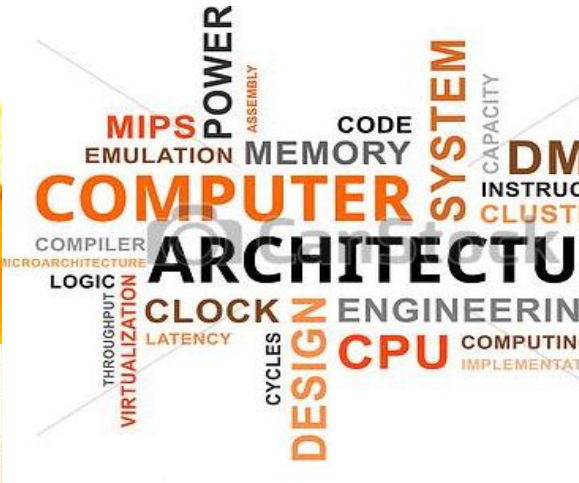
UNIT V
I/O ORGANIZATION AND PARALLELISM

Accessing I/O devices – **Interrupts** – Direct Memory Access – Buses–Interface circuits – Standard I/O Interfaces (PCI, SCSI, USB) –Instruction Level Parallelism : Concepts and Challenges – Introduction to multicore processor – Graphics Processing Unit



Recap the previous Class

Interrupt

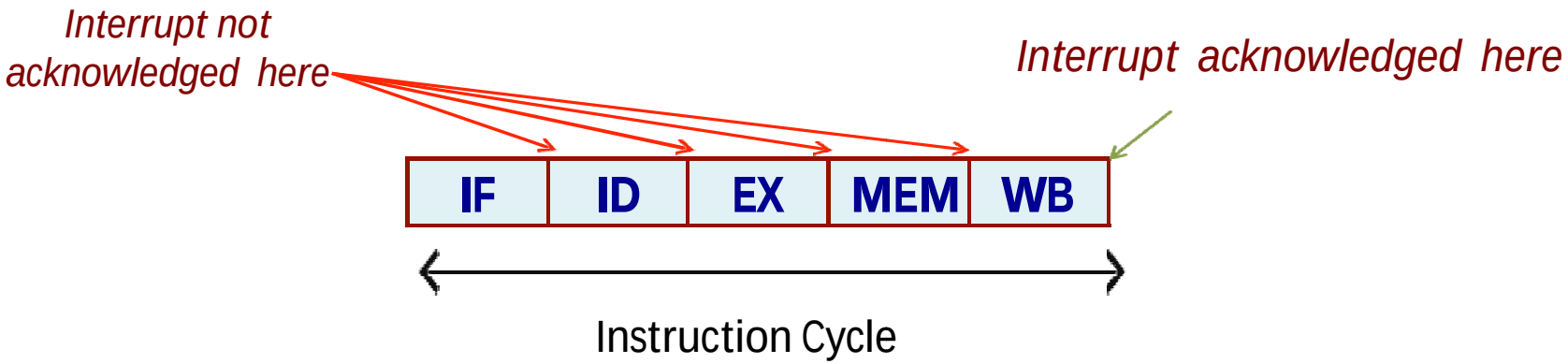
- An **interrupt** is a signal that requests **the processor to suspend its current execution** and service the occurred interrupt.
- To service the interrupt the processor executes the **corresponding interrupt service routine (ISR).**
- After the execution of the interrupt service routine, the **processor resumes** the execution of the suspended program.
- Interrupts can be of two types of **hardware interrupts and software interrupts**.

What happens when an interrupt request arrives?

