

# Module 6-

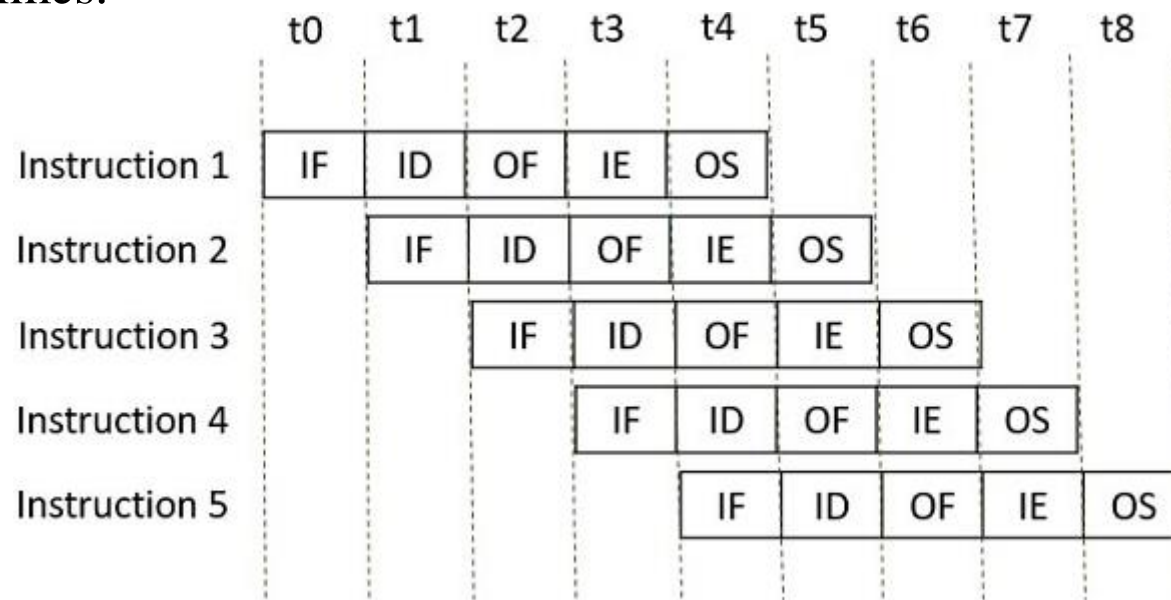
# Principles of Advanced Processor and Buses

-Neha Surti

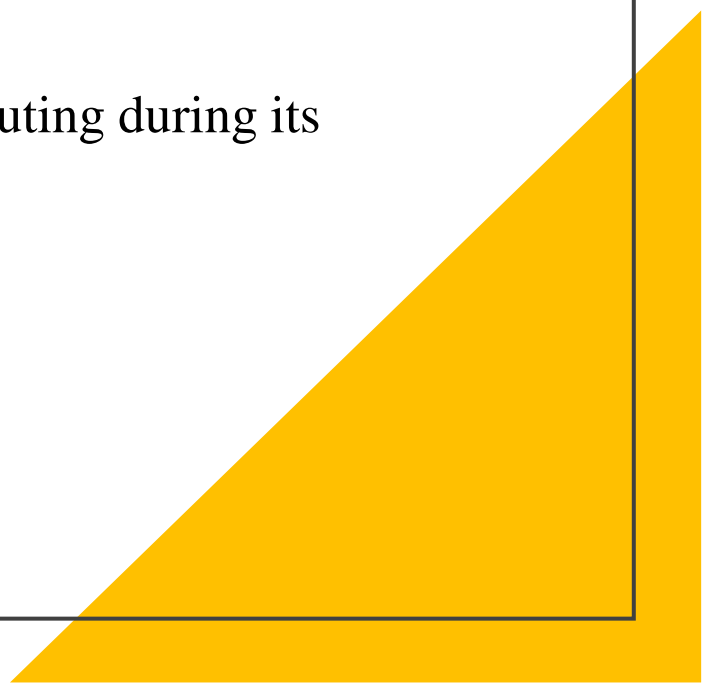
# Basic Pipelined Datapath and control

## Pipelining concepts

- A pipelined processor allows multiple instructions to execute at once, and each instruction uses a different functional unit in the datapath.
- This increases throughput, so programs can run faster.
  - One instruction can finish executing on every clock cycle, & simpler stages also lead to shorter cycle times.

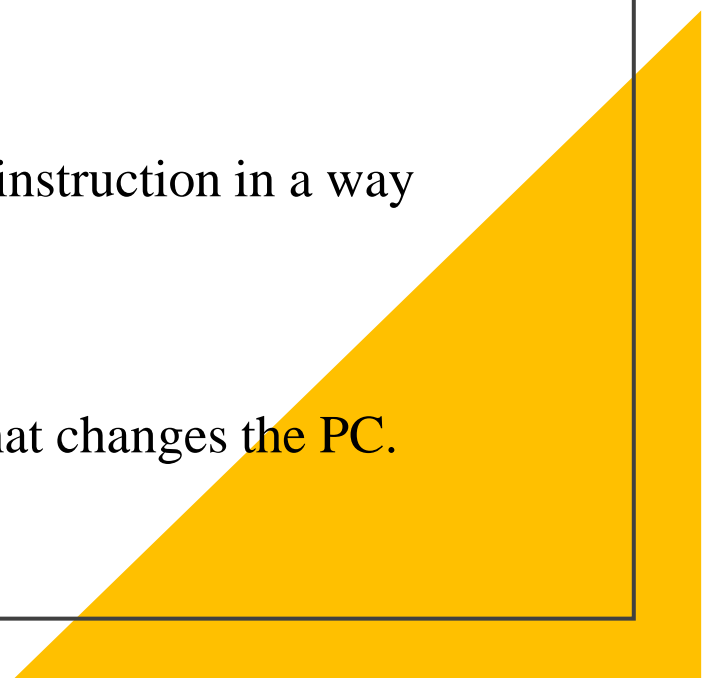


# Data Dependency

- There are three types of dependencies possible in a pipelined processor:
    - 1) Structural Dependency
    - 2) Control Dependency
    - 3) Data Dependency
  - These dependencies may introduce stalls (a cycle in the pipeline without new input) in the pipeline.
  - Any condition that causes the pipeline to stall is called a **hazard**.
  - Hazards that arise in the pipeline prevent the next instruction from executing during its designated clock cycle.
- 
- A large yellow right-angled triangle is positioned in the bottom right corner of the slide, pointing towards the top right.

# Pipeline hazards

There are three types of hazards in a pipeline, they are as follows:

- Structural Hazards:
    - They arise from resource conflicts when the hardware in the pipeline cannot support all the overlapped instructions in the pipeline. (two instructions in the pipeline require the same resource).
  - Data Hazards:
    - They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.
  - Control Hazards:
    - They arise from the pipelining of branches and other instructions that changes the PC.
- 
- A large yellow right-angled triangle is positioned in the bottom right corner of the slide, pointing towards the top right.

