SKDAV GOVT. POLYTECHNIC ROURKELA

COMPUTER SYSTEM & ARCHITECTURE

PREPARED BY
BIJAYALAXMI PADHIARY
SKDAV GOVT. POLYTECHNIC, ROURKELA

Th-1 COMPUTER SYSTEM ARCHITECTURE

Common to (CSE/IT)

Theory	**4 Periods per week**	**Internal Assessment**	**20 Marks**
Total Periods	**60 Periods**	**End Sem Exam**	**80 Marks**
Examination	**3hours**	**Total Marks**	**100Marks**

A. Topic wise distribution of periods

Sl. No.	Topics	Periods
1	BASIC STRUCTURE OF COMPUTER HARDWARE	06
2	INSTRUCTIONS & INSTRUCTION SEQUENCING	07
3	PROCESSOR SYSTEM	10
4	MEMORY SYSTEM	10
5	INPUT – OUTPUT SYSTEM	10
6	I/O INTERFACE & BUS ARCHITECTURE	10
7	PARALLEL PROCESSING	07
9	**TOTAL**	**60**

B. RATIONAL: Now a days the usage of computer has become very essential in various areas like education, entertainment, business, sports etc. This subject will expose the learner to have an idea about the architecture of different components of a computer system and their operation procedure. Further the learner will have idea how the different components integrate to execute a task to get the result. It also gives an idea how to improve the processing capability.

C. OBJECTIVE: After completion of this course the student will be able to**:**

- Understand the basic structure of a computer with instructions.
- Learn about machine instructions and program execution.
- Learn about the internal functional units of a processor and how they are interconnected.
- Understand how I/O transfer is performed.
- Learn about basic memory circuit, organization and secondary storage.
- Understand concept of parallel processing.

D. COURSE CONTENTS:

1. Basic structure of computer hardware

1.1Basic Structure of computer hardware
1.2Functional Units
1.3Computer components
1.4 Performance measures
1.5 Memory addressing & Operations

2. Instructions & instruction Sequencing

2.1 Fundamentals to instructions
2.2 Operands
2.3 Op Codes
2.4 Instruction formats
2.5 Addressing Modes

3. Processor System

3.1 Register Files
3.2 Complete instruction execution

- Fetch

- Decode
- Execution

3.3 Hardware control
3.4 Micro program control

4. Memory System

4.1Memory characteristics
4.2 Memory hierarchy
4.3 RAM and ROM organization
4.4 Interleaved Memory
4.5 Cache memory
4.6 Virtual memory

5. Input – Output System

5.1 Input - Output Interface
5.2 Modes of Data transfer
5.3 Programmed I/O Transfer
5.4 Interrupt driven I/O
5.5 DMA
5.6 I/O Processor

6. I/O Interface & Bus architecture

6.1 Bus and System Bus
6.2 Types of System Bus
- Data
- Address
- Control

7.1 Bus

6.3 Bus Structure
6.4 Basic Parameters of Bus design
6.5 SCSI
6.6 USB

7. Parallel Processing

7.1 Parallel Processing
7.2 Linear Pipeline
7.3 Multiprocessor
7.4 Flynn's Classification

Coverage of Syllabus upto Internal Exams (I.A.)
Chapter 1,2,3,4

Book Recommended:-

Sl.No	Name of Authors	Title of the Book	Name of the Publisher
1	Moris Mano	Computer System Architecture	PHI
2	Er. Rajeev Chopra	Computer Architecture and Organisation	S.Chand
3	Parthasarthy, Senthil Kumar	Fundamentals of Computer Architecture	TMH

UNIT -1

BASIC STRUCTURE OF COMPUTER

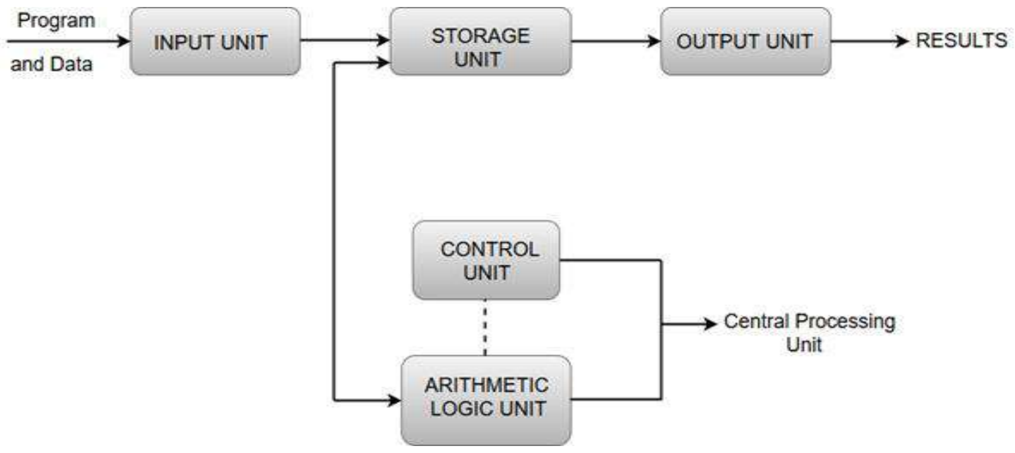
BASIC STRUCTURE OF COMPUTER HARDWARE

- A computer organization describes the functions and design of the various units of a digital system.
- Computer architecture deals with the specification of the instruction set and the hardware units that implement the instructions.
- Computer hardware consists of electronic circuits, displays, magnetic and optic storage media and also the communication facilities.

  Functional units are a part of a CPU that performs the operations and calculations called by the computer program.

FUNCTIONAL UNITS OF DIGITAL SYSTEM

- Functional units of a computer system are parts of the CPU (Central Processing Unit) that performs the operations and calculations called for by the computer program. A computer consists of five main components namely, Input unit, Central Processing Unit, Memory unit Arithmetic & logical unit, Control unit and an Output unit.



INPUT UNIT

- Input units are used by the computer to read the data. The most commonly used input devices are keyboards, mouse, joysticks, trackballs, microphones, etc.
- However, the most well-known input device is a keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

CENTRAL PROCESSING UNIT

- Central processing unit commonly known as CPU can be referred as an electronic circuitry within a computer that carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

MEMORY UNIT

- The Memory unit can be referred to as the storage area in which programs are kept which are running, and that contains data needed by the running programs.
- The Memory unit can be categorized in two ways namely, primary memory and secondary memory.
- It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.
- Primary storage is the fastest memory that operates at electronic speeds. Primary memory contains a large number of semiconductor storage cells, capable of storing a bit of information. The word length of a computer is between 16-64 bits.
- It is also known as the volatile form of memory, means when the computer is shut down, anything contained in RAM is lost.
- Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor.
- The most common examples of primary memory are RAM and ROM.
- Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.

- It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.
- The most common examples of secondary memory are magnetic disks, magnetic tapes, and optical disks.

ARITHMETIC & LOGICAL UNIT

- Most of all the arithmetic and logical operations of a computer are executed in the ALU (Arithmetic and Logical Unit) of the processor. It performs arithmetic operations like addition, subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

CONTROL UNIT

- The control unit is a component of a computer's central processing unit that coordinates the operation of the processor. It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.
- The control unit is also known as the nerve center of a computer system.
- Let's us consider an example of addition of two operands by the instruction given as Add LOCA, RO. This instruction adds the memory location LOCA to the operand in the register RO and places the sum in the register RO. This instruction internally performs several steps.

OUTPUT UNIT

- The primary function of the output unit is to send the processed results to the user. Output devices display information in a way that the user can understand.
- Output devices are pieces of equipment that are used to generate information or any other response processed by the computer. These devices display information that has been held or generated within a computer.
- The most common example of an output device is a monitor.

BASIC OPERATION CONCEPT:

- To perform a given task on the computer, and appropriate program is stored in the memory.

- Individual instructions are brought from the memory into the processor which executes the specified operations.
- Data to be used as the operand are also stored in memory , this instruction requires the performance of several steps:
    - ❖ The operation is performed using the decoded instruction and the operant.
    - ❖ The result is stored back in the memory.
- Transfer of data between the memory and processor takes place sending the address of the memory on the address box and the appropriate control signal on the control bus.
- The processor has no of register which are used for different purpose:
    - ❖ INSTRUCTION REGISTER: it contains the instruction that is being currently executed.
    - ❖ PROGRAM COUNTER: it contains the memory address of next instruction.
    - ❖ GENERAL PURPOSE REGISTER: it is identified by R_0-R_{N-1}
    - ❖ MEMORY ADDRESS REGISTER: it contains the address of the memory location to be accessed.
    - ❖ MEMORY DATA REGISTER: it contains the data to be read / write from the location.

COMPUTER COMPONENTS:

- The main memory stored both data and instruction in binary many ways to store binary data to perform operation on that data.
- A configuration of logic component can be constructed to perform the particular computation. This can be thought of as a form programming use in hardware and named as **“hardware program”.**
- The general-purpose hardware Interprets each instruction and generate the control signal. This is called” **software or microprogrammed control”**

.

PERFORMANCE MEASURES: -

Performance is the ability of the computer to quickly execute a program.

- The speed at which the computer executes a program is decided by the design of its hardware and machine language instruction.
- Computer performance measures is of very big term when used in context of the computer system.
- System that execute program in less time are called to have higher performance.

Basic performance measures:-

The speed of operation of a system is generally decided by two fractions.

i) Response time

ii) Throughput.

Response Time:-

- Response time is the time spends to complete an event or an operation.
- It is also called as execution time or latency.

Throughput:-

- Throughput is the amount of work done per unit of time. i.e. the amount of processing that can be accomplished during a given interval of time.
- It is also called as bandwidth of the system.
- In general, faster response time leads to better throughput.

Elapsed time:-

- Elapsed time is a time spent from the start of execution of the program to its completion is called elapsed time.
- This performance measure is affected by the clock speed of the processor and the concerned input output device.

Percentage of elapsed time = (user CPU time + system CPU time) / elapsed time

MIPS

- A nearly measure of computer performance has the rate at which a given machine executed instruction.
- This is calculated by dividing the no. of instruction and the time required to run the program

CPI/IPC

- CPI – Clock cycle per Instruction
- IPC – Instruction per cycle.
- It is another measuring that which is calculated as the number of clock cycle required to execute one instruction (cycle per instruction) by the instruction executed per cycle.

Speed up:-

- Computer architecture use the speed up to describe the performance of architectural charges as different improvement is made to the system.
- It is defined as ratio of execution time before to the execution time after the charge.

Speed up = execution time before/Execution time after
= performance of system after /Execution time before

Amdahl's law:-

- This law states that "performance improvement to be gained by using a faster mode of execution is limited by the fraction of time the faster made can be used".
- Amdahl's law defines the term speed up.

Speed up = performance of entire task using enhancement /Performance of in time task without using enhancement

Performance = 1/Execution time

Speed up = execution time without using enhancement /Execution time with using enhancement

Performance parameter

The basic performance equation is given by

$$\boxed{T = (N \times S) / R}$$

Where

T – Performance parameter of an application program.

N – No. of instruction required to complete the execution of a program.

R - Clock rate of the processor in cycles per second.

S - Avg. no. of basic step required to execute one machine instruction.

Clock rate:-

Clock rate is one of the important performance measures by improving in the clock rate. There are two ways in which clock rate may be increased.

1) Improving IC technology which makes logic circuits faster thus reading time taken to complete a basic step.

2) By reducing the processing amount in one basic step which by reduces the clock period as

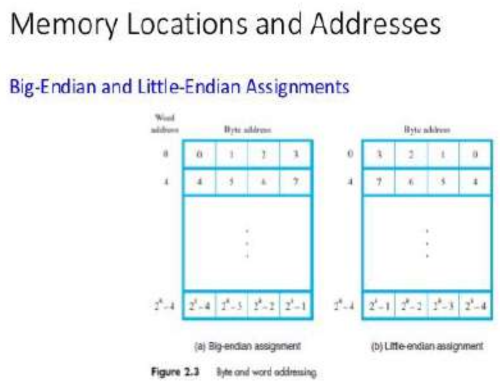
$$R = 1 / T$$

CPU performance Equation:-

Normally the CPUs are constructed by using a clock running at a constant rate. This discrete time events are known as a **clock cycle.**

CPU time = the time of a program may be represented as
= CPU clock cycle for a program X clock cycle time
or
= CPU clock cycle/ clock rate
= instruction count X CPU
= IC X CPI X C

MEMORY ADDRESSING AND MEMORY OPERATION


Memory Locations and Addresses

Big-Endian and Little-Endian Assignments

Figure 2.3 Byte and word addressing

- Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address).
- Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first.
- For example, in a big-endian computer, the two bytes required for the hexadecimal number 4F52 would be stored as 4F52 in storage (if 4F is stored at storage address 1000, for example, 52 will be at address 1001).
- In a little-endian system, it would be stored as 524F (52 at address 1000, 4F at 1001).
- For people who use languages that read left-to-right, big endian seems like the natural way to think of a storing a string of characters or numbers - in the same order you expect to see it presented to you.
- Many of us would thus think of big-endian as storing something in *forward* fashion, just as we read.
- An argument for little-endian order is that as you increase a numeric value, you may need to add digits to the left (a higher non-exponential number has more digits).
- Thus, an addition of two numbers often requires moving all the digits of a big-endian ordered number in storage, moving everything to the right.

- In a number stored in little-endian fashion, the least significant bytes can stay where they are and new digits can be added to the right at a higher address. This means that some computer operations may be simpler and faster to perform.
- Note that within both big-endian and little-endian byte orders, the bits within each byte are big-endian. That is, there is no attempt to be big- or little-endian about the entire bit stream represented by a given number of stored bytes.
- For example, whether hexadecimal 4F is put in storage first or last with other bytes in a given storage address range, the bit order within the byte will be:
- IBM's 370 mainframes, most RISC-based computers, and Motorola microprocessors use the big-endian approach. TCP/IP also uses the big-endian approach (and thus big-endian is sometimes called *network order*).
- On the other hand, Intel processors (CPUs) and DEC Alphas and at least some programs that run on them are little-endian.
- There are also mixed forms of endianness. For example, VAX floating point uses mixed-endian (also referred to as middle-endian).
- The ordering of bytes in a 16-bit word differs from the ordering of 16-bit words within a 32-bit word.
- Bi-endian processors can operate in either little-endian or big-endian mode, and switch between the two.

Read and Write operations in Memory

- A memory unit stores binary information in groups of bits called words.
- Data input lines provide the information to be stored into the memory, Data output lines carry the information out from the memory.
- The control lines Read and write specifies the direction of transfer of data.

  Basically, in the memory organization, there are 2^l memory locations indexing from 0 to 2^{l-1} where l is the address buses. We can describe the memory in terms of the bytes using the following formula:

$$\boxed{N = 2^l Bytes}$$

Where,

I is the total address buses

N is the memory in bytes

For example, some storage can be described below in terms of bytes using the above formula:

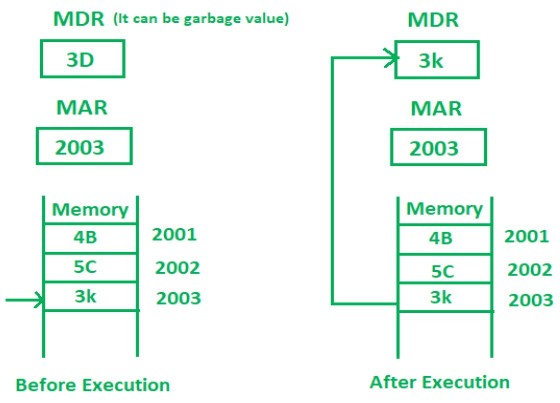
1kB= 2^{10} Bytes
64 kB = 2^6 x 2^{10} Bytes
= 2^{16} Bytes
4 GB = 2^2 x 2^{10}(kB) x 2^{10}(MB) x 2^{10} (GB)
= 2^{32} Bytes

Memory Address Register (MAR) is the address register which is used to store the address of the memory location where the operation is being performed.

Memory Data Register (MDR) is the data register which is used to store the data on which the operation is being performed.

1. **Memory Read Operation:**

- Memory read operation transfers the desired word to address lines and activates the read control line. Description of memory read operation is given below:



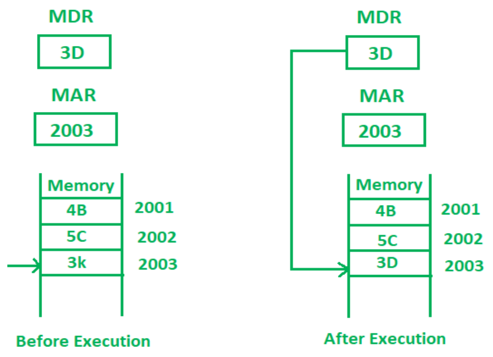
Memory Read Operation

- In the above diagram initially, MDR can contain any garbage value and MAR is containing 2003 memory address.

