_{10}$, thus forming a 2K by 8 memory space.

An Am25LS2521 8-bit comparator is enabled by the $\overline{\text{I/OW}}$ control line. If n matches the settings of the DIP switches at the B inputs of the comparator, an OUT n instruction will write the data into the Am25LS374 "First Address Register".

Figure A2 shows the complete wiring diagram of this interface circuit.

Note: Figure A2 is at back of the book.

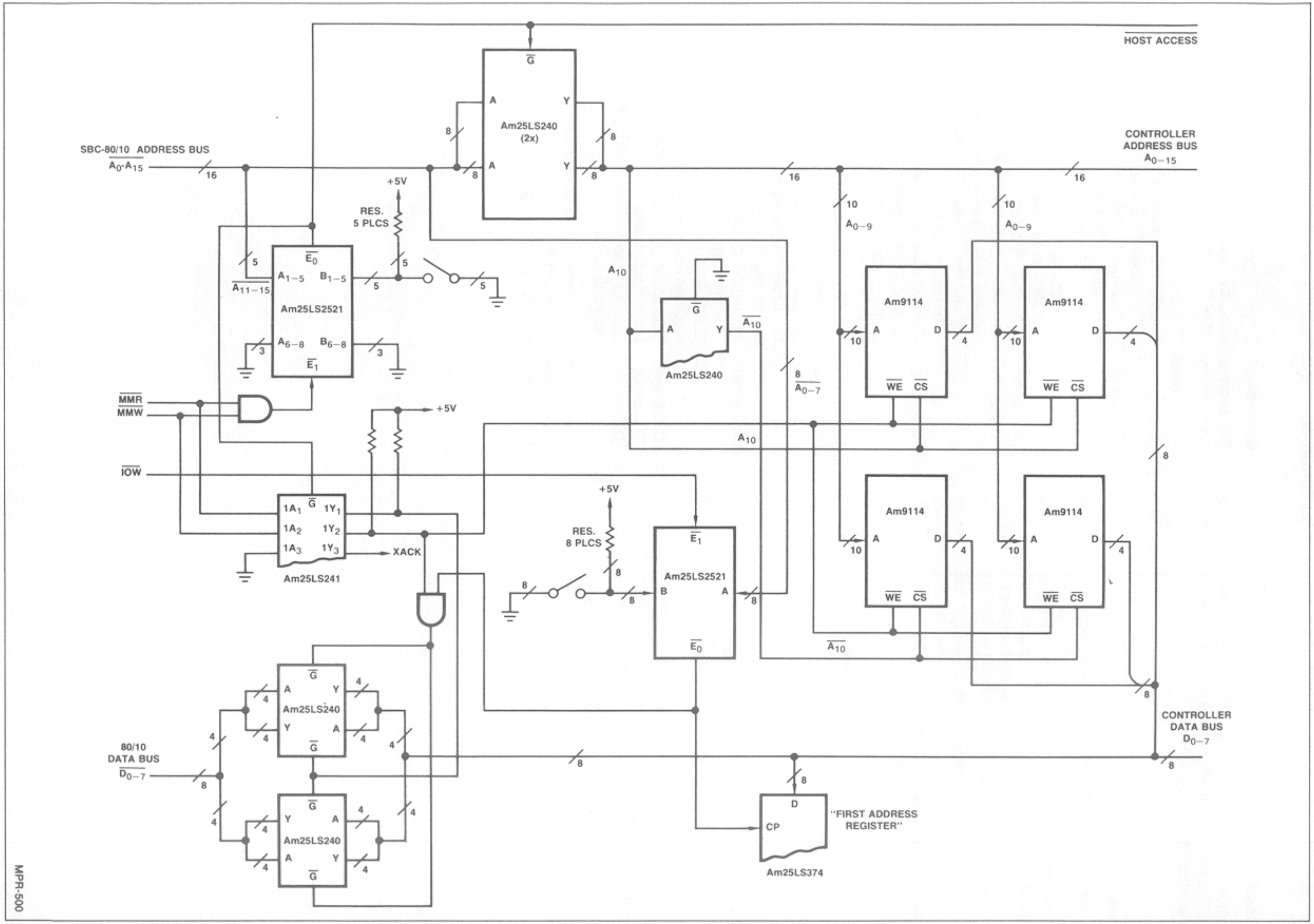


Figure A1. CRT Controller.

# APPENDIX B

## General

A software emulation of the CRT controller was written in BASIC-E and run on the System 29 support processor. Figure B1 is a printout of this program.

## Notations

For reference purposes, each clock pulse (CP) in the program is numbered. The clocks are character-rate clocks. A subscript "10" signifies that this variable belongs to the Am2910 (e.g. R10 = the contents of the Am2910 Register Counter) and similarly a subscript 11 signifies the Am2911 dependent variables (e.g. Y11 – the Y outputs of the two more significant Am2911s).

Usually the normal function names were used though for the active LOW functions the bar was deleted for simplicity. A 0 signifies always a LOW and 1 signifies HIGH. Other abbreviations used in the program:

MA = Microprogram Address (Y output of the Am2910)
CA = Character Address
PC = Program Counter (internal)
R = Register (internal)
F = File (internal)
SP = Stack Pointer (internal)
TENC = The Am25LS168 decade counter
L4B = The 4 least significant bits of CA (the Y outputs of the less significant Am2911
CN = Carry-in into the less significant Am2911
CN4 = Carry-out from the less significant Am2911
CN4 = Carry-in to the next significant Am2911
I10 = The Am2910 instruction
HB = Horizontal Blanking signal (active HIGH)
VB = Vertical Blanking signal (active HIGH)
CPM = Maximum Clock Pulse (at which the program stops)

## Description

The different groups and subroutines of the emulation program are as follows: (See Figure B1).

<1000 series: The microcode. Subroutine 50 is the Am25LS168 decade counter clocking routine. TENTH is the RCO output of this device.

1000 series: This is essentially the Am2910 emulation. Note the definition of the two functions FNFAIL and FNPASS at the beginning of the program, compare to the Am2910 instruction definitions in its data sheet.

2000 series: The Am25LS153 multiplexer emulation.

2500 series: The less significant Am2911 emulation. Note that the only input to this device is ZEROL. CN and the internal PC (called L4B) are controlled in the CLOCK Subroutine (4000 series).

3000 series: The two more significant Am2911's emulation, $S_0$ and $S_1$ are treated as a single number (ranging from 0 through 3) and denoted by S11.

4000 series: The Clocking routine.

5000 series: The main emulation routine. It includes the Am25LS2521 comparator routine and checks the Clock Pulse against CPM to determine end of run.

5500 series: Emulation parameter setup (initialization). The starting and ending CP numbers, MA, TENC, R10 and VECTOR (The "First Address Register") can be set.

6000 series: Sets up the print-out parameters

7000 series: Printout subroutine

9000 series: Sets the program mode: RUN, PRINT or QUIT (return to CP/M)

The emulation was exercised to evaluate fifteen different performance aspects of the CRT Controller. The results indicated that in all cases, the design operated as desired.

```
REM
REV=12
PRINT REV
9000    REM     HEADER
        PRINT
        PRINT
        PRINT " ******************************************************"
        PRINT
        PRINT
        PRINT "    A MICROPROGRAMMED CRT CONTROLLER EMULATION"
        PRINT
        PRINT
        PRINT " *****************************************************"
        PRINT
        PRINT
        PRINT "                        BY MOSHE M. SHAVIT"
        PRINT "                       ADVANCED MICRO DEVICES"
        PRINT "                          FEBRUARY 27, 1978"
        PRINT
        PRINT
REM
        DIM F10(6)
        DEF     FNFAIL=CCEN=0 AND CC=1
        DEF     FNPASS=CCEN=1  OR CC=0
REM
REM
REM     GOTO 6000       REM     PROGRAM PARAMETERS (REMOVED REV 6)
REM
REM     <--REV 6
REM
9100    PRINT
        PRINT
        PRINT
        INPUT "R-UN, P-RINT OR Q-UIT ";MODE$
        IF LEN(MODE$)=0 THEN GOTO 9100
        MODE=ASC(MODE$)-79
        IF MODE<1 OR MODE > 3 \
                THEN    PRINT MODE$; " IS INVALID":\
                        GOTO 9100
        ON MODE GOTO 9110,9120,9130
REM
9120    RETURN
REM
9130    REM     RUN
        PRINT
        INPUT "PUT RESULTS ON FILE (0 IF DIRECT PRINTOUT)= ";WFILE$
        PRINT "CP= ";CP;"MA= ";MA;"VECTOR= ";VECTOR;\
                "CPM= ";CPM;"ROW= ";24-R10
        INPUT "INITIALIZE (Y OR N; CP,MA=0 IF N)";S$
        IF S$="Y" \
                THEN GOSUB 5500 \       REM     INIT.
                ELSE    CP=0 : MA=0
        IF WFILE$="0" \
                THEN    GOTO 6010 \     REM DIRECT PRINTOUT
                ELSE    FILE WFILE$ : GOTO 5000 REM MAIN
REM
9110    REM     PRINT
        PRINT
        INPUT "GET RESULTS FROM FILE=";RFILE$
        FILE RFILE$
REM
6000    REM PRINT PARAMETERS
        PRINT
6010    PRINT "OUTPUT FORMATS:"
```

Figure B1.

```
        PRINT "         A=CP AND CA ONLY"
        PRINT "         B=CP,CA,HB,VB,MA"
        PRINT "         C=CP,CA,MA,TENC,R10"
        PRINT "         D=ALL"
        PRINT
        INPUT "FORMAT=";FORMAT$
        IF LEN(FORMAT$)=0 THEN GOTO 6010
        IF ASC(FORMAT$)<65 OR ASC(FORMAT$)>68 \
                THEN PRINT FORMAT$;" IS ILLEGAL" :\
                        GOTO 6010
        PRINT
REM
6020    REM
        IF WFILE$ NE "0" \
        THEN    CONTROL$="A" :\
                GOTO 6030
        PRINT "CLOCK CONTROL"
        PRINT "         A=CONTINOUS"
        PRINT "         B=STEP"
        INPUT "CONTROL=";CONTROL$
        IF LEN(CONTROL$)=0 THEN GOTO 6020
        IF ASC(CONTROL$)<65 OR ASC(CONTROL$)>66 \
                THEN    PRINT CONTROL$;" IS ILLEGAL" :\
                        GOTO 6020
        PRINT
REM
6030    PRINT "OUTPUT CONTROL"
        PRINT "         A=AT EACH CP"
        PRINT "         B=AT EVERY N-TH CP"
        PRINT "         C=MANUAL CONTROL"
        PRINT "         D=STARTING AT CPS AT EVERY CP"
        PRINT "         E=STARTING AT CPS AT EVERY N-TH CP"
        INPUT "OUTPUT=";OUTPUT$
        IF LEN(OUTPUT$)=0 THEN GOTO 6030
        IF ASC(OUTPUT$)<65 OR ASC(OUTPUT$)>69 \
                THEN    PRINT OUTPUT$;" IS ILLEGAL" :\
                        GOTO 6030
        O.CTL=ASC(OUTPUT$)-64
        ON O.CTL GOTO 6090,6032,6090,6034,6036
6032    INPUT "N=";N
        M=0
        GOTO 6090
6034    INPUT "CPS= ";CPS
        GOTO 6090
6036    INPUT "CPS= ";CPS
        INPUT "N= ";N
        M=0
        GOTO 6090
REM
6090    FORMAT = ASC(FORMAT$)-64
        ON FORMAT GOSUB 6190,6300,6200,6100
        IF WFILE$="0" THEN GOTO 5000    REM     MAIN
REM
6900    PRINT
        IF END #1 THEN 6910
        FOR I=1 TO 2 STEP 0     REM DO UNTIL END OF FILE
        READ #1; CP,R10,F1,SP10,PC10,CA,MUX,CC,CCEN,MA,TENC,\
                CN4,F11,HB,VB
        F10(SP10)=F1
        GOSUB 7000      REM PRINT
        GOSUB 5200      REM ESCAPE      (REV 7)
        IF S=155 THEN PRINT:PRINT "ABORTED AT ";CP : GOTO 6910
        NEXT I
```

**Figure B1 (Cont.)**

```
REM
6910    CLOSE 1
        OUT 100,12       REM     PRINTER PAGE EJECT (REV 7)
        GOTO 9100
REM
6100    PRINT
        PRINT "CP","R10","F10","SP10","PC10"
        PRINT "CA","MUX","CC","CCEN","MA"
        PRINT "TENC","CN4","F11","HB","VB"
        PRINT
6190    RETURN
REM
6200    PRINT
        PRINT "CLOCK","CHAR.ADDR","2910 REG.","LINE CNTR.","NEXT MA"
        RETURN
REM
6300    PRINT
        PRINT "CLOCK","CHAR.ADDR","H.BLANKING","V.BLANKING","NEXT MA"
        RETURN
REM
REM
7000    REM     PRINT SUBROUTINE
        ON O.CTL GOTO 7010,7005,7002,7003,7004
REM
7002    INPUT "OUTPUT (Y OR N)";S$
        IF S$="Y" \
                THEN    GOTO 7010 \
                ELSE    RETURN
REM
7003    IF CP<CPS THEN RETURN ELSE GOTO 7010
REM
7004    IF CP<CPS THEN RETURN ELSE GOTO 7005
REM
7005    M=M+1
        IF M=N THEN M=0 : GOTO 7010 ELSE RETURN
REM
7010    ON FORMAT GOTO 7100,7200,7300,7400
REM
7100    PRINT "CP= ";CP,"CA= ";CA
        RETURN
REM
7200    IF HB=0 THEN HB$="L" ELSE HB$="         H"
        IF VB=0 THEN VB$="L" ELSE VB$="         H"
        PRINT CP,CA,HB$,VB$,MA
        RETURN
REM
7300    PRINT
        PRINT CP,CA,R10,TENC,MA
        RETURN
REM
7400    PRINT
        PRINT CP,R10,F10(SP10),SP10,PC10
        PRINT CA,MUX,CC,CCEN,MA
        PRINT TENC,CN4,F11,HB,VB
        RETURN
REM
REM
5000    REM     MAIN ROUTINE
        REM
        GOSUB 4000       REM     CLOCK
        REM     FETCH MICROCODE
        ON MA+1 GOSUB 30,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22
        GOSUB 2500       REM     2911L
        GOSUB 3000       REM     2911H
```

