COMPUTER ORGANISATION

INTRODUCTION.—MODULE-1

- Computer organization describes the function & design of various units of digital computers that store & process the information
- Most of which deals with the computer H/w & computer Architecture.
- Computer H/w is the electronic circuits and electromechanical equipment that constitutes the computer.
- Computer Architecture is defined as the functional operation of the individual H/w units in a computer system & the flow of information among and the control of those units.
- Computer or a digital computer is a fast electronic calculating machine which accepts digitized "input" information; processes it according to a "program" stored in its "memory" & produces the resultant "output information.
- Computers are classified based on size, speed & cost.
- The most common computers is the personal computer(PC) which has found wide use in homes, schools & Business offices (common form of Desktop computer) which have processing unit, visual display storage units & keyboard that can all be located easing on a home or office Desk.
- Storage media includes harddisks, CD-ROMs, diskettes,
- portable notebook computers are used mainly for word processing & easily fit into brief case.

- The keyboard, panel display and processor are all packaged into a single unit.
- Other than these a range of large & very powerfull computer system exists that are called mainframes at the low & of the range & super computers at the high end.
- Mainframes are typically used for business data processing in medium to large corporations.
- Servers contain sizable database storage units & are capable of handling large volumes of requests to access the data.
- Super computers are used for large-scale numerical calculations found in applications such as weather forecasting & aircraft design & simulations.
- Workstations still retaining the dimensions of desktop computers but have significantly more computational power than P.C.

Basic Functional Units

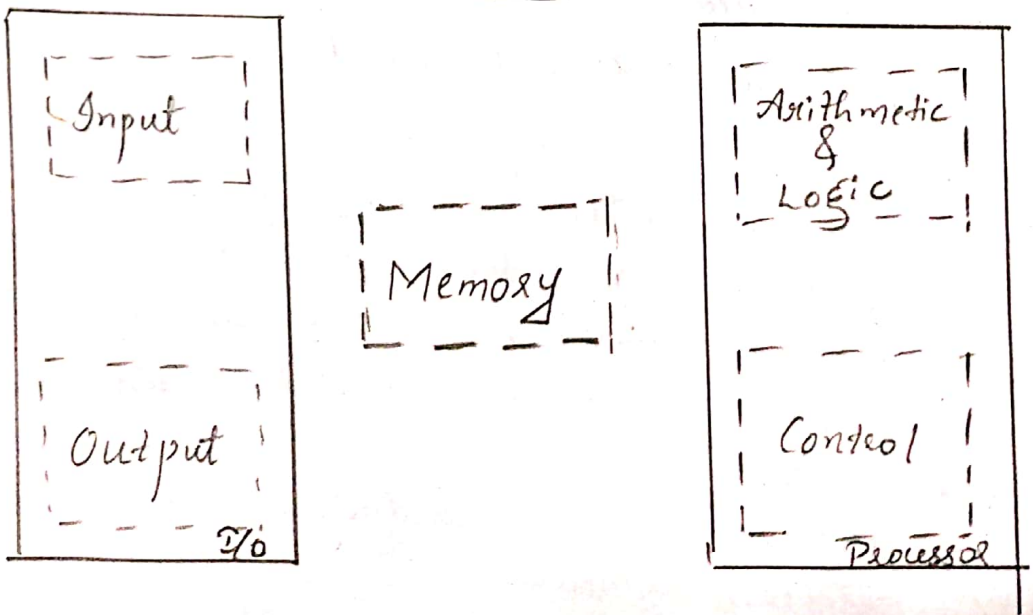


Fig: Basic Functional

- Digital computer in its simplest form consists of five functionally independent main parts
  - Input Unit
  - Memory Unit
  - Arithmetic & logic Unit
  - Output Unit
  - Control Unit. as shown in fig
- The input unit accepts the coded information from human operators, from electromechanical devices or from other computers connected to it over digital communication lines. The information is either stored in the memory for later reference or immediately handled by the arithmetic & logic circuitary which perfoms the desired operations. The processing steps are determined by a program stored in the memory. Finally the result are sent back to the outside world through o/p unit. All these actions are co-ordinated by the control Unit.
- The arithmetic and logic circuits in conjunction with the main control circuits is referred to as control processing unit (CPU) or processor.
- Information fed into the computer is categorized as instructions or data.
- Instructions mainly
  - Govern the transfer of information within a computer as well as between the computer & its I/O devices.
  - Specify the arithmetic & logic operations to be performed.

- A set of instructions that perform a task is called program. Usually the program is stored in the memory. The processor then fetches the instructions that make-up the program from memory, one at a time & performs the desired operation.
- Data are numbers & encoded characters that are used as operands by the instructions.

• Input Unit

- Computer accepts coded information through i/p units which read the data. The most well known i/p device is the keyboard of a Video terminal, which is electronically connected to the processing part of computer.
- Other i/p devices available are joy sticks, track balls & mice.

• Memory Unit

- The function of memory unit is to store programs & data. There are 2 types of memory devices called primary & secondary storage.
- Primary Storage or Main memory is a fast memory that operates at electronic speed. Programs are stored in main memory while they are being executed.
- The memory contains a large no. of semiconductor storage cells, each capable of storing one bit of information.
- The memory is organized so that contents of one word, containing n-bits, can be stored or retrieved in one basic operation.

- To access any word in main memory a distinct address is associated with each word location.
- Addresses are numbers that identify successive locations. A given word is accessed by specifying its address & issuing a control command that starts the storage & retrieval process.
- The no. of bits in each word is often referred to as word length of the computer. Typical word lengths range from 16 to 64 bits.
- Although primary storage is essential, it tends to be expensive. Thus additional cheaper secondary storage is used, when large amount of data has to be stored, particularly if some of the data need not be accessed frequently

Example: Magnetic disks & tapes.

- ALU - Arithmetic & logic Unit

- Most of the computer operations are executed in arithmetic & logic unit of the processor.
- The control & arithmetic logic unit are many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as keyboards, displays magnetic & optical disks, sensors & mechanical controllers

- Output Unit

- The o/p Unit is the counter part of the i/p unit. Its function is to send processed result to the outside world.

Example: Printer.

- Some units such as graphic displays provide both an o/p function and an input function.

Control Unit

- The control unit is effectively the nerve centre that sends control signals to other units & sense their states.
- The data transfer betⁿ the processor & memory are controlled by the control unit through timing signals.
- The operation of computer can be summarized as follows:
  - The computer accepts information in the form of programs & data through an i/p unit & stores it in the memory.
  - The information stored in the memory is fetched under program control, into an arithmetic & logic unit course it is processed
  - Processed information leaves the computer through an o/p unit.
  - All activities inside the machine are directed by the control unit.

Basic Operational Concepts

- The basic function of computer is to execute program, sequence of instructions.
- It is known that the activity within a computer is governed by means of instructions.
- To perform a given task, an appropriate program consisting of a set of instructions is stored in the main memory.

- Individual instructions are brought from the memory into the processor, which executes the specified operations.

- Consider an instruction,

Add LOCA, R0

which adds the operand at memory location LOCA to the operand in a register in the processor called R0, and places the sum into register R0.

- This instruction requires,
- 1st, the instruction must be transferred from the main memory into the processor.
- Then the operand from LOCA must be fetched.
- This operand is added to the contents of R0.
- Finally, the resultant sum is stored in register R0.
- The data is transferred from or to the memory by sending the address of the memory location to be accessed to the memory unit & issuing the appropriate control signals.
- Figure below shows various connection between main memory & processor.

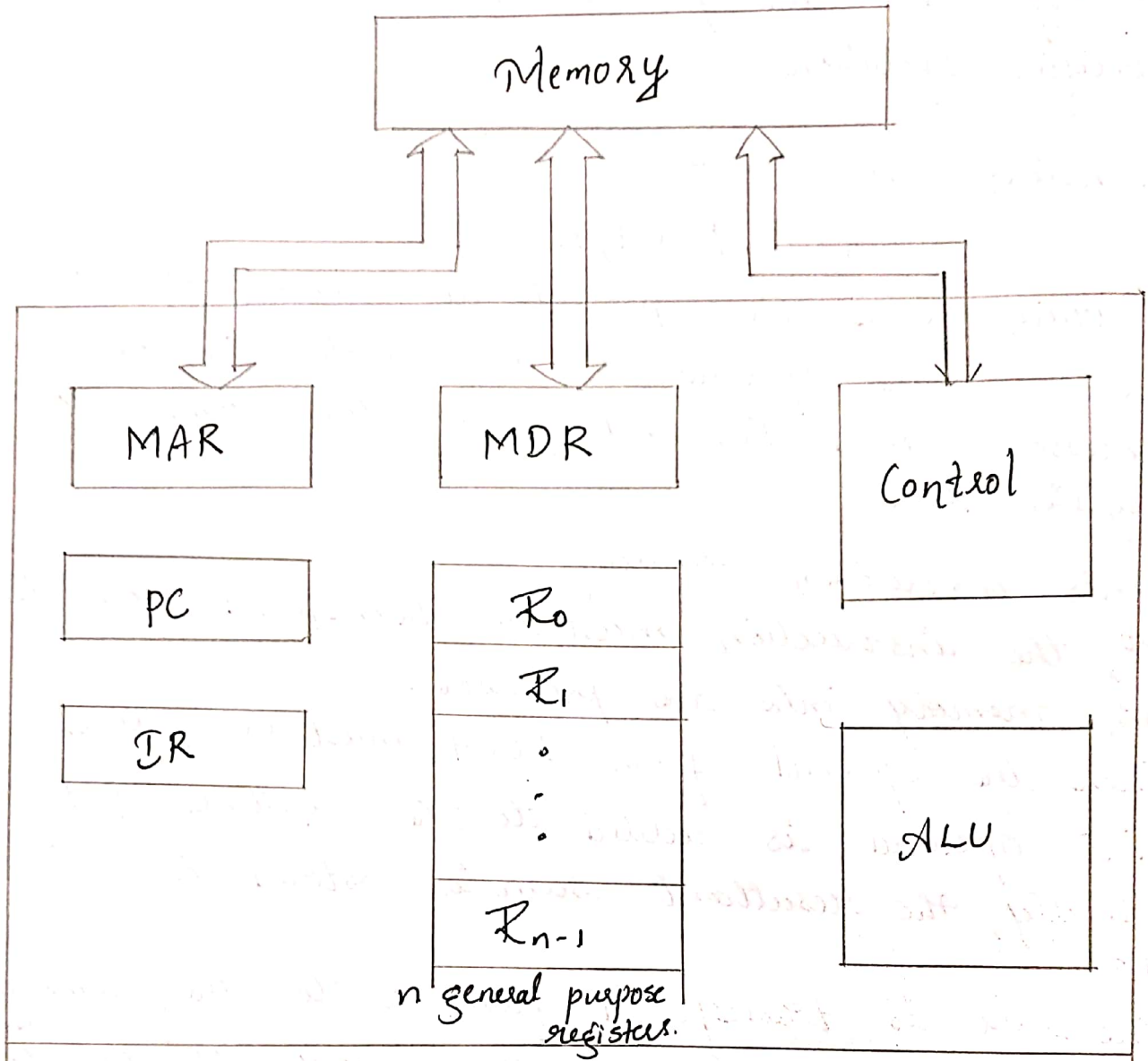


Fig: Connections bet^n the processor & the memory.

- To perform execution of instruction, the processor contains arithmetic & logic circuitry as the main processing elements.
- It also contains a number of registers used for temporary storage of data as shown above which include IR, PC, MAR and MDR.

- IR (Instruction register).
- Contains the instruction that is being executed.
- The contents of IR are available to the control unit, which generates the timing signals.

PC (Program Counter)

- PC is a register which keeps track of the execution of a program.
- It contains memory address of the instruction currently being executed.
- During the execution of current instruction, the contents of PC are updated to correspond to the address of the next instruction to be executed.
- Besides the IR & PC there are usually several other general purpose registers.
- Finally there are 2 registers that facilitate communication with the main memory. These are memory address register (MAR) and memory data register (MDR).
- The MAR is used to hold the address of location to or from which data is to be transferred.
- The MDR contains the data to be written into or read out of the addressed location.

Operation:

- Programs reside in the main memory through i/p unit.
- Execution of a program starts by setting the PC to point to the first instruction of the program.
- The contents of PC are transferred to MAR & read control signal is sent to the memory.
- After some time (memory access time), the i.e the time required to access the memory elapses, the

addressed word is read out of the memory & loaded into MDR.

- Next the contents of MDR are transferred to IR, at which point the instruction is ready to be decoded & executed.
- If the instruction involves in operation to be performed by the ALU, it is necessary to obtain the required operands.
- If an operand resides in the memory [it could also be in GPR (general purpose register) in the processor], it has to be fetched by sending its address to the MAR & initiating a read cycle. when the operand has been read from memory into the MDR, it can be transferred from MDR to the ALU.
- If the result of this operation is to be stored in the memory, it must be sent to the MDR.
- The address of the location where the result is to be stored is sent to the MAR & write cycle is initiated
- In the mean-time the contents of PC are incremented to point to the next instruction to be executed
- In addition to transfering data betn main memory & processor it is necessary to have the ability to accept data from i/p devices & to send data to o/p devices.

