UNIT 5:

Reduced Instruction Set Computer: CISC Characteristics, RISC Characteristics.

Pipeline and Vector Processing: Parallel Processing, Pipelining, Arithmetic Pipeline, Instruction Pipeline, RISC Pipeline, Vector Processing, Array Processor.

MultiProcessors: Characteristics of Multiprocessors, Interconnection Structures, Interprocessor arbitration, Interprocessor communication and synchronization, cache Coherence.

Reduced Instruction Set Computer:

Reduced Instruction Set Architecture (RISC) –
The main idea behind this is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, a store command will store the data.

Complex Instruction Set Architecture (CISC) –
The main idea is that a single instruction will do all loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it's complex.

Earlier when programming was done using assembly language, a need was felt to make instruction do more tasks because programming in assembly was tedious and error-prone due to which CISC architecture evolved but with the uprise of high-level language dependency on assembly reduced RISC architecture prevailed.

Characteristic of RISC –

1. Simpler instruction, hence simple instruction decoding.
2. Instruction comes undersize of one word.
3. Instruction takes a single clock cycle to get executed.
4. More general-purpose registers.
5. Simple Addressing Modes.
6. Fewer Data types.
7. A pipeline can be achieved.

Characteristic of CISC –

1. Complex instruction, hence complex instruction decoding.
2. Instructions are larger than one-word size.
3. Instruction may take more than a single clock cycle to get executed.
4. Less number of general-purpose registers as operations get performed in memory itself.
5. Complex Addressing Modes.
6. More Data types.

Example – Suppose we have to add two 8-bit numbers:

- **CISC approach:** There will be a single command or instruction for this like ADD which will perform the task.
- **RISC approach:** Here programmer will write the first load command to load data in registers then it will use a suitable operator and then it will store the result in the desired location.

So, add operation is divided into parts i.e. load, operate, store due to which RISC programs are longer and require more memory to get stored but require fewer transistors due to less complex command.

Difference – RISC and CISC

RISC	**CISC**
Focus on software	Focus on hardware

RISC	**CISC**
Uses only Hardwired control unit	Uses both hardwired and microprogrammed control unit
Transistors are used for more registers	Transistors are used for storing complex Instructions
Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small
An instruction executed in a single clock cycle	Instruction takes more than one clock cycle
An instruction fit in one word	Instructions are larger than the size of one word

Both approaches try to increase the CPU performance

- **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.
- **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of an increase in the number of cycles per instruction.

Parallel processing

Execution of Concurrent Events in the computing process to achieve faster Computational Speed

Levels of Parallel Processing

- Job or Program level

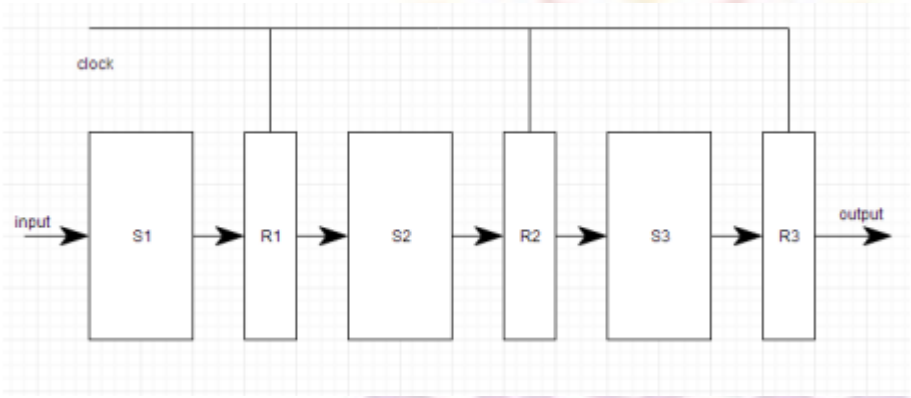
- Task or Procedure level

- Inter-Instruction level

- Intra-Instruction level

PARALLEL COMPUTERS Architectural Classification Flynn's classification » Based on the multiplicity of Instruction Streams and Data Streams » Instruction Stream Sequence of Instructions read from memory » Data Stream Operations performed on the data in the processor.

What is Pipelining? Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing. Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end. Pipelining increases the overall instruction throughput. In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment



Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm. Types of Pipeline It is divided into 2 categories: 1. Arithmetic Pipeline 2. Instruction Pipeline

Arithmetic Pipeline Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is: X = A*2^a Y = B*2^b Here A and B are mantissas (significant digit of floating point numbers), while a and b are exponents. The floating point addition and subtraction is done in 4 parts: 1. Compare the exponents. 2. Align the mantissas. 3. Add or subtract mantissas 4. Produce the result. Registers are used for storing the intermediate results between the above operations.

Instruction Pipeline In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. Advantages of Pipelining 1. The cycle time of the processor is reduced. 2. It increases the throughput of the system 3. It makes the system reliable. Disadvantages of Pipelining 1. The design of pipelined processor is complex and costly to manufacture. 2. The instruction latency is more.

