KLS

Vishwanathrao Deshpande Institute of Technology
Haliyad - 581329

Subject: Microcontrollers & Embedded Systems
[BCO601]

Sixth Semester B.E. / B.Tech. Degree Examination,
June/July 2025

Subject Code : BCO601

Department : Computer Science & Engineering
(AIML)

Sem : VI

Staff Name : Prof. Kanchan Khokale - K.S. Biradar C.

HOD : Dr. Poornima Raikar [signature] 14/3/26

Dean Academic : Dr. Gururaj H. [signature]

[signature]

CBCS SCHEME

USN | | | | | | | | | |

BCO601

Sixth Semester B.E./B.Tech. Degree Examination, June/July 2025

Microcontrollers and Embedded Systems

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	Explain the RISC design philosophy and how it differs from CISC architecture and List out its advantage with data flow.	10	L2	CO1
	b.	Discuss the concept of pipelining in ARM processors. How do interrupts effects the pipeline execution explain with one example.	10	L2	CO1
		OR			
Q.2	a.	With the help of neat supporting Block diagram explain ARM Cox dataflow model.	10	L2	CO1
	b.	Explain the role of registers and also explain 4 fields of Current Program Status Register (CPSR).	10	L2	CO1
		Module – 2			
Q.3	a.	List out all types of data processing institutions in ARM instruction set. Provide examples to illustrate the explanation.	10	L3	CO2
	b.	i) Write a program to add an array of 16-bit numbers and store the 32 bit result in internal RAM. ii) Write a program to find the largest / smallest number in an array of 32 numbers.	10	L3	CO2
		OR			
Q.4	a.	Write a C program to find square of the number between (1 – 10) and convert the same using assembly level program.	10	L3	CO2
	b.	i) Write assembly level program to interface a stepper motor and rotate it in clockwise and anticlockwise direction. ii) Write a assemble level program to find factorial of number.	10	L3	CO2
		Module – 3			
Q.5	a.	Explain the different purpose of Embedded system with examples.	10	L2	CO3
	b.	Write a assembly level code and structural representation to display hex digits O to F on a 7 segment LED interface with an appropriate delay M in between.	10	L3	CO3
		OR			
Q.6	a.	Explain different communication interface for embedded system with neat diagram.	10	L2	CO3
	b.	With the help of interfacing diagram write a assembly level program to interface a 4 x 4 keyboard and display the key code on an LCD.	10	L3	CO3
		Module – 4			
Q.7	a.	Explain the quality attributes of embedded system with different types.	10	L2	CO4
	b.	Explain state machine model with two example : i) FSM model for Automatic Tea / Coffee vending machine. ii) FSM model for coin operated telephone system.	10	L2	CO4

		OR			
Q.8	**a.**	Explain : i) Sequential Program model ii) Concurrent / Communicating Process model	**10**	**L2**	**CO4**
	b.	Explain automotive communication buses and key players of automotive embedded market concepts.	**10**	**L2**	**CO4**
		Module – 5			
Q.9	**a.**	Explain the concept of task process threads with the help of neat diagram.	**10**	**L2**	**CO5**
	b.	Explain the concept of Dead lock and Dining Philosopher's problem.	**10**	**L2**	**CO5**
		OR			
Q.10	**a.**	Explain message passing concept with neat diagram.	**10**	**L2**	**CO5**
	b.	What is semaphore explain binary semaphore concept with supporting diagram.	**10**	**L2**	**CO5**

* * * * *

VTU
SYN

Microcontrollers and Embedded Systems

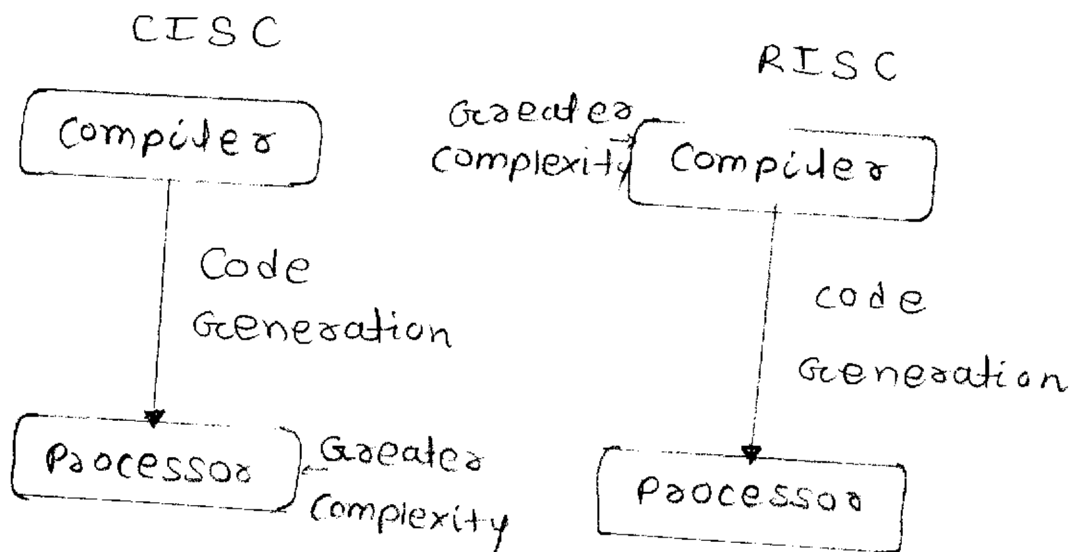
Sixth Semester B.E./B.Tech. Degree Examination, June/July 2025 [BCO601]

Module 1

Q.1 a] Explain the RISC design philosophy and how it differs from CISC architecture and List out its advantage with data flow. 10M

→ RISC (Reduced instruction set computer) is a design philosophy aimed at delivering simple but powerful instructions that execute within a single cycle at a high clock speed.

CISC (complex instruction set computer) relies more on the hardware for instruction functionality, and consequently the CISC instructions are more complicated.



CISC vs RISC. CISC emphisizes hardware complexity. RISC emphasizes compiler complexity.

RISC design philosophy is implemented with four major design rules :-

1) Instructions : RISC processors have a reduced number of instruction classes. These classes provide simple operations that can each execute in a single cycle. The compiler synthesizes complicated operations (ex. divide) by combining

several simple instructions. Each instruction is a fixed length to allow the pipeline to fetch future instructions before decoding the current instruction. In CISC processor instructions are often of variable size and take many cycles to execute.

2) Pipelines : The processing of instruction is broken down into smaller units that can be executed in parallel by pipelines. Ideally the pipeline advances by one step on each cycle for maximum throughput. Instructions can be decoded in one pipeline stage. There is no need for an instruction to be executed by a miniprogram called microcode as on CISC processor.

3) Registers : RISC machines have a large general-purpose register set. Any register can contain either data or an address. Registers act as the fast local memory store for all data processing operations. In contrast, CISC processors have dedicated registers for specific purposes.

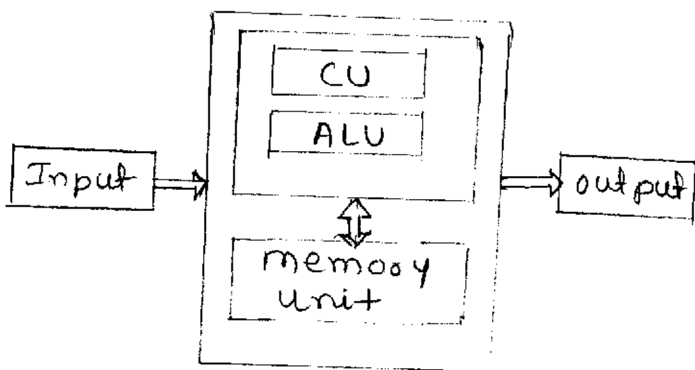
4) Load - Store architecture : The processor operates on data held in registers. Seperate load and store instructions transfer data between the register bank and external memory, memory access are costly, so seperating memory access from data processing provides an advantage because you can use data items held in the register bank multiple times without needing multiple memory accesses. In contrast, with a CISC design, the data processing operations can act on memory directly.

Von Neumann & Harvard architecture:

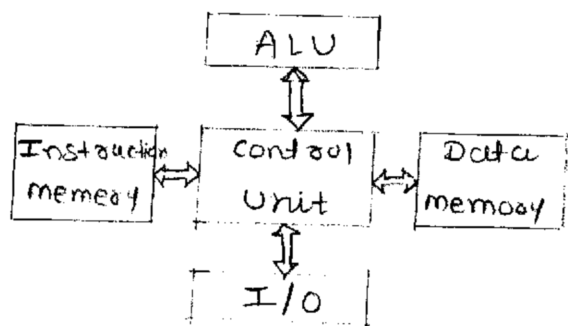
There is no strict rule for following architecture, RISC is most associated with Harvard architecture, and CISC is associated with Von Neumann architecture. Because their design philosophies naturally complement each other to achieve performance goals.

RISC & Harvard: RISC aims for high speed by executing simple, single cycle instructions. To maintain this processor must fetch the next instruction while simultaneously reading or writing data, only possible with the seperate buses of Harvard architecture.

CISC & Von Neumann: CISC uses complex, variable length instructions that often take multiple clock cycles to execute. Since they are not designed for single-cycle execution, they use shared Von Neuman bus architecture.



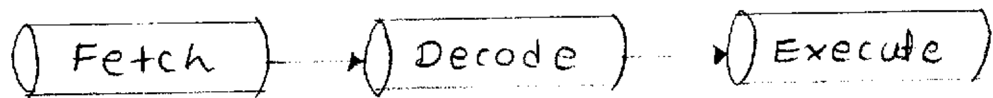
Von Neumann Model



Harvard Model

Q.1 b] Discuss the concept of pipelining in ARM processors. 10M
How do interrupts effect the pipeline execution explain with one example.

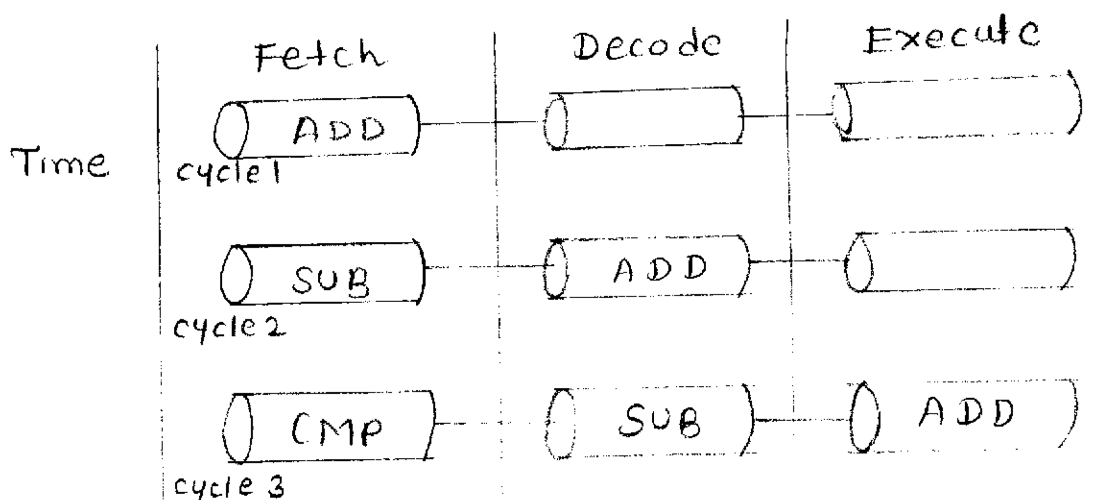
→ It is the mechnism a RISC processor uses to execute instructions. Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed.



ARM7 Three stage pipeline

Fetch loads an instruction from memory. Decode identifies the instruction to be executed. Execute processes the instruction & writes the result back to a register.

Figure shows simple example of, a sequence of three instructions being fetched, decoded & executed by the processor. Each instruction takes a single cycle to complete after the pipeline is filled.

