COMPUTER SCIENCE & Engineering/IT
Quick Revision CAPSULE

Youth Competition Times

UGC-NET/ JRF/ GATE/ IES/ PSU/ UPPSC AE/ Polytechnic Lecturer/LT Grade/ NIELIT/ ISRO

• UPPCL AE • UPRVUNAL AE • UJVNL AE • KVS NVS PGT • DSSSB • PTCUL
• ASSAM PSC • APPSC • BPSC• CGPSC • GUJRAT SC • HPPSC • UPSC
• MPPSC • Rajasthan Basic Computer Teacher • J&KPSC • KERALA PSC
• KARNATAKA PSC • KPCLAE • ISRO Scientist/Engineer • OPSC • RPSC
• TNPSC • PUNJAB PSC • MAHARASHTRA PSC • BHEL ET • TANGEDCO AE
• TELANGANA PSC • JUVNL AEE • CIL MT • RRB SSE • SJVNL ET
• TSGENCO AE • VIZAG STEEL MT • TAMILNADU TRB Polytechnic Lecturer

COMPUTER SCIENCE & ENGINEERING/ INFORMATION TECHNOLOGY

KEY NOTES, TERMS, DEFINITIONS & FORMULAE

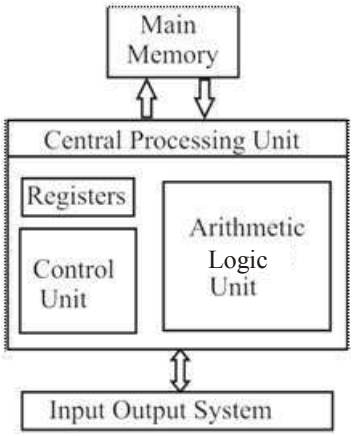
QUICK REVISION CAPSULE

Useful For All Computer Competitive Exams...

INDEX

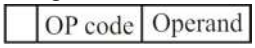
01. Computer Organization and Architecture

➲ **Von Neumann Architecture (Stored Memory Program)**



➲ **Various General Purpose Registers–**

- Address Register
- Data Register
- Accumulator
- Program Counter

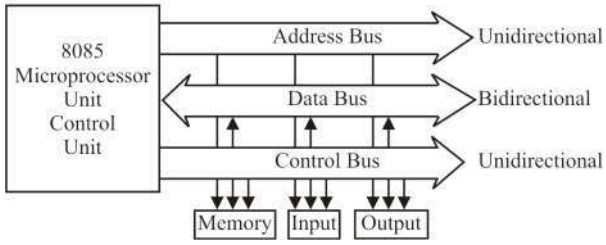


- Instruction Register
- Temporary Register
- Input Register
- Output Register

Type of Bus– There are three type of Bus.

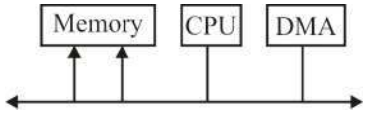
- Address Bus
- Data Bus
- Control Bus

Bus Architecture



- The address bus, which is a unidirectional path way that allows information to travel in only one direction, carries information about where data will be stored in memory.
- The data bus is a bidirectional path way that carries the actual data (information) to and from the main memory.
- The control bus carries the control and timing signals needed to coordinate the activities of the entire computer. Think of this as a traffic cop.

Bus Arbitration–The overall control which decides who will control the master bus line.



There are two approaches to bus arbitration.

- Centralization bus arbitration
- Distributed bus arbitration

Register–

1. Memory Address Register (MAR) hold on address of the memory unit.
2. PC–Program Counter
3. IR–Instruction Register
4. R–Processor Register

Computer register are designated by capital letters.

Register Transfer– $R_2 \leftarrow R_1$

5. DR – Data Register

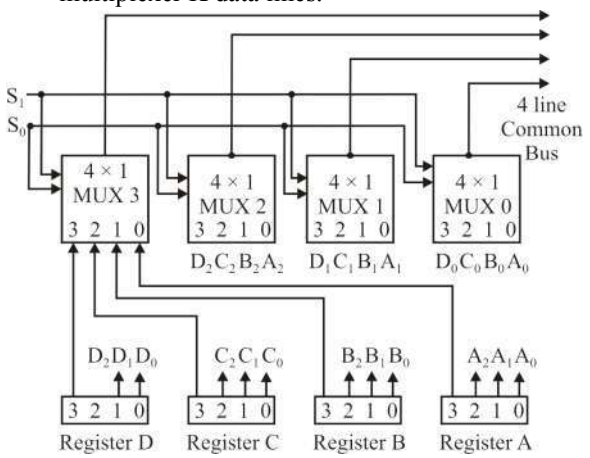
Bus and Memory Transfers–

Common Bus- It consists of a set of common lines one for each bit of a register through which binary information is transfered one at a time.

Control Signal–Determine which register is selected by the bus during each particular register transfer.

Bus System for four Register–

- In general bus system will multiplex **K register of n bits** each to produce an n-line common bus.
- The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register.
- The size of each multiplexer must be K×1 since it multiplexer K data lines.



Bus Transfer–

Bus $\leftarrow R_2$, $R_1 \leftarrow$ BUS

$\Rightarrow R1 \leftarrow R2$

Memory Transfer– Read : DR ← M[AR]

Write : M[AR] ← R_1

Where, DR → Data Register

AR → Address Register

Micro-operations– Micro-operations classified into four categories Address Register ← Program counter

1. Register Transfer Micro-operations
2. Arithmetic Micro-operation
3. Logic Micro-operation
4. Shift Micro-operation

Register Transfer Micro-operations- These type of micro operations are used to transfer from one register to another binary information.

Arithmetic Micro-operation

1. Addition – $R_3 \leftarrow R_1 + R_2$
2. Subtraction – $R_3 \leftarrow R_1 - R_2$

$$R_3 \leftarrow R_1 + \overline{R}_2 + 1$$

3. 1's complement – $R_2 \leftarrow \overline{R}_2$
4. 2's complement – $R_2 \leftarrow \overline{R}_2 + 1$
5. Increment – $R_1 \leftarrow R_1 + 1$
6. Decrement – $R_1 \leftarrow R_1 - 1$

Arithmetic shift – The increment and decrement micro operations are implemented with a binary up-down counter.

Logic micro operation–

1. OR – $R_1 \leftarrow R_1 \vee R_3$
2. AND – $R_1 \leftarrow R_2 \wedge R_3$
3. XOR – $R_1 \leftarrow R_2 \oplus R_3$
4. Compliment – $R_1 \leftarrow \overline{R}$
5. X-NOR – $R_1 \leftarrow R_2 \odot R_3$

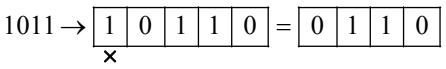
Shift Micro operation–

- Logical shift
- Circular shift (Rotation)
- Arithmetic shift

Logical Shift– There are two types

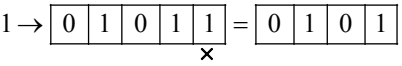
- Left shift

Example–



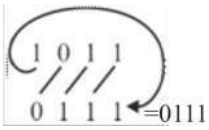
- Right shift.

Example–

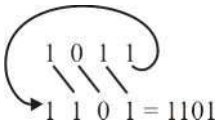


Circular shift– There are two types

- Left shift



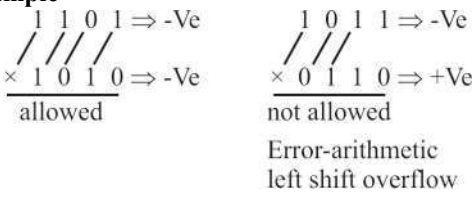
- Right shift



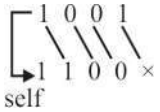
Arithmetic shift– Applied on signed number after the shift sign of the number should remain same.

Left shift– It is same as logical left shift but it is allowed only when sign is not going to change.

Example–



Right shift–



➲ **Instruction format–**

- A group of bits which instructs computer to perform some operation.

ISA (Instruction Set Architecture) Collection of all instruction CPU supports.

Type of Instruction (Based on Operation)

- Data Transfer – MOV, LDI, LDA
- Arithmetic & Logic – ADD, SUB, AND, OR
- Machine Control – EI, DI, PUSH, POP
- Iterative – LOOP, LOOPE, LOOPZ
- Branch – JMP, CALL, RE T, JZ, JNZ

Type of Instruction (Based on Operand Information)

- 4-Address Instruction
- 3-Address Instruction
- 2-Address Instruction
- 1-Address Instruction
- 0-Address Instruction

Types of CPU Organization– The number of address fields in the instruction format of a computer depends on the internal organization of the registers.

There are three types of organization.

