MODULE 3

1) What are addressing modes? Explain each with example.

ANS:

The different addressing modes determine how operands are accessed by the CPU. Each mode

provides flexibility in referencing memory locations and registers, which allows for more efficient and versatile programming.

Below are the addressing modes explained in detail:

1. Register Mode:

In register mode, the operand is located directly in a processor register. The instruction specifies the register where the operand resides. This mode is the fastest, as accessing registers is much quicker than accessing memory.

Example:

ADD R1, R2, R3- Here, R1 and R2 hold the source operands, and R3 is the destination register. The operation ADD adds the contents of R1 and R2 and stores the result in R3.

2. Absolute/Direct Mode

In absolute/direct mode, the operand is located at a specific memory address, and this address is explicitly given in the instruction. This mode is useful when the exact memory location of the operand is known.

Example:

MOV LOC, R1- The instruction moves the value at memory location LOC into register R1.

3. Immediate Mode

In immediate mode, the operand is directly specified in the instruction. The value is a constant or literal that is directly embedded in the instruction.

Example:

ADD #200, R1, R2- This instruction adds the immediate value 200 to the contents of register R1, and places the result into register R2.- The # symbol indicates that 200 is an immediate operand.

4. Indirect Mode

In indirect mode, the instruction specifies a register or memory location that contains the effective address of the operand. This mode allows for more flexible access to data by using pointers.

Example:

ADD (R1), R0- The effective address of the operand is in register R1. The processor first reads the value in R1, which gives the memory address of the operand, and then accesses that memory address.- The operand is added to the contents of register R0.

5. Index Mode:

In index mode, the effective address of the operand is calculated by adding a constant value (displacement) to the contents of a register. The register used in this mode is called the index register.

Example:

MOV 10(R1), R2- The instruction adds 10 to the contents of register R1 to calculate the effective address. It then accesses the operand at that address and moves it into register R2.

6. Relative Addressing Mode:

In relative addressing mode, the effective address is determined by adding the displacement to the program counter (PC). This mode is commonly used for branch instructions.

Example:

BRANCH >0 LOOP- This instruction tells the processor to branch to the address of LOOP if the branch condition is satisfied.- The address of LOOP is calculated relative to the program counter (PC).

7. Auto-Increment Mode:

In auto-increment mode, the effective address of the operand is stored in a register. After accessing the operand, the register is automatically incremented to point to the next item (e.g., in a list or array).

Example:

MOV (R1)+, R2- The operand is accessed from the address in register R1, and then R1 is incremented by the size of the operand (often the size of a word or byte).- The value at the address in R1 is moved into register R2.

8. Auto-Decrement Mode:

In auto-decrement mode, the register is first decremented, and then the effective address of the operand is accessed. This mode is commonly used in stack operations.

Example:

MOV -(R1), R2- The instruction decrements the value in register R1 first, then accesses the memory address pointed to by the updated value of R1.

- The value at the new address is moved into register R2.

Summary of Addressing Modes

Mode	Description	Example
Register Mode	Operand is in a register.	`ADD R1, R2, R3`
Absolute/Direct Mode	Operand is in a memory location, directly specified in the instruction.	`MOV LOC, R1`
Immediate Mode	Operand is directly specified in the instruction.	`ADD #200, R1, R2`
Indirect Mode	Operand is at a memory address stored in a register or memory location.	`ADD (R1), R0`
Index Mode	Effective address is the sum of a constant and the contents of a register.	`MOV 10(R1), R2`
Relative Mode	Effective address is calculated relative to the program counter (PC).	`BRANCH >0 LOOP`
Auto-Increment Mode	Operand is accessed, and the register is incremented after access.	`MOV (R1)+, R2`
Auto-Decrement Mode	Register is decremented before accessing the operand.	`MOV -(R1), R2`

2) Explain the basic operation concepts of the computer with a neat diagram.

