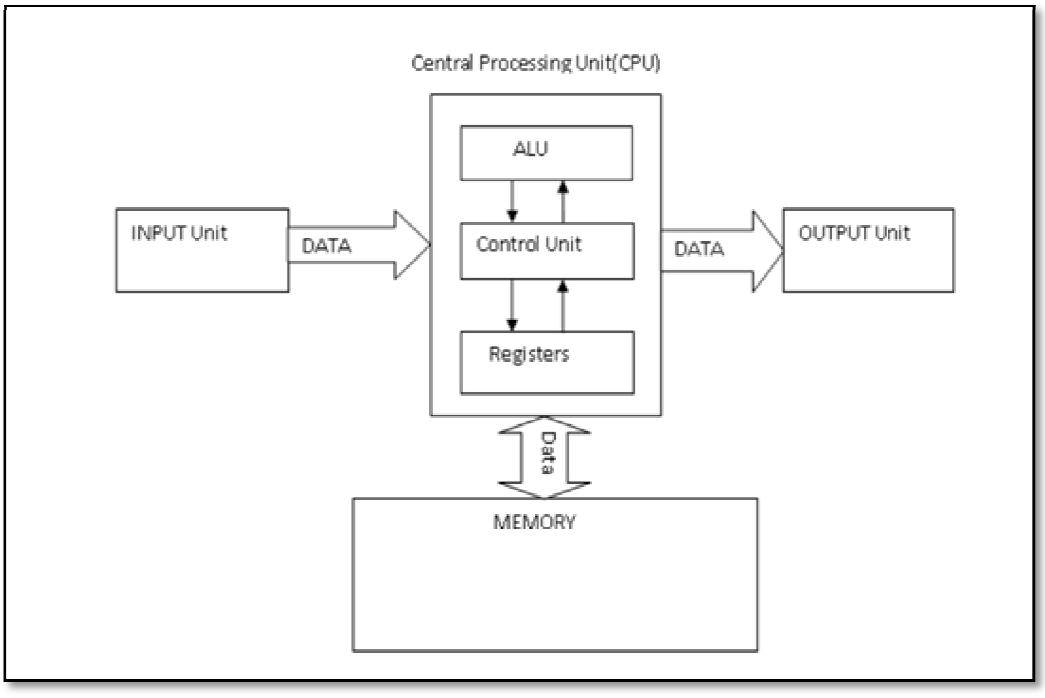
BASIC COMPUTER ORGANIZATION

A computer is a combination of hardware and software resources which integrate together and provides various functionalities to the user. Hardware are the physical components of a computer like the processor, memory devices, monitor, keyboard etc. while software is the set of programs or instructions that are required by the hardware resources to function properly.
There are a few basic components that aids the working-cycle of a computer i.e. the Input- Process- Output Cycle and these are called as the functional components of a computer. It needs certain input, processes that input and produces the desired output. The input unit takes the input, the central processing unit does the processing of data and the output unit produces the output. The memory unit holds the data and instructions during the processing.

A computer is made up of 4 functional components.They are:

1. Input Unit
2. Central Processing Unit (CPU)
    a. Control unit
    b. ALU
    c. Registers
3. Memory Unit
4. Output Unit



1. **Input Unit:**
   Computers need to receive data and instructions to solve any problem. The input unit basically links the external world or environment to the computer system. It consists of one or more input devices. The keyboard and mouse are the most commonly used input devices.

2. **Central Processing Unit(CPU):**

   Once the data and instructions are received from the input device, they are to be processed in this unit. So it can be considered as the heart or brain of the computer system.

   It consists of three major units:

   a) Control Unit :

   Control Unit controls and coordinates the activities of all the units of a computer system. It acts as a supervisor to the computer system.

   It performs the following functions:

   - Fetching the data and instructions from the main memory.
   - Interpreting these instructions.
   - Controlling the transfer of data and instructions to and from the main memory.
   - Controlling input and output devices.
   - The overall supervision of the computer system.

   b) ALU:

   All the arithmetic and logical calculations are carried out in the ALU. An ALU consists of electronic circuitry which performs arithmetic operations such as addition, subtraction, multiplication and division. It also consists of logical circuitry which performs logical operations like AND, OR, NOT. And relational operations like not equal to, = greater than, and == equal to.

   c) **Registers :**

   The CPU consists of several temporary storage units, which are used to store instructions & intermediate data which may be generated during processing.

3. **Memory Unit:**

   The data and instructions required for processing have to be stored in the memory unit before actual processing starts. Similarly, the results generated has to be preserved before it is displayed. The memory unit thus provides space to store input data, intermediate results and final output generated. eg: hard disks, pen drives, floppy disks.

4. **Output Unit :**

   It is used to print or display the result obtained by the execution of a program. Whenever the user wants output from the computer, the control unit sends a signal to this unit to be ready to accept processed data from the memory and to display it. Eg. Monitor, Printer, Speakers etc.

MACHINE INSTRUCTIONS

The words of a machine's language are called instructions,and its vocabulary is called an instruction set.

Computer instructions are a set of machine language instructions that a particular processor understands and executes.A computer performs tasks on the basis of the instruction provided.

Instruction formats:

An instruction format defines the layout of the bits of an instruction, in terms of its constituent fields.An instruction format must include an opcode and ,implicitly or explicitly ,zero or more operands. Each explicit operand is referenced using one of the addressing modes.

An instruction comprises of groups called fields. These fields include:

- The **mode field** which specifies how the operand will be located.
- The **operation code(opcode)** field which specifies the operation to be performed.
- The **address field** which contains the location of the operand,ie,register or memory location.

mode	opcode	address

I. CLASSIFICATION:

1. FUNCTION :

Instructions can be classified into the following four categories based on their functionality.

a) **Data processing :**

Data processing instructions are the ones that perform some mathematical or logical operation on some operands. The Arithmetic Logic Unit performs these operations, therefore the data processing instructions can also be called ALU instructions.

b) **Data storage (main memory) :**

The primary storage for the operands is the main memory. When an operation needs to be performed on these operands, these can be temporarily brought into the CPU registers, and after completion, these can be stored back to the memory. The instructions for data access and storage between the memory and the CPU can be categorized as the data storage instructions.

c) **Data movement (I/O) :**

The ultimate sources of the data are input devices e.g. keyboard. The destination of the data is an output device, for example, a monitor, etc. The instructions that enable such operations are called data movement instructions.

d) **Program flow control:**

A CPU executes instructions sequentially, unless a program flow-change instruction is encountered. This flow change, also called a branch, may be conditional or unconditional. In case of a conditional branch, if the branch condition is met, the target address is loaded into the program counter.

2. ADDRESSES :

Based on the number of address, instructions are classified as:

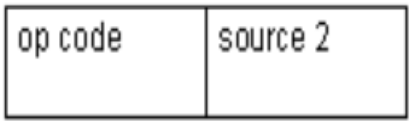
a) Zero address instruction:

A 0-address instruction uses a stack to hold both the operands and the result. Operations are performed on the operands stored on the top of the stack and the second value on the stack. The result is stored on the top of the stack. Just like the use of an accumulator register, the addresses of the stack registers need not be specified, their usage is implicit. Therefore, only one field is required in 0-address instruction; it specifies the op-code.

Example :PUSH

b) One address instruction:

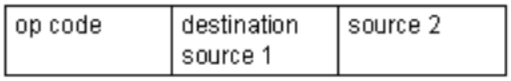
A 1-address instruction has a dedicated CPU register, called the accumulator, to hold one operand and to store the result. There is no need of encoding the address of the accumulator register to access the operand or to store the result, as its usage is implicit. There are two fields in the instruction, one for specifying a source operand address and a destination operand address.



Example :ADD X (AC ← AC + M[X])

c) Two address instruction:

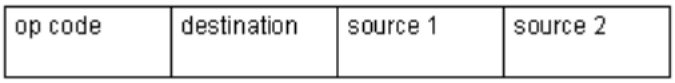
A 2-address instruction has three fields; one for the op-code, the second field specifies the address of one of the source operands as well as the destination operand, and the last field is used for holding the address of the second source operand. So one of the fields serves two purposes; specifying a source operand address and a destination operand address.



Example : ADD R1,R2 (R1← R1 + R2)

d) Three address instruction :

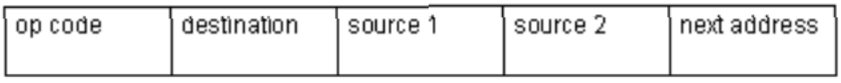
A 3-address instruction specifies the addresses of two operands and the address of the destination operand.



Example : ADD R1,R2,R3 (R1 ← R2 + R3)

e) Four address instruction:

The four address instructions specify the addresses of two source operands, the address of the destination operand and the next instruction address. 4-address instructions are not very common because the next instruction to be executed is sequentially stored next to the current instruction in the memory. Therefore, specifying its address is redundant. These instructions are used in the micro-coded control unit.



3. SIZE:

The basic computer has three instruction code formats. The **Operation code** (opcode) part of the instruction contains 3 bits and remaining 13 bits depends upon the operation code encountered.

