Microprogrammed Control

Compiled by :- Er. Nagendra Karn

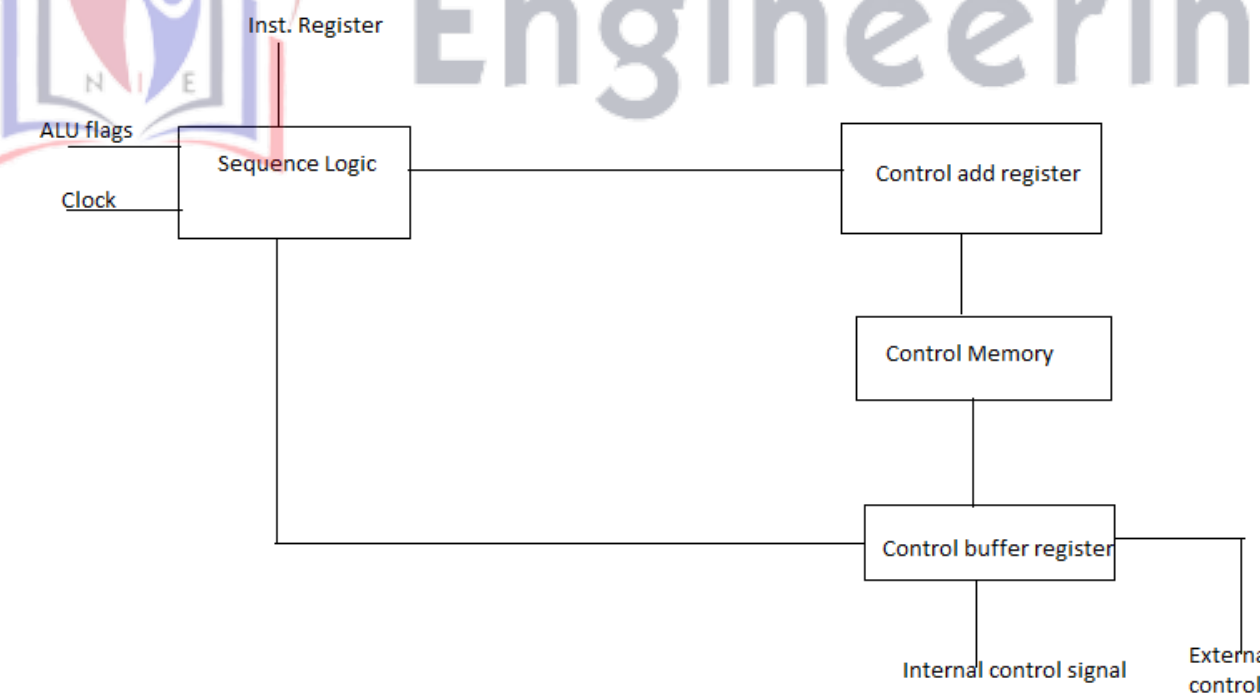
Microprogrammed Control unit :-

- It uses sequence of microinstructions in microprograming language.
- It acts as a mid way between hardware and software.
- It generates a set of control signals.
- It is easy to design, test and implement and it is flexible to modify.
- Microprogrammed control unit consists of set of **control signals**, **control variables**, **control word**, **control memory**, **micro-instructions** and **microprogram**.

Terminologies :-

- **Hardwired Control unit** :- when the control signals are generated by hardware using conventional logic design techniques, control unit is said to be hardwired.
- **Microprogramed control unit** :- A control unit whose binary control variables are stored in a memory is called a microprogrammed control unit.
- **Control memory** :- Control memory is the storage in a microprogrammed control unit to store the microprogram.
- **Control word** :- the control variables at any given time can be represented by a control word string of 1's and 0's called a control word.
- **Micro operation** :- it performs basic operations on data stored in one or more registers, including transferring data between register or between registers and external buses of cpu, and performing logical or arithmetic operation on registers.

- **Microcode** :- a very low level instruction set which is stored permanently in a computer or peripheral controller and controls the operation of the device.
- **Microinstruction** :- a single instruction in microcode. It is the most elementary instruction in the computer, such as moving the contents of a register to the arithmetic logic unit.
- **Microprogram** :- a set or sequence of microinstructions.



