UNIT I:

BLOOM'S LEVEL 2: UNDERSTAND

1. DIFFRENTIATE BETWEEN MICROPROCESSORS AND MICROCONTROLLERS WITH A NEAT BLOCK DIAGRAM.

Sr. No	*Microprocessor*	*Microcontroller*
1.	We need to connect peripherals externally. So it makes circuit bulky.	The presence of peripherals such as RAM, ROM, Input-output, and Timers are In-built. So It is available on a single chip.
2.	It increases the overall cost of the system high.	The overall cost of the system is less.
3.	We can connect external memory in ranges of Mbytes and even Gbytes. But speed is less.	The inbuilt finite memory helps to improve the speed of operations.
4.	You can't use it in a compact system.	You can use it in compact systems.
5.	Due to external components, the total power consumption is high. Therefore, it is not ideal for the devices running on stored power like batteries.	As external components are low, total power consumption is less. So it can be used with devices running on stored power like batteries.
6.	Most of the microprocessors do not have power-saving features.	Most of the microcontrollers offer power-saving mode.
7.	The microprocessor has a smaller number of registers, so more operations are memory-based.	The microcontroller has more register. Hence the programs are easier to write.
8.	These are based on the von Neumann model where program and data are stored in the same memory module.	These are based on Harvard architecture where program memory and data memory are separate.
9.	It is a central processing unit on a single silicon-based integrated chip.	It is a byproduct of the development of microprocessors with a CPU along with other peripherals.
10	It uses an external bus to interface to RAM, ROM, and other peripherals.	It uses an internal controlling bus.
11	Microprocessor-based systems can run at a very high speed because of the technology involved.	Microcontroller based systems run up to 200MHz or more depending on the architecture.
12	It's useful for general purpose applications that allow you to handle loads of data.	It's useful for application-specific systems.
13	It's complex and expensive, with a large number of instructions to process.	It's simple and inexpensive with less number of instructions to process.

2. LIST AND EXPLAIN THE FOUR MAJOR DESIGN RULES OF RISC PHILOSOPHY.

1. **Instructions**

- Reduced number of instruction classes.
- Each class provides simple operations that can each execute in a single cycle.
- Complicated instructions are synthesized by combining simpler instructions. (for example, a divide operation)
- Each instruction has fixed length (to fetch future instructions before decoding the current instruction.)

Note: In CISC –variation in length and take no. of cycles for execution.

2. **Pipelines:**

– The processing of instructions 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.

Note: In CISC – An instruction is executed by a miniprogram called microcode.

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.

Note: In CISC – dedicated registers for specific purposes.

4. **Load-store architecture:**

– The processor operates on data held in registers.

– Separate load and store instructions transfer data between the register bank and external memory.

– As memory accesses are costly, they perform operations only on register data.

Note: In CISC – data processing operations can act on memory directly.

3. DIFFERTIATE BETWEEN RISC AND CISC PROCESSORS.

Comparison of CISC and RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy

4. LIST AND EXPLAIN IN DETAIL THE ARM DESIGN PHILOSOPHY.

The ARM Design Philosophy

• Physical features of ARM processor design:

1. **Power:** The ARM processor has been specifically designed to be small to reduce power consumption and extend battery operation that is essential for applications such as mobile phones and personal digital assistants (PDAs).
2. **High Code Density:** Major requirement since embedded systems have limited memory due to cost and/or physical size restrictions. It is useful for applications that have limited on-board memory, such as mobile phones and mass storage devices.
3. **Price sensitive:** Use slow and low-cost memory devices. For high-volume applications like digital cameras, every cent has to be accounted for in the design. The ability to use low-cost memory devices produces substantial savings.
4. **Area:** Another important requirement is to reduce the area of the die taken up by the embedded processor. The smaller the area used by the embedded processor, the more available space for specialized peripherals. It also reduces the cost of the design and manufacturing.
5. **Use of hardware debug technology** : Incorporated within the processor.

5. JUSTIFY WHY ARM INSTRUCTION SET IS SUITABLE FOR EMBEDDED APPLICATIONS.

Instruction Set for Embedded Systems:

The ARM instruction set differs from the pure RISC definition in following ways.

- **Variable cycle execution** for certain instructions—Not every ARM instruction executes in a single cycle.
- Example: **load-store-multiple** instructions vary in the number of execution cycles depending upon the number of registers being transferred. The transfer can occur on sequential memory addresses, which increases performance. **Code density** is also improved since multiple register transfers are common.
- **Inline barrel shifter leading to more complex instructions**—The inline barrel shifter is a hardware component that preprocesses one of the input registers before it is used by an instruction. This expands the capability of many instructions to improve core performance and code density.

(used to shift and rotate n-bits , typically within a single clock cycle)

- **Thumb 16-bit instruction set**—ARM enhanced the processor core by adding a second 16-bit instruction set called Thumb that permits the ARM core to execute either 16- or 32-bit instructions. The 16-bit instructions improve code density by about 30% over 32-bit fixed-length instructions.

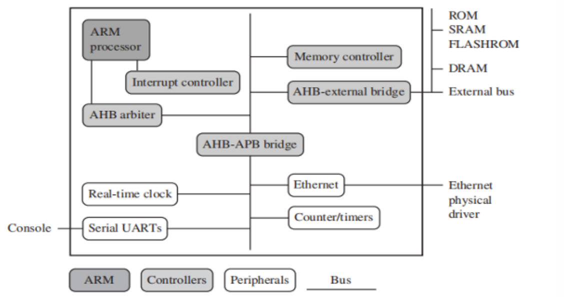
- **Conditional execution**—An instruction is only executed when a specific condition has been satisfied. This feature improves performance and code density by reducing branch instructions.

- **Enhanced instructions**—The enhanced digital signal processor (DSP) instructions were added to the standard ARM instruction set to support fast 16×16-bit multiplier operations and saturation. These instructions allow a faster-performing ARM processor in some cases to replace the traditional combinations of a processor plus a DSP.

6. WITH A NEAT BLOCK DIAGRAM OF AN ARM-BASED EMBEDDED DEVICE, EXPLAIN THE FOLLOWING:

- ARM PROCESSOR
- CONTROLLERS
- PERIPHERALS
- BUS

An ARM-based embedded device : A microcontroller.



Each box represents a feature or function. The lines connecting the boxes are the buses carrying data.

Embedded System Hardware : There are four main hardware components:

- **Core:** An ARM processor comprises a core (the execution engine that processes instructions and manipulates data) plus the surrounding components that interface it with a bus. These components can include memory management and caches.
- **Controllers** coordinate important functional blocks of the system. Two commonly found controllers are interrupt and memory controllers.
- **The peripherals** provide all the input-output capability external to the chip and are responsible for the uniqueness of the embedded device.
- **A bus** is used to communicate between different parts of the device.

ARM Bus Technology

- **Peripheral Component Interconnect (PCI) bus :** Most common PC bus technology, which connects devices such as video cards and hard disk controllers to the x86 processor bus. This type of technology is external or off-chip and is built into the motherboard of a PC.
- In contrast, **embedded devices** use an on-chip bus that is internal to the chip and that allows different peripheral devices to be interconnected with an ARM core.

Two different classes of devices attached to the bus.
The ARM processor core is a bus master—a logical device capable of initiating a data transfer with another device across the same bus.
Peripherals tend to be bus slaves—logical devices capable only of responding to a transfer request from a bus master device.

7. WRITE A NOTE ON THE FOLLOWING:

- ARM BUS TECHNOLOGY
- AMBA BUS PROTOCOL
- MEMORY
- PERIPHERALS

ARM Bus Technology

- **Peripheral Component Interconnect (PCI) bus :** Most common PC bus technology, which connects devices such as video cards and hard disk controllers to the x86 processor bus. This type of technology is external or off-chip and is built into the motherboard of a PC.
- In contrast, **embedded devices** use an on-chip bus that is internal to the chip and that allows different peripheral devices to be interconnected with an ARM core.

Two different classes of devices attached to the bus.
The ARM processor core is a bus master—a logical device capable of initiating a data transfer with another device across the same bus.
Peripherals tend to be bus slaves—logical devices capable only of responding to a transfer request from a bus master device.

Advanced Microcontroller Bus Architecture (AMBA) Bus Protocol: A bus has two architecture levels.

- Physical level that covers the electrical characteristics and bus width (16, 32, or 64 bits).
- Second level deals with protocol—the logical rules that govern the communication between the processor and a peripheral.

(AMBA) is widely adopted as the on-chip bus architecture.

- **ARM System Bus (ASB) (**for external peripherals and requires a bridge).
- **ARM Peripheral Bus (APB)(**for the slower peripherals).
- **ARM High Performance Bus (AHB)(**for high performance peripherals)

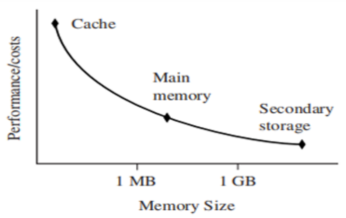
Note: Using AMBA, peripheral designers can reuse the same design on multiple projects.

Memory

An embedded system has to have some form of memory to store and execute code.
Memory characteristics are hierarchy, width, and type. To decide, compare price, performance, and power consumption.

Hierarchy

Figure 1.2 shows a device that supports external off-chip memory. Internal to the processor there is an option of a cache (not shown in Figure 1.2) to improve memory performance.