**Figure B1 (Cont.)**

```
        CA=Y11*16+L4B REM    CHARACTER ADDRESS
                      REM    COMPARATOR NEXT
        IF Y11=120 AND TENTH=0 \    REM REV 8
               THEN   COMP=0 \
               ELSE   COMP=1
        GOSUB 2000    REM    MUX
        GOSUB 1000    REM    2910
REM     REV 6
        IF WFILE$="0" THEN GOSUB 7000 \ REM DIRECT PRINTOUT
               ELSE PRINT #1;CP,R10,F10(SP10),SP10,PC10,CA,MUX,\
                      CC,CCEN,MA,TENC,CN4,F11,HB,VB
        IF CONTROL$="B" THEN INPUT S$   REM    SINGLE STEP
        REM    CHECK END OF RUN
        GOSUB 5200    REM    ESCAPE  (REV 7)
        IF S=155 THEN PRINT:PRINT "ABORTED AT ";CP : GOTO 5100
        IF CP<CPM THEN GOTO 5000    REM    REPEAT MAIN
REM
5100    IF WFILE$ NE "0" THEN CLOSE (1)
        OUT 100,12     REM PRINTER PAGE EJECT (REV 7)
        GOTO 9100
REM
REM     5200 SUB REV 7
5200    REM    ESCAPE SUBROUTINE
        S=INP(97)
        S=INT(S/2)
        S=S/2-INT(S/2)
        IF S NE 0 THEN S = INP(96)
        RETURN
REM
5500    REM    INITIALIZATION
        PRINT
        SP10=1
        PRINT "MA= ";MA
5505    INPUT "NEW MA (Y OR N)";S$
        IF S$="N" THEN GOTO 5510
        INPUT "MA=(0<=MA<22)";MA
        MA=INT(MA)
        IF MA<0 OR MA>21 \
               THEN   PRINT MA;" IS ILLEGAL" :\
                      GOTO 5505
        IF MA=0 THEN TENC=0 : HB=1 : TENTH=1
REM
5510    PRINT
        PRINT "VECTOR= ";VECTOR
5515    INPUT "NEW VECTOR (Y OR N)";S$
        IF S$="N" THEN GOTO 5520
        INPUT "VECTOR=(0<=VECTOR<120)";VECTOR
        VECTOR=INT(VECTOR)
        IF VECTOR<0 OR VECTOR>119 \
               THEN   PRINT VECTOR;" IS ILLEGAL" :\
                      GOTO 5515
REM
5520    PRINT
        PRINT "CP= ";CP
        INPUT "NEW CP (Y OR N) ";S$
        IF S$="N" THEN GOTO 5530
5525    INPUT "CP(>=0)= ";CP
        CP=INT(CP)
        IF CP<0 THEN PRINT CP;" IS ILLEGAL" : GOTO 5525
REM
5530    PRINT
        PRINT "CPM= ";CPM
5535    INPUT "NEW CPM (Y OR N)";S$
        IF S$="N" THEN GOTO 5540
```

**Figure B1. (Cont.)**

```
          INPUT "CPM=(CP+1<CPM)";CPM
          CPM=INT(CPM)
          IF CPM<CP+1 THEN PRINT CPM;" IS ILLEGAL";"CP= ";CP :GOTO 5535
REM
5540      PRINT
          PRINT "TENC= ";TENC
          IF MA=0 THEN GOTO 5550
5545      INPUT "NEW TENC (Y OR N)";S$
          IF S$="N" THEN GOTO 5550
          INPUT "TENC=(0<=TENC<10)";TENC
          TENC=INT(TENC)
          IF TENC<0 OR TENC>9 \
                    THEN     PRINT TENC;" IS ILLEGAL" :\
                             GOTO 5545
          IF TENC=9 THEN TENTH=0 ELSE TENTH=1
REM
5550      PRINT
          PRINT "R10= ";R10
5555      INPUT "NEW R10 (Y OR N)";S$
          IF S$="N" THEN GOTO 5560
          INPUT "R10 (0<=R10<25)=";R10
          R10=INT(R10)
          IF R10<0 OR R10>24 THEN PRINT R10;" IS ILLEGAL" : GOTO 5555
REM
5560      REM
          RETURN
REM
REM
REM
30        I10=6
          CCEN=0
          MUX=3
          S11=3
          FE=0
          ZEROH=1
          ZEROL=0
          CN=0
          HB=1     REM      REV 2
          VB=0
          PL=0
          RETURN
REM
2         I10=12
          S11=0
          FE=1
          ZEROH=1
          ZEROL=0
          CN=0
          HB=1     REM      REV 2
          VB=0
          PL=23
          RETURN
REM
3         I10=14
          S11=2
          FE=1
          ZEROH=1
          ZEROL=0
          CN=1
          HB=1     REM      REV 2
          VB=0
          RETURN
REM
4         I10=3
```

**Figure B1 (Cont.)**

```
        CCEN=0
        MUX=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        HB=0
        VB=0
        PL=3
        RETURN
REM
5       I10=3
        CCEN=0
        MUX=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        HB=0
        VB=0
        PL=4
        RETURN
REM
6       I10=3
        CCEN=0
        MUX=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        HB=0
        VB=0
        PL=5
        RETURN
REM
7       I10=3
        CCEN=0
        MUX=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        HB=0
        VB=0
        PL=6
        RETURN
REM
8       I10=3
        CCEN=0
        MUX=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        HB=0
        VB=0
        PL=7
        RETURN
```

```
REM
9       I10=1
        CCEN=0
        MUX=0
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50 REM TENC
        VB=0
        PL=12
        RETURN
REM
10      I10=1
        CCEN=0
        MUX=2
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=0
        PL=21
        RETURN
REM
11      I10=3
        CCEN=0
        MUX=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=0
        PL=10
        RETURN
REM
12      I10=3
        CCEN=1
        S11=0
        FE=1
        ZEROH=1
        GOSUB 50
        VB=0
        PL=2
        RETURN
REM
13      I10=9
        S11=0
        FE=0    REM      REV 5
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=0
        PL=20
        RETURN
REM
14      I10=6
        CCEN=0
        MUX=3
        S11=3
```

**Figure B1 (Cont.)**

```
        FE=0    REM      REV 10
        ZEROH=1
        ZEROL=0
        GOSUB 50
        VB=1
        RETURN
REM
15      I10=12
        S11=0   REM      REV 10
        FE=1    REM      REV 10
        ZEROH=1
REM     ZEROH=1          REM      REMOVED REV 10
        GOSUB 50
        VB=1
        PL=119
        RETURN
REM
16      I10=4
        CCEN=0
        MUX=3
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=1
        RETURN
REM
17      I10=3
        CCEN=0
        MUX=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=1
        PL=16
        RETURN
REM
18      I10=8
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=1
        RETURN
REM
19      I10=12
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=1
        PL=23
        RETURN
REM
20      I10=10
```

Figure B1 (Cont.)

```
        CCEN=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=1
        RETURN
REM
21      I10=10
        CCEN=1
        S11=0
        FE=1
        ZEROH=1
        ZEROL=1
        CN=1
        GOSUB 50
        VB=0
        RETURN
REM
22      I10=10
        CCEN=1
        FE=0            REM     REV 9
        ZEROH=0
        ZEROL=1         REM     REV 9
        CN=1
        GOSUB 50
        VB=0
        RETURN
REM
50      REM TEN-LINE-COUNTER CLOCKING SUBROUTINE
        IF HB=1 THEN RETURN
        HB=1
        TENC=TENC+1
        IF TENC=9 THEN TENTH=0 ELSE TENTH=1
        IF TENC=10 THEN TENC=0
        RETURN
REM     PUSH AND POP SUBROUTINES REMOVED REV 3
1000    REM     2910 INSTRUCTIONS SUBROUTINE
        ON I10+1 GOTO 1100,1110,1120,1130,1140,1150,1160,1170,1180, \
        1190,1200,1210,1220,1230,1240,1250
REM
1100    REM     JZ
        MA=0    REM     2910 Y
        SP10=0  REM     2910 STACK POINTER (=0 REV 3)
        RETURN
REM
1110    REM     CJS
        IF FNFAIL \
                THEN    MA=PC10 \
                ELSE    MA=PL :\
                        PUSH=1          REM     REV 3
        RETURN
REM
1120    REM     JMAP
        PRINT "JMAP NOT PROGRAMMED"
        RETURN
REM
1130    REM     CJP
        IF FNFAIL \
                THEN    MA=PC10 \
                ELSE    MA=PL
        RETURN
```

Figure B1 (Cont.)

```
REM
1140    REM     PUSH
        IF FNPASS THEN R10=PL   REM     LOAD COUNTER
        MA=PC10
        PUSH=1          REM     REV 3
        RETURN
REM
1150    REM     JSRP
        PRINT "JSRP NOT PROGRAMMED"
        RETURN
REM
1160    REM     CJV
        IF FNFAIL \
                THEN    MA=PC10 \
                ELSE    MA=VECTOR
        RETURN
REM
1170    REM     JRP
        IF FNFAIL \
                THEN MA=R10 \
                ELSE MA=PL
        RETURN
REM
1180    REM     RFCT
        IF R10=0 \
                THEN    MA=PC10 :\
                        POP=1 \
                ELSE    MA=F10(SP10) :\
                        R10=R10-1
        RETURN
REM
1190    REM     RPCT
        IF R10=0 \
                THEN    MA=PC10 \
                ELSE    MA=PL :\
                        R10=R10-1
        RETURN
REM
1200    REM     CRTN
        IF FNFAIL \
                THEN    MA=PC10 \
                ELSE    MA=F10(SP10) :\
                        POP=1           REM     REV 3
        RETURN
REM
1210    REM     CJPP

        PRINT "CJPP NOT PROGRAMMED"
        RETURN
REM
1220    REM     LDCT
        R10=PL
        MA=PC10
        RETURN
REM
1230    REM     LOOP
        IF FNFAIL \
                THEN    MA=F10(SP10) \
                ELSE    MA=PC10 :\
                        POP=1           REM REV 3
        RETURN
REM
1240    REM     CONT
        MA=PC10
        RETURN
```

Figure B1. (Cont.)

```
REM
1250    REM     TWB
        PRINT 'TWB NOT PROGRAMMED'
        RETURN
REM
REM
2000    REM     MUX SUBROUTINE
        ON MUX+1 GOTO 2100,2200,2300,2400
REM
2100    IF TENTH=0 \
                THEN    CC=0 \
                ELSE    CC=1
        RETURN
REM
2200    IF CN4=0 \
                THEN    CC=0 \
                ELSE    CC=1
        RETURN
REM
2300    IF COMP=0 \
                THEN    CC=0 \
                ELSE    CC=1
        RETURN
REM
2400    CC=1
        RETURN
REM
REM
2500    REM     LEAST SIGNIFICANT 2911 (2911L) SUBROUTINE
        IF ZEROL=0 THEN L4B=0
        RETURN
REM
REM
REM
REM
3000    REM     MORE SIGNIFICANT 2911S (2911H) SUBROUTINE
        ON S11+1 GOSUB 3100,3200,3300,3400
        IF ZEROH=0 THEN Y11=0
        RETURN
REM
3100    Y11=PC11
        RETURN
REM
3200    Y11=R11
        RETURN
REM
3300    Y11=F11
        RETURN
REM
3400    IF I10=6 \
                THEN    Y11=VECTOR \
                ELSE    Y11=PL
        RETURN
REM
REM
4000    REM     CLOCK SUBROUTINE
REM     PC10=MA+1       REMOVED REV 4
        IF CN=1 THEN L4B=L4B+1
        IF L4B>15 THEN L4B=0 : CN4=1 ELSE CN4=0
        IF CN4=1 \
                THEN    PC11=Y11+1 \
                ELSE    PC11=Y11
        IF FE=0 THEN F11=PC11
REM     <--REV 3
```

Figure B1 (Cont.)

```
        IF PUSH=1 \
                THEN    SP10=SP10+1 :\
                        F10(SP10)=PC10 :\
                        PUSH=0
        IF SP10>4 \
                THEN    PRINT "2910 STACK FULL " :\
                        SP10=3
        IF POP=1 \
                THEN    SP10=SP10-1 :\
                        POP=0
        IF SP10<0 \
                THEN    PRINT "POP EMPTY FILE? ";CP :\
                        SP10=0
REM     REV 3 -->
        PC10=MA+1       REM     REV 4
        CP=CP+1
        RETURN
REM
REM
```

Figure B1 (Cont.)

## APPENDIX C

A simple circuit was designed to accommodate five different display formats and also to comply with the European 50Hz TV standard. Figure C1 is the circuit diagram of this additional circuit.

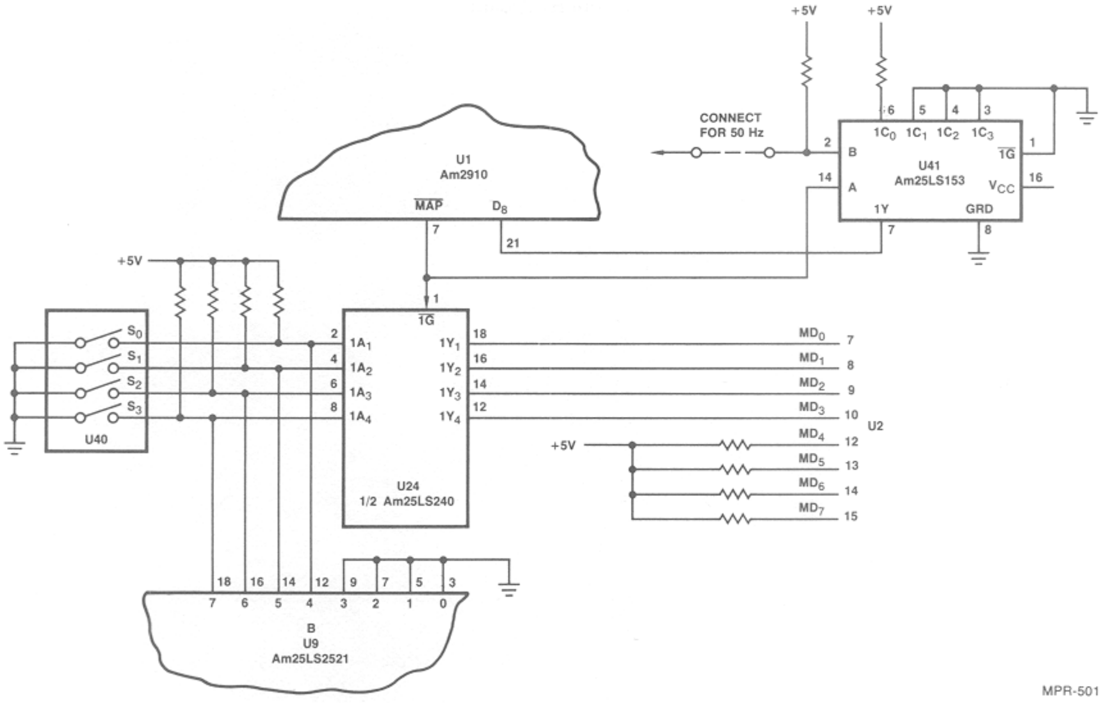
The following parameters change when the format is changed:

1) The number of characters/line.
2) The number of lines/frame.
3) The number of characters to display (i.e., the address of the last character).
4) The line frequency and therefore the dot frequency.

The number of characters/line is counted by the least significant Am2911 sequencer via the microcode. Therefore, the microcode can be changed to change the number of characters/line. The number of lines/frame is counted by a constant, loaded into the Am2910 internal counter by the microcode. The microcode can be changed to vary the number of lines/frame.

