UNIT 4

MICROPROGRAMMED CONTROL

Contents

- ***Control memory***
- ***Address sequencing***
- ***Computer configuration***
- ***Microinstruction format***
- ***Symbolic microinstructions***
- ***Symbolic micro programs***
- ***Control unit operation***
- ***Design of control unit***

Introduction

1. Control Memory

- The function of the ***control unit*** in a digital computer is to initiate ***sequences of micro-operations***.
- ***Micro-operations-****Series of steps involving processor registers (the functional or atomic operations of a processor).*
- Two ways to generate signals from Control Unit:
  1. **Hardwired CU:** use of conventional logic design techniques
  2. **Microprogrammed CU:** use of microprogram, binary control variables
- In a bus-organized systems, the control signals that specify ***micro-operations*** are ***groups of bits*** that select the paths in multiplexers, decoders, and arithmetic logic units.

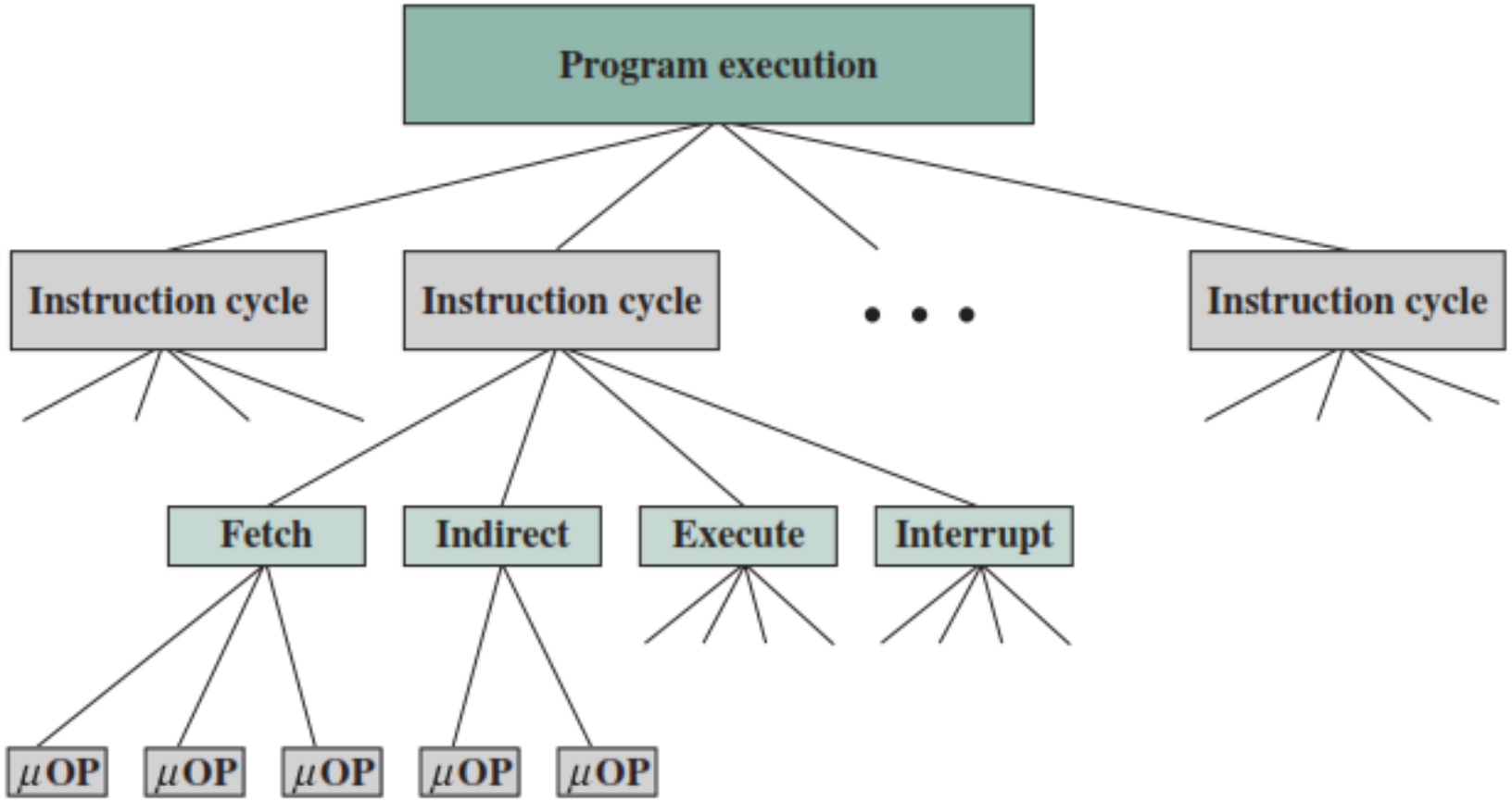


Fig: Constituent Elements of a Program Execution

- A memory that is part of a control unit is referred to as a ***control memory***.
- Each word of Control Memory or set of micro-operations occurring at a time –***Micro-instruction***
- Sequence of micro-instruction- ***Microprogram (strings of 0's and 1's)***
- ***Control Memory- ROM or Dynamic***
- Reading the contents of one location of ***control memory*** at any instance is nothing but a ***microinstruction***.
- Each output line (data line) of microprogram memory corresponds to one control signal.
- If the content of the memory cell is ***0***, ***no signal*** and if it is ***1***, signal is generated at that instant of time.

- Computer that employs a ***microprogrammed control unit*** will have two separate memories:
  - ✓ A main memory
  - ✓ A control memory
- The ***control memory*** is assumed to be ***a ROM***, within which all control information is permanently stored.
  - *The control address register specifies the address of the microinstruction.*
  - *The control data register holds the microinstruction read from memory.*
- Microinstruction contains bits for:
  1. ***initiating micro-operations*** in the processor and
  2. determining the ***address sequence*** for the control memory.

Microprogram Sequencer

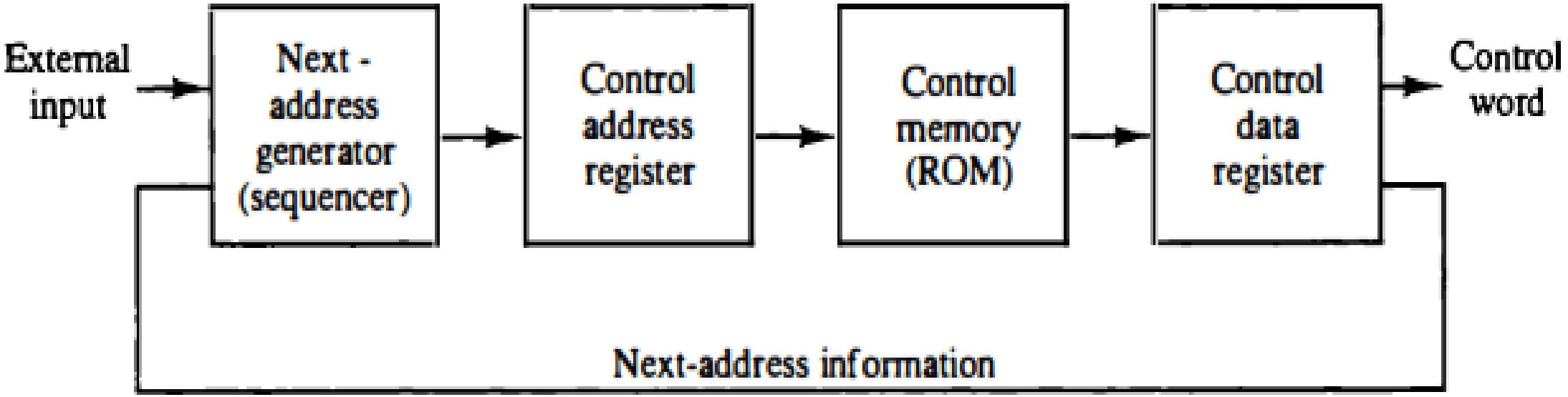


Fig3.1: Microprogrammed control organization

Sequencer

- **Some times called *next address generator*-***determines the address sequence that is read from control memory.***
- **Typical functions:**
  1. *Incrementing the control address register by one*
  2. *Loading an address into the control address register from control memory*
  3. *Transferring an external address*
  4. *Loading an initial address to start the control operations*

Pipeline Register

- ***Data register*** sometimes called ***pipeline register.***
- Allow simultaneously:
  1. The execution of the micro-operations specified by the control word
  2. The generation of the next microinstruction.
  - This configuration requires a *two-phase clock*
- The system can operate by applying a *single-phase clock to the address register.*
  - *Without the control data register*
  - *Thus, the control word and next-address information are taken directly from the control memory.*

- Address sequencer must have capabilities of finding address of next microinstruction 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.***

2. Address Sequencing

- Microinstructions are stored in control memory in groups, with each group specifying a ***routine*.**
- Process of finding ***address of next micro-instruction*** to be executed is called ***address sequencing.***
- Each computer instruction has its own microprogram routine in control memory to generate the micro-operations that execute the instruction.

