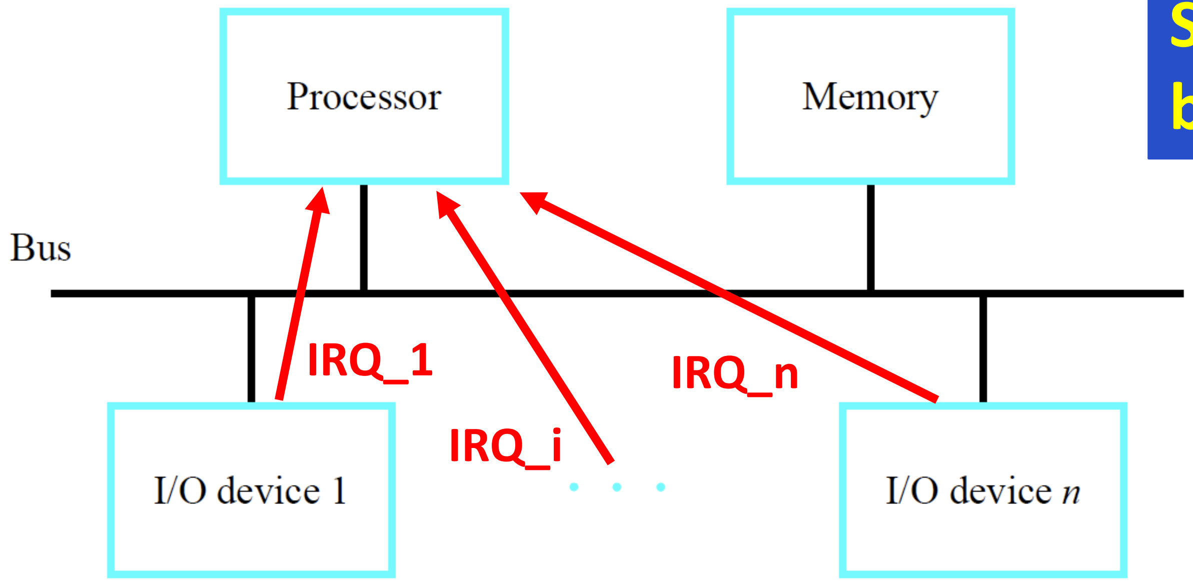
# Multiple I/O Devices



**Some questions to answer in both cases**

1. Which device raises an interrupt?
2. Which ISR to use?
3. Which IRQ-device to serve next if there are multiple requests?

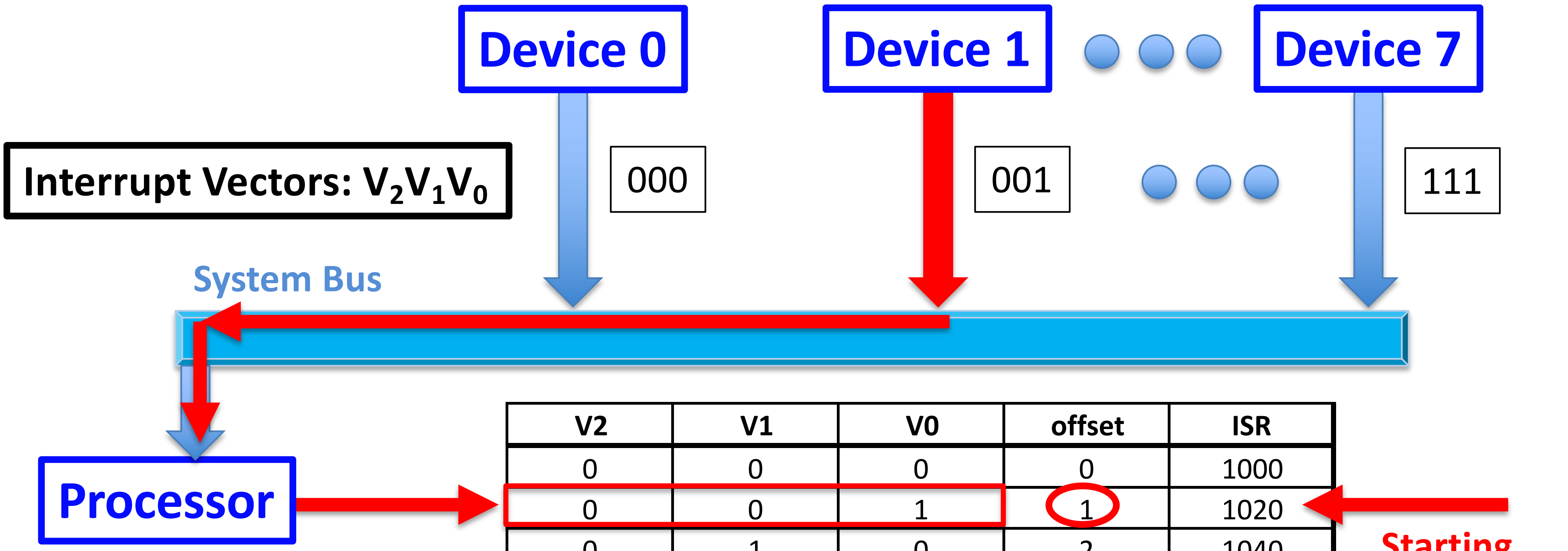
- Two cases:

