Instruction format-

An instruction is a command given to a computer to perform a specified operation on some given data and the format in which the instruction is specified is known as instruction format.

A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control function needed to process the instruction.

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

(1)- An operation code field that specifies the operation to be performed.

(2)- An address field that designates a memory address or a processor register.

(3)- A mode field that specifies the way the operand or the effective address is determined.

The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.

Ex- ADD R1, R0. ADD is the opcode and R1, R0 are the address field.

Operations specified by computer instructions are executed on some data stored in memory or processor registers. Operands residing in memory are specified by their memory address. Operands residing in processor registers are specified by register address. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 proce-

sssor registers R0 through R15 will have a register ~~field~~ address field of four bits. The binary number 0101, for example, will designate register R5.

computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of a computer depends on the internal organization of its registers. Most computers will fall into one of three types of CPU organizations:

(1) Single Accumulator Organization
(2) General Register Organization
(3) Stack organization.

1) Accumulator-Type Organization -

All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field, i.e., only one operand is specified in the instruction. The other operand is in accumulator. The result is placed in the accumulator. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as

ADD X AC ← AC + M[X]

Where X is the address of the operand. The ADD instruction in this case results in the operation AC ← AC + M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

2) General Register Organization -

The instruction format in this type of computer needs three register address fields. Thus the instruction for an arithmetic addition may be written as

ADD R1, R2, R3 R1 ← R2 + R3

The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Thus the instruction

ADD R1, R2 R1 ← R1 + R2

Only registers addresses need be specified in the instruction.

Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction

MOV R1, R2 R1 ← R2

The transfer type instructions need two address fields to specify the source and destination. General type organization computers employ two or three address fields in their instruction format. Each address field may specify a processor register or memory word. An instruction symbolized by

ADD R1, X R1 ← R1 + M[X]

It has two address field, one for register R1 and the other for the memory address X.

(3) Stack Organization -

Stack organized machine do not contain any accumulator or general purpose registers. Computers with stack organization would have PUSH and POP operations which require an address field. Thus the instruction

PUSH X TOP ← M[X]

will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation type instructions do not need an address field in stack organized computers. This is because the operation is performed on the two items that that are on the top of the stack. The instruction

ADD

in a stack computer consists of an operation code only with no address field. This operation has the effect of poping the two items from top of the stack, adding the numbers, and pushing the sum into the stack. There is no need to specify operands with an address field since all operands are implied to be in the stack.

Example - To illustrate the influence of the number of addresses on computer, we will evaluate the arithmetic statement

$$X = (A+B) * (C+D)$$

using zero, one, two or three address instructions. The symbi

which have been used for arithmetic operation operations. are ADD, SUB, MUL and DIV. MOV is for transfer type operations; and LOAD and STORE are for transfers to and from memory and accumulator register. It is assumed that the operands are in memory addresses A, B, C and D and the result must be stored in memory at address X.

(1) Three Address Instructions:-

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates $X = (A+B) * (C+D)$ is shown below

```
ADD  R1, A, B      R1 ← M[A] + M[B]
ADD  R2, C, D      R2 ← M[C] + M[D]
MUL  X, R1, R2     M[X] ← R1 * R2
```

It is assumed that computer has two processor register R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

(2) Two-address Instructions -

Two address instructions are the most common in commercial computers. Here again each address can specify either a processor register or a memory word.

```
MOV  R1, A     R1 ← M[A]
ADD  R1, B     R1 ← R1 + M[B]
MOV  R2, C     R2 ← M[C]
ADD  R2, D     R2 ← R2 + M[D]
MUL  R1, R2    R1 ← R1 * R2
MOV  X, R1     M[X] ← R1
```

The MOV instruction moves or transfer the operands to and from processor registers. The first symbol listed in an instruction is assumed to be a both a source and the destination where the result of the operation is transferred.

(3) One Address Instruction -

One address instruction use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations.

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC

All operations are done between the AC register and a memory operand. T is the address of temporary memory location required for storing the intermediate result

(4) Zero Address Instructions -

A stack organized computer does not use an addre field for the instructions ADD and MUL. The push and pop instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A+B) * (C+D) will be written for a stack organized computer. (TOS stands for top of stack).

PUSH	A	TOS ← A
PUSH	B	TOS ← B
ADD		TOS ← (A+B)
PUSH	C	TOS ← C
PUSH	D	TOS ← D
ADD		TOS ← (C+D)
MUL		TOS ← (C+D) * (A+B)
POP	X	M[X] ← TOS

To evaluate arithmetic operations in a stack computer, it is necessary to convert the expression into reverse polish notation. The name zero address given to this type of computer because of the absence of an address field in the computational instructions.

(A+B) * (C+D)

AB+ * CD+

AB+ CD+ *

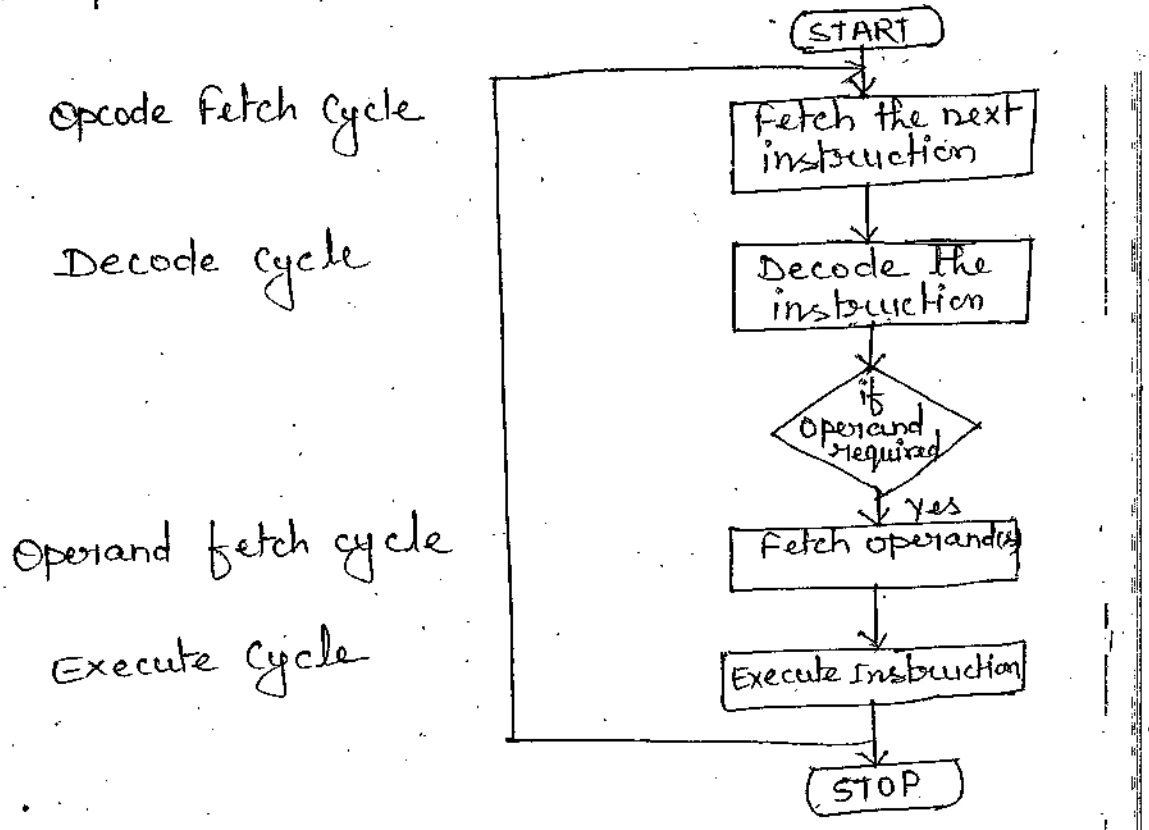
Control Design -

A processor unit is separated into two parts: Data processing unit and control unit. The data processing unit is a collection of functional units capable of performing certain operations on data, where as control unit issues control signals to the data processing part to perform operations on data. These control signals selects the functions to be performed at specific times and route the data through the appropriate functional units.

Fundamental Concepts:

The primary function of a processor unit is to execute sequence of instructions stored in a memory, which is external to processor unit. The sequence of operations involved in processing an instruction constitutes an instruction cycle, which can be subdivided into three major phases: Fetch cycle, decode cycle and execute cycle, ~~which can be subdivided~~

To perform fetch, decode and execute cycles the processor unit has to perform set of operations called microoperations.



(a) Fetch Cycle-

When a program is executed, the program counter (PC) in the CPU is set to the address of the first instruction in the program. This address is then transferred to memory address register (MAR).

The instruction from the memory is fetched and its op-code part is loaded into the instruction register (IR). The operand is placed in MAR. The op-code is then decoded by the instruction decoder to determine as to 'what is to be done': The CU initiates the required signals for the ALU and the registers to carry out the required operation. The contents of PC are automatically incremented to point to the next instruction.

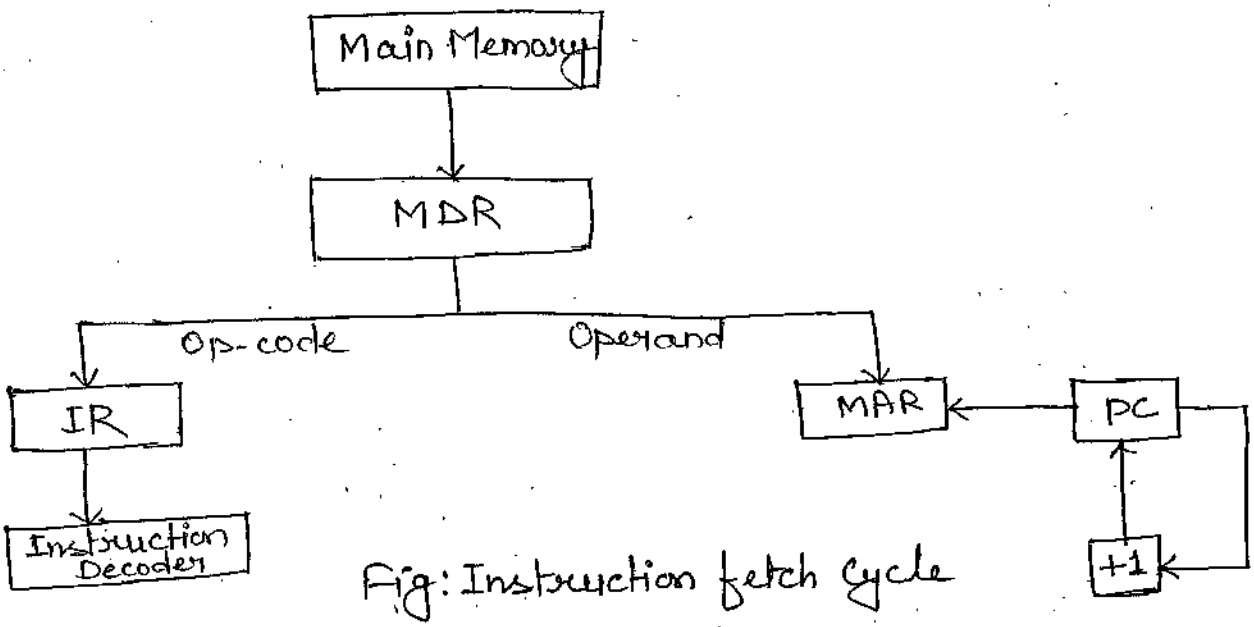


Fig: Instruction fetch cycle

(b) Execution Cycle -

Once an instruction has been fetched and decoded, the instruction execution cycle begins by transferring the required data from the address indicated by MAR. The operation specified by the op-code is then performed on this data in the ALU.

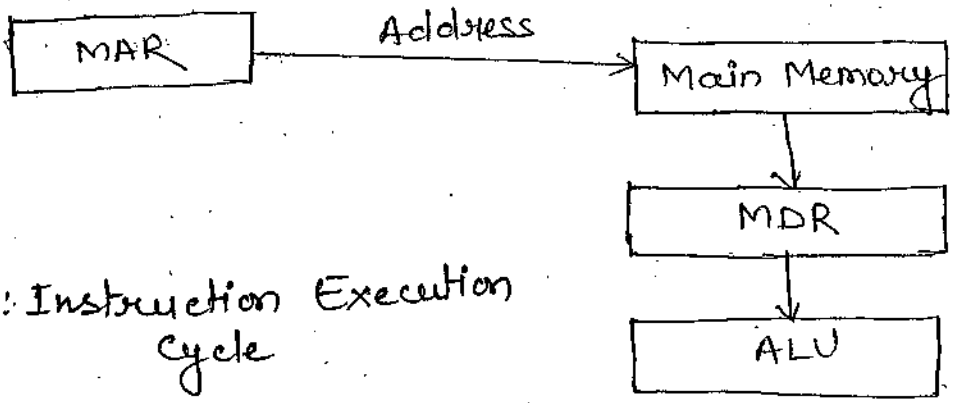


Fig: Instruction Execution Cycle

(1) Register Transfers:

"It is defined as the transfer of data between the registers through a common bus."

