UNIT V

RISC Processors: -

- To execute an instruction, a number of steps are required. By the control unit of the processor, a number of control signals are generated for each step.
- To execute each instruction, if there is a separate electronic circuitry in the control unit, which produces all the necessary signals, this approach of the design of the control section of the processor is called **RISC** design. It is hardware approach.
- It is also called hard-wired approach.

Characteristic of RISC :

1. Simpler instruction, hence simple instruction decoding.
2. Instruction come under size of one word.
3. Instruction take single clock cycle to get executed.
4. More number of general purpose register.
5. Simple Addressing Modes.
6. Less Data types.
7. Pipeling can be achieved.

CISC Processors: -

- If the control unit contains a number of micro electronic circuitry to generate a set of control signals and each micro circuitry is activated by a microcode, this design approach is called CISC design.
- This is a software approach of designing a control unit of the processor.

Characteristic of CISC :

1. Complex instruction, hence complex instruction decoding.
2. Instruction are larger than one word size.
3. Instruction may take more than single clock cycle to get executed.

4. Less number of general purpose register as operation get performed in memory itself.
5. Complex Addressing Modes.
6. More Data types.

Differences between RISC and CISC:

CISC	RISC
• A large number of instructions are present in the architecture.	• Very fewer instructions are present. The number of instructions are generally less than 100.
• Some instructions with long execution times.	• No instruction with a long execution time due to very simple instruction set.
• Variable-length encodings of the instructions.	• Fixed-length encodings of the instructions are used.
• Multiple formats are supported for specifying operands. A memory operand specifier can have many different combinations of displacement, base and index registers.	• Simple addressing formats are supported. Only base and displacement addressing is allowed.
• CISC supports array.	• RISC does not supports array.
• Arithmetic and logical operations can be applied to both memory and register operands.	• Arithmetic and logical operations only use register operands. Memory referencing is only allowed by load and store instructions,
• Implementation programs are hidden from machine level programs.	• Implementation programs exposed to machine level programs.
• Transistors are used for more registers	• Transistors are used for storing complex Instructions
• A instruction execute in single clock cycle	• Instruction take more than one clock cycle
• Code size is large	• Code size is small

Parallel 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

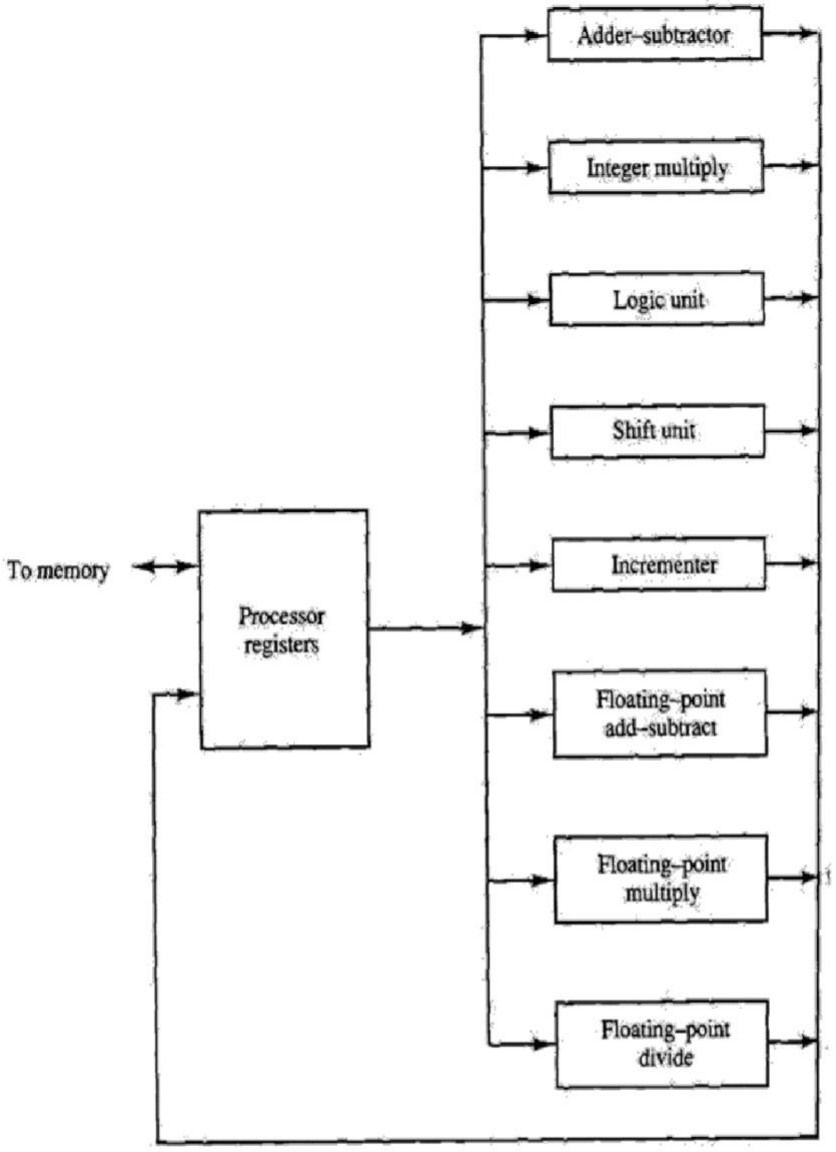
- Goal is to increase the *throughput* – the amount of processing that can be accomplished during a given interval of time
- Parallel processing increases the amount of hardware required

- 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:

  - The internal organization of the processors
  - The interconnection structure between processors o The flow of information through the system

  - 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:

- Single instruction stream, single data stream – SISD
- Single instruction stream, multiple data stream – SIMD
- Multiple instruction stream, single data stream – MISD
- 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: $A_i * B_i + C_i$ for i = 1, 2, 3, …, 7

- The suboperations performed in each segment are:

$R1 \leftarrow A_i$, $R2 \leftarrow B_i$

$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$

$R5 \leftarrow R3 + R4$

Figure 9-2 Example of pipeline processing.

