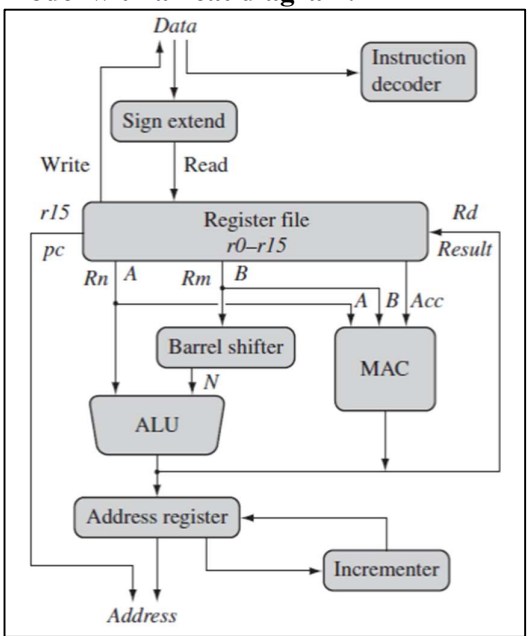
1. Compare and Contrast microprocessor and microcontroller.

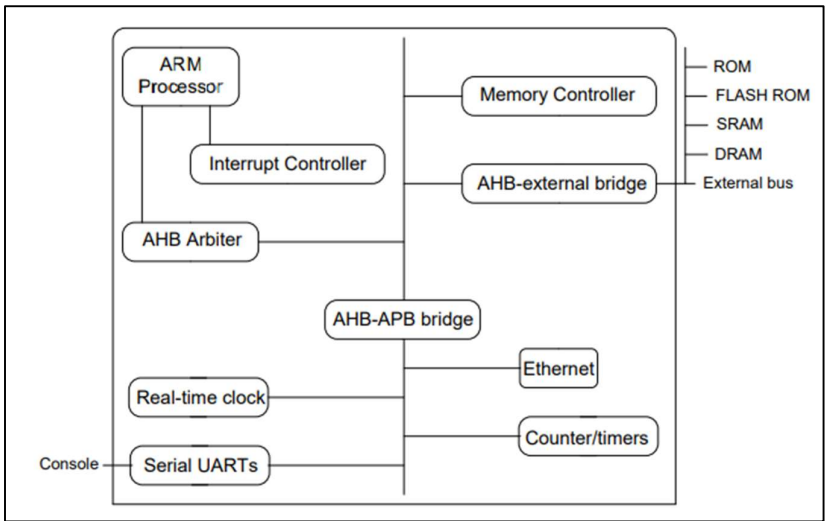
Microprocessor	Microcontroller
It is an electronic component that is used by a computer to do work. It is a CPU on a single IC chip containing millions of transistors, registers and diodes that work together.	It is a compact IC chip designed to monitor a specific operation in an embedded system. A typical microcontroller includes a processor, memory, and I/O peripherals on a single chip.
Generally, does not have RAM, ROM, and I/O pins	Is 'all-in-one'. Processor with RAM, ROM, and I/O pins, all in one chip.
Usually uses its pins as the bus to interface the RAM, ROM, and peripherals. Hence the controlling bus is expandable at the board level.	Controlling bus is internal and not available to the board design.
Generally capable of being built into bigger general-purpose applications.	Usually used for more dedicated applications.
Generally, do not have power saving systems	Have power saving system like idle mode or power saving mode.
Overall cost of system made with microprocessors are high, because of the high number of external components required.	Made using CMOS technology so they are cheaper than microprocessors.
Processing speed is above 1GHz	Processing speed are between 8MHz to 50MHz
Based on Von Neuman model, where program and data are stored in same memory model.	Based on Harvard architecture where program memory and data memory are separate.

2. Explain ARM core data flow model with a neat diagram.



- The arrows represent the flow of data, the lines represent the buses, and the boxes represent either an operation unit or a storage area.
- Data enters the processor core through the Data bus. The data may be an instruction to execute or a data item.
    - Figure shows a Von Neumann implementation of the ARM—data items and instructions share the same bus. (In contrast, Harvard implementations of the ARM use two different buses).
- The 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 load-store architecture—means it has two instruction types for transferring data in and out of the processor:
    - load instructions copy data from memory to registers in the core.
    - store instructions copy data from registers to memory.
- There are no data processing instructions that directly manipulate data in memory. Thus, data processing is carried out in registers.
- Data items are placed in the register file—a storage bank made up of 32-bit registers.
    - Since the ARM core is a 32-bit processor, most instructions treat the registers as holding signed or unsigned 32-bit values. The sign extends 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.
- Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the Address bus.
    - One important feature of the ARM is that register Rm alternatively can be preprocessed in the barrel shifter before it enters the ALU. Together the barrel shifter and ALU can calculate a wide range of expressions and addresses.
- 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 Incrementor 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.

3. **Along with neat diagram of an ARM based embedded system (Microcontroller), explain the hardware components.**



ARM processor based embedded system hardware can be separated into the following four main hardware components:

- **The ARM processor:** The ARM processor controls the embedded device. Different versions of the ARM processor are available to suit the desired operating characteristics.

- **Controllers:** Controllers coordinate important blocks of the system. Two commonly found controllers are memory controller and interrupt controller.

    - **Memory controller:** Memory controllers connect different types of memory to the processor bus. These memory devices allow the initialization code to be executed.

    - **Interrupt controller:** An interrupt controller provides a programmable governing policy that allows software to determine which peripheral or device can interrupt the processor at any specific time. There are two types of interrupt controller available for ARM processors. These are the standard interrupt controller and vector interrupt controller. The interrupt handler determines which device requires servicing by reading a device bitmap register in the interrupt controller.

- **Peripherals:** The peripherals provide all the input-output capability external to the chip and are responsible for the uniqueness of the embedded device. A peripheral device performs input and output functions for the chip by connecting to other devices or sensors that are off chip. All ARM peripherals are memory mapped. Controllers are specialized peripherals that implement higher levels of functionality within the embedded system.

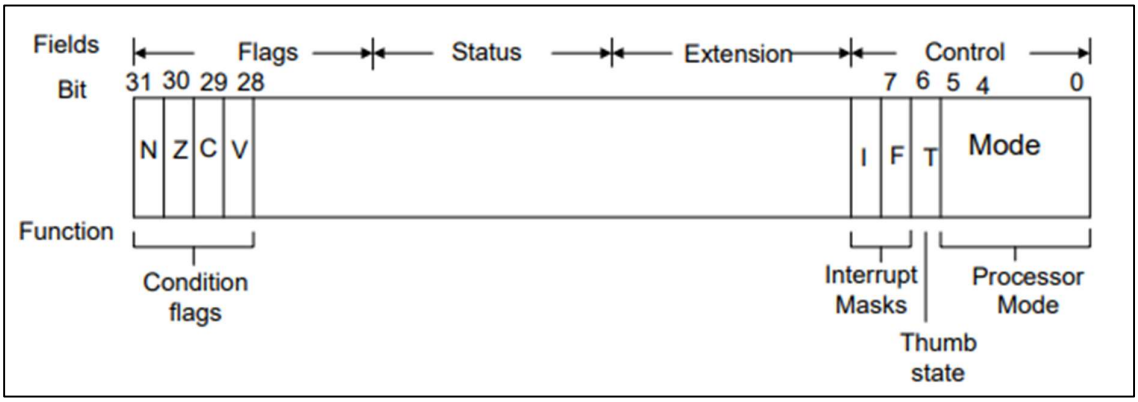
- **Bus:** A bus is used to communicate between different parts of the device. Embedded devices use an on-chip bus that is internal to the chip and that allows different peripheral devices to be interconnected with an ARM core. The ARM processor core is bus master-a logical device capable of initiating a data transfer with another device across the same bus.

- **Memory:** An embedded system has to have some form of memory to store and execute code. There are three types of memory- cache, main memory, and secondary memory. Cache is the fastest memory located near the ARM processor core. Cache is placed between the main memory and the core. It is used to speed up data transfer between the processor and main memory. The main memory is large depending on the application. Secondary memory is the largest and slowest form of memory.

4. Explain the different processor modes provided by ARM7.

- 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.
- A privileged mode allows read/write access to the CPSR.
- A nonprivileged mode only allows read access to the control field in the CPSR but allows read-write access to the conditional flags.
- There are seven processor modes : six privileged modes and one nonprivileged mode.
- The privilege modes are abort, fast interrupt request , interrupt request, supervisor, system and undefined. The nonprivileged mode is user.
- 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 operating system 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.

5. Give the schematic of a Current Program Status Register of ARM7 processor briefing the individual bits.

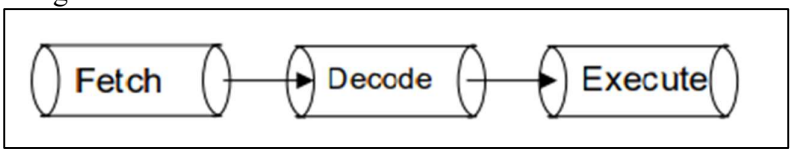


The CPSR is divided into four fields, each 8 bits wide: flags, status, extension, and control. In current designs the extension and status fields are reserved for future use. The control field contains the processor mode, state, and interrupts mask bits. The flag field contains the condition flags. The following table gives the bit patterns that represent each of the processor modes in the CPSR.
When CPSR bit 5 , T=1 , then the processor is in Thumb state. When T=0, the processor is in ARM state.
The CPSR has two internal mask bits,7 and 6 (I and F) which control the masking Interrupt request(IRQ) and Fast Interrupt Request(FIR).
The following table shows the conditional flags.

Flag	Flag Name	Set When
N	Negative	Bit 31 of the result is a binary 1
Z	Zero	The result is 0, frequently used to indicate equality
C	Carry	The result causes an unsigned carry
V	Overflow	The result causes a signed overflow

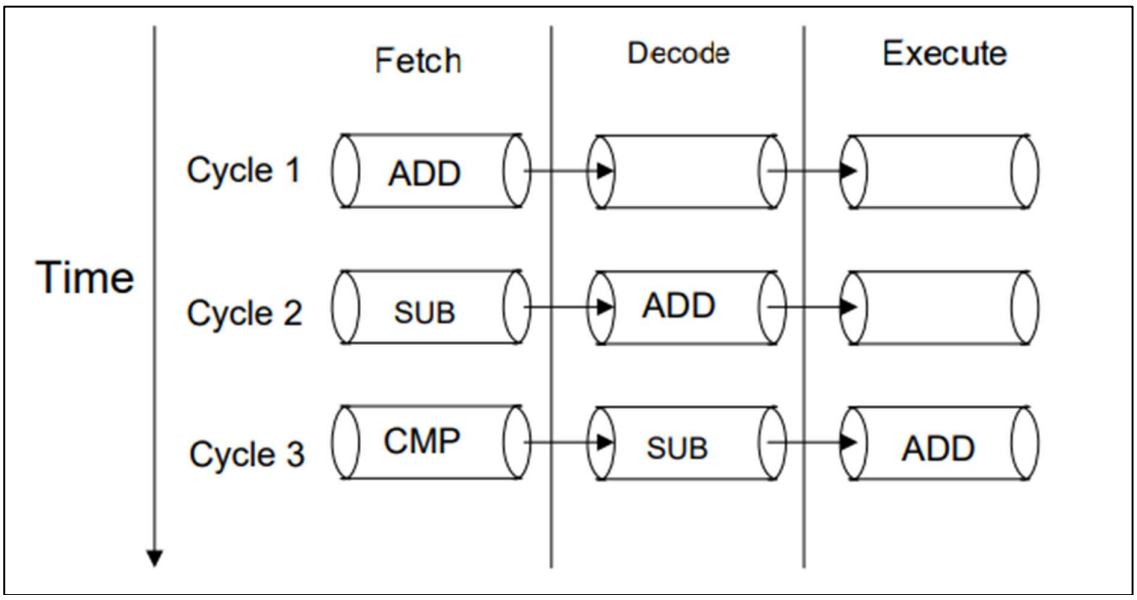
6. What s Pipelining. Explain in detail schematically.

