Advanced Micro Devices

Am2900 Learning and Evaluation Kit User's Manual

Advanced Micro Devices, Inc.

AM2900 EVALUATION AND LEARNING KIT INSTRUCTION MANUAL

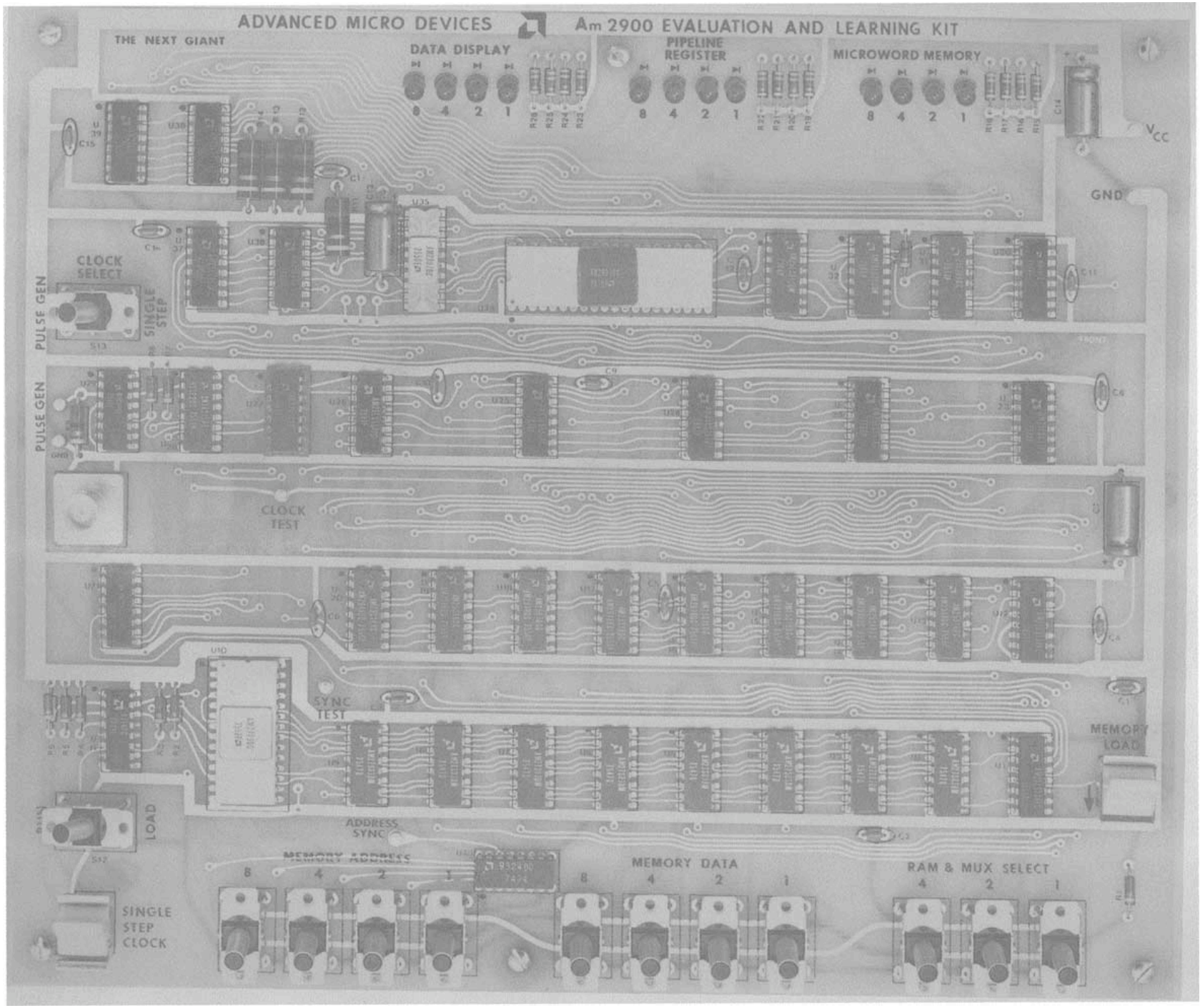
ADVANCED MICRO DEVICES
Am 2900 EVALUATION AND LEARNING KIT
THE NEXT GIANT
DATA DISPLAY
PIPELINE REGISTER
MICROWORD MEMORY
8 4 2 1
V_{CC}
GND
CLOCK SELECT
SINGLE STEP
PULSE GEN
PULSE GEN
CLOCK TEST
SYNC TEST
MEMORY LOAD
ADDRESS SYNC
LOAD
SINGLE STEP CLOCK
MEMORY ADDRESS
MEMORY DATA
RAM & MUX SELECT

TABLE OF CONTENTS

SECTION I

INTRODUCTION

Advanced Micro Devices has designed an educational tool, the Am2900 Evaluation and Learning Kit, to be used by the design engineer learning microprogramming. The purpose of this kit is twofold. First, it is intended to introduce the design engineer to several of the circuits in the Am2900 Bipolar Microprocessor Family. Second, it is intended to be an instructional tool for the engineer faced with his first microprogramming job. The kit consists of one Am2901 Bipolar Microprocessor, one Am2909 Bipolar Microprogram Sequencer, and several memories, registers, and multiplexers organized in a typical CPU (Central Processing Unit) structure. It should be said at the onset that the purpose of this kit is to introduce the design engineer to the Am2900 Family devices and provide a microprogram learning tool. This kit is NOT a four-bit computer.

The organization of the kit is a reduced example of a typical CPU design. In the microprogram control area, one Am2909 Bipolar Microprogram Sequencer is used to address 16 words of microprogrammed memory. The kit uses eight Am27S03 64-bit RAM's (Random Access Memories) connected as the microprogram memory. This provides a total microprogram memory that is 16 words deep where each word is 32 bits wide. This memory is used as the writeable microprogram memory for holding the microinstructions of the program. The kit contains eight Am2918 Registers used as a 32-bit wide pipeline register at the output of the microprogram memory. The kit also has a 32-word by 8-bit PROM (Programmable Read Only Memory) in the Am2909 control path to generate next microprogram address commands.

The ALU (Arithmetic Logic Unit) in the Am2900 kit utilizes one four-bit Am2901 Bipolar Microprocessor slice. One Am25LS08 is used as the status register which holds the four status flags. The ALU section also uses two Am25LS253 Multiplexers for shifting data into either the RAM shift matrix or the Q shift matrix under microprogrammed control. This allows the execution of various types of microinstructions and the results of these operations can be learned in concept and in practice.

The microprogram memory is loaded with data by using a combination of toggle switches that select the memory to be loaded, apply the address for the memory, and also apply the data to be loaded into the memory. The contents of the microprogram memory can be viewed conveniently, four bits at a time, on four LED (Light Emitting Diode) lamps. Four-bit fields associated with

the Am2901, Am2907, Am2909, and Am25LS08 can be viewed on four additional LED lamps. Likewise, four-bit fields of the Am2918 Pipeline Register can be viewed on a separate LED display. This provides the student with the ability to learn the machine operation in a step-by-step program sequence.

All components required in the assembly of the Am2900 Evaluation and Learning Kit are supplied in the kit package. The only item that need be supplied by the user of the kit is a +5V power supply capable of delivering approximately 2 amperes of current. The assembly diagram and assembly instructions in this book show the location of each of the components on the printed circuit board. Each component should be assembled in its position and then soldered in place. The user should assemble the board in steps and check out the assembly as it proceeds. The recommended technique will be to load some of the components onto the printed circuit (PC) board and solder them in place. A +5V power supply will be connected to the PC board and a test procedure followed to ensure proper operation of the devices as installed. This procedure is defined in detail in Section IV during kit assembly.

In order to utilize the Am2900 Family efficiently, a basic understanding of microprogramming is essential. The simplest method of microprogramming is for each microinstruction step to be implemented in a sequential manner. The EXECUTION of the microprogram allows the CPU to operate on a microinstruction in a particular memory location. Then, the microprogram controller increments to the next memory location for the next microinstruction. After each microinstruction fetch, the pipeline register will contain the memory location contents for execution. In order to provide the basic understanding of microprogramming, Section II on this topic is included in this book. Section III describes the basic operation of the Am2900 Evaluation and Learning Kit applying the basics from Section II.

SECTION II

UNDERSTANDING THE BASIC THEORY OF MICROPROGRAMMING

INTRODUCTION

With the advent of the Am2901 four-bit microprocessor slice and the Am2909 bipolar microprogram sequencer, the design engineer can upgrade the performance of existing systems or implement new systems taking advantage of the latest state-of-the-art technology in Low-Power Schottky integrated circuits. These devices, however, utilize a new concept in machine design not familiar to many design engineers. This technique is called microprogramming.

Basically, 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 macroinstruction. All of the little elemental tasks performed by the machine in executing the macroinstruction are called microinstructions. The storage area for these microinstructions or microprogram signals is usually called the microprogram memory.

A microinstruction usually has two primary parts. These are: (1) the definition and control of all 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.

UNDERSTANDING THE MICROPROGRAM MEMORY

The microprogram memory is simply a N word by M bit memory used to hold the various microinstructions. Figure 1 depicts the word number definition of an N word microprogrammed memory. For an N word memory, the address locations are usually defined as location 0 through location N-1. For example, a 256-word microprogram memory will have address locations 0 through 255.