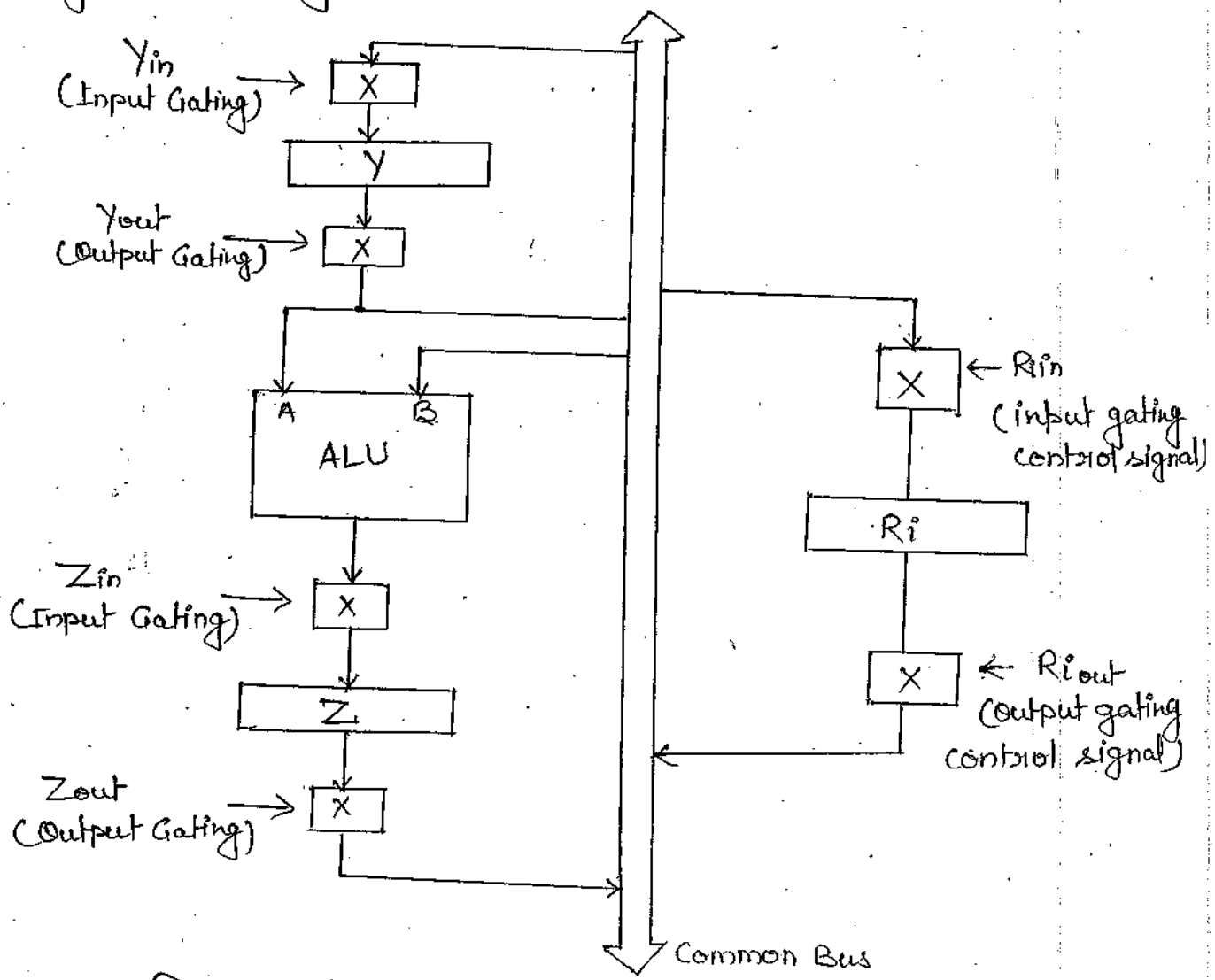


Fig: Input and Output gating for the registers

The transfer between registers and common bus is shown with arrow heads. But in actual practice, each register has input and output gating and these gates are controlled by corresponding control signals.

Performing an Arithmetic or logic Operation -

The following figure shows the internal organization of the processor. It mainly consists of a processor bus and different registers to show how they are organized and interconnected. The data and address lines of the external memory bus are connected to the processor bus via data register and address register (AR). Data can be loaded into DR from processor bus as well as memory bus. Also, the data stored in DR can be placed on either of the bus. The input of the address register is connected to the processor bus and its output is connected to the memory bus. The instruction decoder and control logic is connected to the memory bus by means of control line.

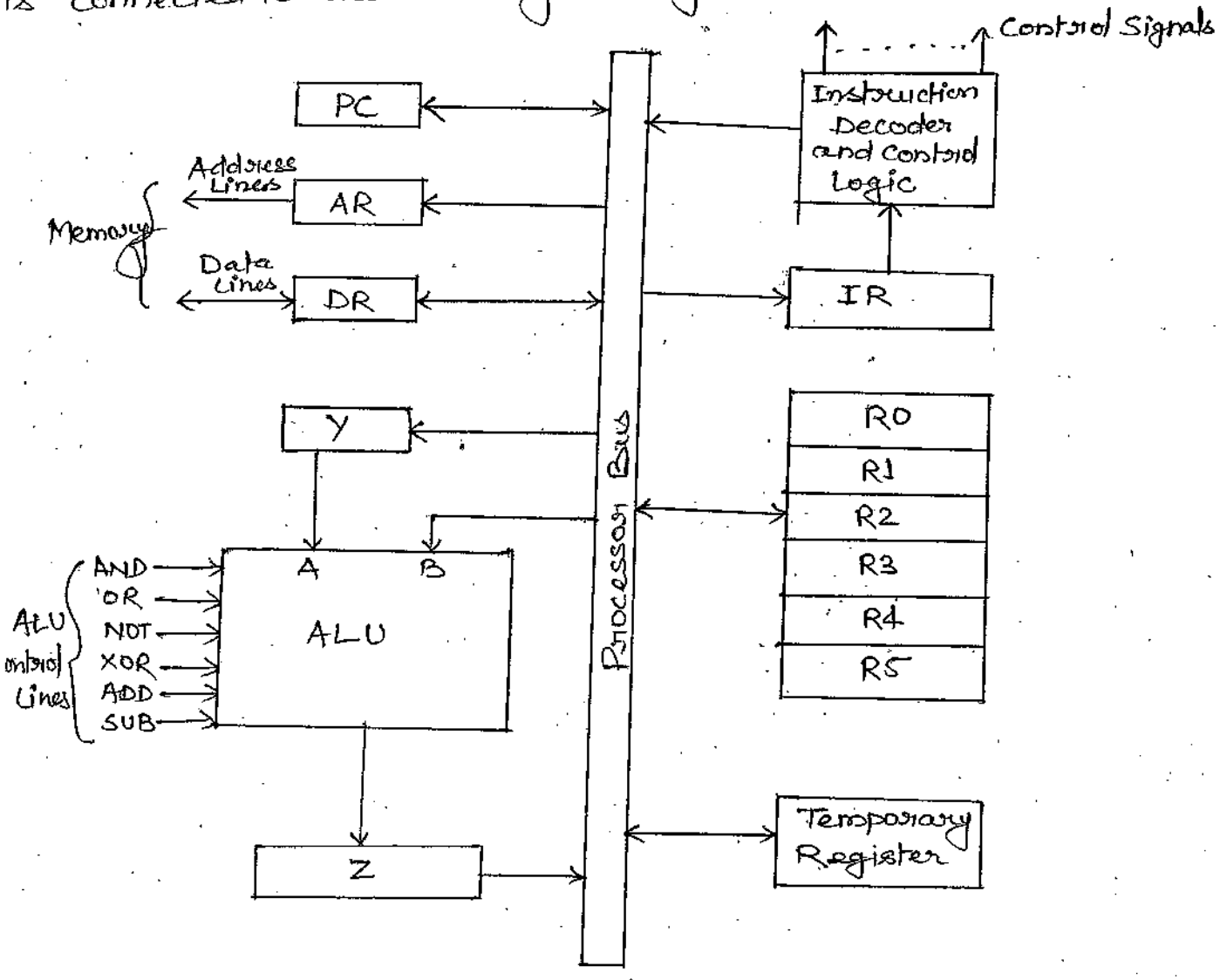


Fig: Bus Organization of CPU

Before execution, the instruction can perform one or more of the following operations:

i) a word of data is transferred from one processor register to another or to the ALU.

(ii) The result is stored in a processor register after performing an arithmetic or logic operation.

(iii) When required, the contents of a given memory location can be fetched and loaded into a processor register.

(iv) The data from the processor register can be stored into a given memory location.

Arithmetic and logic operations are performed by arithmetic and logic unit (ALU) that has no internal storage element. It performs the arithmetic or logical operations on two operands. The two inputs to the ALU is A and B and hence the two operands must be available at the two inputs of the ALU simultaneously. The result of the operation performed by ALU is stored temporarily in register Z. To add the contents of R1 to those of R2 and stored the result in R3, the addition operation can be written as

$$R3 \leftarrow R1 + R2$$

To perform this operation, the output of register R1 is enabled and the content of R1 is transferred to input A of the ALU. Similarly, the contents of R2 is transferred at input B of the ALU. The addition function performed by the ALU depends on the signals applied to its control lines. After performing the operation, the result is transferred into the register Z and then the result is transferred to the destination register R3 by enabling its input line.

Fetching a word from memory -

To fetch a word of information from memory, the CPU has to specify the memory location where that particular information is stored in the memory. The processor transfers that required address into the AR. The output of the data registe (DR) is connected to the memory bus. The control lines of the memory bus is used to indicate the memory read operation. Once the required data is transferred to the data register from memory, they can be transferred into one of the processo registers

Let the read operation be the instruction to move the contents of a memory location stored in register R0 to the register R2. The sequence of operations performed are as follows:

(1) The memory address stored in R0 is transferred to the address register AR as AR ← [R0].
2) A read operation is required to read the contents of AR.
3) Wait for control signal's response.
4) The data from memory bus is transferred into data register.
5) Finally, the data from DR is transferred to the register R2.

Storing a word in Memory-

To store a word in any memory location, the desired address of the memory location is first loaded into the address register (AR). Then, the data word to be written at the specified address by AR is loaded into the data register (DR) and the processor issued a write command. After this the data from the data register (DR) is stored at the address specified by AR.

Let the data to be stored in memory is in register R2 and the register R0 specifies the address location where the data has to be stored. The write operation requires the following sequence.

(1)- The memory address from R0 is transferred into AR i.e AR ← [R0].

(2). The data word from R2 is placed into DR as DR ← [R2] and enables the write operation signal.

(3)- The data word is then transferred at the specified address and wait for the control signal.

Hardwired Control Unit-

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired. Logic gates, flip-flops, decoders and other

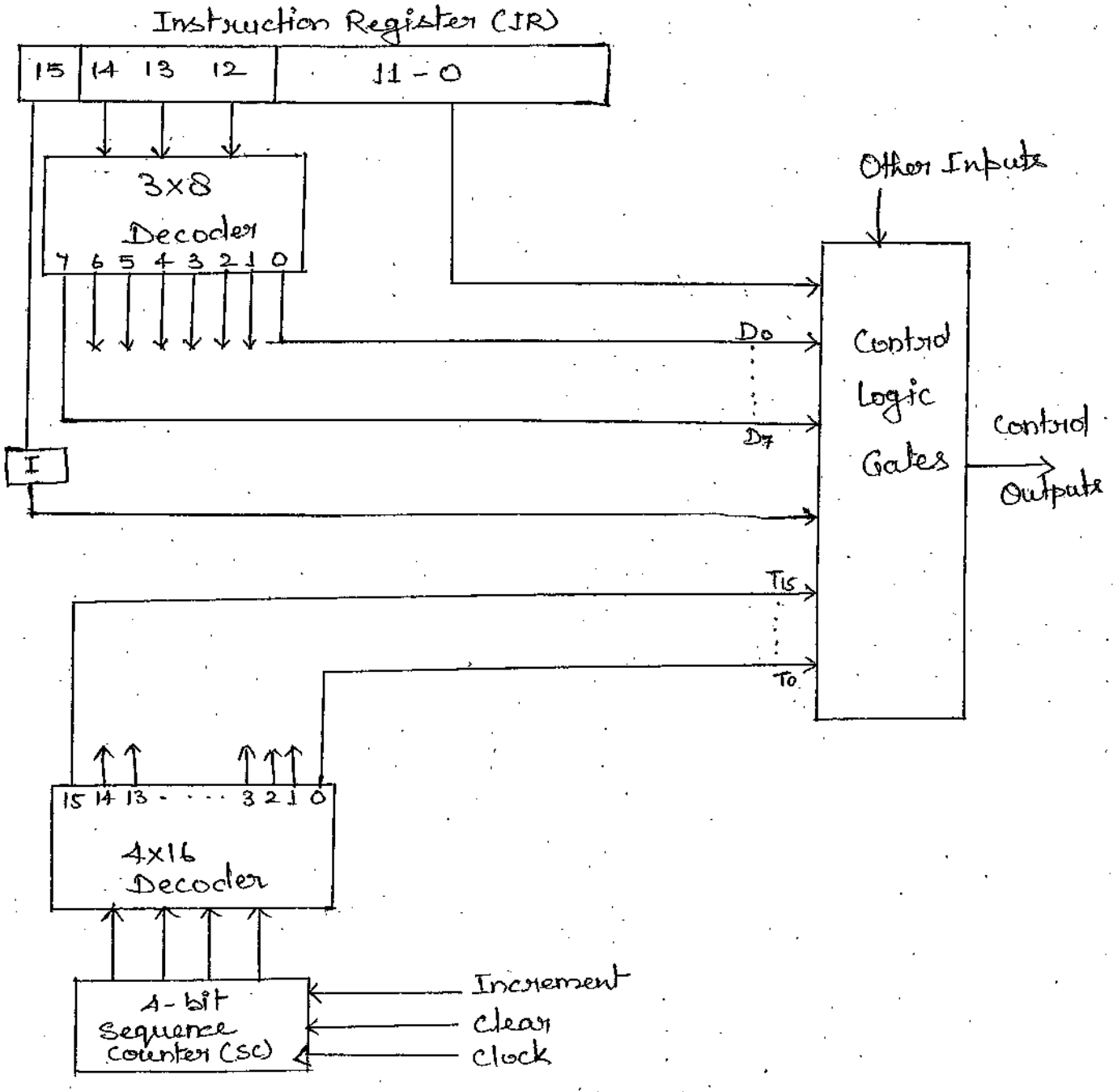


Fig: Hardwired control Unit

digital circuits are used to implement hardwired control organization. As the name suggests, if the design has to be modified or changed, a hardwired control requires changes in the wiring among the various components. The block diagram of hardwired control unit is shown in above figure.