Pipeline is the mechanism to speed up execution by fetching the next instruction while other instructions are being decoded and executed.



- 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.

- Figure shown below illustrates the pipeline using a simple example. It 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.
- In the first cycle, the core fetches the ADD instruction from the memory.
- In the second cycle, the core fetches the SUB instruction and decodes the ADD instruction.
- In the third cycle, the core fetches CMP instruction from the memory, decodes the SUB instruction and executes the ADD instruction.



7. Discuss the ARM design philosophy.

Following are the ARM design philosophy:

- The ARM processor has been specially designed to be small to reduce power consumption and extend battery operation- essentially for applications such as mobile phones and personal digital assistants.
- High code density is a major requirement since embedded systems have limited memory due to cost and physical size restrictions. High code density is useful for applications that have limited on-board memory, such as mobile phones.
- Embedded systems are price sensitive and use low-cost memory devices. The ability to low-cost memory devices produces essential savings.
- Another requirement is to reduce the area of the die taken up by the embedded processor. For a single-chip solution, the smaller the area used by the embedded processor, the more available space for specialized peripherals.
- Another requirement is the hardware debug technology within the processor so that software engineers can view what is happening while the processor is executing code.

8. Describe conditional execution. Write the different code suffix.

- Conditional execution controls whether or not the core will execute an instruction.
- Prior to execution, the processor compares the condition attribute with the condition flags in the CPSR. If they match, then the instruction is executed; otherwise, the instruction is ignored.
- The condition attribute is post-fixed to the instruction mnemonic, which is encoded into the instruction.
- The following Table lists the conditional execution code mnemonics. When a condition mnemonic is not present, the default behavior is to set it to always (AL) execute.

Mnemonic	Name	Condition flags
EQ	equal	*Z*
NE	not equal	*z*
CS HS	carry set/unsigned higher or same	*C*
CC LO	carry clear/unsigned lower	*c*
MI	minus/negative	*N*
PL	plus/positive or zero	*n*
VS	overflow	*V*
VC	no overflow	*v*
HI	unsigned higher	*zC*
LS	unsigned lower or same	*Z* or *c*
GE	signed greater than or equal	*NV* or *nv*
LT	signed less than	*Nv* or *nV*
GT	signed greater than	*NzV* or *nzv*
LE	signed less than or equal	*Z* or *Nv* or *nV*
AL	always (unconditional)	ignored

9. Differentiate between RISC and CISC processors.

CISC	RISC
Complex instructions, taking multiple clock	Simple instructions, taking single clock
Emphasis on hardware, complexity is in the micro-program/processor	Emphasis on software, complexity is in the complier.
Complex instructions, instructions executed by micro-program/processor	Reduced instructions, instructions executed by hardware
Variable format instructions, single register set and many instructions	Fixed format instructions, multiple register sets and few instructions
Many instructions and many addressing modes	Fixed instructions and few addressing modes
Conditional jump is usually based on status register bit	Conditional jump can be based on a bit anywhere in memory
Memory reference is embedded in many instructions	Memory reference is embedded in LOAD/STORE instructions

10. Explain the major design rules to implement the RISC philosophy.

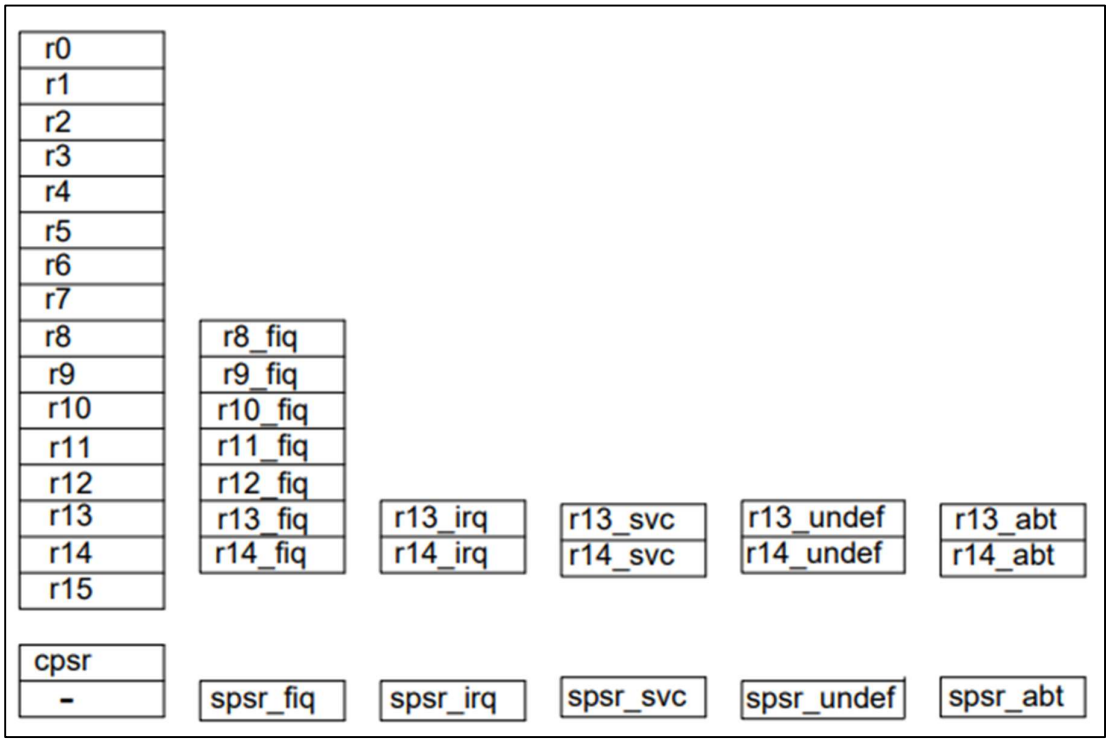
- Instructions—RISC processors have a reduced number of instruction classes. These classes provide simple operations that can each be executed in a single cycle. The compiler or programmer synthesizes complicated operations (for example, a divide operation) by combining several simple instructions. Each instruction is having fixed length to allow the pipeline to fetch future instructions before decoding the current instruction.
    - In contrast, in CISC processors the instructions are often of variable size and take many cycles to execute.
- 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.
    - There is no need for an instruction to be executed by a mini program called microcode as on CISC processors.
- 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.
- 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. Memory accesses are costly, so separating memory accesses 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.
- These design rules allow a RISC processor to be simpler, and thus the core can operate at higher clock frequencies.
    - In contrast, traditional CISC processors are more complex and operate at lower clock frequencies.

11. Briefly describe the concept of exceptions, interrupts, and the vector table.

- When an exception or interrupt occurs, the processor sets the pc to a specific memory address. The address is within a special address range called the vector table.
    - The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt.
    - The memory map address 0x00000000 (or in some processors starting at the offset 0xffff0000) is reserved for the vector table, a set of 32-bit words.
- When an exception or interrupt occurs, the processor suspends normal execution and starts loading instructions from the exception vector table.
- Each vector table entry contains a form of branch instruction pointing to the start of a specific routine:
    - **Reset vector** is the location of the first instruction executed by the processor when power is applied. This instruction branches to the initialization code.
    - **Undefined instruction vector** is used when the processor cannot decode an instruction.
    - **Software interrupt vector** is called when you execute a SWI instruction. The SWI instruction is frequently used as the mechanism to invoke an operating system routine.
    - **Prefetch abort** vector occurs when the processor attempts to fetch an instruction from an address without the correct access permissions. The actual abort occurs in the decode stage.
    - **Data abort vector** is similar to a prefetch abort but is raised when an instruction attempts to access data memory without the correct access permissions.
    - **Interrupt request vector** is used by external hardware to interrupt the normal execution flow of the processor. It can only be raised if IRQs are not masked in the CPSR.
    - **Fast interrupt request vector** is similar to the interrupt request but is reserved for hardware requiring faster response times. It can only be raised if FIQs are not masked in the CPSR.

12. Explain the programmer's model of ARM processors with complete register sets available.

Figure below shows all 37 registers in the register file. Of these, 20 registers are hidden from a program at different times. These registers are called banked registers. They are available only when the processor is in a particular mode, for example, abort mode has banked registers r13_abt, r14_abt and spsr_abt. Banked registers of a particular mode are denoted by an underline character post-fixed to the mode mnemonic.

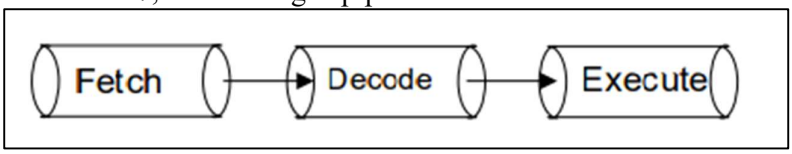


Every processor mode except user mode can change mode by writing directly to the mode bits of the CPSR. All processor modes except system mode have a set of associated banked registers that are subset of the main 16 registers. A banked register maps one-to-one onto a user mode register. If the processor mode is changed, a banked register from the new mode will replace an existing register. For example, when the processor mode is in the interrupt request mode, the instructions you execute still excess registers named r13 and r14. However, these registers are the banked registers r13_irq and r14_irq. The user mode registers r13_usr and r14_usr are not affected by the instruction referencing these registers. A program still has normal access to the other registers r0 to r12. The processor mode can be changed by a program that writes directly to the CPSR, when the processor core is in privilege mode. The following exception and interrupts cause a mode change: reset, interrupt request, fast interrupt request, software interrupt, data abort, prefetch abort and undefined instructions. Exceptions and interrupts suspend the normal execution of sequential instructions and jump to a specific location. The core changing from user mode to interrupt request mode, happens when an interrupt request occurs due to an external device raising an interrupt to the processor core. This change causes user registers r13 and r14 to be banked. The user registers are replaced with registers r13_irq and r14_irq respectively. r14_irq contains the return address and r13_irq contains the stack pointer for interrupt request mode.

13. What is pipeline in ARM? Illustrate with an example. Show the pipeline stages of ARM7, ARM9 and ARM10.

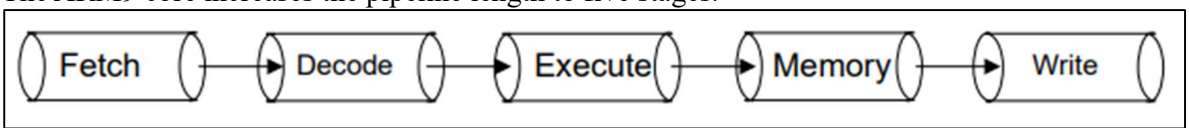
Pipeline is the mechanism to speed up execution by fetching the next instruction while other instructions are being decoded and executed.

In an ARM7, a three-staged pipeline is used.

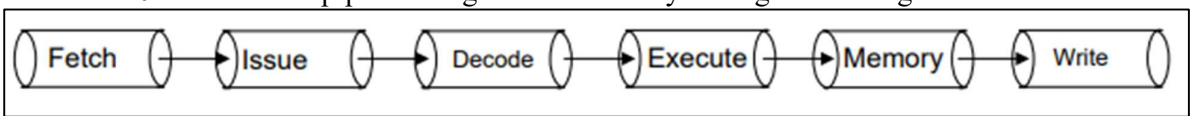


- Fetch loads an instruction from memory.
- Decode identifies the instruction to be executed.
- Execute processes the instruction and writes the result back to a register.

