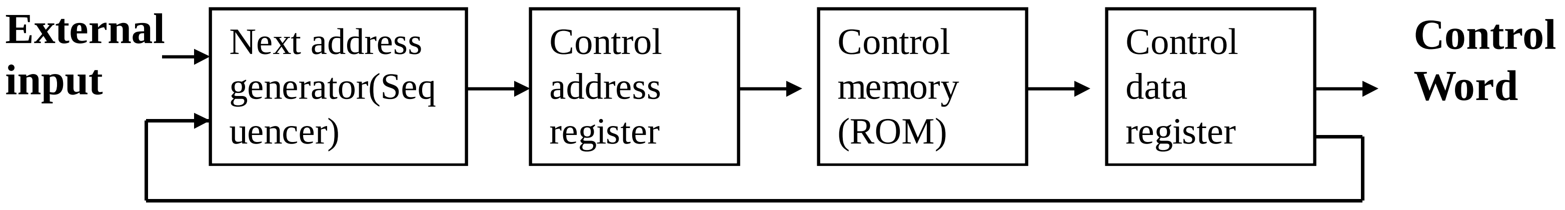
# Microprogrammed Control

- Control Unit
  - Initiate sequences of microoperations. The no. of micro operations in the systems are finite.
    - The control function that specifies a microoperation is a binary variable. When it is in one state the corresponding microoperation is executed. The opposote state does not change the state of registers.
    - Control signal *(that specify microoperations)* in a bus-organized system are
      - groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units
  - Two major types of Control Unit
    - Hardwired Control :
      - The control logic is implemented with gates, F/Fs, decoders, and other digital circuits
      - + Fast operation, - Wiring change(if the design has to be modified)
    - Microprogrammed Control :
      - The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
      - + Any required change can be done by updating the microprogram in control memory,
        - Slow operation
- Control Word: control unit initiates a series of microoperations. during any time certain microoperations are initiated while others are idle.
  - The control variables at any given time can be represented by a string of 1's and 0's is called control word.
- Microprogrammed Control Unit
  - A control unit whose binary control variables are stored in memory (*control memory*).

- Microinstruction : *Each Word in Control Memory contains within it a microinstruction.*
  - The microinstruction specifies one or more microoperations
- Microprogram
  - A sequence of microinstruction
    - » Dynamic microprogramming : *Control Memory* = RAM
      - RAM can be used for writing (*to change a writable control memory*)
      - Microprogram is loaded initially from an auxiliary memory such as a magnetic disk
    - » Static microprogramming : *Control Memory* = ROM
      - Control words in ROM are made permanent during the hardware production.
- Microprogrammed control Organization :



**Next address Information**

- 1) Control Memory: A memory is part of a control unit.
  - » Computer Memory
    - Main Memory : for storing user program (*Machine instruction/data*)
    - Control Memory : for storing microprogram that can not be altered.
    - The microprogram consist of *Microinstruction*
- 2) Control Address Register
  - » Specify the address of the microinstruction
- 3) Sequencer (= *Next Address Generator*)
  - » Determine the address sequence that is read from control memory
  - » Next address of the next microinstruction can be specified several way depending on the sequencer input

- 4) Control Data Register (= *Pipeline Register* )
  - » Hold the present microinstruction while the next add. Is computed & read from control memory
  - » Allows the execution of the microoperations specified by the control word ***simultaneously*** with the generation of the next microinstruction. This requires two phase clock, with one clock applied to add. Register and other to data register.

## Address Sequencing:

Microinstructions are stored in control memory in groups, with each group specifies a *routine*. The hardware that controls the address sequencing of the control memory must be able of sequencing the microinstruction with a routine and be able to branch from one routine to another.

An initial address is loaded into the control address register when power is turned on.this address is the address of the first microinstruction that activates the fetch routine. After the end of fetch routine, the instruction is in the instruction register of the computer.

The control memory next must goes through the routine that determines the effective address of the operand. After computing the effective address the address of the operand is available in the memory address register.

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 upon the operation code part of instruction.