It consists of two decoders, a sequence counter and a number of control logic gates. Instruction read from memory is placed in the Instruction register (IR) which is divided into three parts: the I-bit, the operation code and bits 0 through 11. Bit 15 of the instruction is transferred to a flip-flop designated by I. The operation code bits 12 through 14 are decoded with a 3×8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15, which are decoded into 16 timing signals T_0 through T_{15}. This sequence counter are generally incremented so that it provide the sequence of timing signals. Very rarely, the sequence counter is cleared to 0 and thus, the next active timing signal will be T_0.

Consider a case when sequence counter is incremented to provide timing signals of T_0, T_1, T_2, T_3 and T_4 in sequence and at time T_4, sequence counter SC is cleared to 0 if decoder output D_3 is active.

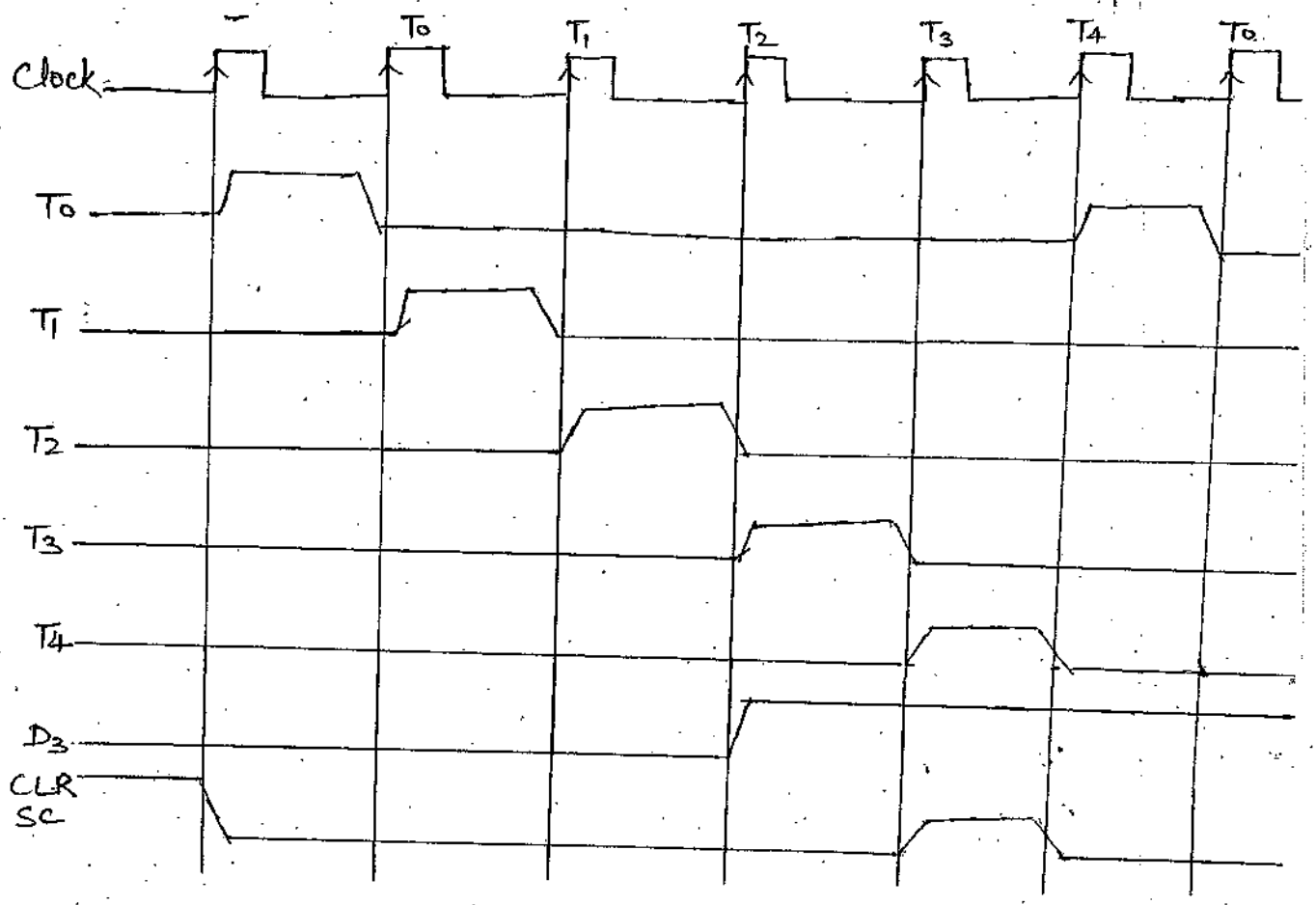


Fig: Timing Diagram

This timing diagram as shown in the figure can be symbolically expressed as

$D_3T_4: SC \leftarrow 0$

Initially the CLR input of the sequence counter is active and SC responds to positive transition of the clock, SC clears to 0 and timing signal T_0 activates and remains active during one clock cycle. SC is incremented with every positive transition unless the clear input becomes active, which produces the sequence as T_0, T_1, T_2, T_3, T_4 and so son. This timing signal will continue upto T_5 and back to T_0, if sequence counter is not cleared. The timing diagram for $D_3T_4: SC \leftarrow 0$ is shown in above diagram. The figure shows how SC is cleared when $D_3T_4 = 1$. At the end of the timing signal T_2 the output D_3 from decoder becomes active. When timing signal T_4 becomes active, the sequence counter cleared to 0 which causes the timing signal T_0 to become active instead of T_5 which would be active signal if sequence counter is incremented instead of cleared.

There are four simplified and systematic method for the design of hardwired controllers:

(i) State table method or one-hot method:
It is the standard algorithm approach to sequential circuit design.

(ii) Delay element method:→
It is a heceristic method based on the use of clocked delay elements for control signal timing.

(iii) Sequence counter Method:
It uses counters for timing purposes.

(iv) PLA (Programmable Logic Array):
It uses programmable logic array.

Microprogram control Unit-

"The control unit which generates control signals according to microprogram rather than using hardware is called a microprogrammed control unit."

Microprogramming is a second alternative for designing the control unit of a digital computer. The control function that specifies a microoperation is a binary variable. When it is in one binary state, the corresponding microoperation is executed. The control variable in opposite binary state does not change the state of the registers in the systems.

Every instruction in a processor is implemented by a sequence of one or more sets of concurrent microoperations. Each microoperation is associated with a specific set of control lines which, when activated, causes that microoperation to take place. Since the number of instructions and control lines is often in the hundreds, the complexity of hardwired control unit is very high. Thus it is very costly and difficult to design.

A computer that employs a microprogrammed control unit will have two separate memories: a control memory and a main memory. The main memory is available to the user for storing the programs. The contents of main memory often may alter when the data are manipulated and every time that the program is changed. On the other hand 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 microinstruction initiates a series of microinstructions in control memory. These microinstructions generate the microoperations to fetch the microinstruction 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 block diagram of a microprogrammed control unit is shown in the figure. The control memory is assumed to be a ROM, within which all control information is permanently stored. The control memory address register specifies the address of the microinstruction and the control data register holds the micro inst-

ruction read from memory. The microinstructions contains a control word that specifies one or more microoperations for the data processor. Once these operations are executed, the control must determine the address of next instruction. The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory. For this reason it is necessary to use some bits of the present microinst ruction to control the generation of the address of the next micro-instruction. The next address may also be a function of external input conditions. While the microoperations 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.

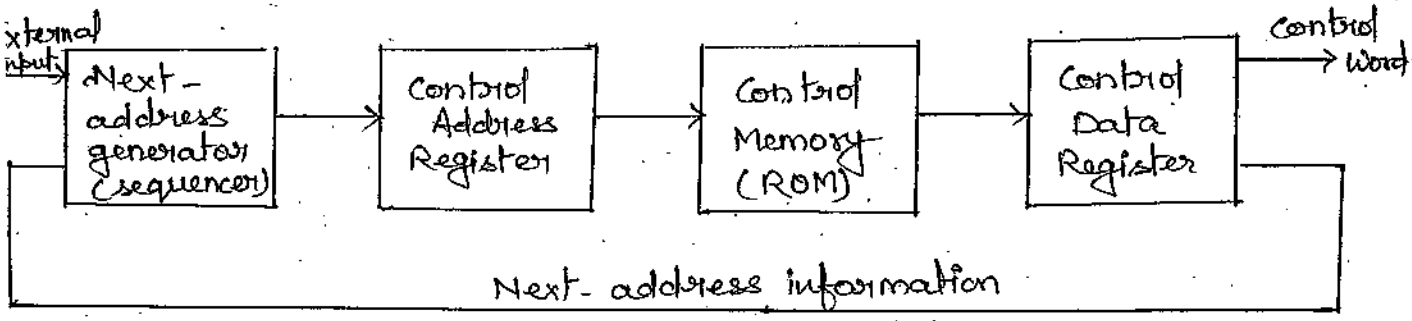


Fig: Microprogrammed control Organization

The next address generator is sometimes called a micro-program sequencer, as it determines the address sequence ~~of~~ that is read from the control memory. The address of the next micro instruction can be specified by several ways, depending on the sequencer inputs. Typical functions of a microprogram sequencer are:

(i) incrementing the control address register by one.
(ii) Loading into control address register an address from control memory.
(iii) transferring an external address.
(iv) Loading an initial address to start the control operations.

Advantage of Microprogrammed control:

(1)- It simplifies the design of control unit.

(2)- Control functions are implemented in software rather than hardware.

(3)- The design process is orderly and systematic.

(4)- More flexible, can be changed to accomodate new system specifications or to correct the design errors quickly.

(5)- Complex function such as floating point arithmetic can be realized efficiently.

Disadvantage of Microprogrammed Control:-

1. A microprogrammed control unit is somewhat slower than the hardwired control unit, because time required to access the microinstruction from control memory (CM).
2. The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.

Comparison Between Hardwired and Microprogrammed Control

Characteristics	Hardwired Control	Microprogrammed Con
Speed	Fast	Slow
Implementation	Hardware	Software
Flexibility	No flexibility	More flexibility
Design Process	Difficult	Easy
Memory	No memory used	RAM & ROM used
Chip Area Efficiency	less area	more area
Ability to support operating system	very difficult	Easy
Ability to handle large/ complex instruction sets	some what difficult	Easy

① Microinstruction -

"An instruction that control the data flow and sequencing in a processor at a more fundamental level than machine instruction is known as microinstruction."

A microinstruction is an instruction in a microprogram. It is the most elementary computer operation that can take place. For example, moving a bit from one register to another. It takes several microinstructions to carry out one machine instruction. Each word in the control memory ~~control~~ contains within it, is a microinstruction and a sequence of microinstruction constitutes a microprogram.

A microinstruction usually consists of four parts:

(i) Microoperation fields designated as F1, F2, F3.
(ii) Condition for branching (CD)
(iii) Branch field (BR)
(iv) Address field (AD)

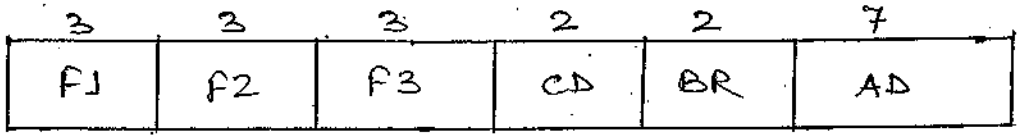


Fig: Microinstruction format (20 bits)

Figure shows the formats for 20-bit microinstruction, which is divided into four functional parts. The fields F1, F2, and F3, each of 3-bits specify microoperations to be performed. The three bits of each field are encoded to specify seven distinct microoperations, which gives a total of 21 microoperations. If the control word needs to specify only one ~~microinstruction~~ microoperation, then other two fields of microoperations have the binary value 000. For example, let us consider a microinstruction that can specify two microoperations from F1 and F3 and none from F2 as:

$AC \leftarrow AC + DR$ with F1 = 001

and $PC \leftarrow AR$ with F3 = 110

Thus, the nine bits of the microoperations field will be 001000110. Also two or more conflicting microoperations cannot be specified simultaneously. For example, a microoperation field 010 001 000 specifies the operation to clear AC to 0 and subtract the content of DR from AC at the same time, and hence no meaning.

