Microprogrammed Control

Unit 4

Design of Accumulator logic

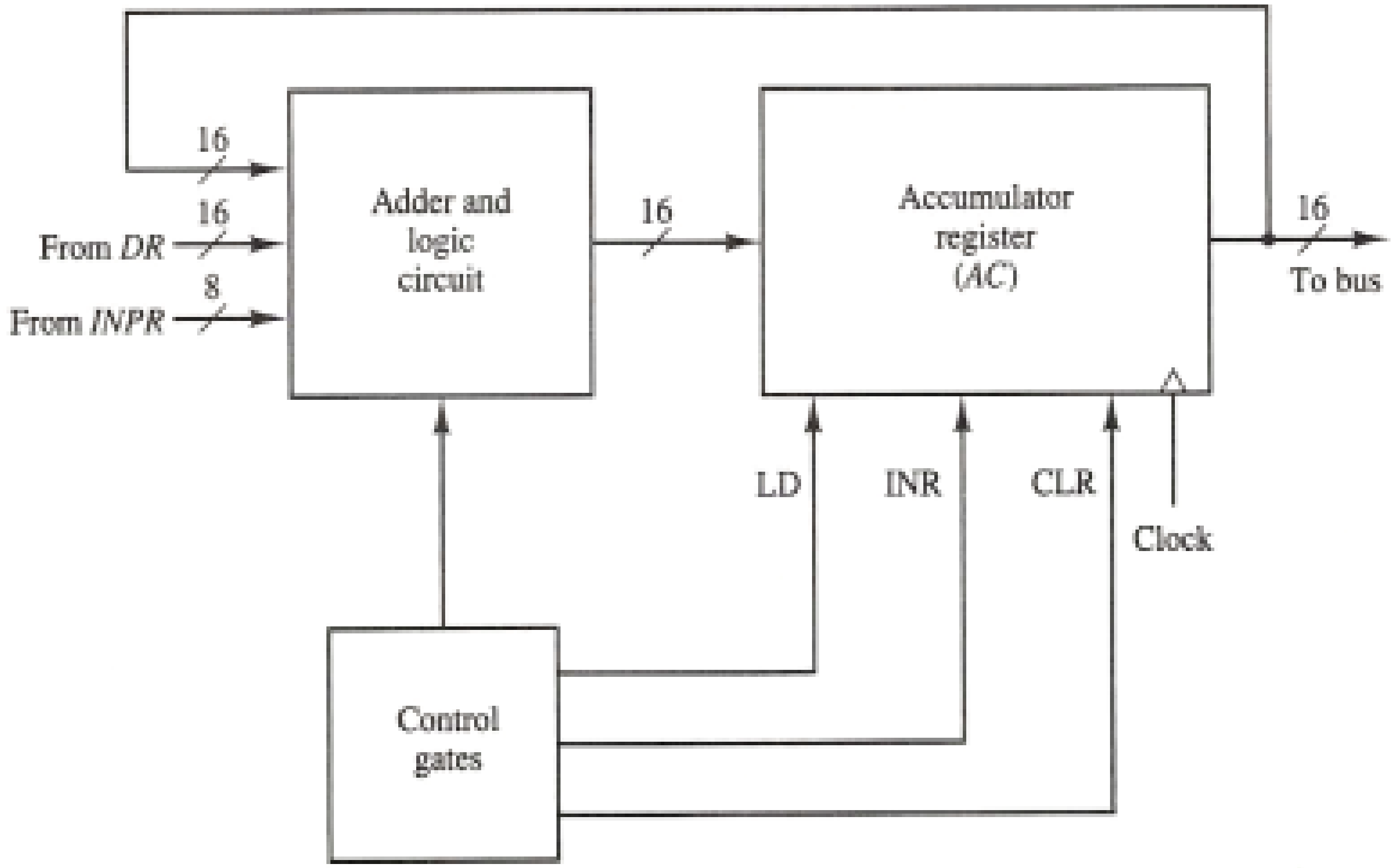
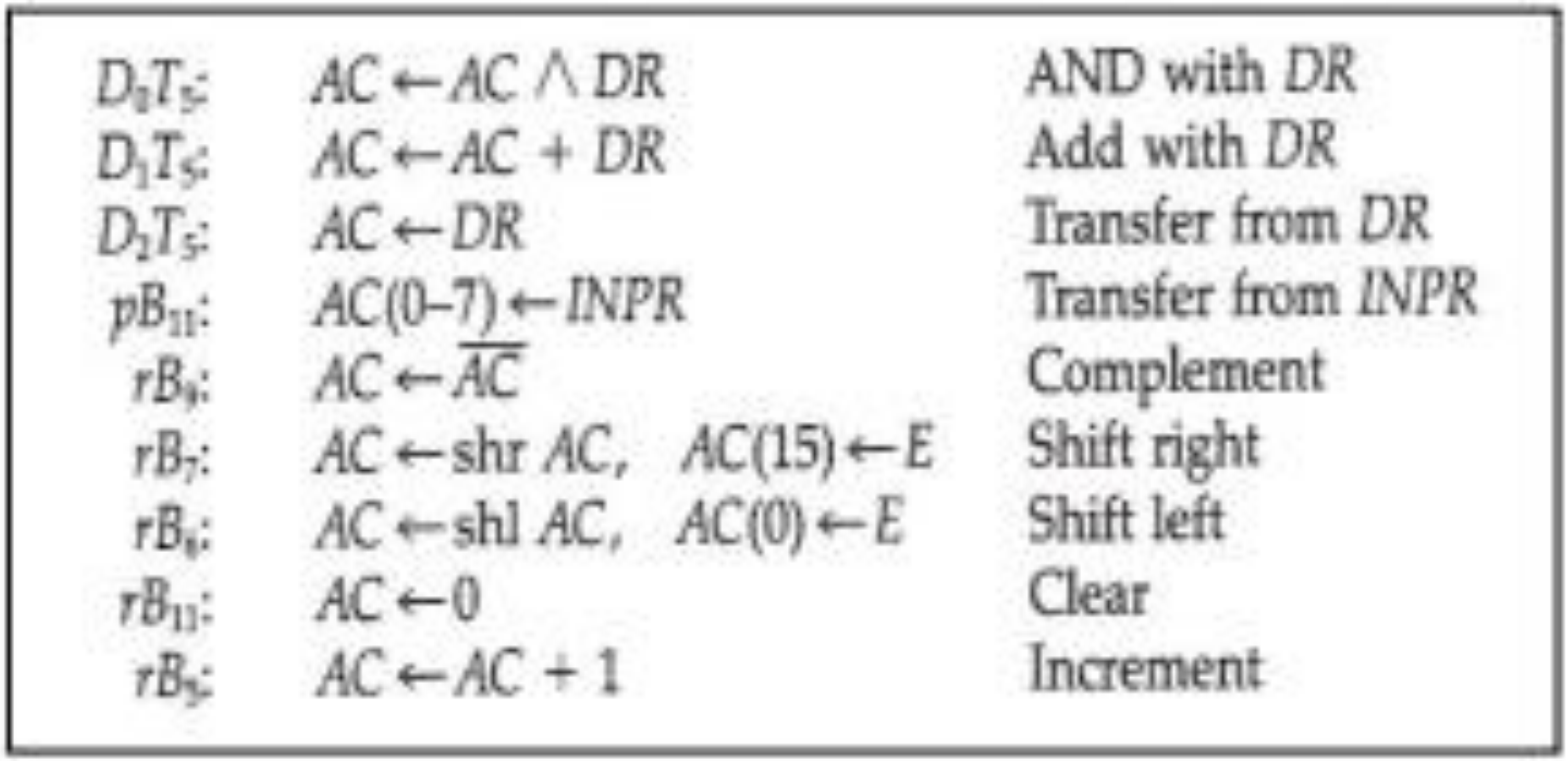