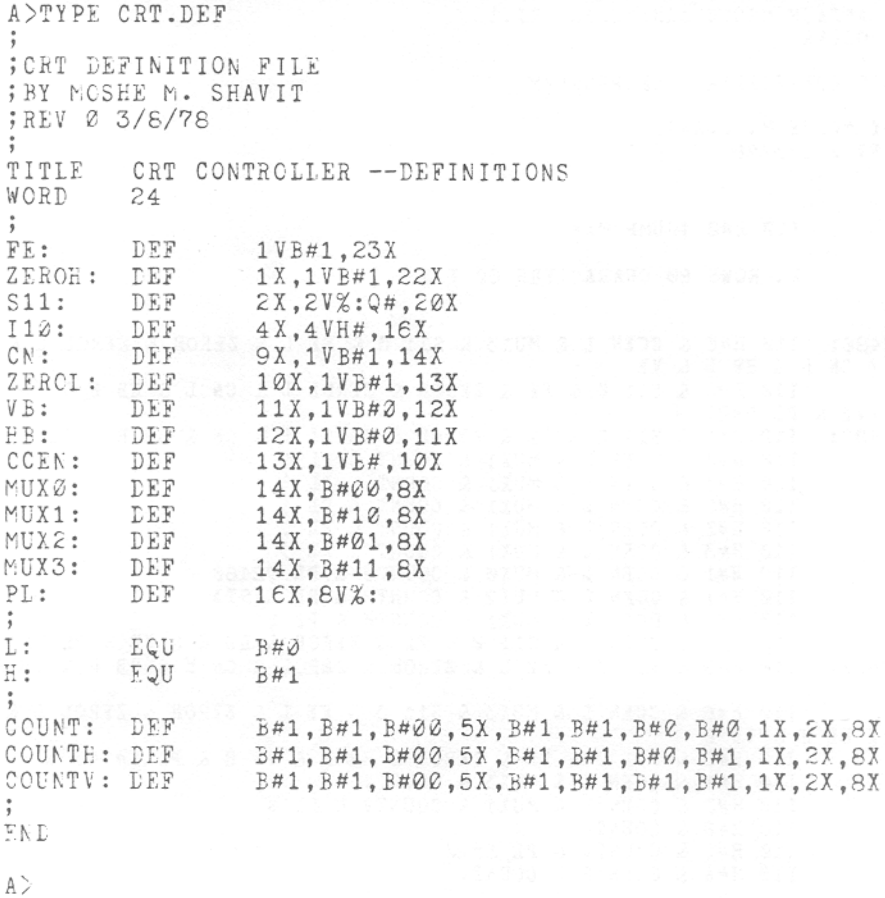
The scan is reinitialized to zero when the last address +1 is attained. $U_9$ (Am25LS2521) detects this address by comparing bits $A_4$ through $A_{10}$ of the character address bus to a constant supplied to its B inputs. A table listing these constants is shown in Figure C1. By setting the DIP switches according to that table, the character scan will reinitialize correctly. The same constant is routed through one half of an Am25LS240 (U24) to the internal data bus. At microprogram address zero, a JUMP MAP instruction enables these outputs thereby putting a starting address on the bus according to the table in Figure C1.

The microprogram is shown on Figure C2.



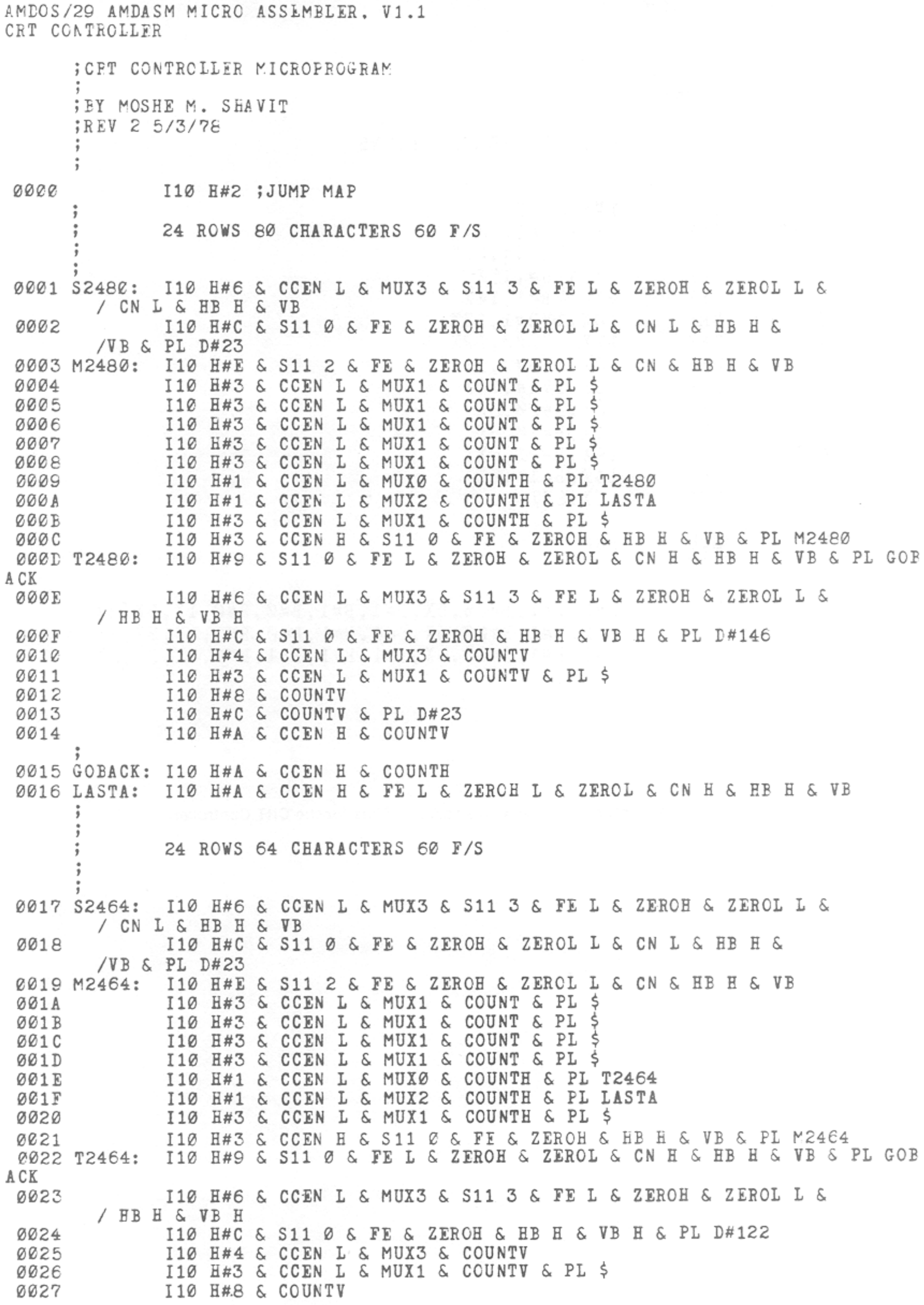
| FORMAT | LAST CHAR. ADD. +1 | COMPARE AT LAST ADD/16 | | $S_3$ $S_2$ $S_1$ $S_0$ | MAP ADDRESS | DOT FREQ. (MHz) |
|---|---|---|---|---|---|---|
| 24 x 80 | 1920 | 120D | 78H | H H H H | 0F0 | 10.86624 |
| 24 x 64 | 1536 | 96D | 60H | H H L L | 0F3 | 9.09216 |
| 24 x 32 | 768 | 48D | 30H | L H H L | 0F9 | 5.544 |
| 16 x 32 | 512 | 32D | 20H | L H L L | 0FB | 5.376 |
| 16 x 16 | 256 | 16D | 10H | L L H L | 0FD | 3.65568 |
| | | | | $A_{10}$ $A_9$ $A_8$ $A_7$ | | |

Figure C1.



```
A>TYPE CRT.DEF
;
;CRT DEFINITION FILE
;BY MOSHE M. SHAVIT
;REV Ø 3/8/78
;
TITLE   CRT CONTROLLER --DEFINITIONS
WORD    24
;
FE:     DEF     1VB#1,23X
ZEROH:  DEF     1X,1VB#1,22X
S11:    DEF     2X,2V%:Q#,2ØX
I1Ø:    DEF     4X,4VH#,16X
CN:     DEF     9X,1VB#1,14X
ZEROL:  DEF     1ØX,1VB#1,13X
VB:     DEF     11X,1VB#Ø,12X
HB:     DEF     12X,1VB#Ø,11X
CCEN:   DEF     13X,1VB#,1ØX
MUXØ:   DEF     14X,B#ØØ,8X
MUX1:   DEF     14X,B#1Ø,8X
MUX2:   DEF     14X,B#Ø1,8X
MUX3:   DEF     14X,B#11,8X
PL:     DEF     16X,8V%:
;
L:      EQU     B#Ø
H:      EQU     B#1
;
COUNT:  DEF     B#1,B#1,B#ØØ,5X,B#1,B#1,B#Ø,B#Ø,1X,2X,8X
COUNTH: DEF     B#1,B#1,B#ØØ,5X,B#1,B#1,B#Ø,B#1,1X,2X,8X
COUNTV: DEF     B#1,B#1,B#ØØ,5X,B#1,B#1,B#1,B#1,1X,2X,8X
;
END

A>
```

**Figure C2. AMDASM Definition and Assembly Files for the CRT Controller.**



```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
CRT CONTROLLER

      ;CRT CONTROLLER MICROPROGRAM
      ;
      ;BY MOSHE M. SHAVIT
      ;REV 2 5/3/78
      ;
      ;

 ØØØØ          I1Ø H#2 ;JUMP MAP
      ;
      ;        24 ROWS 8Ø CHARACTERS 6Ø F/S
      ;
      ;
 ØØØ1 S248Ø:   I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / CN L & HB H & VB
 ØØØ2          I1Ø H#C & S11 Ø & FE & ZEROH & ZEROL L & CN L & HB H &
        /VB & PL D#23
 ØØØ3 M248Ø:   I1Ø H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 ØØØ4          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØØ5          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØØ6          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØØ7          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØØ8          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØØ9          I1Ø H#1 & CCEN L & MUXØ & COUNTH & PL T248Ø
 ØØØA          I1Ø H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 ØØØB          I1Ø H#3 & CCEN L & MUX1 & COUNTH & PL $
 ØØØC          I1Ø H#3 & CCEN H & S11 Ø & FE & ZEROH & HB H & VB & PL M248Ø
 ØØØD T248Ø:   I1Ø H#9 & S11 Ø & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 ØØØE          I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / HB H & VB H
 ØØØF          I1Ø H#C & S11 Ø & FE & ZEROH & HB H & VB H & PL D#146
 ØØ1Ø          I1Ø H#4 & CCEN L & MUX3 & COUNTV
 ØØ11          I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 ØØ12          I1Ø H#8 & COUNTV
 ØØ13          I1Ø H#C & COUNTV & PL D#23
 ØØ14          I1Ø H#A & CCEN H & COUNTV
      ;
 ØØ15 GOBACK:  I1Ø H#A & CCEN H & COUNTH
 ØØ16 LASTA:   I1Ø H#A & CCEN H & FE L & ZEROH L & ZEROL & CN H & HB H & VB
      ;
      ;
      ;        24 ROWS 64 CHARACTERS 6Ø F/S
      ;
      ;
 ØØ17 S2464:   I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / CN L & HB H & VB
 ØØ18          I1Ø H#C & S11 Ø & FE & ZEROH & ZEROL L & CN L & HB H &
        /VB & PL D#23
 ØØ19 M2464:   I1Ø H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 ØØ1A          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØ1B          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØ1C          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØ1D          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 ØØ1E          I1Ø H#1 & CCEN L & MUXØ & COUNTH & PL T2464
 ØØ1F          I1Ø H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 ØØ2Ø          I1Ø H#3 & CCEN L & MUX1 & COUNTH & PL $
 ØØ21          I1Ø H#3 & CCEN H & S11 Ø & FE & ZEROH & HB H & VB & PL M2464
 ØØ22 T2464:   I1Ø H#9 & S11 Ø & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 ØØ23          I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / HB H & VB H
 ØØ24          I1Ø H#C & S11 Ø & FE & ZEROH & HB H & VB H & PL D#122
 ØØ25          I1Ø H#4 & CCEN L & MUX3 & COUNTV
 ØØ26          I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 ØØ27          I1Ø H#8 & COUNTV
```

Figure C2 (Cont.)

```
 0028         I10 H#C & COUNTV & PL D#23
 0029         I10 H#A & CCEN H & COUNTV
      ;
      ;
      ;
      ;       24 ROWS 32 CHARACTERS 60 F/S
      ;
      ;
 002A S2432:  I10 H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / CN L & HB H & VB
 002B         I10 H#C & S11 0 & FE & ZEROH & ZEROL L & CN L & HB H &
        /VB & PL D#23
 002C M2432:  I10 H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 002D         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 002E         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 002F         I10 H#1 & CCEN L & MUX0 & COUNTH & PL T2432
 0030         I10 H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 0031         I10 H#3 & CCEN L & MUX1 & COUNTH & PL $
 0032         I10 H#3 & CCEN H & S11 0 & FE & ZEROH & HB H & VB & PL M2432
 0033 T2432:  I10 H#9 & S11 0 & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 0034         I10 H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / HB H & VB H
 0035         I10 H#C & S11 0 & FE & ZEROH & HB H & VB H & PL D#74
 0036         I10 H#4 & CCEN L & MUX3 & COUNTV
 0037         I10 H#3 & CCEN L & MUX1 & COUNTV & PL $
 0038         I10 H#8 & COUNTV
 0039         I10 H#C & COUNTV & PL D#23
 003A         I10 H#A & CCEN H & COUNTV
      ;
      ;
      ;       16 ROWS 32 CHARACTERS 60 F/S
      ;
      ;
 003B S1632:  I10 H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / CN L & HB H & VB
 003C         I10 H#C & S11 0 & FE & ZEROH & ZEROL L & CN L & HB H &
        /VB & PL D#15
 003D M1632:  I10 H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 003E         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 003F         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 0040         I10 H#1 & CCEN L & MUX0 & COUNTH & PL T1632
 0041         I10 H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 0042         I10 H#3 & CCEN L & MUX1 & COUNTH & PL $
 0043         I10 H#3 & CCEN H & S11 0 & FE & ZEROH & HB H & VB & PL M1632
 0044 T1632:  I10 H#9 & S11 0 & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 0045         I10 H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / HB H & VB H
 0046         I10 H#C & S11 0 & FE & ZEROH & HB H & VB H & PL D#250
 0047         I10 H#4 & CCEN L & MUX3 & COUNTV
 0048         I10 H#3 & CCEN L & MUX1 & COUNTV & PL $
 0049         I10 H#8 & COUNTV
 004A         I10 H#C & COUNTV & PL D#48
 004B         I10 H#4 & CCEN L & MUX3 & COUNTV
 004C         I10 H#3 & CCEN L & MUX1 & COUNTV & PL $
 004D         I10 H#8 & COUNTV
 004E         I10 H#C & COUNTV & PL D#15
 004F         I10 H#A & CCEN H & COUNTV
      ;
      ;
      ;       16 ROWS 16 CHARACTERS 60 F/S
      ;
      ;
```

