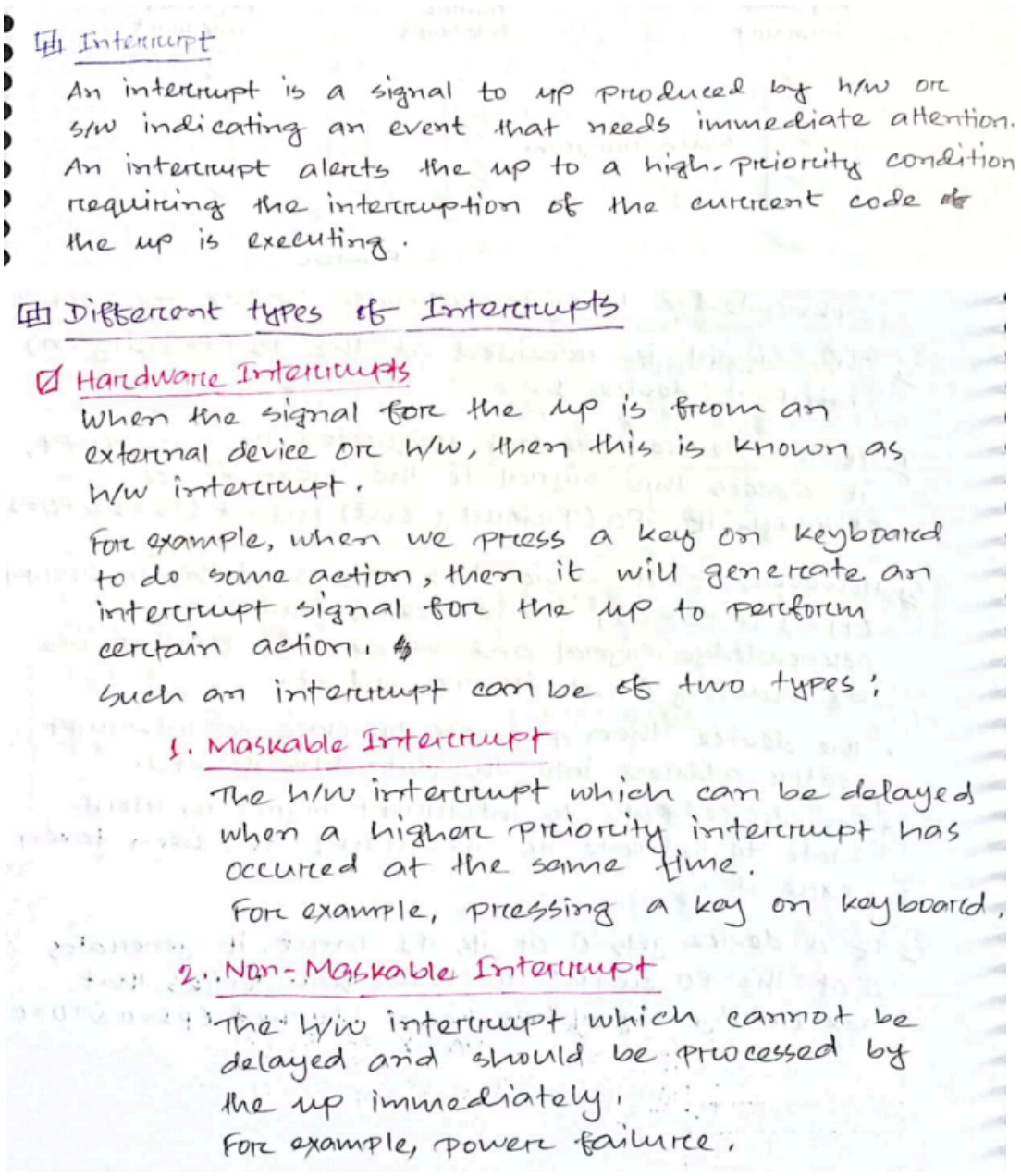
Microprocessor Previous Solve by 5CF (Autumn-23)

1(a) What is interrupt? Describe the types of interrupt with example.

Ans:



Interrupt

An interrupt is a signal to µp produced by h/w or s/w indicating an event that needs immediate attention. An interrupt alerts the µp to a high-priority condition requiring the interruption of the current code the µp is executing.

Different types of Interrupts

☑ Hardware Interrupts

When the signal for the µp is from an external device or h/w, then this is known as h/w interrupt.

For example, when we press a key on keyboard to do some action, then it will generate an interrupt signal for the µp to perform certain action.

Such an interrupt can be of two types;

1. Maskable Interrupt

The h/w interrupt which can be delayed when a higher priority interrupt has occured at the same time.

For example, pressing a key on keyboard,

2. Non-Maskable Interrupt

The h/w interrupt which cannot be delayed and should be processed by the µp immediately.

For example, power failure.

☑ Software Interrupts

The interrupts that is caused by any internal system of computer is known as a s/w interrupt. It can be of two types:

1. Normal Interrupt

The interrupts that are caused by s/w instructions are called normal s/w interrupts.