The ARM9 core increases the pipeline length to five stages.



The ARM10 increases the pipeline length still further by adding a sixth stage.



1. Explain the MOV instruction set provided by ARM7 with the example for each.

Move instruction copies N into a destination register Rd, where N is a register or immediate value. This instruction is useful for setting initial values and transferring 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 a register	$Rd = \sim N$

Example: This example shows a simple move instruction. The MOV instruction takes the contents of register r5 and copies them into register r7, in this case, taking the value 5, and overwriting the value 8 in register r7.

```
PRE:  r5 = 5
      r7 = 8
MOV r7, r5 ; let r7 = r5
POST: r5 = 5
      r7 = 5
```

2. Write a program for forward and backward branch by considering an example.

```
        B forward
        ADD r1, r2, #4
        ADD r0, r6, #2
        ADD r3, r7, #4
forward
        SUB r1, r2, #4
--------------------------------------------------
backward
        ADD r1, r2, #4
        SUB r1, r2, #4
        ADD r4, r6, r7
        B backward
```

3. Write and explain arithmetic instructions with respect to the ARM processor.

The arithmetic instructions implement addition and subtraction of 32-bit signed and unsigned values.

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

Example: The following simple subtract instruction subtracts a 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;
POST  r0 = 0x00000001
```

Example: The SUBS instruction is useful for decrementing loop counters. In this example, we subtract the immediate value one from the value one stored in register r1. The result value zero is written to register r1. The CPSR is updated with the ZC flags being set.

```
PRE   cpsr = nzcvqiFt_USER
      r1 = 0x00000001
SUBS r1, r1, #1
POST  cpsr = nZCvqiFt_USER
      r1 = 0x00000000
```

4. Design ARM assembly language program to perform the addition and multiplication of two 32-bit numbers.

Multiplication:

```
        area multi, code, readonly
entry
start
        ldr r0, = 11111111
        ldr r1, = 2222222
        mul r2,r1,r0
stop b stop
        end
```

Addition:

```
        area add, code, readonly
entry
start
        ldr r0, =11111111
        ldr r1, =22222222
        add r2,r1,r0
stop b stop
        end
```

5. Explain the different branch instructions of ARM processor.

A branch instruction changes the flow of execution or is used to call a routine. This type of instruction allows programs to have subroutines, if-then-else structures, and loops. The change of execution flow forces the program counter pc to point to a new address.

- The address label is stored in the instruction as a signed pc-relative offset and must be within approximately 32 MB of the branch instruction.
- T refers to the Thumb bit in the CPSR. When instructions set T, the ARM switches to Thumb state.

Example: This example shows a forward and backward branch. Because these loops are address specific, we do not include the pre- and post-conditions. The forward branch skips three instructions. The backward branch creates an infinite loop.

```
        B forward
        ADD r1, r2, #4
        ADD r0, r6, #2
        ADD r3, r7, #4
forward
        SUB r1, r2, #4
-------------------------------------------------
backward
        ADD r1, r2, #4
        SUB r1, r2, #4
        ADD r4, r6, r7
        B backward
```

- In this example, forward and backward are the labels. The branch labels are placed at the beginning of the line and are used to mark an address that can be used later by the assembler to calculate the branch offset.
- The branch with link, or BL, instruction is similar to the B instruction but overwrites the link register lr with a return address. It performs a subroutine call.

6. Explain the different barrel shifter operations with suitable examples.

- Data processing instructions are processed within the arithmetic logic unit (ALU).
- A unique and 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 specific number of positions before it enters the ALU.
- Pre-processing or shift occurs within the cycle time of the instruction.
    - This shift increases the power and flexibility of many data processing operations.
    - This is particularly useful for loading constants into a register and achieving fast multiplies or divisions by a power of 2.

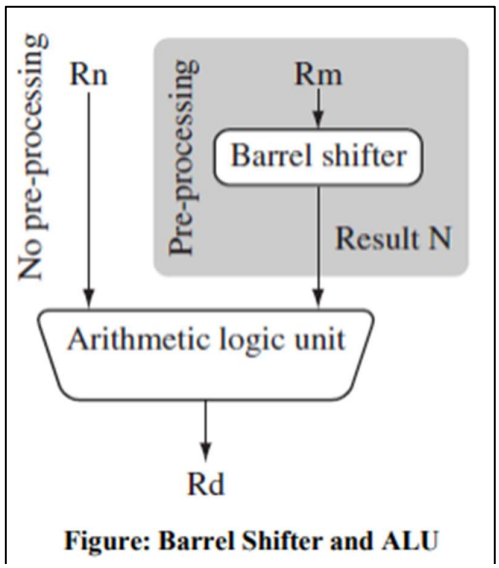


Figure: Barrel Shifter and ALU

- Figure shows the data flow between the ALU and the barrel shifter.
- Register Rn enters the ALU without any pre- processing of registers.
- We apply a logical shift left (LSL) to register Rm before moving it to the destination register. This is the same as applying the standard C language shift operator « to the register.
- The MOV instruction copies the shift operator result N into register Rd. N represents the result of the LSL operation described in the following Table.

Question Bank for Module 2

1) Explain the working of ARM processor with co-processor instructions along with syntax.

Coprocessor instructions are used to extend the instruction set.

- A coprocessor can either provide additional computation capability or be used to control the memory subsystem including caches and memory management.
- The coprocessor instructions include data processing, register transfer, and memory transfer instructions.
- These instructions are only used by cores with a coprocessor.

SYNTAX:

CDP {<cond>} cp, opcode1, Cd, Cn {, opcode2}
<MRC|MCR>{<cond>} cp, opcode1, Rd, Cn, Cm {, opcode2}
<LDC|STC>{<cond>} cp, Cd, addressing

CDP	Coprocessor Data Processing – Perform an operation in a Coprocessor
MRC MCR	Coprocessor Register Transfer – Move data from/to coprocessor registers
LDC STC	Coprocessor Memory Transfer – load and store blocks of memory from/to a coprocessor

- In the syntax of the coprocessor instructions,
  - The cp field represents the coprocessor number between p0 and p15
  - The opcode fields describe the operation to take place on the coprocessor.
  - The Cn, Cm, and Cd fields describe registers within the coprocessor.
- The coprocessor operations and registers depend on the specific coprocessor you are using.
- Coprocessor 15 (CP15) is reserved for system control purposes, such as memory management, write buffer control, cache control, and identification registers.

Example: This example shows a CP15 register being copied into a general-purpose register.

```
; transferring the contents of CP15 register c0 to register r10
        MRC p15, 0, r10, c0, c0, 0
```

Here CP15 register-0 contains the processor identification number. This register is copied into the general-purpose register r10.

2) Explain the working of Profiling and Cycle counting.

- The first stage of any optimization process is to identify the critical routines and measure their current performance. A profiler is a tool that measures the proportion of time or processing cycles spent in each subroutine. You use a profiler to identify the most critical routines.
- A cycle counter measures the number of cycles taken by a specific routine. You can measure your success by using a cycle counter to benchmark a given subroutine before and after an optimization.
- The ARM simulator used by the ADS1.1 debugger is called the ARMulator and provides profiling and cycle counting features.
  - The ARMulator profiler works by sampling the program counter pc at regular intervals.
  - The profiler identifies the function the pc points to and updates a hit counter for each function it encounters. Another approach is to use the trace output of a simulator as a source for analysis.
  - The accuracy of a pc-sampled profiler is limited, as it can produce meaningless results if it records too few samples.
- ARM implementations do not normally contain cycle-counting hardware; so, to easily measure cycle counts you should use an ARM debugger with ARM simulator.
  - You can configure the ARMulator to simulate a range of different ARM cores and obtain cycle count benchmarks for a number of platforms

3) Explain the scheduling of following instructions with respect to the ARM9 TDMI pipeline implementation,
i)STR ii) LDRH iii) B Label

STR (Store Register)

The STR instruction stores a register value to memory. The ARM9TDMI pipeline handles store instructions over multiple stages:

- **Fetch (Cycle 1)**: The STR instruction is fetched from memory.
- **Decode (Cycle 2)**: The instruction is decoded, and operands are read from the register bank.
- **ALU (Cycle 3)**: The effective address for the store is calculated.
- **LS1 (Cycle 4)**: The calculated address is used to store the data in memory.
- **LS2 (Cycle 5)**: No operation for STR as it's not a load instruction.

For a store instruction that stores a single value, the process takes two cycles assuming zero-wait-state memory. Thus, STR executes in five cycles in the pipeline.

LDRH (Load Register Halfword)

The LDRH instruction loads a halfword (16-bit) value from memory into a register:

- **Fetch (Cycle 1)**: The LDRH instruction is fetched from memory.
- **Decode (Cycle 2)**: The instruction is decoded, and operands are read.
- **ALU (Cycle 3)**: The effective address for the load is calculated.
- **LS1 (Cycle 4)**: The halfword is read from memory.
- **LS2 (Cycle 5)**: The halfword is zero- or sign-extended and then written to the register.

The LDRH instruction issues in one cycle, but the load result is not available until two cycles after the issue. Thus, it takes a total of five cycles to complete in the pipeline.

B (Branch)

The B (Branch) instruction changes the program counter to a new address, causing the pipeline to flush and refill:

- **Fetch (Cycle 1)**: The branch instruction is fetched from memory.
- **Decode (Cycle 2)**: The instruction is decoded, and the branch target address is calculated.
- **ALU (Cycle 3)**: The branch target address is applied, causing the pipeline to flush.
- **Fetch (Cycle 4)**: The new instruction at the branch target address is fetched.
- **Decode (Cycle 5)**: The new instruction is decoded and begins execution.

Branch instructions take three cycles due to the pipeline flush and refilling process.

4) Explain the ARM swap instruction with an example code.

- The swap instruction is a special case of a load-store instruction. It swaps the contents of memory with the contents of a register.
- This instruction is an atomic operation—it reads and writes a location in the same bus operation, preventing any other instruction from reading or writing to that location until it completes.
- Swap cannot be interrupted by any other instruction or any other bus access. We say the system "holds the bus" until the transaction is complete. Also, swap instruction allows for both a word and a byte swap.

SYNTAX: SWP {B} {<cond>} Rd, Rm, [Rn]
Example: The swap instruction loads a word from memory into register r0 and overwrites the memory with register r1.