A symbolic notation is assigned to each microoperations. All transfer-type microoperation use five letters, where last two letters specifies the destination register and the third letter is T always.

Symbols and Binary code for Microinstruction field

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	DRTAR
110	AR ← PC	PCTAR
111	M[AR] ← DR	WRITE

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

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

CD	Condition
00	Always = 1
01	DR (15)
10	A (15)
11	AC = 0

Symbol	Comments
U	Unconditional Branch
I	Indirect Address Bit
S	Sign bit of AC
Z	Zero value in AC

BR	Symbol	function
00	JMP	CAR ← AD if condition = 1 CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1 CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR (2-5) ← DR (11-14), CAR (0,1,6) ← 0

Applications of Microprogramming:

The various applications of microprogramming are:

(1)- Realization of Computers
(2)- Emulation
(3)- Operating System Support
(4)- Microdiagnostics
(5)- User tailoring

Techniques for Grouping of control Signals:-

The grouping of control signals can be done either by using technique called vertical organization or by using technique called horizontal organization. Highly encoded scheme that use compact codes to specify only a small number of control functions in each microinstructions are referred to as vertical organization. On the other hand, the minimally encoded scheme, in which resources can be controlled with a single instruction is called a horizontal organization.

The comparison between horizontal and vertical organization:

S.No.	Horizontal	Vertical
1.	Long format	Short format
2.	Ability to express a high degree of parallelism	Limited ability to express parallel microoperations
3.	Little encoding of the control information	Considerable encoding of the control information.
4.	Useful when higher operating speed is desired	Slower operating speed

The advantage and disadvantage of horizontal and vertical organization can be summarized as follows:

1. The horizontal organization approach is suitable when operating speed of computer is a critical factor and where the machine structure allows parallel usage of a number of resources.
2. Vertical approach results in slower operational speed but less bits are required in the microinstruction.
3. In vertical approach the significant factor is the reduced requirement for the parallel hardware required to handle the execution of microinstructions.

Microprogram Sequencer:-

"The circuit that selects the address of next microinstruction in microprogrammed control unit is known as microprogram sequencer."

The basic components of a microprogrammed control unit are the control unit and the circuit that select the next address. The address selection part is called a microprogram sequencer. The purpose of the 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

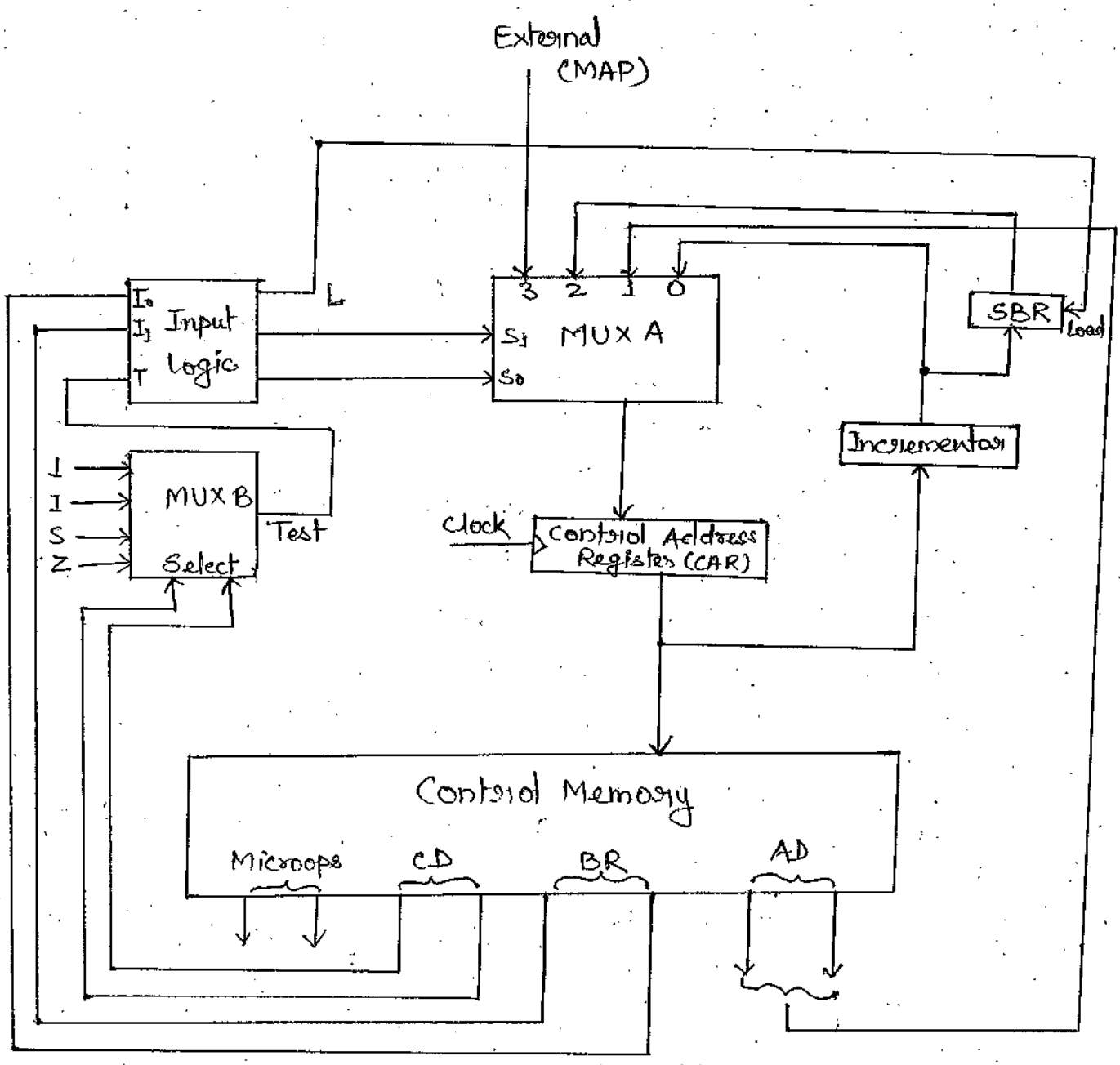


Fig: Block diagram of Microprogram Sequencer

register. The choice of the address source is guided by the next-address information bits that the sequencer receives from the present microinstruction.

The block diagram of the microprogram sequencer is shown in above figure. 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 four sources and routes into a control address register CAR. 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 content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine register (SBR). The other three inputs to the multiplexer number 4 come from the address field of the present microinstruction from the output of SBR, and from an external source that maps the instruction. Although, the diagram shows a single subroutine register, a typical sequencer will have four to eight levels deep. In this way, a number of subroutine can be active at the same time. A push and pop operation, in conjunction with a stack pointer, stores and retrieves the return address during the call and return microinstructions.

The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer. If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise it is equal to 0. The T value together with the two bits from the BR (branch) field go to an input logic circuit. The input logic into a particular sequencer will determine the type of operations that are available in the unit. Typical sequencer operations are: increment, branch, conjunction call and return from subroutine, load an external address, push or pop the stack and, and other address sequencing operations that are available in the unit. With three inputs, the sequencer can provide upto eight address sequencing operations.

The input logic circuit has three inputs, I_0, I_1 and T, and three outputs S_0, S_1 and L, Variables S_0 and S_1 select one of the source addresses of CAR. Variable L enables the load input in SBR. The binary values in the two selection variables determine the path in the multiplexer. For example, with $S_1 S_0 = 10$, multiplexer input number B is selected and establishes a transfer path from SBR to CAR. Each of the four inputs as well as the output of MUX A contains a 7-bit address.

The truth table for the input logic circuit is shown in the table. Inputs I_1 and I_0 are identical to the bit values in the BR field. The bit values for S_1 and S_0 are determined from the state function and the path in the multiplexer that establishes the required transfer. The subroutine register is loaded with the incremented value of CAR during a CALL microinstruction (BR = 01) provided that the status bit condition is satisfied (T = 1). The truth table can be

used to obtain the simplified boolean functions for the input logic circuit.

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I_1' T$$
$$L = I_1' I_0 T$$

BR field		Input			MUX A		Load SBR
		I_1	I_0	T	S_1	S_0	L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	x	1	0	0
1	1	1	1	x	1	1	0

Table: Input logic truth table for Microprogram Sequencer

3) Wide-Branch Addressing -

As the number of branches increases, generating address for each branch becomes difficult. To handle this type of situations, the best and the simple way is to use programmable logic Array (PLA) to generate the required branch address. and this way of generating the branch address is known as wide branch addressing. In this method, the op-code of the instruction is translated into the starting address of the corresponding micro-routine. The opcode bits of the instruction register (IR) is connected as inputs to PLA, hence PLA acts as a decoder and outputs the address of the desired microroutine.

4) Prefetching Microinstructions:

The disadvantage of the microprogrammed control is that it leads to slower operating speed because of the time it takes to fetch the microinstructions from the control memory. This problem can be removed and faster operation can be achieved if the next microinstruction is prefetched while other microinstruction is being executed. Thus the execution can be overlapped with the fetch time.

Sometimes, the address of the next microinstruction is determined from the status flags and from the result of the currently executed microinstruction. In such cases prefetching of microinstruction occasionally prefetches a wrong microinstruction. So, in such cases, the fetch must be repeated with the correct address, which requires more complex hardware. Even though, this difficulty the prefetching technique is oftenly used to increase instruction execution speed.

(5) Microinstruction with Next-Address field:

The microprogram requires several microinstructio These microinstructions perform no useful operation in the data path. They are needed only to determine the address of next microinstruction. The microinstruction in control memory contain a set of bits to initiate microoperations in computer registers an other bits to specify the method by which the next address obtain The control address register receives the address from four sources (status flags, instruction register, condition codes, and next address). The branching is achieved by specifying the branch address in one of the fields of the microinstruction to select a specifying status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic circuit. The return address from a subroutine is stored in a special register whose value is then used when the microprogram wishes to return from the subroutine.

The next address bits are feed through the OR gates to the micro address register (uAR), so that the address can be modified on the basis of the data in the IR, status flags and condition codes. The decoding circuit is used to generate the starting address of a given microroutine on the basis of the opcode in the IR.

x——x——>

Processor Organization -

The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU. The CPU is made up of three major parts as shown in the figure.

(i) Register set: stores intermediate data used during the execution of the instructions.

(ii) Arithmetic Logic Unit (ALU): performs the required microoperations for executing the instructions.

(iii) Control Unit: supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

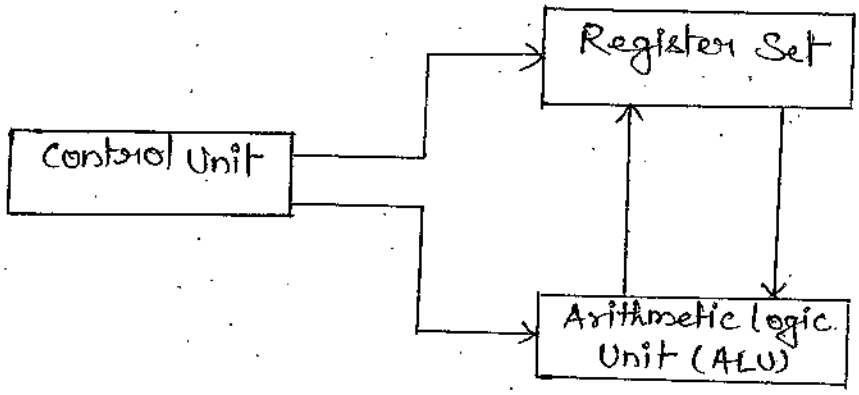


Fig: Major components of CPU

The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer. Computer architecture is sometimes defined as computer structure and behaviour as seen by the programmer that uses machine language instructions. This includes the instruction formats, addressing modes, the instruction set, and the general organization of the CPU registers.

From the designer's point of view, the computer instruction set provides the specifications for the design of the CPU. The design of the CPU is a task that in large part choosing the hardware for implementing the machine instructions.

Components of CPU -

Some of the general components upon which the processor is built are as follows:

1) Buses: In order to communicate with memory, a processor needs three types of connections data, address and control. The data lines are used to send or to receive data from memory. There is an individual connection or wire for each bit of data.

The address lines are controlled by the processor and are used to specify which memory location the processor wants to communicate with. The address is an unsigned binary integer that identifies a unique location where data elements are to be stored or retrieved.

The control lines consists of the signals that manages the transfer of data.

By using a bus, the processor can communicate with exactly one device at a time even though it is physically connected to many devices. If only one device on the bus is enable at a time, the processor can perform a successful data transfer. If two devices are tried to drive the data lines simultaneously, the result will be data lost called bus connection.

2) Registers: The register is a storage device device that is used to store words. The registers are used to transfer data and are also used for some microoperations. The registers' groups is some times called scratch pad memory.