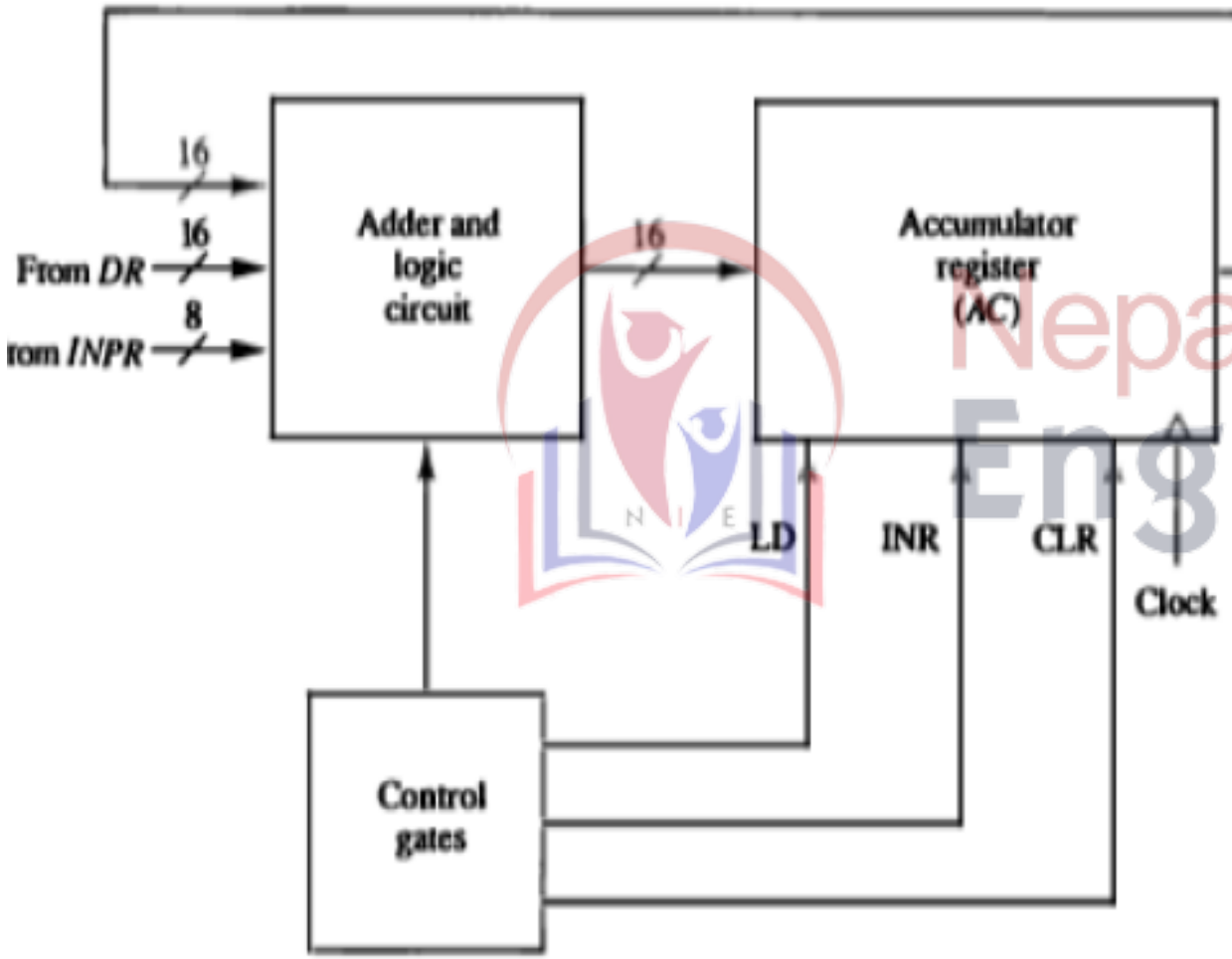
Design of basic computer :-

- The basic computer consists of following hardware components :
  - A memory unit with 4096 words of 16 bits each.
  - Nine registers :- AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
  - Flip-flops : IEN (interrupt enable), I(indirect flip-flop), S(start flip-flop), E(extended flip-flop) ,FGI, and FGO
  - Two decoders : a 3*8 operation decoder and 4*16 timing decoder.
  - A 16-bit common bus.
  - Control logic gates
  - Adder and logic circuit connected to input of AC.

Design of Accumulator Logic

- The adder and logic circuit has three sets of inputs.
- One set of 16 input comes from the outputs of AC.
- Another set of 16 inputs comes from the output of data register (DR).
- A third set of eight inputs comes from the input register INPR.
- The outputs of the adder and logic circuit provide the data inputs for the register.
- In addition, it is necessary to include logic gate for controlling the LD, INR and CLR in the register and for controlling the operation of the adder and logic circuit.

Circuit Associated with 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:	$AC \leftarrow AC'$	Complement
rB_7 :	$AC \leftarrow shr\ AC, AC(15) \leftarrow E$	Shift right
rB_6 :	$AC \leftarrow shl\ AC, AC(0) \leftarrow E$	Shift left
rB_{11} :	$AC \leftarrow 0$	Clear
rB_5 :	$AC \leftarrow AC + 1$	Increment

