ACE
Engineering Academy
(Leading institute for ESE/GATE/PSUs)

ESE – 2019 MAINS
OFFLINE TEST SERIES

ELECTRONICS & TELECOMMUNICATION ENGINEERING (E&T)

TEST – 6
SOLUTIONS

All Queries related to **ESE – 2019 MAINS Test Series** Solutions are to be sent to the following email address
testseries@aceenggacademy.com | Contact Us : 040 – 48539866 / 040 – 40136222

01. (a)

Sol:

(i) In general "no", but possible if you can create your own system software or application to run on system hardware directly.

The Operating system was created to ensure the ease of use of a computer by a common man without having much knowledge of computer internals.
Now if you want to use a computer without OS, then more than saying yes or no, one should understand the purpose of doing so. What you want to do with a computer without OS?. A computer without OS is a bare hardware. You either have to develop your own OS sort of software, which boots your computer and gives you an interface to run your program or you have to use existing operating systems in your computer.

The OS less embedded systems work without OS. As soon as you power on the box, it boots the systems and keeps running the application. These systems are used to perform a fixed set of functions as designed. They cannot be used like generic computer. Similar thing you can implement in Computer also, you write your own custom software, which boots the system and runs your program.

Operating systems are designed to create an abstraction on the underlying hardware in such way that, it facilitates the users to interact easily with the computer and run their programs. It creates a generic framework to run any kind of applications and manages the underlying things. Having an OS, a PC can be used as document editor, development system, multimedia player, communication device etc based on whatever application you run on PC.
Computer is yet another hardware generally based on X86 processor. You can either have an OS and boot the system or you can have your own custom application to start the system, then you need to develop this as per your necessity.

The services provided by operating systems as follows:

1. **Program execution:**
   Operating systems handle many kinds of activities from user programs to system programs like printer spooler, name servers, file server, etc. Each of these activities is encapsulated as a process.
   A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use).

2. **I/O Operation:**
   An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.
   An Operating System manages the communication between user and device drivers.

3. **File system manipulation:**
   A file represents a collection of related information. Computers can store files on the disk, for long-term storage purpose. A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

4. **Communication:**
   In case of distributed systems which are a collection of processors that do not share memory, peripheral devices, or a clock, the operating system manages communications between all the processes. Multiple processes communicate with one another through communication lines in the network. The OS handles routing and connection strategies, and the problems of contention and security

5. Error Handling:

Errors can occur anytime and anywhere. An error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling −

- The OS constantly checks for possible errors.
- The OS takes an appropriate action to ensure correct and consistent computing.

6. Resource Management:

In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management −

- The OS manages all kinds of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

7. Protection:

Considering a computer system having multiple users and concurrent execution of multiple processes, the various processes must be protected from each other's activities. Protection refers to a mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer system.

(ii) In order to ensure the proper execution of the operating system, we must be able to distinguish between the execution of operating system code and user-defined code. The approach taken by most computer systems is to provide hardware support that allows us to differentiate among various modes of execution.

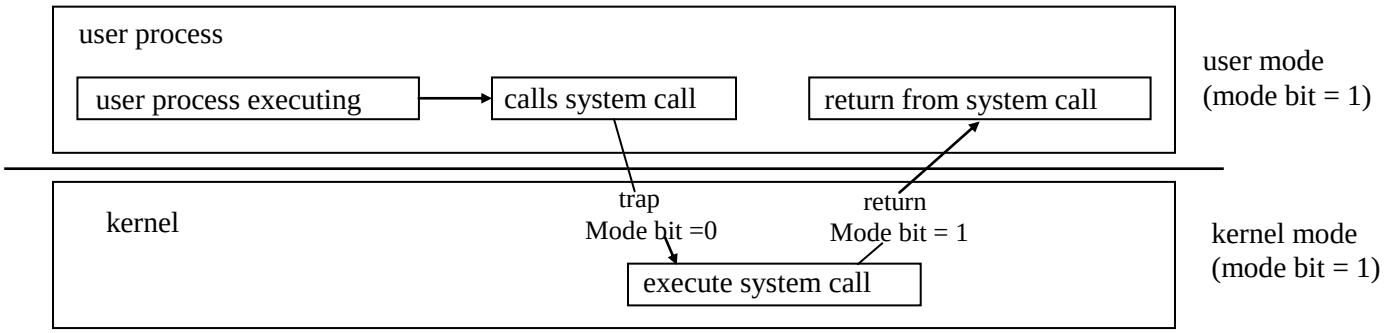


Figure: Transition from user to kernel mode.

At the very least, we need two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user. When the computer system is executing on behalf of a user application, the system is in user mode. However, when a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request. This is shown in Figure. As we shall see, this architectural enhancement is useful for many other aspects of system operation as well.

At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the state of the mode bit to 0). Thus, whenever the operating system gains control of the computer, it is in kernel mode. The system always switches to user mode (by setting the mode bit to 1) before passing control to a user program.

The dual mode of operation provides us with the means for protecting the operating system from errant users and errant users from one another. We accomplish this protection by designating some of the machine instructions that may cause harm as privileged instructions. The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system.

The instruction to switch to kernel mode is an example of a privileged instruction. Some other examples include I/O control, timer management, and interrupt management.

01. (b)

Sol:

(i)

```
find_minimum (int A[ ])
{
    int min, i;
    min = A[0] ;  // considering first element as minimum
   for (i = 1; i < 15; i++) // to check for all other elements
    {
       if (A[i] < min) // if root element is less than min
       {
            min = A[i];  // then update min by this element
       }
     }
     return min;
}
```

(ii)

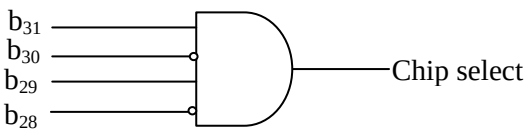
Structures	Union
• A Structure is a collection of related elements, possibly of different types, having a single name.	• A union is a construct that allows a portion of memory to be used by different types of data
• Memory is allocated for all the members	• Memory allocated only for the highest (max) member
• No sharing of data	• Sharing of data
• Any time all values are available	• Only the latest values can be retrieved

01. (c)

Sol: Address bus: $b_{31}, b_{30}, \ldots\ldots b_0$

chip select lines: $b_{31}, b_{30}, b_{22}, b_{29}$ (4-lines)

lines for addressing: $b_{27}, b_{28}, \ldots.. b_0$



Hence in address of 32-bits starting 4 bits (from left) should be 1010 always and remaining 28-bits can be any combination between 28 zeros to 28- ones.

Address range:

1010 0000 …….0000

1010 1111 …….1111

In hexadecimal range will be: A000 0000 to AFFF FFFF

01. (d)

Sol:

(i) Given $\overline{D} = \left(2y^2 + z\right)\hat{a}_x + 4xy\hat{a}_y + x\hat{a}_z C/m^2$

(A) Volume charge density $\rho_v = \nabla.D$

$$= \frac{\partial D_x}{\partial x} + \frac{\partial D_y}{\partial y} + \frac{\partial D_z}{\partial z}$$

$$= 0 \quad + 4x \quad + 0$$

$$\rho_v = 4x\ C/m^3$$

(B) Flux through the cube

$$Q = \int_x \int_y \int_z \rho_v dV$$

$$= \int_0^1 \int_0^1 \int_0^1 4x dx dy dz$$

$$= 4(1)(1)(1/2)$$

$$= 2C$$

$$Q = 2C.$$

(C) Total charge enclosed by the cube = 2C.

(ii) Given $\overline{E} = 20\cos(\omega t - 50x)\,\hat{a}_y\ V/m$

In free space

$\varepsilon = \varepsilon_o$

$\mu = \mu_o$

(A) $J_d = \frac{\partial D}{\partial t} = \frac{\partial}{\partial t}(\varepsilon_o E) = -20\omega\varepsilon_o \sin(\omega t - 50x)\hat{a}_y A/m^2$

(B) From the maxwell's equation,

$\nabla \times H = J_d$

$$\frac{E}{H} = \eta$$

But $\eta = \frac{\beta}{\omega\varepsilon}$

$$\therefore H = \frac{E}{\beta}\omega\varepsilon$$

$$= \left[\frac{20\cos(\omega t - 50x)a_y}{50}\right]\omega\varepsilon_o \qquad \begin{bmatrix} P = E \times H \\ x = y \times z \end{bmatrix}$$

$H = 0.4\omega\varepsilon_o\cos(\omega t - 50x)a_z\ A/m$

(C) $\eta = \frac{\beta}{\omega\varepsilon_o}$

In free space $\eta = 377\Omega$

Given $\beta = 50$

$$\therefore \omega = \frac{\beta}{\eta\varepsilon_o} = \frac{50}{120\pi \times \frac{10^{-9}}{36\pi}} = 1.5 \times 10^{10} rad/s$$

01. (e)

Sol: $f_{c_{21}} = \frac{c}{2}\sqrt{\left(\frac{2}{a}\right)^2 + \left(\frac{1}{b}\right)^2}$

$$= \frac{3\times10^{10}}{2}\sqrt{\left(\frac{2}{6}\right)^2 + \left(\frac{1}{2.5}\right)^2}$$

$f_{c_{21}} = 7.8\text{GHz}.$

$\because$ Operating frequency f = 2 GHz is less than $f_{c_{21}}$.So TM_{21} mode becomes evanescent mode.

In evanescent mode propagation constant 'γ_{21}' becomes a real quantity.

$$\gamma_{21} = \sqrt{\left(\frac{2\pi}{a}\right)^2 + \left(\frac{\pi}{b}\right)^2 - \omega^2\mu\varepsilon}$$

$$\gamma_{21} = \sqrt{\left(\frac{2\pi}{6\times10^{-2}}\right)^2 + \left(\frac{\pi}{2.5\times10^{-2}}\right)^2 - \left(2\pi\times2\times10^9\right)^2\times4\pi\times10^{-7}\times\frac{1}{36\pi}\times10^{-9}}$$

$= \sqrt{25003} = 158.12$

$\gamma_{21} = 158.12 + j0$

$\therefore \alpha_{21} = 158.12\,\text{Np/m}$

Let the amplitude of the E-field at z = d be 20% of that at z = 0 then

$E_z(d) = 0.2 \times 1500 = 300\text{V/m}$

As $E_z(d) = 1500\,e^{-\alpha_{21}d}$ and $E_z(d) = 300\text{V/m}$

(i.e) $300 = 1500\ e^{-\alpha_{21}d}$

$\ell n(0.2) = -\alpha_{21}d$

$\therefore d = -\frac{\ell n(0.2)}{\alpha_{21}} = 0.010178$

= 10.178 mm (or) 1.017 cm.