Fig: circuits associated with AC

Change AC



D_0T_5:	$AC \leftarrow AC \wedge DR$	AND with *DR*
D_1T_5:	$AC \leftarrow AC + DR$	Add with *DR*
D_2T_5:	$AC \leftarrow DR$	Transfer from *DR*
pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$	Transfer from *INPR*
rB_9:	$AC \leftarrow \overline{AC}$	Complement
rB_7:	$AC \leftarrow \text{shr } AC, \quad AC(15) \leftarrow E$	Shift right
rB_6:	$AC \leftarrow \text{shl } AC, \quad AC(0) \leftarrow E$	Shift left
rB_{11}:	$AC \leftarrow 0$	Clear
rB_5:	$AC \leftarrow AC + 1$	Increment

Gate Structure For Controlling AC

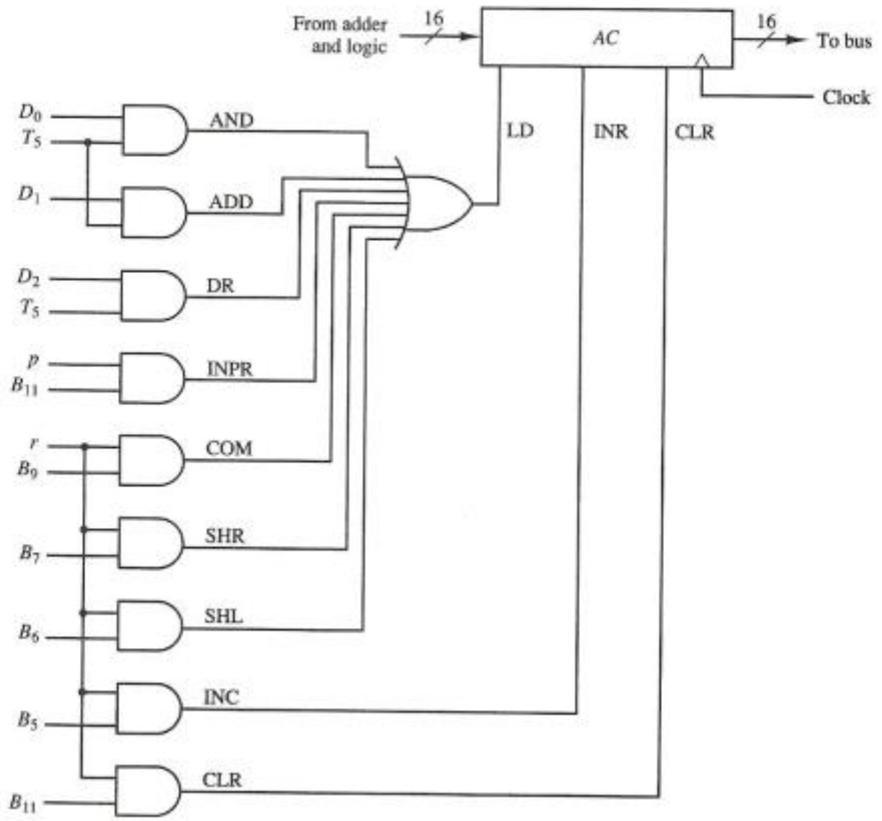


Fig: Gate structure for controlling LD, INR and CLR of AC

Control unit

- Initiate sequence of microoperations
- Number of microoperations are finite

Control Unit

- Control Unit is implemented In 2 ways:-
- *Hardwired* Control

  - fixed instructions, fixed logic block, encoder, decoder.

  -CU is made up of sequential and combinational circuits to generate the control signals.
  - If logic is changed we need to change the whole circuitry
  - Expensive
  - high speed operations.

  -complex

  -no flexibility of adding instructions

  -Intel 8085,motorola 6802,RISC.

- *Micro programmed* **Control**
  - A control memory on the processor contains micro-programs that activate the necessary control signals.
  - If logic is changed we only need to change the micro-program.
  - Cheap
  - Slow

–Intel 8080,Motorola 68000, CISC.

