Unit - 1

Basic Structure of computer.

1. Computer Types

→ A contemporary computer is a fast electronic calculating machine that accepts digitized i/p information, processes it a/c to a list of internally stored instructions, and produces the resulting o/p i/f.

→ The list of instructions is called a computer program, and the internal storage is called computer Memory.

→ Many types of computers exist that differ widely in size, cost, power and use.

↳ The most common computer is the personal computer

1. personal computer :-

→ The most common computer is the personal computer

→ It is a low capacity computer developed for single users.

→ which has found in homes, schools & business offices.

→ It is the most common form of desktop computers

2. Notebook computer :- (or) Laptop computer :-

→ It is the compact version of personal computer with all of these components packaged into a single unit.

→ It is a handy computer that can be easily carried anywhere.

3. Workstation Computer :-

→ The computer of this category is a high end and expensive one. It is exclusively made for complex work purpose.

→ These computers are often used in engineering appl especially for interactive design work.

4. Enterprise system (or) Main frame computer:

→ These are used for business data proccessing.

→ It is high capacity and costly computer.

→ It is largely used by big Organizations where many people can use simultaneously

5. Super Computers :-

→ This category of computer is the fastest and also very expensive.

→ A typical super computer can solve up to 10 trillion individual calculations per second.

→ super computers are used for the large-scale numerical calculations required in applications such as weather forecasting and aircraft design and simulation.

Topic: 2 · Functional Units

A computer consists of 5 functionally independent main parts. 1. Input 2. Memory 3. ALU (Arithmetic logic unit) 4. output & 5. control unit.

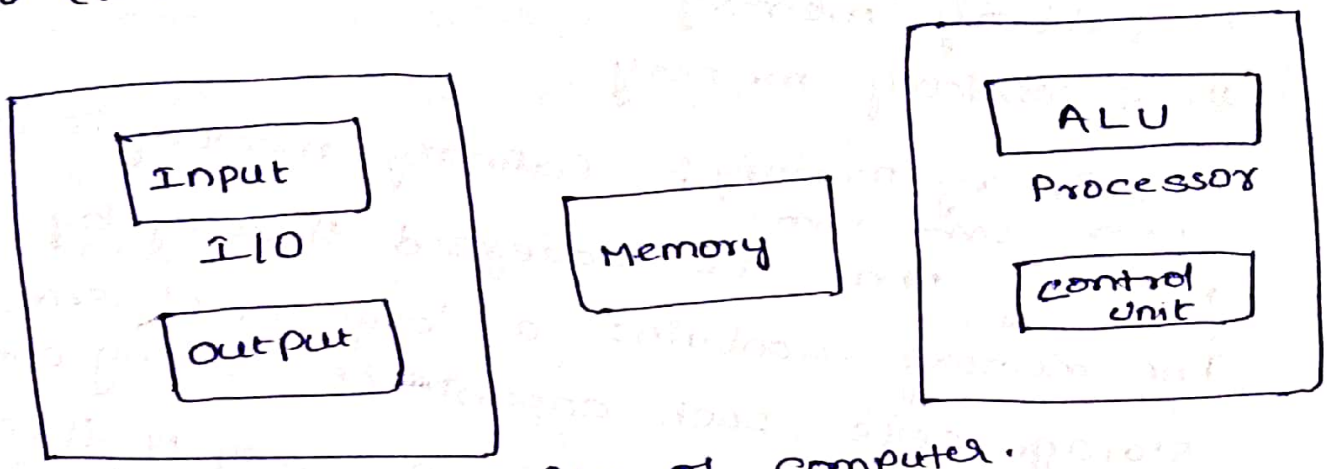


Fig: a: Functional units of computer.

→ Input device accepts the coded i/f as source pgm. (H.L.L). This is either stored in the memory or immediately used by the proccessor to perform the desired operations. The program stored in the memory determines the processing steps. Basically the computer converts one source pgm to an object pgm into machine language.

→ Finally the results are sent to the outside would through output device. All of these actions are co-ordinated by the control unit.

Input unit :-

→ The source pgm / h.l.language pgm / coded i.f / simply data is fed to a computer through input devices.

→ Keyboard is a most common type. Whenever a key is pressed, corresponding word or number is translated into its equivalent binary code over a cable & fed either to memory or processor.

Eg: Keyboard, mouse, scanners, Joysticks.

Memory unit:

Memory unit function is to store programs and data. It is basically to two types.

1. primary memory
2. secondary memory

1. Primary memory :- Primary memory is computer memory that is accessed directly by the CPU. The memory contains a large no. of semiconductors storage cells. Each capable of storing one bit of if These are processed in a group of fixed size called word.

↳ To provide easy access to a word in memory, a distinct address is associated with each word location. Addresses are numbers that identify Memory location.

↳ No. of bits in each word is called "word length" of the computer. programs must reside in the memory during execution.

↳ Instructions and data can be written into the memory or read out under the control of processor

↳ RAM is a type of data storage used in computers that is generally located on the Motherboard. This type of memory is volatile and all if that was stored in RAM is lost when the computer is turned off. volatile memory is temporary memory.

↳ The time required to access one word is called "Memory access time". Memory which is only readable by the user and content's of which can't be altered is called ROM. ROM is non-volatile and holds data permanently when the power is turned off.

2. Secondary Memory :-

Is used where large amounts of data & programs have to be stored, particularly info that is accessed infrequently.

Ex:- Magnetic disks & tapes, optical disks (CD-ROM) floppies etc., hard drive, CD.

3. Arithmetic logic unit (ALU) :-

→ Most of the computer operators are executed in ALU of the processor like addition, subtraction, division, multiplication etc.

→ The operands are brought into the ALU from memory and stored in high speed storage elements called Register.

→ Then alc to the instructions the operation is performed in the required sequence.

→ The control and ALU are many times faster than other devices connected to a computer sys

→ This enables a single processor to control a no.of external devices such as keyboards, displays magnetic and optical disks, and sensors.

4. Output unit :-

These actually are the counterparts of input unit. Its basic function is to send the processed results to the outside world.

Ex:- Printer, speakers, monitor etc.

5. Control unit :-

→ It effectively The control unit is effectively the nerve center that sends signals to other units and senses their states.

→ The actual timing signals that govern the transfer of data blw input unit, processor, memory and output unit are generated by the control unit

Topic :3 Basic Operational Concepts

1. Instructions take a vital role for the proper working of the computer.
2. An appropriate program consisting of a list of instructions is stored in the memory so that the tasks can be started.
3. The memory brings the individual instructions into the processor, which executes the specified operations.
4. Data which is to be used as operands are moreover also stored in the memory.

Example:

Add LOCA, R0 → (1)

→ This instruction requires the perf of several steps
1. The instruction is fetched from memory into the processor.
2. The operand at LOCA is fetched & added to R0
3. Finally the resulting sum is stored in the reg R0.

This instruction adds the operand at memory location LOCA, to the operand which will be present in the Register R0 & places the sum into register.

The above mentioned example can be written as follows: These 2 types of operations are performed by separate instructions:

```
LOAD   LOCA, R1
Add    R1 , R0
```
→ (2)

First instruction sends the contents of the memory location LOCA into Register R0, and meanwhile the second instruction adds the contents of Register R1 and R0 and places the output in the Register R1.

→ The memory and the processor are swapped.
→ The data is then transferred to or from the Memory. and issuing the appropriate signals.

Analysing how processor and memory are connected

→ processors have various registers to perform various functions.
→ Connection B/w processor & Memory.
→ The block diagram consist of the following components.

1. Memory :- MU is responsible for storing all the imp data.
2. MAR
3. MDR
4. PC
5. IR
6. General purpose Registers
7. Control unit 8. ALU.

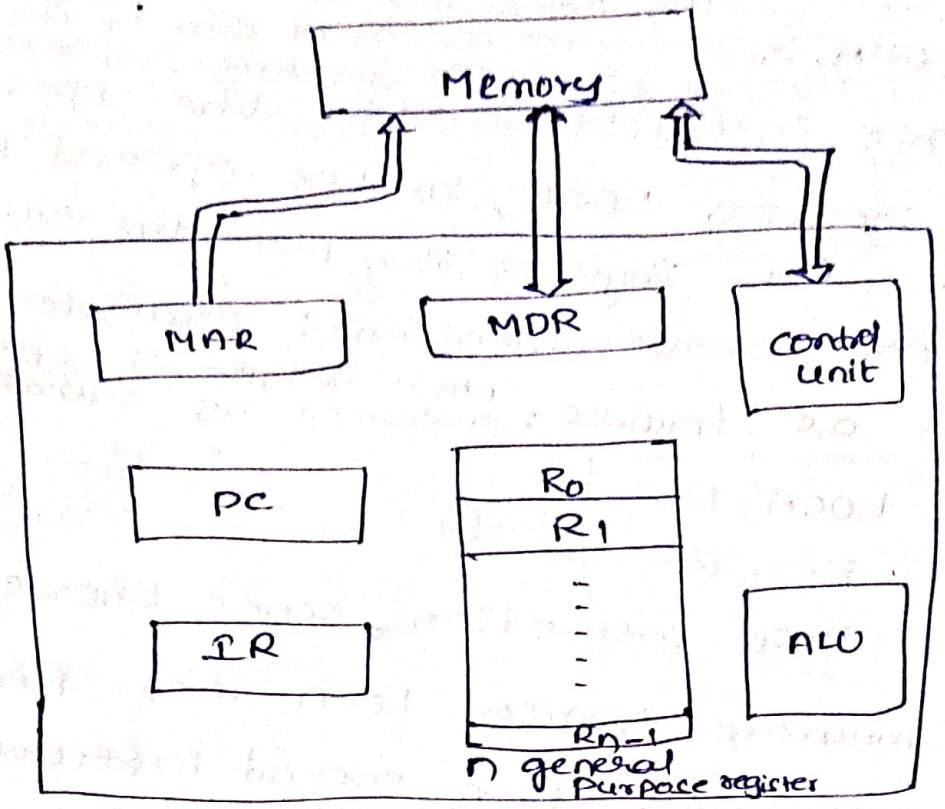


fig: connections between the processor and the memory

→ Processors have various registers to perform various functions

→ Program counter :- It contains the memory address of next instruction to be fetched.

→ Instruction Register :- It holds the instruction which is currently being executed.

→ MDR (Memory Data Register):- It facilities communication with memory. It contains the data to be written into or read out of the addressed location.

→ MAR (Memory Address Register):- It holds the address of the location that is to be accessed.

→ There are 'n' general purpose registers that is R0 to Rn-1. Registers stores the data temporarily

→ MAR & MDR, these two registers facilitate the communication with memory.

→ Memory address of the next instruction to be fetched

executed. Memory is a specialized register. It keeps the record of the programs that are executed.