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

1. Single Accumulator Organization– The instruction format in this type of computer user one address field.

Example–

ADD × AC ← AC + M[X];

AC → Accumulator Register

2. General Register Organization uses three and two address instructions.

Example–

ADD $R_1, R_2, R_3 = R_1 \leftarrow R_2 + R_3$

ADD $R_1, R_2 = R_1 \leftarrow R_1 + R_2$

3. Stack Organization uses zero and one address instruction.

Example–(i) PUSH (ii) ADD

In a general register CPU organization there are eight general purpose register and ALU can perform 32-different operation.

The number of selection line of each multipliers for selecting the operand = 3 (2^3 =8)

The number of bits in operation code= 5(2^5 = 32)

The length of the control word = 5 + 3 + 3 + 3 = 14 (for 3-address instruction)

Addressing Mode

The way of operand are choosen during program execution is dependent on the addressing mode of instruction.

The addressing mode may reduce the number of bits in the addressing field of the instruction.

Instruction Cycle– CPU performs 6 phases to execute an instruction.

Fetch cycle – Instruction fetch

Execution cycle – from decode ..... to write back

Type of addressing mode–

- **Implied Mode–** Operand is specified Implied in the definition of instruction.
  Used for zero address and one address instruction.
- **Immediate Mode–** Operand is directly provided as constant.
  No computation required to calculate effective address.
- **Register Mode–** Operand is present in the register.
  Example– $LDR_1 \Rightarrow AC \leftarrow R_1$
  Register number is written in instruction.
- **Register Indirect Mode–** Register contains address of operand rather than operand itself.
  Example– LD $(R_1) \Rightarrow AC \leftarrow M[R_1]$
- **Auto Increment or Auto Decrement Addressing Mode–** Special case of register indirect addressing mode.
  Example–
  $LD(R_1) + \Rightarrow AC \leftarrow M[R_1], R_1 \leftarrow R_1 + 1$
- **Direct Addressing Mode (Absolute addressing mode)–** Actual address is given in instruction.
  Use to access variables
  Example– LD ADR $\Rightarrow$ AC $\leftarrow$ M[ADR]
- **Indirect Addressing Mode–**
  Using to implement pointers and passing parameters.
  2 memory access required. Where the effective address is stored in memory.
  Example– LD@ADR $\Rightarrow$ AC $\leftarrow$ M[M[ADR]]
- **Relative Address Mode–** In this mode the content of the program counter (PC) is added to the address part of the instruction in order to obtain the effective address.
  Example– LD$ADR $\Rightarrow$ AC $\leftarrow$M[PC + ADR]
  # Relative addressing is often used with branch-type instruction.
- **Base Register Mode–** Used in program relocation of programs in memory.

➲ **Indexed Addressing Mode–**

$\rightarrow$ Use to access or implement array efficiently.

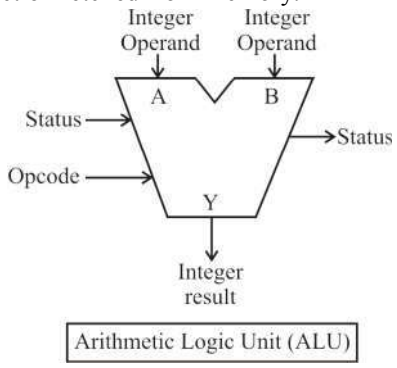
$\rightarrow$ Multiple Registers required to implement.

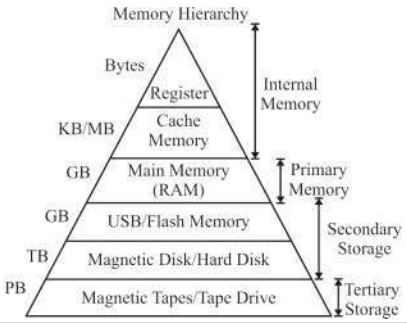
$\rightarrow$ Any element can be accessed without changing instruction.

Example– LD AD R(x) $\Rightarrow$ AC $\leftarrow$ M [ADR + X]

Arithmetic Logic Unit (ALU)

➲ Inside a computer, there is an (ALU) which is capable of performing logical operations (e.g AND, OR, Ex-OR, Invert etc.) in addition to arithmetic operation (Addition, Subtraction etc.). The control unit supplies the data required by the ALU from memory or from input devices and directs the ALU to perform a specific operation based on the instruction fetched from memory.





Cache Memory

Cache memory faster than main memory.

Cache hit– It required element present in cache that called cache hit.

Hit latency– Time taken to find out whether element present on the cache or not that is called hit latency.

Cache miss– It required element not present in cache, that is called 'Cache miss'.

Miss latency– Time taken to get something from main memory and then place it into the cache and then read that's called miss latency.

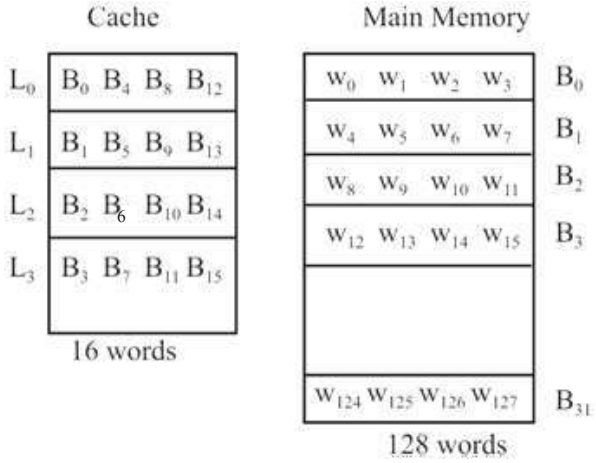
Page hit– It required element present in main memory.

Page fault– It required element not present in main memory.

Tag directory– Tag directory say that required element present in tag on not.

➲ **Introduction to Direct mapping–**

- Taking about disk and main memory we talking about paging.
- Talking about Cache and main memory we talking about blocks.
- Block size = lines size.



- Smallest addressable unit in the memory called word.

Let's

1w = 1B (means system is byte addressable)

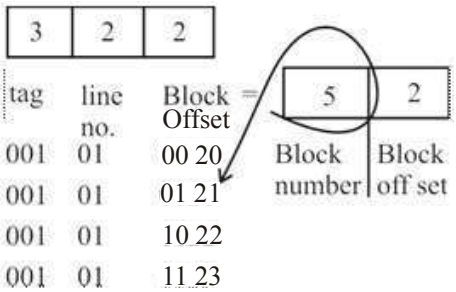
Block size = 4 word

No. of black in main memory = 128/4 = 32 block

No. of Lines in Cache = 16/4 = 4 lines

Physical address contain = 128 = 2^7 = 7 bits

Processure generate address =



Example– Main memory size = 128 KB

Cache size = 16 KB

Block size = 256 B

Tag bits = 3 bit

Tag directory size = ?

Solution – Main memory size 128 KB

$= 2^{10} * 128$ B

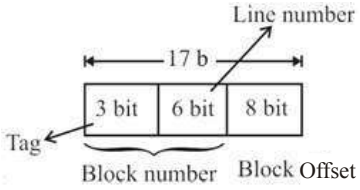
$= 2^{10} * 2^7$ B

$= 2^{17}$ B

Block size = 256 B = 2^8 B

No. of Block $= \frac{2^{17}}{2^8} = 2^9$ bit

Cache size = 16 KB = 2^{14} b

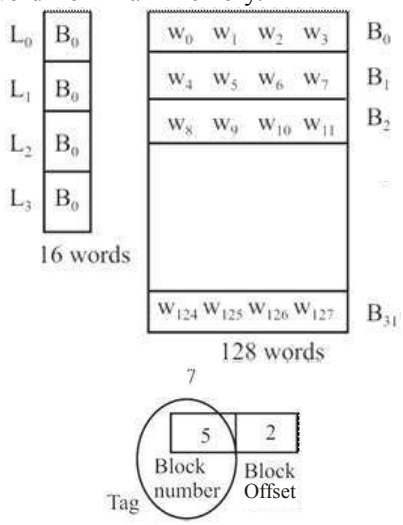


Number of line $= \frac{\text{Cache size}}{\text{Block size}} = \frac{2^{14}}{2^8} = 2^6$ bit

Tag directory size = (Tag size * No. of lines)

$= (3*2^6)$ bit

➲ **Fully Associative–** The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory.



Block no. = 32 = 2^5 = 5 bit

Block offset = 4 = 2^2 = 2 bit

➲ **Set-Associative Mapping–** The set associative mapping is an improvement over the direct mapping in that each word of cache can store two or more word of memory under the same index address.

Total number of set $= \frac{\text{No. of lines}}{K}$

K is number of way

Example–

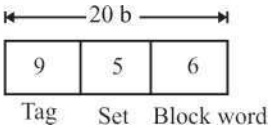
4 way set associative cache lines = 128

Lines size = 64 word

Physical address = 20 bits

Tag, set and word field are?

