UNIT VI – PIPELINE, VECTOR PROCESSING AND MULTIPROCESSORS – 6 HRS

Notes By: Raju Poudel (Mechi Multiple Campus)

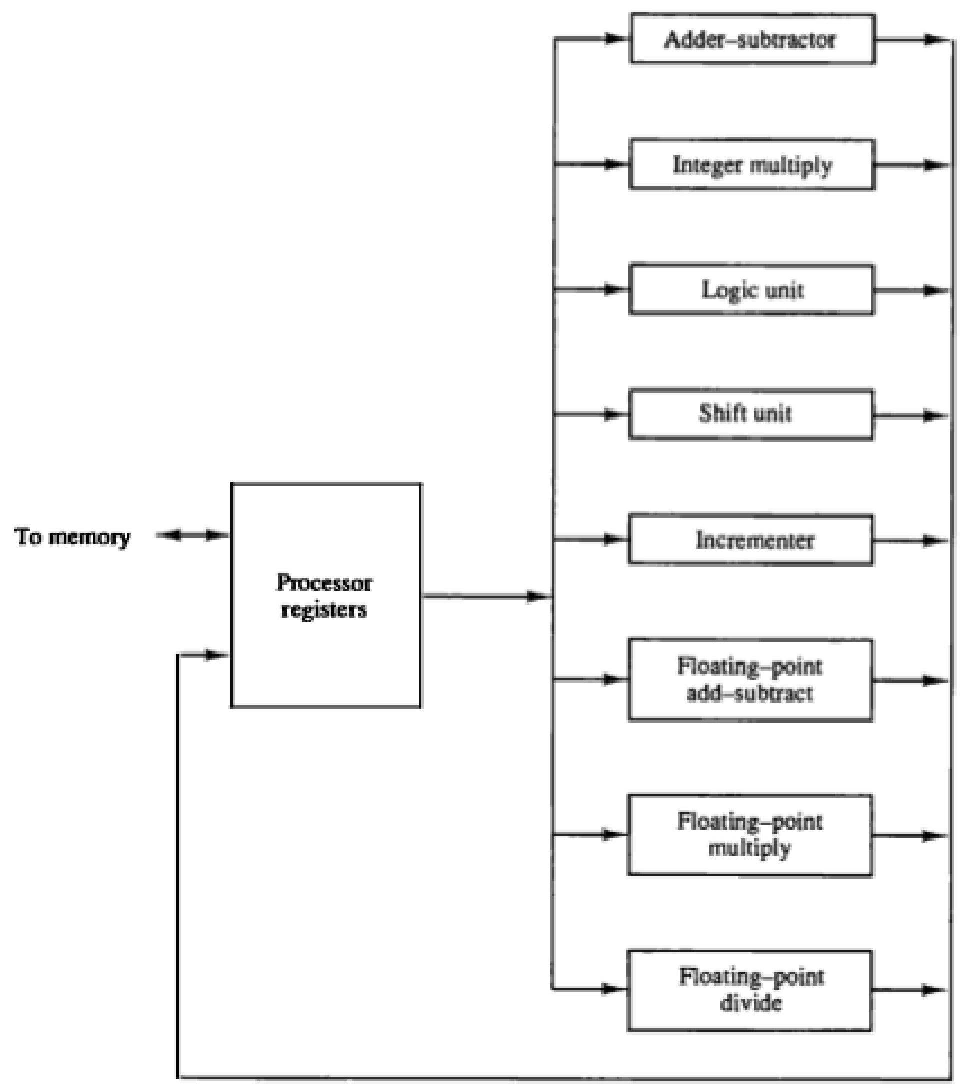
Parallel Processing

- **Parallel processing** is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.
- Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- For example, while an instruction is being executed in the ALU, the next instruction can be read from memory. The system may have two or more ALUs and be able to execute two or more instructions at the same time.
- Furthermore, the system may have two or more processors operating concurrently. The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.
- The amount of hardware increases with parallel processing. and with it, the cost of the system increases. However, technological developments have reduced hardware costs to the point where parallel processing techniques are economically feasible.