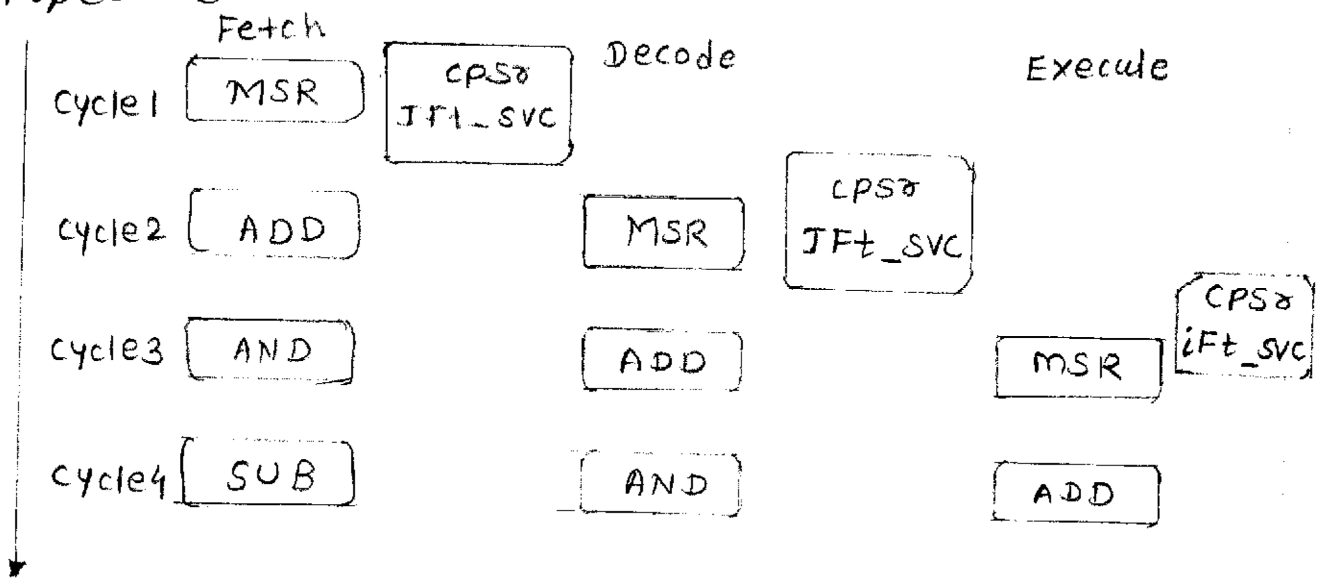


pipelined instruction sequence

The 3 instructions are placed into the pipeline sequentially. In the first cycle core fetches ADD instruction from memory. In the second cycle the core fetches the SUB instruction & decode the ADD instruction. In the third cycle, both the SUB & ADD are moved along the pipeline. The ADD instruction is executed, the SUB instruction is decoded, and the CMP instruction is fetched. This procedure is called filling the pipeline. The pipeline allows core to execute an instruction every cycle.

As the pipeline length increases, the amount of work done at each stage is reduced, which allows the processor to attain a higher operating frequency. Increases the performance. The system latency also increases because it make more cycles to fill the pipeline. The increased pipeline length also means there can be data dependency between some stages.

Pipeline with interrupt example:



ARM instruction sequence

The MSR instruction is used to enable IRQ interrupts, which only occurs once the MSR instruction completes the execute stage of the pipeline.

It clears the I bit in the cpsr to enable the IRQ interrupts.

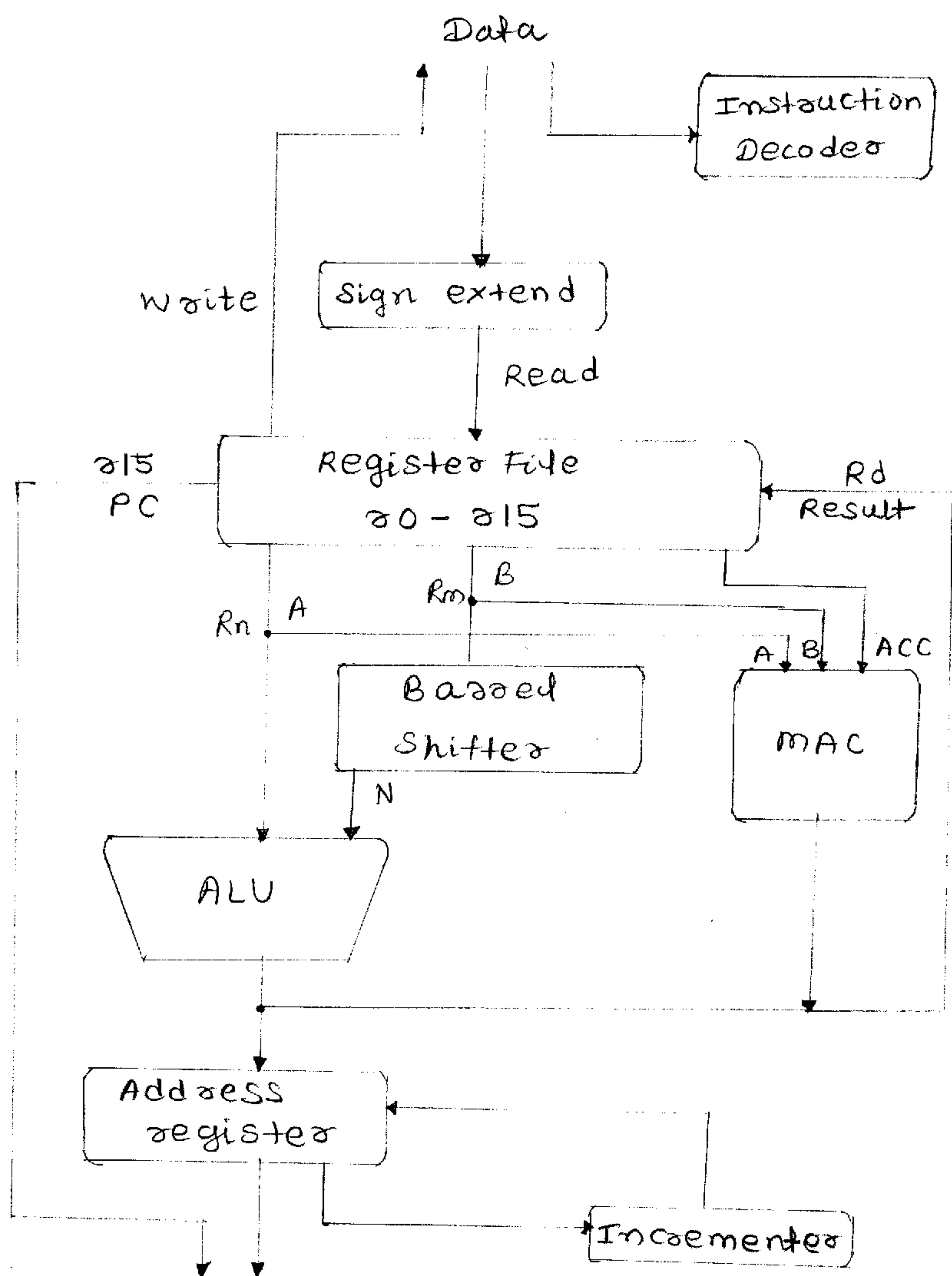
Once the ADD istruction enters the execute stage of the pipeline, IRQ interrupts are enabled.

7

M1

Q. 2. a] with the help of neat supporting Block diagram explain ARM Cox dataflow model.

→



ARM Cox Dataflow model

An ARM core is functional units connected by data buses as shown in figure, where, the arrows represent the flow of data, the lines represent the buses, and the boxes represent an operation unit or a storage area.

Instruction decoder: It translates instructions before they are executed. Each instruction executed belongs to a particular instruction set.

ALU (Arithmetic logic Unit) or MAC (multiply-accumulate unit) takes the register values Rn & Rm, and a single result or destination register, Rd. Source operands are read from the register file using the internal buses A and B respectively.
Data processing instructions write the result in Rd directly to the register file. Load & store Instructions use the ALU to generate an address to be held in the address register & broadcast on the Address bus.

Barrel shifter: Data processing instructions are processed within the ALU. A unique & powerful feature of the ARM processor is the ability to shift the 32-bit binary pattern in one of the source registers left or right by a specified number of positions before it enters the ALU. Pre-processing or shift occurs within the cycle time of the instruction. Useful for loading constants into a register.

Sign Extend Unit : Data items are placed in register file, storage bank made up of 32-bit registers. Registers hold signed or unsigned 32-bit values. The sign extend hardware converts signed 8 bit & 16-bit numbers to 32-bit values.

After passing through functional units the result in Rd is written back to the register file using the result bus. For load & store instructions the incrementer updates the address register before core reads or writes the next register. The processor continues execution until an exception or interrupt changes the normal execution flow.

Q. 2. b] Explain the role of registers & also explain 4 fields of current Program Status Register (CPSR). 10M

→ General purpose registers hold either data or an address.

Registers available in usermode -

r0 - r12
r13 sp
r14 lr
r15 pC
cpsr
-

All registers are 32 bit in size.
There are upto 18 active registers. 16 data & 2 processor status registers. The data registers are visible to the programmer as r0 to r15.

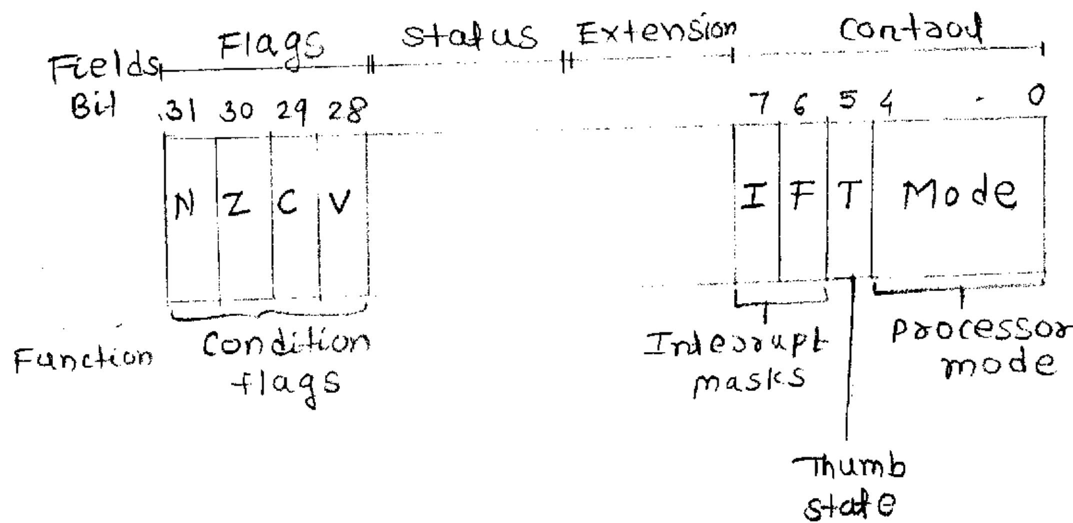
The ARM processor has 3 registers assigned to a particular task or special function : r13, r14 and r15. They are given different labels to differentiate them from the other registers.

r13 : used as the stack pointer (sp) and stores the head of the stack in the current processor mode.

r14 : called as link register (lr) and is where the core puts the return address whenever it calls a subroutine.

r15 : program counter (pc), contains the address of the next instruction to be fetched by the processor. In addition to 16 data registers, two program status registers cpsr & spsr (the current & saved program status registers respectively)

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 sp
r14 lr
r15 pc



A generic program status register (psr)

cpsr
-

Registers Available in User mode

The ARM core uses CPSR to monitor & control internal operations. It is dedicated 32 bit register and resides in the register file. It is divided into 4 fields each of 8 bits wide: flags, status, extension & control.

In current design the extension & status field are reserved for future use. The control field contains the processor mode, state and interrupt mask bits. The flag field contains the condition flags.

processor modes:

It determines which registers are active & the access right to the cpsr register itself. Each processor mode is either priviledged or nonprivileged.

The Jazelle J and Thumb T bit in the cpsr reflects the state of the processor. when both J & T bits are 0 in the precessor is in ARM state & execute ARM instruction.

module 2

Q.3. a] List out all types of data processing instructions in ARM instruction set. Provide examples to illustrate the explanation. — 10M

→ Types of data processing instructions:

1) move instructions
2) Arithmetic instructions
3) Logical Instructions
4) Comparison instructions
5) Multiply instructions

move instructions:- move copies N into a destination register Rd, where N is a register or immediate value. Useful for setting initial values & transfer data between registers.

Syntax: <instruction> {<cond>} {S} Rd, N

mov : move a 32 bit value into a register Rd = N

mvn : move the not of the 32 bit value into register Rd = ~N

N is a register or constant preceded by #

Example: move(copy) content of r5 to register r7

PRE r5 = 5
r7 = 8

mov r7, r5 ; let r7 = r5

POST r5 = 5
r7 = 5

Arithmetic instructions :- Used to implement addition & subtraction of 32-bit signed & unsigned values.

Syntax: <instruction> {<cond>} {S} Rd, Rn, N

ADD → add 2 32 bit values Rd = Rn+N

ADC → add 2 32 bit values & carry Rd = Rn+N+carry

SUB → Subtract 2 32 bit values Rd = Rn-N

N is the result of shifter operation

Example: Subtract value stored in register r2 from a value stored in register r1. The result is stored in register r0.