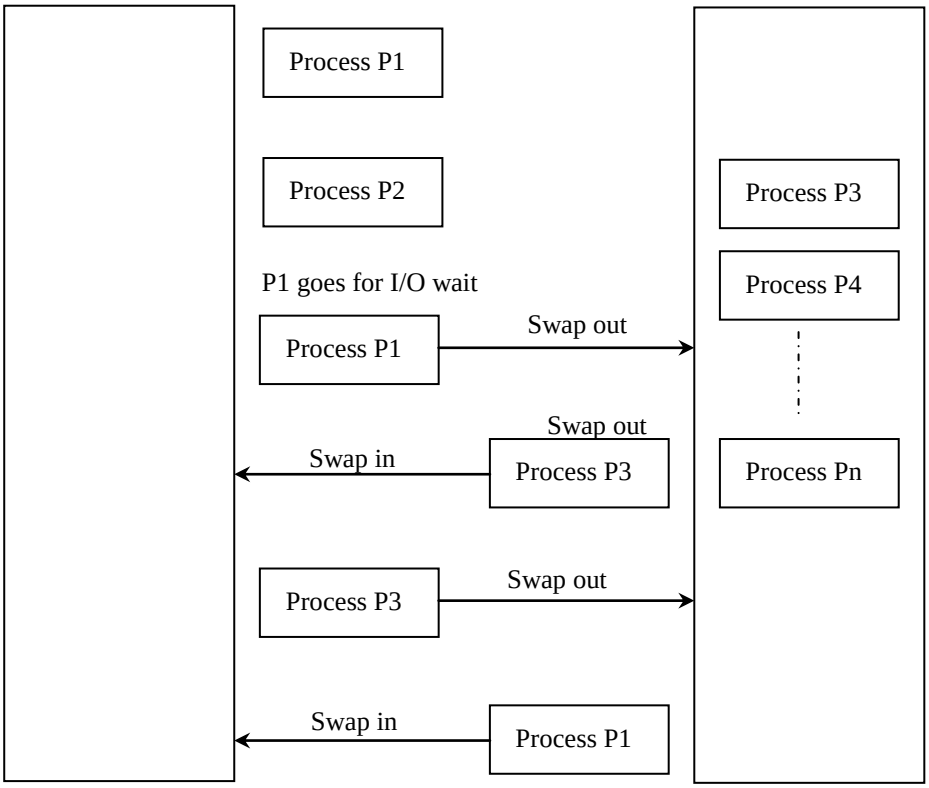
02. (a)

Sol:

(i) Swapping:

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048 KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to from memory will take

2048 KB /1024 KB per second

= 2 seconds

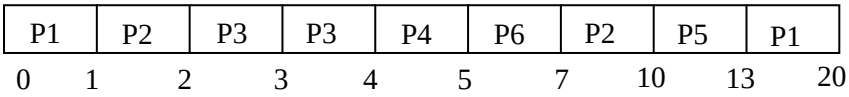
= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

(ii) Preemptive Shortest Remaining Time First (SRTF) policy for process execution provides minimum average waiting time for the process execution. Because the algorithm schedules the smallest available process (least burst time) always to run on CPU; even on the cause of preemption also. Hence any other process will have to wait for least period of time and then it decreases the average waiting time of all processes.

In this Example, there are five jobs P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
1	0	8	20	20	12	0
2	1	4	10	9	5	1
3	2	2	4	2	0	2
4	3	1	5	2	1	4
5	4	3	13	9	6	10
6	5	2	7	2	0	5



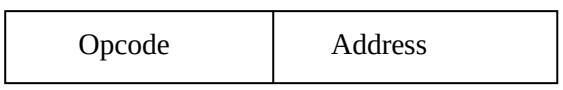
(iii) The Guest account from Windows is a standard, local user account, with very limited permissions. The Guest account has the following restrictions:

- It does not have a password, and you cannot set one for it
- You cannot use it to install programs, universal apps or hardware devices
- It can use only the applications that were already installed on the PC when the Guest account was enabled
- It cannot change its account type, name or picture
- It cannot change the settings of other user accounts
- It cannot access the libraries and user folders of other user accounts
- It can create files only on the desktop and in its user folders - it cannot create folders and files anywhere else on your PC
- The Guest user account can be enabled or disabled only by an administrator

(iv) It is one of the error-detection which OS performs in the system.
Assume a file transfer is going on from a pendrive to local system's harddrive. And during that before the transfer completes user accidently/intentionally plugs out the pendrive. In this case OS detects that the source of file is not available (connected); hence OS services the error with an error message (sometimes with a beep) stating "source of the file is not available" prompted to user.
Based on user's intentions OS performs further operation: like the file is to be transferred completely with plugged-in drive or to be discarded.

02. (b)
Sol:

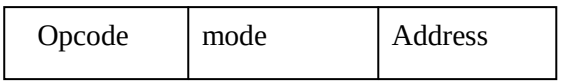
(i) Consider following 1-address instruction



Assume opcode specifies 'load to AC' and address field contains a value 3.
Now the problem is how this value 3 is interpreted by CPU to get operand.

- 3 can be taken as operand value. Which means instruction will be like
  AC ← #3

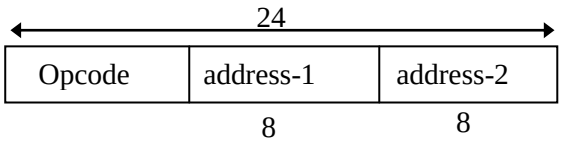
- 3 can be taken as register reference, for which instruction will be like
  AC ← R3
- 3 can be taken as memory address, for that instruction will be like:
  AC ← M [3]
- Even there can be other way to interpret address field for obtaining operand. To distinguish between all these addressing mode is used; which specifies how and from where operand can be obtained using the address field value. Hence after using mode instruction will be:



(ii) If a CPU supports 2 or more type of instructions then we always start analysis from that instruction which contains maximum addresses (least opcode bits).
Assume a system which supports 2-address and 1- address instructions. starting with 2-address instruction, with assumption of

Instruction size = 24-bits
Address size = 8-bits

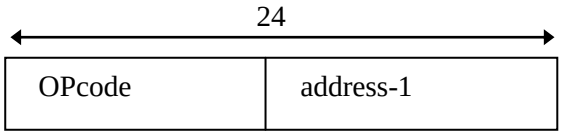
2-address instruction format:



opcode size = 24 – (8 + 8) = 8-bits
Maximum opcode = 2^8 = 256 (Max. Possible 2-address instructions)
If assuming 200 opcodes are used for 2-address instructions; which means there are 200 2-address instructions are supported by system, then

Max. opcodes = 256
Used opcodes = 200
Unused opcodes = 56

1- address instruction format:



opcode = 24 – 8 = 16-bits
but here all 16-bit combinations cannot be used as opcodes in 1-address instructions because there are some opcodes in 2-address instructions which have been already used and those can have same starting 8-bits in opcode as this 16-bit opcode combinations.
Hence, we need to skip those opcode combinations from 1-address instruction opcodes.
In 1-address instruction opcodes are of 16-bits. In those 16-bits starting 8-bits opcodes (remaining) one = 56 from 2-address instruction.

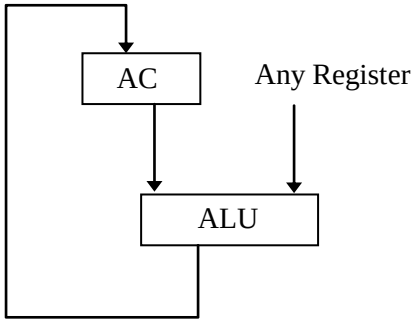
Hence, max. - 1 address instruction = 56 * 2^8

= 14336

2-address instruction	Remaining opcodes combination	1-address instruction
256	256 – 256 = 0	$0 * 2^8 = 0 \Rightarrow$ only 2-add instruction
255	256 – 255 = 1	$1 * 2^8 = 256$
254	256 – 254 = 2	$2 * 2^8 = 512$
.		
.		
.		
.		
0	256 – 0 = 256	$256 * 2^8 = 2^{16} \Rightarrow$ only 1-add instruction

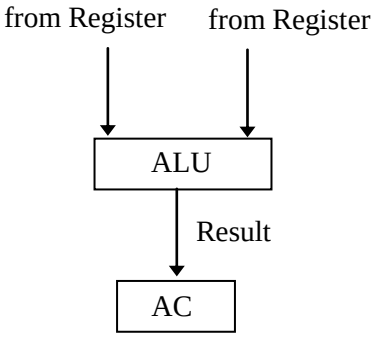
(iii) AC-based architecture:

In accumulator base architecture, ALU takes one of the input from accumulator only. It is shown below.



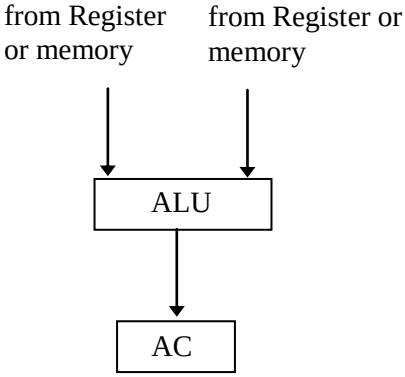
Register based architecture:

In this both the inputs of ALU are taken from registers (general purpose).



Complex memory-based Architecture:

In this architecture both or one input can be taken from memory also.



02. (c)

Sol: For TE_{10} mode:-

$$P_{ang_{TE_{10}}} = \frac{ab\left|E_{y0}\right|^2}{4\eta_{TE_{10}}} \text{----(1)}$$

For TE_{11} mode:-

$$P_{avg_{TE_{11}}} = \frac{ab\left|E_{x_0}^2 + E_{y_0}^2\right|}{8\eta_{TE_{11}}}$$

In TE_{mn} mode (both m and n $\neq 0$) $\Rightarrow \frac{E_{y0}}{E_{x0}} = \frac{mb}{na}$

$\therefore$ TE_{11} mode, $\frac{E_{y0}}{E_{x0}} = \frac{b}{a}$ & $\frac{E_{x0}}{E_{y0}} = \frac{a}{b}$

$$P_{avg_{TE_{11}}} = \frac{ab\left|E_{y0}\right|^2\left(1+\frac{E_{x0}^2}{E_{y0}^2}\right)}{8\eta_{TE_{11}}}$$

$$P_{avg_{TE_{11}}} = \frac{ab\left|E_{y0}\right|^2}{8\eta_{TE_{11}}}\left(\frac{b^2+a^2}{b^2}\right) \text{----(2)}$$

$$\therefore \frac{(1)}{(2)} \Rightarrow \frac{P_{avg_{TE_{10}}}}{P_{avg_{TE_{11}}}} = \frac{\frac{ab\left|E_{y0}\right|^2}{4\eta_{TE_{10}}}}{\frac{ab\left|E_{y0}\right|^2}{8\eta_{TE_{11}}}\left(\frac{b^2+a^2}{b^2}\right)}$$

$$\frac{P_{avg_{TE_{10}}}}{P_{avg_{TE_{10}}}} = \frac{2\eta_{TE_{11}}b^2}{\eta_{TE_{10}}\left(b^2+a^2\right)} \text{---(3)}$$

$$\therefore \eta_{TE_{10}} = \frac{\eta_o}{\sqrt{1-\left(\frac{f_{c_{10}}}{f}\right)^2}} \qquad \text{where } f_{c_{10}} = \frac{c}{2a} = \frac{3\times10^{10}}{2\times10} = 1.5\text{GHz}$$

$$\therefore \eta_{TE_{10}} = \frac{120\pi}{\sqrt{1-\left(\frac{1.5\times10^9}{6\times10^9}\right)^2}}$$

$$\therefore \eta_{TE_{10}} = 389.35\Omega$$

$$\eta_{TE_{11}} = \frac{\eta_o}{\sqrt{1-\left(\frac{f_{c_{11}}}{f}\right)^2}} \qquad \text{where } f_{c_{11}} = \frac{c}{2}\sqrt{\left(\frac{1}{a}\right)^2+\left(\frac{1}{b}\right)^2}$$

$$= \frac{3\times10^{10}}{2}\sqrt{\left(\frac{1}{10}\right)^2+\left(\frac{1}{5}\right)^2} = 3.35 \text{ GHz.}$$

$$\therefore \eta_{TE_{11}} = \frac{120\pi}{\sqrt{1-\left(\frac{3.35\times10^9}{6\times10^9}\right)^2}}$$

$$= 454.41\ \Omega$$

$\therefore$ substitute all values in Equation (3)

$$\frac{P_{avTE_{10}}}{P_{avTE_{11}}} = \frac{2\eta_{TE_{11}}b^2}{\eta_{TE_{10}}\left(a^2+b^2\right)} = \frac{2\times454.41\times(5)^2}{389.35\times\left(10^2+5^2\right)} = 0.466$$

$$\therefore \frac{P_{av_{TE_{10}}}}{P_{avTE_{11}}} = 0.466$$

03. (a)