- After the execution of read instruction, the data of memory location 2003 will be read and the MDR will get updated by the value of the 2003 memory location (3D).

2. Memory Write Operation:

- Memory write operation transfers the address of the desired word to the address lines, transfers the data bits to be stored in memory to the data input lines. Then it activates the write control line.
- Description of the write operation is given below:



Memory Write Operation

- In the above diagram, the MAR contains 2003 and MDR contains 3D. After the execution of write instruction 3D will be written at 2003 memory location.

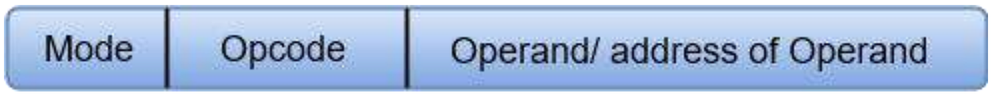
UNIT-2
INSTRUCTIONS & INSTRUCTION SEQUENCING

FUNDAMENTAL TO INSTRUCTION

Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

An instruction comprises of groups called fields. These fields include:

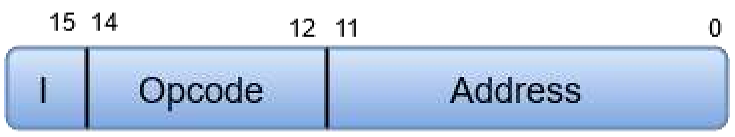
- The Operation code (Opcode) field which specifies the operation to be performed.
- The Address field which contains the location of the operand, i.e., register or memory location.
- The Mode field which specifies how the operand will be located.



A basic computer has three instruction code formats which are:

1. Memory - reference instruction
2. Register - reference instruction
3. Input-Output instruction

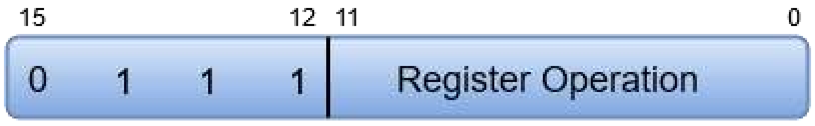
Memory - reference instruction



(Opcode = 000 through 110)

In Memory-reference instruction, 12 bits of memory is used to specify an address and one bit to specify the addressing mode 'I'.

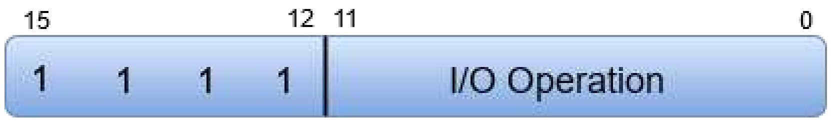
Register - reference instruction



(Opcode = 111, I = 0)

The Register-reference instructions are represented by the Opcode 111 with a 0 in the leftmost bit (bit 15) of the instruction.

Input-Output instruction



(Opcode = 111, I = 1)

Just like the Register-reference instruction, an Input-Output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of the input-output operation or test performed.

Note

- The three operation code bits in positions 12 through 14 should be equal to 111. Otherwise, the instruction is a memory-reference type, and the bit in position 15 is taken as the addressing mode I.
- When the three operation code bits are equal to 111, control unit inspects the bit in position 15. If the bit is 0, the instruction is a register-reference type. Otherwise, the instruction is an input-output type having bit 1 at position 15.

Instruction Set Completeness:-

A set of instructions is said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

- Arithmetic, logical and shift instructions
- A set of instructions for moving information to and from memory and processor registers.
- Instructions which controls the program together with instructions that check status conditions.
- Input and Output instructions

Arithmetic, logic and shift instructions provide computational capabilities for processing the type of data the user may wish to employ.

A huge amount of binary information is stored in the memory unit, but all computations are done in processor registers. Therefore, one must possess the capability of moving information between these two units.

Program control instructions such as branch instructions are used change the sequence in which the program is executed.

Input and Output instructions act as an interface between the computer and the user. Programs and data must be transferred into memory, and the results of computations must be transferred back to the user.

OPERANDS:-

Machine instructions operate on data. The most important general categories of data are:

- Addresses
- Numbers
- Characters

- Logical data

1. **Address:-** Address are the for of number that represent specific location in memory.
2. **Numbers:-** All machine languages include numeric data types. Even in nonnumeric data processing, there is a need for numbers to act as counters, field widths, and so forth. An important distinction between numbers used in ordinary mathematics and numbers stored in a computer is that the latter are limited. Thus, the programmer is faced with understanding the consequences of rounding, overflow and underflow.

   Three types of numerical data are common in computers:

   - Integer or fixed point
   - Floating point
   - Decimal

3. **Characters:-** A common form of data is text or character strings. the most commonly used character code in the International Reference Alphabet (IRA), referred to in the United States as the American Standard Code for Information Interchange (ASCII). Another code used to encode characters is the Extended Binary Coded Decimal Interchange Code (EBCDIC). EBCDIC is used on IBM S/390 machines. It is an 8-bit code. As with IRA, EBCDIC is compatible with packed decimal. In the case of EBCDIC, the codes 11110000 through 11111001 represent the digits 0 through 9.
4. **Logical data:** Normally, each word or other addressable unit (byte, half-word, and soon) is treated as a single unit of data. It is sometimes useful, however, to consider an n-bit unit as consisting 1-bit items of data, each item having the value 0 or I. When data are viewed this way, they are considered to be logic data.

OP CODES:-

The number of different opcodes varies widely from machine to machine. However, the same general types of operations are found on all machines. A useful and typical categorization is the following:

1) Data transfer
2) Arithmetic
3) Logical
4) Conversion
5) I/O
6) System control
7) Transfer of control

1) Data transfer:

The most fundamental type of machine instruction is the data transfer instruction. The data transfer instruction must specify several things.

- The location of the source and destination operands must be specified. Each location could be memory. a register, or the lop of the stack.
- The length of data to be transferred must be indicated.
- As with all instructions with operands, the mode of addressing for each operand must be specified.

In term of CPU action, data transfer operations are perhaps the simplest type. If both source and destination are registers, then the CPU simply causes data to be transferred from one register to another; this is an operation internal to the CPU. If one or both operands are in memory, then (he CPU must perform some or all of following actions:

1. Calculate the memory address, based on the address mode

2. If the address refers to virtual memory, translate from virtual to actual memory address.

3. Determine whether the addressed item is in cache.

4. If not, issue a command lo the memory module.

2) Arithmetic:

Most machines provide the basic arithmetic operations of add, subtract, multiply, and divide. These are invariably provided for signed integer (fixed-point) numbers, Often they are also provided for floating-point and packed decimal numbers.

Other possible operations include a variety of single-operand instructions: for example.

• Absolute: Take the absolute value of the operand.

• Negate: Negate the Operand.

• Increment.: Add 1 to the operand.

• Decrement: Subtract 1 from the operand.

3) Logical:

Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units, often referred to as "bit twiddling." They are based upon Boolean operations.

Some of the basic logical operations that can be performed on Boolean or binary data are AND, OR, NOT, XOR, …

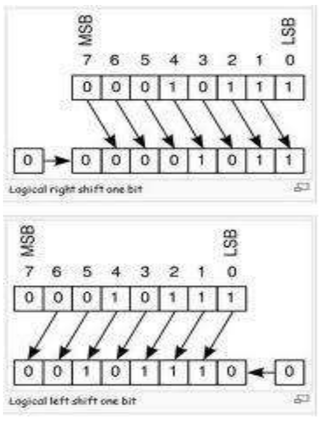
These logical operations can be applied bitwise to n-bit logical data units. Thus, if two registers contain the data

(R1) - 10100101 (R2) - 00001111

then

(R1) AND (R2) – 00000101

In addition to bitwise logical operations, most machines provide a variety of shifting and rotating functions such as shift left, shift right, right rotate, left rotate.....



4) Conversion:

Conversion instructions are those that change the format or operate on the format of data. An example is converting from decimal to binary.

5) Input/Output:

As we saw, there are a variety of approaches taken, including isolated programmed IO, memory-mapped programmed I/O, DMA, and the use of an I/O processor. Many implementations provide only a few I/O instructions, with the specific actions specified by parameters, codes, or command words.

6) System Controls:

System control instructions are those that can he executed only while the processor is in a certain privileged state or is executing a program in a special privileged area of memory, typically, these instructions are reserved for the use of the operating system.

Some examples of system control operations are as follows.

- A system control instruction may read or alter a control register.

- Another example is an instruction to read or modify a storage protection key, such us is used in the S/390 memory system.
- Another example is access to process control blocks in a multiprogramming system.

7) Transfer of control:

For all of the operation types discussed so far. The next instruction to be performed is the one that immediately follows, in memory, the current instruction. However, a significant fraction of the instructions in any program have as their function changing the sequence of instruction execution. For these instructions, the operation performed by the CPU is to update the program counter to contain the address of some instruction in memory.

INSTRUCTION FORMATS:-

Computer perform task on the basis of instruction provided. A instruction in computer comprises of groups called fields. These fields contains different information as for computers everything is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:

- **Operation field** which specifies the operation to be performed like addition.
- **Address field** which contain the location of operand, i.e., register or memory location.
- **Mode field** which specifies how operand is to be founded.

An instruction is of various lengths depending upon the number of addresses it contains. Generally CPU organizations are of three types on the basis of number of address fields:

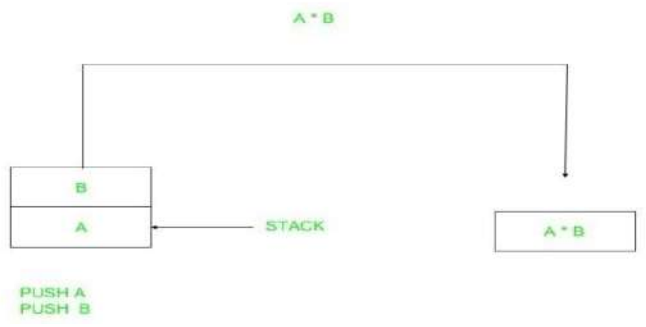
1. Single Accumulator organization
2. General register organization
3. Stack organization

In first organization operation is done involving a special register called accumulator. In second on multiple registers are used for the computation purpose. In third organization the work on stack basis operation due to which it does not contain any address field. It is not necessary that only a single organization is applied a blend of various organization is mostly what we see generally.

On the basis of number of address instruction are classified as:

Note that we will use X = (A+B)*(C+D) expression to showcase the procedure.

- ❖ **Zero Address Instructions : –**



A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

Expression: X = (A+B)*(C+D)
Post fixed : X = AB+CD+*
TOP means top of stack
M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)

POP	X	M[X] = TOP

- **One Address Instructions –**

This use a implied ACCUMULATOR register for data manipulation. One operand is in accumulator and other is in register or memory location. Implied means that the CPU already knows that one operand is in accumulator so there is no need to specify it.

opcode	operand/address of operand	mode

Expression: X = (A+B)*(C+D)
AC is accumulator
M[] is any memory location
M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

- **Two Address Instructions –**

This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

opcode	Destination address	Source address	mode

Here destination address can also contain operand.

Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

- **Three Address Instructions –**

This has three address fields to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

opcode	Destination address	Source address	Source address	mode

Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location

ADD	R1, A, B	R1 = M[A] + M[B]
ADD	R2, C, D	R2 = M[C] + M[D]
MUL	X, R1, R2	M[X] = R1 * R2

ADDRESSING MODES

- The addressing modes is a really important topic to be considered in microprocessor or computer organization. **The addressing modes in computer architecture actually define how an operand is chosen to execute an instruction**. It is the way that is used to identify the location of an operand which is specified in an instruction.
- Whenever an instruction executes, it requires operands to be operated on. An instruction field consisting of opcode and operand. Where operand means the data and opcode means the instruction itself. In case of operations like addition or subtraction, they require two data. So, they are called binary instruction. On the other hand, the increment or decrement operations need only one data and are so called unary instruction. Now the question is **how these data can be obtained.**

Addressing modes in computer architecture:-

- **Implicit**

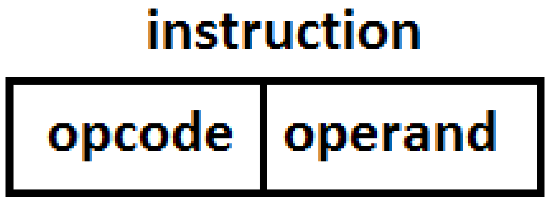
- **Immediate**
- **Direct**
- **Indirect**
- **Register**
- **Register Indirect**
- **Displacement**
    - **Relative**
    - **Base register**
    - **Indexing**
- **Stack**

Implicit addressing mode:-

- The term implicit addressing mode means here we are not mentioning clearly in details that from where the instruction can get the operand. But by default, the instruction itself knows from where it is supposed to access the operand.
- For example, **CMA** stands for complement accumulator. The meaning of the **CMA** instruction is whatever the value present in the accumulator will be replaced by its 1's complement.
- In this instruction **CMA** or along with this instruction, we are not mentioning any operand. So here it knows that the operand has to be accessed from the accumulator implicitly. This is known as implicit addressing modes.

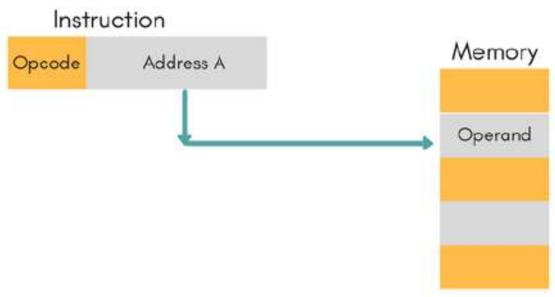
Immediate addressing mode :

- In the immediate addressing mode, the instruction contains two fields. One for the **opcode** and another field contains the **operand** itself. That means in this addressing mode, there is no need to go anywhere to access the operand because of the instruction itself containing the operand. This is known as immediate addressing mode.



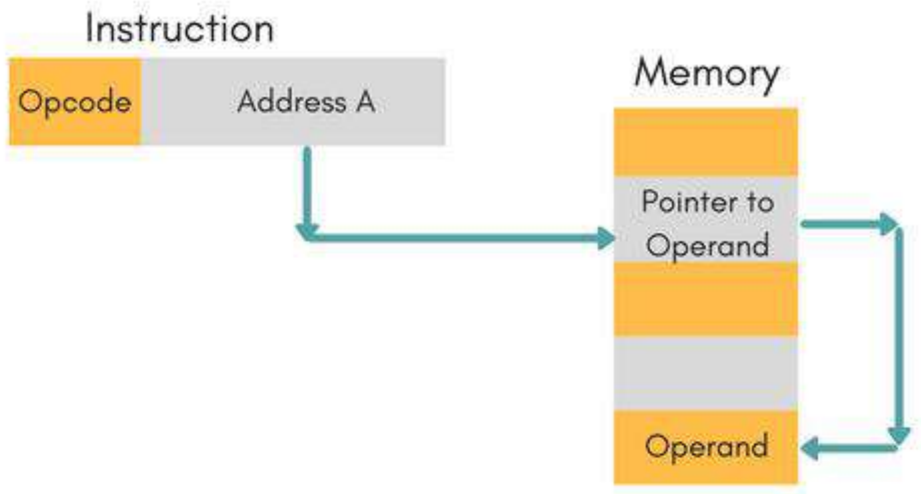
Direct addressing mode :

- In the direct addressing mode, the instruction will have the two parts. One part will contain the **opcode** and another one will contain the **address of the memory location** at where the operand can be found.
- Here **A** is the address of the operand. That means at the **Ath** location in the memory, the operand can be found.



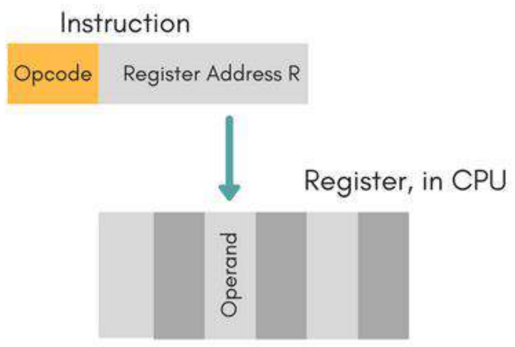
Indirect addressing mode :

- Indirect addressing mode contains the **opcode** and **address field**. But unlike direct addressing mode, it doesn’t contain the address of the operand but contains the address of a memory location in which the actual address of the operand can be found.



