



UNIT IV – MICROPROGRAMMED CONTROL – 10 HRS

Microprogrammed vs Hardwired Control

A hardwired control differs from microprogrammed control in the following ways:

HARDWIRED CONTROL UNIT	MICROPROGRAMMED CONTROL UNIT
The control unit whose control signals are generated by the hardware through a sequence of instructions is called a hardwired control unit.	The control unit whose control signals are generated by the data stored in control memory and constitute a program on the small scale is called a microprogrammed control unit
The control logic of a hardwired control is implemented with gates, flip flops, decoders etc.	The control logic of a micro-programmed control is the instructions that are stored in control memory to initiate the required sequence of microoperations.
Wiring changes are made in the hardwired control unit if there are any changes required in the design.	Changes in a microprogrammed control unit are done by updating the microprogram in control memory.
Hardwired control unit are faster and known to have complex structure.	Microprogrammed control unit is comparatively slow compared but are simple in structure.

Terminologies

Hardwired Control Unit:

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.

Micro programmed control unit:

A control unit whose binary control variables are stored in memory is called a micro programmed control unit.

Control Memory:

Control Memory is the storage in the 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.

Terminologies

Microoperations:

Micro-operations perform basic operations on data stored in one or more registers, including transferring data between registers or between registers and external buses of the central processing unit (CPU), and performing arithmetic or logical operations 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 (ALU).

Microprogram

A set or sequence of microinstructions.

Design of Basic Computer

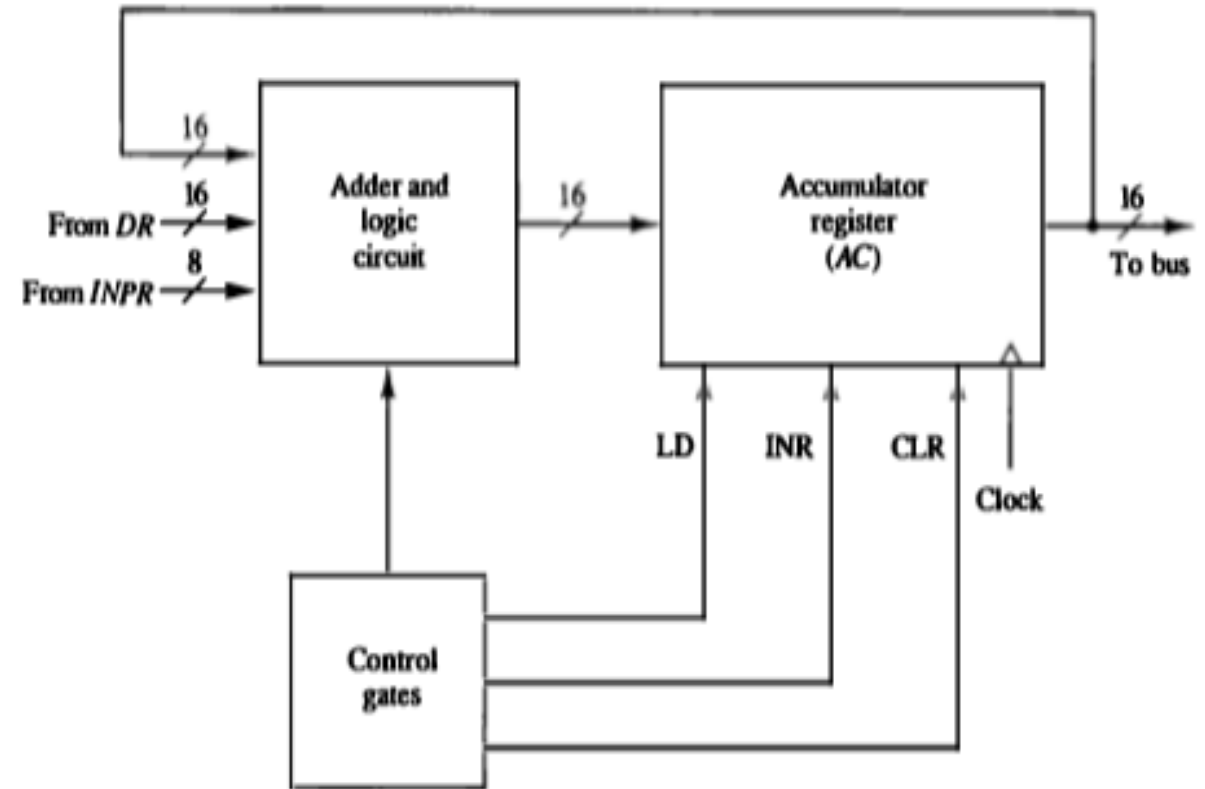
The basic computer consists of the following hardware components:

- ❖ A memory unit with 4096 words of 16 bits each.
- ❖ Nine registers: AR(Address Reg.), PC (Program Counter), DR(Data Reg.), AC (Accumulator), IR (Instruction Reg.), TR (Temp. Reg.), OUTR (Output Reg.), INPR (Input Reg.), and SC (Sequence Counter).
- ❖ Flip-flops: IEN (Interrupt Enable), FGI (Input Flag), and FGO (Output Flag).
- ❖ Two decoders: a 3 x 8 operation decoder and a 4 x 16 timing decoder
- ❖ A 16-bit common bus.
- ❖ Control logic gates.
- ❖ Adder and logic circuit connected to the input of AC.

Design of Accumulator Logic

- ❖ The circuits associated with the AC register are shown in Fig. The adder and logic circuit has three sets of inputs.
- ❖ One set of 16 inputs comes from the outputs of AC.
- ❖ Another set of 16 inputs comes from the 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 gates for controlling the LD, INR, and CLR in the register and for controlling the operation of the adder and logic circuit.

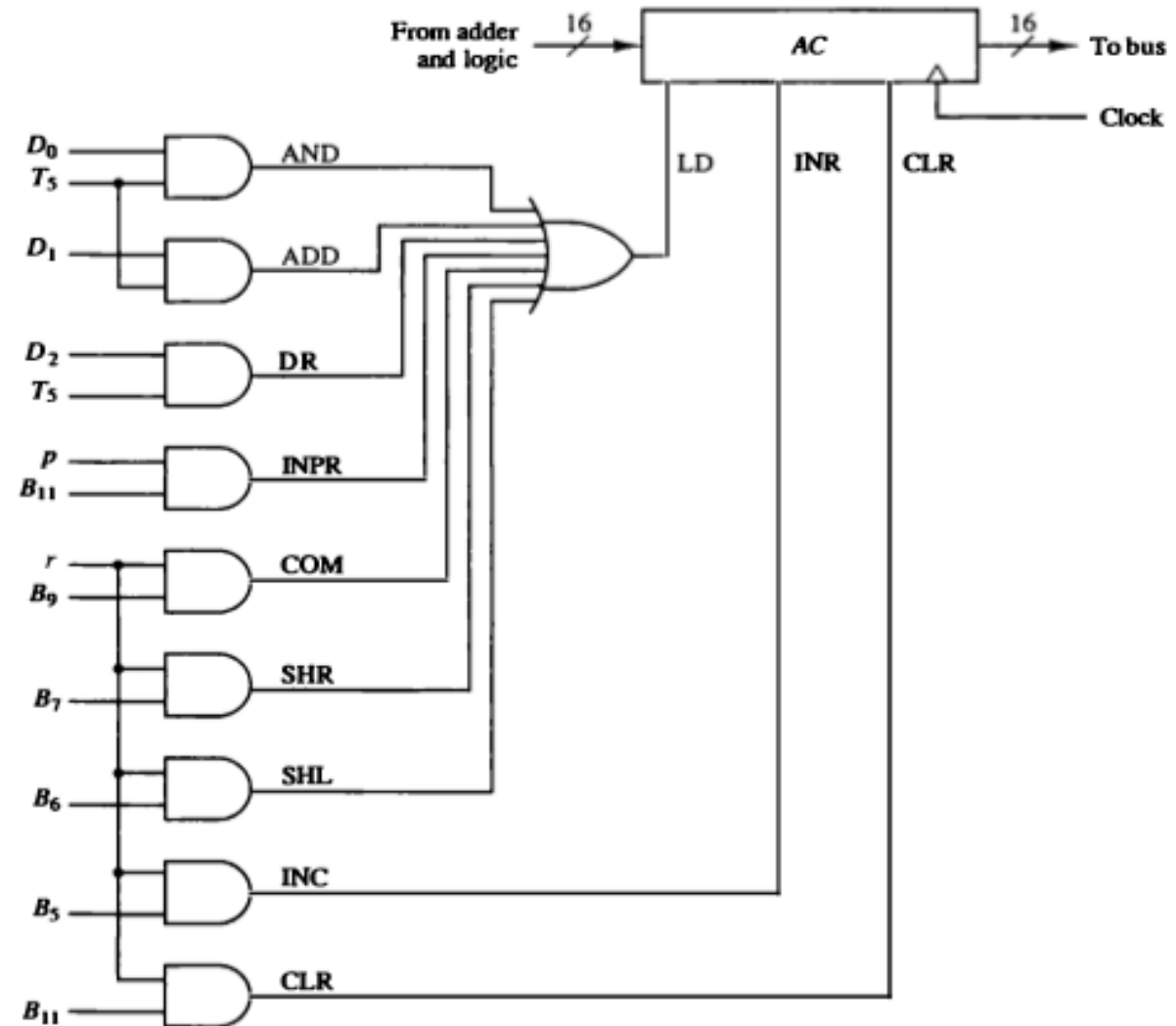
Figure 5-19 Circuits associated with AC.