Control of AC register- Gate Structure

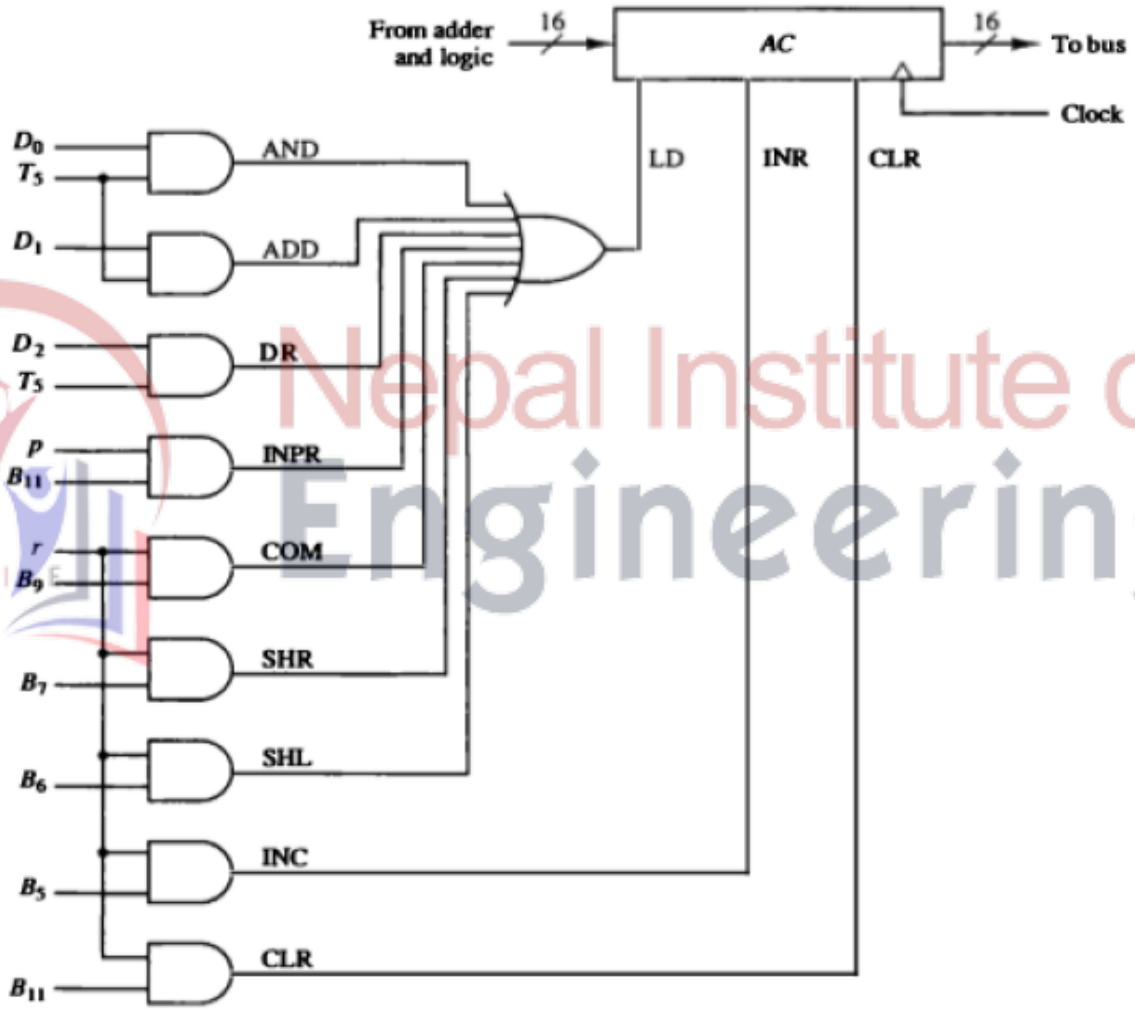


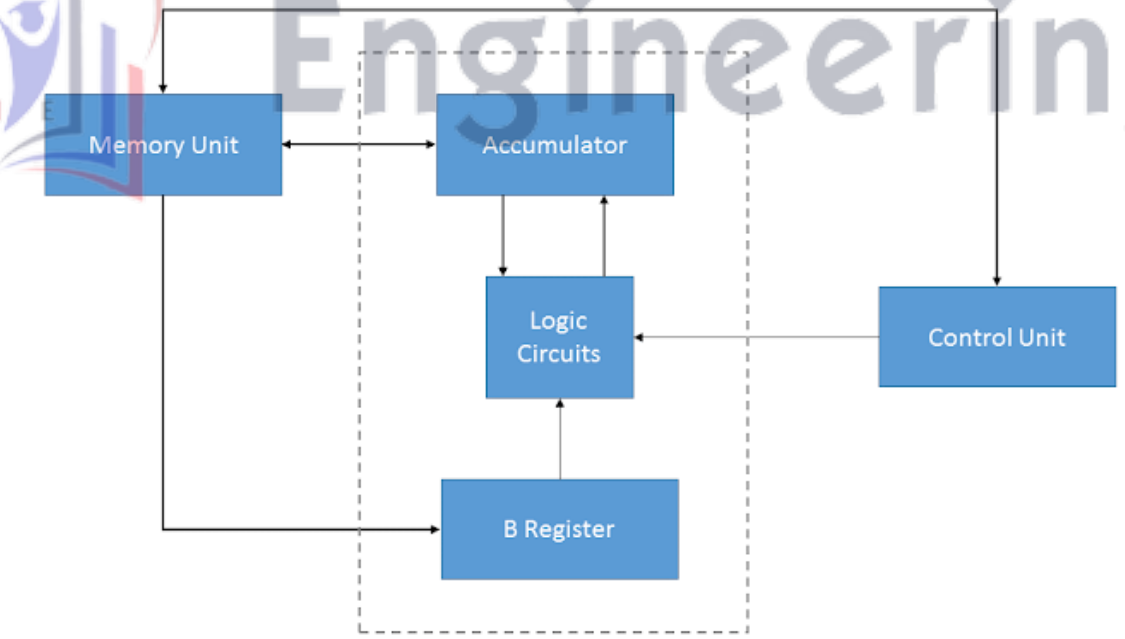
Fig :- Gate structure for controlling the LD, INR, and CLR of AC

Control of AC register-

- The gate structure that controls the LD, INR and CLR inputs of AC is shown in fig.
- The output of the AND gate that generates this control function is connected to the CLR input of the register.
- Similarly, the output of the gate that implements the increment micro operation is connected to the INR input of the register.
- The another seven micro operations are generated in the adder and logic circuit and are loaded into AC at the proper time.
- The outputs of the gates for each control function is marked with symbolic name. These outputs are used in the design of the adder and logic circuit.

ALU organization

- Various circuits are required to process data or perform arithmetical operations which are connected to microprocessor's ALU.
- Accumulator and Data Buffer stores data temporarily. These data are processed as per control instructions to solve problems. Such problems are addition, multiplication etc.



Organization of Arithmetic and Logic Unit

Function of ALU: functions of ALU can be categorized into following 3 categories.

- **Arithmetic operations** :- Additions, multiplications etc. are example of arithmetic operations. Finding greater than or smaller than or equality between two numbers by using subtraction is also a form of arithmetic operations.
- **Logical operations** :- Operations like AND, OR, NOR, NOT etc. using logical circuitry are examples of logical operations.
- **Data manipulations** :- Operations such as flushing a register is an example of data manipulation. Shifting binary numbers are also example of data manipulation.