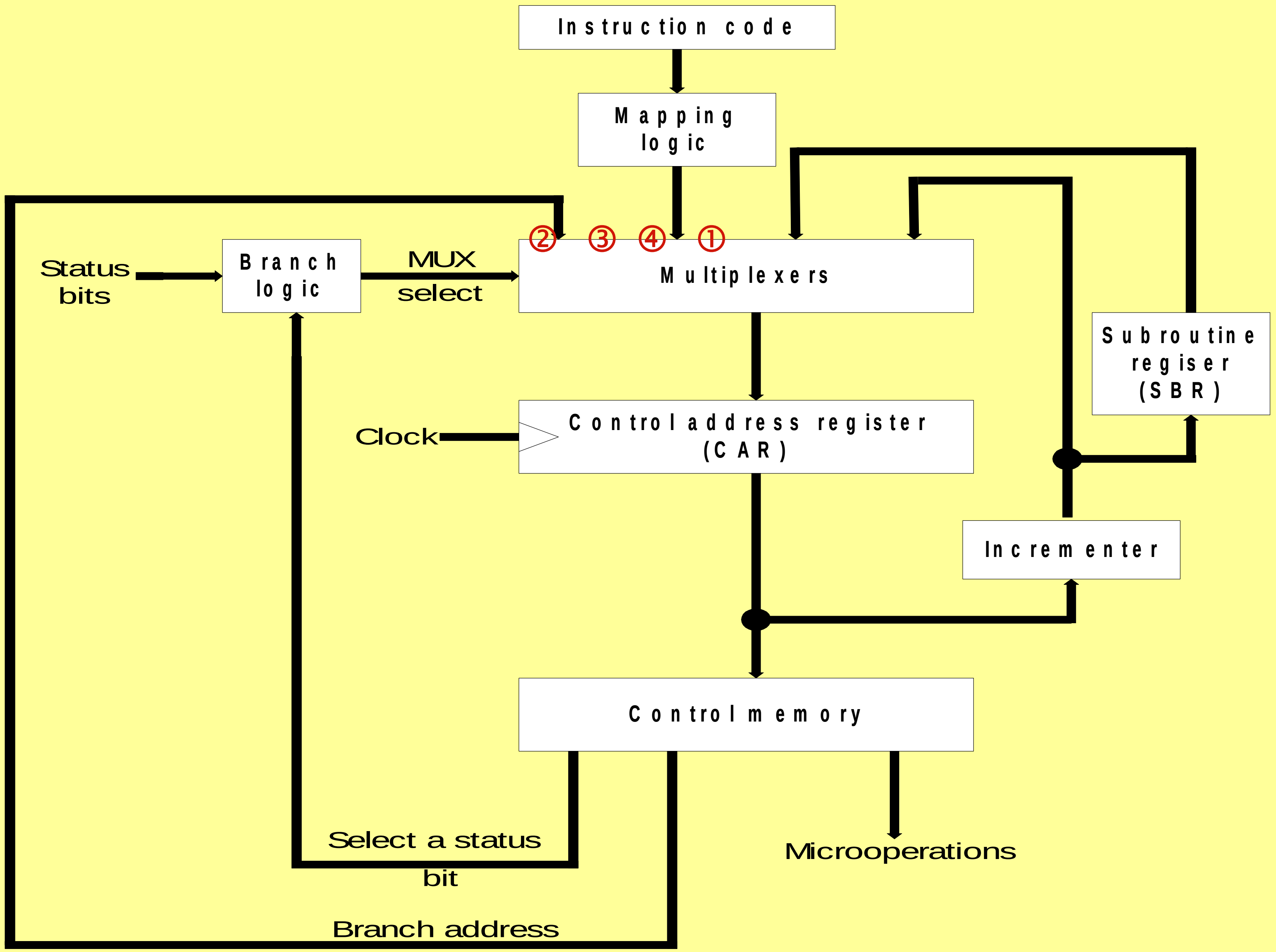
Each instruction has its own microprogram routine stored in a given location of the control memory.

The transformation from the instruction code bits to an address in the control memory where the routine is located is called as ***Mapping.***

After the execution of the instruction control must return to the fetch routine.