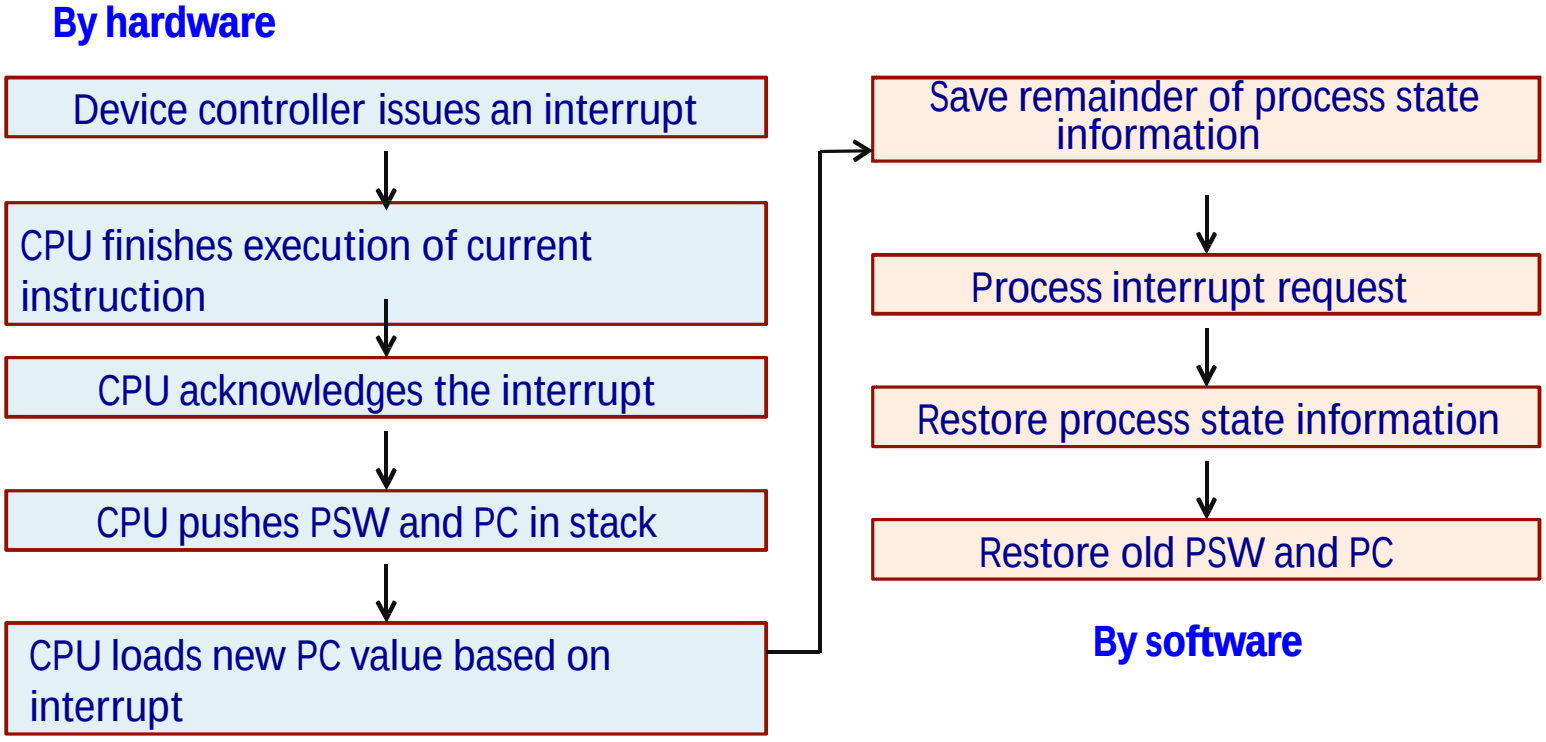
- At the end of the current instruction execution, the **PC and program status word (PSW)** are saved in stack automatically.
- The interrupt is **acknowledged**, the **interrupt vector** obtained, based on which control transfers to the appropriate ISR.
- After handling the interrupt, the ISR executes a special ***Return From Interrupt (RTI)*** instruction.