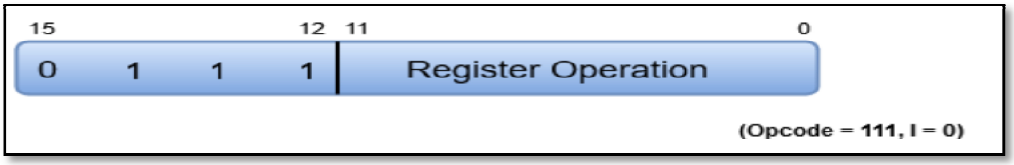
There are three types of formats:

a) Memory Reference Instruction:

It uses 12 bits to specify the address and 1 bit to specify the addressing mode (I). I is equal to 0 for direct address and 1 for indirect address.

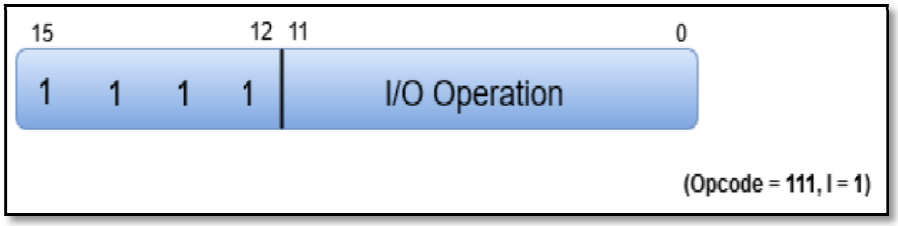
b) Register Reference Instruction :

These instructions are recognized by the opcode 111 with a 0 in the left most bit of instruction. The other 12 bits specify the operation to be executed.



c) Input-Output Instruction:

These instructions are recognized by the operation code 111 with a 1 in the left most bit of instruction. The remaining 12 bits are used to specify the input-output operation.



II. ADDRESSING MODES

The different ways of specifying the location of an operand in an instruction are called as **addressing modes**.

In computer architecture, there are following types of addressing modes:

1. Implied / Implicit Addressing Mode
2. Stack Addressing Mode
3. Immediate Addressing Mode
4. Direct Addressing Mode
5. Indirect Addressing Mode
6. Register Direct Addressing Mode
7. Register Indirect Addressing Mode
8. Relative Addressing Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode
11. Auto-Increment Addressing Mode
12. Auto-Decrement Addressing Mode

1. Implied / Implicit Addressing Mode :

In this addressing mode,the definition of the instruction itself specify the operands implicitly. It is also called as **implicit addressing mode**.

Example:
The instruction “Complement Accumulator” is an implied mode instruction.

2. Stack Addressing Mode:

In this addressing mode, the operand is contained at the top of the stack.

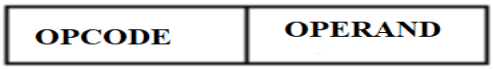
Example :

ADD

This instruction simply pops out two symbols contained at the top of the stack. The addition of those two operands is performed. The result so obtained after addition is pushed again at the top of the stack.

3. **Immediate Addressing Mode:**

In this addressing mode, the operand is specified in the instruction explicitly. Instead of address field, an operand field is present that contains the operand.

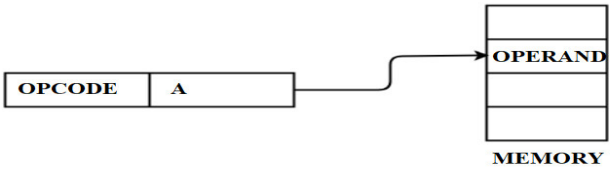


Example :

ADD 10 will increment the value stored in the accumulator by 10.

4. **Direct Addressing Mode :**

In this addressing mode, the address field of the instruction contains the effective address of the operand. Only one reference to memory is required to fetch the operand. It is also called as **absolute addressing mode**.



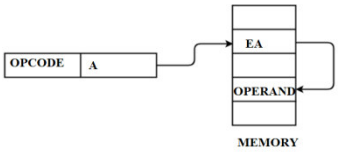
Example:

ADD X will increment the value stored in the accumulator by the value stored at memory location X.

AC ← AC + [X]

5. **Indirect Addressing Mode:**

In this addressing mode, the address field of the instruction specifies the address of memory location that contains the effective address of the operand. Two references to memory are required to fetch the operand.



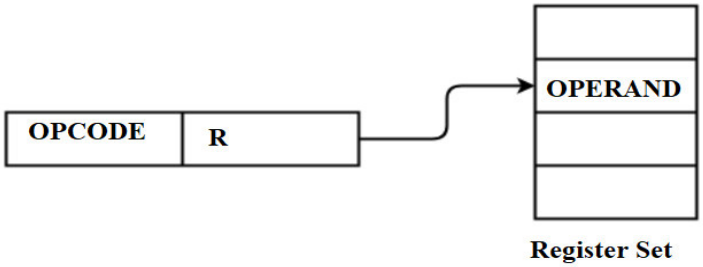
Example:

ADD X will increment the value stored in the accumulator by the value stored at memory location specified by X.

AC ← AC + [[X]]

6. **Register Direct Addressing Mode:**

In this addressing mode, The operand is contained in a register set. The address field of the instruction refers to a CPU register that contains the operand. No reference to memory is required to fetch the operand.



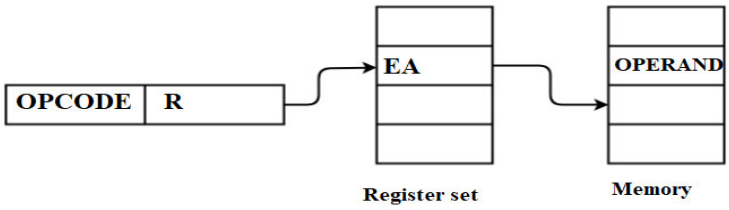
Example:

ADD R will increment the value stored in the accumulator by the content of register R.

AC ← AC + [R]

7. **Register Indirect Addressing Mode:**

In this addressing mode, the address field of the instruction refers to a CPU register that contains the effective address of the operand. Only one reference to memory is required to fetch the operand.



Example:

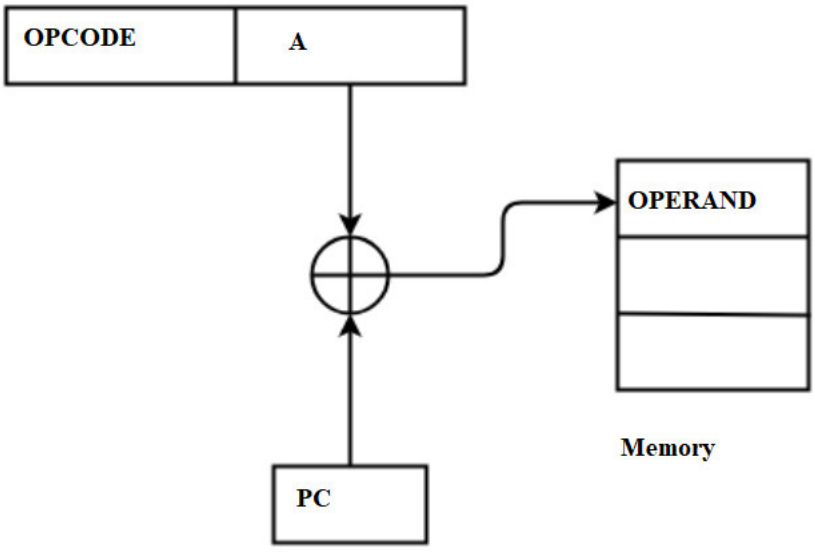
ADD R will increment the value stored in the accumulator by the content of memory location specified in register R.

AC ← AC + [[R]]

8. Relative Addressing Mode:

In this addressing mode, Effective address of the operand is obtained by adding the content of program counter with the address part of the instruction.

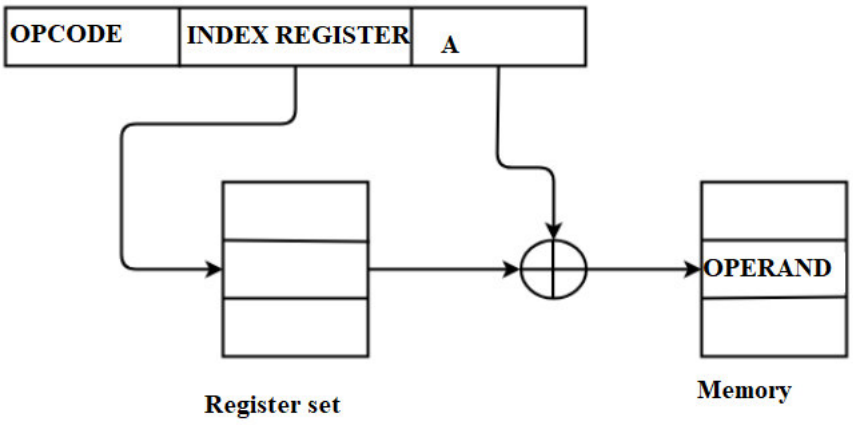
Effective Address = Content of Program Counter + Address part of the instruction



9. Indexed Addressing Mode:

In this addressing mode, Effective address of the operand is obtained by adding the content of index register with the address part of the instruction.