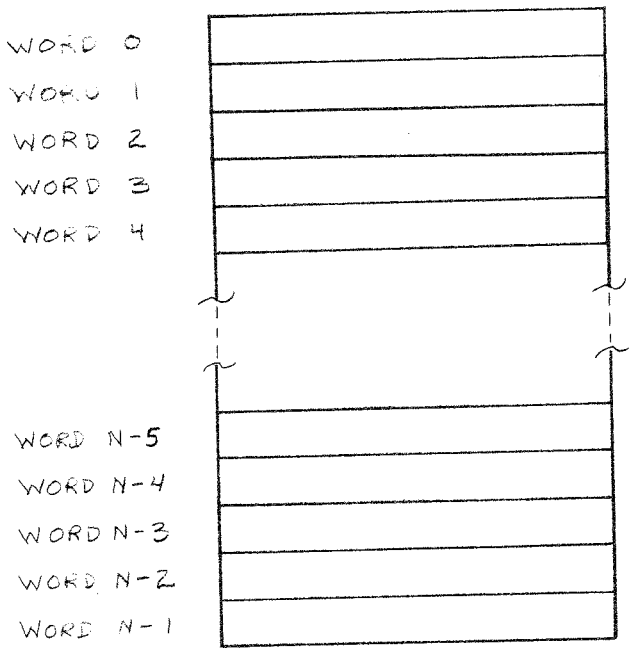

FIGURE 1: Organization of an N-word Microprogram Memory

Each word of the microprogram memory consists of M bits. These M bits are usually broken into various field definitions. A typical example of field definition is shown in Figure 2 where a 32-bit word made up of nine fields is depicted. It should be noted that the fields can consist of various numbers of bits. For example, field number 1 is 5 bits wide, field number 2 is 8 bits wide, field number 6 is 1 bit wide, and so forth. It is the definition of the various fields of a microprogram word that is usually referred to as FORMATTING. Thus, Figure 2 shows the format of a 32-bit microinstruction.

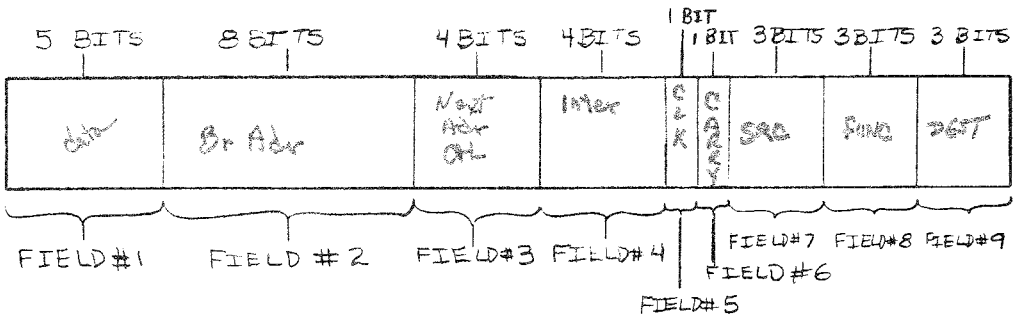

FIGURE 2: Definition of the Fields within one 32-bit MICROPROGRAM Memory Word

Examples of the uses of the field as defined in Figure 2 are as follows:

Field 1 - General purpose
Field 2 - Branch address
Field 3 - Next address control
Field 4 - Interrupt control
Field 5 - Fast clock/slow clock select
Field 6 - Carry control
Field 7 - ALU source operand control
Field 8 - ALU function control
Field 9 - ALU destination control

The above field definition is an example of how microinstruction fields are defined in a typical machine.

SEQUENCING THROUGH MICROINSTRUCTIONS

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. Such a simplified microprogram memory address control is shown in Figure 3. The microprogram address counter simply increments by one on each clock cycle to select the address of the next microinstruction. Figure 4 shows an example of what might be occurring in this mode. 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 but it would be able to execute a fixed pattern of microinstructions.

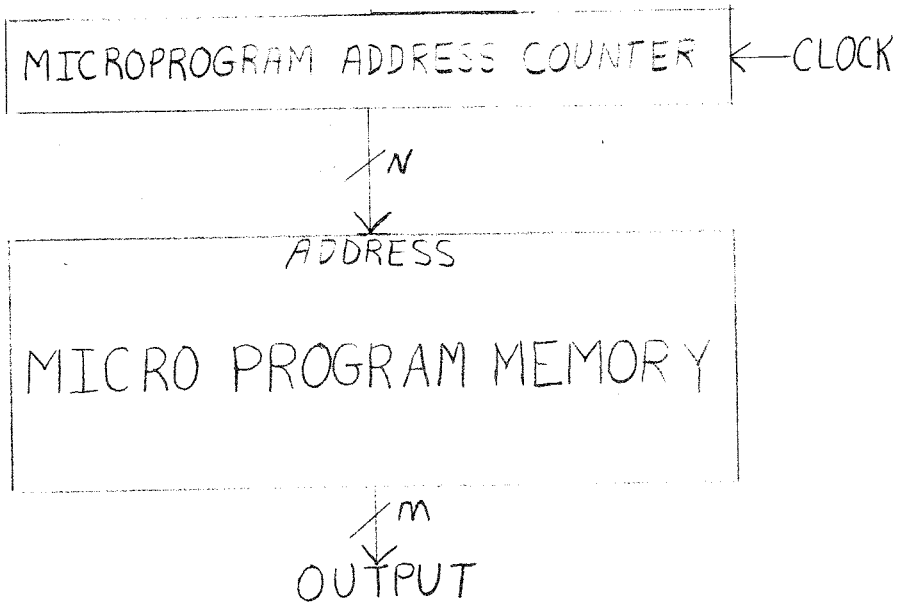

FIGURE 3: Microprogram Memory Address Control Using a Counter

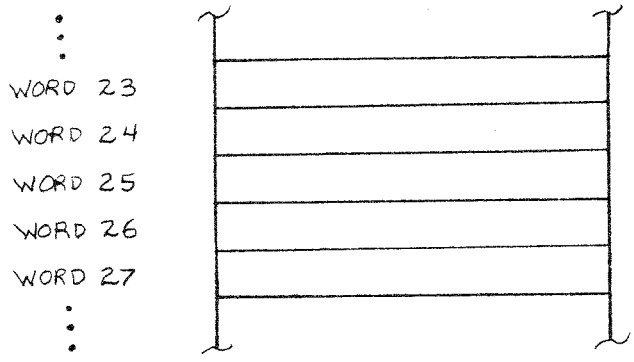

FIGURE 4: Executing Sequential Microprogram Memory Words

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

MICROPROGRAM BRANCHING OR 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 BRANCH address. Figure 5 shows a control unit architecture whereby a BRANCH address can be parallel loaded into the microprogram address counter. The load control is 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 logic 1, a load will be enabled. If the load is enabled, the branch 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 branch to any address in the memory space from word 0 through word 255 can be performed. Note that there may be many other fields in the microprogram.

Let us examine the programming that would be required by the microprogram control unit described in Figure 5. Referring to the microprogram memory map of Figure 6, assume the current address in the microprogram address counter is 51. Since the load control is logic 0, the counter will increment on the next clock cycle and will contain address 52. In fact, the counter will continue to increment through words 53, 54, 55, 56 and 57. When the microprogram address counter contains address 57 as shown in Figure 6, the load control bit changes to a logic 1. When this happens, the microprogram address counter will parallel load the branch address supplied to its parallel data inputs. An example of such a counter is the Am25LS161 binary counter. The branch address shown in the contents of word 57 in Figure 6 is address 90. On the next clock cycle, 90 will be loaded into the microprogram address counter. Thus, the next microinstruction executed will be defined by the contents of the microinstruction word at address 90. Figure 6 now defines the load control bit at instruction 90 to be logic 0. Thus, on the next clock cycle, the counter will increment to address 91. When address 92 is reached, another BRANCH is defined. The branch address at word 92 is address 13.

The simple branching control feature described in Figure 5 and Figure 6 allows a microprogram memory controller to execute sequential microinstructions or perform a BRANCH to any address either before or after the address currently contained in the microprogram address counter.

