UNIT-II
Microprogrammed Control

Control Memory

Introduction

- The major functional parts in a digital computer are Central Processing Unit (CPU), Memory, and Input–output.
- The main digital hardware functional units of CPU are control unit, arithmetic and logic unit, and registers.
- **The function of the control unit in a digital computer is to initiate sequences of microoperations.**
- **The number of different types of microoperations that are available in a given system is finite.**
- **The complexity of the digital system is derived from the number of sequences of microoperations that are performed.**

Implementing Control Unit

- Two methods of implementing control unit are:

1. Hardwired control and
2. Microprogrammed control.

Hardwired Control

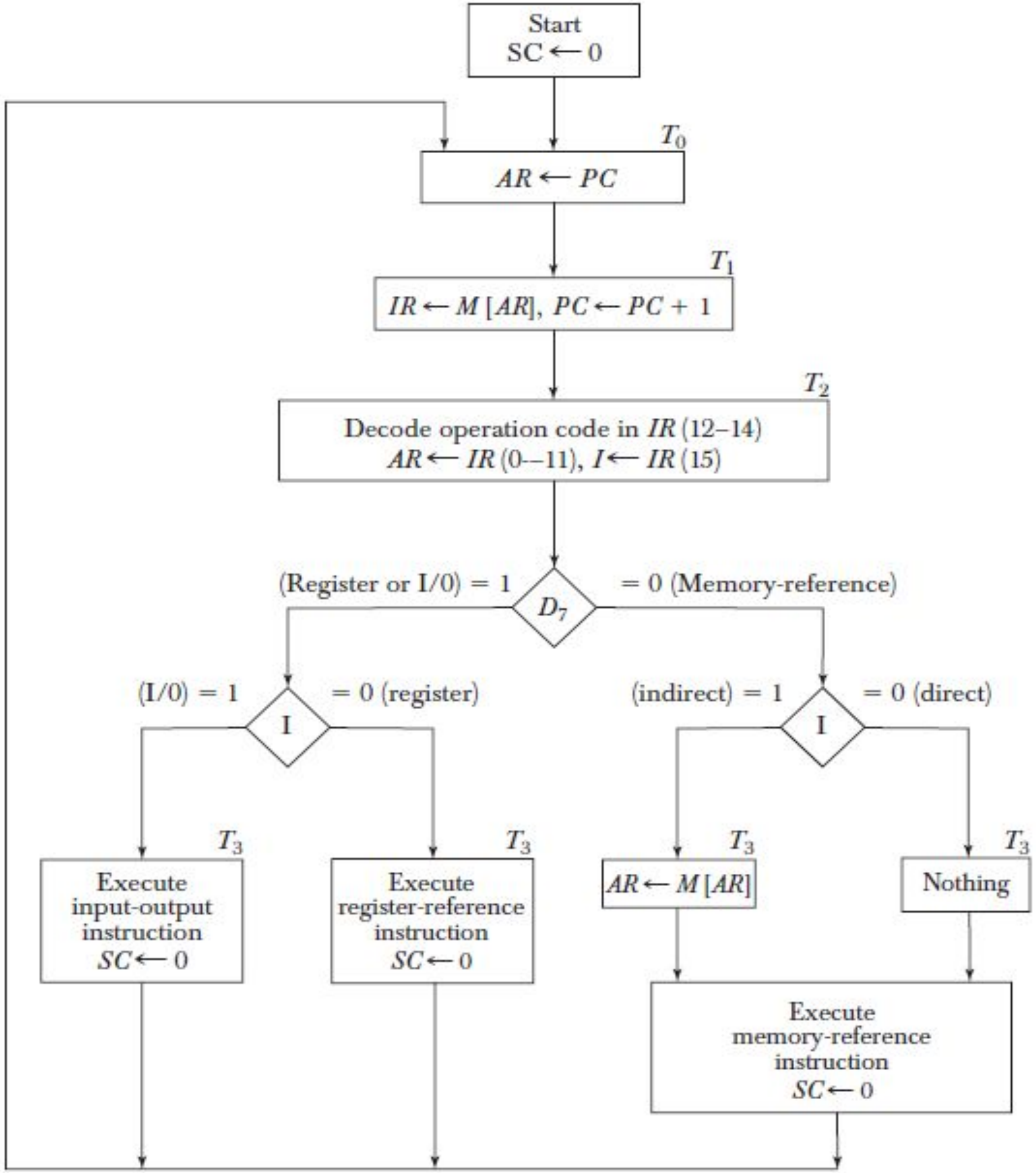
- The design of hardwired control involves the use of fixed instructions, fixed logic blocks of and/or arrays, encoders, decoders, etc.
- **The key characteristics of hardwired control logic are high-speed operation, expensive, relatively complex, and no flexibility of adding new instructions.**
- Example CPUs with hardwired logic control are Intel 8085, Motorola 6802, Zilog 80, and any RISC (Reduced Instruction Set Computer) CPUs.
- **When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.**

Microprogramming

- Microprogramming is a second alternative for designing the control unit of a digital computer.
- **The principle of microprogramming is an elegant and systematic method for controlling the microoperation sequences in a digital computer.**
- For example, CPUs with microprogrammed control unit are Intel 8080, Motorola 68000, and any CISC (Complex Instruction Set Computer) CPUs.

Introduction

- The control function that specifies a microoperation is a binary variable.
- When it is in one binary state, the corresponding microoperation is executed.
- A control variable in the opposite binary state does not change the state of the registers in the system.
- The active state of a control variable may be either the 1 state or the 0 state, depending on the application.
- In a bus-organized system, the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.



Start
SC ← 0
T_0
$AR \leftarrow PC$
T_1
$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2
Decode operation code in IR (12–14)
$AR \leftarrow IR$ (0–11), $I \leftarrow IR$ (15)
(Register or I/0) = 1
D_7
= 0 (Memory-reference)
(I/0) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T_3
Execute input-output instruction $SC \leftarrow 0$
T_3
Execute register-reference instruction $SC \leftarrow 0$
T_3
$AR \leftarrow M[AR]$
T_3
Nothing
Execute memory-reference instruction $SC \leftarrow 0$

Control Word

- The control unit initiates a series of sequential steps of microoperations.
- During any given time, certain microoperations are to be initiated, while others remain idle.
- The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- As such, control words can be programmed to perform various operations on the components of the system.

Microinstruction and Microprogram

- A control unit whose binary control variables are stored in memory is called a microprogrammed control unit.
- Each word in control memory contains within it a microinstruction.
- **The microinstruction specifies one or more microoperations for the system.**
- **A sequence of microinstructions constitutes a microprogram.**

How to Store Microprogram

- Since alterations of the microprogram are not needed once the control unit is in operation, the control memory can be a read-only memory (ROM).
- The content of the words in ROM are fixed and cannot be altered by simple programming since no writing capability is available in the ROM.
- ROM words are made permanent during the hardware production of the unit.
- The use of a microprogram involves placing all control variables in words of ROM for use by the control unit through successive read operations.
- The content of the word in ROM at a given address specifies a microinstruction.

Dynamic Microprogramming

- A more advanced development known as dynamic microprogramming permits a microprogram to be loaded initially from an auxiliary memory such as a magnetic disk.
- Control units that use dynamic microprogramming employ a writable control memory.
- This type of memory can be used for writing (to change the microprogram) but is used mostly for reading.

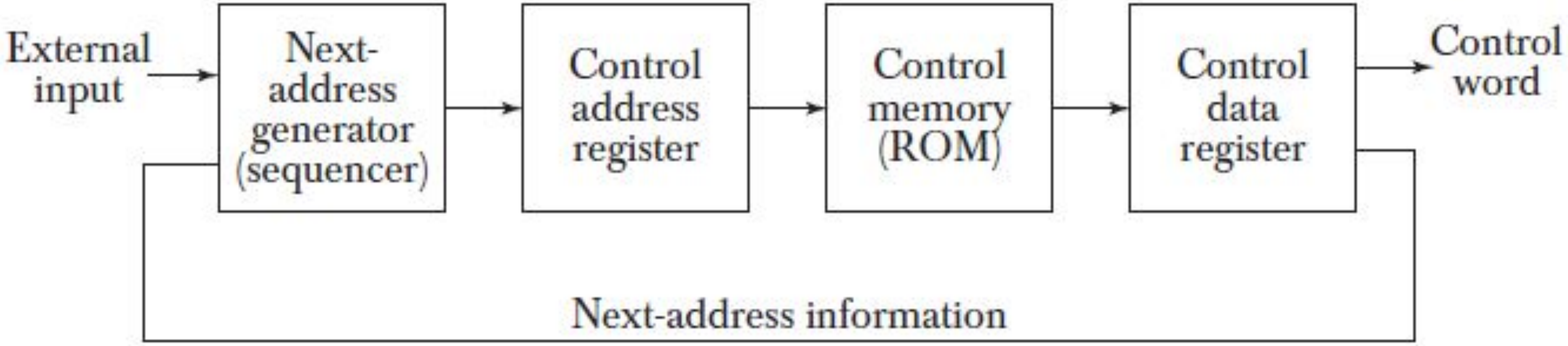
Control Memory

- A memory that is part of a control unit is referred to as a control memory.
- A computer that employs a microprogrammed control unit will have two separate memories: **a main memory and a control memory.**
- The main memory is available to the user for storing the programs.
- The contents of main memory may alter when the data are manipulated and every time that the program is changed.
- The user's program in main memory consists of machine instructions and data.
- In contrast, the control memory holds a fixed microprogram that cannot be altered by the occasional user.

Microprogram

- The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations.
- Each machine instruction initiates a series of microinstructions in control memory.
- These microinstructions generate the microoperations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to repeat the cycle for the next instruction.

Microprogrammed control organization.



Explanation

- The control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more microoperations for the data processor.
- Once these operations are executed, the control must determine the next address.