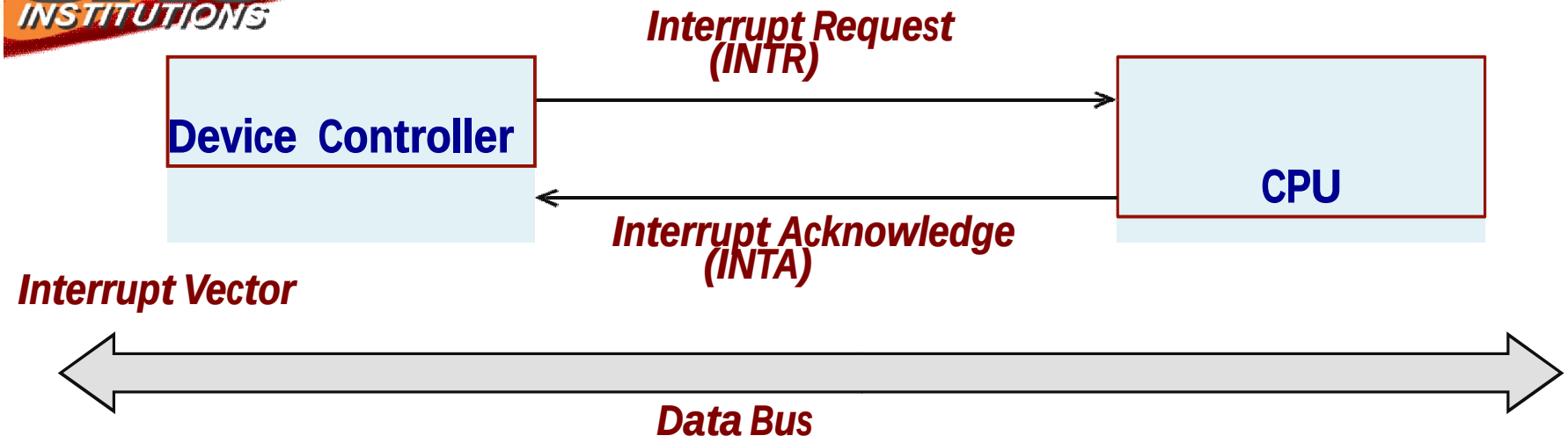
–Restores the PSW and returns control to the saved PC address.

- An instruction cycle typically consists of several machine cycles.
  - For MIPS32, there are 5 machine cycles.



General Interrupt Processing





- The steps:

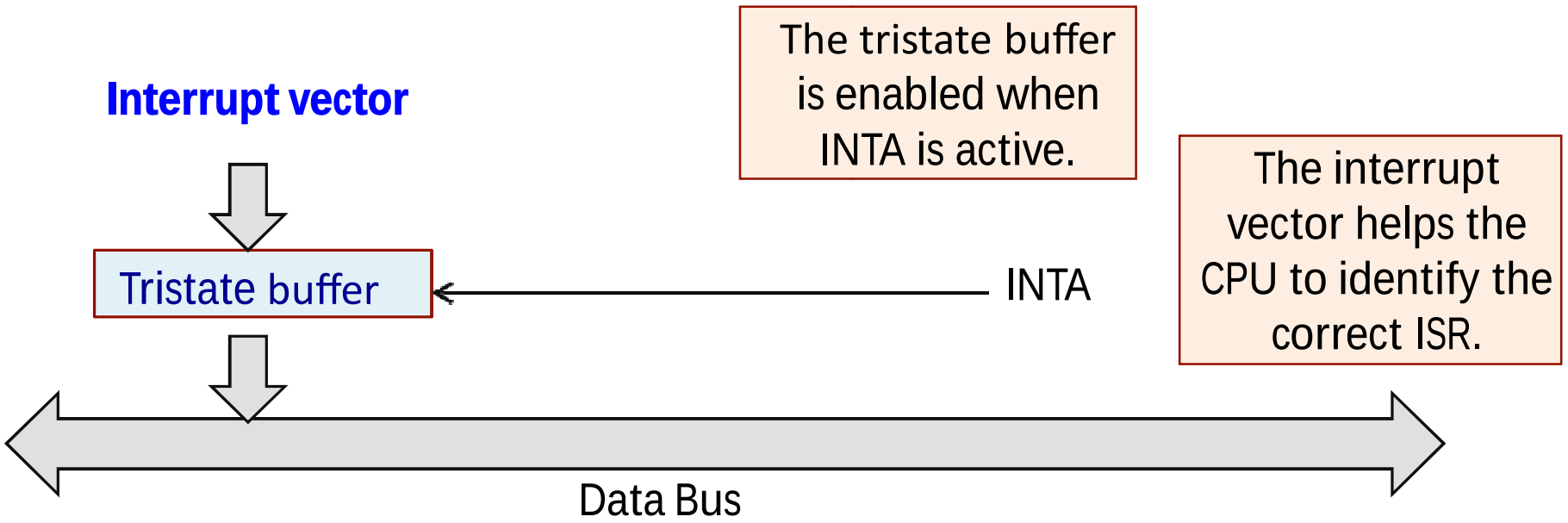
  a) Device controller sends INTR to the CPU.