# Structural Hazards

- Structural dependency arises due to the resource conflict in the pipeline.
- A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle.
- A resource can be a register, memory, or ALU.
- E.g:

Instruction / Cycle	1	2	3	4	5
I <sub>1</sub>	IF(Mem)	ID	EX	Mem	
I <sub>2</sub>		IF(Mem)	ID	EX	
I <sub>3</sub>			IF(Mem)	ID	EX
I <sub>4</sub>				IF(Mem)	ID

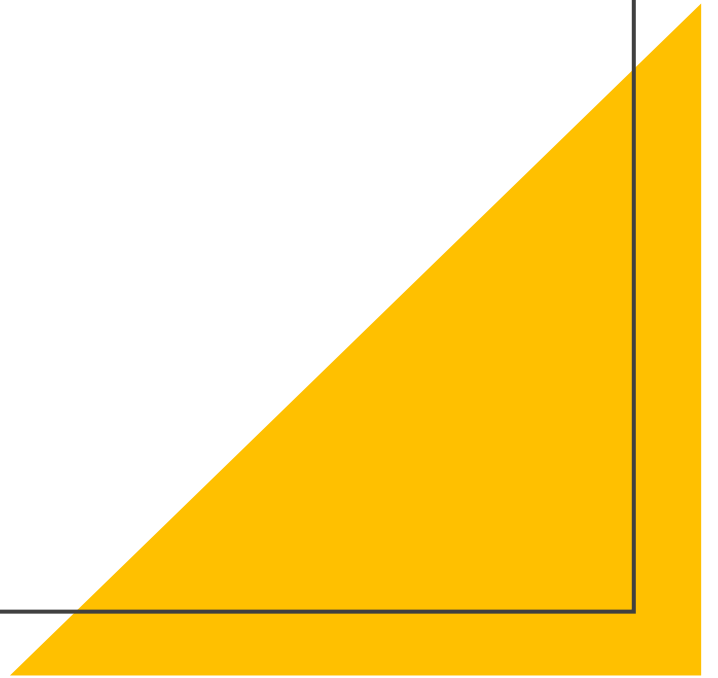
# Structural Hazards

- In the above scenario, in cycle 4, instructions I1 and I4 are trying to access same resource (Memory) which introduces a resource conflict.
- To avoid this problem, we have to keep the instruction on wait until the required resource (memory in our case) becomes available.
- This wait will introduce stalls in the pipeline as shown below:

Cycle	1	2	3	4	5	6	7	8
I <sub>1</sub>	IF(Mem)	ID	EX	Mem				
I <sub>2</sub>		IF(Mem)	ID	EX	Mem			
I <sub>3</sub>			IF(Mem)	ID	EX	Mem		
I <sub>4</sub>				-	-	-	IF(Mem)	

# Data Hazards

- Data hazards occur when instructions that exhibit data dependence, modify data in different stages of a pipeline.
- There are mainly three types of data hazards:
  - 1) RAW (Read after Write)
  - 2) WAR (Write after Read)
  - 3) WAW (Write after Write)



# RAW (Read after Write)

- RAW data hazard occurs when an instruction refers to a result that has not yet been calculated or retrieved.
- E.g.: Consider two instructions  $I_1$  and  $I_2$ , such that  $I_2$  follow  $I_1$ .  
     $I_1: R1 \leftarrow R1 + R2$   
     $I_2: R3 \leftarrow R1 + R4$
- Here instruction  $I_2$  tries to read data before instruction  $I_1$  writes it.

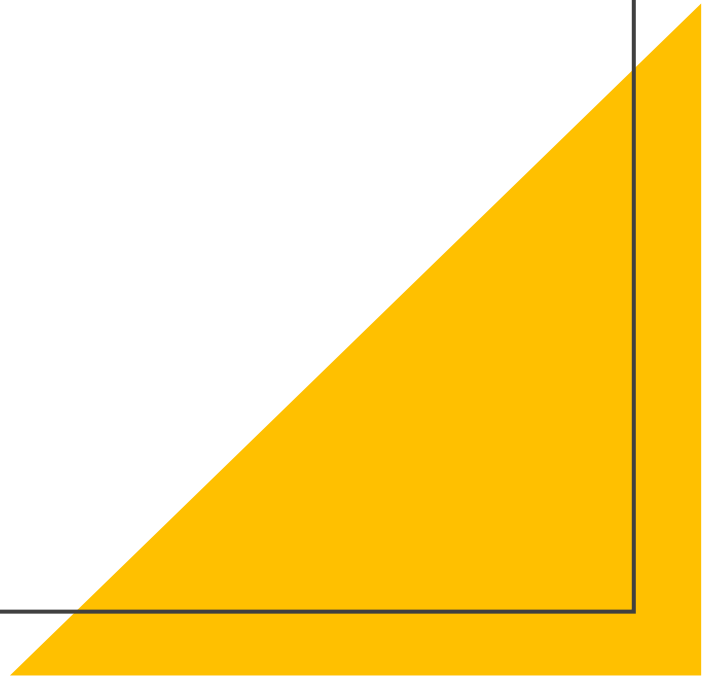


# WAR (Write after Read)

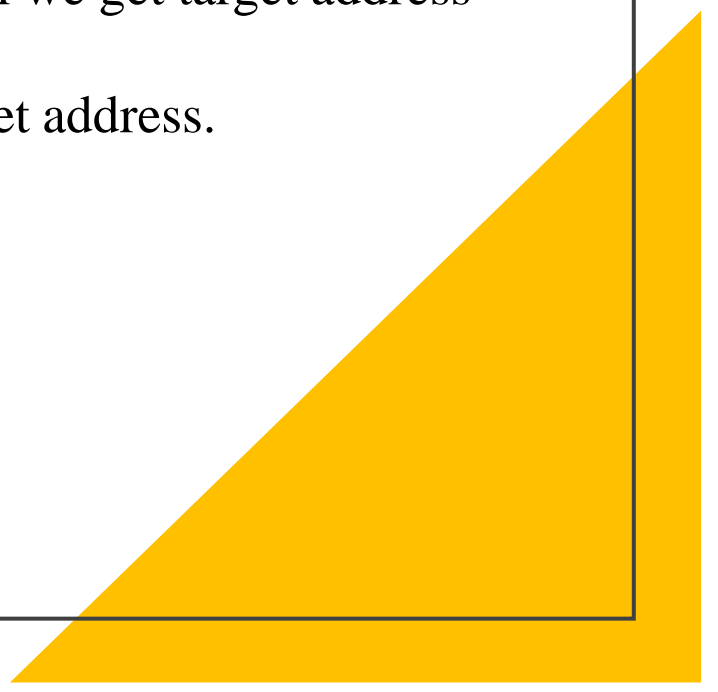
- WAR data hazard occurs either when there are early writes and late reads, or when instructions are re-ordered.
- E.g: Consider two instructions  $I_1$  and  $I_2$ , such that  $I_2$  follow  $I_1$ .  
     $I_1: R1 \leftarrow R2 + R3$   
     $I_2: R2 \leftarrow R4 + R5$
- Here instruction  $I_2$  tries to write data before instruction  $I_1$  reads it.

# WAW (Write after Write)

- WAW data hazard occurs when there are multiple writes .
- E.g: Consider two instructions  $I_1$  and  $I_2$ , such that  $I_2$  follow  $I_1$ .  
     $I_1 : R3 \leftarrow R1 * R2$   
     $I_2 : R3 \leftarrow R4 + R5$
- Here instruction  $I_2$  tries to write output before instruction  $I_1$  writes it.



# Control Hazards

- This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc.
  - On many instruction architectures, the processor will not know the target address of these instructions when it needs to insert the new instruction into the pipeline.
  - Due to this, unwanted instructions are fed to the pipeline.
  - To correct the above problem, we need to stop the Instruction fetch until we get target address of branch instruction.
  - This can be implemented by introducing delay slot until we get the target address.
- 
- A large yellow right-angled triangle is positioned in the bottom right corner of the slide, extending from the bottom edge and the right edge towards the center.