PRE r0 = 0x00000000
r1 = 0x00000002
r2 = 0x00000001

SUB r0, r1, r2 ; let r0 = r1 - r2

POST r0 = 0x00000001

Logical instructions:- Logical instructions perform bitwise logical operations on the two source registers.

Syntax: <instruction> {<cond>} {S} Rd, Rn, N

AND logical bitwise AND of two 32 bit values
Rd = Rn & N

ORR logical bitwise OR of two 32 bit values
Rd = Rn | N

EOR logical exclusive OR of two 32 bit values
Rd = Rn ^ N

BIC logical bit clear (AND NOT) Rd = Rn & ~N

Example: logical OR between registers r1 & r2, r0 holds the result

PRE r0 = 0x00000000
r1 = 0x02040608
r2 = 0x10305070

ORR r0, r1, r2

POST r0 = 0x12345678

Comparison instructions:- Used to compare or test a register value with a 32-bit value. They update CPSR field according to the result, but do not affect other registers. No need to apply S suffix

for comparison instructions to update flags.

syntax : <instruction> {<cond>} Rn, N

CMP	Compare	flag set as result Rn - N
CMN	Compare negated	
TEQ	test for equality of 2 32-bit values	
TST	test bits of a 32-bit value	

Example :- Compare r0 and r9. r0 and r9 are equal Z flag prior execution is 0 and represented by z. After execution z flag changes to 1 or an uppercase Z. This change indication of equality.

```
PRE    cpsr = nzcvqiFt_USER
          r0 = 4
          r9 = 4
          cmp r0, r9
POST   cpsr = nZcvqiFt_USER
```

<u>multiply instructions</u> : The multiply instructions multiply the contents of a pair of registers and accumulate result in another register. The long multiplies accumulate onto a pair of registers representing a 64-bit value. The final result is placed in a destination register or a pair of registers.

syntax : MLA {<cond>} {S} Rd, Rm, RS, Rn

MUL {<cond>} {S} Rd, Rm, RS

MLA - multiply & accumulate, Rd = (Rm * RS) + Rn

MUL - multiply , Rd = Rm * RS

Example : multiply r1 and r2, store result into r0 r1 is value 2, r2 is 2, result 4 is placed into r0

```
PRE   r0 = 0x00000000
      r1 = 0x00000002
      r2 = 0x00000002

      MUL r0, r1, r2 ;   r0 = r1 * r2

POST  r0 = 0x00000004
      r1 = 0x00000002
      r2 = 0x00000002
```

Q.3. b] i) write a program to add an array of 16-bit numbers & store the 32 bit result in internal RAM. 10M

ii) write a program to find largest / smallest number in an array of 32 numbers.

→ i) program to add an array of 16-bit numbers:

```
        AREA ADDITION, CODE, READONLY
        ENTRY
        MOV R5, #6          ; initialize counter N=6
        MOV R0, #0          ; initialize sum to 0
        LDR R1, =VALUE1     ; Load address of value1
        LDR R4, = RESULT    ; Load address to store result
LOOP    LDRH R3, [R1], #2   ; Load 16 bit data & increment R1 by 2
        ADD R0, R0, R3      ; ADD R3 to sum (R0)
        SUBS R5, R5, #1     ; Decrement counter
        BNE LOOP            ; Repeat until counter is 0
        STR R0, [R4]        ; Store 32-bit sum in Result
        B                   ; infinite loop to stop execution
VALUE1 DCW 0x1111, 0x2222, 0x3333, 0xAAAA,
           0xBBBB, 0xCCCC ; 16 bit values Array
```

```
        AREA DATA2, DATA, READWRITE ; Data section for result storage
        RESULT DCD 0x0      ; 32 bit space for result
        END
```

ii) Program to find LARGEST/SMALLEST Number in array of 32 numbers:-

```
        AREA SMALLEST, CODE, RAEDONLY
        ENTRY
START   MOV R5,#6           ; initialize counter to 6
                            ; (N=7 total elements)
        LDR R1,=VALUE1      ; Load address of first value
        LDR R2,[R1],#4      ; Load first element into R2
LOOP    LDR R4,[R1],#4      ; Load next array element
        CMP R2,R4           ; Compare R2 with R4
        BLS LOOP1           ; if R2<=R4, continue
        MOV R2,R4           ; if R4<R2, update R2 to
                            ; hold the smaller value
LOOP1   SUBS R5,R5,#1       ; Decrement counter
        CMP R5,#0           ; Compare counter to 0
        BNE LOOP            ; Loop back untill all elements
                            ; are checked
        LDR R4,=RESOLT      ; Load the address of result
        STR R2,[R4]         ; Store the smallest value in
                            ; result
        NOP
        NOP
        NOP
; Array of 32-bit numbers(N=7)
VALUE1  DCD 0X44444444, DCD 0X1111111
        DCD 0X22222222, DCD 0X33333333
        DCD 0XAAAAAAAA, DCD 0X88888888
        DCD 0X9999999
```

```
AREA DATA2, DATA, READWRITE ; To store result
RESULT DCD 0x0 ; 32-bit space for result
END ; make end of program
```

Q.4. a] write a C program to find square of the number between (1-10) and convert the same using assembly level program.

→ C program:

```
# include <stdio.h>
int square (int i);
int main (void)
{ int i;
    for (i=1; i<11 ; i++)
    {
        printf ("square of %d is %d\n", i, square(i));
    }
}
int square (int i)
{
    return i * i;
}
```

Assembly level program: (file- square.s)

```
        AREA |.text|, CODE , READONLY
        EXPORT square
        ; int square (int i)
square
        MUL r1, r0, r0 ;   r1 = r0 * r0
        mov r0, r1     ;   r0 = r1
        mov PC, lr     ;   return r0
        END
```

The AREA directive names the area or code section that the code lives in. we define a read-only code area called .text.

The EXPORT directive makes the symbol square available for external linking. we define the

symbol square as a code label.
When square is called, the parameter passing is defined by the ATPCS. The input argument is passed in register r0, and the returned value is in register r0. The multiply instruction has a restriction that the destination register must not be the same as the first argument register. Therefore we place the multiply result into r1 and move this to r0.
The END directive makes end of the assembly file.

Build using command line tools:

```
armcc -c main.c
armasm square.s
armlink -o main1.axf main1.o square.o
```

Only work if we compile C as ARM code.
If we compile C as Thumb code, then the assembly routine must return using BX instruction.
Use BX lr instead of MOV pc, lr whenever our processor supports BX (ARMv4T & above) create a new assembly file square2.s as follows:

```
    AREA |.text|, CODE, READONLY
    EXPORT square
    ; int square (int i);
square
    MUL r1, r0, r0   ; r1 = r0 * r0
    MOV r0, r1       ; r0 = r1
    BX  lr           ; return r0
    END
```

Q.4. b] i) write assembly level program to interface a stepper motor and rotate it in clockwise and anticlockwise direction. 10 M

ii) write a assembly level program to find factorial of a number.

→ i) Program to interface a stepper motor : (CLOCKWISE)

```
        AREA STEPPER, CODE, READONLY
        ENTRY
IO0DIR  EQU 0xE0028008
IO0SET  EQU 0xE0028004
IO0CLR  EQU 0xE002800C

START
        LDR R0, =IO0DIR
        MOV R1, #0x0F      ; P0.0 - P0.3 as output
        STR R1, [R0]

CW_LOOP
        LDR R0, =IO0SET
        MOV R1, #0x08
        STR R1, [R0]
        BL DELAY
        LDR R0, =IO0CLR
        MOV R1, #0x04
        STR R1, [R0]

        LDR R0, =IO0SET
        MOV R1, #0x02
        STR R1, [R0]
        BL DELAY
        LDR R0, =IO0CLR
        MOV R1, #0x02
        STR R1, [R0]
```

```
        B CW_LOOP

DELAY
        MOV R2, #0xFFFF
D1      SUBS R2, R2, #1
        BNE D1
        BX LR

        END
```

ANTICLOCKWISE Rotation :-

```
    AREA STEPPER, CODE, READONLY
    ENTRY

IO0DIR   EQU 0xE0028008
IO0SET   EQU 0xE0028004
IO0CLR   EQU 0xE002800C

START
        LDR R0, =IO0DIR
        MOV R1, #0x0F
        STR R1, [R0]
        BL D
ACW-LOOP
        LDR R0, = IO0SER IO0SET
        MOV R1, #0x01
        STR R1, [R0]
        BL DELAY
        LDR R0, = IO0SER IO0CLR
        MOV R1, #0x01
        STR R1, [R0]

        LDR R0, = IO0SET
        MOV R1, #0x02
        STR R1, [R0]
        BL DELAY
```

```
        LDR R0, =IO0CLR
        MOV R1,#0x04
        STR R1, [R0]

        LDR R0, =IO0SET
        MOV R1, #0x08
        STR R1, [R0]

        B ACW-LOOP

DELAY   MOV R2,#0xFFFF
        SUBS R2,R2,#1
        BNE D1
        BX LR

        END
```

ii) Assembly level program to find factorial of a n.

```
        AREA FACTORIAL, CODE, READONLY
        ENTRY
        MOV R0, #3          ; Store factorial number
                            ;   in R0
        MOV R1, R0          ; R1=R0
FACT    SUBS R1, R1, #1     ; Decrement R1 by 1
        MUL R3, R0, R1      ; R3 = R0 * R1
        MOV R0, R3          ; result R3 move to R0
        CMP R1, #1          ; Compare R1 with 1
        BNE FACT            ; Branch to the FACT if
                            ;   not equal
        NOP
        NOP
        NOP
        END                 ; mark END of file
```

This program calculate factorial of number 3.

Q. 5 a] Explain the different purpose of Embedded system with examples. 10m

→ Embedded system is designed to serve the purpose of any one or a combination of the following tasks:

1) Data collection / Storage / Representation
2) Data communication
3) Data (signal) processing
4) Monitoring
5) Control
6) Application specific user interface

1) Data Collection / Storage / Representation:

Embedded system designed for the purpose of data collection performs acquisition of data from the external world. Data collection is usually done for storage, analysis, manipulation and transmission. The term 'Data' refers to all kinds of information viz. text, voice, image, video, electrical signals and any other measurable quantities. Data can be either analog or digital. If the data is digital it is directly captured without any additional interface by digital embedded system. The collected data may be stored directly in the system or may be transmitted to some other system.

Example: Embedded system like Analog & Digital CROs without storage memory, collect data gives meaningful representation of the collected data by means of graphs or quantity value & deletes collected data when new data arrives at data collection terminal.

Some embedded systems store the collected data for processing & analysis. Such systems have a built-in / plug-in storage memory for storing the captured data. They gave meaningful representation of collected data by visual or audible means using LCD, LED, Buzzers, Alarms.
Example: measuring instruments with storage memory and monitoring instruments with storage memory used in medical application.
Certain systems collects data & used for internal processing.
Example: Digital camera is an example of data collection / storage / representation. Images are captured & stored within memory of camera. The captured images can also be presented to the user through a graphical LCD unit.

2) Data communication :- Embedded data communication systems are deployed in applications ranging from complex satellite communication systems to simple home networking systems.
The data collected by embedded terminal may required transferring of the same to some other system located remotely.
The transmission is achieved either by a wire-line medium or by a wire-less medium.
Data can be transmitted by analog means or digital means.
Modern industry trends are setting towards digital communication.
The data collected terminals itself can incorporate data communication units like wireless modules C

Bluetooth, ZigBee, Wi-Fi, EDGE, GPRS) or wire-line modules (RS-232C, USB, TCP/IP, PS2)

Certain system acts as a dedicated transmission unit between the sending & receiving terminals, offering functionalities like data packetizing, encrypting & decrypting.

Network hubs, routers, switches etc.

They act as a mediators in data communication & provide features like security, monitoring etc.

3) Data (signal) processing :- Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech, coding, synthesis, audio video codec etc.

Example: A digital hearing aid is typical example of an embedded system employing data processing.

4) monitoring :- Embedded systems in this category are specifically designed for monitoring purpose. Almost all embedded products coming under the medical domain are with monitoring functions only. They are used to determine the state of some variables using input sensors. They cannot impose control over variables.

Example: ECG machine for monitoring the heartbit of a patient. machine can only monitor heart bit but it can not control heart bit.

other examples are measuring instruments like Digital CRO, Digital multimeters, logic analyzers.