  b) CPU finishes the current instruction and sends back INTA.

  c) Device controller sends interrupt vector (or number) over data bus.

  d) CPU reads the interrupt vector, and identifies the device.

How is the interrupt vector sent on the data bus in response to INTA?



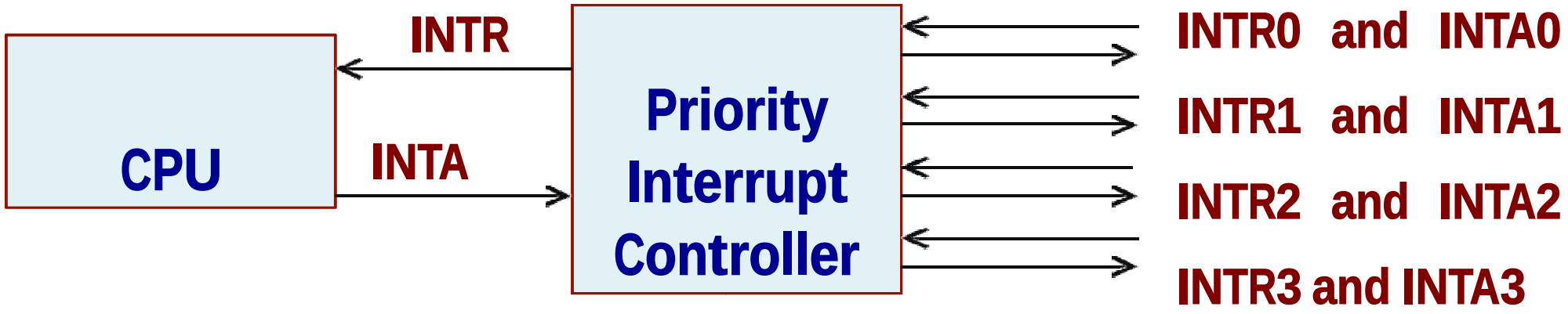
Multiple Devices Interrupting the CPU

- A common solution is to use a priority interrupt controller.

– The interrupt controller interacts with **CPU on one side** and **multiple devices on the other side.**

– For simultaneous interrupt requests, interrupt priority is defined.

– The interrupt controller is responsible for sending the interrupt vector to CPU.



How it works?

- The **INTR line is made active** when some of the device(s) **activate their interrupt request line.**

$$INTR = INTR0 + INTR1 + INTR2 + INTR3$$

- When the **CPU sends back INTR**, the **interrupt controller sends** back the corresponding **acknowledge** to the interrupting device, and puts the interrupt vector on the data bus.
- The interrupt controller is programmable, where the interrupt vectors for the various interrupts can be programmed (specified).
- For **more than one interrupt request** simultaneously active, **a priority mechanism** is used (e.g. INTR0 is highest priority, followed by INTR1, etc.).