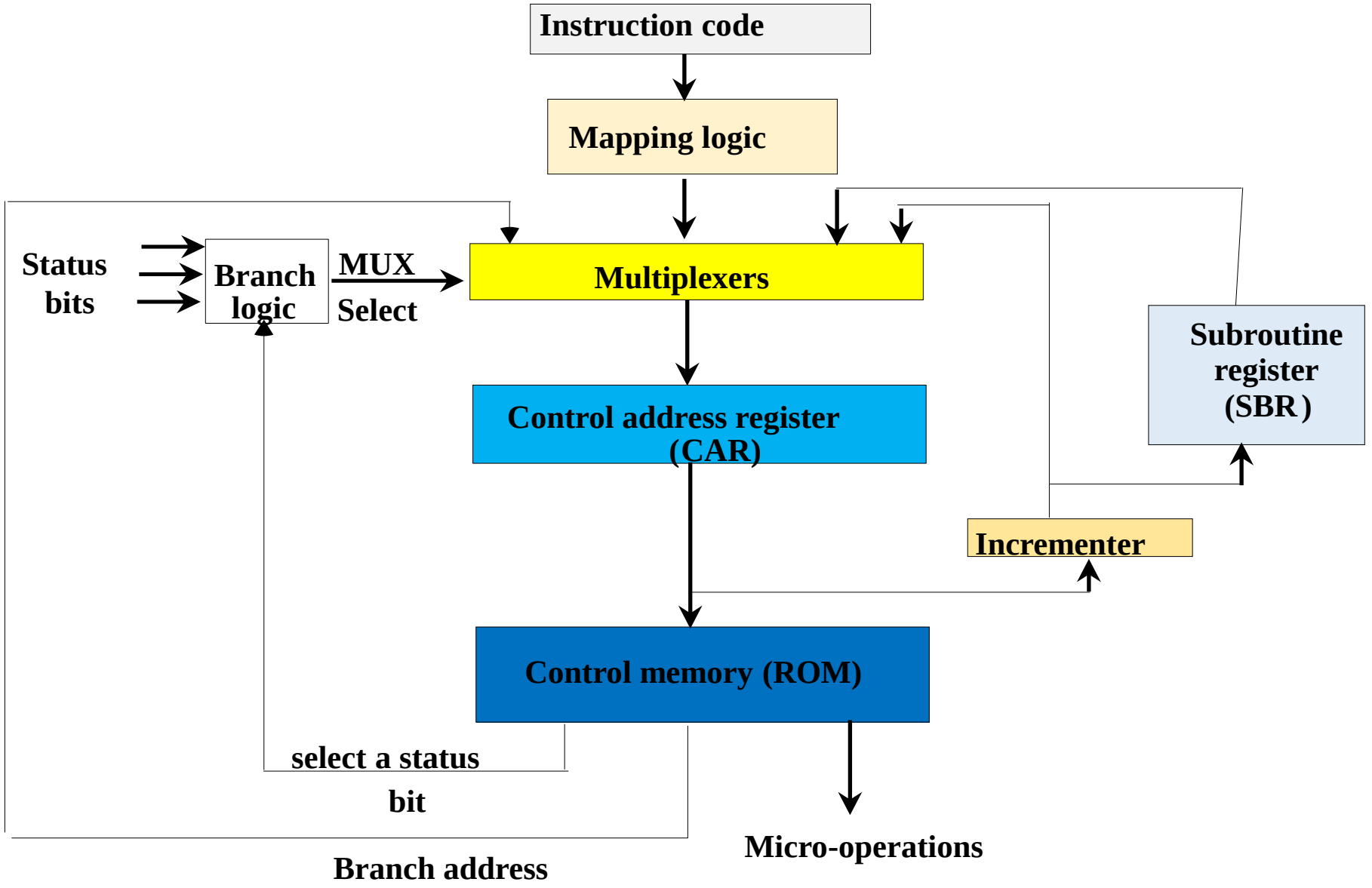


Fig3.2: Selection of address for control memory

Process of sequencing:

- *When power turned on, initial address is loaded into the CAR.*
- *This address is usually the address of the first microinstruction that activates the instruction fetch routine.*
- *The control memory next must go through the routine that determines the effective address of the operand.*
- *The next step is to generate the micro-operations that execute the instruction fetched from memory.*

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

- The microinstruction in control memory contains:
  - *a set of bits to initiate micro-operations in computer registers.*
  - *Other bits to specify the method by which the next address is obtained.*

- The address sequencing capabilities required in a control memory are:
  a) *Incrementing of the control address register.*
  b) *Unconditional branch or conditional branch, depending on status bit conditions.*
  c) *A mapping process from the bits of the instruction to an address for control memory.*
  d) *A facility for subroutine call and return.*

Conditional Branch:

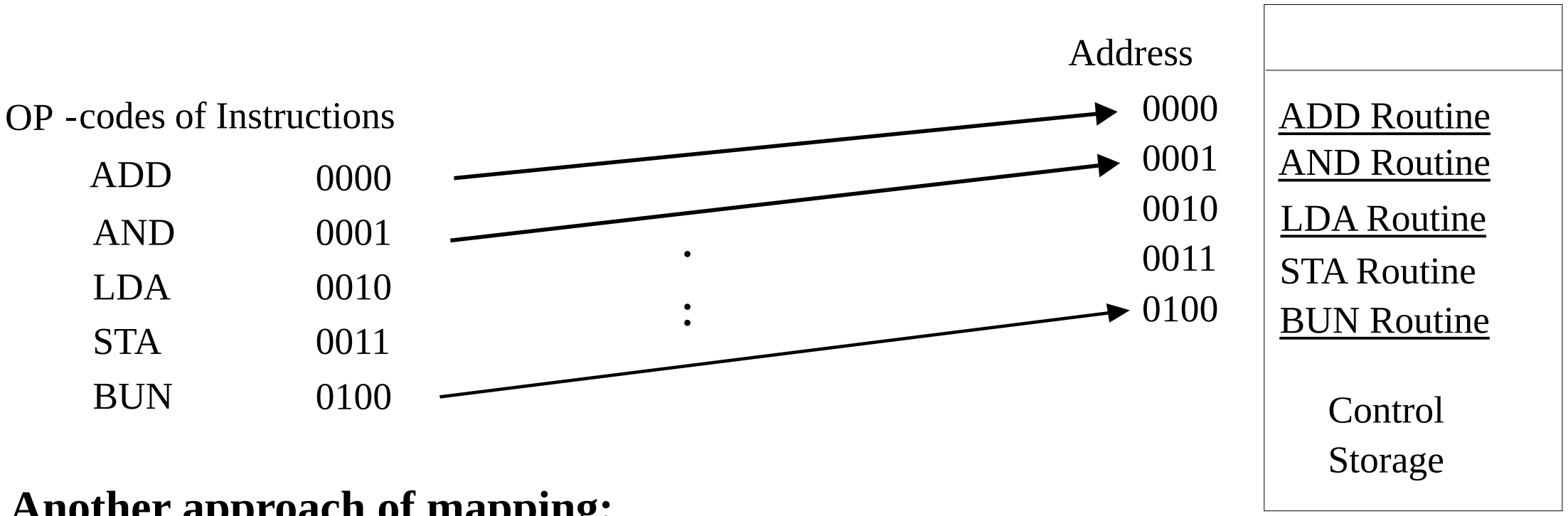
- If Condition is true, set the appropriate field of status register to 1.
- Conditions are tested for O (overflow), N (negative), Z (zero), C (carry), etc.
- Then test the value of that field, if the value is 1 take branch address from the next address field of the current microinstruction, otherwise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

Unconditional Branch:

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

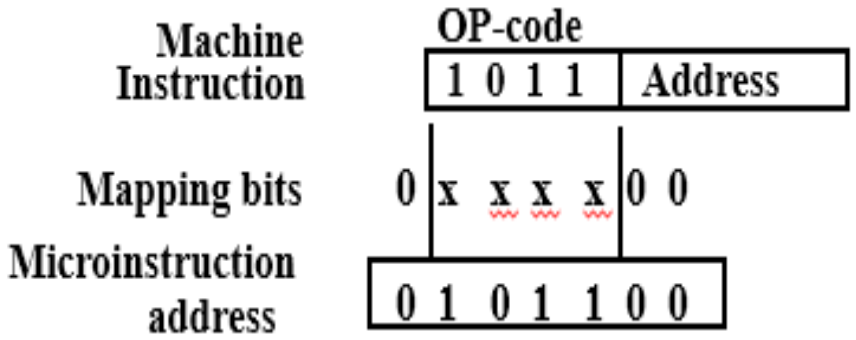
Mapping:

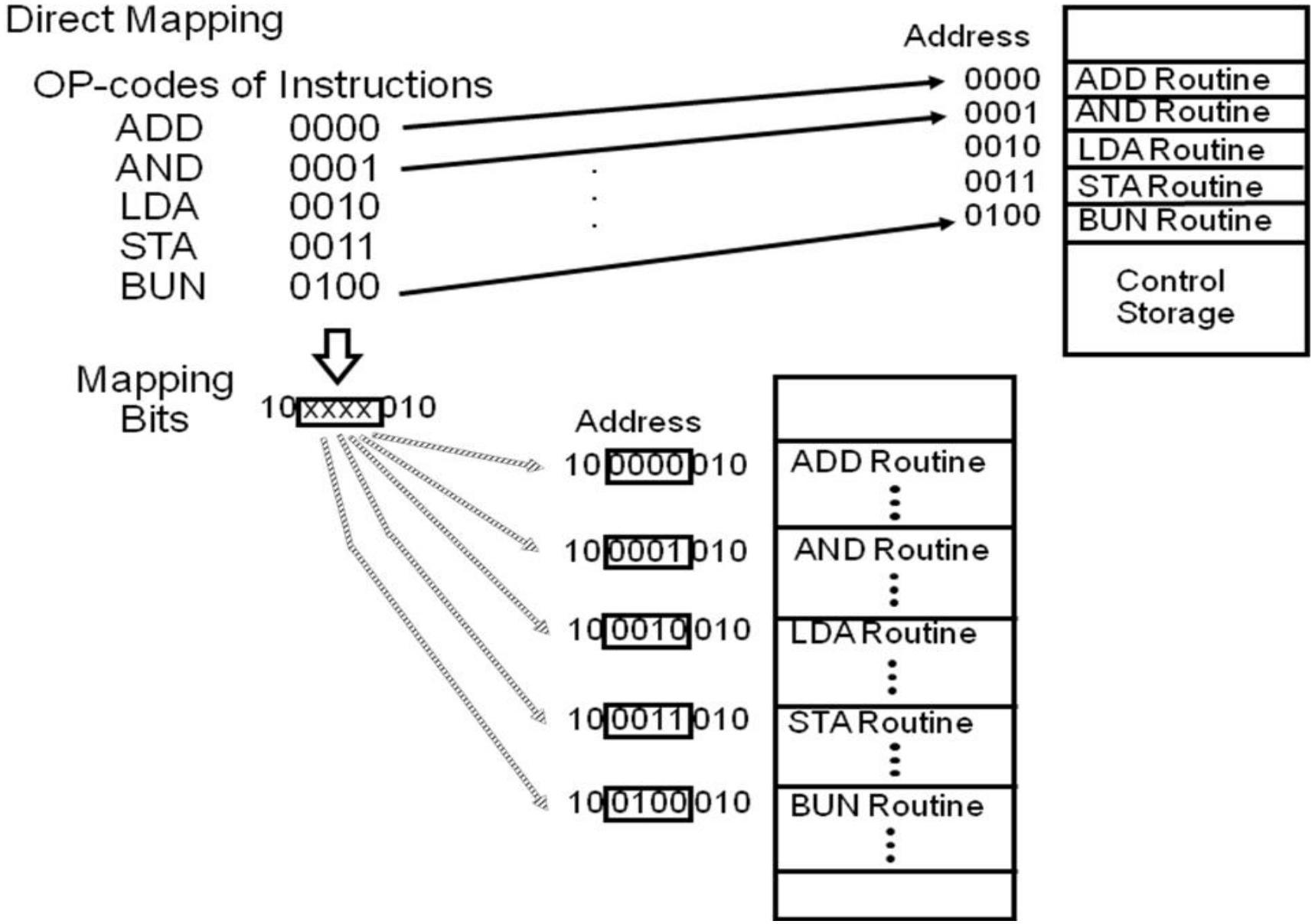
- The transformation of the instruction code bits to an address in control memory where the routine is located is referred to as a *mapping process*.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.
- **Direct mapping:**
  - Directly use opcode as address of Control memory.



Another approach of mapping:

- Modify Opcode to use it as an address of control memory.




Direct Mapping
OP-codes of Instructions
ADD 0000
AND 0001
LDA 0010
STA 0011
BUN 0100
Address
0000
0001
0010
0011
0100
ADD Routine
AND Routine
LDA Routine
STA Routine
BUN Routine
Control Storage
Mapping Bits
10 XXXX 10
Address
10 0000 10
10 0001 10
10 0010 10
10 0011 10
10 0100 10
ADD Routine
AND Routine
LDA Routine
STA Routine
BUN Routine

Mapping function implemented by ROM or PLA:

- The opcode is used to address the ROM address where control address is located and finally the microinstruction address of control memory is generated through mapping process.
- Mapping generally uses AND operation followed by OR operation.

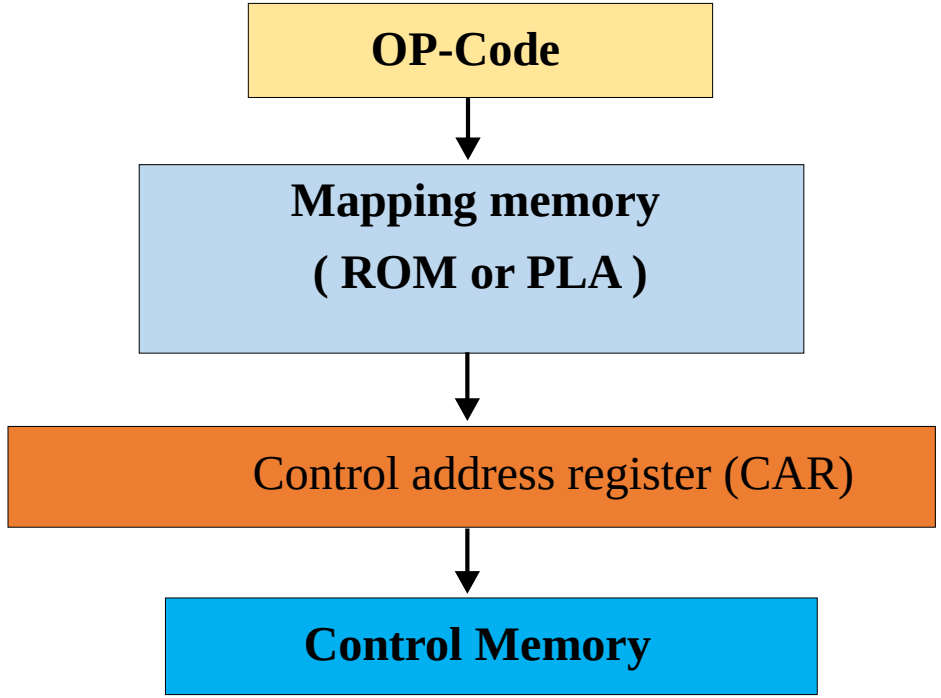


Fig 3.3: Mapping Function Implemented by ROM and PLD

Subroutines:

- Subroutines are programs that are used by other routines to accomplish a particular task.
- Can be called from any point within the main body of the microprogram.
- Designed/written to ease the program flow and for better understanding.
- Help in reducing the repetitive coding.
- Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring that address during return.
- Organize the registers in a last-in, first-out (LIFO) stack.

3. Computer Configuration

- The designer generates the microcode for the control memory.
- This code generation is called microprogramming and similar to conventional machine language programming.
- To illustrate microprogramming we use the block diagram of the computer as shown below. It consists of two memory units:
  - ***a main memory for storing instructions and data, and***
  - ***a control memory for storing microprograms***
- Four registers are associated with the processor unit and two with the control unit.