Vector(Array) Processing There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items, that will take a conventional computer(with scalar processor) days or even weeks to complete. Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors. Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like ADD A to B, and store into C are not practically efficient. Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue. Hence, the concept of Instruction Pipeline comes into picture, in which the instruction passes through several sub-units in turn.

These sub-units perform various independent functions, for example: the first one decodes the instruction, the second sub-unit fetches the data and the thirdsub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line. Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time. A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers(from 0 to n memory location) to another group of numbers(lets say, n to k memory location). This can be achieved by vector processors. In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Applications of Vector Processors

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

1. Petroleum exploration.

2. Medical diagnosis.

3. Data analysis.

4. Weather forecasting.

5. Aerodynamics and space flight simulations.

6. Image processing.

A parallel processing system is able to perform concurrent data processing to achieve faster execution time

• The system may have two or more ALUs and be able to execute two or more instructions at the same time

• Also, the system may have two or more processors operating concurrently

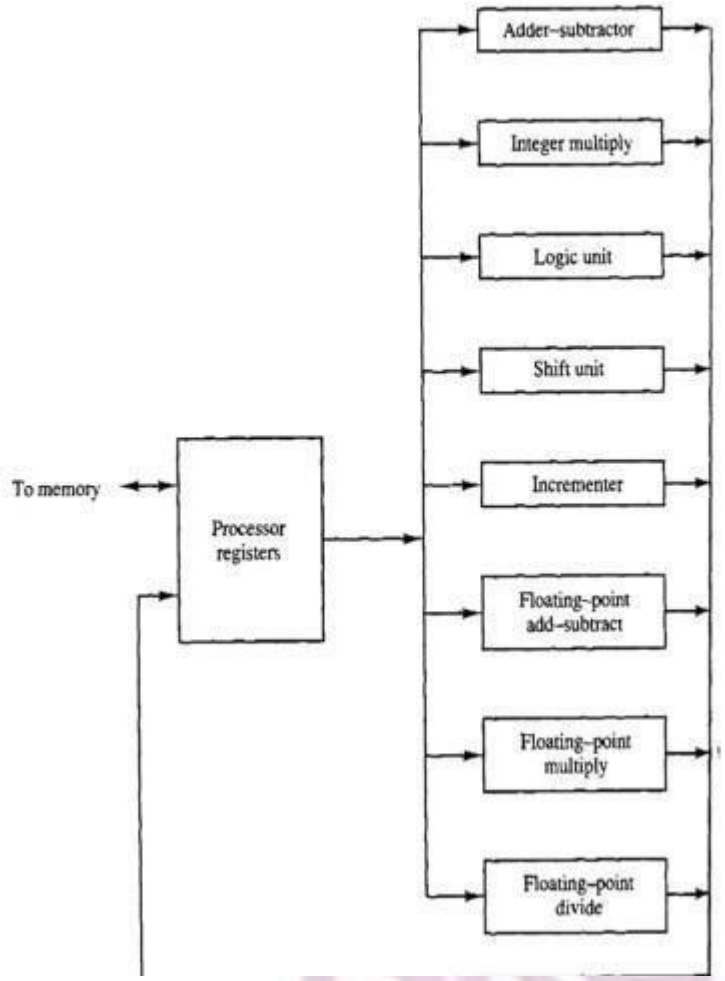
• Goal is to increase the throughput – the amount of processing that can be accomplished during a given interval of time

• Parallel processing increases the amount of hardware required

• Example: the ALU can be separated into three units and the operands diverted to each unit under the supervision of a control unit

• All units are independent of each other • 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.



• Parallel processing can be classified from:

oThe internal organization of the processors

o The interconnection structure between processors

o The flow of information through the system

o The number of instructions and data items that are manipulated simultaneously

• The sequence of instructions read from memory is the instruction stream

• The operations performed on the data in the processor is the data stream

• Parallel processing may occur in the instruction stream, the data stream, or both Computer classification:

o Single instruction stream, single data stream – SISD

o Single instruction stream, multiple data stream – SIMD

o Multiple instruction stream, single data stream – MISD

o Multiple instruction stream, multiple data stream – MIMD

• SISD – Instructions are executed sequentially. Parallel processing may be achieved by means of multiple functional units or by pipeline processing

• SIMD – Includes multiple processing units with a single control unit. All processors receive the same instruction, but operate on different data.

• MIMD – A computer system capable of processing several programs at the same time.

• We will consider parallel processing under the following main topics:

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

• 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

• Can imagine that each segment consists of an input register followed by an combinational circuit

• A clock is applied to all registers after enough time has elapsed to perform all segment activity

• The information flows through the pipeline one step at a time

• Example: Ai * Bi + Ci

for i = 1, 2, 3, …, 7

The suboperations performed in each segment are:

R1 ← Ai

, R2 ← Bi

R3 ← R1 * R2, R4 ← Ci

R5 ← R3 + R4

Figure 9-2 Example of pipeline processing.