**Figure C2 (Cont.)**

```
 0050 S1616:  I10 H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / CN L & HB H & VB
 0051         I10 H#C & S11 0 & FE & ZEROH & ZEROL L & CN L & HB H &
        /VB & PL D#15
 0052 M1616:  I10 H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 0053         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 0054         I10 H#1 & CCEN L & MUX0 & COUNTH & PL T1616
 0055         I10 H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 0056         I10 H#3 & CCEN L & MUX1 & COUNTH & PL $
 0057         I10 H#3 & CCEN H & S11 0 & FE & ZEROH & HB H & VB & PL M1616
 0058 T1616:  I10 H#9 & S11 0 & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 0059         I10 H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / HB H & VB H
 005A         I10 H#C & S11 0 & FE & ZEROH & HB H & VB H & PL D#203
 005B         I10 H#4 & CCEN L & MUX3 & COUNTV
 005C         I10 H#3 & CCEN L & MUX1 & COUNTV & PL $
 005D         I10 H#8 & COUNTV
 005E         I10 H#C & COUNTV & PL D#15
 005F         I10 H#A & CCEN H & COUNTV
      ;
 00F0         ORG     H#0F0   ;24*80
 00F0         I10 H#3 & CCEN H & PL S2480
      ;
      ;
 00F3         ORG     H#0F3   ;24*64
 00F3         I10 H#3 & CCEN H & PL S2464
      ;
      ;
 00F9         ORG     H#0F9   ;24*32
 00F9         I10 H#3 & CCEN H & PL S2432
      ;
      ;
 00FB         ORG     H#0FB   ;16*32
 00FB         I10 H#3 & CCEN H & PL S1632
      ;
      ;
 00FD         ORG     H#0FD   ;16*16
 00FD         I10 .H#3 & CCEN H & PL S1616
      ;
      ;
      ;
      ;50 F/S ROUTINES
      ;
 0100         ORG     H#100
      ;
      ;       24 ROWS 80 CHARACTERS 50 F/S
      ;
      ;
 0100 S2480E: I10 H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
        / CN L & HB H & VB
 0101         I10 H#C & S11 0 & FE & ZEROH & ZEROL L & CN L & HB H &
        /VB & PL D#23
 0102 M2480E: I10 H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 0103         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 0104         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 0105         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 0106         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 0107         I10 H#3 & CCEN L & MUX1 & COUNT & PL $
 0108         I10 H#1 & CCEN L & MUX0 & COUNTH & PL T2480E
 0109         I10 H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 010A         I10 H#3 & CCEN L & MUX1 & COUNTH & PL $
 010B         I10 H#3 & CCEN H & S11 0 & FE & ZEROH & HB H & VB & PL M2480E
 010C T2460E: I10 H#9 & S11 0 & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
```

Figure C2 (Cont.)

```
 Ø1ØD          I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / HB H & VB H
 Ø1ØE          I1Ø H#C & S11 Ø & FE & ZEROH & HB H & VB H & PL D#2ØØ   ;ITERATES
2Ø1 TIMES
 Ø1ØF          I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø11Ø          I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø111          I1Ø H#8 & COUNTV
      ;
 Ø112          I1Ø H#C & COUNTV & PL D#239
 Ø113          I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø114          I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø115          I1Ø H#8 & COUNTV
      ;
 Ø116          I1Ø H#C & COUNTV & PL D#23
 Ø117          I1Ø H#A & CCEN H & COUNTV
      ;
      ;
      ;        24 ROWS 64 CHARACTERS 5Ø F/S
      ;
      ;
      ;
 Ø118 S2464E: I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / CN L & HB H & VB
 Ø119          I1Ø H#C & S11 Ø & FE & ZEROH & ZEROL L & CN L & HB H &
       /VB & PL D#23
 Ø11A M2464E: I1Ø H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 Ø11B          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø11C          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø11D          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø11E          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø11F          I1Ø H#1 & CCEN L & MUXØ & COUNTH & PL T2464E
 Ø12Ø          I1Ø H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 Ø121          I1Ø H#3 & CCEN L & MUX1 & COUNTH & PL $
 Ø122          I1Ø H#3 & CCEN H & S11 Ø & FE & ZEROH & HB H & VB & PL M2464E
 Ø123 T2464E: I1Ø H#9 & S11 Ø & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 Ø124          I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / HB H & VB H
 Ø125          I1Ø H#C & S11 Ø & FE & ZEROH & HB H & VB H & PL D#2ØØ
 Ø126          I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø127          I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø128          I1Ø H#8 & COUNTV
      ;
 Ø129          I1Ø H#C & COUNTV & PL D#167      ;369
 Ø12A          I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø12B          I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø12C          I1Ø H#8 & COUNTV
      ;
 Ø12D          I1Ø H#C & COUNTV & PL D#23
 Ø12E          I1Ø H#A & CCEN H & COUNTV
      ;
      ;
      ;
      ;        24 ROWS 32 CHARACTERS 5Ø F/S
      ;
      ;
 Ø12F S2432E: I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / CN L & HB H & VB
 Ø13Ø          I1Ø H#C & S11 Ø & FE & ZEROH & ZEROL L & CN L & HB H &
       /VB & PL D#23
 Ø131 M2432E: I1Ø H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 Ø132          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø133          I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø134          I1Ø H#1 & CCEN L & MUXØ & COUNTH & PL T2432E
 Ø135          I1Ø H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 Ø136          I1Ø H#3 & CCEN L & MUX1 & COUNTH & PL $
```

Figure C2 (Cont.)

```
 Ø137           I1Ø H#3 & CCEN H & S11 Ø & FE & ZEROH & HB H & VB & PL M2432E
 Ø138 T2432E: I1Ø H#9 & S11 Ø & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 Ø139           I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / HB H & VB H
 Ø13A           I1Ø H#C & S11 Ø & FE & ZEROH & HB H & VB H & PL D#224
 Ø13B           I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø13C           I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø13D           I1Ø H#8 & COUNTV
 Ø13E           I1Ø H#C & COUNTV & PL D#23
 Ø13F           I1Ø H#A & CCEN H & COUNTV
      ;
      ;
      ;         16 ROWS 32 CHARACTERS 5Ø F/S
      ;
      ;
 Ø14Ø S1632E: I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / CN L & HB H & VB
 Ø141           I1Ø H#C & S11 Ø & FE & ZEROH & ZEROL L & CN L & HB H &
       /VB & PL D#15
 Ø142 M1632E: I1Ø H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 Ø143           I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø144           I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø145           I1Ø H#1 & CCEN L & MUXØ & COUNTH & PL T1632E
 Ø146           I1Ø H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 Ø147           I1Ø H#3 & CCEN L & MUX1 & COUNTH & PL $
 Ø148           I1Ø H#3 & CCEN H & S11 Ø & FE & ZEROH & HB H & VB & PL M1632E
 Ø149 T1632E: I1Ø H#9 & S11 Ø & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 Ø14A           I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / HB H & VB H
 Ø14B           I1Ø H#C & S11 Ø & FE & ZEROH & HB H & VB H & PL D#25Ø
 Ø14C           I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø14D           I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø14E           I1Ø H#8 & COUNTV
 Ø14F           I1Ø H#C & COUNTV & PL D#223     ;475
 Ø15Ø           I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø151           I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø152           I1Ø H#8 & COUNTV
 Ø153           I1Ø H#C & COUNTV & PL D#15
 Ø154           I1Ø H#A & CCEN H & COUNTV
      ;
      ;
      ;         16 ROWS 16 CHARACTERS 5Ø F/S
      ;
      ;
 Ø155 S1616E: I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / CN L & HB H & VB
 Ø156           I1Ø H#C & S11 Ø & FE & ZEROH & ZEROL L & CN L & HB H &
       /VB & PL D#15
 Ø157 M1616E: I1Ø H#E & S11 2 & FE & ZEROH & ZEROL L & CN & HB H & VB
 Ø158           I1Ø H#3 & CCEN L & MUX1 & COUNT & PL $
 Ø159           I1Ø H#1 & CCEN L & MUXØ & COUNTH & PL T1616E
 Ø15A           I1Ø H#1 & CCEN L & MUX2 & COUNTH & PL LASTA
 Ø15B           I1Ø H#3 & CCEN L & MUX1 & COUNTH & PL $
 Ø15C           I1Ø H#3 & CCEN H & S11 Ø & FE & ZEROH & HB H & VB & PL M1616E
 Ø15D T1616E: I1Ø H#9 & S11 Ø & FE L & ZEROH & ZEROL & CN H & HB H & VB & PL GOB
ACK
 Ø15E           I1Ø H#6 & CCEN L & MUX3 & S11 3 & FE L & ZEROH & ZEROL L &
       / HB H & VB H
 Ø15F           I1Ø H#C & S11 Ø & FE & ZEROH & HB H & VB H & PL D#2ØØ
 Ø16Ø           I1Ø H#4 & CCEN L & MUX3 & COUNTV
 Ø161           I1Ø H#3 & CCEN L & MUX1 & COUNTV & PL $
 Ø162           I1Ø H#8 & COUNTV
```

Figure C2 (Cont.)

```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
CRT CONTROLLER

          ;
 0163              I10 H#C & COUNTV & PL D#121      ;323
 0164              I10 H#4 & CCEN L & MUX3 & COUNTV
 0165              I10 H#3 & CCEN L & MUX1 & COUNTV & PL $
 0166              I10 H#8 & COUNTV
          ;
 0167              I10 H#C & COUNTV & PL D#15
 0168              I10 H#A & CCEN H & COUNTV
          ;
 01F0              ORG      H#1F0    ;24*80
 01F0              I10 H#3 & CCEN H & PL S2480E
          ;
          ;
 01F3              ORG      H#1F3    ;24*64
 01F3              I10 H#3 & CCEN H & PL S2464E
          ;
          ;
 01F9              ORG      H#1F9    ;24*32
 01F9              I10 H#3 & CCEN H & PL S2432E
          ;
          ;
 01FB              ORG      H#1FB    ;16*32
 01FB              I10 H#3 & CCEN H & PL S1632E
          ;
          ;
 01FD              ORG      H#1FD    ;16*16
 01FD              I10 H#3 & CCEN H & PL S1616E
          ;
          ;
          END
```

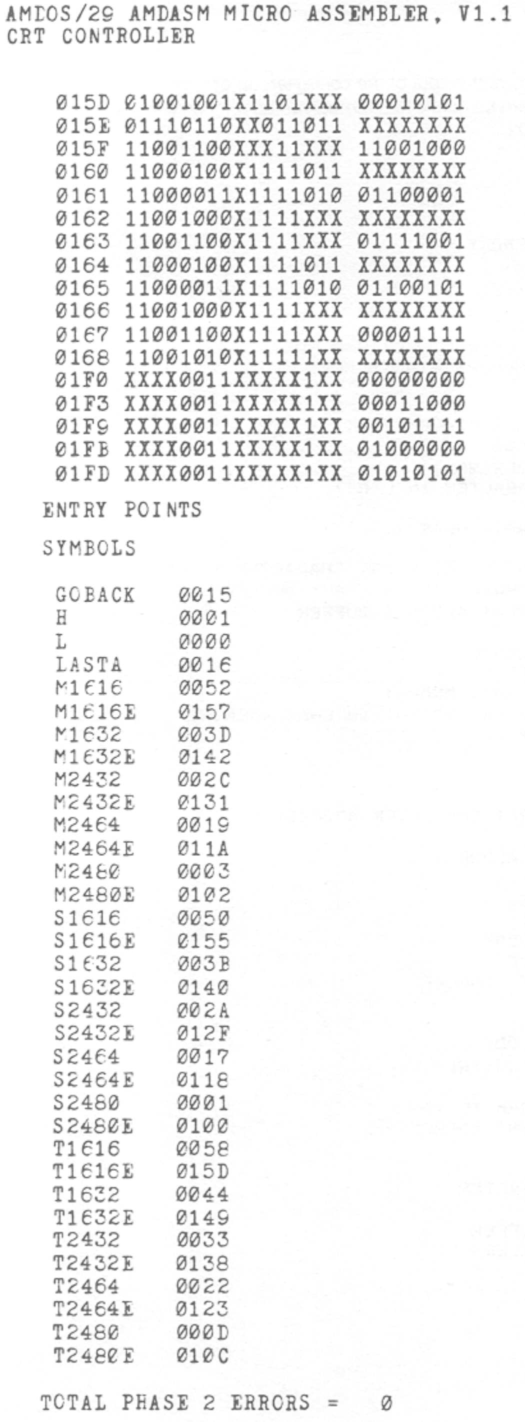
```
    0000 XXXX0010XXXXXXXX XXXXXXXX        0022 01001001X1101XXX 00010101
    0001 01110110X0001011 XXXXXXXX        0023 01110110XX011011 XXXXXXXX
    0002 11001100X0001XXX 00010111        0024 11001100XXX11XXX 01111010
    0003 11101110X1001XXX XXXXXXXX        0025 11000100X1111011 XXXXXXXX
    0004 11000011X1100010 00000100        0026 11000011X1111010 00100110
    0005 11000011X1100010 00000101        0027 11001000X1111XXX XXXXXXXX
    0006 11000011X1100010 00000110        0028 11001100X1111XXX 00010111
    0007 11000011X1100010 00000111        0029 11001010X11111XX XXXXXXXX
    0008 11000011X1100010 00001000        002A 01110110X0001011 XXXXXXXX
    0009 11000001X1101000 00001101        002B 11001100X0001XXX 00010111
    000A 11000001X1101001 00010110        002C 11101110X1001XXX XXXXXXXX
    000B 11000011X1101010 00001011        002D 11000011X1100010 00101101
    000C 11000011XXX011XX 00000011        002E 11000011X1100010 00101110
    000D 01001001X1101XXX 00010101        002F 11000001X1101000 00110011
    000E 01110110XX011011 XXXXXXXX        0030 11000001X1101001 00010110
    000F 11001100XXX11XXX 10010010        0031 11000011X1101010 00110001
    0010 11000100X1111011 XXXXXXXX        0032 11000011XXX011XX 00101100
    0011 11000011X1111010 00010001        0033 01001001X1101XXX 00010101
    0012 11001000X1111XXX XXXXXXXX        0034 01110110XX011011 XXXXXXXX
    0013 11001100X1111XXX 00010111        0035 11001100XXX11XXX 01001010
    0014 11001010X11111XX XXXXXXXX        0036 11000100X1111011 XXXXXXXX
    0015 11001010X11011XX XXXXXXXX        0037 11000011X1111010 00110111
    0016 00XX1010X11011XX XXXXXXXX        0038 11001000X1111XXX XXXXXXXX
    0017 01110110X0001011 XXXXXXXX        0039 11001100X1111XXX 00010111
    0018 11001100X0001XXX 00010111        003A 11001010X11111XX XXXXXXXX
    0019 11101110X1001XXX XXXXXXXX        003B 01110110X0001011 XXXXXXXX
    001A 11000011X1100010 00011010        003C 11001100X0001XXX 00001111
    001B 11000011X1100010 00011011        003D 11101110X1001XXX XXXXXXXX
    001C 11000011X1100010 00011100        003E 11000011X1100010 00111110
    001D 11000011X1100010 00011101        003F 11000011X1100010 00111111
    001E 11000001X1101000 00100010        0040 11000001X1101000 01000100
    001F 11000001X1101001 00010110        0041 11000001X1101001 00010110
    0020 11000011X1101010 00100000        0042 11000011X1101010 01000010
    0021 11000011XXX011XX 00011001        0043 11000011XXX011XX 00111101
```