- It is often the case that some device requires immediate servicing during the normal execution.
- Example: A monitoring device in a computer-controlled industrial process may have detected a dangerous condition.
- To deal with such condition quickly, the normal flow of the program that is being executed by the processors must be interrupted.
- To achieve this, the device can raise an interrupt signal.
- An interrupt is a request from the I/o device for service by the processor.
- The processor provides the requested service by executing an appropriate interrupt-service routine.
- During this condition, it is essential that its state be saved in the main memory before servicing the interrupt.
- This normally involves storing the contents of the PC, the general registers, & some control information.
- Upon termination of the interrupt service routine, the state of the processor is restored.

BUS STRUCTURE

- It is known that CPU, memory Unit & I/o units are Hardware company or modules of the computer.
- To form an operational system, these parts must be connected in organized way.

- To achieve a reasonable speed of operation, a computer must be organized so that all the units can handle one full word of data at a given time.
- When a word of data is transferred between units, all its bits are transferred in parallel, i.e the bits are transferred in HW, simultaneously over many wires or lines, one bit per line.
- A group of lines that serves as a connecting path for several devices is called a Bus.
- A bus that connects major computer components/ module is called system bus.
- The system bus is separated into three functional groups:
  * Data Bus
  * Address Bus
  * Control Bus
- The data bus consists of 8, 16, 32 or more parallel signal lines.
- The data bus lines are bi-directional i.e CPU can read data on these lines from memory or from a port, as well as send data out of these lines to a memory location or to a port.
- The address bus is an Uni-directional bus. The address bus consists of 16, 20, 24 or more HW signal lines. On these lines the CPU sends out the address of the memory location or I/O port that is to be written or read from.

- The Control bus (bi-directional) regulate the activity on the bus. The CPU sends signals on the control bus to evolve the o/p's of the addressed memory devices or port devices.

Example: Memory Read, Memory Write, I/o Read, I/o Write, Bus request, Bus Grant, Interrupt request, Resel, Hold etc.

1. Single Bus Structure

- In a single bus structure all units are connected to common bus called system bus.
- With single bus, only two unit can communicate with each other at a time.
- The bus control lines are used to arbitrate multiple requests for the use of the bus.
- The main advantage of single bus is its low cost & its flexibility for attaching peripheral devices.

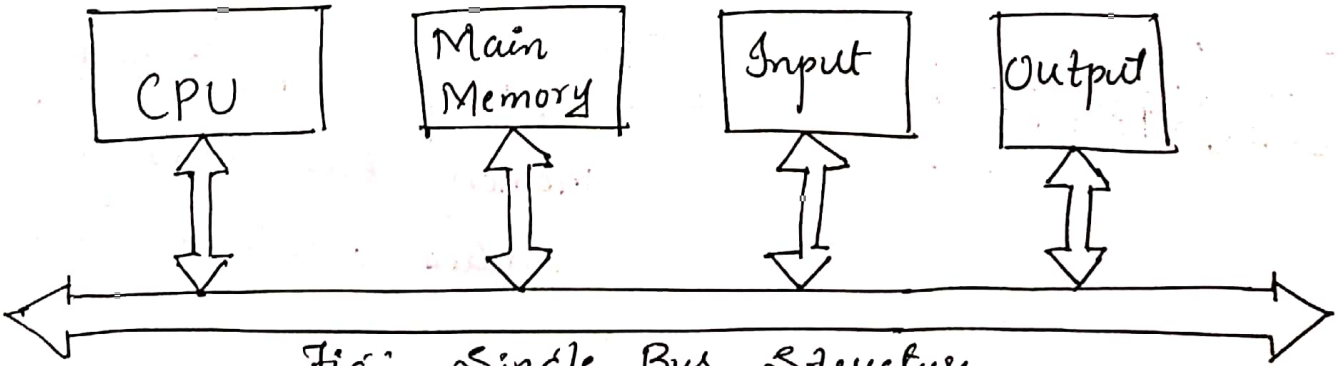


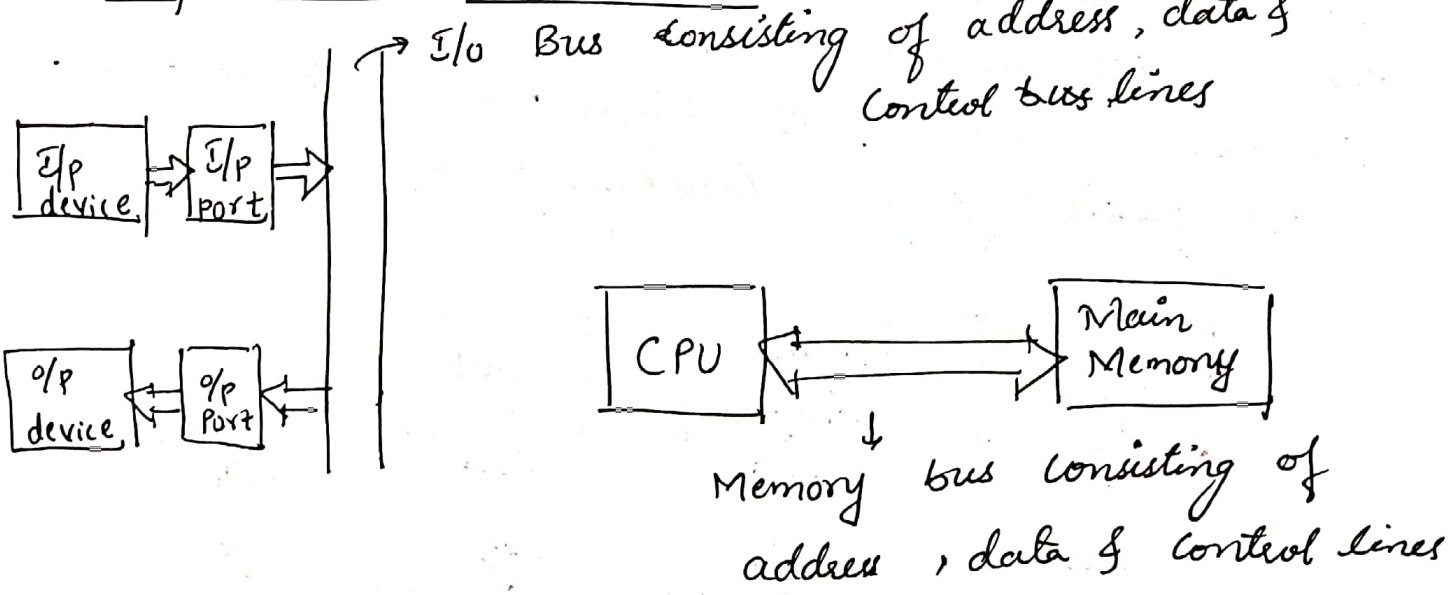
Fig: Single Bus Structure

- The disadvantage is slow speed.
- It is used in small machines like microcomputer & small mini computers
- CPU & Main memory are very fast compared to electromechanical i/p devices such as keyboards, printer etc.

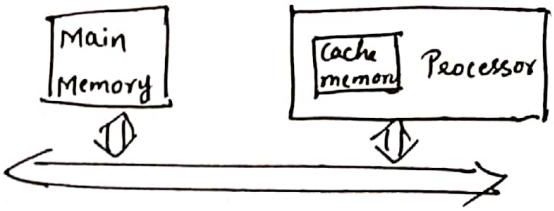
- It is necessary that the single bus is not kept too busy for long time while communicating with I/O devices.
- To achieve this, I/O devices are provided with buffer registers or ports to hold the information during data transfer.
- Now, this happens at electronic speed.

Example: To communicate with printer the CPU loads the print buffer at electronic speed.
- This slowly outputted by printer when printer has finished printing, it requests CPU for further data.

2. Simple Two-Bus Structures



In this case, there are two buses, one for I/O operations & the other for memory operations.
- So CPU can communicate with memory & I/O simultaneously.
- Thus it is faster. It is used in large minicomputers & mainframes.



Performance

- The performance of a computer is measured based on how quickly it can execute programs.
- The speed with which a computer executes program is affected by the design of its H/w & its machine language instructions.
- Because programs are usually written in a high-level language, performance is also affected by the compiler that translates programs into machine language.
- For best performance, it is necessary to design the compiler, the machine instruction set, & the H/w in a coordinated way.
- Program inst's are stored in memory. As execution proceeds, inst's are fetched into processor & copy is placed in cache. Later if same Instr is needed 2nd time, it is read directly from cache.

Processor Clock

- In today's digital computer, the CPU or processor is driven by a clock with a constant cycle time called processor clock & is denoted by P.
- To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in a one clock cycle.
- The period P of one clock cycle is an important parameter that affects processor performance.
- The clock rate is given by $R = \frac{1}{P}$ which is measured in cycles per second (CPS).

- The standard unit for this measurement is Hertz (Hz).
- Today's PC & workstations have clockrate in the range of MHz & GHz.
- The computers having clock rate of 800 MHz have 800 million cycles per second.

Basic Performance Equation

- Let T be the processor time required to execute a program that has been written in some high level language.
- For the execution of the program, processor has to execute number of machine language instructions.
- This number is denoted by N.
- The number N is the actual number of instruction executions & is not necessarily equal to the number of machine instructions in object program.
- This is because some instructions may be executed more than once in the loop & others may not be executed at all.
- Each machine instruction takes one or more cycle time for execution.
- This time is required to perform various steps needed to execute machine instruction.
- The average number of basic steps required execute one machine instruction is denoted by S,

where each basic step is completed in one clock cycle.

- Thus the program execution time is given by

$$T = \frac{N \times S}{R}$$

where $N \rightarrow$ actual number of instructions executed by the processor for execution of program

$R \rightarrow$ clock rate measured in clocks per second

$S \rightarrow$ average number of steps needed to execute one machine instruction.

$T \rightarrow$ Processor time required to execute a program.

- The performance parameter T for an application Program is much more important to the user than the individual values of parameters N, S & R.
- To achieve high performance, the computer designer must seek ways to reduce the value of T, which means reducing N & S & increasing R.
- The value of N is reduced if the source program is compiled into fewer machine instructions.
- The value of S is reduced if instructions have a smaller number of basic steps to perform or if execution of instruction is overlapped.
- Using a higher frequency clock increases the value of R which means that the time required to complete a basic execution steps is reduced.

- But if it is known that N, S, & R are not independent parameters, changing one may affect other.
- With the new design of processor will lead to improved performance only if the overall result is to reduce the value of T.
- A processor advertized as having 900MHz clock does not necessarily provide better performance than a 700 MHz processor because it may have a different value of S.

Clock Rate

- The Clock rate of the processor can be increased by

* Improving IC technology which makes the logic circuits faster, reducing the time required for basic operation. This reduces the clock period P & thus increases the clock rate R.
* Reducing the amount of processing done in one basic step also makes it possible to reduce the clock period P & hence increases clock rate. However if the actions that have to be performed by an instruction remains the same, the number of basic steps needed may increase.

- Increase in clock rate of processor increases the speed of processor operations, however increase in the clock rate does not improve the memory access time. To reduce the memory access time it is necessary to use improved memory technology. Use of cache memory substantially improves the memory access time.

Instruction Rate

- Generally simple instructions requires a small number of basic steps to execute. Complex instructions involve a large number of steps.
- For a processor that has only simple instructions, a large number of instructions may be needed to perform a given programming task.
- This could lead to a large value of N & a small value of S.
- On the other hand if the individual instructions perform more complex operations, fewer instructions will be needed, leading to a lower value of N & a large value of S.
- The processor with simple instructions are called RISC & processor with more complex instructions are called as CISC.

RISC

1) Simple instructions taking one cycle
2) Instructions are executed by Hardwired control unit
3) Few instructions
4) Fixed format instruction
5) Few addressing modes, & most instructions have register to register addressing mode
6) Multiple register set
7) Highly pipelined

CISC

1) Complex instruction taking multiple cycle.
2) Instructions are executed by microprogrammed control unit.
3) Many instructions
4) Variable format instruction
5) Many addressing modes.
6) Single register set
7) No pipelined or less pipelined

Performance Measurements

- Bench mark refers to a standard task used to measure how well a processor operates.
- The performance measure is the time taken by a computer to execute a given benchmark.
- SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC - System Performance Evaluation Corporation) A non-profit organization. SPEC selects & publishes bench mark.
- The SPEC rating is computed as follows

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test.}}$$

- Thus a SPEC rating of 50 means that the computer under test is 50 times as fast as the UltraSPARC10 for this particular benchmark.
- The test is repeated for all the programs in the SPEC suite, & the geometric mean of the results is computed.
- Let $SPEC_i$ be the rating for program i in the suite.
- The overall rating for the computer is given by

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} SPEC_i \right)^{1/n}$$

where $n \rightarrow$ number of programs in the suite.

Problem

- There are 5 processors, P1 is consuming 50μsec & every successive process consumes double the time of previous process. Calculate SPEC rating of each process & SPEC of entire suite (Assume reference system which can execute P1 in 100μsec & each successive process with increase of 50μsec).

→ No of processes = 05.

$T_{exe}(P1) = 50\mu sec$

$T_{exe}(P_2) = 100\mu sec$

$T_{exe}(P_3) = 200\mu sec$

$T_{exe}(P_4) = 400\mu sec$

$T_{exe}(P_5) = 800\mu sec$

$$SPEC\ rating = \frac{Running\ time\ on\ the\ reference\ Computer}{Running\ time\ on\ Computer\ Under\ test.}$$

$$SPEC(P1) = \frac{100\,ms}{50\,ms} = 2$$

$$SPEC(P2) = \frac{150\,ms}{100\,ms} = 1.5$$

$$SPEC(P3) = \frac{200\,ms}{200\,ms} = 1$$

$$SPEC(P4) = \frac{250\,ms}{400\,ms} = 0.625$$

$$SPEC(P5) = \frac{300s}{800\,ms} = 0.375$$

$$SPEC(suite) = \sqrt[5]{2 \times 1.5 \times 0.625 \times 0.375}$$

$$= (0.703125)^{1/5}$$

$$= 0.932.$$

2) If the length of the clock cycle (P) = 2ns Then Calculate the clock rate

$$\Rightarrow R = \frac{1}{P} = \frac{1}{2 \times 10^{-9}} = 0.5 \times 10^9 = 500\,MHz$$

R is 500 million cycles per second.