```
PRE   mem32[0x9000] = 0x12345678
      r0 = 0x00000000
      r1 = 0x11112222
      r2 = 0x00009000
SWP r0, r1, [r2]
POST  mem32[0x9000] = 0x11112222
      r0 = 0x12345678
      r1 = 0x11112222
      r2 = 0x00009000
```

5) All the first five Lab Programs (Part A).

1)To add an array of 16 bit numbers and store the 32 bit result in internal ram

```
      area add,code,readonly
ENTRY
START
     mov r5,#6
     mov r0,#0
     ldr r1,=value
loop
     ldrh r3,[r1],#2
     add r0,r0,r3
     subs r5,r5,#1
     cmp r5,#0
     bne loop
     ldr r4,=result
     str r0,[r4]
     jmp b jmp

     area data2,data,readwrite
Value DCW
     0x1111,0x2222,0x3333,
     0XAAAA,0XBBBB,0XCCCC
Result DCD 0x0000
end
```

2)Squares of 1-10 using look-up table

```
     area square,code,readonly
ENTRY
START
     ldr r1,01
ldr r0,=table1
Repeat
     ldr r2,[r0],#4
     add r1,#1
     cmp R1,#10
     BNE repeat
     BEQ stop
stop b stop
table1
     DCD 0X00000001;
     DCD 0X00000004;
     DCD 0X00000009;
     DCD 0X00000010;
     DCD 0X00000019;
     DCD 0X00000024;
     DCD 0X00000031;
     DCD 0X00000040;
     DCD 0X00000051;
     DCD 0X00000065;
END
```

3)To find the largest value in a given array

```
     area largest,code,readonly
ENTRY
START
     mov r5, #5
     ldr r1,01
ldr r0, =table1
Repeat
     ldr r2, [r0], #4
     add r1, #1
     cmp R1, #10
     BHI loop1
     mov r2, r4
Loop1
     subs r5, r5, #1
     cmp r5, #0
     BNE loop
     ldr r4=result
     str r2, [r4]
     BEQ stop
Stop b stop
Value
     DCD 0X44444444
     DCD 0X22222222
     DCD 0X11111111
     DCD 0XAAAAAAAA
     DCD 0X88888888
     DCD 0X99999999
     area data1, data, readwrite
result
     DCD 0X00000000
```

4)To find the smallest value in a given array

```
     area small,code,readonly
ENTRY
START
     mov r5, #5
     ldr r1,01
ldr r0, =table1
Repeat
     ldr r2, [r0], #4
     add r1, #1
     cmp R1, #10
     BLS loop1
     mov r2, r4
Loop1
     subs r5, r5, #1
     cmp r5, #0
     BNE loop
     ldr r4=result
     str r2, [r4]
     BEQ stop
Stop b stop
Value
     DCD 0X44444444
     DCD 0X22222222
     DCD 0X11111111
     DCD 0XAAAAAAAA
     DCD 0X88888888
     DCD 0X99999999
     area data1,data,readwrite
result
     DCD 0X00000000
```

5)Arranging number is ascending order

```
     area ascend,code,readonly
ENTRY
START
     Mov r8, #4
     Ldr r2, =Cvalue
     Ldr r3, =Dvalue
Loop0
     Ldr r2, [r1], #4
     Str r1, [r], #4
     Sub r8, r5, #1
     Cmp r8, #0
     BNE loop0
Start1
     Mov r5, #3
     Mov r7, #0
     Ldr r1, =Dvalue
Loop
     Ldr r2,[r1],#4
     Ldr r3, r1
     Cmp r2, r3
     BLT loop2
     Mov r7, #1
     Odd r5, #0
Loop2
     subs r5, r5, #1
     cmp r5, #0
     BNE loop
     cmp r7, #0
     BNE start1
Cvalue
     DCD 0X44444444
     DCD 0X11111111
     DCD 0X22222222
     Area data1,data,readwrite
Dvalue
     DCD 0X00000000
End
```

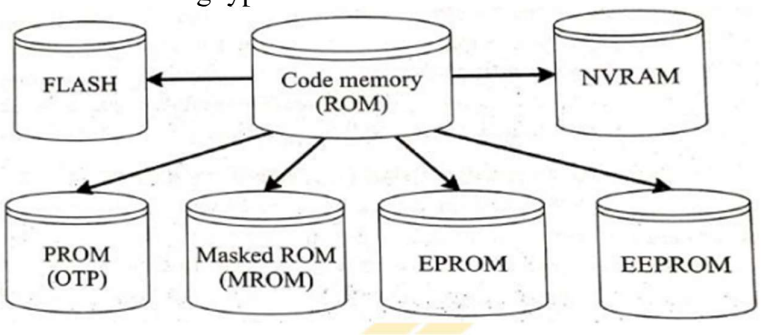
6)Count the number of 0's and 1's stored in consecutive memory locations.