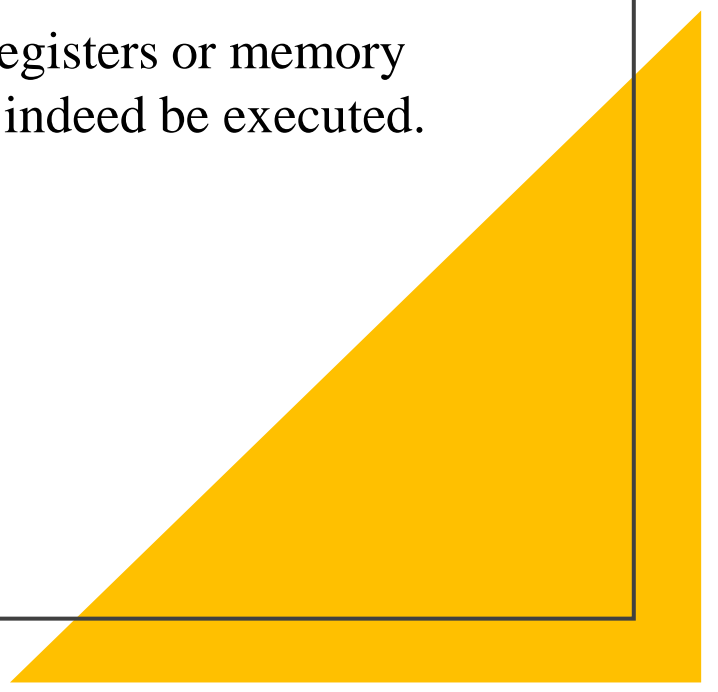
# Delayed Branch

- **Delayed Branching** can minimize the penalty incurred as a result of conditional branch instructions.
- **Idea-** The instructions in the delay slots are always fetched.  
(Branch delay slot- the location following the branch instruction)
- Therefore, fully execute those instructions whether the branch is taken or not.
- The objective is to be able to place useful instructions in these slots.
- If no useful instruction can be placed in the delay slots, fill it with NOP (No-operation) instructions.

# Branch Prediction

- Another technique for reducing branch penalty associated with conditional branch is to predict whether or not a particular branch will be taken.

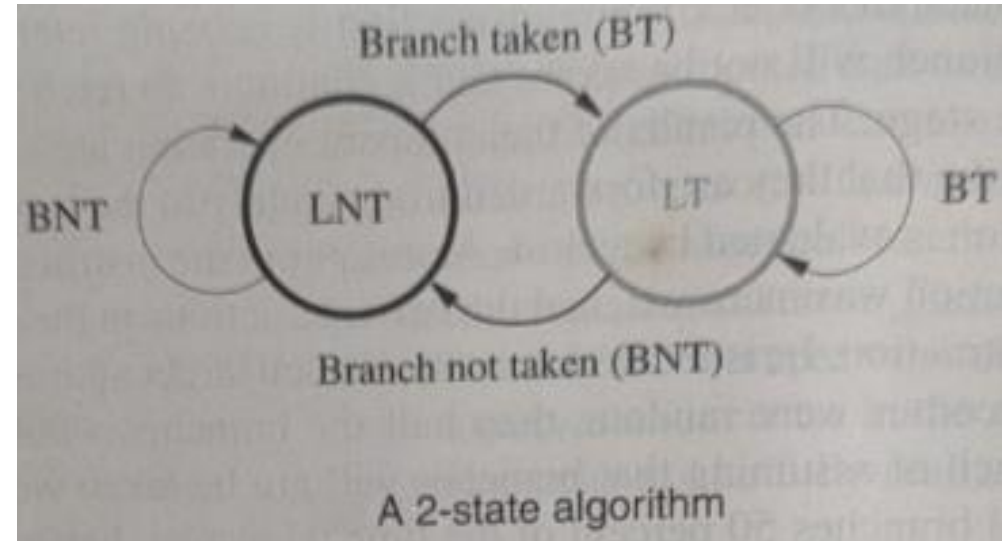
## **Static Branch Prediction**

- Simplest form- assume that the branch will not take place and continue fetching instructions in sequential address order.
  - Speculative execution- therefore, care must be taken that no processor registers or memory locations are updated until it is confirmed that these instructions should indeed be executed.
- 
- A large yellow right-angled triangle is positioned in the bottom right corner of the slide, extending from the bottom edge and the right edge towards the center.

# Branch Prediction

## Dynamic Branch Prediction

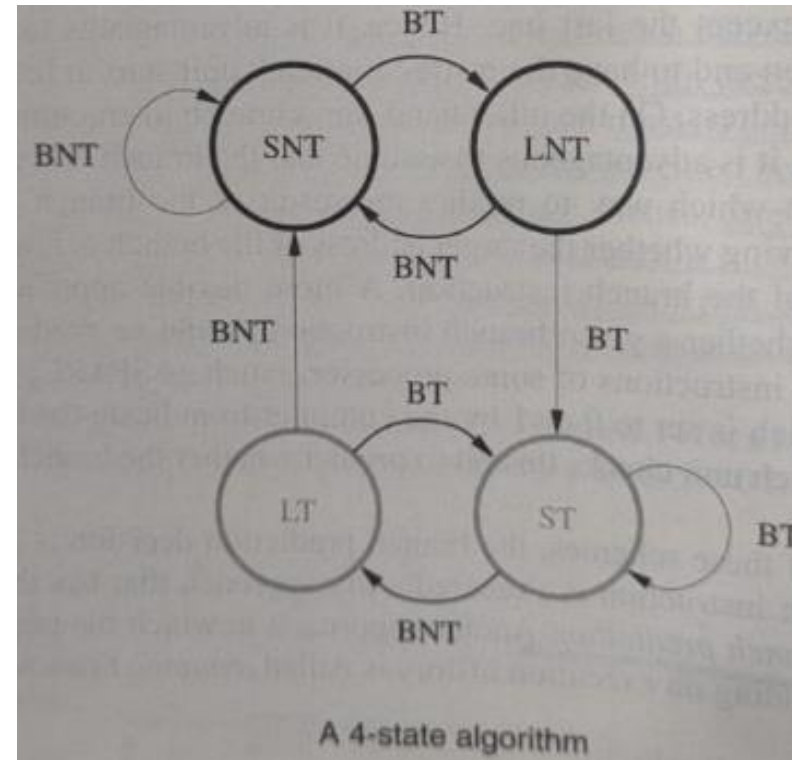
- Simplest form- the execution history is used in predicting the outcome of a given branch instruction.
- Two states:
  - LT: Branch is likely to be taken**
  - LNT: Branch is likely not to be taken**
- Works well inside program loops