Sol:

(i) Given $\bar{E}_x = 5\sin\beta z\sin\omega t\hat{a}_x$ -----(1)

From (1), it is clear that E_x is a function of z and t.

$$\therefore \frac{\partial}{\partial x} = \frac{\partial}{\partial y} = 0$$

From Maxwell's equation $\nabla\times\bar{E} = \frac{-\partial\bar{B}}{\partial t}$

$$\Rightarrow \frac{\partial E_x}{\partial z} = -j\omega\mu H_y$$

$$\Rightarrow \frac{5\beta\cos\beta z\sin\omega t}{-j\omega\mu} = H_y$$

$$\therefore\ H_y = j\frac{\beta}{\omega\mu}5\cos\beta z\sin\omega t$$

$$H_y = \frac{j5\cos\beta z\sin\omega t}{\eta} \qquad \left\{\frac{\beta}{\omega\mu} = \frac{\omega\sqrt{\mu\varepsilon}}{\omega\mu} = \sqrt{\frac{\varepsilon}{\mu}} = \frac{1}{\eta}\right\}$$

Where $\eta = \frac{\eta_0}{\sqrt{\varepsilon_r}} = \frac{\eta_0}{\sqrt{4}} = \frac{\eta_0}{2}$

At the planar interface, z = 0,

$$\therefore H_y = \frac{j10\sin\omega t}{\eta_0}$$

$$\Rightarrow H_y = \frac{j10\sin\omega t}{120\pi} = j0.02652\sin\omega t\,A/m$$

$\therefore$ At the interface, $H_y = j0.02652\sin\omega t\,A/m$

Now, $\bar{J}_s = H_y\,\hat{a}_y\times(\hat{a}_z)$

$$\bar{J}_s = H_y\,(\hat{a}_x)$$

$$\bar{J}_s = j0.02652\sin\omega t\hat{a}_x A/m$$

(ii) Given total current = $6\sin 10^6\pi t$

$\omega = 10^6\pi \Rightarrow f = 0.5\times10^6$Hz.

Skin depth $(\delta) = \dfrac{1}{\sqrt{\pi f\mu\sigma}} = \dfrac{1}{\sqrt{\pi\times0.5\times10^6\times4\pi\times10^{-7}\times3.5\times10^7}}$

$\therefore\ \delta = 0.1203$mm

Effective resistance $R_{ac} = \dfrac{\ell}{\sigma\delta w}$

Since δ is very small, $w = 2\pi\rho_{outer}$

$\rho_{outer} = 12$ mm

$\therefore R_{ac} = \dfrac{\ell}{\sigma.2\pi\rho_{outer}\delta} = \dfrac{40}{3.5\times10^7\times2\pi\times12\times10^{-3}\times0.1203\times10^{-3}}$

$R_{ac} = 0.126\Omega$

03. (b)

Sol:

(i) Since Z_o is real and $\alpha \neq 0$, this is a distortion less line.

$Z_o = \sqrt{\dfrac{R}{G}}$ _____(1)

(or)

$\dfrac{L}{R} = \dfrac{C}{G}$ _____ℓ)

$\alpha = \sqrt{RG}$ _____β)

$\beta = \omega L\sqrt{\dfrac{G}{R}} == \dfrac{\omega L}{Z_0}$ __(4)

(1)×(3) $\rightarrow R = \alpha Z_o = 0.04\times80 = 3.2\Omega/m$

(3) ÷ (1) $\rightarrow G = \dfrac{\alpha}{Z_o} = \dfrac{0.04}{80} = 5\times10^{-4}$S/m

$L = \dfrac{\beta Z_o}{\omega} = \dfrac{1.5\times80}{2\pi\times5\times10^8} = 38.2$nH/m

$C = \dfrac{LG}{R} = \dfrac{38.2\times10^{-9}\times5\times10^{-4}}{3.2} = 5.97$pF/m.

(ii) Given lossless transmission line with

$Z_o = 75\Omega$

$Z_L = 120\Omega$

$\ell = 1.25\lambda$

$\beta\ell = \dfrac{2\pi}{\lambda}(1.25\lambda) = \dfrac{\pi}{2} + 360^o$

$\therefore \tan\beta\ell \rightarrow \infty$

(A) Input impedance

$Z_{in} = Z_o\left[\dfrac{Z_L + jZ_o\tan\beta\ell}{Z_o + jZ_L\tan\beta\ell}\right]$(since lossless line)

As $\tan\beta\ell \rightarrow \infty$

$Z_{in} = \dfrac{Z_o^2}{Z_L} = 46.875\Omega$

(B) magnitude of voltage at z = 0

$$V_o = V(z=0) = \frac{Z_{in}}{Z_{in}+Z_g}V_g$$

$$= \left(\frac{46.875}{46.875+50}\right)100$$

$$= 48.39\text{V}.$$

For lossless line,

Magnitude of load voltage $= |V_L| = |V(z=0)| = 48.39\text{V}.$

03. (c)

Sol:

(i) Void:

- A void type is used when C needs to define a lack of data.
- If a function returns no value, the return type must be declared as void.
- If a function has no parameters, the parameter list must be declared void.
- We highly recommend that every function have a return statement. A return statement is required if the return type is anything other than void.

(ii) The ternary operator is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison. If it helps you can think of the operator as shortened way of writing an if-else statement.

Example:

```
int a = 10, b = 20, c;
if (a < b) {
   c = a;
}
else {
   c = b;
}
printf("%d", c);
```

(iii)

Basis For Comparison	**&**	**&&**
Operator	It is a "Bitwise Operator".	It is a "Logical Operator".
Evaluation	It evaluates both left and right side of the expression.	It only evaluates the left side of the expression.
Operates on	It operates on "Boolean data type" as well as operates on "bits".	It operates only on "Boolean data type".
Use	Use to check logical condition and also used to mask off certain bits such as parity bits.	Used only to check logical condition.

(iv) An array is a collection of fixed number of values of a single type. That is, you need to declare the size of an array before you can use it.

Sometimes, the size of array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.

There are 4 library functions defined under <stdlib.h> makes dynamic memory allocation in C programming. They are malloc(), calloc(), realloc() and free().

malloc( ):

The name "malloc" stands for memory allocation.

The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of type void which can be casted into pointer of any form.

Syntax of malloc()

ptr = (cast-type*) malloc(byte-size)

Example:

ptr = (int*) malloc(100 * sizeof(int));

calloc()

The name "calloc" stands for contiguous allocation.

The malloc() function allocates a single block of memory. Whereas, calloc() allocates multiple blocks of memory and initializes them to zero.

Syntax of calloc()

ptr = (cast-type*)calloc(n, element-size);

Example:

ptr = (float*) calloc(25, sizeof(float));

This statement allocates contiguous space in memory for 25 elements each with the size of float.

realloc()

If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using realloc() function

Syntax of realloc()

ptr = realloc(ptr, x);

Here, ptr is reallocated with new size x.

free( )

Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space.

Syntax of free()

free(ptr);

This statement frees the space allocated in the memory pointed by ptr.

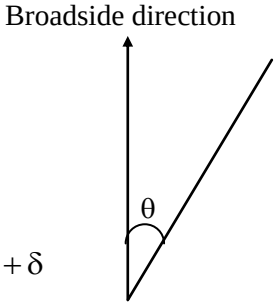
(v) Dangling Pointer in C

1. Dangling pointers arise when an object is deleted or de-allocated, without modifying the value of the pointer, so that the pointer still points to the memory location of the de-allocated memory.
2. In short pointer pointing to non-existing memory location is called dangling pointer.

04. (a)

Sol:

(i) Let us define the angle from the broadside direction instead of from the array axis. We then get



$$\psi = \frac{2\pi}{\lambda} d \sin\theta + \delta$$

Now $\theta_{max} = 30° = \pi/6$, and for maximum radiation $\psi = 0$. We therefore get

$$\delta = -\frac{2\pi}{\lambda} d \sin(\pi/6) = -\frac{\pi d}{\lambda} \quad (1)$$

The nulls of the array are given by $N\psi/2 = \pm m\pi$

The first nulls therefore corresponds to $N\psi/2 = 4\psi/2 = \pm\pi$

$\Rightarrow \psi = \pm \pi/2$

The directions of the first nulls give

$$\frac{2\pi}{\lambda} d \sin\theta_+ - \frac{\pi d}{\lambda} = \pi/2 \quad (2)$$

and $$\frac{2\pi d}{\lambda} \sin\theta_- - \frac{\pi d}{\lambda} = -\pi/2 \quad (3)$$

Adding Equation (2) and (3) we get

$\sin\theta_+ + \sin\theta_- = 1$ (4)

It is given that BWFN $= \theta_+ - \theta_- = \pi/2$

$\Rightarrow \theta_+ = \pi/2 + \theta_-$

$\Rightarrow \sin\theta_+ = \cos\theta_-$

Substituting in Equation (4) we get

$\cos\theta_- + \sin\theta_- = 1$

$\Rightarrow \sqrt{1 - \sin^2\theta_-} + \sin\theta_- = 1$

$\Rightarrow \sin\theta_- = 0$ i.e., $\theta_- = 0$

Therefore $\theta_+ = \pi/2$

The two first nulls are in the directions $\theta = 0$ and $\pi/2$

Substituting $\theta_- = 0$ in (3) we get $d = \lambda/2$, and

$$\psi = \frac{2\pi}{\lambda} . \frac{\lambda}{2} . \sin\theta - \frac{\pi\lambda}{2\lambda} = \pi \sin\theta - \pi/2$$

For directions of other nulls, putting

$$\frac{N\psi}{2} = \frac{4\psi}{2} = 2\psi = \pm m\pi \text{ where } m = 2, 3$$

$\Rightarrow 2(\pi \sin\theta_n - \pi/2) = \pm m\pi$, $m = 2, 3$

$$\Rightarrow \sin\theta_n = \frac{1 \pm m}{2}, m = 2,3$$

$\theta_n = \sin^{-1}(-1/2) = -\pi/6$, $\sin^{-1}(-1) = -\pi/2$

The nulls of the array are located at

$\theta = -\pi/2, -\pi/6, 0, \pi/2$

(ii) $P_{rad} = 100kW$.
$E = 12m\ V/m$
$r = 20\times10^3 m$.
where

$$|E|^2 = \frac{\eta G_d P_{rad}}{2\pi r^2}$$

$$\Rightarrow G_d = \frac{2\pi r^2|E|^2}{\eta P_{rad}} = \frac{2\pi\times(20\times10^3)^2\times(12\times10^{-3})^2}{120\pi\times100\times10^3} = 9.6\times10^{-3}$$

Directivity in dB = $10\log G_d = 10\log(9.6\times10^{-3}) = -20.18$ dB

04. (b)