3) If the length of clock cycle (P) is 0.8ns the calculate the clock rate

$$\Rightarrow R = \frac{1}{P} = \frac{1}{0.8 \times 10^{-9}} = 1.25 \times 10^{9} = 1.25\ GHz$$

R is 1.25 billion cycles per second.

4). A program contains 1000 instructions. Out of that 25% instructions require 4 clock cycles, 40% instructions require 5 clock cycles & remaining require 3 clock cycles for execution. Find the total time required to execute the program running in a 1GHz machine.

$\Rightarrow N = 1000$

25% of N = 250 instructions require 4 clock cycle

40% of N = 400 " " 5 "

35% of N = 350 " " 3 "

$$T = \left(\frac{N \times S}{R}\right)$$

$$= \frac{(250 * 4) + (400 * 5) + (350 * 3)}{1 \times 10^{9}}$$

$$= \frac{1000 + 2000 + 1050}{1 \times 10^{9}}$$

$$= 4.05\mu s.$$

Multiprocessor & Multicomputers

- To achieve high performance in large computer system multiple processors are used.
- Such systems are known as Multiprocessor systems.
- These systems execute a number of different application tasks in parallel or they execute subtasks of a single large task in 11^{el}.
- In multiprocessor systems many times the memory is shared between the processors i.e all the processor have the access to entire memory in the system.
- Such multiprocessor system is known as shared memory multiprocessor system.
- In this system multiple processors & memory units are interconnected with complex interconnection networks.
- This increases the cost of the system.
- Hence higher performance is obtained at higher cost of the system.
- In contrast to multiprocessor system, it is also possible to use an interconnected group of complete computers to achieve high total computational power such systems are called multicomputer systems.
- In this type of systems the computers have

access only to their own memory units.
- When tasks are executing need to communicate data or to access data from other computers, they used message-passing mechanism to exchange the information.
- Message passing mechanism exchanges the messages over a communication n/w.
- Such computer systems are known as Message passing multicomputers.

Memory Locations & Addresses

- Numbers & characters operands as well as instructions are stored in the memory of a computer.
- The memory consists of millions of storage cells each of which store a bit of information having the value of 0 or 1.
- The memory is organized so that a group of n-bits can be stored or retrieved in a single basic operation.
- Each group of n-bits is referred to as a word of information & n is called word length.
- Modern computer have word lengths that typically range from 16 to 64 bits.

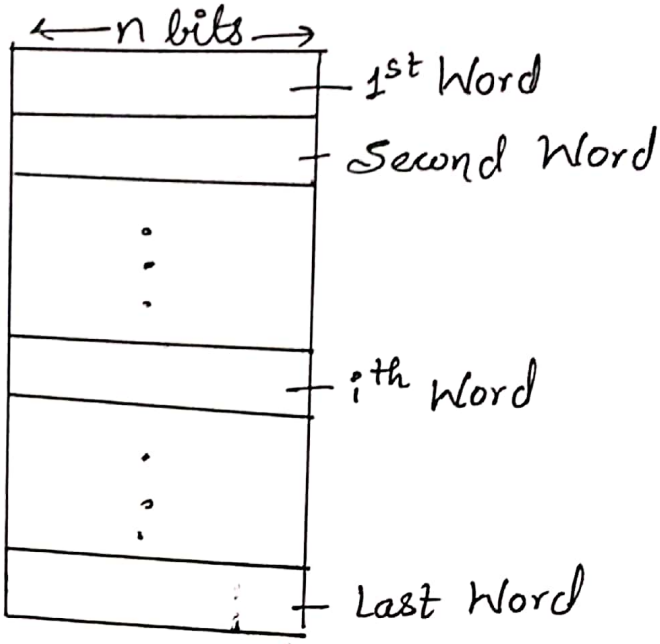


fig: Memory Words.

- If the Word length of a computer is 32-bits a single word can store a 32-bit 2's complement number.
- A unit of 8-bit is called a byte.
- Machine instructions may require one or more words for their representation.
- Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires a distinct name or addresses for each item location.
- Generally, 2^K address constitute the address space of the computer & the memory can have upto 2^K addressable locations.

Example: 24-bit address generates a address space of 2^{24} (16,777,216) locations. This is usually represented as 16 Megabit [1 Mega = 2^{20} = 1,048,576]

- A 32-bit addresses creates an address space of 2^{32} or 4G locations where $1G = 2^{30}$.
- Other notational convention that are commonly used are K for number of 2^{10} [1,024] & T (Tera) for number 2^{40}.

Memory Capacity	Address lines Required
1K = 1024 mem loc	10
2K = 2048 "	11
4K = 4096 "	12
8K = 8192 "	13
16K = 16384 "	14
32K = 32768 "	15
64K = 65536 "	16

Byte Addressability

- It is known that word-length of 8-bit is known as byte
- However word length typically ranges from 16 to 64 bits are simply called word.
- It is impractical to assign distinct address to individual bit locations in the memory.
- ∴ The most widely used way is to assign each byte location of word with a unique address.

- That is each successive byte location is addressed with byte 0, byte 1, byte 2 & so on.
- Thus if 32 bits, a successive words are located at addresses 0, 4, 8, ... with each word consisting of 4 bytes.

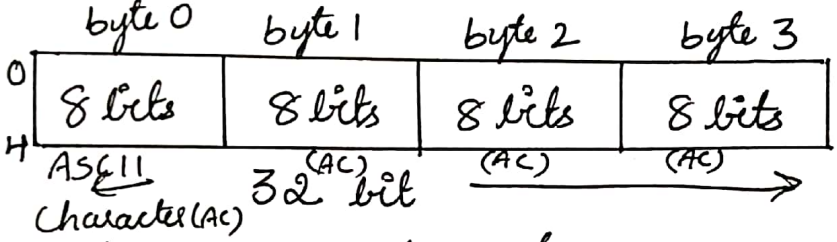


Fig: Four Characters.

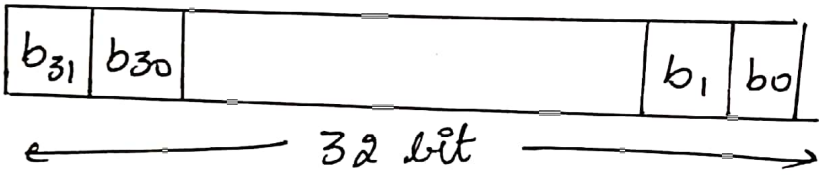


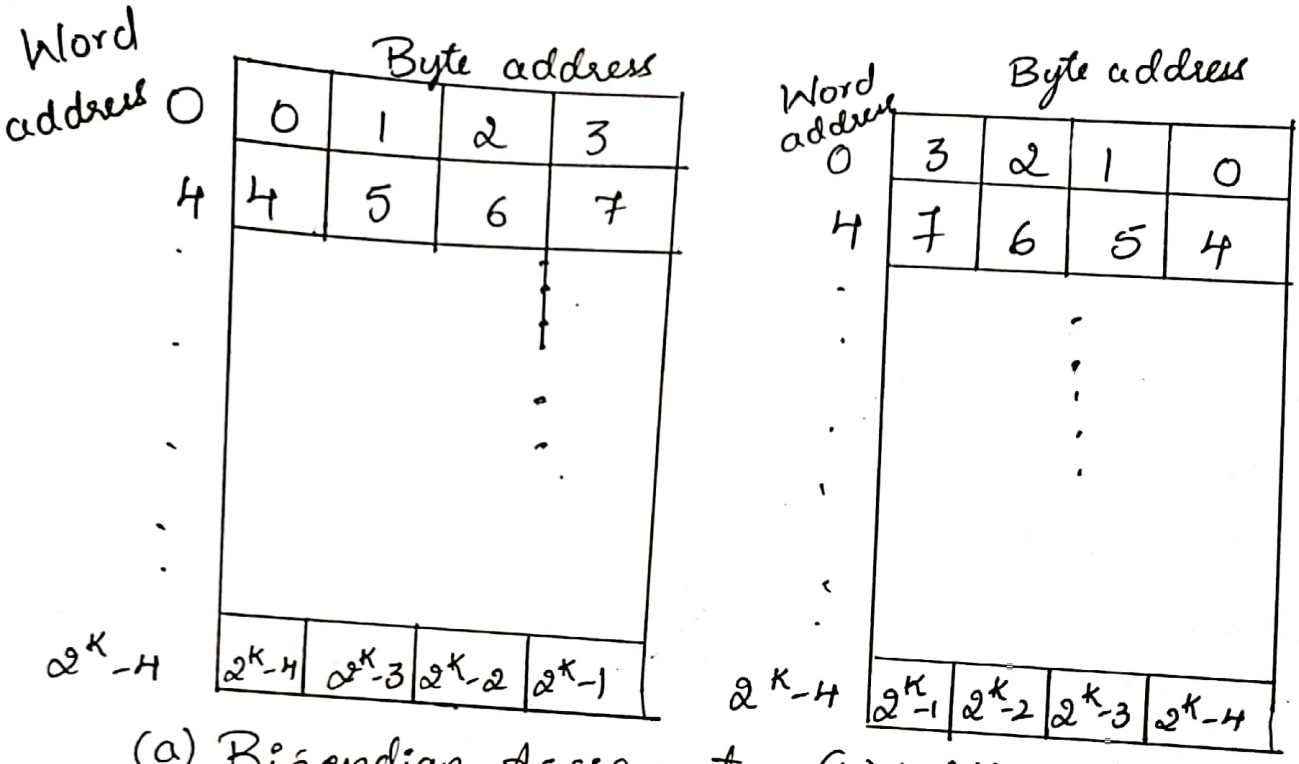
Fig: A Singned Integer.

$b_{31} = 0$ for +ve Number

$b_{31} = 1$ for -ve Number.

Big Endian & Little Endian Assignments

- There are two ways that byte addresses can be assigned across words as shown below.
- The name big endian is used when lower byte addresses are used for the MSB (left most bytes) of the word.
- The name little endian is used for the opposide ordering, where the lower byte addresses are used for the less significant bytes (the right most bytes) of the word



(a) Bigendian Assignment (b) Little endian Assignment.

Fig: Byte & Word addressing.

- In addition to specifying the address ordering of bytes within a word, it is also necessary to specify the labeling of bits within a byte of a word.
- The most common convention is labeling the bits within a byte i.e, $b_7, b_6 \cdots b_0$ from left to right.

Word Alignment

- Generally word locations are aligned addresses.
- In general words are said to be aligned in memory if they begin at a byte address that is multiple of the number byte in a word.
- Hence, if the word length is 16 (2 bytes) aligned words begin at byte addresses

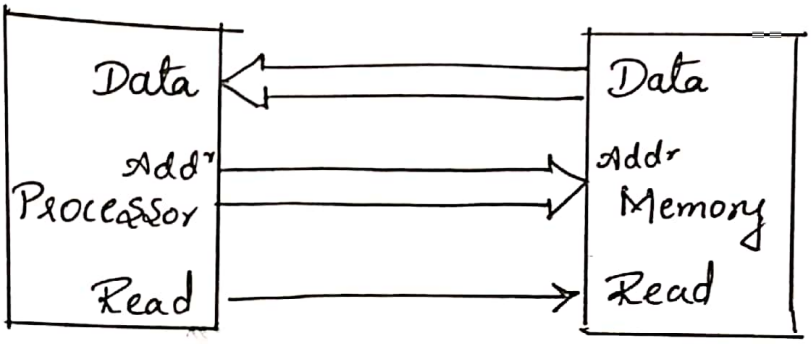
0, 2, 4, 8, 12 & for a word length of 64 (2^3 bytes) aligned words begin at byte 0, 8, 16, 32

Memory operations

- It is known that both program instructions and data operands are stored in the memory.
- To execute an instruction, processor has to fetch instruction & read the operand or operands if necessary from the memory.
- After execution of instruction processor may store the result or modified operand back in the memory.
- Thus there are two basic operations required in memory access
  1. Load [or Read or Fetch]
  2. Store [or Write].

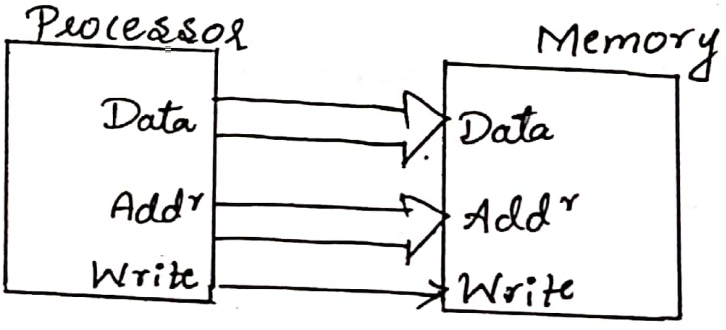
Load operation:

- In the load operation the contents of specified memory location are read by the processor.
- For load operation, processor sends the address of memory location whose contents are to be read & generates the appropriate read signal to indicate that it is a read operation.
- The memory identifies the addressed memory location & sends the contents of the memory location to the processor.