Hardwired Control

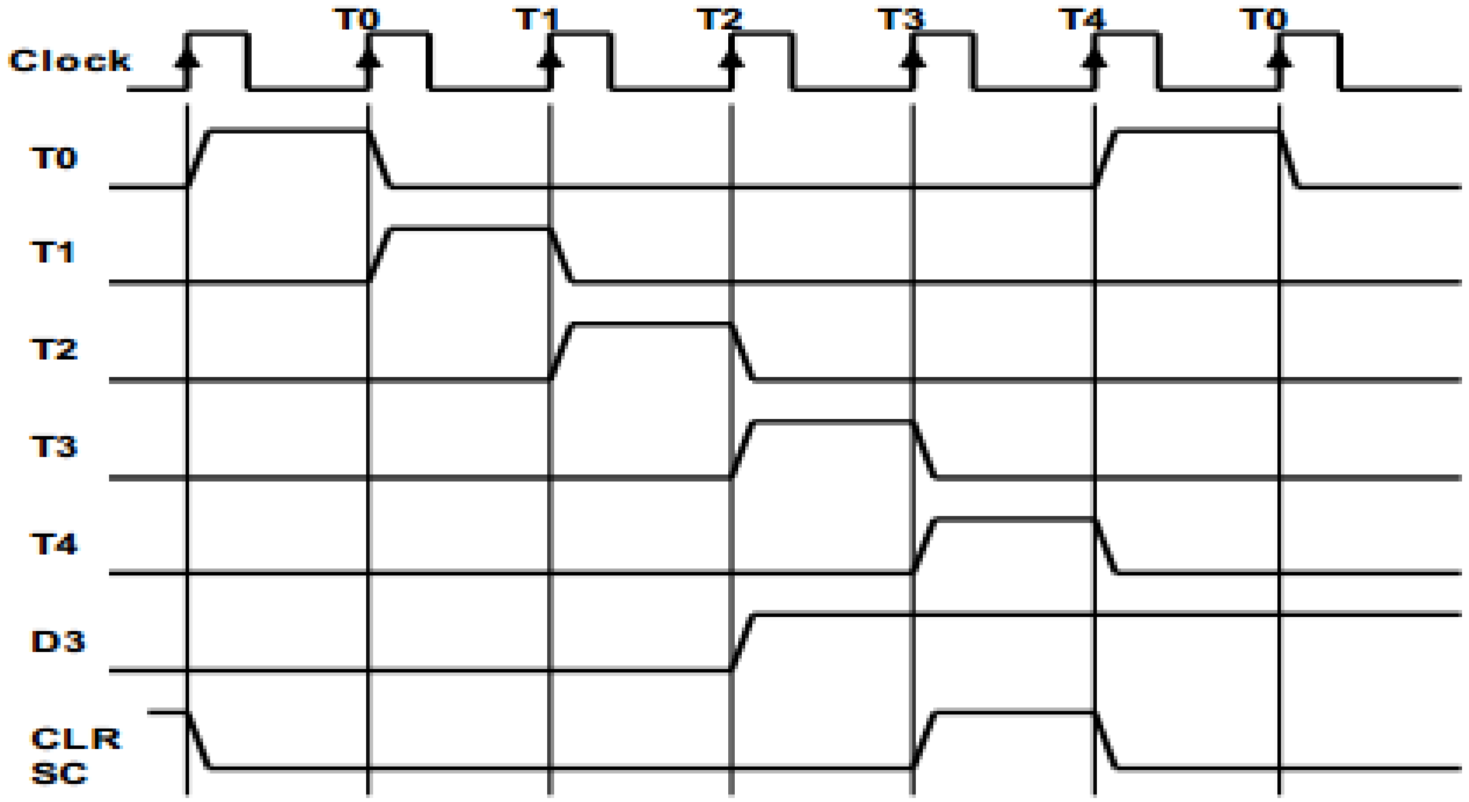
Introduction

- Control logic is implemented with gates, flip flops, decoder and other digital circuits.
- If logic is changed we need to change the whole circuitry
- Expensive
- Fast

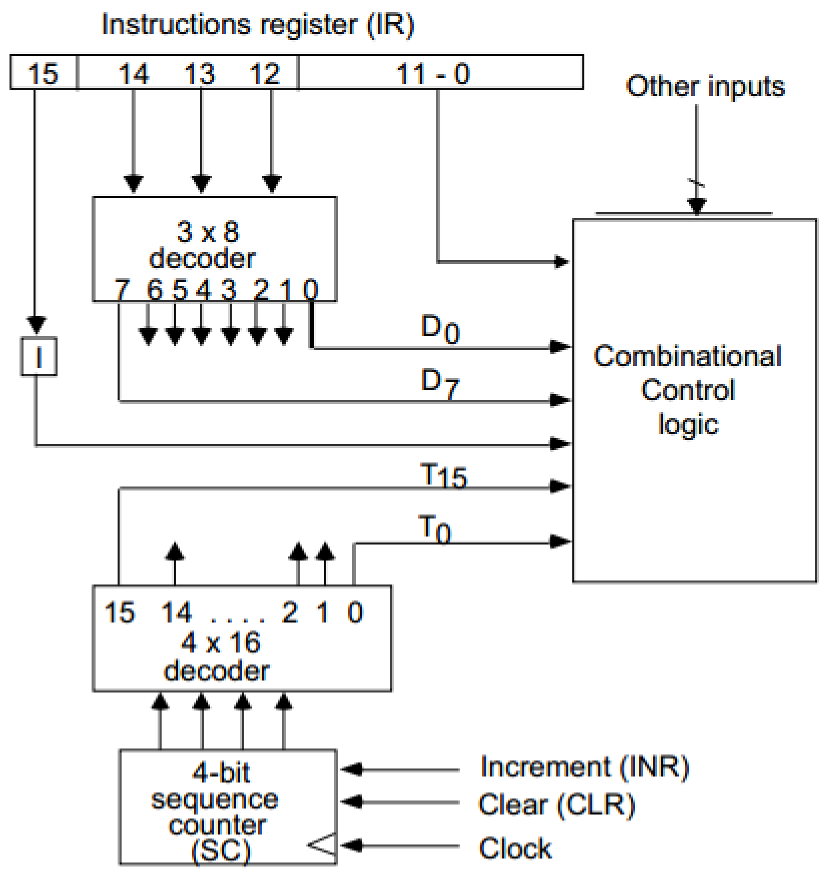
Timing And Control

- Generated by 4-bit sequence counter and 4x16 decoder.
  - The SC can be incremented or cleared.
  - Example: T0, T1, T2, T3, T4, T0, T1 . . .