Sol: Let the intrinsic impedance of quarter wave length waveguide is Z_2 then

$$Z_2 = \sqrt{Z_1 Z_3} = \sqrt{(499.5)(258.36)} = 359.235\Omega$$

$$Z_2 = \frac{\frac{\eta_o}{\sqrt{\epsilon_r}}}{\sqrt{1-\left(\frac{\frac{f_{c_{10}}}{\sqrt{\varepsilon_r}}}{f}\right)^2}} \qquad \left\{\begin{array}{l}\text{where } f_{c_{10}} = \frac{c}{2a}\\ \text{let}\\ f^1_{c_{10}} = \frac{f_{c_{10}}}{\sqrt{\varepsilon_r}} = \frac{c}{2a\sqrt{\varepsilon_r}}\end{array}\right\}$$

$$\Rightarrow Z_2^2\left(1-\left(\frac{f_{c_{10}}}{f}\right)^2.\frac{1}{\varepsilon_r}\right) = \frac{\eta_o^2}{\varepsilon_r} \Rightarrow Z_2^2\left(\frac{f_{c_{10}}}{f}\right)^2.\frac{1}{\varepsilon_r} + \frac{\eta_o^2}{\varepsilon_r} = Z_2^2$$

$$\varepsilon_r = \left(\frac{f_{c_{10}}}{f}\right)^2 + \frac{\eta_o^2}{Z_2^2} \qquad \text{where } f_{c_{10}} = \frac{c}{2a} = \frac{3\times10^{10}}{2\times2.286} = 6.56GHz$$

Given f = 10GHz.

$$\therefore \varepsilon_r = \left(\frac{6.56\times10^9}{10\times10^9}\right)^2 + \frac{(120\pi)^2}{(359.235)^2} \qquad \left\{\begin{array}{l}\because \eta_o = 120\pi\Omega\\ Z_2 = 359.235\end{array}\right\}$$

$\varepsilon_r = 1.531 \rightarrow$ dielectric constant.

$$\therefore \ell = \frac{\lambda_g}{4} = \frac{\lambda}{4\sqrt{1-\left(\frac{f_{c_{10}}}{f}\right)^2.\frac{1}{\varepsilon_r}}}$$

where $\lambda = \frac{c}{f\sqrt{\epsilon_r}} = \frac{3\times10^{10}}{10^{10}\sqrt{1.531}} = 2.4245cm$

$$\therefore \ell = \frac{2.4245}{4\sqrt{1-\left(\frac{6.56}{10}\right)^2.\frac{1}{1.531}}} = 0.7148\,cm$$

$\therefore \ell = 0.7148$ cm

$\therefore \varepsilon_r = 1.531$ & $\ell = 0.7148\,cm$

04. (c)

Sol:

(i) Stack:

Stack is a LIFO (Last-In, First-Out) list, a list-like structure in which elements may be inserted or removed from only one end (last-in, first-out). Stacks are less flexible than lists, but are easier to implement, and more efficient (for those operations they can do). Given a stack, the accessible element of the stack is called the top element. Elements are not said to be inserted; they are pushed onto the stack. When an element (the last one) is removed, an element is said to be popped from the stack.

Queue:

Queue is a FIFO (First-In, First-Out) list, a list-like structure that provides restricted access to its elements: elements may only be inserted at the back and removed from the front. Similarly to stacks, queues are less flexible than lists.

Enqueue: insert elements into queue at the back.

Dequeue: remove elements from the front.

(ii) The C programming language provides many standard library functions for file input and output. These functions make up the bulk of the C standard library header <stdio.h>.

The first thing you will notice is the first line of the file, the #include "stdio.h" line. This is very much like the #define the preprocessor, except that instead of a simple substitution, an entire file is read in at this point.

The system will find the file named "stdio.h" and read its entire contents in, replacing this statement.

Obviously then, the file named "stdio.h" must contain valid C source statements that can be compiled as part of a program.

This particular file is composed of several standard #defines to define some of the standard I/O operations like printf( ) and scanf( ) etc.

By including this stdio.h header file, definitions of scanf( ) and printf( ) functions can be included in our program. Hence to use basic input and output functions we need to include this header file.

(iii) A function declaration in C-language has following syntax:

Return_type function_name (inputtype 1, inputtype 2, …..)

The return type is always one, hence only one value any function can return.

The return type is nothing but output type of the function. For any function evaluation only one output can be possible, hence in C-language also a function returns only 1 value..

(iv) In C-language semicolon is used as termination symbol of the statements. But semicolon is not used at the end of every statement.

When next statement is not related (under) the current statement then at the end of current statement semicolon is placed. Like at the end of printf() statement semicolon is placed always because; next statement will not be under printf() statement.

When next statement is related (or under) the current statement then at the end of such statements semicolon is not placed; to show that the current statement has any upcoming blocks. Like at the end of if, else, for, while statements semicolon is not placed. Because all these statements have their blocks immediately after them.

05. (a)

Sol:

(i) Token is an elementary item in the program that may be a keyword, operator, identifier, punctuation symbol, or a constant.

For example consider the statement: area = 22/7* radius * radius;

Here, area, = , 22, / 7, m radius,; are individual tokens.

Tokens are of 6 types.

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special symbols
6. Operators

Keywords are pre-defined words in a C compiler. Each keyword is meant to perform a specific function in a C program. Since keywords are referred names for compiler, they can't be used as variable name.

Eg: int, void, long.

Identifiers are the name given to each program element in program. Names given to identify Variables, functions and arrays are examples for identifiers.

For example int x; here int is the keyword and x is the identifier.

Constants are variables whose values cannot be modified by the program once they are defined. Constants have a fixed value and they are also known as literals. Keyword const is used before data type.

Syntax: const data_type variable_name;

Eg: const int a ;

Strings are array of characters terminated by a null character. For example a string 'toy' has got 4 characters in it 't' , 'o' , 'y' & the null character.

Symbols other than the Alphabets and Digits and white-spaces are called **Special symbols**.

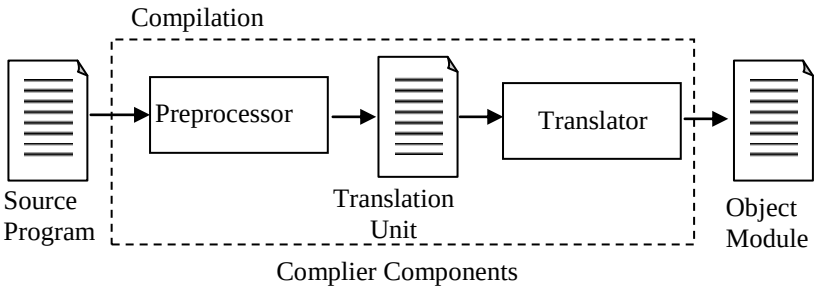
Eg: * , & ,^ ,@, etc.

An **operator** is a symbol which operates on a value or a variable. For example: + is an operator to perform addition. C programming has wide range of operators to perform various operations.

(ii) Preprocessor:

The code in a source file stored on the disk must be translated into machine language. This is the job of the compiler. The C compiler is actually two separate programs: the **preprocessor** and the **translator.**

The preprocessor reads the source code and prepares it for the translator. While preparing the code, it scans for special instructions known as **preprocessor commands.** These commands tell the preprocessor to look for special code libraries, make substitutions in the code, and in other ways prepare the code for translation into machine language. The result of preprocessing is called the **translation unit.**



All **preprocessor commands** start with a pound sign (#).

#define name body

Example:

#define SIZE 9

05. (b)
Sol:
(i)

Basis For Comparison	**Program**	**Process**
Basic	Program is a set of instructions.	When a program is executed, it is known as process.
Nature	Passive	Active
Lifespan	Longer	Limited
Required resources	Program is stored on disk in some file and does not require any other resources.	Process holds resources such as CPU, memory address, disk, I/O etc.

(ii) When the computer system starts CPU executes its first few instructions from ROM only because RAM is a volatile memory and RAM's content used to flushed out while system turn off.

CPU performs POST(Power On Self-Test) to check all hardware's and their working. After that CPU performs booting process which helps to bring OS programs from secondary memory to RAM. All these works are performed while executing the programs from ROM. That is why ROM is very important part of the system. And CPU can execute all these instructions from ROM because ROM is a non-volatile memory.

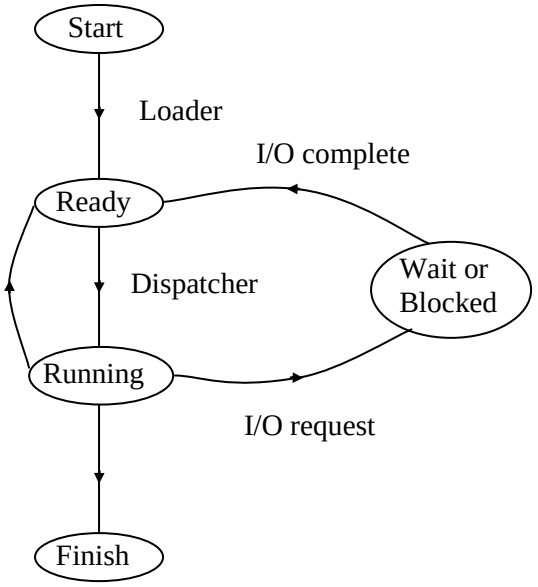
There are two main reasons that read-only memory is used for certain functions within the PC:

Permanence: The values stored in ROM are always there, whether the power is on or not. A ROM can be removed from the PC, stored for an indefinite period of time, and then replaced, and the data it contains will still be there. For this reason, it is called non-volatile storage. A hard disk is also non-volatile, for the same reason, but regular RAM is not.

Security: The fact that ROM cannot easily be modified provides a measure of security against accidental (or malicious) changes to its contents. You are not going to find viruses infecting true ROMs, for example; it's just not possible. (It's technically possible with erasable EPROMs, though in practice never seen.)

Read-only memory is most commonly used to store system-level programs that we want to have available to the PC at all times. The most common example is the system BIOS program, which is stored in a ROM called (amazingly enough) the system BIOS ROM. Having this in a permanent ROM means it is available when the power is turned on so that the PC can use it to boot up the system. Remember that when you first turn on the PC the system memory is empty, so there has to be something for the PC to use when it starts up.

(iii) Process state transition (life cycle) for multitasking operating systems:



Start state:

- When a process is requested to execute, then it is in start state
- Process is in secondary memory (like hard disk)

Ready state:

- When requested process is loaded by loader into main memory, then it is in ready state
- Ready for execution, waiting for its turn
- If more than one processes are waiting for execution in ready state then process (short term) schedule do scheduling among them.

Running state:

- Selected process by process scheduler initialized by dispatcher on CPU for execution
- Process is executing on CPU it is in running state
- CPU fetches instructions of process from main memory one by one and execute it.

Wait or Blocked state:

- When a running process requested for I/O activity (Input/output from secondary devices) then it moved to Blocked state
- Once the process completed its I/O activity it is shifted to ready state

Finish state:

- When a running process completed its execution it is in finish state
- Process execute its last exit instruction and request O.S. to remove it from main memory

05. (c)

Sol:

(i) Locality of reference: If CPU refers the main memory at a specific address then the same address or nearby address will be referred soon, this phenomena is known as locality of reference.

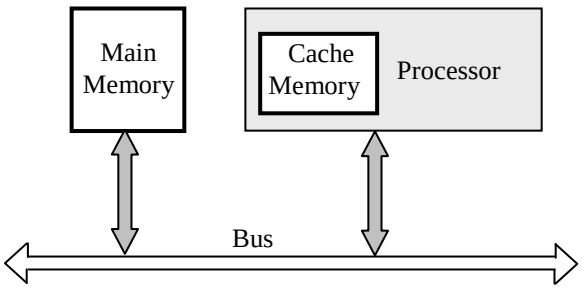
Cache implementation is done based on this concept of locality of reference concept only. During a cache miss not only the requested word/byte but a block is copied from main memory to cache. *Reason behind this is:* if there will be a demand of other nearby content then there will be a cache hit for them and there will be performance improvement in the system.