Control of AC Register – Gate Structure

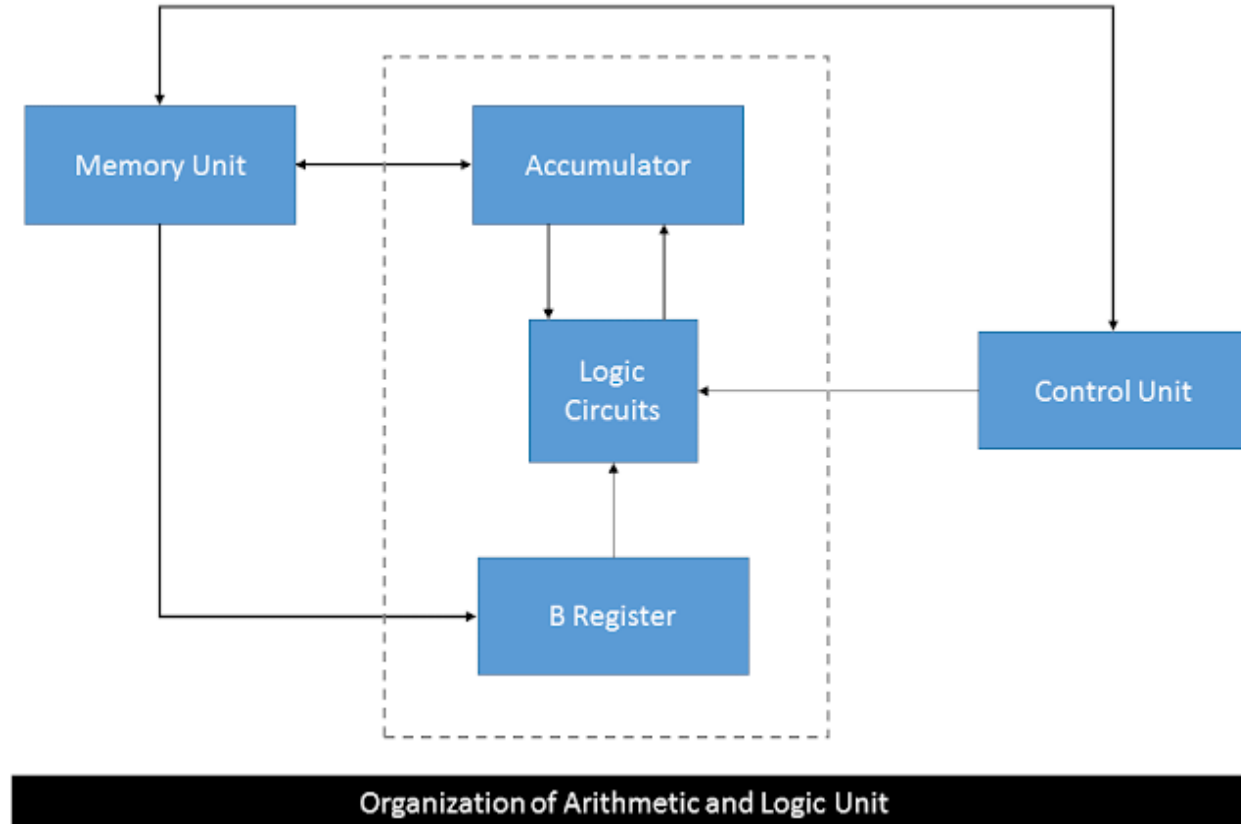
- ❖ The gate structure that controls the LD, INR, and CLR inputs of AC is shown in Fig.
- ❖ The output of the AND gate that generates this control function is connected to the CLR input of the register.
- ❖ Similarly, the output of the gate that implements the increment micro operation is connected to the INR input of the register.
- ❖ The other 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 a symbolic name. These outputs are used in the design of the adder and logic circuit.

