Unit-5

RISC and CISC

Reduced Set Instruction Set Architecture (RISC) –
The main idea behind 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, 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.

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 increase in number of cycles per instruction.

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

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

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. Pipeline can be achieved.

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.

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

- **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 first load command to load data in registers then it will use suitable operator and then it will store result in 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 less transistors due to less complex command.

Difference –

RISC	CISC
Focus on software	Focus on hardware
Uses only Hardwired control unit	Uses both hardwired and micro programmed 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
A instruction execute in single	Instruction take more than one clock

RISC	CISC
clock cycle	cycle
A instruction fit in one word	Instruction are larger than size of one word

Parallel processing:

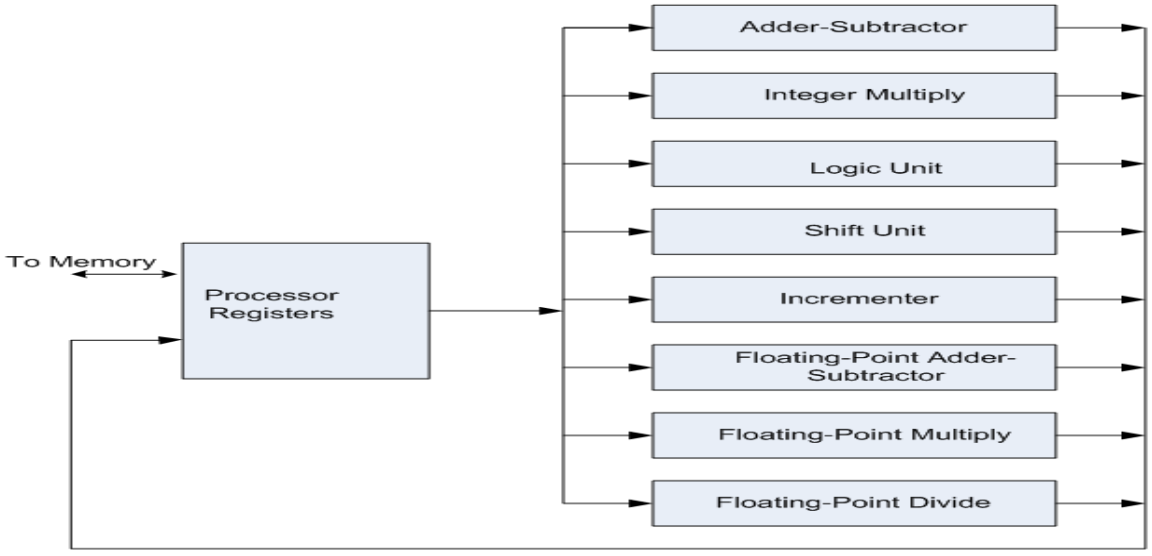
- Parallel processing is a term used for a large class of techniques that

are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.

- It refers to techniques that are used to provide simultaneous data processing.
- The system may have two or more ALUs to be able to execute two or more instruction at the same time.
- The system may have two or more processors operating concurrently.
- It can be achieved by having multiple functional units that perform same or different operation simultaneously.
- Example of parallel Processing:
  - Multiple Functional Unit:

Separate the execution unit into eight functional units operating in parallel.

- There are variety of ways in which the parallel processing can be classified
  - Internal Organization of Processor
  - Interconnection structure between processors
  - Flow of information through system



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

		Number of *Data Streams*	
		Single	Multiple
Number of *Instruction Streams*	Single	SISD	SIMD
	Multiple	MISD	MIMD

- SISD represents the organization containing single control unit, a processor unit and a memory unit. Instruction are executed sequentially and system may or may not have internal parallel processing capabilities.
- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- MISD structure is of only theoretical interest since no practical system has been constructed using this organization.
- MIMD organization refers to a computer system capable of processing several programs at the same time.

The main difference between multicomputer system and multiprocessor system is that the multiprocessor system is controlled by one operating system that provides interaction between processors and all the component of the system cooperate in the solution of a problem.

- Parallel Processing can be discussed under following topics:
  - Pipeline Processing
  - Vector Processing
  - Array Processors

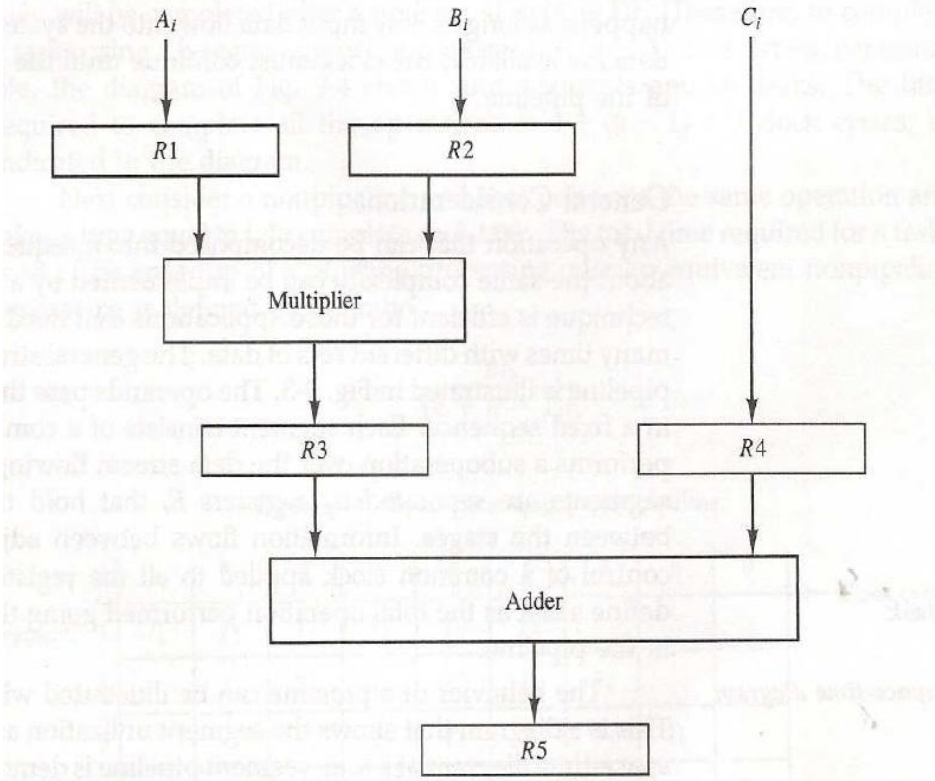
PIPELINING:

- A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.
- It is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segments that operates concurrently with all other segments.
- Each segment performs partial processing dictated by the way task is partitioned.
- The result obtained from each segment is transferred to next segment.
- The final result is obtained when data have passed through all segments.