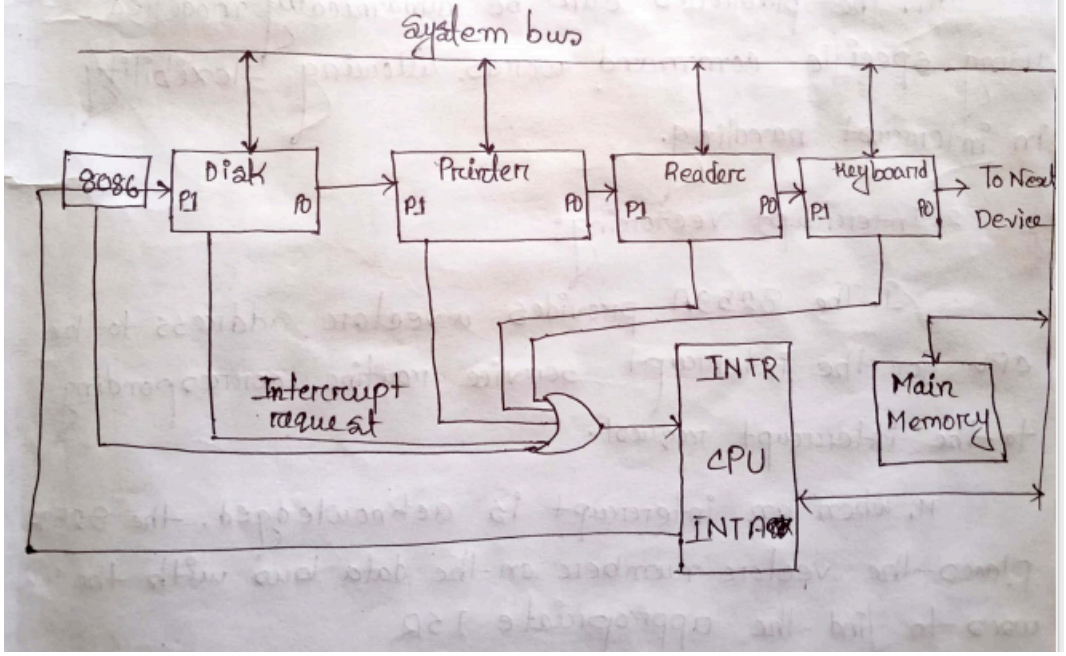
2. Exception

Unplanned interrupts which are produced during the execution of some programs, such as, division by zero.

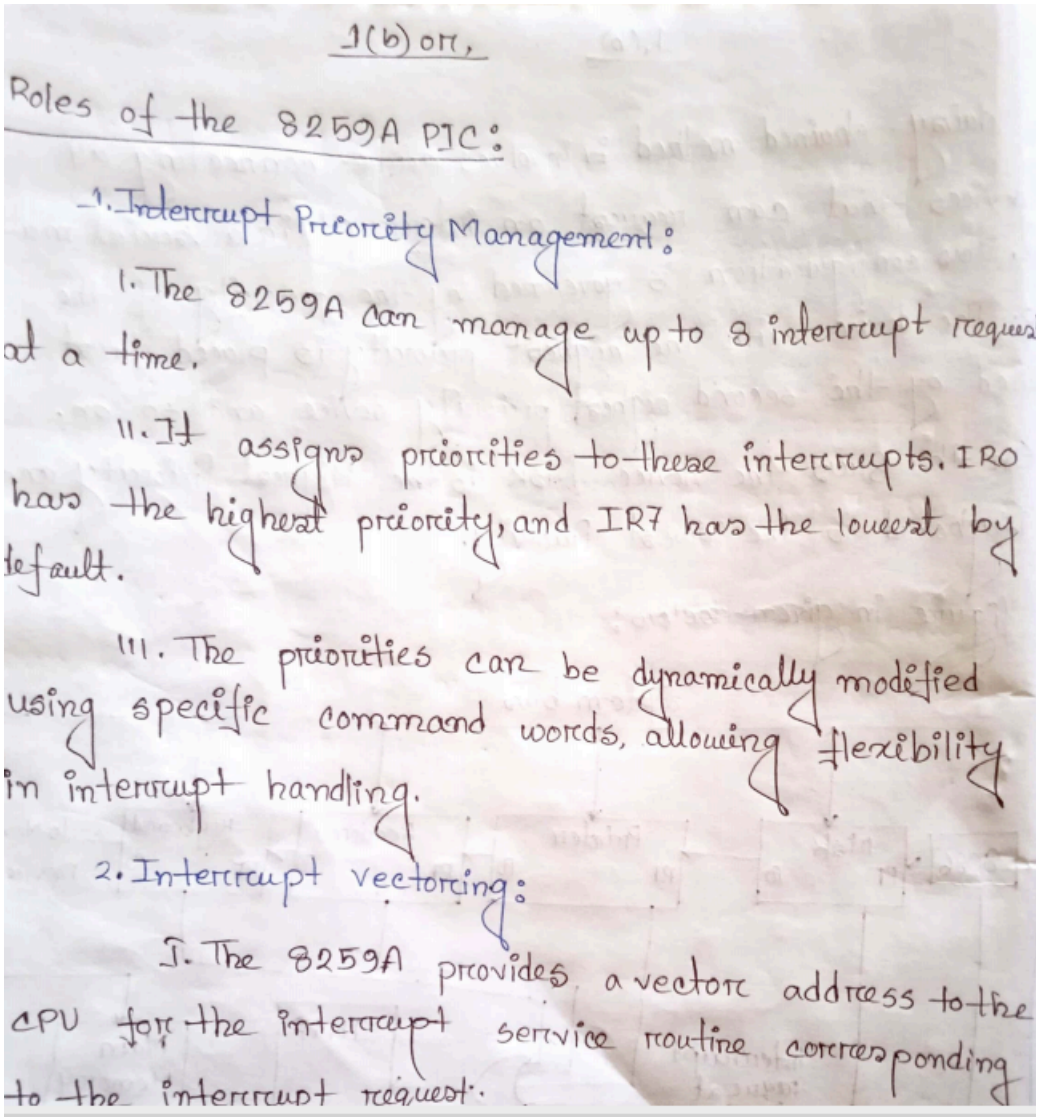
1(b) Suppose you have four input devices: Disk, Printer, Reader, Keyboard. All these devices are sending interrupt to the Microprocessor 8086. Note that the devices names are listed in descending order of priority. How does Microprocessor 8086 handle the above situation in the daisy chain process? Explain the handling process with the proper figure.

The daisy-chained method involves connecting all the devices that can request an interrupt in a serial manner. This configuration is governed by the priority of the devices. The device with the highest priority is placed first followed by the second highest priority device and so on. In the ques the device Disk is the Highest Priority and the keyboard is the lowest Priority.

The figure in given below:



1(b) or: What role does the 8259A Programmable Interrupt controller (PIC) play in managing interrupt priorities and vectoring in microprocessor system and how does the expand –ability of the 8259A to accept up to 64 interrupt request impact the scalability and performance of this system.