Parallel Processing

- Parallel processing is established by distributing the data among the multiple functional units. For example, the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.
- The adder and integer multiplier perform the arithmetic operations with integer numbers.
- The floating-point operations are separated into three circuits operating in parallel.
- The logic, shift, and increment operations can be performed concurrently on different data.
- All units are independent of each other, so one number can be shifted while another number is being incremented.
- A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components.

Figure 9-1 Processor with multiple functional units.



Flynn's Classification of Parallel Processing

- ❖ There are a variety of ways that parallel processing can be classified. It can be considered from the internal organization of the processors, from the interconnection structure between processors, or from the flow of information through the system.
- ❖ One classification introduced by **M. J. Flynn** considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.
- ❖ The normal operation of a computer is to fetch instructions from memory and execute them in the processor. The sequence of instructions read from memory constitutes an **instruction stream**. The operations performed on the data in the processor constitutes a **data stream**. Parallel processing may occur in the **instruction stream, in the data stream, or in both**.
- ❖ **Flynn's** classification divides computers into **four major groups** as follows:
  1. Single instruction stream, single data stream (SISD)
  2. Single instruction stream, multiple data stream (SIMD)
  3. Multiple instruction stream, single data stream (MISD)
  4. Multiple instruction stream, multiple data stream (MIMD)

Flynn's Classification of Parallel Processing

- **SISD** represents the organization of a single computer containing a control unit, a processor unit, and a memory unit. Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities. Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

- **SIMD** represents an organization that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data. The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

- **MISD** structure is only of theoretical interest since no practical system has been constructed using this organization.

- **MIMD** organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multicomputer systems can be classified in this category.

Pipelining

- **Pipelining** is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.
- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline. The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

Pipelining Example

- The pipeline organization will be demonstrated by means of a simple example. Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

$$A_i * B_i + C_i \qquad \text{for } i = 1, 2, 3, \ldots, 7$$

- Each sub-operation is to be implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit as shown in Fig. 9-2.
- R1 through R5 are registers that receive new data with every clock pulse. The multiplier and adder are combinational circuits. The sub-operations performed in each segment of the pipeline are as follows:

$R1 \leftarrow A_i, \quad R2 \leftarrow B_i$	Input A_i and B_i
$R3 \leftarrow R1 * R2, \quad R4 \leftarrow C_i$	Multiply and input C_i
$R5 \leftarrow R3 + R4$	Add C_i to product

Figure 9-2 Example of pipeline processing.