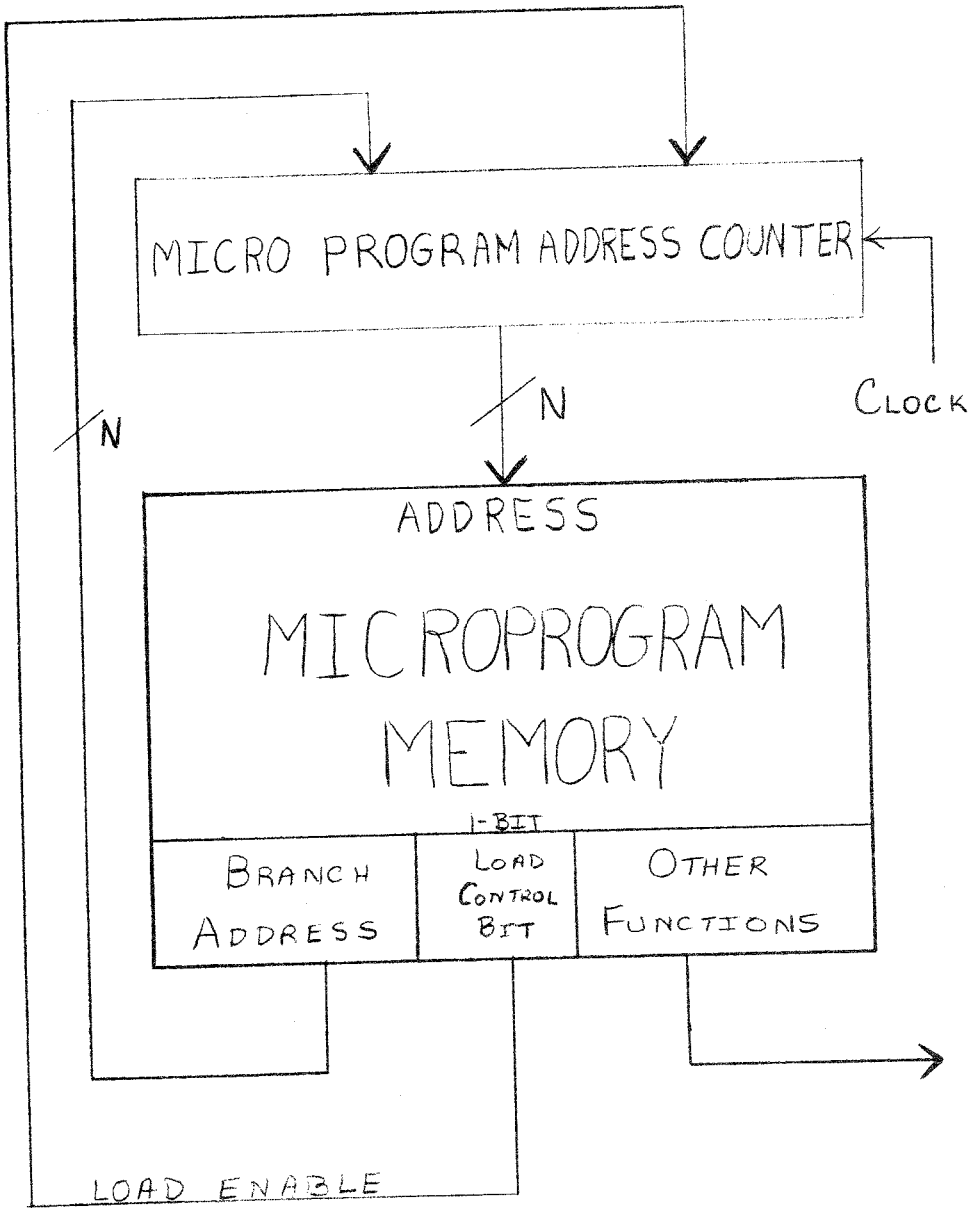

FIGURE 5: Branching or Jumping in a Microprogram Memory Space Requires a Branch Address and a Load Control Signal

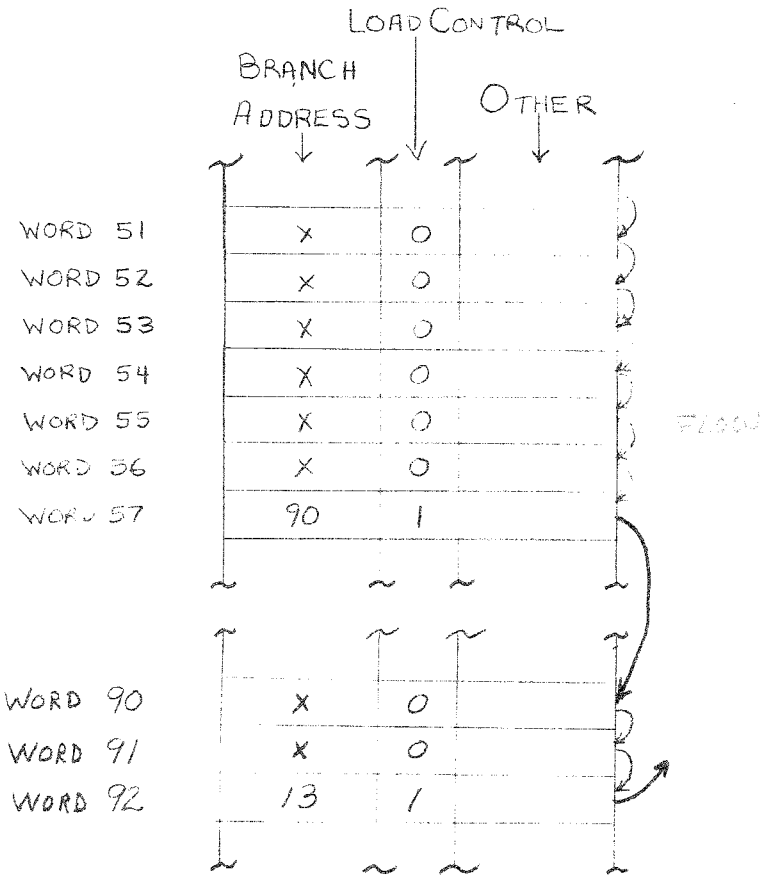

FIGURE 6: Branching or Jumping in a Microprogram Control Space

CONDITIONAL BRANCHING

While the BRANCH or 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 BRANCH instruction. Figure 7 shows a functional block diagram of a microprogram memory/address controller providing the capability to branch on two different conditions. In this example, the load select control is a two-bit field used to control a 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 branch address will be parallel loaded in the next clock cycle. This operation is defined as a CONDITIONAL BRANCH. If the load select input contains binary 2, the D_2 input is selected and the same 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 branch 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 I.

TABLE I

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

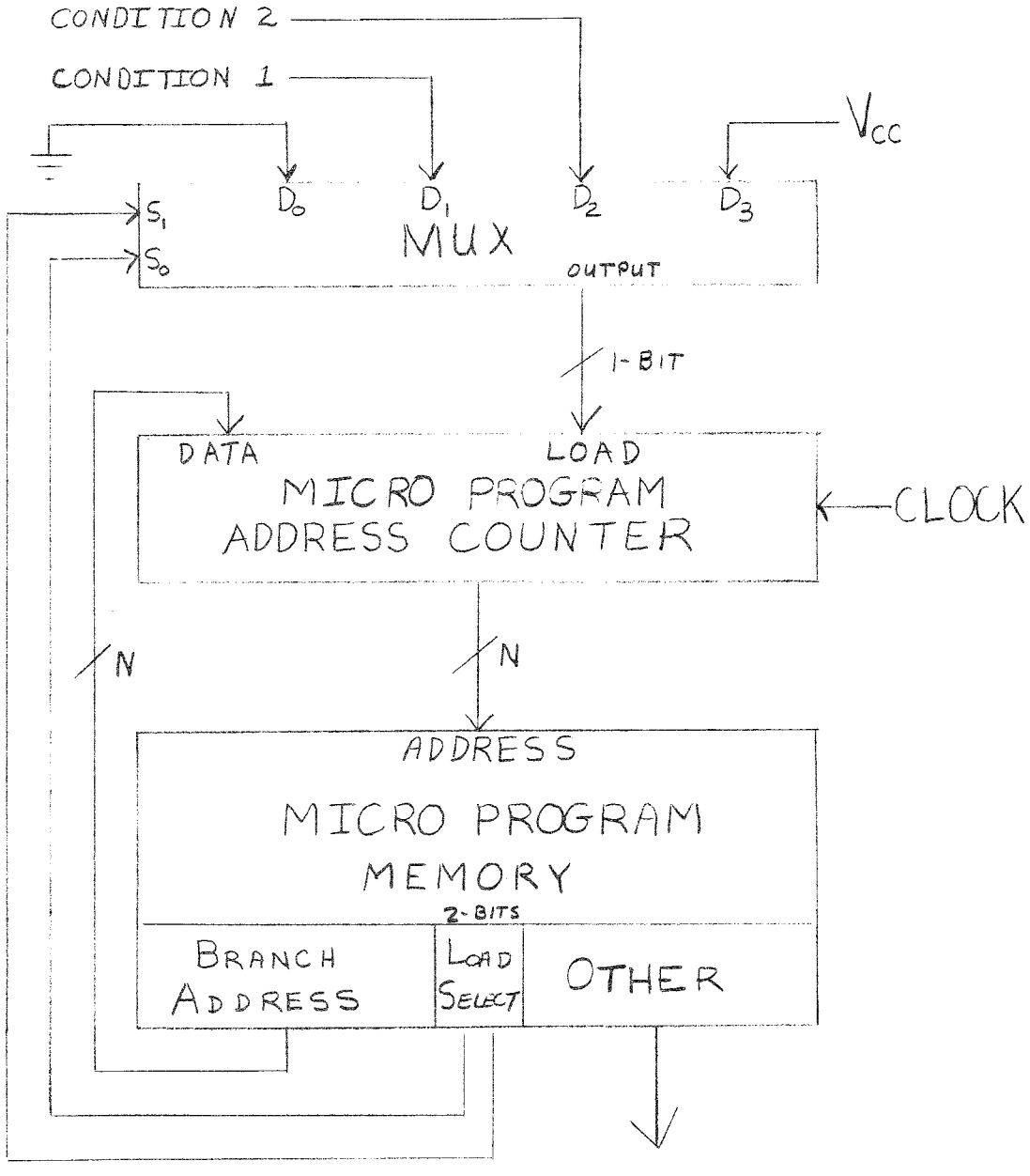

FIGURE 7: A Two-bit Control Field can be used to Select CONTINUE, BRANCH, or CONDITIONAL BRANCH

A microprogram memory map showing the use of conditional branching in microprogram control is depicted in Figure 8. In this map, we assume the microprogram address counter is currently pointing at word 30. The load select field causes the address counter to increment to word 31 on the next clock cycle. At word 31, a CONDITIONAL BRANCH is encountered. If the Condition 1 input is logic 0, the next word selected will be word 32. If the Condition 1 input is logic 1, the next word selected will be word 38. The example of Figure 8 shows other CONDITIONAL BRANCHES at word 32, 35, and 36. It also shows an UNCONDITIONAL BRANCH at word 39. It should be noted that words 35 and 36 result in CONDITIONAL BRANCHES to the same address, address 95. The essence of this function is that if either Condition 1 or Condition 2 is true as tested on sequential microinstructions, the program branches to the sequence beginning at address 95.