Figure C2 (Cont.)

```
0044 01001001X1101XXX 00010101
0045 01110110XX011011 XXXXXXXX
0046 11001100XXX11XXX 11111010
0047 11000100X1111011 XXXXXXXX
0048 11000011X1111010 01001000
0049 11001000X1111XXX XXXXXXXX
004A 11001100X1111XXX 00110000
004B 11000100X1111011 XXXXXXXX
004C 11000011X1111010 01001100
004D 11001000X1111XXX XXXXXXXX
004E 11001100X1111XXX 00001111
004F 11001010X11111XX XXXXXXXX
0050 01110110X0001011 XXXXXXXX
0051 11001100X0001XXX 00001111
0052 11101110X1001XXX XXXXXXXX
0053 11000011X1100010 01010011
0054 11000001X1101000 01011000
0055 11000001X1101001 00010110
0056 11000011X1101010 01010110
0057 11000011XXX011XX 01010010
0058 01001001X1101XXX 00010101
0059 01110110XX011011 XXXXXXXX
005A 11001100XXX11XXX 11001011
005B 11000100X1111011 XXXXXXXX
005C 11000011X1111010 01011100
005D 11001000X1111XXX XXXXXXXX
005E 11001100X1111XXX 00001111
005F 11001010X11111XX XXXXXXXX
00F0 XXXX0011XXXXX1XX 00000001
00F3 XXXX0011XXXXX1XX 00010111
00F9 XXXX0011XXXXX1XX 00101010
00FB XXXX0011XXXXX1XX 00111011
00FD XXXX0011XXXXX1XX 01010000
0100 01110110X0001011 XXXXXXXX
0101 11001100X0001XXX 00010111
0102 11101110X1001XXX XXXXXXXX
0103 11000011X1100010 00000011
0104 11000011X1100010 00000100
0105 11000011X1100010 00000101
0106 11000011X1100010 00000110
0107 11000011X1100010 00000111
0108 11000001X1101000 00001100
0109 11000001X1101001 00010110
010A 11000011X1101010 00001010
010B 11000011XXX011XX 00000010
010C 01001001X1101XXX 00010101
010D 01110110XX011011 XXXXXXXX
010E 11001100XXX11XXX 11001000
010F 11000100X1111011 XXXXXXXX
0110 11000011X1111010 00010000
0111 11001000X1111XXX XXXXXXXX
0112 11001100X1111XXX 11101111
0113 11000100X1111011 XXXXXXXX
0114 11000011X1111010 00010100
0115 11001000X1111XXX XXXXXXXX
0116 11001100X1111XXX 00010111
0117 11001010X11111XX XXXXXXXX
0118 01110110X0001011 XXXXXXXX
0119 11001100X0001XXX 00010111
011A 11101110X1001XXX XXXXXXXX
011B 11000011X1100010 00011011
011C 11000011X1100010 00011100
011D 11000011X1100010 00011101
```

```
011E 11000011X1100010 00011110
011F 11000001X1101000 00100011
0120 11000001X1101001 00010110
0121 11000011X1101010 00100001
0122 11000011XXX011XX 00011010
0123 01001001X1101XXX 00010101
0124 01110110XX011011 XXXXXXXX
0125 11001100XXX11XXX 11001000
0126 11000100X1111011 XXXXXXXX
0127 11000011X1111010 00100111
0128 11001000X1111XXX XXXXXXXX
0129 11001100X1111XXX 10100111
012A 11000100X1111011 XXXXXXXX
012B 11000011X1111010 00101011
012C 11001000X1111XXX XXXXXXXX
012D 11001100X1111XXX 00010111
012E 11001010X11111XX XXXXXXXX
012F 01110110X0001011 XXXXXXXX
0130 11001100X0001XXX 00010111
0131 11101110X1001XXX XXXXXXXX
0132 11000011X1100010 00110010
0133 11000011X1100010 00110011
0134 11000001X1101000 00111000
0135 11000001X1101001 00010110
0136 11000011X1101010 00110110
0137 11000011XXX011XX 00110001
0138 01001001X1101XXX 00010101
0139 01110110XX011011 XXXXXXXX
013A 11001100XXX11XXX 11100000
013B 11000100X1111011 XXXXXXXX
013C 11000011X1111010 00111100
013D 11001000X1111XXX XXXXXXXX
013E 11001100X1111XXX 00010111
013F 11001010X11111XX XXXXXXXX
0140 01110110X0001011 XXXXXXXX
0141 11001100X0001XXX 00001111
0142 11101110X1001XXX XXXXXXXX
0143 11000011X1100010 01000011
0144 11000011X1100010 01000100
0145 11000001X1101000 01001001
0146 11000001X1101001 00010110
0147 11000011X1101010 01000111
0148 11000011XXX011XX 01000010
0149 01001001X1101XXX 00010101
014A 01110110XX011011 XXXXXXXX
014B 11001100XXX11XXX 11111010
014C 11000100X1111011 XXXXXXXX
014D 11000011X1111010 01001101
014E 11001000X1111XXX XXXXXXXX
014F 11001100X1111XXX 11011111
0150 11000100X1111011 XXXXXXXX
0151 11000011X1111010 01010001
0152 11001000X1111XXX XXXXXXXX
0153 11001100X1111XXX 00001111
0154 11001010X11111XX XXXXXXXX
0155 01110110X0001011 XXXXXXXX
0156 11001100X0001XXX 00001111
0157 11101110X1001XXX XXXXXXXX
0158 11000011X1100010 01011000
0159 11000001X1101000 01011101
015A 11000001X1101001 00010110
015B 11000011X1101010 01011011
015C 11000011XXX011XX 01010111
```

Figure C2 (Cont.)

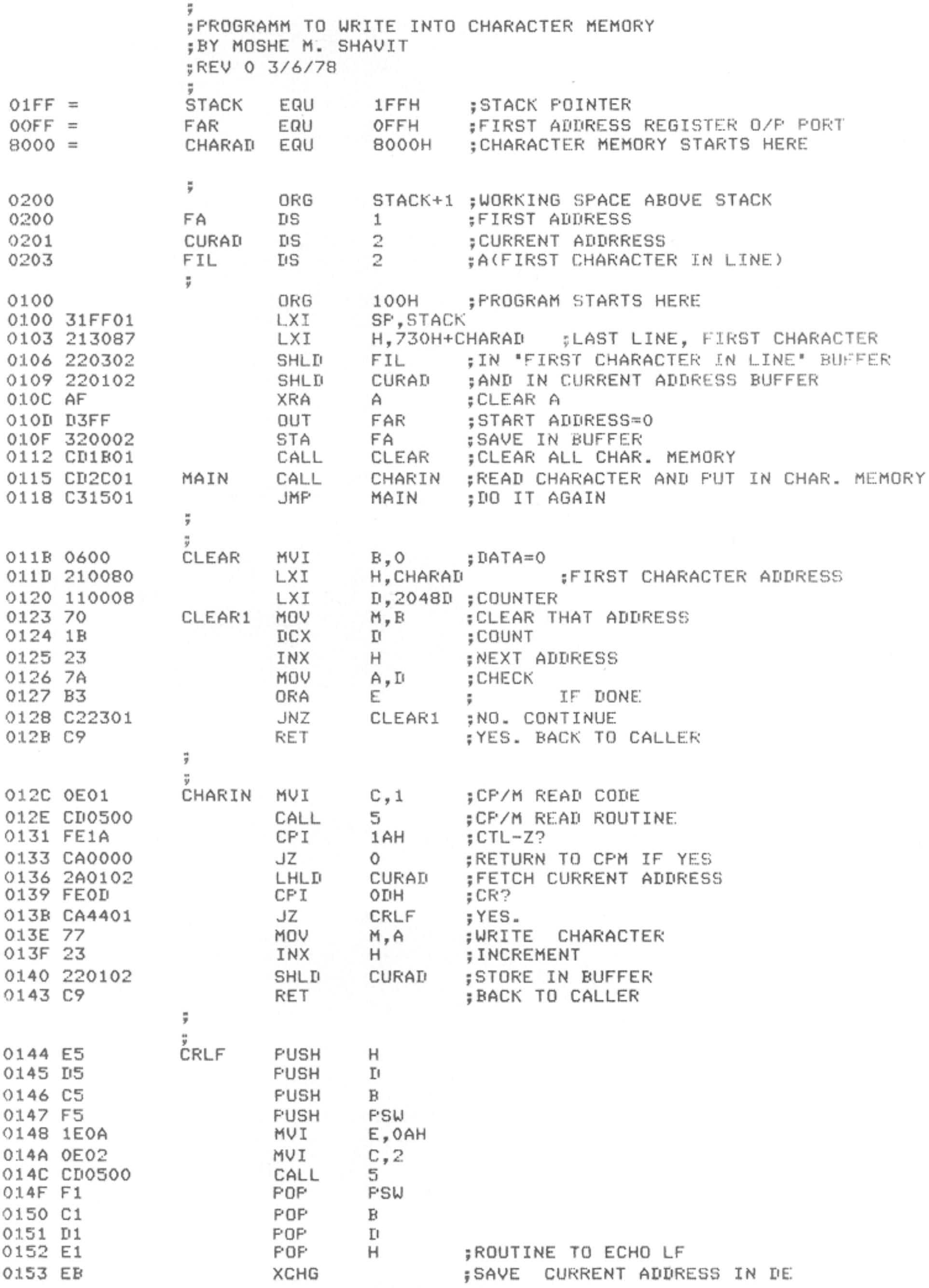


```
AMDOS/29 AMDASM MICRO ASSEMBLER, V1.1
CRT CONTROLLER

    015D 01001001X1101XXX 00010101
    015E 01110110XX011011 XXXXXXXX
    015F 11001100XXX11XXX 11001000
    0160 11000100X1111011 XXXXXXXX
    0161 11000011X1111010 01100001
    0162 11001000X1111XXX XXXXXXXX
    0163 11001100X1111XXX 01111001
    0164 11000100X1111011 XXXXXXXX
    0165 11000011X1111010 01100101
    0166 11001000X1111XXX XXXXXXXX
    0167 11001100X1111XXX 00001111
    0168 11001010X11111XX XXXXXXXX
    01F0 XXXX0011XXXXX1XX 00000000
    01F3 XXXX0011XXXXX1XX 00011000
    01F9 XXXX0011XXXXX1XX 00101111
    01FB XXXX0011XXXXX1XX 01000000
    01FD XXXX0011XXXXX1XX 01010101

   ENTRY POINTS

   SYMBOLS

    GOBACK    0015
    H         0001
    L         0000
    LASTA     0016
    M1616     0052
    M1616E    0157
    M1632     003D
    M1632E    0142
    M2432     002C
    M2432E    0131
    M2464     0019
    M2464E    011A
    M2480     0003
    M2480E    0102
    S1616     0050
    S1616E    0155
    S1632     003B
    S1632E    0140
    S2432     002A
    S2432E    012F
    S2464     0017
    S2464E    0118
    S2480     0001
    S2480E    0100
    T1616     0058
    T1616E    015D
    T1632     0044
    T1632E    0149
    T2432     0033
    T2432E    0138
    T2464     0022
    T2464E    0123
    T2480     000D
    T2480E    010C

   TOTAL PHASE 2 ERRORS =    0
```

Figure C2 (Cont.)

## APPENDIX D

The Microprogrammed CRT Controller was built on a System 29 universal card and exercised by the System 29 support processor. An Am9080A program was written to fill the character memory. Figure D1 is the listing of this program. In order to observe the correct output of the controller, an oscilloscope or CRT monitor can be connected through an adaptation circuit shown in Figure D2.