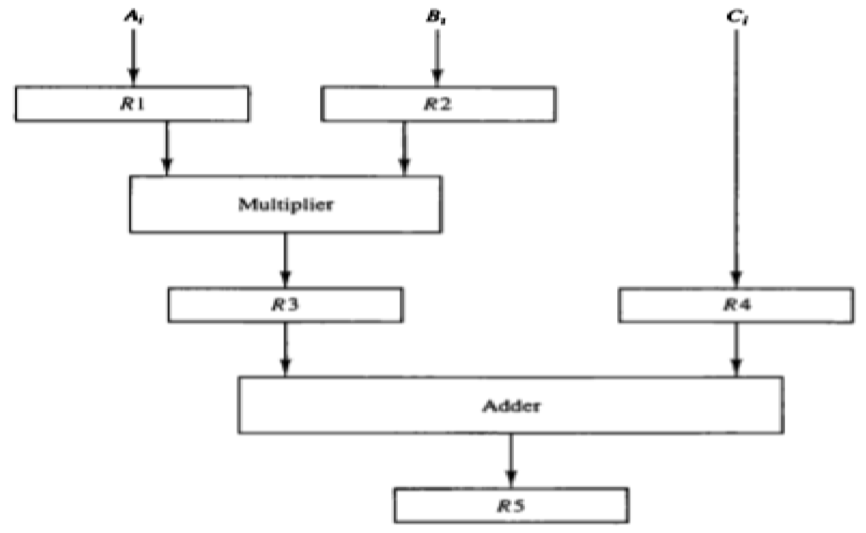
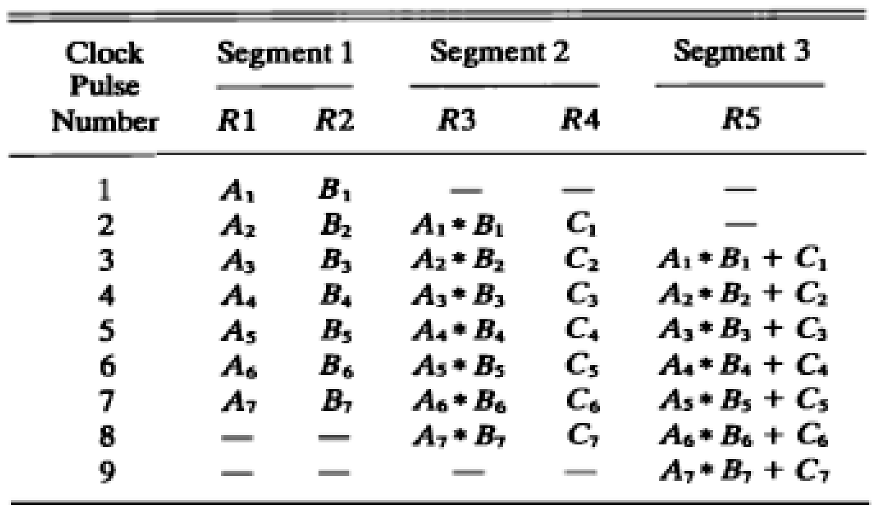


TABLE 9-1 Content of Registers in Pipeline Example



Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

Instruction Pipeline

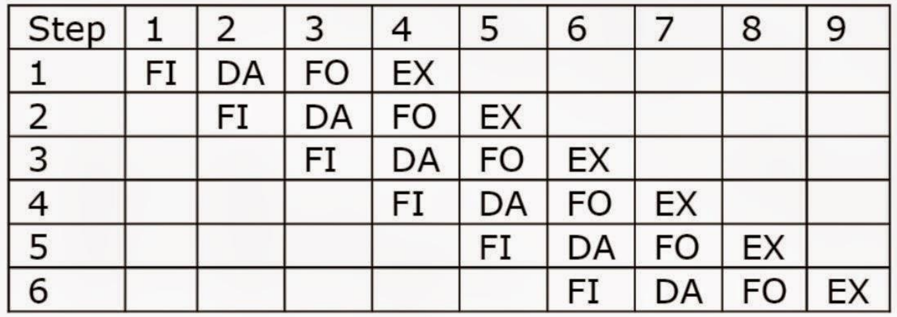
- Pipeline processing can occur not only in the data stream but in the instruction stream as well. **An instruction pipeline** reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and execute phases to overlap and perform simultaneous operations.
- Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely. In the most general case, the computer needs to process each instruction with the following sequence of steps:

  1. Fetch the instruction from memory.
  2. Decode the instruction.
  3. Calculate the effective address.
  4. Fetch the operands from memory.
  5. Execute the instruction.
  6. Store the result in the proper place.

- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration. The time that each step takes to fulfill its function depends on the instruction and the way itis executed.