An example of the power of CONDITIONAL BRANCHING is shown in Figure 9A. Here, ignoring the start-up problem, let us assume the microprogram address counter contains the word 0. The instruction being executed is a conditional test on Condition 1 input. If Condition 1 is true, branch to word 4 and if Condition 1 is false, continue. Word 1 performs a test on Condition 2. This test results in a branch to word 8 if true, or a continue if false. Word 2 performs a condition test on Condition 1. If true, a branch to word 12 occurs, if false a continue to word 3 occurs. Word 3 is an unconditional branch to the starting address at word 0. In this example, if a jump occurs to either word 4, 8, or 12, the net result is that the next four instructions are executed and then an unconditional branch to word 0 occurs. The flow table for this 16-word microprogram is shown in Figure 9B.

The microprogram control flow described in Figures 9A and 9B represent a simple state machine design controller that looks for either Condition 1 to occur, Condition 2 to occur, or not Condition 2 followed by Condition 1 occurring before a reset to word 0. In the event any of the conditions do occur, a small series of tasks (four microinstructions) are to be executed and then the machine returned to word 0. If none of the test conditions occur, the machine is reset at word 3 and returned to the test at word 0. At power-up, an asynchronous reset could be applied to the microprogram address counter to initialize the machine. Although Figure 9A and Figure 9B describe an extremely simple microprogram control unit, it is representative of the microprogram control power contained in only three basic microinstructions. These microinstructions are the CONTINUE instruction, the UNCONDITIONAL JUMP instruction, and the CONDITIONAL JUMP instruction.

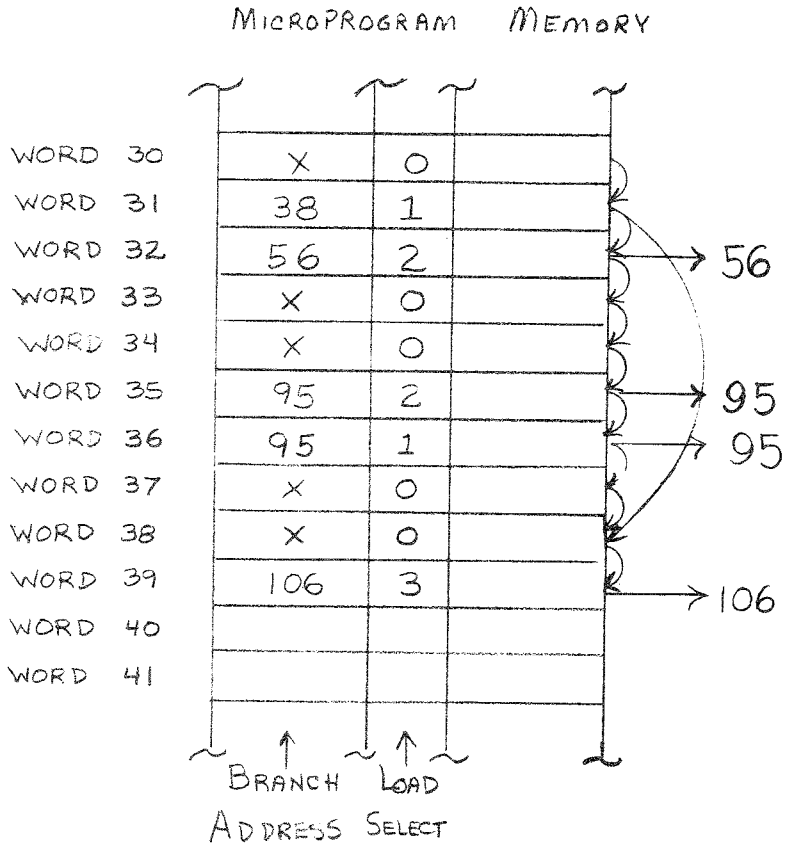

FIGURE 8: CONDITIONAL BRANCHING Using a Two-bit Select Field as shown in Figure 7