5) Control : Embedded systems with control functionalities impose control over some variables according to the changes in input variables. A system with control functionality contains both sensors & actuators.

sensors are connected to the input port for capturing the changes in environmental variables or measuring variable.

The actuators connected to the output port are controlled according to the canges in input variable to put an impact on the controlling variable to bring the controlled variable to the specified range.

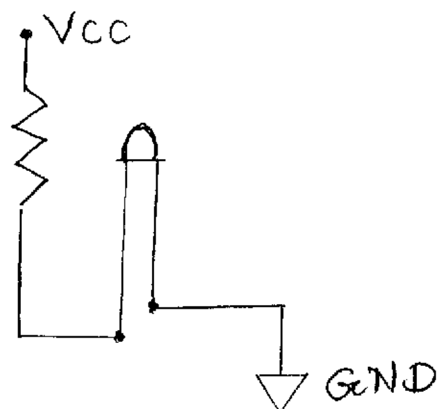
Example: The AC used in our home to control the temperature. AC contains a room temperature sensing element (sensor) which may be a thermistor & a handled unit for setting up (feeding) the desired temperature. Here input variable is the current room temperature & controlled variable is also the room temperature

6) Application Specific User Interface : These are the embedded systems with application-specific user interfaces like buttons, switches, keypad, lights, bells, display units etc.

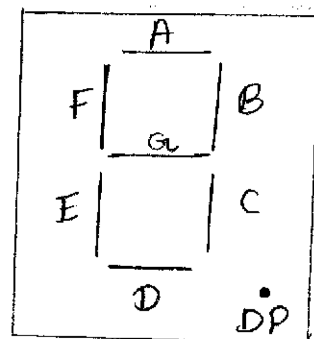
Example: Mobile phone - The user interface is provided through the keypad, graphic LCD module, system speaker, vibration alert etc.

Q.5. b] write a assembly level code & structural representation to display hex digits 0 to F on a 7 segment LCD interface with an appropriate delay M in between. 10M

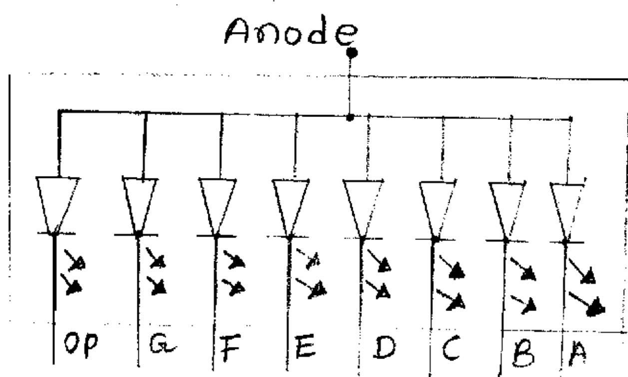
→ LED is an important output device for visual indication in an embedded system. LED can be used as an indicator for the status of various signals or situations.



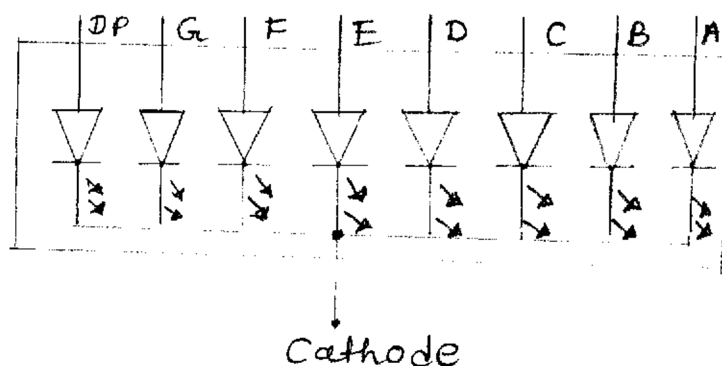
LED Interfacing



7 segment LED Display



Common Anode LED Display



Common cathode LED Display

LED used Indicating the presence of power conditions like 'Device ON', 'charging of battery' for a battery operated handheld embedded devices.

7 segment LED display is an output device for display alpha numeric characters. It contains 8 light - emitting diode (LED) segment arranged in a special format of the 8 LED segment, 7 are used for displaying alpha numeric characters & 1 is used for representing 'decimal point'.

Assembly level code :-

```
        AREA  RESET, CODE, READONLY
        ENTRY
START
        LDR  R0, = 0xE0028008
        LDR  R1, = 0x000000FF   ; Set P0.0 to
                                  P0.7 as output

        STR  R1, [R0]
        LDR  R2, = SEG_TAB
        MOV  R3, #16            ; 16 values (0-F)

LOOP    LDRB R4, [R2], #1       ; Load segment code
        LDR  R5, = 0xE0028004
        STR  R4, [R5]
        BL DELAY
        SUBS  R3, R3, #1
        BNE  LOOP
        B  START

DELAY
        MOV R6, #0xFFFF

D1      SUBS R6, R6, #1
        BNE D1
        BX LR

SEG_TAB  DCB 0x3F ; 0        DCB 0x7D ; 6
         DCB 0x06 ; 1        DCB 0x07 ; 7
         DCB 0x5B ; 2        DCB 0x7F ; 8
         DCB 0x4F ; 3        DCB 0x6F ; 9
         DCB 0x66 ; 4        DCB 0x77 ; A
         DCB 0x6D ; 5        DCB 0x7C ; B
                             DCB 0x39 ; C
                             DCB 0x5E ; D
         DCB 0x79 ; E        DCB 0x71 ; F
         END
```

Q.6 a] Explain different communication interface for embedded system with neat diagram. 10M

→ Communication interfaces in embedded systems enable data exchange between internal components (onboard) and external devices, utilizing protocals like I2C, SPI, UART, CAN, USB and Ethernet. These interfaces are critical for connecting microcontrollers to sensors, displays and networks. Categorized by speed, distance, and wether they are serial or parallel.

Onboard Communication Interfaces: These connets components on the same PCB.

I2C (Inter-Integrated Circuit): A two-wire (SDA, SCL), synchronous, half-duplex bus. suitable for conneting multiple low-speed devices like sensors and EEPROMS.

SPI (Serial Peripheral Interface): A four-wire (MOSI, MISO, SCK, CS), full-duplex, synchronous interface for serial communication of high speed short-distance communication.

UART (Universal Asynchronous Receiver/Transmitter) A two-wire (TX, Rx) asynchronous interface for serial communication.

1-wire: A single-wire interface for communication with simple sensors.

External communication Interfaces: These connect the embedded system to external devices or networks.

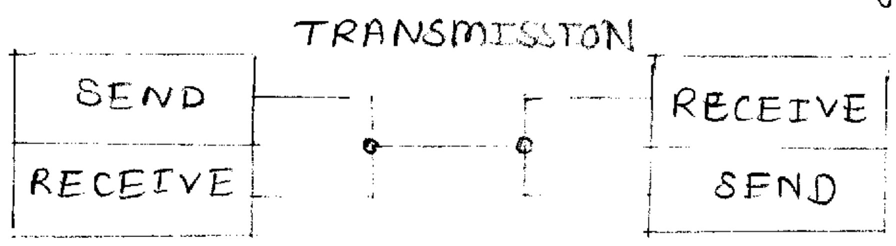
USB (Universal Serial Bus): A widely used, hot-swappable interface for connecting peripherals.

Ethenet / IEEE 802.3 : provides high speed wired network connectivity

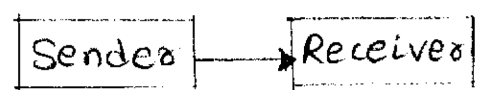
CAN (Controller Area Network) : A robust, multi-master, different protocal common in automotive and industrial applications for inter-device communication.

RS-232 / RS-485 : Serial standards often used for industrial communication.

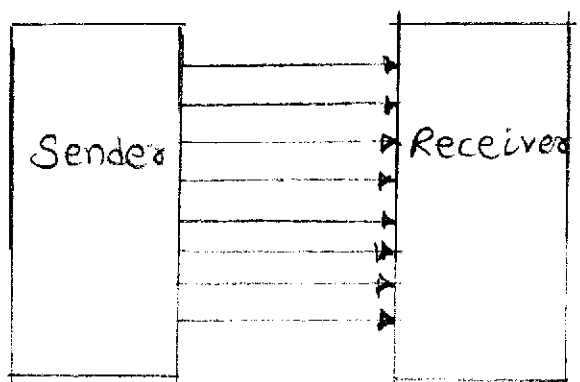
wireless protocals : wi-fi, Bluetooth, ZigBee and NFC are used for short to medium-range wireless.



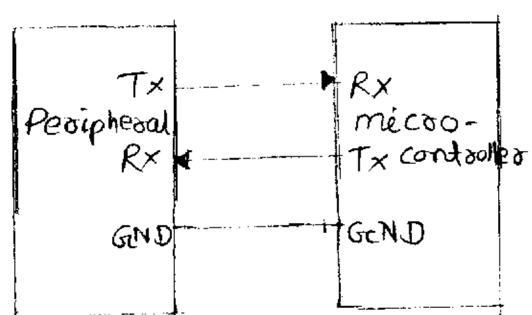
Half - Duplex



Serial Communication
Data is sent 1 bit at a time.

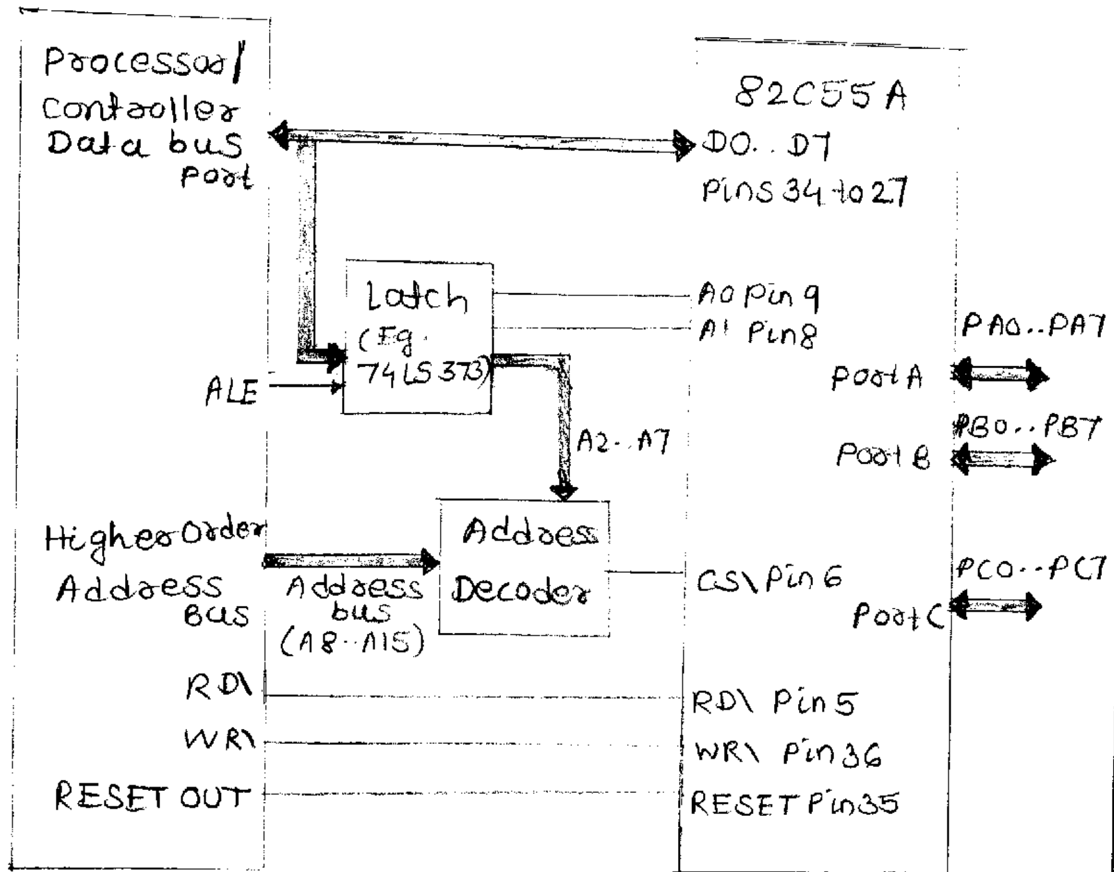


Parallel communication
Data is sent 8 (or more) bits at a time.