Timing Signal



Control Unit Of Basic Computer



Control Memory

Control Word

- Control function specifies a microoperation is a binary variable.
- When it is in one binary state microoperation is executed.(other doesn’t change)
- Active state can be 1 or 0 depending on application.
- Control variables at any given time can be represented by string of 0’s and 1’s called control word.

Control Memory

- Contains 2 memory
  - Main Memory
    I. Used to store program
    II. Content can be altered
  - Control Memory
    I. Located in control unit
    II. Contains fixed microprogrammed that can’t be altered by occasional user.
    III. Consists of microinstructions
    IV. Microinstructions specify various register microoperations.

Microinstruction

- Each word in control memory is ***microinstruction.***
- Microinstruction specifies one or more microoperations.
- Sequence of microinstructions are ***Microprogram.***
- Advance technology-Dynamic programming
  - Microprogram loaded from auxiliary device.
  - Writable control memory.
  - But control memory are mostly used for reading.

Micro programmed Control

- **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 one or more microoperations
  ▪ *Sequencing Word* – Contains information needed to decide the next microinstruction address .

Micro Programmed Control Organization

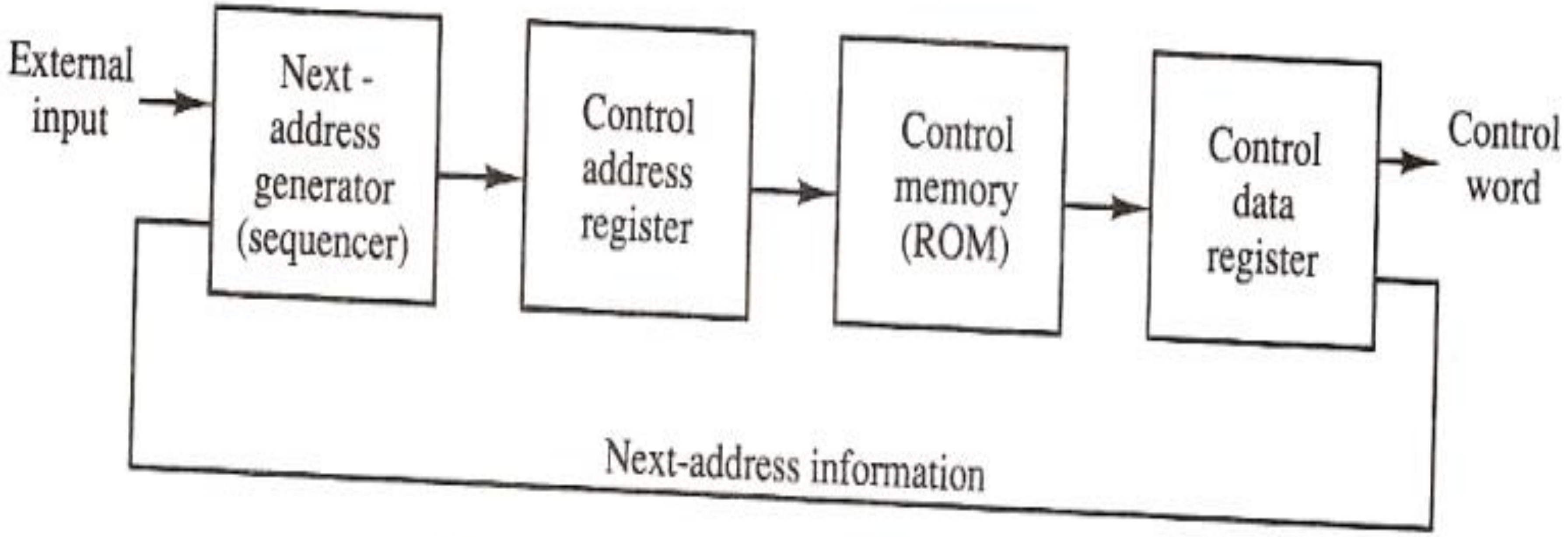


Fig: Microprogrammed control organization

Microprogrammed control Organization

- **Control memory** is ROM, within which all control information is permanently stored.
- **Control memory Address Register** specifies the address of microinstruction.
- **Control Data Register** holds the microinstruction read from memory.

Microprogrammed control Organization

- Microinstructions contains control word that specifies **one or more microoperations.**
- After execution **the location of next microinstruction must be determine**.
  - *Address can be* ***in sequence or out of sequence****.*
  - So, some bits of present microinstruction are used to determine the next address of microinstruction.
- Next address is calculated in next address generator.
  - Also know as sequencer.(Determine next address in sequence)

Address Sequencing (Introduction)

- Microinstructions are stored in control memory in groups.
  - **Called routine.**
  - Every **computer instruction has its own microprogram routine in control memory**, which generate sets of microoperations to execute that instruction.
  - Hardware that control address sequencing of control memory must be **capable of sequencing microinstructions within routine or may able to branch from one routine to other.**

Steps for execution of single computer instruction

- Control address register is loaded with initial address when computer is on– may be fetch routine.
- Fetch routine is sequenced by incrementing CAR.
- At END of routine instruction is in IR.

Calculating Effective Address of operand.

- Can be reached through branch microinstruction.
- At END operand address is available in memory Address Register.