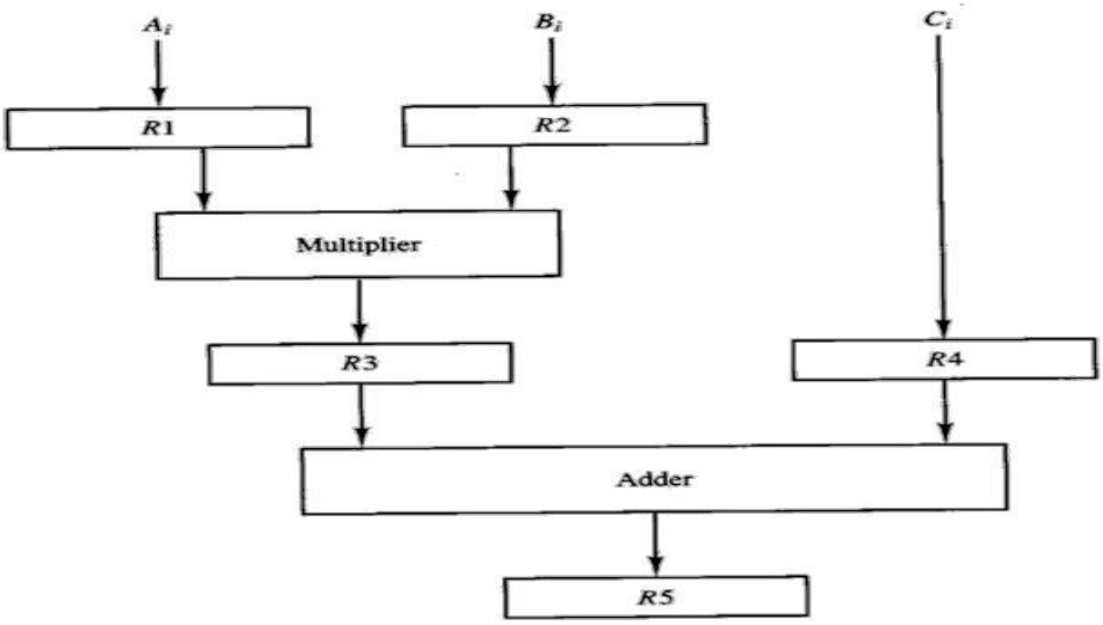


TABLE 9-1 Content of Registers in Pipeline Example

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.

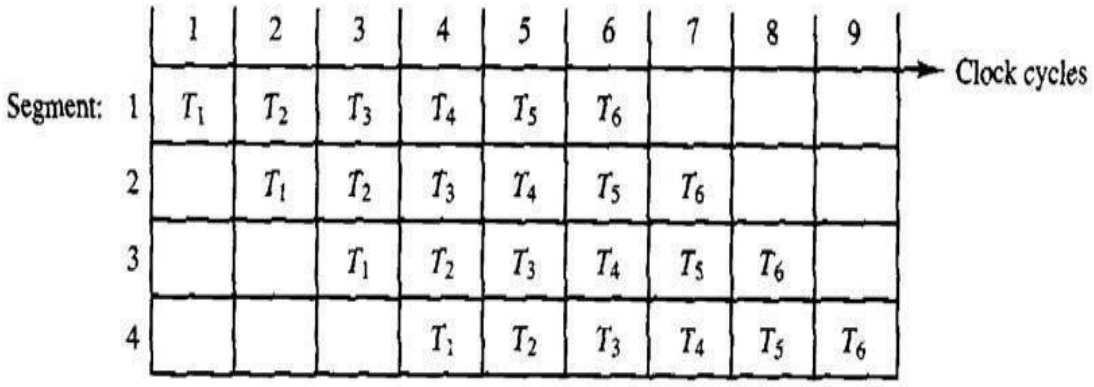


Figure 9-2 Example of pipeline processing.

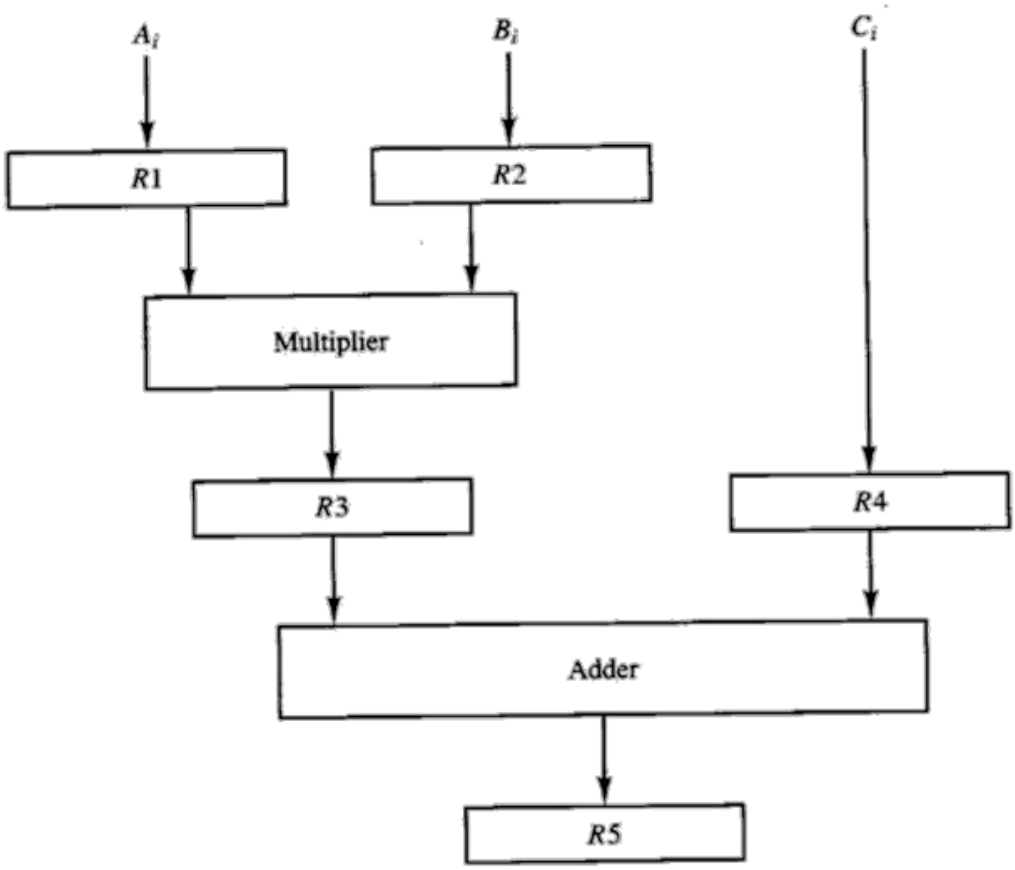


TABLE 9-1 Content of Registers in Pipeline Example

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

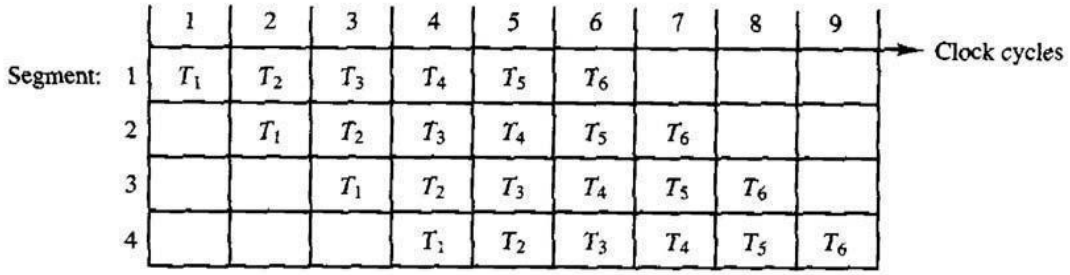
- The technique is efficient for those applications that need to repeat the same task many time with different sets of data
- A task is the total operation performed going through all segments of a pipeline

- 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.