How is interrupt nesting handled?

Consider the scenario:

a) A device D0 has interrupted and the CPU is executing the ISR for D0.

b) In the mean time, another device D1 has interrupted.

Two possible scenarios here:

–D1 will interrupt the ISR for D0, get processed first, and then the ISR for D0 will be resumed. *CREATES PROBLEM FOR MULTI NESTING*

–Disable the interrupt system automatically whenever an interrupt is acknowledged so that handling of nested interrupts is not required.

- Typical instruction set architectures have the following instructions:

  EI : Enable interrupt

  DI : Disable interrupt

- For the second scenario as discussed, the ISR will give an EI instruction just before RTI.
- The DI instruction is sometimes used by the operating system to execute atomic code (e.g. semaphore wait and signal operations).

Cases that make interrupt handling difficult

•For some interrupts, it is not possible to finish the execution of the current instruction.

– A special **RETURN instruction** is required that would return and ***restart*** the interrupted instructions.

Some examples:

a) Page fault interrupt: A memory location is being accessed that is not presently available in main memory.

b) Arithmetic exception: Some error has occurred during some arithmetic operation (e.g. division by zero).

INTERRUPT HANDLING

Handling Multiple Devices

Suppose that a number of devices capable of generating interrupts are connected to the CPU.

The following questions need to be answered.

a)How can the CPU identify the interrupting device?

b)How can the CPU obtain the starting address of the appropriate ISR?

c)Should interrupt nesting be allowed?

d)How should two or more simultaneous interrupt requests be handled?

(a) Device Identification

Suppose that an external device requests an interrupt by activating an INTR line that is common to all the devices. That is,

$INTR = INTR_1 + INTR_2 + ... + INTR_n$

- Each device can have a **status bit** indicating whether it has interrupted.
  - CPU can ***poll*** the status bits to find out who has interrupted.

(b) Find Starting Address of ISR

For a processor with multiple interrupt request inputs, the **address of the ISR can be fixed for each individual input**.

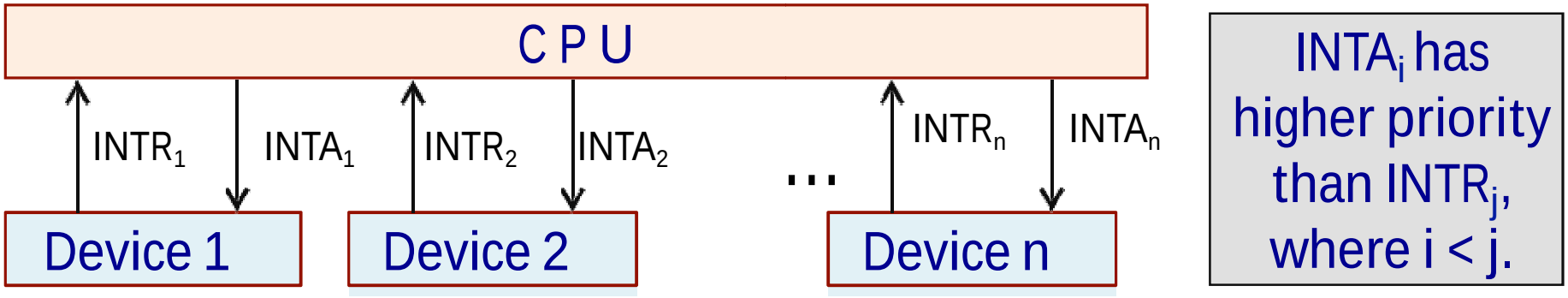
- Lacks flexibility.

(c) Interrupt Nesting

- **A simple approach:**
  - **Disable all interrupts** during the execution of an ISR.
  - This **ensures that the interrupt request from one device** will not cause more that one interruptions.
- **Interrupt priority:**
  - Some interrupting devices may be **assigned higher priorities** than others.
  - *Example*: timer interrupt to maintain a real-time clock.
  - **Higher priority interrupt may interrupt** the ISR of lower priority ones.

