Unit 4
Control Unit

In digital computer, function of control unit is to initiate sequences of microoperations. Types of microoperations for particular system are finite. The complexity of digital system is dependent on the number of sequences of microoperations that are performed. Two complementary techniques used for implementing control unit: hardwired and micro programmed.

Hardwired control

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be *hardwired*. We have already studied about the hardwired control unit of basic computer and timing signals associated with it, so guys, turn back to unit3 (textbook, chapter-5) for this portion.

Microprogrammed control

Basic terminologies:

Control Memory (Control Storage: CS)

✓ Storage in the microprogrammed control unit to store the microprogram.

Control word
✓ It is a string of control variables (0's and 1's) occupying a word in control memory.

Microprogram

✓ Program stored in control memory that generates all the control signals required to execute the instruction set correctly
✓ Consists of microinstructions

Microinstruction

✓ Contains a control word and a sequencing word
✓ Control Word – contains all the control information required for one clock cycle
✓ Sequencing Word - Contains information needed to decide the next microinstruction address

Writable Control Memory (Writable Control Storage: WCS)

✓ CS whose contents can be modified:
  ▪ Microprogram can be changed
  ▪ Instruction set can be changed or modified

A computer that employs a microprogrammed control unit will have two separate memories: main memory and a control memory. The user's program in main memory consists of machine instructions and data whereas control memory holds a fixed micro program that cannot be altered by the user. Each machine instruction initiates a series of microinstructions in control memory.

The general configuration of a microprogrammed control unit is demonstrated in the following block diagram:
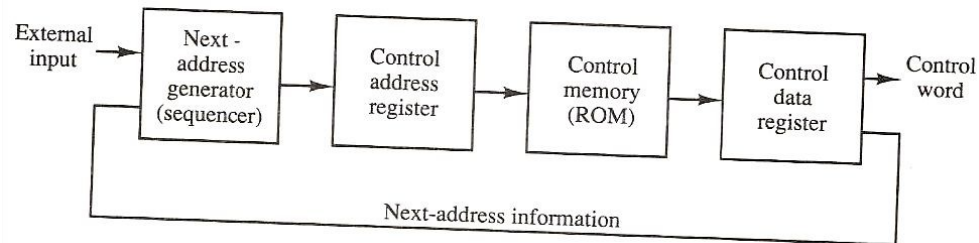

Fig: Microprogrammed control organization

Dynamic Microprogramming

✓ Computer system whose control unit is implemented with a microprogram in WCS.
✓ Microprogram can be changed by a systems programmer or a user

Control Address Register: Control address register contains address of microinstruction.

Control Data Register: Control data register contains microinstruction.

Sequencer

✓ The device or program that generates address of next microinstruction to be executed is called sequencer.

Address Sequencing

Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction. Process of finding address of next micro-instruction to be executed is called address sequencing. Address sequencer must have capabilities of finding address of next micro-instruction in following situations:

- In-line Sequencing
- Unconditional Branch
- Conditional Branch
- Subroutine call and return
- Looping
- Mapping from instruction op-code to address in control memory.

Following is the block diagram for control memory and the associated hardware needed for selecting the next microinstruction address.
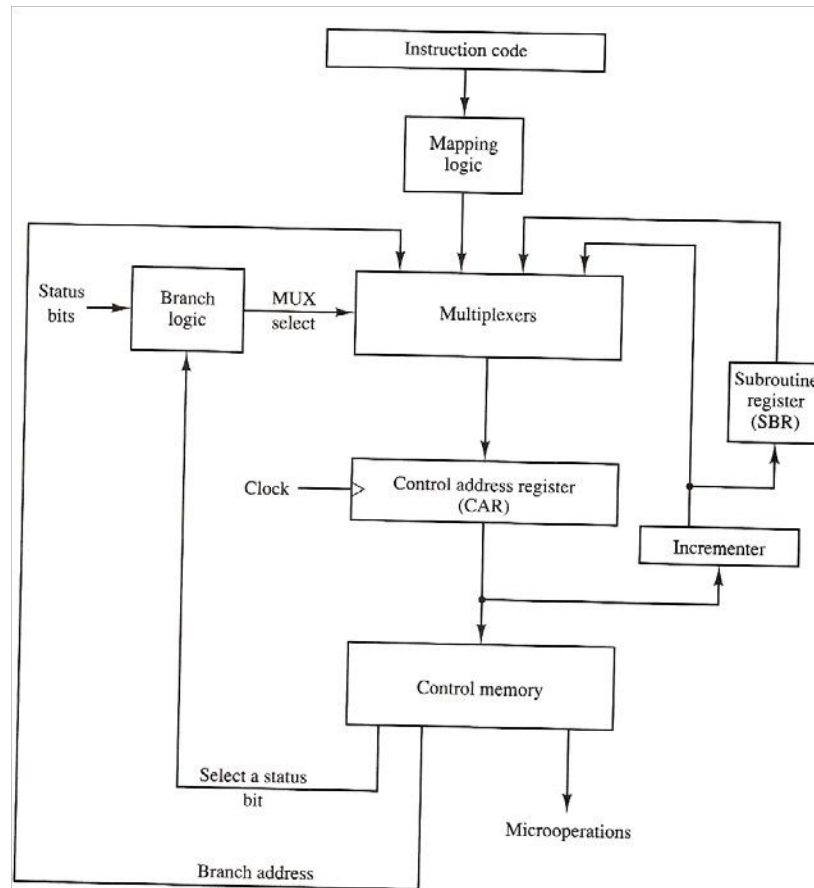
Fig: Block diagram of address sequencer.