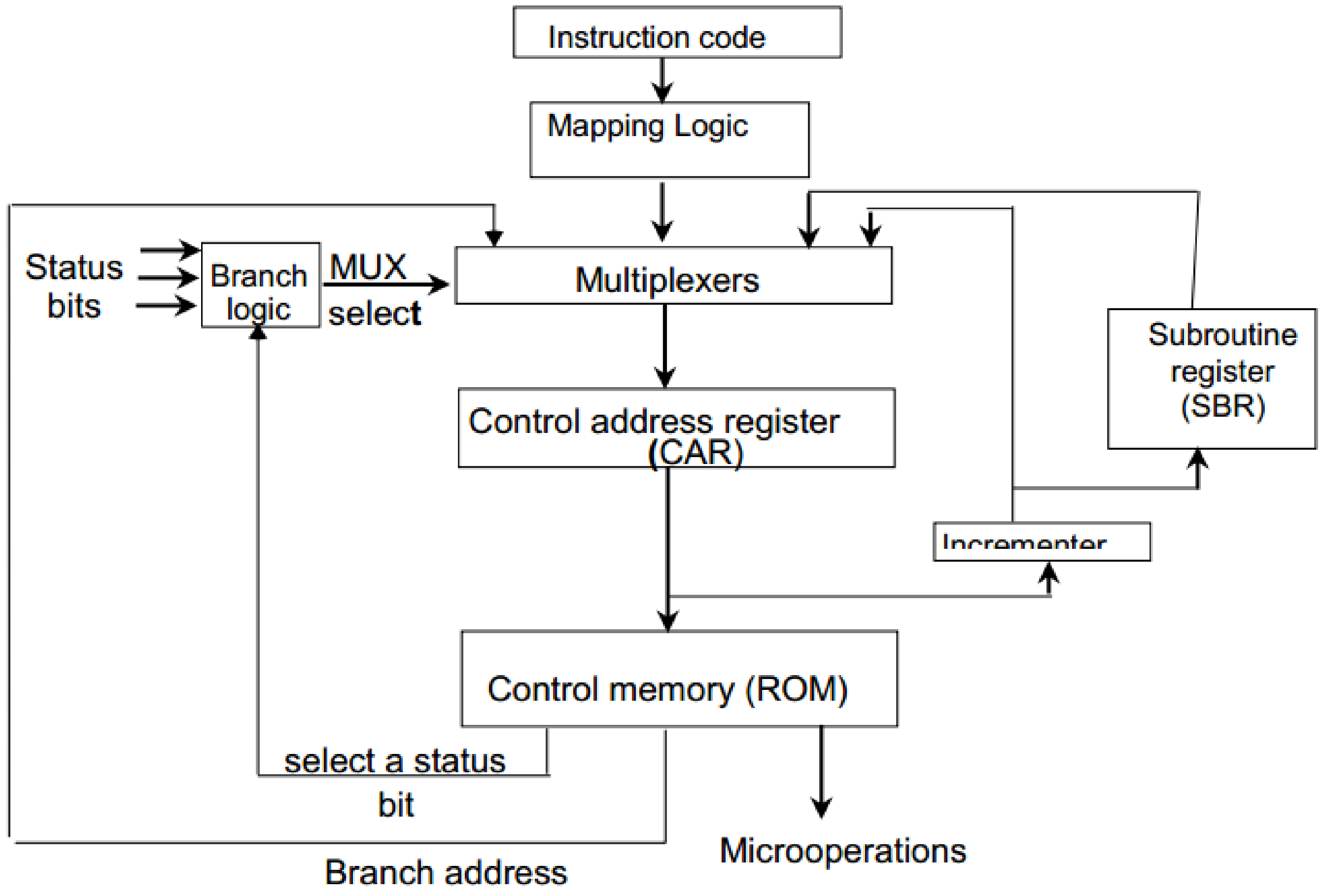
Execute Instruction

- Opcode determine which routine should be run for execution of particular operation.
- Each instruction has its own microprogram routine.
- Mapping process
  - Process of transformation from opcode bit to an address in control memory where routine is located.

Address sequencing capabilities required in control memory

- Incrementing of Control Address register.
- Unconditional branch or conditional branch depending on status bit.
- A mapping process from bit of instruction to control memory address.
- A facility of subroutine call and return.

Block Diagram of Address Sequencer



Conditional Branching

- If Condition is true, set the appropriate field of status register to 1. Conditions are tested for
  - O (overflow),
  - N (negative),
  - Z (zero) and C(carry)
  - Then test the value of that field if the value is 1 take **branch address from the next address field** of the current microinstruction Otherwise **simple increment the address**.

Unconditional branching

- Fix the value of one status bit at the input of the multiplexer to 1. So that always branching is done.

Subroutine

- **Subroutine are program that are used by other routine** to accomplished a particular task.
- Subroutine can be called at any point in main program.
- Some **subroutine are frequently used** in many routine.
- E.g calculation of EA is common to all memory reference instruction.

- When subroutine is called the **incremented address is save in Subroutine register.**
- Register are arranged as stack– used in LIFO manner

Microprogram

- After Microprogrammed Control unit is established
    - Designer generate microcode for control memory(Microprogramming)