Hence cache implementation is done based on locality of reference concept only to improve system's performance.

(ii) Cache Memory:

The purpose of cache memory as follows.

- When the execution of an instruction calls for data located in the main memory, the data are fetched and copy is placed into cache.



- If same instruction or data item is needed a second time, it is read directly from the cache. By using cache the speed of operation will be increased and execution time will be reduced. Cache memory is also called *high speed buffer memory*.

(iii) The types of accesses of cache memory are:

(A) Simultaneous Access Memory Organization:

- In this organization, CPU is directly connected to all the levels of Memory.
- CPU accesses the data from all levels of Memory simultaneously.
- For any *"miss"* encountered in L1 memory, CPU can directly access data from higher memory levels (i.e. L2, L3, …..Ln).
- If H1 and H2 are the Hit Ratios and T1 and T2 are the access times of L1 and L2 memory levels respectively then the *Average Memory Access Time* can be calculated as:

$T = (H1 * T1) + ((1– H1) * H2 * T2)$

(B) Hierarchical Access Memory Organization

- In this organization, CPU is always directly connected to L1 i.e. Level-1 Memory only.
- CPU always accesses the data from Level-1 Memory.
- For any ***"miss"*** encountered in L1 memory, CPU cannot directly access data from higher memory levels(i.e. L2, L3, …..Ln). First the desired data will be transferred from higher memory levels to L1 memory. Only then it can be accessed by the CPU.
- If H1 and H2 are the Hit Ratios and T1 and T2 are the access times of L1 and L2 memory levels respectively then the *Average Memory Access Time* can be calculated as:

$T = (H1 * T1) + ((1– H1) * H2* (T1+ T2))$

05. (d)

Sol: $f_{c_{12}} = \frac{c}{2}\sqrt{\left(\frac{1}{a}\right)^2 + \left(\frac{2}{b}\right)^2}$

$$f_{c_{12}}^2 = \left(\frac{c}{2}\right)^2\left[\frac{1}{a^2} + \frac{4}{b^2}\right] \text{----(1)}$$

similarly

$$f_{21}^2 = \left(\frac{c}{2}\right)^2\left[\frac{4}{a^2} + \frac{1}{b^2}\right] \text{----(2)}$$

(1) - (2)

$$f_{c_{12}}^2 - f_{c_{21}}^2 = \left(\frac{c}{2}\right)^2\left[\frac{-3}{a^2}+\frac{3}{b^2}\right]$$

$$f_{c_{12}}^2 - f_{c_{21}}^2 = 3\left(\frac{c}{2}\right)^2\left[\frac{1}{b^2}-\frac{1}{a^2}\right]$$

$$\frac{1}{b^2}-\frac{1}{a^2} = \frac{f_{c_{12}}^2 - f_{c_{21}}^2}{3\left(\frac{c}{2}\right)^2} = \frac{\left(10\times10^9\right)^2 - \left(6\times10^9\right)^2}{3\left(\frac{3\times10^{10}}{2}\right)^2}$$

$$= \frac{64\times10^{18}\times4}{27\times10^{20}} = 9.48\times10^{-2}$$

$$\frac{1}{b^2}-\frac{1}{a^2} = 0.0948 \text{ ---(3)}$$

(1) + (2)

$$f_{c_{12}}^2 + f_{c_{21}}^2 = \left(\frac{c}{2}\right)^2 5\left[\frac{1}{a^2}+\frac{1}{b^2}\right]$$

$$\therefore \frac{1}{a^2}+\frac{1}{b^2} = \frac{f_{c_{12}}^2 + f_{c_{21}}^2}{5\left(\frac{c}{2}\right)^2}$$

$$= \frac{\left(10\times10^9\right)^2 + \left(6\times10^9\right)^2}{5\left(\frac{3\times10^{10}}{2}\right)^2}$$

$$= \frac{136\times10^{18}\times4}{45\times10^{20}} = 12.088\times10^{-2}$$

$$= 0.1208$$

$$\therefore \frac{1}{b^2}+\frac{1}{a^2} = 0.1208 \text{ ----(4)}$$

(3) + (4)

$$\frac{2}{b^2} = 0.2156$$

$$\therefore b = \sqrt{\frac{2}{0.2156}} = 3.045\text{cm}$$

(4) - (3)

$$\frac{2}{a^2} = 0.026$$

$$\therefore a = \sqrt{\frac{2}{0.026}} = 8.77\text{cm}$$

$$\therefore a = 8.77\,\text{cm} \;\&\; b = 3.045\,\text{cm}$$

05. (e)

Sol: The temperature of the auditorium is given by

$T = x^2 + y^2 - z$

It is given that mosquito is located at (1,1,2) .For the mosquito to get warm as soon as possible it should fly in the direction of maximum space rate of increase of T.

i.e., gradient of scalar field T

$$\nabla T = \frac{\partial T}{\partial x}\hat{a}_x + \frac{\partial T}{\partial y}\hat{a}_y + \frac{\partial T}{\partial z}\hat{a}_z$$

$$\nabla T = 2x\hat{a}_x + 2y\hat{a}_y - \hat{a}_z$$

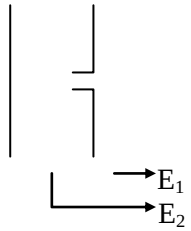
At (1, 1, 2), $\nabla T = (2, 2, -1)$

Hence, mosquito should move in the direction of

$2\hat{a}_x + 2\hat{a}_y - \hat{a}_z$.

05. (f)

Sol:



Without the reflecting rod, the directivity of a half wave dipole is 1.64. After placing the rod, the wave moving in the direction of the arrow consists of two electric field components

$E = E_1 + E_2 \rightarrow (1)$

where E_1 is the field of the radiated wave moving to the right and 'E_2' is the field initially moved to the left and then got reflected by the rod. The two are essentially equal in magnitude but E_2 lags in phase by $2\beta d$ relative to E_1 and also by "π" because the reflection coefficient of the metal rod is –1. Hence, we can write E at any point to the right of antenna as

$E = E_1 + E_1 e^{j\pi} e^{-j2\beta d}$

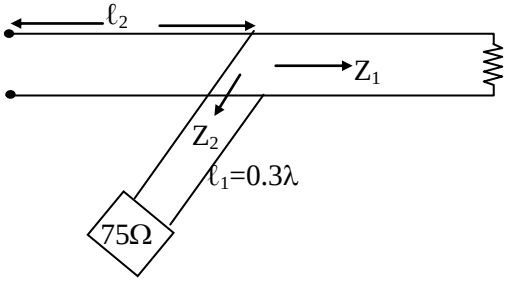
for $d = \frac{\lambda}{4}, 2\beta d = 2\left(\frac{2\pi}{\lambda}\right)\left(\frac{\lambda}{4}\right) = \pi$

$\therefore E = E_1 + E_1 e^{j\pi} e^{-j\pi} = 2E_1$

The directivity is proportional to power (or) $|E|^2$ Hence, 'D' will increase by a factor of 4 (i.e) $D = 1.64 \times 4 = 6.56$

06. (a)

Sol: At the junction we have two impedances Z_1 and Z_2 in parallel. Since the cable has been terminated in its characteristic impedance, Z_1 will be same as the characteristic impedance 50Ω. Z_2 however will be transformed version of 75Ω impedance. Hence, we have



Bus network

$$Z_2 = Z(\ell_1) = Z_0 \frac{75\cos\beta\ell_1 + j50\sin\beta\ell_1}{50\cos\beta\ell_1 + j75\sin\beta\ell_1}$$

and $\beta\ell_1 = 2\pi/\lambda(0.3\lambda) = 0.6\pi = 108°$, giving

$$Z_2 = Z(\ell_1) = 50\frac{75\cos 108° + j50\sin 108°}{50\cos 108° + j75\sin 108°}$$

$$= 35.2008 + j8.621\ \Omega$$

Since, at the junction, the two impedances are connected in parallel, the impedance Z is

$$Z = \frac{Z_1 Z_2}{Z_1 + Z_2} = 20.9549 + j2.9389\,\Omega$$

The impedance at a distance of ℓ_2 from the junction is

$$Z(\ell_2) = Z_0 \frac{Z\cos\beta\ell_2 + j50\sin\beta\ell_2}{50\cos\beta\ell_2 + j\,Z\sin\beta\ell_2}$$

and $\beta\ell_2 = 2\pi/\lambda(0.2\lambda) = 0.4\pi = 72°$, we get

$$Z(\ell_2) = 50\frac{(20.9549 + j2.9389)\cos 72° + j50\sin 72°}{50\cos 72° + j(20.9549 + j2.9389)\sin 72°}$$

$$= 97 + j\ 43.47\Omega$$

The magnitude of the reflection coefficient on the line is

$$|\Gamma| = \left|\frac{Z - Z_0}{Z + Z_0}\right| = \left|\frac{20.9549 + j2.9389 - 50}{20.9549 + j2.9389 + 50}\right| = 0.41$$

$\Rightarrow$ VSWR on the line, $\rho = \dfrac{1+|\Gamma|}{1-|\Gamma|} = 2.389$

06. (b)

Sol: Here we have $V_s = 10$ V, and $Z_s = 50\Omega$

The wavelength on the cable is

$$\lambda = \frac{v}{f} = \frac{2\times10^8}{150\times10^6} = 1.333\text{m}$$

$$\Rightarrow \beta = \frac{2\pi}{\lambda} = 1.5\pi\ \text{rad/m}$$

$\Rightarrow \beta\ell = 1.5\pi \times 2.5 = 11.781$ rad

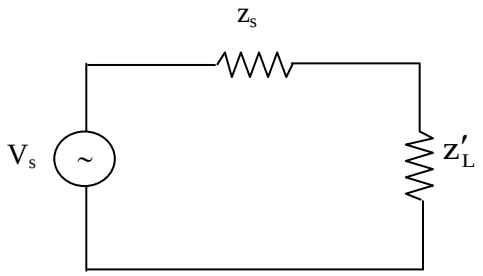
The transformed impedance at the generator-end is

$$Z'_L = Z_0\left(\frac{Z_L\cos\beta\ell + j\,Z_0\sin\beta\ell}{Z_0\cos\beta\ell + j\,Z_L\sin\beta\ell}\right)$$

$$= 54 + j\ 2.96\ \Omega$$

Now, from the equivalent lumped circuit at the generator-end we can calculate the power supplied to the input of the line. However, since the line is loss-less, power supplied by the generator to the line, i.e., the power delivered to the impedance Z'_L, is same as the power delivered to the load.

So,



$$I_A = \frac{V_s}{Z_s + Z'_L} = \frac{10}{50 + 54 + j2.96} = \frac{10}{104 + j2.96} = 0.096 - j0.00273\,A$$

$$V_A = I_A \times Z'_L = (0.096 - j0.00273) \times (54 + j2.96) = 5.192 + j0.137\,V$$

Now,

$$P_L = Re\{V_A I_A^*\} = Re\{(5.192 + j0.137) \times (0.096 + j0.00273)\} = 0.498\,W$$

(Note that there is no factor 1/2, since V_A and I_A are rms values)

06. (c)

Sol:

(i) Banker's Algorithm (Given by Dijkstra)

- Used for Multiple instances of each Resource Type.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker's Algorithm

Let n = number of processes and

m = number of resources types.

Available: Vector of length m. If available [j]= k, there are k instances of resource type Rj available.

Max: n × m matrix. If Max [i, j] = k, then process Pi may request at most k instances of resource type Rj.