# Branch Prediction

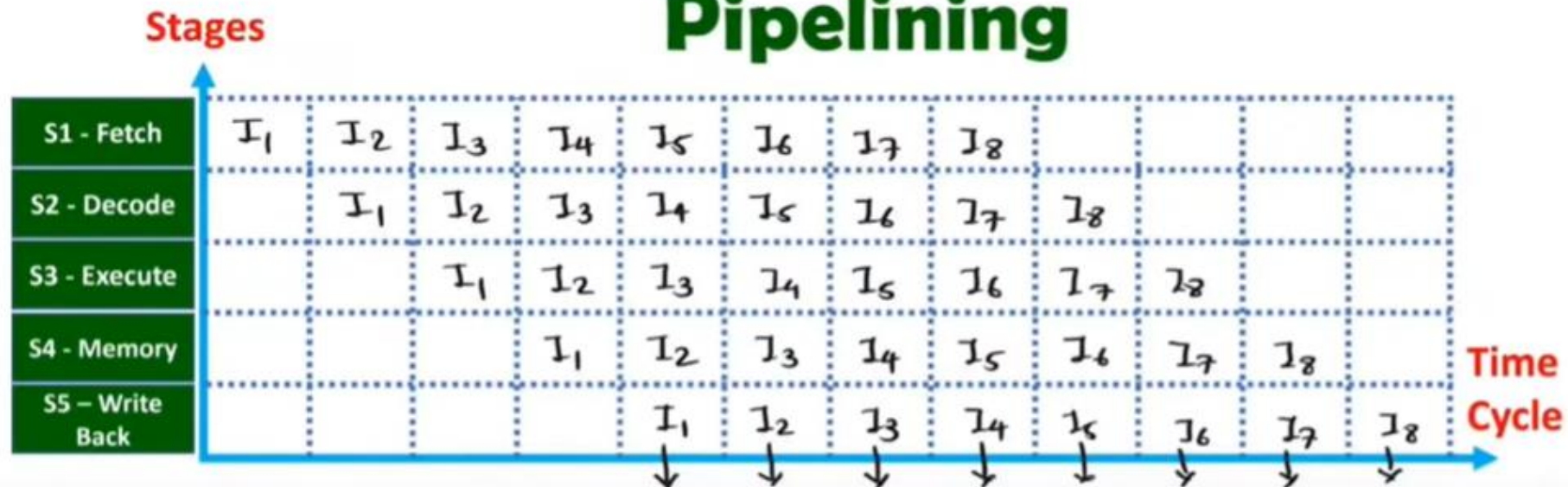
## Dynamic Branch Prediction

- Four states:
  - ST: Strongly likely to be taken**
  - LT: Likely to be taken**
  - LNT: Likely not to be taken**
  - SNT: Strongly likely not to be taken**



# Pipeline Performance

## Instruction Space time Diagram with Pipelining





# Pipeline Performance

***k*-stage pipeline processes *n* tasks in *k* + (*n*-1) clock cycles:**

***k* cycles for the first task and *n*-1 cycles for the remaining *n*-1 tasks**

**Total time to process *n* tasks**

$$T_k = [k + (n-1)] \tau$$

**For the non-pipelined processor**

$$T_1 = n k \tau$$

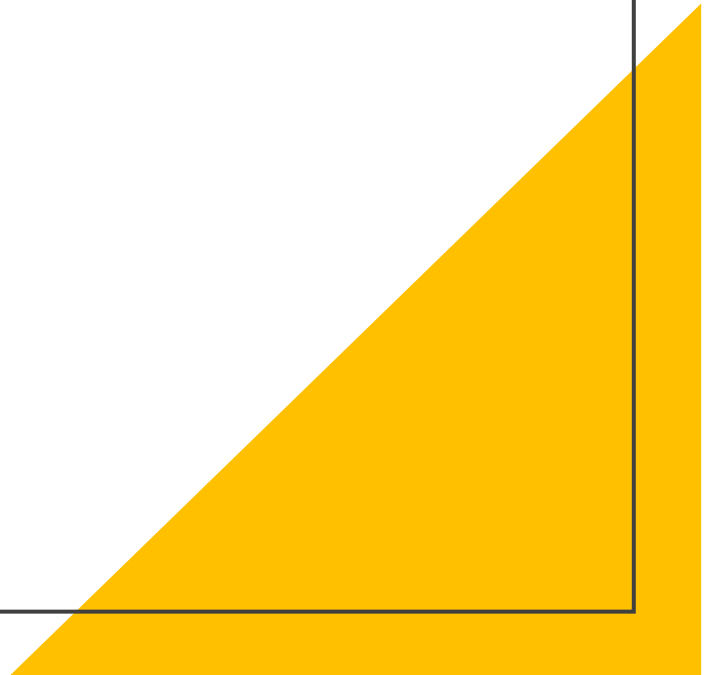
**Speedup factor**

$$S_k = \frac{T_1}{T_k} = \frac{n k \tau}{[k + (n-1)] \tau} = \frac{n k}{k + (n-1)}$$

**Efficiency =  $nk / (k+n-1)k$**

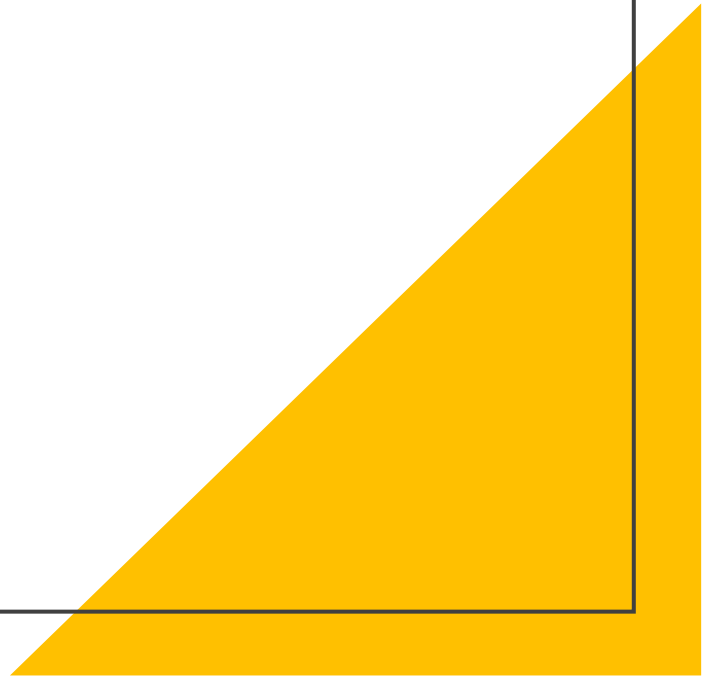
# Flynn's Classification

- Flynn's Classification is the most popular taxonomy of computer architecture.
- In 1966, Michael J Flynn classified computers based on **multiplicity of instruction stream and data streams** in a computer system.
- **Instruction stream:** is the sequence of instructions as executed by the machine
- **Data Stream:** is a sequence of data including input, or partial or temporary result, called by the instruction Stream.



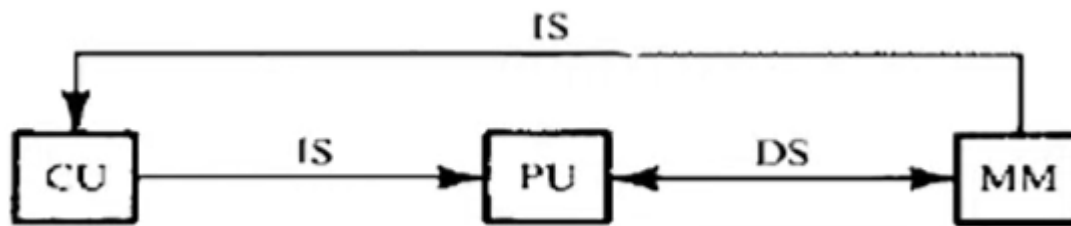
# Flynn's Classification

- According to Flynn's Classification either of the **instruction or data streams can be single or multiple**.
  1. Single Instruction Single Data (SISD)
  2. Single Instruction Multiple Data (SIMD)
  3. Multiple Instruction Single Data (MISD)
  4. Multiple Instruction Multiple Data (MIMD)