Solution–



Number of lines $= \frac{\text{Cache size}}{\text{Lines size}}$

Cache size = $2^6 * 2^7 = 2^{13}$ word

$$\text{Sets} = \frac{\text{Lines}}{\text{Set size}} = \frac{2^7}{2^2} = 2^5$$

Tag = 9, Set = 5, Word = 6

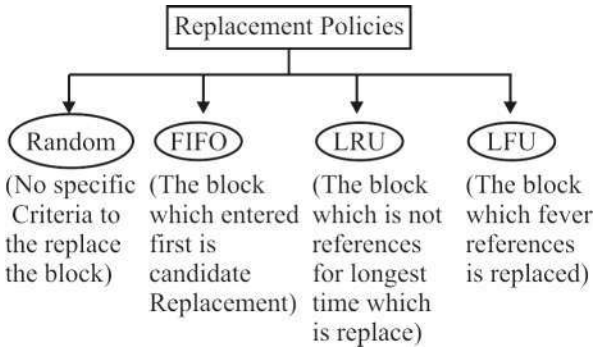
➲ **Locality of Reference–**

1. Spatial locality
2. Temporal locality

- If a word is accessed now then the word adjacent to it (close proscimity) will be accessed next.
- Keeping more words in a block affects spatial locality (block size).
- If a word is referenced now then the same word will be referenced again in future.
- LRU is used in temporal locality.

➲ **Cache Replacement Algorithms–**

- Replacement policy is required for associative mapping and set associative mapping but not for direct mapping.
- Replacement policies are aimed to minimize miss Penalty for future references.



Example– Consider the cache has 8 blocks for the memory reference (5, 12, 13, 17, 4, 12, 13, 17, 2, 13, 19, 13, 43, 61, 19). What is the hit ratio for the following cache replacement algorithms.

(i) FIFO
(ii) LRU
(iii) Direct mapping
(iv) 2-way set associate

(i) FIFO (M M M M M H H H M H / 5, 12, 13, 17, 4, 12, 13, 17, 2, 13, / M M M M H / 19, 13, 43, 61, 19)

S_0	5 ~~4~~	43
S_1	12 ~~2~~	61
S_2	13	19
S_3	17	13

Hit ratio $= \left(\frac{5}{15} \times 100\right) = \frac{100}{3} = 33\frac{1}{3}$

Miss $= \left(\frac{10}{15} \times 100\right) = \frac{2}{3} \times 100 = 66\frac{2}{3}$

(ii) LRU (M M M M M H H H M H / 5, 12, 13, 17, 4, 12, 13, 17, 2, 13, / M H M M H / 19, 13, 43, 61, 19)

(~~5~~, ~~12~~, ~~13~~, ~~17~~, ~~4~~, ~~12~~, 13, ~~17~~, ~~2~~, ~~13~~, ~~19~~, 13, 43, 61, 19)

Hit ratio $= \left(\frac{6}{15} \times 100\right) = 40$

Miss $= \left(\frac{9}{15} \times 100\right) = 60$

(iii) Direct mapping [M / 5,] (M M M M M M M M M M / 5, 12, 13, 17, 4, 12, 13, 17, 2, 13, / M H M M M / 19, 13, 43, 61, 19)

S_0	12	1 ~~12~~ 2
S_1	~~5~~	~~13~~ ~~17~~ ~~13~~ ~~17~~ ~~13~~ 61
S_2		
S_3	~~19~~	~~43~~ 19

Line number = B.No mode 4

Cache hit ratio $= \left(\frac{1}{15} \times 100\right) = 6.66$

Miss ratio $= \left(\frac{14}{15} \times 100\right) = 93.333$

(iv) 2-way set associate

(5 12 13 17 4 12 13 17 2 13 / M, M, M, M, M, H, H, H, M, H, / 19 13 43 61 19 / M, H, M, M, M)

S_0	12 ~~4~~	12 12
S_1	~~5~~ 13	~~17~~ ~~13~~ ~~17~~ ~~13~~ ~~19~~ ~~13~~ ~~13~~ 61 19

Line no. = (i mod 2) i → Block number

Hit ratio $= \left(\frac{5}{15} \times 100\right) = 33.33$

Miss ratio $= \left(\frac{10}{15} \times 100\right) = 66.66$

Control Processing Unit

CPU Control Design– There are two major types of control organization.

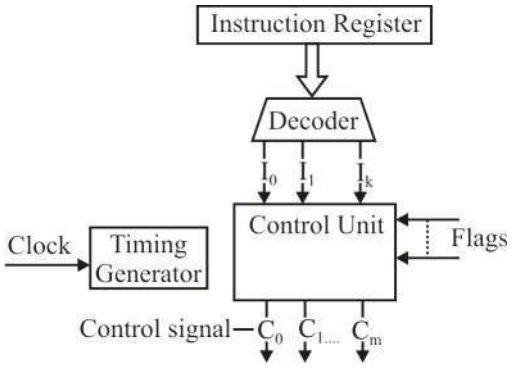
(i) Hardwired Control

(ii) Micro programmed control

Hardwired Control Unit– Control logic is implemented with gates, flip-flops, decoders and other digital circuits.

Advantage– Can be optimized to produce a faster mode of operation.

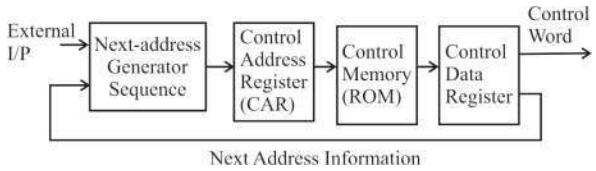
Disadvantage– Rearranging the wires among various components is difficult.



➲ **Micro-Programmed Control Unit–** Control Logic is implemented with micro programmed.

Advantage– Updating the control logic is easy.
Disadvantaged– Slower than hardwired control unit.

Control Word Sequencing



Example– CPU has 64 distinct instructions each instruction takes 8 micro operation micro-instruction–

(i) Signals (128 bits)

(ii) Mux select [16 mux inputs]

(iii) Address

What is size of control memory?

Solution– Total micro operations = 64*8

$= 2^6 * 2^3 = 2^9$

Memory address = 9 bits

Signals	MUX select	Address
128	4	9

Total micro Instruction = 128 + 4 + 9 = 141 bit

so size of control memory = 2^9 *141 bits

Difference between Horizontal Micro Programming and Vertical Micro Programming

	Horizontal Micro Programming	Vertical Micro Programming
1.	Control Word Large	Control word is small
2.	Parallelism is high	No parallalism
3.	Faster	Slower

Difference between Hardwired Control Unit and Micro Programming Unit

	Hardwired Control Unit	Micro Programming Unit
1.	Fixed Instruction	Instruction can flexible
2.	High speed	Slower compared to hardwired instruction
3.	Expensive	Relatively cheap
4.	Complex	Simple
	Example– Intel 8085, Motorola 6802 Tilog 80 and any Reduce Instruction Set Computer (RISC)	**Example–** INTEL 8080, Motorola 68000 and any complex instruction set computer.

Difference between RISC and CISC

RISC	CISC
Less number of instruction	More number of instruction
Fixed length instruction	Variable length instruction
Simple Instruction	Complex Instruction
Limited addressing	More & complex addressing modes
Easy to implement using hardwired control unit	Difficult to implement using hardwired
One cycle per instruction	Multiple cycle per instruction
Register to register arithmetic operation only	Register to memory & memory to register arithmetic operations possible
More number of registers	Less number of registers

Example– Consider the following processor design characteristics.

(i) Register to register arithmetic operation only

(ii) Fixed length instruction format

(iii) Hardwired control unit

Which of the characteristics above are used in the design of a RISC processor?

(a) i and ii only (b) ii and iii only

(c) i and iii only (d) i, ii and iii

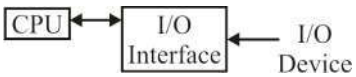
IO Organization

Peripheral Device– Devices connected to processor externally are known as peripherals. There are three type–

- Input Devices
- Output devices
- Storage devices

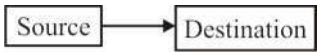
Input/Output Subsystem Of Computer– Provides an efficient mode of communication between control system & outside word.

Input/Output Interface– Provides a method for transferring information between internal storage (memory & registers) & external I/O devices.