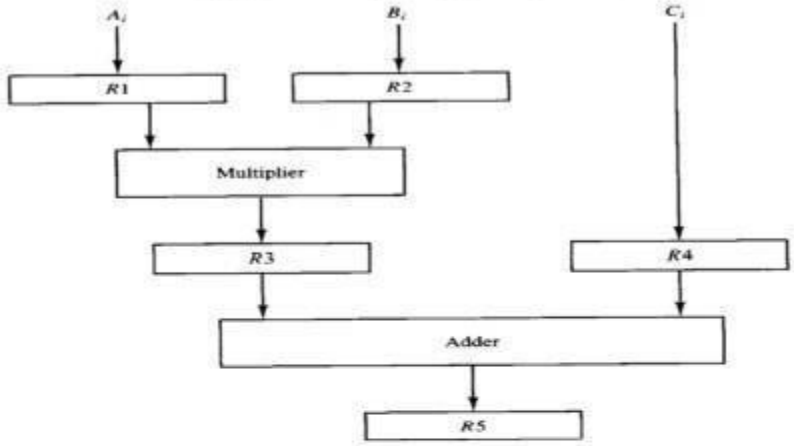


TABLE 9-1 Content of Registers in Pipeline Example

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	—	—	—	—	$A_7 * B_7 + C_7$

Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor

• The technique is efficient for those applications that need to repeat the same task many time with different sets of data

• A task is the total operation performed going through all segments of a pipeline

• The behavior of a pipeline can be illustrated with a space-time diagram

• This shows the segment utilization as a function of time

• Once the pipeline is full, it takes only one clock period to obtain an output

Figure 9-4 Space-time diagram for pipeline.

Segment:	1	2	3	4	5	6	7	8	9	→ Clock cycles
1	T_1	T_2	T_3	T_4	T_5	T_6				
2		T_1	T_2	T_3	T_4	T_5	T_6			
3			T_1	T_2	T_3	T_4	T_5	T_6		
4				T_1	T_2	T_3	T_4	T_5	T_6	

Figure 9-2 Example of pipeline processing.

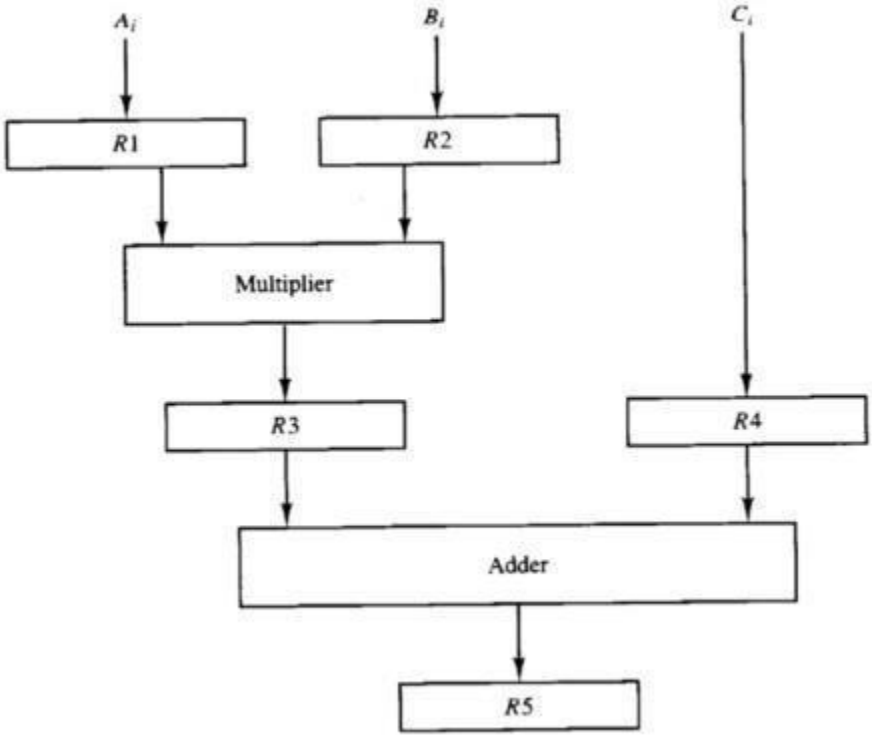


TABLE 9-1 Content of Registers in Pipeline Example

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	—	—	—	—	$A_7 * B_7 + C_7$

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

• Example for floating-point addition and subtraction

• Inputs are two normalized floating-point binary numbers

X = A x 2a

Y = B x 2b

• A and B are two fractions that represent the mantissas

• a and b are the exponents

Instruction Pipeline

• 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

• If a branch out of sequence occurs, the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction mustbe discarded

• Consider a computer with an instruction fetch unit and an instruction execution unit forming a two segment pipeline

• A FIFO buffer can be used for the fetch segment

• Thus, an instruction stream can be placed in a queue, waiting for decoding and processing by the execution segment

• This reduces the average access time to memory for reading instructions

• Whenever there is space in the buffer, the control unit initiates the next instruction fetch phase

• The following steps are needed to process each instruction:

o Fetch the instruction from memory

o Decode the instruction

o Calculate the effective address

o Fetch the operands from memory

o Execute the instruction

o Store the result in the proper place

Up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time

• It is assumed that the processor has separate instruction and data memories

• Reasons for the pipeline to deviate from its normal operation are:

o Resource conflicts caused by access to memory by two segments at the same time.