→ In order to get the best performance it is required to design the compiler, machine instruction set and hardware in a co-ordinated manner.

Working Explanation of

→ A program counter is set to the first instruction of the program.

→ The contents of the PC are transferred to the MAR, and a Read control signal is sent to the memory.

→ The addressed word is fetched from the location which is mentioned in the MAR & loaded into MDR.

→ The contents of PC are incremented so that PC points to the next instruction that is to be executed.

→ Normal execution of a program may be temporarily interrupted if some devices require urgent servicing, to do this one device raises an Interrupt signal.

→ An interrupt is a request signal from an I/O device for service by the processor.

→ The Diversion may change the internal stage of the processor its state must be saved in the memory location before interruption.

→ After completion of this interruption request, processor can continue the interrupted program.

TOPIC 4: BUS Structures

The simplest and most common way of interconnecting various parts of the computer.

To acheive a reasonable speed of operation, a computer must be organized so that all units can handle one word of data at a given time.

A group of lines that carry the data, the bus must have lines for address and control purpose.

Simplest way to interconnect is to use the single bus as shown.

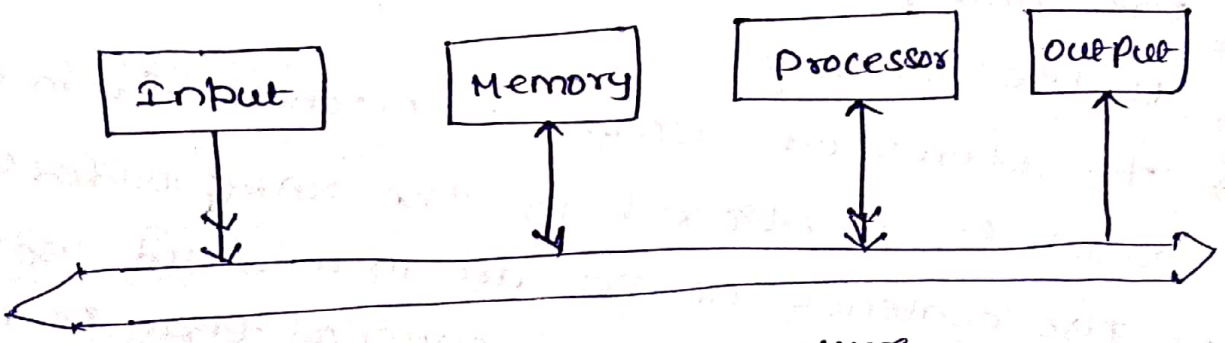


Fig: single bus structure

Since the bus can be used for only one transfer at a time, so only 2 units can actively use the bus at any given time.

→ only one device at a time can successfully transmit

→ single bus structure is a
  * Low cost
  * very flexiable for attaching peripheral devices.

→ Multiple bus structure certainly increases the performance but also increases the cost.

→ All the interconnected devices are not of same speed and time, leads to a bit of a problem.
→ This is solved by using cache registers (buffer reg).
→ These buffers are electronic registers of small capacity when compared to the main memory but of comparable speed.
→ The instructions from the processor at once are loaded into these buffers & then the complete transfer of data at a fast rate will take place.

Topic 5:- Software

→ Software is a program, that enables a computer to perform a specific task.
→ A user to enter and run an application program, the computer must already contain some system software in its memory.
→ System software is a collection of programs that are executed to perform functions such as

1. Receiving and interpreting user commands.
2. Entering and editing application programs and storing them as files in secondary storage devices.
3. Managing the storage and retrieval of files in secondary storage devices.
4. Controlling I/O units to receive I/P if and produce o/p results.
5. Translating programs from source form into object form consisting of machine instructions.
6. Linking and running user-written application programs with existing standard library routines.

→ system software is responsible for the coordination of all activities in a computing system.

→ A programmer using a H.L.L need not to know the details of machine Program instructions.

→ A System software program called a "compiler" translates the high-level language program into machine language program.

→ Another important system program that all programmers use is a "text editor". It is used for entering and editing application programs.

→ A key system software component called the (OS) operating system. This is a large program, or a collection of routines, that is used to control the sharing of and interaction among various computer units.

→ Inorder to understand the basics of O.S, Let us consider a system with one processor, one disk and one printer.

→ The first step is to transfer the file into Memory, when the transfer is Complete, execution of the Pgm is started.

→ When the execution of the program reaches the point where the data file is needed, the program requests the O.S to transfer the data file from the disk to the memory.

→ The OS performs this task and passes execution control back to the application Program, which then proceeds the required computation.

→ When the computation is completed and the results are ready to be printed, the appn pgm again sends a request to the O.S.

→ We have seen how execution control passes back and forth blw the appn pgm & the OS routines.

→ A convenient way to illustrate this sharing of the processor execution time is by a time-line diagram, such as shown in the below figure.

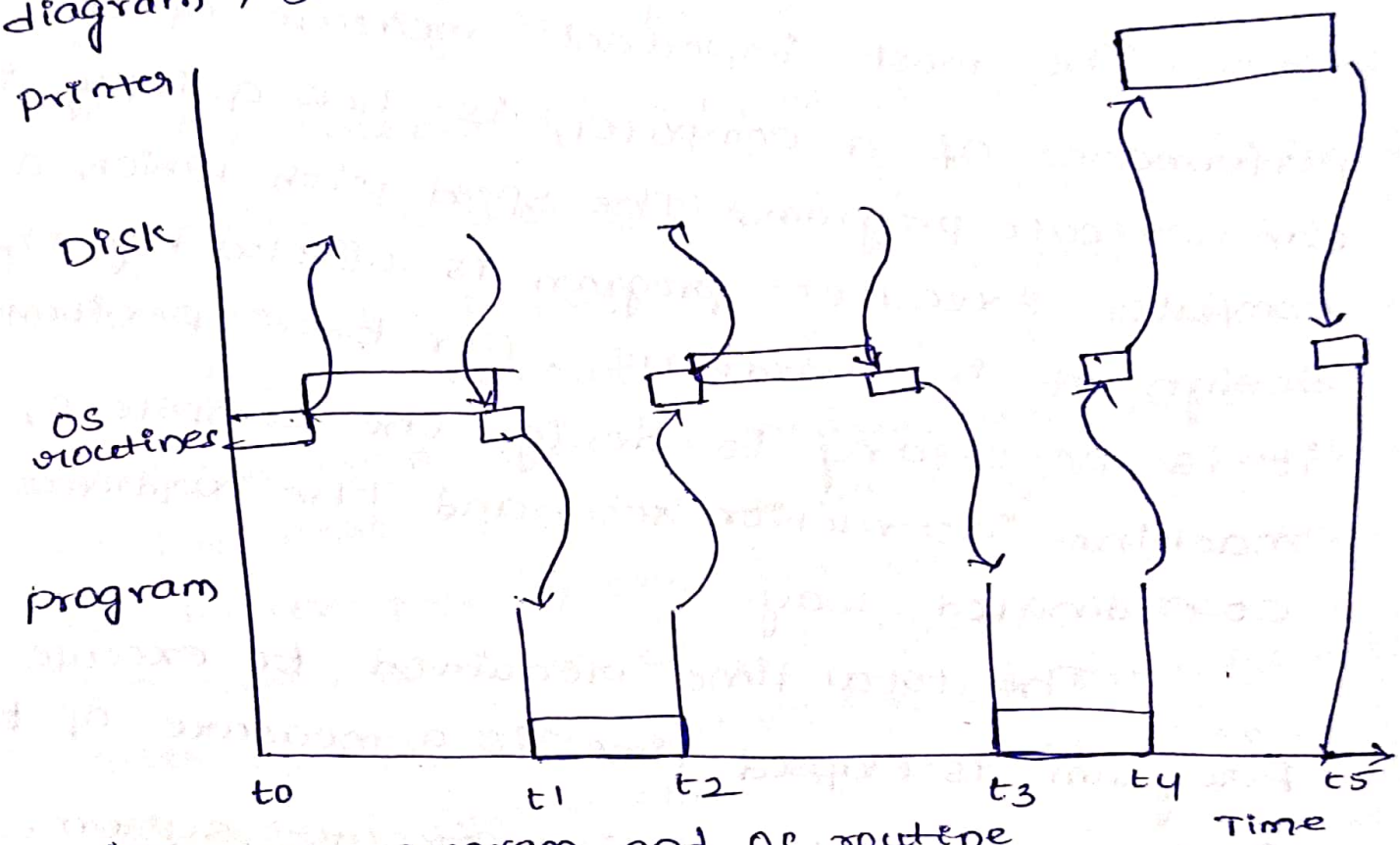


fig: User program and OS routine sharing of the processor.

→ During the time period t0 to t1, an OS routine initiates loading the appn pgm from disk to Memory, waits until the transfer is completed, & then passes execution control to the appn pgm.

→ A similar pattern of activity occurs during period t2 to t3 and period t4 to t5, when the OS transfers the data file from the disk & print the results. At t5 the OS may load & execute another appn pgm.

→ Operating system manages the concurrent execution of several appn pgms to make the best possible use of computer resources.

→ This pattern of concurrent execution is called multiprogramming or multitasking.

Topic:6: Performance

The most important measure of the performance of a computer, is how quickly it can execute programs. The speed with which a computer executes program is affected by the design of its hardware. For best performance it is necessary to design the compilers, the machine instruction set, and the hardware in a co-ordinated way.

The total time required to execute the program is elapsed time is a measure of the performance of the entire computer system. It is affected by the speed of the processor, the disk, and the printer. The time needed to execute a instruction is called the processor time.

Just as the elapsed time for the execution of a program depends on all units in a computer system, the processor time depends on the hardware involved in the execution of individual machine instruction.

This hardware comprises the processor and the memory which are usually connected by the bus. as shown in the fig: single bus structure.

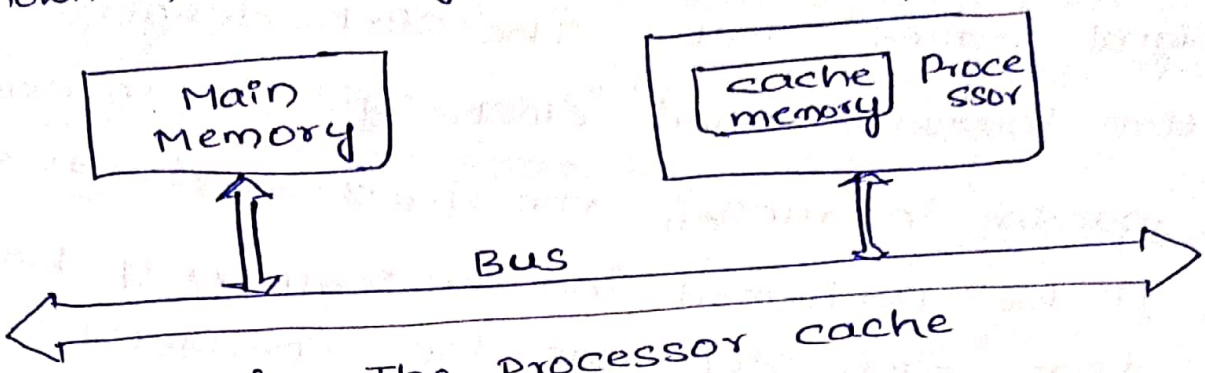


fig: The Processor cache

The pertinent parts of the fig: single bus is repeated in the cache processor which includes cache memory as part of the processor unit.