Different CPU registers are:

i) Accumulator: - A processor register (AC) is required for doing operations on data. This registers holds data on which addition, subtraction, shift and logical operations are to be carried out. The result of an arithmetic and logical operations are automatically stored in the accumulator.

ii) Program Counter (PC) - It deals with the order of the execution of instructions. It holds the address of the next instruction to be executed. Thus, it acts as a pointer which points to the memory location where the next instruction is stored.

(iii) Temporary Register (TR): - A register used for holding temporary data generated during processing.

(iv) Instruction Register (IR) - A register used for storing instructions is called instruction register. The instruction read from the memory is to be placed in some register known as instruction register.

(v) Data Register (DR) - Register used to hold data (operand) read from memory.

(vi) Address Register (AR) - Register used to hold ~~data~~ the address of memory word.

(vii) INPR: - Input register will hold / receives data from an input device.

(viii) OUTR: - It holds the data that need to be sent to output devices

(3) Flags: - There are number of indicators known as flags that shows the processor's status. Most of these flags represent the results of last operations. For example, the addition of two numbers might produce a negative sign, an overflow, a carry, or a value of zero.

These flags are represented by a single bit such as if the result of an addition is negative, the sign flag would set to 1. If the result was not a negative number (zero, or greater than zero), the sign flag would equal to 0.

These flags are organized into a single register called the flag register or the processor status register. Since the value stored in these flags are in the form of bit which are an outcome of an arithmetic or logical operation, and hence the flag registers are connected to mathematical unit of a processor.

(4) Stacks: - During the execution of operation, there are number of times when the processor needs to use a temporary memory to store different data values so that they can be used again when required. Every processor has a finite number of registers but if an application needs more registers than available, the register's values that are not needed immediately by processor processor can be stored in the temporary memory. Also, when

a processor needs to jump to a subroutine or function, it needs to remember the instruction from where it jumped so that it can return back to the same place when the subroutine is completed. Hence, the return address must be stored and this return address are generally stored in this temporary memory. This temporary memory is known as stack.

The stack is a block of memory locations reserved to function as temporary memory. When a processor puts a piece of data, on the top of the stack, the data below it cannot be removed until the data above it is removed. This type of memory location is referred as "Last-In-First-Out" or LIFO.

5) I/O Ports: Input/Output ports, referred as I/O ports, is any connections that exists between the processor and its external devices. Some I/O devices are directly connected to the memory bus. Sending data to the port is done by storing data to a memory address and retrieving data from port is done by reading from a memory address.

Single Accumulator Organization -

Earlier when the hardware were very expensive, the processor was designed usually as a single address machine with accumulator. The accumulator is used to store operand for the instructions. It is used for storing result and for doing operations on the content of the accumulator. Thus, accumulator used as source for the operands as well as the destination for storing the result of the operations performed on its content. Thus, it accumulate datas. The accumulator based organization system results in reduced size of instruction, since the instruction contains only one operand address and the other operand is in accumulator itself. After performing the operations the result is stored in the accumulator. Thus, these accumulator based organization systems are also known as 1-address machine, since only one address is specified in the instruction. But there can be only one or limited numbers of accumulators in the CPU that affects the performance of the systems when an arithmetic expression is to be evaluated containing many different terms.

General Register Organization -

Memory locations are needed for storing pointers, counter return addresses, temporary results, and partial products during multiplications. Having to refer to memory locations for such applications is time consuming because memory access is the most time consuming operation in a computer. It is more convenient and more efficient to store these intermediate values in processor registers. When large number of CPU registers are included in the CPU. It is most efficient to connect them through a common bus system.

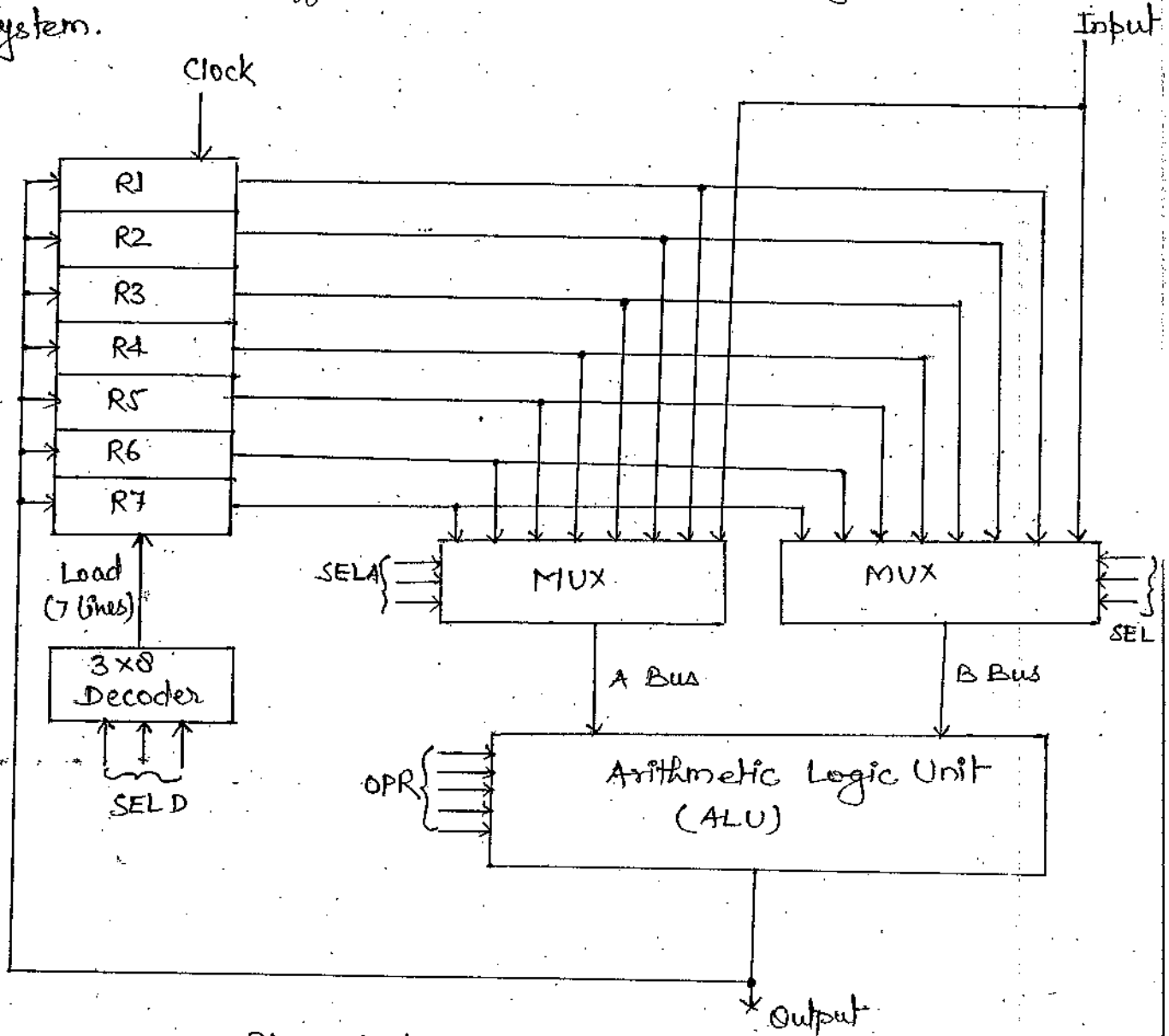


Fig: a. Block Diagram

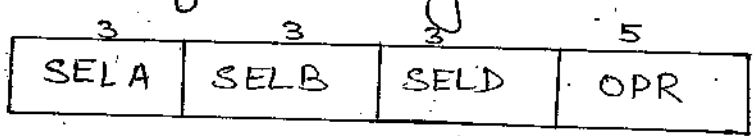


Fig: b. Control Word

Fig: Register set with Common ALU

A bus organization for seven CPU registers is shown in the figure. The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection lines in each multiplexers select one register or the input data for the particular bus. The A and B buses from the inputs to a common arithmetic logic unit (ALU). The operation selected in ALU determines the arithmetic or logic microoperation that is to be performed. This result of the microoperation is available for output data and also goes into the inputs of all registers. The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination registers.

The control unit that operates the cpu bus system directs the information flow through the registers and ALU by selecting the various components in the system, for example, to perform the operation

$$R1 \leftarrow R2 + R3$$

the control must provide binary selection variable to the following selector inputs:

(1)- MUX A selector (SELA): to place the contents of R2 in bus A.

(2)- MUX B selector (SELB): to place the contents of R3 into bus B.

(3)- ALU operation selector (OPR): to provide the arithmetic addition A+B.

(4)- Decoder destination selector (SELD): to transfer the content of output bus into R1.

The four control selection variables are generated in the control unit and must be available at the beginning of a clock cycle. The data from the two source registers propagate through the gates in the multiplexers and the ALU, to the output bus, and into the inputs of the destination register, all during the clock cycle interval. Then when the next clock transition occurs, the binary information from the output bus is transferred into R1.

Control Word:

There are 14 binary selection inputs in the unit and their combined value specifies a control word. The control word consists of 4 fields. Three fields contains 3 bits each, and one field has five bits. The three bits of SELA select a source register for the A input of the ALU. The three bits of SELB select a register for the B input of the ALU. The three bits of SELD select a destination register using the decoder and its seven load outputs. The five bits of OPR select one of the operations in the ALU. The 14-bit control word when applied to the selection inputs specify a particular microoperation.

The encoding of the register selection is specified in the following table.

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Table: Encoding of Register Selection fields

When SELA or SELB is 000, the corresponding multiplexer selects the external input data. When SELD = 000, no destination register is selected and the content of the output bus are available in the external output.

OPR	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Addition	ADD
00101	Subtraction	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift Left A	SHLA

Table: Encoding of ALU Operations

The OPR field has five bits and each operation is designated with a symbol name.

Let the microoperation given by the statement is

$$R1 \leftarrow R4 \wedge R5$$

This statement specifies R4 for the input A of ALU, R5 for the B input of ALU and R1 as the destination register. The microoperation to be performed is AND operation between R4 and R5. The control word for the above statement will be as follows

SELA	SELB	SELD	OPR
R4	R5	R1	AND
100	101	001	01000

Thus, the control word is 100101 001 01000.

Stack Organization -

"A stack is an ordered collection of items which permits the insertion or deletion of an item to occur only at one end."

The stack is also known as last-in, first-out (LIFO) list. A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.

The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it). The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.

The two operations of stacks are the insertion and deletion of items. The operation of insertion is called push operation because it can be thought of as the result of pushing a new item on top. The operation of deletion is called pop because it can be thought as the result of removing an item so that the stack pop-ups. However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

Register Stack -

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words. The figure shows the organization of a 64-word register stack. The stack pointer contains a binary number whose value is equal to the address of the word that is currently on the top of the stack. Three items are placed in the stack: A, B, and C. Item C is on top of the stack. so that the contents of SP is now 3. To remove the top item the stack is popped by reading the memory word at address 3, and decrementing the contents of SP. Item B on top of stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next higher location in the stack.

* Note that the item C has been read out but not physically remove

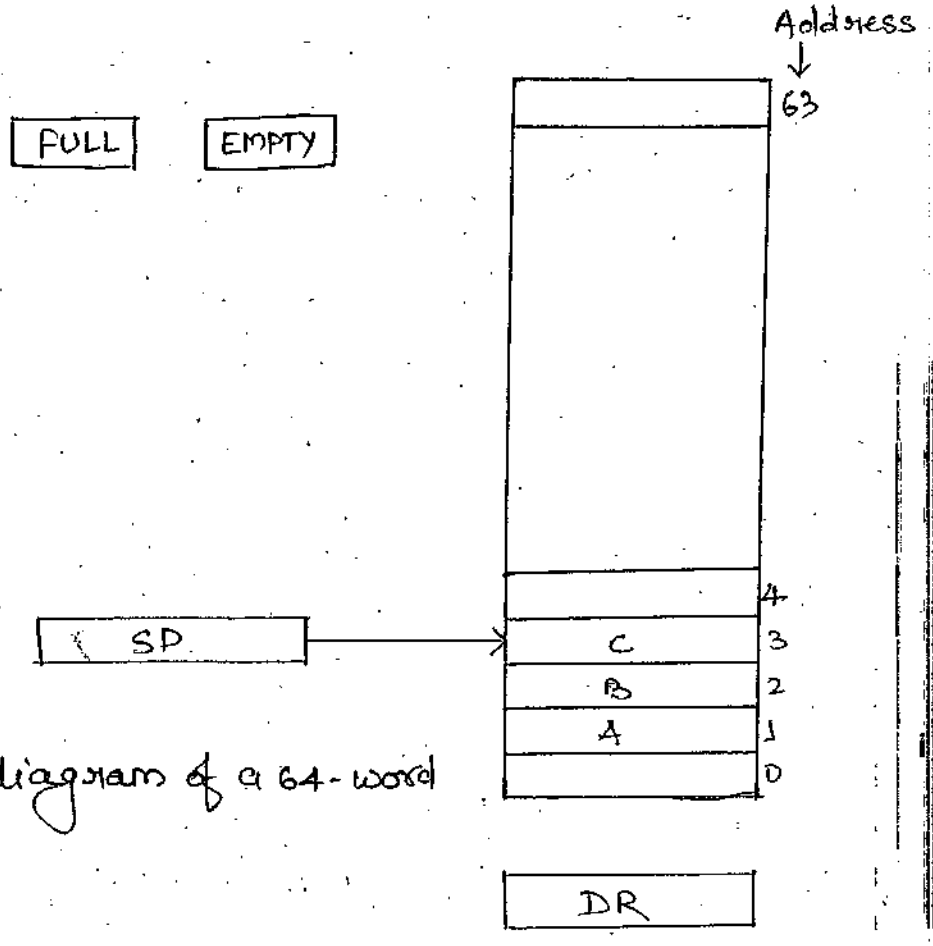


Fig: Block diagram of a 64-word stack

In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$. Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremente by 1, the result is 0 since 111111 + 1 = 1000000 in binary but SP can accomodate only the six least significant bits. Similarly when

000000 is decremented by 1, the result is 111111. The one-bit register FULL is set to 1 when the stack is full, and one-bit register EMPTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written into or read out of the stack.