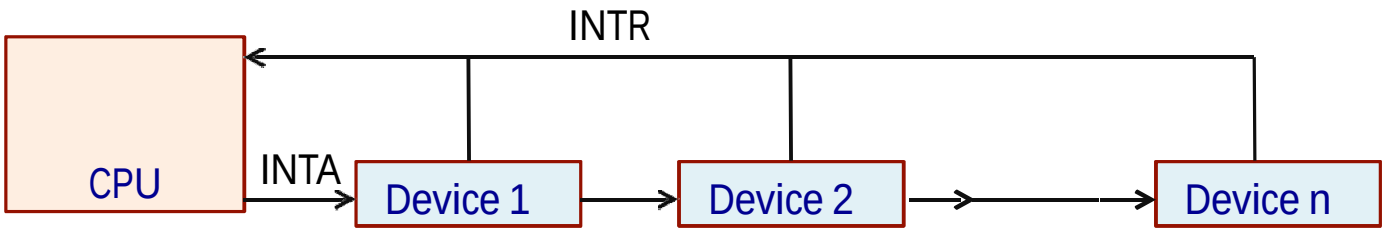
(d) Simultaneous Requests

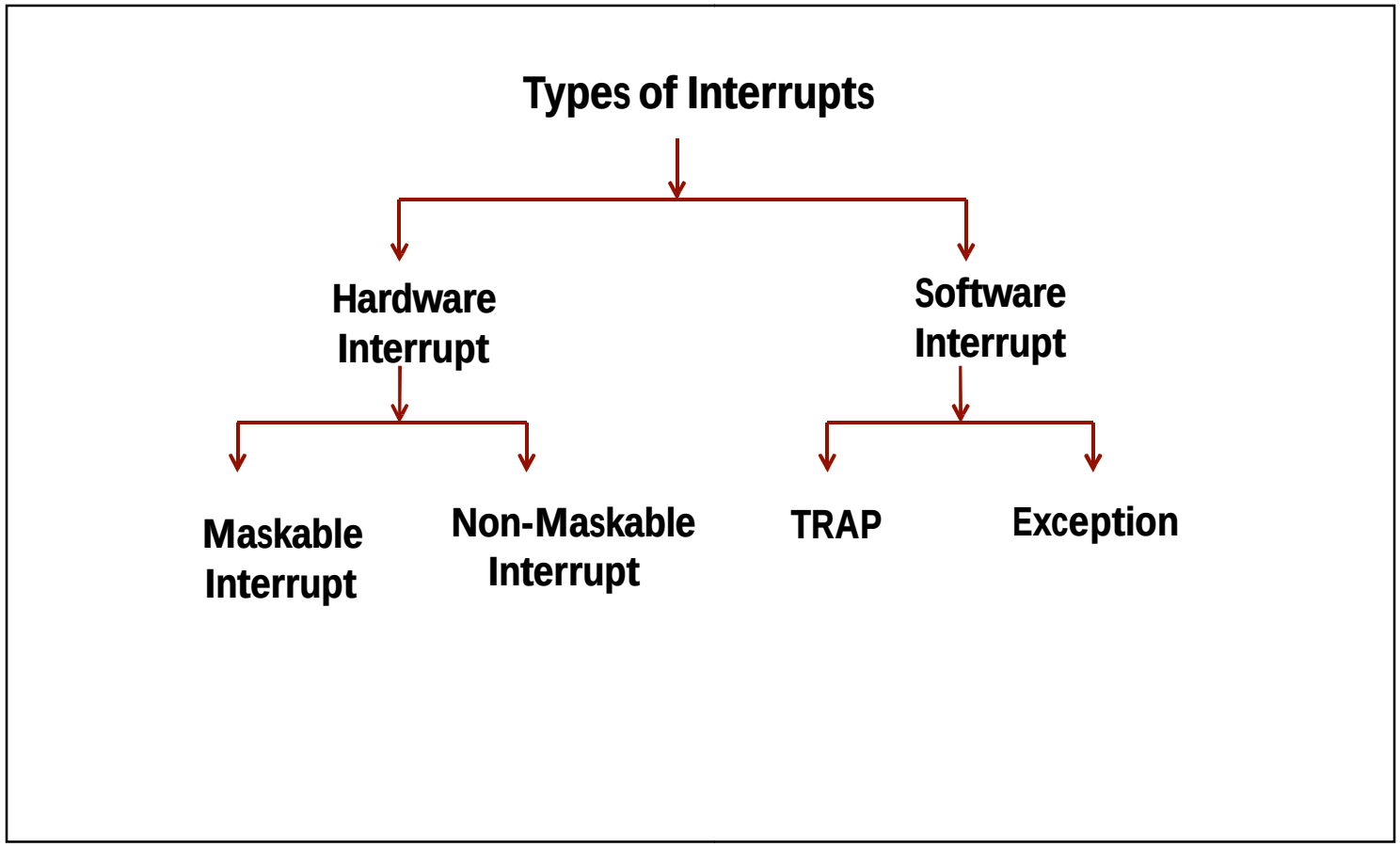
- Here we consider the problem of simultaneous arrivals of interrupt requests from two or more devices.
  - CPU should have some mechanism by which only one request is serviced while the others are **delayed or ignored**.
  - If the CPU has multiple interrupt request lines, it can have a ***priority scheme*** where it accepts the request with the highest priority.