- ✓ Control address register receives address of next micro instruction from different sources.
- ✓ Incrementer simply increments the address by one
- ✓ In case of branching branch address is specified in one of the field of microinstruction.
- ✓ In case of subroutine call return address is stored in the register SBR which is used when returning from called subroutine.

Conditional Branch

Simplest way of implementing branch logic hardware is to test the specified condition and branch to the indicated address if condition is met otherwise address resister is simply incremented. If Condition is true, h/w set the appropriate field of status register to 1. Conditions are tested for O (overflow), N (negative), Z (zero), C (carry), etc.

Unconditional Branch

Fix the value of one status bit at the input of the multiplexer to 1. So that, branching can always be done.

Mapping

Assuming operation code of 4-bits which can specify $16^4 \vert 2$ distinct instructions. Assume further and control memory has 128 words, requiring an address of 7-bits. Now we have to map 4-bit operation code into 7-bit control memory address. Thus, we have to map Op-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its subroutine in memory.

Direct mapping:

Directly use opcode as address of Control memory
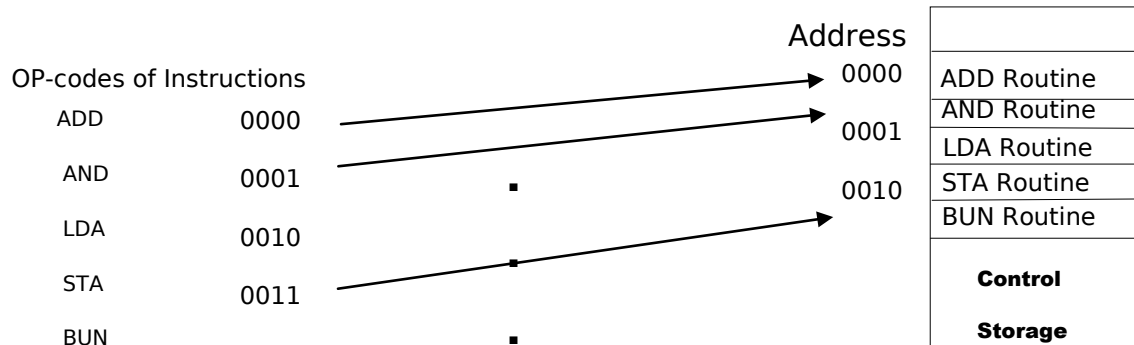
Another approach of direct mapping:

Transfer Opcode bits to use it as an address of control memory.

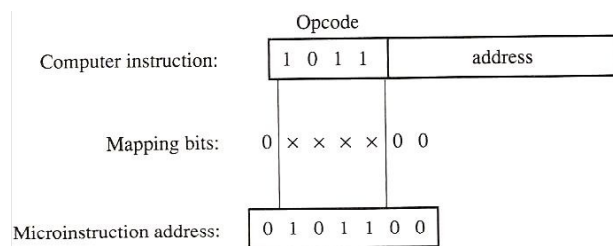Fig: mapping from instruction code to microinstruction address

Extended idea: Mapping function implemented by ROM or PLD(Programmable Logic Device)

Use opcode as address of ROM where address of control memory is stored and than use that address as an address of control memory. This provides flexibility to add instructions for control memory as the need arises.
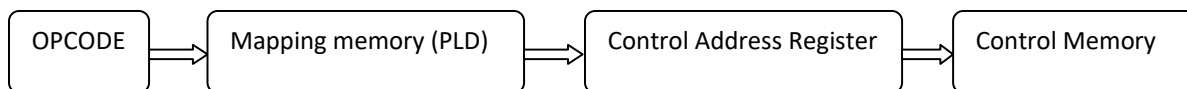
Subroutines

Subroutines are programs that are used by another program to accomplish a particular task. Microinstructions can be saved by employing subroutines that use common sections of micro code.