Computer Configuration

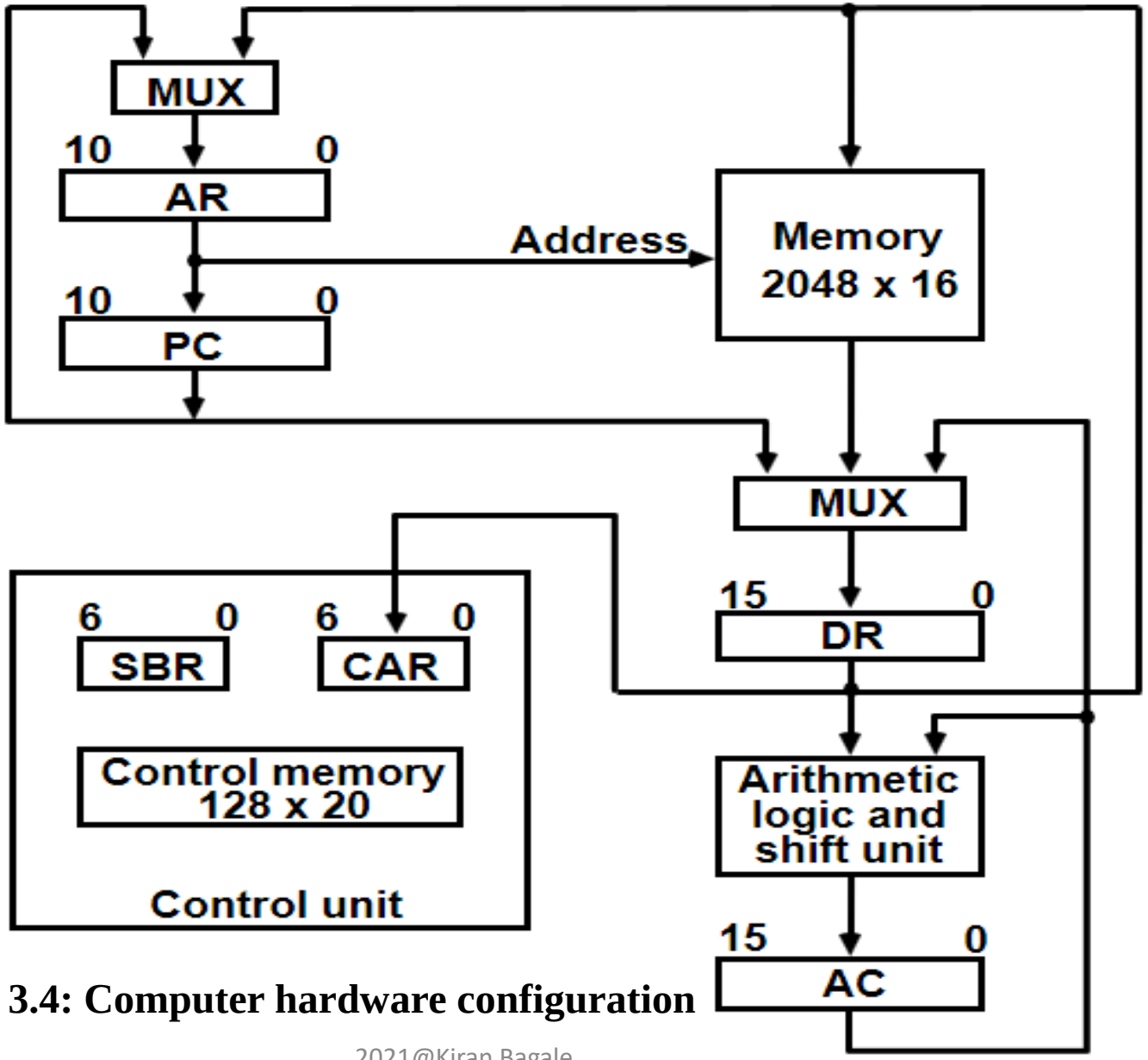


Fig 3.4: Computer hardware configuration

- The processor registers are:
  1. ***Program counter (PC),***
  2. ***Address register (AR),***
  3. ***Data register (DR), and***
  4. ***Accumulator register (AC).***
- The control unit has:
  1. ***Control address register (CAR) and***
  2. ***Subroutine register (SBR).***
- 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.
- Memory receives its address from AR.
- Input data written to memory come from DR, and data read from memory can go only to DR.
- The computer instruction format is depicted in Table1. It consists of three fields:
  a. ***A 1-bit field for indirect addressing symbolized by I,***
  b. ***A 4-bit operation code (opcode),***
  c. ***And an 11-bit address field.***

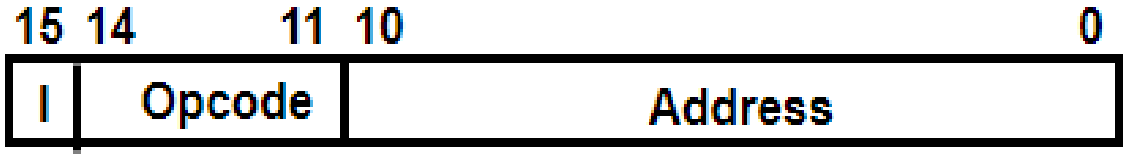


Fig: Machine instruction format

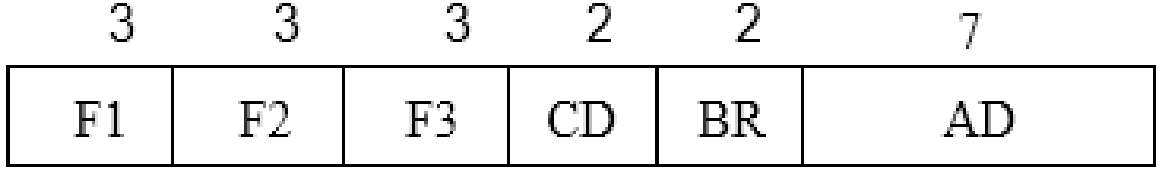
Symbol	OP-code	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC

EA is the effective address

Table1: Simple Instruction format

4. Microinstruction format

- The 20 bits of the microinstruction.
- Divided into four functional parts.
- The three fields F1, F2, and F3 specify microoperations for the computer.
- The CD field selects status bit conditions.
- The BR field specifies the type of 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.



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

Description of F1, F2, F3

Table2: Description of microoperation fields

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

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

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

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

5. Symbolic Microinstruction

- The Symbolic Microinstruction are the symbols used in microinstructions as in assembly language.
- A symbolic microprogram can be translated into its binary equivalent by means of an 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)}

The Fetch Routine

- 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 Fetch Routine (cont.)

- Sequence of microoperations in the fetch cycle:
  1. **AR ←PC**
  2. **DR ← M[AR], PC ← PC + 1**
  3. **AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ←0**
- The fetch routine needs three microinstructions, which are placed in control memory at addresses 64, 65, and 66.
- Symbolic microprogram for the fetch cycle:

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

The Fetch Routine (cont.)

- Binary equivalents translated by an assembler

Table3: Binary word generation

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

6. Symbolic microprogram

- The execution of the microinstruction in the fetch routine with branch results to address OxxxxOO, 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 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.

- In each routine we must provide microinstructions for evaluating the effective address and for executing the instruction.

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 Microinstruction

- The symbolic microprogram is a convenient form for writing microprograms in a way that people can read and understand.
- But this is not the way that the microprogram is stored in memory.
- The symbolic microprogram must be translated to binary either by means of an assembler program or by the user if the microprogram is simple enough as in this example.

Micro Routine	Address		Binary Microinstruction					
	Decimal	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

7. Control Unit Operation

- The main function of the control unit is to fetch and execute instructions from the memory of a computer.
- **Characterization of CU**
  1. Define the basic elements of the processor.
  2. Describe the micro-operations that the processor performs.
  3. Determine the functions that the control unit must perform to cause the micro-operations to be performed.
- **Types of Micro-operation**
  1. Transfer data between registers
  2. Transfer data from register to external interface
  3. Transfer data from external interface to register
  4. Perform arithmetic/logical ops with register for i/p, o/p

Functions of Control Unit

- Sequencing
- Causing the CPU to step through a series of micro-operations
- Execution
- Causing the performance of each micro-op