```
     area count, code, readonly
ENTRY
START
     Ldr r2,=0
     Ldr r3,=0
     Mov r7=#2
     Ldr r6=value
Loop
     Mov r1,#32
     Ldr r0,[r0],#4
Loop0
     r0,ror,#1
     BHI ones
     Add r3,r3,#1
     B loop1
     add r2,r2,#1
loop1
     subs r1,r3,#1
     BNE loop
     Subs r7,r7,#1
     Cmp r7,r7,#1
     Cmp r7,#0
     BNE loop
     Area data,data1,readwrite
Value
     DCD 0x111111111
     DCD 0XAA55AA55
END
```

Question Bank for Module 3

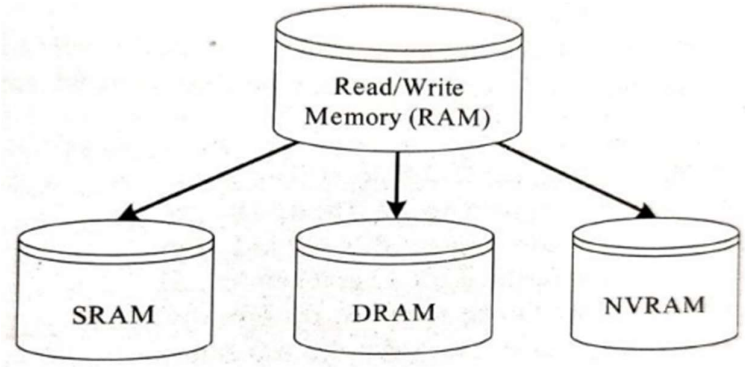
1) **What are the different types of memories used in Embedded System design? Explain the role of each.**

Program Storage Memory (ROM): The program memory or code storage memory of an embedded system stores the program instructions, and it can be classified into different types as per the block diagram representation given in the following Figure.
The code memory retains its contents even after the power to it is turned off. It is generally known as non-volatile storage memory. Depending on the fabrication, erasing and programming techniques, they are classified into the following types



- **Masked ROM (MROM)**
  - **Role:** Used for storing firmware in high-volume, cost-sensitive applications.
  - **Characteristic:** One-time programmable during the manufacturing process using hardwired technology. It is the least expensive type of solid-state memory but cannot be reprogrammed.
- **Programmable Read-Only Memory (PROM) / One-Time Programmable (OTP)**
  - **Role:** Suitable for production where the firmware is finalized and does not require updates.
  - **Characteristic:** Programmable by the end-user using a PROM programmer. Uses fuses that can be selectively burned to store data, making it a one-time programmable memory.
- **Erasable Programmable Read-Only Memory (EPROM)**
  - **Role:** Used during development phases where firmware may need frequent updates.
  - **Characteristic:** Can be erased and reprogrammed using ultraviolet (UV) light. Requires removal from the circuit for reprogramming, making it less convenient than other types.
- **Electrically Erasable Programmable Read-Only Memory (EEPROM)**
  - **Role:** Suitable for applications requiring frequent updates and non-volatile storage.
  - **Characteristics:** Electrically erasable and reprogrammable at the byte level. Provides flexibility as it can be reprogrammed in-circuit but has limited storage capacity compared to standard ROM.
- **FLASH Memory**
  - **Role:** Widely used in modern embedded systems for storing firmware and large amounts of non-volatile data.
  - **Characteristic:** Combines the re-programmability of EEPROM with higher capacity. Data is stored in sectors or pages, which can be erased and reprogrammed electrically.
- **Non-Volatile RAM (NVRAM)**
  - **Role:** Used for storing critical data that must be retained when the power is off.
  - **Characteristic:** Combines static RAM with a battery backup, ensuring data retention for up to 10 years.

Read-Write Memory/ Random Access memory (RAM): RAM is the data memory or working memory of the controller/ processor. Controller/ processor can read from it and write to it. It is volatile, meaning when the power is turned off, all the contents are destroyed. RAM is a direct access memory, meaning we can access the desired memory location directly without the need for traversing through the entire memory locations to reach the desired memory position.



- **Static RAM (SRAM)**
  - **Role:** Used for high-speed memory needs where quick access is essential.
  - **Characteristic:** Stores data using flip-flops made of transistors. It is fast and reliable but has lower density and higher cost compared to DRAM. Commonly used for cache memory.
- **Dynamic RAM (DRAM)**
  - **Role:** Suitable for applications requiring high-density memory.
  - **Characteristic:** Stores data as charge in capacitors, which needs to be refreshed periodically. It offers higher capacity at a lower cost but is slower than SRAM due to the refresh cycles. Typically used for main memory in computers.
- **Non-Volatile RAM (NVRAM)**
  - **Role:** Used for applications requiring non-volatile storage with fast read/write access.
  - **Characteristic:** Combines the speed of SRAM with a battery backup to retain data when power is off. Used for storing settings or operational data that must persist across power cycles

2) List different purposes of embedded system with examples.

Embedded systems are specialized computing systems that perform dedicated functions within larger mechanical or electrical systems. Here are the key purposes of embedded systems:

- **Data Collection/Storage/Representation:**
  - Acquire data from the external environment, which may be stored, analyzed, manipulated, or transmitted.
  - Handle various data types, including text, voice, images, video, and electrical signals.
  - Examples: Digital cameras capture and store images; measuring instruments in medical applications store and display data.

- **Data Communication:**
  - Facilitate the transfer of data between systems, either through wired or wireless means.
  - Used in applications ranging from simple home networks to complex satellite communications.
  - Examples: Wireless network routers, Bluetooth, ZigBee, Wi-Fi modules in embedded terminals.

- **Data (Signal) Processing:**
  - Perform various signal processing tasks such as speech coding, audio/video compression, and signal transmission.
  - Examples: Digital hearing aids process and enhance audio signals for improved hearing.

- **Monitoring:**
  - Continuously observe specific variables or conditions without controlling them.
  - Common in medical devices for monitoring patient vitals, and in instrumentation for measuring electrical parameters.
  - Examples: ECG machines monitor heartbeats; digital oscilloscopes monitor voltage signals.

- **Control:**
  - Actively control variables based on input data to maintain desired states or performance levels.
  - Combine sensors to detect changes and actuators to adjust the controlled variables.
  - Examples: Air conditioners regulate room temperature based on sensor inputs and user settings.

- **Application-Specific User Interface:**
  - Provide tailored interfaces for user interaction, often including buttons, switches, keypads, and display units.
  - Examples: Mobile phones feature keypads, touch screens, and displays for user interaction; smart running shoes with adjustable cushioning and user control buttons.

3) Briefly describe the classification of embedded systems.

Embedded systems can me classified based on various criteria.

Based on Generation

- **First Generation**: Built around 8-bit microprocessors (e.g., 8085, Z80) and 4-bit microcontrollers; used in digital telephone keypads, stepper motor control units.
- **Second Generation**: Utilized 16-bit microprocessors and 8/16-bit microcontrollers; examples include Data Acquisition Systems and SCADA systems.
- **Third Generation**: Employed 32-bit processors and 16-bit microcontrollers, with domain-specific processors like DSPs and ASICs; used in robotics, media, industrial control, networking.
- **Fourth Generation**: Featured System on Chips (SoC), reconfigurable processors, and multicore processors; examples include smartphones and mobile internet devices (MIDs).
- **Future Generations**: Continual advancements expected, focusing on more integration, miniaturization, and performance enhancements.

Classification Based on Complexity and Performance

- Small-Scale Embedded Systems:
  - Embedded systems which are simple in application needs and the performance parameters are not time critical (E.g.: Electronic toy).
  - Small-scale embedded systems are usually built around low performance and low cost 8- or 16-bit microprocessors/ microcontrollers.
  - It may or may not contain an operating system for its functioning.
- Medium-Scale Embedded Systems:
  - Embedded systems which are slightly complex in hardware and firmware (software) requirements.
  - Medium-scale embedded systems are usually built around medium performance, low cost 16- or 32-bit microprocessors/ microcontrollers or digital signal processors.
  - They usually contain an embedded operating system (general purpose/ real-time).
- Large-Scale Embedded Systems/ Complex Systems:
  - Embedded systems which involve highly complex hardware and firmware. They are employed in mission critical applications demanding high performance.
  - Large-scale embedded systems are commonly built around high performance 32- or 64-bit RISC processors/ controllers or Reconfigurable System on Chip (RSoC) or multi-core processors and programmable logic devices.
  - They usually contain a high-performance Real-Time Operating System (RTOS) for task scheduling, prioritization, and management.

4) What is an embedded system? Differentiate between general purpose computing system and 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. Embedded systems are becoming an inevitable part of any product or equipment in all fields including household appliances, telecommunications, medical equipment, industrial control, consumer products, etc.

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

5) Write a short note on :

i. Real Time Clock

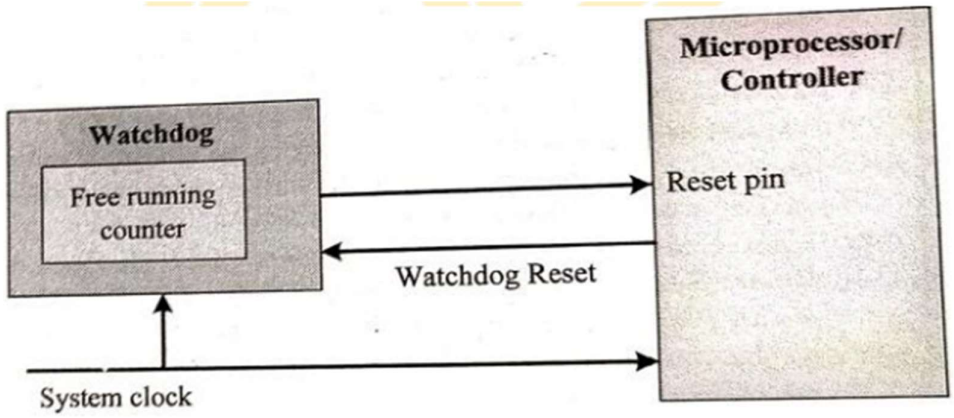
Real-Time Clock (RTC): Real-Time Clock is a system component responsible for keeping track of time. RTC holds information like current time (In hours, minutes and seconds) in 12-hour/ 24-hour format, date, month, year, day of the week, etc. and supplies timing reference to the system.

- RTC is intended to function even in the absence of power. RTCs are available in the form of Integrated Circuits from different semiconductor manufacturers like Maxim/Dallas, ST Microelectronics etc.
- The RTC chip contains a microchip for holding the time and date related information and backup battery cell for functioning in the absence of power, in a single IC package. The RTC chip is interfaced to the processor or controller of the embedded system.
- For Operating System based embedded devices, a timing reference is essential for synchronizing the operations of the OS kernel. The RTC can interrupt the OS .kernel by asserting the interrupt line of the processor/controller to which the RTC interrupt line is connected. The OS kernel identifies the interrupt in terms of the Interrupt Request (IRQ) number generated by an interrupt controller. One IRQ can be assigned to the RTC interrupt, and the kernel can perform necessary operations like system date time updating, managing software timers etc. when an RTC timer tick interrupt occurs.
- The RTC can be configured to interrupt the processor at predefined intervals or to interrupt the processor when the RTC register reaches a specified value (used as alarm interrupt).

ii. Watchdog Time

Watchdog Timer: In desktop Windows systems, if we feel our application is behaving in an abnormally or if the system hangs up, we have the 'Ctrl + Alt + Del' to come out of the situation. What happens to embedded system?

- We have a watchdog to monitor the firmware execution and reset the system processor/ microcontroller when the program execution hangs up. A watchdog timer, or simply a watchdog, is a hardware timer for monitoring the firmware execution. Depending on the internal implementation, the watchdog timer increments or decrements a free running counter with each clock pulse and generates a reset signal to reset the processor if the count reaches zero for a down counting watchdog, or the highest count value for an up-counting watchdog.