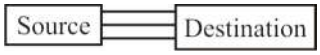


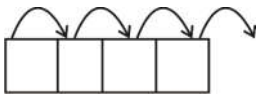
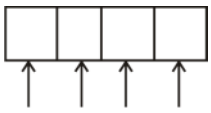
- **Serial and Parallel Transmission**

(i) Serial Transmission



(ii) Parallel Transmission



Difference between Serial and Parallel	
Serial	**Parallel**
Need of shift register 	No need of shift register 
Burst errors	Bit errors
Cheaper	Costless
Less reliable	More reliable
Used to more distance	Used to less distance

Asynchronous Transmission– Data is sent in form of byte or character. This transmission is the half-duplex type transmission. In this transmission start bits and stop bits are added with data.

Example–

- Email
- Forums
- Letters

Synchronous Transmission– Data is sent in form of blocks or frames. This transmission is the full-duplex type. Between sender and receiver, synchronization is compulsory.

Example–

- Chat rooms
- Telephonic conversations

Example-

How many 8 bit characters can be transmitted per second over 9600 baud (bits/sec) serial communication link using a parity synchronous mode of transmission with 1 start bit, 8 data bits, 2 stop bits and 1 parity bit.

Solution– For 1 char = 1 + 8 + 2 + 1 = 12 bits

$$\text{Characters transmitted} = \frac{9600}{12} = 800\,\text{char/sec}$$

➲ **Mode of transfer–** 3-way to perform I/O

- Programmed I/O
- Interrupt driven I/O
- DMA

Programmed I/O

{1. Read the status flag.

2. It data is not available (status = 0) then go to step 1.

3. Move the data}

- Waste time processor
- Mostly devices time to transfer 1 byte.

➲ **Interrupt Initiated IO–**

- IO device has a provision (interrupt signal) to inform to CPU about communication

When CPU receives interrupt–

- It completes execution of current instruction
- Saves the status (PC, PSW etc.) of current process onto the stack
- Branches to service the interrupt
- Resumes the previous process by taking out the values from stack.

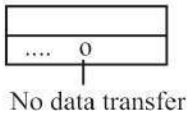
➲ **DMA (Direct Memory Access)**

- Enables data transfer between I/O and memory without CPU intervention.
- Need a hardware : DMAC

Starting Address – Memory Address starting from where data transfer should be perform.

Data Count– No of bytes or word to be transferred.

	Initially	After 1B	After 2B	After 3B
Starting Address	1000	1001	1002	1003 .....
Data Count	500	499	498	497 .....



➲ **Modes of DMA Transfer**

- Burst mode (all data transfer at the same time)
- Cycle stealing (it happen word by word)
- Interleaving DMA (Whenever CPU does not require system buses (doing internal work) then only control of the buses given to DMAC).

➲ **Flynn's Classification of computer**

- SISD– Single Instruction Stream, Single Data Stream

Example– Von-neamann

- SIMD– Single Instruction Stream, Multiple Data Stream

Example– Pipeline processor

- MISD– Multiple Instruction Stream, Single Data Stream only hypothetical
- MIMD – Multiple Instruction Stream, Multiple Data Stream

Example– Multiple Pipelines (Super Scalar Computer (ILP))

➲ **Pipelining**

- Pipelining is useful, when same processing is applied over multiple inputs.
- Technique to decompose a sequential process into sub-operations.
- Sub-operations performed in all segments.
- Task: one operation performed in all segments.

➲ **General Consideration about pipeline**

- Consider a K segment pipeline with clock cycle time = t_p to perform n tasks.

Time required to perform 1st task = $K*t_p$

Time required to perform remaining (n –1) tasks = $(n-1)t_p$

Time required for all n tasks = $(K+n-1)t_p$

(K + n – 1) is total cycle.

- Consider a non-pipeline system that takes t_n time to perform a task

Time required for n task = $n*t_n$

- Performance of a pipeline is given by speed up ratio.

$$\text{Speed up ratio} = \frac{\text{Non}-\text{pipeline time}}{\text{Pipeline time}}$$

$$S = \frac{n * t_n}{(K+n-1)t_p}$$

as the number of task increases, n>>K (Ignore K–1)

$$\boxed{S_{ideal} = \frac{t_n}{t_p}}$$

Latency– Time after machine takes next input.

- Latency in non-pipeline = t_n
- Latency in pipeline = t_p

Throughput– No of input processed per unit of time $= \frac{n}{(K+n-1)t_p}$

Ideal case–

$$\text{Throughput} = \frac{1}{t_p}$$

Example– Consider a 5 stage pipeline with cycle time of 15 ns. Calculate processing time of pipeline for 500 tasks.

n = 500, K = 5, t_p = 15

processing time of pipeline = $(K + n - 1)t_p$

$= (5 + 500 - 1)15$

$= 504 \times 15$

$= 7560$ n sec

➲ **Instruction Pipeline**

If pipeline processing is implemented on instruction cycle.

IF : Instruction fetch

ID : Instruction decode & Address calculation

OF : Operand Fetch

EX : Execution

WB : Write Back

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
I1	IF	ID	OF	EX	WB											
I2		IF	ID	OF	EX	WB										
I3			IF	ID	OF	EX	WB									
I4				IF	ID	OF	EX	WB								
I5					IF	–	–									
I6																
I7																
I8								IF	ID	OF	EX	WB				
I9									IF	ID	OF	EX	WB			

Assume in cycle number 5, I_4 decoded as branch instruction.

By standard $\Rightarrow$ Branch decision is made in 'Execution' phase.

- After execution phase next instruction can be fetched.

Actually only instruction executed (n = 6) K = 5 for n input, no. of cycles = K+ n – 1

$= 5 + 6 - 1$

$= 10$

3 cycles are extra (stall cycles because of branch)

Total = 10 + 3 = 13 cycles

Total execution time = cycle * t_p

➲ **Pipeline Hazards–** Situations that prevent the next instruction from being executing its designated clock cycle.

- Hazard lead to have stall cycles

There are three type–

1. Structural Hazard/Resource Conflict
2. Data Hazard/Data Dependency
3. Control Hazard/Branch difficulty

Structural Hazard– When multiple instruction need same resource.

Control Hazard– All Instruction who change the program counter leads to control hazard because of branch instruction.

Data Hazards Classification–

- RAW (Read After Write) [Flow/true data dependency]
- WAR (Write After Read) [Anti Data dependency]
- WAW (Write After Write) [Output data dependency]

02. DATA STRUCTURE AND ALGORITHM

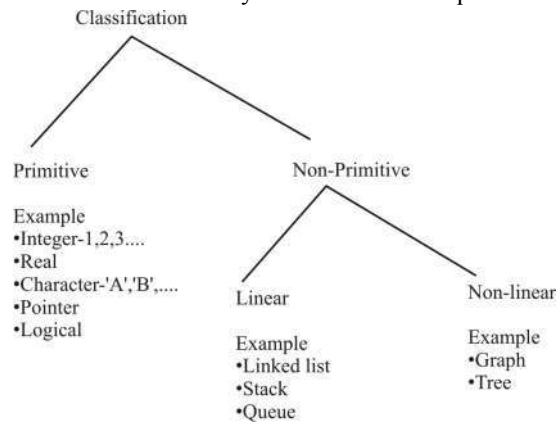
DATA STRUCTURE

Data- Data is a collection of raw, unorganized facts and details like text, observations figures, symbols, and descriptions of things etc.

Data can be record and does not have any meaning unless processed.

Information- Information is processed, organized, and structured data.

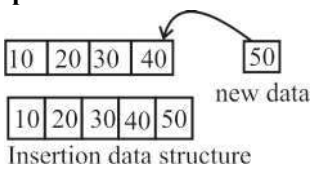
Definition of data structure – A data structure is a collection of data, generally organized so that items can be stored and retrieved by some fixed techniques.



Data structure operation

Insertion- Add a new data in the data structure.

Example–



Deleting- Remove a data from the data structure.

Example-

10	20	30	40	50

Deleting

20	10	30	40

Sorting- Arrange data in increasing or decreasing order.

Example-

data- 20, 30, 10, 5, 6

5	6	10	20	30

Searching- Final the location of data in data structure.

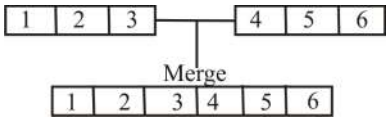
Example-

5	6	10	20	30
a[0]	a[1]	a[2]	a[3]	a[4]

Search = 10 → a[2]

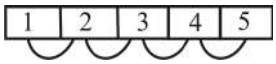
Merging- Combining the data of two different sorted files into a single sorted.

Example-



Traversing- Accessing Each data Exactly one in the data structure so that Each data item is traversed or visited.

Example-

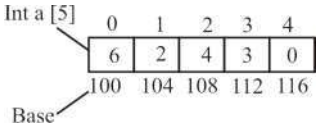


Arrays in Data structure

Array- Array is a fixed size sequenced collection of data items of same type.

- Access a particular element = array name [index]
  = Int a [ ]
- data items are stored in continuous locations.