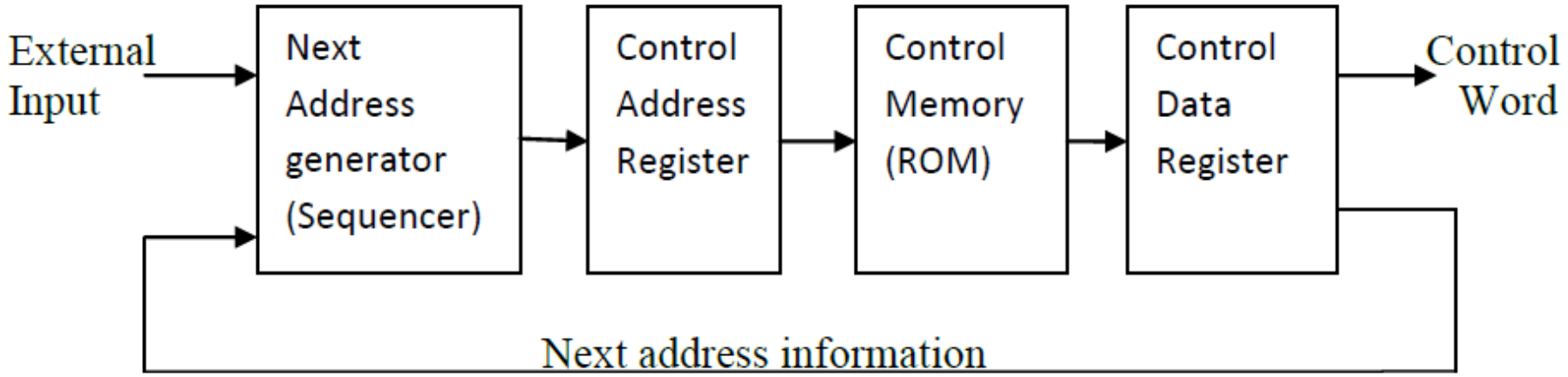
Control Memory

- 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. The microprogram consists of microinstructions that specify various internal control signals for execution of register micro operations.
- 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.

- The control unit initiates a series of sequential steps of micro operations. During any given time, certain micro operations 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.
- The **microinstruction** specifies one or more **microoperations** for the system. A sequence of microinstructions constitutes a **microprogram**.

Microprogrammed control organization

- The general configuration of microprogrammed control unit is demonstrated in the block diagram.
- 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 microinstructions.
- The control data register holds the microinstructions read from memory.

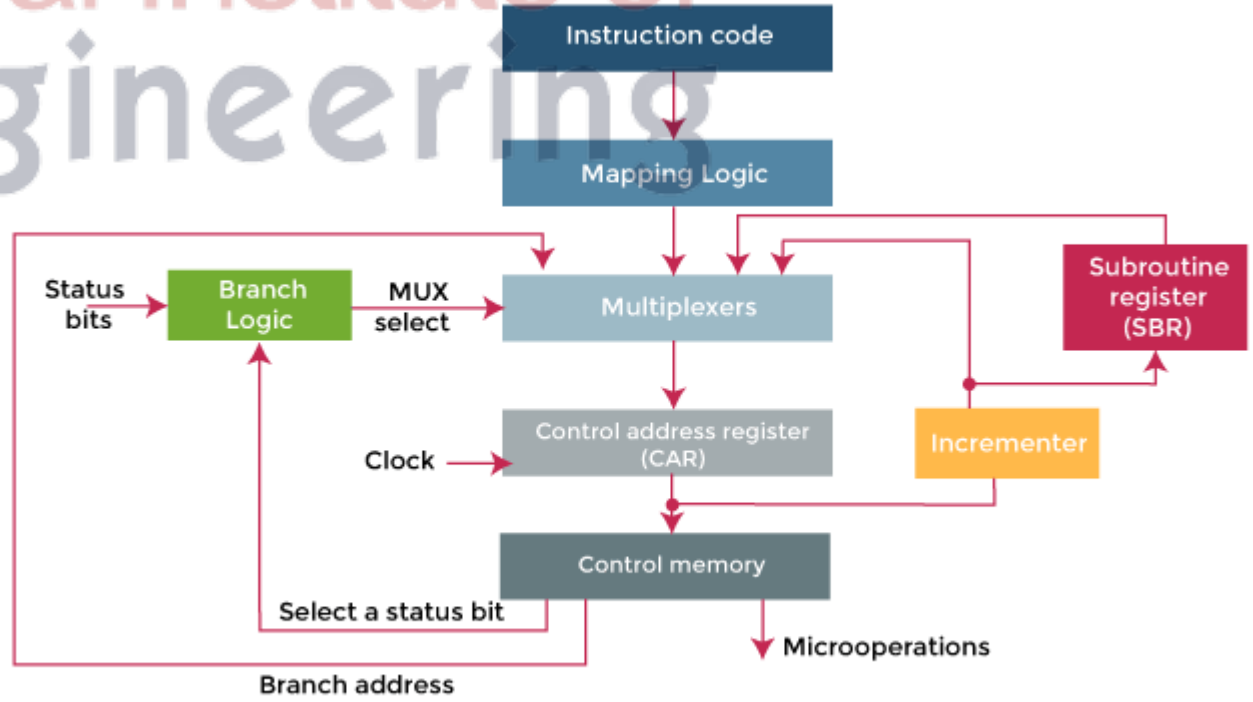


Address sequencing

- 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.
- To appreciate the address sequencing in a microprogram control unit, let us enumerate the steps that the control must undergo during the execution of a single computer instruction.
- 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. At the end of the fetch routine, the instruction is in the instruction register of the computer.
- The control memory next must go through the routine that determines the effective address of the operand. When the effective address computation routine is completed, 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 on the operation code part of the instruction.
- 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.
- In summary, the address sequencing capabilities required in a control memory are:
- Incrementing of the control address register.
- Unconditional branch or conditional branch, depending on status bit conditions.
- A mapping process from the bits of the instruction to an address for control memory.
- A facility for subroutine call and return.
- The diagram shows the block diagram of a control memory and its associated hardware to support in choosing the next microinstruction. The microinstruction present in the control memory has a set of bits that facilitate to start off the micro-operations in registers.