Register addressing mode:

- In case of register addressing mode, the instruction will have the **opcode** and a **register number**. Depending upon the register number, one of the registers will be selected from the available sets of registers by default automatically.
- The unique identification of the register can be done by the register number which is mentioned in the instruction. In that register, the operand can be found.



Advantages

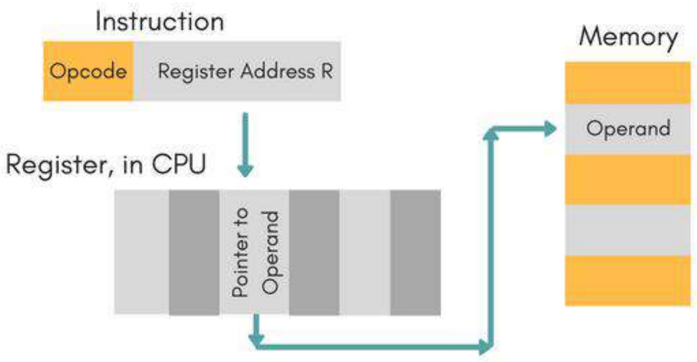
- Shorter instructions and faster instruction fetch.
- Faster memory access to the operand(s)

Disadvantages

- Very limited address space
- Using multiple registers helps performance but it complicates the instructions.

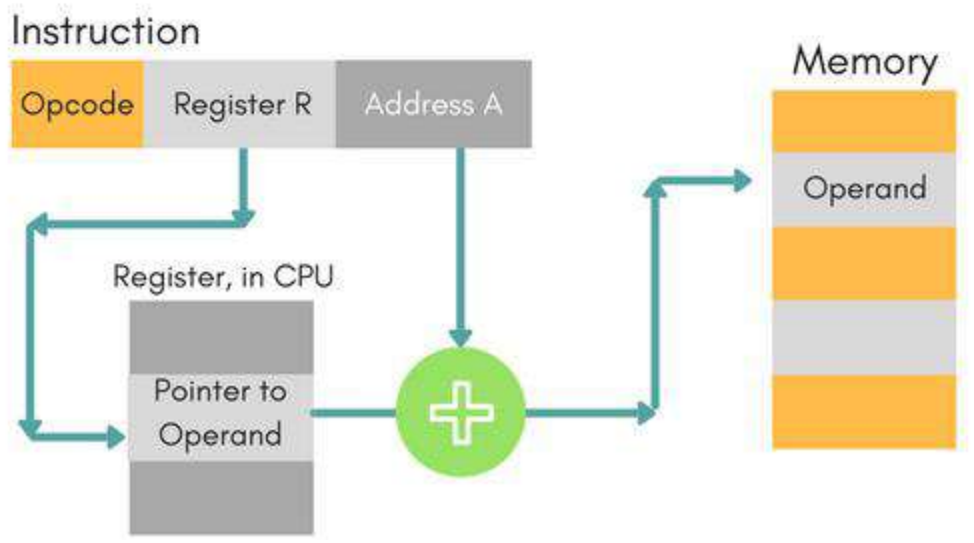
Register indirect addressing mode:

- In the register indirect addressing mode, the instruction will contain the **opcode** as well as a **register number**. Depending upon the register number mentioned in the instruction, the corresponding register will be accessed from the set of registers. But here the register doesn't contain the operand but will contain the address of the operand in the memory at where the operand can be found.



Displacement addressing mode :

- In the displacement addressing mode, the instruction will be having three fields. One for the **opcode**, one for the **register number** and the remaining one for an **absolute address**.
- At first, depending upon the register number the register will be selected from the register set. After that its content will be added with the absolute address and the new address formed will be the actual physical address of the operand in the memory.



- Displacement addressing mode in computer architecture can be categorized into 3 different modes.
  - ✓ **Relative**
  - ✓ **Base register**
  - ✓ **Indexing**

In case of **relative addressing mode,** the register used will be a **program counter**.

In the **base addressing mode**, the register will contain the **base address** and the absolute field will be the **offset or displacement** from the base address. After adding both the actual physical address of the operand can be obtained and mapping this address in the memory we can access the operand.

For example, if the base address is **3000** and the offset is **20**, then after adding both i.e. **3020** will be the actual address of the operand.

In case of **Indexing mode**, the absolute field will contain the **starting base address** of the memory block and the register field will contain the **index value**. Adding both will give the actual physical address of the operand.

Relative Addressing Mode

- It is a version of Displacement addressing mode. In this the contents of PC(Program Counter) is added to address part of instruction to obtain the effective address.
- EA = A + (PC), where EA is effective address and PC is program counter.
  The operand is A cells away from the current cell(the one pointed to by PC).

Base Register Addressing Mode

- It is again a version of Displacement addressing mode. This can be defined as

  EA = A + (R),

  where A is displacement and R holds pointer to base address.

Stack addressing mode:

- In case of stack addressing mode, the instruction knows the **topmost data should be the operand**. If the instruction is a unary instruction then it will select the topmost data as the operand and if the instruction is a binary instruction then it will select the topmost two data as the operands from the top of the stack.

UNIT-3
PROCESSOR SYSTEM

Register File

- A register file is a means of memory storage within a computer's **central processing unit** (**CPU**). The computer's register files contain bits of data and mapping locations. These locations specify certain addresses that are input components of a register file. Other inputs include data, a read and write function and execute function.
- When a user installs a program on a computer, that software application writes a register file on the CPU. Most software programs will contain more than one file. Those files contain execution instructions that the CPU follows when the user launches and uses the application. A register file also lets the CPU know where program is located and what data is needed to perform certain functions.
- Decoders are a part of a register file. When data is extracted from a register, the computer's **hard drive** refers back to the bits that are contained in the file. Part of the extraction process involves reading and deciphering the data bits that are contained in the register. Once a program completes a function, it may write a code or message indicating the results of the operation.
- Register files utilize one of two technologies related to memory. The first is known as static random access memory, or **SRAM**. With static random access memory there are several bits of memory that are labeled according to **binary** code. The status of each memory bit is labeled with a zero or one, indicating an active or inactive state.
- A second type of register memory is dynamic random access memory, or DRAM. Each section of memory contains a capacitor and **transistor**. Data values are equated with different charges and must be constantly updated by the memory chip. The update or "refreshing" will typically take up to 2 percent of the total processing time.
- There are two components to the memory chip's ability to process data. They include cycle and access time. The cycle time is the lowest amount of time that

occurs between data requests. Access time is the amount of seconds or minutes it takes for the CPU to request data from a register file and the time it takes to actually receive that information.

- While SRAM is usually used with memory caches, its cycle time and access time are the same. With DRAM technology, the cycle time is typically longer than its access time. This is because memory reads or extractions involve a destroy and re-write process.

Register Transfer Language

- The symbolic notation used to describe the micro-operation transfers amongst registers is called **Register transfer language**.
- The term **register transfer** means the availability of **hardware logic circuits** that can perform a stated micro-operation and transfer the result of the operation to the same or another register.
- The word **language** is borrowed from programmers who apply this term to programming languages. This programming language is a procedure for writing symbols to specify a given computational process.

Following are some commonly used registers:

1. **Accumulator**: This is the most common register, used to store data taken out from the memory.

2. **General Purpose Registers**: This is used to store data intermediate results during program execution. It can be accessed via assembly programming.

3. **Special Purpose Registers**: Users do not access these registers. These registers are for Computer system,

    - **MAR:** Memory Address Register are those registers that holds the address for memory unit.

- **MBR:** Memory Buffer Register stores instruction and data received from the memory and sent from the memory.
- **PC:** Program Counter points to the next instruction to be executed.
- **IR:** Instruction Register holds the instruction to be executed.

Register Transfer

- Information transferred from one register to another is designated in symbolic form by means of replacement operator.

  R2 ← R1

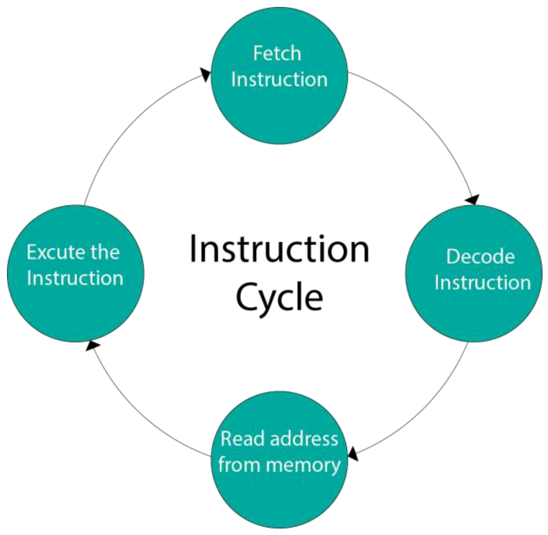
- It denotes the transfer of the data from register R1 into R2.

Instruction Cycle

A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction.

In a basic computer, each instruction cycle consists of the following phases:

1. Fetch instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory.
4. Execute the instruction.



Complete Instruction Execution

1. Fetch cycle

The address of the next instruction to execute is in the Program Counter(PC) at the beginning of the fetch cycle.

Step 1: The address in the program counter is transferred to the Memory Address Register(MAR), as this is the only register that is connected to the system bus address lines.

Step 2: The address in MAR is put on the address bus, now a Read order is provided by the control unit on the control bus, and the result appears on the data bus and is then copied into the memory buffer register. Program counter is incremented by one, to get ready for the next instruction. These two acts can be carried out concurrently to save time.

Step 3: The content of the MBR is moved to the instruction register(IR).

Instruction fetch cycle consist of four micro operation:

T1: MAR← PC

T2: MBR← memory

PC← PC + stepsize or length of instruction

T3: IR← MBR

2. Decode instruction cycle

The next move is to fetch source operands once an instruction is fetched. Indirect addressing (it can be obtained by any addressing mode, here it is done by indirect addressing) is obtained by Source Operand. You don't need to fetch register-based operands. If the opcode is executed, it will require a similar process to store the result in main memory. Micro- operations take place:

T1: MAR🡐 IR(address)

T2: MBR🡐 Memory

T3: IR(address)🡐 (MBR(Address))

Step 1: The instruction address field is passed to the MAR. This is used to fetch the operand 's address.

Step 2: The address field of the IR is updated from the MBR.

Step 3: The IR is now in the state. Now IR is ready for the execute cycle.

3. Execute instruction Cycle

The initial three cycles (Fetch, Indirect, and Interrupt) are predictable and quick. Each requires simple , small, and fixed micro-operation sequences. The same micro-operation is repeated every time around in each event. Execute instruction cycle is different from them. Like, there is N different series of micro-operations that can occur for a computer with different N opcodes.

Example

ADD R , X

T1: MAR🡐 (IR(address))

T2: MBR🡐 Memory

T3: R🡐 (R) + (MBR)

Step 1: The address portion of IR is loaded into the MAR.

Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.

Step 3: Now, the contents of R and MBR are added by the ALU.

Hardwired Control

- Hardwired Control Unit is implemented using various electronic components such as combinational logic units and gates. The circuit uses a fixed architecture. If the instruction set is changed, the wiring should also be changed. As it is hardwired, the instruction set is constant and does not change. Therefore, a Hardwired Control Unit is used in processors that use simple instruction set known as the **Reduced Instruction Set Computers (RISC).**

- Usually, these control units execute faster. However, Hardwired Control Units are difficult to modify and implement. It is also difficult to add new features to the existing design. Therefore, it has minimum flexibility.

Micro programmed Control

- Before talking about Micro programmed Control Unit, it is important to understand some terms. A control variable refers to a binary variable that specifies micro-operations. A control word is a set of ones and zeros in a control variable. The control memory contains control words.

- Micro programmed Control Unit has a sequence of microinstructions stored in control memory. These microinstructions refer to a control word that resides in control memory, containing control signals for execution of micro-operations. Furthermore, a Micro programmed Control Unit is used in processors based on complex instruction set known as Complex Instruction Set Computer (CISC).

- Overall, these control units have a simple structure. Therefore, it is easier to design, implement and test. Moreover, Microprogrammed Control Units are easier to modify.

Difference between Hardwired Control and Micro programmed Control

Hardwired Control	Micro programmed Control
Technology is circuit based.	Technology is software based.
It is implemented through flip-flops, gates, decoders etc.	Microinstructions generate signals to control the execution of instructions.
Fixed instruction format.	Variable instruction format (16-64 bits per instruction).
Instructions are register based.	Instructions are not register based.
ROM is not used.	ROM is used.
It is used in RISC.	It is used in CISC.
Faster decoding.	Slower decoding.
Difficult to modify.	Easily modified.
Chip area is less.	Chip area is large.

UNIT-4
MEMORY SYSTEM

Memory system

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

- **Volatile Memory**: This loses its data, when power is switched off.
- **Non-Volatile Memory**: This is a permanent storage and does not lose any data when power is switched off.

Different Characteristics of Memory System

The different characteristics of memory system includes,

1. Location
2. Capacity
3. Access method
4. unit of transfer
5. Performance
6. Physical types
7. Organization
8. Physical characteristics

1. **Location**

   This characteristic represents whether the location is external or internal to the computer. External memory is the seperate storage device such as disk and tape. It is accessed by the processor through I/O controllers. Internal memory is mostly embedded within the computer. It is in various forms such as registers, cache and main memory.

2. **Capacity**

   It is the most important characteristic of the memory system. It is different for external and internal memory. Capacity of the external memory is measured in terms of bytes. Whereas the capacity of internal memory is measured in terms of bytes or words, wherein word length may be 8, 16 or 32 bits.

3. **Access method**

   This characteristic has four types of method.

   - Direct
   - Sequential
   - Random
   - Associative

- **Direct Access Method**

  In this method every block has its own unique address. Here, the access is made directly by first reaching to vicinity and then following sequential search. Access time of this method is concerned with current and previous location of data. An example of this method is hard disk.

- **Sequential Access Method**

  In this method access to data is made sequentially. Like direct access method, it also makes use of shared read-write mechanism. Tape is an example of this method. Access time is different with that of direct access but depends on same parameters.

- **Random Access Method**

  In this method each block of memory has its unique and physically wired-in addressing mechanism. Unlike direct and sequential access, the access time of this method does not depend on location . RAM is an example of this method.

- **Associative Access Method**

  In this method unlike other methods, access to data is made by comparing the required bit location within a word based on the specified data element. Therefore the access time of this method also depend on location. Cache memory is the example of this method.

4. **Unit of Transfer**

   This characteristic measures the transfer rate of data in and out of the memory. It is different for both the internal and external memory.

(a) For **internal memory**, it is the number of electrical lines going into and coming out of the memory . These lines carry data of word length.

(b) For **external memory**, it is the number of bits read out or written into the memory at a time.

5. **Performance**

   It is also the most important characteristic of the memory system. The following parameters need to be considered for performance,

   (a)Transfer Rate

   It can be defined as the rate at which data is transmitted into or out of the memory. It is different for different types of memory.

   (b)Memory Cycle Time

   It refers to the time required for accessing a block as well as the period prior to the start of the second access. It is mostly required for random access memory. It is related with the system bus but not the processor.

   (c)Access Time

   It is different for different types of memory. In random access memory, it is time required to carry-out read or write operation. In non-random access memory, it is time required to place the read-write mechanism at the desired place.

6. **Physical type**

   There are various physical types of memory such as,

   (a) Magnetic

   (b) Semiconductor

   (c) Magneto-optical

   (d) Optical

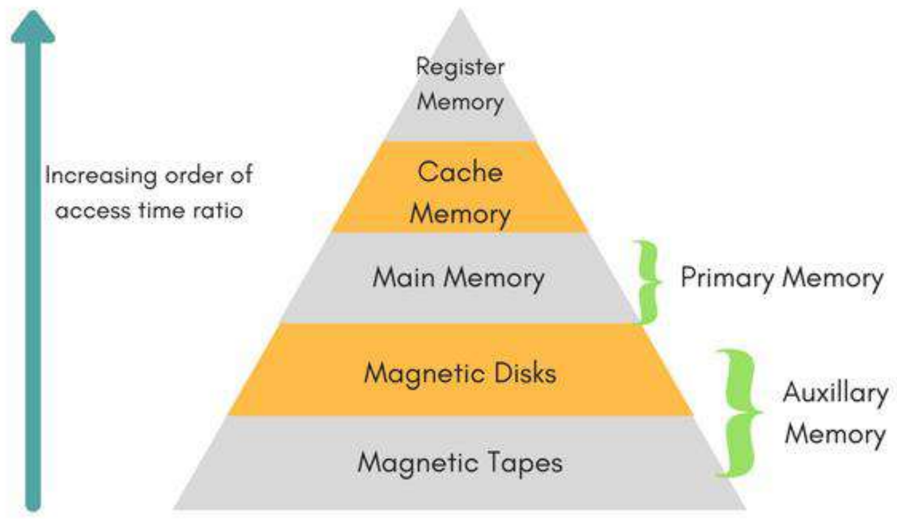
7. **Organization**

   It describes the physical arrangement of the bits. It is very important for random access memory.