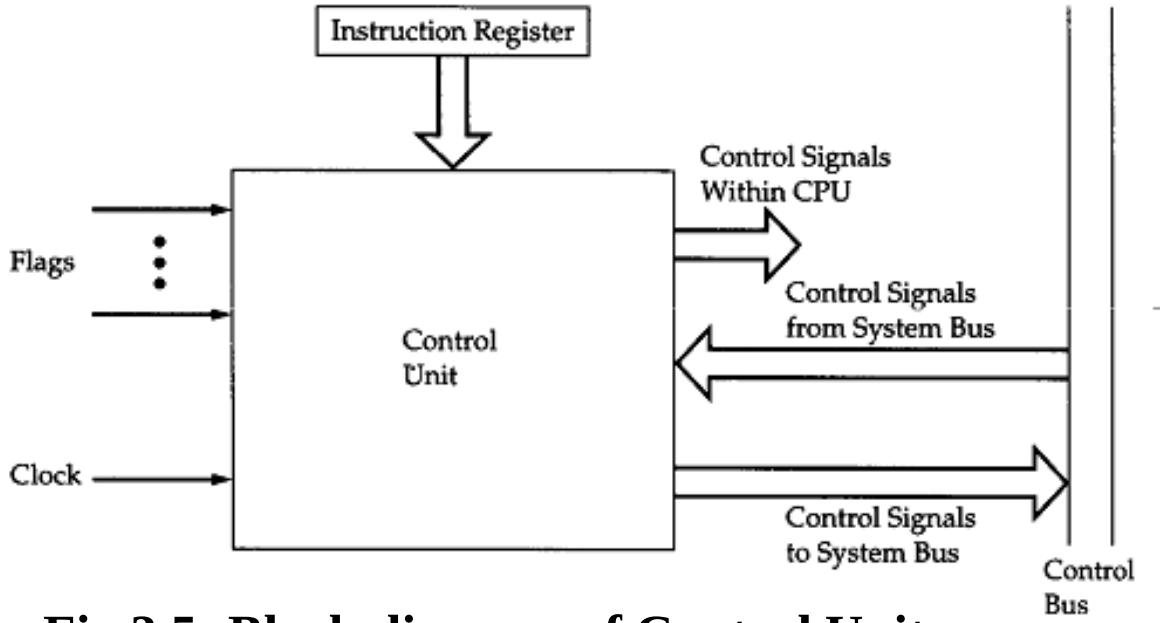


Fig 3.5: Block diagram of Control Unit

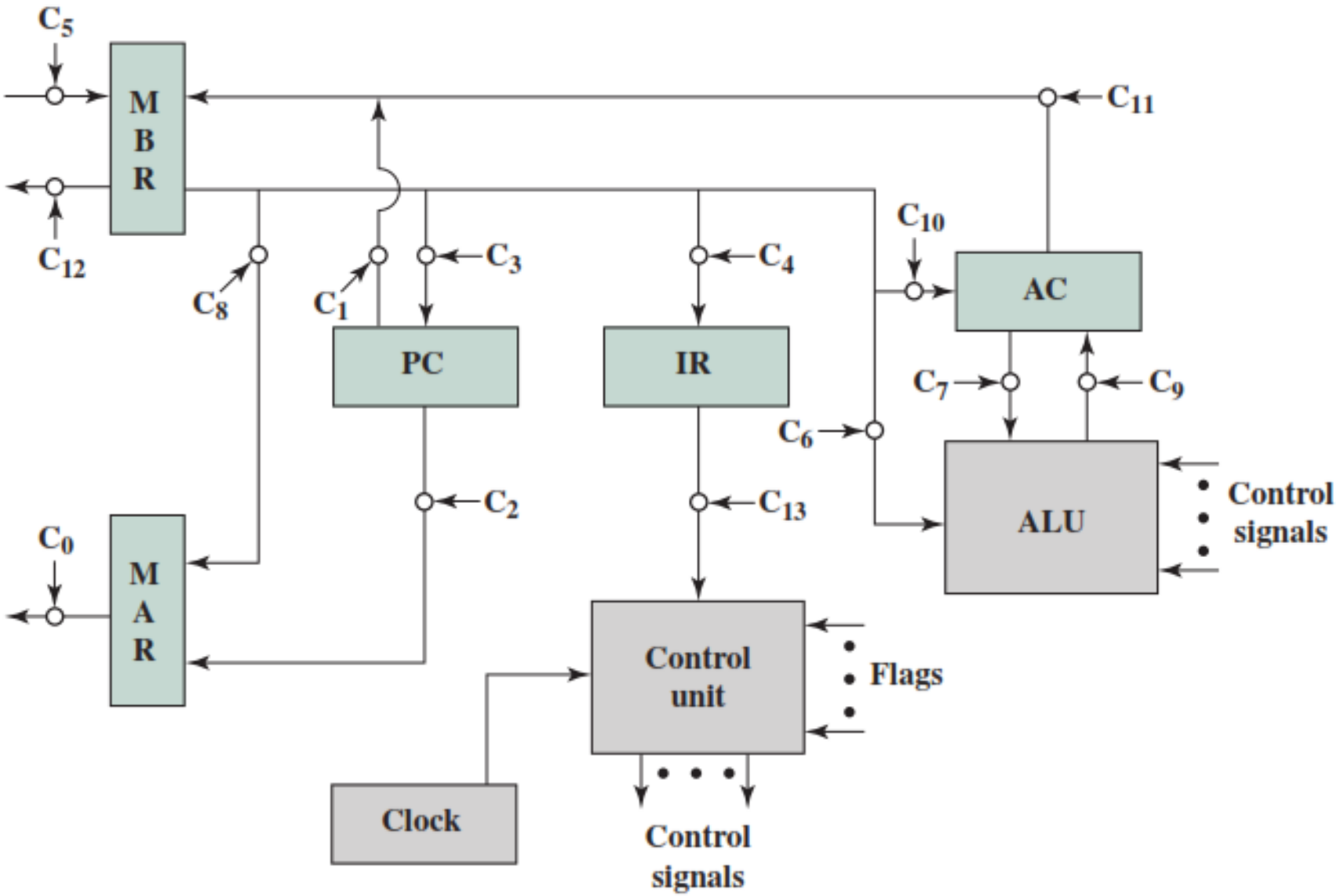


Fig3.6:Data Paths and Control Signals

Table 4: Micro-operations and Control Signals

	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R= Read control signal to system bus.

C_w= Write control signal to system bus .

Hardwired Implementation

- CU is essentially a combinational circuit.
- Its i/p signals are transformed into set of o/p logic signal which are control signals.
- Control unit inputs
- Flags and control bus
  - Each bit means something
- Instruction register
  - Op-code causes different control signals for each different instruction
  - Unique logic for each op-code
  - Decoder takes encoded input and produces single output
  - Each decoder i/p will activate a single unique o/p

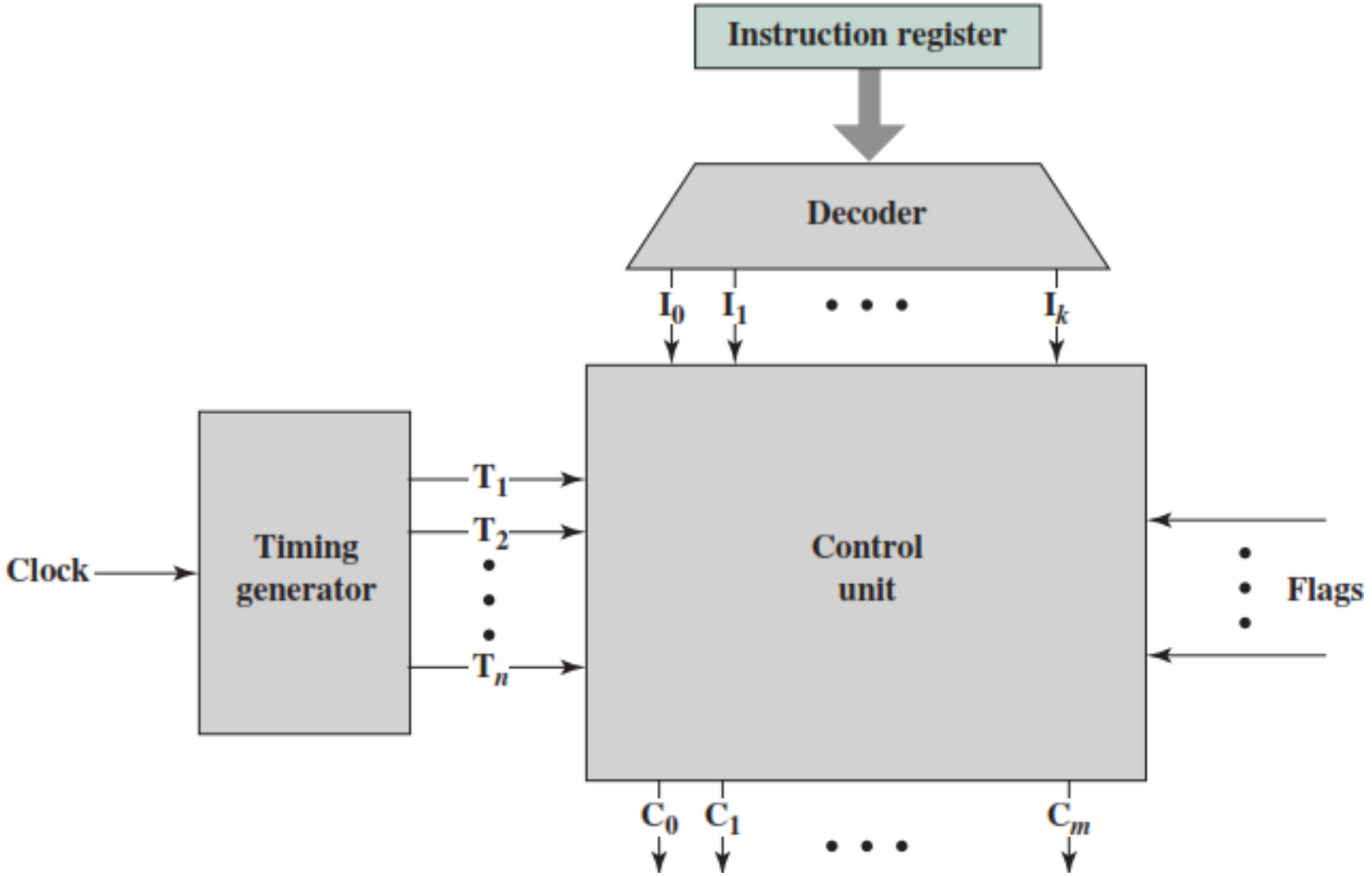


Fig3.7: Control Unit with Decoded Inputs

- Clock
  - ✓ Repetitive sequence of pulses
  - ✓ Useful for measuring duration of micro-ops
  - ✓ Must be long enough to allow signal propagation along data paths and through processor circuitry
  - ✓ Different control signals at different times within instruction cycle
  - ✓ Need a counter as i/p to control unit with different control signals being used for t1, t2 etc.
  - ✓ At end of instruction cycle, counter is re-initialised

Implementation

- For each control signal, a Boolean expression of that signal as a function of the inputs is derived
- With that the combinatorial circuit is realized as control unit.
- *Let us consider a control signal C5 (reads data from external data bus into the MBR).*

- *Used twice (fetch and indirect cycle) in table 4.*
- *Let P and Q be two new control signals having following interpretation:*

  PQ=00 Fetch cycle

  PQ=01 Indirect cycle

  PQ=10 Execute cycle

  PQ=11 Interrupt cycle
- Then the following Boolean expression defines C5:

  C5= (P'.Q'.T2) + (P'.Q.T2)

Problems With Hard Wired Designs

- Complex sequencing & micro-operation logic
- Difficult to design and test
- Inflexible design
- Difficult to add new instructions

Micro-programmed Implementation

- An alternative to hardwired CU
- Common in contemporary CISC processors
- Use sequences of instructions to perform control operations performed by micro-operations called micro-programming or firmware
- Microprogram is a mid-way between hardware and software.
- The control unit functions as follows to execute an instruction :

1. ***The sequencing logic unit issues a READ command to the control memory.***
2. ***The word whose address is specified in the control address register is read into the control buffer register.***

3. *The content of the control buffer register generates control signals and next-address information for the sequencing logic unit.*
4. *The sequencing logic unit loads a new address into the control address register based on the next-address information from the control buffer register and the ALU flags.*

All this happens during one clock pulse.