Let us examine the flow of program instructions and data blw the memory and bhe processor. As the start of execution, all program instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache later if the same instruction or data item is needed a second time, it is read directly from the cache.

A pgm will be executed faster if the movement of instructions and data blw the main memory & the processor is minimized, which is achieved by using the cache.

for ex:- suppose a no of instructions are executed repeatedly over a short period of time ga a pgm loop. If these instructions are available in the cache, they can be fetched quickly during the period of repeted use. same appiles to data that arecued repetedly.

1. Processor Clocks:-

Processor circuits are controlled by a timing signal called "clock". The clock designer the regular time intervals called "clock cycles". To execute a machine instruction the processor divedes the action to be performed into a sequence of basic steps that each step can be completed in one cycle clock. The length P of one clock cycle is an important parameter that affects the Processor performance.

Processor used in today's personal computer and work station have a clock rates that range from a few hundred million to over a billion cycles per second.

2. Basic Performance Equation:-

We now focus our attention on the processor time component of the total elapsed time.

Let 'T' be the processor time required to execute a pgm.

→ The compiler generates a machine language that corresponds to the source Program.

→ Assume that complete execution of the pgm requires the execution of N machine cycle language instructions.

→ The number N is the actual no of instruction execution.

Suppose that the average no. of basic steps needed to execute one machine cycle instruction is 'S', each basic step is completed in one clock cycle. If clock rate is 'R' cycles per second, The program execution time is given by $T = \frac{N \times S}{R}$.

this is often referred to as the basic performance

We must emphasize that N, S & R are not independent parameters, changing one may affect another.

Introducing a new feature in the design of a processor will lead to improved performance only if the overall results is to reduce the value of T.

3. Pipelining and super scalar operation:-

We assume that instructions are executed one after the other. Hence the value of S is the total no. of basic steps, or clock cycles, required to execute one instruction.

A substantial improvement in performance can be achieved by overlapping the execution of successive instructions using a technique called "pipelining".

Consider Add $R_1 R_2 R_3$

This adds the contents of R_1 & R_2 and places the sum into R_3.

4. Clock rate :-

These are the two possibilities for increasing the clock rate 'R'.

1. Improving the IC technology makes logical circuits which reduces the time of execution. This allows the clock period P, to be reduced & the clock rate R to be increased.

2. Reducing the amount of processing done in one basic step & also makes it possible to reduce the clock period P, the no. of basic steps needed may increase.

→ Increase in the value 'R' that are entirely caused by improvements in IC. technology

5 Instruction Set CISC & RISC :-

→ Simple instructions require a small no. of basic steps to execute.

→ complex instructions involve a large no. of steps.

→ For a processor that has only simple instructions, a large no. of instructions may be needed to perform a given programming task.

→ But complex instructions combined with pipelining would achieve the best performance.

→ However, it is much easier to implement Efficient pipelining in processors with simple instruction set

6. Compiler

→ A compiler translates a high-level language program into a sequence of machine instructions

→ An optimizing compiler takes advantages of various features of the target processor to reduce the product NXS.

→ The no. of clock cycles needed to execute a pgm.

→ The compiler may rearrange program instructions to acheive better performance.

→ A compiler appears as a separate entity from the processor.

→ The compiler and the processor are often designed at the same time, with much interaction blw the designers to acheive best results.

→ CISC & RISC

CISC	RISC
1. CISC is complex Instruction set computers.	1. RISC is a Reduced Instruction set computer.
2. CISC architecture gives more importance to hardware	2. RISC architecture gives more importance to s/w.
3. Complex Instructions	3. Reduced Instructions
4. coding in CISC processor is simple.	4. coding in RISC processor requires more no. of lines.
5. It consists of complex instructions, it takes multiple cycles to execute	5. It consist of simple that take single cycle execute
6. Complexity lies in Micro Pgm.	6. complexity lies in compiler
7. Speed is less.	7. speed is more.

TOPIC 7: Multiprocessors And Multicomputers

Multicomputer

1. A computer made up of several computers. Each processor has its own memory rather than a multiple processors with a shared memory.
2. Distributed computing deals with hardware & software systems.
3. Multiprocessor computer have one physical address space/CPU
4. It can run faster
5. A Multi computer is Multiple computers, each of which can have Multiple processors.

Multi processors

Def: It contain a no of processor units are called Multi Processor sys.

1. A Multiprocessor system is simply a computer that has more than one CPU on its Mother board.
2. Multiprocessing is the use of two (or) more central processor units with in a single computer system.
3. It has a single physical address space (Memory) share by all the CPUs.
4. A Multiprocessor would run slower, because it would be in one computer.
5. A Multi-processor is a single system with Multiple CPU's.

Differences b/w System S·W & App^n software:-

System software	Application software
1. Sys S·W is used for operating computer hardware.	1. A·S is used by user to perform specific task.
2. System S·W are installed on the computer when O·S is installed.	2. A·S are installed a/c to user's requirement.
3. In general, the user does not interact with system software because it works in the background.	3. In general, the user Interact with application softwares.
4. System S·W can run independently. It Provides platform for running app^n S·W	4. A·S·W can't run independently. They can't run with out the presence of system S·W.
5. some examples of System S·W are compiler, assembler, debugger, driver etc.	5. some Examples of App^n S·W are word processor, web browser, Media player etc.

Machine Instructions and Programs

TOPIC 1:- NUMBERS, ARITHMETIC OPERATIONS, AND CHARACTERS

→ Computers are built using logic circuits that operate on if represented by two values 0 & 1:

@ Number Representation :-

→ Consider a n-bit vector $B = b_{n-1} \cdots b_1 b_0$

→ The unsigned integer values V in the range 0 to 2^{n-1}.

where $V(B) = b_{n-1} \times 2^{n-1} + \cdots + b_1 \times 2^1 + b_0 \times 2^0$

→ We need to represent in both positive & negitive

→ Three systems are used for representing such Numbers:

* sign and Magnitude
* 1's complement
* 2's complement.

+0
−1

→ In all three systems, the left most bit is '0' for positive numbers and 1 for negative numbers.

B (b_3 b_2 b_1 b_0)	values Represented: sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Fig: Binary, signed-integer representations

→ The positive values have same identical representation in all the three systems, but Negative values will have different Representation.

(b) Addition of positive Numbers:-

→ Let us consider two 1-bit numbers

0	1	0	1
+ 0	0	1	1
0	1	1	10

↑ carry-out

fig:- Addition of 1-bit numbers.

(c) Addition And subtraction of signed Numbers:-

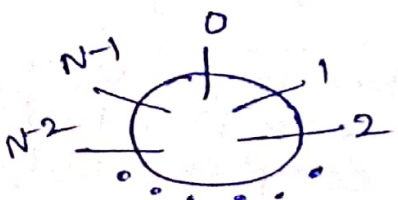
→ In this, the most efficient method for performing addition & subtraction operation is 2s complement.

→ Let us apply addition technique to the simple example of adding +7 to -3.

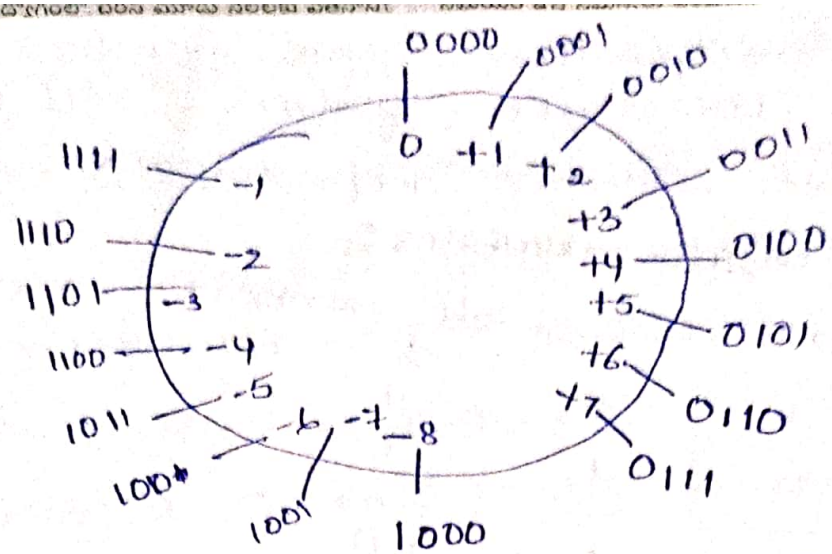
→ The 2's complement representation of these two numbers is +7 = 0111 −3 = 0011 → 1's comp, 1100 → 2's comp, 1101

Given Number −3 = +3 = 0011
↳ 1's comp = 1100
2s +1 1
2's comp = 1101

So 7 ⇒ 0111
−3 ⇒ 1101
1 11
10100
↑
Carry-out.



(d) Circle Representation of integers mod N



(b) Mod 16 system for 2's complement numbers

fig:- Modular number systems and the 2'-complement

Rules governing the addition & subtraction of n-bit signed Numbers using the 2's complement Representation system :-

1. To add two numbers, add their n-bit representation ignoring the carry-out signal from the Most Significant bit (MSB) position.
The range of the value is -2^{n-1} through $+2^{n-1}-1$

2. To Subtract two Numbers x and y, that is to perform x-y, form the 2's complement of y & then add it to x.
→ Again the result will be the algebraically correct value in the 2's complement Representation.
→ The range of the value is -2^{n-1} through $+2^{n-1}-1$

Eg:- (a) (+2) 0010
(+3) 0011
(+5) 0101

(b) (+4) 0100
(-6) 1010 → Take tve, 6 → 0110
(-2) 1110 Comp 1's → 1001
Comp 2's → 1010

(c) (-5) 1011
(-2) 1110
(+)
(-7) 1001

(d) (+7) 0111
(-3) (+) 1101
+4 10100
Carry

fig: 2's complement add and subtract operations

(d) Overflow in Integer Arithmetic :-

→ In the 2's complement Number representation system n bits can represent values in the range -2^{n-1} to $+2^{n-1}-1$

→ When the result of an arithmetic operation is outside the representable range, an arithmetic overflow has occured.

Overflow :-

1. Overflow can occur only when adding two nos that have the same sign.
2. The carry-out signal from the sign bit position is not a sufficient indicator of overflow when adding signed numbers.