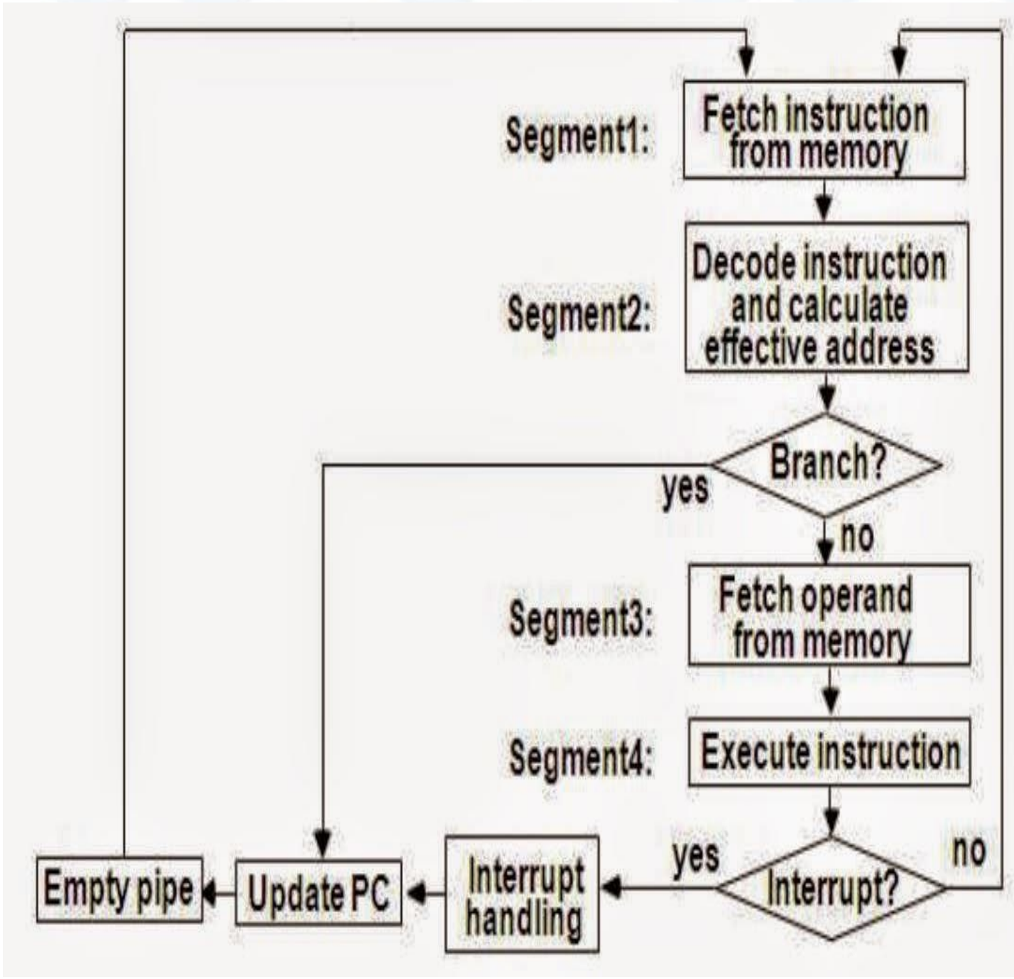
Example: Four-Segment Instruction Pipeline

- The above figure shows operation of 4-segment instruction pipeline. The four segments are represented as:
  1. **FI**: segment 1 that fetches the instruction.
  2. **DA**: segment 2 that decodes the instruction and calculates the effective address.
  3. **FO**: segment 3 that fetches the operands.
  4. **EX**: segment 4 that executes the instruction.
- The space time diagram for the 4-segment instruction pipeline is given below:



Step	1	2	3	4	5	6	7	8	9
1	FI	DA	FO	EX					
2		FI	DA	FO	EX				
3			FI	DA	FO	EX			
4				FI	DA	FO	EX		
5					FI	DA	FO	EX	
6						FI	DA	FO	EX

Fig: timing diagram for 4-segment instruction pipeline



Pipeline Conflicts (Hazards)

❖ A pipeline hazard occurs when the instruction pipeline deviates at some phases, some operational conditions that do not permit the continued execution. In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. **Resource conflicts** caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2. **Data dependency** conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

3. **Branch difficulties** arise from branch and other instructions that change the value of PC.

Data Dependency

- ❖ It arises when instructions depend on the result of previous instruction but the previous instruction is not available yet.
- ❖ **For example** an instruction in segment may need to fetch an operand that is being generated at same time by the previous instruction in the segment.
- ❖ The most common techniques used to resolve data hazard are:

(a) **Hardware interlock** - a hardware interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline. It then inserts enough number of clock cycles to delays the execution of such instructions.