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

Types of Multiprocessors

There are mainly two types of multiprocessors i.e. symmetric and asymmetric multiprocessors. Details about them are as follows −

Symmetric Multiprocessors

In these types of systems, each processor contains a similar copy of the operating system and they all communicate with each other. All the processors are in a peer to peer relationship i.e. no master - slave relationship exists between them.

An example of the symmetric multiprocessing system is the Encore version of Unix for the Multimax Computer.

Asymmetric Multiprocessors

In asymmetric systems, each processor is given a predefined task. There is a master processor that gives instruction to all the other processors. Asymmetric multiprocessor system contains a master slave relationship.

Asymmetric multiprocessor was the only type of multiprocessor available before symmetric multiprocessors were created. Now also, this is the cheaper option.

Advantages of Multiprocessor Systems

There are multiple advantages to multiprocessor systems. Some of these are −

More reliable Systems

In a multiprocessor system, even if one processor fails, the system will not halt. This ability to continue working despite hardware failure is known as graceful degradation. For example: If there are 5 processors in a multiprocessor system and one of them fails, then also 4 processors are still working. So the system only becomes slower and does not ground to a halt.

Enhanced Throughput

If multiple processors are working in tandem, then the throughput of the system increases i.e. number of processes getting executed per unit of time increase. If there are N processors then the throughput increases by an amount just under N.

More Economic Systems

Multiprocessor systems are cheaper than single processor systems in the long run because they share the data storage, peripheral devices, power supplies etc. If there are multiple processes that share data, it is better to schedule them on multiprocessor systems with shared data than have different computer systems with multiple copies of the data.

Disadvantages of Multiprocessor Systems

There are some disadvantages as well to multiprocessor systems. Some of these are:

Increased Expense

Even though multiprocessor systems are cheaper in the long run than using multiple computer systems, still they are quite expensive. It is much cheaper to buy a simple single processor system than a multiprocessor system.

Complicated Operating System Required

There are multiple processors in a multiprocessor system that share peripherals, memory etc

Characteristics of Multiprocessor

There are the major characteristics of multiprocessors are as follows −

- **Parallel Computing** − This involves the simultaneous application of multiple processors. These processors are developed using a single architecture to execute a common task. In general, processors are identical and they work together in such a way that the users are under the impression that they are the only users of the system. In reality, however, many users are accessing the system at a given time.
- **Distributed Computing** − This involves the usage of a network of processors. Each processor in this network can be considered as a computer in its own right and have the capability to solve a problem. These processors are heterogeneous, and generally, one task is allocated to a single processor.
- **Supercomputing** − This involves the usage of the fastest machines to resolve big and computationally complex problems. In the past, supercomputing machines were vector computers but at present, vector or parallel computing is accepted by most people.
- **Pipelining** − This is a method wherein a specific task is divided into several subtasks that must be performed in a sequence. The functional units help in performing each subtask. The units are attached serially and all the units work simultaneously.
- **Vector Computing** − It involves the usage of vector processors, wherein operations such as 'multiplication' are divided into many steps and are then applied to a stream of operands ("vectors").
- **Systolic** − This is similar to pipelining, but units are not arranged in a linear order. The steps in systolic are normally small and more in number and performed in a lockstep manner. This is more frequently applied in special-purpose hardware such as image or signal processors.

Interconnection structures :

The processors must be able to share a set of main memory modules & I/O devices in a multiprocessor system. This sharing capability can be provided through interconnection structures. The interconnection structure that are commonly used can be given as follows –

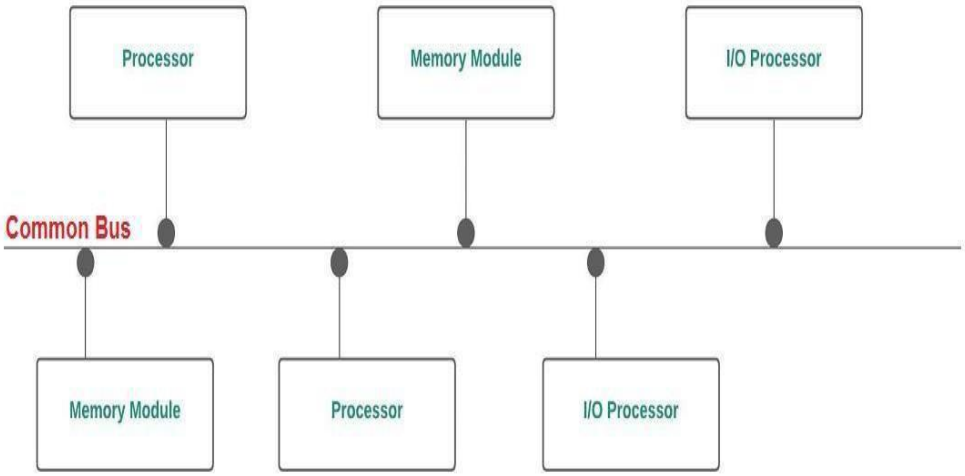
1. Time-shared / Common Bus
2. Cross bar Switch
3. Multiport Memory
4. Multistage Switching Network (Covered in 2nd part)
5. Hypercube System

