**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
```

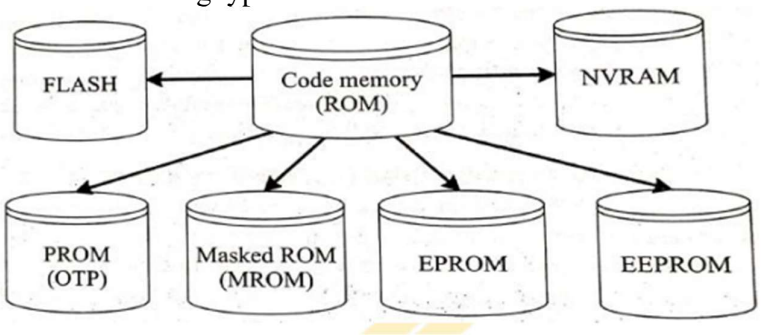
**6)Count the number of 0's and 1's stored in consecutive memory locations.**

```
     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**

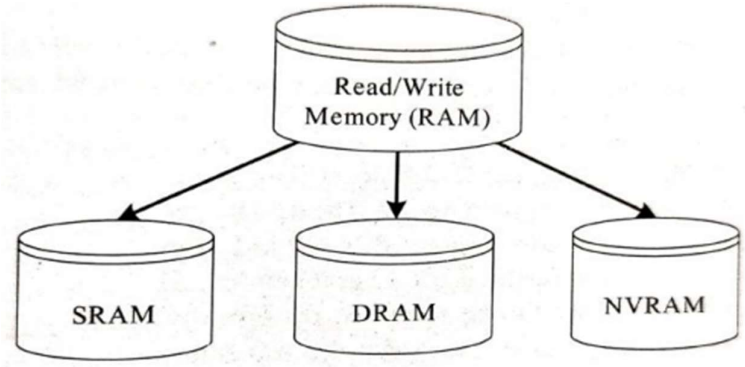
1) **What are the different types of memories used in Embedded System design? Explain the role of each.**

**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**

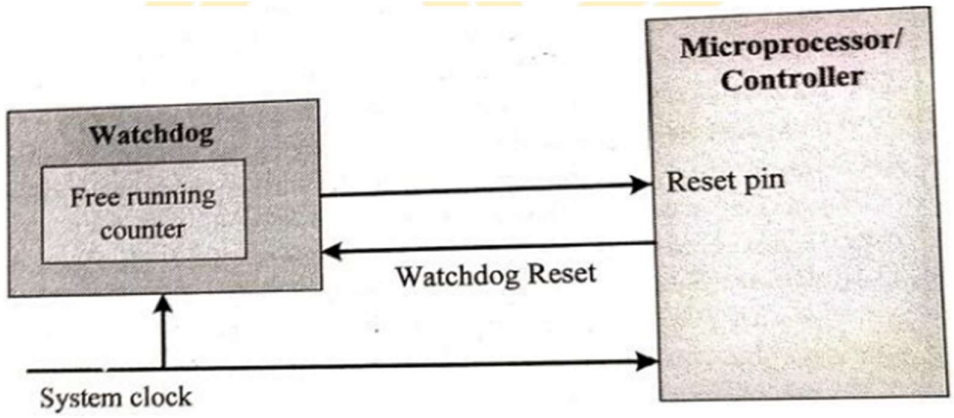
**Real-Time Clock (RTC):** Real-Time Clock is a system component responsible for keeping track of time. RTC holds information like current time (In hours, minutes and seconds) in 12-hour/ 24-hour format, date, month, year, day of the week, etc. and supplies timing reference to the system.

- 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

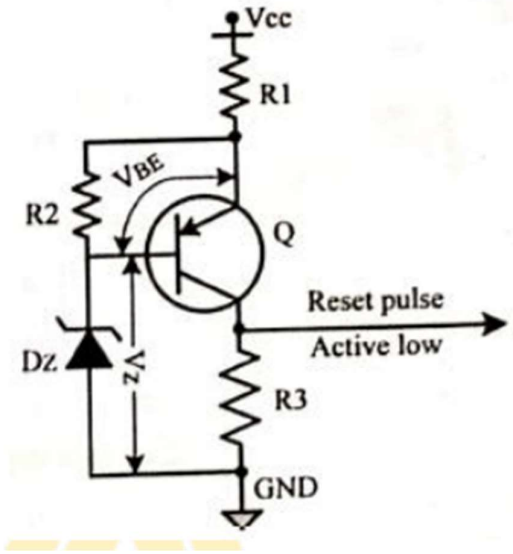
**Reset Circuit:** The reset circuit is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON.

- 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.