8. **Physical characteristics**

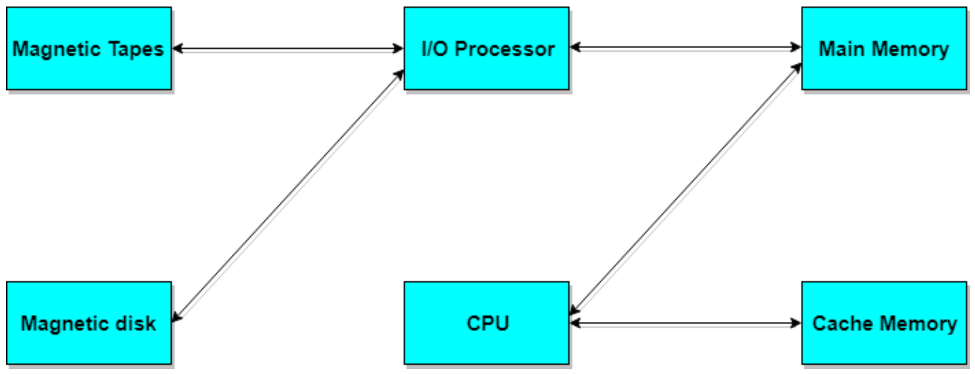
It describes the nature of memory that whether the memory is volatile, non-volatile erasable or non-erasable. Non-volatile memory does not ensure the persistence of data unlike volatile memory. Erasable memory allows one to change or modify data unlike non erasable memory.

Memory Hierarchy



- The total memory capacity of a computer can be visualized by hierarchy of components. The memory hierarchy system consists of all storage devices contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.
- **Auxiliary memory** access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.
- The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).
- When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use.

- The **cache memory** is used to store program data which is currently being executed in the CPU. Approximate access time ratio between cache memory and main memory is about **1 to 7~10**



Main Memory

The memory unit that communicates directly within the CPU, Auxillary memory and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.

- RAM: Random Access Memory
    - **DRAM**: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
    - **SRAM**: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
    - **NVRAM**: Non-Volatile RAM, retains its data, even when turned off. Example: Flash memory.
- ROM: Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on. **PROM**(Programmable

ROM), **EPROM**(Erasable PROM) and **EEPROM**(Electrically Erasable PROM) are some commonly used ROMs.

Auxiliary Memory

- Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks.
- It is not directly accessible to the CPU, and is accessed using the Input / Output channels.

Cache Memory

- The data or contents of the main memory that are used again and again by CPU are stored in the cache memory so that we can easily access that data in shorter time.
- Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory then the CPU moves onto the main memory. It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accommodate the new one.

Hit Ratio

- The performance of cache memory is measured in terms of a quantity called **hit ratio**. When the CPU refers to memory and finds the word in cache it is said to produce a **hit**. If the word is not found in cache, it is in main memory then it counts as a **miss**.
- The ratio of the number of hits to the total CPU references to memory is called hit ratio.

$$\text{Hit Ratio} = \text{Hit}/(\text{Hit} + \text{Miss})$$

Associative Memory

- It is also known as **content addressable memory (CAM)**. It is a memory chip in which each bit position can be compared. In this the content is compared in each bit cell which allows very fast table lookup. Since the entire chip can be compared, contents are randomly stored without considering addressing scheme. These chips have less storage capacity than regular memory chips.

Main Memory

The main memory acts as the central storage unit in a computer system. It is a relatively large and fast memory which is used to store programs and data during the run time operations.

The primary technology used for the main memory is based on semiconductor integrated circuits. The integrated circuits for the main memory are classified into two major units.

1. RAM (Random Access Memory) integrated circuit chips
2. ROM (Read Only Memory) integrated circuit chips

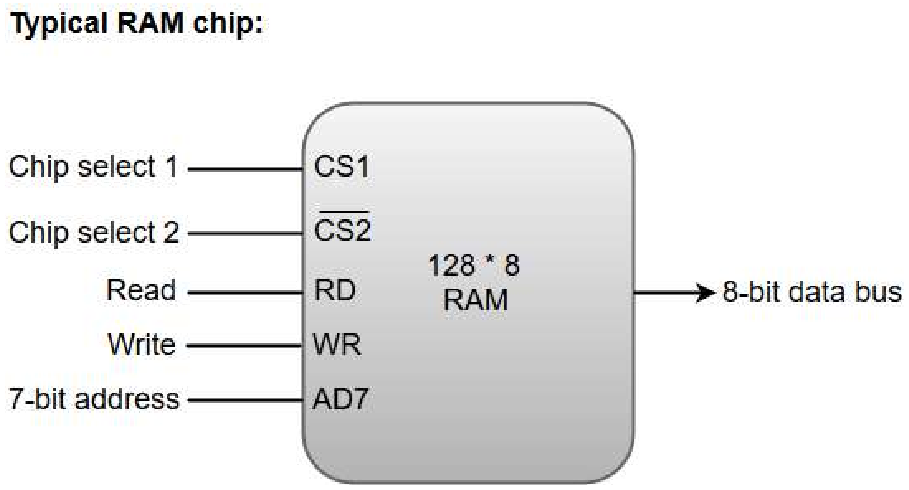
RAM Integrated Circuit Chips

The RAM integrated circuit chips are further classified into two possible operating modes, **static** and **dynamic**.

The primary compositions of a static RAM are flip-flops that store the binary information. The nature of the stored information is volatile, i.e. it remains valid as long as power is applied to the system. The static RAM is easy to use and takes less time performing read and write operations as compared to dynamic RAM.

The dynamic RAM exhibits the binary information in the form of electric charges that are applied to capacitors. The capacitors are integrated inside the chip by MOS transistors. The dynamic RAM consumes less power and provides large storage capacity in a single memory chip.

RAM chips are available in a variety of sizes and are used as per the system requirement. The following block diagram demonstrates the chip interconnection in a 128 * 8 RAM chip.



- A 128 * 8 RAM chip has a memory capacity of 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus.
- The 8-bit bidirectional data bus allows the transfer of data either from memory to CPU during a **read** operation or from CPU to memory during a **write** operation.
- The **read** and **write** inputs specify the memory operation, and the two **chip select (CS)** control inputs are for enabling the chip only when the microprocessor selects it.
- The bidirectional data bus is constructed using **three-state buffers**.
- The output generated by three-state buffers can be placed in one of the three possible states which include a signal equivalent to logic 1, a signal equal to logic 0, or a high-impedance state.

Note: The logic 1 and 0 are standard digital signals whereas the high-impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.

The following function table specifies the operations of a 128 * 8 RAM chip.

CS1	$\overline{\text{CS2}}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedance
0	1	x	x	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data to RAM
1	1	x	x	Inhibit	High-impedance

From the functional table, we can conclude that the unit is in operation only when CS1 = 1 and CS2 = 0. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0.

DRAM	SRAM
1. Constructed of tiny capacitors that leak electricity.	1.Constructed of circuits similar to D flip-flops.
2.Requires a recharge every few milliseconds to maintain its data.	2.Holds its contents as long as power is available.
3.Inexpensive.	3.Expensive.
4. Slower than SRAM.	4. Faster than DRAM.
5. Can store many bits per chip.	5. Can not store many bits per chip.
6. Uses less power.	6.Uses more power.
7.Generates less heat.	7.Generates more heat.
8. Used for main memory.	8. Used for cache.

Difference between SRAM and DRAM

ROM integrated circuit

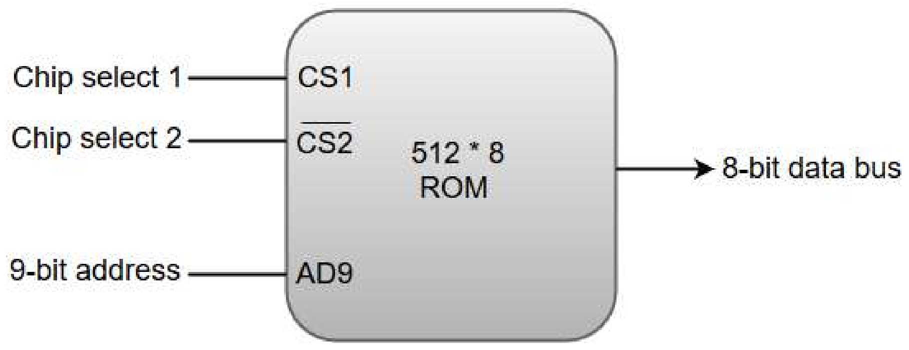
The primary component of the main memory is RAM integrated circuit chips, but a portion of memory may be constructed with ROM chips.

A ROM memory is used for keeping programs and data that are permanently resident in the computer.

Apart from the permanent storage of data, the ROM portion of main memory is needed for storing an initial program called a **bootstrap loader**. The primary function of the **bootstrap loader** program is to start the computer software operating when power is turned on.

ROM chips are also available in a variety of sizes and are also used as per the system requirement. The following block diagram demonstrates the chip interconnection in a 512 * 8 ROM chip.

Typical ROM chip:



- A ROM chip has a similar organization as a RAM chip. However, a ROM can only perform read operation; the data bus can only operate in an output mode.
- The 9-bit address lines in the ROM chip specify any one of the 512 bytes stored in it.
- The value for chip select 1 and chip select 2 must be 1 and 0 for the unit to operate. Otherwise, the data bus is said to be in a high-impedance state.

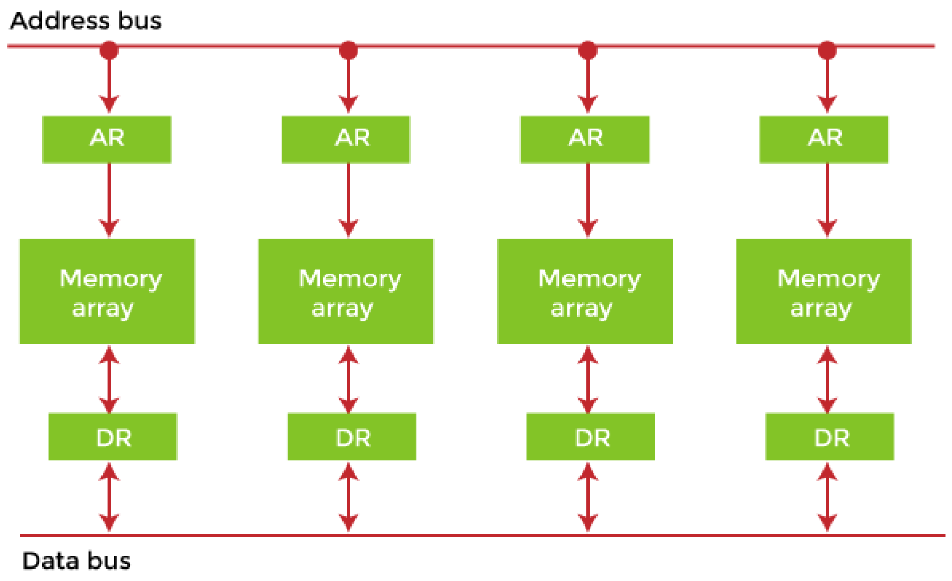
Types of Read Only Memory (ROM) –

1. **PROM (Programmable read-only memory)** –
   It can be programmed by the user. Once programmed, the data and instructions in it cannot be changed.
2. **EPROM (Erasable Programmable read only memory)** –
   It can be reprogrammed. To erase data from it, expose it to ultraviolet light. To reprogram it, erase all the previous data.
3. **EEPROM (Electrically erasable programmable read only memory)** –
   The data can be erased by applying an electric field, with no need for ultraviolet light. We can erase only portions of the chip.
4. **MROM(Marked ROM)** –
   The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kinds of ROMs are known as masked ROMs, which are inexpensive.

Interleaved Memory

Interleaved memory is designed to compensate for the relatively slow speed of dynamic random-access memory (DRAM) or core memory by spreading memory addresses evenly across memory banks. In this way, contiguous memory reads and writes use each memory bank, resulting in higher memory throughput due to reduced waiting for memory banks to become ready for the operations.

It is different from multi-channel memory architectures, primarily as interleaved memory does not add more channels between the main memory and the memory controller. However, channel interleaving is also possible, for example, in ***Free scale*** i.MX6 processors, which allow interleaving to be done between two channels. With interleaved memory, memory addresses are allocated to each memory bank.



Example of Interleaved Memory

It is an abstraction technique that divides memory into many modules such that successive words in the address space are placed in different modules.

Suppose we have 4 memory banks, each containing 256 bytes, and then the Block Oriented scheme (no interleaving) will assign virtual addresses 0 to 255 to the first bank and 256 to 511 to the second bank. But in Interleaved memory, virtual address 0 will be with the first bank, 1 with the second memory bank, 2 with the third bank and 3 with the fourth, and then 4 with the first memory bank again.

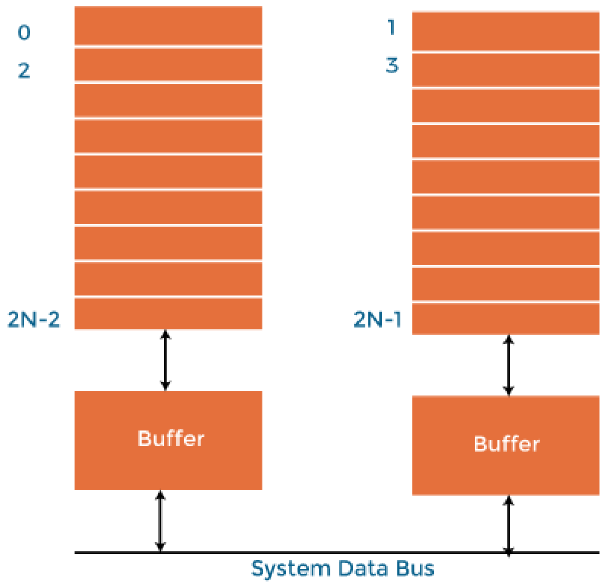
Hence, the CPU can access alternate sections immediately without waiting for memory to be cached. There are multiple memory banks that take turns for the supply of data.

In the above example of 4 memory banks, data with virtual addresses 0, 1, 2 and 3 can be accessed simultaneously as they reside in separate memory banks. Hence we do not have to wait to complete a data fetch to begin the next operation.

An interleaved memory with n banks is said to be ***n-way interleaved***. There are still ***two banks of DRAM*** in an interleaved memory system, but logically, the system seems one bank of memory that is twice as large.

In the interleaved bank representation below with 2 memory banks, the first long word of bank 0 is flowed by that of bank 1, followed by the second long word of bank 0, followed by the second long word of bank 1 and so on.

The following image shows the organization of two physical banks of n long words. All even long words of the logical bank are located in physical bank 0, and all odd long words are located in physical bank 1.



Why do we use Memory Interleaving?

When the processor requests data from the main memory, a block (chunk) of data is transferred to the cache and then to processor. So whenever a cache miss occurs, the data is to be fetched from the main memory. But main memory is relatively slower than the cache. So to improve the access time of the main memory, interleaving is used.

For example, we can access all four modules at the same time, thus achieving parallelism. The data can be acquired from the module using the higher bits. This method uses memory effectively.

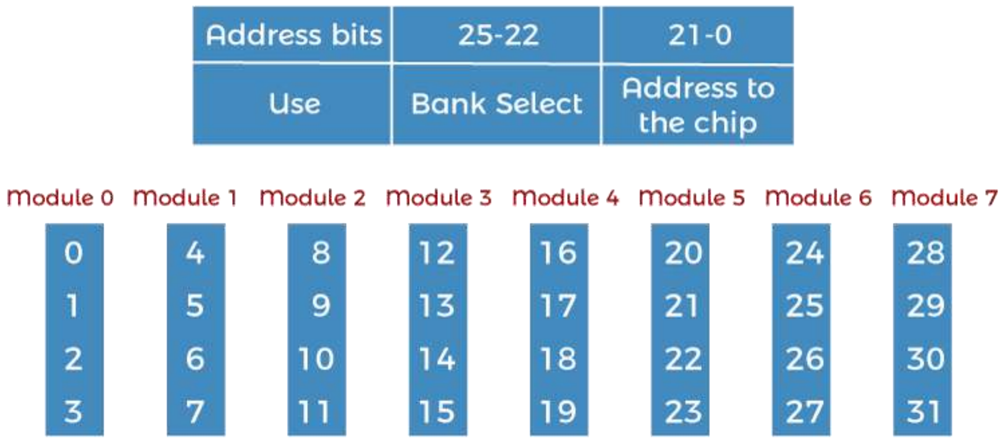
Types of Interleaved Memory

There are two types of interleaved memory, such as:

1. High order interleaving:

In high order memory interleaving, the most significant bits of the memory address decides memory banks where a particular location resides. But, in low order interleaving the least significant bits of the memory address decides the memory banks.