- Consider a *k*-segment pipeline with a clock cycle time t_p to execute *n* tasks

- The first task T_1 requires time kt_p to complete
- The remaining $n - 1$ tasks finish at the rate of one task per clock cycle and will be completed after time $(n - 1)t_p$

- The total time to complete the *n* tasks is $[k + n - 1]t_p$

- The example of Figure 9-4 requires [4 + 6 – 1] clock cycles to finish

- Consider a nonpipeline unit that performs the same operation and takes t_n time to complete each task

- The total time to complete *n* tasks would be nt_n

- The *speedup* of a pipeline processing over an equivalent nonpipeline processing is defined by the ratio

$$S = \frac{nt_n}{(k + n - 1)t_p}$$

- As the number of tasks increase, the speedup becomes $S = \frac{t_n}{t_p}$

- If we assume that the time to process a task is the same in both circuits, $t_n = k\ t_p$ $S = \frac{kt_n}{tp} = k$
- Therefore, the theoretical maximum speed up that a pipeline can provide is *k*

- Example:
  - Cycle time = t_p = 20 ns
  - # of segments = k = 4
  - # of tasks = n = 100

The pipeline system will take $(k + n - 1)t_p = (4 + 100 - 1)20\text{ns} = 2060$ ns

Assuming that $t_n = kt_p = 4 * 20 = 80$ ns,

A nonpipeline system requires $nkt_p = 100 * 80 = 8000$ ns The speedup ratio = 8000/2060 = 3.88

- The pipeline cannot operate at its maximum theoretical rate
- One reason is that the clock cycle must be chosen to equal the time delay of the segment with the maximum propagation time
- Pipeline organization is applicable for arithmetic operations and fetching instructions

As the number of tasks increase, the speedup becomes S = $\underline{t_n}$

^{t}p

- Therefore, the theoretical maximum speed up that a pipeline can provide is *k*

- Example:

  p Cycle time = t_p = 20 ns o # of segments = *k* = 4
  p # of tasks = *n* = 100

The pipeline system will take $(k + n - 1)t_p = (4 + 100 - 1)20\text{ns} = 2060$ ns

Assuming that $t_n = kt_p = 4 * 20 = 80$ ns,

A nonpipeline system requires $nkt_p = 100 * 80 =$ 8000 ns The speedup ratio = 8000/2060 = 3.88

- The pipeline cannot operate at its maximum theoretical rate
- One reason is that the clock cycle must be chosen to equal the time delay of the segment with the maximum propagation time
- Pipeline organization is applicable for arithmetic operations and fetching instructions

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

a $X = A \times 2^a$

$Y = B \times 2^b$

- A and B are two fractions that represent the mantissas
- a and b are the exponents
- Four segments are used to perform the following:
  - Compare the exponents
  - Align the mantissas
  - Add or subtract the mantissas
  - Normalize the result

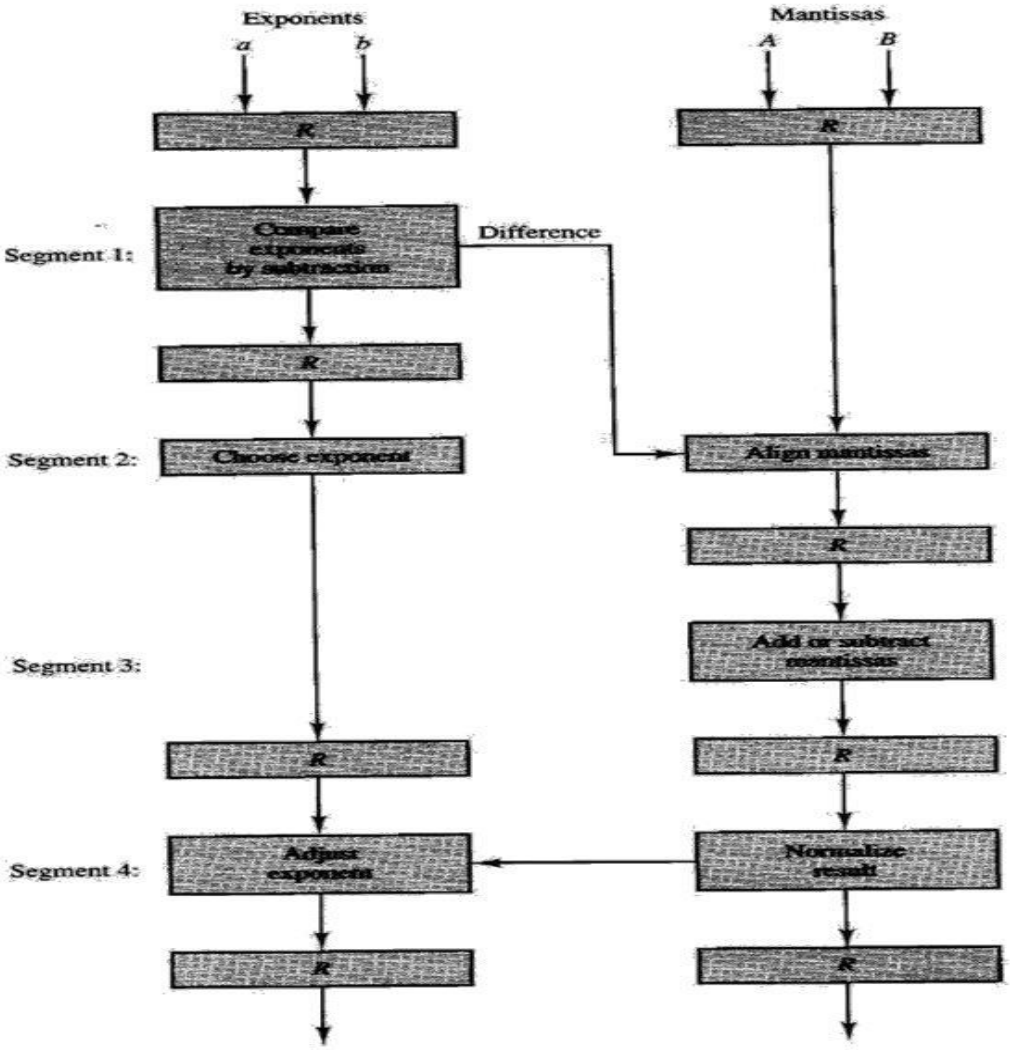


Figure 9-6 Pipeline for floating-point addition and subtraction.

- X = 0.9504 x 10^3 and Y = 0.8200 x 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
- Segment 2 shifts the mantissa of Y to the right to obtain Y = 0.0820 x 10^3
- The mantissas are now aligned
- Segment 3 produces the sum Z = 1.0324 x 10^3
- Segment 4 normalizes the result by shifting the mantissa once to the right and incrementing the exponent by one to obtain Z = 0.10324 x 10^4

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 must be 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:
    - Fetch the instruction from memory
    - Decode the instruction
    - Calculate the effective address
    - Fetch the operands from memory
    - Execute the instruction
- Store the result in the proper place

• The pipeline may not perform at its maximum rate due to:

- o Different segments taking different times to operate
- o Some segment being skipped for certain operations
- oMemory access conflicts

- Example: Four-segment instruction pipeline

- Assume that the decoding can be combined with calculating the EA in one segment

- Assume that most of the instructions store the result in a register so that the execution and storing of the result can be combined in one segment