Eg: +7 0111
+4 0100
1011 (-5)

If we add the nos +7 & +4, the O/P sum vector s is 1011, which is the code for -5, an incorrect result.

(e) Characters :- In addition to numbers, computers must be able to handle non-numeric text i.e consisting of characters.

→ characters can be letters of the alphabet, decimal digits, punctuations etc.

→ They are represented by codes that are usually eight bits and the most widely used code is the ASCII (American standards committee on Information Interchange)

TOPIC: 2 Instructions and Instruction sequencing

→ A computer Program consist of a sequence of small steps (such as adding 2 nos, testing for a particular condition, reading a cha from keyboard) A computer must have instructions capable of performing four types of operations:-

1. Data transfers blw the memory & the processor regis
2. Arithmetic & Logic operations on data.
3. Program sequencing & control
4. I/O transfer (↳ Execution of Pgm, ↳ Reading a cha from the keyboard (or) sending a cha to be displayed on screen)

1. Register transfer Notation :-

→ Transfer of information from one Location in the computer to another.

→ Locations that may be involved in such transfer are memory Locations, processor registers, (or) registers in the I/O sub system.

→ The address of memory Locations may be LOC, PLACE, A, VAR2.

→ Processor registers names may be R0 to R5.

→ I/O register names may be DATA IN, OUT STATUS etc.

→ The contents of a location are denoted by placing the square brackets around the name of the Location. Thus, the expression is given as

$R1 \leftarrow [LOC]$ (memory location (Hallow c))

It means the contents of memory Location Loc are transferred in to the Processor register R1.

Another operation that adds the contents of register R_1 & R_2 & then places their sum into Register R_3.

$$R_3 \leftarrow [R_1] + [R_2]$$

This type of notation is known as Register Transfer Notation. (RTN)

→ The right hand side of a RTN expression always denote a value, and the Left hand side is the name of Location, where the value is placed.

Assembly Language Notation :-

→ An another type of notation to represent machine instructions & Programs called assembly Language format.

Eg: MOV LOC, R_1 (LOC → R_1)

Load R_2, LOC (contents read from memory location are loaded into register)

The contents of LOC are unchanged by the execution of the instruction; but the old contents of register R_1 are overwritten.

Eg:- Add R_1, R_2, R_3

Adding two numbers contained in processor registers R_1 & R_2 and placing their sum in R_3.

Basic Instruction types :-

4 → 1. 3-address instruction
2. 2- "
3. one- "
4. zero address "

Three - Address Instruction :-

→ The operation of adding two numbers is a fundamental capability in any computer. The Statement $C = A + B$ It is to add the correct values of the two variables called A & B and to assign the sum into a third variable.

→ When this stmt is compiled, the three variables A, B and C are assigned to distinct locations in the memory.

$$C \leftarrow [A] + [B].$$

→ To carry out this action, the contents of memory locations

A and B are fetched from the memory and transferred into the processor where their sum is computed.

→ Assume that this instruction contains the memory addresses of the three operands A, B and C.

→ This three-address instruction can be Represented as,

```
Add A,B,C
```

Operands A and B are called the source operands, C is called the destination operand. and Add is the operation to be performed on the operands.

→ A general instruction of this type has the format

```
Operation source1, source 2, destination
```

Two address instruction :- It is of the form

```
Operation source, destination
```

An Add instruction of this type is

```
Add A,B
```

→ The operation B ← [A] + [B].

When the memory sum is calculated the result is sent to the memory and stored in Location B, replacing the original contents of this location.

→ Operand B is both a source & a destination.

```
MOV B,C
```

which performs the operation C ← [B] Leaving the contents of location B unchanged.

→ The Operation C ← [A] + [B] can now be performed by the two-instruction sequence.

```
MOV B,C
Add A,C
```

In all the instructions above, the source operands

are specified first, followed by the destination.

One Address Instruction :-

In the one address register, a processor register, usually called as the Accumulator is used for this purpose.

eg :- Add A

In this Add the contents of memory Location A to the contents of the accumulator register & place the sum back in to the Accumulator.

→ Load A & store A are the one address instructions.

→ The Load A instruction copies the contents of memory location A in to the accumulator.

→ The store instruction copies the contents of the accumulator into memory Location.

Using the only-one address instructions, the operation C ← [A] + [B] can be performed by executing the sequence of instructions.

```
LOAD   A
Add    B
Store  C
```

Registers are used to store data temporarily in the processor during Processing.

Let Ri represent a general-purpose register. The instructions, Load A, Ri.
store Ri, A.

and Add A, Ri.

These are the generalizations of the Load store & Add instructions for the single Accumulator.

When a processor has several general purpose registers, Many instructions involve only operands that are in the registers.

Instructions such as,

```
Add Ri, Rj
```

(or)

```
Add Ri, Rj, Rk
```

In above instructions, the source operands are the contents of registers Ri & Rj.

→ If the data has to be transfer b/w different locations, it is given as,

```
Mov source, destination
```

This instruction copy the contents of source in to the Destination.

→ When data are moved to (or) from a processor register, the Move instruction can be used rather than the Load (or) store instructions because the Order of the source & destination operands determine which operation is intended. Thus,

```
Mov A, Ri
```

is same as, `Load A, Ri` and `Mov Ri, A` is same as `Store Ri, A`

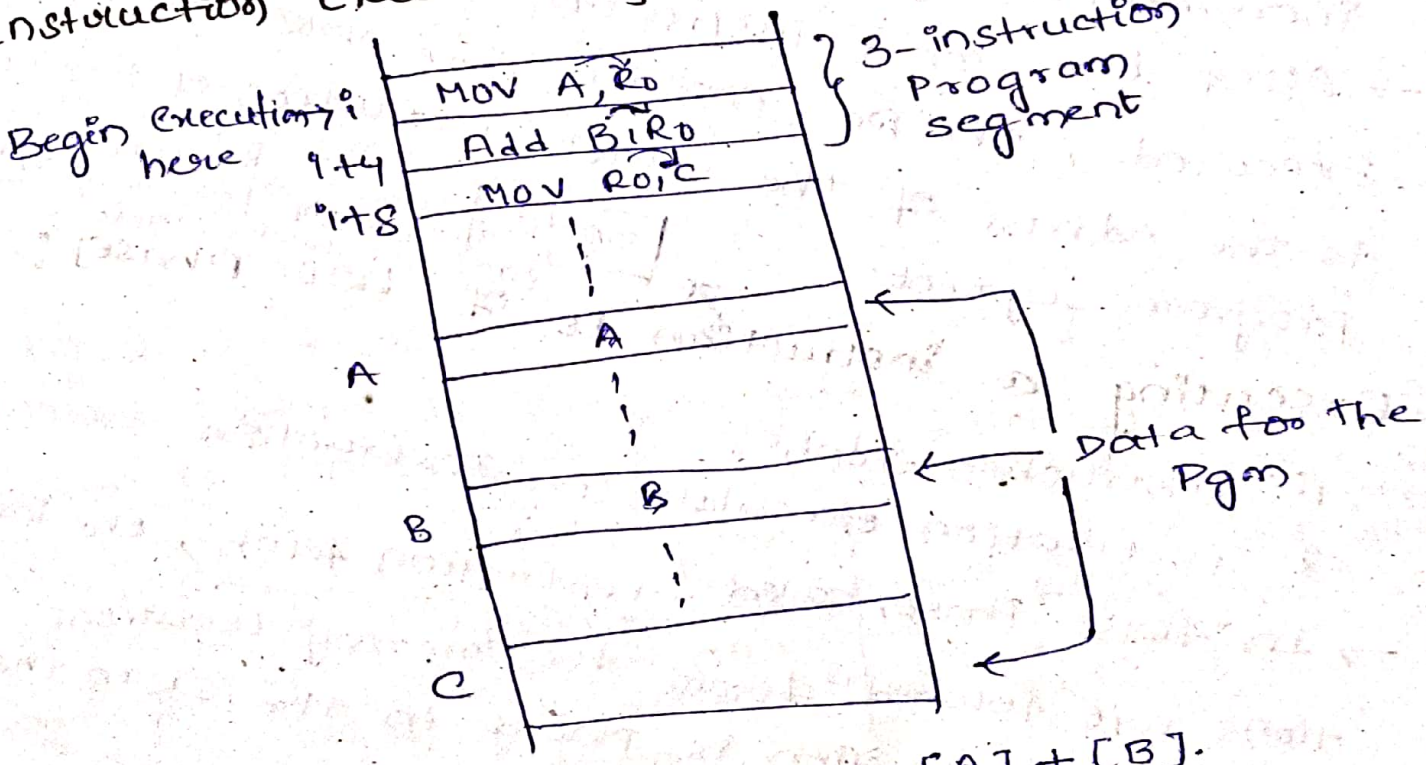
→ The arithmetic operations are allowed only one operands that are in processor registers, the C=A+B can be performed by the instruction sequence.

```
MOV   A, Ri
MOV   B, Rj
Add   Ri, Rj
MOV   Rj, C
```

→ In Processors where one operand may be in memory but the other must be in a register, an instruction sequence for the required task would be,

```
MOV   A, Ri
Add   B, Ri
MOV   Ri, C.
```

Instruction Execution & Straight-Line sequencing :-



A Program for C ← [A] + [B].

→ The task has to perform is C ← [A] + [B].
Assume that the computer allows one memory operand per instruction and has a no. of Processor registers.

→ word length is 32 bits & the memory is byte addressable. The starting instruction location is 'i', second & third instructions start at address i+4, & i+8.

Execution of a Program :-

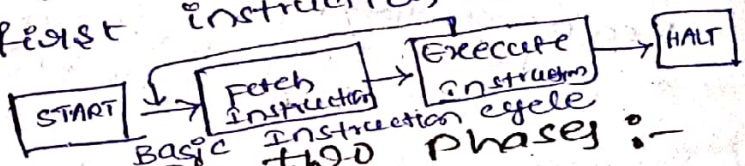
The processor contains a register called the Pgm counter (PC), which holds the address of the Next instruction to be executed.

→ To execute a pgm, the address of the first instruction is placed into the PC.

Processor control circuits use the PC to fetch & execute instruction, one at a time, in the order of increasing address, it is known as straight-line sequence.

→ The execution of each instruction, the PC is incremented by 4 to point to the next instruction.

→ After the move instruction at location i+8 is executed, the PC contains the value i+12, which is the address of the first instruction of the next program segment.



Basic Instruction cycle

Executing a instruction is of two phases :-

1. Instruction fetch
2. Instruction execution (or) Instruction execute.

→ In first phase, called instruction fetch, the instructions are fetched from the memory location.

→ And the instruction is placed in the IR in the processor.

→ In the 2nd phase, called Instruction execution is to determine which operation is to be performed.