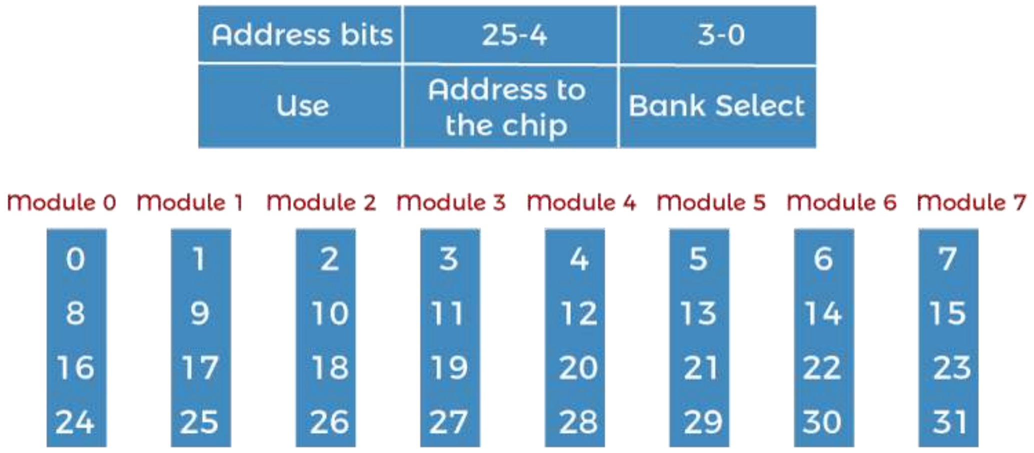
The least significant bits are sent as addresses to each chip. One problem is that consecutive addresses tend to be in the same chip. The maximum rate of data transfer is limited by the memory cycle time. It is also known as ***Memory Banking***.



Address bits	25-22	21-0
Use	Bank Select	Address to the chip

Module 0	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6	Module 7
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

2. Low order interleaving: The least significant bits select the memory bank (module) in low-order interleaving. In this, consecutive memory addresses are in different memory modules, allowing memory access faster than the cycle time.



Address bits	25-4	3-0
Use	Address to the chip	Bank Select

Module 0	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6	Module 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

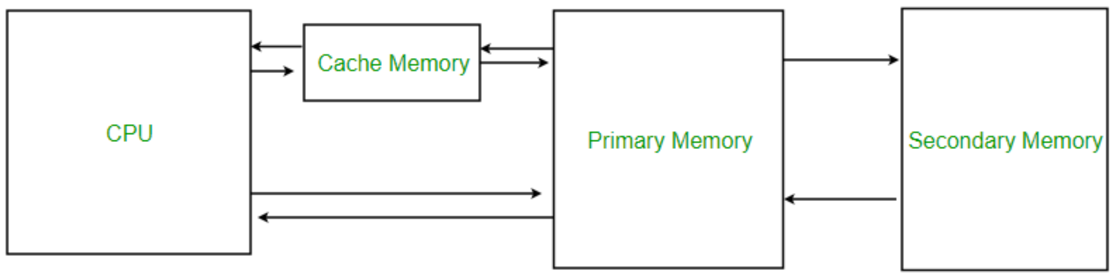
Benefits of Interleaved Memory

An instruction pipeline may require instruction and operands both at the same time from main memory, which is not possible in the traditional method of memory access. Similarly, an arithmetic pipeline requires two operands to be fetched simultaneously from the main memory. So, to overcome this problem, memory interleaving comes to resolve this.

- It allows simultaneous access to different modules of memory. The modular memory technique allows the CPU to initiate memory access with one module while others are busy with the CPU in reading or write operations. So, we can say interleave memory honors every memory request independent of the state of the other modules.
- So, for this obvious reason, interleave memory makes a system more responsive and fast than non-interleaving. Additionally, with simultaneous memory access, the CPU processing time also decreases and increasing throughput. Interleave memory is useful in the system with pipelining and vector processing.
- In an interleaved memory, consecutive memory addresses are spread across different memory modules. Say, in a byte-addressable 4 way interleave memory, if byte 0 is in the first module, then byte 1 will be in the 2nd module, byte 2 will be in the 3rd module, byte 3 will be in the 4th module, and again byte 4 will fall in the first module, and this goes on.
- An n-way interleaved memory where main memory is divided into n-banks and system can access n operands/instruction simultaneously from n different memory banks. This kind of memory access can reduce the memory access time by a factor close to the number of memory banks. In this memory interleaving memory location, i can be found in bank **i mod n**.

Cache Memory

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



Levels of memory:

- **Level 1 or Register –**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory –**
  It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Cache Mapping:

There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

1. **Direct Mapping –**

   The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or
   In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

   i = j modulo m

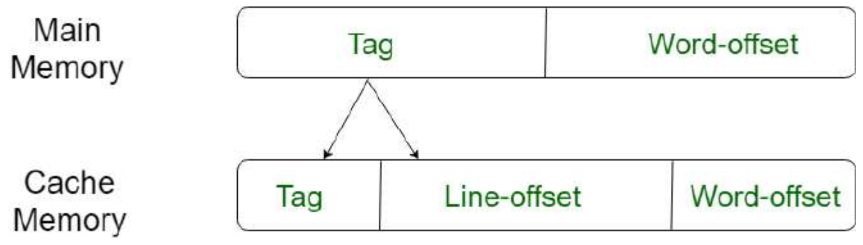
   where

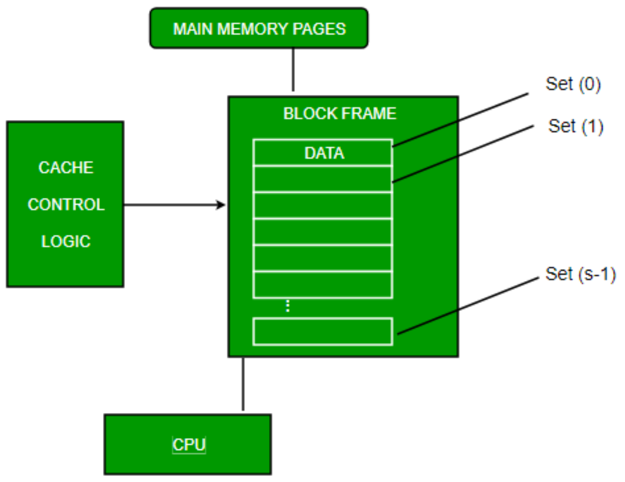
   i=cache line number

   j= main memory block number

   m=number of lines in the cache

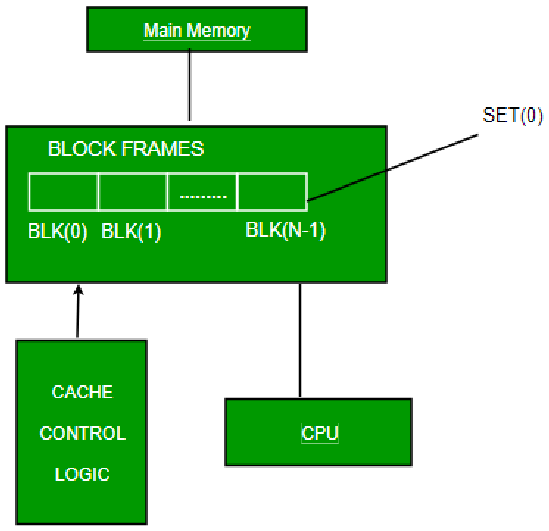
For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache.





2. **Associative Mapping –**

   In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.



3. **Set-associative Mapping –**

   This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set..Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.

   In this case, the cache consists of a number of sets, each of which consists of a number of lines. The relationships are

   $m = v * k$

   $i= j \bmod v$

   where

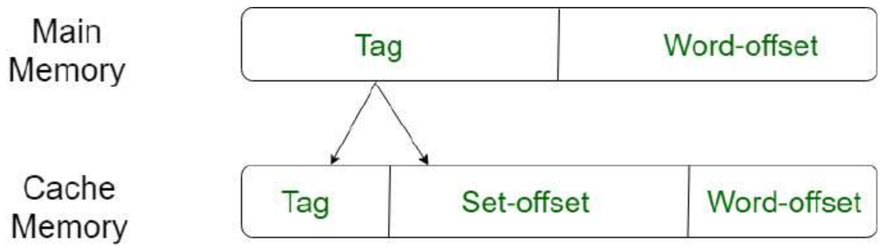
   i=cache set number

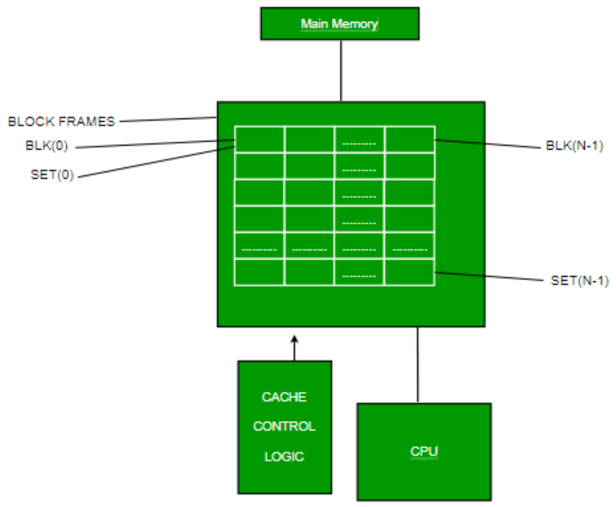
   j=main memory block number

   v=number of sets

   m=number of lines in the cache number of sets

   k=number of lines in each set





Application of Cache Memory –

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache –

- **Primary Cache –**

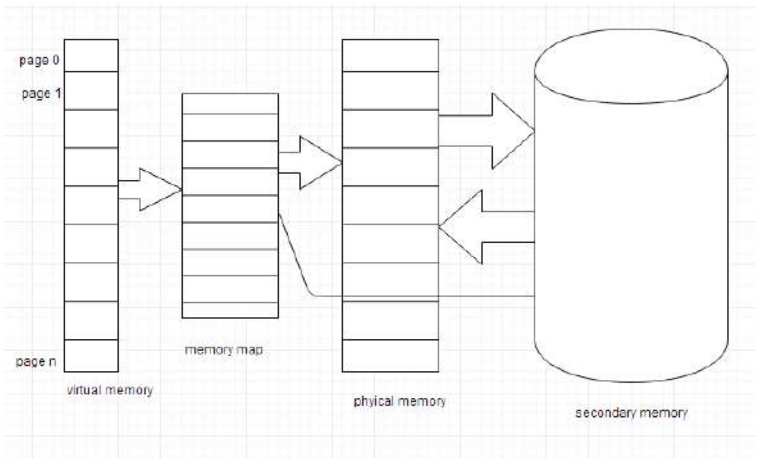
  A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
- **Secondary Cache –**

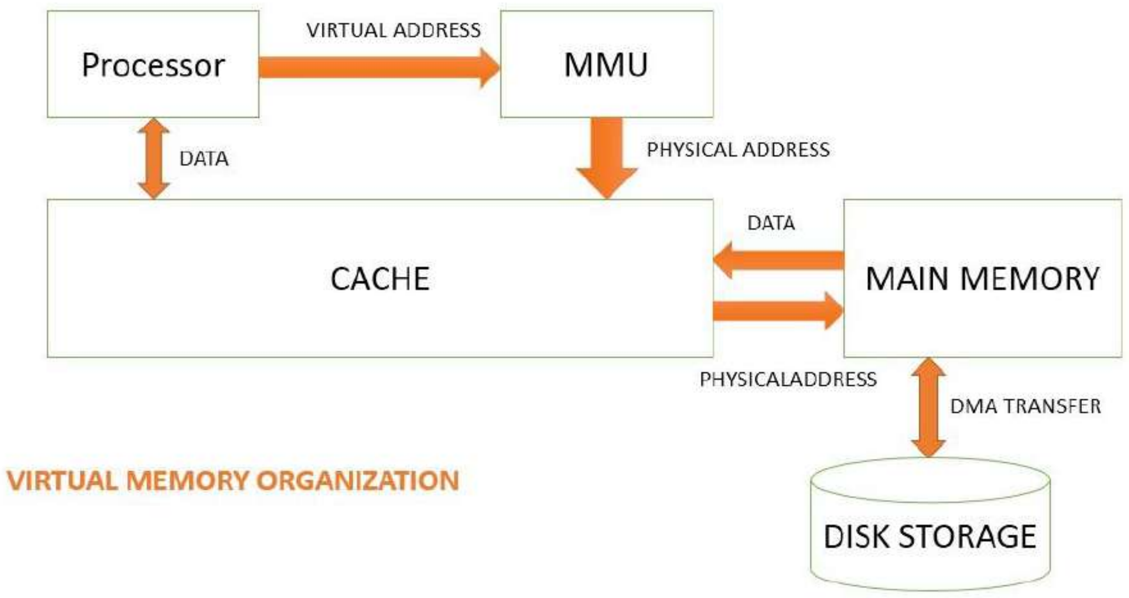
  Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

VIRTUAL MEMORY

Virtual memory is the separation of logical memory from physical memory. This separation provides large virtual memory for programmers when only small physical memory is available.

Virtual memory is used to give programmers the illusion that they have a very large memory even though the computer has a small main memory. It makes the task of programming easier because the programmer no longer needs to worry about the amount of physical memory available.



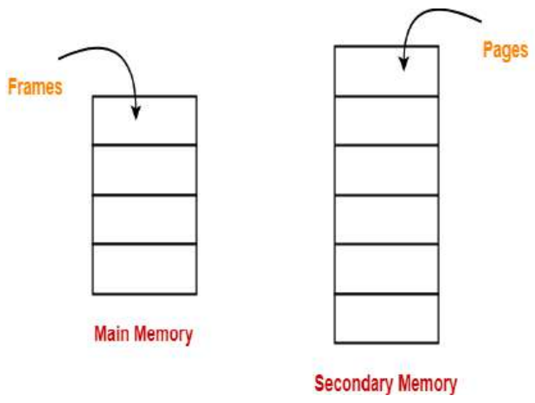


A virtual memory can be **configured** using any of the following technique:

1. **Paging technique**
2. **Segmentation technique**

PAGING:-

- Paging is a fixed size partitioning scheme.
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as **pages**.
- The partitions of main memory are called as **frames**.

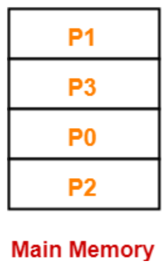


- Each process is divided into parts where size of each part is same as page size.
- The size of the last part may be less than the page size.
- The pages of process are stored in the frames of main memory depending upon their availability.

<u>Example-</u>

Consider a process is divided into 4 pages P_0, P_1, P_2 and P_3.

- Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-



Translating Logical Address into Physical Address-

CPU always generates a logical address.

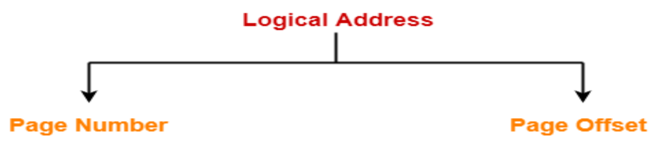
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Page Number
2. Page Offset



- Page Number specifies the specific page of the process from which CPU wants to read the data.
- Page Offset specifies the specific word on the page that CPU wants to read.