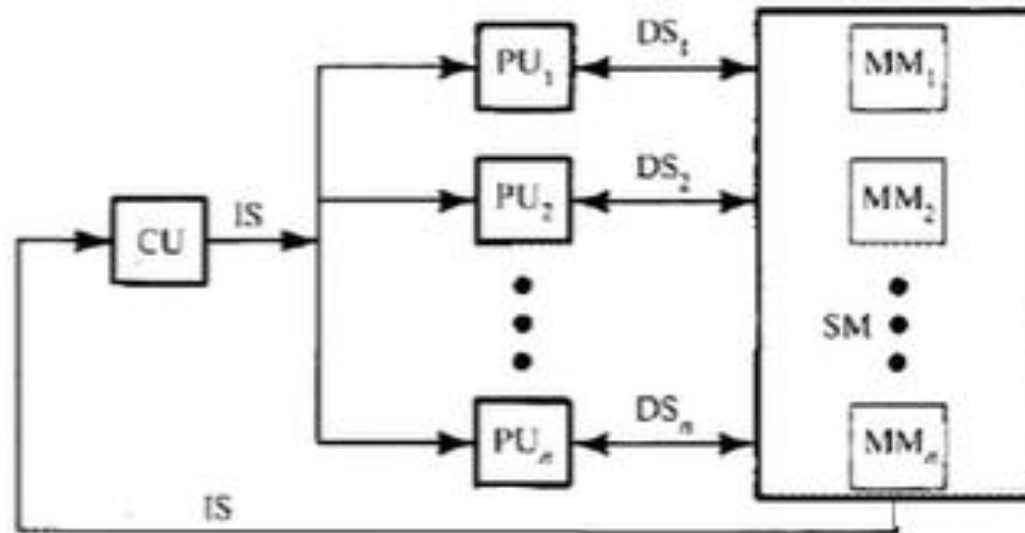
# Single Instruction Single Data (SISD)

- In a SISD architecture, there is a single processor that executes a single instruction stream and operates on a single data stream.
- Also called as sequential computers.
- This is the simplest type of computer architecture and is used in most traditional computers.
- Instructions are executed sequentially but may be overlapped in their execution stages (Pipelining).
- Examples of SISD architecture are the traditional uniprocessor machines like a PC or old mainframes.



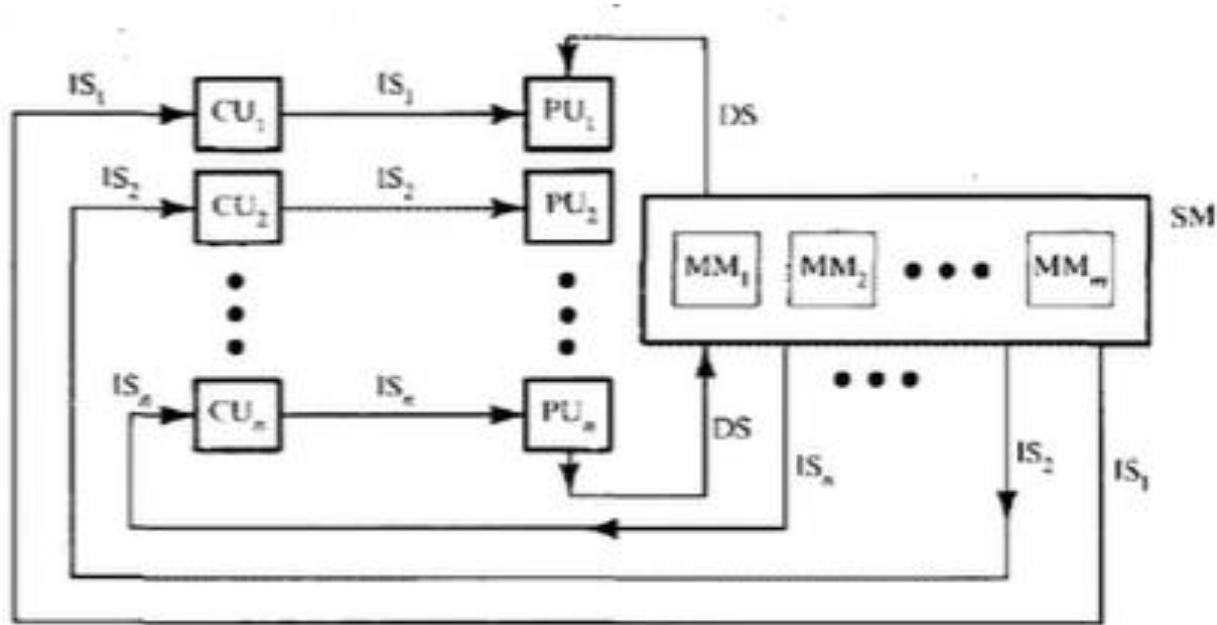
# Single Instruction Multiple Data (SIMD)

- In a SIMD architecture, there is a single processor that executes the same instruction on multiple data streams in parallel.
- SIMD computer has single control unit which issues one instruction at a time, but it has multiple ALU's or processing units to carry out on multiple data sets simultaneously.
- For example, vector and array processors.
- This type of architecture is used in applications such as image and signal processing.



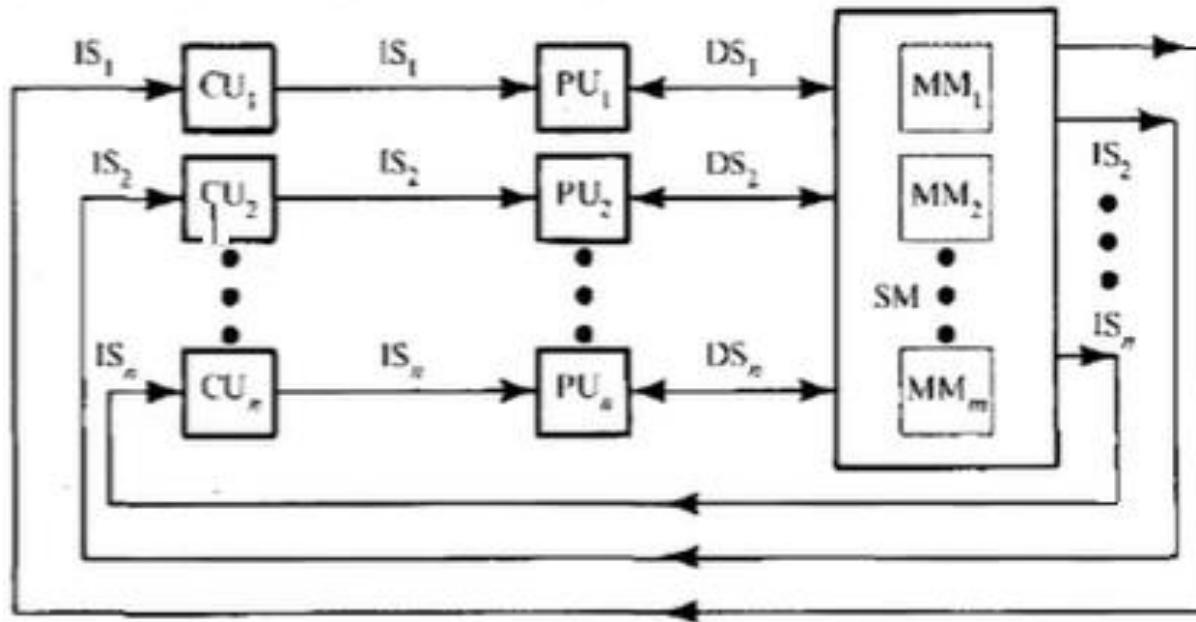
# Multiple Instruction Single Data (MISD)

- In a MISD architecture, multiple processors execute different instructions on the same data stream.
- This type of architecture is not commonly used in practice, as it is difficult to find applications that can be decomposed into independent instruction streams.
- Example: systolic arrays

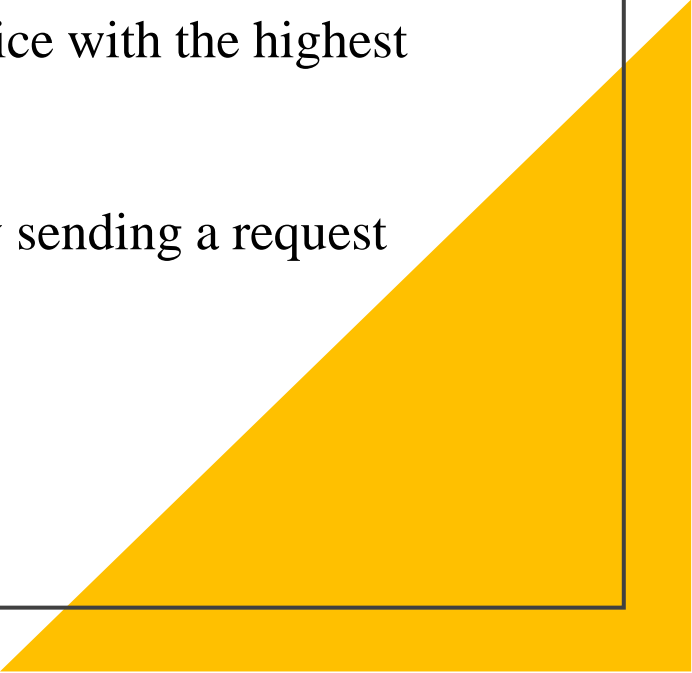


# Multiple Instruction Multiple Data (MIMD)

- In a MIMD architecture, multiple processors execute different instructions on different data streams.
- This type of architecture is used in distributed computing, parallel processing, and other high-performance computing applications.
- Example: most current supercomputers, IBM 370.