Figure 5-20 Gate structure for controlling the LD, INR, and CLR of AC.



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.



Functions of ALU:

Functions of ALU or Arithmetic & Logic Unit can be categorized into following 3 categories:

1. 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.

2. Logical Operations:

Operations like AND, OR, NOR, NOT etc. using logical circuitry are examples of logical operations.

3. 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 microoperations.
- ❖ 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.

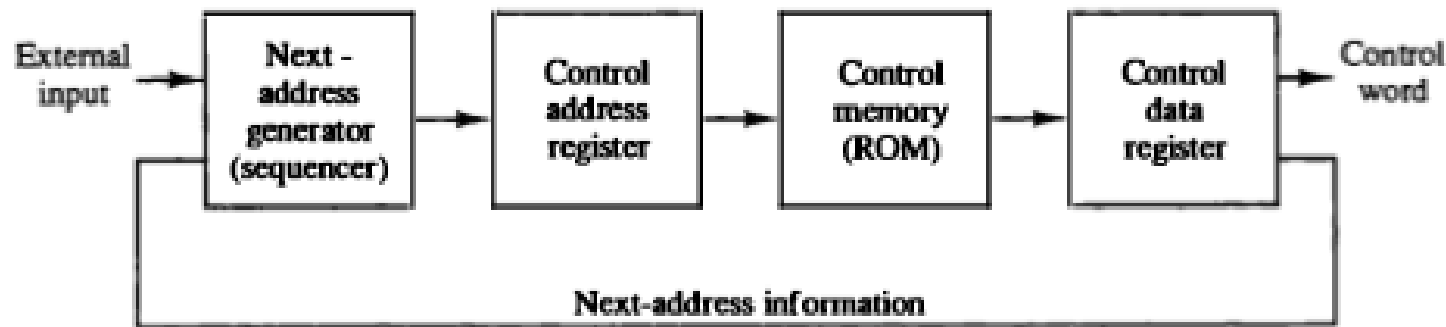
Control Memory

- ❖ The control unit initiates a series of sequential steps of microoperations. During any given time, certain microoperations 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 a microprogrammed control unit is demonstrated in the block diagram of Fig. The control memory is assumed to be a ROM, within which all control information is permanently stored.

Figure 7-1 Microprogrammed control organization.



- ❖ The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction 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.

Address Sequencing

- ❖ 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:
 1. Incrementing of the control address register.
 2. Unconditional branch or conditional branch, depending on status bit conditions.
 3. A mapping process from the bits of the instruction to an address for control memory.
 4. A facility for subroutine call and return.

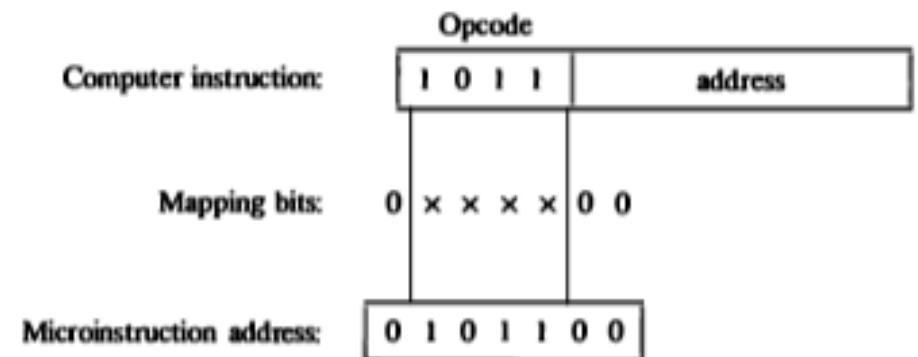
Conditional Branching

- ❖ 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. 7-3 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.