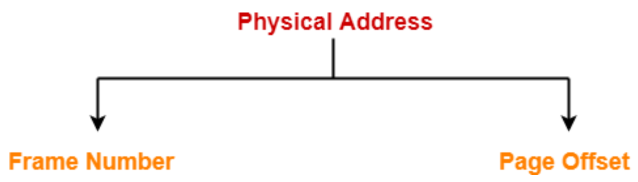
Step-02:

For the page number generated by the CPU,

- **Page Table** provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

Step-03:

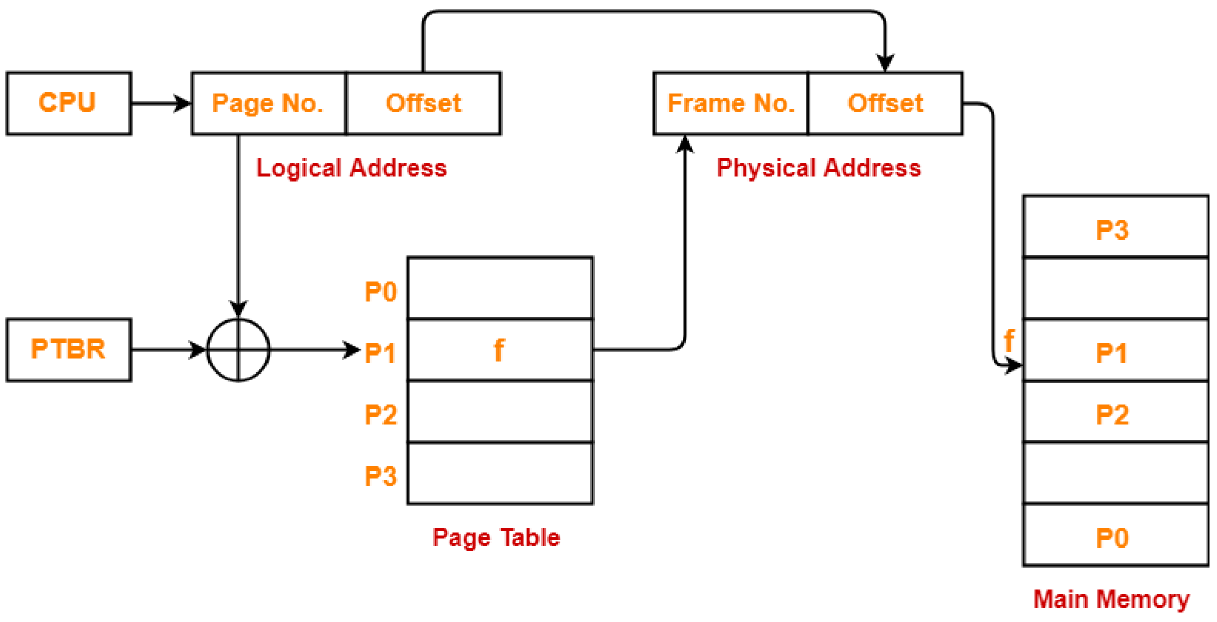
The frame number combined with the page offset forms the required physical address.



- Frame number specifies the specific frame where the required page is stored.
- Page Offset specifies the specific word that has to be read from that page.

<u>Diagram-</u>

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

<u>Advantages-</u>

- It allows to store parts of a single process in a non-contiguous fashion.
- It solves the problem of external fragmentation.

<u>Disadvantages-</u>

- It suffers from internal fragmentation.
- There is an overhead of maintaining a page table for each process.

- The time taken to fetch the instruction increases since now two memory accesses are required.

SEGMENTATION-

Like Paging, Segmentation is another non-contiguous memory allocation technique.

- In segmentation, process is not divided blindly into fixed size pages.
- Rather, the process is divided into modules for better visualization.

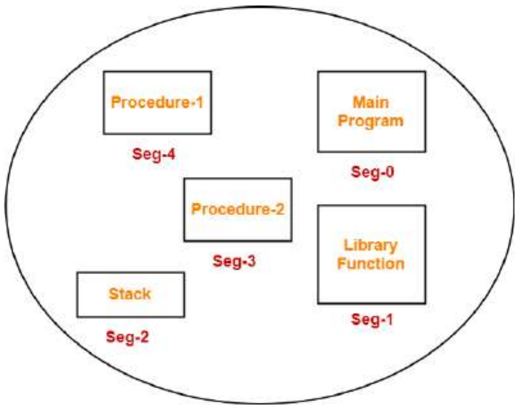
<u>Characteristics-</u>

Segmentation is a variable size partitioning scheme.

- In segmentation, secondary memory and main memory are divided into partitions of unequal size.
- The size of partitions depends on the length of modules.
- The partitions of secondary memory are called as **segments**.

<u>Example-</u>

Consider a program is divided into 5 segments as-

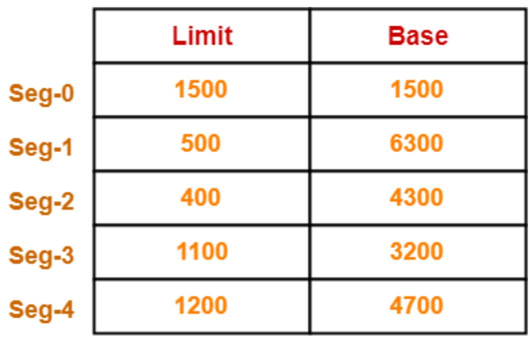


<u>Segment Table-</u>

- Segment table is a table that stores the information about each segment of the process.
- It has two columns.
- First column stores the size or length of the segment.

- Second column stores the base address or starting address of the segment in the main memory.
- Segment table is stored as a separate segment in the main memory.
- Segment table base register (STBR) stores the base address of the segment table.

For the above illustration, consider the segment table is-

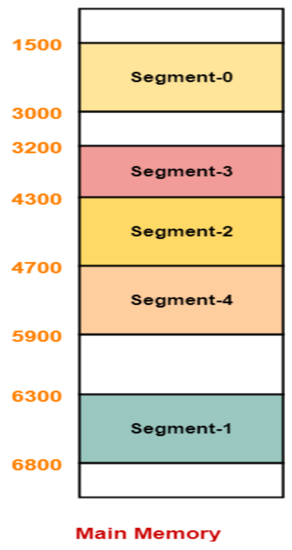


	Limit	Base
Seg-0	1500	1500
Seg-1	500	6300
Seg-2	400	4300
Seg-3	1100	3200
Seg-4	1200	4700

Segment Table

- Limit indicates the length or size of the segment.
- Base indicates the base address or starting address of the segment in the main memory.

In accordance to the above segment table, the segments are stored in the main memory as-



Translating Logical Address into Physical Address-

CPU always generates a logical address.

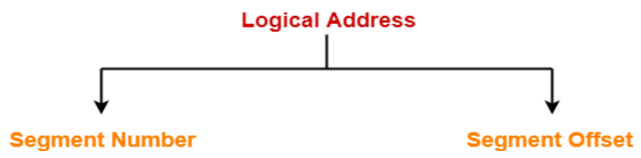
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Segment Number
2. Segment Offset



- Segment Number specifies the specific segment of the process from which CPU wants to read the data.
- Segment Offset specifies the specific word in the segment that CPU wants to read.

Step-02:

For the generated segment number, corresponding entry is located in the segment table.

- Then, segment offset is compared with the limit (size) of the segment.

Now, two cases are possible-

Case-01: Segment Offset >= Limit

If segment offset is found to be greater than or equal to the limit, a trap is generated.

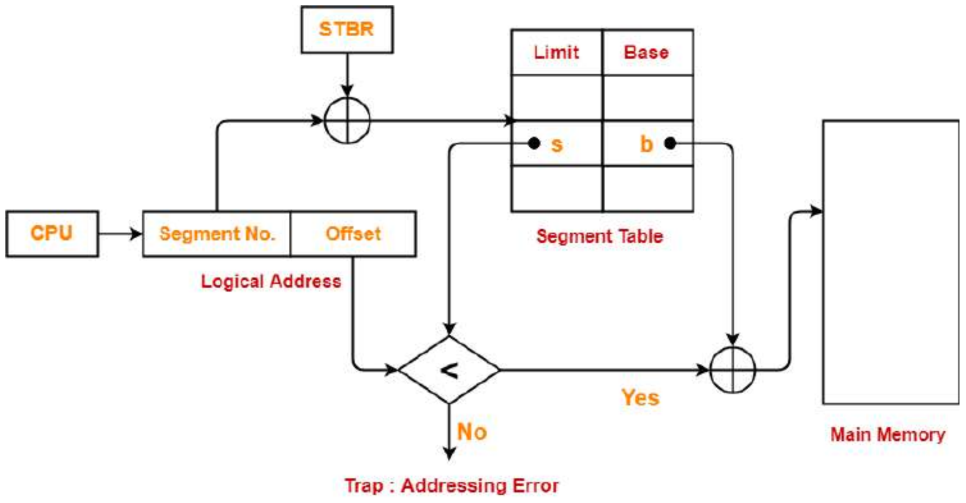
Case-02: Segment Offset < Limit

If segment offset is found to be smaller than the limit, then request is treated as a valid request.

- The segment offset must always lie in the range [0, limit-1],
- Then, segment offset is added with the base address of the segment.
- The result obtained after addition is the address of the memory location storing the required word.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages-

- It allows to divide the program into modules which provides better visualization.
- Segment table consumes less space as compared to **Page Table** in paging.
- It solves the problem of internal fragmentation.

Disadvantages-

- There is an overhead of maintaining a segment table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.
- Segments of unequal size are not suited for swapping.
- It suffers from external fragmentation as the free space gets broken down into smaller pieces with the processes being loaded and removed from the main memory.

Advantages of Virtual Memory

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

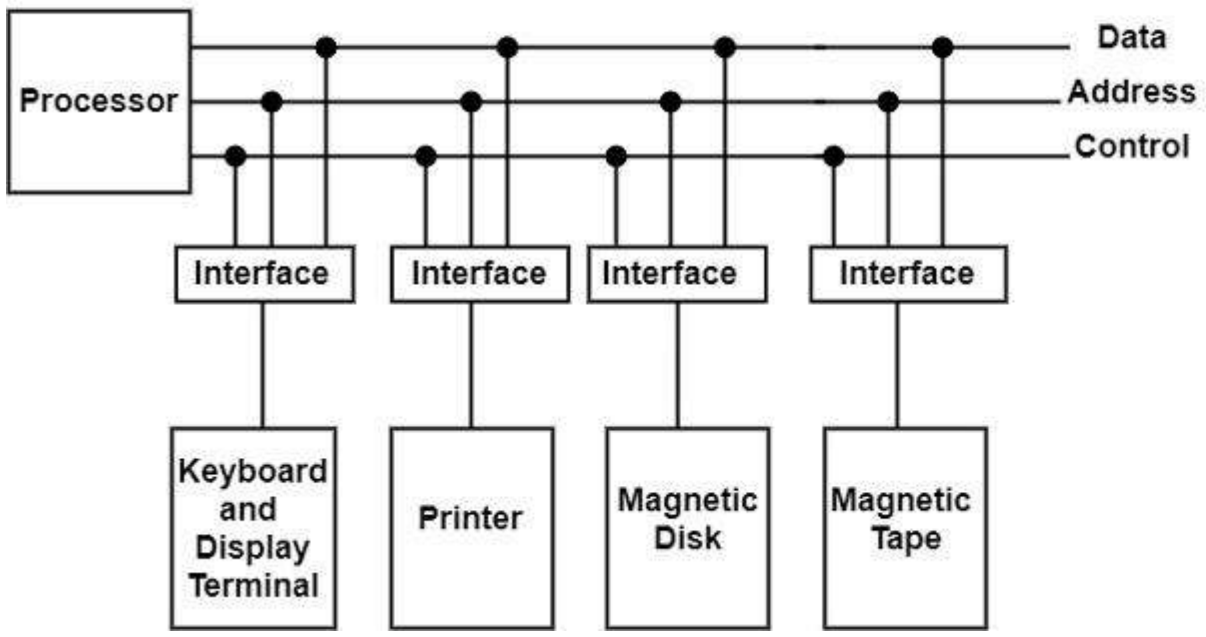
UNIT – 5
INPUT-OUTPUT INTERFACE

Input-Output Interface is used as a method which helps in transferring of information between the internal storage devices i.e. memory and the external peripheral device. A peripheral device is that which provide input and output for the computer, it is also called Input-Output devices.

Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral.

For Example: A keyboard and mouse provide Input to the computer are called input devices while a monitor and printer that provide output to the computer are called output devices. Just like the external hard-drives, there is also availability of some peripheral devices which are able to provide both input and output.

Connection of I/O Bus to I/O Devices



The I/O bus includes data lines, address lines, and control lines. In any general-purpose computer, the magnetic disk, printer, and keyboard, and display terminal are commonly employed. Each peripheral unit has an interface unit associated with it. Each interface decodes the control and address received from the I/O bus.

It can describe the address and control received from the peripheral and supports signals for the peripheral controller. It also conducts the transfer of information between peripheral and processor and also integrates the data flow.

The I/O bus is linked to all peripheral interfaces from the processor. The processor locates a device address on the address line to interact with a specific device. Each interface contains an address decoder attached to the I/O bus that monitors the address lines.

When the address is recognized by the interface, it activates the direction between the bus lines and the device that it controls. The interface disables the peripherals whose address does not equivalent to the address in the bus.

An interface receives any of the following four commands −

- **Control** − A command control is given to activate the peripheral and to inform its next task. This control command depends on the peripheral, and each peripheral receives its sequence of control commands, depending on its mode of operation.
- **Status** − A status command can test multiple test conditions in the interface and the peripheral.
- **Data Output** − A data output command creates the interface counter to the command by sending data from the bus to one of its registers.
- **Data Input** − The data input command is opposite to the data output command. In data input, the interface gets an element of data from the peripheral and places it in its buffer register.

In micro-computer base system, the only purpose of peripheral devices is just to provide **special communication links** for the interfacing them with the CPU. To resolve the differences between peripheral devices and CPU, there is a special need for communication links.
The I/O communication is simplified to a set of requirements consisting of three parts of Where, What and How? i.e.

- Where? - Identification of the device under communication – The device and location within the device Ex: Select the Printer.
- What? - What is the type of communication? i.e sender? Receiver? (Read/Write?)
- How? - How much to be communicated? (Number of bytes to/from Memory/file)

The major differences are as follows:

1. The nature of peripheral devices is electromagnetic and electro-mechanical. The nature of the CPU is electronic. There is a lot of difference in the mode of operation of both peripheral devices and CPU.
2. There is also a synchronization mechanism because the data transfer rate of peripheral devices are slow than CPU.
3. In peripheral devices, data code and formats are differing from the format in the CPU and memory.
4. The operating modes of peripheral devices are different and each may be controlled so as not to disturb the operation of other peripheral devices connected to CPU.

There is a special need of the additional hardware to resolve the differences between CPU and peripheral devices to supervise and synchronize all input and output devices.

These components are called Interface Units because they interface between theν processor bus and the peripheral devices. I/O BUS and Interface Module It define the typical link between the processor and several peripherals.

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O:

i. Use two Separate buses , one for memory and other for I/O.

ii. Use one common bus for both memory and I/O but separate control lines for each.

iii. Use one common bus for memory and I/O with common control lines.

Functions of Input-Output Interface:

1. It is used to synchronize the operating speed of CPU with respect to input-output devices.
2. It selects the input-output device which is appropriate for the interpretation of the input-output device.
3. It is capable of providing signals like control and timing signals.
4. In this data buffering can be possible through data bus.
5. There are various error detectors.
6. It converts serial data into parallel data and vice-versa.
7. It also converts digital data into analog signal and vice-versa.

Bus Design Characteristics

Few of the important characteristics of interest are Bus Type, Bus Width, Clock Rate, Protocol and Arbitration mechanism.

- **Bus Type** - Dedicated or Multiplexed

  Dedicated - The address and data lines are separate.

  Multiplexed - The address and data lines are sent on the same physical cable but at different timings.

- **Width –** The number of lines that carry Data and Address. More width on address lines increases the addressing range. More width on data lines increases Data Bandwidth.
- **Access Protocol** - Synchronous or Asynchronous

Synchronous	Asynchronous
One of the control line is Clock	There is no common clock
All activities in the bus are synchronized to the clock and at predefined clock cycles	The bus transfers information based on handshake protocol.

Synchronous	Asynchronous
Useful when all the devices in the bus are in the same speed range.	Since there is a handshake, any device may interact with any other device.
Internal system bus is an example of synchronous bus.	Generally used for slow speed device communications.

- **Arbitration** - The protocol to gain access to the bus amongst the eligible competing devices is called *bus arbitration*. During the bus operation, there is a Bus Master (Initiator) and a Bus Slave (Responder). Thus a device wanting to initiate communication has to become bus master. Always, the communication happens between two devices. Since there are more devices connected on a bus, there is a bus arbitrator and an identified arbitration mechanism.
- **Clock Rate** – The speed of the bus is determined by the Synchronous clock. It is a design decision to fix the clock rate.

There are few other physical parameters like how the wires are to be run, what type of connectors to be used, length of the wires, etc. The electrical characteristics define the voltage levels of the signals and the power source.

I/O System Organization

The CPU executes a program in a Fetch-Decode –Execute mode by bringing instructions from Memory. So your program has to get loaded into memory, from the Disk. Generally, both program and executables file is stored in the Disk. This is where the I/O comes into the picture. It is interesting to know the functional requirements of I/O communication with CPU and Memory.

I/O is also a Slave component in the computer. An I/O operation is initiated by CPU and the I/O controllers take care of transferring and completing the I/O operation.