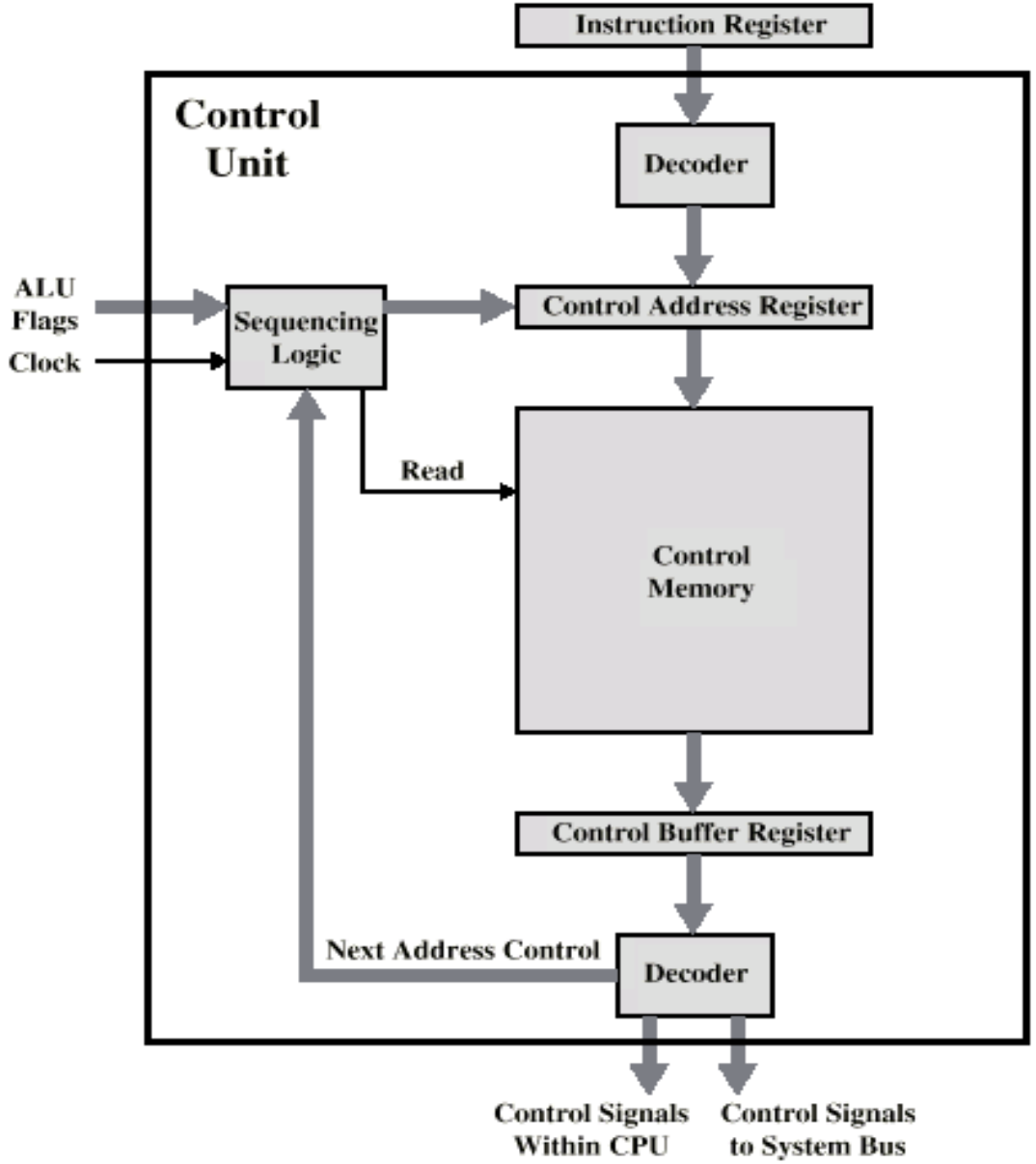


Fig 3.8: Functioning of Microprogrammed Control Unit

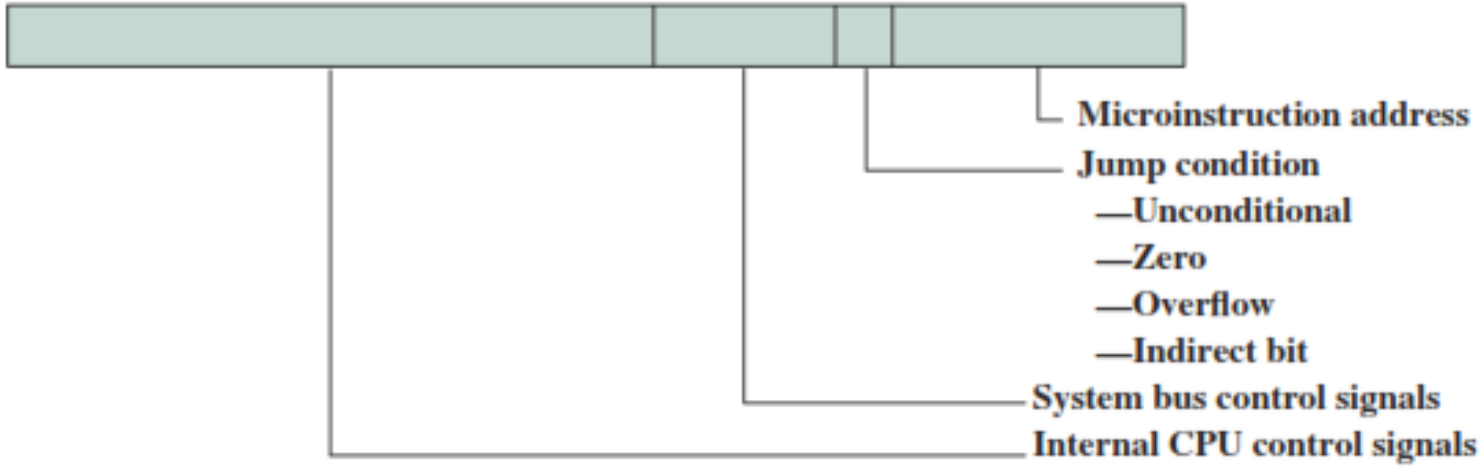
- The last step just listed needs elaboration.
  1. **Get the next instruction:** Add 1 to the control address register.
  2. **Jump to a new routine based on a jump microinstruction:** Load the address field of the control buffer register into the control address register.
  3. **Jump to a machine instruction routine:** Load the control address register based on the opcode in the IR.
- Upper decoder translates the opcode of IR into control memory address.
- Lower decoder used for vertical microinstructions.