Example: the sequence of microoperations needed to generate the effective address is common to all memory reference instructions. Thus, this sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Subroutine resister is used to save a return address during a subroutine call which is organized in LIFO (last in, first out) stack.

Microprogram (An example)

Once we have a configuration of a computer and its microprogrammed control unit, the designer generates the microcode for the control memory. Code generation of this type is called microprogramming and is similar to conventional machine language programming. We assume here a simple digital computer similar (but not identical) to Manos' basic computer.

Computer configuration

Block diagram is shown below; it consists of two memory units: a main memory for storing instructions and data, and a control memory for storing the microprogram. 4 resisters are with processor unit and 2 resisters with the control unit.
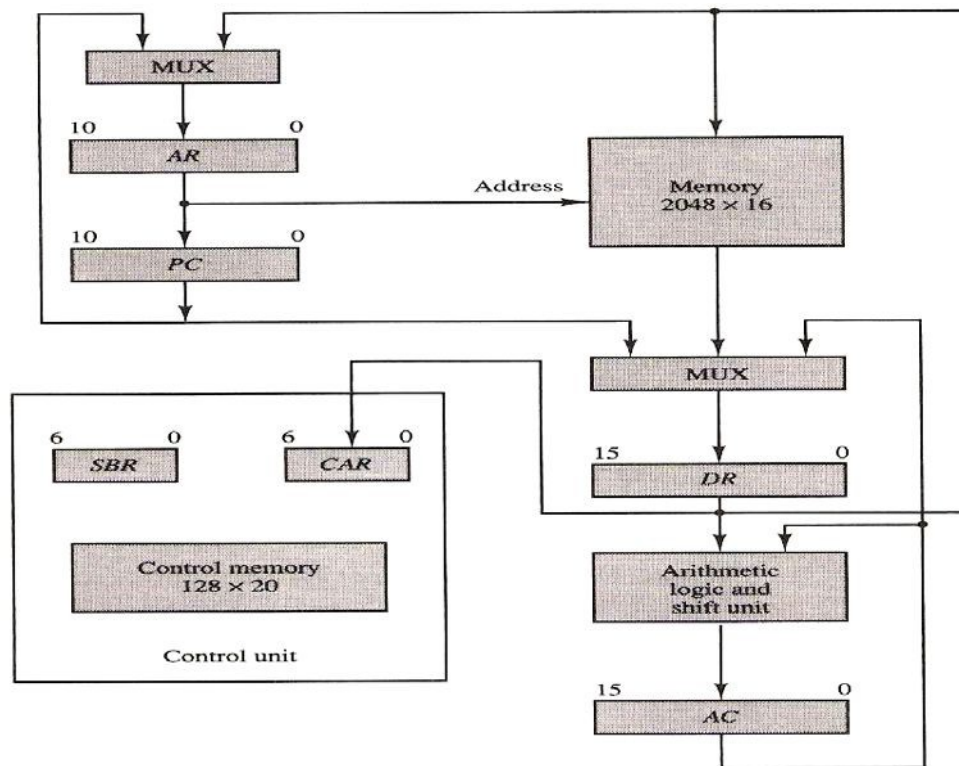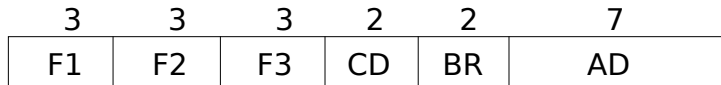

Fig: Computer hardware configuration

Microinstruction Format

We know the computer instruction format (explained in unit3) for different set of instruction in main memory. Similarly, microinstruction in control memory has 20-bit format divided into 4 functional parts as shown below.

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

Each microoperation below is defined using resister transfer statements and is assigned a symbol for use in symbolic microprogram.

Description of CD

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

Description of BR

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	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

CD (condition) field consists of two bits representing 4 status bits and BR (branch) field (2-bits) used together with address field AD, to choose the address of the next microinstruction.

Microinstruction fields (F1, F2, F3)

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	$AR \leftarrow DR(0\text{–}10)$	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	$AC \leftarrow AC \lor DR$	OR
011	$AC \leftarrow AC \land DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	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	

Here, microoperations are subdivided into three fields of 3-bits each. These 3 bits are used to encode 7 different microoperations. No more than 3 microoperations can be chosen for a microinstruction, one for each field. If fewer than 3 microoperations are used, one or more fields will contain 000 for no operation.