Store operation:

- In the store operation the data from processor is stored in the specified memory location.
- For store operation processor sends the address of the memory location where the data is to be written on the address lines, & generates the appropriate write signal to indicate the store operation & data to be written on the data lines.
- The memory identifies the addressed memory location & writes the data sent by the processor at that location destroying the former contents of that location.



Instructions and Instruction Sequencing

- Each instruction of the CPU contain specific information fields which are required to execute it.
- These information fields of instruction are called elements of instructions.

These are:

* Operation Code:

- The operation code field in the instruction specifies the operation to be performed

Example: In 8085, Opcode for Add B → 80H.

* Source/Destination Operand:

- The source/destination operand field directly specifies the source/destination operand for the instruction.

Example: In 8085, MOV A, B

B → Source
A → Destination operand

* Source Operand Address:

- It is known that operation specified by the instruction may require one or more operands.
- The source operand may be in the CPU register or in the memory.

- Many times the instruction specifies the address of the source operand so that operands can be accessed & operated by the CPU according to the instructions.

Example: In 8085, source operand address for instruction Add M is given by HL reg. pair.

* Destination operand Address:

- The operation executed by the CPU may produce the result.
- Generally the result is stored in one of the operand.
- Such operand is known as destination operand.
- The instruction which produces the result specifies the destination operand address.

Example: In 8085, the destination operand address for instruction INR M is given by HL reg pair.

* Next Instruction Address

- The next instruction address tells the CPU from where to fetch the next instruction after completion of execution of current instruction.

Example: In 8085, the instruction JMP 2000H specifies the next instruction address as 2000H

- It is seen that each instruction in a program specifies the operation to be performed & data to be processed.
- Hence, the instruction is divided into two parts: Opcode & operand.
- The operand is an another form for data.
- It may appear in different forms like, Addresses, Numbers, characters & logical data.
- The tasks carried out by a computer program consists of a sequence of small steps, such as adding two numbers, reading a character from the keyboard.
- A computer must have instructions capable of performing 4 types of operation.
  * Data transfer between memory & processor register.
  * Arithmetic & logic operations on data
  * Program Sequencing & control
  * I/O control.
- Before going to discuss these instructions first let us see some notations used to represent these instructions.

- It is

Register Transfer Notation

- It is seen that in a computer system data transfer takes place between processor register & memory & between processor registers & I/O system.
- These data transfers can be represented by standard notation given below.

* Processor registers are represented by notations $R_0, R_1, R_2 \ldots$ & so on.
* The addresses of the memory locations are represented by names such as LOC, PLACE, MEM.
* I/O registers are represented by names such as DATAIN, DATAOUT & so on.
* The contents of register or memory location are denoted by placing square brackets around the name of register or memory location.

Example: (i) $R_2 \leftarrow [LOC]$

- This expression states that the contents of memory location LOC are transferred to the register R2.

(ii) $R3 \leftarrow [R1] + [R2]$.

- This expression states that the contents of processor register R1 & R2 are added & result is stored into the processor register R3.

- The above notations are all commonly known as RTN.
- In these notations the data represented by RHS of the expression is transferred to the location specified by LHS of the expression, overwriting the old contents of that location.

Assembly language Notation

- Assembly language notations are another type of notations used to represent machine instructions & programs.

Example : (i) Move LOC, R1
(Move ← operation, LOC ← src, R1 ← dest)

- This expression states that the contents memory location LOC are transferred to processor register R1.
- Thus the contents of memory location LOC are unchanged but contents of register R1 are overwritten.

[The flow is always from left to right].

(ii) Add R1, R2, R3.

- This expression states that the contents of processor register R1 & R2 are added & result is stored in the register R3.

Basic Instruction types

- The processor instructions can be classified according to their operation & according to the number of address references required by them.
- In this case, let us consider basic instruction types according to number of address reference required by the instruction.
- Accordingly, there are four types
  - * Three Address machine.
  - * Two Address machine
  - * One Address machine
  - * Zero Address Instructions.

Three Address Instruction

- The three address Instruction can be represented Symbolically as

ADD A, B, C ; [A] + [B] ⟶ C.

where A, B, C are variables.

- In this instruction operands A & B are called Source operands & C is called destination Operand & ADD is the operation performed on the operands
- Thus the general instruction format for 3 address instruction is

Operation Source 1, Source 2, Destination

- Here the instruction will be having 3 operands.
- No GPR or Accumulator is required for 3-address instruction execution.
- Number of 3 address instruction = Number of arithmetic operations.
- Instruction length is very long & memory space required for program storage is more.
- The CPU is complicated & it is costlier.
- The instructions are

```
ADD  X, Y, Z  ; [X] + [Y] → Z
DIV  X, Y, Z  ; [X] / [Y] → Z
SUB  X, Y, Z  ; [X] - [Y] → Z
MPY  X, Y, Z  ; [X] * [Y] → Z
```

PROBLEM

1) Write the program to realise the following expression using 3-address instruction.

A + B * C + D

```
START : MPY  B, C, T  ; [B] * [C] → T1
        ADD  A, T1, T2 ; [A] + [T1] → T2
        ADD  T2, D, T3 ; [T2] + [D] → T3
        HLT
```

Two Address Instruction

- Here the instruction will be having 2 operands
- No GPR or Accumulator are needed for 2-address instruction execution.
- Number of two-address instructions = Number of arithmetic operations + one more instruction.
- This is true if original values of the variables are allowed to be destroyed.
- If the original values are to be preserved then number of instruction will be more.
- Instruction length is smaller compared to 3-address instruction & so memory space required for program storage is less.

- Two-address machine are costlier compared to one address machine & are used in main frames & minicomputers
- IBM-370 & PPP-11 are 2-address machine with very good speed of execution of instruction.

Example: MPY X, Y ; [X] * [Y] → [Y].

ADD X, Y ; [X] + [Y] → [Y].

MOV X, Y ; [X] ⟷ [Y] [X] → Y

Problem:

1) A * B + C * D

START: MPY A, B ; A * B → [B]

MPY C, D ; [C] * [D] → [D]

ADD A, D ; [A] + [D] → [D]

HLT [D ← A * B + C * D]

2) A = B * [C + D * E - F/G]

MPY D, E ; [D] * [E] → [E]

DIV F, G ; [F] / [G] → [G]

SUB E, G ; E - G → G

D * E - F/G → G

ADD C, G ; C + G → G

MPY B, G ; B * G → G

MOV R, G A

HLT.

One Address Instruction

- Here there will be only one operand in each instruction.

Opcode Source/destination.

- The accumulator is a must & destination is also the accumulator, But accumulator is not specified in the instruction.
- These instructions are much faster, because one operand will be in accumulator & only one operand has to be fetched from memory.
- Instruction length is less so the execution speed is more.
- It is cheaper compared to 3 or 2 address machine & is used in microcomputer.
- Here original values of the variables is preserved whether we want it or not.
- Some of the instructions used are

```
LOAD X  ; [Acc] ← [X]
STORE X ; X ← [ACC]
ADD  X  ; [ACC] ← [X] + [ACC]
MPY  X  ; [ACC] ← [ACC] * [X].
```

Problem

1) $A * B + C * D$

```
START : LOAD A ;   ACC ← [A]
        MPY  B ;  [ACC] ← [ACC] * [B]
        STORE X ;   X ← [ACC]
        LOAD  C ;   [ACC] ← [C]
        MPY   D ;   [ACC] ← [ACC] * [D]
        ADD   X ;   ACC ← A*B + C*D
        STORE X ;   X ← [ACC]
        HLT         [X = A*B + C*D]
```

2) $A = B * (C + D * E - F/G)$.

```
START : LOAD D ;  ACC ← [D]
        MPY  E ;  ACC ← ACC * E
        STORE X ;  X ← ACC  [X = D*E]
        LOAD  F ;  ACC ← F
        SUB   Y ;  ACC ← D * E - F/G
        ADD   C ;  ACC ← C + D * E - F/G
        MPY   B ;  B * [C + D*E - F/G]
        STORE A ;  A ← ACC.
        HLT
```

DIV G ; ACC ← ACC/G ←
STORE Y ; Y ← ACC
& LOAD X ; ACC ← X

Zero Address machine or Stack machine

- Here the arithmetic instruction contains no operands.
- Thus all the operations are performed in the Stack. Thus it is called as Stack machine also.
- A Stack is a LIFO (last in first out) data Structure.
- Stack Pointer is a register which points to the Top most filled location in the Stack.
- The location where Stack pointer addresses is called as Top level (TL) one below the top of the Stack is called as Second level. Each of these level can hold the data.
- When an arithmetic operation is performed on the contents of top of Stack register, result is Stored in the Second level of the register.
- The SP is incremented by 1 & it points to the SL, Thus SL becomes the top level now for new arithmetic operations.
- The one address instructions used are

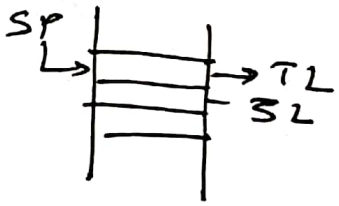
LOAD X ; is used to fetch the contents of [PUSH X] of memory location X & store it on top of Stack.

STORE X ; is used to pop the top of Stack into memory location X & SP

will be incremented by 1.

- Instruction length is smallest but the number of instructions in a program is more.
- Since the operand are in the stack register within the CPU, the speed of execution is very fast.
- But in reality stack is generally implemented in main memory so that we have large stack area. than the increased speed advantage is ~~lost~~ cost.
- Stack machine are not popular as compared to 2 or 1 address machine.

Example: Some of Instructions



LOAD X ; TL ← [X]
STORE Y ; Y ← [TL]
MPY ; [SL] ← [SL] * (TL]
ADD ; SL ← SL + TL
SUB ; SL ← SL − TL

Problem

1) Write a program in zero address machine to evaluate the above expression.

$$X = A*B + C*D.$$

⇒ START : LOAD A
LOAD B
MPY
LOAD C
LOAD D
MPY
ADD
STORE X
HLT.

Q4 A = B * (C + D * E - F/G)

LOAD C
LOAD D
LOAD E
MPY
LOAD F
LOAD G
DIV
SUB
ADD
LOAD B
MPY
STORE A
HLT

Addressing Modes

- Programs are usually written in high level language, which enables the programmers to use constants, local & global variables, pointers & arrays.
- When translating a high level language program into assembly language the compiler must be able to implement these constructs using the facilities provided in the instruction set of the computer in which the program will be run.
- The different ways in which the location of an operand is specified in an instruction are referred to as Addressing modes

Name	Assembler Syntax	Addressing function
1) Immediate	#Value	Operand = Value
2) Register	Ri	$E_A = R_i$
3) Absolute (Direct)	LOC	$E_A = LOC$
4) Indirect	(Ri) (LOC)	$E_A = [R_i]$ $E_A = [LOC]$
5) Inden	X(Ri)	$E_A = [R_i] + X$
6) Base with Inden	(Ri, Rj)	$E_A = [R_i] + [R_j]$

Name	Assembler Syntax	Addressing function
7) Base with Index offset	X(Ri, Rj)	EA = Ri + Rj + X
8) Relative	X(PC)	EA = [PC] + X
9) Auto increment	(Ri)+	EA = [Ri]; Increment Ri
10) Auto decrement	-(Ri)	Decrement Ri; EA = [Ri]

I Implementation of Variables & Constants

- The variables & constants are the most commonly used data types on every computer

Variable Representation

1. Register Mode

- The operand is the contents of processor register. The name (Address) of the register is specified in the instruction.

Example: MOVE R1, R2.

- This instruction copies the contents of register R1 to register R2.

2. Absolute Mode or Direct Mode

- The operand is in memory location, the address of this location is given explicitly

in the instruction

Example: MOVE LOC, R2

- This instruction copies the contents of LOC to register R2.

Constants Representation

3. Immediate Mode

- The operand is given explicitly in the instruction

Example: MOVE #200, R0

- This instruction places the value 200 in register R0.

Example: A = B+6.

A, B are declared as variables.

```
Move  B, R1
ADD   # 6, R1
Move  R1, A.
```

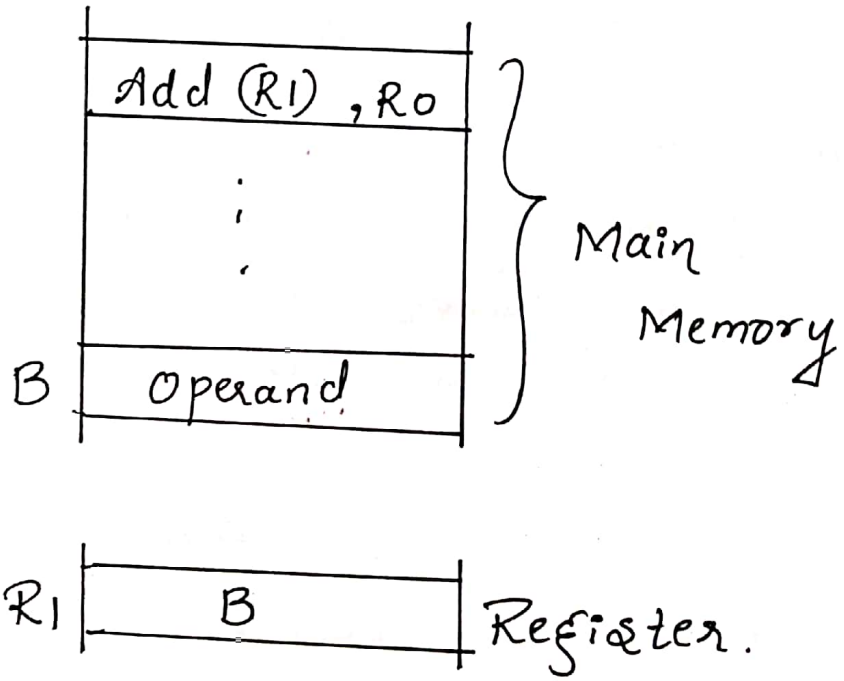
II Indirection & pointers

- Some processor instruction doesnot provide an operand or its address explicitly.
- Instead they provide the information from which the memory address of the operand can be determined. Such address is referred to as Effective Address of the operand.