Explanation

- The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory.
- For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.
- The next address may also be a function of external input conditions.
- While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

Finally,

- Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.

sequencer

- The next address generator is sometimes called a microprogram sequencer, as it determines the address sequence that is read from control memory.
- The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs.
- Typical functions of a microprogram sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.

pipeline register

- The control data register holds the present microinstruction while the next address is computed and read from memory.
- The data register is sometimes called a pipeline register.
- It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.
- This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.

Another Approach

- The system can operate without the control data register by applying a single-phase clock to the address register.
- The control word and next-address information are taken directly from the control memory.
- It must be realized that a ROM operates as a combinational circuit, with the address value as the input and the corresponding word as the output.
- The content of the specified word in ROM remains in the output wires as long as its address value remains in the address register.
- No read signal is needed as in a random-access memory.
- **Each clock pulse will execute the microoperations specified by the control word and also transfer a new address to the control address register.**

Note

- In the example that follows we assume a single-phase clock and therefore we do not use a control data register.
- **In this way the address register is the only component in the control system that receives clock pulses.**
- **The other two components: the sequencer and the control memory are combinational circuits and do not need a clock.**

Advantage (Microprogrammed Control)

- The main advantage of the microprogrammed control is the fact that once the hardware configuration is established, there should be no need for further hardware or wiring changes.
- If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.
- The hardware configuration should not be changed for different operations; the only thing that must be changed is the microprogram residing in control memory.

Address Sequencing

Routine

- Microinstructions are stored in control memory in groups, with each group specifying a routine.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction.
- The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another.

The steps that the control must undergo during the execution of a single computer instruction

STEP-1

- An initial address is loaded into the control address register when power is turned on in the computer.
- This address is usually the address of the first microinstruction that activates the instruction fetch routine.
- The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.
- At the end of the fetch routine, the instruction is in the instruction register of the computer.

STEP-2

- The control memory next must go through the routine that determines the effective address of the operand.
- A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.
- The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
- When the effective address computation routine is completed, the address of the operand is available in the memory address register.

STEP-3 (Mapping)

- The next step is to generate the microoperations that execute the instruction fetched from memory.
- The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
- Each instruction has its own microprogram routine stored in a given location of control memory.
- **The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a mapping process.**

STEP-3 (Mapping)

- A mapping procedure is a rule that transforms the instruction code into a control memory address.
- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register, but sometimes the sequence of microoperations will depend on values of certain status bits in processor registers.

STEP-4

- Microprograms that employ subroutines will require an external register for storing the return address.
- Return addresses cannot be stored in ROM because the unit has no writing capability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.

Summary

In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.

2. Unconditional branch or conditional branch, depending on status bit conditions.

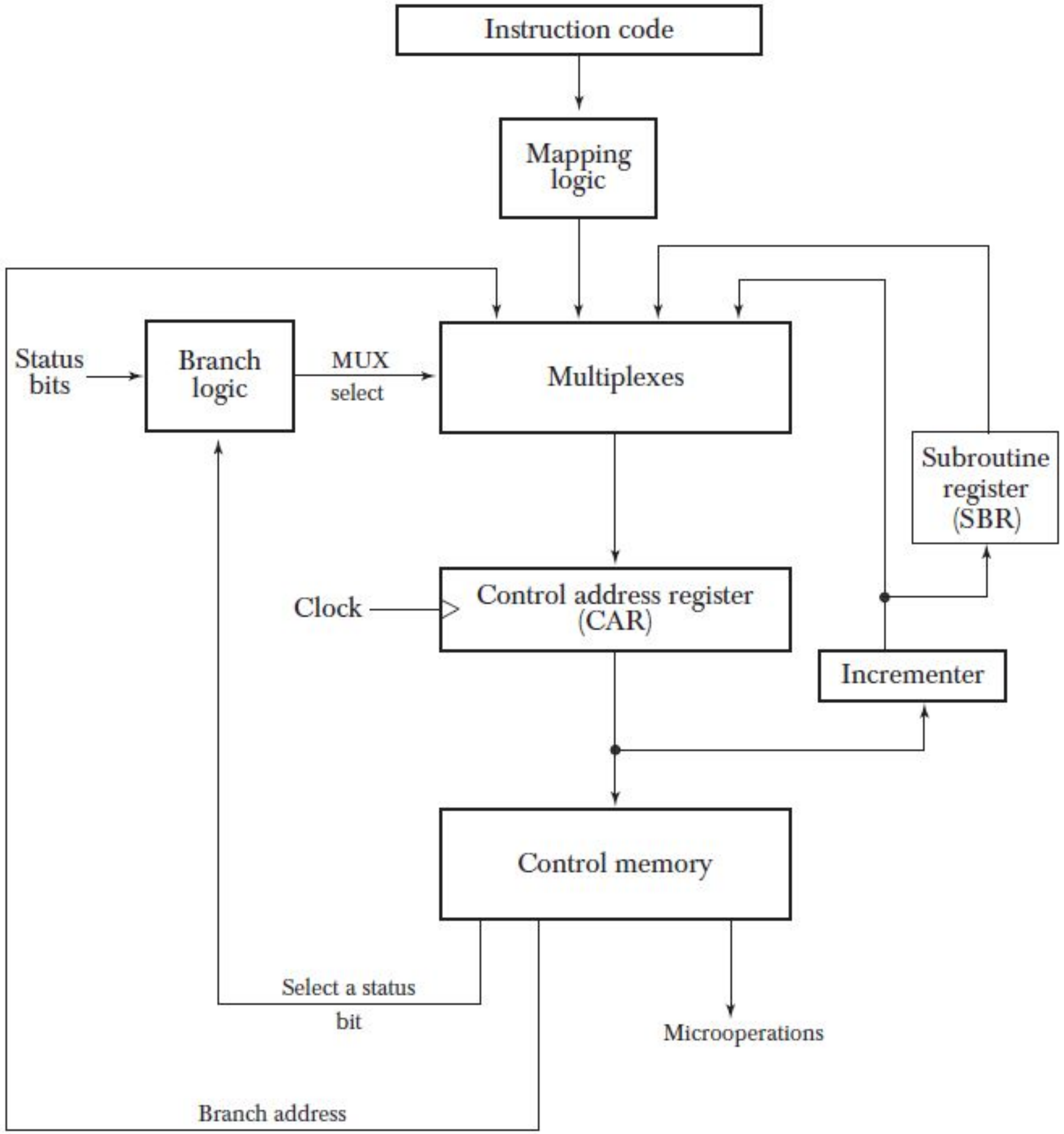
3. A mapping process from the bits of the instruction to an address for control memory.

4. A facility for subroutine call and return.

NOTE

- The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.

Block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address



The diagram shows four different paths from which the control address register (CAR) receives the address.

The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.

Branching is achieved by specifying the branch address in one of the fields of the microinstruction.

Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.

An external address is transferred into control memory via a mapping logic circuit.

The return address for a subroutine is stored in a special register whose value is then used

Conditional Branching

- The branch logic provides decision-making capabilities in the control unit.
- The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions.
- Information in these bits can be tested and actions initiated based on their condition: whether their value is 1 or 0.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

branch logic

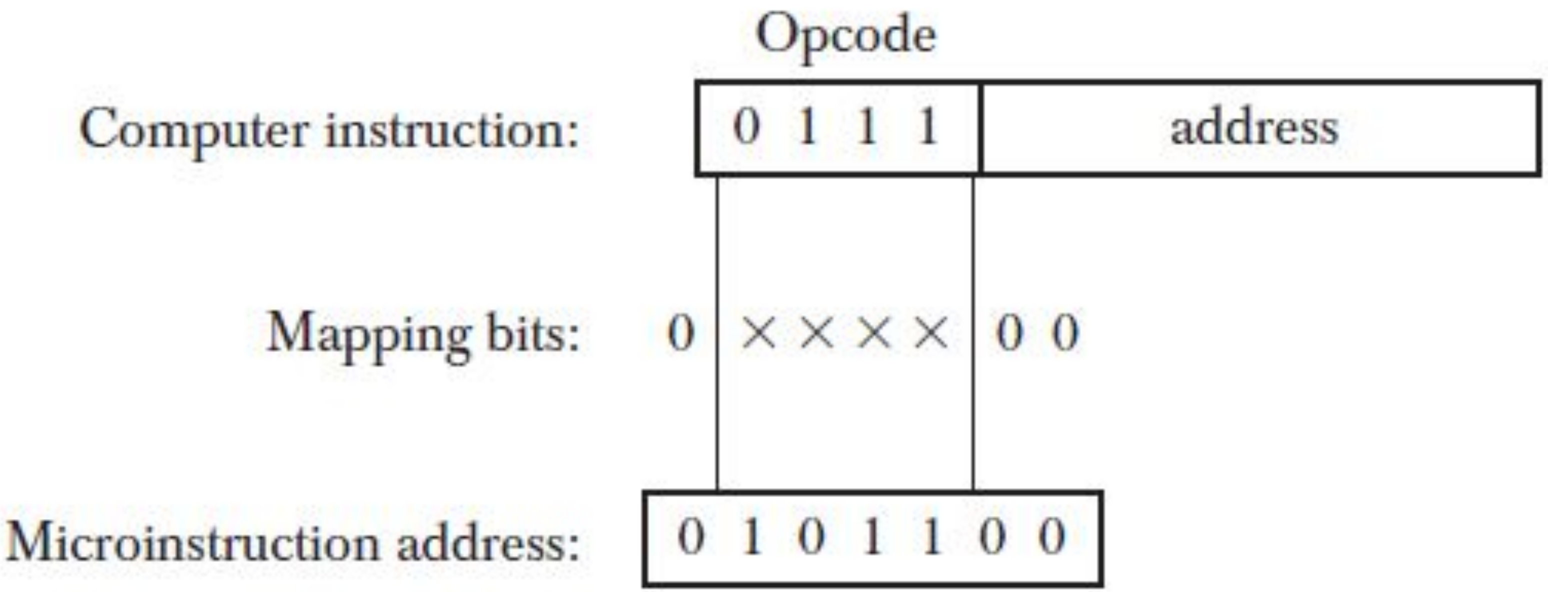
- The branch logic hardware may be implemented in a variety of ways.
- The simplest way is to test the specified condition and branch to the indicated address if the condition is met; otherwise, the address register is incremented.
- This can be implemented with a multiplexer.
- Suppose that there are eight status bit conditions in the system.
- Three bits in the microinstruction are used to specify any one of eight status bit conditions.
- These three bits provide the selection variables for the multiplexer.
- If the selected status bit is in the 1 state, the output of the multiplexer is 1; otherwise, it is 0.
- A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register.
- A 0 output in the multiplexer causes the address register to be incremented.

branch logic

- In this configuration, the microprogram follows one of two possible paths, depending on the value of the selected status bit.
- An unconditional branch microinstruction can be implemented by loading the branch address from control memory into the control address register.
- This can be accomplished by fixing the value of one status bit at the input of the multiplexer, so it is always equal to 1.
- A reference to this bit by the status bit select lines from control memory causes the branch address to be loaded into the control address register unconditionally.