# Bus Contention and Arbitration

- Bus arbitration is the process of resolving conflicts that arise when multiple devices attempt to access the bus at the same time.
  - The Bus Arbiter decides who would become the current bus master.
  - Types-
    1. **Centralized-** bus controller is responsible for managing access to the bus
    2. **Decentralized-** each device has its own priority level, and the device with the highest priority is given access to the bus
    3. **Distributed arbitration-** devices compete for access to the bus by sending a request signal and waiting for a grant signal.
- 

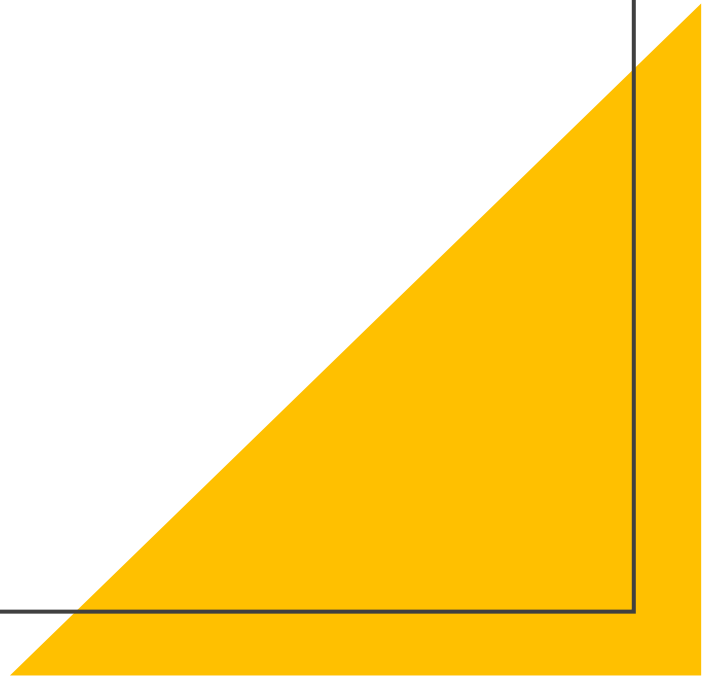


# Bus Contention and Arbitration

## **Centralized**

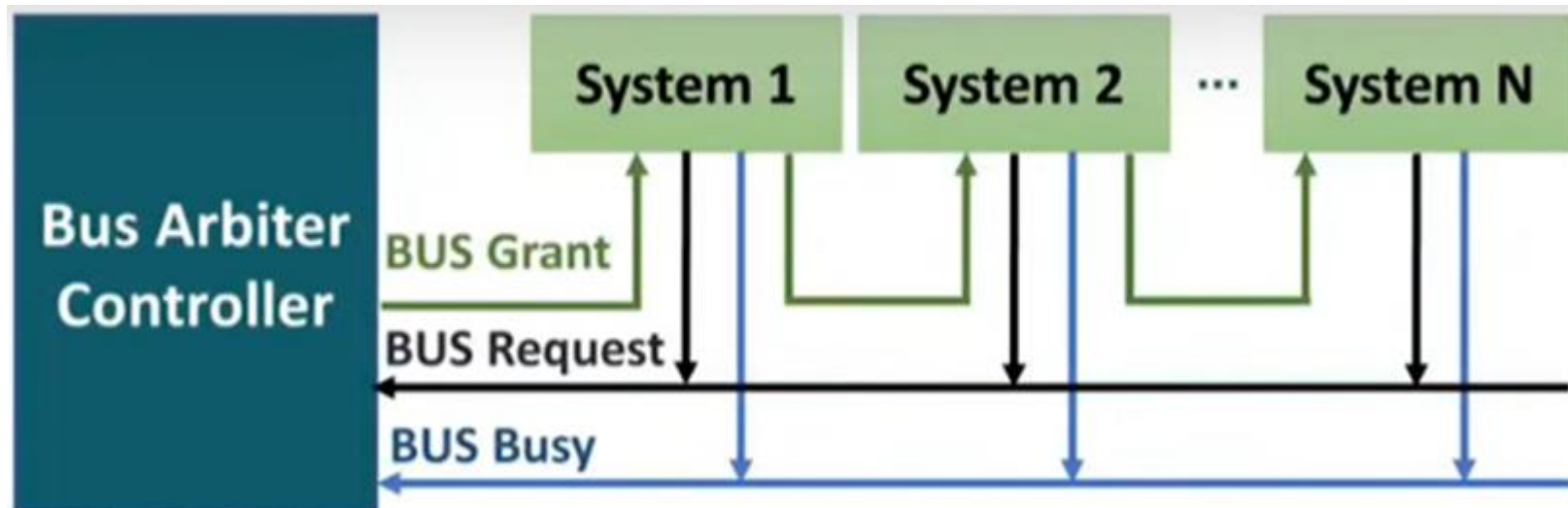
There are three bus arbitration methods:

- Daisy Chain method
- Polling method or Rotating Priority method
- Fixed priority or Independent Request method



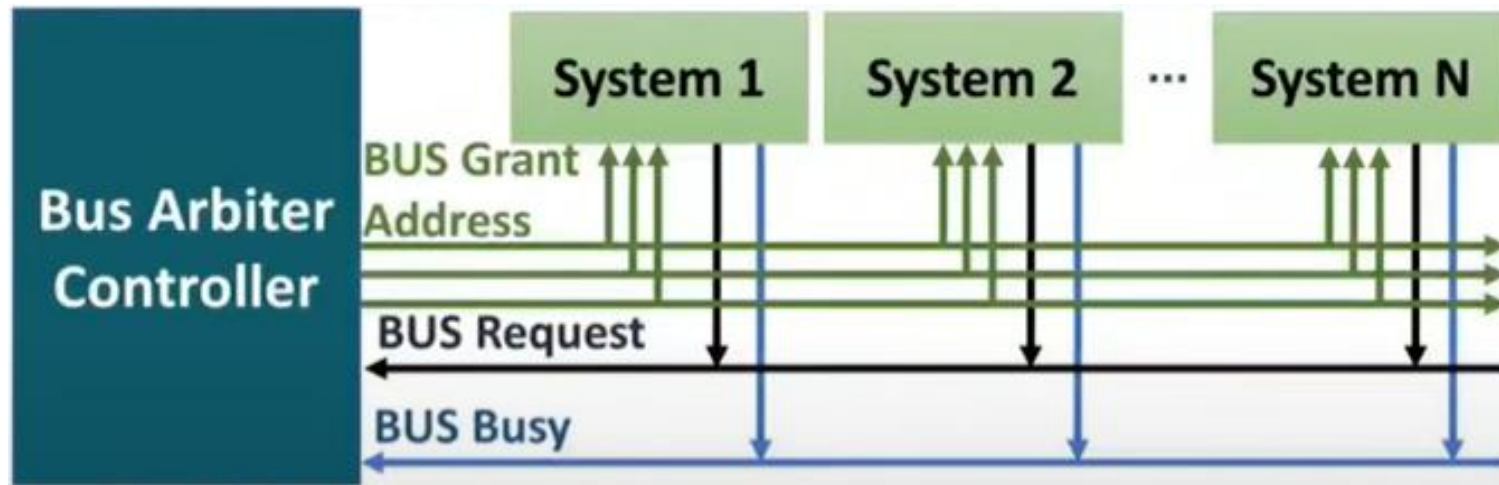
# Daisy Chain method

- All bus Masters have same line for bus request.
- If bus busy line is inactive then bus controller gives the bus grant signal.
- Bus grant signal is transmitted serially through all the systems or bus masters.
- If bus master requires system bus, then it will activate the bus busy signal and take control of system bus.