Symbolic Microinstructions

Symbols are used in microinstructions as in assembly language. A symbolic microprogram can be translated into its binary equivalent by a microprogram assembler.

Format of Microinstruction:

Contains five fields: label; micro-ops; CD; BR; AD

Label: may be empty or may specify a symbolic address terminated with a colon

Micro-ops: consists of one, two, or three symbols separated by commas

CD: one of {U, I, S, Z},

Where   U: Unconditional Branch

I: Indirect address bit

S: Sign of AC

Z: Zero value in AC

BR: one of {JMP, CALL, RET, MAP}

AD: one of {Symbolic address, NEXT, empty (in case of MAP and RET)}

Symbolic Microprogram (example)

FETCH Routine: During FETCH Read an instruction from memory and decode the instruction and update PC

Sequence of microoperations in the *fetch cycle*:

```
AR ←   PC

DR ←   M[AR], PC ← PC + 1

AR ←   DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0
```

Symbolic microprogram for the fetch cycle:

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

- Control Storage: 128 20-bit words

- The first 64 words: Routines for the 16 machine instructions

- The last 64 words: Used for other purpose (e.g., fetch routine and other subroutines)

- Mapping: OP-code XXXX into 0XXXX00, the first address for the 16 routines are

  0(0 0000 00), 4(0 0001 00), 8, 12, 16, 20, …, 60

Partial Symbolic Microprogram

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

E.g. the execution of ADD instruction is carried out by the microinstructions at addresses 1 and 2. The first microinstruction reads operand from into DR. The second microinstruction performs an add microoperation with the content of DR AC and then jumps back to the beginning of the fetch routine.

Binary Microprogram

Symbolic microprogram is a convenient form for writing microprograms in a way that people can understand. But this is not a way that the microprogram is stored in memory. It must be translated into binary by means of assembler.

Binary equivalent of a microprogram translated by an assembler for fetch cycle:

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

Binary program for control memory

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

Design of Control Unit

F-field decoding

The 9-bits of the microoperation field are divided into 3 subfields of 3 bits each. The control memory output of each subfield must be decoded to provide distinct microoperations. The outputs of the decoders are connected to the appropriate inputs in the processor unit.

Fig below shows 3 decoders and connections that must be made from their outputs.
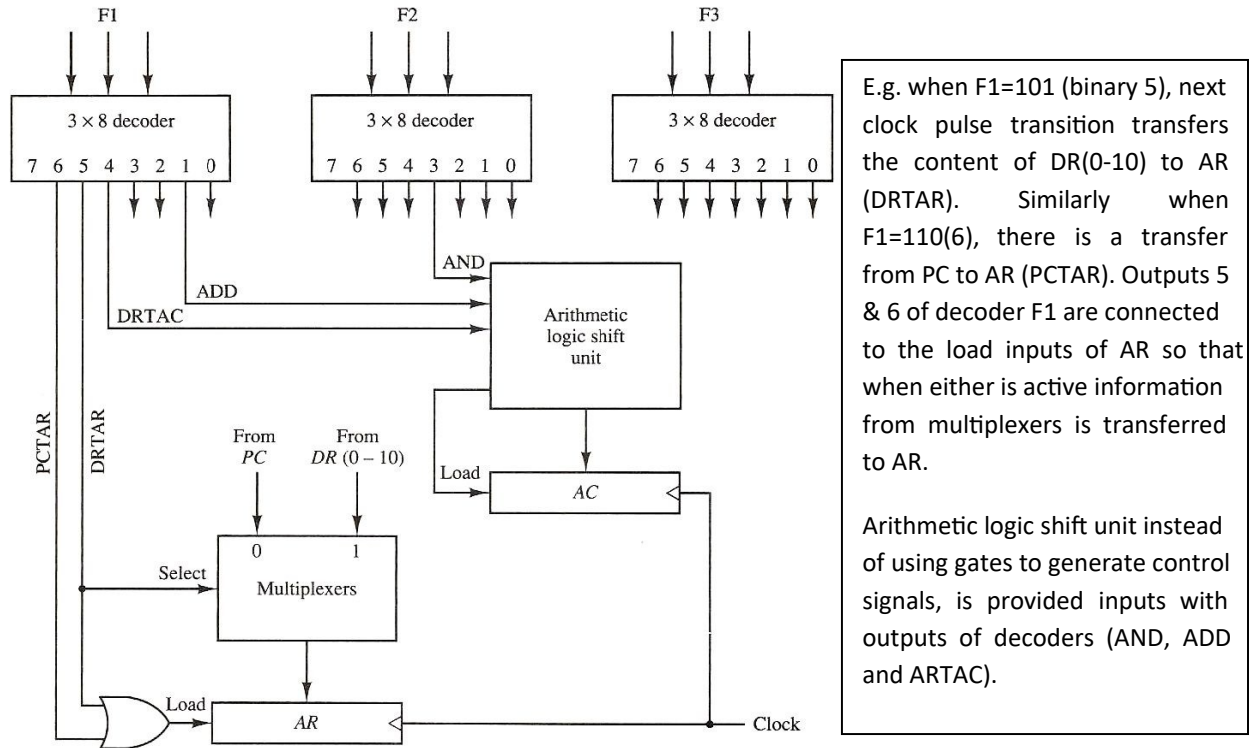
E.g. when F1=101 (binary 5), next clock pulse transition transfers the content of DR(0-10) to AR (DRTAR). Similarly when F1=110(6), there is a transfer from PC to AR (PCTAR). Outputs 5 & 6 of decoder F1 are connected to the load inputs of AR so that when either is active information from multiplexers is transferred to AR.