Mapping of Instruction

Mapping from instruction code to microinstruction address.



Mapping of Instruction

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
- The status bits for this type of branch are the bits in the operation code part of the instruction.
- For example, a computer with a simple instruction format as shown in Fig. has an operation code of four bits which can specify up to 16 distinct instructions.
- Assume further that the control memory has 128 words, requiring an address of seven bits.
- For each operation code there exists a microprogram routine in control memory that executes the instruction.
- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown.

Mapping of Instruction

- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.
- If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

Mapping of Instruction

- One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function.
- In this configuration, the bits of the instruction specify the address of a mapping ROM.
- The contents of the mapping ROM give the bits for the control address register.
- In this way the microprogram routine that executes the instruction can be placed in any desired location in control memory.
- The mapping concept provides flexibility for adding instructions for control memory as the need arises.

Mapping of Instruction

- The mapping function is sometimes implemented by means of an integrated circuit called programmable logic device or PLD.
- A PLD is similar to ROM in concept except that it uses AND and OR gates with internal electronic fuses.
- The interconnection between inputs, AND gates, OR gates, and outputs can be programmed as in ROM. A mapping function that can be expressed in terms of Boolean expressions can be implemented conveniently with a PLD.

Subroutines

- Subroutines are programs that are used by other routines to accomplish a particular task.
- A subroutine can be called from any point within the main body of the microprogram.
- Frequently, many microprograms contain identical sections of code.
- Microinstructions can be saved by employing subroutines that use common sections of microcode.
- For example, the sequence of microoperations needed to generate the effective address of the operand for an instruction is common to all memory reference instructions.
- This sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Subroutine Register

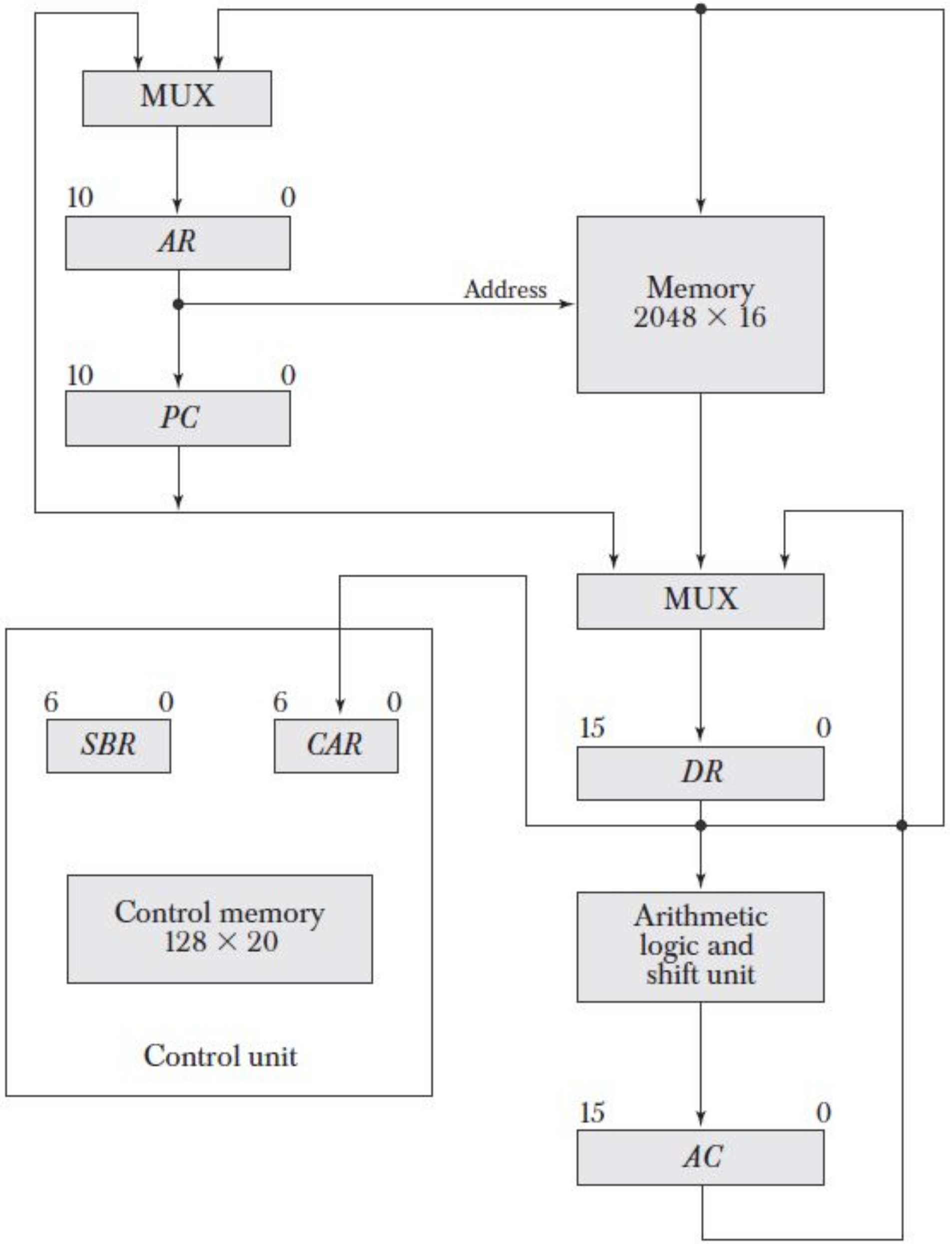
- Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return.
- This may be accomplished by placing the incremented output from the control address register into a subroutine register and branching to the beginning of the subroutine.
- The subroutine register can then become the source for transferring the address for the return to the main routine.
- The best way to structure a register file that stores addresses for subroutines is to organize the registers in a last-in, first-out (LIFO) stack.

Microprogram Example

Introduction

- Once the configuration of a computer and its microprogrammed control unit is established, the designer’s task is to generate the microcode for the control memory.
- This code generation is called microprogramming and is a process similar to conventional machine language programming.
- To appreciate this process, we present here a simple digital computer and show how it is microprogrammed.

Computer Hardware Configuration

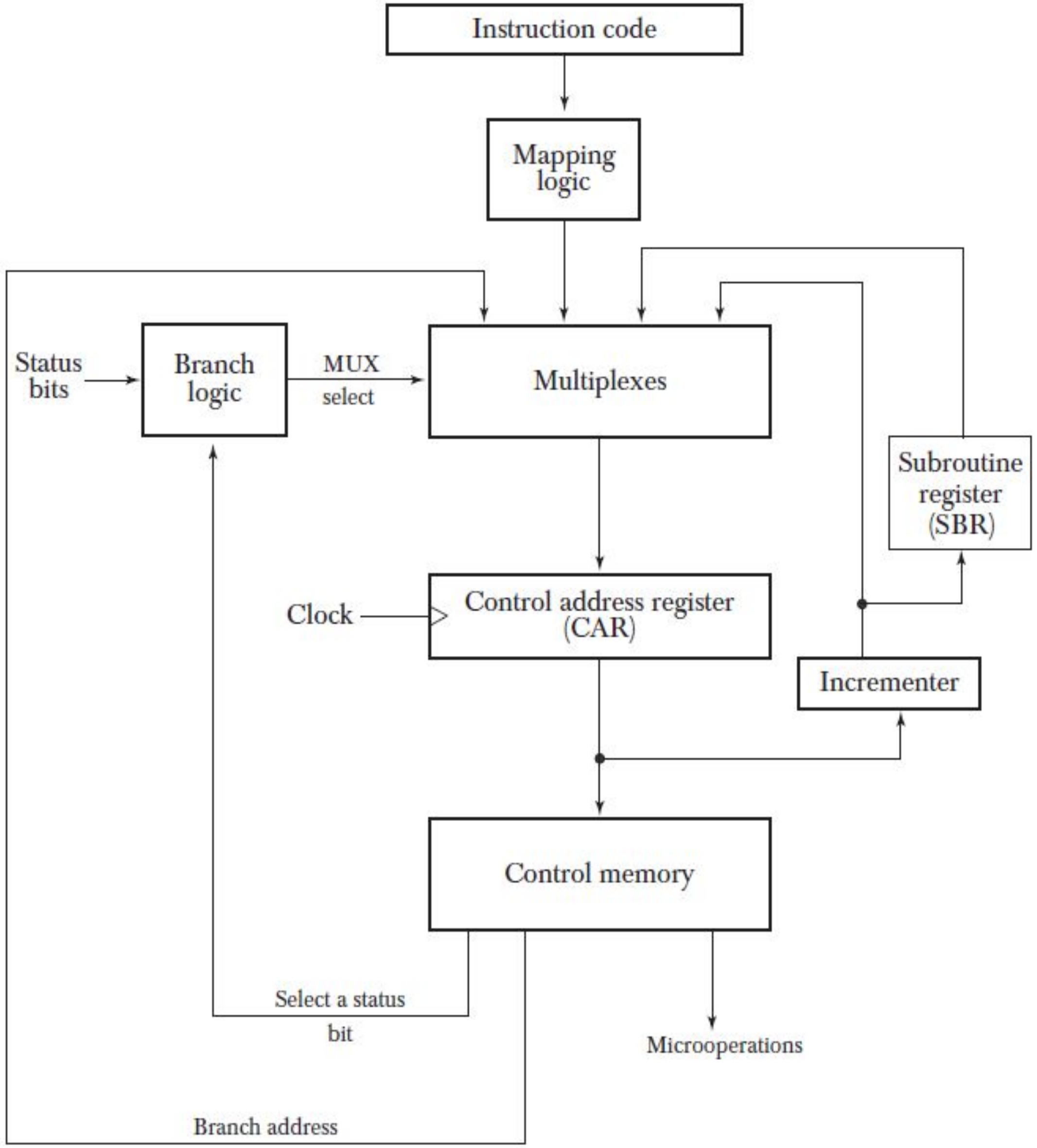


It consists of two memory units: a main memory for storing instructions and data, and a control memory for storing the microprogram.