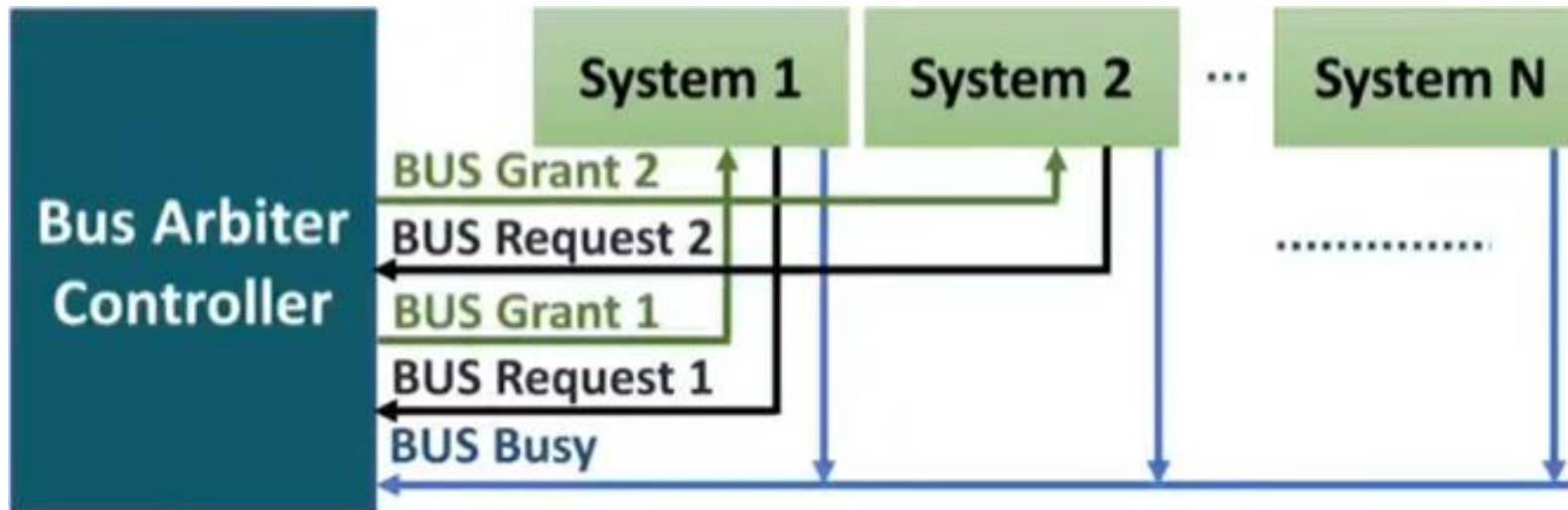
# Polling method

- All bus Masters have same line for bus request.
- If bus busy line is inactive then bus controller gives the bus grant address based on bus request.
- Bus controller polls the system in well define order as per priority.
- Once system receives its own address, it will active the bus busy signal and take control of system bus.



# Independent Request method

- All bus Masters have individual bus request lines.
- So, controller knows which system is asking for bus request.
- Priority of all systems is predefined.
- So based on bus availability and priority, bus grant is given to systems.
- Controller consists encoder and decoder logic for priorities.

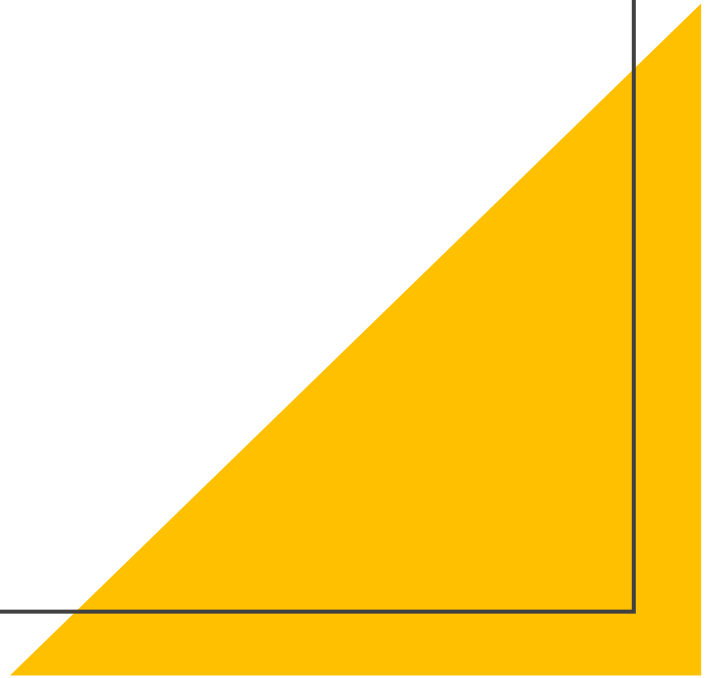


# Comparison of Daisy Chain, Polling & Independent Request

Parameters	Daisy Chain	Polling	Independent Request
Design	Simplest	Little Complex	Complex
Reliability	Not Reliable if any system fails.	Fully Reliable	Fully Reliable
Performance	Weak	Better then Daisy Chain	Excellent
Upgradation	Easy	Reconfiguration requires	Complexity Increases
Cost	Minimum	Less	High
Lines for 8 systems	Bus Grant = 1 Bus Request = 1 Bus Busy =1	Bus Grant = 3 Bus Request = 1 Bus Busy =1	Bus Grant = 8 Bus Request = 8 Bus Busy =1

# PCI (Peripheral Component Interconnect) Bus

- PCI bus was developed by Intel to replace the Industry Standard Architecture (ISA).
- One of the fastest types of PC expansion bus.
- The PCI bus is a high-performance bus widely used, from embedded systems to enterprise servers.
- The PCI bus supports higher speed devices/applications such as audio, streaming video, interactive gaming, modems, etc.