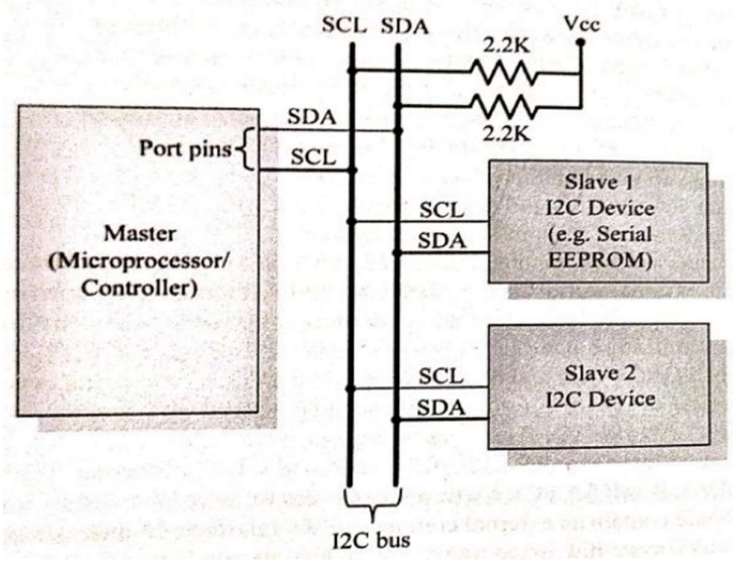
- It is essential for battery powered devices since there are greater chances for the battery voltage to drop below the required threshold. The processor behavior may not be predictable if the supply voltage falls below the recommended operating voltage. It may lead to situations like data corruption.
- 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).

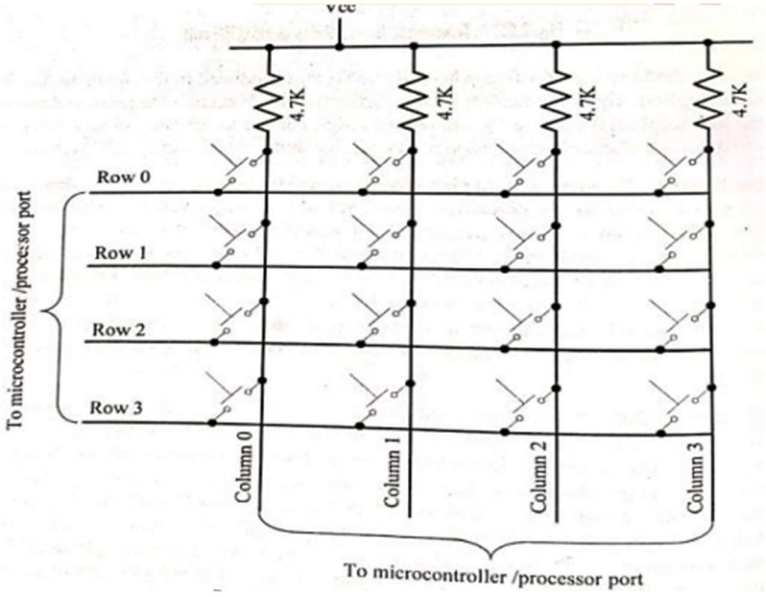



- **Serial Peripheral Interface (SPI)**
- **Universal Asynchronous Receiver Transmitter (UART)**
- **1-Wire Interface**
- **Parallel Interface**

**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.

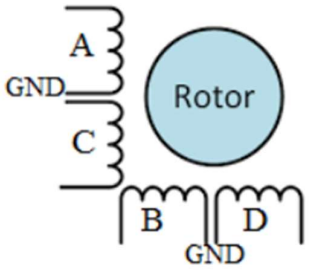
**9) Explain the working of the Stepper Motor.**

**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.

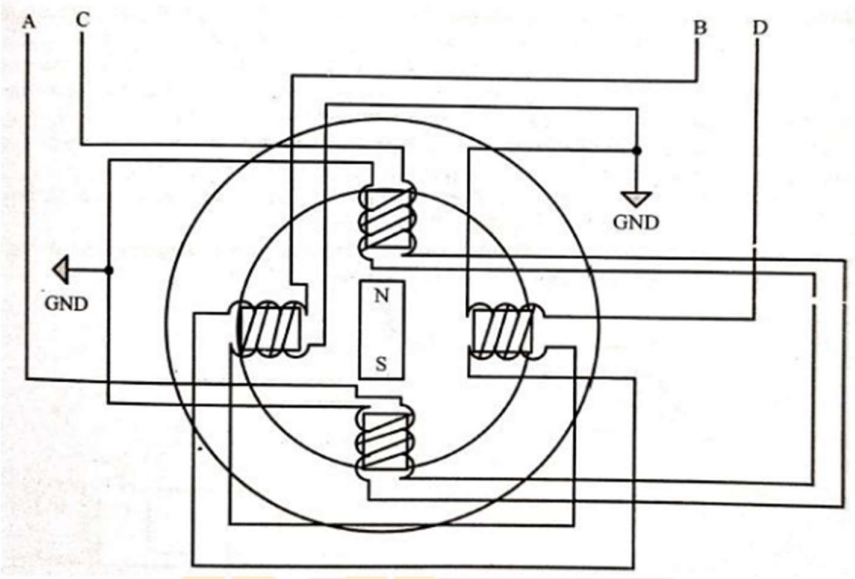
**Wave Step:** In the wave step mode, only one phase is energized at a time and each coils of the phase is energized alternatively. The A, B, C and D are energized in the order, as shown in the following Table.

| 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.