Allocation: n × m matrix. If Allocation [i, j] = k then Pi is currently allocated k instances of Rj.

Need: n × m matrix. If Need[i, j] = k, then Pi may need k more instances of Rj to complete its task.

Need [i, j] = Max[i, j] – Allocation [i, j].

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i.

If $Request_i$ [j] = k then process P_i wants k instances of resource type R_j.

1. If $Request_i \le Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \le$ Available, go to step 3. Otherwise P_i must wait, since resources are not available.

3. Pretend to allocate requested resources to P_i by modifying the state as follows:

   Available = Available – $Request_i$;

   $Allocation_i = Allocation_i + Request_i$;

   $Need_i = Need_i – Request_i$;

   If safe $\Rightarrow$ the resources are allocated to P_i.

   If unsafe $\Rightarrow$ P_i must wait and the old resource allocation state is restored

Example: Banker's Algorithm

– Consider 5 processes P0 through P4; and
– 3 resource types:
A (10 instances),
B (5 instances) and
C (7 instances)

Snapshot at time T_0:

	Allocation	Max	Available
	A B C	A B C	A B C
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Matric Need = Max – Allocation

	A B C
P0	7 4 3
P1	1 2 2
P2	6 0 0
P3	0 1 1
P4	4 3 1

– The above system is in a safe state since the sequence < P1, P3, P4, P2, P0> satisfies safety criteria.
– Further if P1's request (1,0,2) arrives, we check that Request≤ Available that is,
(1,0,2) ≤ (3,3,2) $\Rightarrow$ true.

	Allocation	Need	Available
	A B C	A B C	A B C
P0	0 1 0	7 4 3	2 3 0
P1	3 0 2	0 2 0	
P2	3 0 1	6 0 0	
P3	2 1 1	0 1 1	
P4	0 0 2	4 3 1	

Executing safety algorithm shows that sequence <P1, P3, P4, P0, P2> satisfies safety requirement.

– Next if request for (3, 3, 0) by P4 arrives, we find it cannot be granted as
Request > Available i.e. (3,3,0) ≤ (2,3,0) $\Rightarrow$ false.
– But request for (0,2,0) by P0 be granted as Request ≤ Available (that is, (0,2,0) ≤ (2,3,0)) $\Rightarrow$ true.

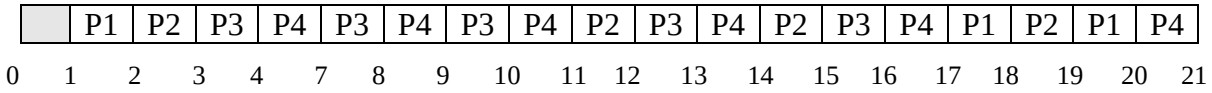
(ii)

Basis For Comparison	Deadlock	Starvation
Basic	Deadlock is where no process proceeds, and get blocked.	Starvation is where low priority processes get blocked, and high priority process proceeds.
Arising condition	The occurrence of Mutual exclusion, Hold and wait, No preemption and Circular wait simultaneously.	Enforcement of priorities, uncontrolled resource management.
Other name	Circular wait.	Lifelock.
Resources	In deadlocked, requested resources are blocked by the other processes.	In starvation, the requested resources are continuously used by high priority processes.
Prevention	Avoiding mutual exclusion, hold and wait, and circular wait and allowing preemption.	Aging.

06. (d)

Sol:

(i) **Gantt Chart**



Process	Arrival time (AT)	Burnt time (AT)	Completion time (CT)	Turn Around Time (TAT = CT – AT)	Waiting Time (WT = TAT – BT)
P1	1	2	18	17	15
P2	2	4	12	17	13
P3	3	6	20	17	11
P4	4	8	21	17	9

Average turn around time = $\frac{17+17+17+17}{4} = 17\,\text{msec}$

Average waiting time = $\frac{15+13+11+9}{4} = 12\,\text{msec}$

(ii) In demand paging environment, for a memory reference 2 times main memory is accessed. One for page table access (translation of logical address into physical address) and one for content access. Hence, effective access time = 2 × main memory access time

The standard solution to this problem is to use a special, small, fast-lookup hardware cache called a translation look-aside buffer (TLB). The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts: a key (or tag) and a value. When the associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. The search is fast; a TLB lookup in modern hardware is part of the instruction pipeline, essentially adding no performance penalty. To be able to execute the search within a pipeline step, however, the TLB must be kept small. It is typically between 32 and 1024 entries in size. Some CPUs implement separate instruction and data address TLBs. That can double the number of TLB entries available, because those lookups occur in different pipeline steps. We can see in this development an example of the evolution of CPU technology: systems have evolved from having no TLBs to having multiple levels of TLBs, just as they have multiple levels of caches.

The TLB is used with page tables in the following way. The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory. As just mentioned, these steps are executed as part of the instruction pipeline within the CPU, adding no performance penalty compared with a system that does not implement paging.

If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. Depending on the CPU, this may be done automatically in hardware or via an interrupt to the operating system. When the frame number is obtained, we can use it to access memory. In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference. If the TLB is already full of entries, an existing entry must be selected for replacement. Replacement policies range from least recently used (LRU) through round-robin to random. Some CPUs allow the operating system to participate in LRU entry replacement, while others handle the matter themselves. Furthermore, some TLBs allow certain entries to be wired down, meaning that they cannot be removed from the TLB. Typically, TLB entries for key kernel code are wired down.

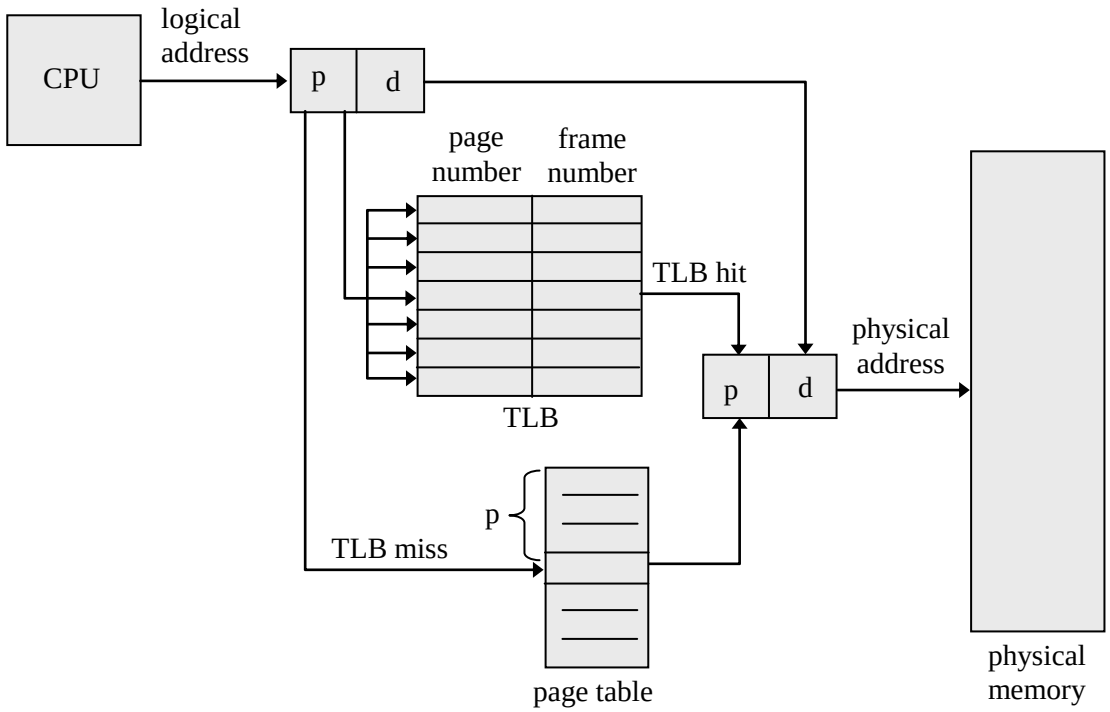
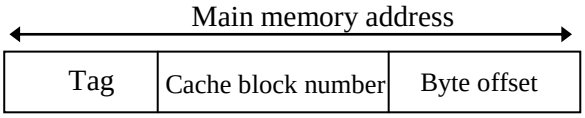


Fig: Paging hardware with TLB

07. (a)

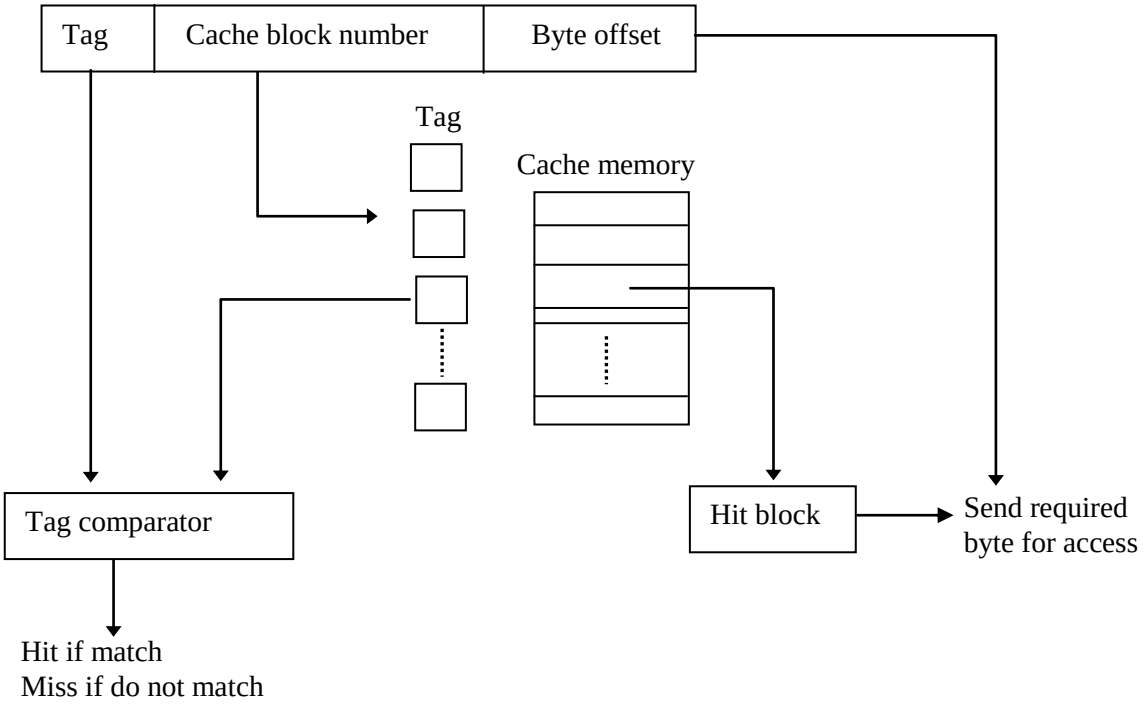
Sol:

(i) In direct-mapped cache the main memory address is divided in 3-parts as follows



Following are the steps to access cache with this address:
1. With cache block number first the mapped block number is checked from cache.
2. On this block present tag is compared with the tag in address generated.
3. If the tag is matched then hit otherwise miss.

All the steps are shown as below diagram.



(ii) Multiple memory chips are used to provide more capacity in the system.