- Suppose we have to perform the following task:
- Each sub operation is to be performed in a segment within a pipeline. Each segment has one or two registers and a combinational circuit.

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

Figure Example of pipeline processing.

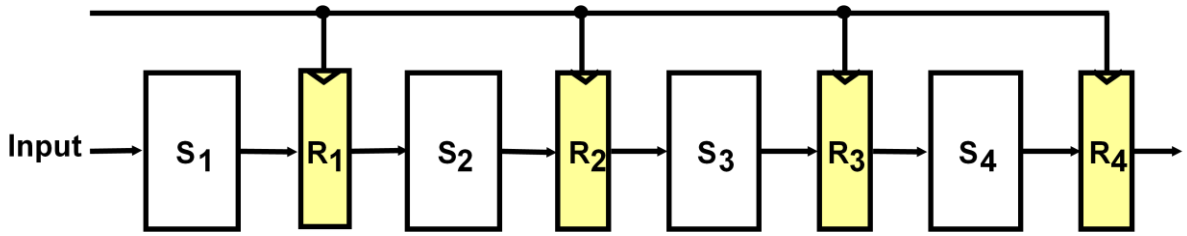


Suboperations in each segment:	$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Load A_i and B_i
	$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply and load C_i
	$R5 \leftarrow R3 + R4$	Add

OPERATIONS IN EACH PIPELINE STAGE:

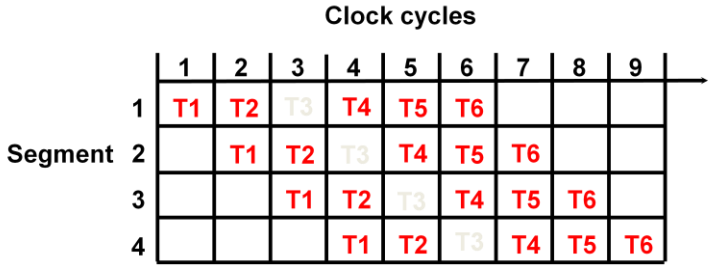
Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	---	---	-------
2	A2	B2	A1 * B1	C1	-------
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

- General Structure of a 4-Segment Pipeline



- Space-Time Diagram

The following diagram shows 6 tasks T1 through T6 executed in 4segments.



No matter how many segments, once the pipeline is full, it takes only one clock period to obtain an output.

PIPELINE SPEEDUP:

Consider the case where a k-segment pipeline used to execute n tasks.

- n = 6 in previous example

- k = 4 in previous example

- Pipelined Machine (k stages, n tasks)
  - The first task t1 requires k clock cycles to complete its operation since there are k segments
  - The remaining n-1 tasks require n-1 clock cycles
  - The n tasks clock cycles = k+(n-1) (9 in previous example)
- Conventional Machine (Non-Pipelined)
  - Cycles to complete each task in nonpipeline = k
  - For n tasks, n cycles required is
- Speedup (S)
  - S = Nonpipeline time /Pipeline time

- For n tasks: S = nk/(k+n-1)
- As n becomes much larger than k-1; Therefore, S = nk/n = k

PIPELINE AND MULTIPLE FUNCTION UNITS:

Example:

- 4-stage pipeline

- 100 tasks to be executed

- 1 task in non-pipelined system; 4 clock cycles

Pipelined System : k + n - 1 = 4 + 99 = 103 clock cycles

Non-Pipelined System : n*k = 100 * 4 = 400 clock cycles

Speedup : S_k = 400 / 103 = 3.88

Types of Pipelining:

- Arithmetic Pipeline
- Instruction Pipeline

ARITHMETIC PIPELINE:

- Pipeline arithmetic units are usually found in very high speed computers.
- They are used to implement floating point operations.

- We will now discuss the pipeline unit for the floating point addition and subtraction.
- The inputs to floating point adder pipeline are two normalized floating point numbers.
- A and B are mantissas and a and b are the exponents.
- The floating point addition and subtraction can be performed in four segments.

Floating-point adder:

[1] Compare the exponents

[2] Align the mantissa

[3] Add/sub the mantissa

[4] Normalize the result

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

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

1) Compare exponents :

$3 - 2 = 1$

2) Align mantissas

$X = 0.9504 \times 10^3$

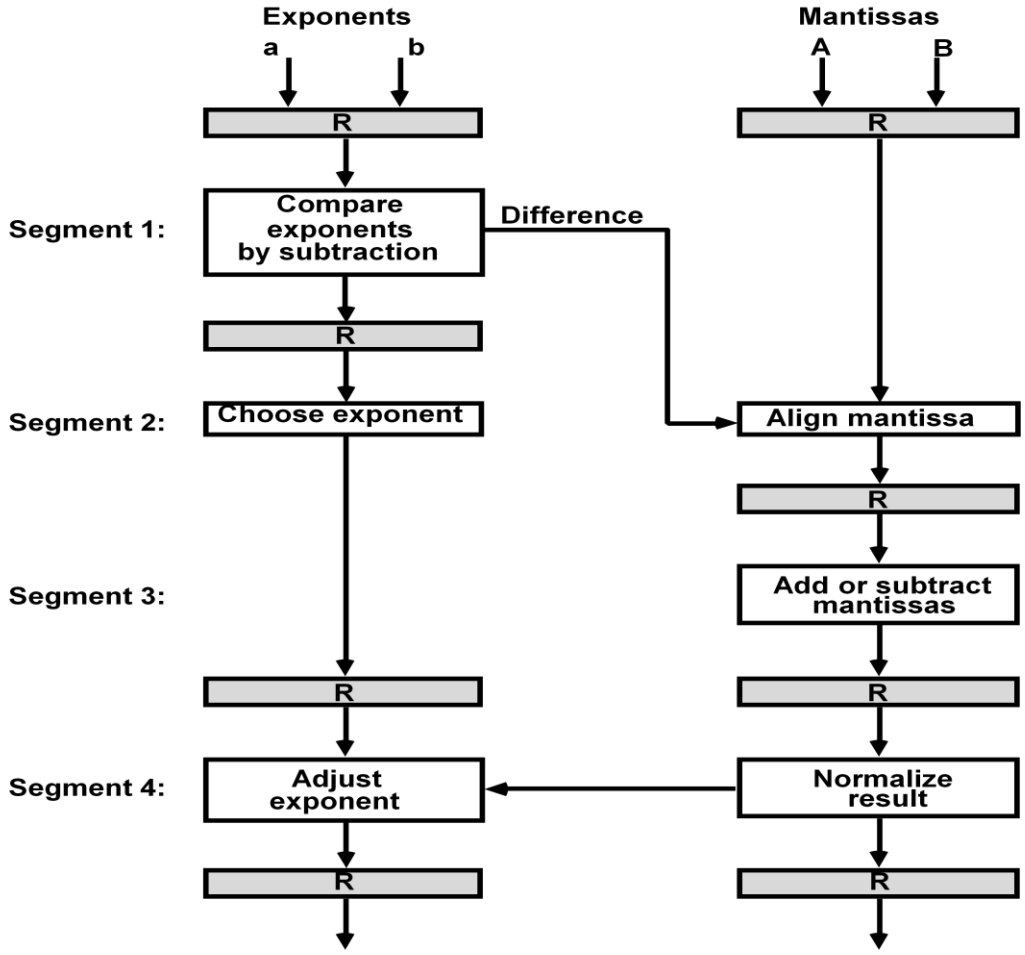
$Y = 0.08200 \times 10^3$

3) Add mantissas

$Z = 1.0324 \times 10^3$

4) Normalize result

$Z = 0.10324 \times 10^4$



Instruction Pipeline:

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- An instruction pipeline reads consecutive instruction from memory while previous instruction are being executed in other segments.
- This caused the instruction fetch and execute segments to overlap and perform simultaneous operation.