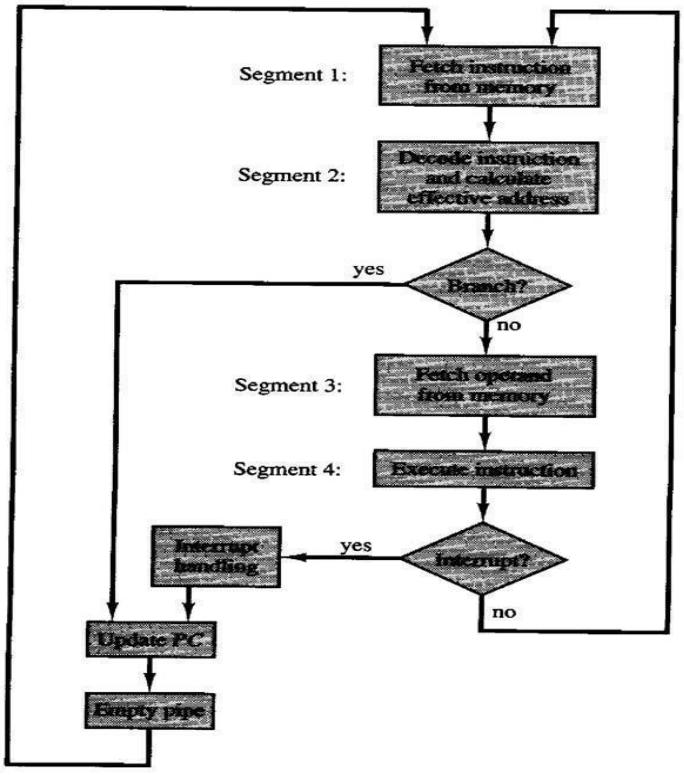


Figure 9-7 Four-segment CPU pipeline.

- 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:
  - *Resource conflicts* caused by access to memory by two segments at the same time.
  - *Data dependency* conflicts arise when an instruction depends on the result of a previous instruction, but his result is not yet available
- Assume that most of the instructions store the result in a register so that the execution and storing of the result can be combined in one segment

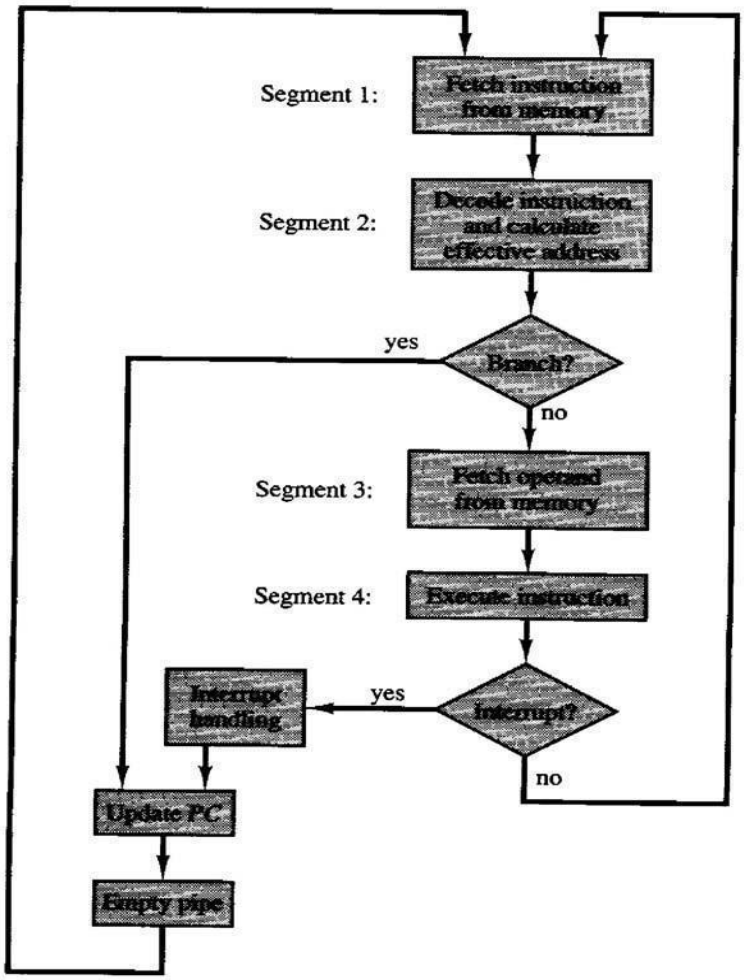


Figure 9-7 Four-segment CPU pipeline.

- 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:

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

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

Branch difficulties arise from program control instructions that may change the value of PC

- **Methods to handle data dependency:**

  - Hardware interlocks are circuits that detect instructions whose source operands are destinations of prior instructions. Detection causes the hardware to insert the required delays without altering the program sequence.

  - Operand forwarding uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments. This requires additional hardware paths through multiplexers as well as the circuit to detect the conflict.

  - Delayed load is a procedure that gives the responsibility for solving data conflicts to the compiler. The compiler is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operation instructions.

- **Methods to handle branch instructions:**

  - ***Prefetching the target instruction*** in addition to the next instruction allows either instruction to be available.

  - ***A branch target buffer*** is an associative memory included in the fetch segment of the branch instruction that stores the target instruction for a previously executed branch. It also stores the next few instructions after the branch target instruction. This way, the branch instructions that have occurred previously are readily available in the pipeline without interruption.

  - ***The loop buffer*** is a variation of the BTB. It is a small very high speed register file maintained by the instruction fetch segment of the pipeline. Stores all branches within a loop segment.

  - ***Branch prediction*** uses some additional logic to guess the outcome of a conditional branch instruction before it is executed. The pipeline then begins prefetching instructions from the predicted path.

Vector processing

The need to increase computational power is a never-ending requirement. In scientific and research areas, the computational involved are quite extensive and hence high power computers are the must.

The areas like structural engineering, petroleum exploration, aerodynamics, hydrodynamics, nuclear research, tomography, VLSI design, AI can have data in the form of matrices which suits vector processors to process it at a high speed.

Some examples of its are:

1. In radar and signal processing for detection of space / underwater targets.
2. In remote sensing for earth resources exploration.
3. In computational wind tunnel experiments.
4. In 3D stop-action computer assisted tomography.
5. Weather forecasting.
6. Medical diagnosis.

Characteristics of Vector processing

1. A vector is an ordered set of elements. A vector operand contains an ordered set of n elements, where n is called the length of the vector. Each element in a vector is a scalar quantity, which may be floating point number, an integer, a logical value, or a character (byte).
2. In **vector processing**, two successive pairs of elements are processed each clock period. In dual vector pipes and dual sets of vector functional units allow two pairs of elements to be processed during the same clock period. As each pair of operations is completed, the results are delivered to the appropriate elements of the result register. The operation continues until the number of elements processed is equal to the count specified by the vector length register.
   For example: C (1:50) = A (1:50) + B (1:50)
   This vector instruction includes the initial addresses of the two source operands, one destination operand, the length of the vectors and the operation to be performed.
3. Vector instructions are classified into for basic types:

F1: V = V f2: V = S

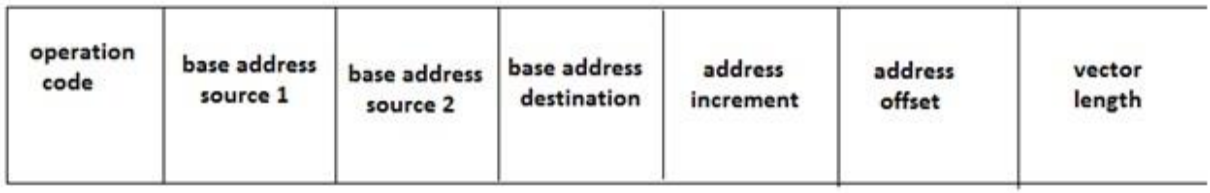
F3: V * V = V f4: V*S = V

Where V indicates vector operand and S indicates scalar operand. The operations f1 and f2 are unary operations such as vector square root, vector sine, vector complement, vector summation and so on. On the other hand, operations f3 and f4 are binary operations such as vector add, vector multiply, vector scalar adds and so on.