(b) **Operand forwarding -** This method uses a special hardware to detect conflicts in instruction execution and then avoid it by routing the data through special path between pipeline segments. For example, instead of transferring an ALU result into a destination result, the hardware checks the destination operand, and if it is needed in next instruction, it passes the result directly into ALU input, bypassing the register.

(c) **Delayed load -** It is software solutions where the compiler is designed in such a way that it can detect the conflicts; re-order the instructions to delay the loading of conflicting data by inserting no operation instruction.

Handling of Branch Instructions

- **Branch hazard** arises from branch and other instruction that change the value of program counter (PC). The conditional branch provides plenty of instruction branch line and it is difficult to determine which branches will be taken or not taken. A variety of approaches have been used to deal with branch hazard and they are described below.

(a) **Multiple streaming -** It is a brute-force approach which replicates the initial portions of the pipeline and allows the pipeline to fetch both instructions, making use of two streams (branches).

(b) **Prefetch branch target -** When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch. This target is then saved until the branch instruction is executed. If the branch is taken, the target has already been prefetched.

(c) **Branch prediction -** uses additional logic to prediction the outcomes of a (conditional) branch before it is executed. The popular approaches are - predict never taken, predict always taken, predict by opcode, taken/not taken switch and using branch history table.

Handling of Branch Instructions

d) **Loop buffer -** A loop buffer is a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the n most recently fetched instructions, in sequence. If a branch is to be taken, the hardware first checks whether the branch target is within the buffer. If so, the next instruction is fetched from the buffer.

e) **Delayed branch -** This technique is employed in most RISC processors. In this technique, compiler detects the branch instructions and re-arranges the instructions by inserting useful instructions to avoid pipeline hazards.

Vector Processing

- **Vector processing** is a procedure for speeding the processing of information by a computer, in which pipelined units perform arithmetic operations on uniform, linear arrays of data values, and a single instruction involves the execution of the same operation on every element of the array.

- There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems are characterized by the fact that they require a vast number of computations that will take a conventional computer days or even weeks to complete.

- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing.
- To achieve the required level of high performance it is necessary to utilize the fastest and most reliable hardware and apply innovative procedures from vector and parallel processing techniques.

Application Areas of Vector Processing

Computers with vector processing capabilities are in demand in specialized applications. The following are representative application areas where vector processing is of the utmost importance.

- Long-range weather forecasting
- Petroleum explorations
- Seismic data analysis
- Medical diagnosis
- Aerodynamics and space flight simulations
- Artificial intelligence and expert systems
- Mapping the human genome
- Image processing

Vector Operations

- Many scientific problems require arithmetic operations on large arrays of numbers. These numbers are usually formulated as vectors and matrices of floating-point numbers.
- A **vector** is an ordered set of a one-dimensional array of data items. A vector V of length n is represented as a row vector by $V = [V_1 , V_2 , V_3, \cdot\cdot\cdot V_n]$.
- A conventional sequential computer is capable of processing operands one at a time. Consequently, operations on vectors must be broken down into single computations with subscripted variables. The element V_i of vector V is written as V(I) and the index I refers to a memory address or register where the number is stored.
- To examine the difference between a conventional scalar processor and a vector processor, consider the following Fortran DO loop:

```
      DO 20 I = 1, 100
20    C(I) = B(I) + A(I)
```

Conventional computer

```
     Initialize I = 0
20   Read A(I)
     Read B(I)
     Store C(I) = A(I) + B(I)
     Increment I = i + 1
     If I ≤ 100 goto 20
```

- This is a program for adding two vectors A and B of length 100 to produce a vector C.
- .A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop. It allows operations to be specified with a single vector instruction of the form

$$C(1 : 100) = A(1 : 100) + B(1: 100)$$

- The vector instruction includes the initial address of the operands, the length of the vectors, and the operation to be performed, all in one composite instruction.