Four Segment CPU Pipeline:

- FI segment fetches the instruction.
- DA segment decodes the instruction and calculate the effective address.
- FO segment fetches the operand.
- EX segment executes the instruction.

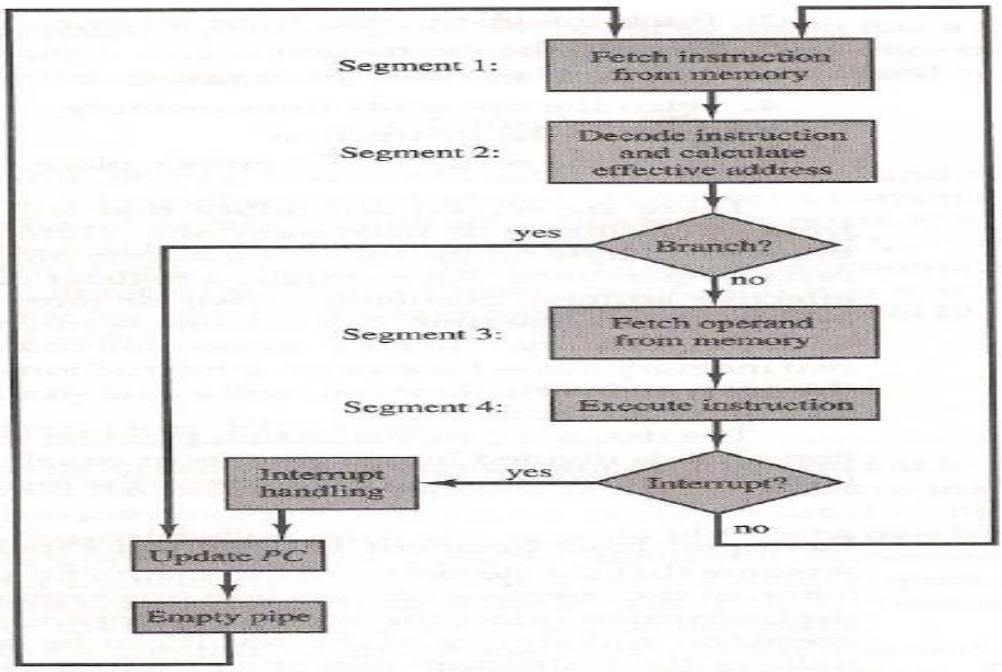


Figure Four-segment CPU pipeline.

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

Figure Timing of instruction pipeline.

INSTRUCTION CYCLE:

Pipeline processing can occur also in the instruction stream. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.

Six Phases* in an Instruction Cycle

[1] Fetch an instruction from memory

[2] Decode the instruction

[3] Calculate the effective address of the operand

[4] Fetch the operands from memory

[5] Execute the operation

[6] Store the result in the proper place

* Some instructions skip some phases

* Effective address calculation can be done in the part of the decoding phase

* Storage of the operation result into a register is done automatically in the execution phase

==> 4-Stage Pipeline

[1] FI: Fetch an instruction from memory

[2] DA: Decode the instruction and calculate the effective address of the operand

[3] FO: Fetch the operand

[4] EX: Execute the operation

Pipeline Conflicts :

- Pipeline Conflicts : 3 major difficulties

In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. *Resource conflicts* caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
2. *Data dependency* conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. *Branch difficulties* arise from branch and other instructions that change the value of *PC*.

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

2) Data dependency: when an instruction depend on the result of a previous instruction, but this result is not yet available.

Example: an instruction with register indirect mode cannot proceed to fetch the operand if the previous instruction is loading the address into the register.

3) Branch difficulties: branch and other instruction (interrupt, ret, ..) that change the value of PC.

Handling Data Dependency:

- This problem can be solved in the following ways:
    - Hardware interlocks: It is the circuit that detects the conflict situation and delayed the instruction by sufficient cycles to resolve the conflict.
    - Operand Forwarding: It uses the special hardware to detect the conflict and avoid it by routing the data through the special path between pipeline segments.
    - Delayed Loads: The compiler detects the data conflict and reorder the instruction as necessary to delay the loading of the conflicting data by inserting no operation instruction.

Handling of Branch Instruction:

- Pre fetch the target instruction.
- Branch target buffer(BTB) included in the fetch segment of the pipeline
- Branch Prediction
- Delayed Branch

RISC Pipeline:

- Simplicity of instruction set is utilized to implement an instruction pipeline using small number of sub-operation, with each being executed in single clock cycle.

Since all operation are performed in the register, there is no need of effective address calculation.

Three Segment Instruction Pipeline:

- I: Instruction Fetch
- A: ALU Operation
- E: Execute Instruction

Delayed Load:

Consider now the operation of the following four instructions

1. LOAD: $R1 \leftarrow M$[address 1]
2. LOAD: $R2 \leftarrow M$[address 2]
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: M[address 3] $\leftarrow R3$

Clock cycles:	1	2	3	4	5	6
1. Load $R1$	I	A	E			
2. Load $R2$		I	A	E		
3. Add $R1 + R2$			I	A	E	
4. Store $R3$				I	A	E

Pipeline timing with data conflict

Clock cycle:	1	2	3	4	5	6	7
1. Load $R1$	I	A	E				
2. Load $R2$		I	A	E			
3. No-operation			I	A	E		
4. Add $R1 + R2$				I	A	E	
5. Store $R3$					I	A	E

Pipeline timing with delayed load

Delayed Branch:

Let us consider the program having the following 5 instructions

Load from memory to $R1$
Increment $R2$
Add $R3$ to $R4$
Subtract $R5$ from $R6$
Branch to address X

Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to *X*					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in *X*								I	A	E

Using no-operation instructions

Organization of Intel 8085 Micro-Processor:

The microprocessors that are available today came with a wide variety of capabilities and architectural features. All of them, regardless of their diversity, are provided with at least the following functional components, which form the central processing unit (CPU) of a classical computer.

1. Register Section : A set of registers for temporary storage of instructions, data and address of data .
2. Arithmetic and Logic Unit : Hardware for performing primitive arithmetic and logical operations .
3. Interface Section : Input and output lines through which the microprocessor communicates with the outside world .
4. Timing and Control Section : Hardware for coordinating and controlling the activities of the various sections within the microprocessor and other devices connected to the interface section .

The block diagram of the microprocessor along with the memory and Input/Output (I/O) devices is shown in the Figure 11.1.

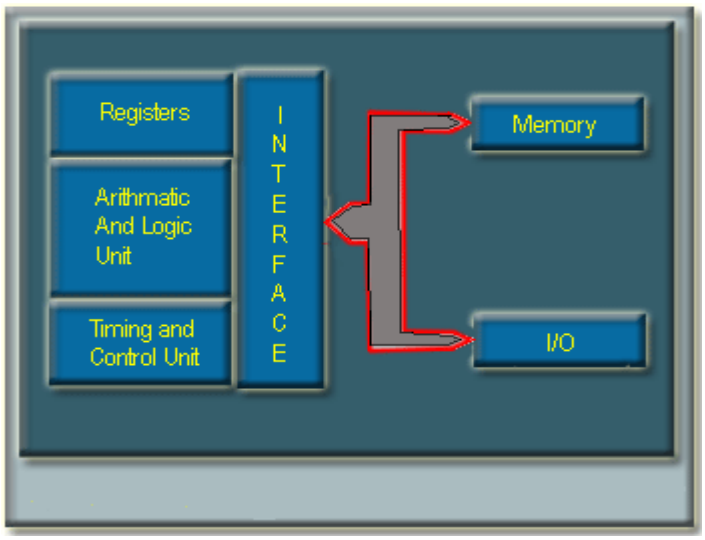


Figure 11.1: Block diagram of Micorprocessor with memory and I/O.

Intel Microprocessors:

Intel 4004 is the first 4-bit microprocessor introduced by Intel in 1971. After that Intel introduced its first 8-bit microprocessor 8088 in 1972.

These microprocessors could not last long as general-purpose microprocessors due to their design and performance limitations.

In 1974, Intel introduced the first general purpose 8-bit microprocessor 8080 and this is the first step of Intel towards the development of advanced microprocessor.

After 8080, Intel launched microprocessor 8085 with a few more features added to its architecture, and it is considered to be the first functionally complete microprocessor.

The main limitations of the 8-bit microprocessors were their low speed, low memory capacity, limited number of general purpose registers and a less powerful instruction set .

To overcome these limitations Intel moves from 8-bit microprocessor to 16-bit microprocessor.

In the family of 16-bit microprocessors, Intel's 8086 was the first one introduced in 1978 .

8086 microprocessor has a much powerful instruction set along with the architectural developments, which imparted substantial programming flexibility and improvement over the 8-bit microprocessor.

Microprocessor Intel 8085 :

Intel 8085 is the first popular microprocessor used by many vendors. Due to its simple architecture and organization, it is easy to understand the working principle of a microprocessor.

Register in the Intel 8085:

The programmable registers of 8085 are as follows -

- One 8-bit accumulator A.
- Six 8-bit general purpose register (GPR's)
  B, C, D , E , H and L.
- The GPR's are also accessible as three 16-bit register pairs BC, DE and HL.
    - There is a 16-bit program counter(PC), one 16-bit stack pointer(SP) and 8-bit flag register . Out of 8 bits of the flag register , only 5 bits are in use.

The programmable registers of the 8085 are shown in the Figure 11.2-

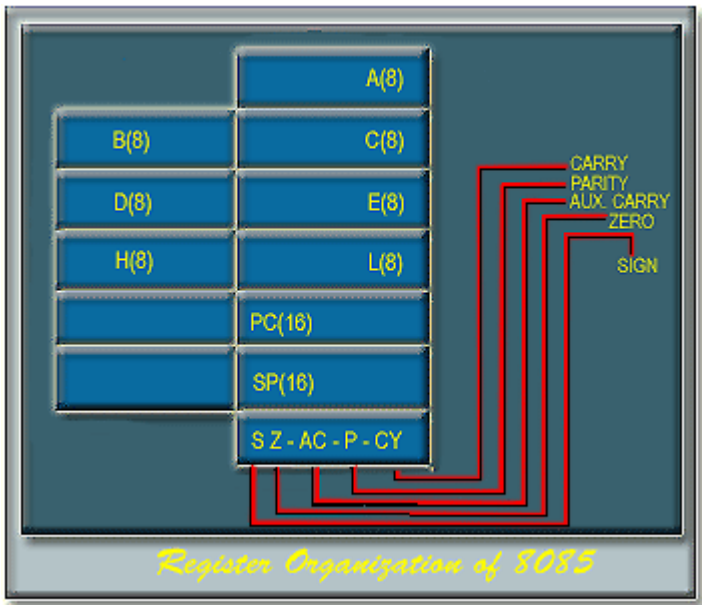


Figure 11.2: Register Organisation of 8085

Apart from these programmable registers , some other registers are also available which are not accessible to the programmer . These registers include -

- Instruction Register(IR).
- Memory address and data buffers(MAR & MDR).
  - MAR: Memory Address Register.
  - MDR: Memory Data Register.
- Temporary register for ALU use.

ALU of 8085 :

The 8-bit parallel ALU of 8085 is capable of performing the following operations –

Arithmetic : Addition, Subtraction, Increment, Decrement, Compare.

Logical : AND, OR, EXOR, NOT, SHIFT / ROTATE, CLEAR.

Because of limited chip area , complex operations like multiplication, division, etc are not available, in earlier processors like 8085.

The operations performed on binary 2's complement data.

The five flag bits give the status of the microprocessor after an ALU operation.

The carry (C) flag bit indicates whether there is any overflow from the MSB.

The parity (P) flag bit is set if the parity of the accumulater is even.

The Auxiliary Carry (AC) flag bit indicates overflow out of bit –3 (lower nibble) in the same manner, as the C-flag indicates the overflow out of the bit-7.

The Zero (Z) flag bit is set if the content of the accumulator after any ALU operations is zero.

The Sign(S) flag bit is set to the condition of bit-7 of the accumulator as per the sign of the contents of the accumulator(positive or negative).

The Interface Section:

Microprocessor chips are equipped with a number of pins for communication with the outside world. This is known as the system bus.
The interface lines of the Intel 8085 microprocessor are shown in the Figure 11.3 –

Address and Data Bus

The AD0 - AD7 lines are used as lower order 8-bit address bus and data bus , in time division multiplexed manner .

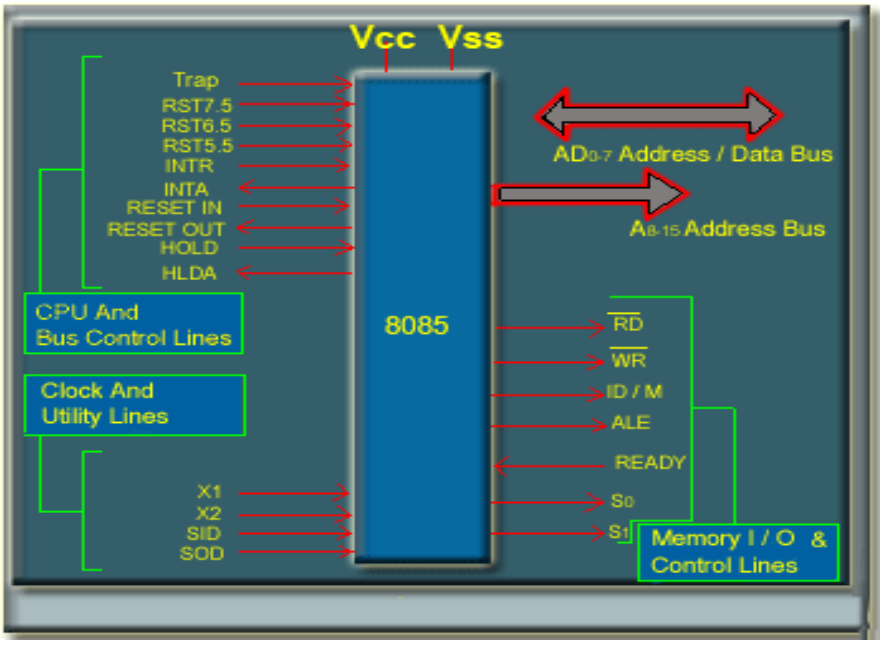
The A8 - A15 lines are used for higher order 8 bit of address bus.

There are seven memory and I/O control lines -

RD : indicates a READ operation when the signal is LOW .

WR : indicates a WRITE operation when the signal is LOW .

IO/M : indicates memory access for LOW and I/O access for HIGH .



ALE : ALE is an address latch enable signal , this signal is HIGH when address information is present in AD0-AD7 . The falling edge of ALU can be used to latch the address into an external buffer to de-multiples the address bus .

READY : READY line is used for communication with slow memory and I/O devices .

S0 and S1 : The status of the system bus is difined by the S0 and S1 lines as follows -

S1	S0	Operation Specified
0	0	Halt
0	1	Memory or I/O WRITE
1	0	Memory or I/O READ
1	1	Instruction Fetch

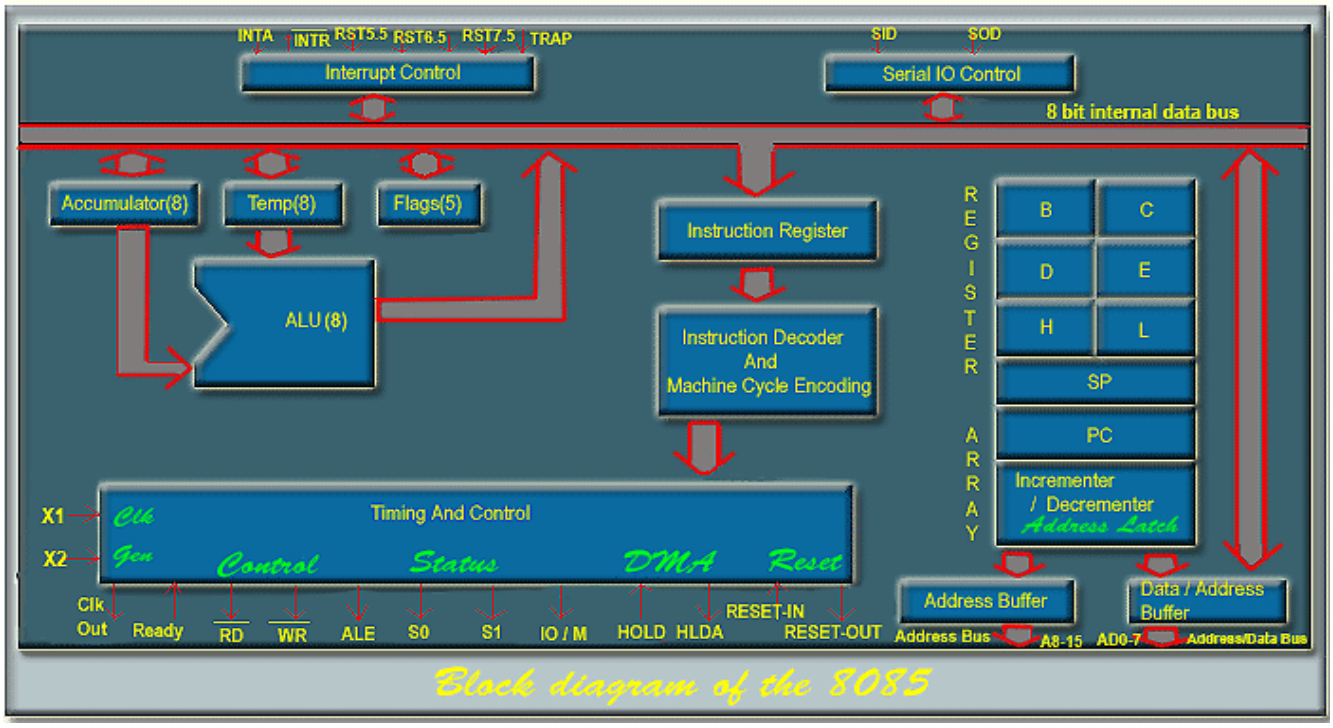
There are ten lines associated with CPU and bus control-

- TRAP , RST7.5 , RST6.5 , RST5.5 and INTR are the Interrupt lines.
- INTA: Interrupt acknowledge line.
- RESET IN : This is the reset input signal to the 8085.
- RESET OUT : The 8085 generates the RESET-OUT signal in response to RESET-IN signal , which can be used as a system reset signal .
- HOLD : HOLD signal is used for DMA request.
- HLDA : HLDA signal is used for DMA grant .

- Clock and Utility Lines :

X1 and X2: X1 and X2 are provided to connect a crystal or a RC network for generating theclockinternaltothe chip.
Sid: input line for serial data communication.
Sod: output line for serial data communication.
V_{cc} and v_{ss}: power supply.

- The block diagram of the Intel 8085 is shown in the Figure 11.4 -



Addressing Modes :

The 8085 has four different modes for addressing data stored in memory or in registers -

Direct: Bytes 2 and 3 of the instruction contains the exact memory address of the data item(the low-order bits of the address are in byte 2 , the high-order bits in byte 3).

Register: The instruction specifies the register or register pair in which the data are located.

Register Indirect: The instruction specifies a register pair which contains the memory address where the data are located .(the high-order bits of the address are in the first register of the pair and the low order bits in the second).

Immediate: The instruction contains the data itself . This is either and 8-bit quantity or a 16-bit quantity (least significant byte first , most significant byte second).

Unless directed by an interrupt or branch instruction the execution of instructions proceeds through consecutively increasing memory locations.

A branch instruction can specify the address of the next instruction to be executed in one of two ways -

Direct: The branch instruction contains the address of the next instruction to be executed .

REFERENCE :

1. COMPUTER SYSTEM ARCHITECTURE, MORRIS M. MANO, 3RD EDITION, PRENTICE HALL INDIA.
2. HTTP://NPTEL.AC.IN/COURSES

Chapter – 8
Multiprocessors

8.1 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 (IOP).
- Multiprocessors are classified as *multiple instruction stream, multiple data stream* (MIMD) systems
- 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.
- Multiprocessing improves the reliability of the system.
- 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.
- 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.
- 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*

8.2 Interconnection Structures

- The components that form a multiprocessor system are CPUs, IOPs connected to input-output devices, and a memory unit.
- 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
  - Among the processing elements in a loosely coupled system
- There are several physical forms available for establishing an interconnection network.
  - Time-shared common bus
  - Multiport memory
  - Crossbar switch
  - Multistage switching network
  - 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.
- *Disadv.*:
  - Only one processor can communicate with the memory or another processor at any given time.
  - As a consequence, the total overall transfer rate within the system is limited by the speed of the single path
- A more economical implementation of a dual bus structure is depicted in Fig. below.
- Part of the local memory may be designed as a *cache memory* attached to the CPU.

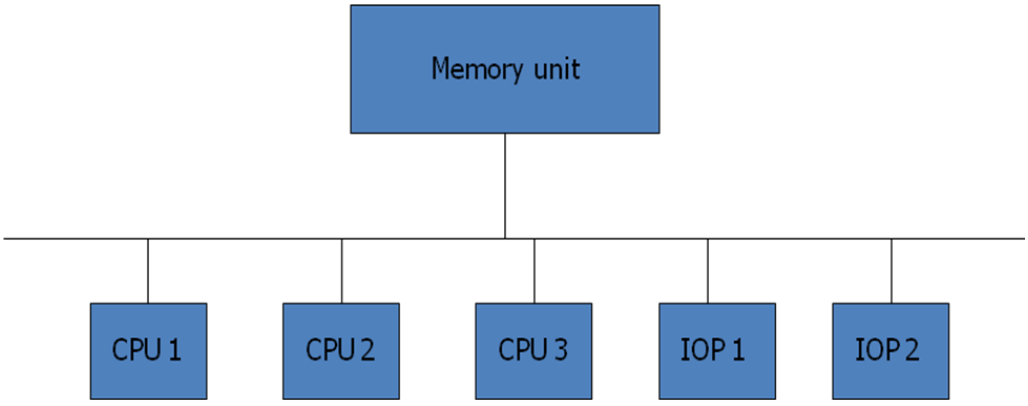


Fig: Time shared common bus organization

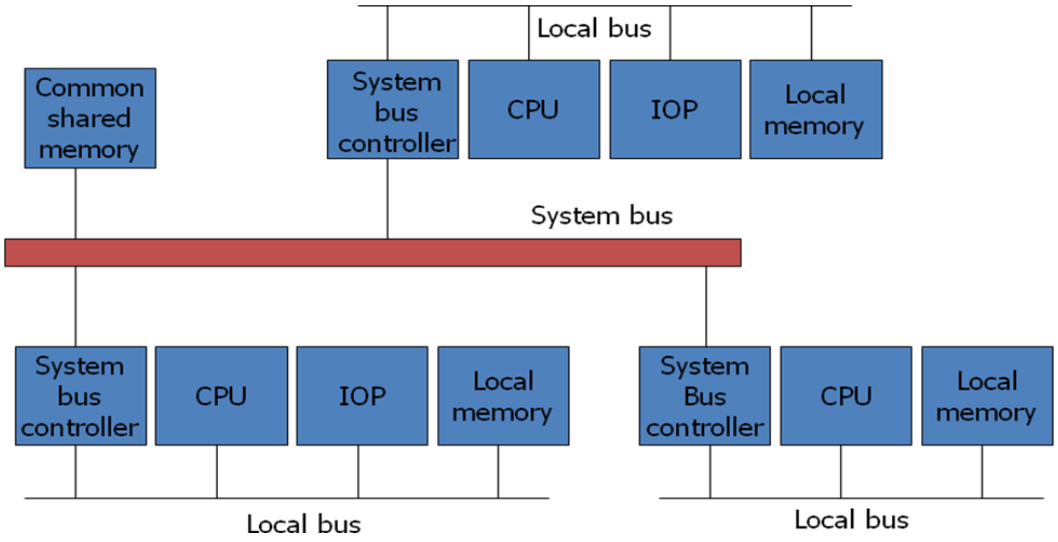


Fig: System bus structure for multiprocessorsa

Multiport Memory

- A multiport memory system employs separate buses between each memory module and each CPU.
- The module must have internal control logic to determine which port will have access to memory at any given time.
- Memory access conflicts are resolved by assigning fixed priorities to each memory port.
- *Adv.*:
  - The high transfer rate can be achieved because of the multiple paths.
- *Disadv.*:
  - It requires expensive memory control logic and a large number of cables and connections

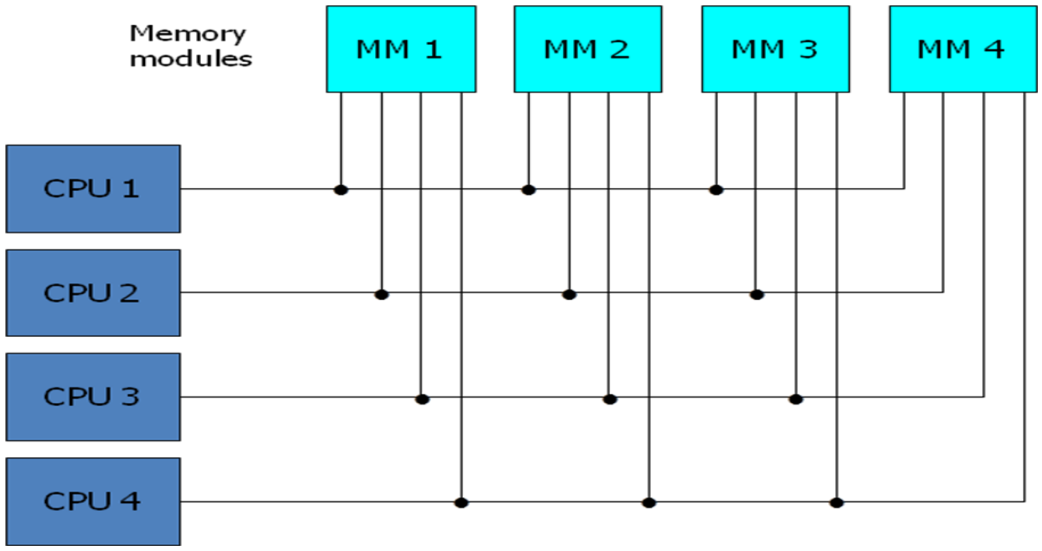


Fig: Multiport memory organization

Crossbar Switch

- Consists of a number of *crosspoints* that are placed at intersections between processor buses and memory module paths.
- The small square in each crosspoint is a *switch* that determines the path from a processor to a memory module.
- Adv.:
  - Supports simultaneous transfers from all memory modules
- Disadv.:
  - The hardware required to implement the switch can become quite large and complex.
- Below fig. shows the functional design of a crossbar switch connected to one memory module.

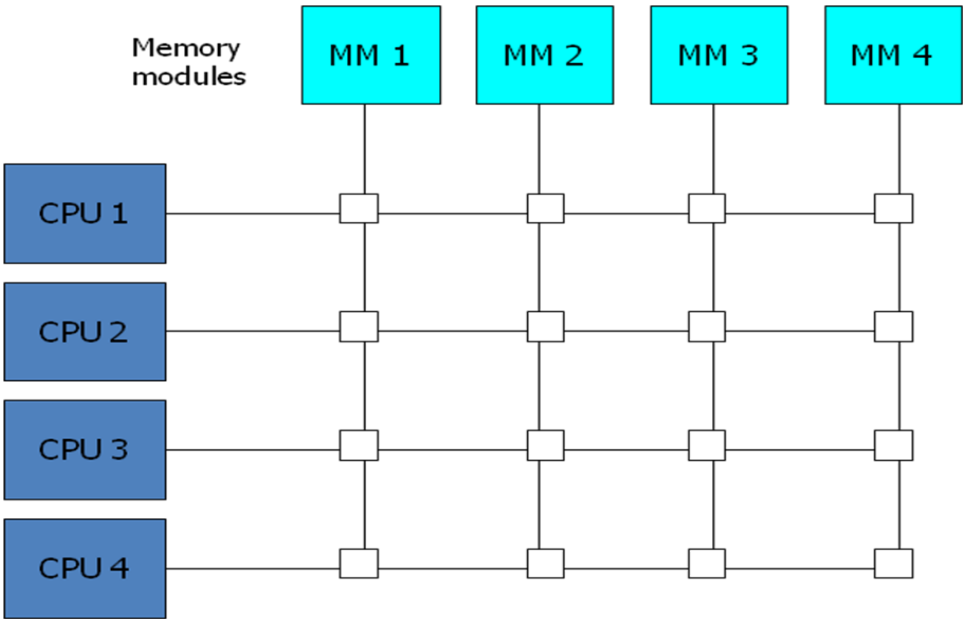


Fig: Crossbar switch

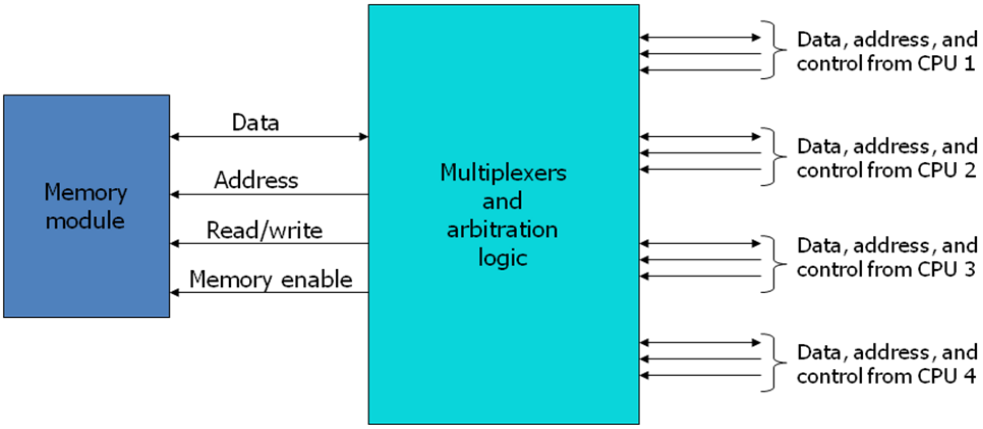
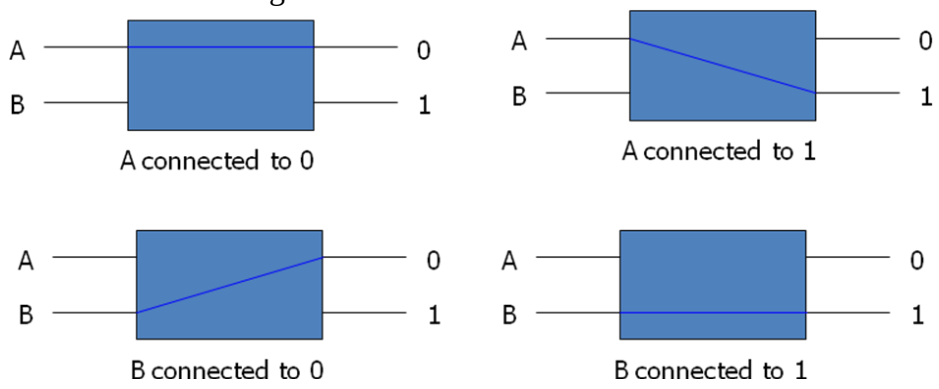


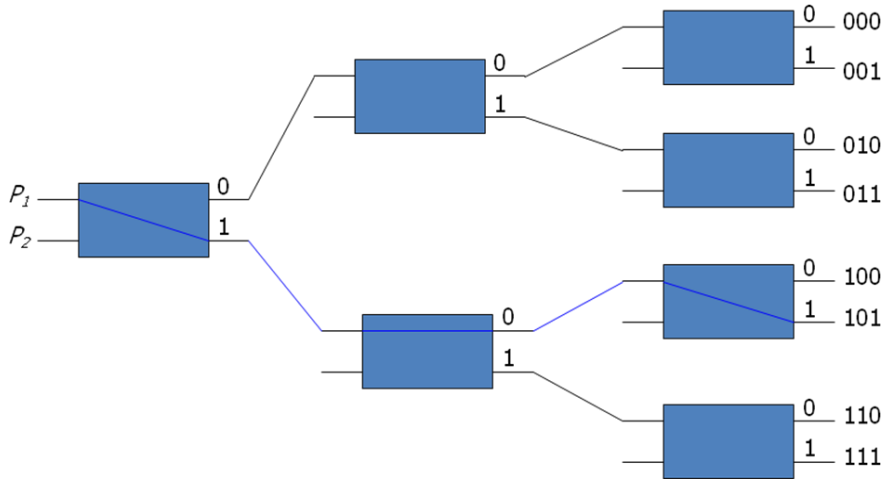
Fig: Block diagram of crossbar switch

Multistage Switching Network

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



- 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.
  - Certain request patterns cannot be satisfied simultaneously. i.e., if P_1 → 000~011, then P_2 → 100~111