4. In **vector processing**, identical processes are repeatedly invoked many times, each of which can be subdivided into subprocesses.
5. In **vector processing**, successive operands are fed through the pipeline segments and require as few buffers and local controls as possible. This parallel vector processing allows the generation of more than two results per clock period. The parallel vector operations are automatically initiated either when successive vector instructions use different functional units and different vector registers, or when successive vector instructions use the result stream from one vector register as the operand of another operation using different functional units. This process is known as chaining.
6. Because of the startup delay in a pipeline, a vector processor performs better with longer vectors.
7. **Vector processing** is usually faster and more efficient than scalar processing because it reduces the overhead associated with maintenance of the loop control variables.

Vector Instruction Fields

Vector instructions are usually specified by the following fields:



1. **Opcode (operation code):**
   This field is used to select the functional unit or to reconfigure a multifunctional unit to perform the specified operation.
2. **Base addresses:**
   In case of memory reference instruction, this field specifies the base addresses needed for source operands and result vectors. If the operands and results are located in the vector register file, the designated vector registers must be specified.
3. **Address increment:**
   This field specifies the space between the two elements in the main memory. Usually, the elements are consecutively stored thus the increment is 1. However, with variable increment higher flexibility can be offered in the applications.
4. **Address offset:**
   This field specifies the offset to the base address. Using the base address and the offset, the effective memory address can be calculated. The offset can be either positive or negative.

5. **Vector length:**
   this field determines the termination of a vector instruction. Vector length affects the processing efficiency because the additional subdividing is required for long vectors.

Array Processor :

Array processors are also known as multiprocessors or vector processors. They perform computations on large arrays of data. Thus, they are used to improve the performance of the computer.

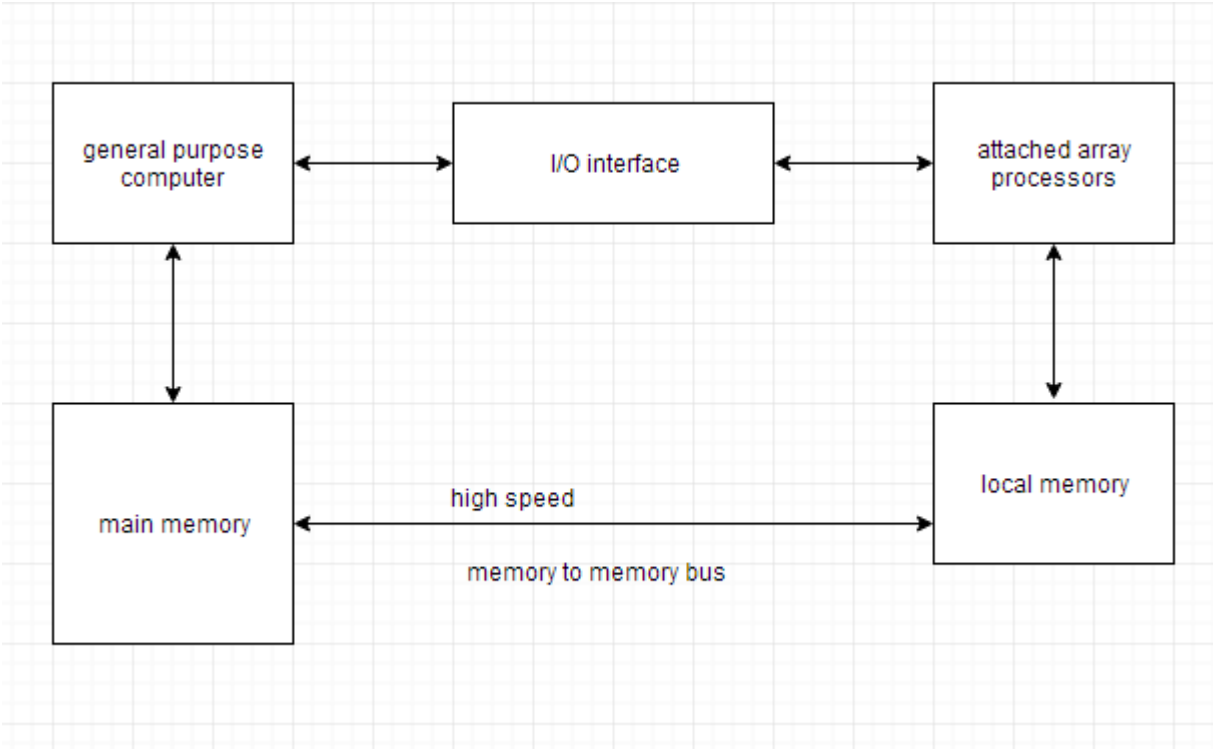
Types of Array Processors

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.