Types of memory

- Read-only memory (ROM) is the least flexible of all memory types because it cannot be reprogrammed. Many devices also use a ROM to hold **boot code**.
- Dynamic random access memory (DRAM) is the most commonly used RAM for devices. It has the **lowest cost per megabyte** compared with other types of RAM. DRAM is dynamic— it needs to have its storage cells **refreshed . S**o you need to set up a DRAM controller before using the memory.
- Static random access memory (SRAM) is **faster** than the more traditional DRAM, but requires more **silicon area**. SRAM is static— the RAM **does not require refreshing**.
- Synchronous dynamic random access memory (SDRAM) is one of many subcategories of DRAM. It can run at **much higher clock speeds** than conventional memory. SDRAM synchronizes itself with the processor bus because it is clocked.

Peripherals : A peripheral device performs input and output functions for the chip by connecting to other devices or sensors that are off-chip. Each peripheral device usually performs a single function and may reside on-chip.

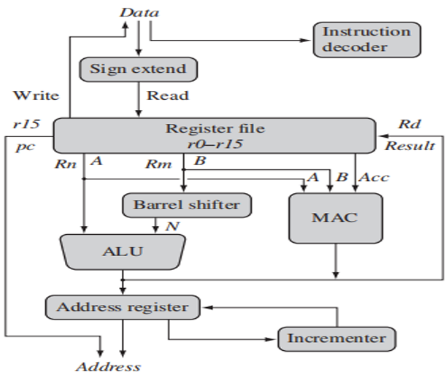
Examples: Peripherals range from a simple serial communication device to a more complex 802.11 wireless device.

- **Controllers are specialized peripherals** that implement higher levels of functionality within an embedded system.
- Two important types of controllers are **memory controllers and interrupt controllers.**

8. WITH A NEAT BLOCK DIAGRAM EXPLAIN THE ARM CORE DATA FLOW MODEL.

ARM Processor Fundamentals: ARM core dataflow model

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



1-3

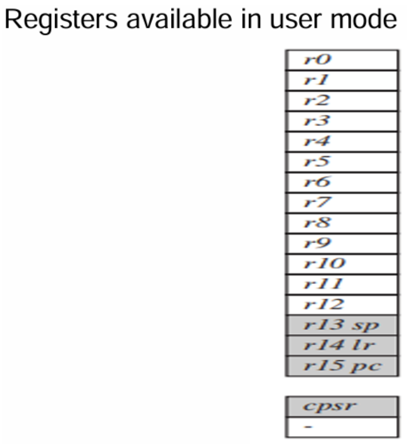
Abstract components of an ARM core:

- Data enters the processor core through the Data bus. The data may be an instruction to execute or a data item.
- Data items and instructions share the same bus (Von Neumann impl). In contrast, Harvard implementations of the ARM use two different buses.
- Instruction decoder translates instructions before they are executed. Each instruction executed belongs to a particular instruction set.
- The ARM processor, like all RISC processors, uses a load-store architecture. Load instructions copy data from **memory to registers** in the core, and conversely the store instructions copy data from **registers to memory**.
- **Note:** There are no data processing instructions that directly **manipulate data in memory.** Thus, data processing is carried out solely in registers.

- Data items are placed in the register file—a storage bank made up of 32-bit registers.
- ARM core is a 32-bit processor, most instructions treat the registers as holding signed or unsigned 32-bit values. The sign extend hardware converts signed 8-bit and 16-bit numbers to 32-bit values as they are read from memory and placed in a register.
- ARM instructions typically have two source registers, Rn and 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.
- The ALU (arithmetic logic unit) or MAC (multiply-accumulate unit) takes the register values Rn and Rm from the A and B buses and computes a result. Data processing instructions write the result in Rd directly to the register file.

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

9. LIST OUT THE VARIOUS REGISTERS OF ARM 7. COMMENT ON ITS WIDTH, AND SPECIAL PURPOSE OF REGISTERS R13, R14 AND R15.



There are up to 18 active registers: 16 **data registers** and 2 **processor status registers**. The data registers are visible to the programmer as r0 tor15.

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

In above Figure the shaded registers identify the assigned special-purpose registers:

■ Registerr13 is traditionally used as the stack pointer (sp) and stores the head of the stack in the current processor mode.

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

■ Registerr15 is the program counter (pc) and contains the address of the next instruction to be fetched by the processor.

Depending upon the context, registers r13 and r14 can also be used as general-purpose registers, which can be particularly useful since these registers are banked during a processor mode change. However, it is dangerous to use r13 as a general register when the processor is running any form of operating system because operating systems often assume that r13 always points to a valid stack frame.

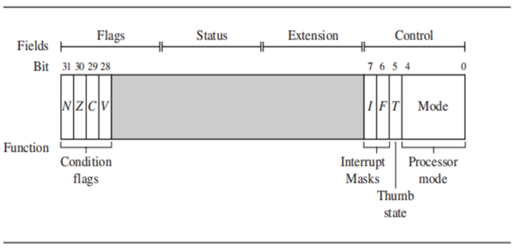
In ARM state the registers r0 to r13 are orthogonal—any instruction that you can apply to r0 you can equally well apply to any of the other registers. However, there are instructions that treat r14 and r15 in a special way.

In addition to the 16 data registers, there are two program status registers: cpsr and spsr (the current and saved program status registers, respectively).

The register file contains all the registers available to a programmer. Which registers are visible to the programmer depend upon the current mode of the processor.

10. DRAW THE NEAT BLOCK DIAGRAM OF CPSR AND COMMENT ON THE SIGNIFICANCE OF **N, Z, C AND V** FLAGS?

Current Program Status Register



The ARM core uses the cpsr to monitor and control internal operations. The cpsr is a dedicated 32-bit register and resides in the register file. Figure above shows the basic layout of a generic program status register.

The CPSR is divided into four fields, each 8 bits wide: flags, status, extension, and control.
In current designs the extension and status fields are reserved for future use. The control field contains the processor mode, state, and interrupt mask bits. The flags field contains the condition flags.

11. LIST THE VARIOUS MODES OF OPERATION OF ARM 7.

Processor Modes : The processor mode determines which registers are active and the access rights to the cpsr register itself. Each processor mode is either privileged or nonprivileged.
Privileged mode : It allows full read-write access to the cpsr.
Nonprivileged mode : It only allows read access to the control field in the cpsr but still allows read-write access to the condition flags.

Seven processor modes: 6 privileged modes (abort, fast interrupt request, interrupt request, supervisor, system, and undefined) and one nonprivileged mode (user).

Introduction 1-47

- **Abort mode:** The processor enters abort mode when there is a failed attempt to access memory.
- **Fast interrupt request and interrupt request modes** correspond to the two interrupt levels available on the ARM processor.
- **Supervisor mode** is the mode that the processor is in after reset and is generally the mode that an o.s kernel operates in.
- **System mode** is a special version of user mode that allows full read-write access to the cpsr.
- **Undefined mode** is used when the processor encounters an instruction that is undefined or not supported by the implementation.
- **User mode** is used for programs and applications.

12. DEFINE PIPELINE. HOW MANY STAGES OF PPELINE IS AVAILABLE FOR ARM7. ILLUSTRATE THE PIPELINE OPERATION FOR THE FOLLOWING INSTRUCTIONS:

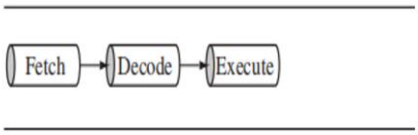
a. ADD R0,R1,R2

b. AND R3,R4,R5

c. SUB R6,R7,R8

Pipeline: A pipeline is the mechanism 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. One way to view the pipeline is to think of it as an automobile assembly line, with each stage carrying out a particular task to manufacture the vehicle.

ARM7 Three-stage pipeline



Introduction 1-6

The three instructions are placed into the pipeline sequentially.
In the first cycle the core fetches the ADD instruction from memory.
In the second cycle the core fetches the SUB instruction and decodes the ADD instruction.
In the third cycle, both the SUB and ADD instructions 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 the 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. This in turn increases the performance. The system latency also increases because it takes more cycles to fill the pipeline before the core can execute an instruction. The increased pipeline length also means there can be data dependency between certain stages.

Introduction 1-62

Figure above shows a three-stage pipeline:

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

Example :

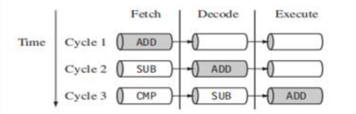
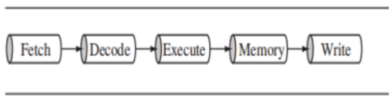
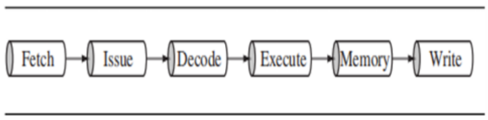


Figure shows a sequence of three instructions being fetched, decoded, and executed by the processor. Each instruction takes a single cycle to complete after the pipeline is filled.

ARM9 five-stage pipeline



ARM10 six-stage pipeline



BLOOM'S LEVEL 3: APPLY

1. WHICH OF THE FOLLOWING STATEMENTS ARE TRUE WITH RESPECT TO ARM 7 ARCHITECTURE.

   a. EACH PROCESSOR MODE IS EITHER PREVILEGED OR NONPREVILEGED.

   b. PREVILEGED MODE ALLOWS FULL READ WRITE ACCESS TO THE CPSR.