→ It fetches the operands from the memory (or) from processor registers, perform an ALU operation and stores the result in the destination register.

Branching :-

→ Consider a task of adding 'n' numbers. The addresses of the memory locations containing the 'n' numbers are symbolically given as NUM1, NUM2, ... NUMn.

→ A separate add instruction is used to add each number to the contents of Register R_0.

→ After all the numbers are added, the result is placed in memory location SUM.

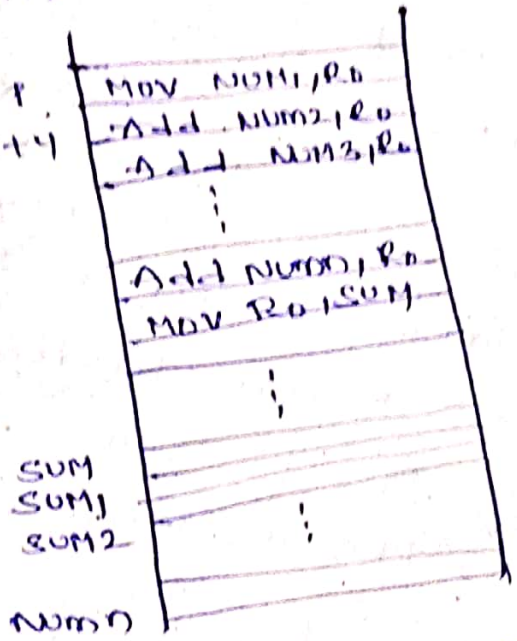


fig: 2(A): Straight line pgm for adding 'n' numbers.

→ In this instead of using a Add instruction, it is possible to place a single Add instruction in a pgm loop.

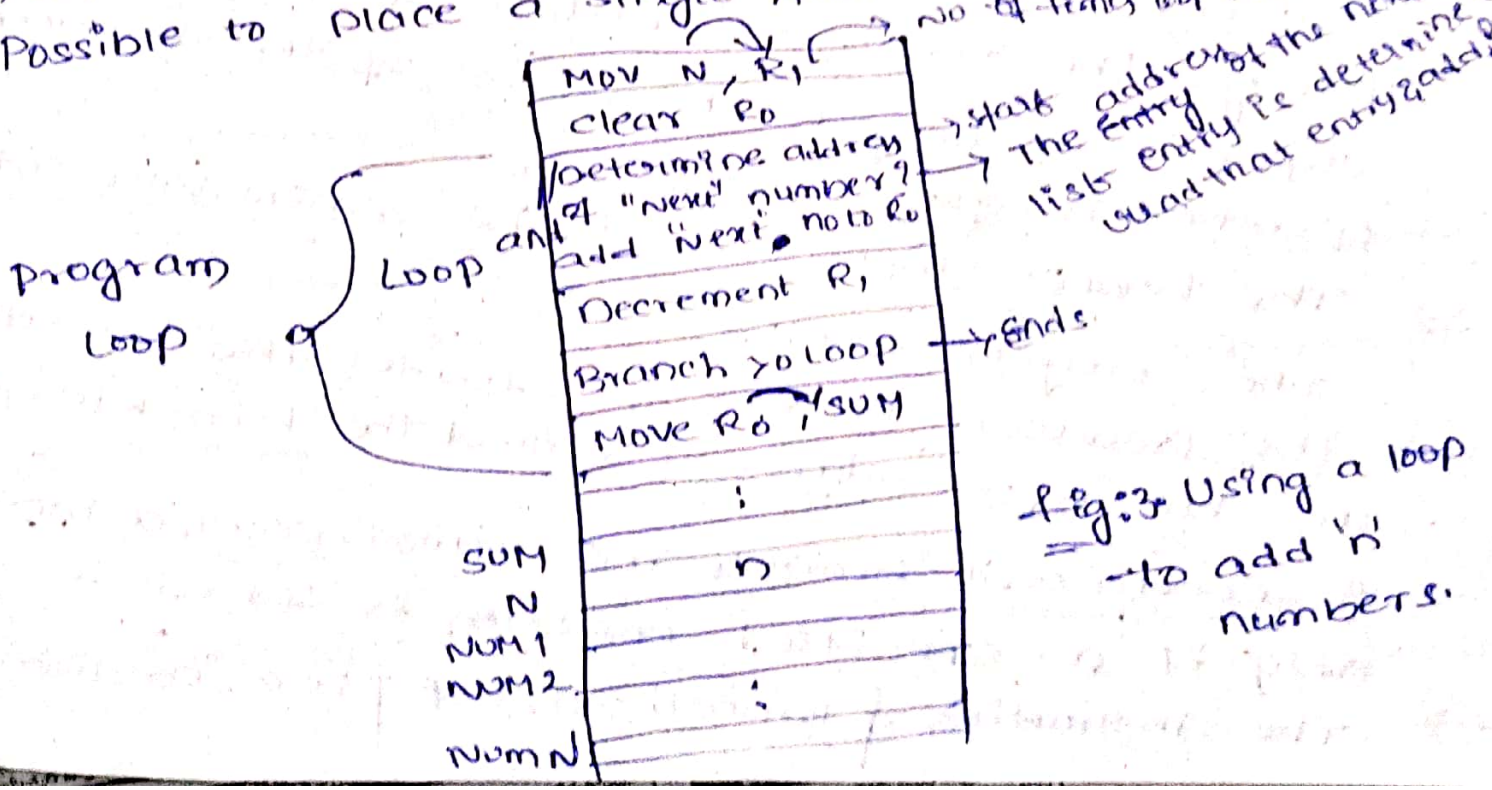


fig: 3 Using a loop to add 'n' numbers.

→ The Loop is a straight-line sequence of instructions executed as many times as needed.

→ It starts at location Loop & ends at the instruction Branch >0.

→ During each pass through this loop, the address of the next list entry is determined and that entry is fetched & added to R0.

Let us see how to create & control a Pgm Loop

→ Assume that the no. of entries in the List is stored in memory location N as shown in figure 3.

→ Register R1 is used as a counter to determine the number of times the loop is executed.

→ contents of location N are loaded into register R1 at the beginning of the Pgm. With in the body of the loop, the instruction is given as,

Decrement R1

reduces the contents of R1 by 1 each time through the loop. execution of the loop is repeated as long as the result of the decrement operation is greater than Zero.

→ The branch instruction Loads a new value into the program counter.

→ The Processor fetches & executes the instruction at this new address, called the branch target.

→ A conditional branch instruction cause a branch only if a specified condition is satisfied.

→ The instruction | Branch >0 Loop | is a conditional

branch instruction that causes a branch to location Loop if the result of the immediately preceding instruction, which is the decremented value in Register R_1, is greater than zero.

→ This means, the loop is repeated as long as there are entries in the list that are yet to be added to R_0.

→ At the end of the n^{th} pass through the loop, the decrement instruction produces a value of zero & hence branching does not occur.

→ MOVes the final result from R_0 into Memory Location "SUM".

Condition Codes :-

The Processor keep track of ilf about the result of various operations for use by subsequent conditional branch instructions.

→ Accomplished by recording the required ilf in individual bits, often called condition code flags.

→ These flags are usually grouped together in a special processor register called the condition code register or status register.

FOUR commonly used flags are :-

N (Negative) :- Set to '1' if the result is negative; other cleared to 'zero'.

Z (Zero) :- Set to '1' if the result is 0; Otherwise cleared to zero.

V (Overflow) :- set to 1 if arithmetic overflow occurs otherwise cleared to zero.

C (Carry) :- set to 1 if a arithmetic Overflow carry-out results from the operation otherwise cleared to zero.

→ The N & Z flags indicate wheather the result of an arithmetic (or) logic operation is negative (or) zero.

→ The N and Z flags may also be affected by instructions that transfer data, such as Move, Load (or) store.

→ The V-flag indicates whether overflow has taken place & overflow occurs when the result of an arithmetic operation is outside the range of values.

→ The processor sets the V flag to allow the programmer to test wheather overflow has occured (or) not.

→ The c flag is set to '1' if a carry occure from the most significant bit position during an arithmetic operation.

Generating Memory Addresses:-

The purpose of the instruction block at loop is to add a different number from the list during each pass through the loop.

Def: The different ways in which the location of an operand is specified in an Instruction are referred as Addressing Mode.

TOPIC 3: Addressing Modes :-

→ An addressing mode is a method of specifying an operand. A computer program is a sequence of instructions.

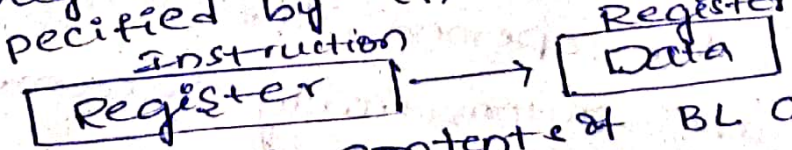
→ Each instruction specifies the operation to be performed in the operation code field on the data specified by the operands in the operand field.

EA - Effective address
value - signed no.

Generic Addressing modes		
Name	Assembler Syntax	Addressing function
Immediate	#value (a signed no between -128 to +127)	Operand = value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri), (LOC)	EA = [Ri], EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri, Rj)	EA = [Ri] + [Rj]
Base with index & offset	X(Ri, Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] + X
Auto increment	(Ri)+	EA = [Ri]; Increment Ri
Auto decrement	-(Ri)	Decrement Ri: EA = [Ri]

fig: Generic Addressing modes.

Register Mode :- In register addressing the operand is placed in one of 8 bit (or) 16 bit general purpose registers. The data is in the register that is specified by the instruction.

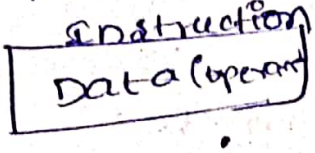


Ex: 1. MOV AL, BL; Content of BL are moved into AL

2. MOV AX, CX (move the contents of CX register to AX register)

Immediate Mode :-

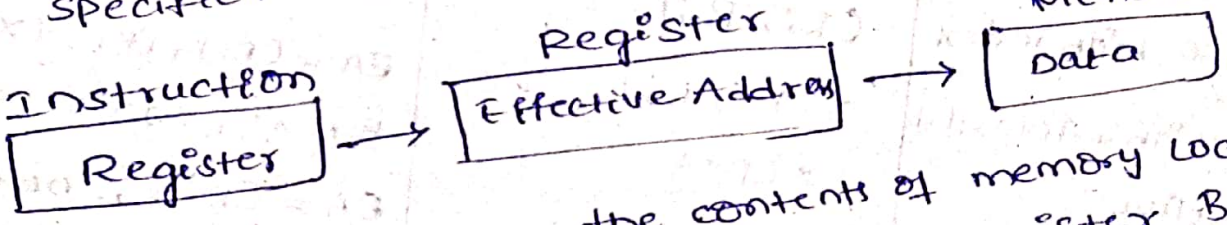
In Immediate addressing mode the operand is specified in the instruction itself. In this mode the data is 8-bits (or) 16 bits long and data is the part of instruction.