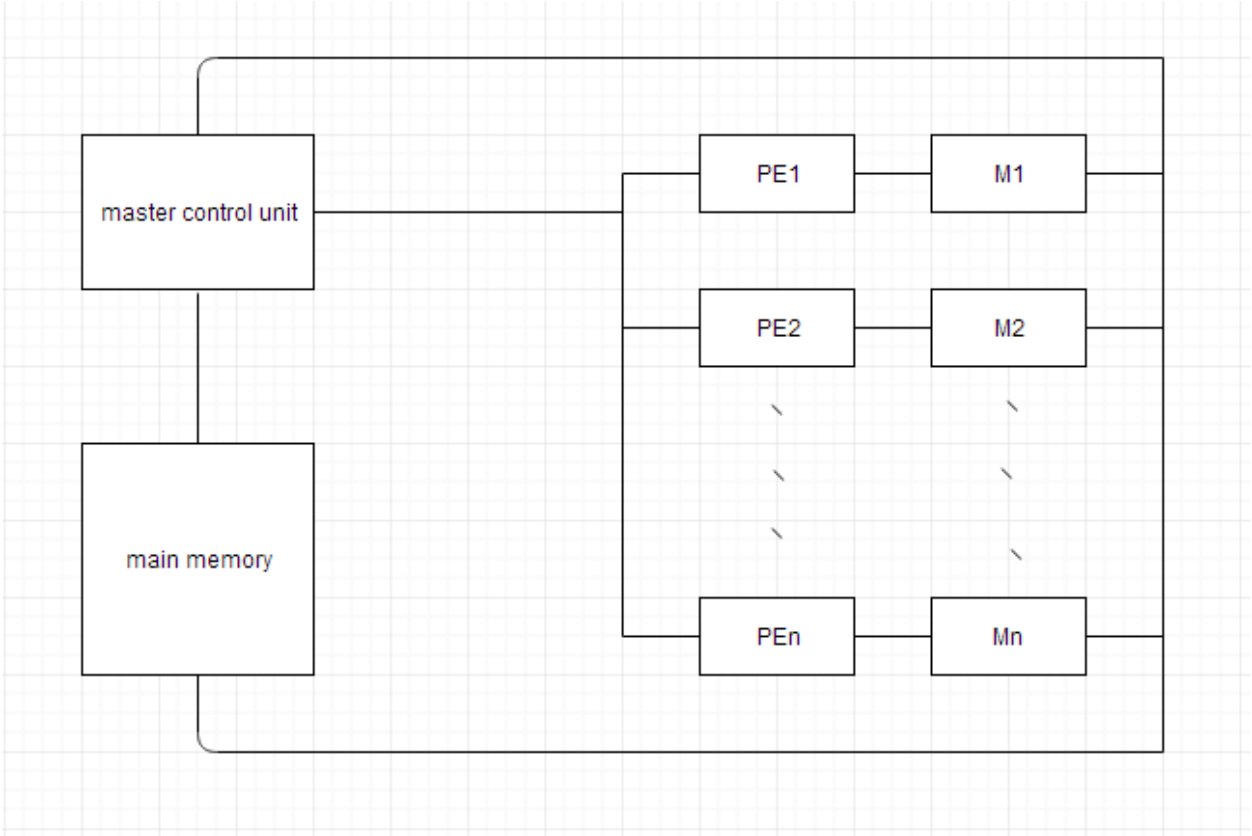
SIMD Array Processors

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an **ALU** and **registers**. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the **ILLIAC IV** computer developed by the **Burroughs corps**. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.



Multiprocessor Systems:

Most computer systems are single processor systems i.e they only have one processor. However, multiprocessor or parallel systems are increasing in importance nowadays. These systems have multiple processors working in parallel that share the computer clock, memory, bus, peripheral devices etc. An image demonstrating the multiprocessor architecture is:

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. So, it is much more complicated to schedule processes and impart resources to processes.than in single processor systems. Hence, a more complex and complicated operating system is required in multiprocessor systems.

Large Main Memory Required

All the processors in the multiprocessor system share the memory. So a much larger pool of memory is required as compared to single processor systems.

Characteristics of multiprocessors

1. A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment.
2. The term “processor” in multiprocessor can mean either a central processing unit (CPU) or an input-output processor (IOP).
3. Multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems

4. The similarity and distinction between multiprocessor and multicomputer are ``

- Similarity
- Both support concurrent operations
- Distinction
- The network consists of several autonomous computers that may or may not communicate with each other.
- A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.

5. Multiprocessing improves the reliability of the system.

6. The benefit derived from a multiprocessor organization is an improved system performance.

- Multiple independent jobs can be made to operate in parallel.
- A single job can be partitioned into multiple parallel tasks.

7.Multiprocessing can improve performance by decomposing a program into parallel executable tasks.

- The user can explicitly declare that certain tasks of the program be executed in parallel.
- This must be done prior to loading the program by specifying the parallel executable segments.
- The other is to provide a compiler with multiprocessor software that can automatically detect parallelism in a user's program.

8. Multiprocessor are classified by the way their memory is organized.

- A multiprocessor system with common shared memory is classified as a shared-memory or tightly coupled multiprocessor.
- Tolerate a higher degree of interaction between tasks.
- Each processor element with its own private local memory is classified as a distributed-memory or loosely coupled system.
- Are most efficient when the interaction between tasks is minimal

Interconnection Structures

1. The components that form a multiprocessor system are CPUs, IOPs connected to input-output devices, and a memory unit.
2. The interconnection between the components can have different physical configurations, depending on the number of transfer paths that are available

- Between the processors and memory in a shared memory system o Among the processing elements in a loosely coupled system

3. There are several physical forms available for establishing an interconnection network.

- Time-shared common bus o Multiport memory o Crossbar switch o Multistage switching network o Hypercube system Time Shared Common Bus

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

2. Disadv.:

- Only one processor can communicate with the memory or another processor at any given time. o As a consequence, the total overall transfer rate within the system is limited by the speed of the single path

3. A more economical implementation of a dual bus structure is depicted in Fig. below.
4. Part of the local memory may be designed as a cache memory attached to the CPU.

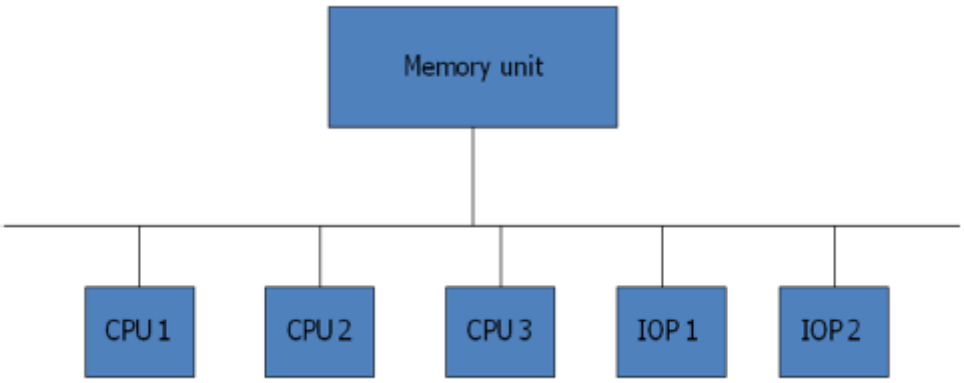
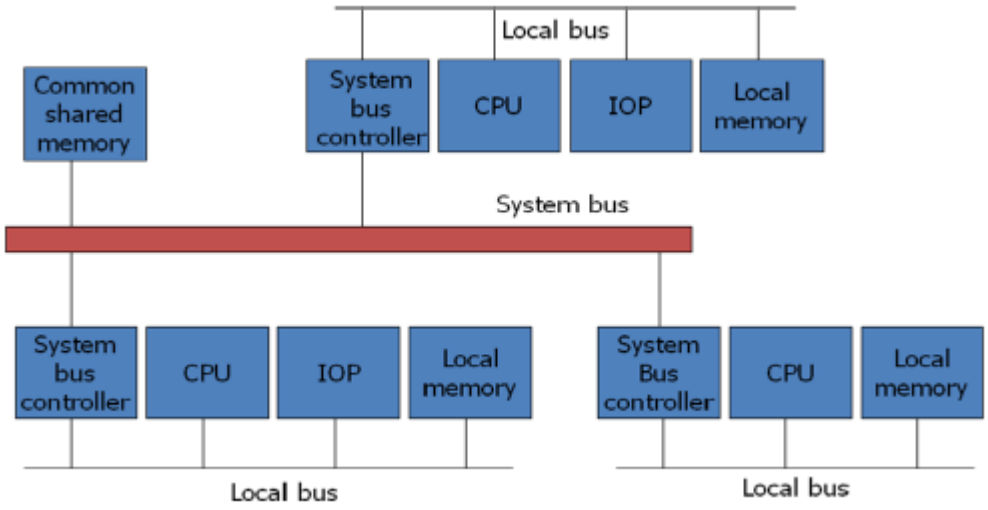


Fig: Time shared common bus organization



Multiport Memory

1. A multiport memory system employs separate buses between each memory module and each CPU.
2. The module must have internal control logic to determine which port will have access to memory at any given time.