ANS:

Operating Steps in a Computer System

These steps outline how the processor fetches, decodes, and executes instructions, as well as how input/output (I/O) operations are handled through interrupts.

Steps for Program Execution:

1. **Program in Memory**: The program is loaded into memory, usually through the **Input Unit**. The processor will start execution based on the instructions provided in the program.
2. **Program Counter (PC) Initialization**: Execution starts when the **Program Counter (PC)** is set to point to the first instruction of the program. The PC is responsible for holding the address of the instruction that needs to be executed next.
3. **Fetching Instruction**:
    - The contents of the **PC** (address of the next instruction) are transferred to the **Memory Address Register (MAR)**.
    - A **Read Control Signal** is sent to the memory to fetch the instruction at the address held in **MAR**.

4. **Memory Access**: After the memory access time has elapsed, the instruction is read from memory and transferred into the **Memory Data Register (MDR)**.
5. **Loading Instruction to Instruction Register (IR)**: The contents of the **MDR** are transferred to the **Instruction Register (IR)**. The instruction is now ready to be decoded and executed.
6. **Operation by ALU**: If the instruction involves an operation to be performed by the **Arithmetic Logic Unit (ALU)**, the required operands must be obtained.
7. **Fetching Operand from Memory**:
    - If the operand is stored in memory, its address is sent to **MAR**, and a read cycle is initiated to fetch the operand.
8. **Loading Operand to ALU**: Once the operand is fetched from memory into the **MDR**, it is transferred to the **ALU** for processing.
9. **ALU Operation**: After performing the required operations, the ALU produces the result.
10. **Storing the Result in Memory**: If the result from the ALU needs to be stored back in memory:
    - The result is transferred to the **MDR**.
    - The address where the result should be stored is sent to **MAR**, and a **Write Cycle** is initiated to store the result.
11. **Update Program Counter**: After completing the execution of the current instruction, the **PC** is incremented so it points to the next instruction to be executed.

Interrupts:

- **Interrupt**: An interrupt is a signal from an I/O device requesting immediate attention from the processor. When an interrupt is raised, it temporarily halts the current execution and switches to handle the interrupt.
- **Interrupt Service Routine (ISR)**: When an interrupt occurs, the processor executes a predefined set of instructions called the interrupt service routine to handle the interrupt. After the interrupt is serviced, the processor returns to the previous program execution.

3) Explain basic performance equations of computer

ANS :

Basic Performance Equation of a Computer

The **basic performance equation** is used to estimate the **processor time** required to execute a program in a computer system. This time is an important factor when measuring the performance of a processor, as it directly influences the overall speed of the computer when running a program.

Basic Performance Equation:

The processor time T is given by the equation:

$$T = \frac{N \times S}{R}$$

Where:

- **N** is the number of machine instructions,
- **S** is the number of basic steps per instruction (each step takes one clock cycle),
- **R** is the processor's clock rate (in cycles per second, Hz).

Performance Equation Variables:

- **T**: **Processor time** required to execute a program that has been prepared in a high-level language.
- **N**: **Number of actual machine language instructions** needed to complete the execution. This includes the total number of instructions, including those in loops and any other operations performed during program execution.
- **S**: **Average number of basic steps** needed to execute one machine instruction. Each step completes in one clock cycle. These basic steps could include operations like fetching, decoding, executing, and storing results.
- **R**: **Clock rate** of the processor, usually measured in cycles per second (Hertz, Hz). This indicates the speed at which the processor's clock operates.

Explanation:

1. **Number of Instructions (N)**: This is the total count of instructions required by the machine to execute the program. The number of instructions depends on the program's complexity, including loops, branches, and the operations involved in the program.
2. **Average Number of Steps per Instruction (S)**: This is the number of fundamental steps (like fetching, decoding, executing, and writing back) that need to be performed for each machine instruction. These steps depend on the processor architecture and how efficiently the processor can handle each instruction.
3. **Clock Rate (R)**: The clock rate determines how fast each cycle occurs, essentially controlling how fast each step can be completed. A higher clock rate means more cycles per second, thus reducing the time required for each step.