In this article, we will cover Time shared / Common Bus in detail.

1. Time-shared / Common Bus (Interconnection structure in Multiprocessor System) :

In a multiprocessor system, the time shared bus interconnection provides a common

communication path connecting all the functional units like processor, I/O processor, memory unit etc. The figure below shows the multiple processors with common communication path (single bus).



Single-Bus Multiprocessor Organization

To communicate with any functional unit, processor needs the bus to transfer the data. To do so, the processor first need to see that whether the bus is available / not by checking the status of the bus. If the bus is used by some other functional unit, the status is busy, else free.

A processor can use bus only when the bus is free. The sender processor puts the address of the destination on the bus & the destination unit identifies it. In order to communicate with any functional unit, a command is issued to tell that unit, what work is to be done. The other processors at that time will be either busy in internal operations or will sit free, waiting to get bus. We can use a bus controller to resolve conflicts, if any. (Bus controller can set priority of different functional units)

This Single-Bus Multiprocessor Organization is easiest to reconfigure & is simple. This interconnection structure contains only passive elements. The bus interfaces of sender & receiver units controls the transfer operation here.

To decide the access to common bus without conflicts, methods such as static & fixed priorities, First-In-Out (FIFO) queues & daisy chains can be used.

Advantages –

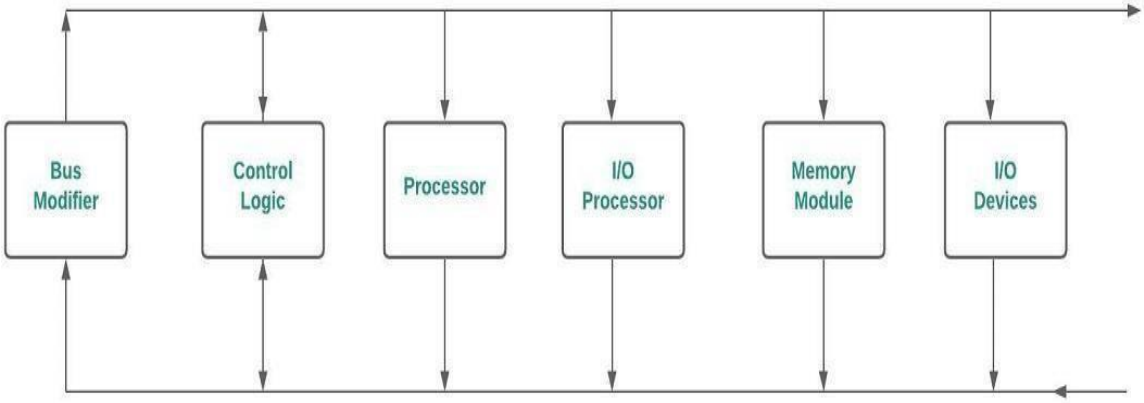
- Inexpensive as no extra hardware is required such as switch.
- Simple & easy to configure as the functional units are directly connected to the bus .

Disadvantages –

- Major fight with this kind of configuration is that if malfunctioning occurs in any of the bus interface circuits, complete system will fail.
- **Decreased throughput** —
  At a time, only one processor can communicate with any other functional unit.

- **Increased arbitration logic** —

As the number of processors & memory unit increases, the bus contention problem increases. To solve the above disadvantages, we can use two uni-directional buses as :

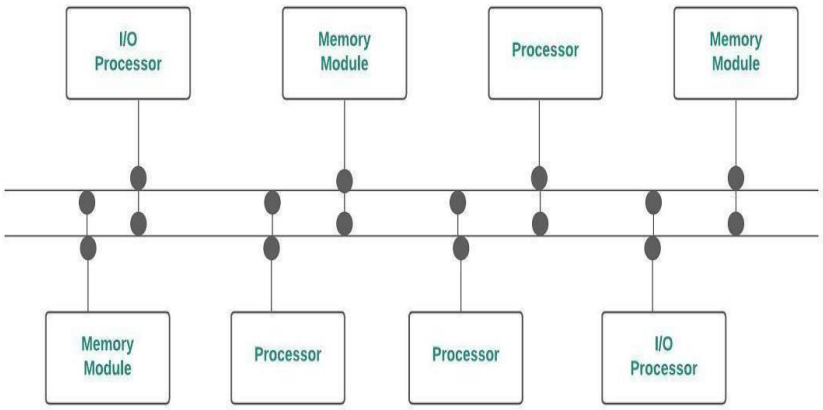


Multiprocessor System with unidirectional buses

Both the buses are required in a single transfer operation. Here, the system complexity is increased & the reliability is decreased, The solution is to use multiple bi-directional buses.