3. Memory access conflicts are resolved by assigning fixed priorities to each memory port.

4. Adv.:

- The high transfer rate can be achieved because of the multiple paths.

5. Disadv.:

- It requires expensive memory control logic and a large number of cables and connections

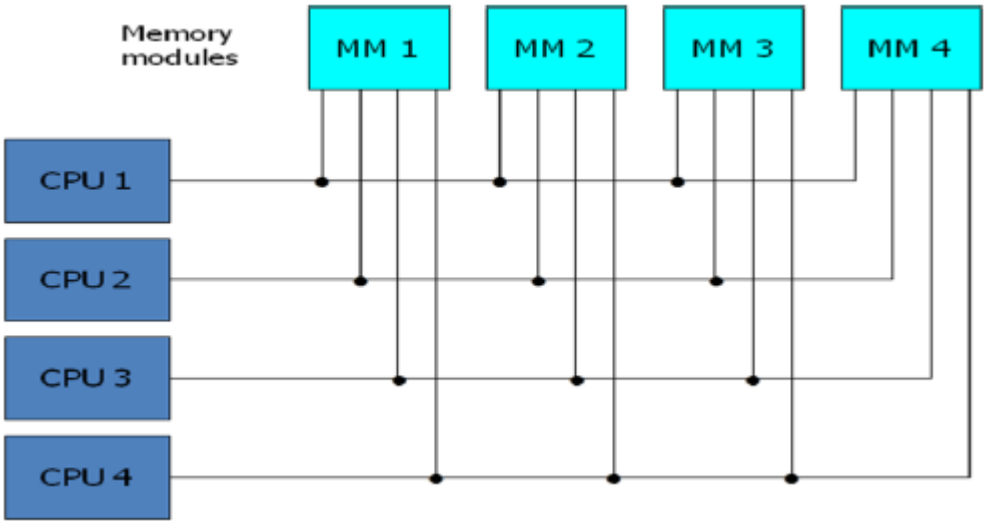


Fig: Multiport memory organization

Crossbar Switch

1. Consists of a number of crosspoints that are placed at intersections between processor buses and memory module paths.
2. The small square in each crosspoint is a switch that determines the path from a processor to a memory module.

3. Adv.:

- Supports simultaneous transfers from all memory modules

4. Disadv.:

- The hardware required to implement the switch can become quite large and complex.

4. Below fig. shows the functional design of a crossbar switch connected to one memory module.

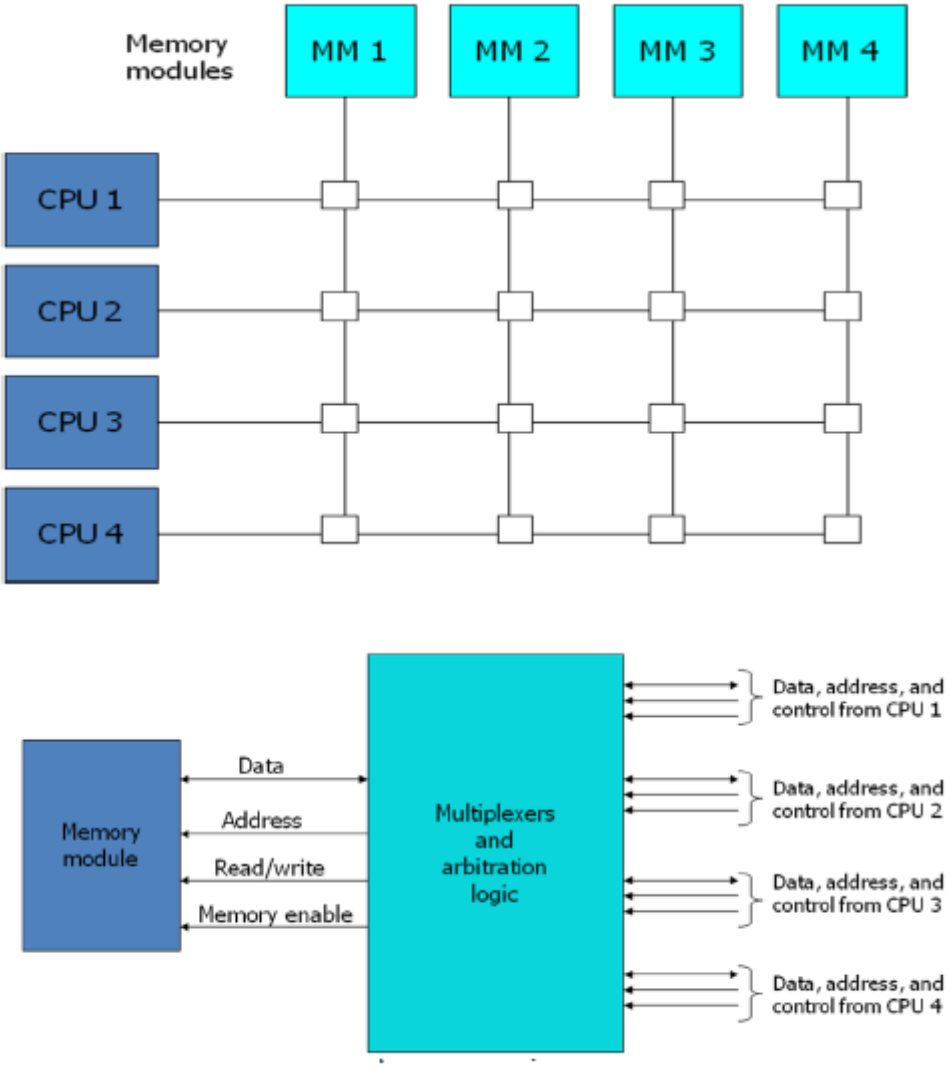
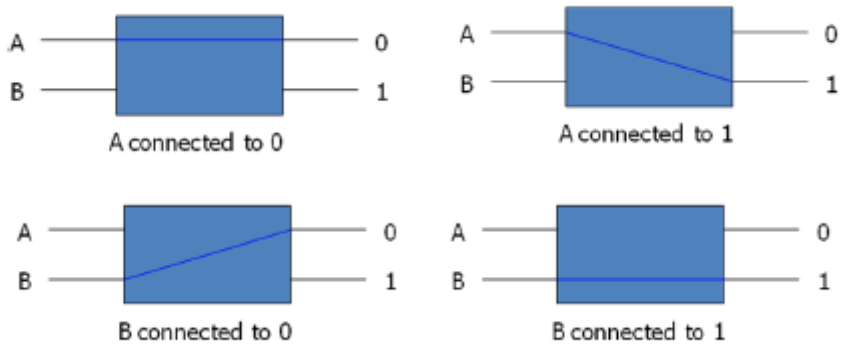


Fig: Block diagram of crossbar switch

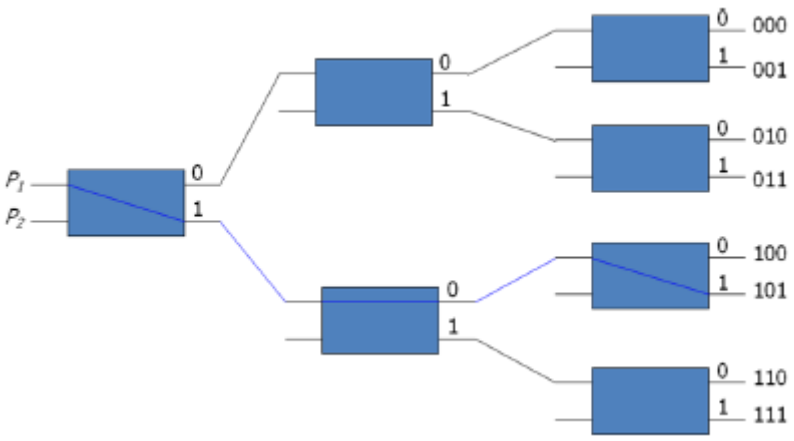
Multistage Switching Network

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



2. Using the 2x2 switch as a building block, it is possible to build a multistage network to control the communication between a number of sources and destinations.