Matrix Multiplication

- **Matrix multiplication** is one of the most computational intensive operations performed in computers with vector processors. An n x m matrix of numbers has n rows and m columns and may be considered as constituting a set of n row vectors or a set of m column vectors. Consider, for example, the multiplication of two 3 x 3 matrices A and B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

- For example, the number in the first row and first column of matrix C is calculated by letting i = 1, j = 1, to obtain $c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$

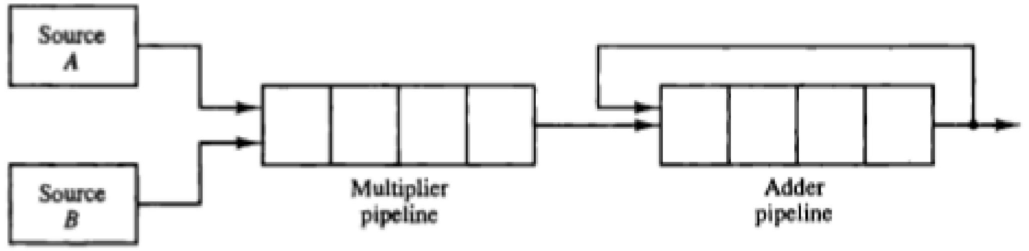
Inner Product

In general, the inner product consists of the sum of k product terms of the form

$$C = A_1B_1 + A_2B_2 + A_3B_3 + A_4B_4 + \cdots + A_kB_k$$

In a typical application k may be equal to 100 or even 1000. The inner product calculation on a pipeline vector processor is shown below:

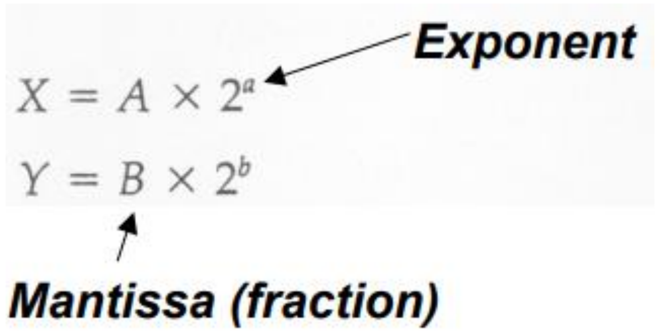
$$\begin{aligned} C = {} & A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \cdots \\ & + A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \cdots \\ & + A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \cdots \\ & + A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \cdots \end{aligned}$$



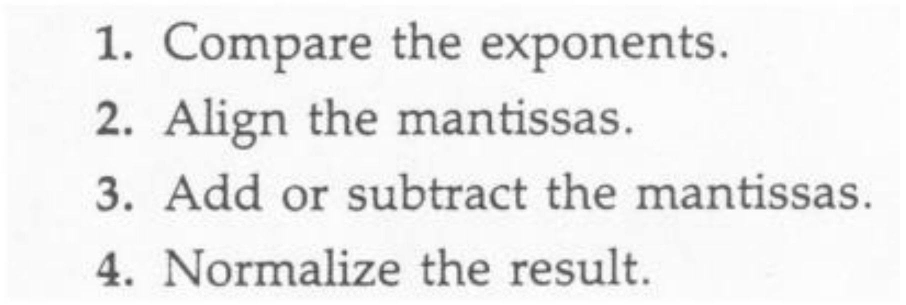
Arithmetic Pipeline

- **Pipeline arithmetic** units are usually found in very high speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems.

- Let’s take an example of a pipeline unit for floating-point addition and subtraction. The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.