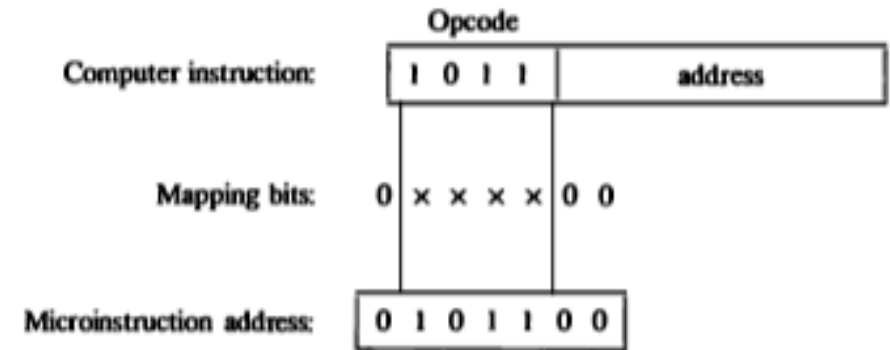
Figure 7-3 Mapping from instruction code to microinstruction address.



Mapping of 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.

Figure 7-3 Mapping from instruction code to microinstruction address.



Subroutines

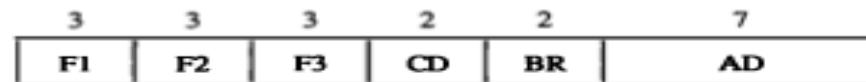
- ❖ **Subroutines** are programs that are used by other routines to accomplish a particular task.
- ❖ A subroutine can be called from any point within the main body of the microprogram.
- ❖ Frequently, many microprograms contain identical sections of code. Microinstructions can be saved by employing subroutines that use common sections of microcode.
- ❖ For example, the sequence of microoperations 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.
- ❖ Microprograms that use subroutines must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return. This may be accomplished by placing the incremented output from the control address register into a **subroutine register** and branching to the beginning of the subroutine. The subroutine register can then become the source for transferring the address for the return to the main routine.

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.

Microinstruction Format

The microinstruction format for the control memory is shown in Fig. 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 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

Figure 7-6 Microinstruction code format (20 bits).

Symbols & Binary Code for Microoperations

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	$AR \leftarrow DR(0-10)$	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	$DR(0-10) \leftarrow PC$	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	$PC \leftarrow PC + 1$	INCP
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

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

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	$CAR(2-5) \leftarrow DR(11-14), CAR(0,1,6) \leftarrow 0$

Description

- ❖ Each microoperation in Table is defined with a register transfer statement and is assigned a symbol for use in a symbolic microprogram. All transfer-type microoperations symbols use five letters. The first two letters designate the source register, the third letter is always a T, and the last two letters designate the destination register. For example, the microoperation that specifies the transfer $AC \leftarrow DR$ ($F1 = 100$) has the symbol DRTAC, which stands for a transfer from DR to AC.
- ❖ The **CD (condition)** field consists of two bits which are encoded to specify four status bit conditions as listed in Table. The first condition is always a 1, so that a reference to $CD = 00$ (or the symbol U) will always find the condition to be true. We will use the symbols U, I, S, and Z for the four status bits when we write microprograms in symbolic form.
- ❖ The **BR (branch)** field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction. As shown in Table, when $BR = 00$, the control performs a jump (JMP) operation (which is similar to a branch), and when $BR = 01$, it performs a call to subroutine (CALL) operation. The return from subroutine is accomplished with a BR field equal to 10. This causes the transfer of the return address from SBR to CAR. The mapping from the operation code bits of the instruction to an address for CAR is accomplished when the BR field is equal to 11.