1(b) or,

Roles of the 8259A PIC:

1. Interrupt Priority Management:

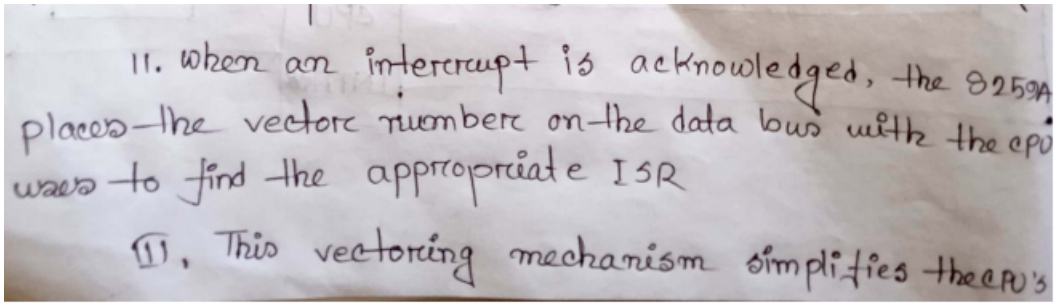
i. The 8259A can manage up to 8 interrupt reques at a time.

ii. It assigns priorities to these interrupts. IR0 has the highest priority, and IR7 has the lowest by default.

iii. The priorities can be dynamically modified using specific command words, allowing flexibility in interrupt handling.

2. Interrupt vectoring:

i. The 8259A provides a vector address to the CPU for the interrupt service routine corresponding to the interrupt request.



ii. When an interrupt is acknowledged, the 8259A places the vector number on the data bus with the CPU uses to find the appropriate ISR

iii. This vectoring mechanism simplifies the CPU's

task of identifying and Jumping to the corrcect ISR.

Expanability to 64 interrupt:

I. By cascading a master 8259A with up to 8 slave 8259A controllers, the system can handle up to 64 interrup requests.

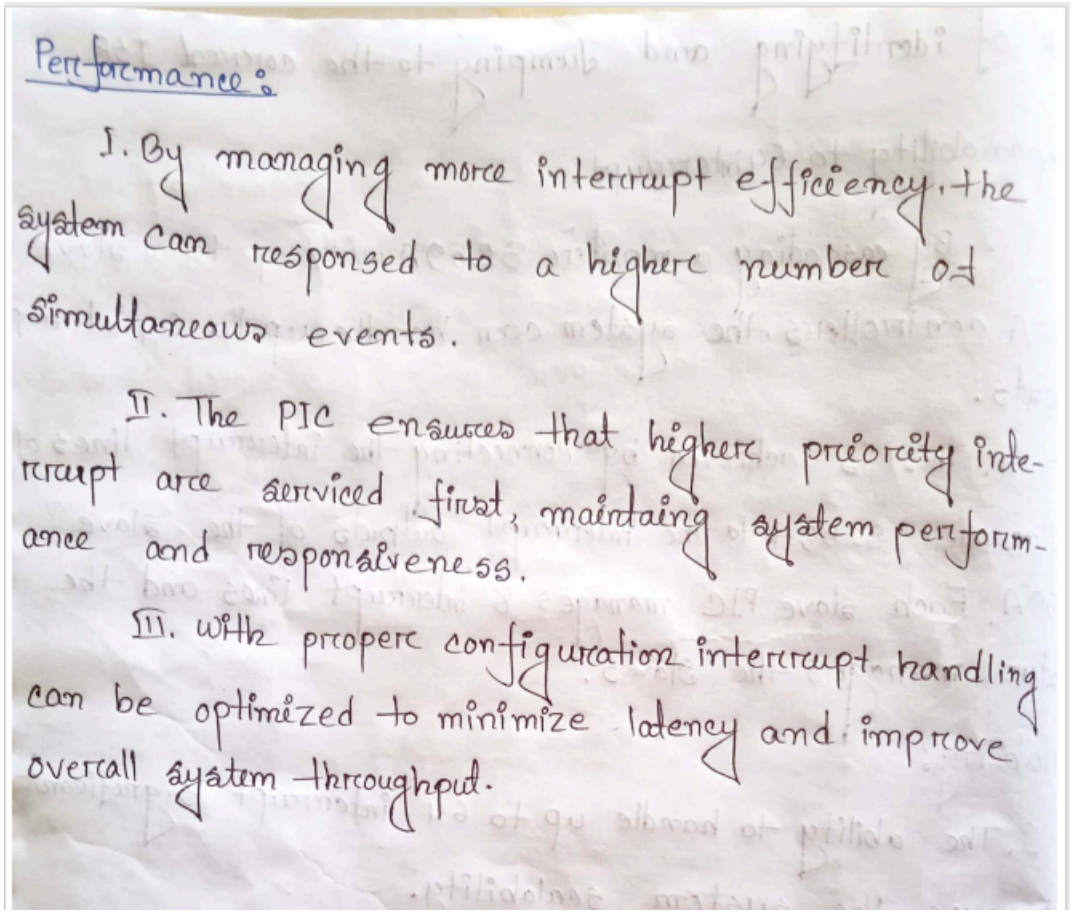
II. This is achieved by connecting the interrupt lines of the master 8259A to the interrupt outputs of the slave 8259A. Each slave PIC manages 8 interrupt lines and the master manages the slaves.

Scalability:

I. The ability to handle up to 64 interrupt significant enhances the system scalability.

II. It allows the additional of morre peripherals and devices without needing major changes to the system architecture.

III. As a new device can be added each with its own interrupt line, the system can grow in complexity &



2(a) What is the significant of using interrupt in the context of keyboard input and how does the generation of a single pulse by the keyboard encode of the system? Explain with timeline diagram.

Answer:

Using interrupts for keyboard input in an embedded system is significant because it allows the system to respond immediately to key presses without constantly checking the keyboard status, which would waste valuable processing time. Interrupts make the system more efficient and responsive.

Significance of Using Interrupts for Keyboard Input

1. Efficiency: The CPU can perform other tasks while waiting for a key press. It doesn't have to continuously check the keyboard status (polling).

2. Immediate Response: The system can respond quickly to key presses because the interrupt signals the CPU to handle the input as soon as it occurs.

3. Resource Management: Interrupts free up system resources for other processes, improving overall performance.

How a Keyboard Interrupt Works

1. Key Pressed: When a key on the keyboard is pressed, it generates an electrical signal.

2. Interrupt Signal: This signal triggers an interrupt request (IRQ) to the CPU.

3. Interrupt Handler: The CPU temporarily stops its current task and executes a special function called an interrupt handler or interrupt service routine (ISR).