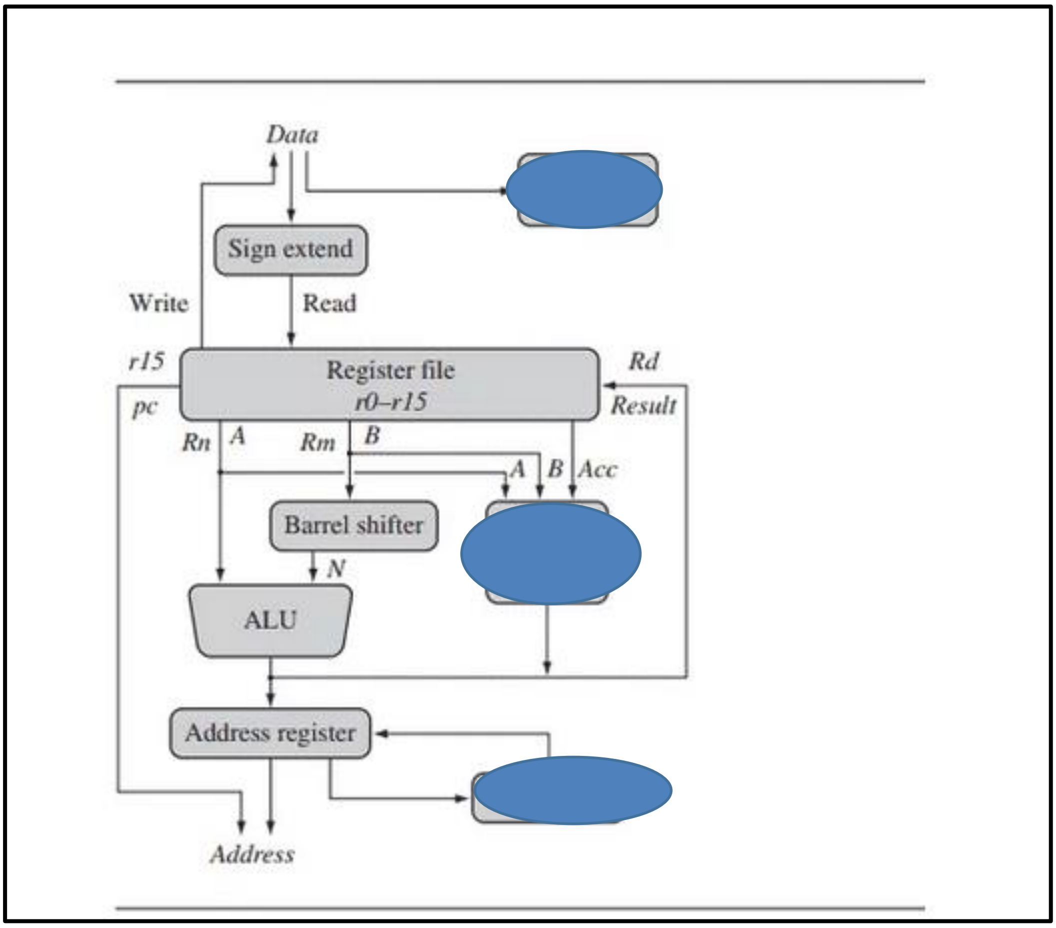
   c. THE NEGATIVE FLAG 'N' IS SET WHEN BIT 31 OF THE RESULT IS BINARY 1.

   d. THE ZERO FLAG 'Z' IS USED TO INDICATE EQUALITY.

   e. THE CARRY FLAG 'C' IS SET WHEN THE RESULT CAUSES AN UNSIGNED CARRY.

   f. THE OVERFLOW FLAG 'V' IS SET WHEN THE RESULT CAUSES SIGNED OVERFLOW.

BLOOM'S LEVEL 4: ANALYZE: ANALYZE THE ARM CORE DATAFLOW MODEL SHOWN IN FIGURE BELOW AND IDENTIFY THE MASKED BLOCKS AND THEIR SIGNIFICANCE.



UNIT 2:

BLOOM'S LEVEL 2: UNDERSTAND

1. LIST AND EXPLAIN THE VARIOUS DATA TRANSFER INSTRUCTIONS OF ARM7 WITH PROPER SYNTAX AND AN EXAMPLE.

2. WITH A NEAT BLOCK DIAGRAM EXPLAIN THE SIGNIFICANCE OF BARRE SHIFTER AND ALU.

3. LIST AND EXPLAIN THE BASIC 'C' DATA TYPES.

4. LIST AND EXPLAIN THE FOLLOWING INSTRUCTIONS OF ARM7 WITH PROPER SYNTAX AND AN EXAMPLE FOR EACH.

   a. SHIFT INSTRUCTIONS

   b. ROTATE INSTRUCTIONS

   c. ARITHMETIC INSTRUCTIONS

   d. LOGICAL INSTRUCTIONS

e. COMPARISON INSTRUCTIONS

f. MULTIPLY INSTRUCTIONS

g. BRANCH INSTRUCTIONS

h. LOAD STORE INSTRUCTIONS

i. SWAP INSTRUCTION

j. PROGRAM STATUS REGISTER INSTRUCTIONS

BLOOM'S LEVEL 3: APPLY

1. DEVELOP AN ASSEMBLY LANGUAGE PROGRAM (ALP) TO PERFORM BLOCK DATA TRANSFER.

2. DEVELOP AN ALP TO GENERATE THE SERIES: 5, 10,15,20,25. HINT: USE MLA INSTRUCTION.

3. DEVELOP AN ALP TO COMPUTE THE FACTORIAL OF A GIVEN NUMBER AND STORE THE RESULT IN RAM LOCATION.

4. DEVELOP AN ALP FIND THE LARGEST NUMBER IN AN ARRAY AND STORE IT IN RAM LOCATION.

5. DEVELOP AN ALP TO ILLUSTRATE THE SIGNIFICANCE OF LOGICAL OPERATIONS.

6. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF SHIFT AND ROTATE INSTRUCTIONS.

7. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF SWAP INSTRUCTION.

8. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF LOAD STORE INSTRUCTIONS

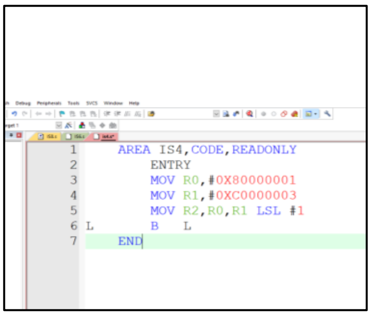
9. DEVELOP AN ALP TO ILLUSTRATE THE WORKING OF PROGRAM STATUS REGISTER INSTRUCTIONS

BLOOM'S LEVEL 4: ANALYZE:

1. ANALYZE THE GIVEN PIECE OF CODE AND ANSWER THE FOLLOWING:

a. WHAT IS THE CONTENT OF R0,R1 AND R2.

b. COMMENT ON THE STATUS OF NZCV FLAGS AFTER EXECUTING THE LAST INSTRUCTION.



2. ANALYZE THE GIVEN PIECE OF CODES ('C' CODE AND COMPILER OUTPUT) AND ANSWER THE FOLLOWING:

```c
int checksum_v1(int *data)
{
  char i;
  int sum=0;

  for (i=0; i<64; i++)
  {
    sum += data[i];
  }
  return sum;
}
```

```
checksum_v1
        MOV     r2,r0               ; r2 = data
        MOV     r0,#0               ; sum = 0
        MOV     r1,#0               ; i = 0
checksum_v1_loop
        LDR     r3,[r2,r1,LSL #2]   ; r3 = data[i]
        ADD     r1,r1,#1            ; r1 = i+1
        AND     r1,r1,#0xff         ; i = (char)r1
        CMP     r1,#0x40            ; compare i, 64
        ADD     r0,r3,r0            ; sum += r3
        BCC     checksum_v1_loop    ; if (i<64) loop
        MOV     pc,r14              ; return sum
```

- WHAT IS THE DRAWBACK OF USING CHAR DATA TYPE FOR DECLARING THE LOCAL VARIABLES IN ARM7 'C' PROGRAM?
- IN THE COMPILER OUTPUT HOW CAN WE AVOID THE INSTRUCTION *AND R1,R1,#0XFF*
- WHAT IS THE USE OF BCC INSTRUCTION?
- WHY PC IS UPDATED WITH R14 CONTENT? CAN WE REPLACE R14 BY ANY OTHER REGISTER?