How the data is to be accessed, how you can calculate the address.



a[0]=6, a[1]=2, a[2]=4, a[3]=3, a[4]=0

Formula-

Address of a[i] = Base +i * (size of data type)

i = 0, 1, 2……..(n - 1)

n = size of array

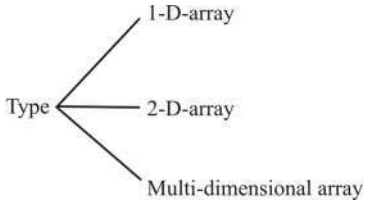
so if want to, i = 2;

Int size = 4 bit

= 100+2×4

= 108

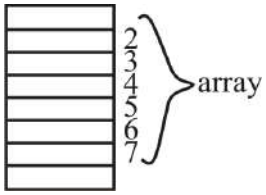
- Random Access taking Constant time.



- Name [LB:UB] LB ≤ UB

LB = Lower Bound

UB = Upper Bound



Size of array = UB-LB+1

Consider an array A [6:18].

Total number of element in array

By formula

Size of array = UB – LB+1

= 18 – 6 + 1

= 13

Searching Array–

- Searching an array means to find a particular element in the array. The search can be used to return the position of the element or check if it exists in the array.

Linear Search	**Time complexity**
Average	O (n)
Best	O (1)
Worst	O (n)

Binary search	**Time complexity**
Average	O (log n)
Best	O (1)
Worth	O (n)

- Pointer is an address of another variable

Example–

int b = 10;

int* P; b P

10	200

P = & b; 200 204

➲ 2-D Array

The two-dimensional array can be defined as an array of array.

2D array syntax–

data type array_name [row][columns];

Representation of 2-D Arrays in memory

Row major order-

a [i][j] = B+((i*n)+ j)*size.

IF start (0,0)

A [i][j] = [((i-1)*n+(j-1))*size + Base if start (1,1)

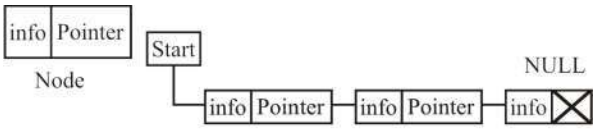
Colom major

A [i][j] = ((i*m) + i)*size + Base

Linked List

A Linked list is a Linear data structure in which the elements are not stored at contiguous memory location.

- A liked list is a dynamic data structure.
- Each element is called a node which has two part, info part stores the information and pointer part which point to the next element.

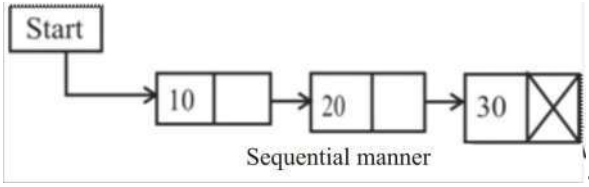


Operation On Linked List-

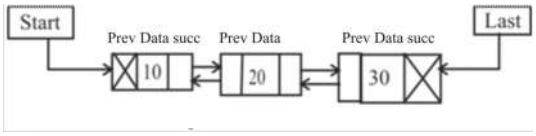
1. **Creation -** This operation are used to created a linked list in this node is created and linked to the Another node.
2. **Insertion-** This operation is used to insert a new node in the linked list.
3. **Deletion-** In this operation, elements can be deleted at the starting of the list.
4. **Traversing-** It is a process of going through all the nodes of linked list from one End to the other end.

Type of Linked List

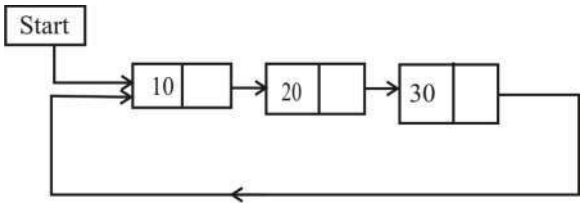
- **Singly Linked List-** A singly linked list is a unidirectional linked list. It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type.



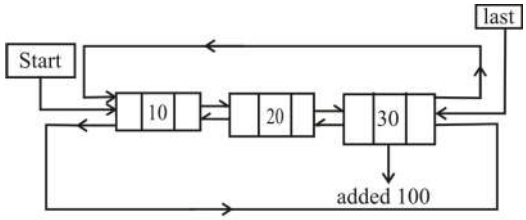
Doubly Linked List- A doubly linked list is a bidirectional linked list that contains a pointer to the next as well as the previous node in sequence.



- **Circular Linked List-** A circular linked list is that in which the last node contains the pointer to the first node of the list.



- **Circular Doubly Linked List-** A circular doubly linked list is a mixture of a doubly linked list and a circular linked list.



Representation of Linked List–

- A data item
- An address of another node

We wrap both the data item and the next reference in a struct as.

Syntax–

```
Struct node
{ Int data;
Struct node*next;
};
```

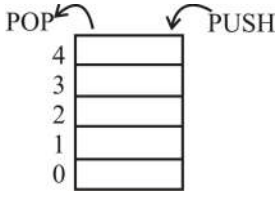
➲ **Linked list Applications**

- Dynamic memory allocation.
- Implemented in stack and queue.
- In undo functionality of software.
- Hash tables, graphs.

STACKS

Stack is a non-primitive linear data structure.

It is an ordered list in which addition of new data item and deletion of already existing data item is done from only one End know as Top of Stack (TOS).



- It follows the LIFO pattern, which means the last added element will be the first to be Removed from the stack.

➲ **Stack has two operation–** 1. PUSH Operation
2. POP Operation

PUSH Operation- Every PUSH operation TOP is incremented by one.

TOP = TOP + 1

In case the Array is full no new Element is added. This condition is called stack full or stack over flow condition.

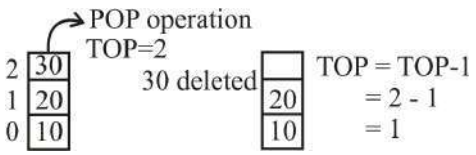
- The process of adding a new element of the TOP of stack is called PUSH operation.

POP Operation- The process of Deleting an element from the top of stack is called POP.

After Every POP operation the stack TOP is decremented by one.

TOP = TOP-1

(This is called operation stack underflow).



Stack Notation- There are three stack notation.

- **Infix Notation-** Where the operator is written in between the operands.

Example- A + B + operator
A, B operands

- **Prefix Notation-** In this operator is written before the operands. It is also know as polish Notation.
  Example- +AB
- **Post fix Notation-** In this operator is written After the operands. It is also know as suffix Notation.
  Example- AB+
- **Convert the following Infix to prefix and postfix for (A + B)* C/D+E ^ F/G**

Prefix –

(A+B)* C/D + E ^ F/G

+ AB* C/D + E ^ F/G

Let + AB = R_1R_1

* C/D + E^F/G

R_1 * C/D+^ EF/G

Let ^ EF = R_2

R_1 * C/D+R_2/G

R_1 */CD + R_2/G

Let /CD = R_3

R_1*R_3 + R_2/G

R_1*R_3 + /R_2G

Let /R_2G = R_4

R_1*R_3 + R_4

*R_1R_3 + R_4

Let *R_1R_3 = R_5

R_5 + R_4

+R_5R_4

Now enter the value of R_5, R_4, R_3, R_2, R_1

+*R_1R_3/R_2G

+*+AB/CD/AEFG

Post fix-

(A+B)*C/D+E ^ F/G

AB+*C/D+E ^ F/G

Let AB+=R_1

R_1*C/D+EF ^ /G

Let EF ^ =R_2

R_1*C/D+R_2/G

R_1*CD/+R_2/G

Let CD/ = R_3

R_1*R_3+R_2/G

R_1*R_3+R_2G/

Let R_2G/=R_4

R_1*R_3+R_4

R_1R_3*+R_4

Let R_1R_3*=R_5

R_5+R_4+

Now Enter the value of R_5, R_4, R_3, R_2, R_1

R_5R_4+

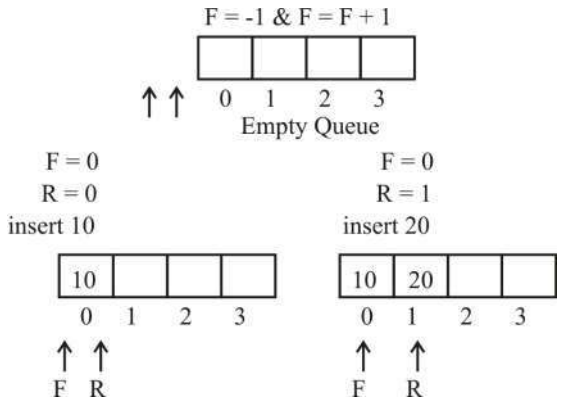
$R_1R_3*R_4+$

$AB+CD*R_2G/+$

AB+CD* EF ^ G/+

QUEUE

- Queue is a Non primitive Linear data structure.
- The first added Element will be the first to be remove from the queue that is the reason queue is called (FIFO) type list.
- In queue Every insert operation Rear is incremented by one. (R = R + 1) and every delete operation front is increment by one.