Eg:- MOV AL, 35H ; MOVe the data 35H in to AL register.

Register Indirect Mode :-

In this addressing the operands offset is placed in any one of the registers BX, BP, SI, DI as specified in the instruction.



Eg: MOV AX,[BX] ; [Move the contents of memory locations addressed by the register BX to the register AX].

Auto - increment Mode :-

Effective Address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of the register are automatically incremented to point to the next item in a list.

Auto increment mode is written as, (Ri)+

Eg: Add R1,(R2)+

Auto-decrement Mode :-

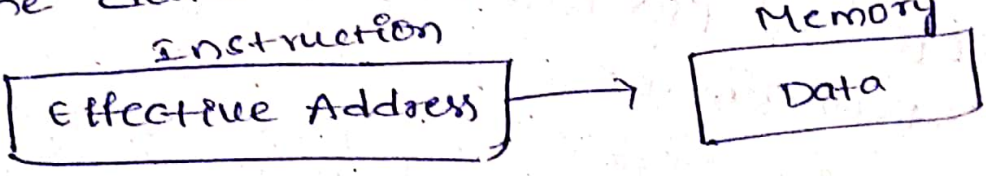
Effective Address of the operand is the contents of a register specified in the instruction. Before accessing the operand, the contents of this register are automatically decremented to point to the previously consecutive memory location.

→ Auto decrement mode is written as, -(Ri)

Eg: Add R1, -(R2)

Direct Mode :-

The operands offset is given in the instruction as an 8bit (or) 16bit effective address of the data is the part of the instruction.



Eg: ADD AL, [0301] ; add the content of offset address 0301 to AL register.

Base Addressing :-

In this type of addressing mode the 16bit effective address contents of BX (or) BP registers constitute the effective address of the instruction.

Eg: MOV CL, [BX]
MOV DX, [BP]
MOV AL, [BX+05]

In the above register, suppose BX contain 0301. The offset will be 0301+05 = 0306. i.e content of the memory location 0306 will move to AL.

Index addressing Mode :-

The operands offset is the sum of the content of an index register SI (or) DI

Eg: MOV AX, [SI+05]

Based Indexed Addressing :-

The operands offset is sum of the content of a base register BX (or) BP and an index register SI (or) DI.

Eg: ADD AX, [BX+SI]

Based Indexed Plus displacement Addressing Mode :-

In this mode of addressing the operands offset is given by Offset = [BX (or) BP] + [SI (or) DI] + 8 bit (or) 16 bit displacement.

Eg: MOV AX, [BX+SI+05]

~~Based~~ Branch Related Addressing Modes :-

Intra segment Direct :- The effective branch address is sum of 8 (or) 16 bit displacement and the current contents of IPC (Instruction Pointer). It can be used with either conditional (or) unconditional branching.

TOPIC 4:
Basic Input/Output Operations:-

→ consider a task that reads in character input from a keyboard & produces character output on a display screen.
→ The way of performing such I/O tasks is to use a method known as program-controlled I/O.
→ The rate of data transfer from the keyboard to a computer is limited by the typing speed of the user.
→ The rate of O/P transfers from the computer to the display is much higher.

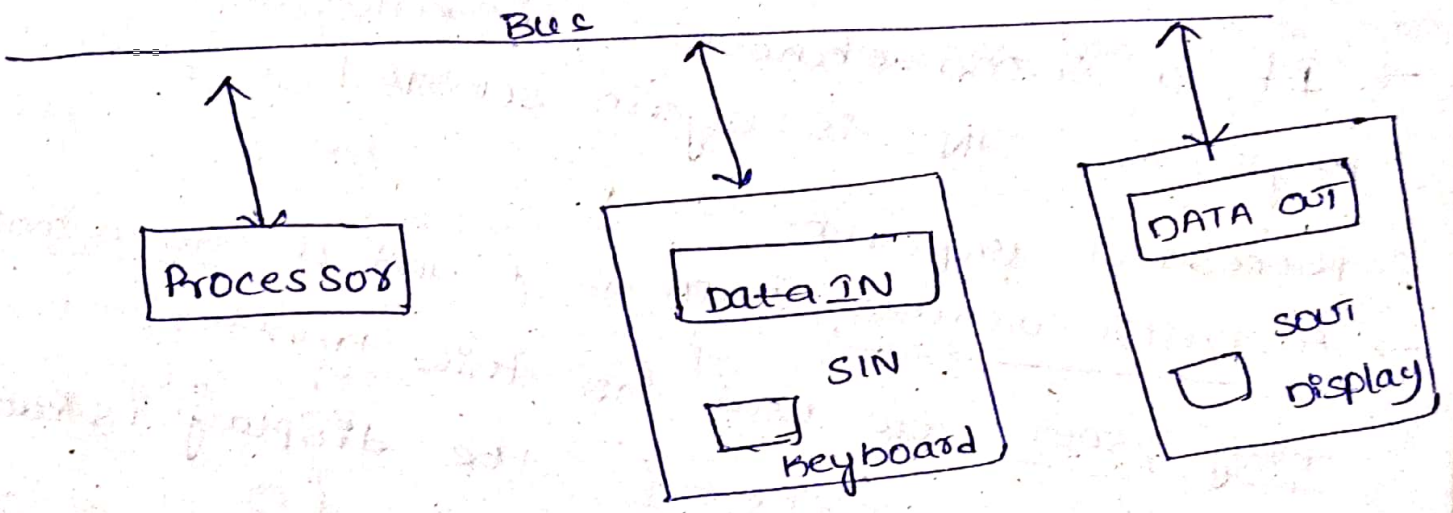


fig: Bus connection for processor, keyboard & Display

→ The keyboard & display are separate devices as shown in above figure.
→ Striking a key on the keyboard does not automatically cause the corresponding character to be displayed on the screen.
→ One block of instructions in the I/O program transfers the character in to the processor & another associated block of instructions causes the character to be displayed.

→ Striking a key stores the corresponding character code in an 8-bit buffer register associated with the keyboard.

→ The register DATA IN, inform the processor that a valid character is in DATA IN, a status control flag SIN is set to 1.

→ When SIN is set to 1, the processor reads the contents of DATA IN.

→ When the character is transferred to the processor SIN is automatically cleared to zero.

→ If a second character is entered at the keyboard SIN is again set to 1, & the Processor repeats.

→ A buffer register, DATA OUT, and a status control flag, SOUT are used for this transfer.

→ When SOUT equals to 1, the display is ready to receive a character.

Eg: The processor can Monitor the keyboard status flag SIN & transfer a character from DATAIN to register R_1 by the following sequence of operations.

```
READWAIT Branch to READWAIT if SIN=0
         Input from DATAIN to R1
```

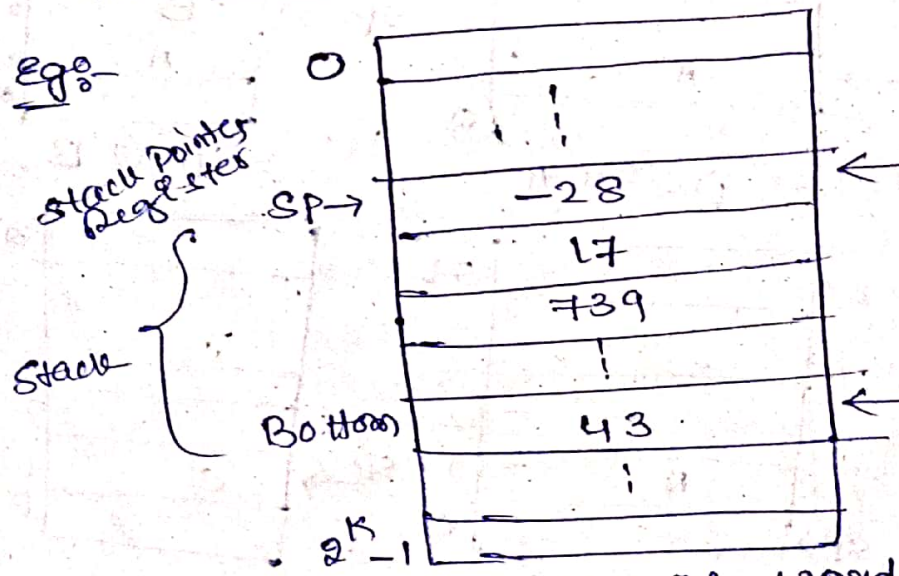
The operations is used for transferring O/P to the display is

```
WRITE WAIT Branch to WRITEWAIT if SOUT=0
           output from R1 to DATAOUT.
```

TOPIC 5: Stacks & Queues:-

→ A computer program often needs to perform a particular sub task using the familiar subroutine structure.
→ To organize & control the i/f b/w the main program & the subroutine, a data structure called a stack is used
→ A stack is a list of data elements, usually words, with the accessing instruction that elements can be added (or) removed at one end list.
→ One end is called the top of the stack, and the other end is called the bottom of the stack.
→ The LIFO (Last-in-first-out) stacks is used to describe the type of storage mechanism.
→ The last data item placed on the top of stack is the first one removed.
→ The terms push & pop are used to describe placing a new item on the stack & removing the top item from the stack.

Eg:-



A stack of words in the Memory.

→ The above figure contains Numerical values with 43 at the bottom & -28 at the top.
→ A processor register is used to keep track of the address of the element of the stack that is at TOP any

Eg: 2:

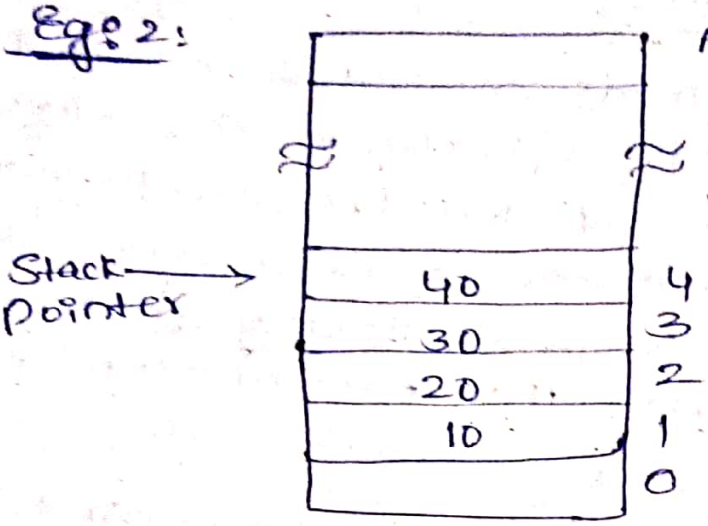


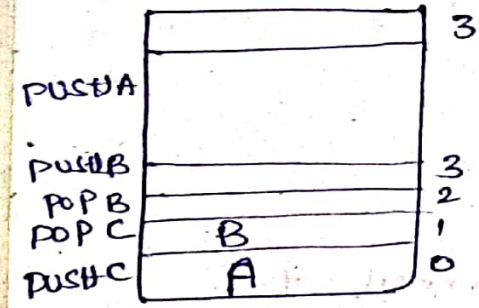
fig: 32-bit word Register stack

→ The Stack Pointer holds the address of the register that is currently the top of a stack.