3. ANALYZE THE GIVEN PIECE OF CODE AND ANSWER THE FOLLOWING:

- WHICH DATA TYPE IS USED TO DECLARE THE LOCAL VARIABLE?
- WHAT IS THE MODIFICATION THAT IS REQUIRED IN THE COMPILER OUTPUT IF THE VARIABLE SUM IS 16-BIT.

```
checksum_v2
        MOV     r2,r0               ; r2 = data
        MOV     r0,#0               ; sum = 0
        MOV     r1,#0               ; i = 0
checksum_v2_loop
        LDR     r3,[r2,r1,LSL #2]   ; r3 = data[i]
        ADD     r1,r1,#1            ; r1++
        CMP     r1,#0x40            ; compare i, 64
        ADD     r0,r3,r0            ; sum += r3
        BCC     checksum_v2_loop    ; if (i<64) goto loop
        MOV     pc,r14              ; return sum
```

4. ANALYZE THE GIVEN PIECE OF CODES ('C' CODE AND COMPILER OUTPUT) AND ANSWER THE FOLLOWING:

```
short checksum_v3(short *data)
{
 unsigned int i;
 short sum=0;

 for (i=0; i<64; i++)
 {
   sum = (short)(sum + data[i]);
 }
 return sum;
}
```

```
checksum_v3
        MOV     r2,r0              ; r2 = data
        MOV     r0,#0              ; sum = 0
        MOV     r1,#0              ; i = 0
checksum_v3_loop
        ADD     r3,r2,r1,LSL #1    ; r3 = &data[i]
        LDRH    r3,[r3,#0]         ; r3 = data[i]
        ADD     r1,r1,#1           ; i++
        CMP     r1,#0x40           ; compare i, 64
        ADD     r0,r3,r0           ; r0 = sum + r3
        MOV     r0,r0,LSL #16
        MOV     r0,r0,ASR #16      ; sum = (short)r0
        BCC     checksum_v3_loop   ; if (i<64) goto loop
        MOV     pc,r14             ; return sum
```

- HOW CAN WE REDUCE THESE INSTRUCTIONS IN THE COMPILER OUTPUT?
  - `ADD r3,r2,r1,LSL #1`

    `MOV r0,r0,LSL #16`
  - `MOV r0,r0,ASR #16`
- REWRITE THE 'C' CODE TO REDUCE THESE INSTRUCTIONS.

UNIT 3:

LEVEL 2:

1. LIST AND EXPLAIN THE VARIOUS REGISTERS OF ARM 7 USED FOR CONFIGURING PORTS AS GPIO, INPUT/OUTPUT AND SET/CLEAR .
2. WHAT VALUE HAS TO BE LOADED INTO THE REGISTERS
   - TO CONFIGURE PORT 0 (P0.0-P0.15) PINS AS INPUT?
   - TO CONFIGURE PORT 0 (P0.15-P0.31) PINS AS INPUT?
   - TO CONFIGURE PORT 0 (P0.0-P0.15) PINS AS OUTPUT?
   - TO CONFIGURE PORT 0 (P0.15-P0.31) PINS AS OUTPUT?
   - TO CONFIGURE PORT 1 (P1.16-P1.31) PINS AS INPUT?
   - TO CONFIGURE PORT 1 (P1.16-P1.31) PINS AS OUTPUT?

- WHETHER THE PINS P1.0-P1.15 ARE AVAILABLE AS GPIO?

3. LIST AND EXPLAIN THE REGISTERS USED TO CONTROL GPIO REGISTERS(IOPIN, IODIR,IOSET,IOCLR)

LEVEL 3:

1. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO BLINK THE LEDS CONNECTED TO PORT 0 PINS(P0.16-P0.23).
2. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO IMPLEMENT 8-BIT BINARY COUNTER ON PORT 0 PINS (P0.16-P0.23).
3. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO INTERFACE DAC WITH ARM7 TO GENERATE THE FOLLOWING WAVEFORMS:
    - SQUARE WAVE
    - TRIANGULAR WAVE
4. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO INTERFACE THE RELAY WITH ARM7.
5. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO BLINK THE BUILTIN LED CONNECTED TO PIN NUMBER 5 OF ARDUINO UNO.
6. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO INTERFACE LDR SENSOR CONNECTED TO PIN NUMBER 13 OF ARDUINO UNO.
7. DEVELOP AN EMBEDDED ‘C’ PROGRAM TO INTERFACE BUZZER CONNECTED TO PIN NUMBER 9 OF ARDUINO UNO.

UNIT 4:

Q1). DEFINE AN EMBEDDED SYSTEM.

An embedded system is an electronic/ electro-mechanical system designed to perform a specific function and is a combination of both hardware and firmware (software). Every embedded system is unique, and the hardware as well as the firmware is highly specialized to the application domain.

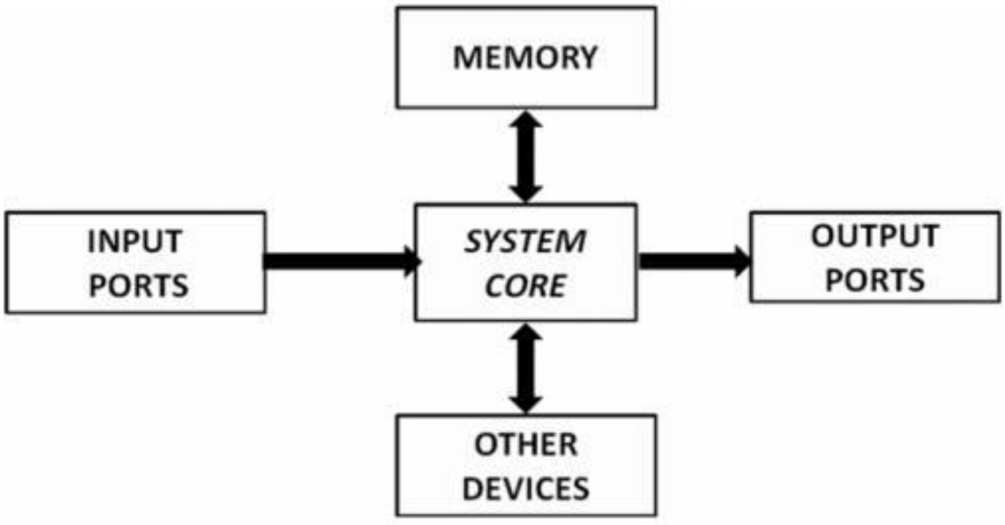


Figure 2.0 : Elements of an Embedded System

Characteristics of Embedded Systems: (if they ask)

- highly reliable and stable.
- have minimal or no user interface.
- are usually feedback oriented or reactive.
- meet real time constraints.
- have limited memory and limited number of peripherals.
- are designed for specific application or purpose.
- Systems are designed for low power consumption, as they use battery power.

Q2). DIFFERENTIATE BETWEEN EMBEDDED SYSTEMS AND GENERAL COMPUTING SYSTEMS.

General Computing System	Embedded System
1. A combination of generic hardware and a General Purpose Operating System (GPOS) for executing a variety of applications.	1. A combination of special purpose hardware embedded OS for executing a specific set of applications.
2. Applications are alterable (programmable) by the user.	2. The firmware is pre-programmed and it is non-alterable by the end-user (there may be exceptions).
3. Performance is the key deciding factor in the selection of the system. Always, 'Faster is Better'.	3. Application-specific requirements (like performance, power requirements, memory usage, etc.).
4. Less/ not at all tailored towards reduced operating power requirements.	4. Highly tailored to take advantage of the power saving modes supported by the hardware and the operating system.
5. Need not be deterministic in execution behavior; response requirements are not time critical.	5. Execution behavior is deterministic for certain types of embedded systems like 'Hard Real Time' systems.

Q3). EXPLAIN THE DOMAINS AND AREAS OF APPLICATIONS OF EMBEDDED SYSTEMS.

The application areas and the products in the embedded domain are countless. A few of the important domains and products are listed below:

1. **Consumer Electronics**: cameras, etc.
2. **Household Appliances**: Television, washing machine, fridge, etc.
3. **Home Automation and Security Systems**: Air conditioners, sprinklers, fire alarms, etc.
4. **Automotive Industry**: engine control, ignition systems, automatic navigation systems, etc.
5. **Telecom:** Cellular telephones, telephone switches, handset multimedia applications, etc.
6. **Computer Peripherals**: Printers, scanners, etc.
7. **Computer Networking Systems**: Network routers, switches, firewalls, etc.

8. **Healthcare**: Different kinds of scanners, EEG, ECG machines, etc.
9. **Measurement & Instrumentation**: Digital multi meters, digital CROs, etc.
10. **Banking & Retail**: Automatic teller machines (ATM) and currency counters.
11. **Card Readers**: Barcode, smart card readers, hand held devices, etc.
12. **Wearable Devices**: Health and fitness trackers, Smartphone screen extension for notifications, etc.
13. **Cloud Computing and Internet of Things (IoT).**

Q4). IDENTIFY AND EXPLAIN THE PURPOSES OF EMBEDDED SYSTEMS.

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

1. Data Collection, Storage, Representation

- Data collection is usually done for storage, analysis, manipulation, and transmission.
- Embedded systems with analog data capture directly collect analog signals, while those with digital data collection convert analog to digital using A/D converters.
- If the data is digital, it can be directly captured by digital embedded system. Ex: A digital camera

2. Data Communication

- Embedded data communication systems are deployed in applications ranging from simple home networking systems to complex satellite communication systems.
    - Network hubs, routers, switches are examples of dedicated data transmission embedded systems.
- Data transmission is in the form of wire medium or wireless medium.
    - USB, TCP/ IP are examples of wired communication;
    - BlueTooth and Wi-Fi are examples for wireless communication.
- Data can be transmitted by analog means or by digital means.

3. Data (Signal) Processing

• are employed in applications demanding signal processing like speech coding, transmission applications, etc. eg: A digital hearing aid.

4. Monitoring

- Almost all embedded products coming under the medical domain are with monitoring functions.
  - Patient heart beat is monitored by Electro cardiogram (ECG) machine.

• Digital CRO, digital multi-meters, and logic analyzers are examples of monitoring embedded systems.