BASIC OPERATION OF QUEUE

- **Enqueue:** Add an element to the end of the queue.
- **Dequeue:** Remove an element from the front of the queue.
- **Is Empty:** Check if the queue is Empty.
- **Is full:** Check if the queue is full.
- **Peek:** Get the value of the front of the queue without removing it.

Application of Queue-

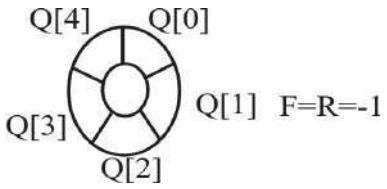
- CPU scheduling, Disk scheduling
- Handling of interrupts in real time system
- Call center phone system use Queues to hold people calling them in order.
- Unlike stack, Queue is also considered as the ordered list of the data the ordered list of the data that has a similar data type.

Priority Queue- Priority Queue is an Extension of queue with following properties. An element with high priority is dequeued before an element with low priority.

Operations-

- **Insert (item, priority):** Inserts an item with given priority time (O(logn)).
- **Get highest priority ( ):** Returns the highest priority item. operation implemented by linear searching the highest priority item in array. O(1).
- **Delete highest priority ():** Removes the highest priority operation can be implemented by first linearly searching an item, the removing the item by moving all subsequent items one position back. (time O(logn)).
- **Heap** is generally preferred for priority queue implementation because heaps provide better performance compared array or linked list.

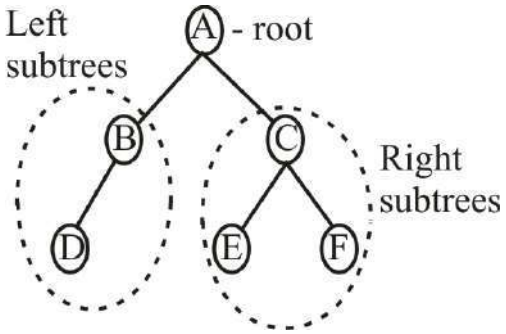
Circular Queue- A circular queue is one in which the insertion of a new element is done at very first location of queue is full.



- A circular queue overcome the problem of utilized space in linear queues implemented as arrays.

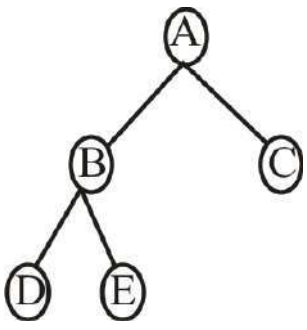
BINARY TREE

- Binary tree is a finite set of data item which is either empty of consists of a single item called root and two disjoint binary tree called the left sub tree and right sub tree.
- In Binary tree, every node can have maximum of two children which are known as left child and right child.

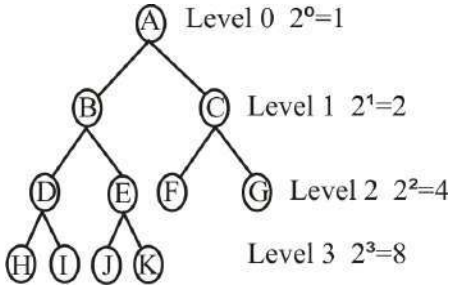


Types of Binary Tree

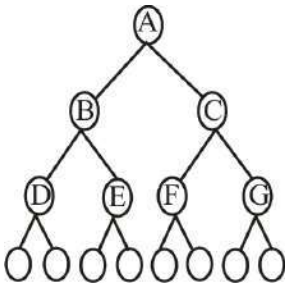
Full Binary Tree- A Binary tree is full if every node has 0 or 2 child.



Complete Binary Tree- A Binary tree is complete Binary tree if all level are completely filled.



Perfect Binary Tree- A Tree in which all internal nodes has two children and all leaves are at the same level in which all level has 2^n children.



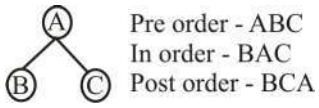
Traversal of Binary Tree

1. Preorder traversal (NLR) Node left Right

2. In order traversal (LNT) Left Node Right

3. Post order traversal (LRN) Left Right Node

Example-



B-Tree

B-Tree is a self-balancing tree. in most of the other self-balancing searching (like AVL and Red-Black tree). It assumed that everything is in the main memory.

Time complexity of B-Tree

Algorithm	Time complexity
Search	O (logn)
Insert	O (logn)
Delete	O (logn)

Properties of B-Tree

- All nodes of the leaf must be on the same level.
- At least two root nodes are required.
- Each B-Tree node a maximum of m children.
- Each node in a B- tree includes at least m/2 children, except the root and the leaf node.
- Maintains sorted data.
- Minimum children:

leaf = 0

root = 2

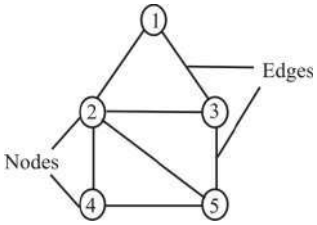
$$\text{Internal nodes} = \left\lceil \frac{m}{2} \right\rceil$$

- Every node has max. (m-1) keys
- Min key:- root node $\rightarrow$ 1

$$\text{all other} = \left\lceil \frac{m}{2} \right\rceil - 1$$

GRAPH

Graph in data structure are non linear data structure made up of a finite number of vertices and the edges that connect them. Graph in data structure are used to address real problem in which it represents the real problem in which it represents the problem area as a network like telephone networks, circuit network and social network.



This graph has a set of vertices

V = { 1, 2, 3, 4, 5} and a set of edges

E = {(1,2), (1,3), (2,3), (2,4), (2,5), (3,5), (4,5)}

Operation Of Graph In Data Structure

- Creating graphs
- Insert vertex
- Delete vertex
- Insert edge
- Delete edge

Graph Traversal Algorithm–

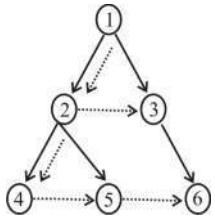
Graph traversal is a subset of tree traversal. There are two techniques to implement a graph traversal Algorithm.

1. Breadth - first Search or BFS

2. Depth – first – search or DFS

BFS- Two data structure for traversing the graph.

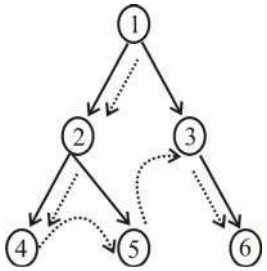
- Visited array (size of the graph)
- Queue data structure
- Using the FIFO concept.
- Until the queue is not empty and no vertex is left to be visited.



BFS	1	2	3	4	5	6

DFS- The (DFS) algorithm traverses or explores data structure such as trees and graphs. The DFS algorithm begins at the root nodes and examines each branch as for feasible before backtracking.

- Examine any two data structures for traversing the graph.
- Visited array (size of the graph)
- Stack data structure
- Using the FIFO principle.
- Stack data structure is not empty.



DFS	1	2	4	5	3	6

Application of graph

- The friend suggestion system on facebook is based on graph theory.
- Graph transformation system manipulate graph in memory using rules. Graph databases store and query graph–structure data in a transaction–safe permanent manner.

Difference between stack and Queue	
Stack	**Queue**
The collection of element in last in first out (LIFO).	The collection of Element in first in first out (FIFO)
Objects are inserted and removed at the same end called TOP of sack.	Objects are inserted and removed from different ends called Front and Rear end.
Insert operation is called PUSH operation.	Insert operation is called Enqueue operation.
Delete operation is called POP operation.	Delete operation is called Dequeue operation.
In stack There is no wastage of memory space.	In Queue there is a wastage of memory space.
Plate counter at marriage Reception is an Example of stack.	Students standing in a line at fees counter is an Example of Queue.