- If the watchdog counter is in the enabled state, the firmware can write a zero (for up counting watchdog implementation) to it before starting the execution of a piece of code and the watchdog will start counting. If the firmware execution doesn't complete due to malfunctioning, within the time required by the watchdog to reach the maximum count, the counter will generate a reset pulse, and this will reset the processor. If the firmware execution completes before the expiration of the watchdog, you can reset the count by writing a 0 (for an up-counting watchdog timer) to the watchdog timer register.



iii. Reset Circuit

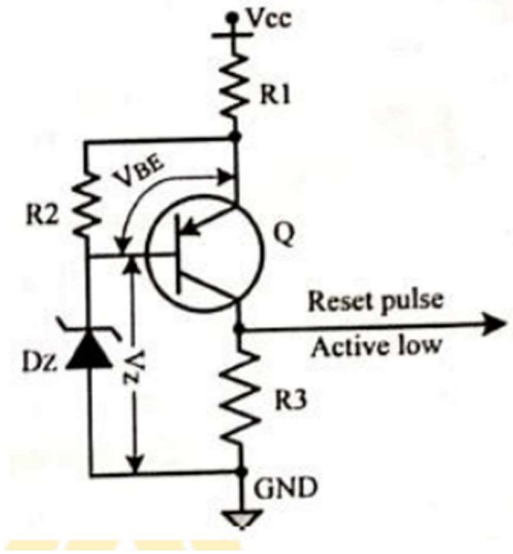
Reset Circuit: The reset circuit is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON.

- The reset signal brings the internal registers and the different hardware systems of the processor/ controller to a known state and starts the firmware execution from the reset vector (Normally from vector address 0x0000 for conventional processors/ controllers
- The reset signal can be either active high (The processor undergoes reset when the reset pin of the processor is at logic high) or active low (The processor undergoes reset when the reset pin of the processor is at logic low).
- Since the processor operation is synchronized to a clock signal, the reset pulse should be wide enough to give time for the clock oscillator to stabilize before the internal reset state starts.
- The reset signal to the processor can be applied at power ON through an external passive reset circuit comprising a Capacitor and Resistor or through a standard Reset IC like MAX810 from Maxim Dallas. Select the reset IC based on the type of reset signal and logic level (CMOS/ TTL) supported by the processor/ controller in use.
- Some microprocessors /controllers contain built-in internal reset circuitry, and they don't require external reset circuitry.

6) **Explain brown out protection.**

Brown-out Protection Circuit: Brown-out protection circuit prevents the processor/ controller from unexpected program execution behavior when the supply voltage to the processor/ controller falls below a specified voltage.

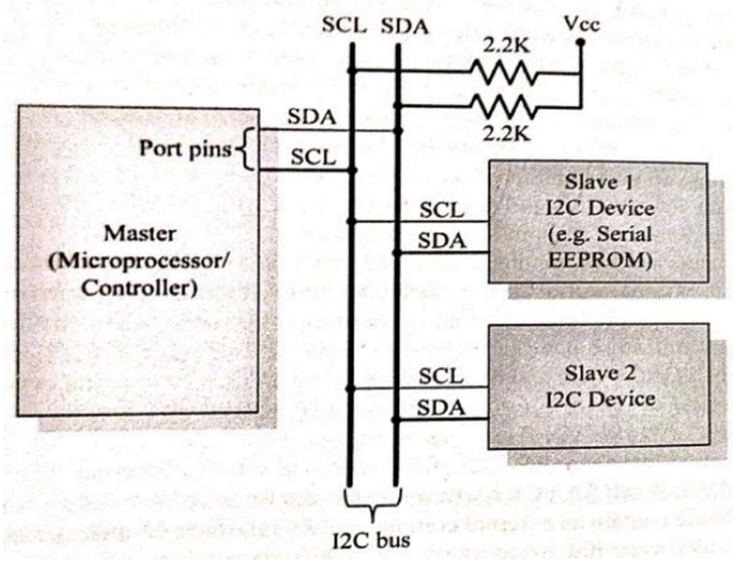
- It is essential for battery powered devices since there are greater chances for the battery voltage to drop below the required threshold. The processor behavior may not be predictable if the supply voltage falls below the recommended operating voltage. It may lead to situations like data corruption.
- A brown-out protection circuit holds the processor/ controller in reset state, when operating voltage falls below the threshold, until it rises above the threshold voltage.
- Certain processors/ controllers support built in brown-out protection circuit which monitors the supply voltage internally.
- If the processor/ controller don't integrate a built-in brown-out protection circuit, the same can be implemented using external passive circuits or supervisor ICs. The following Figure illustrates a brown-out circuit implementation using Zener diode and transistor for processor/ controller with active low Reset logic.



7) **List four onboard communication interfaces. Explain any one in detail.**

Inter-Integrated Circuit (I2C) Bus: The Inter-Integrated Circuit (I2C) Bus is a synchronous, bi-directional, half-duplex, two-wire serial communication interface.

- Developed by Philips Semiconductors in the 1980s, I2C is widely used for connecting microcontrollers to peripheral devices in embedded systems.
- The bus consists of two lines: Serial Clock (SCL) and Serial Data (SDA).
- The SCL line generates synchronization clock pulses, while the SDA line transmits serial data between devices.
- I2C supports multiple devices on the same bus, which can operate as either 'Master' or 'Slave'.
- The Master controls the communication by initiating and terminating data transfers, sending data, and generating clock pulses, while Slave devices respond to Master commands.
- I2C supports multi-master configurations and offers various data rates, including Standard Mode (up to 100 kbps), Fast Mode (up to 400 kbps), and High-Speed Mode (up to 3.4 Mbps).

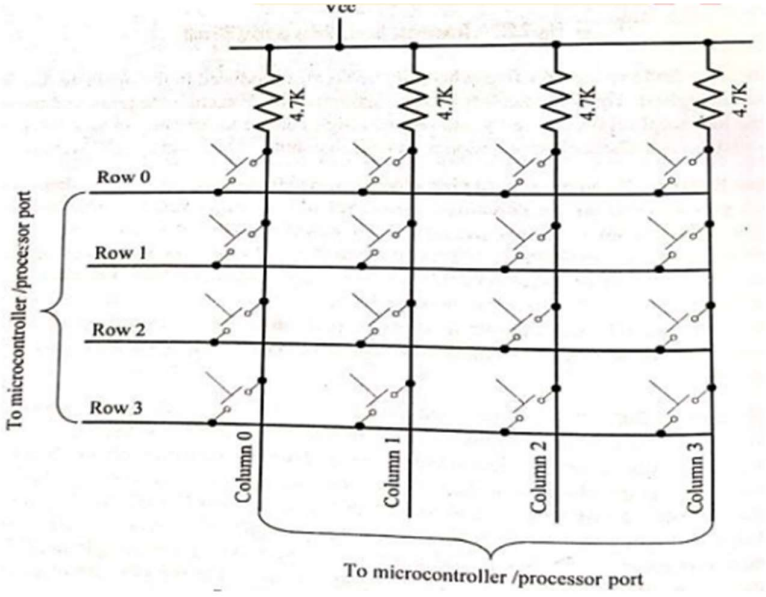


- **Serial Peripheral Interface (SPI)**
- **Universal Asynchronous Receiver Transmitter (UART)**
- **1-Wire Interface**
- **Parallel Interface**

8) Explain matrix keyboard interfacing.

Keyboard: Keyboard is an input device 'HIGH' Pulse generator for user interfacing.

- If the number of keys required is very limited, push-button switches can be used and they can be directly interfaced to the port pins for reading.
- However, there may be situations demanding a large number of keys for user input (e.g. PDA device with alpha-numeric keypad for user data entry).
    - In such situations it may not be possible to interface each keys to a port pin due to the limitation in the number of general-purpose port pins available for the processor/ controller in use and moreover it is wastage of port pins.
    - Matrix keyboard is an optimum solution for handling large key requirements. It greatly reduces the number of interface connections.



In a matrix keyboard, the keys are arranged in matrix fashion. For detecting a key press, the keyboard uses the scanning technique, where each row of the matrix is pulled low, and the columns are read. After reading the status of each columns corresponding to a row, the row is pulled high, and the next row is pulled low, and the status of the columns are read. This process is repeated until the scanning for all rows are completed. When a row is pulled low and if a key connected to the row is pressed, reading the column to which the key is connected will give logic 0. Since keys are mechanical devices, proper key de-bouncing technique should be applied.

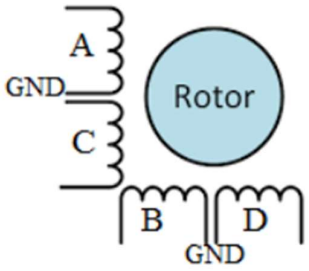
9) Explain the working of the Stepper Motor.

Stepper Motor: A stepper motor is an electro-mechanical device which generates discrete displacement (motion) in response to de electrical signals. It differs from the normal DC motor in its operation. The DC motor produces continuous rotation on applying DC voltage, whereas a stepper motor produces discrete rotation in response to the DC voltage applied to it.

Based on the coil winding arrangements, a two-phase stepper motor is classified into two.

Unipolar

A unipolar stepper motor contains two windings per phase. The direction of rotation (clockwise or anticlockwise) of a stepper motor is controlled by changing the direction of current flow. Current in one direction flows through one coil and in the opposite direction flows through the other coil. It is easy to shift the direction of rotation by just switching the terminals to which the coils are connected.



The coils are represented as A, B, C and D. Coils A and C carry current in opposite directions for phase 1 (only one of them will be carrying current at a time). Similarly, B and D carry current in opposite directions for phase 2 (only one of them will be carrying current at a time).

Bipolar

A bipolar stepper motor contains a single winding per phase. For reversing the motor rotation, the current flow through the windings is reversed dynamically. It requires complex circuitry for current flow reversal.

The stepping of stepper motor can be implemented in different ways by changing the sequence of activation of the stator windings. The different stepping modes supported by stepper motor are explained below:

Full Step: In the full step mode both the phases are energized simultaneously. The coils A, B, C and D are energized in the order, as shown in the following Table.

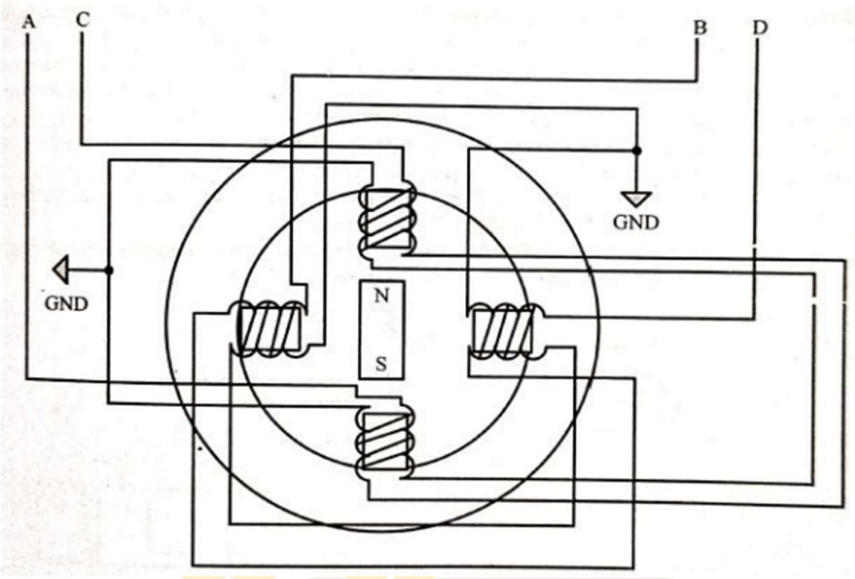
Wave Step: In the wave step mode, only one phase is energized at a time and each coils of the phase is energized alternatively. The A, B, C and D are energized in the order, as shown in the following Table.

Step	Full Step				Wave Step			
	Coil A	Coil B	Coil C	Coil D	Coil A	Coil B	Coil C	Coil D
1	H	H	L	L	H	L	L	L
2	L	H	H	L	L	H	L	L
3	L	L	H	H	L	L	H	L
4	H	L	L	H	L	L	L	H

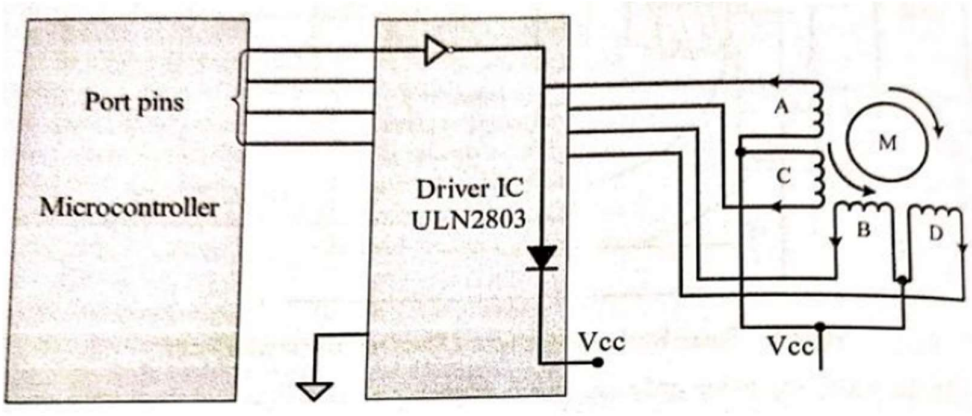
Half Step: It uses the combination of wave and full step. It has the highest torque and stability. The coil energizing sequence for half step is given in the Table below.

Step	Coil A	Coil B	Coil C	Coil D
1	H	L	L	L
2	H	H	L	L
3	L	H	L	L
4	L	H	H	L
5	L	L	H	L
6	L	L	H	H
7	L	L	L	H
8	H	L	L	H

The rotation of the stepper motor can be reversed by reversing the order in which the coil is energized. The following Figure shows the stator winding details of Stepper motor:



Two-phase unipolar stepper motors are the popular choice for embedded applications. The current requirement for stepper motor is little high and hence the port pins of a microcontroller/ processor may not be able to drive the directly. Also, the supply voltage required to operate stepper motor varies normally in the range 5V to 24V. Depending on the current and voltage requirements, special driving circuits are required to interface the stepper motor with microcontroller/ processors.



Module 4

1. What are the operational and non-operational qualities attributes of an embedded systems.

Operational Quality Attributes

The operational quality attributes pertain to the performance and functionality of embedded systems when they are in operation or 'online'. The following are the key attributes:

- **Response**
  - **Definition**: Measure of the system's quickness in tracking input changes.
  - **Importance**: Many embedded systems require fast, real-time responses.
    - **Example**: Flight control systems must respond in real time to ensure safety.
  - **Exceptions**: Not all systems need real-time responses.
    - **Example**: Electronic toys do not have critical response time requirements.
- **Throughput**
  - **Definition**: Efficiency of the system, measured as the rate of production or operation over time.
  - **Measurement**: Units of products, batches, etc.
    - **Example**: For a Card Reader, throughput is measured by the number of transactions it can perform in a given time.
  - **Benchmark**: Reference point for performance criteria. Used to compare products within the same line.
- **Reliability**
  - **Definition**: Measure of how dependable the system is.
  - **Metrics**:
    - **Mean Time Between Failures (MTBF)**: Frequency of failures.
    - **Mean Time to Repair (MTTR)**: Time allowed for the system to be out of order.
  - **Application**: Critical systems need high reliability with minimal downtime.
- **Maintainability**
  - **Definition**: Support and maintenance for technical issues and routine checkups. Higher reliability reduces the need for corrective maintenance.
  - **Types of Maintenance**:
    - **Scheduled/Periodic (Preventive)**: Regular checkups and replacements. Eg: Replacing printer ink cartridges.
    - **Corrective**: Addressing unexpected failures. Eg: Repairing the paper feeding part of a printer.
- **Security**
  - **Aspects**: Confidentiality, Integrity, Availability (distinct from availability in maintainability).
    - **Confidentiality**: Protecting data from unauthorized access.