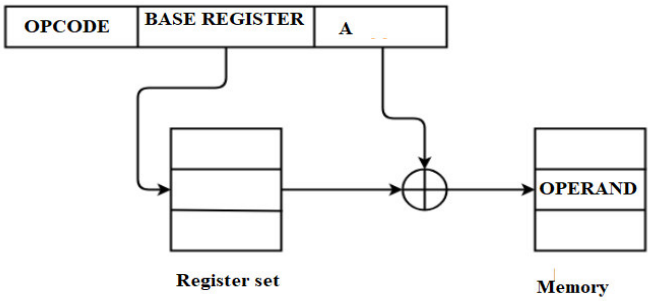
Effective Address= Content of Index Register + Address part of the instruction



10. Base Register Addressing Mode:

In this addressing mode, Effective address of the operand is obtained by adding the content of base register with the address part of the instruction.

Effective Address= Content of Base Register + Address part of the instruction

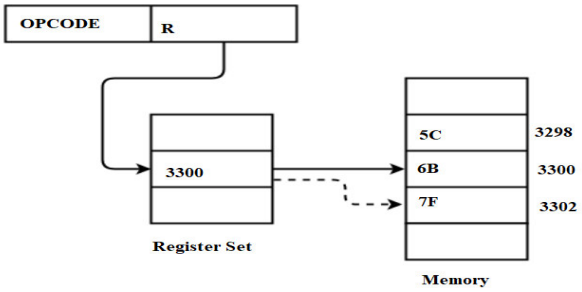


11. Auto-Increment Addressing Mode:

This addressing mode is a special case of Register Indirect Addressing Mode where-

Effective Address of the Operand= Content of Register

In auto-increment addressing mode,First, the operand value is fetched.Then, the instruction register R_{AUTO} value is incremented by step size 'd'.



Assume operand size = 2 bytes.

Here, After fetching the operand 6B, the instruction register R_{AUTO} will be automatically incremented by 2.

Then, updated value of R_{AUTO} will be 3300 + 2 = 3302.

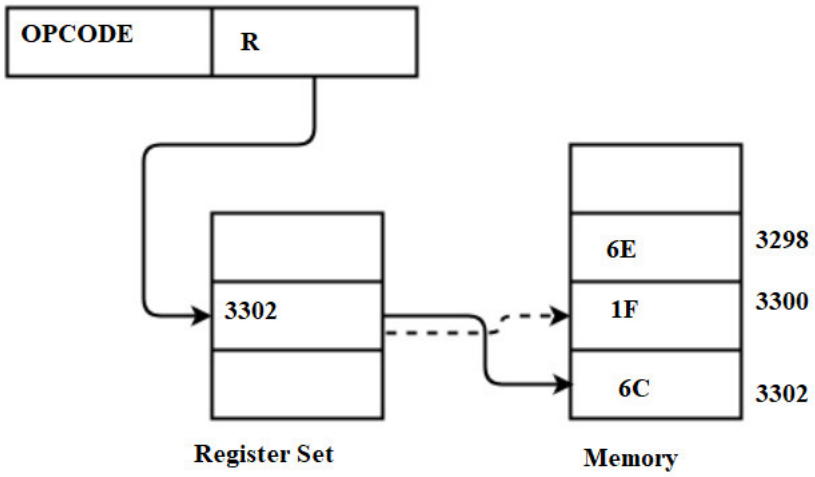
At memory address 3302, the next operand will be found.

12. Auto-Decrement Addressing Mode

This addressing mode is again a special case of Register Indirect Addressing Mode where-

Effective Address of the Operand= Content of Register – Step Size

In auto-decrement addressing mode,First, the instruction register R_{AUTO} value is decremented by step size 'd'.Then, the operand value is fetched.



Assume operand size = 2 bytes.

Here,

First, the instruction register R_{AUTO} will be decremented by 2.

Then, updated value of R_{AUTO} will be 3302 – 2 = 3300.

At memory address 3300, the operand will be found.

III.INSTRUCTION CYCLE

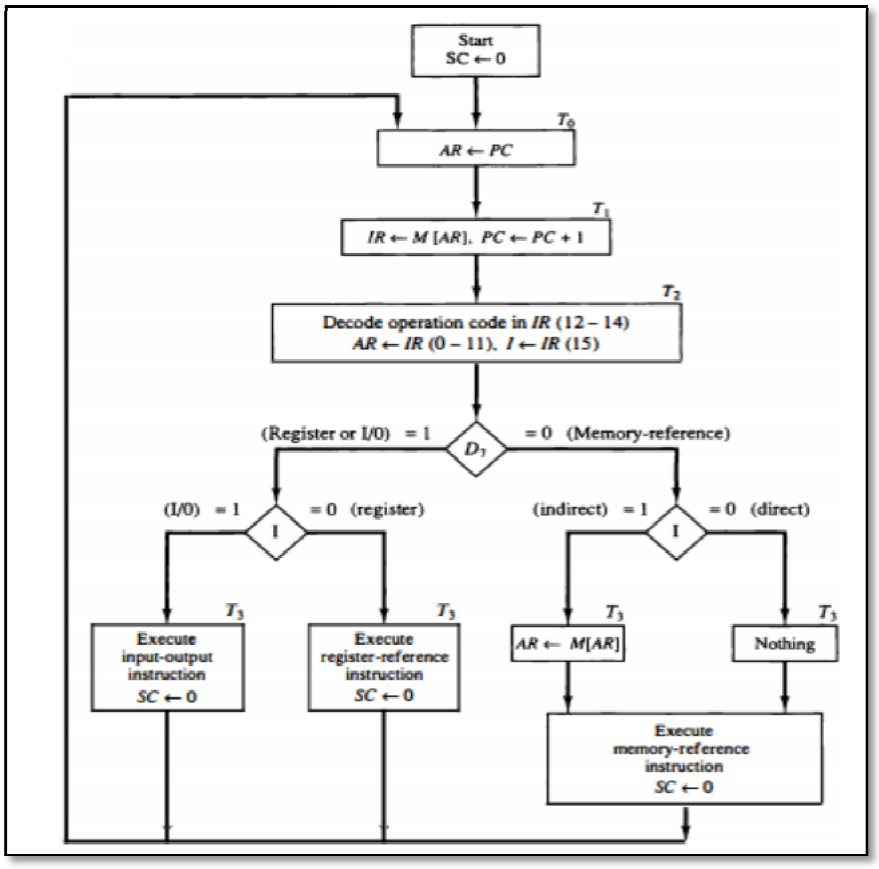
A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory
2. Decode the instruction
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

1. **Fetch an instruction from memory:**
   The instruction is fetched from memory address that is stored in PC and stored in the instruction register IR.At the end of the fetch operation,PC is incremented by 1 and it then points to the next instruction to be executed.
2. **Decode the instruction:**
   The instruction in the IR is executed by the decoder.
3. **Read the effective address:**
   If the instruction has an indirect address,the effective address is read from the memory.Otherwise operands are directly read in case of immediate operand instruction.
4. **Execute the instruction:**
   In the last phase the processor execute the instruction.

Flow chart :



Registers Involved In Each Instruction Cycle:

- Memory address registers(MAR) : It is connected to System Bus address lines. It specifies the address of a read or write operation in memory.
- Memory Buffer Register(MBR) : It is connected to the data lines of the system bus. : It is connected to the system bus Data Lines. It holds the memory value to be stored, or the last value read from the memory.
- Program Counter(PC) : Holds the address of the next instruction to be fetched.
- Instruction Register(IR) : Holds the last instruction fetched.

IV.INSTRUCTION SEQUENCING

The order in which the instructions in a program are carried out. Normally the sequence proceeds in a linear fashion through the program, and the address of the instructions is obtained from the program counter in the control unit. This sequence is interrupted when a branch instruction is executed; at such a time the address field of the branch instruction is inserted into the program counter and the process continues. In the case of an indirect branch instruction, the memory content referred to by the address field of the instruction is inserted into the program counter.

A computer program consist of a sequence of small steps, such as adding two numbers,testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen. A computer must have instructions capable of performing four types of operations:

1. Data transfers between the memory and the processor registers
2. Arithmetic and logic operations on data
3. Program sequencing and control
4. I/O transfers

REGISTER TRANSFER NOTATION:

In general, the information is transferred from one location to another in a computer. Possible locations for such transfers are memory locations, processor registers, or registers in the I/O subsystem. For example, Names for the addresses of memory locations may be LOC, PLACE, A, VAR2; processor register names may be RO, R5; and I/O register names may be DATAIN, OUTSTATUS, and so on. The contents of a location are denoted by placing square brackets around the name of the location. Thus, the expression

$$R1 \longleftarrow [LOC]$$

means that the contents of memory location LOC are transferred into processor register R1. As another example, consider the operation that adds the contents of registers R1 and R2, and then places their sum into register R3. This action is indicated as

$$R1 \leftarrow [R1] + [R2]$$

This type of notation is known as Register Transfer Notation (RTN). .Note that the right hand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be placed, overwriting the old contents of that location.

ASSEMBLY TRANSFER NOTATION:

The another type of notation to represent machine instructions and programs is assembly transfer notation. For example, an instruction that causes the transfer described above, from memory location LOC to processor register RI, is specified by the statement.

Move LOC,Rl

The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten. The second example of adding two numbers contained in processor registers Rl and R2 and placing their sum in R3 can be specified by the assembly language statement.