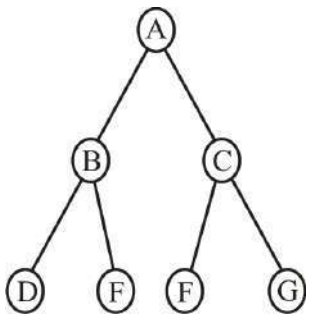
Non-Linear Data structure	Linear Data structure
The non- linear data structure are comparatively difficult implement and understand as compared to linear data structure.	The linear data structure are comparatively easier to implement
The data element connect to each other hierarchically.	The data element connect to each other sequentially.
It is not easy to traverse in multiple runs	You can traverse in a single run.
It is memory friendly.	It is not very memory friendly.
Map, Graph, Tree	List, Array, Stack, Queue

Array	Linked list
Array is a collection of Homogeneous (same) data type.	Linked-list is a collection of node (data & address)
Size of an Array is fixed	Size of list is not fixed.
Memory is allocated from stack.	Memory is allocated from heap.
Work with static data structure	Work with dynamic data structure.
Array Element are independent to each other.	Linked list Element are depend to each other.
Array take more time (Insertion & Deletion)	Linked- list take less time (Insertion & Deletion)

Tree	Graph
There is a unique node called root in tree.	There is no unique node.
Tree is a collection of node and edges. Ex- T{node, Edges}	Graph is a collection of vertices/nodes and edges. Ex G = {V,E}
There will not be any cycle/loops.	There can be loops/cycle.
In this Pre-order, In-order and post order Traversal.	In this BFS and DFS traversal.

HEAP DATA STRUCTURE

A Heap is a special Tree-based structure in which the tree is complete binary tree.



A	B	C	D	E	F	G
1	2	3	4	5	6	7

Formula-

if a Node is at index i
its left child is at = 2*i
its right child is at = 2*i+1

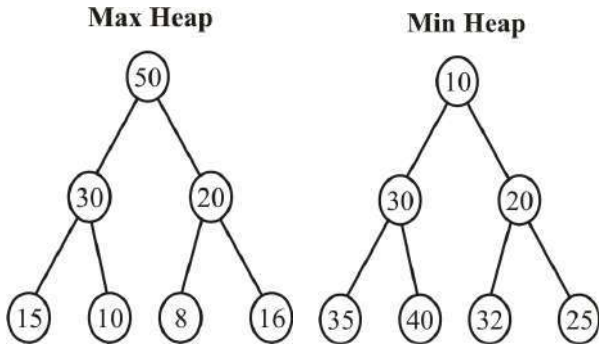
it parent is at = $\left[\frac{i}{2}\right]$

(This is true when your heap is starting from index 1.)

If you represent a binary tree in an array then they should not be any empty locations or gaps in between the elements bins from first element to last element in between anywhere.

- Height of a complete binary tree will be minimum only that is log n.

HEAP



Max Heap- A max heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of the node. Max heap data structure is useful for sorting data using heap sort.

50 > 30, 20 30 > 15, 10 20 > 8, 16

Min Heap- A min heap is a heap where every single parent node including the root, is less than or equal to the value of its children nodes. The most important property of a min heap is that the node with the smallest, or minimum value, will always be the root node.

10 < 30, 20 30 < 35, 40 20 < 32, 25

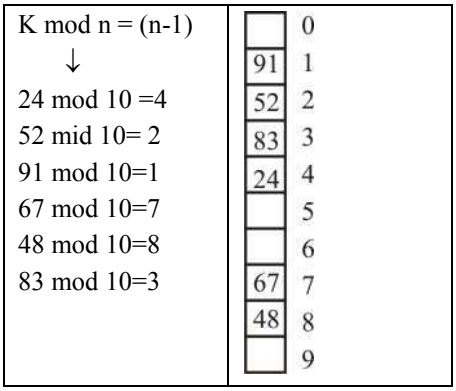
Operations of Heap

- **Heapify–** a process of creating a heap from an array.
- **Insertion–** time complexity O (log N)
- **Deletion –** time complexity O (log N)
- **Peek–**To check or find the most prior element in the heap.

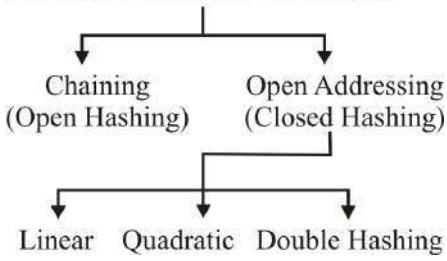
HASHING

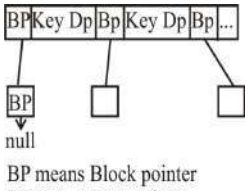
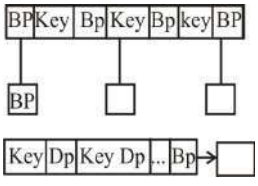
Hashing is a technique or process of mapping keys and values into the hash table by using a hush function. It is done for faster access to elements.

- Hashing storing and Retrieving data in O (1) time.
- Search key (24, 52, 91, 67, 48, 8 3)
- Hash Table
- Hash Function (K mod 10, K mod n, Mid, Square) folding method.



Collision in Hashing
or
Collision Resolution Techniques

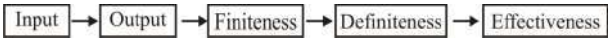


B-Tree & B^+Tree	
Data is stored in leaf as well as internal nodes	Data is stored only in leaf nodes.
Leaf nodes not linked together	Linked together like Linked list.
Searching is slower deletion complex	Searching is faster, deletion easy (directly from leaf node)
	

ALGORITHM

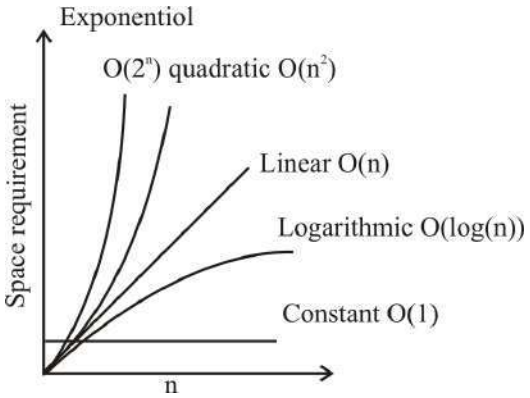
Analysis of Algorithms

- A well defined procedure to solve a specific problem is called Algorithm.
- Data structure + Algorithm = Programming
- Algorithms Criteria



Asymptotic Notations

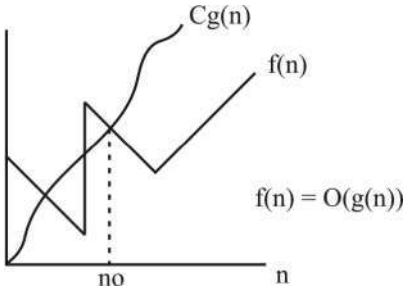
Asymptotic notations are abstract notation for describe the behavior of Algorithm and determine the rate of growth of a function.



1. Big O Notation

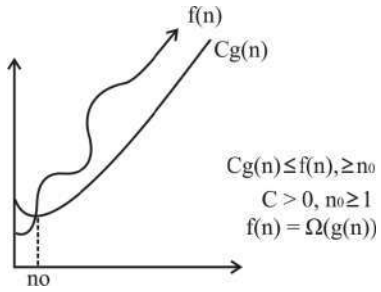
- The Big O Notation defines an upper bound of an algorithm. Therefore, it gives the worst-case complexity of an algorithm.
- O (g(n)) = {f(n): there exist positive constants C and n_0 such that

$$O \le f(n) \le Cg(n) \quad \text{for all } n > n_0$$



2. Omega (Ω) Notation:

Omega notation represents the lower bound of the running time of an algorithm. thus, it provides the best case complexity of an algorithm.

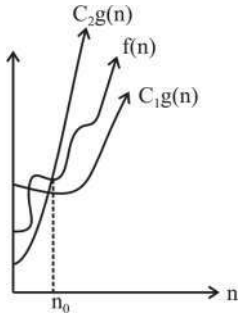


3. Theta (θ) Notation:

Theta notation encloses the function from above and below. It is used for analyzing the average case complexity of an algorithm.

$$C_1 g(n) \le f(n) \le C_2 g(n)$$

$$C_1 C_2 \ge 0, n \ge n_0, n_0 \ge 1$$

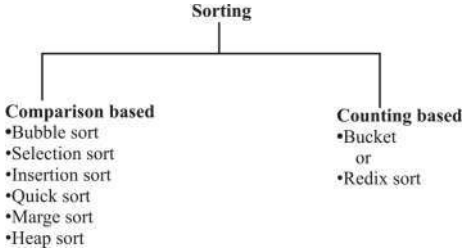


f(n) = θ(g(n))

Worst Case Analysis– The case that causes a maximum number of operation to be executed.

Best Case Analysis– The case that causes a minimum number of operations to be executed .

Average Case Analysis– We take all possible inputs and calculate the computing time for all of the inputs, sum all calculate values and divide the sum by the total number of inputs.