- To see how this is done, consider the binary tree shown in Fig. below. o Certain request patterns cannot be satisfied simultaneously. i.e., if P1 :000~011, then P2 : 100~111



3. One such topology is the omega switching network shown in Fig. below

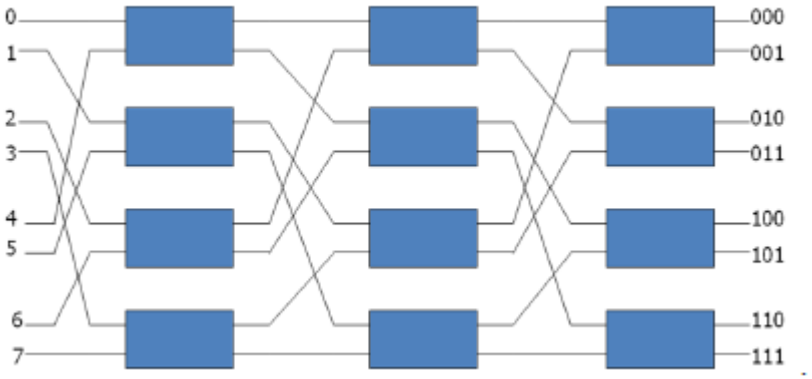


Fig: 8 x 8 Omega Switching Network

4. Some request patterns cannot be connected simultaneously. i.e., any two sources cannot be connected simultaneously to destination 000 and 001
5. In a tightly coupled multiprocessor system, the source is a processor and the destination is a memory module.
6. Set up the path transfer the address into memory transfer the data
7. In a loosely coupled multiprocessor system, both the source and destination are processing elements.

Hypercube System

1. The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of N=2n processors interconnected in an n-dimensional binary cube.
   - Each processor forms a node of the cube, in effect it contains not only a CPU but also local memory and I/O interface.
   - Each processor address differs from that of each of its n neighbors by exactly one bit position.

2. Fig. below shows the hypercube structure for n=1, 2, and 3.
3. Routing messages through an n-cube structure may take from one to n links from a source node to a destination node.
    - A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address.
    - The message is then sent along any one of the axes that the resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ.
4. A representative of the hypercube architecture is the Intel iPSC computer complex.
    - It consists of 128(n=7) microcomputers, each node consists of a CPU, a floating-point processor, local memory, and serial communication interface units.

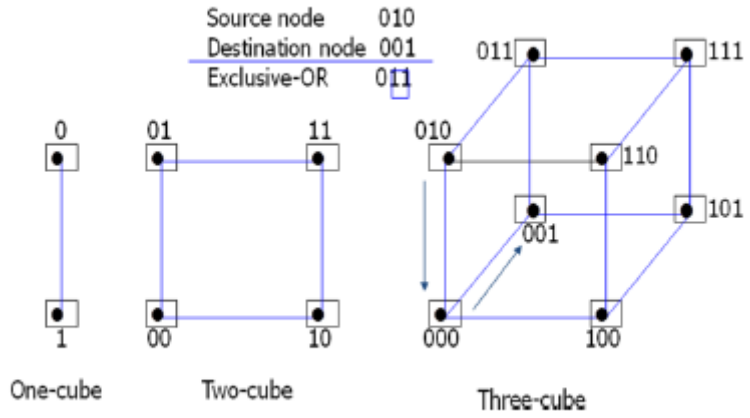


Fig: Hypercube structures for n=1,2,3

Inter processor Communication and Synchronization

1. The various processors in a multiprocessor system must be provided with a facility for communicating with each other.
    - A communication path can be established through a portion of memory or a common input-output channels.
    2. The sending processor structures a request, a message, or a procedure, and places it in the memory mailbox.
    - Status bits residing in common memory o The receiving processor can check the mailbox periodically. o The response time of this procedure can be time consuming.
    3. A more efficient procedure is for the sending processor to alert the receiving processor directly by means of an interrupt signal.
    4. In addition to shared memory, a multiprocessor system may have other shared resources. e.g., a magnetic disk storage unit.

5. To prevent conflicting use of shared resources by several processors there must be a provision for assigning resources to processors. i.e., operating system.

6. There are three organizations that have been used in the design of operating system for multiprocessors: master-slave configuration, separate operating system, and distributed operating system.

7. In a master-slave mode, one processor, master, always executes the operating system functions.

8. In the separate operating system organization, each processor can execute the operating system routines it needs. This organization is more suitable for loosely coupled systems.

9. In the distributed operating system organization, the operating system routines are distributed among the available processors. However, each particular operating system function is assigned to only one processor at a time. It is also referred to as a floating operating system.

Loosely Coupled System

1. There is no shared memory for passing information.
2. The communication between processors is by means of message passing through I/O channels.
3. The communication is initiated by one processor calling a procedure that resides in the memory of the processor with which it wishes to communicate.
4. The communication efficiency of the interprocessor network depends on the communication routing protocol, processor speed, data link speed, and the topology of the network.

Interprocess Synchronization

1. The instruction set of a multiprocessor contains basic instructions that are used to implement communication and synchronization between cooperating processes.
   - Communication refers to the exchange of data between different processes.
   - Synchronization refers to the special case where the data used to communicate between processors is control information.
2. Synchronization is needed to enforce the correct sequence of processes and to ensure mutually exclusive access to shared writable data.
3. Multiprocessor systems usually include various mechanisms to deal with the synchronization of resources.

- Low-level primitives are implemented directly by the hardware.
- These primitives are the basic mechanisms that enforce mutual exclusion for more complex mechanisms implemented in software.
- A number of hardware mechanisms for mutual exclusion have been developed.

4. A binary semaphore

Mutual Exclusion with Semaphore

1. A properly functioning multiprocessor system must provide a mechanism that will guarantee orderly access to shared memory and other shared resources.
   - Mutual exclusion: This is necessary to protect data from being changed simultaneously by two or more processors. o Critical section: is a program sequence that must complete execution before another processor accesses the same shared resource.
2. A binary variable called a semaphore is often used to indicate whether or not a processor is executing a critical section.
3. Testing and setting the semaphore is itself a critical operation and must be performed as a single indivisible operation.
4. A semaphore can be initialized by means of a test and set instruction in conjunction with a hardware lock mechanism.

5. The instruction TSL SEM will be executed in two memory cycles (the first to read and the second to write) as follows: R □ M[SEM], M[SEM] □ 1 6.

6 Note that the lock signal must be active during the execution of the test-and-set instruction.