a) One device raises an interrupt (IRQ)
b) Multiple devices raise IRQ

## 1. Which Device? – 2. Which ISR?

- **a) Single and b) multiple IRQ cases:**
- The interrupting device:
  - ➢ Sends an identification code or ***Interrupt Vector***
  - ➢ Example: 000, 001, 011, etc. to the processor
- The processor:
  - Uses the vector to index into an ***interrupt vector table***
  - Retrieves the starting address of the corresponding ISR

# I/O Device Self Identification



| V2 | V1 | V0 | offset | ISR |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1000 |
| 0 | 0 | 1 | 1 | 1020 |
| 0 | 1 | 0 | 2 | 1040 |
| 0 | 1 | 1 | 3 | 1060 |
| 1 | 0 | 0 | 4 | 1080 |
| 1 | 0 | 1 | 5 | 1100 |
| 1 | 1 | 0 | 6 | 1120 |
| 1 | 1 | 1 | 7 | 1140 |

Starting Address of ISR #1

Vector Interrupt Table

# Interrupt Vector

Vector=010

Vector from Interrupting Device

Vector=110

| V2 | V1 | V0 | offset | ISR |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1000 |
| 0 | 0 | 1 | 1 | 1020 |
| 0 | 1 | 0 | 2 | 1040 |
| 0 | 1 | 1 | 3 | 1060 |
| 1 | 0 | 0 | 4 | 1080 |
| 1 | 0 | 1 | 5 | 1100 |
| 1 | 1 | 0 | 6 | 1120 |
| 1 | 1 | 1 | 7 | 1140 |