Multiple bi-directional buses :

The multiple bi-directional buses means that in the system there are multiple buses that are bi-directional. It permits simultaneous transfers as many as buses are available. But here also the complexity of the system is increased.



Multiple Bi-Directional Multiprocessor System

Apart from the organization, there are many factors affecting the performance of bus. They are –

- Number of active devices on the bus.
- Data width
- Error Detection method
- Synchronization of data transfer etc.

Advantages of Multiple bi-directional buses –

- Lowest cost for hardware as no extra device is needed such as switch.
- Modifying the hardware system configuration is easy.
- Less complex when compared to other interconnection schemes as there are only 2 buses & all the components are connected via that buses.

Disadvantages of Multiple bi-directional buses –

- System Expansion will degrade the performance because as the number of functional unit increases, more communication is required but at a time only 1 transfer can happen via 1 bus.
- Overall system capacity limits the transfer rate & If bus fails, whole system will fail.
- Suitable for small systems only.

2. Crossbar Switch :

A point is reached at which there is a separate path available for each memory module, if the number of buses in common bus system is increased. Crossbar Switch (for multiprocessors) provides separate path fro each module.

3. Multiport Memory :

In Multiport Memory system, the control, switching & priority arbitration logic are distributed throughout the crossbar switch matrix which is distributed at the interfaces to the memory modules.

4. Hypercube Interconnection :

This is a binary n-cube architecture. Here we can connect 2n processors and each of the processor here forms a node of the cube. A node can be memory module, I/O interface also, not necessarily processor. The processor at a node has communication path that is direct goes to n other nodes (total 2n nodes). There are total 2n distinct n-bit binary addresses.

Conclusion :

Interconnection structure can decide overall system's performance in a multi processor environment. Although using common bus system is much easy & simple, but the availability of only 1 path is its major drawback & if the bus fails, whole system fails. To overcome this & improve overall performance, crossbar, multi port, hypercube & then multistage switch network evolved.

Computer systems contain a number of buses at various levels to facilitate the transfer of information between components. The CPU contains a number of internal buses for transferring information between processor registers and ALU.

Inter Processor Arbitration

The processor, main memory and I/O devices can be interconnected by means of a common bus. A bus is set of lines (wires) defined to transfer all bits of a word from a specified source to a specified destination. Thus, bus provides a communication path for the transfer of data.

The bus includes data lines, address lines and control lines. Such a bus known as system bus. Different types of arbitration: ***Serial (Daisy Chain) arbitration, Parallel arbitration, Dynamic arbitration***

Serial (Daisy Chain) arbitration

In this type of arbitration, processors can access bus based on priority. In serial arbitration, bus access priority resolving based on the serial connection of the processors. This technique is obtained from daisy chain (serial) connection of processors. The serial priority resolving technique is obtained from daisy-chain connection similar to the daisy chain priority interrupt logic. The processors connected to the system bus are assigned priority according to their position along the priority control line.

When multiple devices concurrently request the use of the bus, the device with the highest priority is granted access to it. Each processor has its own bus arbiter logic with priority-in and priority-out lines. The priority out (PO) of each arbiter is connected to the priority in (PI) of the next-lower-priority arbiter. The PI of the highest-priority unit is maintained at a logic value 1. The highest-priority unit in the system will always receive access to the system bus when it requests it. The processor whose arbiter has a PI = 1 and PO = 0. That processor accesses the system bus.

Advantages

Simple and cheaper method

Least number of lines.

Disadvantages

Higher delay

Priority of the processor is fixed

Not reliable

Inter Process Communication (IPC)

A process can be of two types:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently, in reality, there are many situations when co-operative nature can be utilized for increasing computational speed, convenience, and modularity. Inter-process communication (IPC) is a mechanism that allows processes to

communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

1. Shared Memory
2. Message passing

Figure 1 below shows a basic structure of communication between processes via the shared memory method and via the message passing method.

An operating system can implement both methods of communication. First, we will discuss the shared memory methods of communication and then message passing. Communication between processes using shared memory requires processes to share some variable, and it completely depends on how the programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously, and they share some resources or use some information from another process. Process1 generates information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.
Let's discuss an example of communication between processes using the shared memory method.