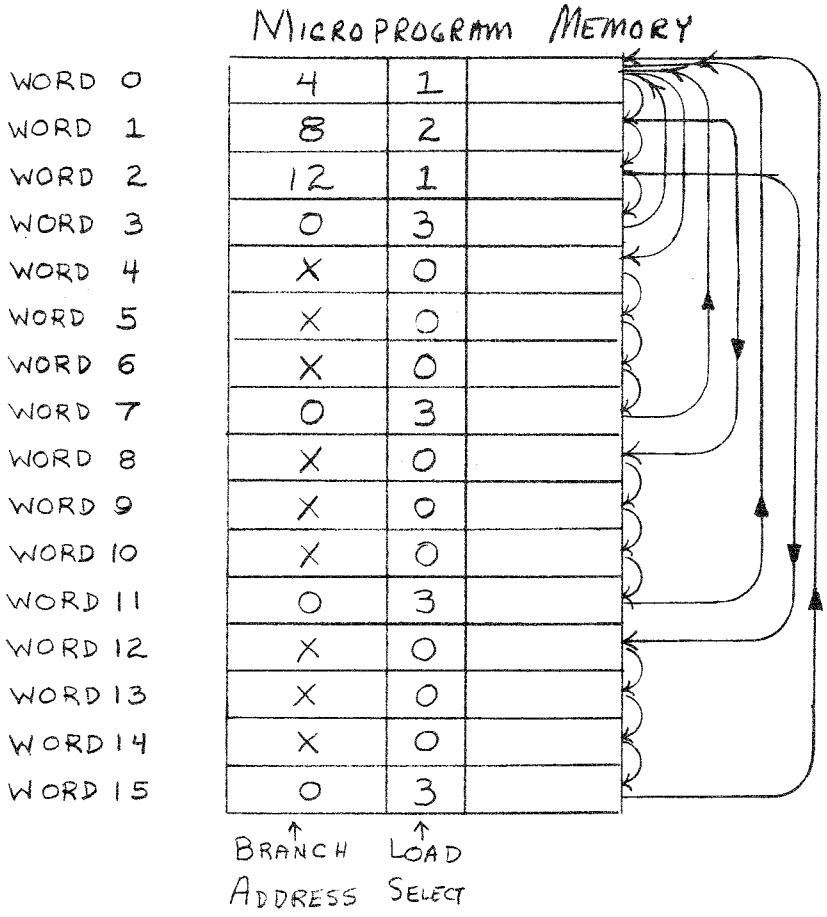

FIGURE 9A: CONDITIONAL BRANCHING Example

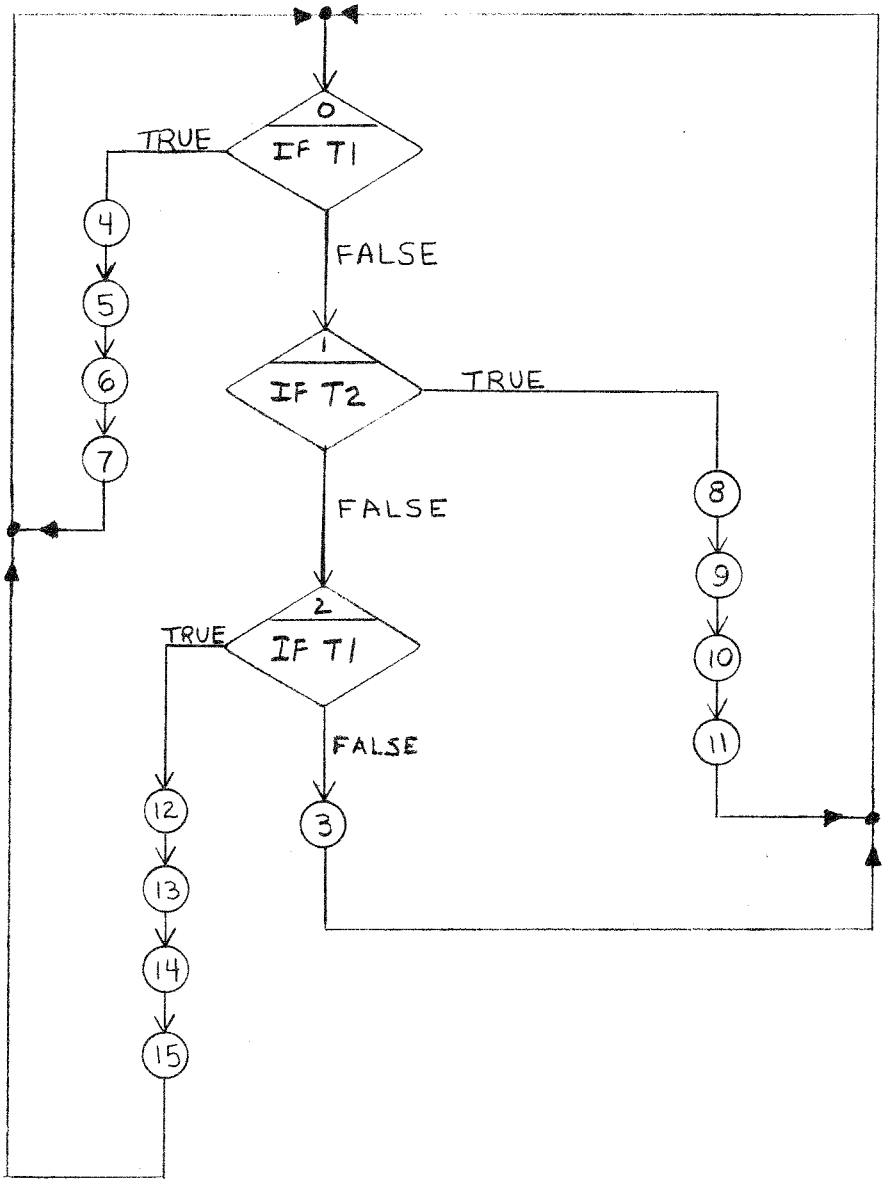

FIGURE 9B: Flow Table for CONDITIONAL BRANCH Example

OVERLAPPING THE MICROPROGRAM INSTRUCTION FETCH

Now that a few basic microprogram address control instructions have been defined, let us examine the 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 10. 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 microprogram fetch, the current microinstruction being executed and the results of the previous microinstruction are available with respect to each other simultaneously. Said another way, 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.

Let us now compare the block diagram of Figure 10 with that shown in Figure 7. 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, it is possible to always generate the current microprogramming address "plus one" at the input of the microprogram counter register. In Figure 7, the microprogram address counter was described as a device such as the Am25LS161 counter. In the implementation as shown in Figure 10, the Am25LS161 counter is not appropriate. Instead, an incrementer and register are used to give the equivalent effect of a counter.

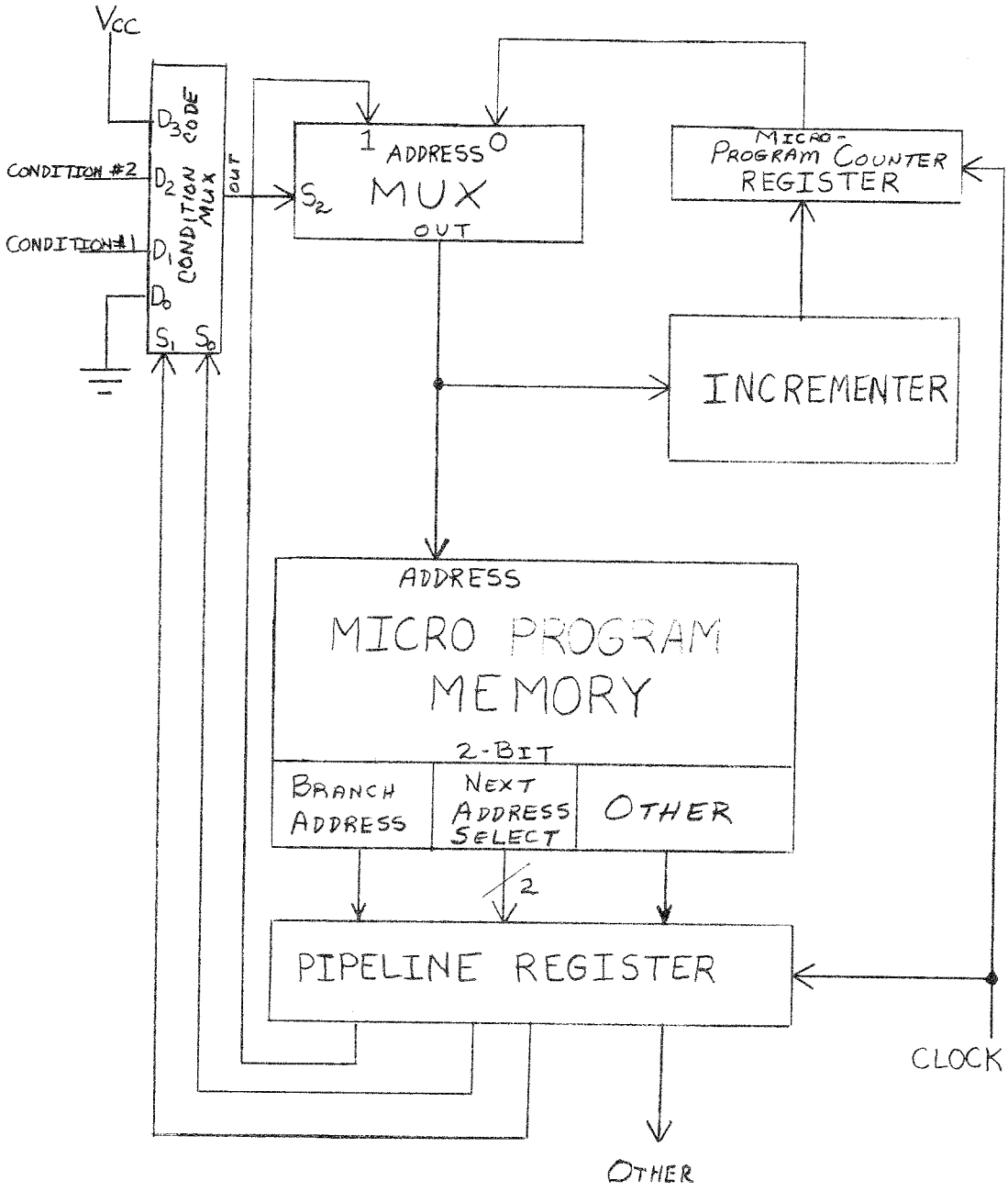

FIGURE 10: OVERLAPPING (or PIPELINING) the Fetch of the Next Microinstruction

The key difference between using a true binary counter and the incrementer register described here is as follows. When the branch 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 mutliplexer 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 7. 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 mutliplexer via D_1 or D_2. Thus, a CONDITIONAL BRANCH can be executed. Binary three in the next address select portion of the pipeline register causes an UNCONDITIONAL BRANCH instruction to be executed via D_3.

When the overall machine timing is studied, it will be noticed 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 MICROPROGRAM CONTROL

Thus far, we have examined the CONTINUE instruction as well as the CONDITIONAL and UNCONDITIONAL BRANCH instructions. 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 can be shared by several instruction sequences of microcode. This results in the 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 11, we see the