4) Indirect Mode

- The effective address of the operand is the contents of a register or the main memory location whose address is given explicitly in the instruction.
- The Indirection is denoted by placing the name of register or the memory address given in the instruction in parenthesis.

a) Through a GPR

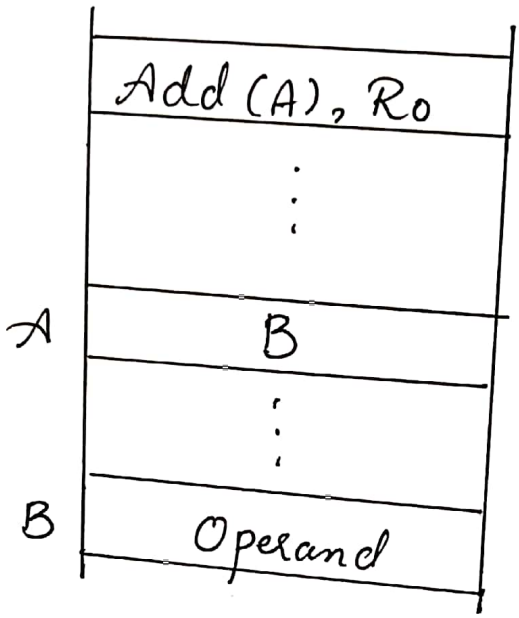


- To execute the Add instruction, the processor uses the value B, which is in register R1, as the effective address of the operand.
- It requests a read operation from the memory to get read the contents of location B.
- The value read is the desired operand which

the processor adds to the contents of register R0.

$$EA = [Ri].$$

b) Through a memory location



- In this case the processor first reads the contents of memory location A, then request second read operation using the value B as an address to obtain the operand

$$EA = [LOC].$$

- The register or memory location that contains the address of an operand is called a pointer.
- Indirect addressing through register is used extenssively.

III Indexing & Arrays

- The Indexing is a technique that allows programmer to point or refer the data (operand) stored in Sequential memory location one by one.

5). Index Mode

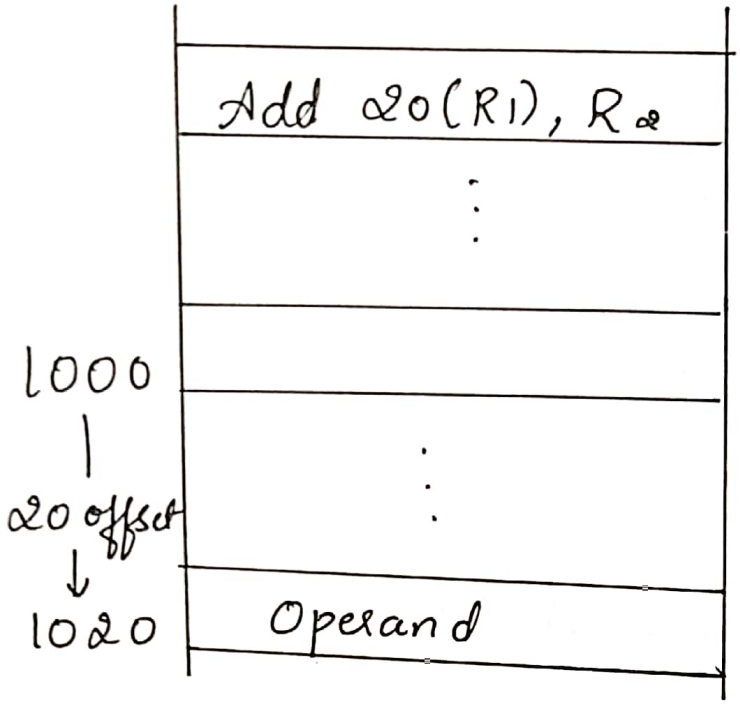
- The effective address of the operand is generated by adding a constant value to the contents of a register.
- The register used may be a special register provided for this purpose, or more commonly, it may be one of GPR in the processor.
- In either case it is referred to as an Index Register.
- The Index mode is indicated symbolically as

$$X(R_i).$$

where X denotes the constant value contained in the instruction & R_i is the name of the register involved.

- The effective address of the operand is given by $EA = X + [R_i]$
- The contents of index register are not changed in the process of generating the effective address.

i) The Index register R1 contains the address of memory location & the value X defines an offset (displacement) from this address to the location where the operand is stored.



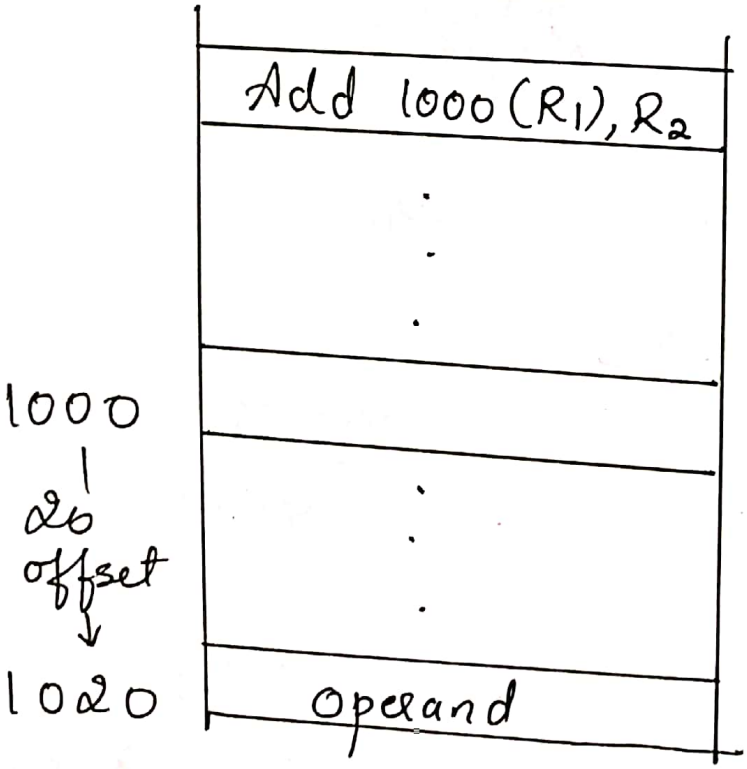
1000	R1

$EA = X + [R_1]$

$= 20 + 1000$

$= 1020.$

ii). Here the Constant X corresponds to a memory address, & contents of index register defines the offset to the operand.



20	R1

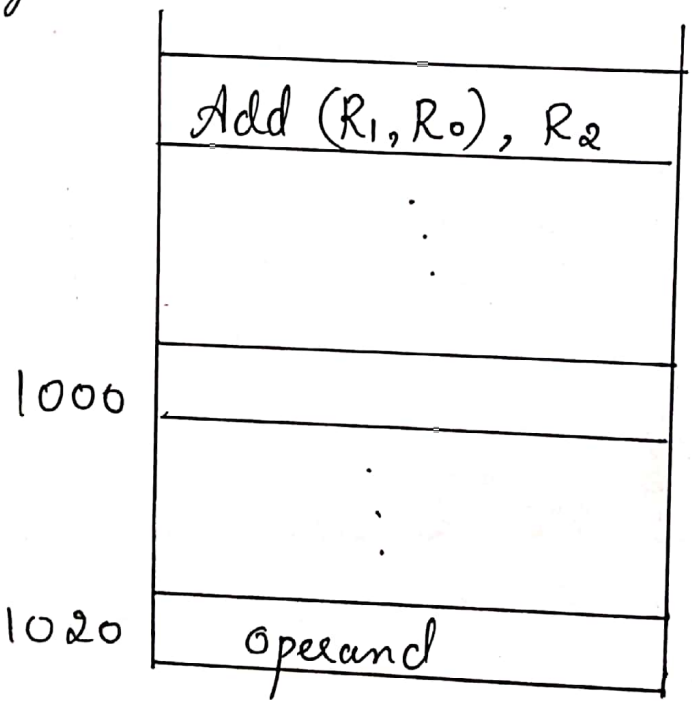
6) Base with Index

- A second register may be used to contain the offset X.
- In this case, Index mode can be written as

$$(R_i, R_j)$$ (R_i: Index reg, R_j: Base reg)

$$E_A = [R_i] + [R_j].$$

- The 2nd register is usually called the base register.



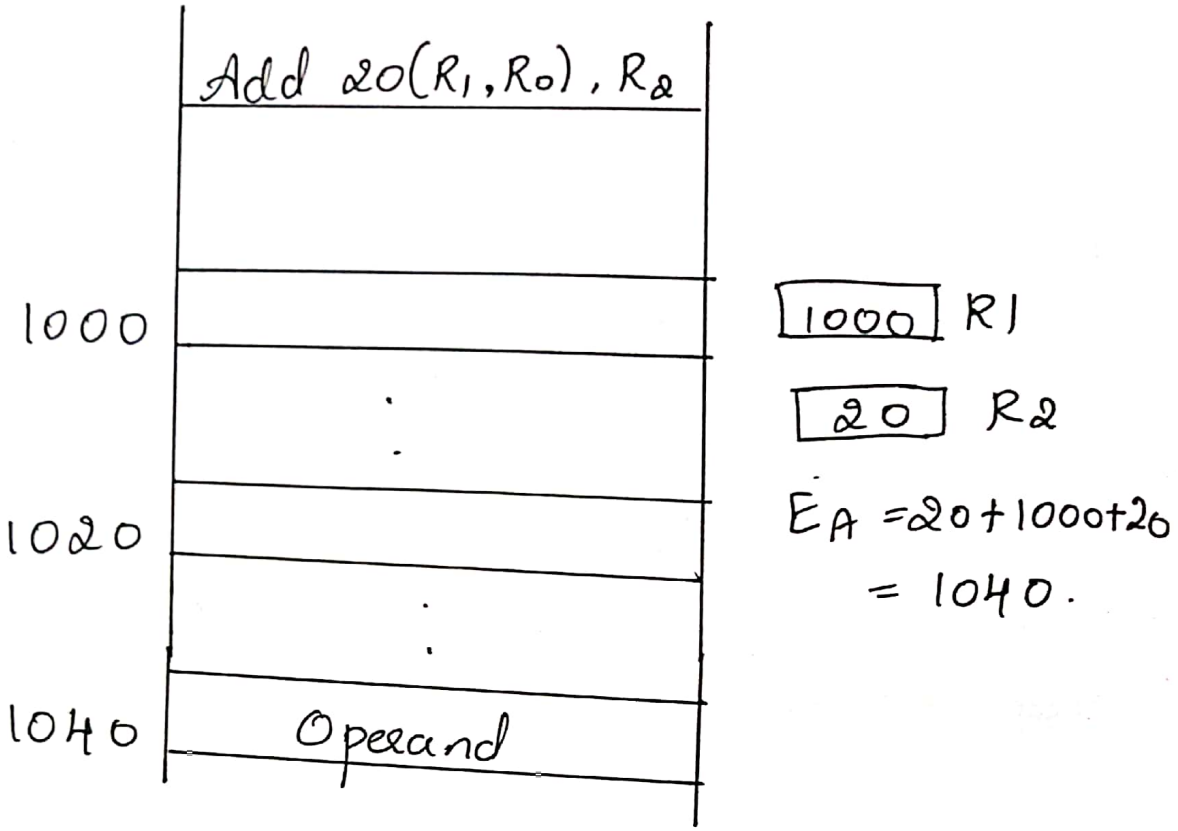
$E_A = [R_1] + [R_2]$
$= 1000 + 20$
$= 1020$

7) Base with Index & offset

- In this mode, there are two registers plus a constant which is denoted as

$$X(R_i, R_j)$$

$$E_A = X + R_i + R_j.$$



IV Relative Addressing Mode

- It is sum in the previous case Index mode using GPR.
- If in this case PC is used instead of GPR then it is called Relative addressing

$$EA = X + [PC]$$

- Relative mode can be used to access data operands.
- But is commonly used to specify the target address in branch instruction

Example: Branch >0 Loop.

Causes the program execution to go to the branch target location identified by the name

LOOP if the branch condition is not satisfied.
- Since the branch target may be either before or after the branch instruction, the offset is given as signed number.

V Additional Modes

i) Auto increment mode

- The effective address of the operand is the contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.

$(R_i)+$

$EA = [R_i]$; Increment R_i after accessing the operand.

Example: Add $(R_2)+, R_0$

- The above instruction adds the contents of memory location addressed by R2 with the contents of R0.
- After add operation, the contents of register R2 are automatically incremented by 1.

[8 bit - Incremented by 1
16 bit - " 2
32 bit - " 4]

ii) Auto decrement mode

- The contents of a register specified in the instruction are first automatically decremented & are then used as the effective address of the operand

Ex: $-(R_i)$

$EA = [R_i]$

Add $-(R_2)$, R_0

- The above instruction initially decrements the contents of register R2 & then the decremented contents of register R2 are used to address the memory location.
- Finally the contents from the addressed memory location are added to contents of register R_0.