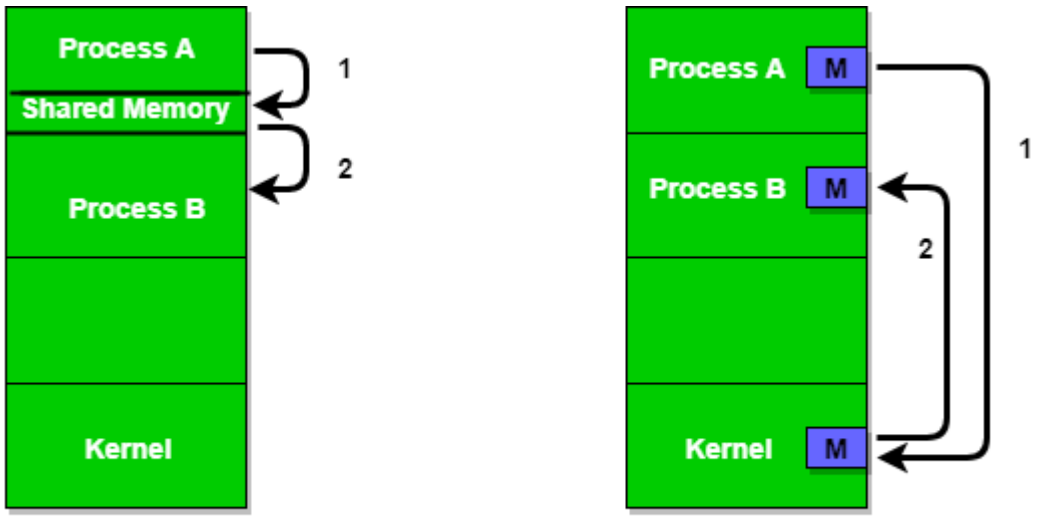


Figure 1 - Shared Memory and Message Passing

i) Shared Memory Method

Ex: Producer-Consumer problem

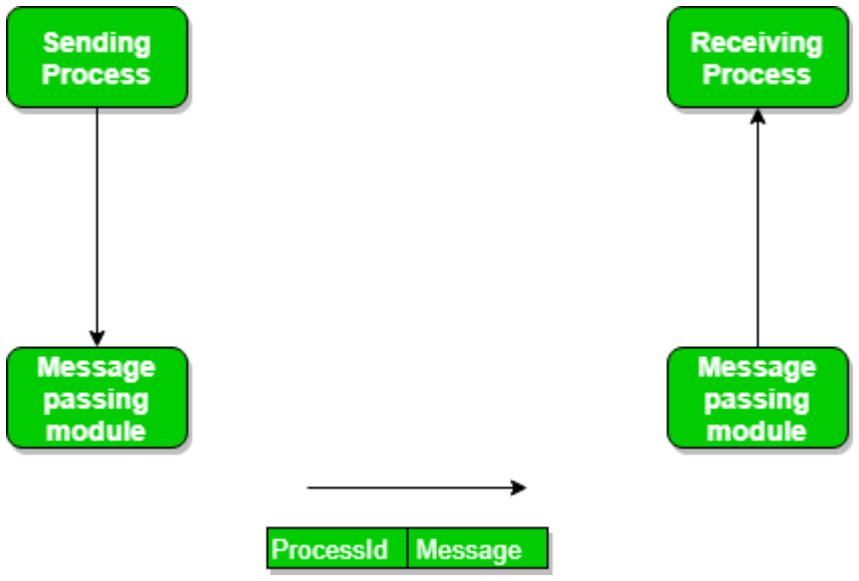
There are two processes: Producer and Consumer. The producer produces some items and the Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by the Producer is stored and from which the Consumer consumes the item if needed. There are two versions of this problem: the first one is known as the unbounded buffer problem in which the Producer can keep on producing items and there is no limit on the size of the buffer, the second one is known as the bounded buffer problem in which the Producer can produce up to a certain number of items before it starts waiting for Consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, then the producer will start producing items. If the total produced item is equal to the size of the buffer, the producer will wait to get it consumed by the Consumer. Similarly, the consumer will first check for the availability of the item. If no item is available, the Consumer will wait for the Producer to produce it. If there are items available, Consumer will consume them. The pseudo-code to demonstrate is provided below:

Shared Data between the two Processes

ii) Messaging Passing Method

Now, We will start our discussion of the communication between processes via message passing. In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destination) or **send**(message)
  – **receive**(message, host) or **receive**(message)



The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: **header and body.**

The **header part** is used for storing message type, destination id, source id, message length, and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

Message Passing through Communication Link.

Direct and Indirect Communication link

Now, We will start our discussion about the methods of implementing communication links. While implementing the link, there are some questions that need to be kept in mind like :

1. How are links established?
2. Can a link be associated with more than two processes?
3. How many links can there be between every pair of communicating processes?
4. What is the capacity of a link? Is the size of a message that the link can accommodate fixed or variable?
5. Is a link unidirectional or bi-directional?

A link has some capacity that determines the number of messages that can reside in it temporarily for which every link has a queue associated with it which can be of zero capacity, bounded capacity, or unbounded capacity. In zero capacity, the sender waits until the receiver informs the sender that it has received the message. In non-zero capacity cases, a process does not know whether a message has been received or not after the send operation. For this, the sender must communicate with the receiver explicitly. Implementation of the link depends on the situation, it can be either a direct communication link or an in-directed communication link.