Arithmetic logic shift unit instead of using gates to generate control signals, is provided inputs with outputs of decoders (AND, ADD and ARTAC).

Fig: Decoding of microoperation fields

Microprogram Sequencer

Basic components of a microprogrammed control unit are control memory and the circuits that select the next address. This address selection part is called a microprogram sequencer. The purpose of microprogram sequencer is to load CAR so that microinstruction may be read and executed. Commercial sequencers include within the unit an internal resister stack to store addresses during microprogram looping and subroutine calls.

Internal structure of a typical microprogram sequencer is shown below in the diagram. It consists of input logic circuit having following truth table.

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	$\times$	1	0	0
1	1	1	1	$\times$	1	1	0

Fig: Input Logic Truth for Microprogram Sequencer

Fig: Microprogram sequencer for a control memory
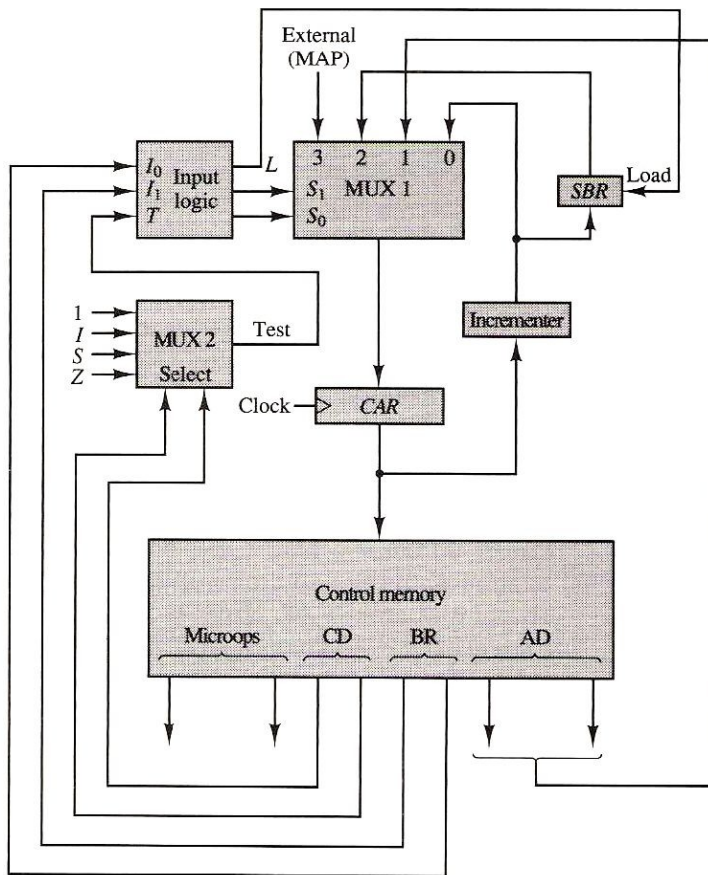
-MUX1 selects an address from one of four sources of and routes it into CAR.

-MUX2 tests the value of selected status bit and result is applied to input logic circuit.

-Output of CAR provides address for the control memory

-Input logic circuit has 3 inputs I_0, I_1 and T and 3 outputs S_0, S_1 and L. variables S_0 and S_1 select one of the source addresses for CAR. L enables load input of SBR.

-e.g. when $S_1S_0=10$, MUX input number 2 is selected and establishes a transfer path from SBR to CAR.