Initially, SP is cleared to 0, EMPTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack pointer points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

```
SP ← SP + 1          Increment stack pointer
M[SP] ← DR           Write item on top of stack
if (SP=0) then (FULL←1)  Check if stack is full.
EMPTY ← 0            Mark the stack not empty.
```

The stack pointer is incremented so that it points to the address of the next higher word. A memory write operation inserts the word from DR from DR into the top of the stack. The first item stored in the stack is at address 1. The last item is stored at address 0. If SP reaches to zero, the stack is full of items, so ~~that~~ FULL is set to 1. This condition is reached, if top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in 0. Once an item is stored in 0, there are no more empty registers in the stack. If an item is written back in the stack, obviously the stack cannot be empty, so EMPTY is cleared to 0.

A new item is deleted from the stack if the stack is not empty (if EMPTY = 0). The pop operation consists of the following sequence of microoperations:

```
DR ← M[SP]           Read item from the top of the stack
SP ← SP - 1          Decrement stack pointer
if (SP=0) then (EMPTY=1)  Check if stack is empty.
FULL ← 0             Mark the stack not full.
```

The top item is read from the stack into DR. The SP is then decremented. If its value reaches zero, the stack is empty.

so EMPTY is set to 1. This condition is reached if the item read was in location 1. Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.

Memory Stack -

A stack can also be implemented in a random-access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. The figure shows a portion of computer memory partitioned into three segments: program, data and stack. The program counter (PC) points at the memory location of the next instruction in the program. The address register (AR) points at an array of data. The stack pointer (SP) points at the top of the stack. The three registers are connected to a common addressing bus, either one can provide an address for memory. PC is used during the fetch phase to read an instruction. SP is used to push and pop items into or from the stack. AR is used during the execute phase to read an operand.

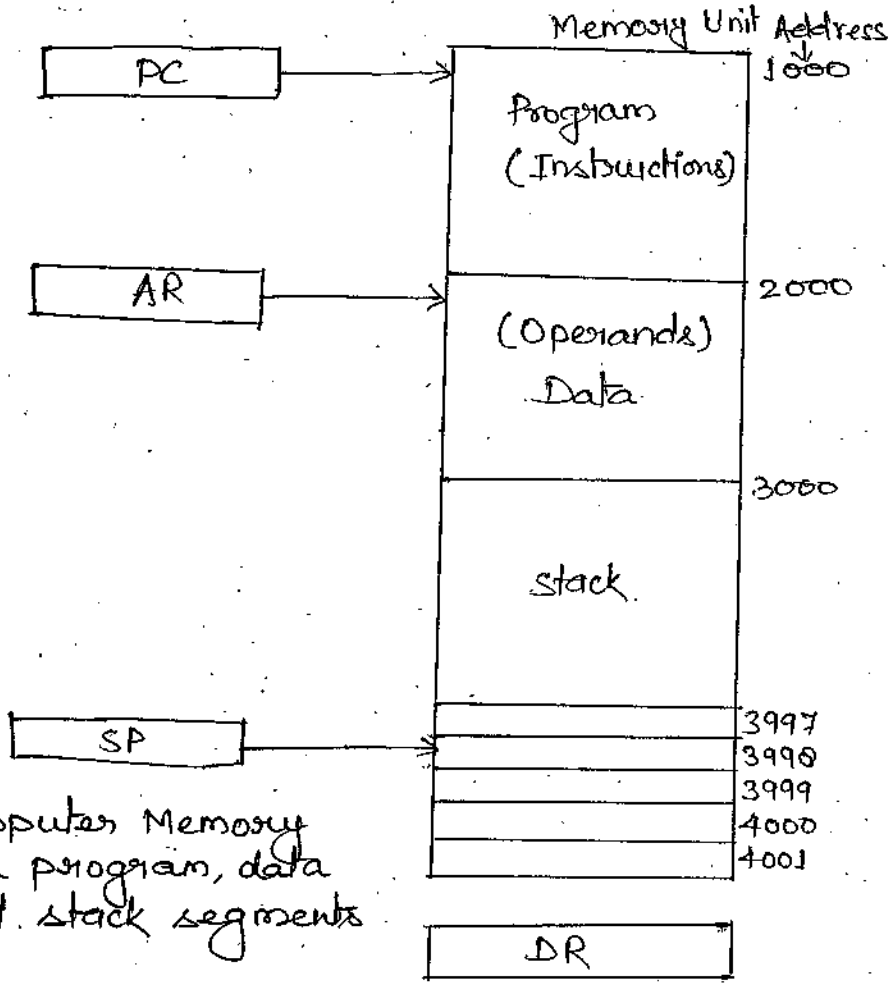


Fig: Computer Memory with program, data and stack segments

The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000. No provisions are available for the stack limits checks.

It is assumed that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows:

$$SP \leftarrow SP - 1$$
$$M[SP] \leftarrow DR$$

The stack pointer is decremented so that it points the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with a pop operation as follows:

$$DR \leftarrow M[SP]$$
$$SP \leftarrow SP + 1$$

The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.

Most computer do not provide hardware to check for stack overflow (full stack) or underflow (empty stack). The stack limits can be checked by using two processor registers: one to hold the upper limit (3000 in this case) and the other to hold the lower limit (4001 in this case). After a push operation, SP is compared with the upper limit register, and after a pop-operation SP is compared with the lower limit register.

The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP.

Reverse Polish Notation -

A stack organization is very effective for evaluating arithmetic expressions. The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer. The common arithmetic expressions are written in infix notation, when each operator written between the operands. Consider the simple arithmetic expression

$$A * B + C * D$$

The star (denoting multiplication is placed between two operands A and B or C and D. The plus is between the two products. To evaluate this arithmetic expression, it is necessary to evaluate the product A*B, store this product while computing C*D, and the sum the two products. Such a notation is known as infix notation. If the operator is placed before the two operands as +xy, the notation is said to be prefix notation, also known as polish notation. If the operator is placed after the two operands as xy+, the notation is said to be postfix notation, also known as Reverse Polish Notation (RPN). Thus the three notations are

A + B	Infix notation
+ AB	Prefix notation
AB +	Postfix Notation

For stack manipulation, the reverse polish notation is best suited. The reverse polish notation for the expression A*B + C*D is AB* CD*+.

Conversion to Reverse Polish Notation -

The conversion from infix notation to reverse polish notation must take into consideration the operational heirarchy adopted for infix notation. This heirarchy dictates that we first perfor all inner parenthesis, then inside outer parenthesis, and do multiplication and division before addition and subtraction operations. Consider the expression.

$$(A+B) * [C*(D+E)+F]$$

To evaluate the expression we must first perform the arithmetic inside the parenthesis (A+B) and (D+E). Next we calculate the expression inside the square brackets. The multiplication of [C*(D+E must be done prior to the addition of F since multiplication has precedence over addition. The last operation is the multiplication of two terms between the parenthesis and brackets. The expression can be converted to reverse polish notation without the use of parenthesis, by taking into consideration the operation heirarchy. The converted expression is:

AB+ DE+C* F+ *

AB+ CDE+* F+*

Ex- A * B + A * (B*D + C*E) → AB* ABD*CE*+*+

Evaluation of Arithmetic Expressions:

Consider an expression A*B + C*D in infix notation. Its reverse polish notation is AB* CD*+. The postfix expression will be evaluated as follows: Scan the ~~operation~~ expression from left to right. Whenever an operator is found, perform the operation with the two operands on the left side of the operator. Remove the operator and two operands and replace them by the result obtained by performing that operation. Continue in the same manner and repeat the procedure for every operator found until there are no more operators.

Thus, for the reverse polish notation AB* CD*+ first we find the operator * and the two operands to the left of * are A and B. Thus we perform A*B and replace A, B and * by the product, we get

(A*B) CD*+

The next operator is * and the two operands to the left of * are C and D. Thus, we perform C*D, and replace C, D and * by the product, we get

(A*B) (C*D) +

The next operator is + and the two operands to the left of + are the two products (A*B) and (C*D), hence the result obtained is

A*B + C*D

To illustrate this consider the expression ~~(A+B~~ (3*4) + (5*6). In reverse polish notation, the expression is "34* 56* +". The stack operation is shown in figure. The arrow(→) points to the top of the stack.

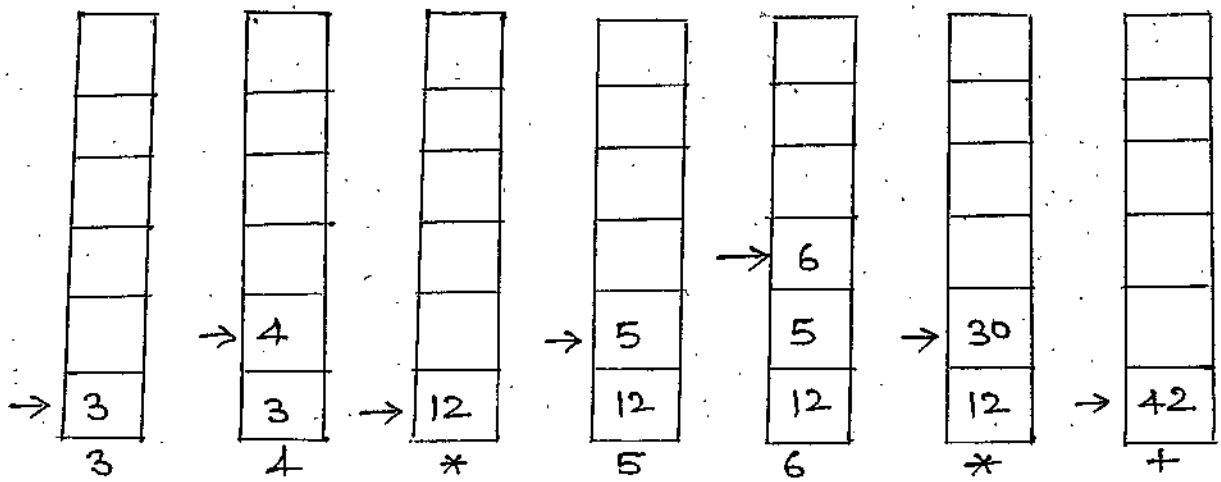


Fig: Stack Operation to evaluate 3*4 + 5*6.

Addressing Modes:

"The techniques for specifying the address of the operand are known as addressing mode."

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way that operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting and or modifying the address-field of the instruction before the operand is actually referenced. Computers use addressing mode techniques for the purpose of accomodating one or both of the following provisions:

(i) To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.

(ii) To reduce the number of bits in the addressing field of instruction

The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

(i) Fetch the instruction from memory.
(ii) Decode the instruction.
(iii) Execute the instruction.

There is one register in the computer called the program counter (PC) that keeps track of the instructions stored in memory. PC holds the address of instruction to be executed next and is incremented each time an instruction is fetched from memory. The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction and, the location of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.

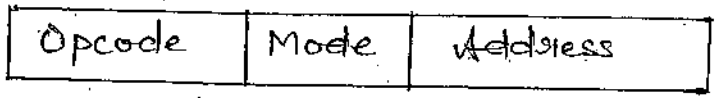


Fig: Instruction format with field

Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all. These are implied mode and immediate mode.

(1) Implied Mode -

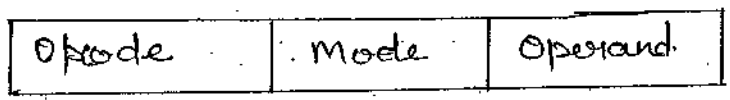
In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of instruction. In fact, all register reference instructions that use an accumulator are implied mode instructions.

Ex. Shift Microinstructions, zero address instructions in a ~~in~~stack-organized computer since the operands are implied ~~on~~ to be on top of the stack.

(2) Immediate Mode -

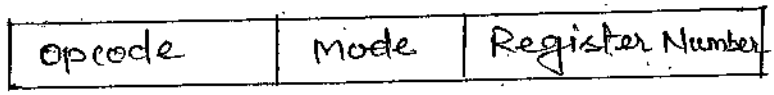
In this mode the instruction, itself, contains the operand. In other words, an immediate-mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate mode are useful for initializing register a constant value.