Add RI,R2,R3

INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING

The program is executed as follows:

1) Initially, the address of the first instruction is loaded into PC .

2) Then, the processor control circuits use the information in the PC to fetch and

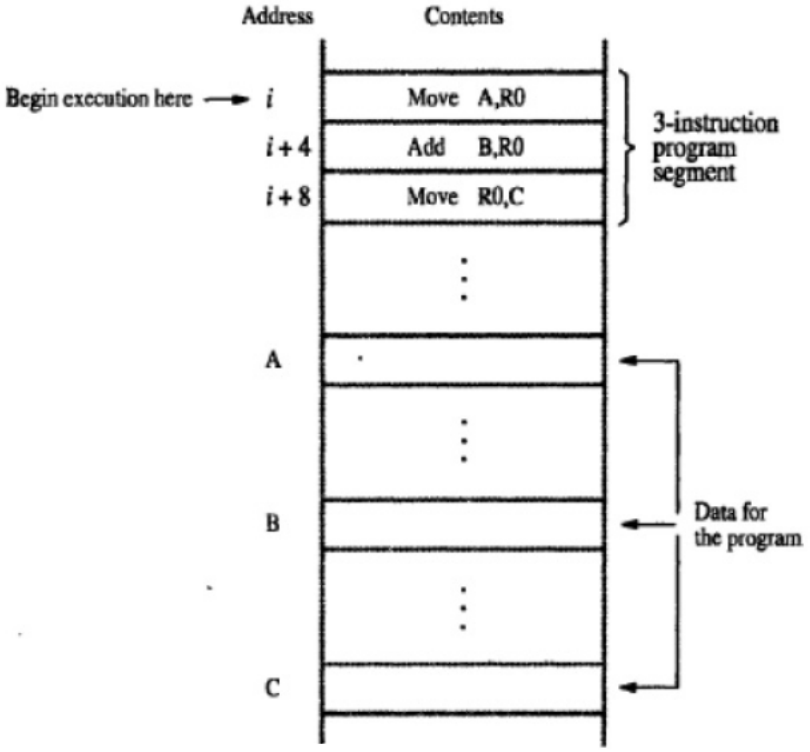
execute instructions, one at a time, in the order of increasing addresses. This is called Straight-Line sequencing.

3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.

There are 2 phases for Instruction Execution:

1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.

2) Execute Phase: The contents of IR is examined to determine which operation is to beperformed. The specified-operation is then performed by the processor.



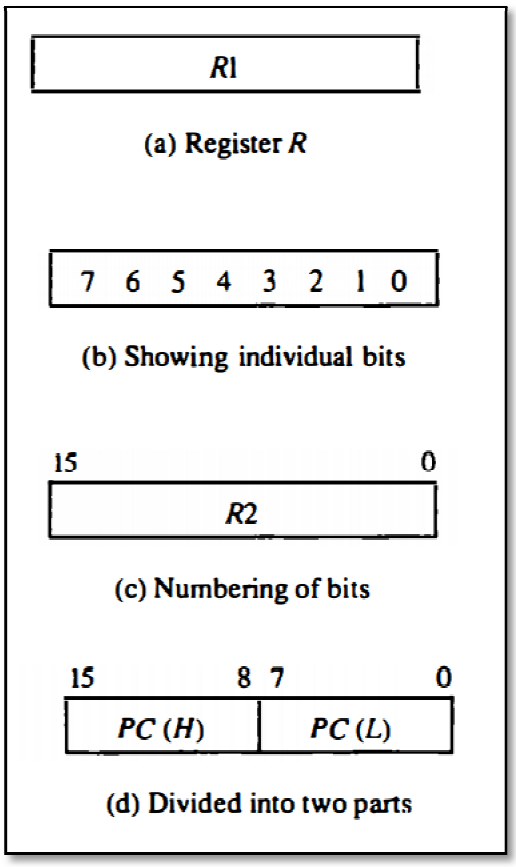
A program for C ← [A] + [B].

FUNDAMENTAL CONCEPTS

I. REGISTERS

Computer registers are designated by capital letters to denote the function of the register. For example, the register that holds an address for the memory unit is usually called a memory address register and is designated by the name MAR. Other designations for registers are PC (for program counter), IR (for instruction register, and R_1 (for processor register).

Block Diagram:



II. REGISTER TRANSFERS

Information transfer from one register to another is designated in symbolic form by means of a replacement operator. The statement **R2 ← R1** denotes a transfer of the content of register R1 into register R2.

It designates a replacement of the content of R2 by the content of Rl. By definition, the content of the source register R1 does not change after the transfer.

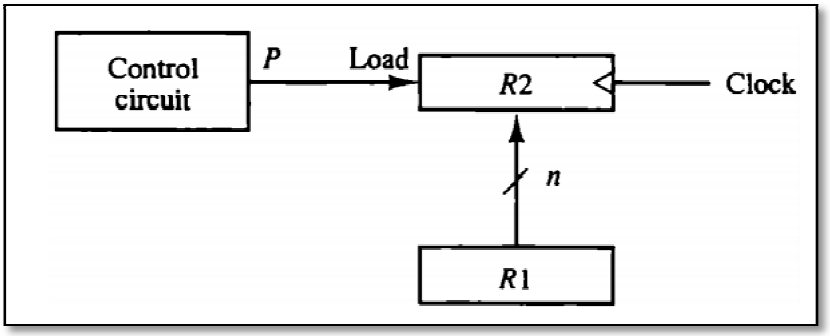
Control function:

A control function is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

P: R2 ← R1

The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1.

Block diagram(Transfer from Rl to R2 when P = 1.)



The above diagram depicts the transfer from R1 to R2. The n outputs of register R1 are connected to the n inputs of register R2. The letter n will be used to indicate any number of bits for the register. It will be replaced by an actual number when the length of the register is known. Register R2 has a load input that is activated by the control variable P. It is assumed that the control variable is synchronized with the same clock as the one applied to the register.

Basic Symbols for Register Transfers:

Symbol	Description	Examples
Letters	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Comma ,	Separates two microoperations	R2 ← R1, R1 ← R2

III. PERFORMING ARITHMETIC OR LOGIC OPERATIONS

ALU is responsible to perform the operation in the computer. The basic operations are implemented in hardware level. ALU is having collection of two types of operations: Arithmetic operations and Logical operations .

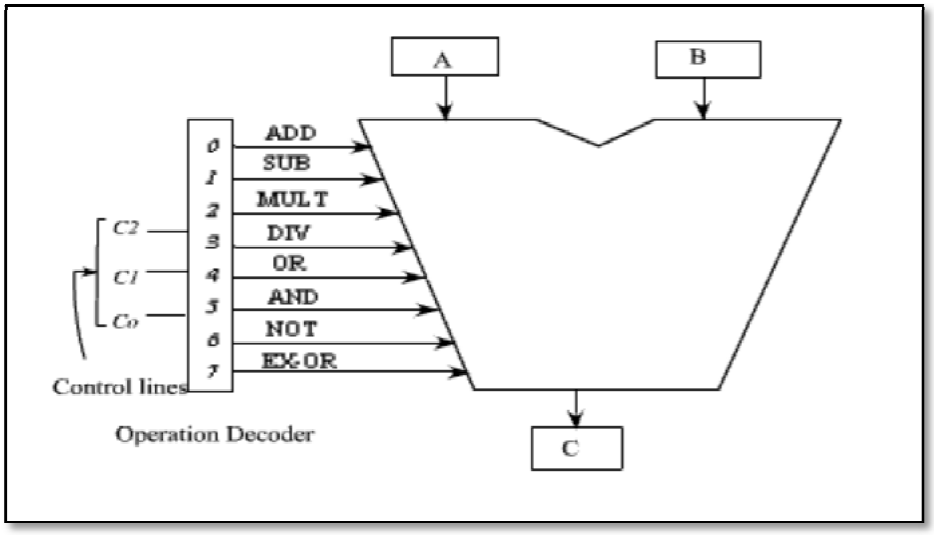
Consider an ALU having 4 arithmetic operations and 4 logical operation.

To identify any one of these four logical operations or four arithmetic operations, two control lines are needed. Also to identify the any one of these two groups-arithmetic or logical, another control line is needed. So, with the help of three control lines, any one of these eight operations can be identified.

Consider an ALU is having four arithmetic operations. Addition, subtraction, multiplication and division. Also consider that the ALU is having four logical operations: OR, AND, NOT & EX-OR.

C_1	C_0	Arithmetic $C_2 = 0$	Logical $C_2 = 1$
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	**EX-OR**

Block Diagram:



The ALU has got two input registers named as A and B and one output storage register, named as C. It performs the operation as:

C = A op B

The input data are stored in A and B, and according to the operation specified in the control lines, the ALU perform the operation and put the result in register C.

As for example, if the contents of controls lines are, 000, then the decoder enables the addition operation and it activates the adder circuit and the addition operation is performed on the data that are available in storage register A and B . After the completion of the operation, the result is stored in register C.

Arithmetic Microoperation:

It perform arithmetic operations on numeric data stored in registers.

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

Logic microoperation:

It perform bit manipulation operations on nonnumeric data stored in registers.