Computer Configuration

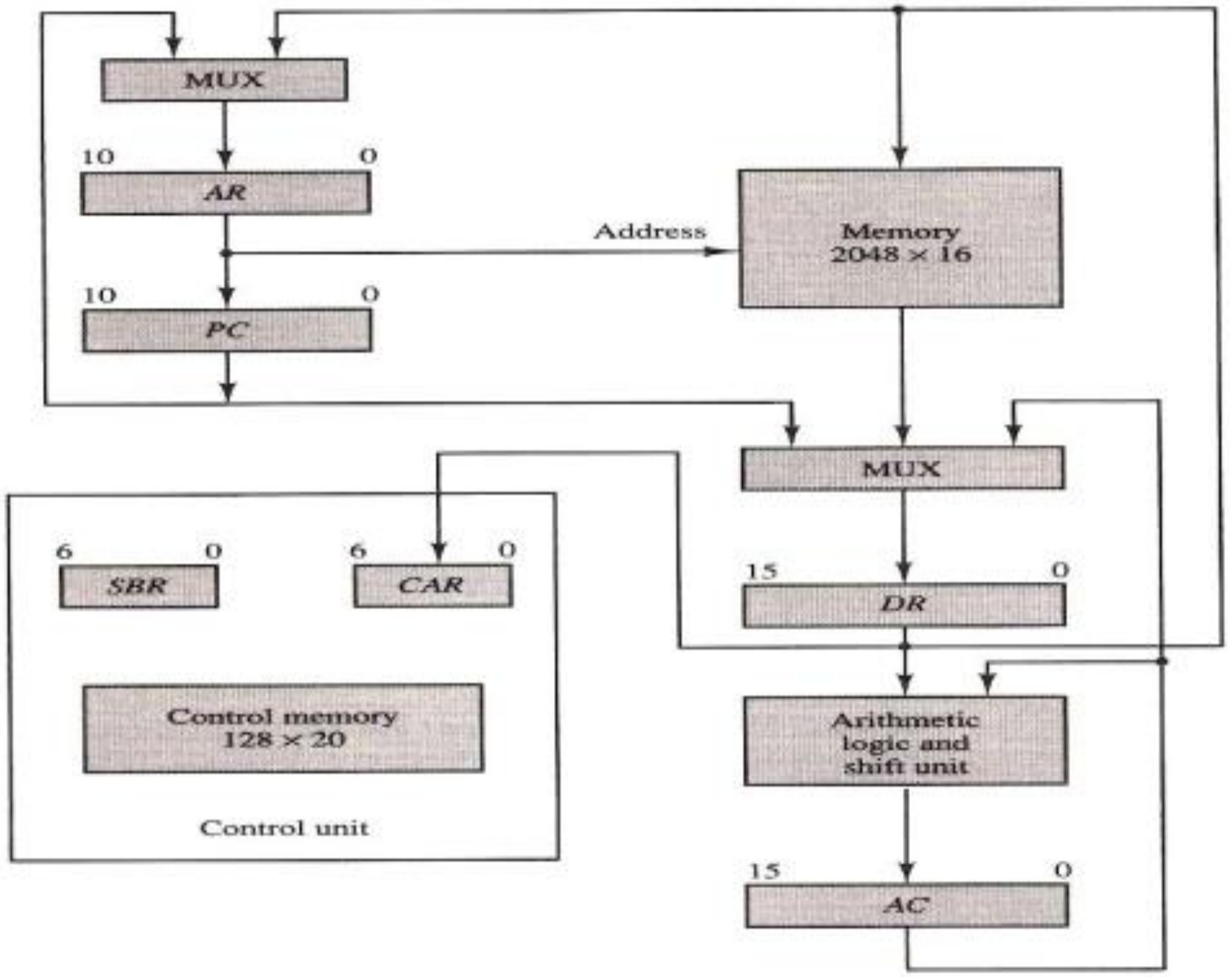


Fig: Computer hardware configuration

Microinstructions

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

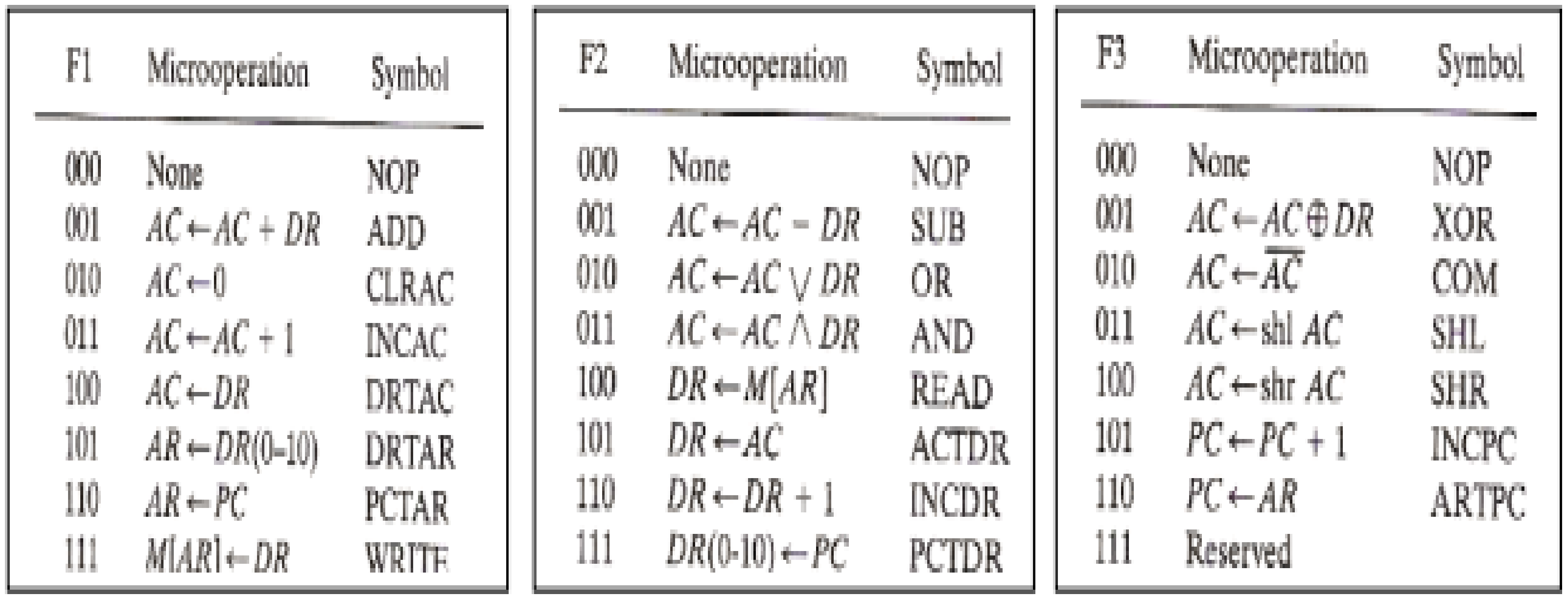
Description of CD

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 ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR (2-5) ← DR (11-14), CAR (0, 1, 6) ← 0

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	$AC \leftarrow AC \vee DR$	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	$DR(0\text{-}10) \leftarrow PC$	PCTDR

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

Symbolic Microinstruction

- Symbols like ADD,CLRAC,READ,COM etc are used to specify microinstructions.
- Assembler is used to translate symbolic microinstruction to its binary equivalent.
- Contains five fields:-
  - **Label**:-This field may be empty or it may specify a symbolic address. A label is terminated with colon(:).
  - **Microoperations field**:- contains one, two or three symbols separated by commas.
  - **CD**:- one of U, I,S,Z.
  - **BR**- can contains 4 symbols. JMP,CALL,RET,MAP.