MODULE 4

1) What are the different modes of data transfer between CPU and io device .Explain interrupt driven io technique.

ANS :

Modes of Data Transfer Between CPU and I/O Devices

There are several modes for transferring data between the CPU and I/O devices. These modes define how data is moved between the CPU and memory, and how I/O devices interact with the system. The main modes are:

1. **Programmed I/O (PIO)**
2. **Interrupt-Driven I/O**
3. **Direct Memory Access (DMA)**

Interrupt-Driven I/O Technique:

In interrupt-driven I/O, the CPU is alerted by the I/O device when it needs to perform a data transfer, rather than the CPU constantly checking the device's status. This method uses interrupts, which are signals generated by hardware (I/O devices) or software to notify the CPU of an event that needs immediate attention.

How Interrupt-Driven I/O Works:

1. **Device Ready for Data Transfer:**
    - **The I/O device sends an interrupt request (IRQ) signal to the CPU when it is ready for data transfer (e.g., data is ready to be read from an I/O device or space is available for data to be written).**
2. **Interrupt Handling:**
    - **When the CPU receives the interrupt signal, it temporarily halts its current task (suspends the execution of the current program).**
    - **The CPU saves its state (e.g., register values, program counter) so that it can return to the current task after handling the interrupt.**
3. **Interrupt Service Routine (ISR):**
    - **The CPU branches to the Interrupt Service Routine (ISR), which is a special program written to handle the interrupt. The ISR will take the necessary actions, such as reading data from an I/O device or transferring data to memory.**
    - **After completing the I/O transfer, the ISR signals the CPU that the operation is complete.**
4. **Return to Main Program:**

- **Once the ISR completes the I/O task, the CPU restores its previous state (using the saved program counter and registers) and returns to the task it was executing before the interrupt**

Example of Interrupt-Driven I/O:

Imagine a scenario where a CPU is reading data from a disk:

1. The CPU sends a read command to the disk.
2. While the disk is processing the read request, the CPU can perform other operations.

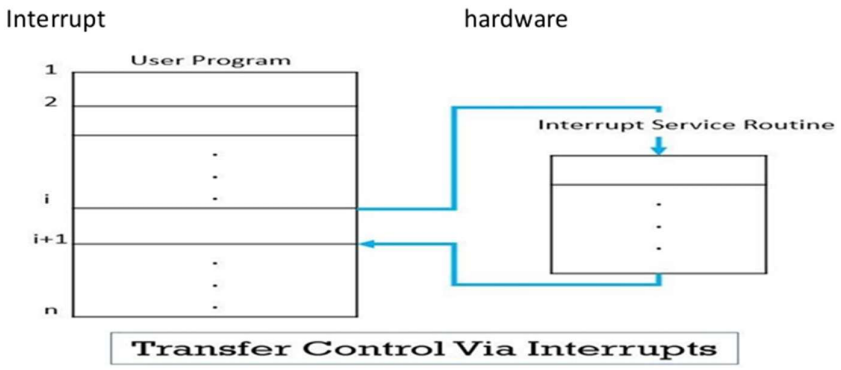


Fig 4.3 Transfer control via Interrupts

2) What are the different techniques to handle multiple interrupts (poling, vectors, interrupt nesting explain full).

ANS:

Handling Multiple Devices with Interrupts

When multiple devices generate interrupt requests (IRQ) at the same time, the system must decide how to handle them effectively. The processor needs a way to prioritize and identify which interrupt request should be serviced first. To handle this, there are several methods:

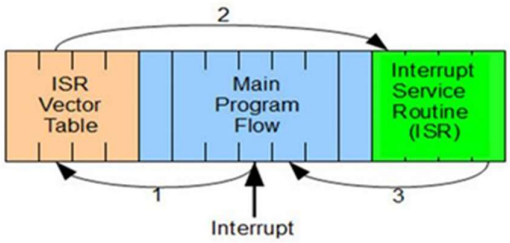
1. **Polling**
2. **Vectored Interrupts**
3. **Interrupt Nesting**

1. Polling:

In **polling**, the processor repeatedly checks the interrupt request (IRQ) bits of all devices to see if any device needs attention. When a device has its IRQ bit set, the CPU services it by calling the appropriate interrupt service routine (ISR).