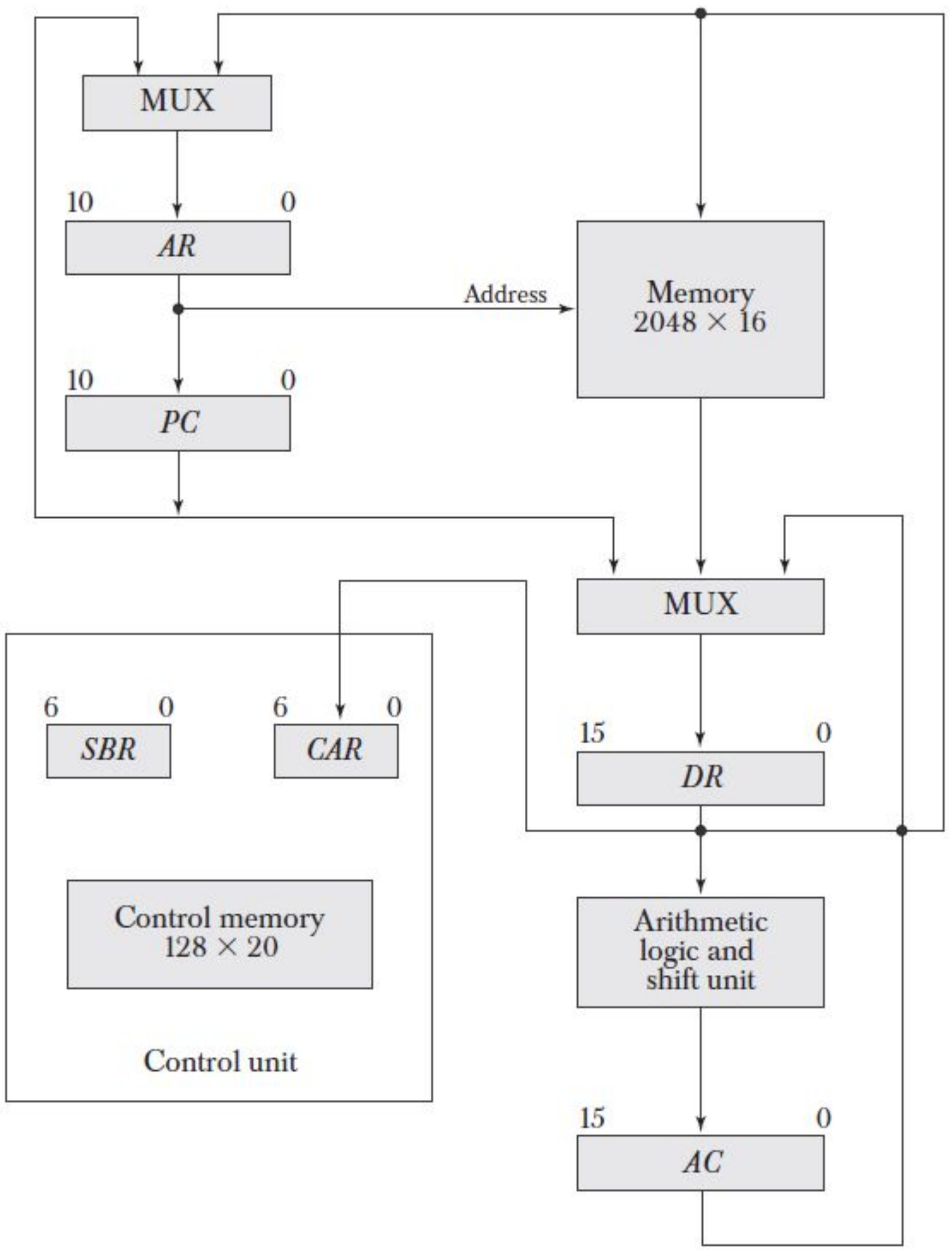
Four registers are associated with the processor unit and two with the control unit.

The processor registers are program counter *PC*, address register *AR*, data register *DR*, and accumulator register *AC*.

The control unit has a control address register *CAR* and a subroutine register *SBR*.



The control memory and its registers are organized as a microprogrammed control unit



The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

DR can receive information from *AC*, *PC*, or memory.

AR can receive information from *PC* or *DR*. *PC* can receive information only from *AR*.

The arithmetic, logic, and shift unit performs microoperations with data from *AC* and *DR* and places the result in *AC*.

Note that memory receives its address from *AR*. Input data written to memory come from *DR*, and data read from memory can go only to *DR*.

Instruction Format

15	14 … 11	10 … 0
I	Opcode	Address

(a) Instruction format

Symbol	Opcode	Description
ADD	0000	$AC \rightarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA]$, $M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

The computer instruction format is depicted in Fig. (a).

Instruction Format consists of three fields: a 1-bit field for indirect addressing symbolized by *I*, a 4-bit operation code (opcode), and an 11-bit address field.

Figure (b) lists four of the 16 possible memory-reference instructions.
-The ADD instruction adds the content of the operand found in the effective address to the content of *AC*.
-The BRANCH instruction causes a branch to the effective address if the operand in *AC* is negative. The program proceeds with the next consecutive instruction if *AC* is not negative. The *AC* is negative if its sign bit (the bit in the leftmost position of the register) is a 1.

15	14	11	10	0
I	Opcode		Address	

(a) Instruction format

Symbol	Opcode	Description
ADD	0000	$AC \rightarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

(b) Four computer instructions

-The STORE instruction transfers the content of *AC* into the memory word specified by the effective address.

-The EXCHANGE instruction swaps the data between *AC* and the memory word specified by the effective address.

"Each computer instruction must be microprogrammed."

Microinstruction code format (20 bits)

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

The microinstruction format for the control memory is shown.

The 20 bits of the microinstruction are divided into four functional parts.

The three fields Fl, F2, and F3 specify microoperations for the computer.

The CD field selects status bit conditions.

The BR field specifies the type or branch to be used.

The AD field contains a branch address.

The address field is seven bits wide, since the control memory has 128 $=2^7$ words.

Symbols and Binary Code for Microinstruction Fields

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

The microoperations are subdivided into three fields of three bits each.
The three bits in each field are encoded to specify seven distinct microoperations
This gives a total of 21 microoperations.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow AC$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

No more than three microoperations can be chosen for a microinstruction, one from each field.

If fewer than three microoperations are used, one or more of the
fields will use the binary code 000 for no operation.

As an il… specify two sim… F2 and F3 and r…

$$DR \leftarrow M[AR] \quad \text{with F2} = 100$$

and

$$PC \leftarrow PC + 1 \quad \text{with F3} = 101$$

The nine bits of the microoperation fields will then be 000 100 101.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow AC$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

It is important to realize that two or more conflicting microoperations cannot be specified simultaneously.

For example, a microoperation field 010 001 000 has no meaning because it specifies the operations to clear *AC* to 0 and subtract *DR* from *AC* at the same time.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow AC$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Each microoperation in is defined with a register transfer statement and is assigned a symbol for use in a symbolic microprogram.

All transfer- type microoperations symbols use five letters.

The first two letters designate the source register, the third letter is always a T, and the last two letters designate the destination register.

For example, the microoperation that specifies the transfer $AC \leftarrow DR$ (F1 =100) has the symbol DRTAC, which stands for a transfer from DR to AC.

condition field

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

The *CD* (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	*DR*(15)	I	Indirect address bit
10	*AC*(15)	S	Sign bit of *AC*
11	*AC* = 0	Z	Zero value in *AC*

The first condition is always a 1, so that a reference to *CD* 00 (or the symbol U) will always find the condition to be true. When this condition is used in conjunction with the BR (branch) field, it provides an unconditional branch operation.

The indirect bit *I* is available from bit 15 of *DR* after an instruction is read from memory.

The sign bit of *AC* provides the next status bit.

The zero value, symbolized by *Z*, is a binary variable whose value is equal to 1 if all the bits in *AC* are equal to zero.

We will use the symbols U, I, S, and Z for the four status bits when we write microprograms in

branch field

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2-5) \leftarrow DR(11-14)$, $CAR(0,1,6) \leftarrow 0$

The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction.
when BR 00, the control performs a jump (JMP) operation (which is similar to a branch), and when BR 01, it performs a call to subroutine (CALL) operation.
The two operations are identical except that a call microinstruction stores the return address in the subroutine register *SBR*.
The jump and call operations depend on the value of the CD field.
It the status bit condition specified in the CD field is equal to 1, the next address in the AD field is transferred to the control address register *CAR*. Otherwise, *CAR* is incremented by 1.
The return from subroutine is accomplished with a BR field equal to 10. This causes the transfer of the return address from *SBR* to *CAR*.

branch field

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3: Microoperation fields

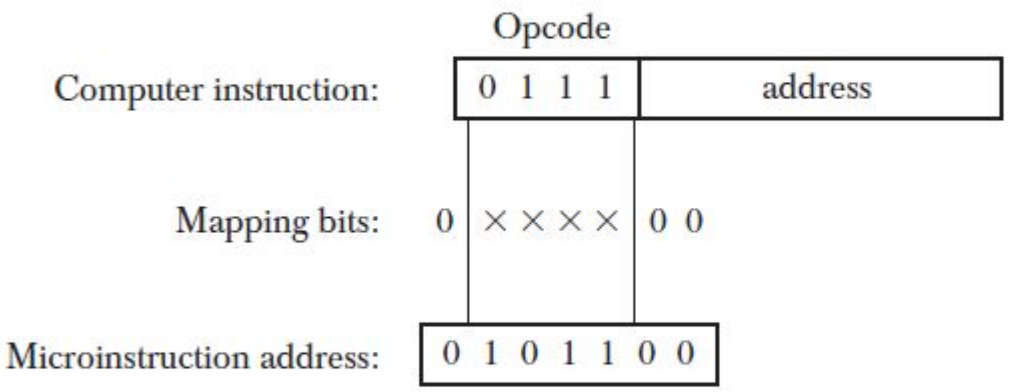
CD: Condition for branching

BR: Branch field

AD: Address field

15	14 11	10 0
I	Opcode	Address

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1
		$CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2-5) \leftarrow DR(11-14)$, $CAR(0,1,6) \leftarrow 0$



The mapping. from the operation code bits of the instruction to an address for *CAR* is accomplished when the BR field is equal to 11.