- There are four different directions are showed in the figure from where the control address register recovers its address. The CAR is incremented by the incrementer and selects the next instruction. In multiple fields of microinstruction, the branching address can be determined to result in branching.
- It can specify the condition of the status bits of microinstruction, conditional branching can be applied. A mapping logic circuit can share an external address. A special register can save the return address so that when the microprogram needs to return from the subroutine, it can need the value from the unique register.



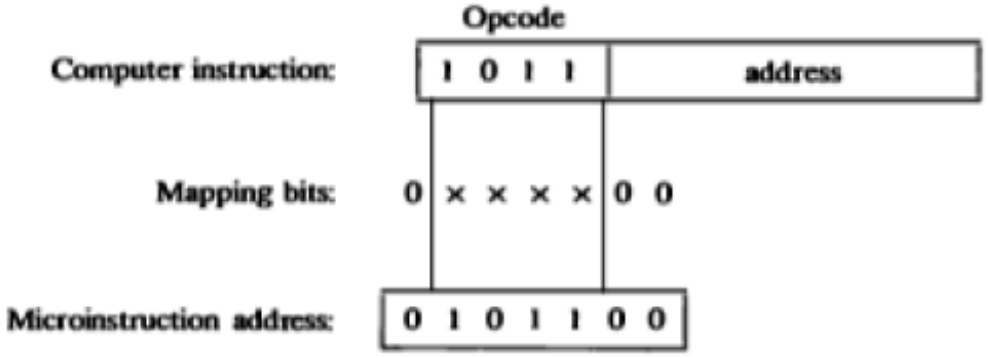
Selection of Address for Control Memory

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.
- 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.

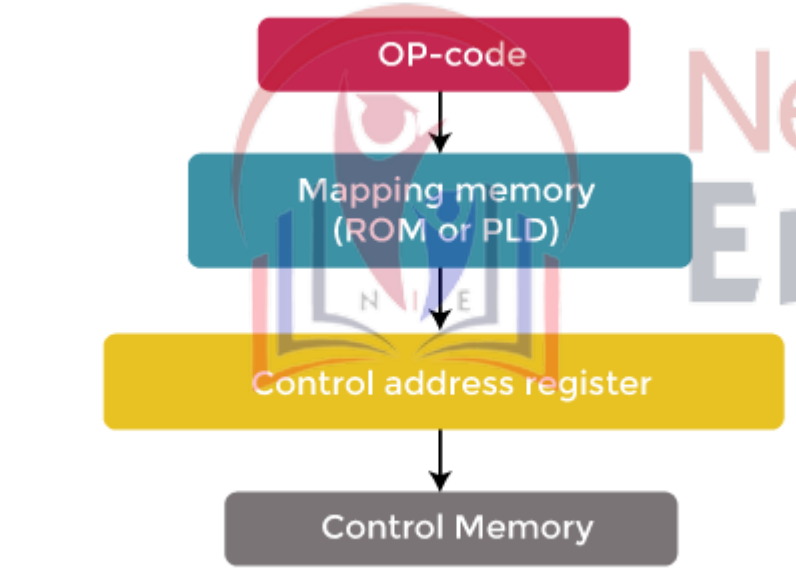
Mapping of Instruction :-

- 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**.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.
- For example, a computer with a simple instruction format as shown in Fig has an operation code of four bits. 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.

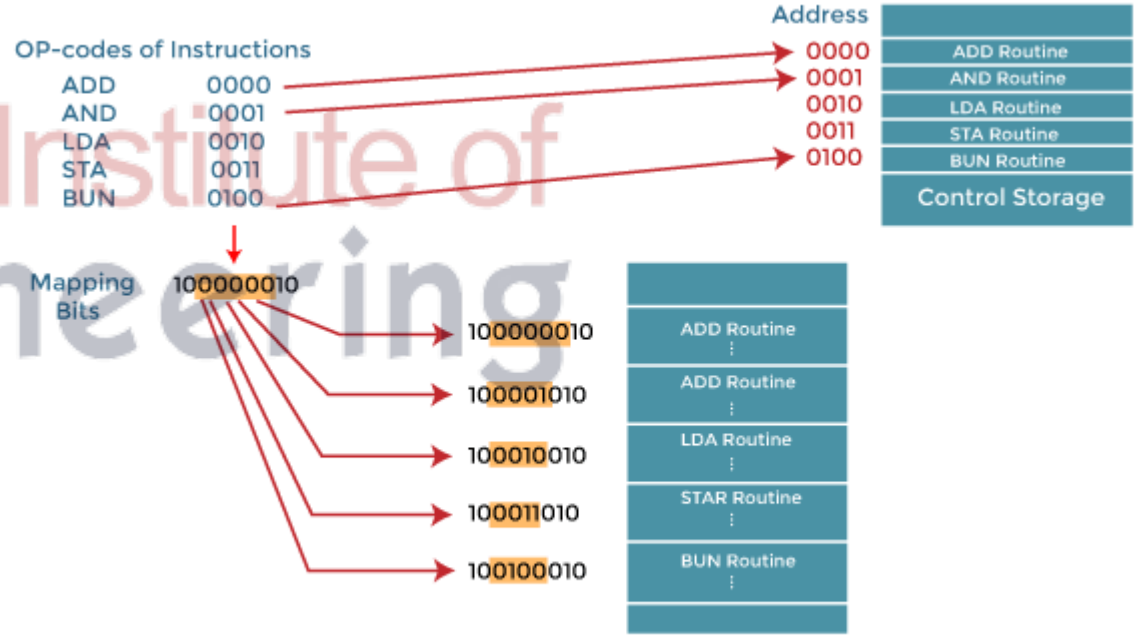


- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register. 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.