4-Segment Pipeline :



1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

Arithmetic Pipeline

Floating-point Add/Subtraction Pipeline Example :

$$X = 0.9504 \times 10^3$$

$$Y = 0.8200 \times 10^2$$

The two exponents are subtracted in the first segment to obtain $3 - 2 = 1$. The larger exponent 3 is chosen as the exponent of the result. The next segment shifts the mantissa of Y to the right to obtain

$$X = 0.9504 \times 10^3$$

$$Y = 0.0820 \times 10^3$$

This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum

$$Z = 1.0324 \times 10^3$$

The sum is adjusted by normalizing the result so that it has a fraction with a nonzero first digit. This is done by shifting the mantissa once to the right and incrementing the exponent by one to obtain the normalized sum.

$$Z = 0.10324 \times 10^4$$

Exponent

$$X = A \times 2^a$$

$$Y = B \times 2^b$$

Mantissa (fraction)

4-Segment Pipeline :

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

Multiprocessor System

- A **multiprocessor** is a computer system with two or more central processing units (CPUs), with each one sharing the common main memory as well as the peripherals. This helps in simultaneous processing of programs.
- The key objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.
- A multiprocessor is regarded as a means to improve computing speeds, performance and cost-effectiveness, as well as to provide enhanced availability and reliability.

Characteristics:

- Consists of more than one CPU.
- Fast processing.
- Reliability
- Cost – Effective
- Simultaneous processing of programs.

Interconnection Structures for Multiprocessor System

- ❖ The components that form a multiprocessor system are CPUs, IOPs(Input Output Processors) connected to input output devices, and a memory unit.
- ❖ There are several physical forms available for establishing an interconnection network.

  - Time-shared common bus
  - Multiport memory
  - Crossbar switch
  - Multistage switching network

1. Time Shared Common Bus

A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit.

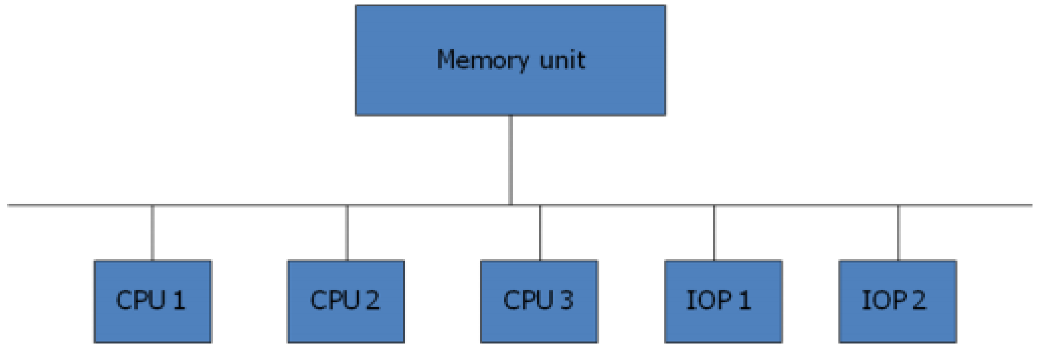


Fig: Time shared common bus organization

Interconnection Structures for Multiprocessor System

2. Multiport Memory

A multiport memory system employs separate buses between each memory module and each CPU.

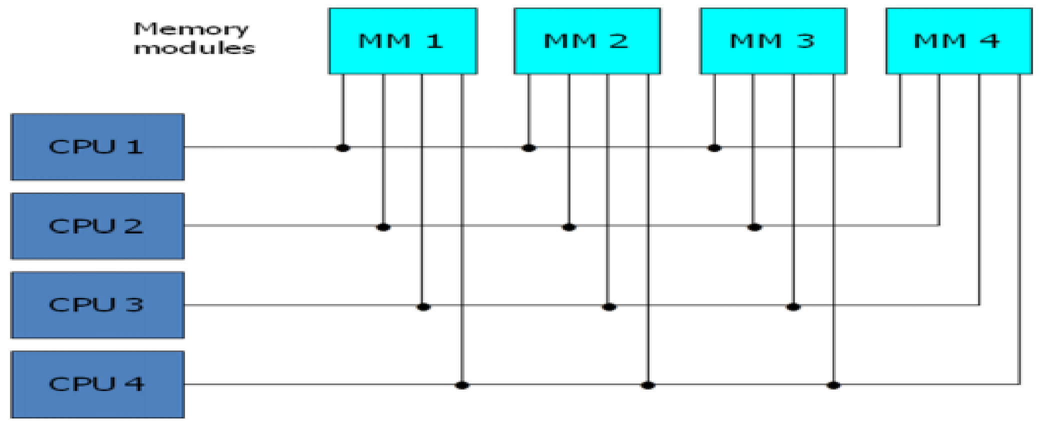


Fig: Multiport memory organization

3. Crossbar Switch

Consists of a number of cross points that are placed at intersections between processor buses and memory module paths.