4. Process Key Input: The ISR reads the key code from the keyboard buffer and processes it (e.g., storing it in memory).

5. Resume Task: After processing the key press, the CPU resumes its previous task.

Timeline Diagram

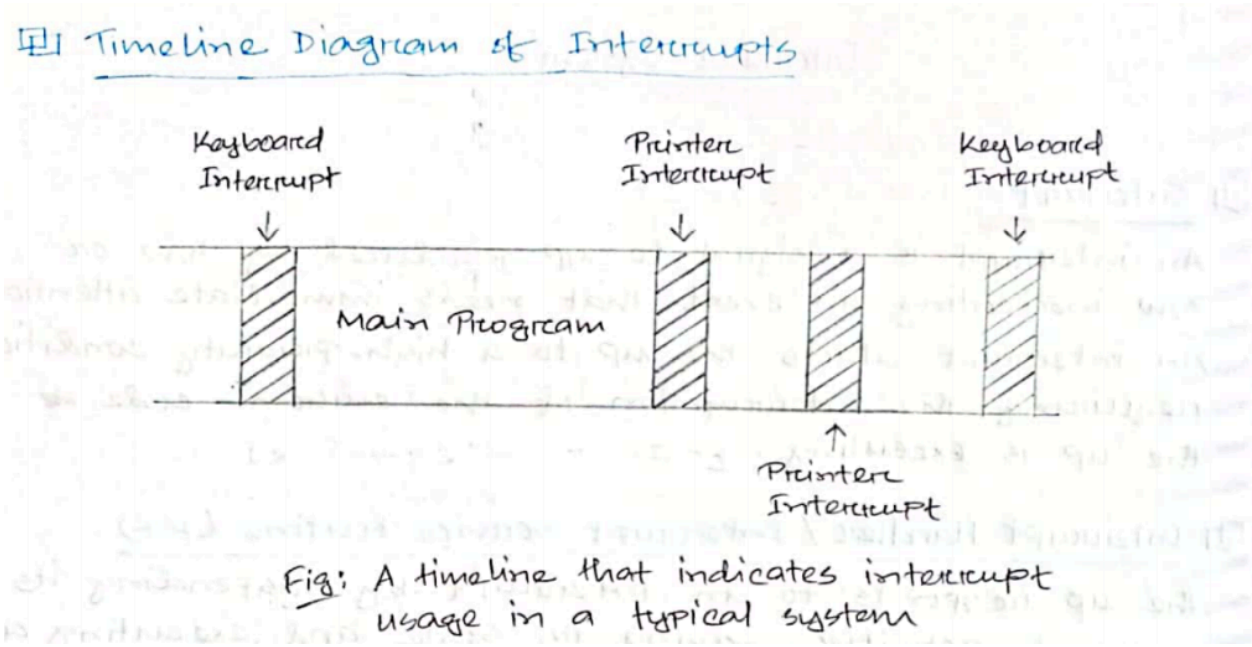


Fig: A timeline that indicates interrupt usage in a typical system

Explanation with Example

The timeline above shows typing on a keyboard, a printer removing data from memory and a program execution. The keyboard interrupt service procedure and the printer interrupt service procedure take little time to execute.

3.a)

What are the fundamental differences between isolated I/O and memory-mapped I/O methods, and how do these two approaches affect the interaction between a microprocessor and external devices in a computer system?

Isolated I/O	Memory-mapped I/O
Different address spaces are used for computer memory and I/O devices. I/O devices have dedicated address space.	Same address space is used for memory and I/O devices.
Separate control unit and control instructions are used in case of I/O devices.	Control units and instructions are same for memory and I/O devices.
More complex and costlier than memory -mapped I/O as more bus are used.	Easier to build and cheap as it's less complex.
Entire address space can be used by memory as I/O devices have separate address space.	Some part of the address space of computer memory is consumed by I/O devices as address space is shared.
Computer memory and I/O devices use different control instructions for read write.	Computer memory and I/O devices can both use same set of read and write instructions.
Separate control bus is used for computer memory and I/O devices. Though same address and data bus are used.	Address, data and control bus are same for memory and I/O devices.

Impact on Microprocessor and External Device Interaction:

Isolated I/O:

Addressing and Access:

Complexity in Addressing: The microprocessor uses specific instructions and dedicated control signals for I/O operations, which can add complexity but ensure clear separation between memory and I/O.

Performance:

Potential Latency: Access to I/O ports might be slower compared to memory due to the need for specific I/O instructions and control signals.

Device Compatibility:

Easier Management: Easier to manage and avoid address conflicts since I/O devices are in a separate address space.

Software Design:

Specific Instructions Required: Device drivers must use specific I/O instructions, which can complicate software development and maintenance.

Memory- Memory-Mapped I/O:

Addressing and Access:

Simplified Addressing: The same set of instructions and control signals are used for both memory and I/O operations, simplifying the CPU design and programming model.

Performance:

Improved Efficiency: Potentially faster access to I/O devices as the same memory access instructions are optimized for speed.

Address Conflicts:

Careful Management Needed: Requires careful planning to avoid address conflicts since I/O devices share the same address space as memory.

Software Design:

Standard Instructions: Device drivers can use standard memory instructions, simplifying development and providing greater flexibility.

Both isolated I/O and memory-mapped I/O have their advantages and disadvantages, affecting how the microprocessor interacts with external devices:
Choosing between these two approaches depends on the specific requirements of the system, including performance needs, complexity, and the number of I/O devices.

3.b)
How can you perform Multiplication and Division operations in Microcontroller 8051? Explain the process.

Multiplication and division operations are performed in the **8051 microcontroller**.

Multiplication in 8051:

The 8051 microcontroller provides an **MULAB** instruction to perform multiplication. Here's how it works:

1. Registers Used:

- **A (Accumulator)**: Holds one of the operands.
- **B:** Holds the other operand.

2. Steps for Multiplication:

- Load the two 8-bit numbers (operands) into registers **A** and **B**.
- Execute the **MULAB** instruction, which multiplies the contents of **A** and **B**.

- The result is stored in the accumulator (**A**), and the higher-order byte is stored in register **B**.

3. Example:

- Let's multiply two 8-bit numbers: **FFH** and **FFH**.
- After multiplication, the result will be stored as follows:
- Higher-order byte (from **B**): **FEH**
- Lower-order byte (from **A**): **01H**

Division in 8051:

To perform division in the 8051 microcontroller, we'll use the following steps:

1.Registers Used:

- **A (Accumulator)**: Contains the dividend (number to be divided).
- B: Contains the divisor (number by which division is performed).