- The below image shows the mapping of address of microinstruction from the opcode of an instruction. In the execution program, this microinstruction is the starting microinstruction.



Mapping Function Implemented by ROM and PLD



Direct Mapping

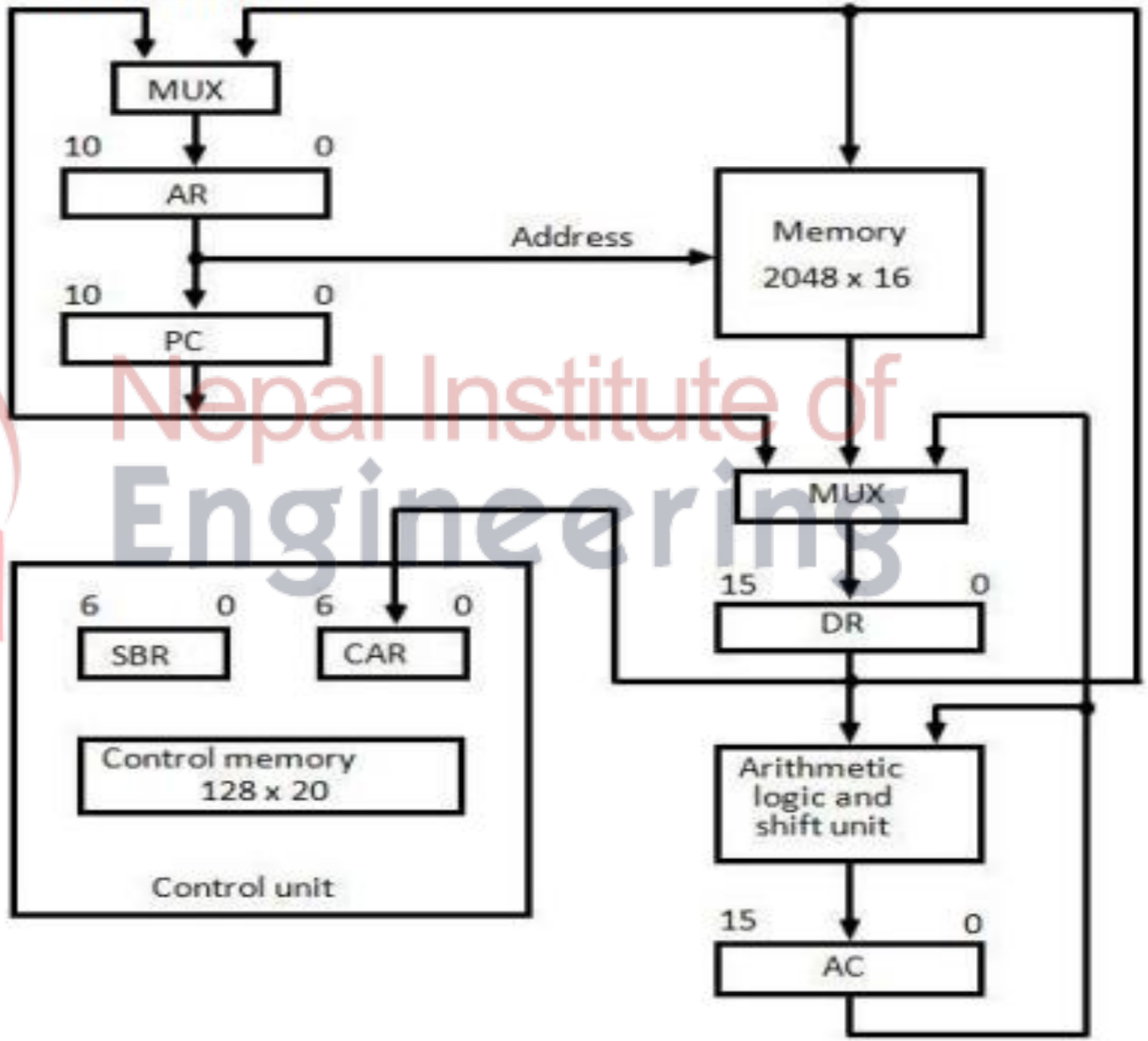
Subroutine :-

- Subroutines are programs that are used by other routines to accomplish a particular task.
- Microinstructions can be saved by employing subroutines that use common section of microcode. e.g. effective address computation.
- 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 subroutine is to organize the registers in last-in, first-out (LIFO) stack.

Computer Configuration :-

- The block diagram of the computer is shown below. It has two memory unit.
  - Main memory for storing instructions and data.
  - Control memory for storing the microprogram.
- Four registers are associated with the processor unit :- program counter (PC), address register(AR), data register (DR), accumulator register (AC).
- The control unit has a control address register CAR, and a subroutine register SBR.
- The control memory and its register are organized as microprogrammed control unit, as shown in figure.
- The transfer of information among the registers in the processor is done through multiplexer rather than a common bus.

Computer Hardware Configuration

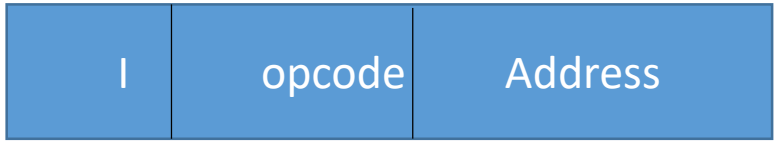


Microprogram:-

- **Microprogram** is a sequence of microinstructions that controls the operation of an arithmetic and logic unit so that machine code instructions are executed.
- It is a microinstruction program that controls the functions of a central processing unit or peripheral controller of a computer.