Direct Communication links are implemented when the processes use a specific process identifier for the communication, but it is hard to identify the sender ahead of time.

For example the print server.

In-direct Communication is done via a shared mailbox (port), which consists of a queue of messages. The sender keeps the message in mailbox and the receiver picks them up.

Message Passing through Exchanging the Messages.

Synchronous and Asynchronous Message Passing:

A process that is blocked is one that is waiting for some event, such as a resource becoming available or the completion of an I/O operation. IPC is possible between the processes on same computer as well as on the processes running on different computer i.e. in networked/distributed system. In both cases, the process may or may not be blocked while sending a message or attempting to receive a message so message passing may be blocking or non-blocking. Blocking is considered **synchronous** and **blocking send** means the sender will be blocked until the message is received by receiver. Similarly, **blocking receive** has the receiver block until a message is available. Non-blocking is considered **asynchronous** and Non-blocking send has the sender sends the message and continue. Similarly, Non-blocking receive has the receiver receive a valid message or null. After a careful analysis, we can come to a conclusion that for a sender it is more

natural to be non-blocking after message passing as there may be a need to send the message to different processes. However, the sender expects acknowledgment from the receiver in case the send fails. Similarly, it is more natural for a receiver to be blocking after issuing the receive as the information from the received message may be used for further execution. At the same time, if the message send keep on failing, the receiver will have to wait indefinitely. That is why we also consider the other possibility of message passing. There are basically three preferred combinations:

- Blocking send and blocking receive
- Non-blocking send and Non-blocking receive
- Non-blocking send and Blocking receive (Mostly used)

In Direct message passing, The process which wants to communicate must explicitly name the recipient or sender of the communication.
e.g. **send(p1, message)** means send the message to p1.
Similarly, **receive(p2, message)** means to receive the message from p2.
In this method of communication, the communication link gets established automatically, which can be either unidirectional or bidirectional, but one link can be used between one pair of the sender and receiver and one pair of sender and receiver should not possess more than one pair of links. Symmetry and asymmetry between sending and receiving can also be implemented i.e. either both processes will name each other for sending and receiving the messages or only the sender will name the receiver for sending the message and there is no need for the receiver for naming the sender for receiving the message. The problem with this method of communication is that if the name of one process changes, this method will not work.
In Indirect message passing, processes use mailboxes (also referred to as ports) for sending and receiving messages. Each mailbox has a unique id and processes can communicate only if they share a mailbox. Link established only if processes share a common mailbox and a single link can be associated with many processes. Each pair of processes can share several communication links and these links may be unidirectional or bi-directional. Suppose two processes want to communicate through Indirect message passing, the required operations are: create a mailbox, use this mailbox for sending and receiving messages, then destroy the mailbox. The standard primitives used are: **send(A, message)** which means send the message to mailbox A. The primitive for the receiving the message also works in the same way e.g. **received (A, message)**. There is a problem with this mailbox implementation. Suppose there are more than two processes sharing the same mailbox and suppose the process p1 sends a message to the mailbox, which process will be the receiver? This can be solved by either enforcing that only two processes can share a single mailbox or enforcing that only one process is allowed to execute the receive at a given time or select any process randomly and notify the sender about the receiver. A mailbox can be made private to a single sender/receiver pair and can also be shared between multiple sender/receiver pairs. Port is an implementation of such mailbox that can have multiple senders and a single receiver. It is used in client/server applications (in this case the server is the receiver). The port is owned by the receiving process and created by OS on the request of the receiver process and can be destroyed either on request of the same receiver processor when the receiver terminates itself. Enforcing that only one process is allowed to execute the receive can be done using the concept of mutual exclusion. **Mutex mailbox** is created which is shared by n process. The sender is non-

blocking and sends the message. The first process which executes the receive will enter in the critical section and all other processes will be blocking and will wait.
Now, let's discuss the Producer-Consumer problem using the message passing concept. The producer places items (inside messages) in the mailbox and the consumer can consume an item when at least one message present in the mailbox. The code is given below:

Examples of IPC systems

1. Posix : uses shared memory method.
2. Mach : uses message passing
3. Windows XP : uses message passing using local procedural calls

Communication in client/server Architecture:

There are various mechanism:

- Pipe
- Socket
- Remote Procedural calls (RPCs)

Cache Coherence.

In computer architecture, **cache coherence** is the uniformity of shared resource data that ends up stored in multiple local caches. When clients in a system maintain caches of a common memory resource, problems may arise with incoherent data, which is particularly the case with CPUs in a multiprocessing system.

In the illustration on the right, consider both the clients have a cached copy of a particular memory block from a previous read. Suppose the client on the bottom updates/changes that memory block, the client on the top could be left with an invalid cache of memory without any notification of the change. Cache coherence is intended to manage such conflicts by maintaining a coherent view of the data values in multiple caches.