An analogy to this is a classroom environment, wherein a teacher (CPU) assigns work; the student (I/O) submits the solution in a Notebook or uploads at the prescribed place(Memory). Probably the student informs the teacher that it is done or it is done for some reason(Interrupt).

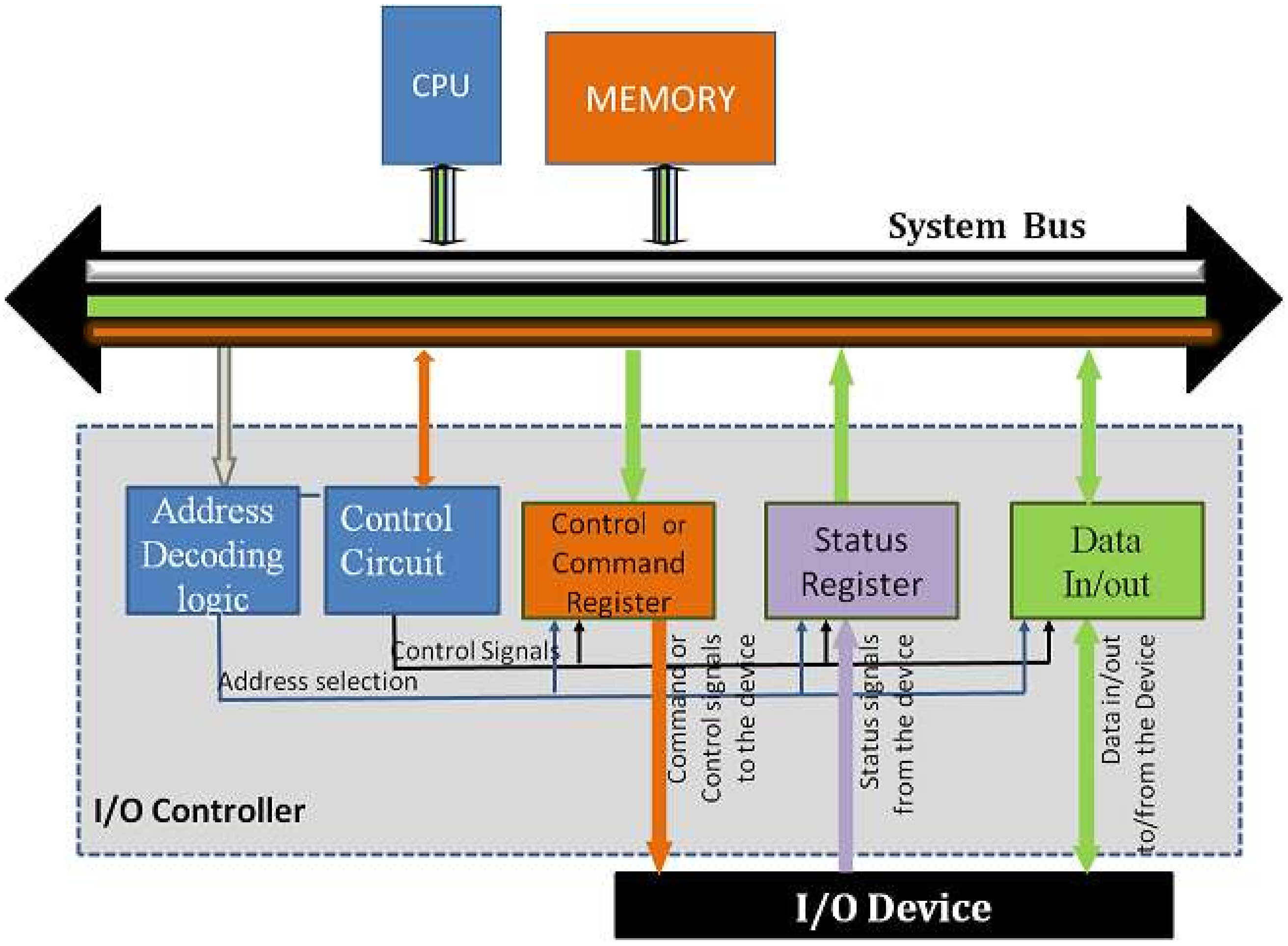
I/O Controllers

- The devices operate at wide-ranging data transfer speed and also many different interface standards. The Keyboard, Mouse have very small data rates and are asynchronous in data transfer to computer. Disk, Solid State Disks have high data rates. USB has a mediocre data rate. And we know each one has a different connector and interface standard.
- It is overloading on the part of CPU to deal with these devices directly. I/O controllers play a bridging role between CPU, Memory and I/O Device by taking care of all kinds of communication.
- Due to heterogeneity of the devices, each device /type of interface requires an I/O Controller
- I/O controllers also act as a buffer during data transfer

A data transfer from an I/O device involves:

- Initiate the operation (i.e. addressing the device)
- Direct the device (i.e. communicate the operation to be done and control the data transfer)
- Closing of the IO operations (i.e notifying the CPU that it is done or not done with status).

To do the above, each IO Controller will typically have Data Register(s), Status Register(s), Control Register(s), Address decoding logic and Control Circuitry. The I/O Controller is connected to the system bus. Whenever the I/O controller wants to use the bus, it has to contend and obtain. All communication from the CPU and Memory happens via these registers shown in the diagram. These registers are given a unique address for each I/O controller.



I/O Controller

The **address decoding logic** is connected to the address bus. The value on the address bus indicates the register to be accessed. The decoding logic converts this as an address selection signal to one of these registers.

The **control circuitry** is connected to the control signals of the system bus. These signals are MEMW, MEMR, IOR, IOW, INTERRUPT, BREQ, etc. These signals ensure the synchronization and validation of address and Data on the bus, demanding the Bus for data transfer, sending interrupt after the normal or abnormal end operation.

The **Data Registers** take care of Data Transfer. There may be Data In and/or Data Out Registers depending on the device. Also, fast devices like Disk will have a Buffer so that the fast bulk data from the disk is stored and then sent to Memory when the system bus is available.

The **Control Register** has information like:

- Which is the device to be accessed? (Address of the device)
- What is the Operation (Command) to be carried out on the device?
    - Printer Commands – Form Feed for a new page, Print, Line Feed, status check, etc.
    - Disk – Seek, Write, Read, Reset, status check
- How this operation is to be carried out?
    - Number of Bytes in case of Data Transfer
    - What communication is expected by CPU after data transfer to memory
    - Mode Settings applicable to devices, etc.

The purpose of **Status Register** is:

- To have the information about the device being selected
    - Whether the device is online and ready?
    - If not online have the information about the error from the device. Example - a) Printer may not have a cartridge or paper. This error status is collected in the status register. b) The Disk may not be getting ON
- The status of the data transfer on the device side is stored as a flag in the Status Register.
    - Whether the data transfer was successful?
    - Data Overrun or Underrun may occur with fast devices like DISK. The buffer operates on PUSH – PULL mode. So in case of an error in PUSH or PULL side Data Overrun or Underrun occurs.
    - The status register keeps track of the Word Count pending to be transferred from/to Memory. A Counter is maintained in the I/O controller for this purpose. When Word Count becomes ZERO, it is implied that the data transfer is successfully done.
    - Interrupt bit is part of the Status Register. An Interrupt is reasoned with the content of the status register.

I/O Addressing Schemes

This is essentially about allocating addresses to the I/O registers. Each Controller has a set of registers. Two methods in practice are I/O mapped I/O and Memory-mapped I/O.

I/O Mapped I/O

The registers are assigned port numbers and accessed via special instructions namely IN and OUT. This is also Intel's method. IN is for reading from the Data or Status Registers. OUT is for writing onto the DATA or Command Registers.

Memory-Mapped I/O

Portions of the Memory address space are assigned to I/O device. This method reduces the number of control signals required on the System Bus for READ/WRITE.

I/O Mapped I/O	**Memory-Mapped I/O**
Exclusive I/O Address space for I/O addressing which are called port addresses	Part of memory address space is reserved for I/O addressing
Total addressable space is available for Memory	Available memory is addressable memory space less the reserved space I/O addressing
On the Control bus, I/O or Memory operation is differentiated by a separate set of control signals like IOR/IOW, MEMR/MEMW.	On the Control bus, there is only one set of control signals i.e. MEMR/MEMW. Memory or I/O operation is differentiated by decoding the address.
The number of lines required for addressing I/O ports is less. Hence address decoding logic on the I/O controllers is smaller and simpler.	The number of lines used for addressing is equal to the maximum addressable space; hence address decoding logic on the I/O controller is larger.
Separate I/O instructions like IN/OUT are used for I/O communication.	No such separate instructions used; the same set of STORE/LOAD instructions that are used for

I/O Mapped I/O	Memory-Mapped I/O
	memory operations are used for I/O operations too.
More opcodes needed in the CPU	Does not increase the opcodes.

Input/output Data Transfer

Data from Peripherals have to reach Memory so that the CPU can use it. Similarly, the processed output has to reach peripherals like printer, internet devices, DISK, etc.

Thus, I/O data transfer is about how the three subsystems i.e. CPU, Memory and I/O Controller, are involved in achieving the data exchange with peripherals. The word Data Exchange means successful Data in/Data Out with necessary handshake and coordination.

Data Exchange with a peripheral involves few steps and not straight forward as devices are mostly electromechanical and/or operate at a different speed than CPU and Memory.

- The device to be initiated and checked for status whether it is ready for Data Exchange.
- The data transfer has to happen at the acceptable capsule and speed of the device using the applicable protocol.
- This may happen in more than one sequence, as necessitated.
- If the device status is an Error status, it is to be suitably handled.

I/O Controller's role is to ensure seamless data exchange by ensuring the following :

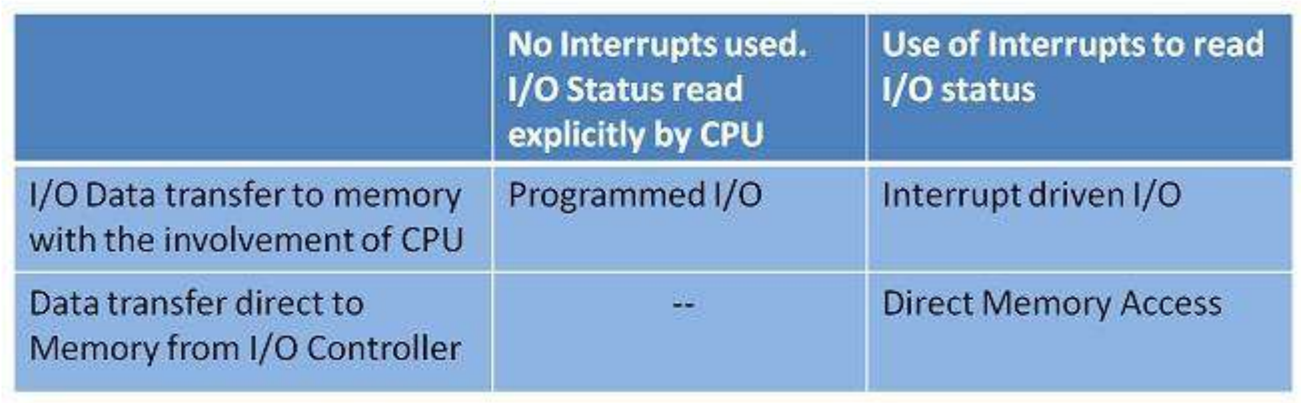
- Processor/Memory Communication
- Device Communication
- Control timing and Status processing – Coordination of the data traffic between CPU, Memory and Io devices
- Data Buffering – To manage the data transfer speed mismatch

- Error management (Transmission errors are corrected other status errors are communicated)

I/O Data Transfer Techniques

CPU can be used very efficiently in Multitasking. Ideally, the involvement CPU in I/O data transfer is expected to be minimal. There are three possibilities for I/O Data exchange.

- Programmed I/O
- Interrupt driven I/O
- Direct Memory Access



	No Interrupts used. I/O Status read explicitly by CPU	Use of Interrupts to read I/O status
I/O Data transfer to memory with the involvement of CPU	Programmed I/O	Interrupt driven I/O
Data transfer direct to Memory from I/O Controller	--	Direct Memory Access

I/O Data exchange methods and features

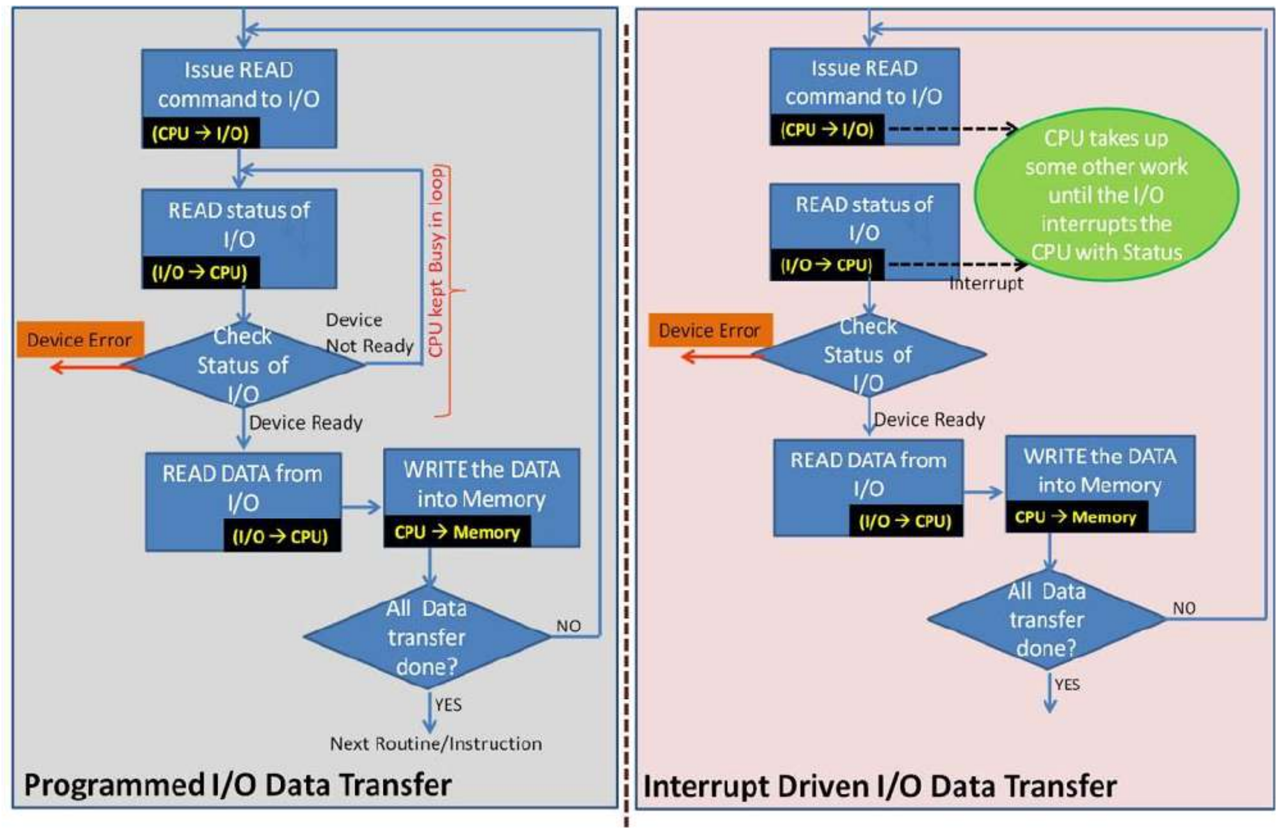
Programmed I/O

A program controls the I/O Operation; hence CPU is fully involved. CPU Monitors the status of the device and the I/O data transfer. This keeps the CPU hooked until the I/O operation is completed for the desired number of bytes. In the case of READ from I/O device, the final destination for Data is Memory. CPU writes the data collected from the device into Memory. It happens vice versa in the case of WRITE onto the device.

Programmed I/O is used for tiny data transfer i.e. few bytes. Before an actual Data transfer to a device, selecting, readying the device and verifying the status are to be done. Ex: Selecting the Printer and initialization before printing. This is shown by the decision box loop in figure 20.2.

In the diagram, at each step, the direction of information flow between the subsystems is marked to reinforce your understanding.

The CPU loop during status check is eliminated in the Interrupt Driven I/O Operation.



Programmed I/O and Interrupt Driven I/O

Interrupt Driven I/O data transfer

The steps involved in I/O data transfer are the same but for a change in freeing the CPU until the device is ready for data transfer.