UART communication (Full-Duplex)

Q.6. b] With the help of interfacing diagram write a assembly level program to interface a 4x4 keyboard and display the key code on LCD. — 10M



Interfacing of 8255 with an 8 bit microcontroller

Port Assignment:

Port A – LCD Data

Port B – Keypad Columns

Port C – keypad Rows + LCD Control signals

Working : i) Rows are set as output & columns as input.

ii) The program sends 0 to one row at a time

iii) If a key is pressed, the corresponding columns becomes low

iv) The microprocessor/controller reads the column and determines which key is pressed.

v) The key code is then sent to the LCD for display.

ARM Assembly Program:

```
AREA KEYPAD_LCD, CODE, READONLY
ENTRY
; GPIO Addresses (LPC2148)
 IO0DIR   EQU  0xE0028008
 IO0PIN   EQU  0xE0028000
 IO0SET   EQU  0xE0028004
 IO0CLR   EQU  0xE002800C

 IO1DRR   EQU  0xE0028018
 IO1SET   EQU  0xE0028014
 IO1CLR   EQU  0xE0028D1C
START
   ; Configure keypad rows as output & columns as input
     LDR R0, =IO0DIR
     MOV R1, #0x0F       ; P0.0 - P0.3 output
     STR R1, [R0]
   ; Configure LCD port as output
     LDR R0, = IO1DRR
     MOV R1, #0x7FF
     STR R1, [R0]
     MAIN_LOOP
      ; Scan Row1
     LDR R0, =IO0CLR
     MOV R1, #0x01
     STR R1, [R0]
     LDR R0, IO0PIN
     LDR R2, [R0]
     AND R2, R2, #0xF0
     CMP R2, #0xE0
     BEQ KEY1
```

```
B MAIN_LOOP
KEY 1
MOV R3, #'1'
BL LCD_DISPLAY
B MAIN_LOOP
; LCD Display Subroutine
LCD_DISPLAY
LDR R0, =IO1SET
STR R3, [R0]
BL DELAY
LDR R0, =IO1CLR
STR R4, [R0]
BX LR
DELAY
MOV R5, #5000
D1
SUBS R5, R5, #1
BNE D1
BX LR
END
```

Q.7. a] Explain the quality attributes of embedded system with different types. 10M

→ Quality attributes of embedded system are of two types:-

1) Operational quality Attributes
2) Non-operational quality Attributes

Operational Quality Attributes :-

Attributes related to the embedded system when it is in the operational mode or 'online' mode.

1) Response: It is a measure of quickness of the system. It gives you an idea about how fast your system is tracking the changes in input variables. Most of system demand fast response which should be almost Real Time.
eg. An embedded system deployed in flight control application should respond in a Real time manner. Any response delay in the system will create potential damages to the safety of flight as well as passengers.
It is not necessary that all system should be Real Time in response eg. Electronic toy is not at all time critical.

2) Throughput : Deals with efficiency of a system. It can be defined as the rate of production or operation of a defined process over stated period of time.
The rates can be expressed in terms of units of products, batches produced, or any other meaningful measurements.
Example: In the case of card Reader, throughput means how many transactions the Reader can perform in a minute or in an hour or in a day.

3) Reliability: measure of how much % you can rely upon the proper functioning of the system or what is the % susceptibility of the system to failures.

Mean Time Between Failures (MTBF) & Mean Time To Repair (MTTR) are terms used in defining the system reliability.

4) Maintainability: Deals with support & maintainance to the end user, or client in case of technical issues & product failures or on the basis of a routine system checkup. Reliability & maintainability are two complementry desplines. A more reliable system means, system with less corrective maintainability requirement. Maintainability is closely related to the system availability. It can be classified into two categories:

   i) Scheduled periodic maintainance
   ii) Maintainance to unexpected failures.

Example: Printer is example for both

5) Security :- 'Confidentiality', 'Integrity', 'Availability' are the three major measures of information security.

Confidentiality deals with protection of data & application from unauthorized disclosure.

Integrity deals with the protection of data & application from unauthorized modification.

Availability deals with protection of data & application from unauthorized users.

6) Safety :- Safety deals with the possible damages that can happen to the operations, public and environment due to the breakdown of an embedded system.

Non - Operational Quality Attributes :- Attributes that needs to be addressed for the product 'not' on the basis of operational aspects are grouped under this category.

i) Testability & Debug-Ability: It deals with how easily one can test his/her design, application, and by which means he/she can test it. Testability is applicable to both the embedded hardware & firmware. Hardware testing ensures that the peripherals and the total hardware functions in the desired manner. Firmware testing ensures that firmware is functioning in expected way.

Debugging can be hardware level or firmware level. Hardware debugging is used to figure out the issue created by hardware. Firmware debugging is employed to figure out portable errors that appear as a result of flaws in firmware.

ii) Portability :- Is a measure of 'system independence'. An embedded system is said to be portable if the product is capable of functioning 'as such' in various environments, target processors/controllers & embedded operating systems. Standard product should always be flexible & portable.

iii) Time-to-prototype & market :- Time-to-market is the time elapsed between the conceptualisation of a product and the time at which the product is ready for selling (for commercial products) or use (for non-commercial products). Time-to-market is very critical factor. As market is very competitive. Product prototyping helps a lot reducing time-to-market.

iii) Per unit Cost & Revenue :- Cost is a factor which is closely monitored by both end user (those who buy the product) and product manufacturer (those who build the product). Cost is highly sensitive factor for commercial products. Proper market study & cost benefit analysis should be carried out before taking a decision on the per-unit cost.

Every embedded product has a product life cycle which starts with the design and development phase.

Design & development phase : The product idea generation, prototyping, Roadmap defenition, actual product design and development are the activitees carried out in this stage. In this stge there is only investment & no returns.

Product Introduction Phase :- Once product is ready to sell it is introduced to the market. There not be much competeion. Product sales & revenue is increased with time.

Growth phase : Product grabs high market share

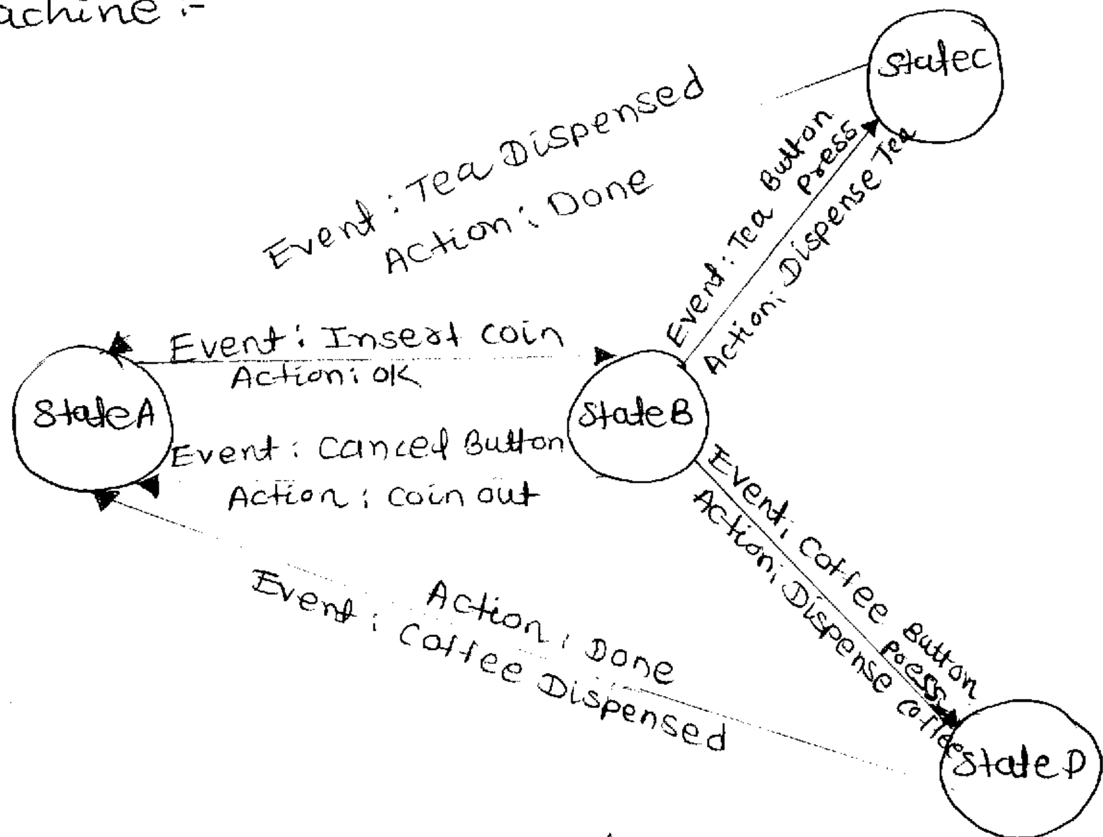
Maturity phase : the growth and sales will be steady and the revenue reaches at its peak.

Product Retirement / Decline Phase :- starts with drop in sales volume, market share and revenue.

iv) Evolvability :- Evolvability is referred as non-heritable variation. For embedded system it refers to the ease with which the product can be modified to take advantage of new hardware or software technologies.

Q.7. b] Explain state machine model with two example 10 M
i) FSM model for Automatic Tea / coffee vending mahine
ii) FSM model for coin operated telephone system.

→ State machine model : It is used for modelling reactive or event-driven embedded system whose processing behaviour are dependent on state tranitions. The state machine model describes the system behavior with 'states', 'Events', 'Actions' and 'Transitions'
i) FSM model for Automatic Tea/coffee vending machine :-



FSM model for Automatic Tea/ coffee vending mahine

The tea/coffee vending is initiated by user inserting 5 rupee coin. After inserting the coin the user can either select 'coffee' or 'Tea' or press 'cancel' to cancel the order and take back the coin. It contain 4 states namely, 'wait for coin', 'wait for user input', 'Dispense Tea' and 'Dispense coffee' The event 'Insert coin' (5 rupee coin insertion), transition the state to 'wait for user Input'.

ii) FSM model for coin operated telephone system:

The FSM model shown is a simplest representation and it doesn't take care of scenarios like user doesn't insert a coin within the specified time after waiting.

Most of the time state machine model translates the requirements into sequence driven program and it is difficult to implement concurrent processing with FSM. This limitation is addressed by the Heirarchical concurrent Finite state machine model (HCFSM). It is extension of FSM for supporting concurrency and hierarchy.

i) The calling process is initiated by lifting the receiver (off-hook) of the telephone unit.

ii) After lifting the phone the user needs to insert a 1 rupee coin to make the call

iii) If the line busy, the coin is returned on placing the receiver back on the hook (on-hook)

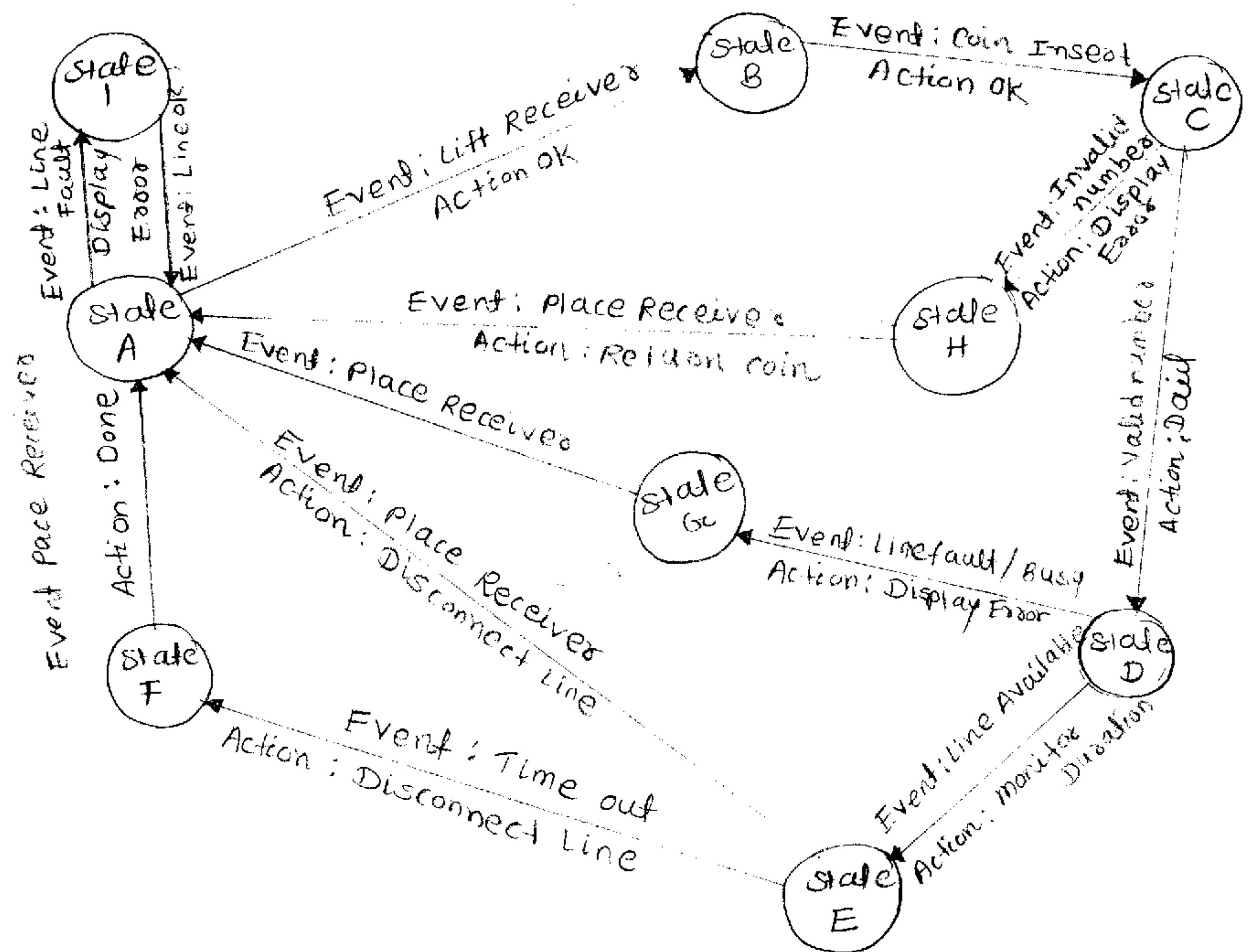
iv) If the line is through, the user is allowed to talk till 60 seconds and at the end of 45th second, prompt for inserting another 1 rupee coin for continuing the call is initiated.

v) If the user doesn't insert another 1 rupee coin the call is terminated on completing the 60 seconds time slot.

vi) The system goes to the 'out of order' state when there is a line fault.