### Address Sequencing Capabilities :

- 1) Incrementing of the control address register
- 2) Unconditional branch or conditional branch, depending on status bit conditions
- 3) Mapping process ( *bits of the instruction address for control memory* )
- 4) A facility for subroutine return



Instruction code
Mapping logic
②
③
④
①
Status bits
Branch logic
MUX select
Multiplexers
Subroutine register (SBR)
Clock
Control address register (CAR)
Incrementer
Control memory
Select a status bit
Microoperations
Branch address

## ◆Selection of address for control memory :

- Multiplexer
  - ① CAR Increment
  - ② JMP/CALL
  - ③ Mapping
  - ④ Subroutine Return
- CAR : Control Address Register
  - »CAR receive the address from 4 different paths
    1) Incrementer
    2) Branch address from
       control memory
    3) Mapping Logic
    4) SBR : Subroutine Register
- SBR : Subroutine Register
  - »Return Address can not be stored in ROM
  - »Return Address for a subroutine is stored in SBR

# Conditional Branching

**Special bits:** Status conditions are special bits in  the system that provide parameter information such as carry-out of an adder, sign bit of number, mode bits of an instruction, input /output status condition.

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.

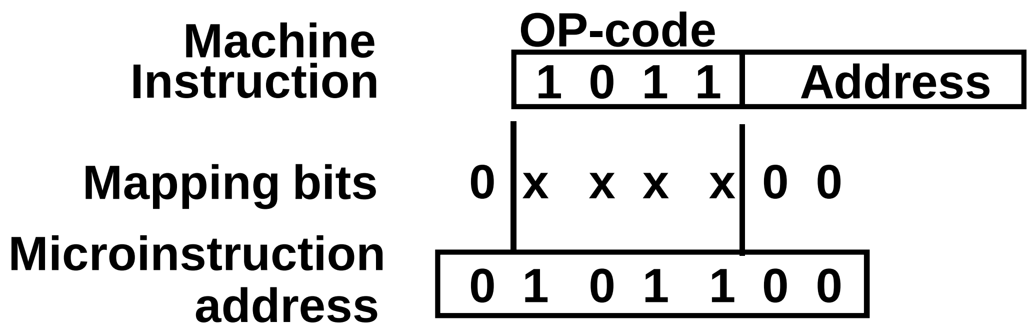
**Branch Logic:** branch logic is used to test the specified condition and branch to the indicated address if the condition is met; otherwise the address register is incremented.this can be implemented with the help of multiplexer.

Ex. Let there are eight status bit conditions in the system. Three bits in the microinstruction are used to specify one of eight status bits. If the selected status bit is in the 1 state the output of the multiplexer is 1, otherwise 0. The 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register otherwise address register to be incremented.

The unconditional branch microinstruction can be implemented by loading the branch address from control memory into control address register.