Computer Instruction Format:-

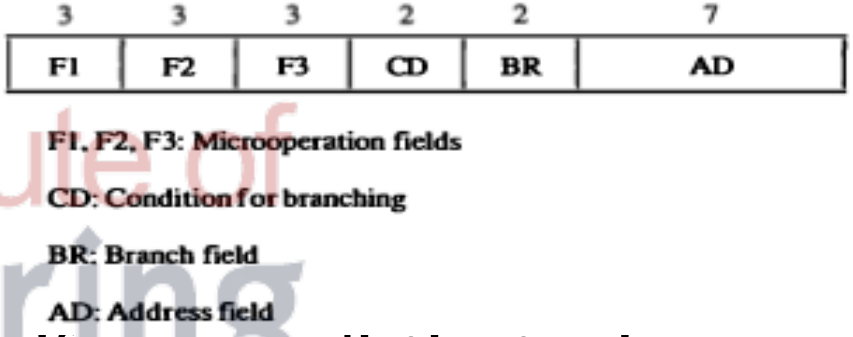
- It consists of three fields :-
  - A 1-bit filed for indirect addressing symbolized by I
  - 4 bit operation code (opcode).
  - An 11-bit address field .



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

Microinstruction Format :-

- The 20 bits of the microinstruction are 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.
  - The AD field contains branch address.



- The three bits in each field are encoded to specify seven distinct micro operations as listed in table.
- No more than three micro operations can be chosen for a microinstructions, one from each fields.
- If fewer than three micro instructions are used, one or more of the fields will use binary code 000 for no operation.

- It is important to realize that two or more conflicting microoperations cannot be specified simultaneously. e.g. 010 001 000
- Each microoperation in the table is defined with register transfer statement and is assigned a symbol for use in a symbolic micro program.

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 v 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	SHR
100	AC←shr AC	SHR
101	PC←PC+1	INCPC
110	PC←AR	ARTPC
111	RESERVED	

Condition Field

- A condition field includes 2 bit. They are encoded to define four status bit conditions. As stated in table, the first condition is always 1, with CD=0. the symbol that can indicate this condition is U. the table displays the multiple condition fields and their summary in an easy manner.
- **Condition field symbols and description:-**

	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

Branch Field

- The BR (Branch field) includes 2 bits. It can be used by connecting with the AD (Address) field. The reason for connecting with the AD field is to select the address for the next microinstruction. The table illustrates the various branch fields and their functions.
- **Branch field symbols and descriptions :-**

BR	Symbol	Comments
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

Symbolic Microinstruction :-

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

Sample format :-

Five fields :- label; micro-ops; CD; BR; AD

Label : may be empty or may specific 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}

Symbolic microprogram :- 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 equivalents 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

Symbolic Microprogram :-

- 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	Micro ops	CD	BR	AD
ADD:	ORG 0			
	NOP	I	CALL	INDIRECT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
BRANCH	ORG 4			
	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDIRECT
	ARTPC	U	JMP	FETCH
STORE:	ORG 8			
	NOP	I	CALL	INDIRECT
	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	

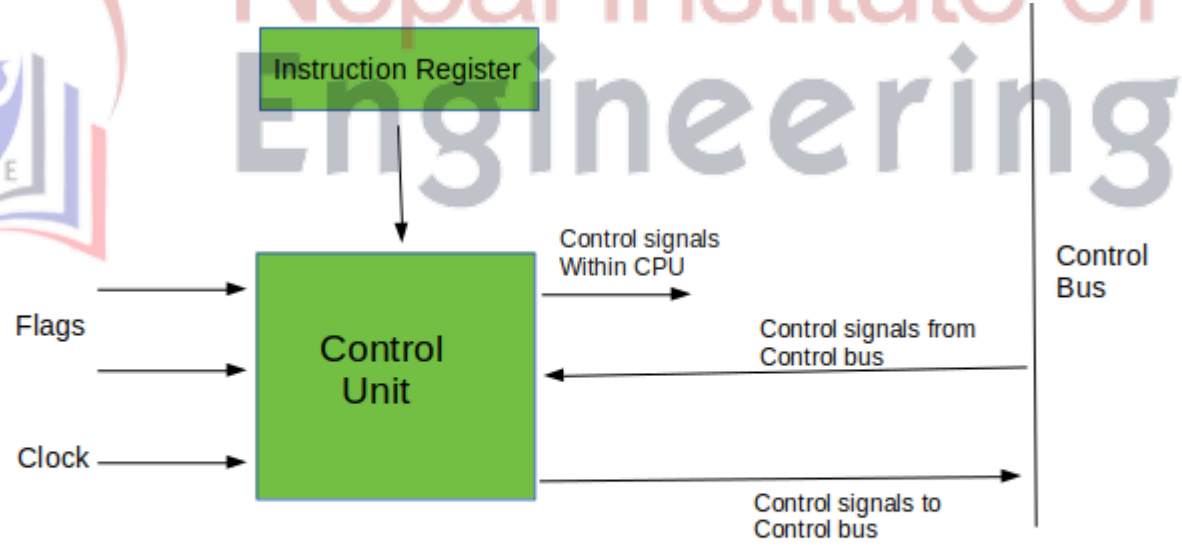
Micro Routine	Address		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	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

Symbolic vs Binary Microprogram:-

Symbolic Microprogram	Binary Microprogram
It is a set of microinstructions written in a symbolic form.	It is a set of microinstructions written in a binary form.
It is a convenient form for writing microprograms in a way that people can read and understand.	It is written in binary form so can be difficult for people to read and understand.
To be stored in memory symbolic microprogram must be translated to binary either by means of an assembler program or by the user if the microprogram is simple enough.	Translation is not required for storing in memory because it is already written in binary form.
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.
Example: Symbolic microprogram for fetch routine: FETCH: ORG 64 PCTAR U JMP NEXT READ, INCPC U JMP NEXT DRTAR U MAP	Example: Translation of symbolic microprogram to 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

Design of Control Unit/ Structure of CU :-

- Control unit generates timing and control signals for the operations of the computer. The control unit communicates with ALU and main memory. It also controls the transmission between processor, memory and the various peripherals. It also instructs the ALU which operation has to be performed on data.