5. Control

• Sensors and actuators are used for controlling the system.

- Sensors are connected to the input port for capturing the changes in environmental variable or measuring variable.
- Actuators connected to output port are controlled according to the changes in input variable.

• Air conditioner system used in our home to control the room temperature to a specified limit.

6. Application Specific User Interface

- Examples: buttons, switches, keypad, lights, bells, display units, etc.
- Mobile phone is an example for this. In mobile phone, the user interface is provided through the keypad, graphic LCD module, system speaker, vibration alert, etc.

Q5) LIST AND EXPLAIN THE ELEMENTS OF A TYPICAL EMBEDDED SYSTEM.

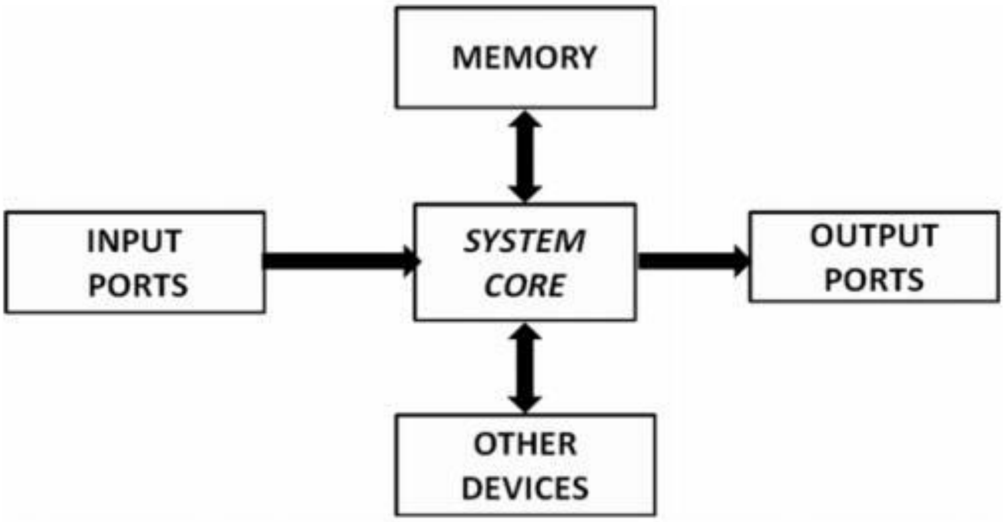


Figure 2.0 : Elements of an Embedded System

The core of the embedded system falls into any one of the following categories:

1) General Purpose and Domain Specific Processors

   a. Microprocessors
   b. Microcontrollers
   c. Digital Signal Processors

2) Application Specific Integrated Circuits (ASICs)

3) Programmable Logic Devices (PLDs)

4) Commercial off-the-shelf Components (COTS)

Embedded system processors:

Microprocessors: General-purpose, execute diverse tasks, serve as the system's brain for computation, control, and communication.

Microcontrollers: Integrated circuits with a microprocessor core, memory, and peripherals. Specialized for specific applications, commonly used in embedded systems.

Digital Signal Processors (DSPs):

A typical digital signal processor incorporates the following four key units:

1. Program Memory

2. Data Memory
3. Computational Engine
4. I/O Unit

- Efficiently process digital signals for tasks like audio, video processing, communications, and signal analysis. Optimized for numerical computations.

Application Specific Integrated Circuits (ASICs):

Custom-designed for specific applications, highly optimized for performance and power efficiency in high-volume production.

Programmable Logic Devices (PLDs):

Includes FPGAs and CPLDs, programmable to implement digital logic circuits, offering flexibility for prototyping and changing requirements.

Commercial Off-the-Shelf Components (COTS):

Pre-manufactured, standardized components like sensors and communication modules, readily available, reducing development time and cost.

Q6). DIFFERENTIATE BETWEEN MICROPROCESSOR AND MICROCONTROLLER.

Microprocessor

1. It is simply the CPU of any general purpose computer.

2. Used for General purpose (Browsing, gaming, documentation, etc)

3. Can be programmed using ALP, Higher level programming languages

4. structure:

CPU	Registers
RAM/ ROM	?

5. Architecture:

Von Neumann architecture (Single memory is used for storing program/instructions & data)

Most of the microprocessors use Von-Neumann architecture (ie data & instructions are stored in same memory).

Microcontroller

1. It is a Complete computer on chip (CPU + I/O devices + memory) So, we call it as System on Chip (SoC).

2. Used for specific purpose applications (eg:- washing machine, Smart TV, Refrigerator, etc.)
specific purpose means their functionality is fixed.

3. Can be programmed using Assembly, Higher level programming languages.

4. Structure:

CPU	I/O devices	Memory
Serial port	Timers/ Counter	Interrupts controller

5. Architecture:

Harvard architecture (Separate memory is used for storing program & data)

↳ It is best bcz fast access is possible.

↑ses performance

Most of the microcontrollers

Q7). DIFFERENTIATE BETWEEN RISC AND CISC.

Differentiate b/w RISC & CISC

RISC	CISC
1) Reduced Instruction Set Computer	1) Complex Instruction Set Computer
2) Less number of instructions.	2) More Number of Instructions.
3) Fixed length Instruction format (means I'm using 32-bit instruction format. So this point gives benefit to program counter bcz program counter knows that instruction start from here & ends here. & also where the next instruction starts & ends.)	3) Variable length Instruction format. (means the instructions which we are using in this are of 16 bit, 32 bit & 64 bit, we've kept instructions of different sizes.)
4) Few number of Addressing Modes.	4) Large number of Addressing modes.
5) Less Cost	5) Cost is High
6) Less Powerful (Here you'll not get maximum number of instructions but you can use available instructions & make complex instructions. But by default easy instructions are available to you. You can use them multiple times & make it as a complex. let's say you've only addition & subtraction. Then to do multiplication, you can do addition multiple times.	6) More Powerful (~~bcz~~ simple reason you know that is no. of instruction set is large. So if you want to perform any operation like multiplication, division, addition, subtraction, if you're to check any bit or anything then you'll get maximum or complex instructions in this.)
7) Single Cycle Instructions (But here we try to keep CPI value 1 using the pipeline architecture. So that single cycle में instruction को complete करना है।)	7) Several cycle Instructions (means instruction, जो है, ज़रूरी नहीं की एक cycle में काम complete होगा। you can also take multiple cycles.) CPI की value इसीलिए थोड़ा गड़बड़ रहती है इधर। CPI means cycle per Instruction)

P.T.O

OPPO F19 Pro+

RISC

8). Manipulation of data is done using registers.
(means here we are using register to register instructions more although load & store will be used by default. bcz data आप memory से load ~~करोगे~~ और final result को आप store करोगे।)

9) Hardwired control unit based.

10). Examples:
~~Mainframes, Motorola 6800, Intel 8080~~
MIPS, ARM, SPARC, Fugaku

CISC

8). Manipulation of data directly in the memory.
(means we are manipulating data directly in RAM.)

~~9) Microcontrolled Program~~

9) Microprogrammed control unit based.

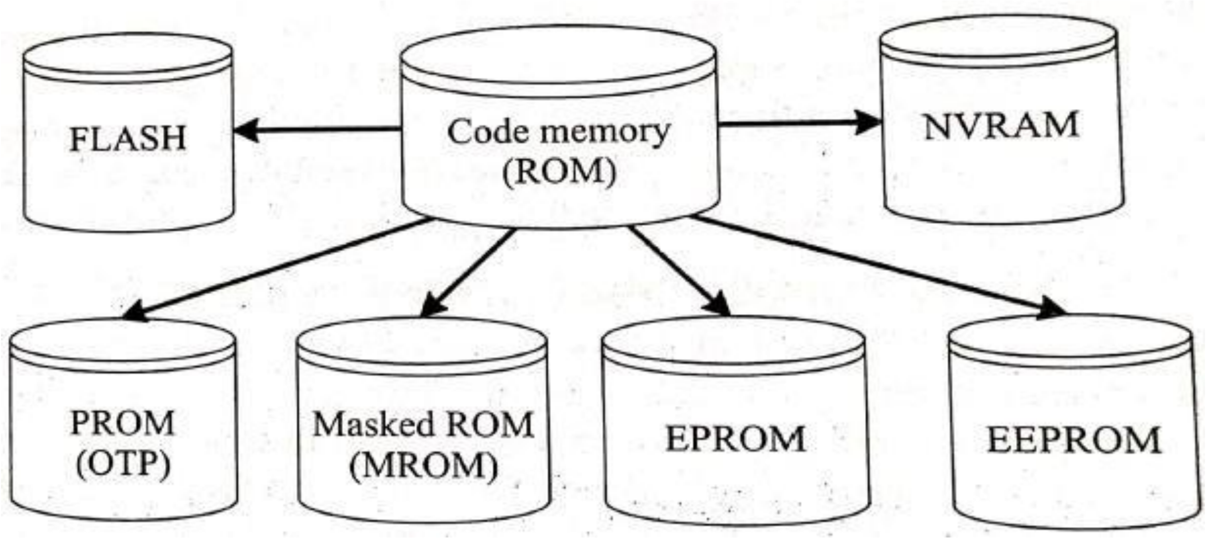
10) Examples:
~~MIPS, ARM, SPARC, Fugaku~~ ~~supercomputer~~
Mainframes, Motorola 6800, Intel 8080

Q8). DIFFERENTIATE BETWEEN HARVARD AND VON NEUMANN ARCHITECTURE.