2. Steps for division:

- Load the dividend into the accumulator (**A**) and the divisor into register **B**.
- Execute the **DIV AB** instruction, which divides the contents of **A** by **B**.
- The quotient is stored in the accumulator (**A**), and the remainder is stored in register **B.**

2. Example:

- Let's divide two 8-bit numbers: **40H** (dividend) and **0F0H** (divisor).
- After division, the result will be as follows:

Quotient (from A): 42H
Remainder (from **B**): **0F0H**

3. b) OR

What are the primary components involved in basic input and output operations within a microprocessor system, and how are these operations defined in terms of IN and OUT, referring to data movement between the microprocessor and I/O devices? Explain with proper diagram.

Answer:

Components Involved in I/O Operations:

7. Central Processing Unit (CPU):

- The CPU (also known as the microprocessor) is the heart of the system. It performs all data processing and control functions.
- It fetches instructions from memory, decodes them, and executes them using its Arithmetic and Logical Unit (ALU).
- The CPU interacts with I/O devices through specific instructions.

8. Memory:

- Memory stores both program instructions and data.
- The CPU reads instructions from memory and writes data back to memory during I/O operations.

9. Input and Output Devices (I/O Devices):

- I/O devices include peripherals such as keyboards, mice, displays, printers, sensors, and communication interfaces.
- These devices allow data to move between the microprocessor and the external world.

10. Interfacing Devices:

- Interfacing devices connect the CPU to I/O devices.
- They handle the electrical and protocol-level communication between the CPU and peripherals.

Data Movement Using IN and OUT Instructions:

IN Instruction:

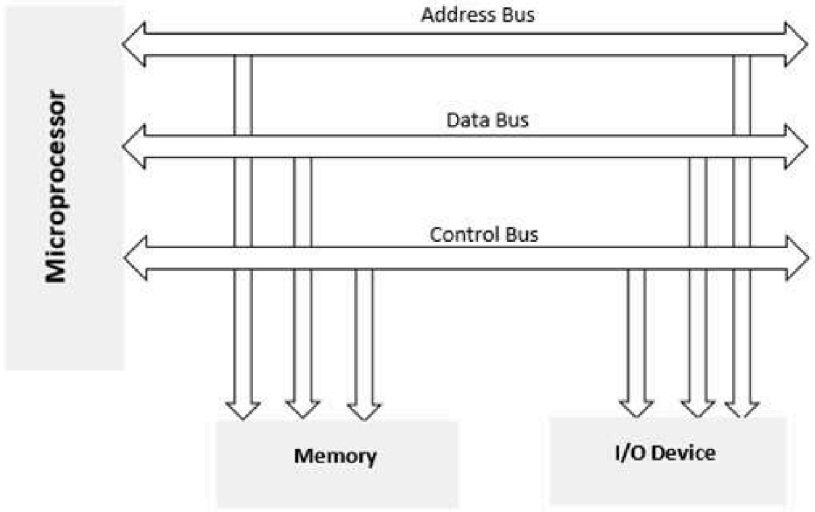
- The **IN** instruction transfers data from an I/O port (external device) to the accumulator (register A) of the microprocessor.

- Syntax: `IN A, port number`
- The **port number** specifies the I/O port from which data is read.
- Example: If we want to read data from a keyboard, we use the **IN** instruction to read the key code from the keyboard controller.

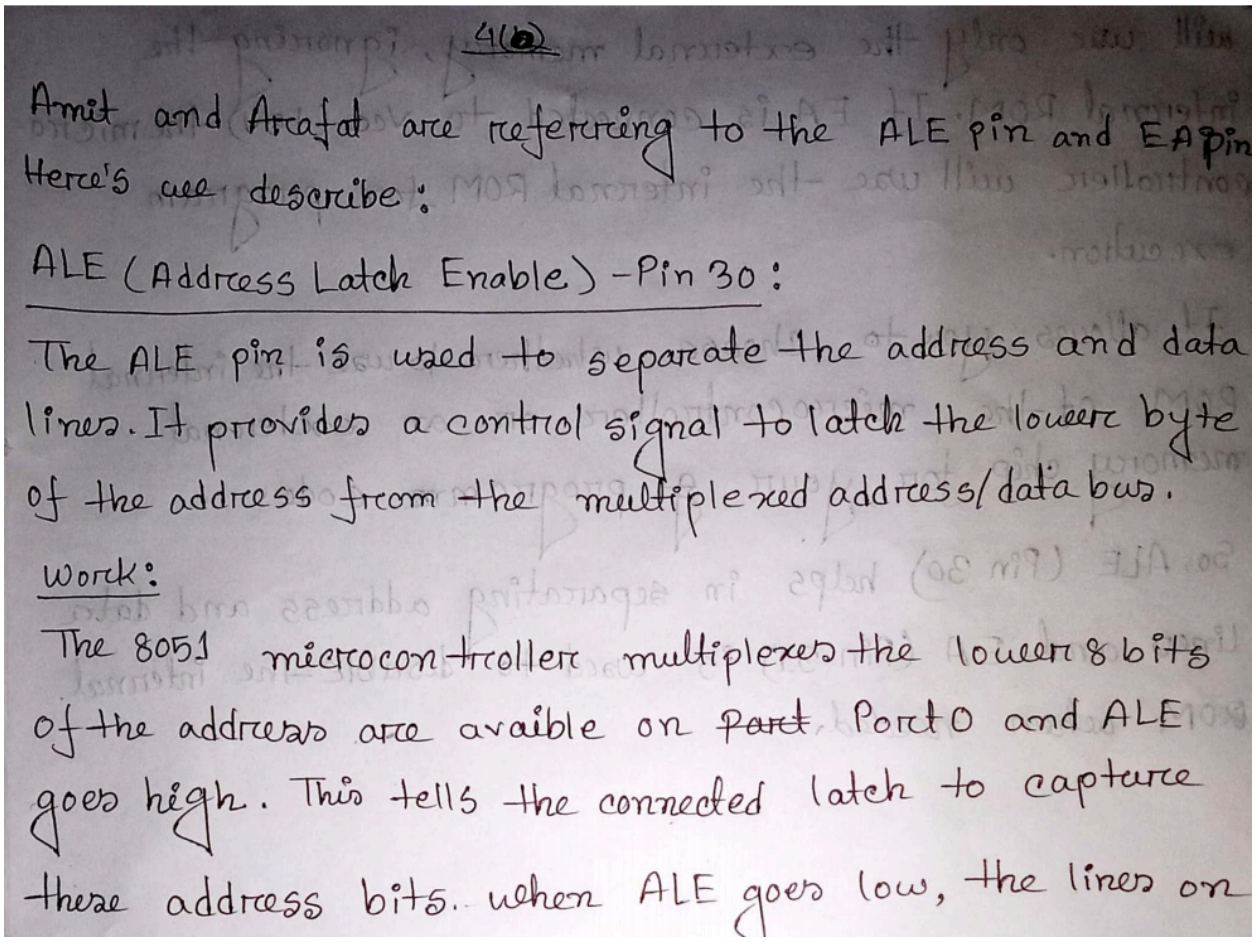
OUT Instruction:

- The **OUT** instruction transfers data from the accumulator (register A) to an I/O port (external device).
- Syntax: `OUT port number, A`
- The **port number** specifies the I/O port to which data is written.
- Example: To display a character on a monitor, we use the **OUT** instruction to send the character code to the display controller.

Diagram of I/O Operations: (not sure)



4(a) Recently Amit and Arafat learned about 8051 microcontroller. However they are aware that there is a pin that separates address and data lines and another pin that disables internal ROM, but they are having trouble recognizing them. Help them by describing these pins.



4(a)

Amit and Arafat are referring to the ALE pin and EA pin. Here's we describe:

ALE (Address Latch Enable) - Pin 30:

The ALE pin is used to separate the address and data lines. It provides a control signal to latch the lower byte of the address from the multiplexed address/data bus.

Work:

The 8051 microcontroller multiplexes the lower 8 bits of the address are avaible on ~~Port~~ Port 0 and ALE goes high. This tells the connected latch to capture these address bits. when ALE goes low, the lines on

Port 0 switch to carrying data instead of the address

It helps to demultiplexing the address data bus making it possible to use the same lines for both address and data but at different times.

EA (External Access) - Pin 31 :

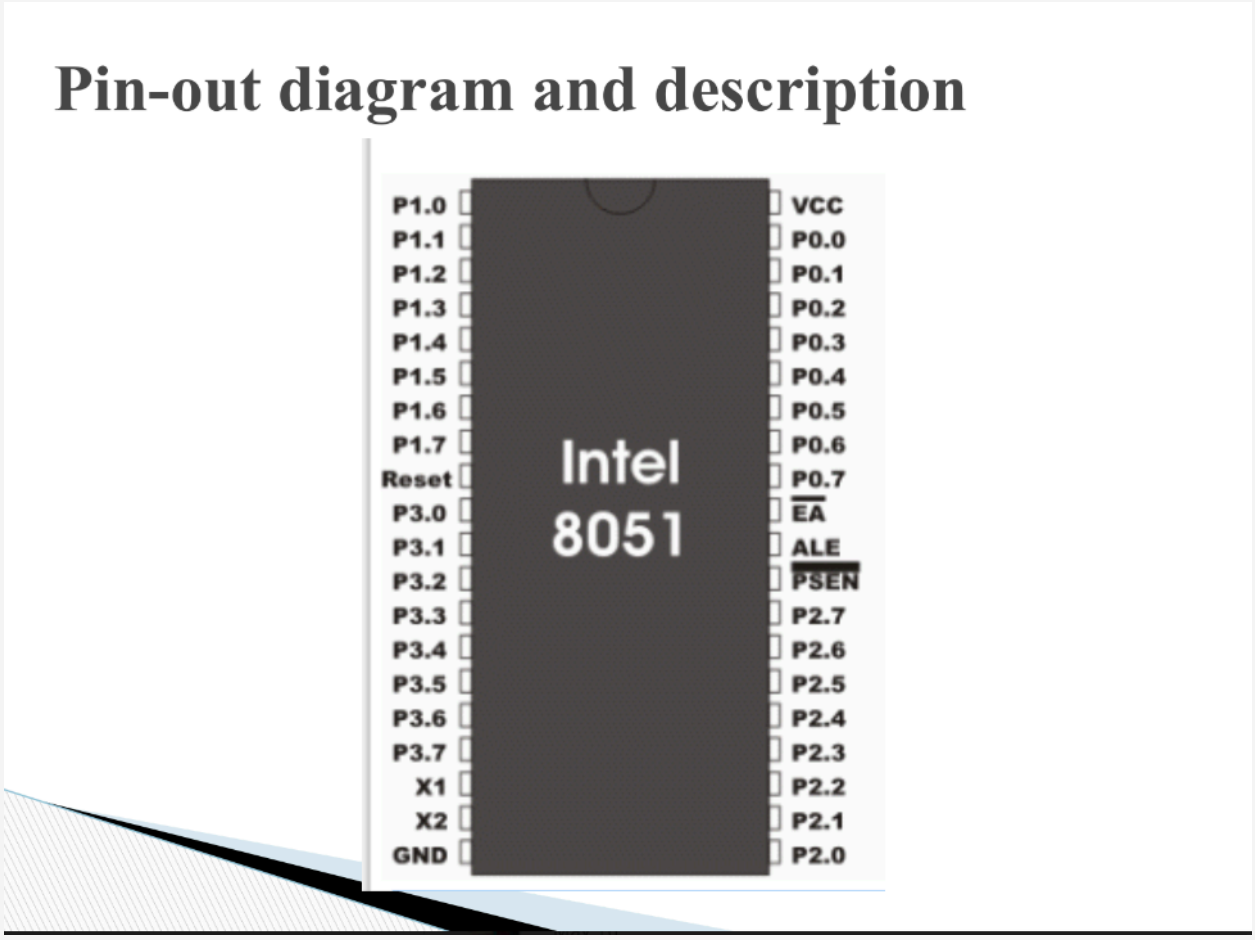
This pin is used to enable or disable the internal ROM.

If EA is connected to ground (0V) the 8051 microcontroller will use only the external memory, ignoring the internal ROM. If EA is connected to Vcc (+5V) the mic controller will use the internal ROM for program execution.

It allows you to choose whether to use the internal ROM of the microcontroller or an external memory chip for your ~~if~~ programme code.