Ex (1)

Add $(R_2)+$, R_0

Initially

R_2	1040

R_0	10

103C	
1040	10
1044	20
1048	

$(1040) = 10$

Add $(R_0) = 10$

$R_0 = 20H$

After execution

R_2	1044

R_0	20H

Ex c) Add $-(R_2), R_0$

Initially

R_2	1040

R_0	10

103C	08
1040	10
1044	:
1048	:

$(R_0) = 10$

$(103C) = 08$

Add

$R_0 = 18$

After execution

R_2	103C
:	:

R_0	18
:	:

PROBLEM

1) Registers R1 & R2 of a computer contain the den decimal values 1200 & 4600. What is the effective address of the memory operand in each of the following instructions.

a) Load 20(R1), R5

b) Move #3000, R5

c) Store R5, 30(R1, R2)

d) Add $-(R_2)$, R5

e) Subtract $(R_1)+$, R5.

⇒ a) Load 20 (R1), R5

EA = 20 + 1200

EA = 1220

b) Move #3000, R5

No memory operand

c) Store R5, 30 (R1, R2)

R1 = 1200

R2 = 4600

EA = 30 + 1200 + 4600

EA = 5830

d) Add −(R2), R5

[R2] = 4600

−[R2] = 4596

EA = 4596

e) Subtract (R1)+, R5

R1 = 1200

EA = 1200

Assembly Language

- A program which has a sequence of binary codes for the instructions is called machine level language program.
- However to write a program in machine language, programmer has to memorize the thousands of binary instruction codes for a processor.
- To make a programming easier usually programmers write program in assembly language.
- They then translate ALL to Machine so that it can be loaded into memory & executed when writing programs for a specific computer, those words are normally [Move, Add, Increments & Branch] replaced by acronyms called mnemonics such as Mov, ADD, INC, III^y we use R3 to refer Register 3 & LOC to refer to a mem location.
- A complete set of such symbolic names & rules for their use constitute programming language, generally referred to as assembly language
- The set of rules for using the mnemonics in the specification of complete instructions & programs is called the syntax of the language.
- Programs written in Assembly language can be automatically translated into a sequence of machine instructions by a program called Assembler.

- The assembler like any other program is stored as a sequence of machine instructions in the memory of a computer.
- The user program in its original alphanumeric text format is called a Source program, & the assembled machine language program is called an object program.

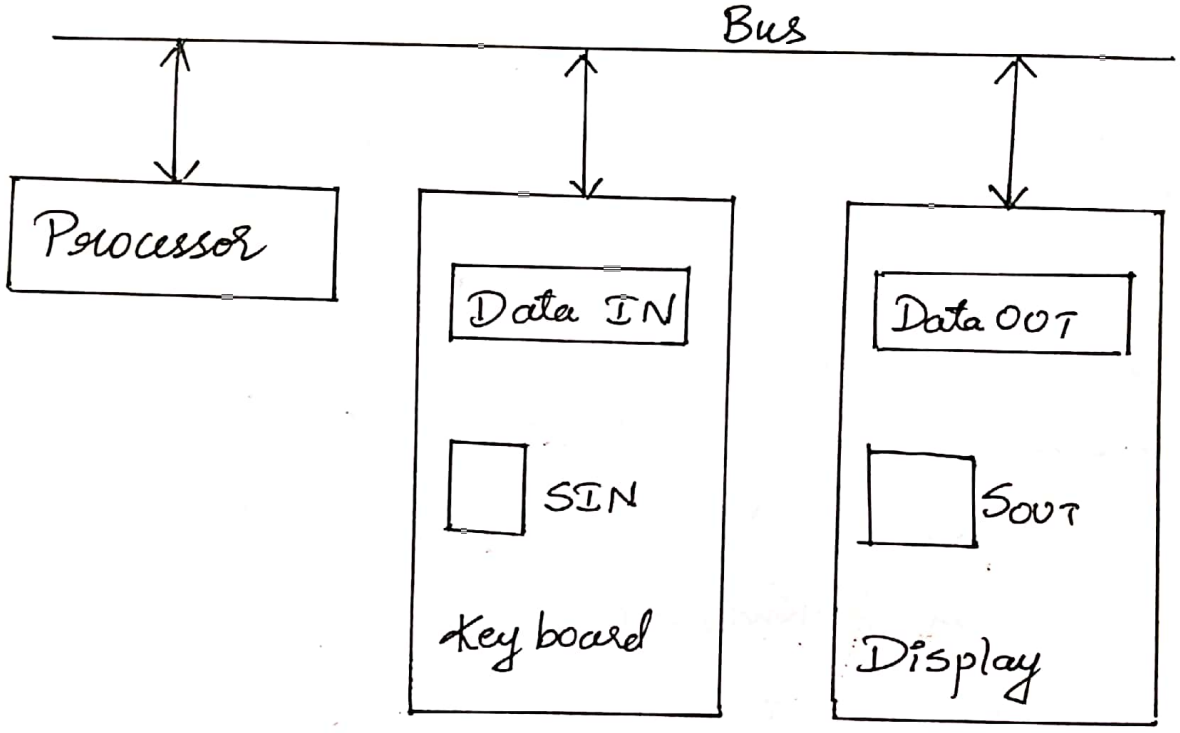
Assembler Directives

- In addition to pseu

Basic I/O Operation

- This topic deals with means by which data are transferred between memory of a computer & outside world.
- Consider a task that reads in character input from a keyboard & produces character output on a display screen.
- This is generally done by program-controlled I/O.
- It is known that rate of data transfer from the keyboard to a computer is limited by the typing speed of the user.
- The rate of output transfers from the computer to the display is much higher.

- The difference in speed between the processor & I/O devices creates the need for mechanisms to synchronize the transfer of data between them.
- The figure below shows the typical bus connection for processor, keyboard & display.



- The DATAIN & DATAOUT are the registers by which the processor reads the contents from keyboard & sends the data for display.
- SIN & SOUT are status control flags used to synchronize data transfer between keyboard & processor, & data transfer between display & processor.
- Whenever a key is pressed, the corresponding character code is stored in the DATAIN register & SIN states status bit is set to 1

to indicate that the valid character code is available in the DATAIN register.

- A program monitors SIN & when SIN is set to 1, the processor reads the contents of DATAIN register.
- After completion of read operation SIN is automatically cleared to 0.
- If another key is pressed, the corresponding character code is stored in DATAIN register, SIN is again set to 1 & process repeats.
- Whenever character is to be transferred from the processor to the display, buffer register DATAOUT & SOUT status control flag are used.
- Under program control, processor checks SOUT bit when SOUT equals 1, the display is ready to receive a character.
- ∴ Whenever the processor wants to transfer data to the display device & SOUT = 1 processor transfer data to be displayed to the DATAOUT register & Clears SOUT status bit to 0.
- The display device then reads the character from DATAOUT register.
- After the acceptance of the data by the display device, the SOUT bit is automatically set to 1 & display device is ready to accept next data types.

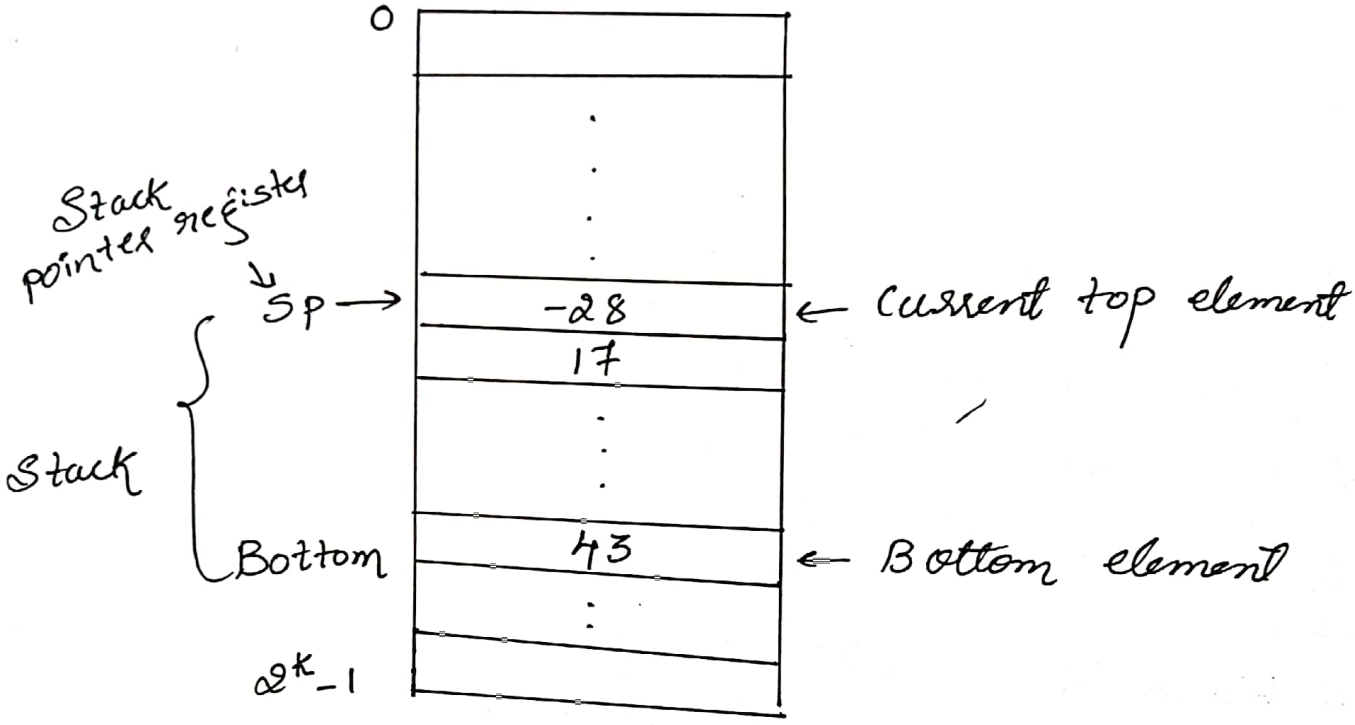
READWAIT Branch to READWAIT if SIN = 0
Input from DATAIN to R1

WRITEWAIT Branch to WriteWAIT if SOUT = 0
Output from R1 to DATAOUT.

Stacks & Queues

- In order to organize the control & information linkage between the main program & the subroutine, a data structure called a Stack is used.
- A Stack is a list of data elements (usually words or bytes) with the accessing restriction that elements can be added or removed at one end of the list only.
- This end is called top of the Stack & other end is called bottom of Stack.
- The last data item placed on the Stack is the first one removed when retrieval begins which is referred to as LIFO Stack.
- The PUSH & POP are terms used to describe placing a new item on the Stack & removing the top item from the Stack.
- Data stored in the memory of a computer can be organized as a Stack, with successive element occupying successive memory locations.

- Assume that the first element is placed in the location BOTTOM, & when new elements are pushed onto the stack, they are placed in successively lower address locations.



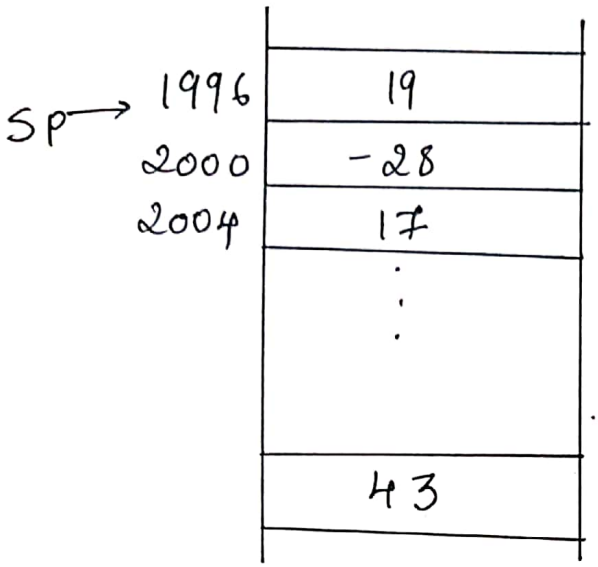
- This stack is implemented with the help of special memory register called Stack pointer.
- During PUSH & POP operation, stack pointer register gives the address of memory where the information is to be stored or to be read.
- PUSH operation can be performed by single instruction

MOVE NEW ITEM, -(SP) } Subtract #4, SP
Move NEW ITEM, (SP)

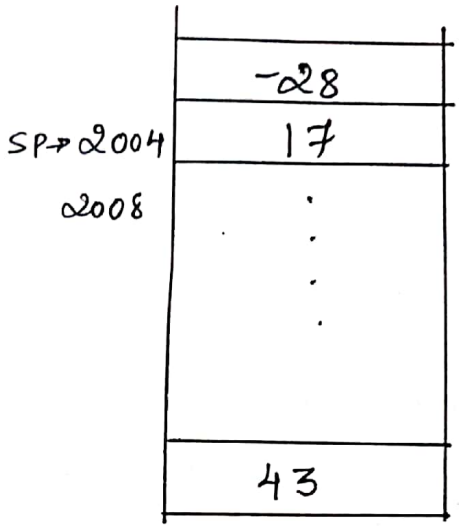
& POP operation can be performed by

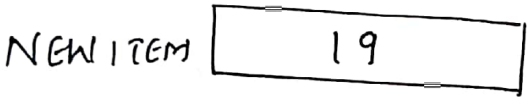
MOVE (SP)+, ITEM } Move (SP), ITEM
Add #4, SP

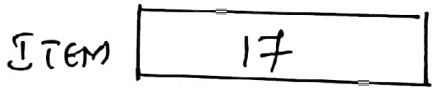
a) After PUSH from NEWITEM



b) After POP into ITEM







EA = 2004
after execution
SP → 2008

- Another useful data structure that is similar to the stack is called queue.
- Data are stored in & retrieved from a queue on a FIFO basis.
- New data are added at the back (high-address end) & retrieved from the front (low-address end) of the queue.