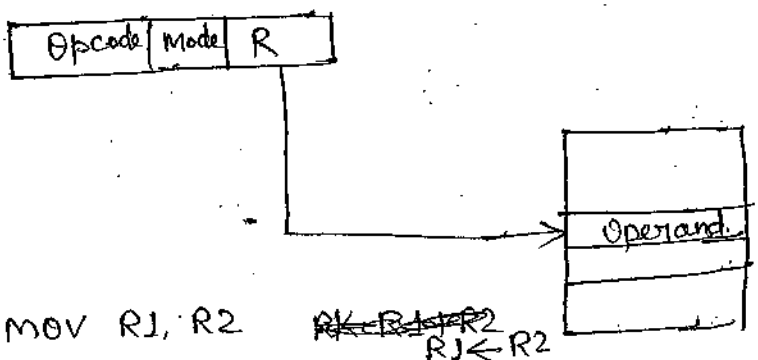
Ex. MVI 06 Move 06 to the accumulator
ADD 05 Add 05 to the content of Accumulator.



(3) Register Addressing Mode -

In this mode, the operands are in registers that reside within the CPU. The particular register is selected from a register field in the ~~CPU~~ instruction. The instruction contains the register number that has the operand. A k-bit field can specify 2^k registers.



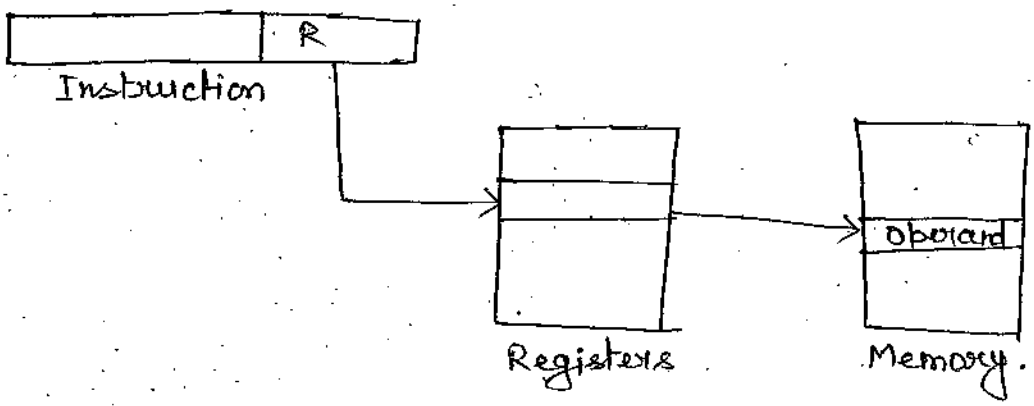


Ex. MOV R1, R2 ~~R1 ← R2~~ R1 ← R2
ADD R1 AC ← AC + R1

(4) Register Indirect Mode -

In this mode, the instruction specifies a CPU register whose contents give the address of the operand in memory. In other words the selected register contains the address of the operand, rather than the operand itself.

Ex. LD (R1) AC ← M[R1]



(5) Autoincrement or Autodecrement Mode :-

This is similar to the register indirect mode except that the register is incremented or decremented (or before) after its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction.

Effective Address -

The address field of an instruction is used by the control unit in the CPU to obtain the operand from the memory. Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.

"The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode."

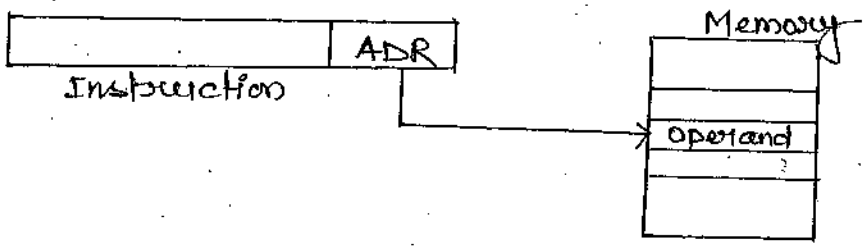
The effective address is the address of the operand in a computational type instruction. It is the address where control branches in response to a branch type instruction.

(6) Direct Addressing Mode:

In this mode the effective address is equal to the address part of the instruction. The operand resides in memory

and its address is given directly by the address field of the instruction. In a branch type instruction the address field specifies the actual branch address.

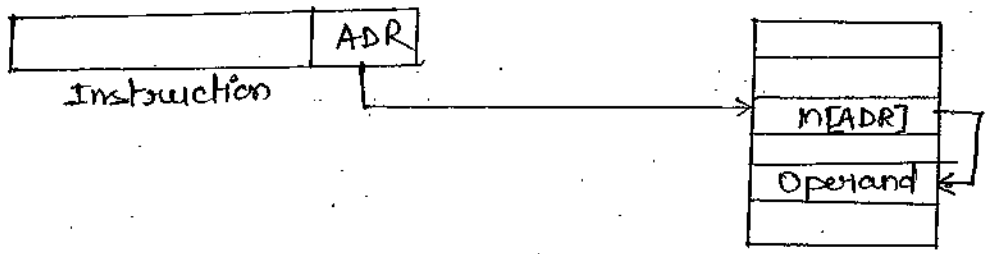
Ex. LD ADR AC ← M[ADR]



(7)- Indirect Addressing Mode -

In this mode the address field of the instruction gives the address where the effective address address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

Ex. LD @ADR AC ← M[M[ADR]]



(8). Relative Addressing Mode:

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

Effective Address = Address Part of instruction + content of PC.

(9)- Indexed Addressing Mode -

In this mode, the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The beginning ~~field~~ address of a data array in memory is defined by the address field of the instruction. Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in index register.

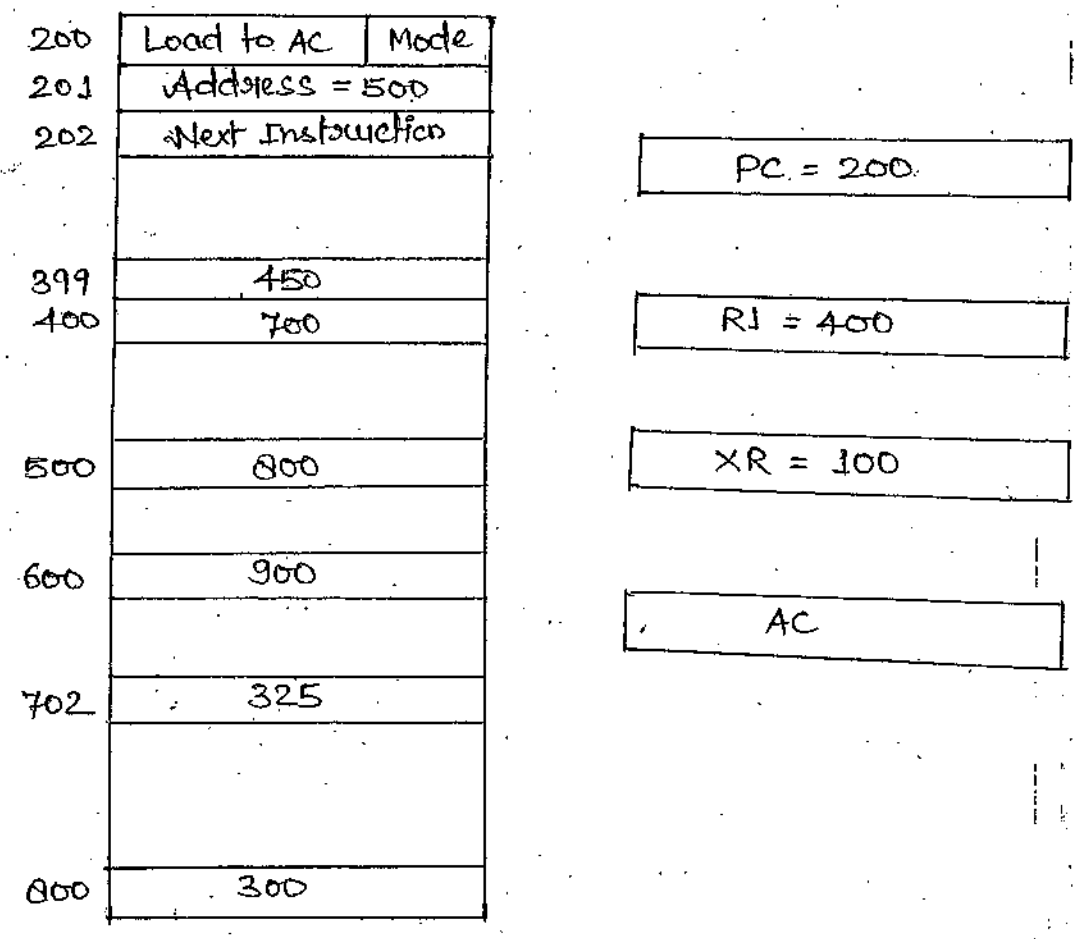
Effective Address = Address part of Instruction + Content of Index Register

(10)- Base Register Addressing Mode -

In this mode, the content of a base register is added to the address part of the instruction to obtain the effective address. A base register is assumed to hold a base address and the address field of an instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

Effective Address = Address part of instruction + content of Base Reg.

Example- The two word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500. The first word of the instruction specifies the operation code and mode, the second word specifies the address part. PC has the value 200 for fetching this instruction. The content of processor register R1 is 400, and the content of an index register XR is 100. AC receives the operand after the instruction is executed. The figure lists a few pertinent addresses and shows the memory content at each of these addresses. Find the effective address and operand that must be loaded into AC for each possible mode.



Solution -

(i) Immediate - In immediate mode, the second word of the instruction is operand rather than address, so 500 is loaded into AC. Effective address in this case is 201.

(ii) Register - In this mode, the operand is in R1 and hence 400 is loaded into AC.

(iii) Register Indirect - In this, the effective address is in processor register R1, i.e. 400 is the effective address and hence the operand is 700 loaded into AC.

(iv) Direct :- In this mode the effective address is the address part of the instruction i.e. 500. and the operand loaded into AC is 800.

(v) Indirect - In this mode, the effective address is stored in memory at address 500. So the effective address is at 800 and the operand loaded into AC is 300.

(vi) Index - In this mode

Effective Address = Address part of instruction + Content of XR

= 500 + 100 = 60

So, 900 is loaded into AC.

(vii) Relative - The instruction stored at 200 and 201 is now executing the content of the PC will be 202, because a single instruction occupies two memory word.

Effective Address = Address part of instruction + Content of PC.

= 500 + 202 = 702

So, 325 is loaded into AC.

Data Transfer Instructions:

Computers provide an extensive set of instructions to give the user the flexibility to carry out various computation tasks. The instruction set of different computers differ from each other mostly in the way the operands are determined from the address and mode fields. The actual operations available in the instruction set are not very different from one computer to another. It so happens that the binary code assignments in the operation code field is different in different computers, even for the same operation. The symbolic name may also be different in different computers for the same operation.

Most computers instructions can be classified into three categories:

(1) - Data Transfer Instructions

(2) - Data Manipulation Instructions

(3) - Program Control Instructions

(1) Data Transfer Instructions:-

Data transfer instructions move data from one place in the computer to another without changing the content. The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves. Different computers use different mnemonics for the same instruction name. Different data transfer instruction (with their mnemonic) are given in the table.

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Table: Data Transfer Instructions

The load instruction has been used mostly to designate a transfer from memory to a processor register usually an accumulator. The store instruction designate a transfer from a processor register into memory. The move instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used in data transfers between CPU registers and memory or between two memory words. The exchange ~~information~~ instruction swaps information between two registers or a register and a memory word. The input and output instructions transfer data among processor registers and input or output terminals. The push and pop instructions transfer data between processor register and a memory stack.

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ ADR	AC ← M[M[ADR]]
Relative address	LD $ ADR	AC ← M[PC + ADR]
Immediate Operand	LD # NBR	AC ← NBR
Index Addressing	LD ADR(X)	AC ← [ADR + XR]
Register	LD RJ	AC ← RJ
Register Indirect	LD (RJ)	AC ← M[RJ]
Autoincrement	LD (RJ)+	AC ← M[RJ], RJ ← RJ+1

Table: Eight Addressing modes for the load instruction

The character @ before memory address indicates indirect address. In case of register indirect mode, the register that holds the memory address is enclosed in parentheses. The character $ before memory address makes the address relative to the program counter (PC). The character # before the operand indicates immediate mode instruction.

(2) Data Manipulation Instructions -

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions into a typical computer are usually divided into three groups:

(1) Arithmetic Instructions
(2) Logical and bit manipulation Instructions.
(3) Shift Instructions.

(1) Arithmetic Instructions -

The four basic arithmetic operations are addition, subtraction, multiplication and division. Most computer provide instructions for all four operations. Some small computers have only addition and possibly subtraction operations. The multiplication and division must then be generated by means of software subroutines. Increment (or decrement) instructions add 1 (or subtract 1) to the value stored in a register or some memory word. A list of instructions is shown in table.

Name of Instruction	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Table: Typical Arithmetic Instructions

(2) Logical and Bit Manipulation Instructions:

Logical instructions perform binary operations on strings of bits stored in registers. They are useful for manipulating individual bits or a group of bits that represent binary-coded information. The logical instructions consider each bit of the operand separately and treat it as boolean variable. Some typi

logical and bit manipulation instructions are listed in table. The instruction (clear) causes the specified operand to be replaced by 0's. The complement instruction produces the 1's complement by inverting all the bits of the operand. The AND, OR and XOR instructions produce the corresponding logical operations on individual bits of operands.