    - **Integrity**: Ensuring data is not altered without authorization.
    - **Availability**: Ensuring authorized users have access to data.
    - **Example**: Personal Digital Assistants (PDAs). User profiles accessible via username and password. Some data may be read-only for users. Administrators have different access levels than regular users.
- **Safety**
  - **Definition**: Protection from potential damage to operators, the public, and the environment.
  - **Concerns**:
    - **Hardware or Firmware Failures**: Can lead to system breakdowns.
    - **Hazardous Emissions**: Potential for gradual or sudden damage.
  - **Safety Analysis**: Essential to evaluate and mitigate potential damages.

Non-operational Quality Attributes

These quality attributes pertain to aspects of the embedded system that are not directly related to its operational performance. The following are the key non-operational quality attributes:

- **Testability & Debug-ability**
  - **Testability**: Ease of testing the design, application, embedded hardware, and firmware.
    - **Hardware Testing**: Ensures peripherals and hardware function correctly.
    - **Firmware Testing**: Ensures firmware operates as expected.
  - **Debug-ability**: Ability to debug the product to identify sources of unexpected behavior.
    - **Hardware Debugging**: Identifies issues caused by hardware problems.
    - **Firmware Debugging**: Identifies errors in the firmware.
- **Evolvability**
  - **Definition**: Ease of modifying the embedded product to incorporate new firmware or hardware technologies.
  - **Context**: Similar to biological evolvability, which refers to non-heritable variation.
- **Portability**
  - **Definition**: Measure of 'system independence'.
  - **Characteristics**:
    - Functionality across various environments, processors, and operating systems.
    - Flexibility and ease of porting the product to new platforms.
  - **Porting Example**: Migrating firmware from one target processor to another (e.g., from Intel x86 to ARM Cortex M3).
    - **High-Level Language (e.g., C)**: Easier to port with minimal target-specific adjustments.
    - **Assembly Language**: Difficult to port due to processor-specific instructions.
  - **Comparison with Desktop Applications**:
    - **Microsoft Visual C++**: Runs only on Microsoft platforms.
    - **Java**: Runs on any operating system supporting Java standards.
- **Time to Prototype and Market**
  - **Definition**: Time between product conceptualization and readiness for selling or use.
  - **Importance**: Critical in a competitive market with rapid technology changes.
  - **Prototyping**: Informal rapid development of key product features.
    - **Benefits**: Reduces overall development time and time-to-market.
    - **Strategies**: Use off-the-shelf components and reusable assets to speed up prototyping.
- **Per Unit and Total Cost**
  - **Definition**: Costs monitored by end users and manufacturers.
  - **Commercial Sensitivity**: Pricing affects product success in the market.
  - **Manufacturer's Perspective**: Aim for marginal profit by balancing budget and system cost.
  - **Market Study**: Conduct cost-benefit analysis to determine optimal pricing.

2. Explain the fundamental issues in hardware software co-design.

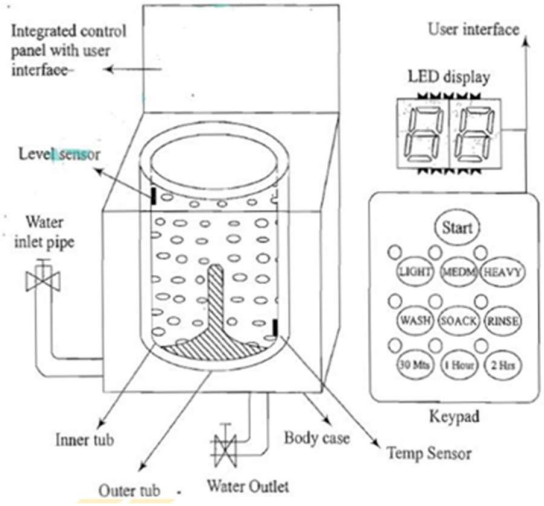
When addressing the problem statement of hardware-software co-design in real-life scenarios, several fundamental issues may arise. The key issues include:

- **Selecting the Model**
  - **Definition**: Models capture and describe system characteristics.
  - **Decision Making**: Difficult to choose the right model for each design phase.
    - **Example**: During specification, the focus is on system functionality. During implementation, focus shifts to system components and structure.
- **Selecting the Architecture**
  - **Definition**: Architecture specifies the number, types, and interconnection of system components.
  - **Types of Architectures**:
    - **Controller Architecture**: Implements the finite state machine model using a state register and combinational circuits.
    - **Datapath Architecture**: Implements the data flow graph model using registers, counters, memories, and ports.
    - **Finite State Machine Datapath (FSMD)**: Combines controller and datapath architectures.
    - **CISC (Complex Instruction Set Computing)**: Uses complex operations with single instructions, requiring additional silicon for microcode decoders.
    - **RISC (Reduced Instruction Set Computing)**: Uses simple operations with multiple instructions for complex operations, supporting extensive pipelining.
    - **VLIW (Very Long Instruction Word)**: Implements multiple functional units in the data path, packaging one instruction per unit.
    - **Parallel Processing**: Implements multiple concurrent processing elements (PEs) with local memory.
      - **SIMD (Single Instruction Multiple Data)**: Executes single instruction in parallel across PEs with a single controller.
      - **MIMD (Multiple Instruction Multiple Data)**: Executes different instructions concurrently across PEs, forming the basis of multiprocessor systems with shared memory or message passing.
- **Selecting the Language**
  - **Definition**: Programming languages capture computational models and map them into architecture.
  - **Language Choice**: No strict rules; multiple languages can capture models, and some languages are better suited for specific models.
    - **Example**: C++ is suitable for object-oriented models. Hardware implementations often use VHDL, SystemC, or Verilog.
- **Partitioning System Requirements into Hardware and Software**
  - **Decision Making**: Tough to decide whether to implement requirements in hardware or software.
  - **Trade-offs**: Various hardware-software trade-offs are used to make partitioning decisions.

These issues are crucial in the process of hardware-software co-design, affecting the overall system performance, flexibility, and efficiency.

3. **With the functional block diagram, explain the operation of Washing Machine as Application-Specific Embedded system.**

- **Washing Machine as an Embedded System**: A washing machine is a typical example of an embedded system used in home automation applications.
- **Components**: It contains sensors, actuators, a control unit, and application-specific user interfaces like keyboards and display units, some of which are visible while others are not.
- **Actuators**: The actuator part includes a motorized agitator, tumble tub, water drawing pump, and inlet valve to control water flow into the unit.
- **Sensors**: The sensor part consists of a water temperature sensor, level sensor, etc.
- **Control Unit**: The control part contains a microprocessor/controller-based board with interfaces to the sensors and actuators.
- **Feedback Mechanism**: Sensor data is fed back to the control unit, which generates the necessary actuator outputs.
- **User Interface**: The control unit connects to user interfaces like a keypad for setting washing time and selecting material types, with user feedback shown through display units and LEDs.
- **Washing Machine Models**: Washing machines come in two models, top loading and front loading.
- **Top Loading Models**: The agitator twists back and forth, pulling clothes down to the bottom of the tub, where they work their way back up.
- **Front Loading Models**: Clothes are tumbled and plunged into the water repeatedly.
- **Washing Phases**: The first phase involves washing the clothes, while the second phase, called the 'spin phase', uses centrifugal force to wring out more water by spinning the tub at high RPM.
- **Keyboard Panel**: The keyboard panel has buttons like Wash, Spin, and Rinse for configuring washing stages.
- **Spin Cycle**: The inner drum, with multiple holes, spins during the spin cycle, forcing water out through the holes into the stationary outer tub, from which it is drained.
- **Design Variations**: While designs vary among manufacturers, the general working principle remains the same.
- **Basic Controls**: These include a timer, cycle selector, water temperature selector, load size selector, and start button.
- **Mechanism**: The mechanism consists of a motor, transmission, clutch, pump, agitator, inner tub, outer tub, and water inlet valve, which connects to the home water supply.
- **Control Panel**: The integrated control panel includes a microprocessor/controller-based board with I/O interfaces and a control algorithm.
- **Input Interface**: The keyboard allows settings for wash type (Wash, Spin, Rinse), cloth type (Light, Medium, Heavy duty), and washing time.
- **Output Interface**: It includes LED/LCD displays and status indication LEDs connected to the controller's I/O bus.
- **Invisible Interfaces**: These include sensor interfaces (water temperature, water level) and actuator interfaces (motor control for agitator, tub movement control, and water flow control).



4. Explain unique characteristics of embedded systems.

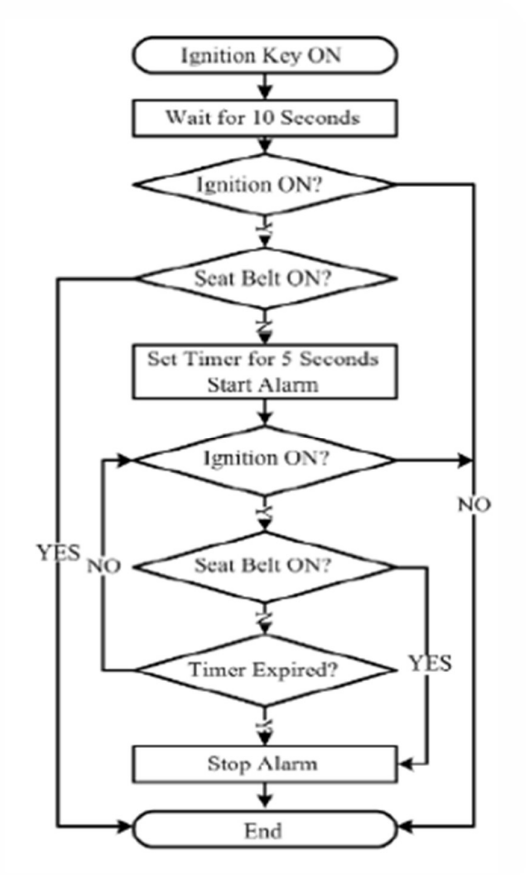
- **Application and Domain Specific**:
  - Embedded systems perform specific functions and cannot be used for other purposes.
  - For example, the embedded control unit of a microwave oven cannot replace that of an air conditioner.
- **Reactive and Real-Time**:
  - Constantly interact with the real world through sensors and user-defined input devices.
  - Systems capture real-time events and react to maintain controlled output.
  - They are reactive systems and may need to operate in real-time, especially for mission-critical applications like flight control systems and Antilock Brake Systems (ABS).
- **Operates in Harsh Environments**:
  - Embedded systems may be deployed in dusty, high-temperature, or high-vibration areas.
  - Design must consider operating conditions, component durability, power supply fluctuations, corrosion, and component aging.
- **Distributed**:
  - Embedded systems can be part of larger systems, forming a single large control unit.
  - Examples include vending machines, ATMs, and SCADA systems in control and instrumentation applications.
- **Small Size and Weight**:
  - Compactness and low weight are significant factors in product aesthetics and convenience.
  - Many applications demand small, lightweight embedded systems.
- **Power Concerns**:
  - Design should minimize heat dissipation to avoid the need for bulky cooling solutions.
  - Use ultra-low-power components like low dropout regulators and processors with power-saving modes.
  - Critical for battery-operated applications where power consumption affects battery life.

5. **What is sequential processing model? Draw a sequential processing model for car seat belt warning system using flow chart.**

Sequential Program Model: In the sequential programming Model, the functions or processing requirements are executed in sequence. It is same as the conventional procedural programming.

- Here the program instructions are iterated and executed conditionally, and the data gets transformed through a series of operations.
- FSMs are good choice for sequential program modeling.
- Another important tool used for modeling 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 functions in a sequential program model for the 'Seat Belt Warning' system is illustrated below.