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

IV. MEMORY READ AND WRITE

A memory word will be symbolized by the letter M. The particular memory word among the many available is selected by the memory address during the transfer. It is necessary to specify the address of M when writing memory transfer operations. This will be done by enclosing the address in square brackets following the letter M.

Read operation:

The transfer of information from a memory word to the outside environment is called a read operation.

Consider a memory unit that receives the address from a register, called the address register, symbolized by AR . The data are transferred to another register, called the data register, symbolized by DR . The read operation can be stated as follows:

Read: DR ←M[AR]

This causes a transfer of information into DR from the memory word M selected by the address in AR .

Wrire operation:

The transfer of new information to be stored into the memory is called a write operation.

The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR.The write operation can be stated symbolically as follows:

Write: M[AR] ← R1

V. EXECUTION OF A COMPLETE INSTRUCTION

Various steps are involved in executing a complete instruction:

1. Fetch the instruction
2. Fetch the first operand
3. Perform the action
4. Load the result

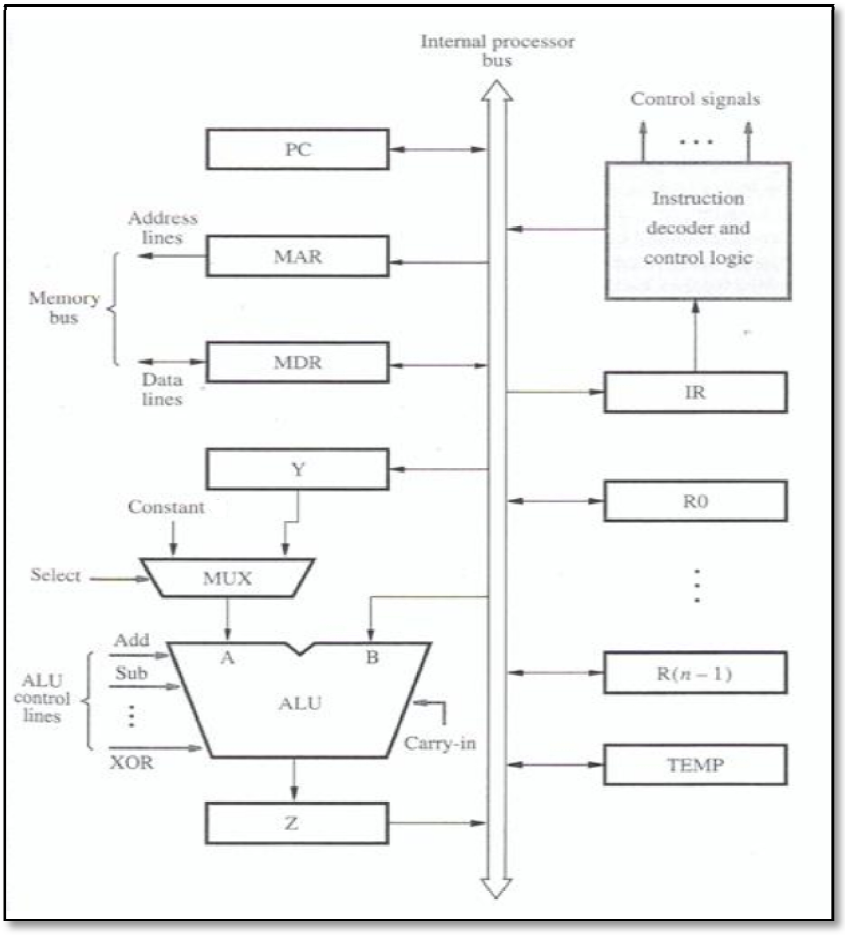
Example :

Let us find the complete control sequence for execution of the instruction add R1,(R2) for the single bus processor..This instruction add the contents of register R1 and the contents of memory location specified by register R2 and store result in the register R1.

To execute bus instruction it is necessary to perform following actions:

1. Fetch the instruction
2. Fetch the operand from memory location pointed by R2.
3. Perform the addition
4. Store the result in R1

The sequence of control steps required to perform these operations for the single bus architecture are as follows:



Step	Action
1	PC $_{out}$, MAR $_{in}$,Read, Y $_{in}$,Select C, Add, Z $_{in}$
2	Z $_{out}$, PC $_{in}$, WFMC
3	MDR $_{out}$, IR $_{in}$
4	R2 $_{out}$, MAR $_{in}$, Read
5	R1 $_{out}$, Y $_{in}$, WMFC
6	MDR $_{out}$, Select Y , Add, Z $_{in}$
7	Z $_{out}$,R1 $_{in}$, End

Step 1:

In step1,the instruction fetch operation is initiated by loading the contents of the PC into the MAR and sending a read request to memory.PC contents are also loaded into register Y and added with constant number by activating Select C input of multiplexer and add input of the ALU.By activating Z_{in} signal result is stored in the register Z.

Step 2:

In step 2 the contents of register Z are transferred to PC register by activating Z_{out} and PC_{in} signal.This completes the PC increment operation and PC will now point to next instruction.After receiving WMFC signal,the contents of specified location are available in MDR register.

Step 3:

In step3 contents of MDR register are transferred to the instruction register(IR) of the processor.The step1 through 3 constitute the instruction fetch place.

At the beginning of step 4 ,the instruction decoder interprets the contents of the IR.This enables the control circuiting to activate the control signal for step 4 through 7,which constitute the execution phase.

Step 4:

In step 4,the contents of register R2 are transferred to register MAR by activating $R2_{out}$ and MAR $_{in}$ signals are read signal is activated.

Step 5:

In step 5,the contents of register R1 are transferred to register Y by activating $R1_{out}$ and Y_{in} signals.After receiving WMFC signal,the contents of specified location are available in MDR register.

Step 6:

In step 6,MDR_{out},Select Y,Add and Z_{in} signals are activated to perform addition of contents of register Y and the contents of MDR.The result is stored in the register Z.

Step 7:

In step 7,the contents of register Z are transferred to register R1 by activating Z_{out} and $R1_{in}$ signals.The End signal causes a new instruction fetch cycle to begin by returning to step 1.

VI. BRANCH INSTRUCTION

With the help of branching instruction, the control of the execution of the program is transferred from one particular position to some other position, due to which the sequence flow of control is broken. Branching is accomplished by replacing the current contents of the PC by the branch address, that is, the address of the instruction to which branching is required.

Consider a branch instruction in which branch address is obtained by adding an offset X, which is given in the address field of the branch instruction, to the current value of PC.

Consider the following unconditional branch instruction

JUMP X

i.e., the format is

op- code offset of jump

The control sequence that enables execution of an unconditional branch instruction using the single bus organization is as follows

Step	Action
1	PC_{out} , MAR_{in} ,Read, Y_{in} ,Select C, Add, Z_{in}
2	Z_{out} , PC_{in} , WFMC
3	MDR_{out} , IR_{in}
4	Offset of IR_{out} ,Add,Z_{in}
5	Z_{out} ,PC_{in} , End

In step 4,the offset X of the instruction is gated to the bus and the addition operation is performed.

In step 5,The result of the addition ,which represents the branch address is loaded into the PC.It generates the End signal to indicate the end of execution of the current instruction.

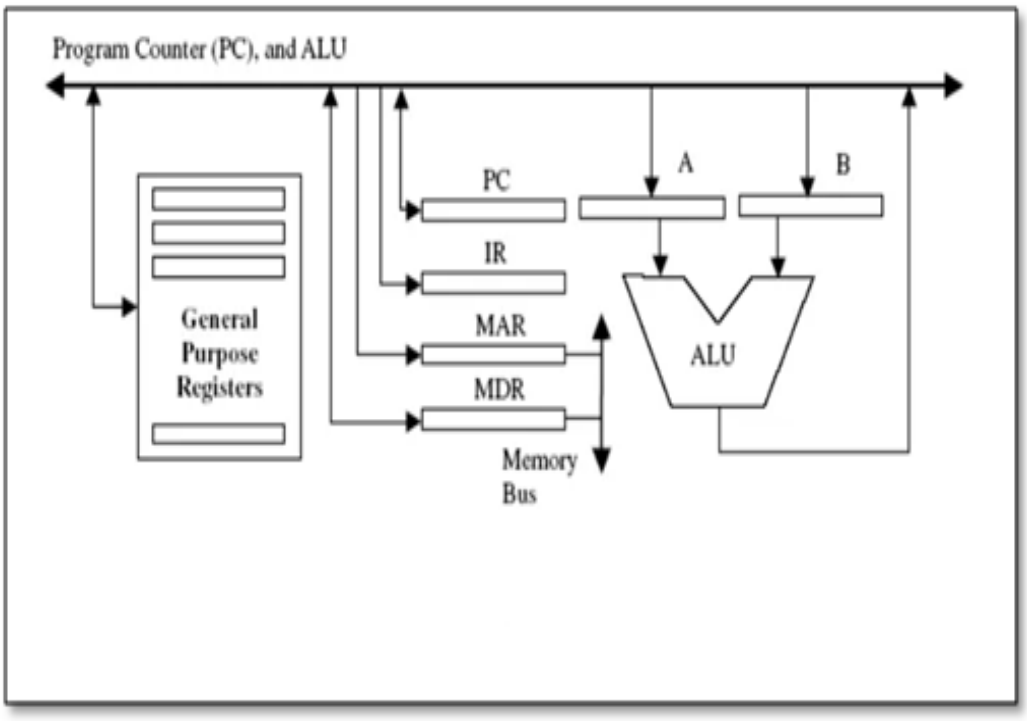
VII.SINGLE BUS, TWO BUS, THREE BUS ORGANIZATION

Internal data movement among registers and between the ALU and registers may be carried out using different organizations including one-bus, two-bus, or three-bus organizations.