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear Carry	CLRC
Set Carry	SETC
Complement Carry	COMC
Enable Interrupt	EI
Disable Interrupt	DI

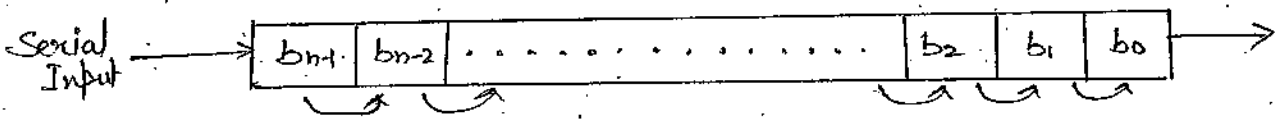
Table: Typical Logical and Bit manipulation Instruction.

(3) Shift Instructions:

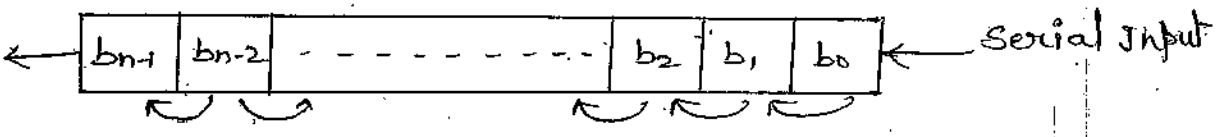
Shifts are operations in which the bits of a word are moved to the left or right. The bit shifted in at the end of the word determines the type of shift used. Shift instructions may specify either logical shifts, arithmetic shifts, or rotate-type operations.

(i) Logical Shift Instruction -

The logical shift inserts 0 to the end bit position. The end position is the left most bit for shift right and the right most position bit for shift left.



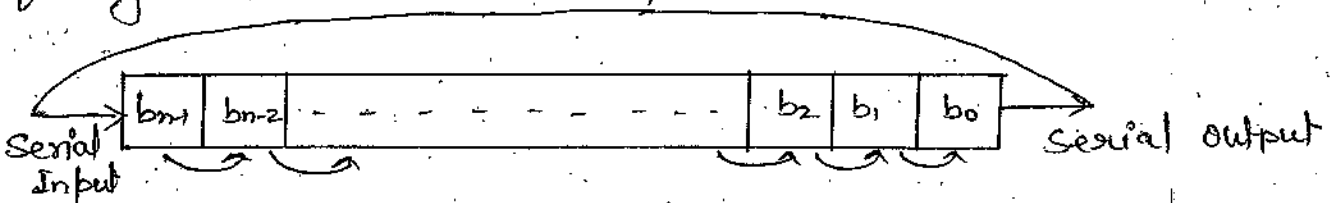
(a) shift right operation



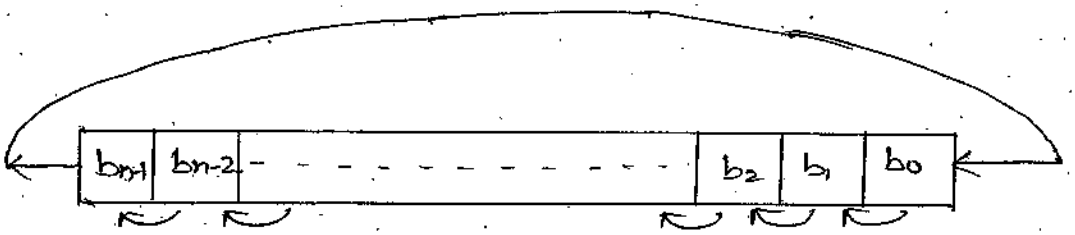
(a) Shift left Operation.

(ii) Circular shift -

The rotate instruction produces a circular shift. Bits shifted out at one end of the word are not lost as in logical shift but are circulated back into the other end. This circular shift is achieved by connecting the serial input of the shift register to the serial output.



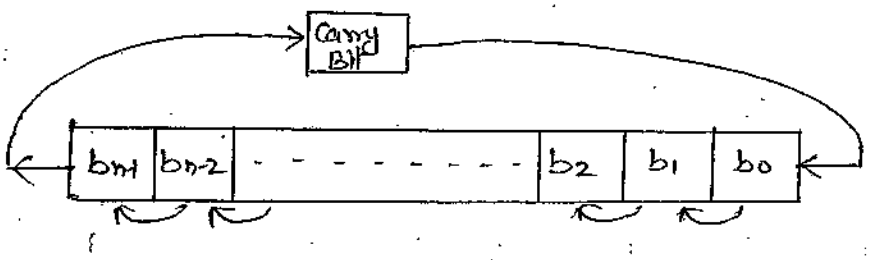
(a) Rotate Right



(b) Rotate Left

(iii) Rotate Through Carry :-

The rotate through carry instruction treats a carry bit as an extension of the register whose word is being rotated. Thus a rotate-left through carry instruction transfers the carry bit into the rightmost bit position of the register, transfers the leftmost bit position into the carry, and shifts the entire register to the left. Similarly, the rotate-right through carry transfers the carry bit into the leftmost bit position and transfers the rightmost bit into the carry and at the same time all other bits are shifted to the right.



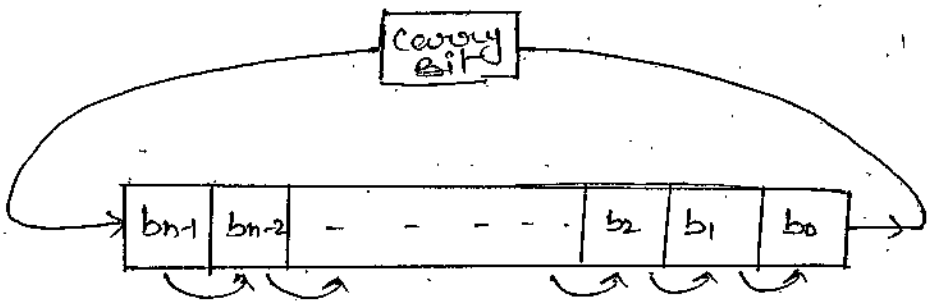


Fig: Rotate through left carry and Rotate right through carry

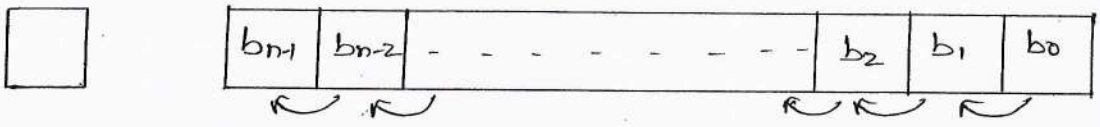
(iv) Arithmetic shift -

The arithmetic shifts operation shifts a signed binary number to the left or to the right. An arithmetic shift right operation divides the binary number by two and an arithmetic shift left operation multiplies the binary number by two. The two arithmetic shift operations leave the sign bit unchanged. Multiplying or dividing the binary number by two does not change the sign of the original number. Hence the sign of the number must be same.

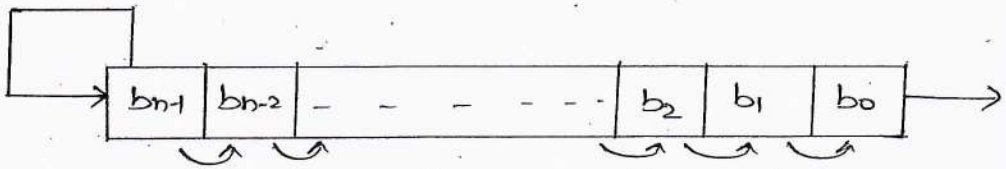
The most significant bit, b_{n-1}, holds the sign bit and the remaining bits represents the binary number. The arithmetic shift right leaves the sign bit unchanged and shifts the bit, including the sign bit, to the right. The rightmost bit is lost. The arithmetic shift left inserts 0 into b_0 and all other bits are shifted towards left. The bit, b_{n-1}, is shifted into a flag register and is replaced by bit b_{n-2}. If the bit b_{n-1} and b_{n-2} differ in their values a sign reversal occurs. Thus, b_{n-1} changes in value after the arithmetic shift left operation This happen if the multiplication by 2 results in an overflow. The overflow occurs if before shift operation bit $b_{n-1} \neq b_{n-2}$. This overflow situation can be found using

$$V_s = b_{n-1} \oplus b_{n-2}.$$

If $V_s = 0$, it means $b_{n-1} = b_{n-2}$ in values and there will be no overflow. But if $V_s = 1$, it means $b_{n-1} \neq b_{n-2}$ and there will be an overflow and hence a sign reversal is required after the shift operation.



(a) Arithmetic Shift Left



(b) Arithmetic Shift Right

Name	Mnemonic
Logical shift right	SHR
Logical shift Left	SHL
Arithmetic shift right	SHRA
Arithmetic shift Left	SHLA
Rotate Right	ROR
Rotate Left	ROL
Rotate right through carry	RORC
Rotate Left through Carry	ROLC

Table: Typical shift Instructions

(3) Program Control -

Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed. Each time an instruction is fetched from memory, the program counter is incremented so that it contains the address of next instruction in sequence. After the execution of a data transfer or data manipulation instruction, control returns to the fetch cycle with the program counter containing the address of next instruction in sequence. On the other hand a program control type instruction, when executed may change the address value in the program counter and cause the flow of control to be altered. Program control instructions specify conditions for altering the content of program counter, while data transfer and manipulation instructions specify conditions for data-processing operations. The change in value of PC as a result of execution

of a program control instruction causes a break in the sequence of instruction execution.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Table: Typical Program Control Instructions

The branch and jump instructions are used interchangeably to mean the same thing, but sometimes they are used to denote different addressing modes. The branch is usually a one address instruction written as BR ADR. When executed the branch instruction causes a transfer of the value of ADR into the program counter. Since the program counter contains the address of the instruction to be executed, the next instruction will come from location ADR.

Branch and jump instruction may be conditional or unconditional. An unconditional branch instruction causes a branch to the specified address without any conditions. The conditional branch instruction specifies a condition such as branch if positive or branch if zero. The skip instruction does not need an address field and is therefore a zero-address instruction. A conditional skip instruction will skip the next instruction if the condition is met. The call and return instructions are used in conjunction with subroutines. The compare and test instructions do not change the program sequence directly. The compare instruction performs a subtraction between two operands, but the result of the operation is not retained. However, certain ~~bit~~ status bits conditions are set as a result of the operation. Similarly, the test instruction performs the logical AND of two operands. ~~The status bits of interest~~ and updates certain status bits without retaining the result of changing the operands. The status bits of interest are carry bit, sign bit, a zero indication and an overflow condition.

The following table shows the conditional branch instructions.

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0.

Unsigned Compare Condition (A-B)

BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Signed Compare Condition (A-B)

BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less than or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Subroutine Call and Return -

A subroutine is a self-contained sequence of instructions that performs a given computational task. During the execution of a program, a subroutine may be called to perform its function many times at various points in the main program. Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions. After the subroutine has been executed, a ~~program~~ branch is made back to the main program.

The instruction that transfers program control to a subroutine is known by different names like call subroutine, jump to subroutine, branch to subroutine or branch and save address. A call subroutine instruction consists of an opcode together with an address that specifies the beginning of subroutine. The instruction is executed by performing two operations:

(1) The address of the next instruction available in the program counter (the return address) is saved in a temporary location so the subroutine knows where to return.

(2) Control is transferred to the beginning of subroutine.

The last instruction of every subroutine, commonly called return from subroutine, transfers the return address from the temporary location into the program counter. This results in a transfer of program control to the instruction whose address was originally stored in temporary location.

Program Interrupt-

Program interrupts refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. control returns to the original program after the service program is executed.

The interrupt procedure is quite similar to a subroutine call except for three variations:

(1)- The interrupt is usually initiated by an external or internal signal rather than from the execution of an instruction (except for software interrupt).

(2). The address of the interrupt service program is determined by the hardware rather than the ~~software~~ address field of an inst.

(3). An interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.

After a program has been interrupted and the service routine been executed, the CPU must return to return exactly

the same state that it was when the interrupt occurred. The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined from

(i) The content of PC
(ii) The content of all processor registers
(iii) The content of certain status conditions

There are three major types of interrupt that cause a break in the normal execution of the program. They can be classified as:

(i) External Interrupts: External interrupts come from input-output (I/O) devices, from a timing device, from any other external sources.
Ex- I/O device requesting for transfer of data, I/O device finished transfer of data, elapsed time of an event or a failure.

(ii) Internal Interrupt:- Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps.
Ex- Register overflow, attempt to divide by zero, an invalid opcode, stack overflow, and protection violation. These error conditions usually occur as a result of premature termination of instruction execution. The service program that processes the internal interrupt determines the corrective measure to be taken.

(iii) Software Interrupt:- External and internal interrupts are initiated from signals that occur in the hardware of the CPU. A software interrupt is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than software subroutine call. It can be used by the programmer to initiate an ~~instruction~~ interrupt procedure at any desired point in the program. The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides ~~for~~ means for switching from a CPU user mode to the supervisor mode. When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction.