UNIT-V

PIPELINE AND MULTIPROCESSORS

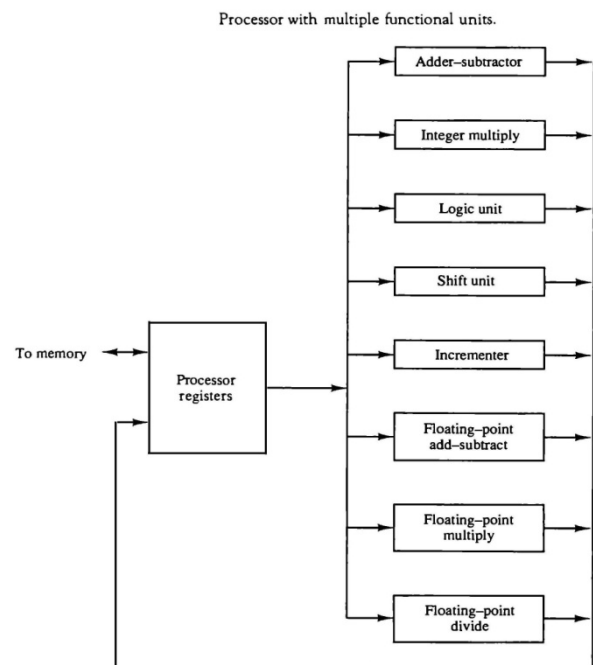**Pipeline**

- **Parallel Processing**
- **Pipelining**
- **Arithmetic Pipeline**
- **Instruction Pipeline.**

Multiprocessors

- **Characteristics of Multiprocessors**
- **Interconnection Structures**
- **Inter Processor Arbitration**
- **Inter Processor Communication and Synchronization**

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

Processor with multiple functional units.

- ✓ Parallel processing can be classified from:
  - ➢ The internal organization of the processors
  - ➢ The interconnection structure between processors
  - ➢ 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

- ✓ Flynn's 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.

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

- ✓ 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 sub operations performed in each segment are:

$$R1 \leftarrow Ai , R2 \leftarrow Bi$$
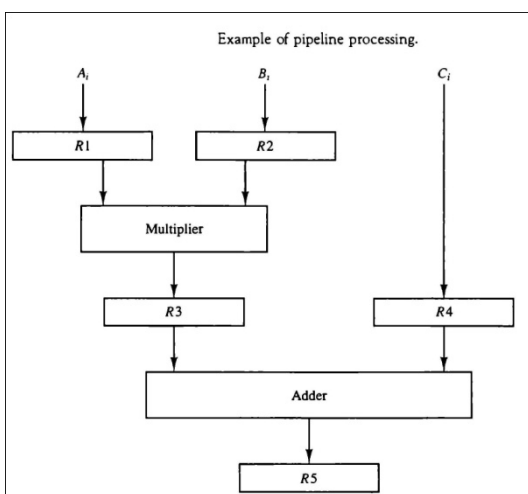$$R3 \leftarrow R1 * R2, R4 \leftarrow Ci$$
$$R5 \leftarrow R3 + R4$$

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

- ✓ The general structure of a four-segment pipeline is as shown in fig;

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

Four-segment pipeline.

- ✓ The behavior of a pipeline can be illustrated with a *space-time* diagram
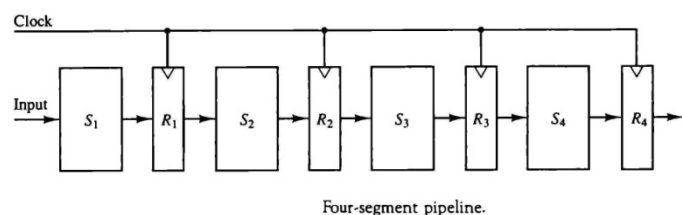
- ✓ This shows the segment utilization as a function of time
- ✓ Once the pipeline is full, it takes only one clock period to obtain an output

Space-time diagram for pipeline.

Segment:	1	2	3	4	5	6	7	8	9
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

→ Clock cycles

- 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 above example requires $[4 + 6 - 1]$ clock cycles to finish
- ✓ Consider a non-pipeline 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 non-pipeline 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, $t_n = k\, t_p$

$$S = \frac{kt_p}{t_p} = k$$

- ✓ Therefore, the theoretical maximum speedup 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 non-pipeline 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

$X = A \times 2^a$

$Y = B \times 2^b$

✓ 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

✓ $X = 0.9504 \times 10^3$ and $Y = 0.8200 \times 10^2$

✓ 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

✓ Segment 2 shifts the mantissa of Y to the right to obtain $Y = 0.0820 \times 10^3$

✓ The mantissas are now aligned

✓ Segment 3 produces the sum $Z = 1.0324 \times 10^3$

✓ Segment 4 normalizes the result by shifting the mantissa once to the right and incrementing the exponent by one to obtain $Z = 0.10324 \times 10^4$

✓ The comparator, shifter, adder-subtractor, incrementer, and decrementer in the floating-point pipeline are implemented with combinational circuits.

✓ Suppose that the time delays of the four segments are t1 = 60 ns, t2 = 70 ns, t3 = 100 ns, t4 = 80 ns, and the interface registers have a delay of tr = 10 ns. The clock cycle is chosen to be tp = t3 + tr = 110 ns. An equivalent non-pipeline floating point adder-subtractor will have a delay time tn = t1 + t2 + t3 + t4 + tr = 320ns. In this case the pipelined adder has a speedup of 32011 10 = 2. 9 over the non-pipelined adder.
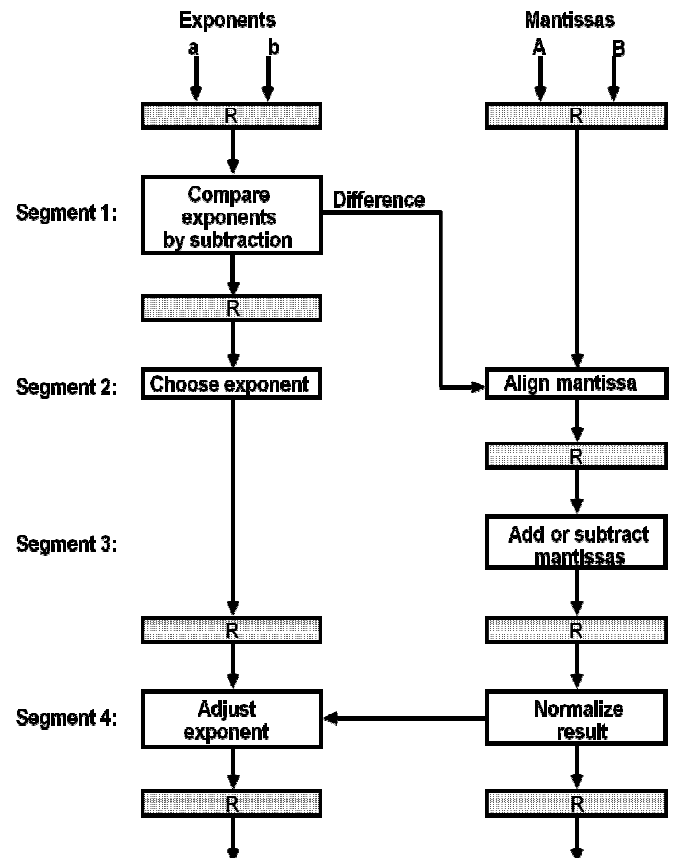
INSTRUCTION PIPELINE

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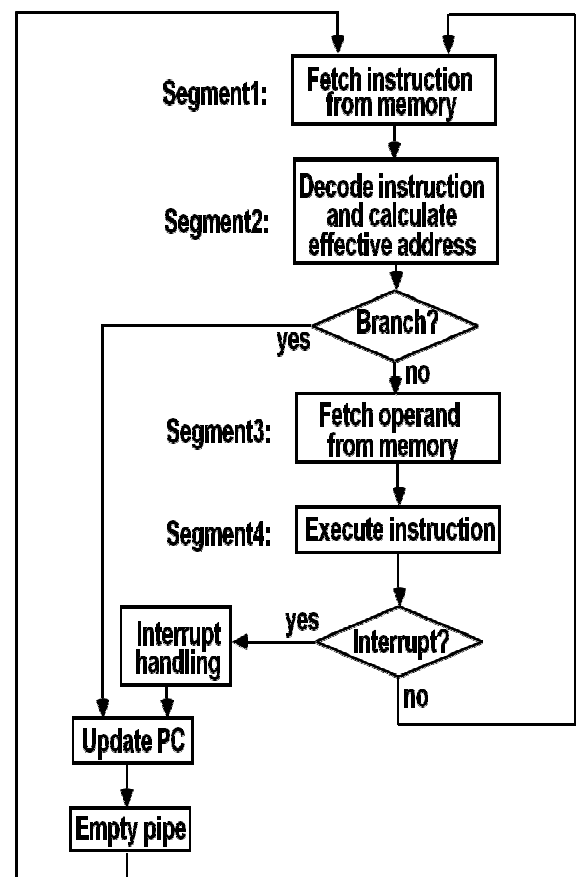
    1. Fetch the instruction from memory
    2. Decode the instruction
    3. Calculate the effective address
    4. Fetch the operands from memory
    5. Execute the instruction
    6. Store the result in the proper place

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

- ➢ Different segments taking different times to operate

- ➢ Some segment being skipped for certain operations

- ➢ Memory 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

- ✓ While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3. The effective address may be calculated in a

separate arithmetic circuit for the third instruction, and whenever the memory is available, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO

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

✓ The following figure shows the operation of the instruction pipeline. The four segments are represented in the diagram with an abbreviated symbol.

➢ FI: Fetch an instruction from memory

➢ DA:Decode the instruction and calculate the effective address of the operand

➢ FO: Fetch the operand

➢ EX: Execute the operation

Step:	1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction 1	FI	DA	FO	EX									
2		FI	DA	FO	EX								
(Branch) 3			FI	DA	FO	EX							
4				FI	-	-	FI	DA	FO	EX			
5					-	-	-	FI	DA	FO	EX		
6									FI	DA	FO	EX	
7										FI	DA	FO	EX

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

✓ Assume now that instruction 3 is a branch instruction. As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step 6. If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken, the instruction fetched previously in step 4 can be used. The pipeline then continues until a new branch instruction is encountered.

✓ Another delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand. In that case, segment FO must wait until segment EX has finished its operation.

✓ 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. Most of these instructions can be resolved by using separate instruction and data memories.

✓ **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 (BTB)** 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.

✓ **Delayed branch** is used in most RISC processors so that the compiler rearranges the instructions to delay the branch.

CHARACTERISTICS OF MULTIPROCESSORS

- ✓ A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment. The term "processor" in multiprocessor can mean either a central processing unit (CPU) or an input-output processor (lOP).

- ✓ As it is most commonly defined, a multiprocessor system implies the existence of multiple CPUs. Multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems.

- ✓ There are some similarities between multiprocessor and multicomputer systems since both support concurrent operations. 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.

- ✓ Although some large-scale computers include two or more CPUs in their overall system. Microprocessors take very little physical space and are very inexpensive brings about the feasibility of interconnecting a large number of microprocessors into one composite system.

- ✓ Very-large-scale integrated circuit technology has reduced the cost of computer components

- ✓ Multiprocessing improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system.

- ✓ The benefit derived from a multiprocessor organization is an improved system performance. The system derives its high performance in one of two ways.

  1. Multiple independent jobs can be made to operate in parallel.

  2. A single job can be partitioned into multiple parallel tasks.

- ✓ Multiprocessors 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**. Most commercial tightly coupled multiprocessors provide a cache memory with each CPU.

- ✓ An alternative model of microprocessor is the **distributed-memory or loosely coupled system**. Each processor element in a loosely coupled system has its own private local memory.

- ✓ Loosely coupled systems are most efficient when the interaction between tasks is minimal, whereas tightly coupled systems can tolerate a higher degree of interaction between tasks.

INTERCONNECTION STRUCTURES

- ✓ The components that form a multiprocessor system are CPUs, lOPs connected to input-output devices, and a memory unit that may be partitioned into a number of separate modules.

- ✓ 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 or among the processing elements in a loosely coupled system.

- ✓ There are several physical forms available for establishing an interconnection network. Some of these schemes are:

  1. Time-shared common bus
  2. Multiport memory
  3. Crossbar switch
  4. Multistage switching network
 5. Hypercube system

  Time-Shared Common Bus

- ✓ A common-bus multiprocessor system consists of a number of processors connected through a common path to a memory unit. A time-shared common bus for five processors is shown in Fig.

Time-shared common bus organization.

- ✓ Only one processor can communicate with the memory or another processor at any given time.

- ✓ Transfer operations are conducted by the processor that is in control of the bus at the time.

- ✓ A command is issued to inform the destination unit what operation is to be performed. The receiving unit recognizes its address in the bus and responds to the control signals from the sender, after which the transfer is initiated.

- ✓ The transfer conflicts must be resolved by incorporating a bus controller that establishes priorities among the requesting units.

- ✓ A single common-bus system is restricted to one transfer at a time.

- ✓ The processors in the system can be kept busy more often through the implementation of two or more independent buses to permit multiple simultaneous bus transfers.

- ✓ A more economical implementation of a dual bus structure is depicted in Fig.

- ✓ Each local bus may be connected to a CPU, an lOP, or any combination of processors.

- ✓ A system bus controller links each local bus to a common system bus.

- ✓ The IO devices connected to the local lOP, as well as the local memory, are available to the local processor.

- ✓ If an lOP is connected directly to the system bus, the IO devices attached to it may be made available to all processors. Only one processor can communicate with the shared memory and other common resources through the system bus at any given time.

- ✓ The other processors are kept busy communicating with their local memory and IO devices.

- ✓ Part of the local memory may be designed as a cache memory attached to the CPU

Multiport Memory

- ✓ A multiport memory system employs separate buses between each memory module and each CPU. This is shown in Fig. for four CPUs and four memory modules (MMs).

- ✓ Each processor bus is connected to each memory module. A processor bus consists of the address, data, and control lines required to communicate with memory.

- ✓ The memory module is said to have four ports and each port accommodates one of the buses. The module must have internal control logic to determine which port will have access to memory at any given time.

Multiport memory organization.

- ✓ Memory access conflicts are resolved by assigning fixed priorities to each memory port. The priority for memory access associated with each processor may be established by the physical port position that its bus occupies in each module.

- ✓ The advantage of the multi port memory organization is the high transfer rate that can be achieved because of the multiple paths between processors and memory.

- ✓ The disadvantage is that it requires expensive memory control logic and a large number of cables and connectors.
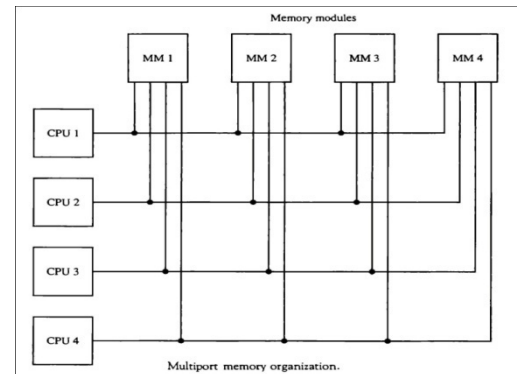
Crossbar Switch

- ✓ The crossbar switch organization consists of a number of cross points that are placed at intersections between processor buses and memory module paths.

- ✓ The small square in each cross point is a switch that determines the path from a processor to a memory module.

- ✓ Each switch point has control logic to set up the transfer path between a processor and memory.


Crossbar switch.
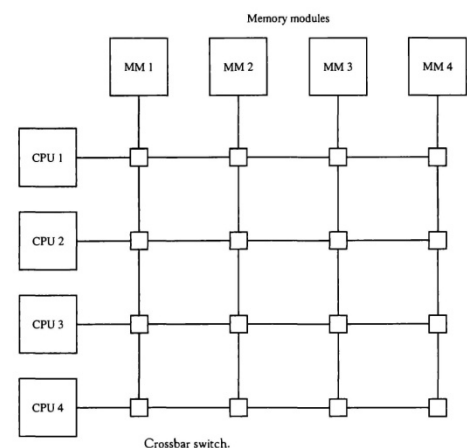
- ✓ It examines the address that is placed in the bus to determine whether its particular module is being addressed.
- ✓ It also resolves multiple requests for access to the same memory module on a predetermined priority basis.
- ✓ The functional design of a crossbar switch connected to one memory module is shown in figure.
- ✓ The circuit consists of multiplexers that select the data address, and control from one CPU for communication with the memory module.
- ✓ Priority levels are established by the arbitration logic to select one CPU when two or more CPUs attempt to access the same memory.
- ✓ A crossbar switch organization supports simultaneous transfers from memory modules because there is a separate path associated with each module.

Block diagram of crossbar switch.
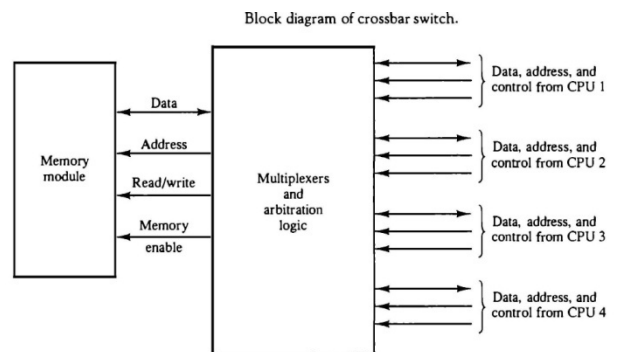
Multistage Switching Network

- ✓ The basic component of a multistage network is a two-input, two-out interchange switch.
- ✓ The switch has the capability of connecting input A to either of the outputs. Terminal B of the switch behaves in a similar fashion. The switch also has the capability to arbitrate between conflicting requests.
- ✓ Using the 2 x 2 switch as a building block, it is possible to build multistage network to control the communication between a number of sources and destinations.
- ✓ Consider the binary tree shown Fig. The two processors P1 and P2 are connected through switches to eight memory modules marked in binary from 000 through 111.
- ✓ The path from source to a destination is determined from the binary bits of the destination number. The first bit of the destination number determines the switch output in the first level. The second bit specifies the output of the switch in the second level, and the third bit specifies the output of the switch in the third level.
- ✓ Many different topologies have been proposed for multistage switching networks to control processor-memory communication in a tightly coupled multiprocessor system or to control the communication between the processing elements in a loosely coupled system.
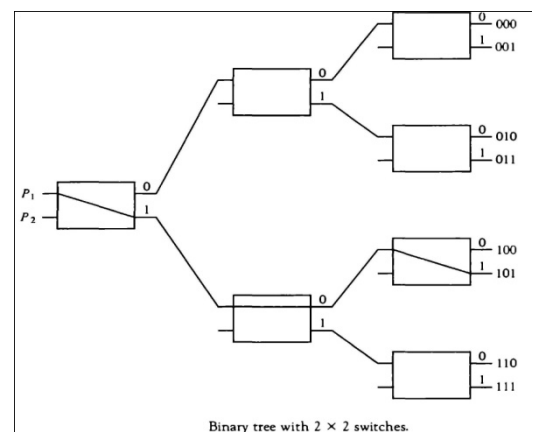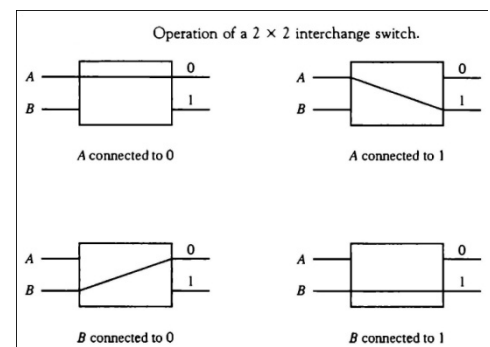
- ✓ One such topology is the omega switching network shown in Fig.

8x8 Omega Switching Network

- ✓ In this configuration, there is exactly one path from each source to any particular destination.

- ✓ Some request patterns, however, cannot be connected simultaneously. For example, any two sources cannot be connected simultaneously to destinations 000 and 001.
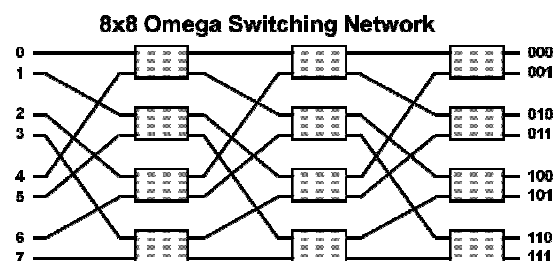
Hypercube Interconnection

- ✓ The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of $N = 2^n$ processors interconnected in an n-dimensional binary cube.

- ✓ Each processor forms a node of the cube.

- ✓ Each processor has direct communication paths to n other neighbor processors. These paths correspond to the edges of the cube.

- ✓ Fig shows the hypercube structure for n = 1, 2, and 3.

- ✓ A one-cube structure has n = 1 and $2^n = 2$. It contains two processors interconnected by a single path.

- ✓ A two-cube structure has n = 2 and $2^n = 4$. It contains four nodes interconnected as a square.

- ✓ A three-cube structure has eight nodes interconnected as a cube.

- ✓ An n -cube structure has 2^n nodes with a processor residing in each node. Each node is assigned a binary address in such a way that the addresses of two neighbors differ in exactly one bit position.

- ✓ Routing messages through an n-cube structure may take from one to n links from a source node to a destination node.

- ✓ For example, in a three-cube structure, node 000 can communicate directly with node 001. It must cross at least two links to communicate with 011 (from 000 to 001 to 011 or from 000 to 010 to 011).

- ✓ A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address. The resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ. The message is then sent along any one of the axes.

- ✓ For example, in a three-cube structure, a message at 010 going to 001 produces an XOR of the two addresses equal to 011 . The message can be sent along the second axis to 000 and then through the third axis to 001.
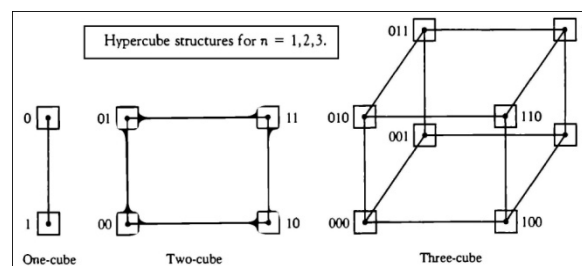
INTERPROCESSOR ARBITRATION

- ✓ 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.

- ✓ A memory bus consists of lines for transferring data, address, and read/write information.

- ✓ An I/O bus is used to transfer information to and from input and output devices.

- ✓ A bus that connects major components in a multiprocessor system, such as CPUs, IOPs, and memory, is called a **system bus**.

- ✓ The processors in a shared memory multiprocessor system request access to common memory or other common resources through the system bus. If no other processor is currently utilizing the bus, the requesting processor may be granted access immediately.

- ✓ Other processors may request the system bus at the same time. Arbitration must then be performed to resolve this multiple contention for the shared resources. The arbitration logic would be part of the system bus controller placed between the local bus and the system bus.
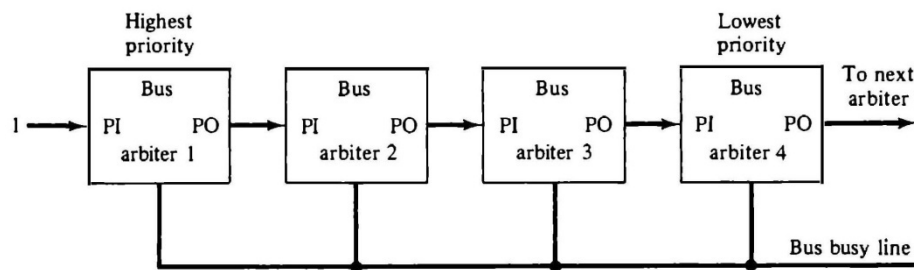
System Bus

- ✓ A typical system bus consists of approximately 100 signal lines. These lines are divided into three functional groups: data, address, and control. In addition, there are power distribution lines that supply power to the components.

- ✓ For example, the IEEE standard 796 multibus system has 16 data lines, 24 address lines, 26 control lines, and 20 power lines, for a total of 86 lines.

- ✓ Data transfers over the system bus may be **synchronous or asynchronous**.

- ✓ In a **synchronous bus**, each data item is transferred during a time slice known in advance to both source and destination units. Synchronization is achieved by driving both units from a common clock source.

- ✓ In an **asynchronous bus**, each data item being transferred is accompanied by handshaking control signals to indicate when the data are transferred from the source and received by the destination

- ✓ The following table lists the 86 lines that are available in the IEEE standard 796 multibus.

IEEE Standard 796 Multibus Signals	
	Signal name
Data and address	
Data lines (16 lines)	DATA0–DATA15
Address lines (24 lines)	ADRS0–ADRS23
Data transfer	
Memory read	MRDC
Memory write	MWTC
IO read	IORC
IO write	IOWC
Transfer acknowledge	TACK
Interrupt control	
Interrupt request (8 lines)	INT0–INT7
Interrupt acknowledge	INTA
Miscellaneous control	
Master clock	CCLK
System initialization	INIT
Byte high enable	BHEN
Memory inhibit (2 lines)	INH1–INH2
Bus lock	LOCK
Bus arbitration	
Bus request	BREQ
Common bus request	CBRQ
Bus busy	BUSY
Bus clock	BCLK
Bus priority in	BPRN
Bus priority out	BPRO
Power and ground (20 lines)	

Serial Arbitration Procedure

✓ Arbitration procedures service all processor requests on the basis of established priorities. A hardware bus priority resolving technique can be established by means of a serial or parallel connection of the units requesting control of the system bus.

✓ The serial priority resolving technique is obtained from a daisy-chain connection of bus arbitration circuits similar to the priority interrupt logic.

✓ The processors connected to the system bus are assigned priority according to their position along the priority control line.

✓ The device closest to the priority line is assigned the highest priority. When multiple devices concurrently request the use of the bus, the device with the highest priority is granted access to it.


Serial (daisy-chain) arbitration.

✓ The processor whose arbiter has a PI = 1 and PO = 0 is the one that is given control of the system bus

✓ A processor may be in the middle of a bus operation when a higher priority processor requests the bus. The lower-priority processor must complete its bus operation before it relinquishes control of the bus.

✓ When an arbiter receives control of the bus (because its PI = 1 and PO = 0) it examines the busy line. If the line is inactive, it means that no other processor is using the bus. The arbiter activates the busy line and its processor takes control of the bus. However, if the arbiter finds the busy line active, it means that another processor is currently using the bus.

✓ The arbiter keeps examining the busy line while the lower-priority processor that lost control of the bus completes its operation.

✓ When the bus busy line returns to its inactive state, the higher-priority arbiter enables the busy line, and its corresponding processor can then conduct the required bus transfers.

Parallel Arbitration Logic

Parallel arbitration.

✓ The parallel bus arbitration technique uses an external priority encoder and a decoder as shown in Fig. Each bus arbiter in the parallel scheme has a bus request output line and a bus acknowledge input line.

✓ Each arbiter enables the request line when its processor is requesting access to the system bus. The processor takes control of the bus if its acknowledge input line is enabled.
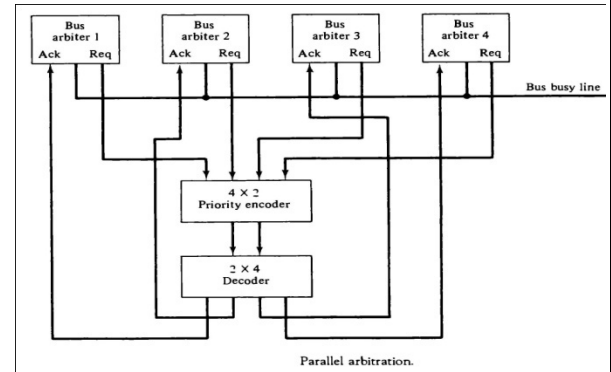
Dynamic Arbitration Algorithms

✓ A dynamic priority algorithm gives the system the capability for changing the priority of the devices while the system is in operation.

✓ The **time slice algorithm** allocates a fixed-length time slice of bus time that is offered sequentially to each processor, in round-robin fashion. The service given to each system component with this scheme is independent of its location along the bus.

✓ In a bus system that uses **polling**, the bus grant signal is replaced by a set of lines called poll lines which are connected to all units. These lines are used by the bus controller to define an address for each device connected to the bus.

✓ When a processor that requires access recognizes its address, it activates the bus busy line and then accesses the bus. After a number of bus cycles, the polling process continues by choosing a different processor. The polling sequence is normally programmable, and as a result, the selection priority can be altered under program control.

✓ The **least recently used (LRU) algorithm** gives the highest priority to the requesting device that has not used the bus for the longest interval. The priorities are adjusted after a number of bus cycles according to the LRU algorithm.

✓ In the **first-come, first-serve** scheme, requests are served in the order received. To implement this algorithm, the bus controller establishes a queue arranged according to the time that the bus requests arrive. Each processor must wait for its turn to use the bus on a first-in, first-out (FIFO) basis.

✓ The rotating daisy-chain procedure is a dynamic extension of the daisy chain algorithm. In this scheme there is no central bus controller, and the priority line is connected from the priority-out of the last device back to the priority-in of the first device in a closed loop.

✓ Each arbiter priority for a given bus cycle is determined by its position along the bus priority line from the arbiter whose processor is currently controlling the bus. Once an arbiter releases the bus, it has the lowest priority.

INTERPROCESSOR COMMUNICATION AND SYNCHRONIZATION

- ✓ 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 common input-output channels.

- ✓ In a shared memory multiprocessor system, the most common procedure is to set aside a portion of memory that is accessible to all processors. The primary use of the common memory is to act as a message center similar to a mailbox, where each processor can leave messages for other processors and pick up messages intended for it.

- ✓ The sending processor structures a request, a message, or a procedure, and places it in the memory mailbox. Status bits residing in common memory are generally used to indicate the condition of the mailbox, whether it has meaningful information, and for which processor it is intended.

- ✓ The receiving processor can check the mailbox periodically to determine if there are valid messages for it. The response time of this procedure can be time consuming since a processor will recognize a request only when polling messages.

- ✓ A more efficient procedure is for the sending processor to alert the receiving processor directly by means of an interrupt signal. This can be accomplished through a software-initiated interprocessor interrupt by means of an instruction in the program of one processor which when executed produces an external interrupt condition in a second processor. This alerts the interrupted processor of the fact that a new message was inserted by the interrupting processor.

- ✓ In addition to shared memory, a multiprocessor system may have other shared resources. For example, a magnetic disk storage unit connected to an lOP may be available to all CPUs. This provides a facility for sharing of system programs stored in the disk.

- ✓ A communication path between two CPUs can be established through a link between two lOPs associated with two different CPUs. This type of link allows each CPU to treat the other as an IO device so that messages can be transferred through the IO path.

- ✓ To prevent conflicting use of shared resources by several processors there must be a provision for assigning resources to processors. This task is given to the operating system. 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.

- ✓ In a master-slave mode, one processor, designated the master, always executes the operating system functions. The remaining processors, denoted as slaves, do not perform operating system functions. If a slave processor needs an operating system service, it must request it by interrupting the master and waiting until the current program can be interrupted.

- ✓ 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 where every processor may have its own copy of the entire operating system.

- ✓ 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. This type of organization is also referred to as a floating operating system since the routines float from one processor to another and the execution of the routines may be assigned to different processors at different times.

- ✓ In a loosely coupled multiprocessor system the memory is distributed among the processors and there is no shared memory for passing information.

- ✓ The communication between processors is by means of message passing through IO channels. The communication is initiated by one processor calling a procedure that resides in the memory of the processor with which it wishes to communicate. When the sending processor and receiving processor name each other as a source and destination, a channel of communication is established.

- ✓ A message is then sent with a header and various data objects used to communicate between nodes. There may be a number of possible paths available to send the message between any two nodes.

- ✓ The operating system in each node contains routing information indicating the alternative paths that can be used to send a message to other nodes. The communication efficiency of the interprocessor network depends on the communication routing protocol, processor speed, data link speed, and the topology of the network.

Interprocessor Synchronization

- ✓ 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. For example, parameters passed to a procedure in a different processor constitute interprocessor communication.

- ✓ *Synchronization* refers to the special case where the data used to communicate between processors is control information. Synchronization is needed to enforce the correct sequence of processes and to ensure mutually exclusive access to shared writable data.

- ✓ 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.

- ✓ One of the most popular methods is through the use of a binary semaphore. Mutual Exclusion with a Semaphore

- ✓ A properly functioning multiprocessor system must provide a mechanism that will guarantee orderly access to shared memory and other shared resources.

- ✓ This is necessary to protect data from being changed simultaneously by two or more processors. This mechanism has been termed mutual exclusion. Mutual exclusion must be provided in a multiprocessor system to enable one processor to exclude or lock out access to a shared resource by other processors when it is in a critical section.

- ✓ A **critical section** is a program sequence that, once begun, must complete execution before another processor accesses the same shared resource.

- ✓ A binary variable called a **semaphore** is often used to indicate whether or not a processor is executing a critical section. A semaphore is a software controlled flag that is stored in a memory location that all processors can access.

- ✓ When the semaphore is equal to 1, it means that a processor is executing a critical program, so that the shared memory is not available to other processors.

- ✓ When the semaphore is equal to 0, the shared memory is available to any requesting processor. Processors that share the same memory segment agree by convention not to use the memory segment unless the semaphore is equal to 0, indicating that memory is available . They also agree to set the semaphore to 1 when they are executing a critical section and to clear it to 0 when they are finished.

- ✓ Testing and setting the semaphore is itself a critical operation and must be performed as a single indivisible operation. If it is not, two or more processors may test the semaphore simultaneously and then each set it, allowing them to enter a critical section at the same time. This action would allow simultaneous execution of critical section, which can result in erroneous initialization of control parameters and a loss of essential information.

- ✓ A semaphore can be initialized by means of a test and set instruction in conjunction with a hardware lock mechanism.

- ✓ A hardware lock is a processor generated signal that serves to prevent other processors from using the system bus as long as the signal is active. The test-and-set instruction tests and sets a semaphore and activates the lock mechanism during the time that the instruction is being executed.

- ✓ This prevents other processors from changing the semaphore between the time that the processor is testing it and the time that it is setting it. Assume that the semaphore is a bit in the least significant position of a memory word whose address is symbolized by SEM.

- ✓ Let the mnemonic TSL designate the "test and set while locked" operation. The instruction TSL SEM will be executed in two memory cycles (the first to read and the second to write) without interference as follows:

  $R \leftarrow M[SEM]$ Test semaphore

  $M[SEM] \leftarrow 1$ Set semaphore

- ✓ The semaphore is tested by transferring its value to a processor register R and then it is set to 1. The value in R determines what to do next.

- ✓ If the processor finds that R = 1, it knows that the semaphore was originally set. (The fact that it is set again does not change the semaphore value.) That means that another processor is executing a critical section, so the processor that checked the semaphore does not access the shared memory.

- ✓ If R = 0, it means that the common memory (or the shared resource that the semaphore represents) is available. The semaphore is set to 1 to prevent other processors from accessing memory. The processor can now execute the critical section.

- ✓ The last instruction in the program must clear location SEM to zero to release the shared resource to other processors. Note that the lock signal must be active during the execution of the test-and-set instruction. It does not have to be active once the semaphore is set.

- ✓ Thus the lock mechanism prevents other processors from accessing memory while the semaphore is being set. The semaphore itself, when set, prevents other processors from accessing shared memory while one processor is executing a critical section.