The bits of the operation code are in *DR*(11–14) after an instruction is read from memory.

Note that the List two conditions in the BR field are independent of the values in the CD and AD fields.

Symbolic Microinstructions

The symbols defined in Table can be used to specify microinstructions in symbolic form.

A symbolic microprogram can be translated into its binary equivalent by means of an assembler.

A microprogram assembler is similar in concept to a conventional computer assembler.

The simplest and most straightforward way to formulate an assembly language for a microprogram is to define symbols for each field of the microinstruction and to give users the capability for defining their own symbolic addresses.

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Symbolic Microinstructions

- Each line of the assembly language microprogram defines a symbolic microinstruction.
- Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD.

The fields specify the following information:

1. The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).

2. The microoperations field consists of one, two, or three symbols, separated by commas, from those defined in Table. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations. This will be translated by the assembler to nine zeros.

3. The CD field has one of the letters U, I, S, or Z.

4. The BR field contains one of the four symbols defined in Table.

5. The AD field specifies a value for the address field of the microinstruction in one of three possible ways:

   a. With a symbolic address, which must also appear as a label.

   b. With the symbol NEXT to designate the next address in sequence.

   c. When the BR field contains a RET or MAP symbol, the AD field is left empty and is

ORG

We will use also the pseudo instruction ORG to define the origin, or first address, of a microprogram routine.

Thus the symbol ORG 64 informs the assembler to place the next microinstruction in control memory at decimal address 64, which is equivalent to the binary address 1000000.

The Fetch Routine (fetch and decode)

- The control memory has 128 words, and each word contains 20 bits.
- To microprogram the control memory, it is necessary to determine the bit values of each of the 128 words.
- The first 64 words (addresses 0 to 63) are to be occupied by the routines for the 16 instructions.
- The last 64 words may be used for any other purpose.
- A convenient starting location for the fetch routine is address 64.

The microinstructions needed for the fetch routine are

$$AR \leftarrow PC$$
$$DR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$
$$AR \leftarrow DR(0-10), \quad CAR(2-5) \leftarrow DR(11-14), \; CAR(0,1,6) \leftarrow 0$$

The address of the instruction is transferred from *PC* to *AR* and the instruction is then read from memory into *DR*.

Since no instruction register is available, the instruction code remains in *DR*.

The address part is transferred to *AR* and then control is transferred to one of 16 routines by mapping the operation code part of the instruction from *DR* into *CAR*.

$$AR \leftarrow PC$$
$$DR \leftarrow M[AR], \quad PC \leftarrow PC + 1$$
$$AR \leftarrow DR(0-10), \quad CAR(2-5) \leftarrow DR(11-14), \; CAR(0,1,6) \leftarrow 0$$

The fetch routine needs three microinstructions, which are placed in control memory at addresses 64, 65, and 66.

Using the assembly language conventions defined previously, we can write the symbolic microprogram for the fetch routine as follows:

```
        ORG 64
FETCH:  PCTAR           U   JMP   NEXT
        READ, INCPC     U   JMP   NEXT
        DRTAR           U   MAP
```

$AR \leftarrow PC$

$DR \leftarrow \mathrm{M}[AR], \quad PC \leftarrow PC + 1$

$AR \leftarrow DR(0{-}10), \quad CAR(2{-}5) \leftarrow DR(11{-}14),\ CAR(0,1,6) \leftarrow 0$

```
         ORG 64
FETCH:   PCTAR           U    JMP    NEXT
         READ, INCPC     U    JMP    NEXT
         DRTAR           U    MAP
```

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD,\ SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2-5) \leftarrow DR(11-14),\ CAR(0,1,6) \leftarrow 0$

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

The translation of the symbolic microprogram to binary produces the following binary microprogram

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

NOTE

- The three microinstructions that constitute the fetch routine have been listed in three different representations.
- The register transfer representation shows the internal register transfer operations that each microinstruction implements.
- The symbolic representation is useful for writing microprograms in an assembly language format.
- The binary representation is the actual internal content that must be stored in control memory.
- It is customary to write microprograms in symbolic form and then use an assembler program to obtain a translation to binary.

```
         ORG 64
FETCH:   PCTAR              U    JMP    NEXT
         READ, INCPC        U    JMP    NEXT
         DRTAR              U    MAP
```

The execution of the third (MAP) microinstruction in the fetch routine results in a branch to address 0xxxx00, where xxxx are the four bits of the operation code.

For example, if the instruction is an ADD instruction whose operation code is 0000, the MAP microinstruction will transfer to *CAR* the address 0000000, which is the start address for the ADD routine in control memory.

The first address for the BRANCH and STORE routines are 0 0001 00 (decimal 4) and 0 0010 00 (decimal 8), respectively.

The first address for the other 13 routines are at address values 12, 16, 20, . . . , 60.

This gives four words in control memory for each routine.

Evaluating Effective Address

- In each routine we must provide microinstructions for evaluating the
- effective address and for executing the instruction.
- The indirect address mode is associated with all memory-reference instructions.
- A saving in the number of control memory words may be achieved if the microinstructions for the indirect address are stored as a subroutine.
- This subroutine, symbolized by INDRCT, is located right after the fetch routine

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow AC$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

$AR \leftarrow PC$

$DR \leftarrow \mathrm{M}[AR], \quad PC \leftarrow PC + 1$

$AR \leftarrow DR(0-10), \quad CAR(2-5) \leftarrow DR(11-14), \; CAR(0,1,6) \leftarrow 0$

To see how the transfer and return from the indirect subroutine occurs, assume that the MAP microinstruction at the end of the fetch routine caused a branch to address 0, where the ADD routine is stored.

The first microinstruction in the ADD routine calls subroutine INDRCT, conditioned on status bit *I*.

If I 1, a branch to INDRCT occurs and the return address (address 1 in this case) is stored in the subroutine register *SBR*.

The INDRCT subroutine has two microinstructions:

```
INDRCT:   READ   U   JMP   NEXT
          DRTAR  U   RET
```

Symbolic Microprogram (Partial)

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Symbol	Opcode	Description
ADD	0000	$AC \rightarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

EA is the effective address

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD, SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2-5) \leftarrow DR(11-14),\ CAR(0,1,6) \leftarrow 0$

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

Binary Microprogram for Control Memory (Partial)

Micro Routine	Address Decimal	Address Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

The binary microprogram listed in specifies the word content of the control memory.
When a ROM is used for the control memory, the microprogram binary list provides the truth table for fabricating the unit.
This fabrication is a hardware process and consists of creating a mask for the ROM so as to produce the l's and 0's for each word.
The bits of ROM are fixed once the internal links are fused during the hardware production.
The ROM is made of IC packages that can be removed if necessary and replaced by other packages.
To modify the instruction set of the computer, it is necessary to generate a new microprogram and mask a new ROM.

Note

- If a writable control memory is employed, the ROM is replaced by a RAM.
- The advantage of employing a RAM for the control memory is that the microprogram can be altered simply by writing a new pattern of l's and 0's without resorting to hardware procedures.
- A writable control memory possesses the flexibility of choosing the instruction set of a computer dynamically by changing the microprogram under processor control.
- However, most microprogrammed systems use a ROM for the control memory because it is cheaper and faster than a RAM and also to prevent the occasional user from changing the architecture of the system.

Design of Control Unit

Issue

- The bits of the microinstruction are usually divided into fields, with each field defining a distinct, separate function.
- The various fields encountered in instruction formats provide control bits to initiate microoperations in the system, special bits to specify the way that the next address is to be evaluated, and an address field for branching.
- **The number of control bits that initiate microoperations can be reduced by grouping mutually exclusive variables into fields** and encoding the K bits in each field to provide 2^K microoperations.

Issue

- Each field requires a decoder to produce the corresponding control signals.
- This method **reduces the size of the microinstruction bits** but requires additional hardware external to the control memory.
- It also **increases the delay time of the control signals** because they must propagate through the decoding circuits.
- The encoding of control bits was demonstrated in the programming example of the preceding section.
- The nine bits of the microoperation field are divided into three subfields of three bits each.
- The control memory output of each subfield must be decoded to provide the distinct microoperations.
- The outputs of the decoders are connected to the appropriate inputs in the processor unit.

The encoding of control bits

3	3	3	2	2	7
F1	F2	F3	CD	BR	AD

F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

AD: Address field

The nine bits of the microoperation field are divided into three subfields of three bits each.

The control memory output of each subfield must be decoded to provide the distinct microoperations.

The outputs of the decoders are connected to the appropriate inputs in the
processor unit.