```
#define ON 1
#define OFF 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()==ON))
                        stop_alarm();
            }
      }
}
```

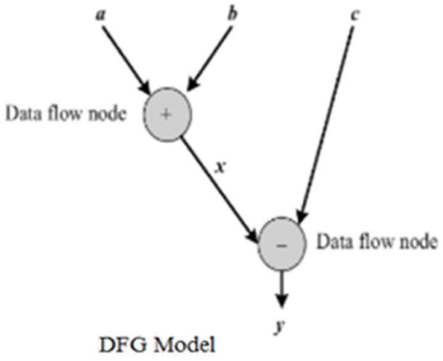


6. Explain Data flow graph and control data flow graph computational model with neat diagram.

Data Flow Graph/ Diagram (DFG) Model: The Data Flow Graph (DFG) model translates data processing requirements into a visual graph where program execution is determined by data flow. In this model, operations on the data (processes) are represented by circles, and data flow is depicted with arrows. An inward arrow to a process represents input data, while an outward arrow represents output data. This model is particularly suited for computationally intensive, data-driven embedded applications.

For example, for computations like x = a + b and y = x - c, the DFG model visually represents these operations with processes and data flow arrows. A data path in a DFG model is the flow from input to output.

A DFG model is said to be acyclic (ADFG) if it doesn't have multiple values for input variables and output values for given inputs. Non-acyclic inputs include feedback loops where output is fed back to input. The DFG model translates the program into a single sequential process execution.

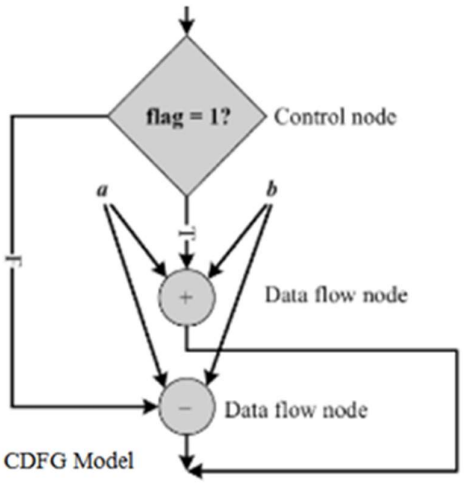


DFG Model

Control Data Flow Graph/ Diagram (CDFG) Model: The Control DFG (CDFG) model is used for applications involving conditional program execution, incorporating both data and control operations. Unlike the Data Flow Graph (DFG) model, which is solely data-driven and lacks conditionals, the CDFG contains both data flow nodes and decision nodes.

In the CDFG model, a decision-making process is represented by a 'Diamond' block, similar to decision elements in flow charts. For instance, the requirement "If flag = 1, x = a + b; else y = a - b" involves a control node that determines which process to execute.

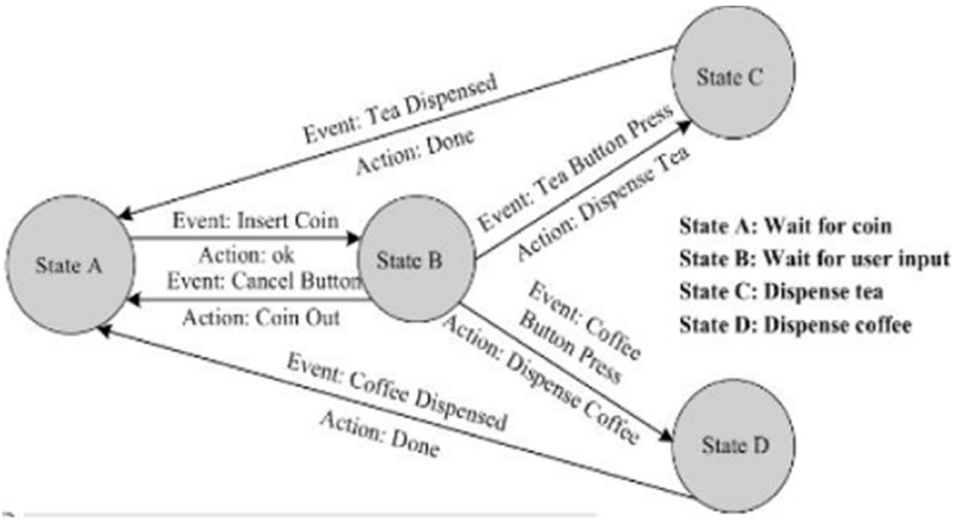
A practical example of the CDFG model is in digital still cameras. The process of capturing and saving images is data-driven, with operations like analog-to-digital conversion and media processing for auto-correction and white balance. The decision on the image format (e.g., JPEG, TIFF, BMP) is controlled by the camera settings configured by the user.



CDFG Model

7. Design an FSM model for Tea/Coffee vending machine.

- The tea/ coffee vending is initiated by user inserting a 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. The FSM representation for the above requirement is given in the following Figure.
- The FSM representation contains four states namely; 'Wait for coin' 'Wait for User Input', 'Dispense Tea' and 'Dispense Coffee'.
- The event ‘Insert Coin’ (5-rupee coin insertion) transitions the state to 'Wait for User Input'. The system stays in this state until a user input is received from the buttons 'Cancel', 'Tea' or 'Coffee'.
- If the event triggered in 'Wait State' is 'Cancel' button press, the coin is pushed out and the state transitions to 'Wait for Coin'. If the event received in the 'Wait State' is either 'Tea' button press, or 'Coffee' button press, the state changes to 'Dispense Tea' or 'Dispense Coffee' respectively.
- Once the coffee/ tea vending is over, the respective states transitions back to the 'Wait for Coin' state.



Module 5

1. Explain the process of choosing an RTOS.

The decision of choosing an RTOS for an embedded design is very crucial. A lot of factors need to be analyzed carefully before deciding on the selection of an RTOS. These factors can be either functional or non-functional.

Functional Requirements:

- **Processor Support:** Ensure the RTOS supports the required processor architecture.
- **Memory Requirements:** Evaluate the minimal ROM and RAM requirements.
- **Real-time Capabilities:** Analyze task/process scheduling policies and real-time standards met by the OS.
- **Kernel and Interrupt Latency:** Minimal interrupt latency is crucial for systems with high response requirements.
- **Inter Process Communication and Task Synchronization:** Availability and implementation of IPC and synchronization mechanisms, including priority inversion handling.
- **Modularization Support:** Ability to choose and compile essential modules to fit the embedded product's needs.
- **Support for Networking and Communication:** Ensure the OS provides stack implementation and driver support for necessary communication interfaces.
- **Development Language Support:** Availability of runtime libraries and virtual machines for languages like Java and C#.

Non-functional Requirements:

- **Custom Developed or Off the Shelf:** Decide between a custom-built OS or an off-the-shelf (commercial or open source) OS based on cost, licensing, development time, and resource availability.
- **Cost:** Evaluate the total cost of development, purchase, and maintenance.
- **Development and Debugging Tools Availability:** Ensure the availability of necessary development and debugging tools.
- **Ease of Use:** Consider the user-friendliness of the commercial RTOS.
- **After Sales Support:** Analyze the availability and quality of after-sales support for bug fixes, critical updates, and production issues.

2. Explain the working of target hardware debugging

Hardware debugging is not similar to firmware debugging. Hardware debugging involves the monitoring of various signals of the target board (address/ data lines, port pins, etc.), checking the inter connection among various components, circuit continuity checking, etc. The various hardware debugging tools used in Embedded Product Development are explained below.

- **Magnifying Glass (Lens):**
  - Used for visual inspection of the target board.
  - Helps identify dry soldering, missing components, improper placement, soldering issues, and track damage.
- **Multimeter:**
  - Measures electrical quantities: voltage (AC and DC), current (AC and DC), resistance, capacitance, and continuity.
  - Essential for physical contact-based debugging and initial hardware diagnostics.
- **Digital Cathode Ray Oscilloscope (CRO):**
  - Captures and analyzes waveforms and signal strength.
  - Useful for analyzing interference noise, power supply lines, and crystal oscillator signals.
  - Digital versions offer high frequency support, advanced waveform recording, and measurement features.
- **Logic Analyzer:**
  - Captures digital data (logic 1 and 0) from digital circuitry.
  - Measures states of port pins, address bus, and data bus.
  - Provides detailed insights into firmware behavior by capturing address and data line logic.
- **Function Generator:**
  - Simulates periodic waveforms (sine, square, saw-tooth) with various frequencies and amplitudes.
  - Used to provide required input signals to the target board during debugging.

3. Show the working of Emulators, Simulator and Debugging

Simulators

Working:

- **Purpose:** Simulators provide a software-based representation of the target hardware. They allow developers to test and debug firmware without needing the actual hardware.
- **Operation:**
  - **Modeling:** The simulator models the CPU and peripherals of the target hardware in software. This model replicates the functionality of the actual hardware.
  - **Firmware Execution:** Developers load the firmware into the simulator. The simulator executes the firmware as if it were running on the real hardware.
  - **Interface:** Simulators typically provide a graphical user interface (GUI) that allows developers to interact with and test the firmware, including I/O operations and UI elements.
  - **Debugging:** Debugging features may include setting breakpoints, stepping through code, and inspecting memory and register states.

Advantages:

- **No Hardware Required:** Firmware development can start without the actual hardware.
- **Simulate Peripherals:** Allows simulation of I/O peripherals and manipulation of I/O values.

Limitations:

- **Deviates from Real Behavior:** Simulated environments might not perfectly replicate real hardware conditions.
- **Lack of Real-Time Behavior:** Simulators cannot fully replicate real-time constraints and variations in hardware behavior.

Emulators

Working:

- **Purpose:** Emulators are hardware devices that replicate the functionality of the target CPU and enable real-time debugging of firmware on the actual hardware.
- **Operation:**
    - **Emulation Device:** Contains hardware that mimics the target CPU's functionality. It receives signals from the target board and executes firmware under debug control.
    - **Emulation Memory:** The emulator has its own RAM that replaces the target board's ROM. This allows for dynamic code updates and avoids the need for ROM burning.
    - **Control Logic:** Implements advanced debugging features such as hardware breakpoints, trace buffers, and logic analyzer functions.
    - **Device Adapters:** Connect the emulator to the target board, providing physical and electrical connections for communication and signal routing.

Advantages:

- **Real-Time Debugging:** Allows debugging of firmware in a real hardware environment.
- **ROM Emulation:** Facilitates code testing and modification without physical ROM changes.
- **Advanced Features:** Supports complex debugging features like hardware breakpoints and trace analysis.

Limitations:

- **Cost:** Emulators can be expensive due to their advanced hardware and capabilities.
- **Setup Complexity:** Requires proper setup and calibration with the target hardware.

Debugging

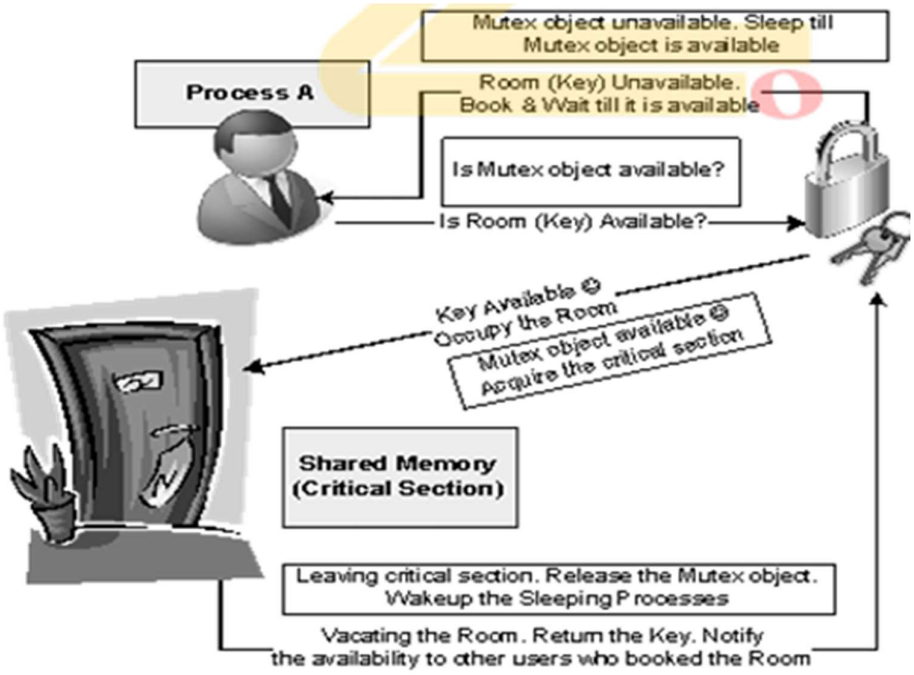
Types and Techniques:

- **Incremental EEPROM Burning:**
    - **Process:** Code is divided into functional units and burned into EEPROM incrementally. This approach simplifies debugging by isolating functional blocks.
    - **Use Case:** Early-stage firmware development where the complete firmware isn't ready.
- **Inline Breakpoint Debugging:**
    - **Process:** Involves inserting printf() statements or similar debug code into the firmware. This code helps in tracking execution flow and verifying behavior at specific points.
    - **Use Case:** Simple debugging to ensure code execution reaches desired points.
- **Monitor Program Based Debugging:**
    - **Process:** A monitor program runs on the target hardware to control firmware download, register/memory inspection, and single stepping. Communication typically happens via a serial interface.
    - **Features:** Includes command set interface, firmware download, register inspection, and debug information transfer.
- **In-Circuit Emulator (ICE) Based Debugging:**
    - **Process:** Uses an emulator hardware device to interface with the target board. The emulator replicates the target CPU and facilitates real-time debugging.
    - **Components:** Emulation device, memory, control logic, and device adapters.
- **On-Chip Debugging (OCD):**
    - **Process:** Utilizes built-in debug modules within the processor/controller for efficient debugging. Features vary by chip vendor and may include proprietary technologies like Background Debug Mode (BDM) or OnCE.
    - **Advantages:** Provides integrated, efficient debugging support directly within the chip.

4. Explain the concept of Binary Semaphore.

Binary Semaphore: Implements exclusive access to shared resource by allocating the resource to a single process at a time and not allowing the other processes to access it when it is being used by a process.

- Only one process/ thread can own the binary semaphore at a time.
- The state of a binary semaphore object is set to signaled when it is not owned by any process/ thread and set to non-signaled when it is owned by any process/ thread.
- The implementation of binary semaphore is OS kernel dependent. Under certain OS kernel it is referred as mutex.



The concept of Binary Semaphore

5. Explain the role of Integrated Development Environment (IDE) for Embedded Software development.

In embedded system development context, Integrated Development Environment (IDE) stands for an integrated environment for developing and debugging the target processor specific embedded firmware. IDE is a software package which bundles

- Text Editor (Source Code Editor)
- Cross-complier (for cross platform development and complier for same platform development)
- Linker
- Debugger.

Some IDEs may provide

- interface to target board emulators,
- target processor's/ controller's Flash memory programmer, etc.

IDE may be command line based or GUI based.