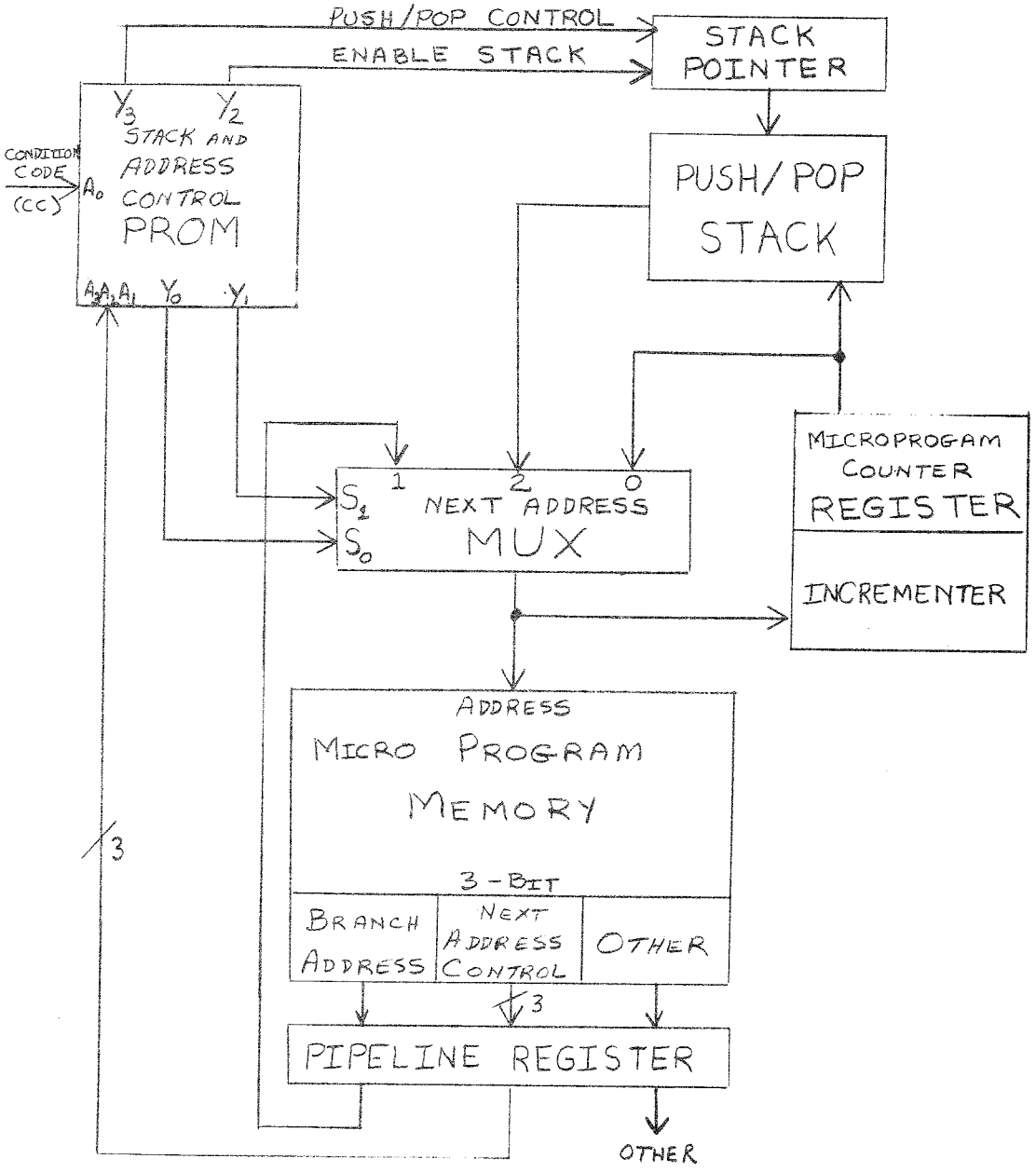

FIGURE 11: Block Diagram for SUBROUTINE Control Using a PUSH/POP Stack

addition of a push/pop stack and its associated stack pointer. The control signals required by the stack are an enable stack signal which will be used to tell the file whenever we wish to perform a push or a pop, and a push/pop control 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 into the file. This allows the "next address multiplexer" to read the file at any time via port 2 on the next address MUX. 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 stack and address control programmable read only memory (PROM). The next address control field of the microprogram word has been expanded to a three-bit field. This three-bit field in conjunction with one condition code bit form the four-bit address control for the stack and address control PROM. The PROM has four outputs used to control the next address multiplexer and the stack pointer. The net result is that in this example, eight instructions have been defined where three of the instructions are conditional. Table II shows these next address select control functions. They include continue, branch, branch on condition, push, jump-to-subroutine, jump-to-subroutine on condition, return-from-subroutine, and test end of loop. Note that this architecture does not provide the ability to select any particular starting address. Also , either a master reset or a known address must be provided to the system to somehow handle power-up turn-on. A system using the Am2909 can provide this feature. Table III shows a detailed definition of the instruction set defined in Table II.

TYPICAL COMPUTER CONTROL UNIT ARCHITECTURE USING THE AM2909

The microprogram memory control described in Figure 11 is easily implemented using the Am2909 as shown in the block diagram of Figure 12. Note the addition of a fourth input (D) for selecting the starting address of a sequence of microinstructions. The Am2909 thereby provides four different inputs from which the next address can be selected. These are the direct input (D), the register input (R), the program counter (PC), and the file (F). A detailed logic diagram of the Am2909 is shown for clarity in Figure 13.

The architecture of Figure 12 shows a macroinstruction register capable of being loaded with a macroinstruction word from the data bus. The op code portion of the instruction is decoded in a mapping PROM to arrive at a starting address for the microinstruction sequence required to execute the macroinstruction. When the microprogram memory address is to be the first microinstruction of the macroinstruction sequence, the microsequencer control PROM selects the multiplexer D input.

TABLE II

Next Address Select Control Functions

$A_3A_2A_1$	Function
000	Continue
001	Branch
010	Branch on condition
011	Push
100	Jump to subroutine
101	Jump to subroutine on condition
110	Return from subroutine
111	Test end of loop

TABLE III

PROM Control for Table II Function

Function	$A_3A_2A_1$	A_0	Next Address MUX S_1S_0	Enable Stack	Push/ Pop
Continue	000	0 1	0	No	X
Branch	001	0 1	1	No	X
Conditional Branch	010	0 1	0 1	No	X
Push	011	0 1	1	Yes	Push
Jump to Subroutine	100	0 1	1	Yes	Push
Conditional Jump to Subroutine	101	0 1	0 1	No Yes	X Push
Return from Subroutine	110	0 1	2	Yes	Pop
Test end of loop	111	0 1	2 0	No Yes	X Pop