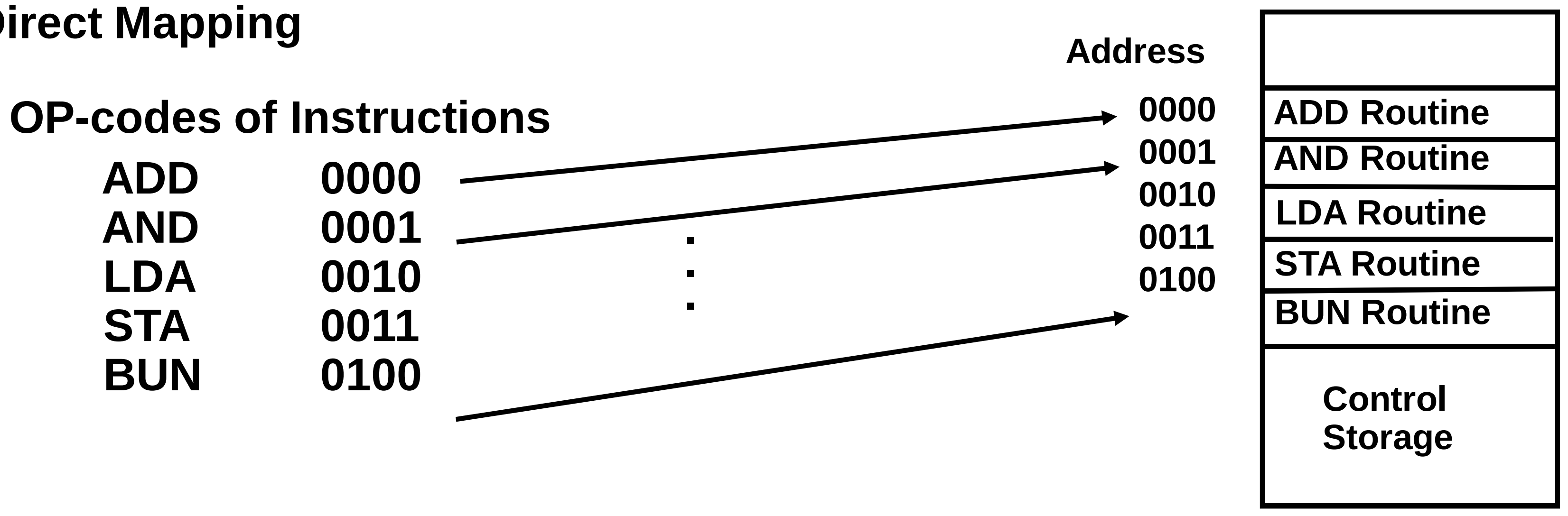
## MAPPING OF INSTRUCTIONS TO MICROROUTINES

**Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram**



- 4 bit Opcode = specify up to 16 distinct instruction
- Mapping Process : Converts *the 4-bit Opcode* to *a 7-bit control memory address*
  - »1) Place a “0” in the most significant bit of the address
  - »2) Transfer 4-bit Operation code bits
  - »3) Clear the two least significant bits of the CAR

## Direct Mapping



Mapping Function : Implemented by *Mapping ROM* or *PLD. A PLD is similar to ROM except that it uses AND and OR gates with internal electronic fuses.the interconnection between AND, OR and outputs can be programmed as in ROM*

## Subroutine

- Subroutines are program 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 microprogram contain identical section of code. Microinstruction can be saved by employing subroutines that used common section of micro-code. Ex. The sequence of micro operations 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.

## Design of Control Unit

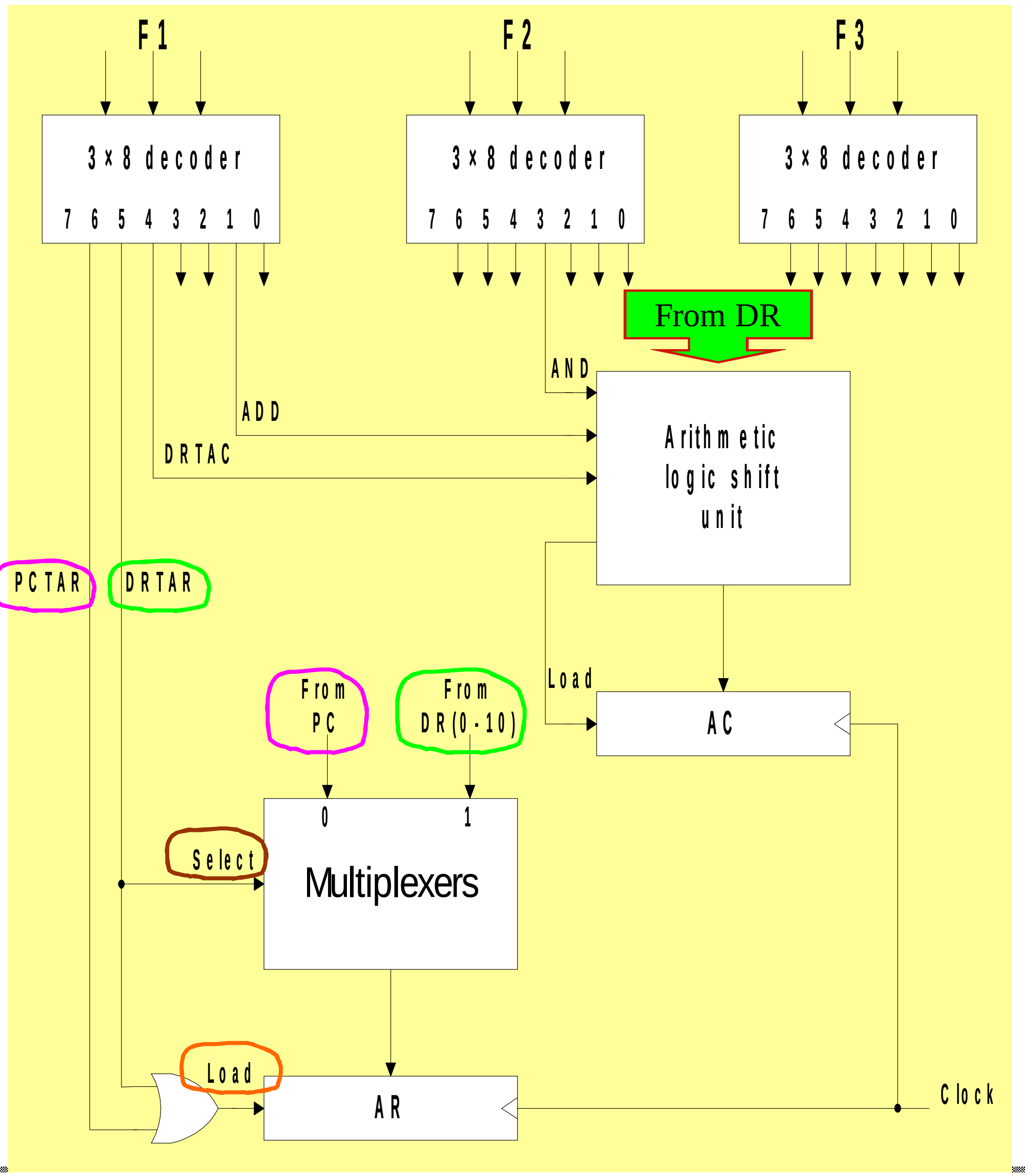
The bits of microinstruction are usually divided into fields, with each field defining a distinct, separate function. The various fields available in the instruction format provide control bits to initiate the microoperation.

- Decoding of Microinstruction Fields :
  - F1, F2, and F3 of Microinstruction are decoded with a 3 x 8 decoder
  - Output of decoder must be connected to the proper circuit to initiate the corresponding microoperation (***Tab. 7-1***)

F1 = 101 (5) : **DRTAR (The next clock pulse transition transfer the content of DR(0-10) to AR)**