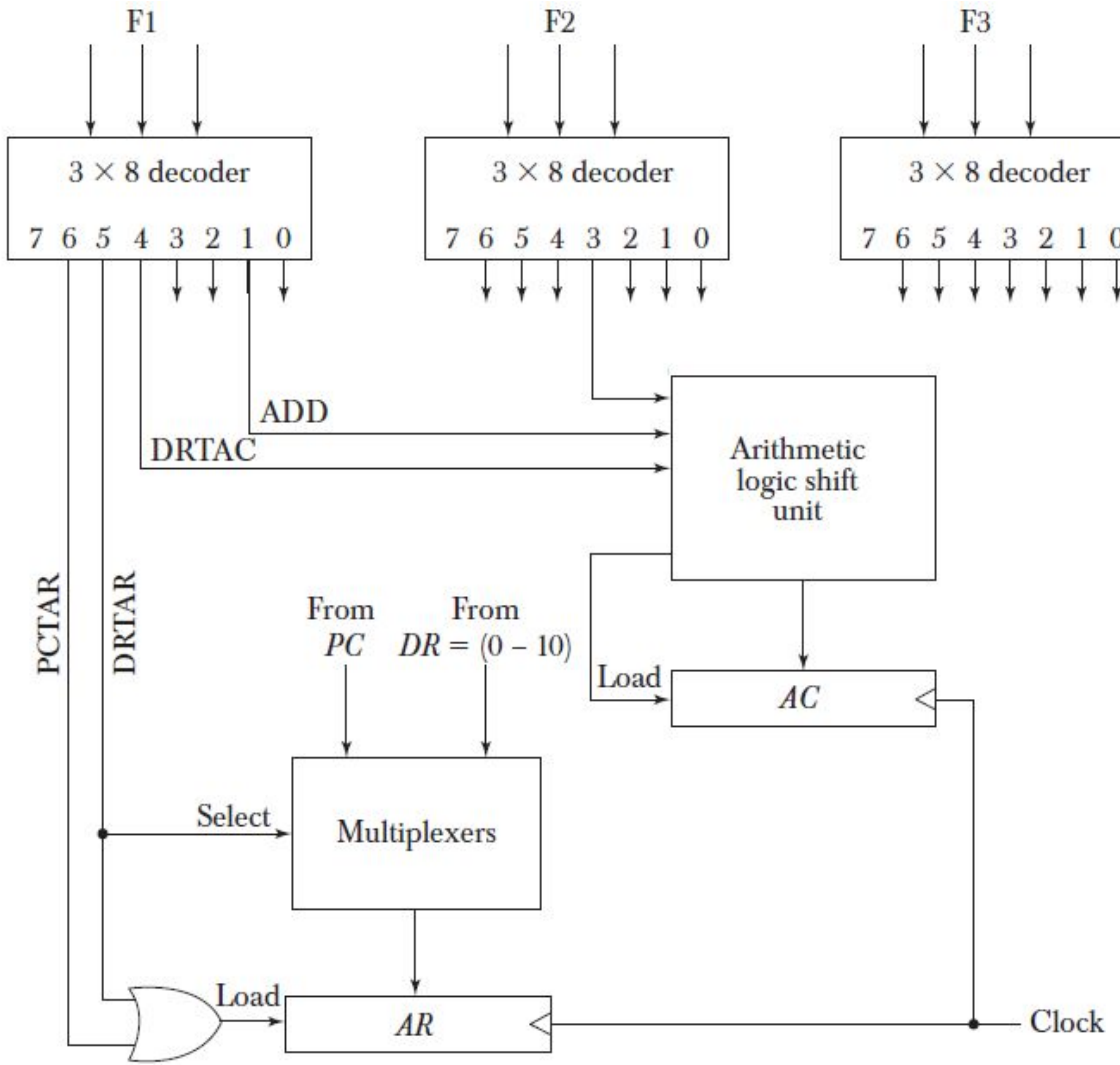
F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE
F2	**Microoperation**	**Symbol**
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR
F3	**Microoperation**	**Symbol**
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

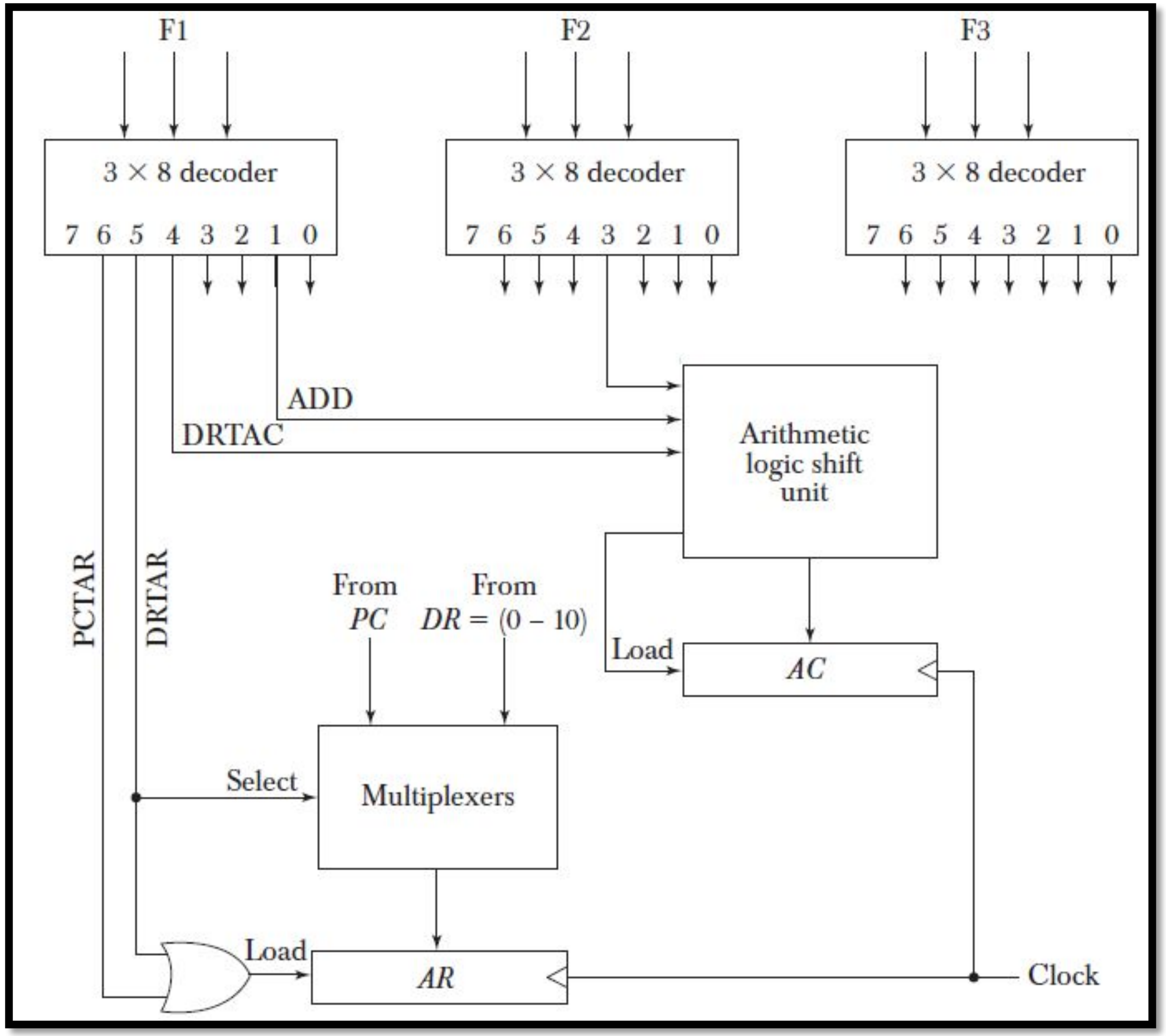
Decoding of microoperation fields

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	





Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3 x 8 decoder to provide eight outputs.

Each of these outputs must be connected to the proper circuit to initiate the corresponding microoperation.

For example, when *F*1 =101 (binary 5), the next clock pulse transition transfers the content of *DR*(0–10) to *AR* (symbolized by DRTAR).

Similarly, when *F*1 =110 (binary 6) there is a transfer from *PC* to *AR* (symbolized by PCTAR).

As shown in Fig., outputs 5 and 6 of decoder *F*1 are connected to the load input of *AR* so that when either one of these outputs is active, information from the multiplexers is transferred to *AR*.

The multiplexers select the information from *DR* when output 5 is active and from *PC* when output 5 is inactive.

The transfer into *AR* occurs with a clock pulse transition only when output 5 or output 6 of the decoder are active.

The other outputs of the decoders that initiate transfers between registers must be connected in a similar fashion.

Microprogram Sequencer

- The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a microprogram sequencer.
- A microprogram sequencer can be constructed with digital functions to suit a particular application.
- However, just as there are large ROM units available in integrated circuit packages, so are general-purpose sequencers suited for the construction of microprogram control units.
- To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range of applications.

Microprogram Sequencer

- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next-address logic of the sequencer determines the specific address source to be loaded into the control address register.
- The choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction.
- Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutine calls.
- Some sequencers provide an output register which can function as the address register for the control memory.

Microprogram sequencer for a control memory

The control memory is included in the diagram to show the interaction between the sequencer and the memory attached to it.