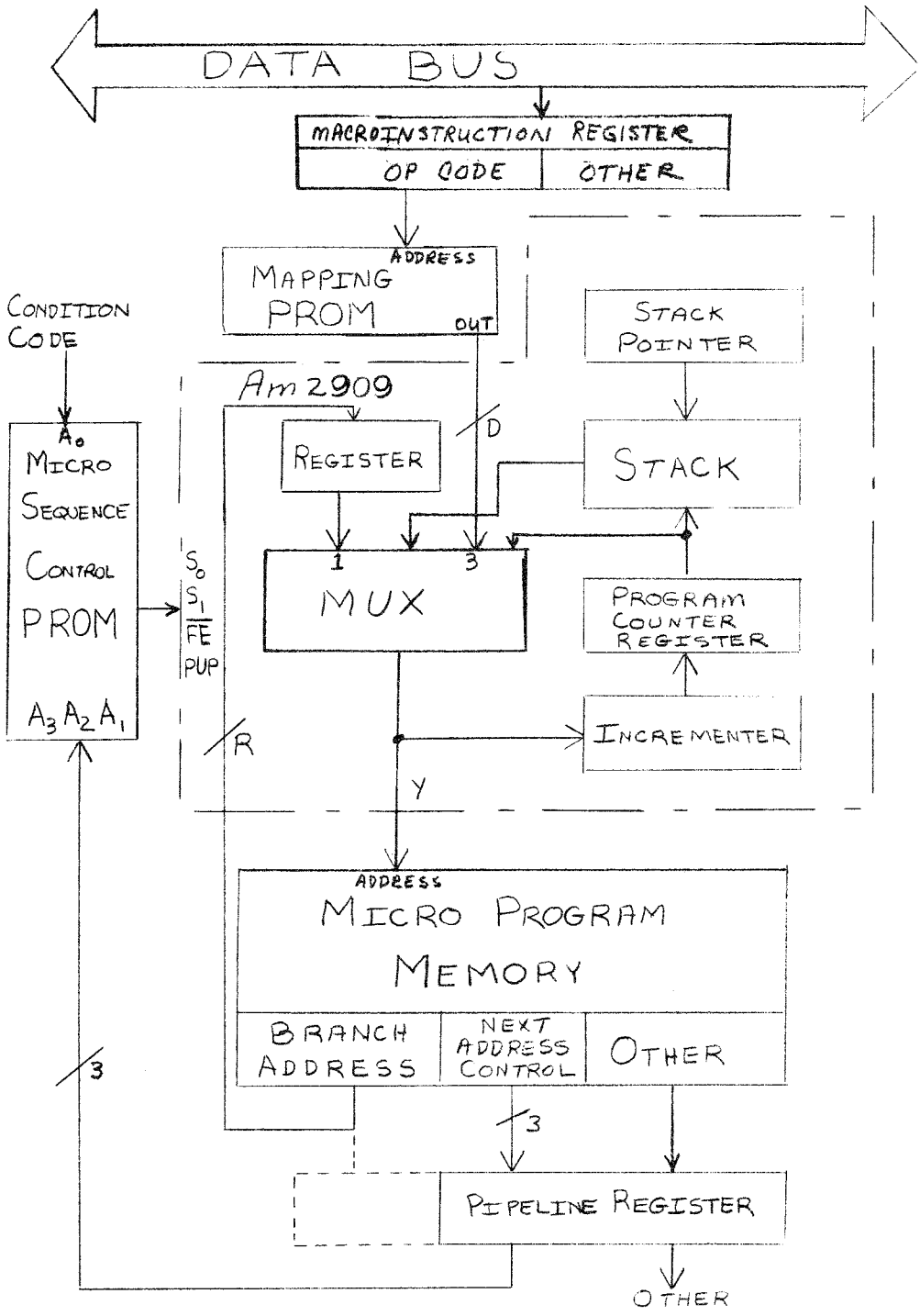

FIGURE 12: A Typical Computer Control Unit Architecture Using Am2909

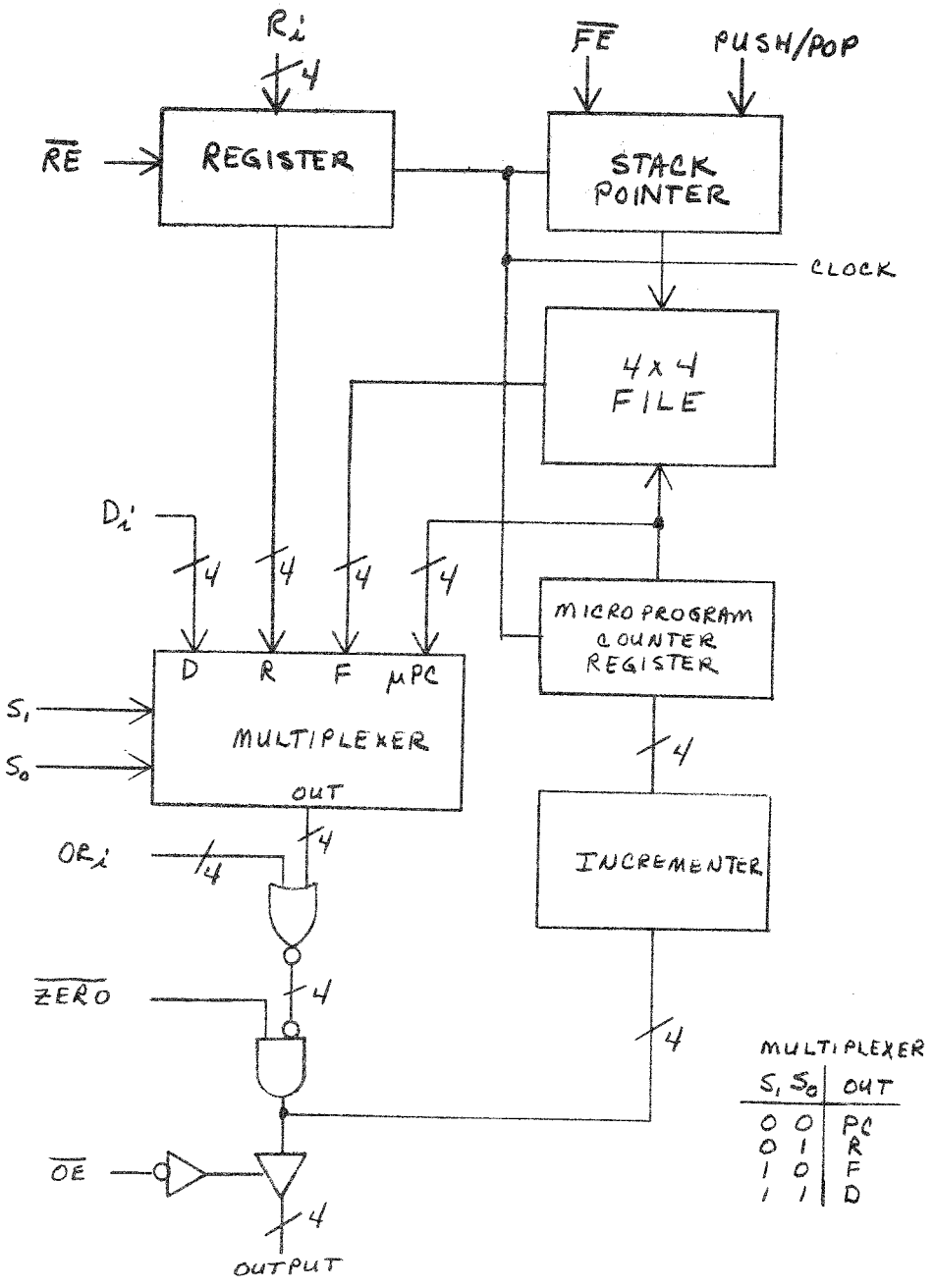

FIGURE 13: Detailed Logic Diagram of the Am2909

The register internal to the Am2909 should be thought of as the pipeline register. When the current microinstruction being executed is selecting the next microinstruction address as a branch function, the branch address will be contained in the register internal to the Am2909. When sequencing through continuous instruction in microprogram memory, the program counter in the Am2909 is used. The 4 x 4 stack in the Am2909 is used for looping and subroutining in microprogram operations.

If a three-bit field for the next address control is used for the architecture shown in Figure 12, the recommended eight functions are shown in Table IV. The eight functions providing good control for selection of the address of the next microinstruction include continue, branch, branch on condition, push, jump-to-subroutine, starting address, return-from-subroutine, and test end of loop. The detailed control required to implement these eight functions is shown in Table V. The actual PROM coding for this example is shown in Table VI. The PROM selected for this implementation is an Am29751 32-word by 8-bit Programmable Read Only Memory.

If more flexibility is required, the architecture shown in Figure 12 can be expanded such that a four-bit next address field format is used to drive the microsequencer control PROM. This provides the ability to have 16 different control functions for selecting the address of the next microinstruction. Table VII shows an example of these 16 functions providing a comprehensive set of instructions for microprogram address control. Table VIII shows the actual PROM coding for this instruction set and Figure 14 shows a detailed functional connection diagram for this microsequencer instruction set.

Am2911 MICROPROGRAM SEQUENCER

The Am2911 Microprogram Sequencer is similar in design to the Am2909 device. The difference between the Am2909 and the Am2911 involves the D inputs, R inputs, and OR inputs. On the Am2911, the OR inputs have been eliminated to save four pins. Also on the Am2911, the R inputs and D inputs have been connected together to save an additional four pins. This allows the 28-pin Am2909 to be packaged in a 20-pin package.

When using the Am2911, the normal technique in a computer control unit is to multiplex the output of the mapping PROM and the output of the pipeline register for the branch address. This is easily accomplished using three-state control at each of these points. The enable signal or the mapping PROM and the branch address field three-state control can be supplied from the Am29751 Control PROM associated with the microprogram sequencer.

TABLE IV

Next Address Select Control Functions

$A_3A_2A_1$	Function
000	Continue
001	Branch
010	Branch on condition
011	Push
100	Jump to subroutine
101	Starting address
110	Return from subroutine
111	Test end of loop

TABLE V

PROM Control for Table IV Function

Function	$A_3A_2A_1$	CC A_0	Next Address MUX S_1S_0	Enable Stack	Push/ Pop
Continue	000	0 1	0	No	X
Branch	001	0 1	1	No	X
Conditional Branch	010	0 1	0 1	No No	X X
Push	011	0 1	0	Yes	Push
Jump to Subroutine	100	0 1	1	Yes	Push
Starting Address	101	0 1	3	No	X
Return	110	0 1	2	Yes	Pop
Test end of loop	111	0 1	2 0	No Yes	X Pop

TABLE VI

Am29751 PROM Coding for Table V Example

Address $A_4A_3A_2A_1A_0$	PROM Output O_3 O_2 O_1 O_0
0 0 0 0 0	X 1 0 0
0 0 0 0 1	X 1 0 0
0 0 0 1 0	X 1 0 1
0 0 0 1 1	X 1 0 1
0 0 1 0 0	X 1 0 0
0 0 1 0 1	X 1 0 1
0 0 1 1 0	1 0 0 0
0 0 1 1 1	1 0 0 0
0 1 0 0 0	1 0 0 1
0 1 0 0 1	1 0 0 1
0 1 0 1 0	X 1 1 1
0 1 0 1 1	X 1 1 1
0 1 1 0 0	0 0 1 0
0 1 1 0 1	0 0 1 0
0 1 1 1 0	X 0 1 0
0 1 1 1 1	0 1 0 0

0 = LOW 1 = HIGH

X = Don't care

Notes:

1. Am2909 Connections:

PROM O_0 to S_0 (Pin 16)

PROM O_1 to S_1 (Pin 17)

PROM O_2 to $\overline{FE}$ (Pin 25)

PROM O_3 to PUP (Pin 26)

2. Ground A_4 (Pin 14) of Am29751

TABLE VII

PROM Control for 16 Next Address Functions

Function	$A_3A_2A_1A_0$	CC	Next Address	Stack Enable	Push Pop	Zero	OR
Start Address	0000	0 1	D	No	X	H	L
Continue	0001	0 1	PC	No	X	H	L
Branch	0010	0 1	R	No	X	H	L
Jump zero	0011	0 1	X	No	X	L	X
Push	0100	0 1	PC	Yes	Push	H	L
Pop	0101	0 1	PC	Yes	Pop	H	L
JSB	0110	0 1	R	Yes	Push	H	L
RTS	0111	0 1	F	Yes	Pop	H	L
Test end of loop	1000	0 1	F PC	No Yes	X Pop	H H	L
Conditional Branch	1001	0 1	PC R	No	X	H	L
Conditional JSB	1010	0 1	PC R	No Yes	X Push	H	L
Conditional RTS	1011	0 1	PC F	No Yes	X Pop	H	L
Conditional Jump MAX	1100	0 1	PC X	No	X	H H	L H
Jump MAX	1101	0 1	X	No	X	H	H
Conditional Jump Zero	1110	0 1	PC X	No No	X X	H L	L
	1111	0 1	TO BE DETERMINED BY USER				