So, ALE (Pin 30) helps in separating address and data lines and EA (Pin 31) is used to disable the internal ROM when needed.

4(b) What are the key functions of the main pins in the INTEL 8051 microcontroller , as depicted is the pinout diagram and how do these pins contribute to the microcontroller's operation and its ability to interface with external devices.



The ~~8501~~ 8051 microcontroller is a widely used microcontroller with a variety of applications in embedded system. Understanding the pinout diagram is essential for effectively using the 8051 in interfacing with external devices.

Overview of the main pins and their contribution:

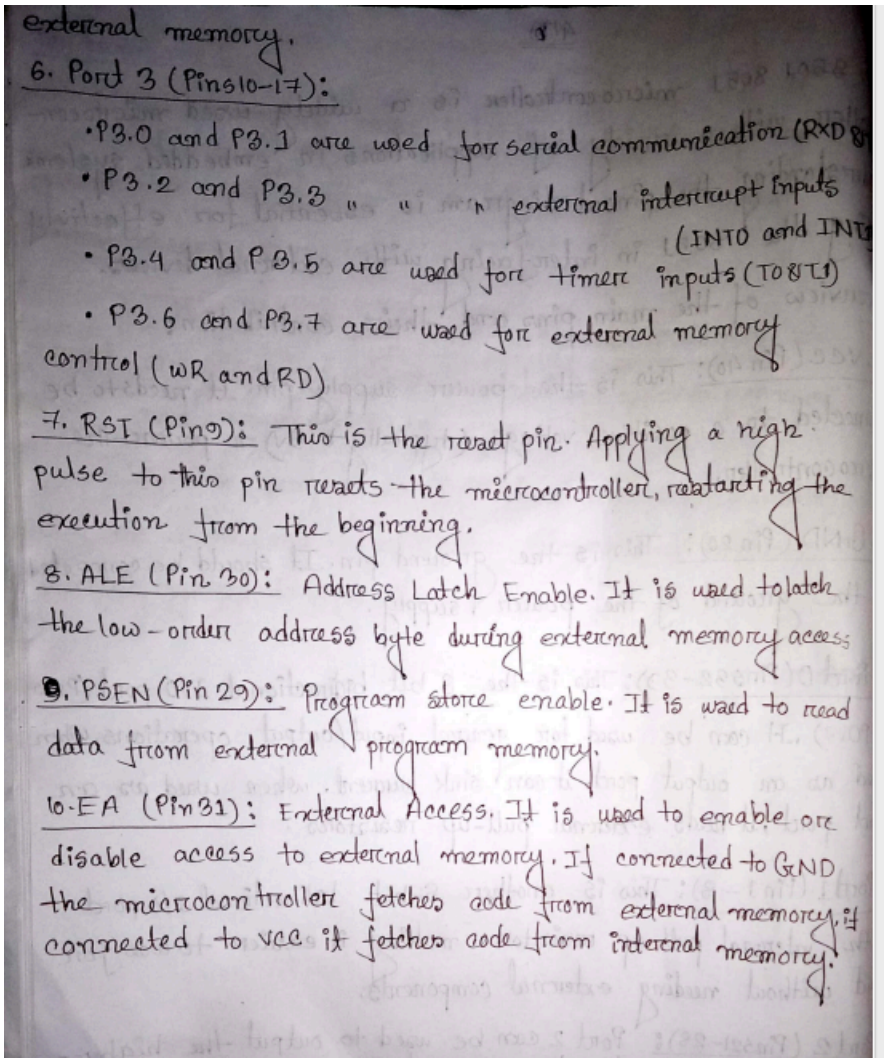
1. VCC (Pin 40): This is the power supply pin. It needs to be connected to a positive voltage (typically +5V) to power the microcontrollen.

2. GND. (Pin 20): This is the ground pin. It should be connect to the ground of the power supply.

3. Port 0 (Pins 32-39): This is the 8 bit bidirectional I/O port (P0 to P0.7). It can be used for general input/output operations. Wh used as an output port, it can sink current. when used as an input port, it needs external pull-up resistors.

4. Port 1 (Pin 1 - 8): This is another 8-bit bidirectional I/O port with internal pull-up resistor. making it easier to use for input without needing external components.

5. Port 2 (Pins 21-28): Port 2 can be used to output the higher-order address byte when the microcontroller is accessing



external memory.

6. Port 3 (Pins 10–17):

- P3.0 and P3.1 are used for serial communication (RXD &
- P3.2 and P3.3 " " " external interrupt inputs (INT0 and INT
- P3.4 and P3.5 are used for timer inputs (T0 & T1)
- P3.6 and P3.7 are used for external memory control (WR and RD)

7. RST (Pin 9): This is the reset pin. Applying a high pulse to this pin resets the microcontroller, restarting the execution from the beginning.

8. ALE (Pin 30): Address Latch Enable. It is used to latch the low-order address byte during external memory access

9. PSEN (Pin 29): Program store enable. It is used to read data from external program memory.

10. EA (Pin 31): External Access. It is used to enable or disable access to external memory. If connected to GND the microcontroller fetches code from external memory, if connected to VCC it fetches code from internal memory.

5(a) What is System-on-a-chip (SoC)? Describe each of its components with a diagram.

SoC

A system on chip (SoC) is an IC that integrates all components of a computer or other electronic system into a single chip.

SoC components

1. A MC or MP
2. Memory blocks including ROM, RAM, Flash
3. Timing sources including oscillators
4. Peripherals including counter-timers
5. External interfaces including industry standards such as USB, Ethernet etc.
6. Voltage regulators.

5(b) What is the significance of the PLD (Programmable Logic Device) in the context of decoding I/O addresses in embedded system? How does the addition of eight address lines (A15-A8) affect the decoding process, and how does the PLD generate address strobes for I/O ports?

Answer:

In embedded systems, a Programmable Logic Device (PLD) is important for decoding I/O addresses because it can be programmed to recognize specific address patterns. This helps in determining which devices should respond when the CPU access a particular address. Here's a simple explanation of its significance and how it works:

Significance of PLD:

1. PLDs can be programmed to implement custom logic functions. This flexibility allows designers to tailor the I/O address decoding to their specific system requirements.

2. By using a PLD, multiple decoding functions can be integrated into a single device, reducing the need for multiple discrete logic chips. This can save space and reduce complexity.