Total memory capacity = No. of chips * 1 chip capacity

Types of arrangements:

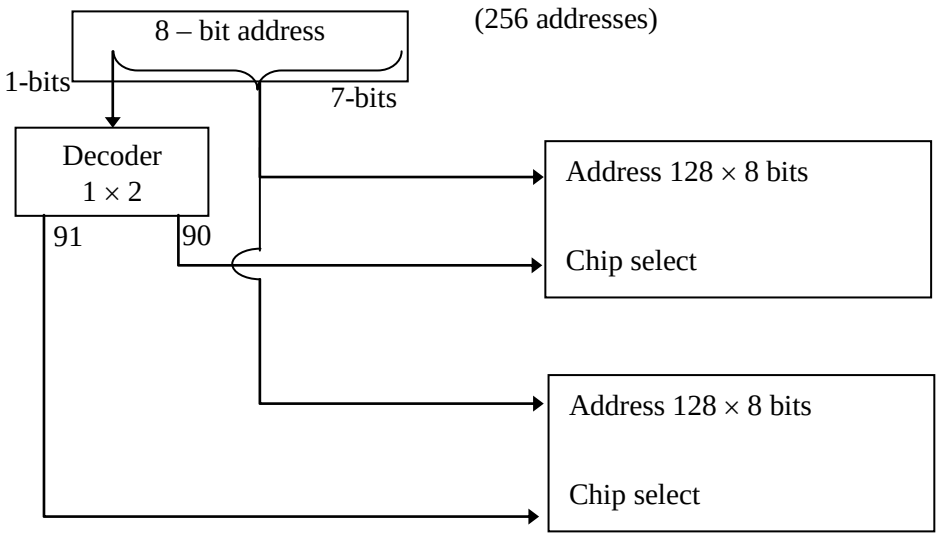
(A) Vertical: When number of addresses required is more as compared to one-chip addresses.

Ex: 1 chip size = 128 × 8 -bits

Total memory required = 256 × 8 bits

Hence no. of chips required = $\frac{256 \times 8\,\text{bits}}{128 \times 8\,\text{bits}} = 2$

2 chips here are arranged in vertical way



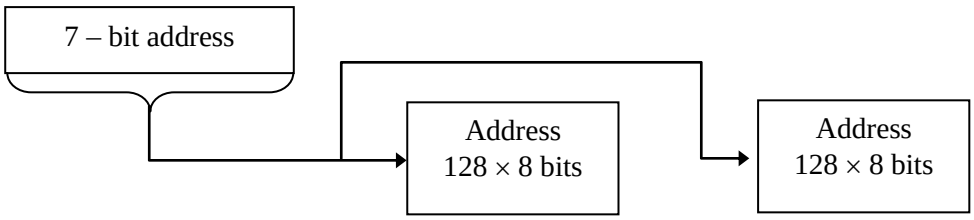
(B) Horizontal: This arrangement is used when data required on one address is more than one chips capacity.

Ex:

One chip capacity = 128 × 8 bits

Total capacity = 128 × 16 bits

No. of chips required = $\frac{128\times16}{128\times8}=2$



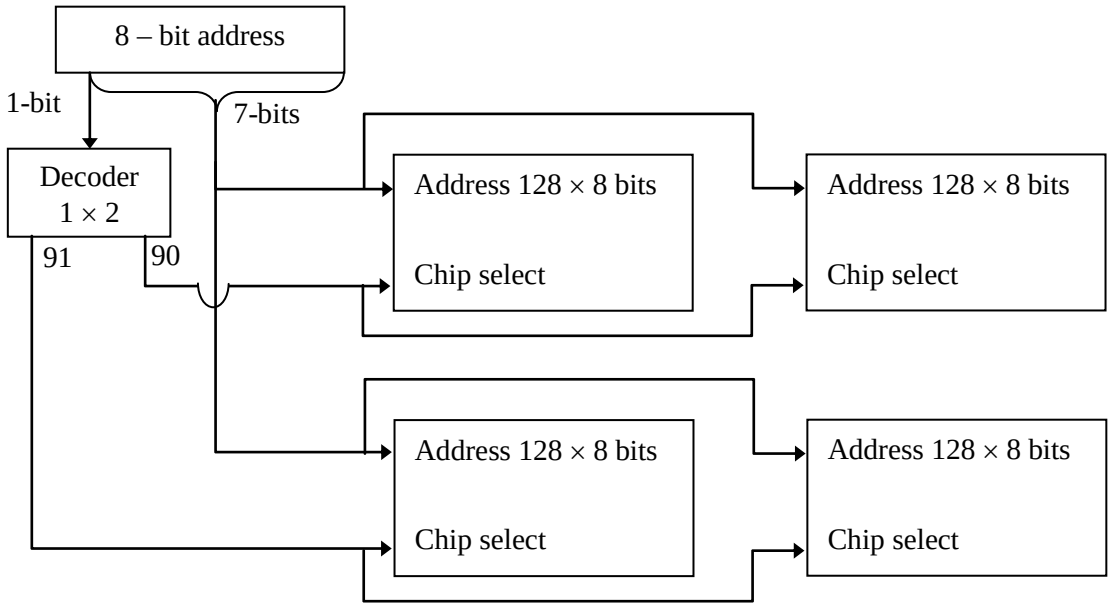
(C) Hybrid or mixed: This arrangement is used when both number of addresses and data size required one more than one chip's address and data.

Ex:

One chip capacity = 128 × 8 bits

Total capacity = 256 × 16 bits

No. of chips = $\frac{256\times16}{128\times8}=4\,\text{chips}$



(iii)

Basis For Comparison	**SRAM**	**DRAM**
Speed	Faster	Slower
Size	Large	Small
Cost	Expensive	Cheap
Used in	Cache memory	Main memory
Density	Less dense	Highly dense
Construction	Complex and uses transistors and latches.	Simple and uses capacitors and very few transistors.
Single block of memory requires	6 transistors	One transistor and one capacitor.
Charge leakage property	Not present	Present hence require power refresh circuitry
Power consumption	High	Low

07. (b)

Sol:

(i)
```
#include <stdio.h>
void main()
{
   int x, y, result = 1;
   printf("Enter x: ");
   scanf("%d", &x);
   printf("Enter y: ");
   scanf("%d", &y);
   int i = 1;
   while(i <= y)
   {
      result *= x;
      i++;
   }
   printf("%d^%d = %d", x, y, result);
}
```

(ii)
```
#include <stdio.h>
void main()
{
   int i;
   float num[10], sum = 0.0, average;

   for(i = 0; i < 10; ++i)
   {
      printf("%d. Enter number: ", i+1);
      scanf("%f", &num[i]);
      sum + = num[i];
   }
   average = sum / 10;
   printf("Average = %f", average);
}
```

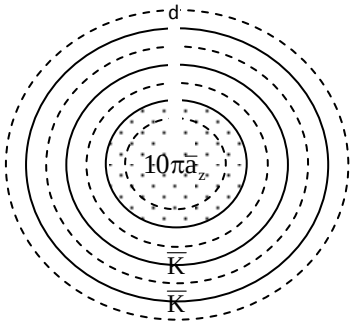
(iii)
```
#include <stdio.h>
void main()
{
   int i, first, num[10];
   for(i = 0; i < 10; ++i)
   {
      printf("%d. Enter number: ", i+1);
      scanf("%f", &num[i]);
    }
   printf("before the shift the array is: \n");
```

```
 for(i = 0; i < 10; ++i)
 {
    printf("%d\n", &num[i]);
  }
 int i, first;
 first = num[0];
 for(i=0; i<10; i++)
 {
    /* Move each array element to its left */
    num[i] = num[i + 1];
 }
 /* Copies the first element of array to last */
 num[SIZE-1] = first;
 printf("After the shift the array is: \n");
 for(i = 0; i < 10; ++i)
 {
    printf("%d\n", &num[i]);
  }
}
```

07. (c)

Sol:

(A) As the currents are directed in $\pm\ \hat{a}_z$ direction, H_ϕ component of magnetic field intensity is present. Consider a circular path of radius 'a' where a < 1mm. Apply Ampere's circuital law to this circular path.



$\oint \overline{H}.\overline{dL} = I_{encl}$

L.H.S. $= \oint \overline{H}.\overline{dL}$

$$= \int_0^{2\pi} \left(H_\phi \hat{a}_\phi\right)\left(\rho d\phi \hat{a}_\phi\right)\Big|_{\rho=a} = \int_0^{2\pi} \rho H_\phi d\phi\Big|_{\rho=a}$$

$= 2\pi a H_\phi$

R.H.S. $= I_{encl}$.

The current of 10π A is uniformly distributed through a circle of radius 1mm.

Hence

$$J = \frac{I}{\text{Area}} = \frac{10\pi}{\pi\left(10^{-3}\right)^2} = 10^7\ \text{A/m}^2$$

$\therefore$ RHS $= I_{encl} = J \times$ area .

$= 10^7 \times \pi a^2 = \pi a^2 10^7$

$\therefore 2\pi a H_\phi = \pi a^2 10^7$

$\therefore H_\phi = 0.5 \times 10^7 a$

$\therefore \overline{H} = 0.5 \times 10^7 a \hat{a}_\phi$ where a < 1mm

Replacing a by ρ, we have

$\overline{H} = 0.5 \times 10^7 \rho \hat{a}_\phi$ A/m for $\rho < 1$mm.

(B) Consider a circle of radius 'b' where 1mm < b < 2mm.
Apply Ampere's circuital law to this circular path.

$\oint \overline{H}.\overline{dL} = I_{encl}$

L.H.S. = $\oint \overline{H}.\overline{dL} = \int_0^{2\pi} \left(H_\phi \hat{a}_\phi\right).\left(\rho d\phi \hat{a}_\phi\right)\Big|_{\rho=b} = \int_0^{2\pi} \rho H_\phi d\phi\Big|_{\rho=b}$

$= 2\pi b H_\phi$

R.H.S. = $I_{end} = 10\pi A$

$\therefore 2\pi b H_\phi = 10\pi$

or $H_\phi = \frac{5}{b}$

or $\overline{H} = \frac{5}{b}\hat{a}_\phi$

where 1mm < b < 2mm.
Replacing b by ρ, we have

$\overline{H} = \frac{5}{\rho}\hat{a}_\phi A/m$, $1mm < \rho < 2mm$.

(C) Considers a circle of radius 4m where 2mm < c < 4 mm.
Apply Ampere's circuital law to this circular path.

$\oint \overline{H}.\overline{dL} = I_{encl}$

L.H.S. = $\oint \overline{H}.\overline{dL}$

$= \int_0^{2\pi} \left(H_\phi \hat{a}_\phi\right).\left(\rho d\phi \hat{a}_\phi\right)\Big|_{\rho=c} = \int_0^{2\pi} \rho H_\phi d\phi\Big|_{\rho=c}$

$= 2\ \pi c H_\phi$

R.H.S. = $I_{encl.}$

The current of 10πA is in $\hat{a}_z$ direction where as $\overline{K} = -1000\hat{a}_z$ at ρ = 2mm is in $-\hat{a}_z$ direction. The current at ρ = 2mm is equal to 1000 × circumference of ρ = 2mm circle

$= 1000 \times 2\ \pi(0.002) = 4\pi$ A directed in $-\hat{a}_z$ direction

$\therefore$ RHS = $I_{encl} = 10\pi - 4\pi = 6\pi$A.

$\therefore 2\pi c H_\phi = 6\pi$

$\therefore H_\phi = \frac{3}{c}$

$\therefore \overline{H} = \frac{3}{c}\hat{a}_\phi$ for $2mm < c < 4mm$

Replacing c by ρ we have

$\overline{H} = \frac{3}{\rho}\hat{a}_\phi$ for $2mm < \rho < 4mm$

(D) If $\overline{H} = 0$ for ρ > 4mm, then I_{encl} must be equal to zero for ρ > 4mm.
Let $\overline{K}$ be the current sheet located at ρ = 4mm.