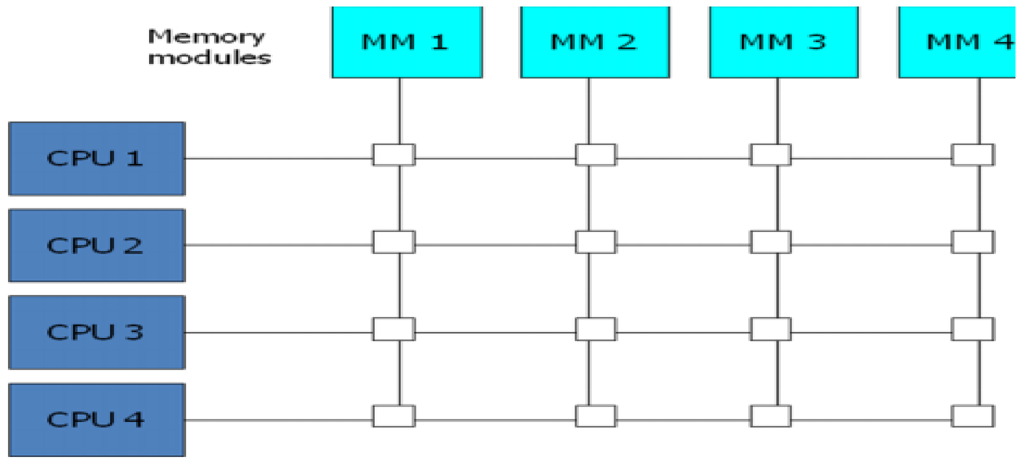
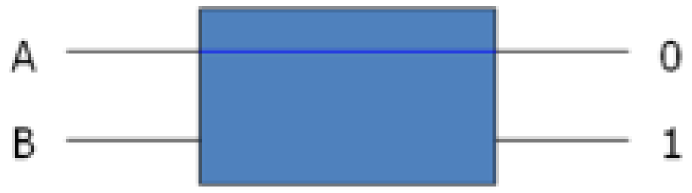


Fig: Crossbar switch

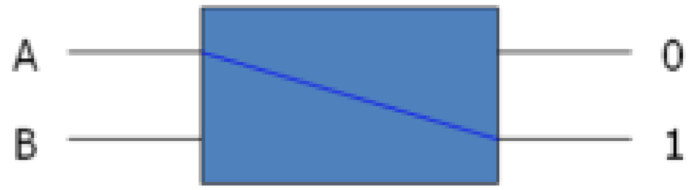
Interconnection Structures for Multiprocessor System

4. Multistage Switching Network

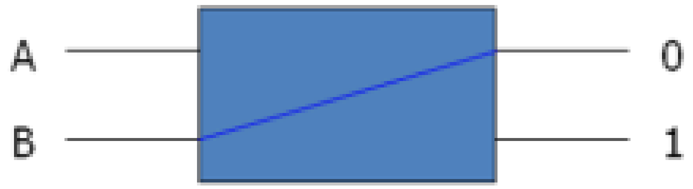
- The basic component of a multistage network is a two-input, two-output interchange switch as shown in Fig. below.



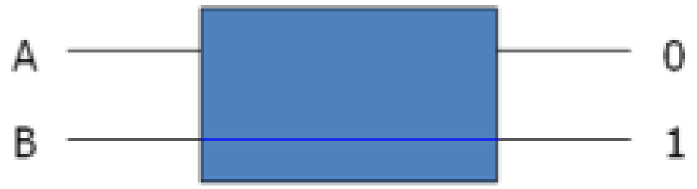
A connected to 0



A connected to 1



B connected to 0



B connected to 1