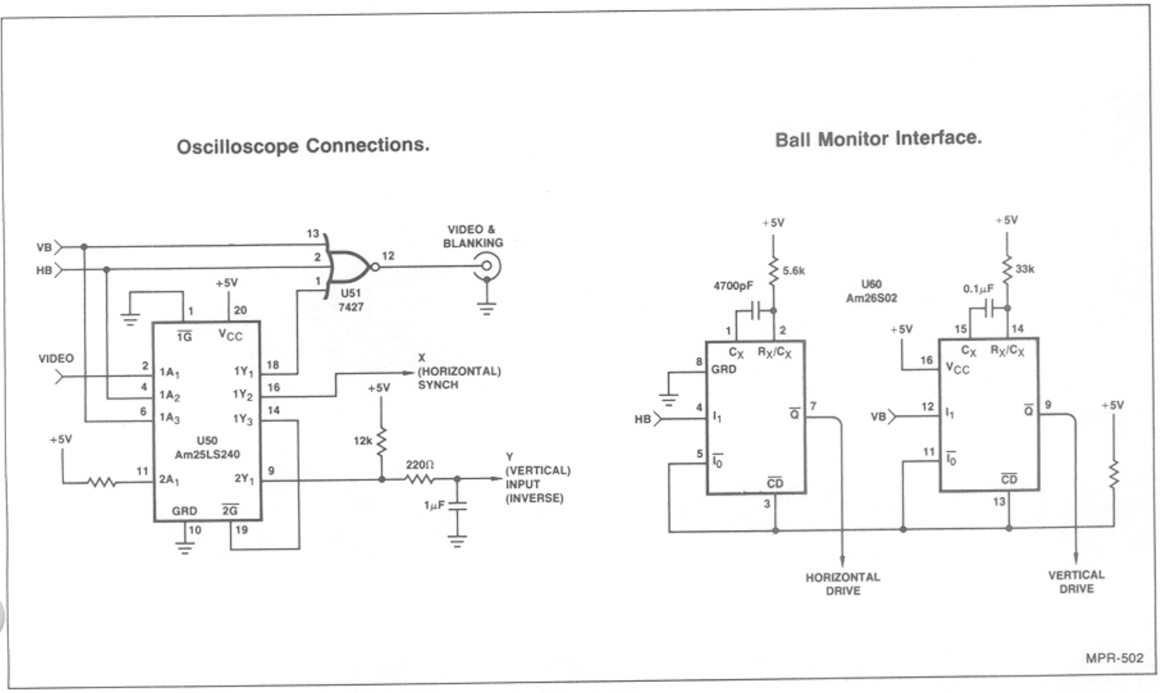


```
                ;
                ;PROGRAMM TO WRITE INTO CHARACTER MEMORY
                ;BY MOSHE M. SHAVIT
                ;REV 0 3/6/78
                ;
01FF =          STACK   EQU     1FFH    ;STACK POINTER
00FF =          FAR     EQU     0FFH    ;FIRST ADDRESS REGISTER O/P PORT
8000 =          CHARAD  EQU     8000H   ;CHARACTER MEMORY STARTS HERE


                ;
0200                    ORG     STACK+1 ;WORKING SPACE ABOVE STACK
0200            FA      DS      1       ;FIRST ADDRESS
0201            CURAD   DS      2       ;CURRENT ADDRRESS
0203            FIL     DS      2       ;A(FIRST CHARACTER IN LINE)
                ;
0100                    ORG     100H    ;PROGRAM STARTS HERE
0100 31FF01             LXI     SP,STACK
0103 213087             LXI     H,730H+CHARAD   ;LAST LINE, FIRST CHARACTER
0106 220302             SHLD    FIL     ;IN "FIRST CHARACTER IN LINE" BUFFER
0109 220102             SHLD    CURAD   ;AND IN CURRENT ADDRESS BUFFER
010C AF                 XRA     A       ;CLEAR A
010D D3FF               OUT     FAR     ;START ADDRESS=0
010F 320002             STA     FA      ;SAVE IN BUFFER
0112 CD1B01             CALL    CLEAR   ;CLEAR ALL CHAR. MEMORY
0115 CD2C01     MAIN    CALL    CHARIN  ;READ CHARACTER AND PUT IN CHAR. MEMORY
0118 C31501             JMP     MAIN    ;DO IT AGAIN
                ;
                ;
011B 0600       CLEAR   MVI     B,0     ;DATA=0
011D 210080             LXI     H,CHARAD        ;FIRST CHARACTER ADDRESS
0120 110008             LXI     D,2048D ;COUNTER
0123 70         CLEAR1  MOV     M,B     ;CLEAR THAT ADDRESS
0124 1B                 DCX     D       ;COUNT
0125 23                 INX     H       ;NEXT ADDRESS
0126 7A                 MOV     A,D     ;CHECK
0127 B3                 ORA     E       ;       IF DONE
0128 C22301             JNZ     CLEAR1  ;NO. CONTINUE
012B C9                 RET             ;YES. BACK TO CALLER
                ;
                ;
012C 0E01       CHARIN  MVI     C,1     ;CP/M READ CODE
012E CD0500             CALL    5       ;CP/M READ ROUTINE
0131 FE1A               CPI     1AH     ;CTL-Z?
0133 CA0000             JZ      0       ;RETURN TO CPM IF YES
0136 2A0102             LHLD    CURAD   ;FETCH CURRENT ADDRESS
0139 FE0D               CPI     0DH     ;CR?
013B CA4401             JZ      CRLF    ;YES.
013E 77                 MOV     M,A     ;WRITE  CHARACTER
013F 23                 INX     H       ;INCREMENT
0140 220102             SHLD    CURAD   ;STORE IN BUFFER
0143 C9                 RET             ;BACK TO CALLER
                ;
                ;
0144 E5         CRLF    PUSH    H
0145 D5                 PUSH    D
0146 C5                 PUSH    B
0147 F5                 PUSH    PSW
0148 1E0A               MVI     E,0AH
014A 0E02               MVI     C,2
014C CD0500             CALL    5
014F F1                 POP     PSW
0150 C1                 POP     B
0151 D1                 POP     D
0152 E1                 POP     H       ;ROUTINE TO ECHO LF
0153 EB                 XCHG            ;SAVE  CURRENT ADDRESS IN DE
```

**Figure D1**

```
0154 015000          LXI    B,80D     ;80 CHARACTERS/LINE
0157 2A0302          LHLD   FIL       ;FETCH FIRST CH. IN LINE ADDRESS
015A 09              DAD    B         ;HL= A(NEXT LINE'S FIRST CH. ADD.)
015B EB              XCHG             ;HL=CURRENT ADDR.,DE=A(NEXT LINE FIRST CH. ADDR)
015C 0600            MVI    B,0       ;DATA=0
015E 7C      CRLF2   MOV    A,H       ;MORE SIGNIFICANT CURRENT ADDRESS
015F BA              CMP    D         ;=NEXT LINE FIRST ADDRESS?
0160 C26801          JNZ    CRLF3     ;NO
0163 7D              MOV    A,L       ;LESS SIGNIFICANT CURRENT ADDRESS
0164 BB              CMP    E         ;IS CURRENT LINE FULL?
0165 CA6D01          JZ     CRLF4     ;YES
0168 70      CRLF3   MOV    M,B       ;STORE 0 AT THAT ADDRESS
0169 23              INX    H         ;INCREMENT ADDRESS
016A C35E01          JMP    CRLF2     ;GO CHECK AGAIN
016D 7C      CRLF4   MOV    A,H       ;MORE SIGNIFICANT PART OF ADDRESS
016E E607            ANI    7         ;ONLY 3 LESS SIGNIFICANT BITS
0170 FE07            CPI    7         ;LAST LINE PASSED?
0172 C27E01          JNZ    CRLF5     ;NOT YET
0175 7D              MOV    A,L       ;LESS SIGNIFICANT BYTE OF ADDRESS
0176 FE80            CPI    80H       ;ARE WE AT 780H=1920D?
0178 C27E01          JNZ    CRLF5     ;NOT YET, SKIP
017B 210080          LXI    H,CHARAD        ;YES, START WRITING AT BEGINNING OF CH. MEM.
017E 220302  CRLF5   SHLD   FIL       ;STORE IN FIRST CH. IN LINE BUFFER
0181 220102          SHLD   CURAD     ;AND IN CURRENT ADDRESS BUFFER
0184 3A0002          LDA    FA        ;FETCH FIRST VISIIBLE CHARACTER ADDRESS
0187 C605            ADI    5         ;SCROLL
0189 FE78            CPI    120D      ;TOO MUCH?
018B CC9401          CZ     CRLF0     ;YES
018E 320002          STA    FA        ;STORE IN FIRST ADDRESS BUFFER
0191 D3FF            OUT    FAR       ;LOAD REGISTER
0193 C9              RET              ;RETURN TO CALLER
             ;
             ;
0194 AF      CRLF0   XRA    A         ;FIRST ADDRESS=0
0195 C9              RET
             ;
             ;
```

Figure D1 (Cont.)


Oscilloscope Connections.

Ball Monitor Interface.

Figure D2.

### MULTIPLEXER SELECT

| $R_{20}$ | $R_{19}$ | $R_{18}$ | $R_{17}$ | SELECT |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | TEST 0 |
| 0 | 0 | 0 | 1 | TEST 1 |
| 0 | 0 | 1 | 0 | TEST 2 |
|  | ⋮ |  |  | ⋮ |
| 1 | 1 | 1 | 1 | TEST 15 |

### POLARITY CONTROL

| $R_{16}$ | OUTPUT |
|---|---|
| 0 | COMPLEMENT OF TEST |
| 1 | TRUE TEST |

### NEXT ADDRESS CONTROL

| $R_{15}$ | $R_{14}$ | $R_{13}$ | $R_{12}$ | FUNCTION |
|---|---|---|---|---|
| X | X | X | X | NEXT INSTRUCTION |

### MACHINE INSTRUCTION REGISTER

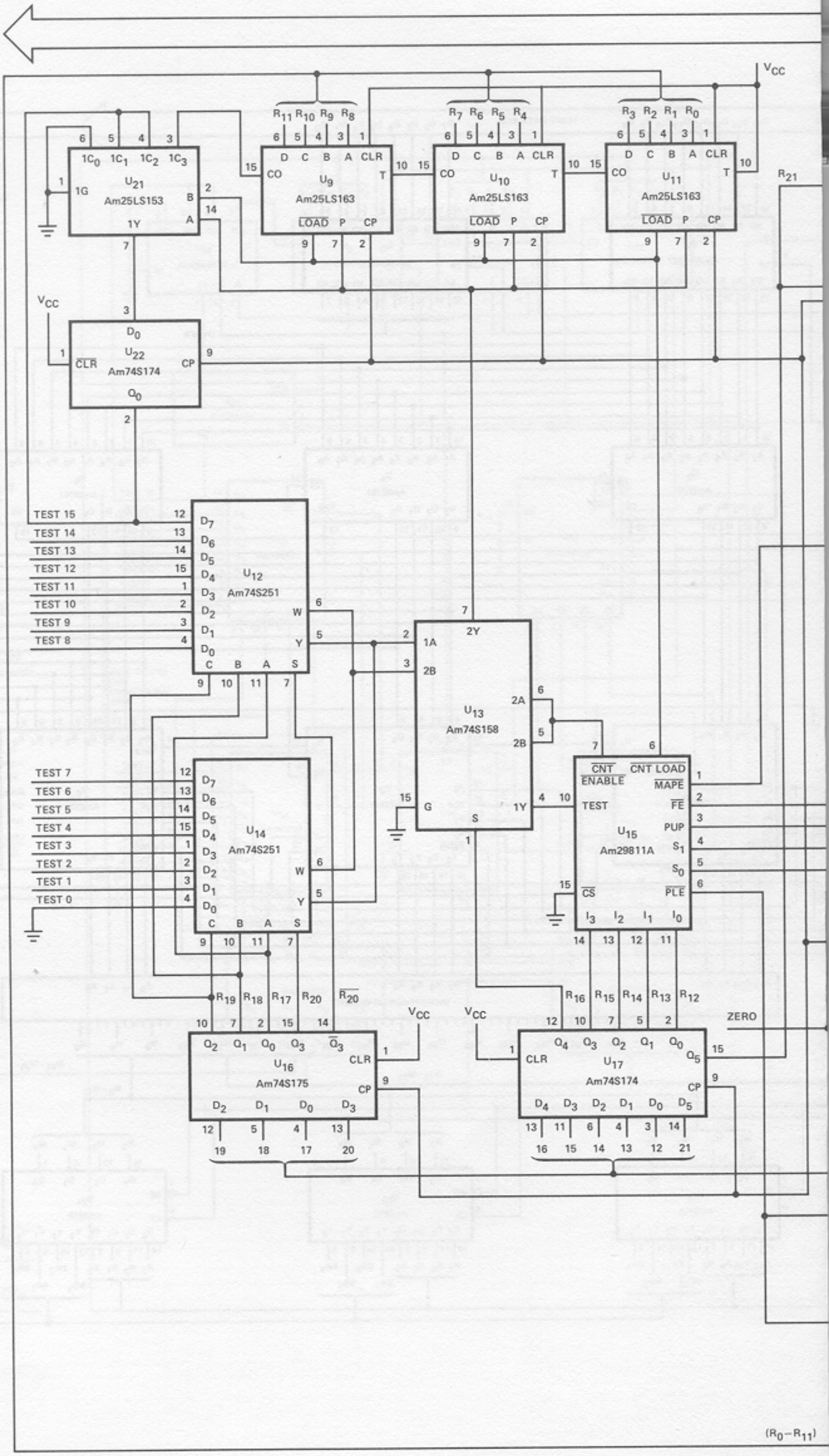
| $R_{21}$ | FUNCTION |
|---|---|
| 0 | LOAD |
| 1 | HOLD |

### CONTROL VALUE

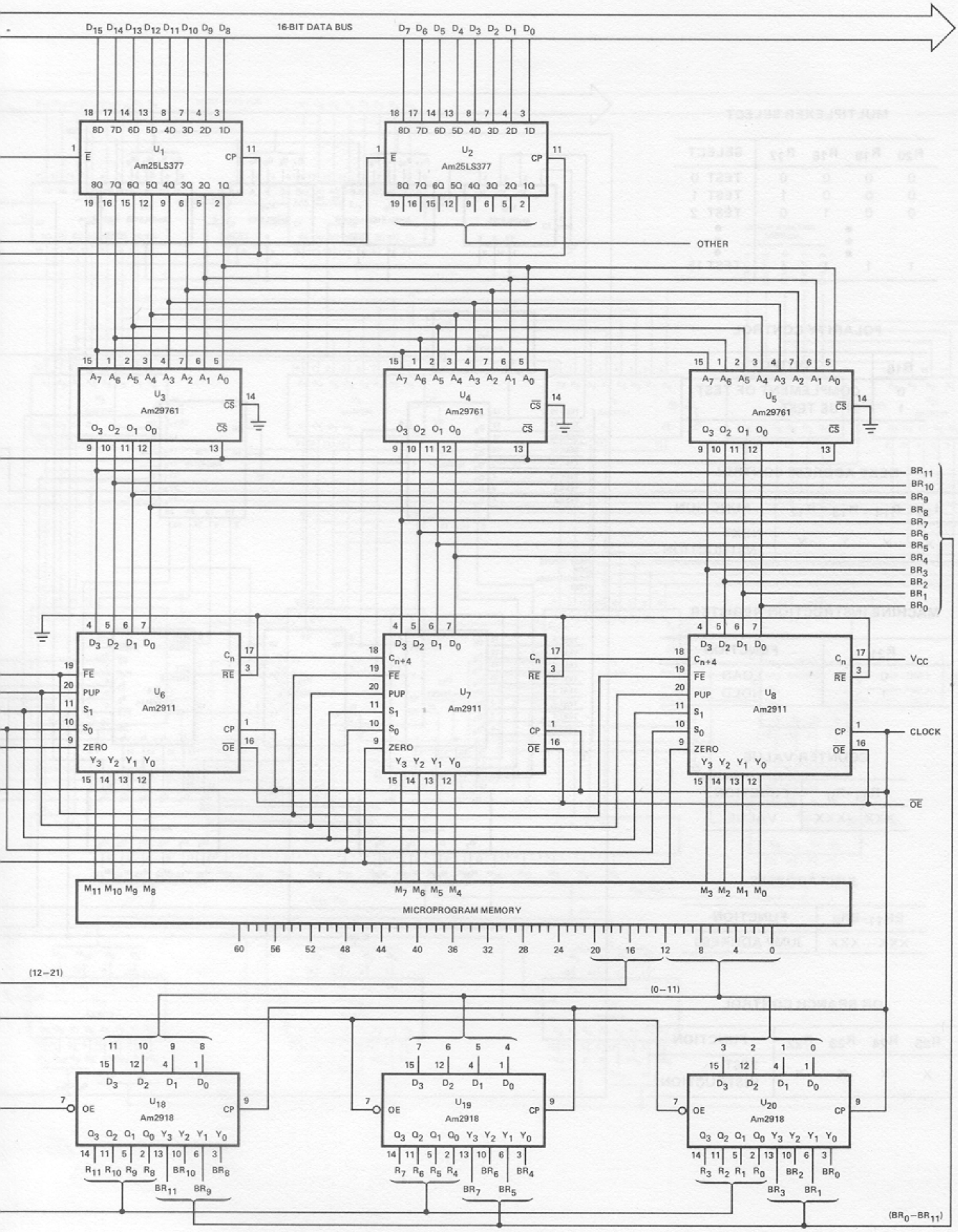
| $R_{11}$–$R_0$ | FUNCTION |
|---|---|
| XXX---XXX | VALUE |

### JUMP ADDRESS

| $BR_{11}$–$BR_0$ | FUNCTION |
|---|---|
| XXX---XXX | JUMP ADDRESS |



Figure



7. Computer Control Unit with Am2911.

**MULTIPLEXER SELECT**

| $R_{20}$ | $R_{19}$ | $R_{18}$ | $R_{17}$ | SELECT |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | TEST 0 |
| 0 | 0 | 0 | 1 | TEST 1 |
| 0 | 0 | 1 | 0 | TEST 2 |
| | ⋮ | | | ⋮ |
| 1 | 1 | 1 | 1 | TEST 15 |