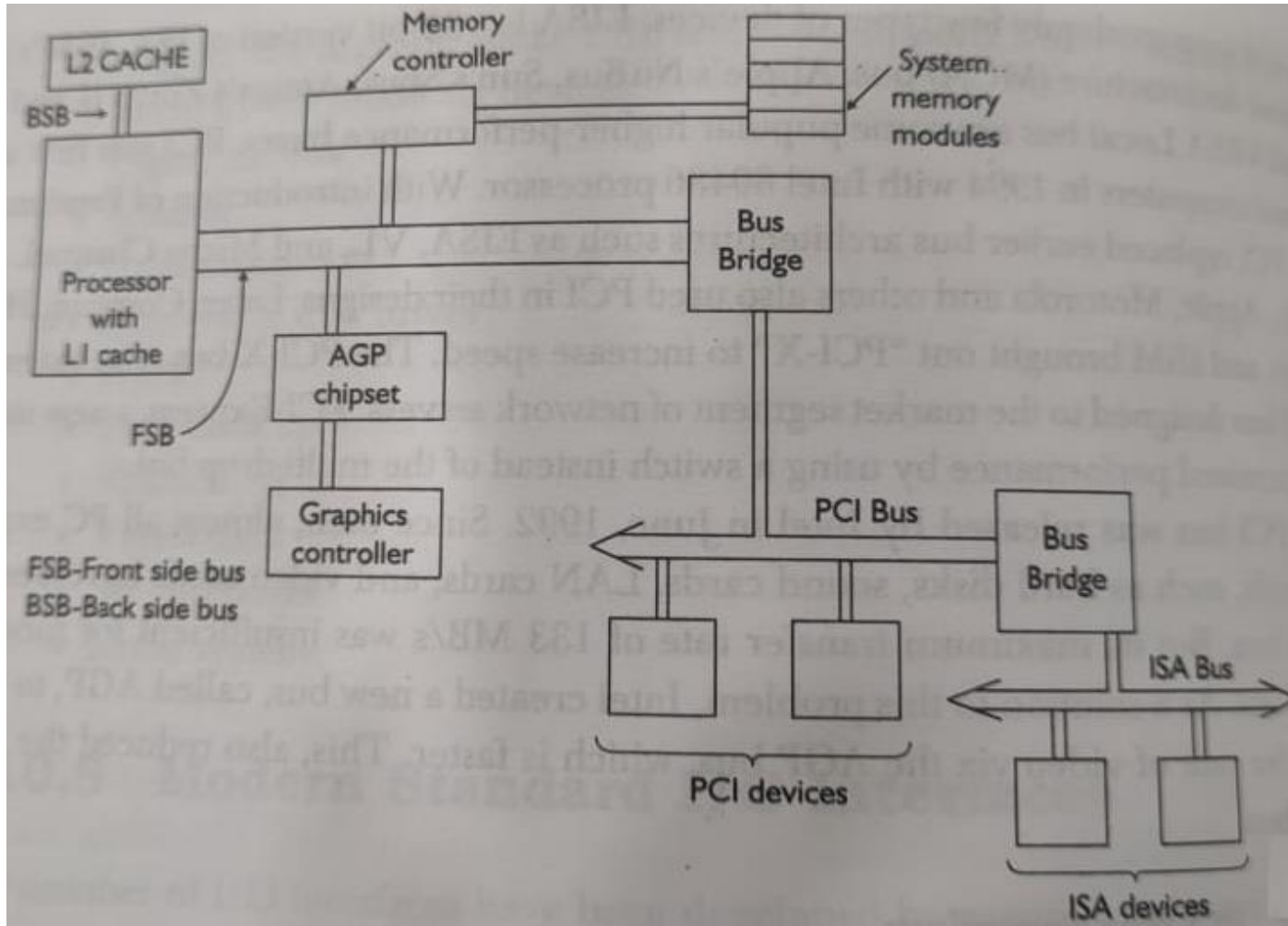


# PCI Bus

PCI bus has become very popular because of several attractive features.

1. **Speed:** PCI bus provides extremely high-speed transfers.
2. **Burst mode:** A burst of data means a series of words
3. **PCI Bridge:** To add more expansion slots, PCI-to-PCI bridges are used
4. **PCI advantages:**
  - PCI devices can have direct access to system memory without involving the CPU
  - A single PCI bus can have up to five devices connected to it
  - PCI supports bridges to handle large number of devices
  - PCI supports auto configuration
  - PCI bus is processor independent and hence can be used with any processor
5. **Voltage Requirements:** Original PCI used 5V power. Subsequent PCI versions supported both 5 Volt and 3.3 Volt.
6. **PCI Variants and Clock speed:** The initial PCI supported maximum clock rate of 33 MHz. At 33 MHz, a 32-bit slot gives a maximum transfer rate of 132 MBytes/sec, and a 64-bit slot gives 264 MBytes/sec. PCI revision 2.1 supported maximum clock at 66 MHz

# PCI Bus





# PCI Bus Features

PCI bus has become very popular because of several attractive features.

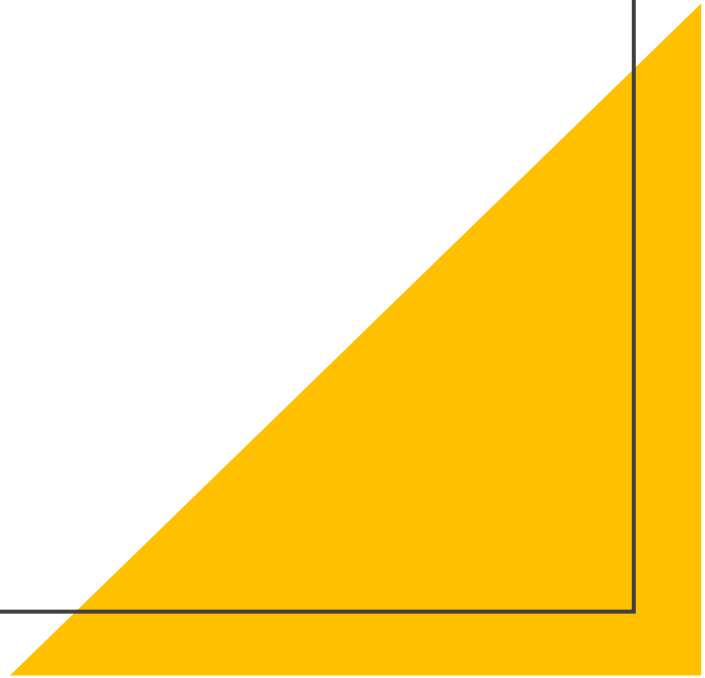
- 1. Bus Mastering and Bus Arbitration:** The arbitration mechanism allows any device to request control of the processor bus. PCI has an arbitrator who can grant the use of the bus. Any device can get control of the bus. REQ# and GNT# are unique for every slot.
- 2. Plug and Play and Auto Configuration:** Plug and play feature allows addition of a device without need for any manual configuration.
- 3. Multiplexed Bus:**
  - Initiator and Target
  - Address Phase and Data Phase
  - Termination

# USB (Universal Serial Bus)

- It is a modern serial interfaces which provides high performance and flexibility.
- The features of USB are as follows:
  1. **Multiple devices:** Upto 127 different devices can be connected on a single USB bus.
  2. **Transfer rate:** Initial-12 Mbps transfer rate. The USB 2.0 supports higher rates.
  3. **Support for wide range of peripherals:** Low bandwidth devices such as keyboard, mouse, joystick, game pad, floppy disk drive, zip drive, printer, scanner etc.
  4. **Hub architecture:** Each device is connected to an USB hub. The USB hub is an intelligent unit interacting to the PC on one side and the peripheral devices on other sides. It is more like a multi “tiered star topology”. Hence, a single USB hub establishes presence of multiple USB devices (upto 127)
  5. **Hot pluggability:** A USB device can be connected without powering-off a PC. The ‘Plug and Play’ feature take care of detection, device recognition and handling. The user is totally relieved of configuration procedures.

# USB (Universal Serial Bus)

- 6. **Power allocation:** USB controller in PC detects the presence (attachment) or absence (detachment) of the USB devices and allocates of appropriate levels of electrical power.
- 7. **Ease of installation:** A 4-pin cable carries signals
- 8. **Host centric:** The CPU/software initiates every transaction on the USB bus. Hence the overhead on the software increases when large number of peripherals, involving large number of transactions are connected.



Thank You