Above all requirements are taken care in diagram. Events occured, state transitions and action takens are shown in diagram.

Diagram on next page.



State A : Ready

State B : wait for coin

State C : wait for number

State D : Dailing

State E : Call in progress

State F : call terminated

State G : Unable to make call

State H : Invalid number input

State I : out of order

FSM model for Coin Operated Telephone System

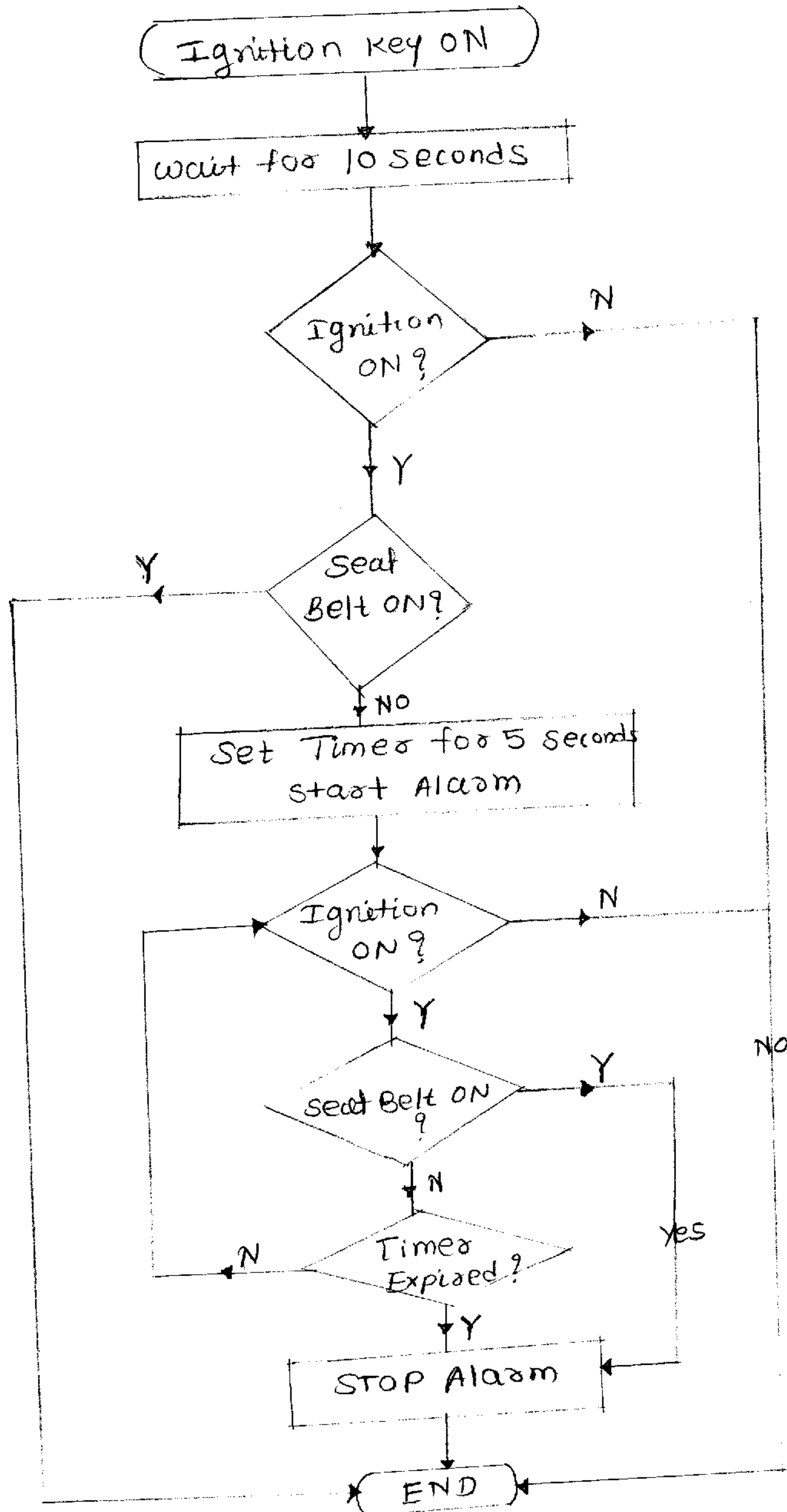
Q. 8. a] Explain:
i) Sequential program model
ii) Concurrent / communicating process model

10 M

→ i) Sequential Program Model :- The functions or processing requirements are executed in sequence. It is same as the conventional procedural programming. Here the program instructions are iterated & executed conditionally and the data gets transformed through a series of operations. FSMs are good choice for sequential program modelling. Another important tool used for modelling sequential program is Flow charts. The FSM approach represents the states, events, transitions and actions, whereas the Flow chart models the execution flow. The execution of function in a sequential program model for the 'Seat Belt warning' system is as below :-

Example:

```
# define No 0
Void seat_belt_warn()
{
   wait_10sec();
   if ( check_ignition_key()==ON)
   {
      if (check_seat_belt() == OFF)
      {  set_timer(5);
         start_alarm();
         while ((check_seat_belt()==OFF)
          && (check_ignition_key()==OFF)
          && (timer_expire() = No));
         stop_alarm();
      }
   }
}
```


Ignition Key ON
wait for 10 seconds
Ignition ON?
N
Y
Y
Seat Belt ON?
NO
Set Timer for 5 seconds
start Alarm
Ignition ON?
N
Y
NO
Seat Belt ON?
Y
N
Yes
Timer Expired?
N
Y
STOP Alarm
END

ii) Concurrent / Communicating Process model :-

The concurrent or communicating process model models concurrently executing tasks/processes. It is easier to implement certain requirements in concurrent processing model than the conventional sequential execution. sequential execution leads to a single sequential execution of task and thereby leads to poor processor utilization, when the task involves I/O waiting, sleeping for specified duration etc. If the task is split into multiple subtasks it is possible to tackle the CPU usage effectively, when the subtask under execution goes to wait or sleep mode, by switching the tasks execution. However concurrent processing model requires additional overheads in task scheduling, task synchronisation and communication.

Example:

Implement the 'seat belt warning' system in Concurrent processing model we can split task into:

1) Timer task for waiting 10 seconds (wait timer task)
2) Task for checking the ignition key status (ignition Key status monitoring task)
3) Task for checking the seat belt status (seat belt status monitoring task)
4) Task for starting & stopping the alarm (alarm control task)
5) Alarm timer task for waiting 5 seconds (Alarm timer task)

we have 5 ~~steps~~ tasks and we cannot execute them randomly or sequentially. we need to synchronise their execution through some mechanism. we need to start the alarm only after the expiration of the 10 sec wait timer and that too only when wait timer is expired.

Create & initialize events
wait-timer_expire,
ignition_on, ignition-off,
seat_belt-on, seat-belt-off
alarm_timer_start,
alarm_timer_expire
create task wait Timer
create task Ignition key status
monitor, create task seat Belt
Status monitor, create task
Alarm control, Alarm Timer

wait Timer Task
sleep(10S);
//signal wait_timer_expire
set Event wait timer_expire

Alarm control Task
wait for signalling of
wait_timer_expire

if (ignition_on &
seat_belt_off) {
start Alarm();
set Event Alarm start;
wait for signalling of
alarm_timer_expire or
ignition_off or
seat_belt_on;
Stop Alarm(); }

Alarm Timer Task
wait for event
alarm start;
sleep (10S);
Set Event
alarm_timer expire

Ignition key status
monitor Task
while (1) {
if (Ignition key ON)
{
Set Event Ignition on;
Reset Event Ignition off;
}
else
{
set Event ignition off
Reset Event
ignition on;
}}

Ignition seat belt
status monitor
Task
while(1) {
if (Seat Belt ON)
Set Event seat_belt_on;
Reset Event seat-belt_off;
}
else
{
set Event seat-belt-off;
Reset Event seat_belt_on;
}
}

'Seat Belt warning System' concurrent processing program model

Q.8. b] Explain automotive communication buses and key players of automative embedded market system. 10M

→ Automotive applications uses serial buses for communication, which greatly reduces the amount of wiring required inside a vehicle.

Controller Area Network (CAN): The CAN bus was originally proposed by Robert Bosch. It supports medium speed (ISO11519 - class B with data rates upto 125 kbps. & high speed (ISO11898 - Class C with support for error handling in data transmission. It is used in safety system like airbag control & ABS Navigation system like GPS.

Local Interconnet Network (LIN): LIN bus is a single master multiple slave (upto 16 independent slave nodes) communication interface. LIN is slow speed, single wire communication interface with support for data rates upto 20kbps and is used for sensor/actuator interfacing. LIN bus follows the master communication triggering technique to eliminate the possible bus arbitration. LIN bus employed in applications like mirror controls, fan controls, seat positioning controls, windows controls & position controls where respone time is not a critical issue.

Media-Oriented System Transport (MOST) BUS: It is targeted for automotive audio/video equipment interfacing, used primarily in European cars. A MOST is a multimedia fibre optic point-to-point network implemented in a star, ring or daisy chained topology over optical fibre cables.

Key players of automotive embedded market system:-
mainly visualised in three verticals namely:
i) Silicon providers
ii) Solution providers
iii) Tool & platform providers

Silicon Providers: Responsible for providing necessary chips which are used in the control application development. The chip may be a Standard product like microcontroller or DSP or ADC / DAC chips. Some applications may require specific chips & they are manufactured as Application Specific Integrated chip (ASIC).
The leading silicon providers in the industry are:

Analog Devices (www.analog.com): providers of world class digital signal processing chips precision and microcontrollers, programmable inclinometer / accelerometer, LED Drivers etc.

Xilinx (www.xilinx.com): suppliers of high performance FPGAs, CPLDs, automotive specific IP cores for GPS navigation systems

Atmel (www.atmel.com): supplier of cost-effective high-density. Flash controllers and memories. Atmel provides a series of high performance microcontroller namely ARM®1, AVR®2, 80C51.
Automotive networking products CAN, LIN & FlexRay are supplied by Atmel.

NXP Semiconductor (www.nxp.com): supplier of 8/16/32 Flash microcontrollers

NEC (www.nec.co.jp): providers of high performance microcontrollers

Solution providers: supply OEM & complete solution for automotive applications making use of the chips, platforms & different development tools. The major players of this domain are as below:-

BOSCH Automotive (www.boschindia.com): Bosch is providing complete automotive solution ranging from body electronics, diesel engine control, gasoline engine control, powertrain systems, safety systems, in-car navigation system & infotainment systems.

DENSO Automotive (www.globaldensoproducts.com) Infosys is a solution provider for automotive embedded hardware & software. Infosys provides the competitive edge in integrating technology change through cost-effective solutions.

Delphi (www.delphi.com): Delphi is the complete solution provider for engine control, safety, infotainment etc. & OEM for spark plugs, bearings.