Differences between stack & Queue

- One end of the stack is fixed, while the other end rises & falls as data are pushed & popped. A single pointer is needed to point to the top of stack at any given time. On the other hand both ends of a queue move to higher

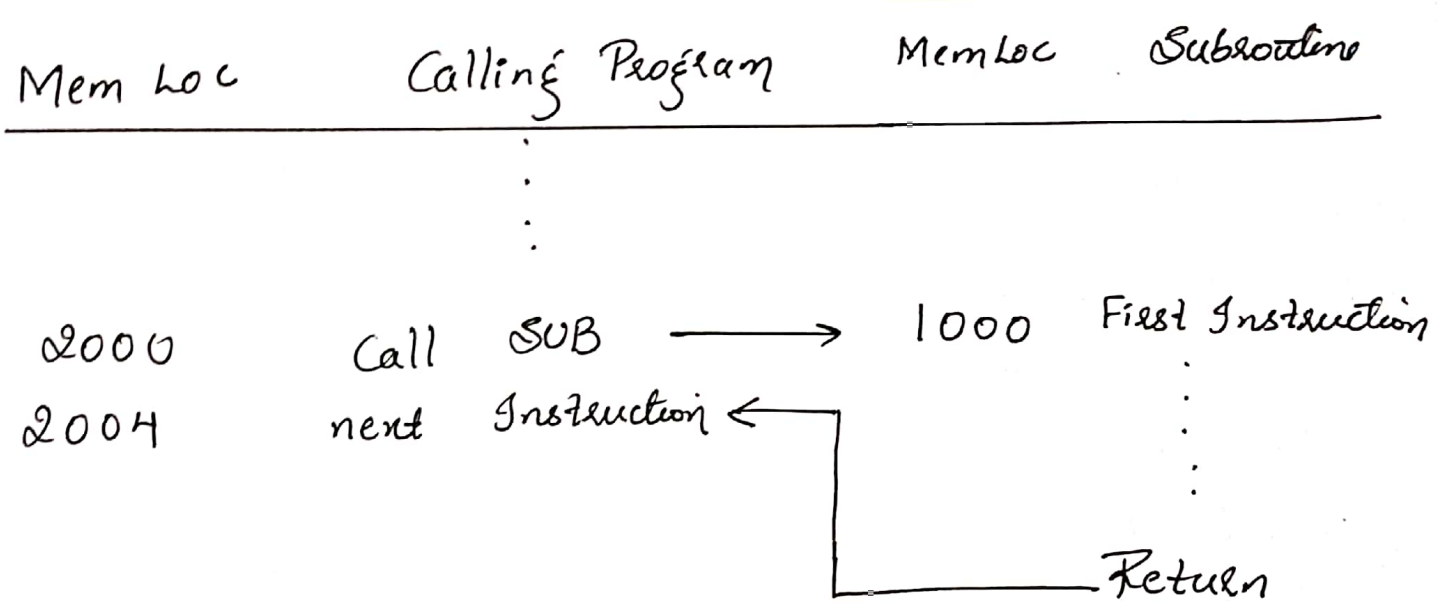
addresses as data are added at the back & removed from the front so two pointers are needed to keep track of the two end of queue.

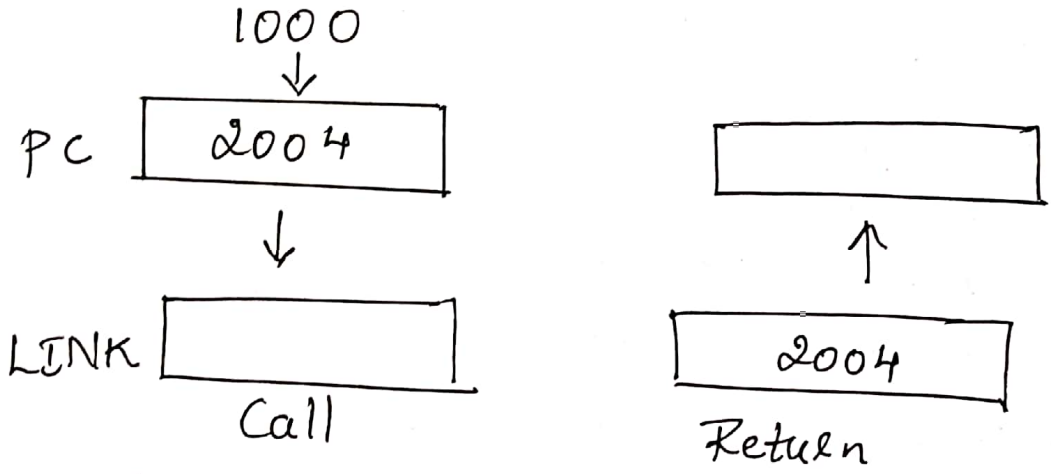
- Without further control, a queue would continuously move through the memory of a computer in the direction of higher addresses one way to limit the queue to a fixed region in memory is to use a circular buffer.

Subroutines

- In a given program, it is often necessary to perform a particular subtask many times on different data values.
- Such subtask is usually called subroutine when a program branches to a subroutine, then it is said that it is calling the subroutine the instruction that perform this branch operation is named as Call Instruction.
- After subroutine has been executed, the calling program must resume execution, continuing immediately after the instruction that called the subroutine.
- The subroutine is said to return to the program that called it by executing a Return Instruction.

- The Contents of the PC must be saved by the Call instruction to enable correct return to the Calling Program.
- The way in which a computer makes it possible to Call & return from Subroutine is referred to as its Subroutine linkage method.
- In the Subroutine linkage method Save the return address in a specific location, which may be a register dedicated to this function.
- Such a register is called link register when the Subroutine completes its task, the return instruction returns to the Calling program by branching indirectly through a link register.
- The Call instruction peform the following operation.
  * Store the Contents of PC in the link register.
  * Branch to the target address Specified by the instruction.
- The return instruction performs the following operation.
  * Branch to the address contained in the link register





Subrouline Nesting & the processor Stack

- When one Subroutine calls another Subroutine to complete a particular task, the operation is called Subroutine nesting.
- In this case, the return address of the second call is also stored in the link register, destroying its previous contents.
- Hence it is essential to save the contents of the link register in some other location before calling another Subroutine.

- Subroutine nesting can be carried out to any depth. That is return addresses are generated & used in LIFO order.
- ∴ The return addresses associated with Subroutine calls should be pushed onto Stack.
- A particular register designated as stack pointer, SP to be used for this operation.
- The SP points to a Stack called the processor Stack.
- The call instruction pushes the contents of the PC onto the processor stack & loads the Subroutine address into the PC.
- The return instruction POPS the return address from the processor stack into the PC.

Additional Instructions

- Few more important instructions that are found in most instruction sets.

Logic Instructions

- Logic operations such as AND, OR & NOT applied to individual bits, are the basic building blocks of digital circuits.
- It is also useful to be able to perform logic operations in Software, which is done using instructions that apply these operations to all bits of a word or byte independently & in

parallel.

Example : Not dst
Add #$FF000000, R0
Compare #$5A000000, R0

Shift & Rotate Instructions

Shift Instruction

- There are 2 types of Shifts
  1) Logical Shift
     a) Logical left Shift
     b) Logical right Shift
  2) Arithmetic Shift
     a) Arithmetic right Shift.

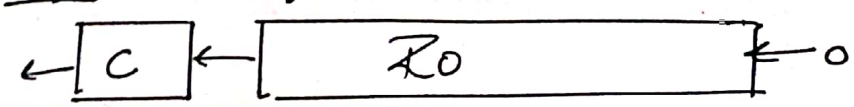
→ Instead of multiplication logical left Shift can be used because left Shift is always multiply by 2.

→ Instead of division logical right Shift can be used since right Shift is divide by 2.

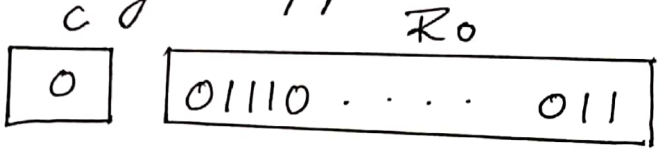
1) Logical Left Shift

- It is represented by the instruction
  LShiftL Count, destination
- LShiftL means logical Shift left.
- Count means the number of times left Shift has to be perfolmed.
- destination is the number to be Shifted

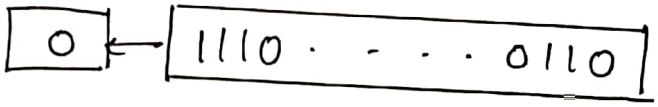
Example: LShiftL #2, R0



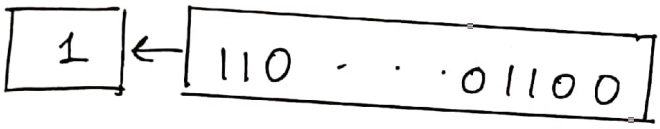
- When LShiftL is done zero is padded to LSB & the bit at MSB is stored in carry flag & eventually dropped.



LShiftL #2, R0



When shifted for 1st time



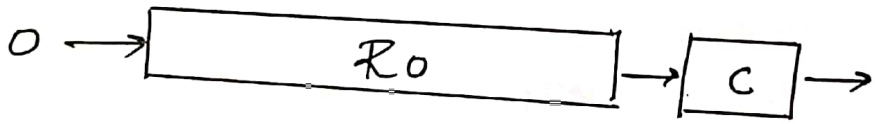
Shifted for 2nd time.

Logical shift right

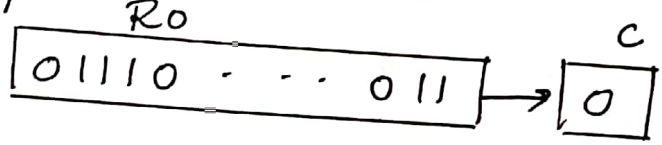
- It is represented by the instruction

LShiftR Count, destination

Example: LShiftR #2, R0

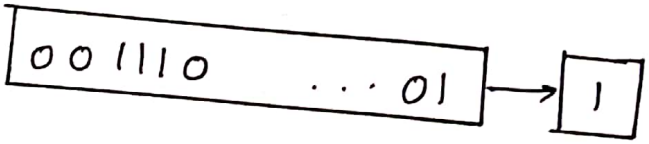


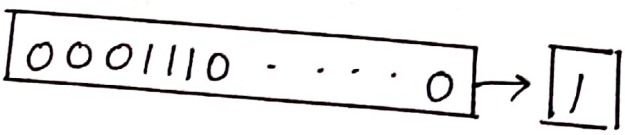
When logical right shift is done the bit at the LSB is stored in carry & eventually dropped & zero is padded to MSB



LShiftR #2, R0

Shifted for the first time





Digit Packing

- BCD stands for binary coded decimal
  There are two types of BCD
  → Unpacked BCD
  → Packed BCD.

Unpacked BCD

- In Unpacked BCD every decimal digit takes 8 bits

Example: 4 When stored in memory looks like

8bits	8bits	8bits	0000 0100

2 When stored in memory looks like

8bits	8bits	8bits	0000 0010

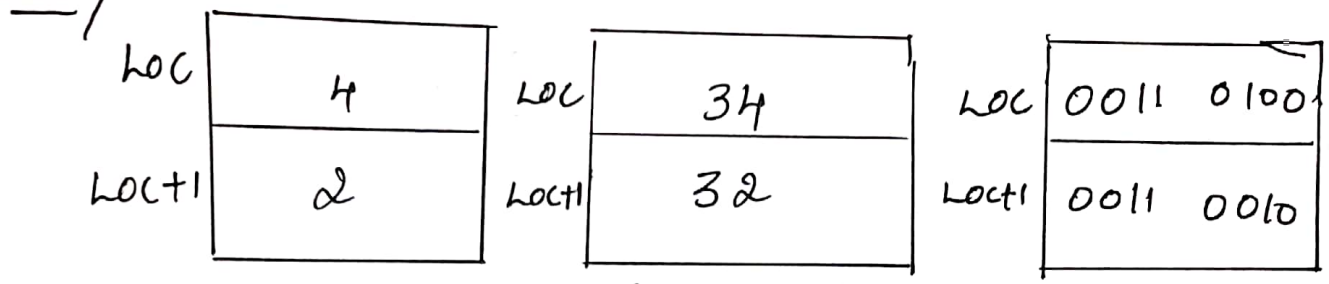
Packed BCD

- In Packed BCD every decimal digit takes 4bits.

Note: Since 8 bits are required to represent one decimal digit in unpacked bcd & packed bcd requires only four bits to represent one decimal digit, Using packed BCD we can represent two digits using 8 bits.

- Consider an example where two numbers are represented in unpacked bcd stored at the memory LOC & LOC+1 we want to pack them into a single memory location.

⇒



Entered Characters | ASCII value in hex | BCD format in memory.

- Two numbers 4 & 2 are entered by the user. The ASCII of these numbers in hex is 34 & 32 which is stored in the memory as BCD format

- In Order to pack them into single memory location follow the steps.