Symbolic Vs Binary Microprogram

Symbolic Microprogram	Binary Microprogram																																																
1. It is a set of microinstructions written in a symbolic form.	1. It is a set of microinstructions written in a binary form.																																																
2. It is a convenient form for writing microprograms in a way that people can read and understand.	2. It is written in binary form so can be difficult for people to read and understand.																																																
3. 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.	3. Translation is not required for storing in memory because it is already written in binary form.																																																
4. The symbolic representation is useful for writing microprograms in an assembly language format.	4. The binary representation is the actual internal content that must be stored in control memory.																																																
5. Example: Symbolic microprogram for fetch routine: <div><table><tr><td>FETCH:</td><td>ORG 64</td><td></td><td></td><td></td></tr><tr><td></td><td>PCTAR</td><td>U</td><td>JMP</td><td>NEXT</td></tr><tr><td></td><td>READ, INCPC</td><td>U</td><td>JMP</td><td>NEXT</td></tr><tr><td></td><td>DRTAR</td><td>U</td><td>MAP</td><td></td></tr></table></div> <p>Each line of the assembly language microprogram defines a symbolic microinstruction. Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD. Here ORG 64 refers to the origin address or staring address.</p>	FETCH:	ORG 64					PCTAR	U	JMP	NEXT		READ, INCPC	U	JMP	NEXT		DRTAR	U	MAP		5. Example: Translation of symbolic microprogram to binary microprogram. <div><table><tr><th>Binary Address</th><th>F1</th><th>F2</th><th>F3</th><th>CD</th><th>BR</th><th>AD</th></tr><tr><td>1000000</td><td>110</td><td>000</td><td>000</td><td>00</td><td>00</td><td>1000001</td></tr><tr><td>1000001</td><td>000</td><td>100</td><td>101</td><td>00</td><td>00</td><td>1000010</td></tr><tr><td>1000010</td><td>101</td><td>000</td><td>000</td><td>00</td><td>11</td><td>0000000</td></tr></table></div>	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
FETCH:	ORG 64																																																
	PCTAR	U	JMP	NEXT																																													
	READ, INCPC	U	JMP	NEXT																																													
	DRTAR	U	MAP																																														
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.
- ❖ Control unit can be designed by two methods:
 1. Hardwired control Unit
 2. Micro programmed Control Unit

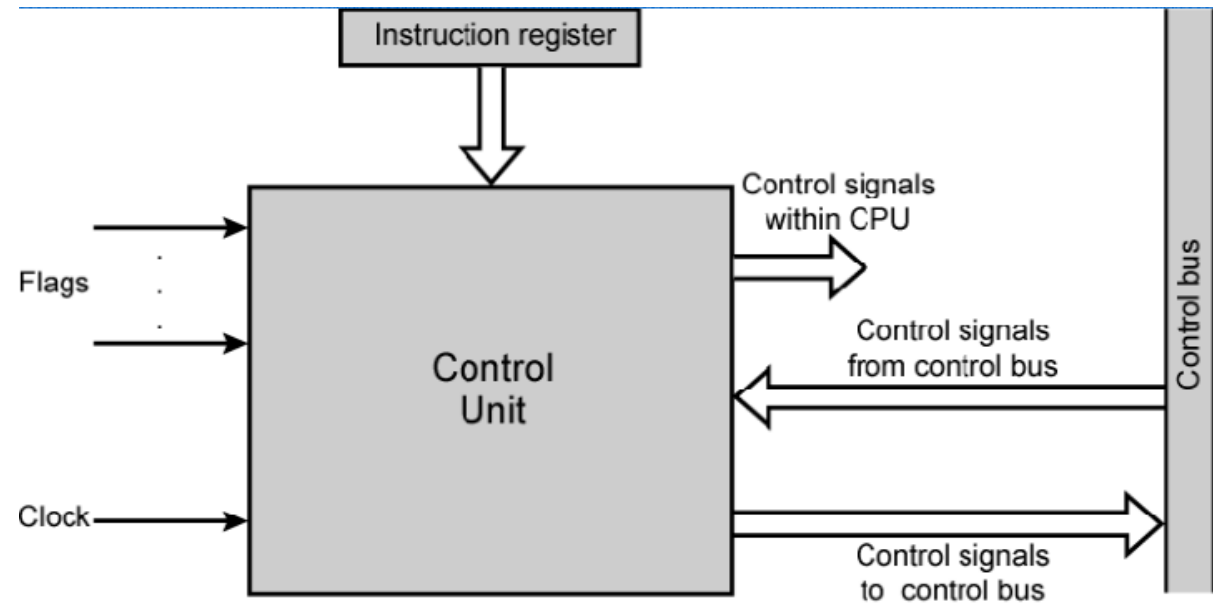


Fig: Block Diagram of Control Unit

Basic Requirement of Control Unit

- ❖ The functional requirements of control unit are those functions that the control unit must perform and these are the basis for the design and implementation of the control unit.
- ❖ A three step process that lead to the characterization of the Control Unit:
 1. Define the basis 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.