How Polling Works:

- The CPU checks each device in a sequential manner to see if the IRQ bit is set.
- The first device with an IRQ bit set is serviced by calling its corresponding ISR.
- The process continues until all devices are checked.



Disadvantages of Polling:

- **Inefficiency**: The CPU wastes time checking the IRQ bits of all devices, even if most devices are not requesting service.

Example of Polling:

In polling, the CPU might check devices like this:

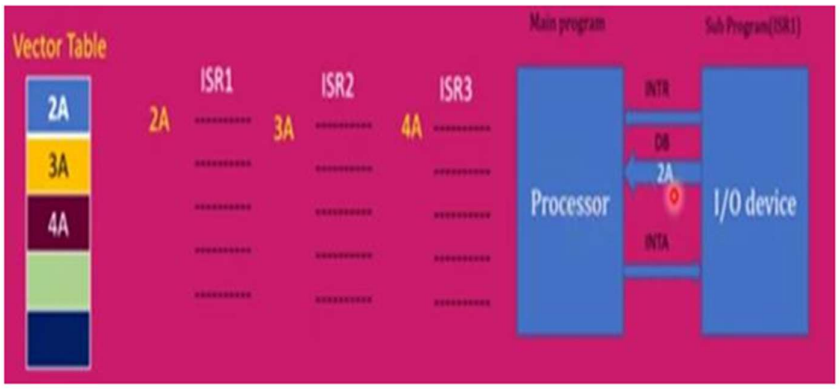
- Check Device 1: Is IRQ set? Yes → Service Device 1.
- Check Device 2: Is IRQ set? No → Skip to next device.
- Check Device 3: Is IRQ set? Yes → Service Device 3.

2. Vectored Interrupts:

In **vectored interrupts**, each device that raises an interrupt sends a **special code** (a vector) to the processor over the bus. This code helps the CPU directly identify the device requesting service and allows the CPU to jump to the corresponding ISR without checking each device individually.

How Vectored Interrupts Work:

- Each device is assigned a unique **vector code** (often 4 to 8 bits), which the device sends to the CPU.
- The vector code tells the CPU the starting address of the appropriate ISR in memory.
- The CPU then jumps to the specific ISR associated with the device.



Advantages of Vectored Interrupts:

- **Faster Response**: The CPU directly knows which ISR to execute, without having to check all devices.
- **Efficiency**: It reduces the time spent by the CPU checking each device.

Example of Vectored Interrupts:

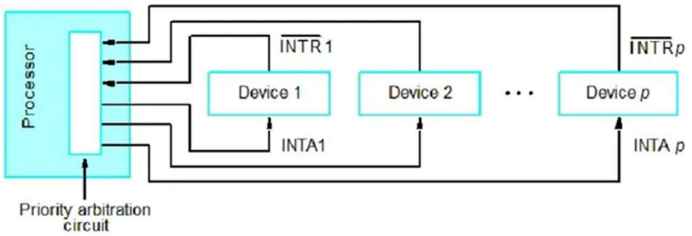
- Device 1 sends a vector "2A" to the processor. The processor knows that "2A" corresponds to ISR1 and directly jumps to ISR1.
- Device 2 sends "3A", and the processor jumps to ISR2, and so on.

3. Interrupt Nesting:

In **interrupt nesting**, interrupts are prioritized. A higher-priority interrupt can interrupt a lower-priority interrupt being serviced by the CPU. This method ensures that critical interrupts are handled immediately, even if a less important interrupt is currently being processed.