3. PLDs can perform address decoding very quickly, which is essential for maintaining efficient communication within an embedded system.

Adding Address Lines (A15-A8):

When decoding I/O addresses, the address lines (A15-A8) represent the higher-order bits of the address. Here's what happens when these lines are used:

1. Adding address lines increases the number of unique addresses that can be decoded. For example, using lines A15-A8 means there are 256 possible combinations (since 2^8 = 256).

2. With more address lines, the PLD can more specifically identify which I/O port or peripheral should be accessed, reducing the chance of address conflicts.

PLD Generating Address Strobes:

Here's how a PLD can generate these strobes:

1. The PLD is programmed with logic equations that define when specific address ranges (combinations of A15-A8) should activate a strobe signal.

2. When the microcontroller or processor outputs an address, the PLD evaluates the address lines (A15-A8) against its programmed logic.

3. If the address lines match a predefined pattern, the PLD generates the corresponding strobe signal, indicating that the associated I/O port should be accessed.

Example:

Suppose you have an I/O port that should respond to addresses where A15-A8 are `01010101` (85 in hexadecimal). The PLD is programmed to recognize this pattern. When the address bus holds this value, the PLD activates the strobe for this I/O port, allowing data to be read from or written to the port.

5(b) or, Providing a brief explanation of modeling embedded system behavior, illustrated with an example of a finite state machine model.

Answer:

Modeling the behavior of an embedded system can be effectively done using a finite state machine (FSM). An FSM is a mathematical model used to design both computer programs and sequential logic circuits. It consists of a finite number of states, transitions between these states, and actions.

Components of an FSM

1. States: Distinct modes in which the system can exist.

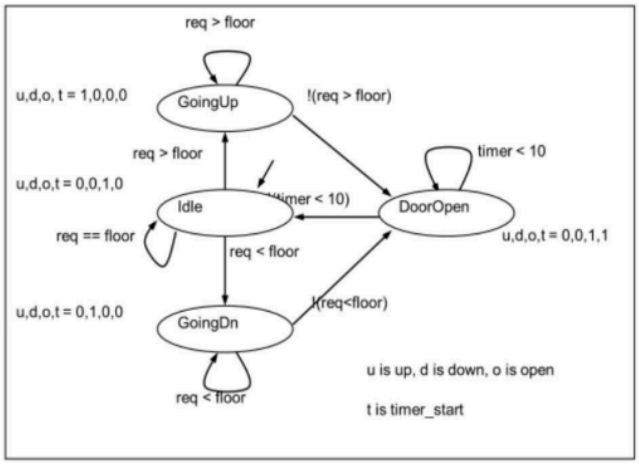
2. Transitions: Rules that determine how the system moves from one state to another.

3. Events/Inputs: Triggers that cause state transitions.

4. Actions: Operations performed during transitions or while in a state.

Modeling Embedded system behavior with Finite State machine models

1. List all possible states
2. Declare all variables
3. For each state, list possible transitions, with conditions, to other states
4. For each state and/or transition, list associated actions
5. For each state, ensure exclusive and complete exiting transition conditions
   - No two exiting conditions can be true at same time
     - Otherwise nondeterministic state machine
   - One condition must be true at any given time



Example: FSM for a Simple Traffic Light System

Imagine a traffic light system with three states:

1. **Green Light**
2. **Yellow Light**
3. **Red Light**

Here's how the FSM works for this traffic light:

1. **States**:

- **Green**: Traffic can move.
- **Yellow**: Traffic should slow down and prepare to stop.
- **Red**: Traffic must stop.

2. **Transitions**:

- **From Green to Yellow**: This transition occurs after a set time (e.g., 60 seconds).
- **From Yellow to Red**: This transition occurs after a shorter time (e.g., 5 seconds).
- **From Red to Green**: This transition occurs after another set time (e.g., 60 seconds).

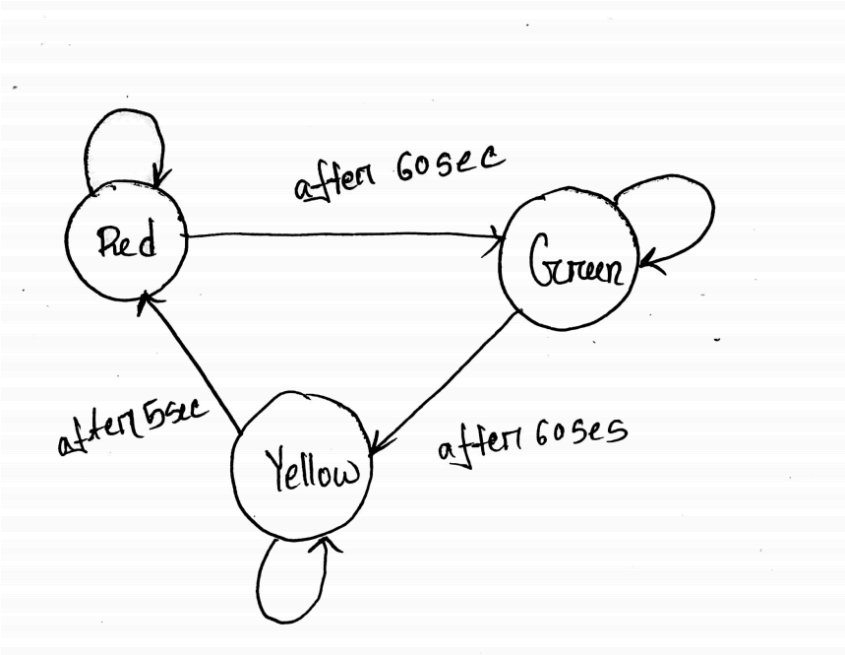
3. **Events**:

- **Timer Expiration**: Each state change is triggered by the expiration of a timer set for each state duration.

4. **Actions**:

- **In Green State**: Turn on green light.
- **In Yellow State**: Turn on yellow light.
- **In Red State**: Turn on red light.

Finite state Machine Model (not Sure):



How It Works

1. **Start at Green**: The traffic light starts in the Green state.
2. **Transition to Yellow**: After 60 seconds, the system transitions to the Yellow state.
3. **Transition to Red**: After 5 seconds in the Yellow state, the system transitions to the Red state.

4. **Transition back to Green**: After 60 seconds in the Red state, the system transitions back to the Green state, and the cycle repeats.