**POLARITY CONTROL**

| $R_{16}$ | OUTPUT |
|---|---|
| 0 | COMPLEMENT OF TEST |
| 1 | TRUE TEST |

**NEXT ADDRESS CONTROL**

| $R_{15}$ | $R_{14}$ | $R_{13}$ | $R_{12}$ | FUNCTION |
|---|---|---|---|---|
| X | X | X | X | NEXT INSTRUCTION |

**MACHINE INSTRUCTION REGISTER**

| $R_{21}$ | FUNCTION |
|---|---|
| 0 | LOAD |
| 1 | HOLD |

**COUNTER VALUE**

| $R_{11}$–$R_0$ | FUNCTION |
|---|---|
| XXX---XXX | VALUE |

**JUMP ADDRESS**

| $BR_{11}$–$BR_0$ | FUNCTION |
|---|---|
| XXX---XXX | JUMP ADDRESS |

**OR BRANCH CONTROL**

| $R_{25}$ | $R_{24}$ | $R_{23}$ | $R_{22}$ | FUNCTION |
|---|---|---|---|---|
| X | X | X | X | TEST INSTRUCTION |

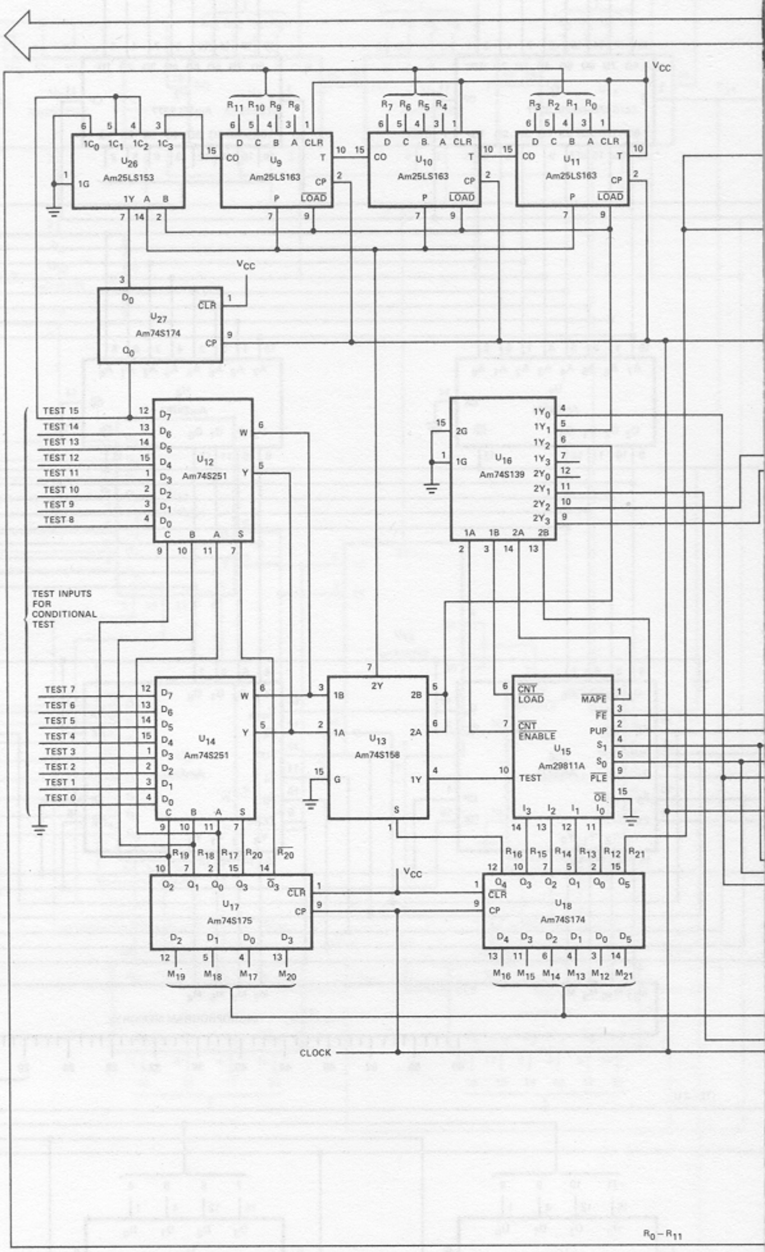
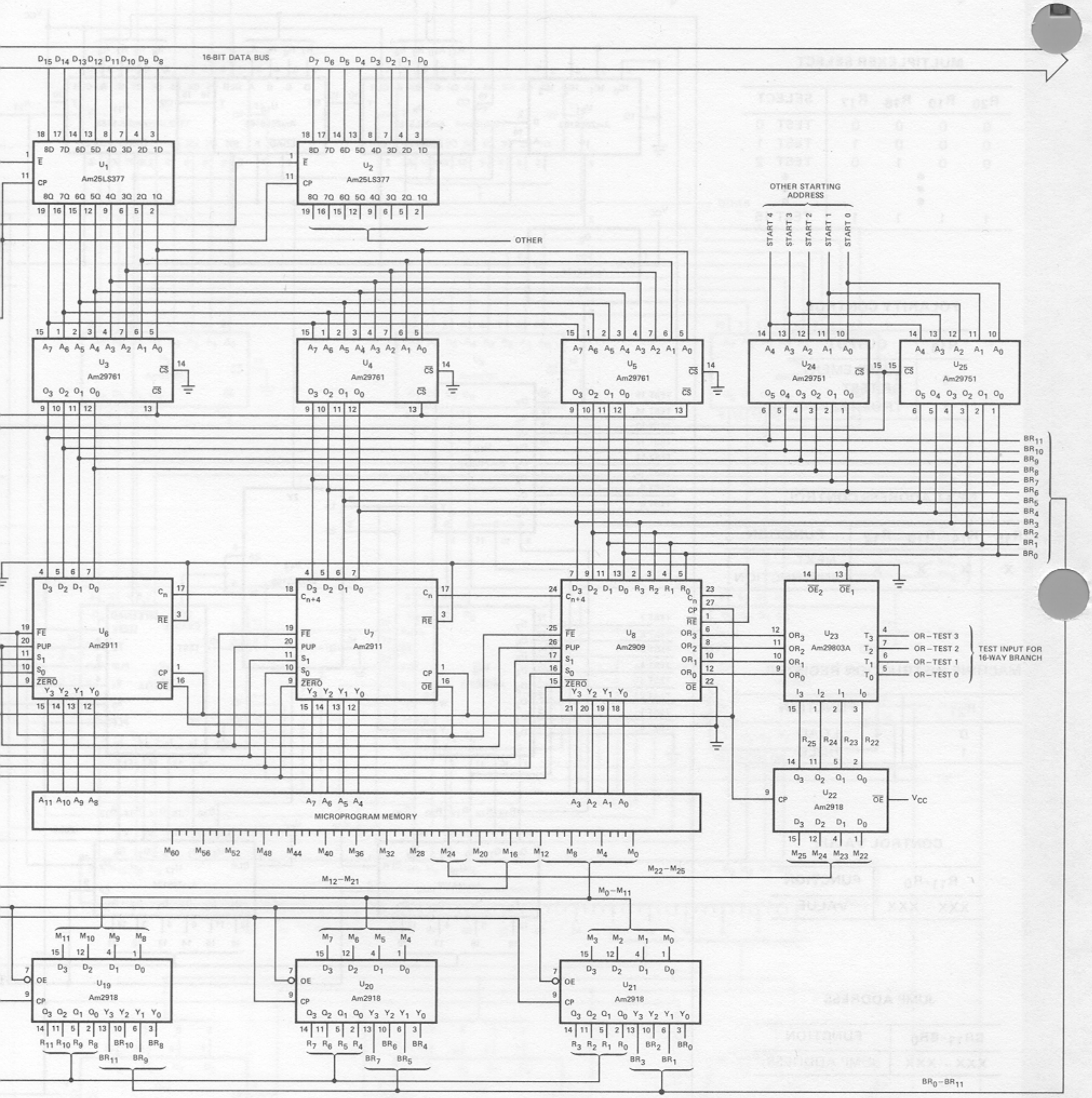


Figure 18



High Performance Computer Control Unit with Am2909/2911.

**MULTIPLEXER SELECT**

| $R_{20}$ | $R_{19}$ | $R_{18}$ | $R_{17}$ | SELECT |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | TEST 0 |
| 0 | 0 | 0 | 1 | TEST 1 |
| 0 | 0 | 1 | 0 | TEST 2 |
| | | • | | • |
| | | • | | • |
| | | • | | • |
| 1 | 1 | 1 | 1 | TEST 15 |