TABLE VIII

Am29751 PROM Coding for Table VII Example

Address $A_4A_3A_2A_1A_0$	PROM Output O_5 O_4 O_3 O_2 O_1 O_0
0 0 0 0 0	0 1 X 1 1 1
0 0 0 0 1	0 1 X 1 1 1
0 0 0 1 0	0 1 X 1 0 0
0 0 0 1 1	0 1 X 1 0 0
0 0 1 0 0	0 1 X 1 0 1
0 0 1 0 1	0 1 X 1 0 1
0 0 1 1 0	0 0 X 1 X X
0 0 1 1 1	0 0 X 1 X X
0 1 0 0 0	0 1 1 0 0 0
0 1 0 0 1	0 1 1 0 0 0
0 1 0 1 0	0 1 0 0 0 0
0 1 0 1 1	0 1 0 0 0 0
0 1 1 0 0	0 1 1 0 0 1
0 1 1 0 1	0 1 1 0 0 1
0 1 1 1 0	0 1 0 0 1 0
0 1 1 1 1	0 1 0 0 1 0
1 0 0 0 0	0 1 X 1 1 0
1 0 0 0 1	0 1 0 0 0 0
1 0 0 1 0	0 1 X 1 0 0
1 0 0 1 1	0 1 X 1 0 1
1 0 1 0 0	0 1 X 1 0 0
1 0 1 0 1	0 1 1 0 0 1
1 0 1 1 0	0 1 X 1 0 0
1 0 1 1 1	0 1 0 0 1 0
1 1 0 0 0	0 1 X 1 0 0
1 1 0 0 1	1 1 X 1 X X
1 1 0 1 0	1 1 X 1 X X
1 1 0 1 1	1 1 X 1 X X
1 1 1 0 0	0 1 X 1 0 0
1 1 1 0 1	0 0 X 1 X X
1 1 1 1 0	To be determined
1 1 1 1 1	by user

0 = LOW 1 = HIGH X = Don't care

Am2909 Connections:

PROM O_0 to S_0 (Pin 16)
PROM O_1 to S_1 (Pin 17)
PROM O_2 to $\overline{FE}$ (Pin 25)
PROM O_3 to PUP (Pin 26)
PROM O_4 to Zero (Pin 15)
PROM O_5 to OR_0, OR_1, OR_2, OR_3 (Pins 12, 10, 8 and 6)

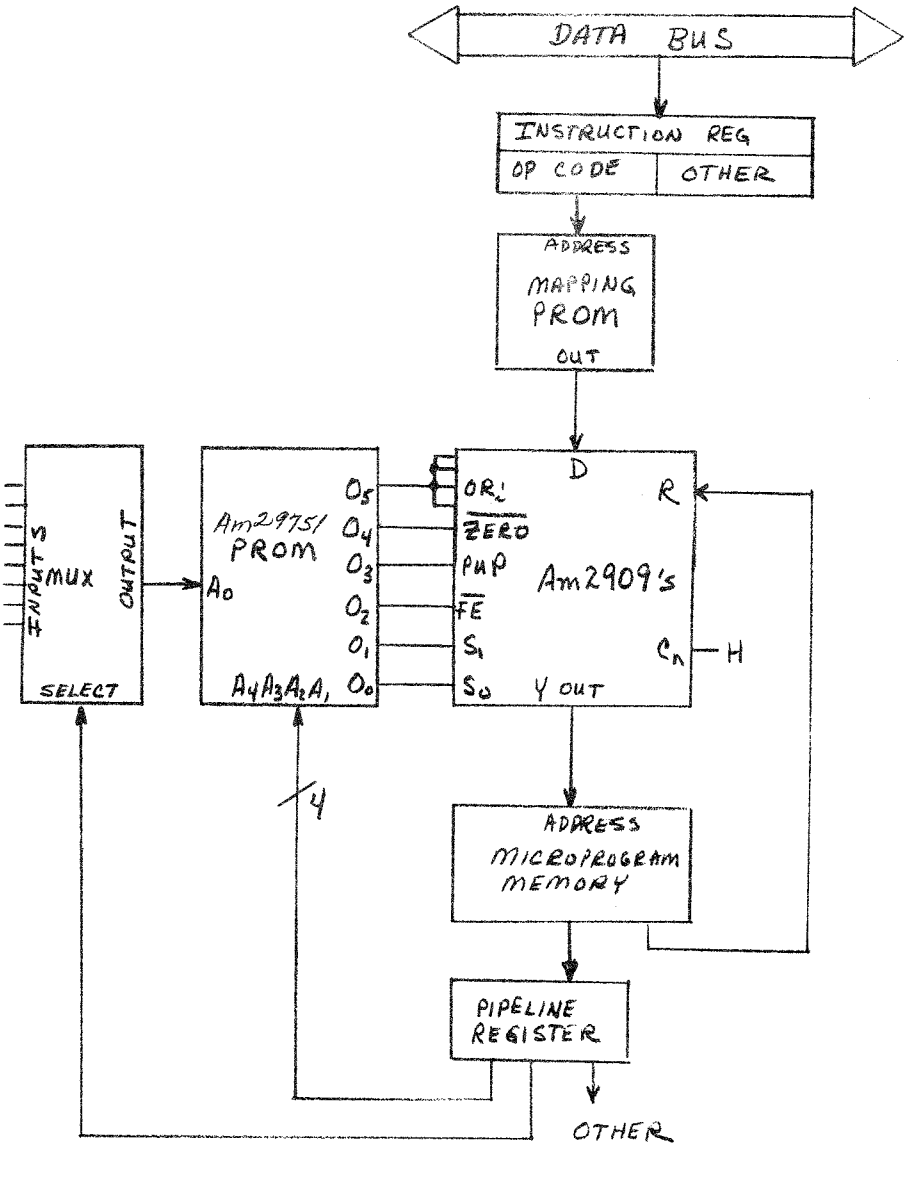

FIGURE 14: Functional Connection Diagram for Typical Computer Control Unit Using Four-bit Next Address Control Field

FORMATTING IN MICROPROGRAM MEMORY

Many engineers designing a microprogrammed machine for the first time find difficulty in determining an approach to the microprogramming problem. At Advanced Micro Devices, we have found the following approach to be the most useful for engineers attempting their first microprogramming design job. This approach involves initially assuming only one format for the microprogram word will be used. That is, the machine architecture should be determined as required for the performance desired and the total word width of the microprogram memory temporarily ignored. This can result in a fairly wide microprogram word at the onset of the design. However, by taking this approach, the design engineer does not become immediately bogged down in making several trade-offs as to the various formats useful for the microprogram words. If the machine architecture is designed to achieve the performance required and the microprogram word is expanded such that bits are available to control the various portions of the machine as required, then the initial design task becomes straight-forward.

After the initial architecture has been laid out, the design should proceed such that small groups of microinstructions are written for the key macroinstructions. Particular emphasis should be placed on the macroinstruction set requiring the highest use in the firmware operation. For example, particular emphasis should be placed on the fetch routine since it is used over and over on every machine instruction cycle. Likewise, if the machine will typically execute many register-to-register arithmetic operations, the microprogram sequence used in these instructions should be reviewed carefully. As small sequences of firmware are written, it will become apparent that perhaps some fields of the microprogram could be shared in the memory. For example, it may become apparent that the interrupt control microprogram bits and the A source operand microprogram bits never occur in the same microinstruction. Thus, perhaps if four bits are used for interrupt control and four bits are used for A source operand select, then these four bits might use the same four-bit field in the memory. This would save a four-bit field throughout the entire microprogram memory addressing space. However, it will be necessary to determine which field is present in the word in the pipeline register. Thus, one bit may be dedicated in the entire microprogram word field to distinguish between format number 1 and format number 2. When this bit is a zero, the machine recognizes format 1 and applies these four bits to the source operand control. When this bit is a logic 1, the machine recognizes that format number 2 is in the format register and applies this field to the interrupt control.

From this discussion, it should be apparent that perhaps four different formats might be utilized by the machine and a two-bit field used to identify which of the formats is in the pipeline register. In some designs, parts of the pipeline register may be repeated more than one time for some control fields. If this is the case, then it becomes necessary to select the appropriate register to be loaded by the microinstruction coming from the memory for this field. Thus, the two bits determining the format may be decoded using a two-line to four-line decoder such as the Am25LS139. The appropriate output from the Am25LS139 can enable a register with a clock enable function such as on the Am25LS07, Am25LS08 or Am25LS377 such that the microprogram memory word is loaded into the appropriate register for use by the machine.

The primary advantage of formatting the microprogram word is to save bits in width for the microprogram memory. Occasionally, as the formatting becomes more complex, it will be found that two fields are required simultaneously. Thus, this can result in an extra microinstruction being required to be able to perform all of the functions required by the microprogram control. Normally, anywhere from two to eight different formats can be found useful in the microprogramming of many machines.

For the design engineer doing his first microprogramming job, the emphasis should be on layout of the architecture of the machine to meet the required specification. Should the design accidentally use too many bits in microword width, it would be unfortunate but not catastrophic. The machine would still perform properly and meet specification and, in many cases, only one or two extra integrated circuits would have been used. Based on the cost of Field Programmable Read Only Memories, this cost is not unacceptable. Also, it is possible that such an approach may considerably reduce the overall design time of the machine. As the designer gains experience, he will become more proficient in formatting microprogram words and will soon find he can generate microprogram control architectures with reasonable speed.

SUMMARY

The Am2909 provides a unique solution to the microprogram memory sequence control problem. It is particularly well suited for high-performance computer control units using overlap fetch of the next address in microprogram memory. By using a microsequencer control PROM, the functional control of the next address selection can be uniquely defined. In addition, operations involving the internal stack can be selected as desired. The Am2909 is also a useful device in state machine control design. That is, control of the microprogram memory can be programmed as needed for the design.

The Am2911, being similar in design to the Am2909, provides a similar type of control in the 20-pin, 0.3" centers package. This package provides lower cost because of its reduced size and complexity. In addition, this package requires less printed circuit board area than the 28-pin Am2909 but provides almost the same functional capability.