Single bus organization:

Using one bus, the CPU registers and the ALU use a single bus to move outgoing and incoming data. Since a bus can handle only a single data movement within one clock cycle, two-operand operations will need two cycles to fetch the operands for the ALU. Additional registers may also be needed to buffer data for the ALU.

This bus organization is the simplest and least expensive, but it limits the amount of data transfer that can be done in the same clock cycle, which will slow down the overall performance.



It consisting of a set of general-purpose registers, a memory address register (MAR), a memory data register (MDR), an instruction register (IR), a program counter (PC), and an ALU.

Example : Add R1,R2

Pc_{out} ,AR_{in} ,Read , Set carry of ALU, Clear y,Add, Z_{in}.

Z_{out} , Pc_{in} , WMFC (Wait for Memory-Function-Completed signal).

DR_{out} , IR_{in} .

$R1_{out}$, y_{in}.

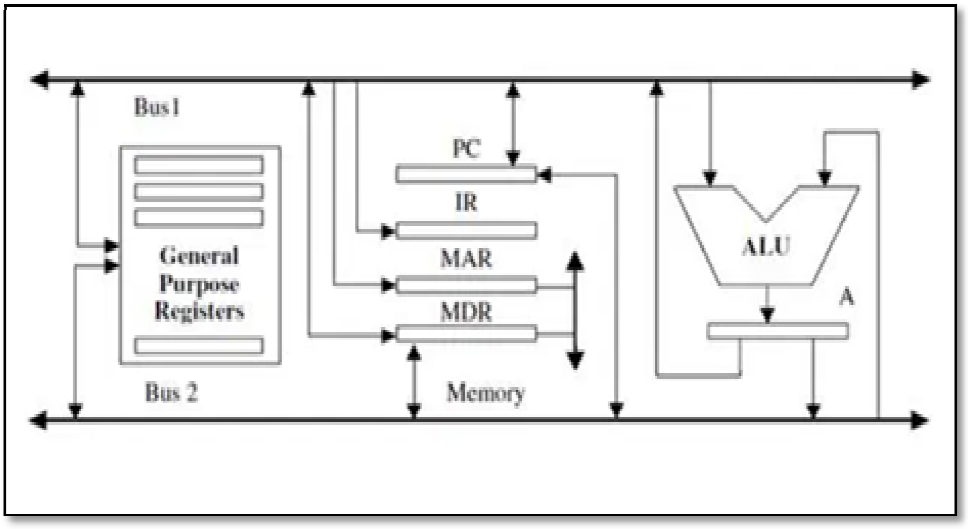
$R2_{out}$, add ,Z_{in}

Z_{out} , $R1_{in}$

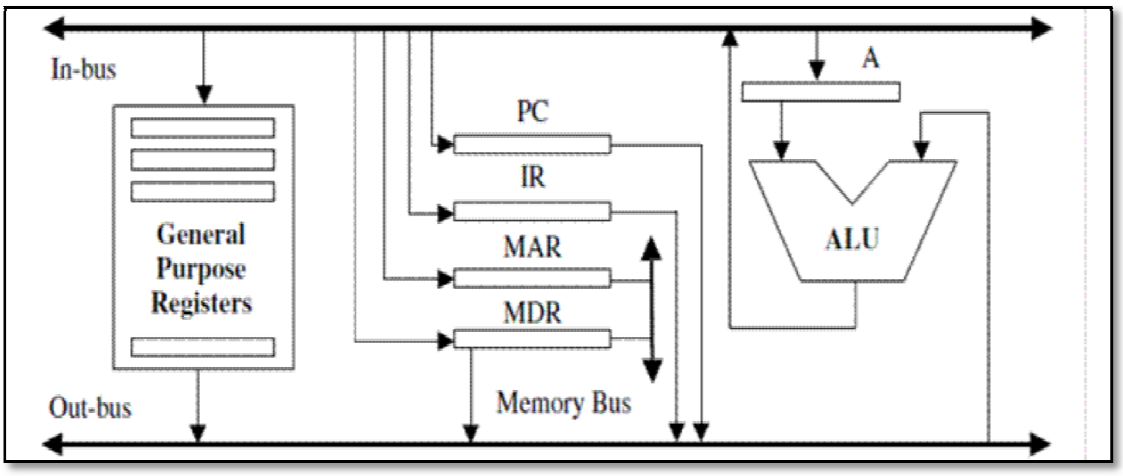
End.

Two-Bus Organization:

Using two buses is a faster solution than the one-bus organization. In this case, general-purpose registers are connected to both buses. Data can be transferred from two different registers to the input point of the ALU at the same time. Therefore, a two operand operation can fetch both operands in the same clock cycle. An additional buffer register may be needed to hold the output of the ALU when the two buses are busy carrying the two operands.



In some cases, one of the buses may be dedicated for moving data into registers (in-bus), while the other is dedicated for transferring data out of the registers (out-bus). In this case, the additional buffer register may be used, as one of the ALU inputs, to hold one of the operands. The ALU output can be connected directly to the in-bus, which will transfer the result into one of the registers.



Example : Add R1,R2

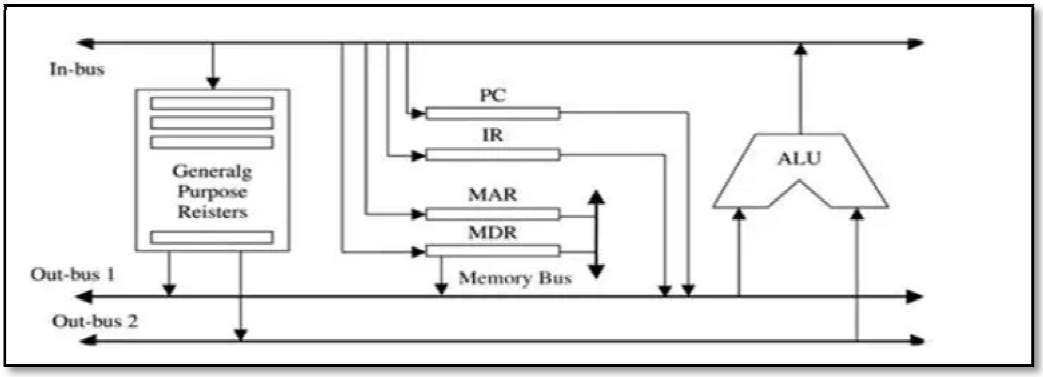
$R1_{out}$, y_{in}.

R2out, add ,R1in

End.

Three-Bus Organization:

In a three-bus organization, two buses may be used as source buses while the third is used as destination. The source buses move data out of registers (out-bus), and the destination bus may move data into a register (in-bus). Each of the two outbuses is connected to an ALU input point. The output of the ALU is connected directly to the in-bus. As can be expected, the more buses we have, the more data we can move within a single clock cycle. However, increasing the number of buses will also increase the complexity of the hardware.



Example : Add R1,R2

$R1_{out}$, $R2_{out}$,add, $R1_{in}$

End.

A COMPLETE PROCESSOR

A processor (CPU) is the logic circuitry that responds to and processes the basic instructions that drive a computer.

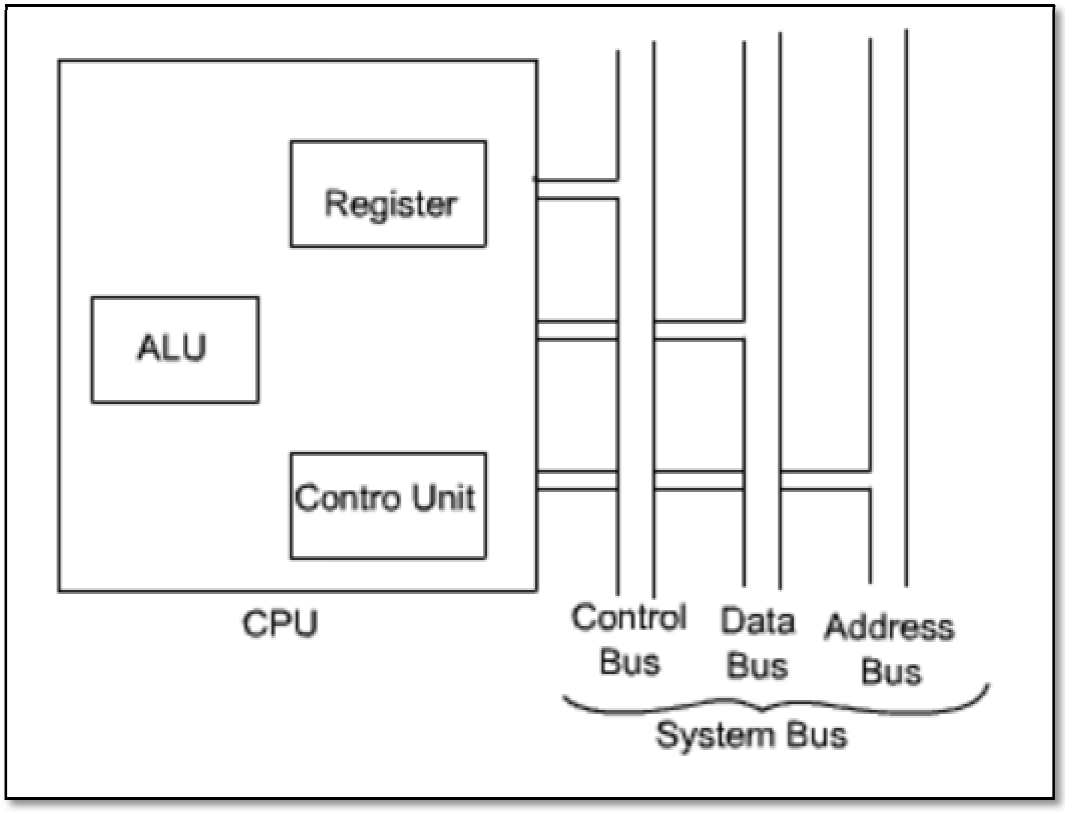
The major components of the CPU are an arithmatic and logic unit (ALU),registers and a control unit (CU).