- Another way to assign priority is to use polling using ***daisy chaining*.**
  - In daisy chain connection, **the INTR line is common to all the devices**, but the **INTA line is connected in a daisy chain fashion** allowing it to propagate serially through the devices.
  - A device when it receives INTA, passes the signal to the next device only if it had not interrupted. Else, it stops the propagation of INTA, and puts the identifying code on the data bus.
  - Thus, the device that is electrically closest to the CPU will have the highest priority.

Daisy Chain Arrangement




Types of Interrupts
Hardware Interrupt
Software Interrupt
Maskable Interrupt
Non-Maskable Interrupt
TRAP
Exception

- **Hardware Interrupt:**
  - The interrupt signal is coming from a device external to the CPU.
  - Example: keyboard interrupt, timer interrupt, etc.
- **Maskable Interrupt:**
  - Hardware interrupts that can be **masked or delayed** when a higher priority interrupt request arrives.
  - There are processor instructions that can selectively mask and unmask the interrupt request lines of the CPU.
- **Non-Maskable Interrupt:**
  - Interrupts that **cannot be delayed** and should be handled by the CPU immediately.
  - Examples: power fail interrupts, real-time system interrupts, etc.

- **Software Interrupt:**
  - They are caused due to execution of some instructions.
  - Not caused due to external inputs.
- **TRAP:**
  - They are special instructions used to request services from the operating system.
  - Also called ***system calls*****.**
- **Exception:**
  - These are **unplanned interrupts** generated while executing a program.
  - They are generated from within the system.
  - Examples: invalid opcode, divide by zero, page fault, invalid memory access, etc.

TEXT BOOK

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, McGraw-Hill, 6th Edition 2012.

REFERENCES

1. David A. Patterson and John L. Hennessey, “Computer organization and design”, MorganKauffman ,Elsevier, 5th edition, 2014.
2. William Stallings, “Computer Organization and Architecture designing for Performance”, Pearson Education 8th Edition, 2010
3. John P.Hayes, “Computer Architecture and Organization”, McGraw Hill, 3rd Edition, 2002
4. M. Morris R. Mano “Computer System Architecture” 3rd Edition 2007
5. David A. Patterson “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann; 5th edition 2011

THANK YOU