Tools & Platform Providers: suppliers of various kinds of development tools & Real time Embedded Operating Systems for developing & debugging different control unit related applications.

Categories:

Embedded Software application development tools
Embedded hardware development tools
Sometimes the silicon suppliers provide the development suite for application development using their chip.

ENEA (www.enea.com): Developer of the OSE Real-Time operating system. The OSE RTOS supports both CPU & DSP, also been specially developed to support multi-core & fault-tolerant system development.

The math works (www.mathwork.com): It offers a wide range of tools, consultancy & training for numeric computation, visualization, modelling and simulation across many different industries. math work's breakthrough product is MATLAB - a high level programming language & environment for technical computation & numerical analysis.

SIMULINK, stateflow and Real-Time workshop provide top quality tools for data analysis, test & measurement, application development & deployment, image processing & development of dynamic and reactive systems for DSP & control applications.

Keil Software: It is a powerful embedded software design tool for 8051 & C166 family of microcontrollers.

Lauterbach (http://www.lauterbach.com/ supplier of collaborative modelling tool for requirement analysis, specification, design & development of complex applications.

microsoft (www.microsoft.com): Platform provider for automotive embedded applications microsoft windows CE is a powerful RTOS platform for automotive applications. Automotive features are included in the new win CE version for providing support for automotive application developers.

Module 5

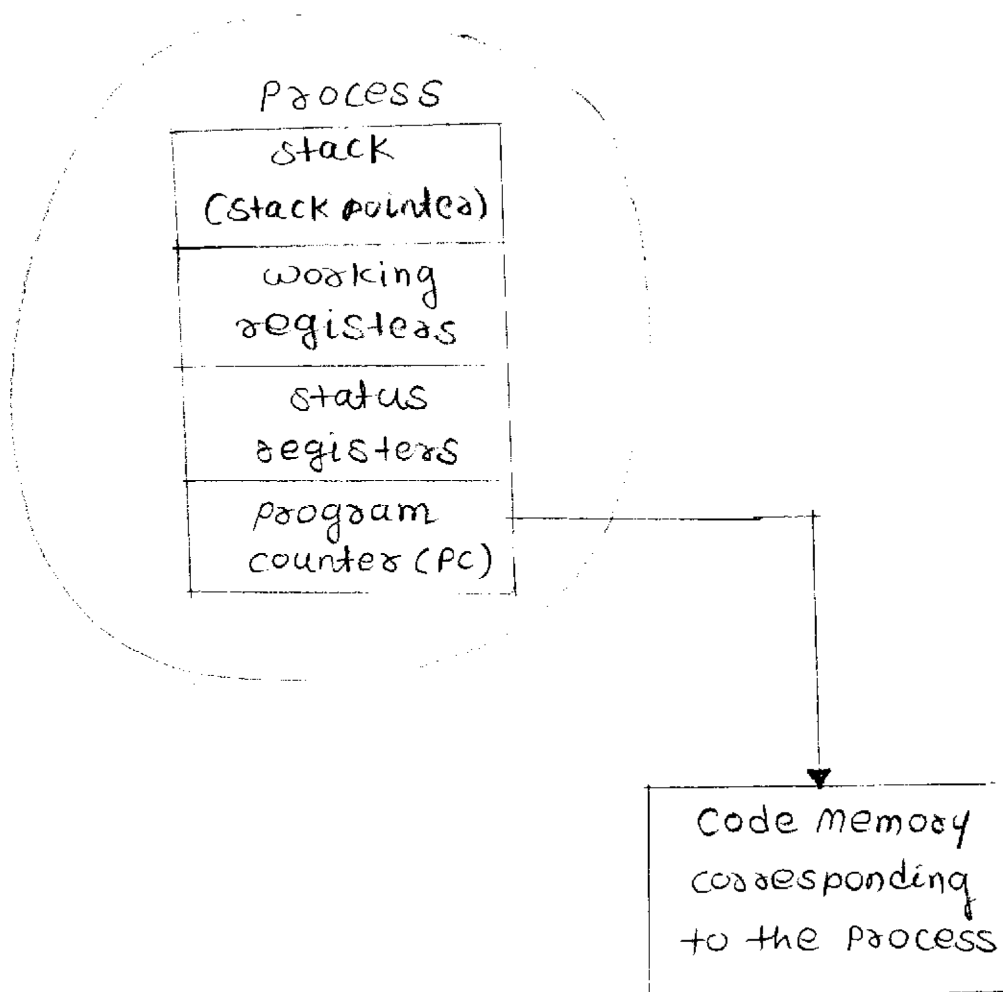
Q.9. a] Explain the concept of task process threads with the help of neat diagram. 10M

→ Task: Task refers to something that needs to be done. In day-to-day life, we are bound to the execution of a number of tasks. we will have an order of priority and schedule / timeline for executing these tasks.

In the OS context, a task is defined as the program in execution & related information maintained by the OS for the program. Also known as job. A program or part of it in execution is also called a 'Process'. The term 'Job', 'Process' & 'Task' refers to the same entity.

Process: A 'Process' is a program, or part of it, in execution. Also known as instance of a program in execution. multiple instances of the same program can execute simultaneously. A process requires various system resources like CPU for executing the process, memory for storing the code corresponding to the process and associated variables, I/O devices for information exchange. A process is sequential in execution.

The structure of a process: The concept of 'process' leads to concurrent execution (pseudo parallelism) of tasks & thereby the efficient utilisation of the CPU and & other system resources. Concurrent execution is achieved through the sharing of CPU among the processes. A process mimics a processor in properties & holds a set of registers process status, a program counter (PC) to point to the next executable instruction of the process.

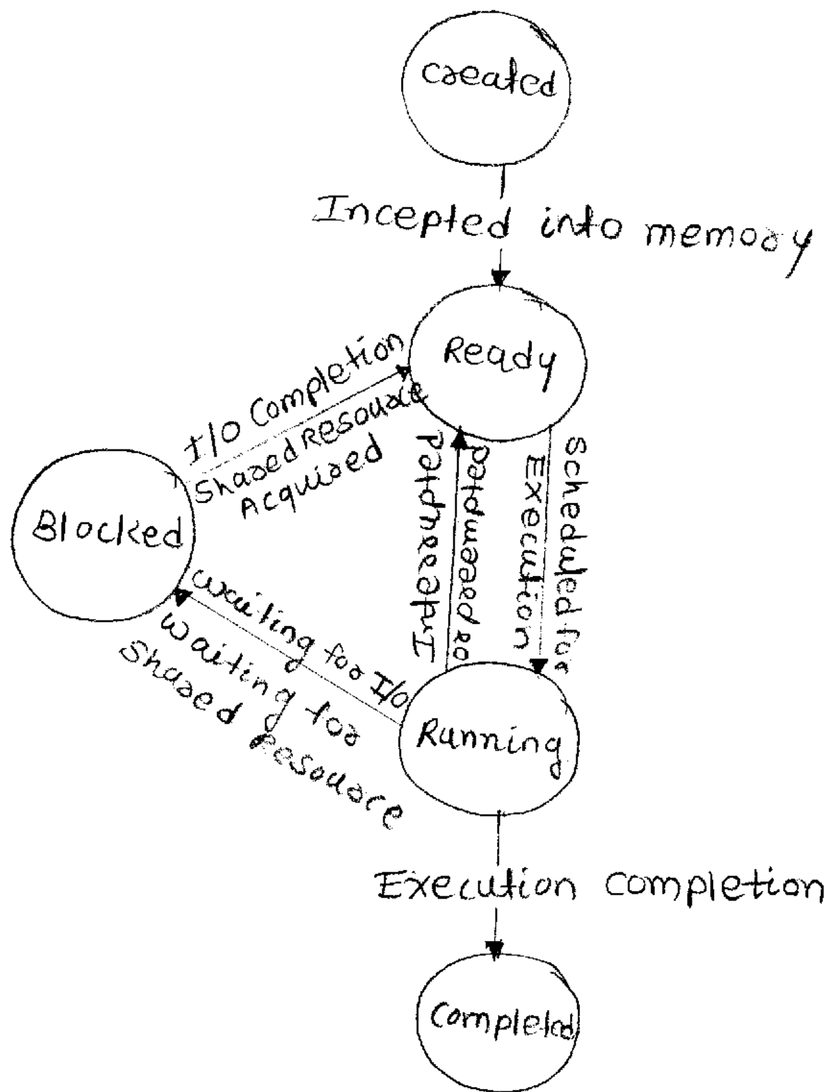


Structure of a process

Process states & State Transition :- The process traverses through a series of states during its transition from the newly created state to the terminated state. The cycle through which a process changes its state from 'newly created' to 'execution completed' is known as 'process Life cycle'.

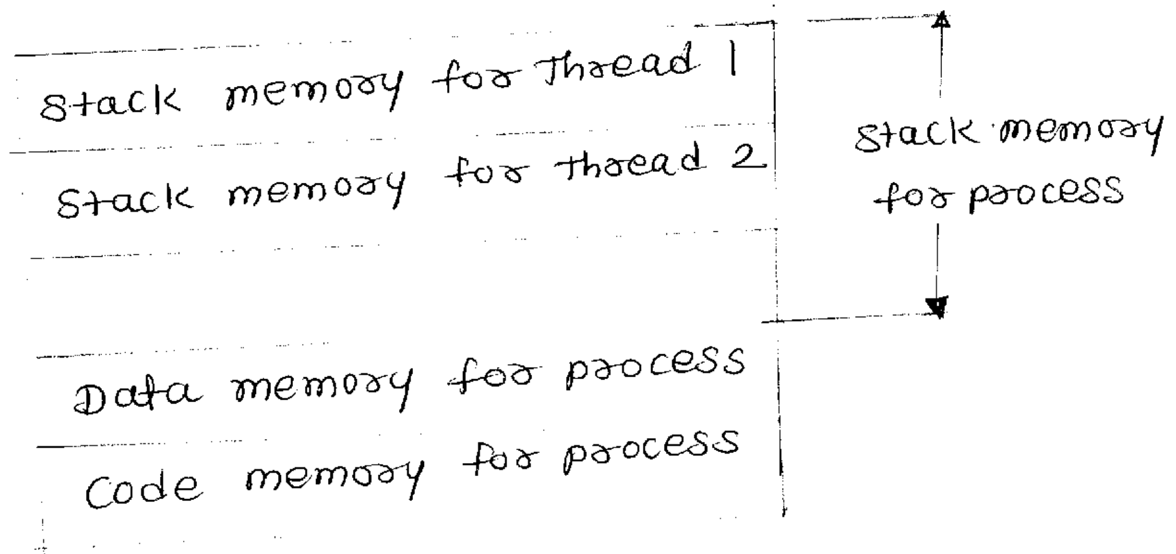
The state indicates the current status of the process with respect to time & also provide information on what it is allowed to do next.

The tansition of a process from one state to another is known as 'state transition'. when process changes its state from Ready to running or from running to block, the CPU allocation for the process may also changes.



Process States & State transition representation

Threads: A thread is primitive that can execute code. A thread is a single sequential flow of control within a process. 'Thread' is also known as light weight process. A process can have many threads of execution. Different threads which are part of a process, share the same address space; meaning they share the data memory, code memory and heap memory area. Threads maintain their own thread status (CPU register values), program counter (PC) and stack. The memory model for a process and its associated threads are as shown in figure:



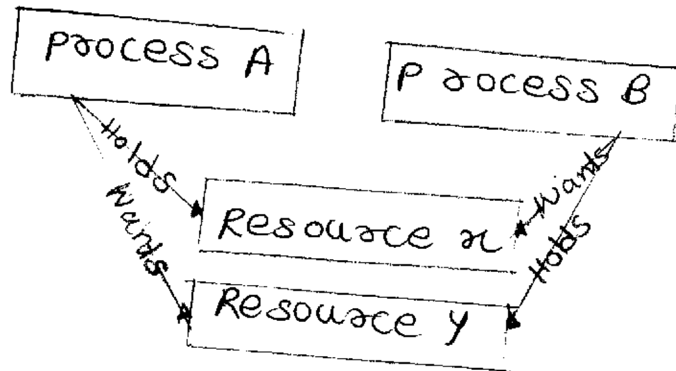
Memory Organisation of a process & its Associated Threads

Multithreading :- A process / task in embedded application may be a complex or lengthy one & it may contain various suboperations like getting input from I/O devices connected to the processor performing some internal calculations, updating some I/O devices etc. In all subfunctions of a task are executed in sequence, the CPU utilization may not be efficient. Instead of this single sequential execution of the whole process if the task / process is split into different threads carrying out the different subfunctionalities of the process, the CPU can be effectively utilized & when the thread corresponding to I/O operation enter the wait state, another thread which do not require I/O event for their operation can be switched into execution.