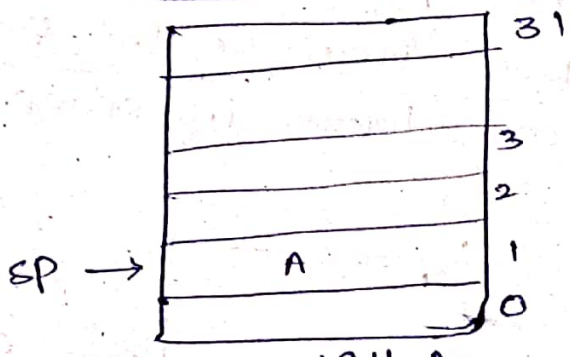
→ Initially it is clear to zero, & stack is said to be empty

→ If SP reaches 31 & one more data element is pushed then SP is incremented to '0' & data element is stored in the register '0'.

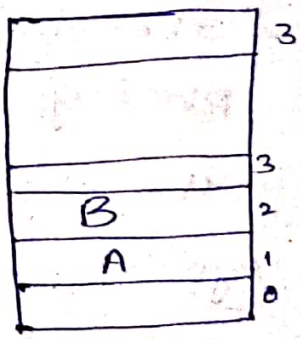
→ There is no more empty registers in the stack & it is said to be "FULL".



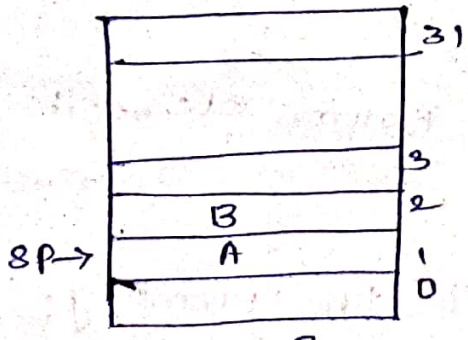
Initial status



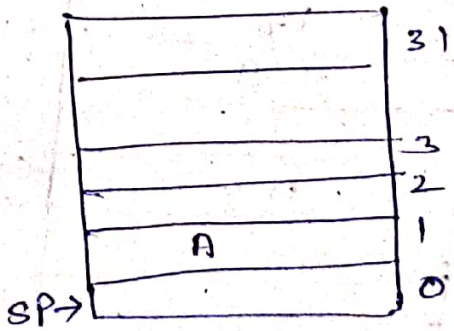
PUSH A



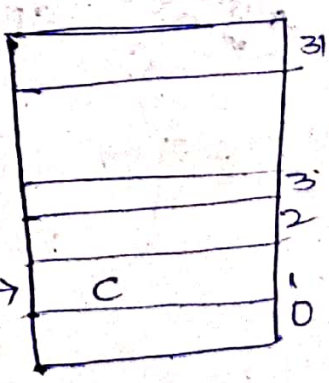
PUSH B



POP B



POP A



PUSH C

Queue:

→ Another useful ds that is similar to the stack is called "Queue".

→ Data are stored in & retrieved from a queue on a FIFO basis.

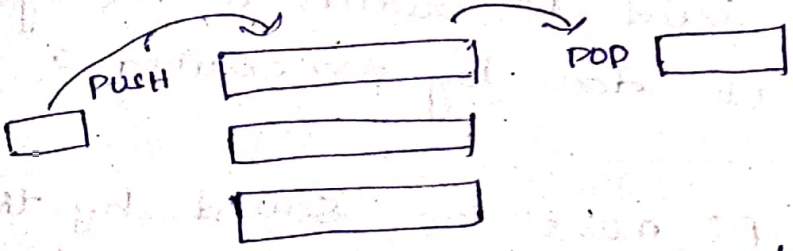
→ Assume that the queue grows in the direction of increasing address in the memory.

→ In which new data are added at the back (End) & retrieved from the front of the queue.

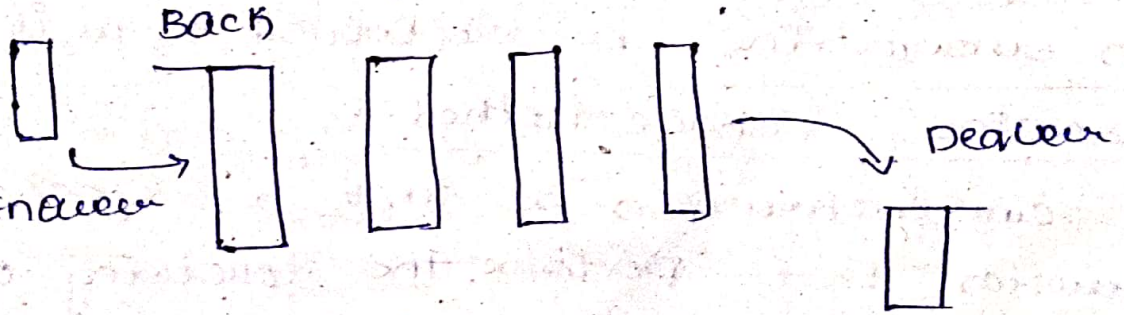
Basic differences b/w Stack and Queue:

Eg: Imagine a stack of paper. The first piece put in to the stack is at the bottom, so it is the last one to come out. This is LIFO.

→ Adding a piece of paper is called "pushing" & removing a piece of paper is called "popping".



→ Imagine a queue at the stores. The first person in line is the first person to get out of line & person getting out of line is "dequeued".



Topic 6: Sub routines

→ It is often necessary to perform a particular subtask many times on different data values.

→ Such a sub task is usually called a subroutine.

→ To reproduce the block of instructions that constitute a subroutine at every place where it is needed in the Pgm.

→ To save space, only one copy of this block is placed in the Memory & any Pgm that requires the use of the subroutine simply branch to its starting location.

→ When a Pgm branches to a subroutine we call it as the calling the subroutine.

→ The instruction that performs this branch operation is named a call instruction.

→ The subroutine is said to return to the Pgm that called it, & it does by executing a return instruction.

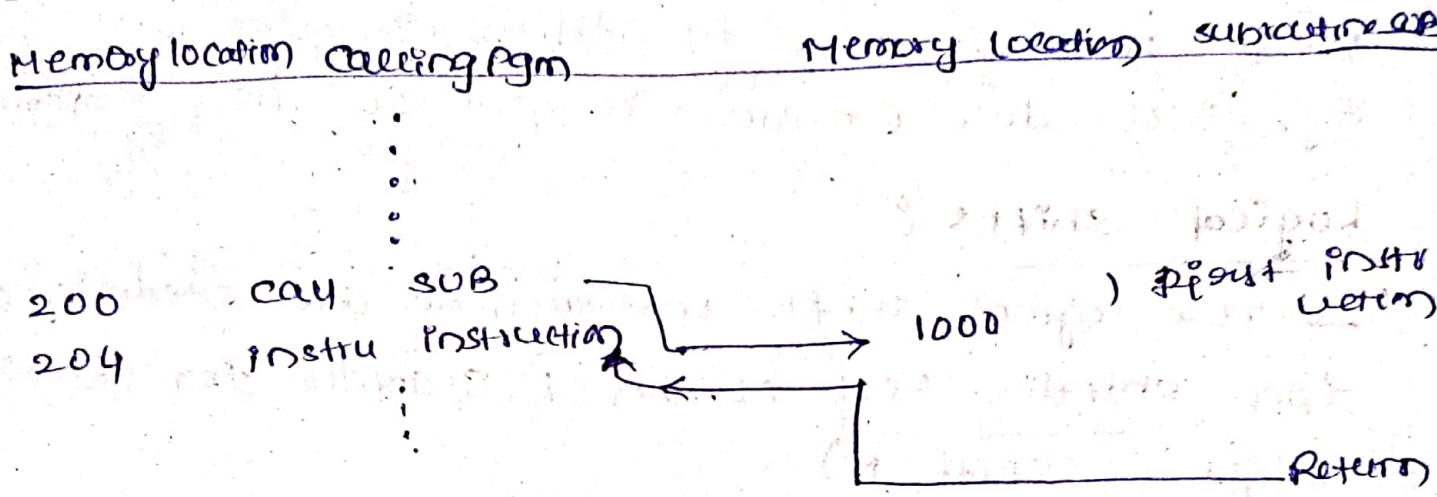
→ The content of the PC must be saved by the call instruction to enable correct return to the calling program.

→ The computer makes it possible to call & return from subroutine, is referred to as its subroutine Linkage Method.

→ The call instruction is just a special branch instruction that performs the following operations

1. Store the contents of PC in the link register
2. Branch to the target address specified by the instruction.

→ The Return instruction is a special branch instruction that performs the operation → Branch to the address contained in the link Register.