Micro-instruction Types

- Each micro-instruction specifies single or few micro-operations to be performed -*vertical micro-programming.*
- Each micro-instruction specifies many different micro-operations to be performed inparallel - *horizontal micro-programming.*

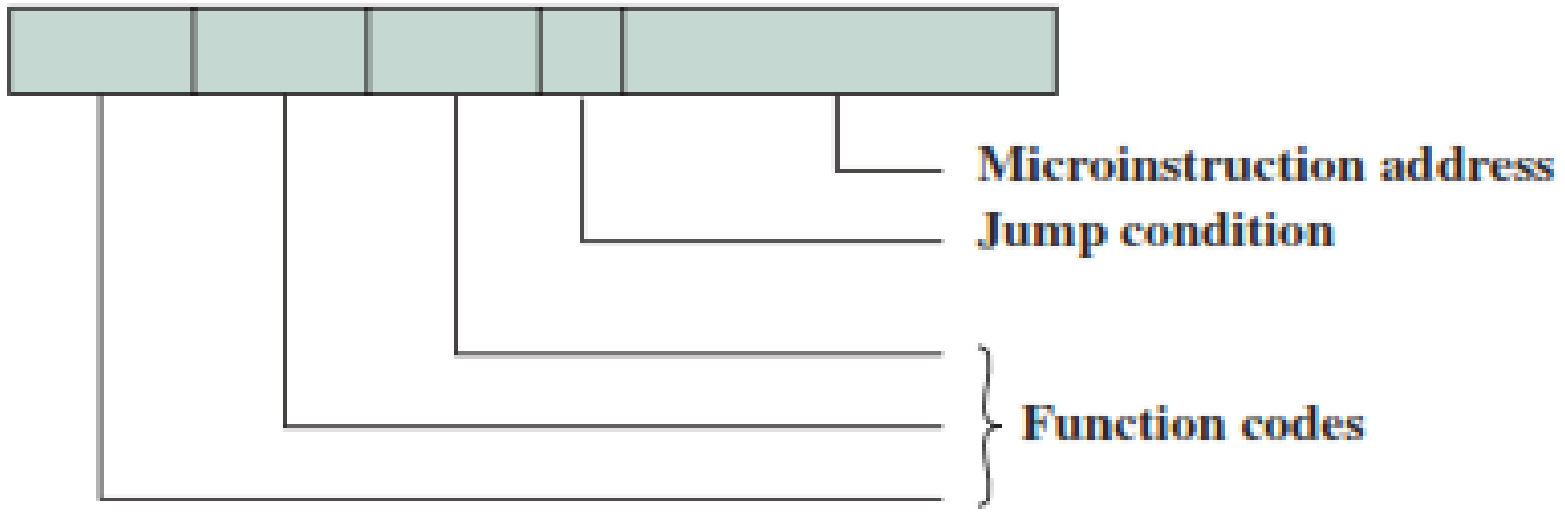
Horizontal Micro-programming

- Wide memory word
- High degree of parallel operations possible
- Little encoding of control information



Vertical Micro-programming

- Width is narrow
- n control signals encoded into $\log_2 n$ bits
- Limited ability to express parallelism
- Considerable encoding of control information requires external memory word decoder to identify the exact control line being manipulated



8. Design of Control Unit

- 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
  - ✓An address field for branching .
- The number of control bits that initiate microoperations can be reduced by grouping *mutually exclusive* variables into fields by encoding the k bits in each field to provide 2^k microoperations.

- Each field requires a decoder to produce the corresponding control signals.
  - Reduces the size of the microinstruction bits.
  - Requires additional hardware external to the control memory.
  - Increases the delay time of the control signals.

F Field Decoding

- Three microoperation fields need 3 decoders.
- 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.

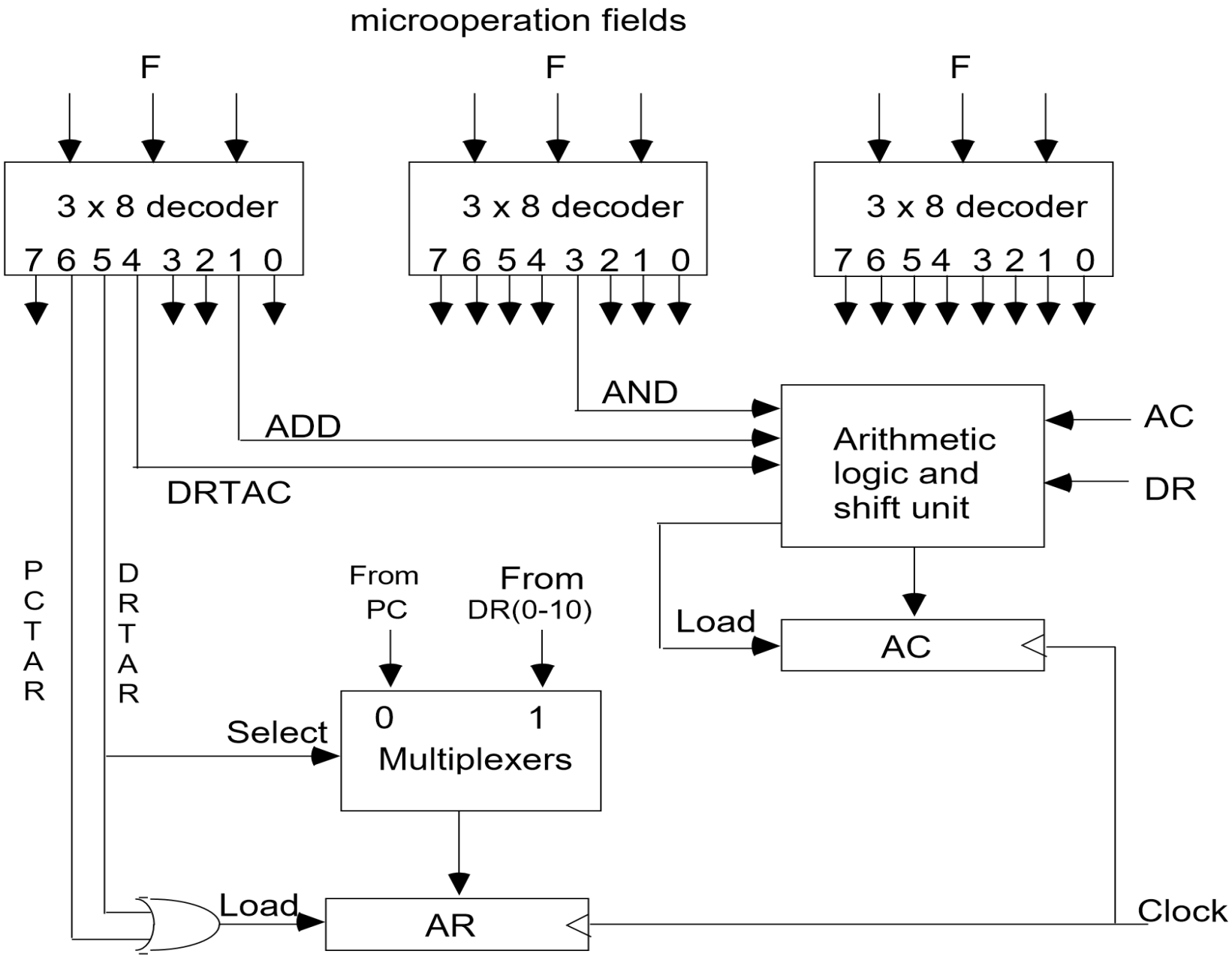
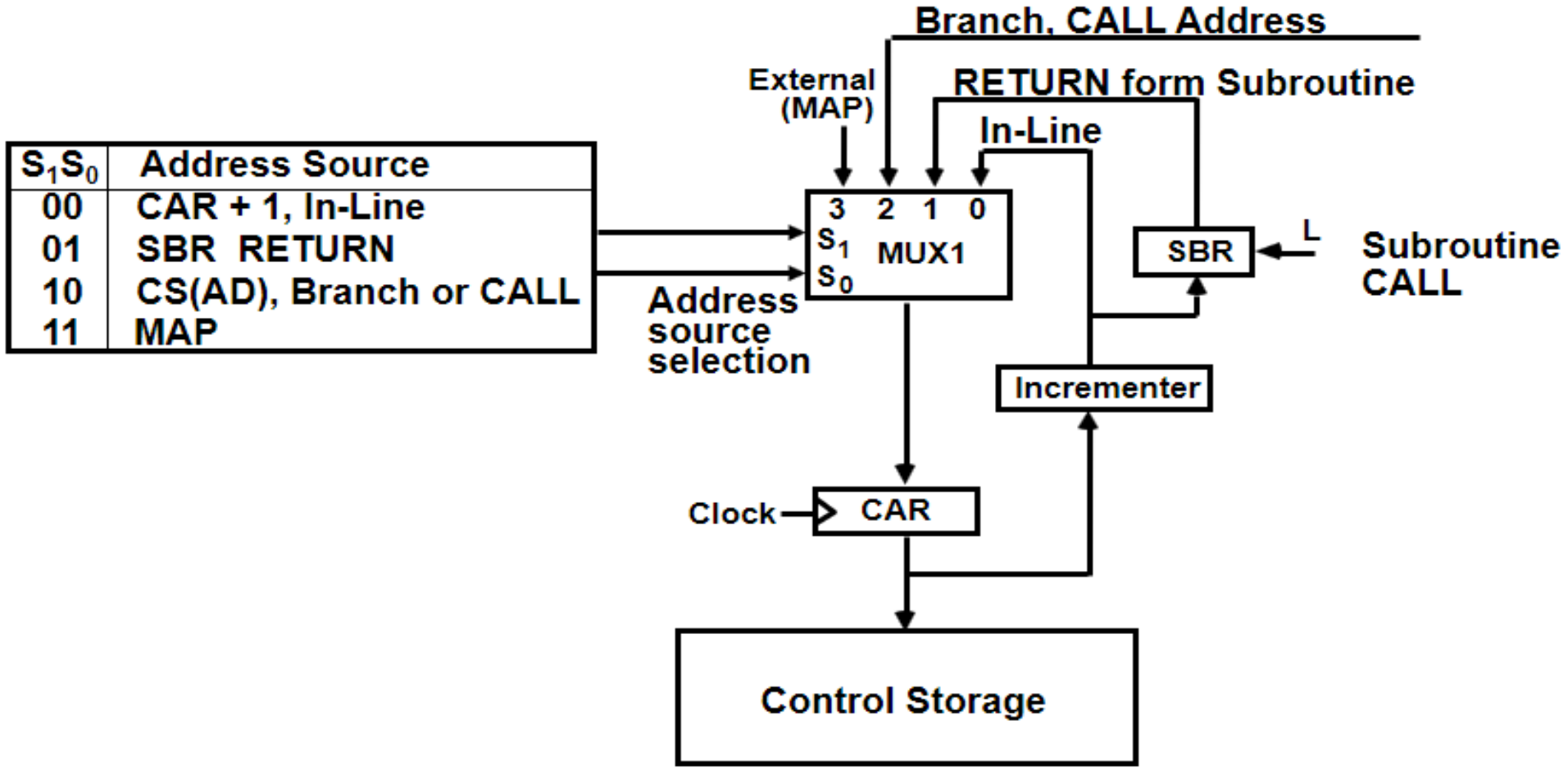


Fig 3.9: Decoding of microoperation fields

- For example, when F1 = 101 (binary 5), the next clock pulse transition transfers the content of DR (0-10) to AR (symbolized by DRTAR in Table2.
- Similarly, when F1 = 110 (binary 6) there is a transfer from PC to AR (symbolized by PCTAR).
- Outputs 5 and 6 of decoder F1 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.
- The other outputs of the decoders that are associated with an AC operation must also be connected to the arithmetic logic shift unit in a similar fashion.

Microprogram Sequencer



S_1S_0	Address Source
00	CAR + 1, In-Line
01	SBR RETURN
10	CS(AD), Branch or CALL
11	MAP

- 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.
- 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 application.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.

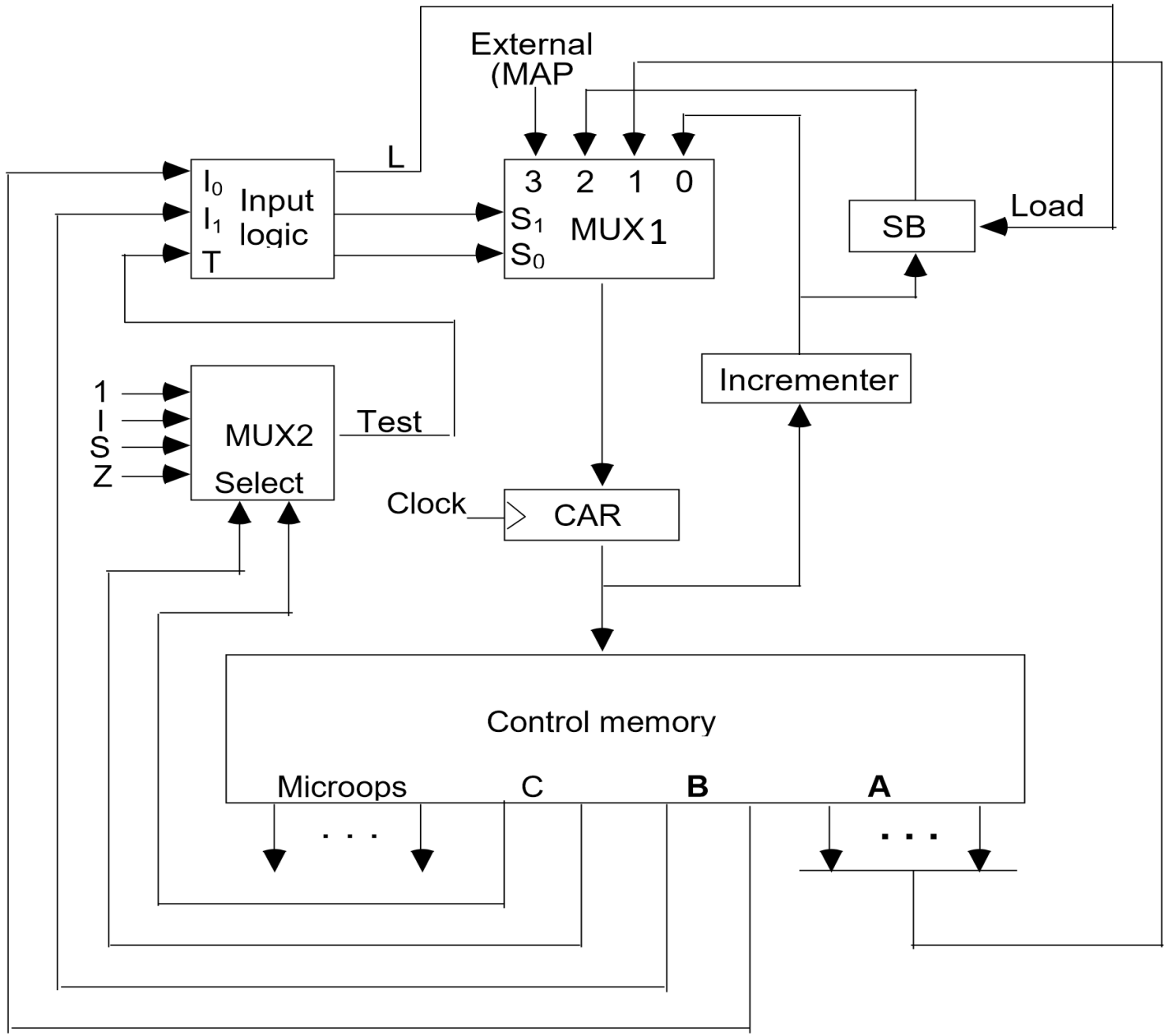


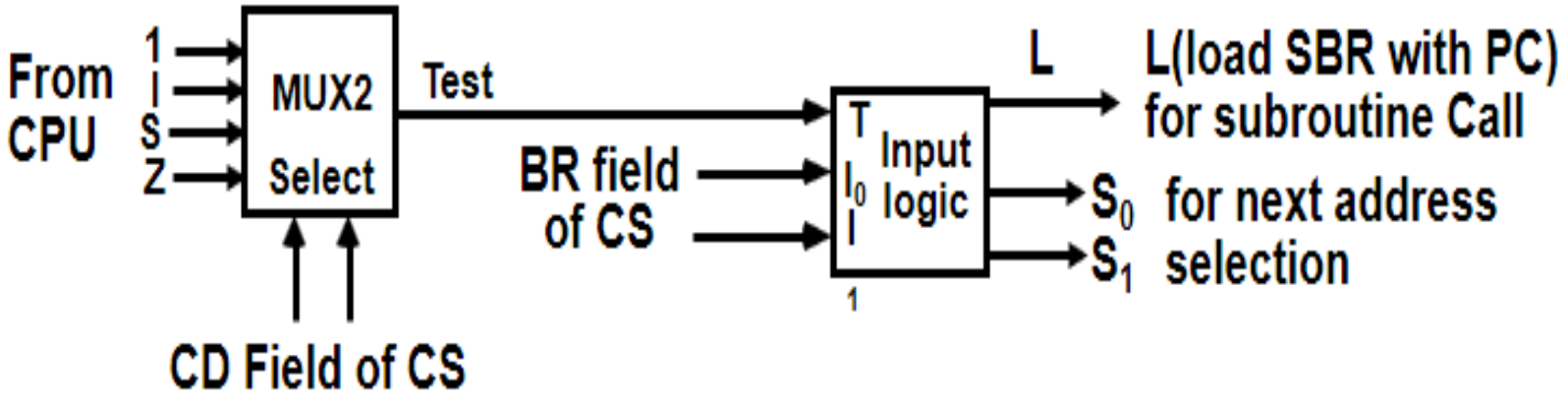
Fig 3.10: Micro program Sequencer

Components and Working:

- The ***control memory*** is included to show the *interaction between the sequencer* and *the different units attached to it*.
- ***Two multiplexers***:
  1. **MuX1**-selects an address from one of the four sources and routes to CAR,
     - *In-Line Sequencing* ← *CAR + 1*
     - *Branch, Subroutine Call* ← *Take address from AD field*
     - *Return from Subroutine* ← *Output of SBR*
     - *New Machine instruction* ← *MAP*

  2. **Mux2**-tests the value of the selected status bit and result is applied to an input logic circuit.

- MUX-2 Controls the condition and branching as below:



Design of Input Logic:

- 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*.

- Based on the function listed in each entry, the truth table for the input logic circuit is shown in Table below.

Table5: Input logic truth table for microprogram sequencer

I_1I_0T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	01	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	01	1
10x	RET	SBR	10	0
11x	MAP	DR(11-14)	11	0

$$S_1 = I_1$$
$$S_0 = I_1I_0 + I_1'T$$
$$L = I_1'I_0T$$

- The bit values for S1 and S0 are determined from the stated function and the path in the multiplexer that establishes the required transfer.
- Note that the incrementer circuit in the sequencer of Fig.4.10 is not a counter constructed with flip-flops but rather a combinational circuit constructed with gates.