- **AD**-With symbolic address
  - With Symbol NEXT to designate the next address in sequence.
  - When BR field contains RET or MAP Symbol, the AD field is left empty and is converted to 7 zeroes by assemblers.
- **ORG**- origin address

FETCH ROUTINE Example

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

Sequence of microoperations in the *fetch cycle*:

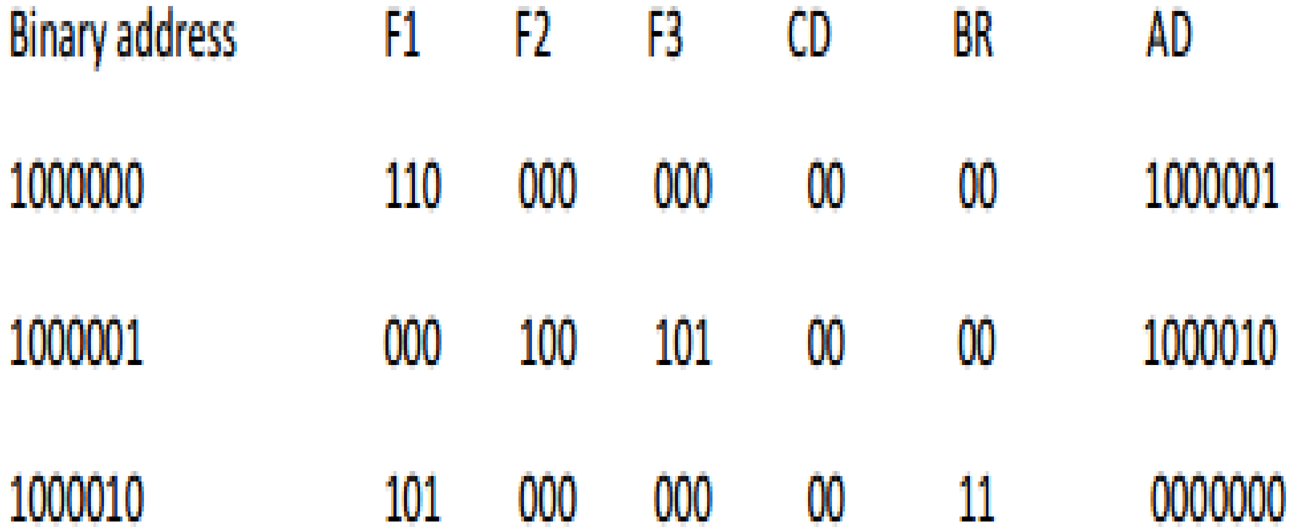
```
AR ← PC

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
```