Feature	Harvard Architecture	von Neumann Architecture
Memory Structure	Separate data and instruction memory spaces.	Single memory space for both data and instructions.
Data Bus and Address Bus	Separate buses for data and instructions.	Single bus for both data and instructions.

Feature	Harvard Architecture	von Neumann Architecture
Speed and Efficiency	Potential for higher speed and efficiency.	May face potential bottlenecks in simultaneous access.
Complexity	More complex due to separate memory spaces.	Simpler implementation with a single memory space.
Applications	Common in embedded systems, DSPs, microcontrollers.	Widely used in general-purpose computers.

Q9). EXPLAIN DIFFERENT MEMORY TECHNOLOGIES AND MEMORY TYPES USED IN EMBEDDED SYSTEM DEVELOPMENT.



➢ <u>**Program Storage Memory (ROM):**</u>

1. **Masked Memory (MROM):**
   - One-time programmable, factory-programmed during production.
   - Low-cost for high-volume production.
   - Limitation: Permanent, cannot be modified for firmware upgrades.

2. **Programmable Read Only Memory (PROM) / One Time Programmable Memory (OTP):**
   - User-programmable, uses fuses for data storage.
   - OTP version for commercial production, cost-effective.
   - Limitation: Not suitable for development due to lack of reprogramming.

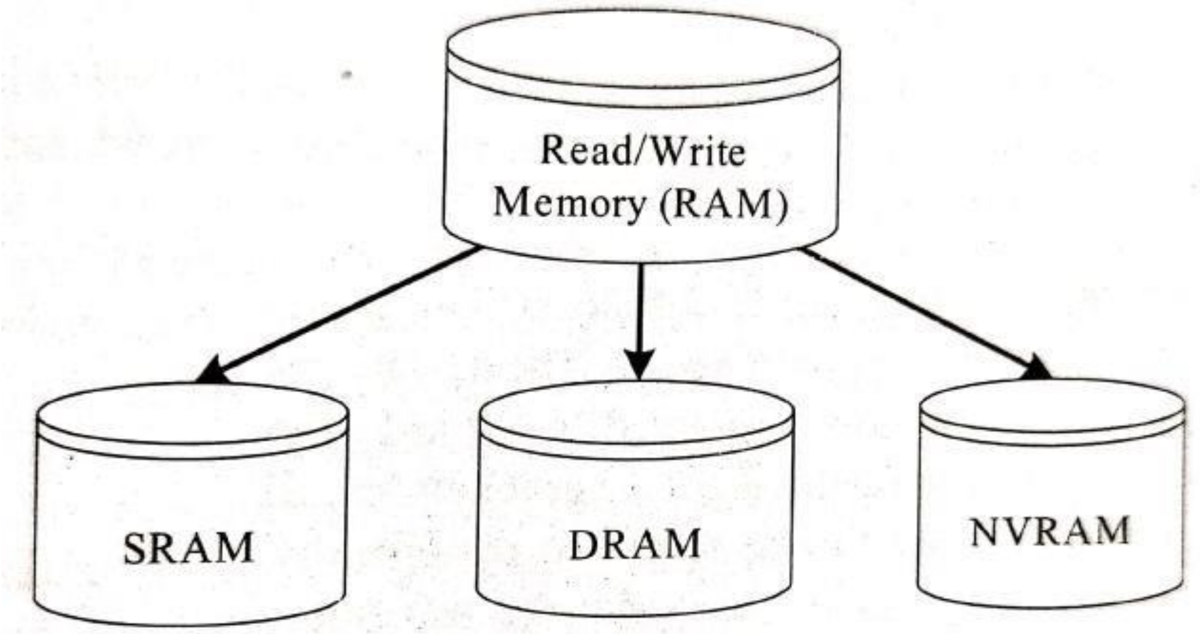
3**. Erasable Programmable Read Only Memory (EPROM):**
   - Reprogrammable in-chip by charging the floating gate.
   - Erased by exposing to UV light.

- Limitation: Requires removal for UV erasing, a time-consuming process.

4. **Electrically Erasable Programmable Read Only Memory (EEPROM):**
   - Altered electrically at the register/byte level.
   - In-circuit reprogramming with chip erase mode.
   - Limited capacity compared to standard ROM.

5. **FLASH:**
   - Combines re-programmability of EEPROM with high capacity.
   - Organized in sectors/pages, erasable at sector/page level.
   - Popular for embedded designs with 1000+ erase cycles.

6. **NVRAM (Non-Volatile RAM):**
   - Static RAM with battery backup for non-volatility.
   - 10-year lifespan, maintains data in power absence.
   - Example: DSJ644 from Maxim/Dallas (32KB).



➢ <u>Read-Write Memory / Random Access Memory (RAM)</u>:

1. **Static RAM (SRAM):**
   - Stores data using flip-flops.
   - Fastest RAM type, typically implemented with six transistors.
   - Volatile, used for temporary data storage.

2. **Dynamic RAM (DRAM):**
   - Stores data as charge in capacitors.
   - Requires periodic refreshing.
   - Common in general-purpose computers.

3. **Non-Volatile RAM (NVRAM):**
   - Static RAM with battery backup.
   - Maintains data in the absence of power.

- Limited lifespan, e.g., DSJ644 with a 10-year expectancy.

Q10). ANALYZE THE ROLE OF SENSORS, ACTUATORS IN THE EMBEDDED SYSTEM DEISGN.

Role of Sensors and Actuators in Embedded Systems:

Sensors:

1. Data Acquisition:

- Function: Gather information, converting real-world phenomena into electrical signals.
- Example: Temperature sensors, accelerometers.

2. Environmental Monitoring:

- Function: Monitor conditions, provide data for decision-making.
- Example: Humidity sensors, gas sensors.

3. Feedback Mechanism:

- Function: Provide feedback for system adjustment.
- Example: Proximity sensors in touchscreens.

4. Navigation and Positioning:

- Function: Assist in determining device location and orientation.
- Example: GPS modules, accelerometers.

5. Security and Surveillance:

- Function: Detect unauthorized access, enhance security.
- Example: Motion detectors, infrared sensors.

Actuators:

1. **Execution of Commands:**
    a. Function: Carry out actions based on system commands.
    b. Example: Motors for moving parts, solenoids.
2. **Feedback and Control:**
    a. Function: Provide feedback for system control.
    b. Example: Servo motors adjusting camera position.
3. **Physical Output:**
    a. Function: Produce physical effects in the external environment.
    b. Example: Speakers for audio, motors for movement.
4. **Energy Conversion:**
    a. Function: Convert electrical signals into energy.

b. Example: Piezoelectric actuators for precise movements.

5. **Automation and Control:**
   a. Function: Crucial for automating processes and device control.
   b. Example: Servo motors in robotics.

(Extra):

Integration of Sensors and Actuators:

- Closed-Loop Systems:

- Example: Thermostats adjusting based on temperature sensors.

- Human-Machine Interaction:

- Example: Touchscreens, gesture recognition.

- Adaptive Systems:

- Example: Adaptive lighting adjusting based on ambient light sensors.

- Energy Efficiency:

- Example: Smart thermostats adjusting based on occupancy sensors.

In summary, the symbiotic role of sensors and actuators is vital for the functionality, adaptability, and efficiency of embedded systems in various applications.

UNIT 5:

Q1). EXPLAIN THE NEED FOR OPERATING SYSTEM WITH A NEAT SYSTEM ARCHITECTURE.

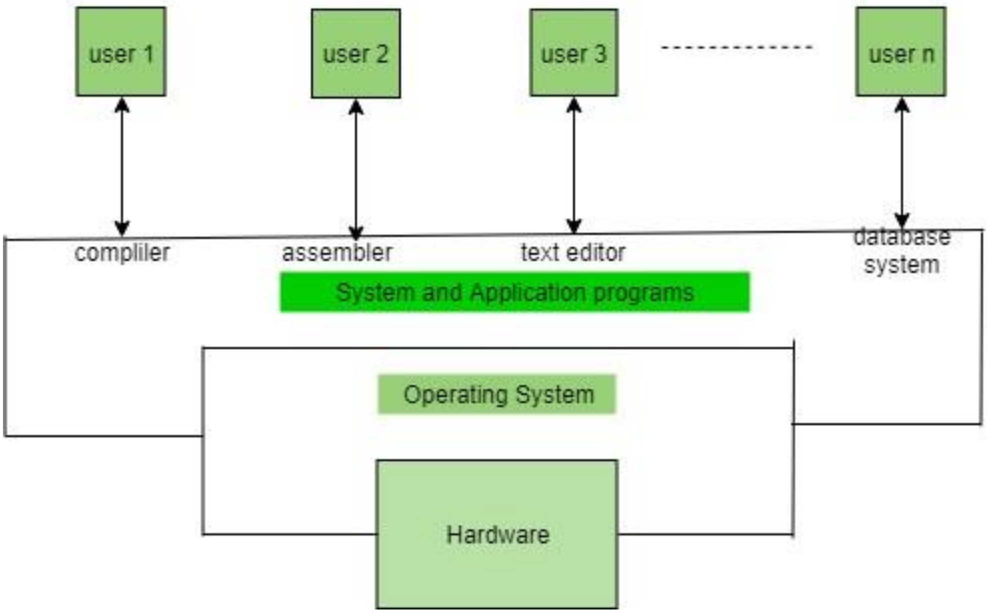


Fig: Abstract view of the components of a computer system

Need for Operating System:

<u>OS as a platform for Application programs</u>: The operating system provides a platform, on top of which, other programs, called application programs can run. These application programs help users to perform a specific task easily. It acts as an interface between the computer and the user. It is designed in such a manner that it operates, controls, and executes various applications on the computer.