The ALU does the actual computation or processing of data.

A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor. A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters).

The CU controls the movement of data and instruction into and out of the CPU and controls the operation of the ALU.

CPU with the System bus:



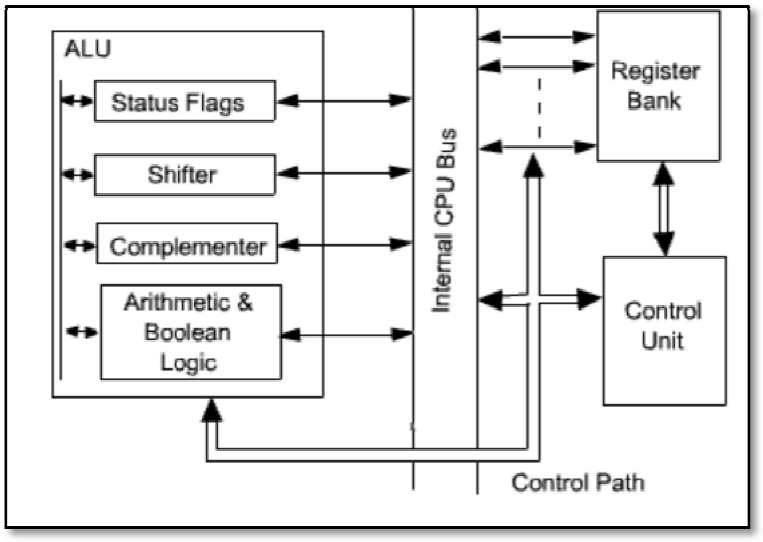
The CPU is connected to the rest of the system through system bus. Through system bus, data or information gets transferred between the CPU and the other component of the system. The system bus may have three components:

- **Data Bus:** Data bus is used to transfer the data between main memory and CPU.
- **Address Bus:** Address bus is used to access a particular memory location by putting the address of the memory location.
- **Control Bus:** Control bus is used to provide the different control signal generated by CPU to different part of the system.

Internal structure of the processor(CPU):

There are three basic components of CPU: register bank, ALU and Control Unit. There are several data movements between these units and for that an internal CPU

bus is used. Internal CPU bus is needed to transfer data between the various registers and the ALU.



The operation or task that must perform by CPU are:

1. **Fetch Instruction:** The CPU reads an instruction from memory.
2. **Interpret Instruction:** The instruction is decoded to determine what action is required.
3. **Fetch Data:** The execution of an instruction may require reading data from memory or I/O module.
4. **Process data:** The execution of an instruction may require performing some arithmatic or logical operation on data.

Write data: The result of an execution may require writing data to memory or an I/O module.

CONTROL UNIT

The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

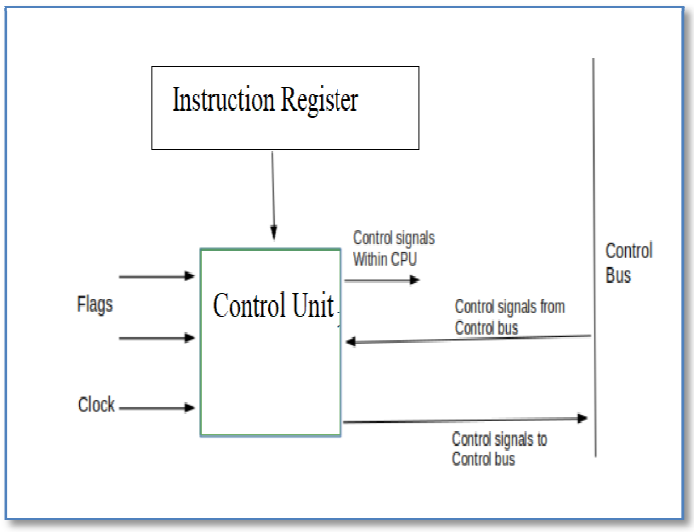
It performs two tasks:

- **Sequencing:**Initiate sequence of micro operations based on the program.
- **Execution:**Each micro operation to be performed.

Functions of the Control Unit:

1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
2. It interprets instructions.
3. It controls data flow inside the processor.
4. It receives external instructions or commands to which it converts to sequence of control signals.
5. It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU.
6. It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

Block Diagram:



The inputs are:

- **Clock:** This helps to keep time in control unit. Control unit perform operation for each clock pulse.
- **Flags:** This specify the status of the CPU.
- **Instruction Register:** It used to store instruction.
- **Control signals from control bus.**

The outputs are:

- Control signals within CPU.
  - Data to be moved from one register to another.
  - Those that activate specific ALU functions.
- Control signals to system bus
  - Control signal to memory.
  - Control signals to I/O modules.

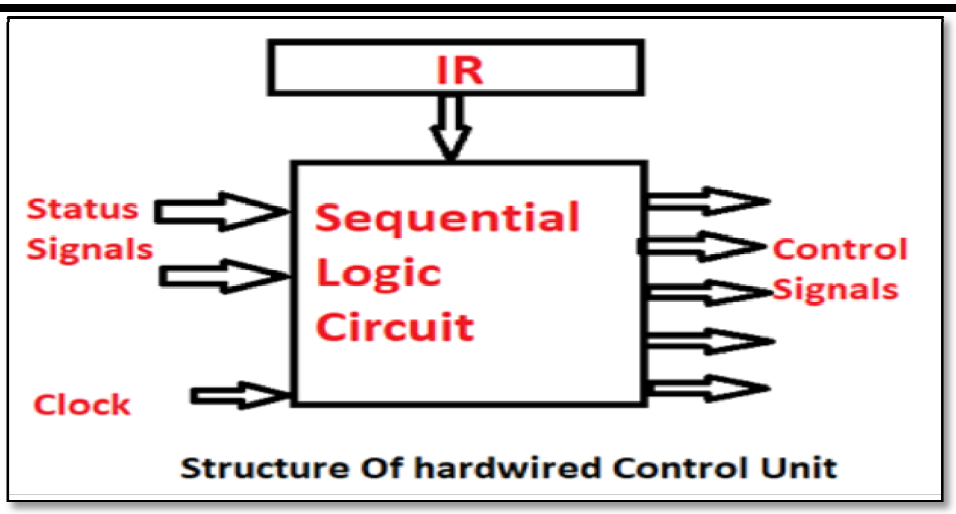
I. HARDWIRED CONTROL UNIT

The **hardwired control unit** is implemented as a sequential logic circuit or a finite state machine which will generate a specific sequence of the control signal to execute an instruction.

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals.

The hardwired control unit uses the logic to interpret the instruction and generate control signals. While designing such type of control unit various factors are considered which are:

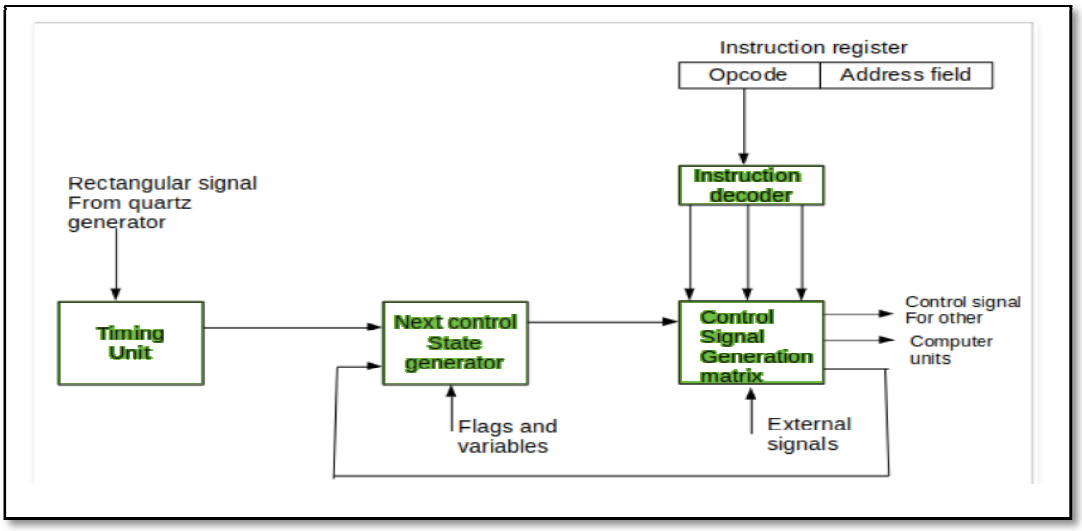
- Cost of designing
- Number of hardware used
- Speed of the operation (performance)



Generally, there are four techniques for designing a Hardwired control unit they are as follows:

1. **State table method:** It is a simple method of sequential circuit design. Its attempt is to minimize the amount of hardware.
2. **Delay element method:** It is a method which is based on the use of D flip-flop for control signal timing.
3. **Sequence counter method:** It uses the counter for the timing purposes.
4. **PLA method:** It uses the programmable logic array.