F1 = 110 (6) : **PCTAR**

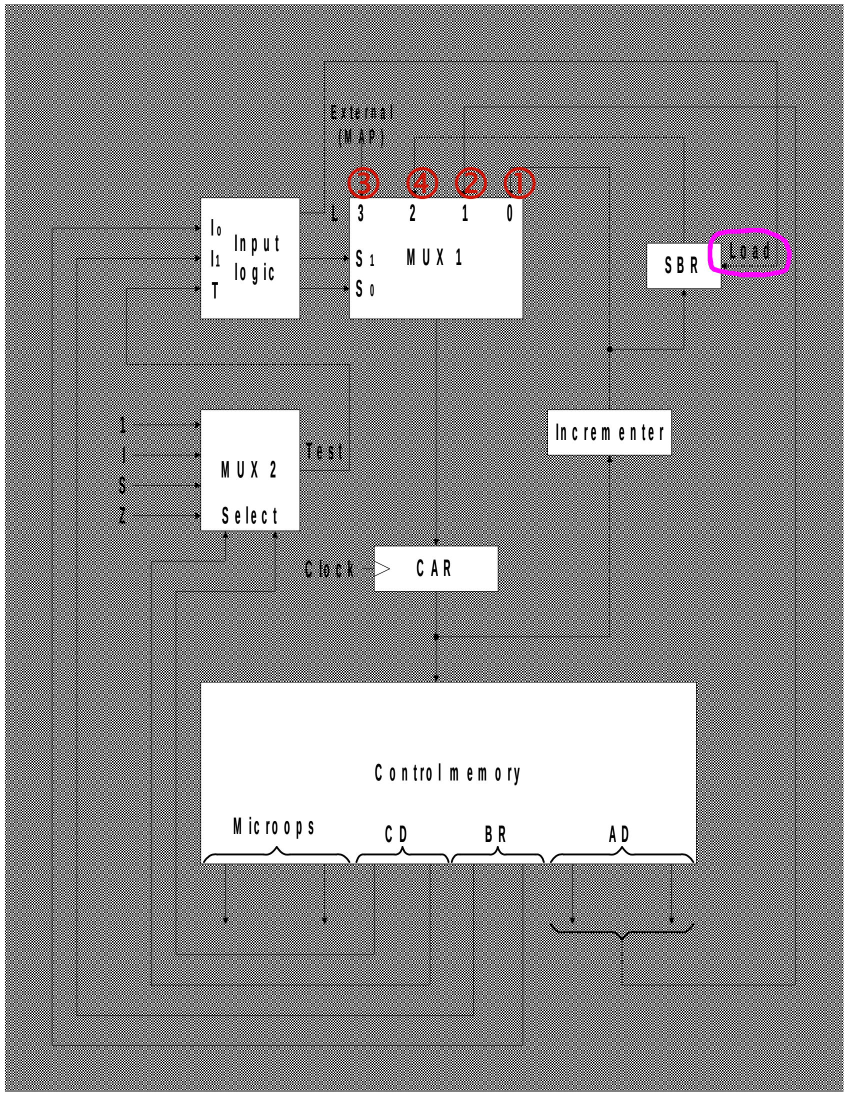
- Output 5 and 6 of decoder F1 are connected to the load input of AR (*two input of OR gate*)
- Multiplexer select the data from DR when output 5 is active
- Multiplexer select the data from AC when output 5 is inactive



- Arithmetic Logic Shift Unit
  - » Instead of using the gates to generate the control signal marked by AND, ADD, and DR Control signal will be now come from the *output of the decoders* associated with the AND, ADD, and DRTAC respectively.

## Microprogram Sequencer : *Fig. 7-8*

- Microprogram Sequencer select the next address for control memory
- MUX 1
  - » Select an address source and route to CAR
    - ① CAR + 1
    - ② JMP/CALL
    - ③ Mapping
    - ④ Subroutine Return
  - » JMP 와 CALL의 차이점
    - ▪ JMP : AD가 MUX 1의 2번을 통해 CAR로 전송
    - ▪ CALL : AD가 MUX 1의 2번을 통해 CAR로 전송되고, 동시에 CAR + 1(Return Address) 이 LOAD 신호에 의해 SBR에 저장된다.
- MUX 2
  - » Test a status bit and the result of the test is applied to an input logic circuit
  - » One of 4 Status bit is selected by Condition bit (**CD**)
- Design of Input Logic Circuit
  - » Select one of the source address(**$S_0$, $S_1$**) for CAR
  - » Enable the load input(**L**) in SBR



- Input Logic Truth Table : ***Tab. 7-4***
  - » Input :
    - $I_0$, $I_1$ from Branch bit (**BR**)
    - T from MUX 2 (**T**)
  - » Output :
    - MUX 1 Select signal (**$S_0$, $S_1$**)

      $S1 = I_1I_0' + I_1I_0 = I_1(I_0' + I_0) = I_1$

      $S0 = I_1'I_0'T + I_1'I_0T + I_1I_0$

      $= I_1'T(I_0' + I_0) + I_1I_0$

      $= I_1'T + I_1I_0$
    - SBR Load signal (**L**)

      $L = I_1'I_0T$

| BR Field | | Input | | | MUX 1 | | Load SBR |
|---|---|---|---|---|---|---|---|
| | | I1 | I0 | T | S1 | S0 | 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 | x | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | x | 1 | 1 | 0 |

① CAR + 1
② JMP
① CAR + 1
② CALL
③ MAP
④ RET

**CALL**