**NEXT ADDRESS CONTROL**

| $R_{15}$ | $R_{14}$ | $R_{13}$ | $R_{12}$ | FUNCTION |
|---|---|---|---|---|
| X | X | X | X | NEXT INSTRUCTION |

**CONTROL VALUE**

| $R_{11}$-$R_0$ | FUNCTION |
|---|---|
| XXX - - - XXX | VALUE |

**POLARITY CONTROL**

| $R_{16}$ | OUTPUT |
|---|---|
| 0 | COMPLEMENT TEST |
| 1 | TRUE TEST |

**MACHINE INSTRUCTION REGISTER**

| $R_{21}$ | FUNCTION |
|---|---|
| 0 | LOAD |
| 1 | HOLD |

**JUMP ADDRESS**

| $BR_{11}$-$BR_0$ | FUNCTION |
|---|---|
| XXX - - - XXX | JUMP ADDRESS |

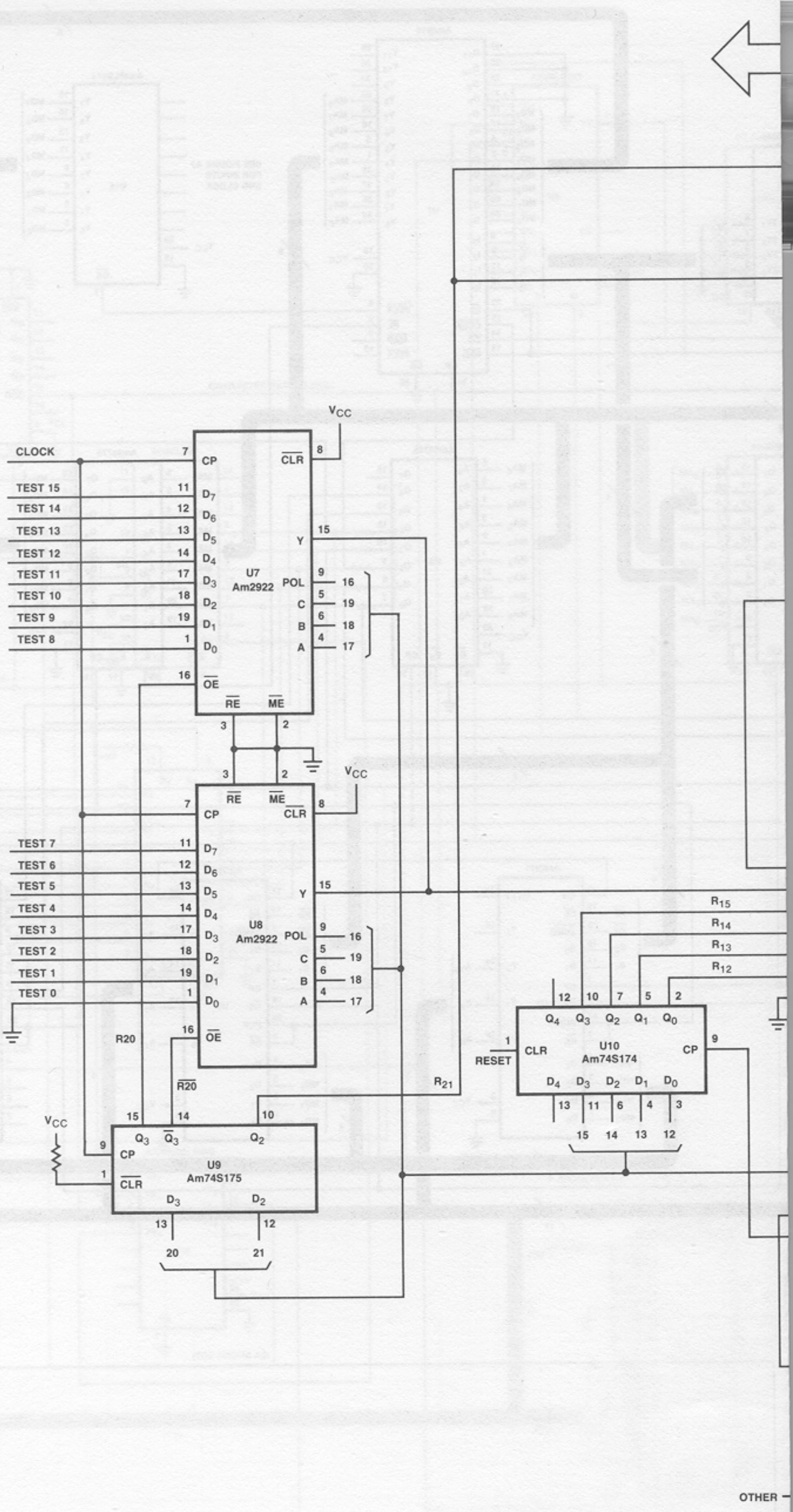
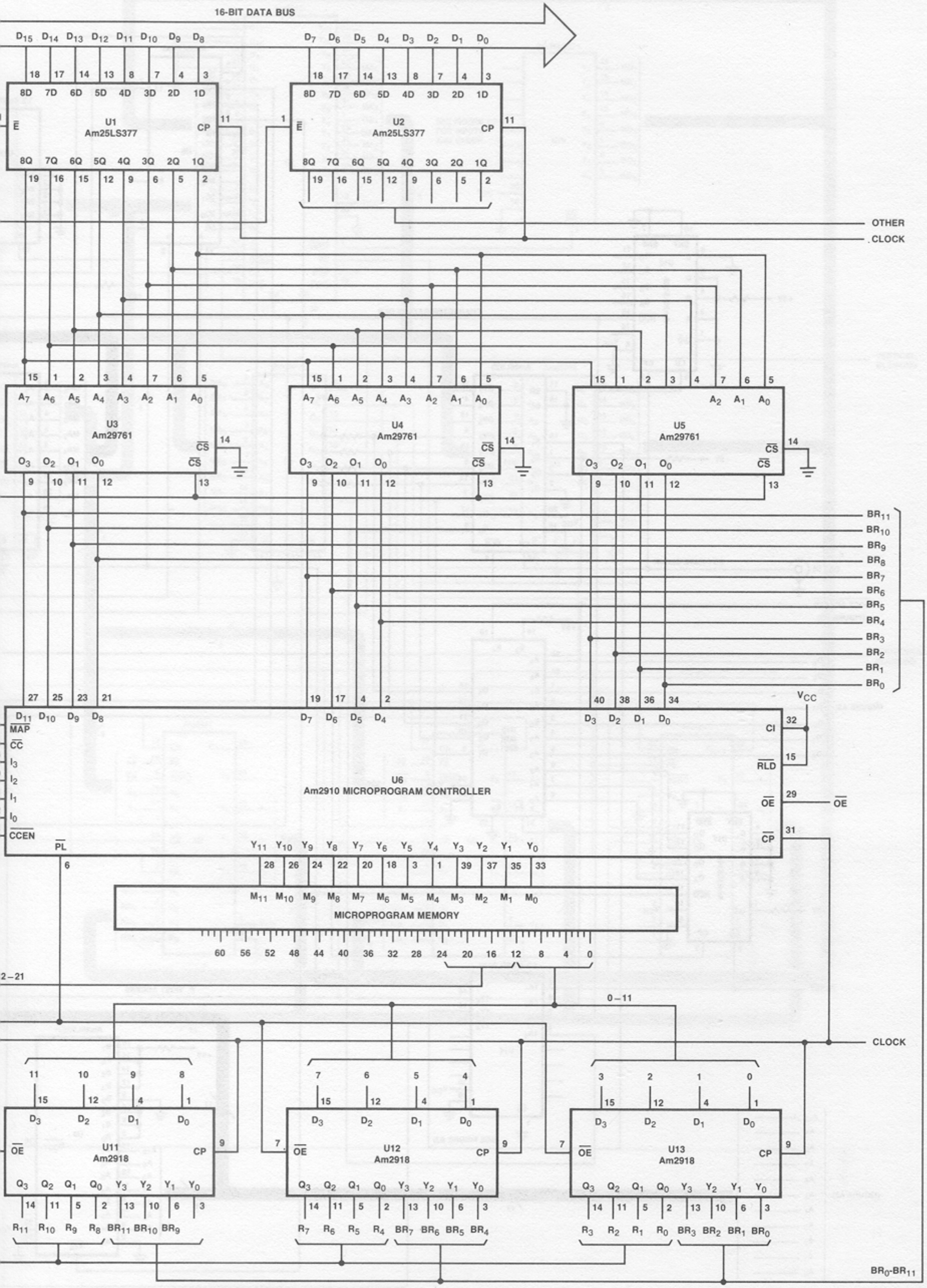
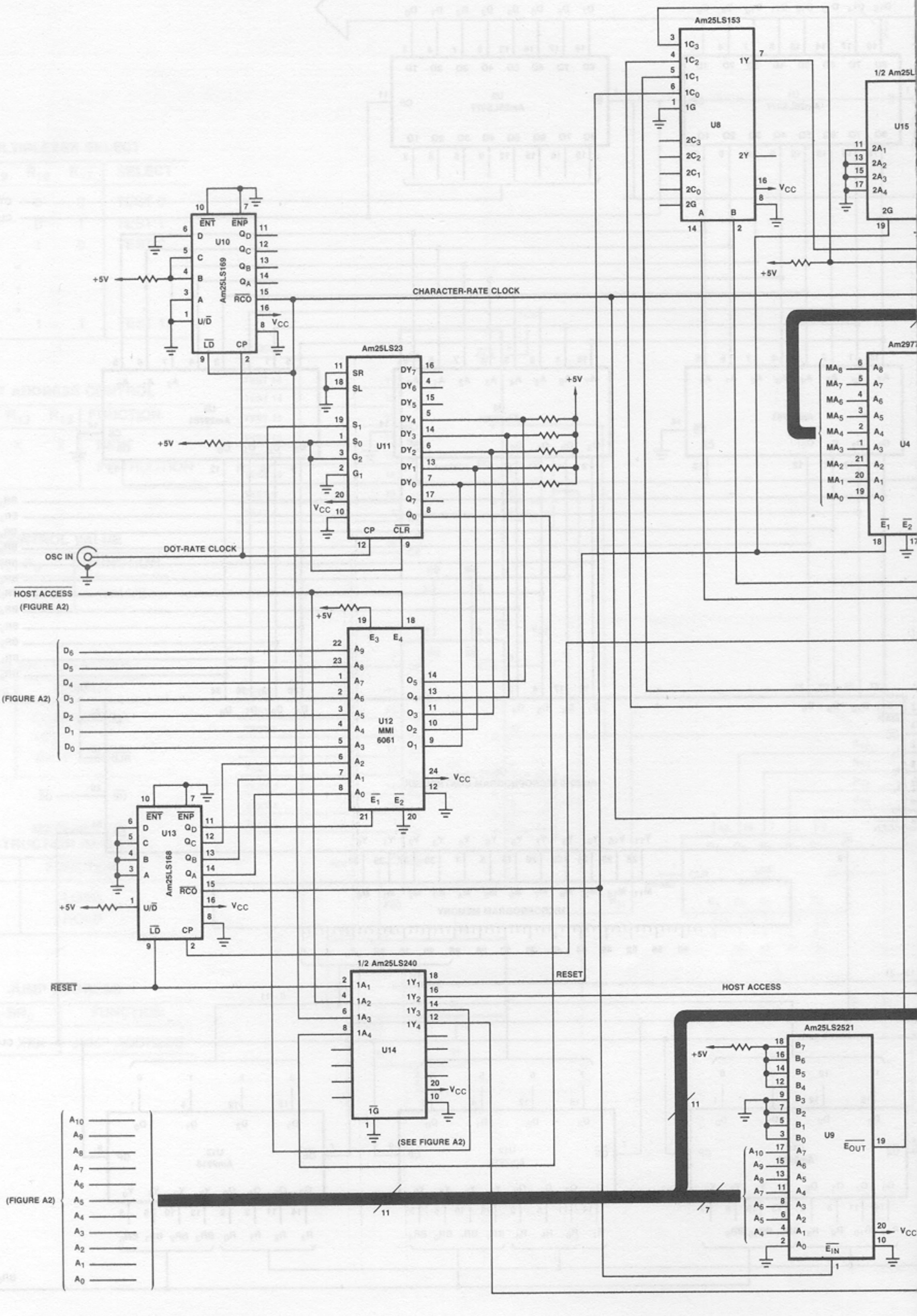


Figure 20. Com



**outer Control Unit with Am2910.**

MPR-505



Am25LS153
U8
1C3
1C2
1C1
1C0
1G
2C3
2C2
2C1
2C0
2G
1Y
2Y
A
B
VCC
1/2 Am25L
U15
2A1
2A2
2A3
2A4
2G
+5V
Am25LS169
U10
ENT
ENP
D
C
B
A
U/D
LD
CP
QD
QC
QB
QA
RCO
VCC
CHARACTER-RATE CLOCK
Am25LS23
U11
SR
SL
S1
S0
G2
G1
DY7
DY6
DY5
DY4
DY3
DY2
DY1
DY0
Q7
Q0
CP
CLR
Am297
MA8
MA7
MA6
MA5
MA4
MA3
MA2
MA1
MA0
A8
A7
A6
A5
A4
A3
A2
A1
A0
U4
E1
E2
OSC IN
DOT-RATE CLOCK
HOST ACCESS
(FIGURE A2)
D6
D5
D4
D3
D2
D1
D0
U12
MMI
6061
E3
E4
A9
O5
O4
O3
O2
O1
Am25LS168
U13
RESET
1/2 Am25LS240
U14
1A1
1A2
1A3
1A4
1Y1
1Y2
1Y3
1Y4
1G
(SEE FIGURE A2)
HOST ACCESS
Am25LS2521
U9
B7
B6
B5
B4
B3
B2
B1
B0
A7
A6
A5
A4
A3
A2
A1
A0
EOUT
EIN
A10
11
7

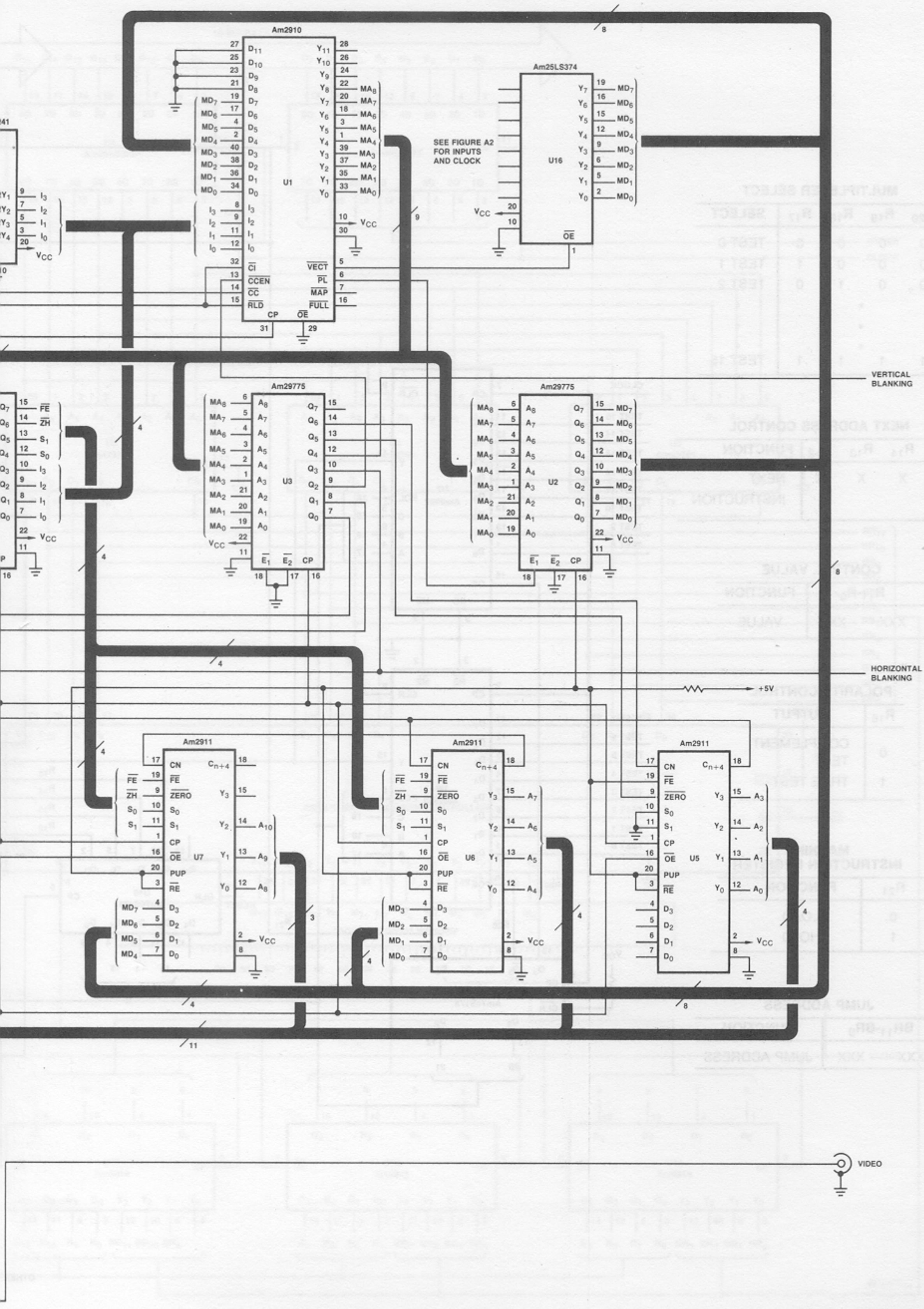
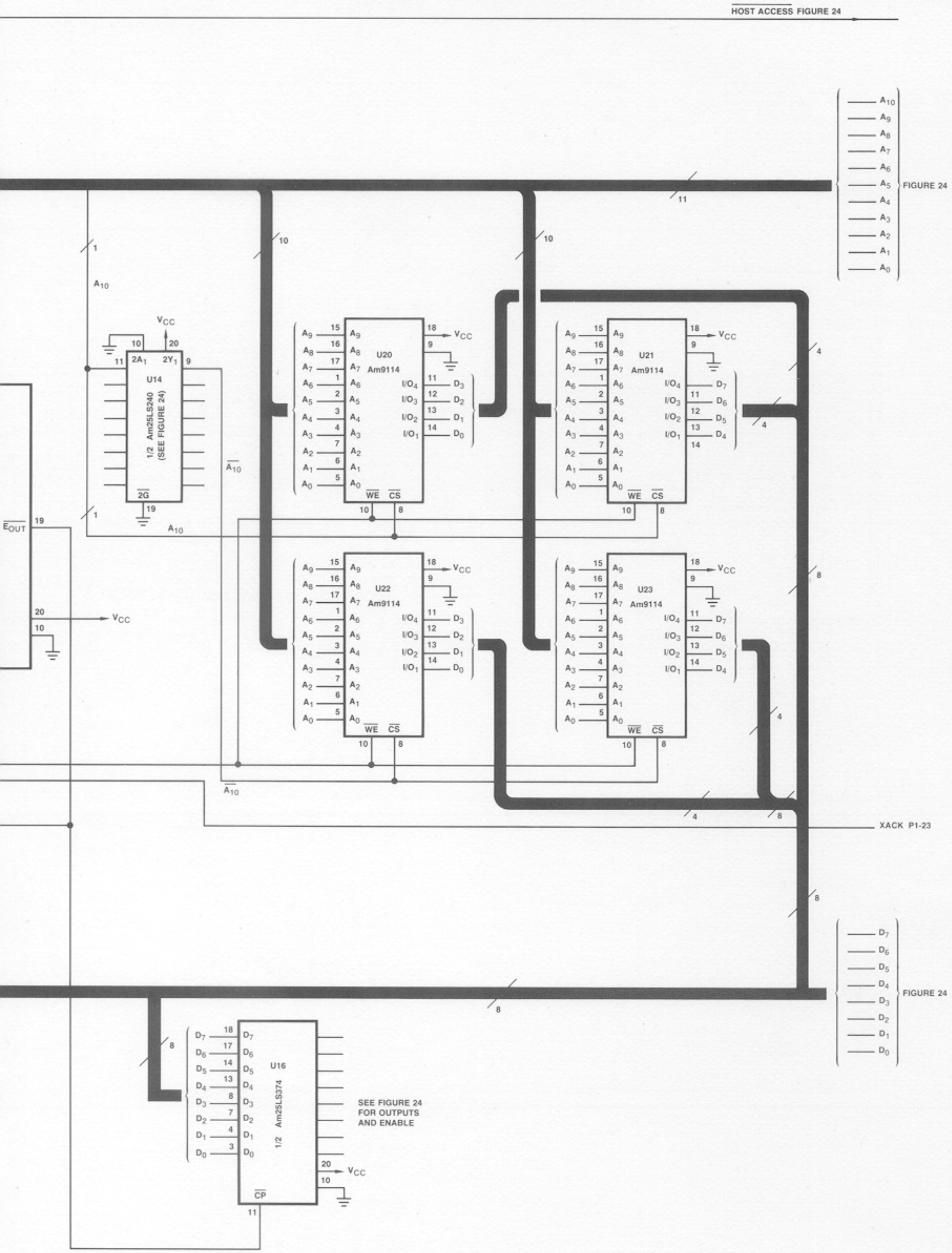


Figure 24. CRT Controller.



T Controller.

**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*