1. Basic Elements of Processor

The following are the basic functional elements of a CPU:

- ❖ **ALU**: is the functional essence of the computer.
- ❖ **Registers**: are used to store data internal to the CPU.

Basic Requirement of Control Unit

2. Types of Micro-operation

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

3. Functions of Control Unit

The control unit perform two tasks:

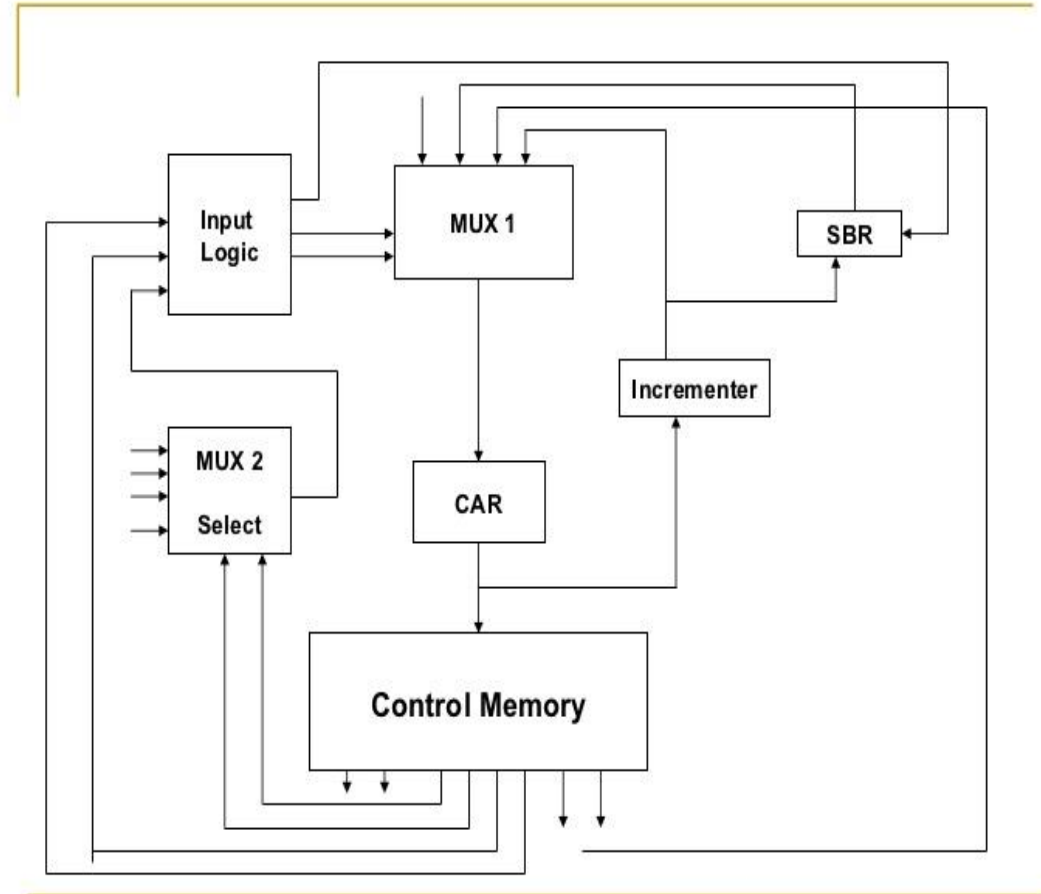
- ❖ **Sequencing**: The control unit causes the CPU to step through a series of micro-operations in proper sequence based on the program being executed.
- ❖ **Execution**: The control unit causes each micro-operation to be performed.

Microprogram 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.
- ❖ The next-address logic of the sequencer determines the specific address source to be loaded into the control address register. The choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction.
- ❖ Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutine calls. Some sequencers provide an output register which can function as the address register for the control memory.

Microprogram Sequencer

- ❖ 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.



Microprogram sequencer for a control Memory