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

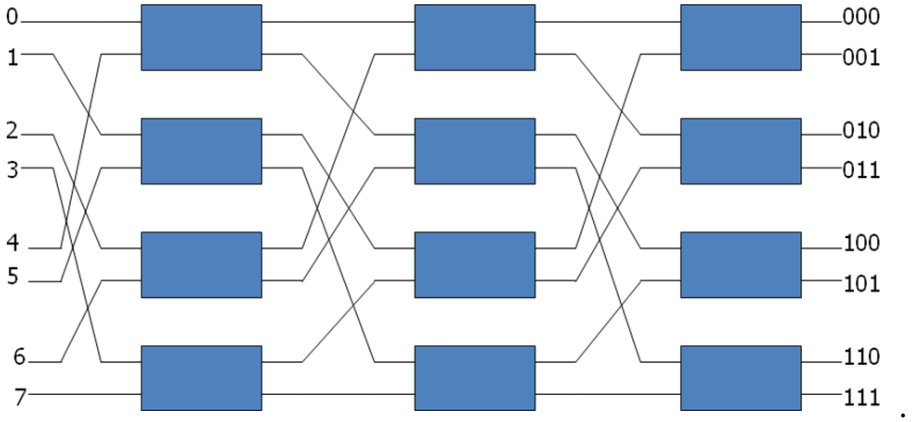


Fig: 8 x 8 Omega Switching Network

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

Hypercube System

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

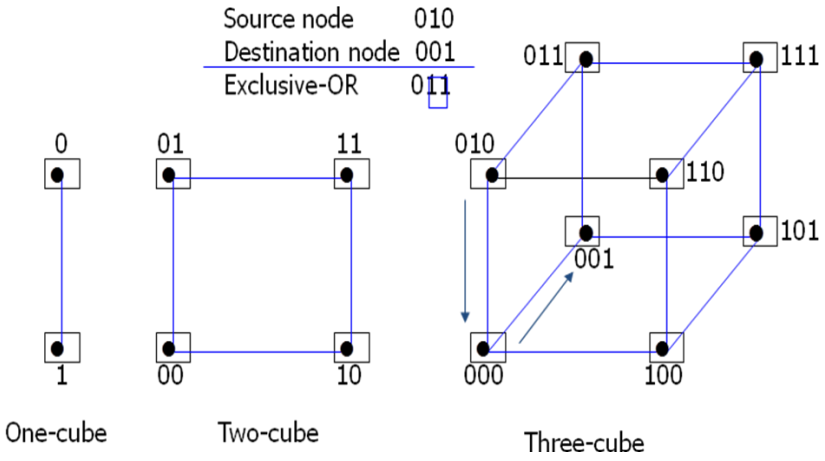


Fig: Hypercube structures for n=1,2,3

8.3 Inter processor 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 *a portion of memory* or *a common input-output channels*.
- The sending processor structures a request, a message, or a procedure, and places it in the memory mailbox.
    - *Status bits* residing in common memory
    - The receiving processor can check the mailbox *periodically*.
    - The response time of this procedure can be time consuming.
- A more efficient procedure is for the sending processor to alert the receiving processor directly by means of an *interrupt signal*.
- In addition to shared memory, a multiprocessor system may have other shared resources. e.g., a magnetic disk storage unit.
- To prevent conflicting use of shared resources by several processors there must be a provision for assigning resources to processors. i.e., 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, master, always executes the operating system functions.
- 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*.
- 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

- There is *no shared memory* for passing information.
- The communication between processors is by means of message passing through *I/O 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.
- 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

- 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.

- 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.
        - A binary semaphore

Mutual Exclusion with Semaphore

- 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.
    - Critical section: is a program sequence that 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.
- Testing and setting the semaphore is itself a critical operation and must be performed as a single indivisible operation.
- A semaphore can be initialized by means of a *test and set instruction* in conjunction with a hardware *lock* mechanism.
- 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
- Note that the lock signal must be active during the execution of the test-and-set instruction.

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

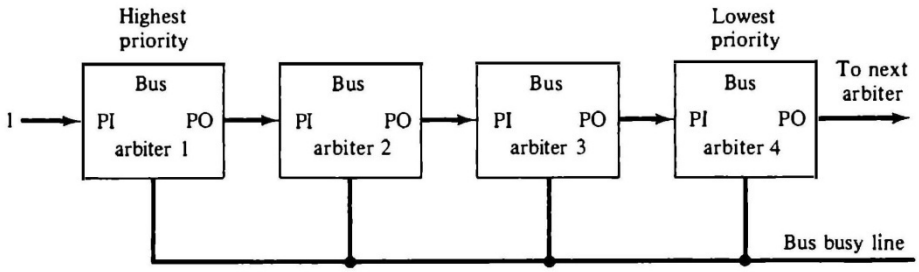
- ✓ A typical system bus consists of approximately 100 signal lines. These lines are divided into three functional groups: data, address, and control. In addition, there are power distribution lines that supply power to the components.
- ✓ For example, the IEEE standard 796 multibus system has 16 data 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

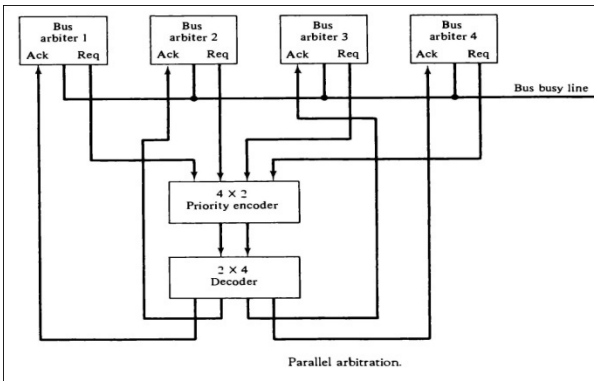
- ✓ Arbitration procedures service all processor requests on the basis of established priorities. A hardware bus priority resolving technique can be established by means of a serial or parallel connection of the units requesting control of the system bus.
- ✓ The serial priority resolving technique is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