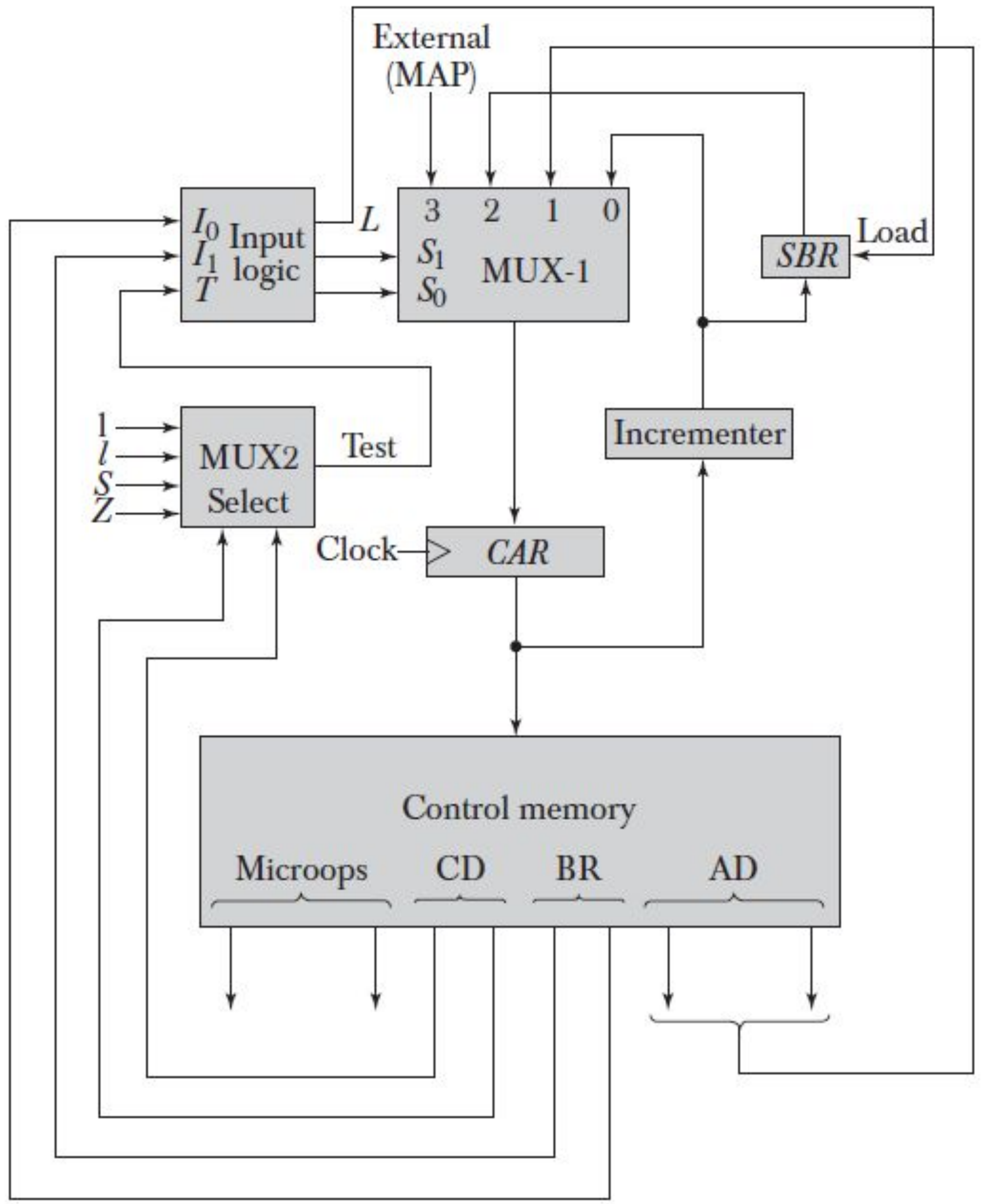
There are two multiplexers in the circuit.

The first multiplexer selects an address from one of four sources and routes it into a control address register *CAR*.

The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.

The output from *CAR* provides the address for the control memory.

The content of *CAR* is incremented and applied to one of the multiplexer inputs and to the subroutine register *SBR*.



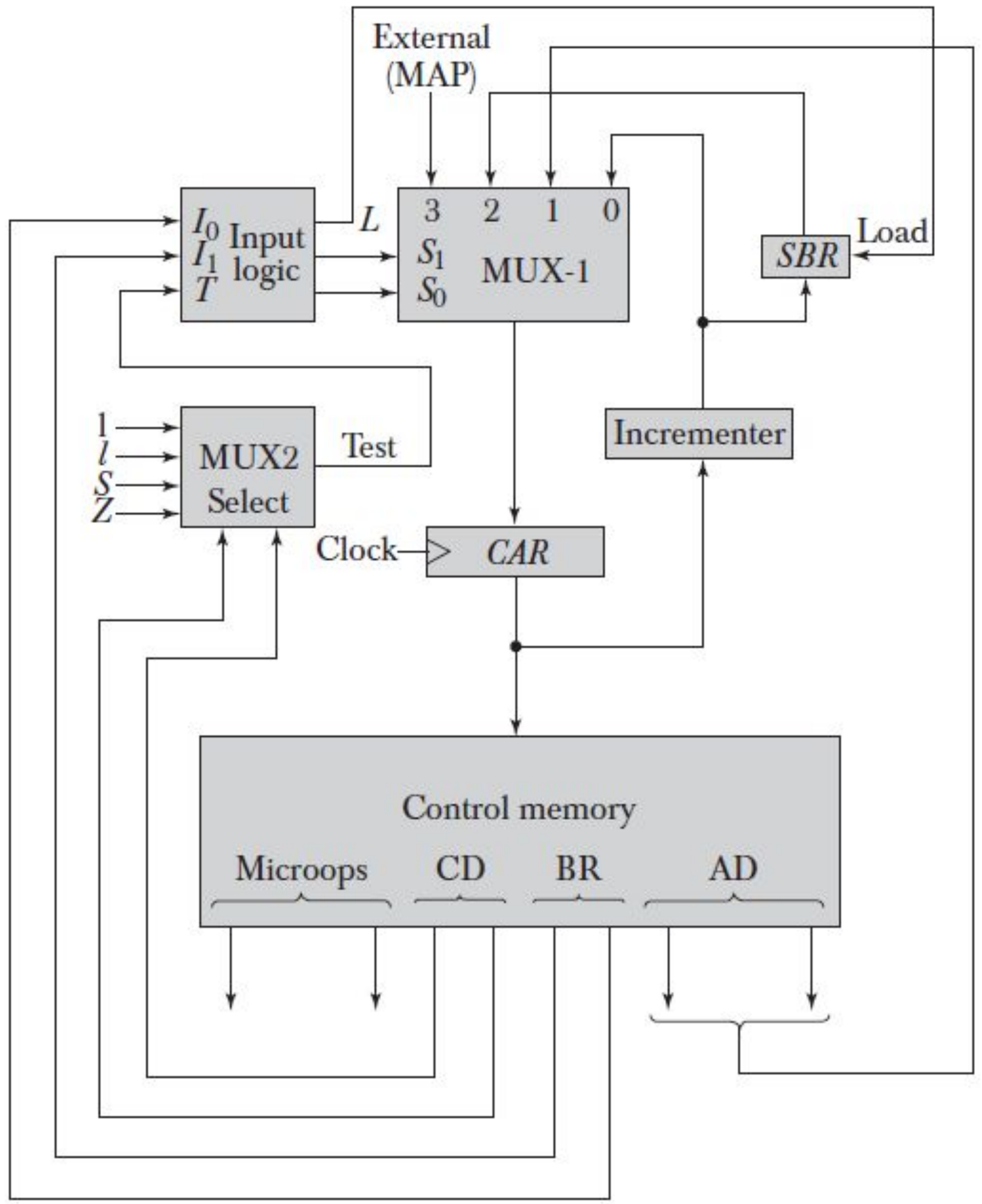
Microprogram sequencer for a control memory

The other three inputs to multiplexer number 1 come from the address field of the present microinstruction, from the output of *SBR*, and from an external source that maps the instruction.

Although the diagram shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep.

In this way, a number of subroutines can be active at the same time.

A push and pop operation, in conjunction with a stack pointer, stores and retrieves the return address during the call and return microinstructions.



Microprogram sequencer for a control memory

The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer.

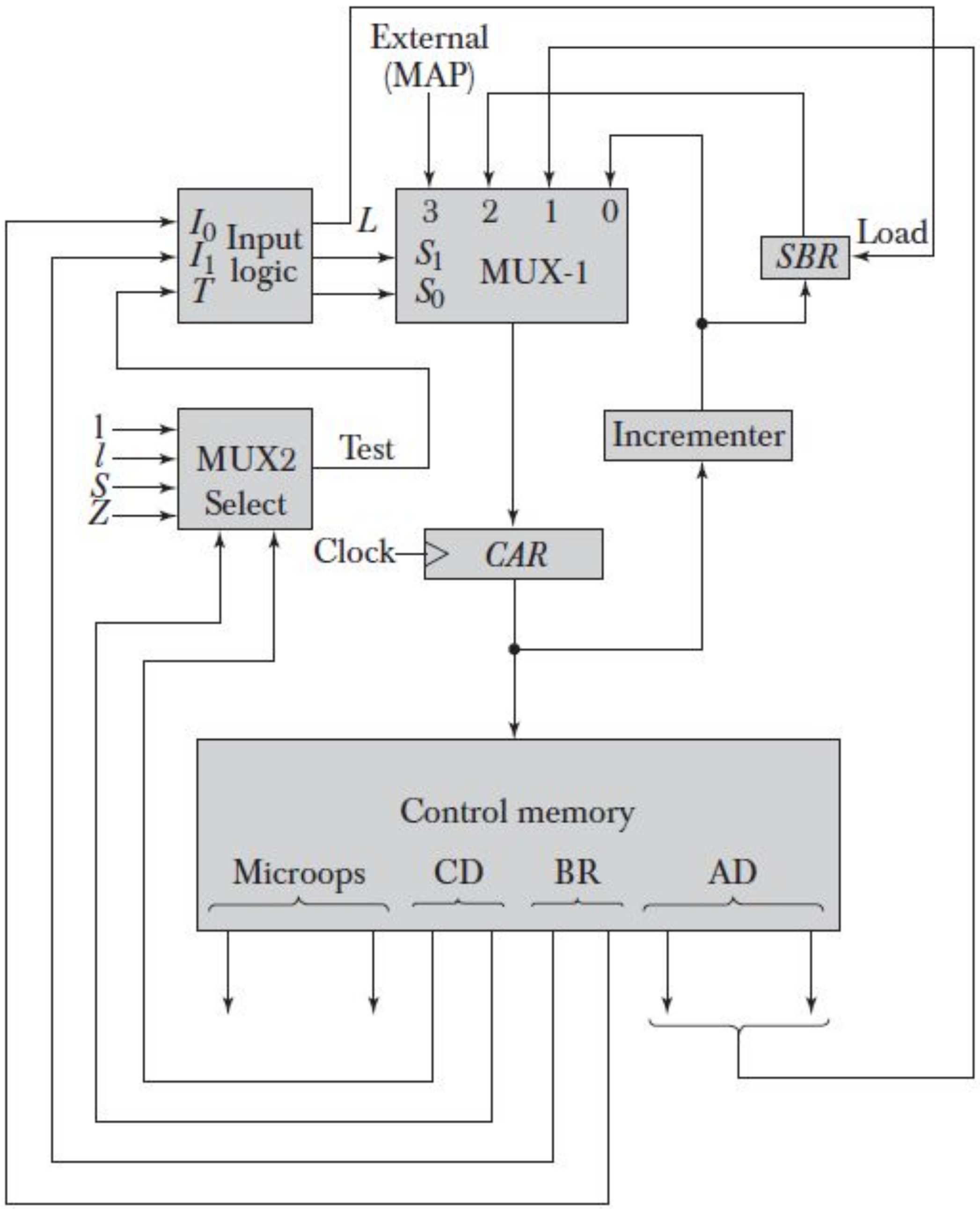
If the bit selected is equal to 1, the *T* (test) variable is equal to 1; otherwise, it is equal to 0.