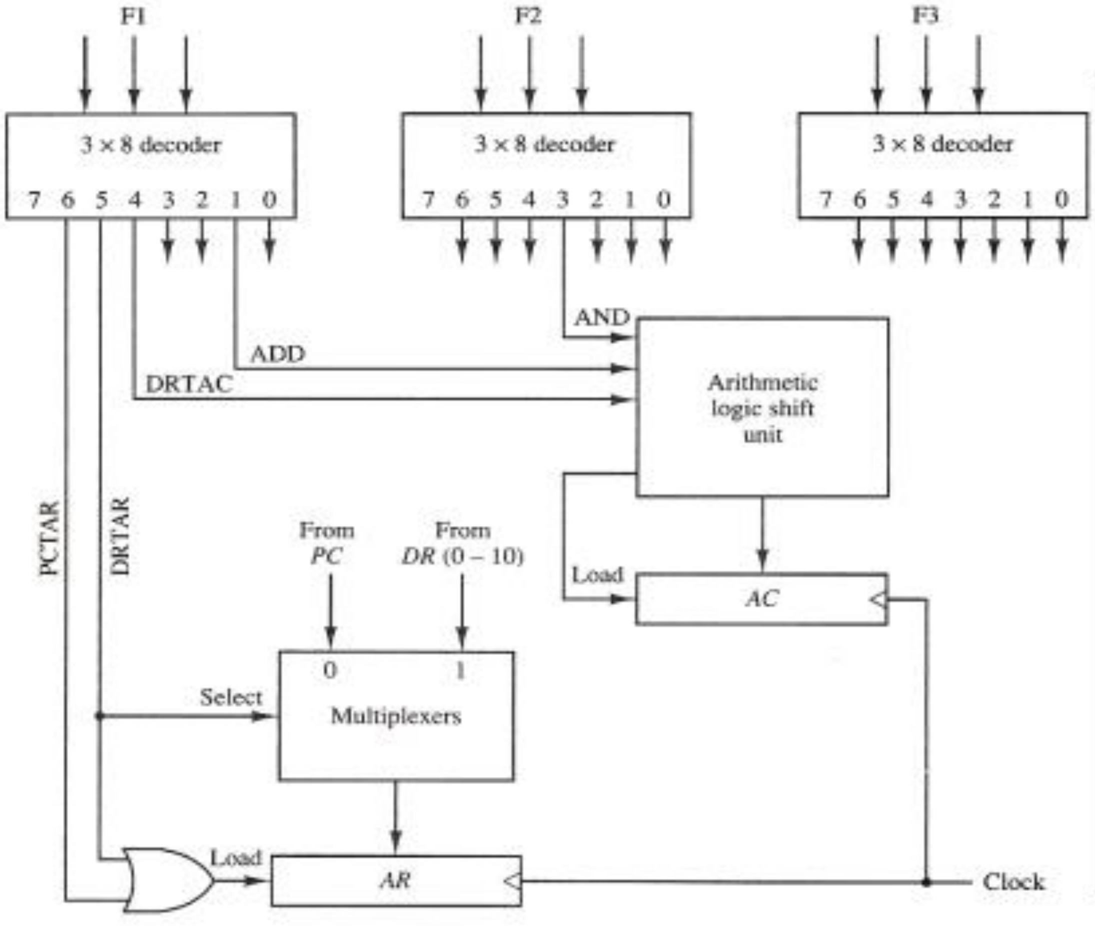
Binary Microprogram

- Program not Stored in memory in Symbolic form.
- Translated by assembler.



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

Design of Control Unit



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 component of microprogrammed are
  - Control memory and next address generator
- Address selection part is microprogram sequencer.
- It load CAR with address of microinstruction to fetch and execute next microprogram.
- Control unit doesn’t contain one register but stack of SBR.

Microprogram Sequencer

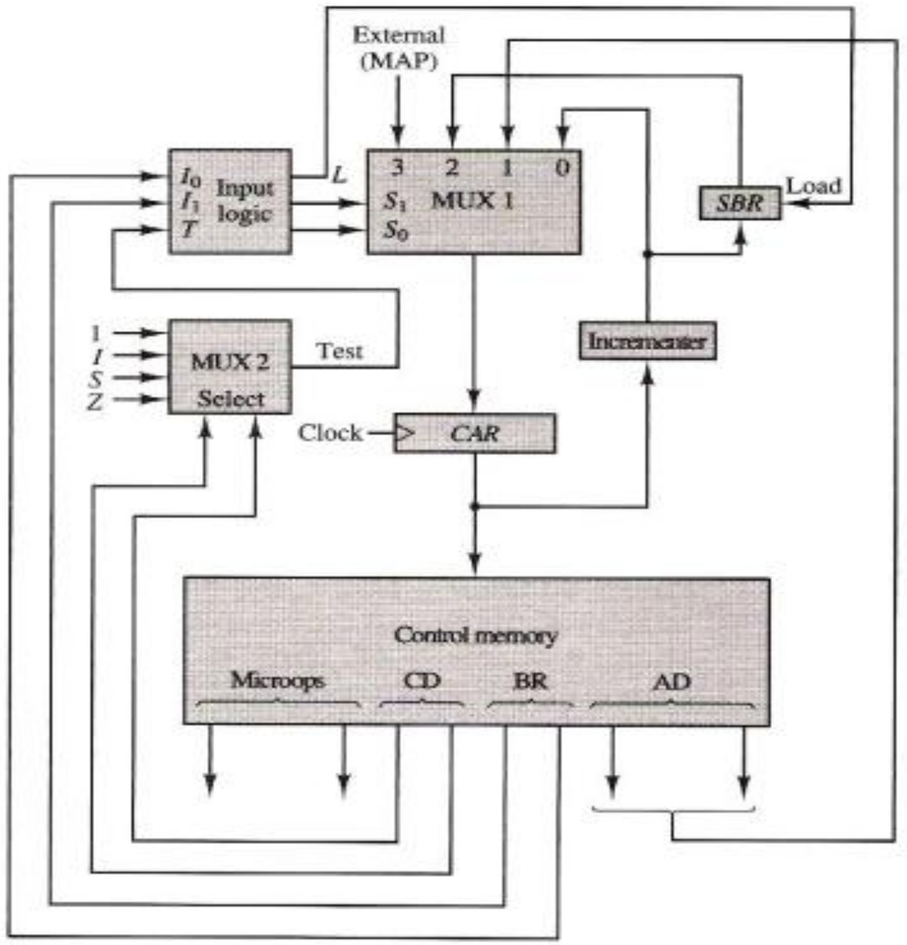


Fig: Microprogram sequencer for a control memory

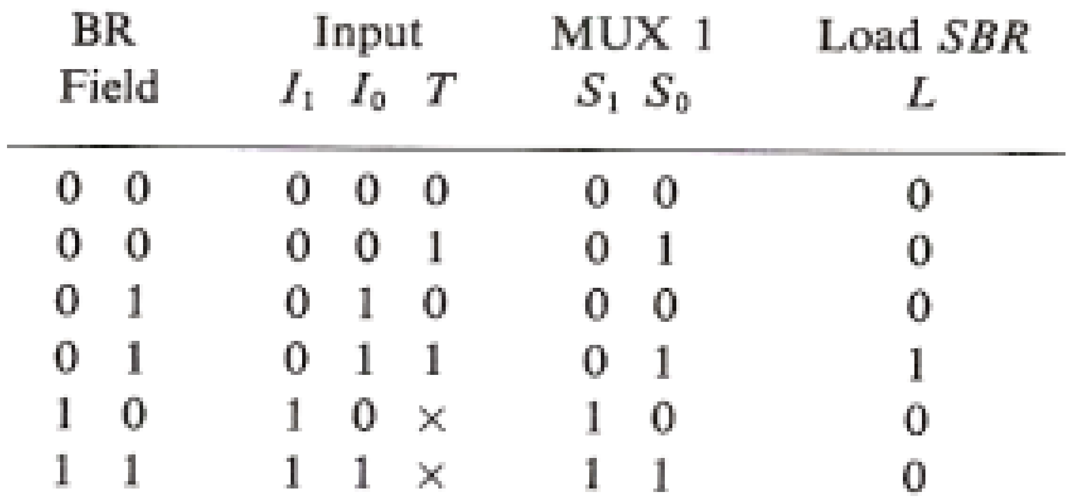
-MUX1 selects an address from one of four sources of and routes it into CAR.

-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.



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

Fig: Input Logic Truth for Microprogram Sequencer