$I_{end} = 10\pi - 4\pi + K \times$ circumference of ρ = 4mm circle

$\therefore 0 = 10\pi - 4\pi + K \times 2\pi\ (0.004)$

$\therefore K = -750$

Hence $\overline{K} = -750\hat{a}_z\ A/m$ must be present at ρ = 4mm

So that $\overline{H} = 0$ for ρ > 4mm.

08. (a)

Sol:

(i) (A) Given at y = 0

$$\overline{E} = 30\cos\left(10^9\pi t + \frac{\pi}{4}\right)\hat{a}_z$$

E at any 'y', for wave travelling in positive 'y' direction.

$$\overline{E} = 30e^{-\alpha y}\cos\left(\omega t - \beta y + \frac{\pi}{4}\right)\hat{a}_z \text{ V/m, } \omega = 10^9\pi$$

Given a lossy medium,

$$\alpha = \omega\sqrt{\frac{\mu\varepsilon}{2}\left[\sqrt{1+\left(\frac{\sigma}{\omega\varepsilon}\right)^2}-1\right]} = 0.942\,\text{Np/m}$$

$$\beta = \omega\sqrt{\frac{\mu\varepsilon}{2}\left[\sqrt{1+\left(\frac{\sigma}{\omega\varepsilon}\right)^2}+1\right]} = 20.97\,\text{rad/m}$$

$$\therefore E = 30e^{-0.942y}\cos\left(10^9\pi t - 20.97y + \pi/4\right)\hat{a}_z\,\text{V/m}$$

At t = 2ns, y = 1m

$E = 30e^{-0.942\times2}\cos(2\pi - 20.96\times1 + \pi/4)a_z$

$$\therefore E = 2.787\hat{a}_z\,\text{V/m}$$

(B) $\beta y = 10^\circ = \frac{10\pi}{180}\text{rad}$

$$y = \frac{10\pi}{180\times20.97} = 8.325mm$$

$\therefore$ distance travelled by the wave to have a phase shift of 10^o is 8.325mm.

(C) If amplitude is to be reduced by 40%

$\Rightarrow$ Net Amplitude is 30(0.6)

Distance travelled is $30(0.6) = 30e^{-\alpha y}$

$$y = \frac{1}{0.942}\ell n\frac{1}{0.6} = 542mm$$

(D) $$\eta = \sqrt{\frac{j\omega\mu}{\sigma + j\omega\varepsilon}} = \sqrt{35339}\angle(5.135/2)$$

$$\eta = 188\angle2.57^\circ$$

$a_k = a_E\times a_H \Rightarrow \hat{y} = \hat{z}\times\hat{x}$

$$H = \frac{E}{|\eta|}e^{-\alpha y}\cos(\omega t - \beta y + \pi/4 - \angle\eta)\hat{a}_x$$

$$H = \frac{30}{188}e^{-0.942y}\cos\left(10^9\pi t - 20.97y + \pi/4 - 2.57\right)\hat{a}_x$$

At y = 2m, t = 2ns

$$H = -22.6\hat{a}_x\,\text{mA/m}$$

(ii) (A) Give plane $x + z = 1 \Rightarrow x + z - 1 = 0$

Unit vector of plane is $\frac{a_x + a_z}{\sqrt{2}}$

Total power $P = \int P_{avg}.ds$ S : area of plate

$= P_{avg}.S\hat{a}_n$

$P_{avg} = \frac{1}{2}\eta H_o^2 \hat{a}_x$ $(\because S = (0.1)^2)$

$\therefore P_{total} = P_{avg}.S\hat{a}_n$

$= \left(\frac{1}{2} \times 120\pi \times (0.2)^2 \hat{a}_x\right).(0.1)^2\left(\frac{\hat{a}_x + \hat{a}_y}{\sqrt{2}}\right)$

$= \frac{1}{2\sqrt{2}}(120\pi)(0.2)^2(0.1)^2$

$P_{total} = 53.31mW$

(B) Given circular disc of radius 0.05m.

Area is $\pi r^2 = \pi(0.05)^2 = S$

$P_t = P_{avg}.S\,\hat{a}_x$

$= \left(\frac{1}{2} \times 120\pi \times (0.2)^2 \hat{a}_x\right).(\pi(0.05)^2)\hat{a}_x$

$P_t = 59.22mW$

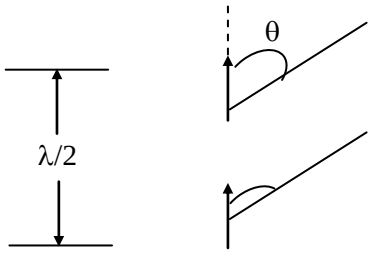
08. (b)

Sol: Since the antenna elements is the Hertz dipole, the primary element radiation pattern is sinθ.

The radiation pattern of the array is

$|E| = 2\cos(\psi/2)\sin\theta$

Where $\psi = \frac{2\pi}{\lambda} d\cos\theta + \delta$



Array of collinear Hertz dipoles

Elements are excited in phase and hence $\delta = 0$. The array phase ψ is

$$\psi = \frac{2\pi}{\lambda}\frac{\lambda}{2}\cos\theta = \pi\cos\theta$$

The normalized field pattern is

$$\Rightarrow |E| = \sin\theta\cos\left(\frac{\pi\cos\theta}{2}\right)$$

$$\text{Directivity } D = \frac{4\pi}{\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi} \sin^2\theta\cos^2\left(\frac{\pi\cos\theta}{2}\right).\sin\theta.d\theta d\phi}$$

$$D = \frac{4\pi}{2\pi\int_0^{\pi} \sin^3\theta\cos^2\left(\frac{\pi\cos\theta}{2}\right)d\theta} = \frac{2}{I}(\text{say})$$

Substituting for $\frac{\pi\cos\theta}{2} = t$, $\sin\theta d\theta = -\frac{2}{\pi}dt$

$$I = \int_{-\pi/2}^{\pi/2} \frac{2}{\pi}\left(1-\frac{4t^2}{\pi^2}\right)\cos^2 t.dt$$

$= 0.8677$

The directivity of the array is

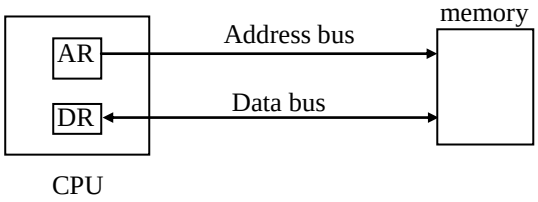
$$D = \frac{2}{0.8677} = 2.3$$

08. (c)

Sol:

(i) The data transfer between memory unit and CPU takes place with the help of data register DR. When CPU wants to read some information from memory unit, the information first brings to DR, and after that it goes to appropriate position. Similarly, data to be stored to memory must put into DR first, and then it is stored to appropriate location in the memory unit.

The address of the memory location that is used during memory read and memory write operations are stored in the memory register AR.



Read:

In case of memory read operation address is sent to memory using address bus. But before that, the address should be copied to AR in CPU.

Memory performs read on provided address and returns data to CPU using data bus. The data reaches to DR first in CPU, from there it is transferred to desired location.

Write:

In case of memory write address and data are sent from CPU to memory. CPU has to copy address to AR and data to DR, then only these 2 can be sent to memory via buses.

(ii) (A) Accumulator (AC): The accumulator is an internal CPU register used as the default location to store any calculations performed by the **arithmetic and logic unit**.

(B) Program counter (PC):

It is program counter that holds the address of the next instruction to be fetched; It's size is equal to the address bus size of the processor. After fetching an instruction, PC content is automatically incremented to point the address of the next instruction to be fetched.

(C) Stack pointer (SP): It is an address managing Register for stack memory, which provides the stack address while performing PUSH R and POP R instructions.
In Basic processor; SP content increments after executing PUSH R instruction and Decremented after executing POP R instruction

(D) Instruction Register (IR):
- It is an instruction Register that holds the opcode of the instruction after it's fetching
- After fetching an instruction, opcode will be placed in IR (from MDR) later it sends to control Register for completing it's Decode and execution.
- Size of the IR equal to the MDR size

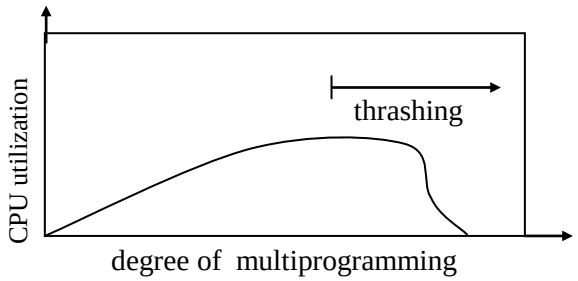
08. (d)
Sol:

(i) Initially when degree of multiprogramming increases, CPU utilization also increases but after a certain limit CPU utilization starts decreasing if the degree of multiprogramming increases more. This happens because in memory all process's content cannot be stored completely if number of processes are so many and in this scenario most of time, max. of processe's content is kept in hard disk. While executing a process CPU spends more time for being process content from hard disk as compared to execution. Hence CPU utilization will be less. This problem is known as thrashing.
Thrashing results in severe performance problem. Consider the following scenario, which is based on the actual behaviour of early paging systems.

The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page-replacement algorithm is used; it replaces pages without regard to the process to which they belong. Now suppose that a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames away from other processes. These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging divide, the ready queue empties. As processes wait for the paging device, CPU utilization decreases.

The CPU scheduler sees the deceasing CPU utilization and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more. Thrashing has occurred, and system throughput plunges. The page-fault rate increases tremendously. As a result, the effective memory-access time increases. No work is getting done, because the processes are spending all their time paging.

This phenomenon is illustrated in below figure, in which CPU utilization is plotted against degree of multiprogramming. As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming.



(ii) If synchronization is not provided between co-operating or communicating processes then following problems may arise

(A) Loss of data

(B) Inconsistency

(C) Deadlock

Operating system is a resource allocator. There are many resources that can be allocated to only one process at a time, and we have seen several operating system features that allow this, such as mutexes, semaphores or file locks.

Sometimes a process has to reserve more than one resource. For example, a process which copies files from one tape to another generally requires two tape drives. A process which deals with databases may need to lock multiple records in a database.

In general, resources allocated to a process are not preemptable; this means that once a resource has been allocated to a process, there is no simple mechanism by which the system can take the resource back from the process unless the process voluntarily gives it up or the system administrator kills the process.

This can lead to a situation called *deadlock*. A set of processes or threads is deadlocked when each process or thread is waiting for a resource to be freed which is controlled by another process. Here is an example of a situation where deadlock can occur.

```
Mutex M1, M2;
/* Thread 1 */
while (1) {
    NonCriticalSection( )
    Mutex_lock(&M1);
    Mutex_lock(&M2);
    CriticalSection();
    Mutex_unlock(&M2);
    Mutex_unlock(&M1);
}
/* Thread 2 */
while (1) {
NonCriticalSection( )
Mutex_lock(&M2);
Mutex_lock(&M1);
CriticalSection();
Mutex_unlock(&M1);
Mutex_unlock(&M2);
}
```

Suppose thread 1 is running and locks M1, but before it can lock M2, it is interrupted. Thread 2 starts running; it locks M2, when it tries to obtain and lock M1, it is blocked because M1 is already locked(by thread 1). Eventually thread 1 starts running again, and it tries to obtain and lock M2, but it is blocked because M2 is already locked by thread 2. Both threads are blocked; each is waiting for an event will never occur.