<u>Managing Input-Output unit</u>: The operating system also allows the computer to manage its own resources such as memory, monitor, keyboard, printer, etc. Management of these resources is required for effective utilization. The operating system controls the various system input-output resources and allocates them to the users or programs as per their requirements.

<u>Multitasking</u>: The operating system manages memory and allows multiple programs to run in their own space and even communicate with each other through shared memory. Multitasking gives users a good experience as they can perform several tasks on a computer at a time.

<u>A platform for other software applications</u>: Different application programs are needed by users to carry out particular system tasks. These applications are managed and controlled by the OS to ensure their effectiveness. It serves as an interface between the user and the applications, in other words.

<u>Controls memory</u>: It helps in controlling the computer's main memory. Additionally, it allows and deallocates memory to all tasks and applications.

Looks after system files: It helps with system file management. As far as we are aware, all of the data on the system exists as files. It facilitates simple file interaction.

Provides Security: It helps to maintain the system and applications safe through the authorization process. Thus, the OS provides security to the system.

Q2). EXPLAIN THE BASIC FUNCTIONS OF A REAL-TIME KERNEL.

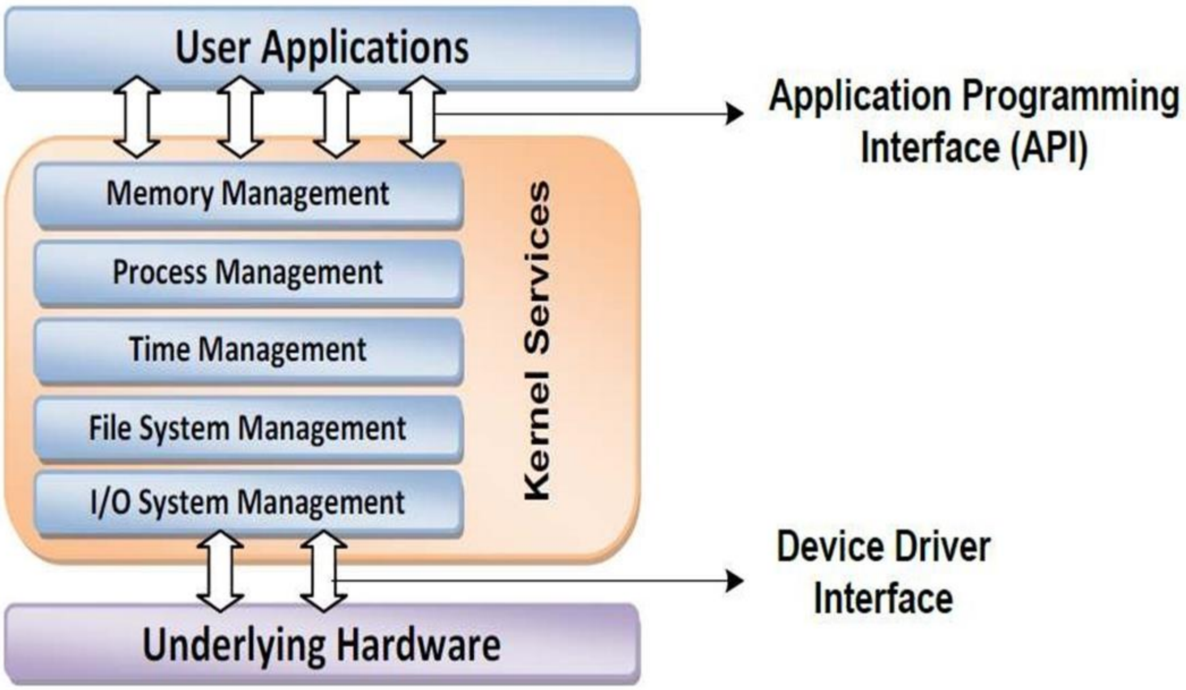


Fig. SYSTEM ARCHITECTURE

The Kernel: The kernel is the core of the operating system. It is responsible for managing the system resources and the communication among the hardware and other system services. Kernel acts as the abstraction layer between system resources and user applications.

• Kernel contains a set of system libraries and services. For a general-purpose OS, the kernel contains different services like memory management, process management, Time management, file system management, I/O System management.

The kernel of an operating system performs various crucial functions:

1. Process Management:
- Handles tasks by setting up memory, loading process code, allocating resources, scheduling execution, and managing Process Control Blocks (PCBs).
- Manages inter-process communication, synchronization, and process termination.

2. Primary Memory Management:
- Oversees volatile memory (RAM) where processes and shared data reside.
- Managed by the Memory Management Unit (MMU) to track memory usage and allocate/deallocate space as needed.

3. File System Management:
- Manages files and directories, including creation, deletion, and alteration.
- Saves files in secondary storage, allocates file space, and provides flexible naming conventions.

4. I/O System (Device) Management:
- Routes I/O requests from user applications to appropriate devices through an Application Programming Interface (API).
- Device Manager handles loading/unloading of device drivers and manages I/O operations.

5. Secondary Storage Management:
- Manages secondary storage devices (e.g., disks) for backup.
- Handles disk storage allocation, scheduling, and free disk space management.

6. Protection Systems:
- Implements security policies to restrict access for different users and applications.

7. Interrupt Handler:
- Manages external/internal interrupts, ensuring proper handling of system events.

8. Time Management:
Efficiently schedules and allocates resources, ensuring timely execution of tasks, with specialized mechanisms in real-time systems for meeting strict deadlines.

9. I/O Management:
Coordinates input and output operations between the computer and external devices, utilizing buffering, caching, and scheduling for optimized data transfer and system performance.

Q3). CLASSIFY THE TYPES OF OPERATING SYSTEM

Types of Operating Systems:

- Batch Operating System
- Multi-Programming System
- Multi-Processing System
- Multi-Tasking Operating System
- Time-Sharing Operating System
- Distributed Operating System
- Network Operating System
- Real-Time Operating System

1. Batch Operating System:

- Manages similar jobs grouped into batches without direct user interaction.

- Advantages include efficient processor use and reduced idle time, but debugging can be challenging.

2. Multi-Programming Operating System:

- Allows multiple programs in main memory for improved resource utilization.

- Enhances system throughput and reduces response time.

3. Multi-Processing Operating System:

- Utilizes multiple CPUs for resource execution, increasing system throughput.

- Offers redundancy if one processor fails.

4. Multi-Tasking Operating System:

- Enables simultaneous execution of multiple programs using round-robin scheduling.

- Enhances productivity with proper memory management but may lead to system heating.

5. Time-Sharing Operating Systems:

- Allocates CPU time to tasks in a round-robin fashion, reducing idle time.

- Facilitates resource sharing, improves productivity, but complexity and security risks exist.

6. Distributed Operating System:

- Connects autonomous computers through a shared network, allowing remote access.

- Enables fast computation, resource sharing, but faces challenges like networking delays and security.

7. Network Operating System:

- Runs on a server, managing data, users, and networking functions for shared access.

- Provides stability, security through servers, but requires regular maintenance.

8. Real-Time Operating System:

- Serves real-time systems with strict time constraints.

- Differentiates into hard and soft real-time systems, excelling in maximum resource utilization but having limited multitasking capability.

Q4). DIFFERENTIATE BETWEEN THE MEMORY MANAGEMENT OF GENERAL-PURPOSE OPERATING SYSTEM AND REAL TIME OPERATING SYSTEM.

Real-Time Operating System	General Purpose Operating System
The RTOS always uses priority-based scheduling.	Task scheduling in a GPOS isn't necessarily based on which application or process is the most important. Threads and processes are often dispatched using a "fairness" policy.
The time response of the RTOS is deterministic.	The time response of the general-purpose operating system is not deterministic.
A low-priority job in an RTOS would be pre-empted by a high-priority one if required, even executing a kernel call.	A high-priority thread in a GPOS cannot preempt a kernel call.
The real-time operating system optimizes memory resources.	The GPOS does not optimize the memory resources.
The RTOS is mainly used in the embedded system.	GPOS is mainly used in PC, servers, tablets, and mobile phones.
The real-time operating system has a task deadline.	The general-purpose operating system has no task deadline.
It doesn't have large memory.	It has a large memory.
GPOS code is not often modular in nature when it comes to development.	RTOS kernel code is intended to be scalable, allowing developers to selectively select kernel objects.
RTOS is designed and developed for a single-user environment.	GPOS is designed for a multi-user environment.
Examples: FreeRTOS, Contiki source code, etc.	Examples: Linux, Windows, IOS, etc.

Q5). DIFFERENTIATE BETWEEN THE HARD-REAL TIME AND SOFT REAL-TIME EMBEDDED SYSTEMS WITH AN EXAMPLE FOR EACH.