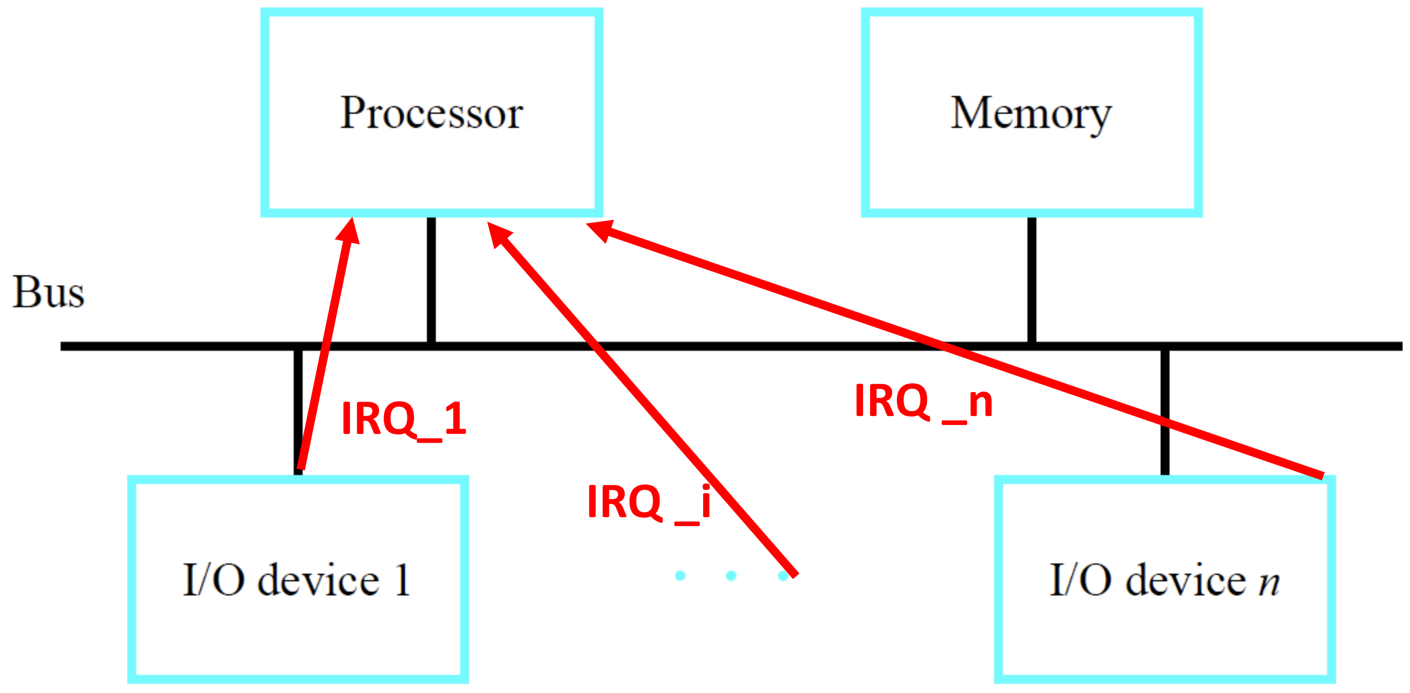
Offset 2 = 40

Offset 6 = 120

**Starting Address**
**= 1000 + Vector# X 20**

# Interrupts from Multiple Devices

**3. Which IRQ-device to serve next if there are simultaneous requests?**



**Options:**
- FIFO?
- LIFO?
- Priority?

**Priority:**
1. Polling Device Order
2. Pre-assigned in HW

## Which Device to Serve Next: Priority and Interrupt Nesting

Multiple simultaneous interrupt requests scenarios:

1. No interrupt allowed (DI) while executing an ISR
   - Simple but not efficient; may miss a more important task
2. 'User' prioritizes I/O devices
   - CPU accepts higher priority interrupts while it is
     a. Executing a main program (that has a lower priority) or
     b. Serving a lower priority device (in ISR now, nested interrupts)
   - Enables (EI) <u>higher</u> priority interrupts

# Disable and Enable Interrupts

- Disable further interrupts <u>temporarily</u>
- Enable interrupt again at the right moment

1. Normal case:
    - **DI**: disable all interrupts
    - **$EI_a$**: enable **<u>a</u>**ll interrupts
2. Priority case:
    - **DI**: disable all interrupts
    - **$EI_h$**: enable all interrupts with **<u>h</u>**<u>igher priority</u>

# Handling Multiple IRQ Scenario 1

**1. Normal Scenario: No Priority**

While processor is serving ISR ***A***, ***Device B raises interrupt request***:

- Processor completes ISR ***A***, then handles ***B***
  - Use of Disable and Enable Interrupt (DI & EI)
  - In ISR ***A***:

a. DI; disable all interrupts

b. **Initialization; DO NOT DISTURB**

**c.** **Critical Region/Section ; DO NOT DISTURB**

d. $EI_a$; enable all interrupts

e. Return

# Handling Multiple IRQ Scenario 2

**2. Scenario: With Prioritized Interrupts**

While processor is serving ISR ***A***, ***Device B raises interrupt request***:

- Handles ***B*** if ***B*** has higher priority
  - Use of Disable and Enable Interrupt
  - In ISR ***A***:
    - a. DI; disable all interrupts
    - **b. Initialization; DO NOT DISTURB**
    - c. $EI_h$; enable all **h**igher priority interrupts
    - d. Execute instructions (could be interrupted by higher priority)
    - e. Return