This helps in speedy execution of the process & efficient utilization of the processor time & resources

Q. 9. b] Explain the concept of Dead lock & Dining Philosopher's problem. 10M

→ **Deadlock**: A race condition produces incorrect results whereas a deadlock condition creates a situation where none of the processes are able to make any progress in their execution, resulting in a set of deadlocked processes. A situation very similar to our traffic jam issues in a junction.

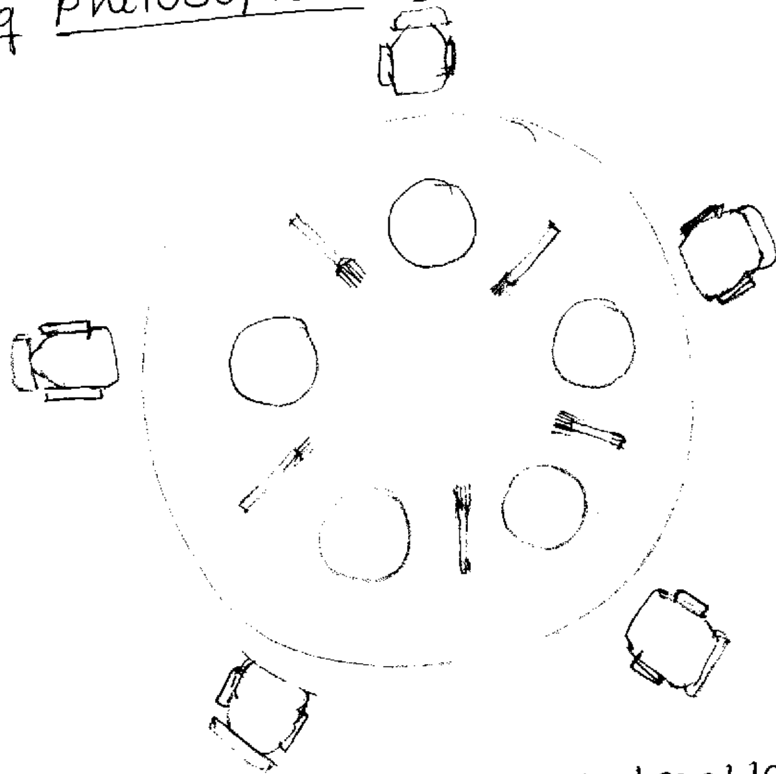


Deadlock is condition in which a process is waiting for resource held by another process which is waiting for a resource held by first process.

Process A holds resource x & it wants a resource y hold by process B. Process B is currently holding resource y & it wants resource x which is held by process A. Both hold the respective resources & they ~~complete~~ compete each other to get resource held by the respective processes.

The result of competion is 'deadlock'. None of the competing process will be able to access the resources held by other processes since they are locked by the respective processes. Conditions favouring deadlock: Mutual Exclusion, Hold & wait No Resource Preemption & circular wait.

Dining philosophers problem :-



The 'Dining' philosophers 'problem'

It is an example for synchronization issues in
resource utilisation. The term 'dining', 'philosophers'
etc. may sound awkward in OS context, but it is
the best way to explain technical things abstractly
using non-technical terms.
Five philosophers (It can be 'n') are sitting around a
round table, involved in eating & brainstorming.
At any point of time each philosopher will be in
any of the three ~~sets~~ states : eating, hungry &
brainstorming. For eating, each philosopher
requires 2 forks. There are only 5 forks available
on the dining table) and they are arranged in
fashion one fork is between two philosophers
The philosopher can only use the forks on his/her
immediate left and right that too in the order
pickup the left fork first & then right fork.
Let's analyze various scenarios that may
occur in this situation.

Scenario 1 : All the philosophers involve in brain-storming together & try to eat together. Each philosopher picks up the left fork and is unable to proceed since two forks are required for eating the spaghetti present in the plate. Philosopher 1 thinks that philosopher 2 sitting to the right of him/her will put the fork down & waits for it. Philosopher 2 think Philosopher 3 sitting to right of him/her will put the fork down & wait for it. and so on. This forms a circular chain of un-guarded requests. If philosopher continue in this state waiting for the fork from the philosopher sitting to the right of each, they will not make any progress in eating & this will result in starvation of the philosophers & deadlock.

Scenario 2 : All the philosophers start brainstorming together. One of the philosophers is hungry, and he/she pick up the left fork. When the philosopher is about to pick up the right fork, the philosopher sitting his right also become hungry & tries to grab the left fork which is the right fork of his neighbouring philosopher who is trying to lift it resulting in 'Race condition'

We need to find out alternative solutions to avoid the deadlock, livelock, racing and starvation condition that may arise due to the concurrent access of forks by philosophers. This situation can be handled in many ways by allocating the fork in different allocation techniques including Round Robin allocation or FIFO allocation etc.

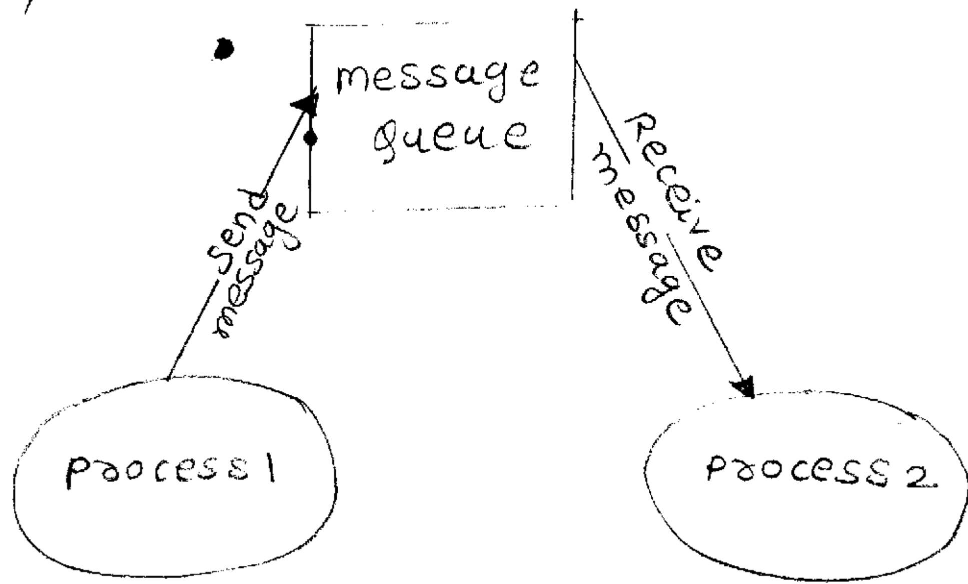
Q.10 a] Explain message passing concept with neat diagram. 10 M

→ message passing is an (a) synchronous information exchange mechanism used for process / thread communication.

The major difference between shared memory & message passing technique is that, through shared memory lots of data can be shared whereas only limited amount of info / data is passed through message passing. Also message passing is free from the synchronization overheads compared to shared memory.

message passing is relatively fast compared to shared memory.

Based on message passing operation between the processes message passing is classified as: i) message queue
ii) Mailbox



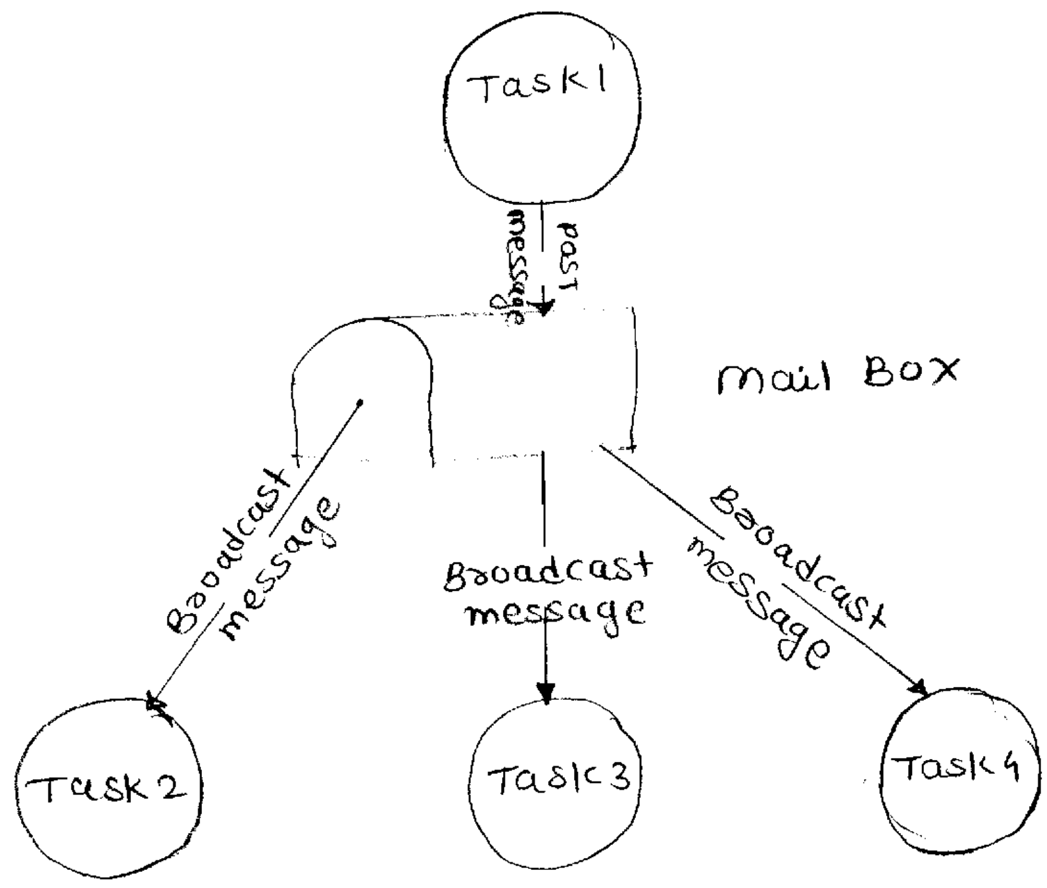
Concept of message queue based messaging for IPC.

<u>Message Queue</u> :- Usually the process which wants to talk to another process posts the message to a First-In-First-Out (FIFO) queue called '<u>Message Queue</u>', which stores the messages temporarily in a system defined memory object, to pass it to the desired process. Messages are sent and received through send (Name of the process to which the message is to be sent) and received through send (Name of the process from which the message is to be received, message) methods. The messages are exchanged through a message queue. The implementation of message queue send and receive methods are OS kernel dependent. The windows XP OS kernel maintains a single system message queue and one process / thread. A thread which wants to communicate with another thread posts the message to the system message queue.

The kernel picks up the message from the system message queue one at a time and examines the message for finding the destination thread, and then post the message to the system message queue.

<u>Mailbox</u> :- Mailbox is an alternate form of 'message queues' and it is used in certain Real Time Operating systems for IPC. mailbox technique for IPC in RTOS is usually used for one way messaging.

The task / thread which wants to send a message to other tasks / threads creates a mailbox for posting the messages. The threads which are interested in receiving the messages posted to the mailbox by the mailbox creator thread can subscribe to the mailbox. The thread which creates the mailbox is known as 'mailbox server' and the threads which subscribe to the mailbox are known as 'mailbox clients'. The mailbox server post messages to the mailbox and notifies it to the clients which are subscribed to the mailbox. The mailbox creation, subscription, messages reading and writing are a achieved through OS kernel provided API calls.



concept of mailbox based indirect messaging for IPC

Q. 10. b] What is semaphore? Explain binary semaphore concept with supporting diagram. 10 m

→ Semaphore :- Semaphore is a sleep and wakeup based mutual exclusion implementation for shared resource access. Semapore is system resource and process which wants to access the shared resource can first acquire this system object to indicate the other processes which wants the shared resource that the shared resource is currently acquired by it.
The display device of an embedded system is a typical example for the shared resource which needs exclusive access by a process.

Binary semaphore (Mutex): It is a synchronization object provided by OS for process / thread synchronisation. Any process / thread can create a 'mutex object' and other processes / threads of the system can use this 'mutex object' for synchronising the access to critical sections. Only one process / thread can own the 'mutex object' at a time. The state of mutex object is set to signalled when it not owned by any process or thread and set to non-signalled when it is owned by any process / thread.

Example:
For the mutex concept is Hotel accommodation system (lodging system)
Diagram on next page.

THE CONCEPT OF BINARY SEMAPHORE (MUTEX)