1) Take the first number & left shift it four times.

i.e, when we left shift 0011 0100 four times we get

0100 0000

2) Take the second number & perform AND operation to clear higher order bits

i.e, 0011 0010

AND 0000 1111

0000 0010

3) Now perform OR operation with the result of Step 1 & Step 2

0100 0000

OR 0000 0010

0100 0010

4 2

Assembly language code for digit packing

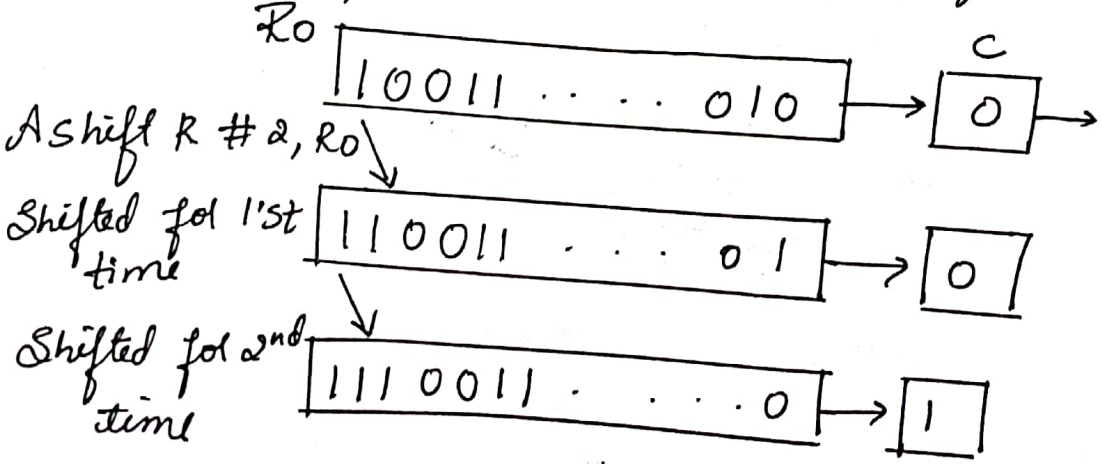
```
Move      #LOC, R0
Movebyte  (R0)+, R1
LShiftL   #4, R1
Movebyte  (R0), R2
AND       #0F, R2
OR        R1, R2
Movebyte  R2, PACKED
```

Arithmetic Right Shift

- It ensures that sign of the number is maintained.
- It is same as logical right shift in case of a +ve number.
- As the bits are shifted zero is padded to MSB.

For a Negative Number

- As we right shift the number n times the vacant places at MSB is filled with 1.



Rotate Instruction

- There are two types of Rotate
  1) Rotate without carry
     a) Rotate Left
     b) Rotate right
  2) Rotate with carry
     a) Rotate left
     b) Rotate right.

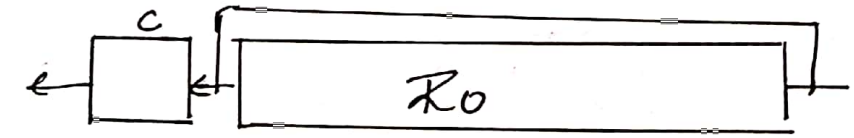
- In rotate without carry, carry bit will not participate.
- The bit that will rotated will be stored on carry flag & eventually dropped.

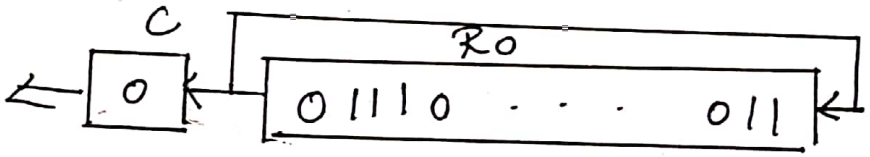
1) Rotate Left without carry.

Instruction is as follows

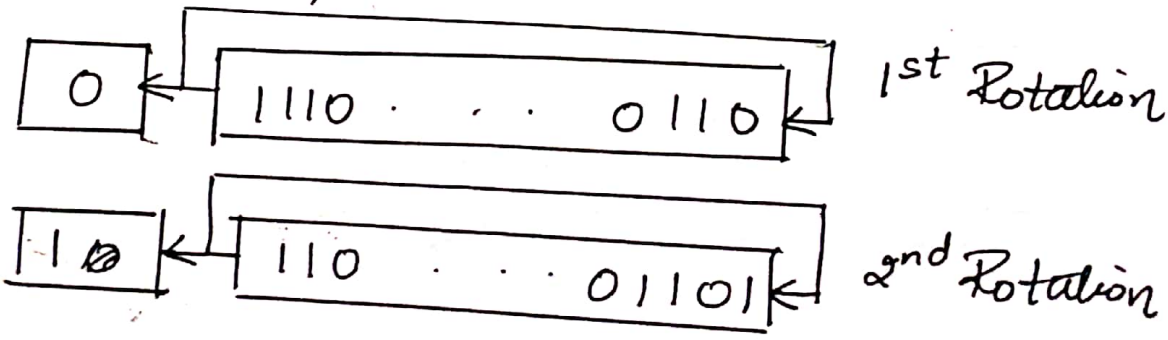
Rotate L Count, destination

Rotate L, #2, R0





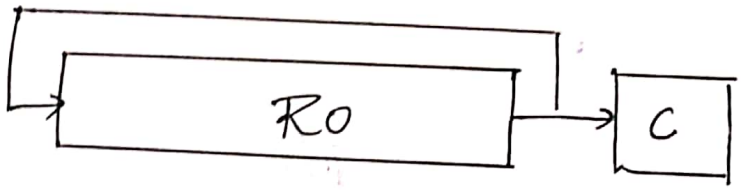
Rotate L #2, R0

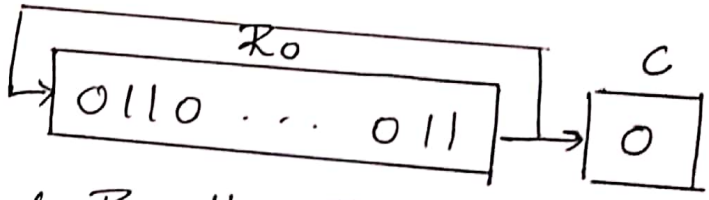


Rotate right without carry

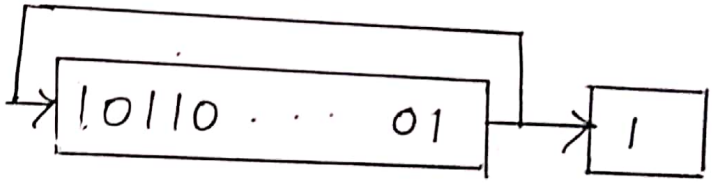
Instruction : Rotate R Count, destination

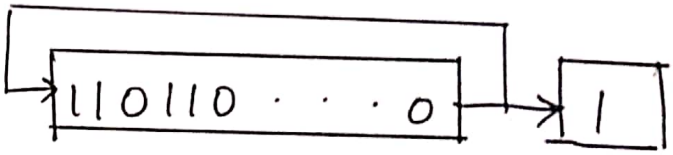
Example : Rotate R #2, R0





Rotate R #2, R0



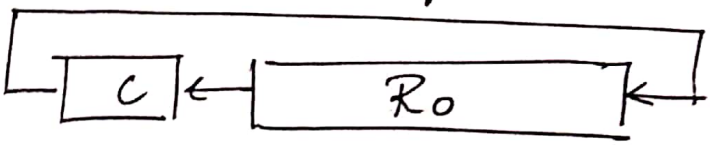


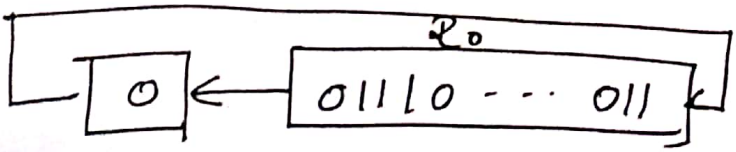
Right Rotate with Carry

- Even carry bit will participate in rotation

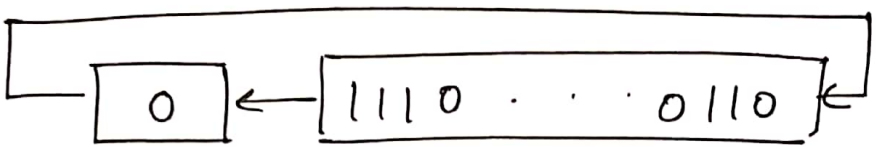
Instruction : RotateLC Count, destination

RotateLC → RotateLeft with Carry.

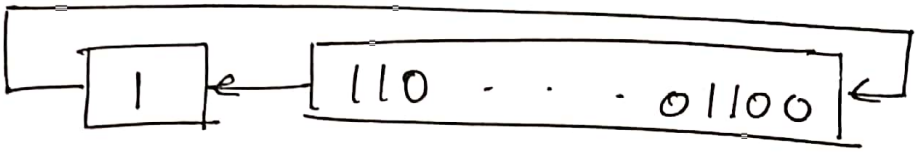




Rotate LC #2, R0



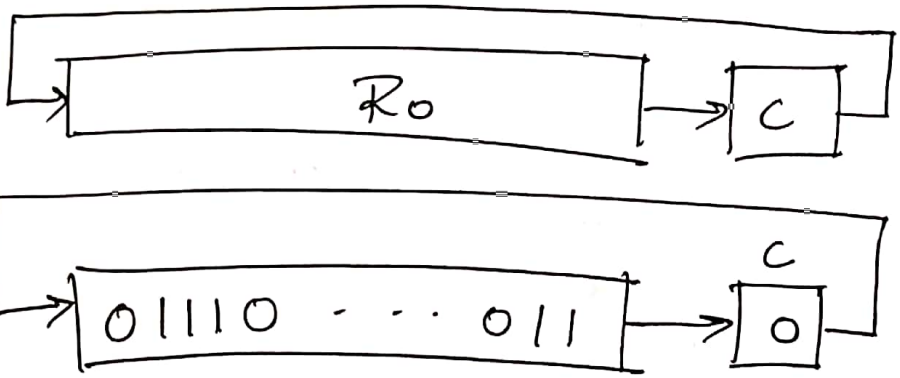
1st Rotation



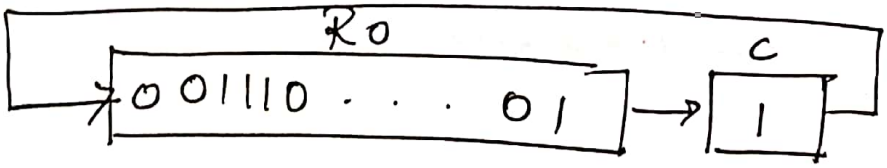
2nd Rotation.

2) Rotate right with carry

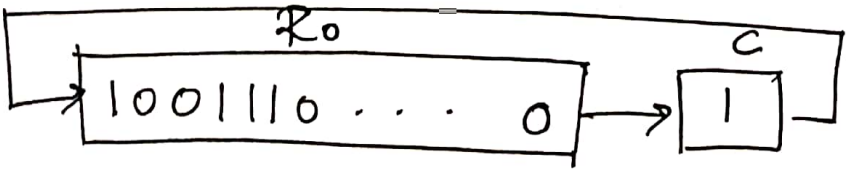
Instruction: RotateRC Count, destination



Rotate RC #2, R0



1st Rotation



2nd Rotation.

Encoding of Machine Instruction

- Binary pattern of Assembly language instruction which will be stored in memory & gets executed is known as machine instruction.
- An instruction is made up of opcode, source, destination & addressing modes of source & destination
- Unless these opcodes, source, destination & addressing modes are converted to binary computer cannot understand the instruction.
- Specific identification will be given to all of these or sr
- Suppose one bit is used then we can represent two opcodes

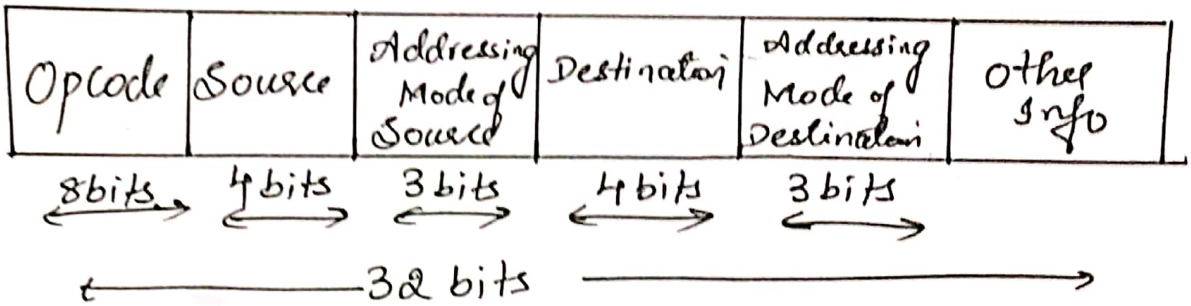
  i.e, 0 → Move
  1 → Add } example

- If two bits are used then four opcodes can be represented

  00 → move
  01 → add
  10 → Branch
  11 → Call

- To represent an opcode 8 bits are used

  ie $2^8 = 256$ different opcodes can be represented using 8 bits



Example

Add R1, R2

- Opcode Add takes 8 bits
- To Specify register R2 (source) needs 4 bits
- Addressing mode of R2 needs 3 bits
- To Specify register R2 (destination) needs 4 bits
- Addressing mode of R2 needs 3 bits
- This complete instruction can fit in the memory