Block Diagram:



In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The **instruction decoder** constitutes a set of many decoders that decode different fields of the instruction opcode.

As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer.

This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle.

A number of signals generated by the control signal generator matrix are sent back to inputs of the **next control state generator matrix.** This matrix combines these signals with the timing signals, which are generated by the **timing unit** based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching.

Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the signals stimulates the change of the control unit state.

This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle.

The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

Advantages:

1. It is faster than the micro programmed control unit.
2. It can be optimized to produce the fast mode of operation.

Disadvantages:

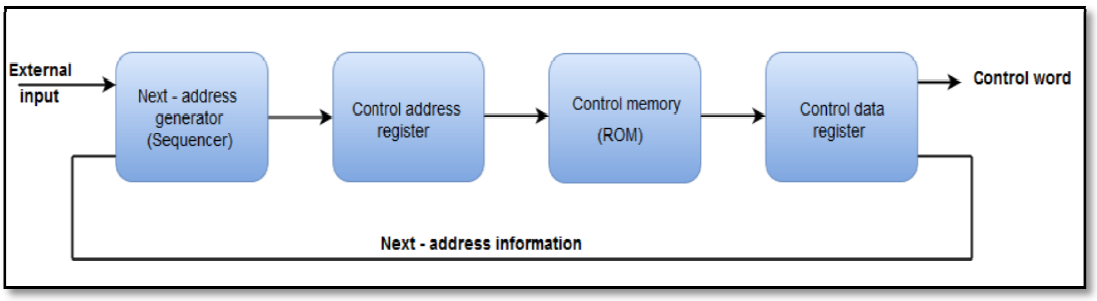
1. Instruction set, the control logic is directly implemented.
2. Requires change in wiring if the design has to be controlled.
3. An occurrence of an error is more.
4. Complex decoding and sequencing logic.
5. It requires a more chip area, therefore, it is a costlier control unit.

II. MICROPROGRAMMED CONTROL UNIT

The Micro programmed Control organization is implemented by using the programming approach.

It is a method of control unit design in which the control signal selection and sequencing information is stored in a ROM called control memory. This control signal is activated by microinstruction stored in memory.

Block diagram of a Micro programmed Control organization:



The **next address generator** is often referred to as a micro-program sequencer, as it determines the address sequence that is read from control memory.

The **Control memory address register** specifies the address of the micro-instruction.

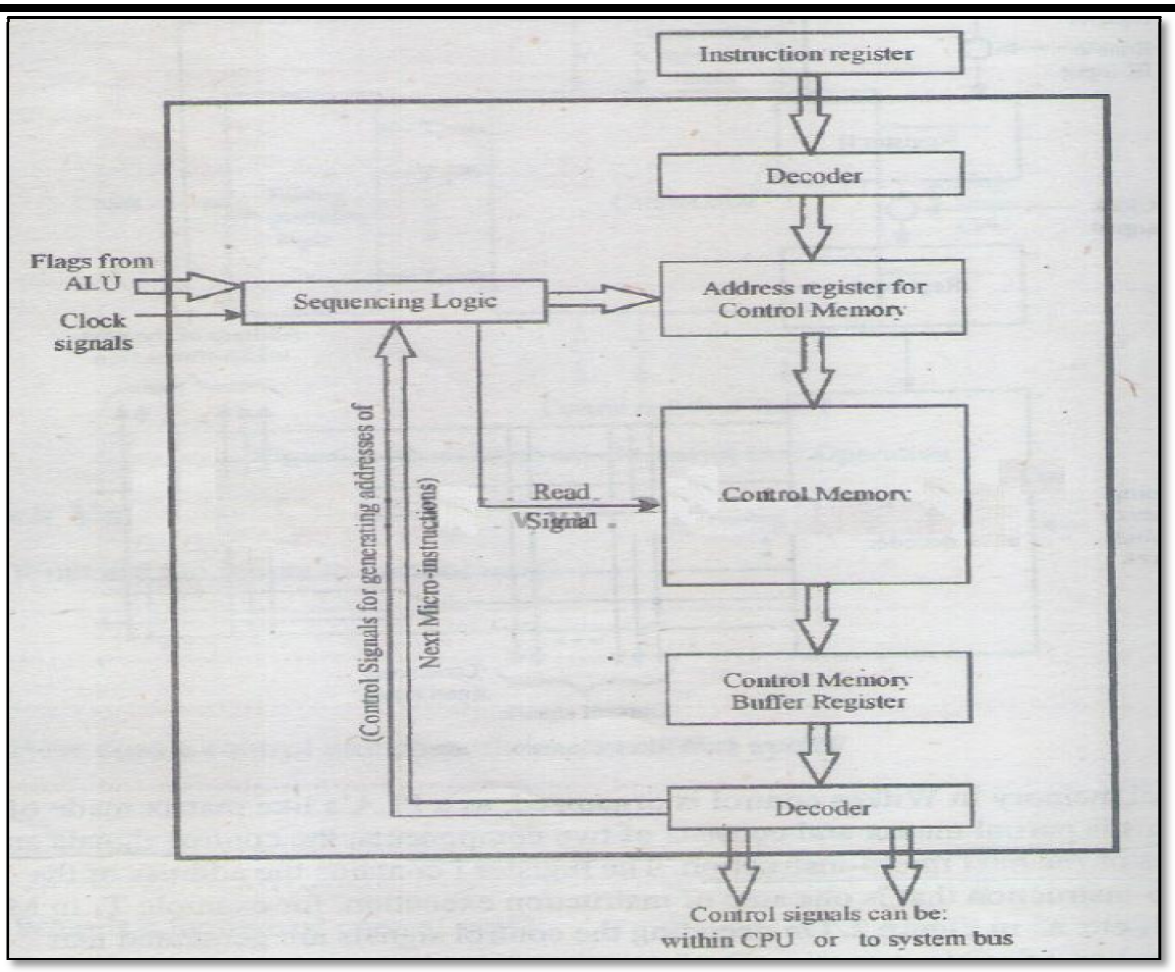
The **Control memory** is assumed to be a ROM, within which all control information is permanently stored.

The **control register** holds the microinstruction fetched from the memory.

The micro-instruction contains a control word that specifies one or more micro-operations for the data processor.

While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

Operation of micro programmed control unit:



The micro-instructions are stored in the control memory.

The address register for the control memory contains the address of the next instruction that is to be read.

The control memory Buffer Register receives the micro-instruction that has been read.

A micro-instruction execution primarily involves the generation of desired control signals and signals used to determine the next micro-instruction to be executed.

The sequencing logic section loads the control memory address register. It also issues a read command to control memory.

The following functions are performed by the micro-programmed control unit:

1. The sequence logic unit specifies the address of the control memory word that is to be read, in the Address Register of the Control Memory. It also issues the READ signal.
2. The desired control memory word is read into control memory Buffer Register.
3. The content of the control memory buffer register is decoded to create control signals and next-address information for the sequencing logic unit.
4. The sequencing logic unit finds the address of the next control word on the basis of the next-address information from the decoder and the ALU flags.

This decoder translates the opcode of the IR into a control memory address.

III. MICRO INSTRUCTIONS-TYPES

Each micro-operation is described in symbolic notation. Each line describes a set of micro-operations occurring at one time and is known as a microinstruction. A sequence of instructions is known as a microprogram, or firmware.

There are two types of microinstruction formats:

1. **Horizontal Format :**
   - ❖ **T**he control signals are represented in the decoded binary format that is 1 bit/CS.
   - ❖ Example: If 53 Control signals are present in the processor than 53 bits are required.
   - ❖ More than 1 control signal can be enabled at a time.
   - ❖ It supports longer control word.
   - ❖ It is used in parallel processing applications.
   - ❖ It allows higher degree of parallelism. If degree is n, n CS are enabled at a time.
   - ❖ It requires no additional hardware(decoders).It means it is faster than Vertical Micro programmed.
   - ❖ It is more flexible than vertical micro programmed.

2. **Vertical Format:**

- ❖ The control signals re represented in the encoded binary format.
- ❖ For N control signals- Log2(N) bits are required.
- ❖ It supports shorter control words.
- ❖ It supports easy implementation of new control signals therefore it is more flexible.
- ❖ It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
- ❖ Requires an additional hardware (decoders) to generate control signals, it implies it is slower than horizontal micro programmed.
- ❖ It is less flexible than horizontal but more flexible than that of hardwired control unit.