Block Diagram of the Control Unit

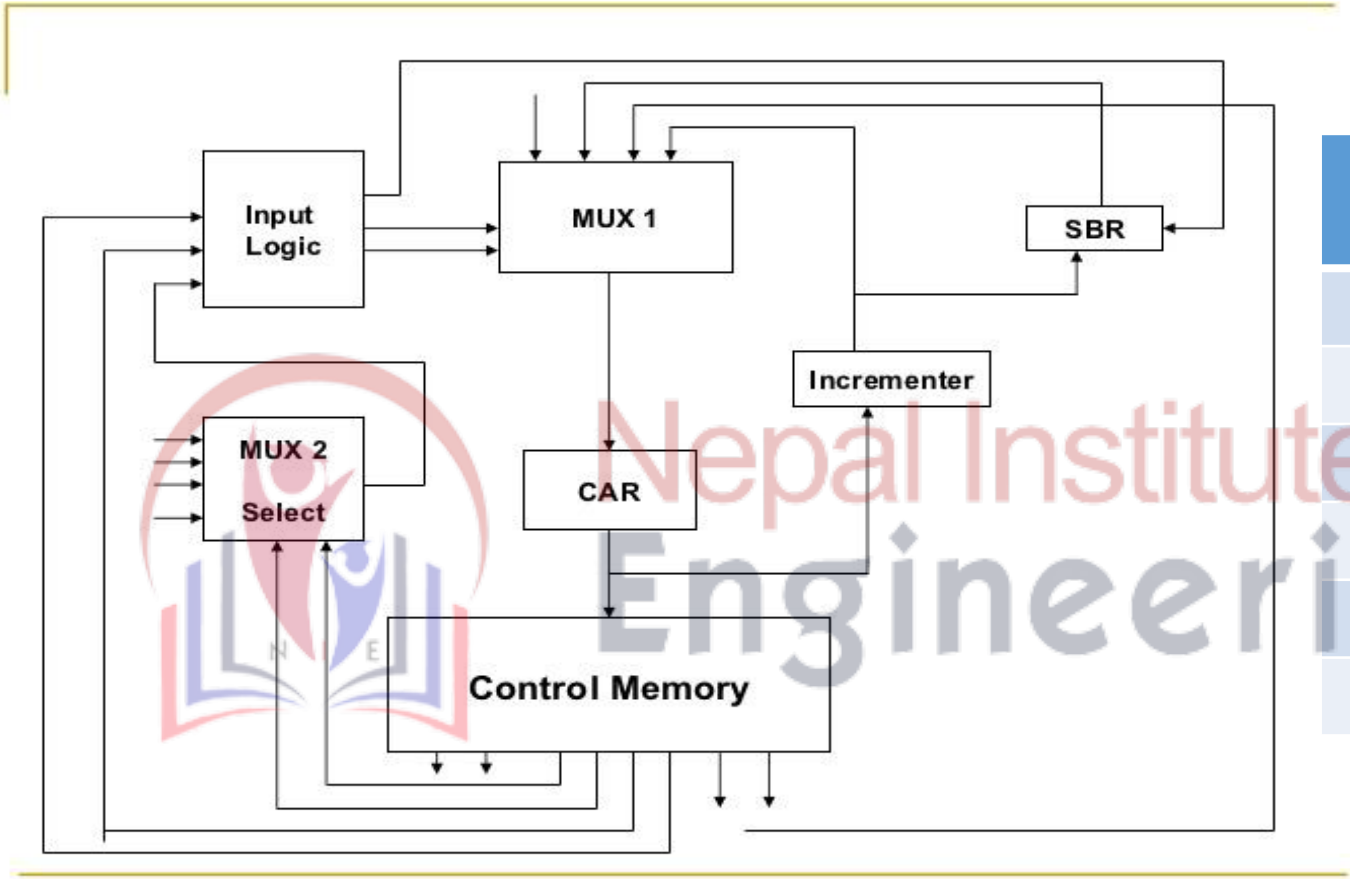
- A three step process that lead to the characterization of the Control Unit:
  - Define the basis elements of the processor.
  - Describe the micro-operations that the processor performs.
  - Determine the functions that the control unit must perform to cause the micro-operations to be performed.
- **Types of a Micro-operations :-** These operations consist of a sequence of micro operations. All micro instructions fall into one of the following categories:
  - Transfer data between registers
  - Transfer data from register to external
  - Transfer data from external to register
  - Perform arithmetic or logical operations

Function of Control unit :-

- The control unit perform two tasks:
- **Sequencing**: The control unit causes the CPU to step through a series of micro-operations in proper sequence based on the program being executed.
- **Execution**: The control unit causes each micro-operation to be performed.

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



Microprogram sequencer for a control Mmemory

BR field	Input I_1I_0T	MUX 1 $S1S_0$	Load SBR L
00	000	00	0
00	001	01	0
01	010	00	0
01	011	01	1
10	10X	10	0
11	11X	11	0

- 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 the four sources and routes it into the **CAR(Control Address Register)**.
- 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 contents of **CAR** is incremented and applied to one of the multiplexer inputs and to the **SBR**.
- The other three input come from the address field of the present microinstruction, from the output of **SBR(Subroutine Register)** and from an external source that maps the instruction.