The *T* value together with the two bits from the BR (branch) field go to an input logic circuit.

The input logic in a particular sequencer will determine the type of operations that are available in the unit.

Typical sequencer operations are: increment, branch or jump, call
and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations.

With three inputs, the sequencer can provide up to eight address sequencing operation.



Design of Input Logic

The input logic circuit in has three inputs, I_0, I_1, and T, and three outputs, S_0, S_1, and L.

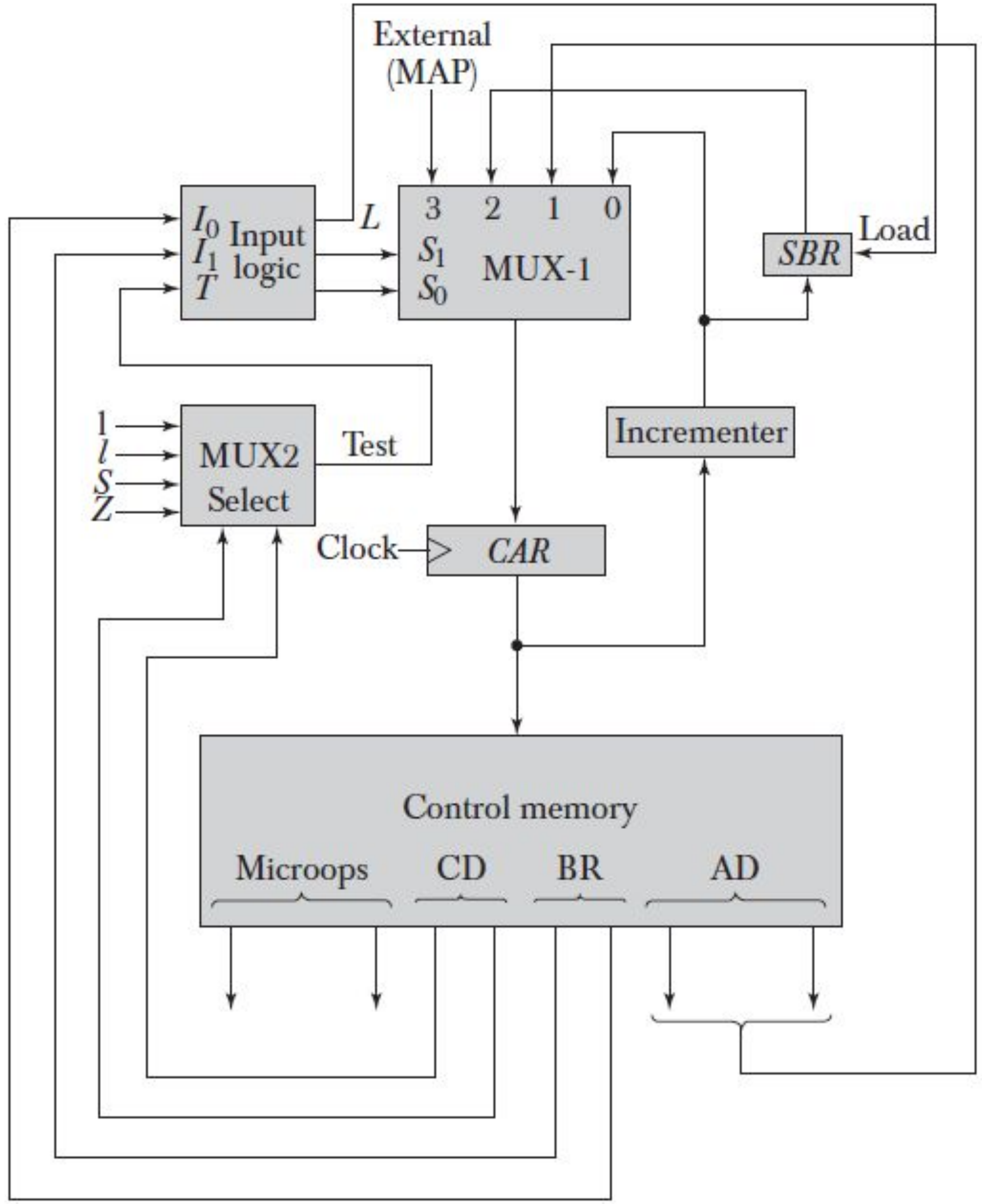
Variables S_0 and S_1 select one of the source addresses for *CAR*.

Variable L enables the load input in *SBR*.

The binary values of the two selection variables determine the path in the multiplexer.

For example, with S_1, S_0 =10, multiplexer input number 2 is selected and establishes a transfer path from *SBR* to *CAR*.

Note that each of the four inputs as well as the output of MUX 1 contains a 7-bit address.

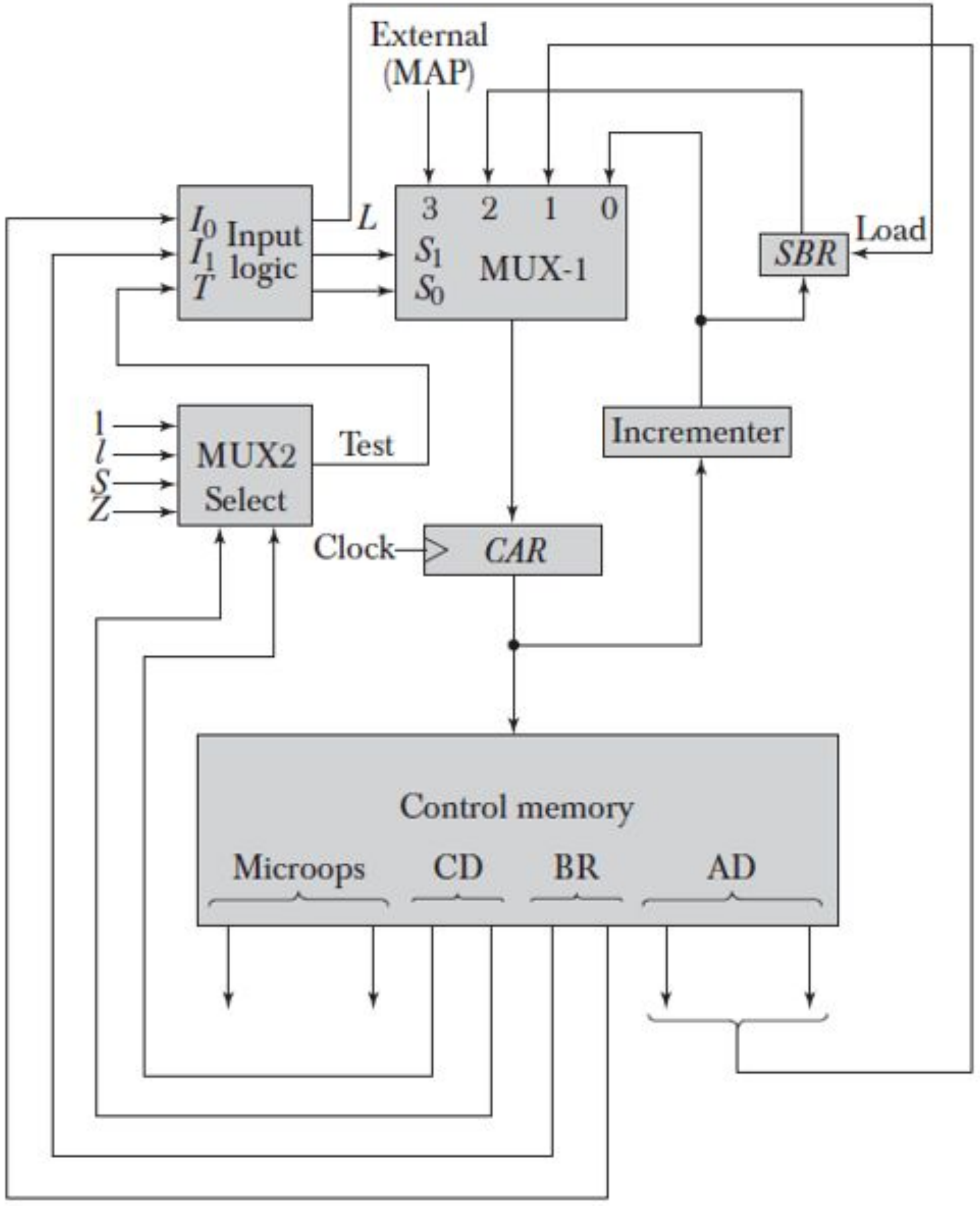


Input Logic Truth Table for Microprogram Sequencer

BR Field		Input I_1	I_0	T	MUX 1 S_1	S_0	Load *SBR* L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	×	1	0	0
1	1	1	1	×	1	1	0

The truth table can be used to obtain the simplified Boolean functions for the input logic circuit:

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I_1' T$$
$$L = I_1' I_0 T$$



The bit values for S_1 and S_0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.

The subroutine register is loaded with the incremented value of *CAR* during a call microinstruction (BR =01) provided that the status bit condition is satisfied (T =1).