BUBBLE SORT

Algorithm–

1. Start with an array of unsorted numbers
2. Defines a function called 'bubblesort" that takes in the array and the length of the array as parameters.
3. In the function, create a variable called "sorted" that is set to false.
4. Create a for loop that iterates through the array starting at index 0 and ending at the length of the array –1
5. Within the for loop, compare the current element with the next element in the array.
6. If the current element is greater than the next elements, swap their positions and set "sorted" to true.
7. After the for loop, check if "sorted" is true.
8. If "sorted" is true, call the "bubblesort" function again with the same array and length as parameters.
9. If "sorted" is false, the array is now sorted and the function will return the sorted array.
10. Call the "bubbleSort" function with the initial unsorted array and its length as parameters to befin the sorting process.

- It is stable sorting techniques.
- Recursive relation in Bubble sort.

$$T(n) = T(n-1) + n$$

- It is In place sorting
- Time complexity = $\theta(n^2)$

Space complexity = 1 + n

= θ (n)

Best case = θ (n)

Worst case = $\theta(n^2)$

Average case = $\theta(n^2)$

SELECTION SORT

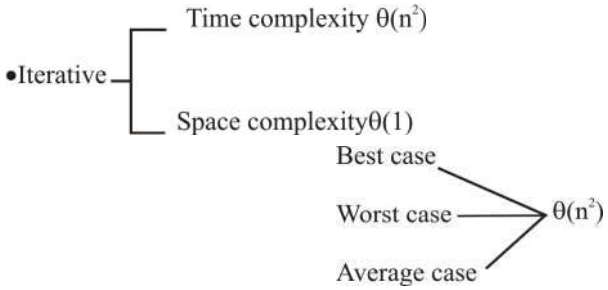
Algorithm–

1. Initialize minimum value(min_idx) to location 0.
2. Traverse the array to find the minimum element in the array.
3. While traversing if any element smaller than min_idx is found then swap both the values.
4. Then, increment min_idx to point to the next element.
5. Repeat until the array is sorted.

- Recursive relation

$T(n) = T(n-1) + (n-1)$

- Max swap = (n-1) Complexity $\theta(n)$
- Min swap = 0



- It is unstable sorting

INSERTION SORT

Algorithm– To sort an array of size N in ascending order:

1. Iterate from arr[1] to arr[N] over the array.
2. Compare the current element (key) to its predecessor.
3. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements before. Move the greater elements one position up to make space for the swapped element.

- Number of comparison– $\max = \frac{n(n-1)}{2}$

$\min = n - 1$

- Best case = $\theta(n)$

Worst case & Average case = $\theta(n^2)$

- Insertion sort = Position +Shifting
- It is highly affected due to order of input.
- Recursive time T (n) = T (n-1)+n

$= \theta(n^2)$

space = $\theta(n)$

- Stable sorting

QUICK SORT

- It is Divide and conquer technique.
- It is unstable sorting
- Best case = θ (n logn) = Average case

Worst case = $\theta(n^2)$

- Recursive

$T(n) = 2T(n/2)+n$

Using master theorem

θ (n logn)

- Number of comparison = $\frac{n(n-1)}{2}$

Merge Sort

Algorithm–

MergeSort(arr[],l,r)

If r > 1

1. Find the middle point to divide the array into two halves.

middle m = l + (r – 1)/2

2. Call mergeSort for first half:

Call mergeSort(arr, l, m)

3. Call mergeSort for second half:

Call mergeSort(arr, m + 1, r)

4. Merge the two halves sorted in steps 2 and 3:

Call merge(arr, l, m, r)

- It is stable sorting.
- It is pure divide & conquer.
- Recursive–

$T(n)= 2T(n/2)+n$

- Complexity–

Best case = Average case = Worst case = θ(n logn)

RADIX SORT

- It is counting Based sorting.
- It is outplace sorting
- Q (n k)
  - If n>>K=Q(n)
  - If n=K=Q(n²)

  ] Time complexity
- Q (n*R) space complexity.

RECURSION

A recurrence is an equation or inequality that describes a function in terms of its values on smaller inputs. To solve a Recurrence Relation means to obtain a function defined on the natural numbers that satisfy the recurrence.

- There are four methods for solving Recurrence

(i) Substitution method (iii) Iteration method

(ii) Recursion tree (vi) Master method

(i) Substitution method–

$$T(n)=\begin{cases}1 & n=0\\ T(n-1)+1 & n>o\end{cases}$$

$$T(n)=T(n-1)+1 \quad \begin{cases}\because T(n)=T(n-1)+1.....(i)\\ \therefore T(n-1)=T(n-2)+1.....(ii)\end{cases}$$

Substitution $T(n-1)$

$T(n)=[T(n-2)+1]+1$ {From equ^{x} (ii)}

$T(n)=T(n-2)+2$

$T(n)=T(n-3)+3$

⋮

⋮Continue for k time

⋮

$\boxed{T(n)=T(n-k)+k}$ Assume $n-k=0$

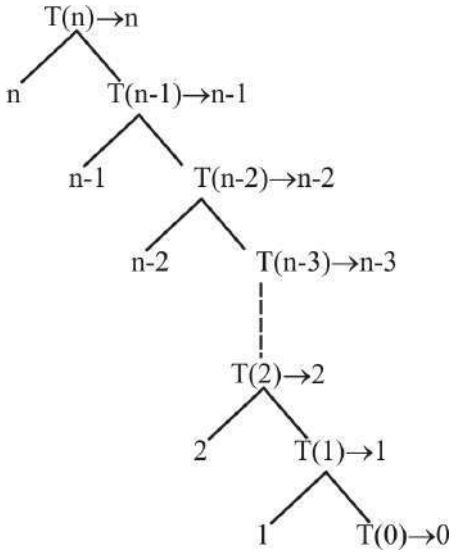
$T(n)=T(n-n)+n \quad \therefore n=k$

$T(n)=T(0)+n$

$T(n)=1+n \quad \boxed{\therefore \theta(n)}$

(ii) Recursion Tree –

$$T(n)=\begin{cases}1 & n=0\\ T(n-1)+n & n>0\end{cases}$$



$0+1+2.....n-1+n=\frac{n(n+1)}{2}$

$T(n)=\frac{n(n+1)}{2}$

$\boxed{\theta(n^2)}$

(iii) Iteration Method–

$$T(n)=\begin{cases}1 & n=0\\ T(n-1)+n & n>0\end{cases}$$

$T(n)=T(n-1)+n$ $\{\because T(n)=T(n-1)+2$

$T(n)=[T(n-1)+n-1]+n.....(i)$ {T(n-1)=T(n-2)+n-1

$T(n)=T(n-2)+n-1]+n.....(ii)$

$T(n)=T(n-3)+(n-2)+(n-1)+n.....(iii)$

⋮

: Continue of k time

⋮

$T(n)=T(n-k)+(n-(k-1))+(n-(k-2)+.....+(n-1)+n$

Assume $n-k=0$

$$\therefore n=k$$

$T(n)=T(0)+1+2+3+.....n-1+n$

$$=1+\frac{n(n+1)}{2}$$

$$T(n)=1+\frac{n(n+1)}{2}=\theta(n)^2$$

- $T(n)=2T(n-1)+1$

By solving recursion tree

$1+2+2^3+.....+2^K=2^{k+1}$ (This is GP series)

$=\theta(2^n)$

(iv) Master theorem for Decreasing function

$T(n)=aT(n-b)+f(n)$

$a>0$, $b>0$ and $f(n)=O(n^K)$ where $K\geq 0$

Case

• If a<1	• If a=1	• If a>n
$O(n^k)$	$O(n^{K+1})$	$O(n^K a^{n/b})$
$O(f(n))$	$O(n*f(n))$	$O(f(n)a^{n/b})$

- **Master Theorem for Dividing function**

(i) $\log_b{}^a$ $T(n)=a\,T(n/b)+f(n)$

$a\geq 1$

(ii) $b>$ $f(n)=\theta(n^k \log^P n)$

Case1: if $\log_b{}^a>k$ them $\theta(n\log_b{}^a)$

Case2: if $\log_b{}^{a=k}$

If $P>-1$ $\theta(n^k \log^{p+1} n)$

If $P=-1$ $\theta(n^k \log\log n)$

If $P<-1$ $\theta(n^k)$

Case3: If $\log_b{}^a<K$ If $P\geq 0$ $\theta(n^k \log^p n)$

If $P<0$ $O(n^k)$

$T(n)=2T(n/2)+1$

$a=2$, $b=2$, $f(n)=O(1)=\theta(n^0 \log^0 n)$

$\log_2{}^2=1>K=0$

Case1: $\theta(n^1)$

Recurrence Relation

$T(n)=T(n-1)+1$	$O(n)$
$T(n)=T(n-1)+1$	$O(n^2)$
$T(n)=T(n-1)+\log n$	$O(n\log n)$
$T(n)=T(n-1)+n^2$	$O(n^3)$
$T(n)=T(n-2)+1$	$\frac{n}{2}O(n)$
$T(n)=T(n-100)+n$	$O(n^2)$