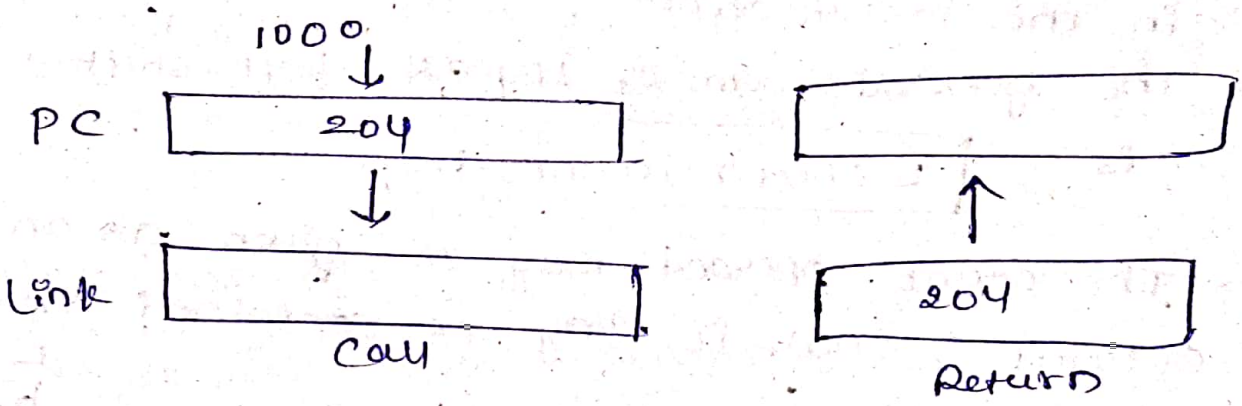
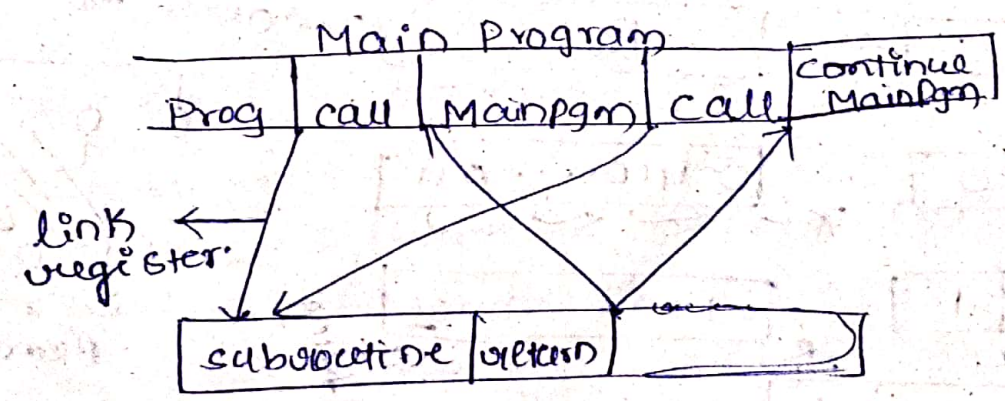


fig: subroutine linkage using a link register

Concept:



Topic 7: Additional Instructions

Logic Instructions:- Logic operations such as AND, OR and NOT applied to individual bits, are the basic basic building blocks of digital circuits.
Eg: Not dest (means it will do is complement)

Logical shifts:-

→ Two logical shift instructions are needed, one for shifting Left (LShiftL) & another for shifting right. (LShift R)

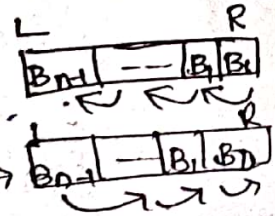
→ These instructions shift an operand over a Number of bit positions specified in a count operand contained in the instruction.

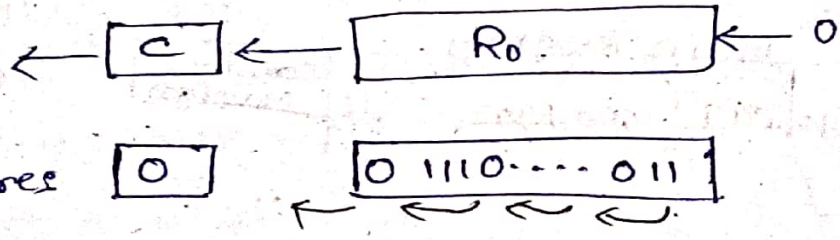
→ The general form of Logical Left shift instruction is

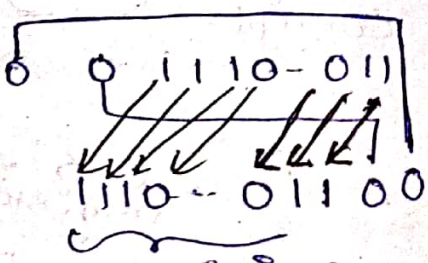
LShiftL, count, dest

→ The count operand may be given as an immediate operand, (or) it may be contained in a processor register.

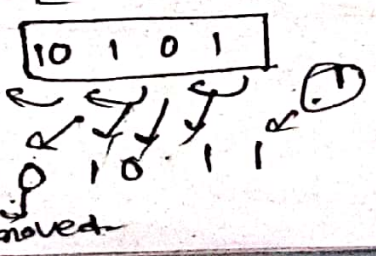
Eg: @Logical shift left LShiftL #2, R0.



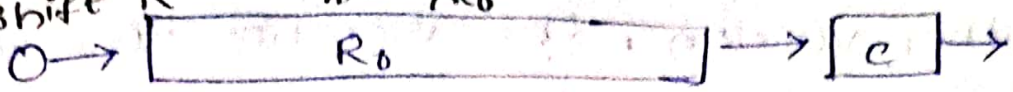




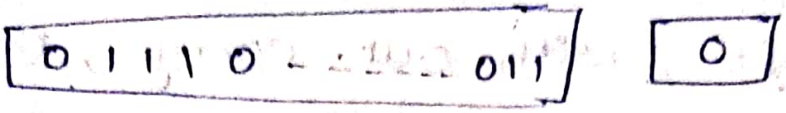
ex:



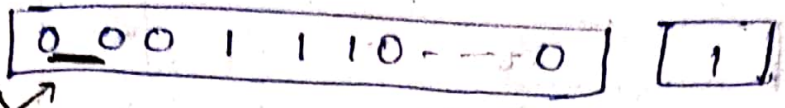
2. L shift R #2, R_0



before :



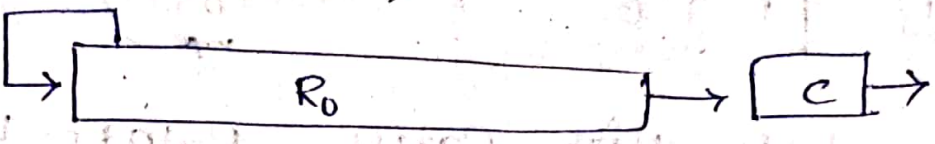
After :



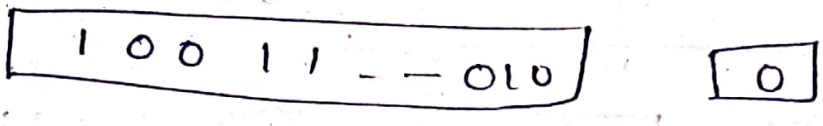
Adding 0 (b) Logical shift right.

1, 0 removed (correct)

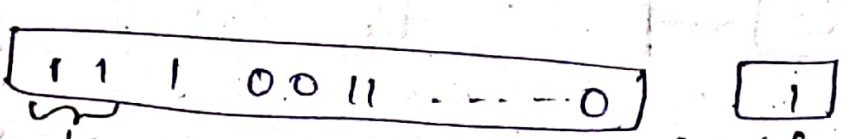
3. A shift R #2, R_0



before :



After :



↳ 1 Repeat two times bcz itself

00 removed

(c). Arithmetic shift right.

Rotate operations :-

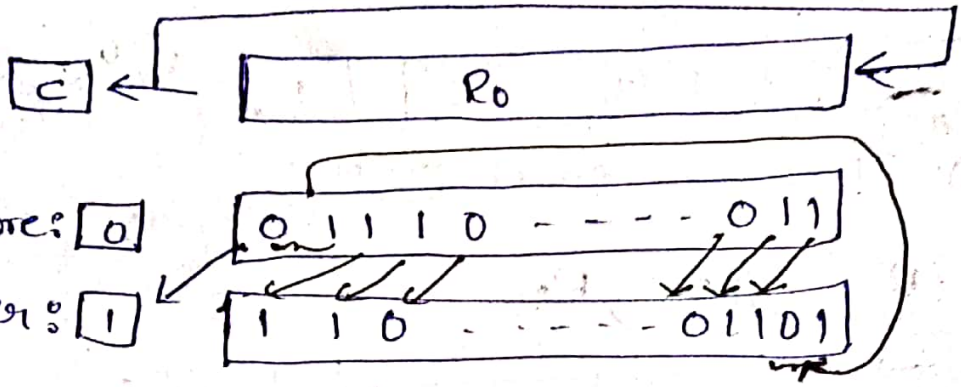
→ In the shift operations, the bits shifted out of the operand are lost, except for the last bit shifted out which is retained in the carry flag C. To preserve all bits, a set of rotate instructions can be used.

→ They move the bits that are shifted out of one end of the operand back in to the other end.

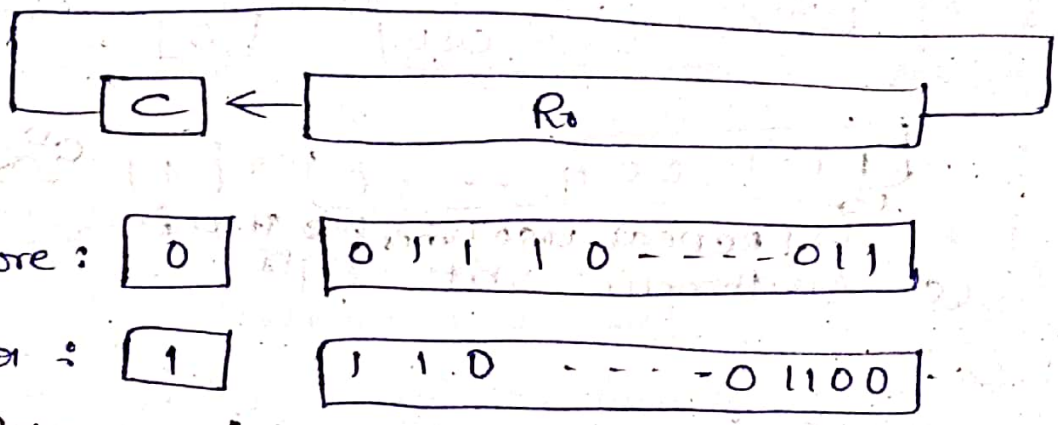
→ Two versions of both the Left & Right rotate instructions are usually provided. In one version, the bits of the operand are simply rotated.

In the other version, the rotation includes the 'C' flag.

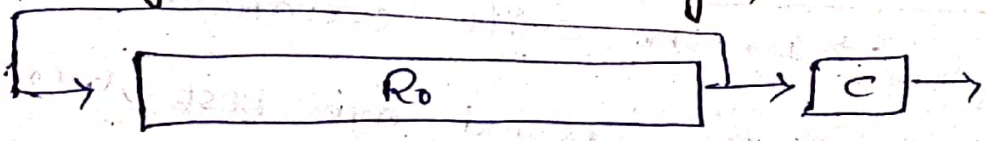
(a) Rotate Left with out carry, Rotate L #2, R0



(b) Rotate Left with Carry, Rotate LC #2, R0



(c) Rotate right with out Carry, Rotate R #2, R0



(d) Rotate Right with Carry Rotate RC #2, R0

Multiplication and Division :-

Two signed integers can be multiplied (or) divided by machine instructions with the same format :

The instruction,

Multiply Ri, Rj

performs the operation,

$$Rj \leftarrow [Ri] \times [Rj]$$

→ The signed integer Divide instruction,

Divide Ri, Rj

which performs the operation,

$$Rj \leftarrow [Rj] / [Ri]$$

placing the quotient in Rj. The remainder may be placed in R(j+1) (or) it may be lost.