Terms	Hard Real-Time System	Soft Real-Time System
Definition	A hard-real time system is a system in which a failure to meet even a single deadline may lead to complete or appalling system failure.	A soft real-time system is a system in which one or more failures to meet the deadline are not considered complete system failure, but that performance is considered to be degraded.
File size	In a hard real-time system, the size of a data file is small or medium.	In a soft real-time system, the size of the data file is large.
Response time	In this system, response time is predefined that is in a millisecond.	In this system, response time is higher.
Utility	A hard-real time system has more utility.	A soft real-time system has less utility.
Database	A hard real-time system has short databases.	A soft real-time system has enlarged databases.
Performance	Peak load performance should be predictable.	In a soft real-time system, peak load can be tolerated.
Safety	In this system, safety is critical.	In this system, safety is not critical.
Integrity	Hard real-time systems have short term data integrity.	Soft real-time systems have long term data integrity.
Restrictive nature	A hard real-time system is very restrictive.	A Soft real-time system is less restrictive.
Computation	In case of an error in a hard real-time system, the computation is rolled back.	In a soft real-time system, computation is rolled back to a previously established checkpoint to initiate a recovery action.
Flexibility and laxity	Hard real-time systems are not flexible, and they have less laxity and generally provide full deadline compliance.	Soft real-time systems are more flexible. They have greater laxity and can tolerate certain amounts of deadline misses.
Validation	All users of hard real-time systems get validation when needed.	All users of soft real-time systems do not get validation.
Examples	Satellite launch, Railway signalling systems, and Safety-critical systems are good examples of a hard real-time system.	DVD player, telephone switches, electronic games, Linux, and many other OS provide a soft real-time system.

BEFORE GOING TO NEXT QUESTION, HAVE A LOOK AT THIS:

Thread	Process
Thread is a single unit of execution and is part of process.	Process is a program in execution and contains one or more threads.
A thread does not have its own data memory and heap memory.	Process has its own code memory, data memory, and stack memory.
A thread cannot live independently; it lives within the process.	A process contains at least one thread.
There can be multiple threads in a process; the first (main) thread calls the main function and occupies the start of the stack memory of the process.	Threads within a process share the code, data and heap memory; each thread holds separate memory area for stack.
Threads are very inexpensive to create.	Processes are very expensive to create; involves many OS overhead.
Context switching is inexpensive and fast.	Context switching is complex and involves lots of OS overhead and comparatively slow.
If a thread expires, its stack is reclaimed by the process.	If a process dies, the resource allocated to it are reclaimed by the OS and all associated threads of the process also dies.

Q6). DEFINE THE FOLLOWING:

a. **TASK**
b. **PROCESS**
c. **THREAD**

Task:

<u>Definition:</u>

Program in execution, maintained by OS.

Also known as a "Job."

<u>Execution Nature:</u>

Abstract and high-level.

Resource Requirements:

Less defined, may lack detailed resource needs.

Concurrency:

May or may not involve concurrent processing.

Process:

Definition:

Active instance of a program in execution.

Requires CPU, memory, and I/O resources.

Execution Nature:

Concrete and specific.

Resource Allocation:

Well-defined requirements.

Concurrency:

Can involve concurrent execution.

Creation and Structure:

Structured format with CPU properties, registers, etc.

Memory Segmentation:

Segregated into Stack, Data, and Code memory.

Process Life Cycle:

Traverses through various states (New, Ready, Running, Waiting, Terminated).

Threads:

Definition:

Primitive code executor within a process.

Single sequential flow of control.

Execution Nature:

Lightweight process.

Concurrency:

Enables concurrent execution within a process.

Resource Allocation:

Maintains own thread status, shares resources.

Multithreading:

Allows splitting a process into multiple threads.

Main thread and additional threads created within it.

Advantages of Multithreading:

Better memory utilization.

Efficient CPU utilization.

Thread Standards:

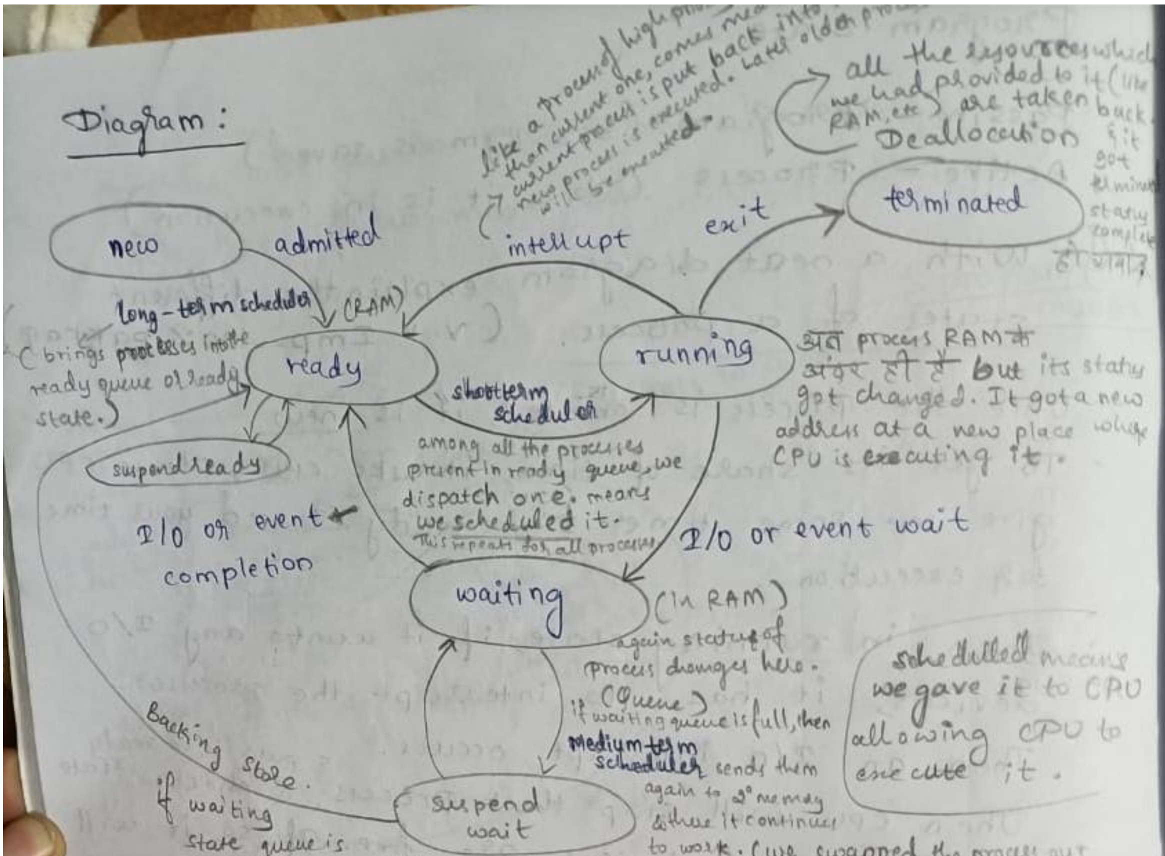
POSIX Threads: Portable Operating System Interface.

Win32 Threads: Supported by Windows OS.

Java Threads: Supported by Java programming language.

Q7). EXPLAIN THE PROCESS LIFE CYCLE WITH A NEAT STATE TRANSITION DIAGRAM.

Diagram:



if waiting state queue is empty then process will go from suspend wait state to there. Else it is sent to ready queue so that from there it can go to running state. We call it as backing store. अगर ready queue भी भरी है तो थोड़ी देर के लिए suspend ready queue में चला जायेगा, later it goes to ready state & then to running state & gets terminated at last.

Here, you can observe that

- Short term scheduler (or CPU scheduler) – selects which process should be executed next & allocates CPU
- Long term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.
- Medium term scheduler can be added if degree of multiple programming needs to decrease.
  - Remove process from memory, store on disk, bring back in from disk to continue execution: swapping.

As a process executes, it changes state

- **new:** The process is being created
- **running:** Instructions are being executed
- **waiting:** The process is waiting for some event to occur
- **ready:** The process is waiting to be assigned to a processor
- **terminated:** The process has finished execution

Q8). DIFFERENTIATE BETWEEN TASK AND PROCESS.

Aspect	Task	Process
Definition	In OS context, program in execution, often used interchangeably with "Job" and "Process."	Program or part of it in execution, with specific system resources allocated.
Specificity	More general term, may not specify resource requirements or have a well-defined lifecycle.	Specifies resource needs and undergoes distinct states during its execution lifecycle.
Resource Allocation	May not explicitly involve detailed resource allocation or management.	Involves specific resource requirements such as CPU, memory, and I/O devices.
Execution Unit	Represents a unit of work without specifying the details of its execution.	Represents a program actively running, utilizing system resources, and having a defined execution context.
Concurrency	May not inherently imply concurrent execution or parallel processing.	Can involve concurrent execution, allowing efficient utilization of system resources.

Creation to Termination	Implies something to be done, but details about its lifecycle may vary.	Involves a well-defined lifecycle from creation to termination.
States and Transitions	May not explicitly go through different states during its execution.	Typically undergoes states like Created, Ready, Running, Blocked, and Completed.
Interchangeability	Terms like "Task," "Job," and "Process" are often used interchangeably.	The term "Process" is more specific, referring to the execution instance of a program.
Scope	Can be a broader term referring to a unit of work, not limited to program execution.	Specifically refers to the execution of a program.
Usage Context	Used in various contexts, including general project management.	Primarily used in the context of operating systems and software execution.

1.