How Interrupt Nesting Works:

- Devices are organized in a **priority structure**, where each device is assigned a priority level.
- Interrupt requests from higher-priority devices are serviced first, even if the CPU is already servicing a lower-priority interrupt.
- Interrupt requests are passed to a **priority arbitration circuit** that determines the priority of each interrupt and allows the processor to service the most urgent request first.



Advantages of Interrupt Nesting:

- **Prioritization**: Critical interrupts are handled immediately, ensuring that time-sensitive tasks are not delayed.

- **Efficiency**: Devices with higher priority are serviced first, improving system responsiveness.

Example of Interrupt Nesting:

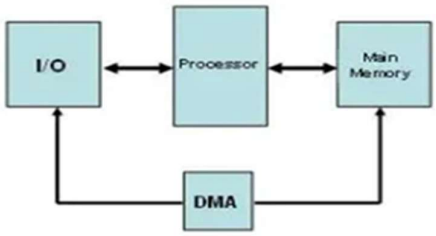
- **Device 1 (high priority)** sends an interrupt request while **Device 2 (low priority)** is being serviced. Since Device 1 has a higher priority, its interrupt will interrupt the service of Device 2, and the processor will handle Device 1 first.

3) Explain DMA(direct memory access)

ANS :

Direct Memory Access (DMA)

Direct Memory Access (DMA) is a method of data transfer that allows peripheral devices to directly access the main memory without involving the CPU. DMA enables faster data transfers by bypassing the CPU, allowing it to perform other tasks while the transfer is happening. This process is managed by a special control unit known as the DMA controller.



Key Concepts of DMA:

1. **DMA Controller**: A hardware component that handles the data transfer between memory and I/O devices without involving the CPU.
2. **Data Transfer**: The DMA controller directly manages data transfers between peripheral devices (like hard drives or network cards) and memory. The CPU is not involved in these transfers, except for initiating the process and handling completion.
3. **Efficiency**: By offloading data transfer tasks to the DMA controller, the CPU can continue performing other tasks, improving overall system efficiency.

How DMA Works:

1. The CPU initiates the DMA transfer by setting up the DMA controller with the starting memory address and the number of data bytes to transfer.
2. The DMA controller takes control of the system's bus to perform the data transfer between the peripheral device and memory.
3. The DMA controller reads or writes data directly between the I/O device and the main memory without the intervention of the CPU.
4. Once the transfer is complete, the DMA controller signals the CPU via an interrupt, allowing the CPU to resume its normal operations.

DMA and System Performance:

DMA allows systems to perform data transfers at high speed by reducing the load on the CPU. It is especially beneficial in data-intensive tasks like disk I/O operations or large data transfers between devices and memory.

Speed, Size, and Cost of Memory

In computer systems, the trade-offs between speed, size, and cost of memory are significant considerations:

1. **Speed and Size**:
    - To achieve faster memory access, SRAM (Static RAM) can be used, which is faster but expensive. Due to its high cost, it's not practical to use SRAM for large memory systems.
    - On the other hand, DRAM (Dynamic RAM) is slower than SRAM but much cheaper and can be used for large main memory systems.
2. **Impact of DRAM**:
    - DRAM is slower compared to the CPU, which can result in the need for wait states during memory read/write cycles. This can slow down the execution of programs.
3. **Cache Memory**:
    - Cache memory is used to address the performance limitations of DRAM. It stores frequently accessed code and data to speed up access times.
    - The active portion of the program is moved from main memory (DRAM) to cache memory, where it can be accessed more quickly.
    - There are two types of cache:
        - **Primary Cache**: Built into modern processors.
        - **Secondary Cache**: Located between the processor and main memory.
    - The **DMA controller** plays a role in managing the swapping of data between the main memory and cache memory, ensuring efficient performance.

Memory Hierarchy