- The CPU initiates the command on the device and takes up other tasks.
- Once the device is ready with data, the I/O controller generates an Interrupt.
- On recognizing the pending Interrupt signal from I/O controller, the CPU temporarily suspends the ongoing execution, goes in for interrupt service routine and collects the data and stores in memory or vice versa and then resumes the

suspended execution. In this mode also, it is the CPU which interacts with memory for storing the device data.

Most of the devices are electromechanical and have high latency. The period for which the CPU is freed is proportional to this. Hence the benefit achieved is reasonable. The interrupt is asynchronous. The I/O controller waits until the interrupt is serviced by CPU. Thus, what we have achieved is the I/O controller is made to wait instead of the CPU. CPU time is more precious. Details of Interrupt servicing will be dealt with in the next chapter.

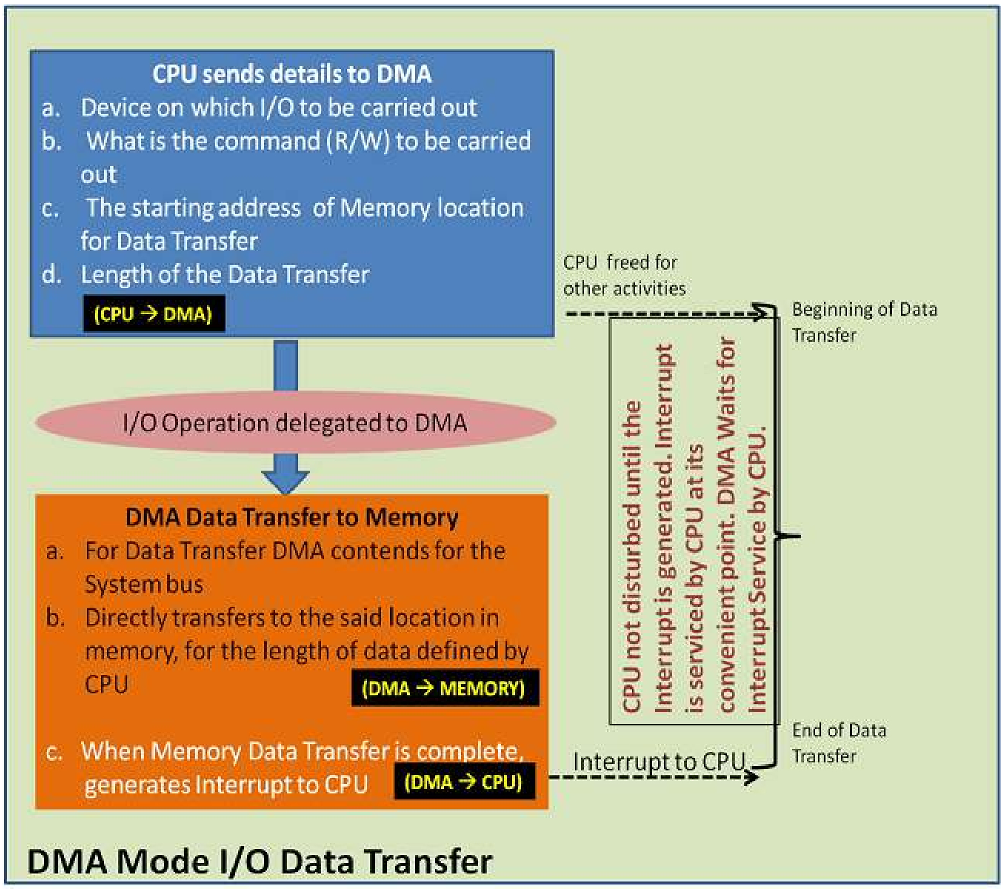
Interrupt service routine is also a complex and costly matter involving OS. This method is also feasible for a small quantum of data transfer. Direct Memory access can resolve the CPU availability in a hardware environment.

Direct Memory Access (DMA) Data Transfer

Interrupt servicing of the device is still accomplished using the software. This may not be fast enough for a large chunk of data transfer from DISK. Recall that the Virtual memory implementation is supported by Disk. This involves large data exchange to memory. The software routine is replaced with a specialized hardware processor, called the Direct Memory Access Controller (DMAC). DMA technique facilitates I/O data transfer between Main Memory and devices connected to DMAC, Fast data rate devices are connected to DMAC.

- The CPU passes necessary information to DMA and goes on with any other work.
- The DMA takes care of data transfer to Memory.
- On completion, DMA generates an Interrupt to inform the CPU of the job done.

Note that the CPU as a boss gives the work and informed of Job done but not involved for doing the job. It is independently taken care of by DMAC.



CPU sends details to DMA
a. Device on which I/O to be carried out
b. What is the command (R/W) to be carried out
c. The starting address of Memory location for Data Transfer
d. Length of the Data Transfer
(CPU → DMA)
I/O Operation delegated to DMA
DMA Data Transfer to Memory
a. For Data Transfer DMA contends for the System bus
b. Directly transfers to the said location in memory, for the length of data defined by CPU
(DMA → MEMORY)
c. When Memory Data Transfer is complete, generates Interrupt to CPU
(DMA → CPU)
CPU freed for other activities
Beginning of Data Transfer
CPU not disturbed until the Interrupt is generated. Interrupt is serviced by CPU at its convenient point. DMA Waits for Interrupt Service by CPU.
End of Data Transfer
Interrupt to CPU
DMA Mode I/O Data Transfer

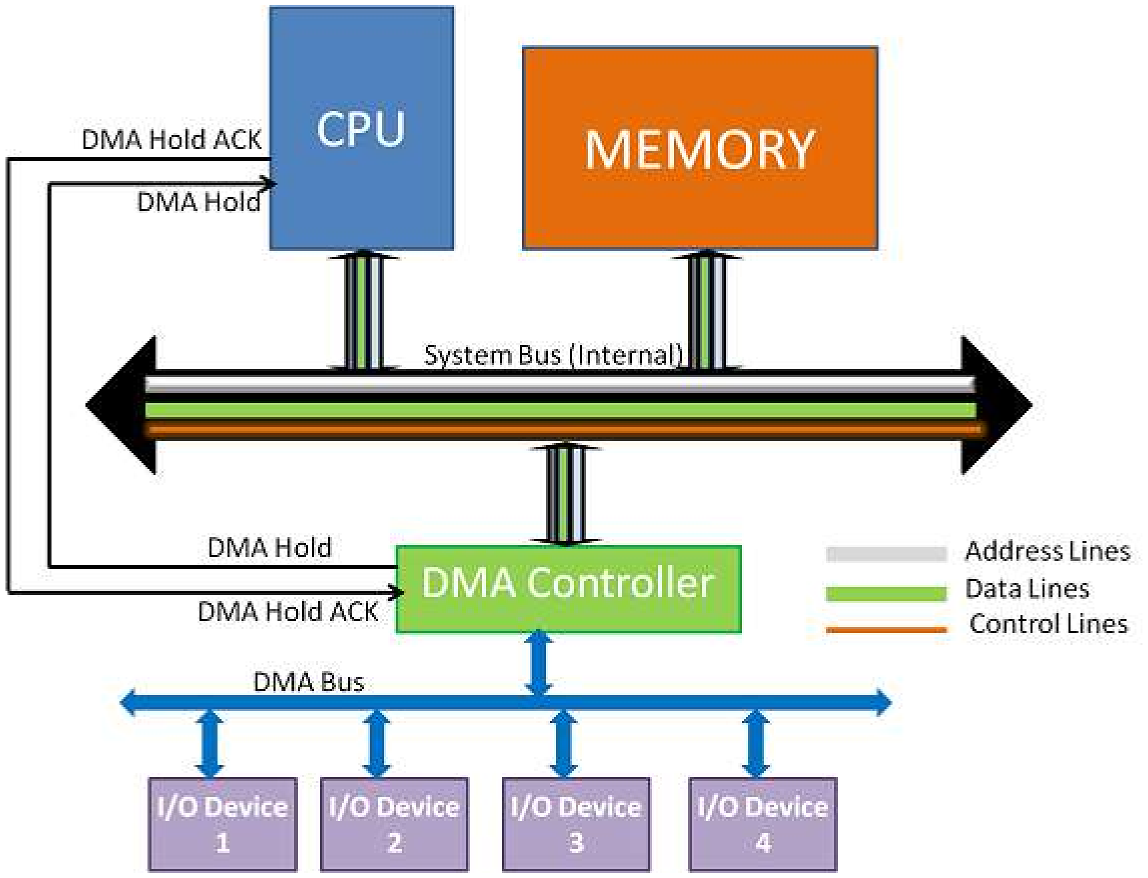
The DMA method improvises the interrupt method with optimal efficiency as the data is not routed via CPU. Thus DMA unburdens CPU.

Direct Memory Access controller

DMA is Direct Memory Access. As the name implies, DMA facilitates data transfer between I/O and Memory directly, instead of involving CPU as in the other two cases of I/O data transfer. DMA is faster and bulk data transfer technique. The system bus is common between CPU, Memory, DMA and maybe few I/O controllers. At an instant, the system bus can be used for communication between any two members only. Further, at a time, a resource can be used by only one entity. Thus when DMA communicates with Memory, the CPU will be on hold to use the System Bus. However, the CPU may do any other machine cycle internally using ALU and other resources. DMA Controller (DMAC) is special hardware which manages the above functions.

DMAC operation

In the below figure, the DMAC is connected to the system bus. The I/O devices are connected to the DMA ports called ***Channels***. A DMAC can generally support 4 I/O devices. The coordination between CPU and DMA happens with DMA raising DMA HOLD signal and the CPU responding with DMA HOLD ACK signal.



A System configuration with DMAC

The sequence of events described below clarifies how an end to end data transfer happens between I/O device and Memory using DMA.

- CPU delegates the responsibility of data transfer to DMA by sending the following details:
    - A device on which I/O to be carried out
    - What is the command (R/W) to be carried out
    - The starting address of Memory location for Data Transfer

  - Length of the Data Transfer (Byte Count) - The first two information is given to the device controller while the last two information is stored in the channel register in DMAC.

- The I/O controller initiates the necessary actions with the device and requests DMAC when it is ready with data.
- DMAC raises the HOLD signal to CPU conveying its intention to take the System bus for data transfer.
- At the end of the current machine cycle, the CPU disengages itself from the system bus. Then, the CPU responds with a HOLD ACKNOWLEDGE signal to DMAC, indicating that the system bus is available for use.
- DMAC places the memory address on the address bus, the location at which data transfer is to take place.
- A read or write signal is then generated by the DMAC, and the I/O device either generates or latches the data. Then DMA transfers data to memory.
- A register is used as a **byte count**, decremented for each byte transferred to memory.
- Increment the memory address by the number of bytes transferred and generate new memory address for the next memory cycle.
- Upon the **byte count** reaches zero, the DMAC generates an Interrupt to CPU.
- As part of the Interrupt Service Routine, the CPU collects the status of Data transfer.

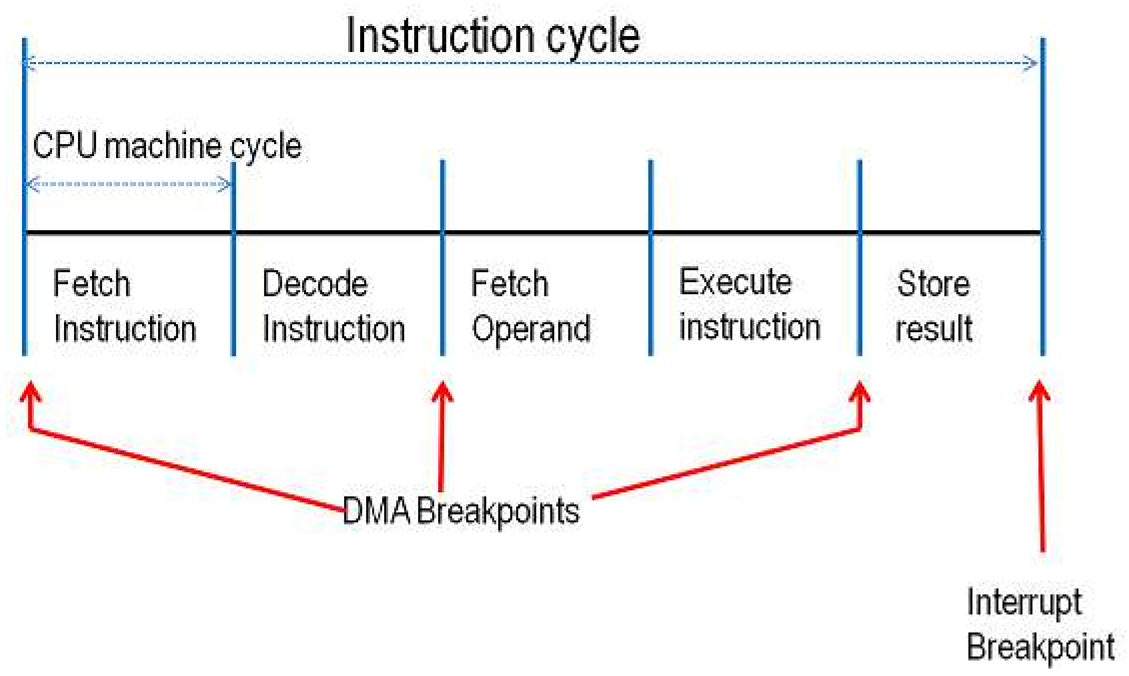
DMA to Memory Data Transfer Modes

The data transfer modes are about how does the CPU acquires the system bus for its activity. There are three options available in the DMAC, namely Cycle Stealing, Burst Mode and Hidden Mode.

Cycle Stealing: The DMAC steals the system bus (Memory cycles) from the CPU, during its non-memory machine cycles like Decode or execute, to transfer a byte/word to/from memory. Cycle stealing is possible when there is a separate arbitrator logic for the system bus. It is to be noted that, in the case of HOLD and HOLD ACK sequence, CPU acts as bus arbitrator. The DMAC steals the bus for every transaction, hence the name *cycle stealing*.

Burst Mode: Once the DMAC takes the bus, it completes the data transfer with the necessary number of cycles. Holds the system bus until the Byte Count reaches zero, after which it releases the bus. The CPU is halted during the data transfer. Since the intended block is transferred in one bus access, Burst mode is also called **Block Transfer Mode**. In this case, the DMAC is assigned an equal priority with CPU.

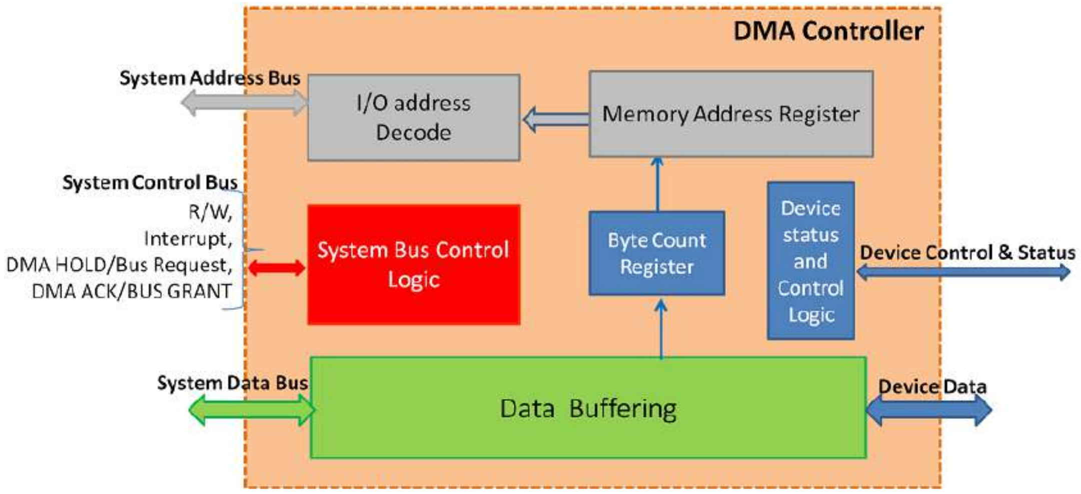
Transparent or Hidden Mode: There are some internal states during which the CPU frees the bus, disengaging itself. At these times, DMA takes up data transfers to memory. In this mode, the DMA is assigned the lowest priority, rather it never contends for the bus. The CPU speed is never affected by DMA. Not only DMA requires extra logic to detect the floating bus but also CPU requires extra logic to indicate its status. The throughput of DMA is poor and the data transfer is slowest of all the modes. In the below figure explains when the CPU can recognise DMA or Interrupt. These points are called breakpoints.



Breakpoints in CPU

DMA Controller Functional Components

It has all the functional components of any I/O controller. Since DMA communicates on the system bus, required additional logic is in-built. The bus control logic generates R/W signal and Interrupt. Depending on the DMA data transfer mode, bus arbitration signals are generated. The bus arbitration signals are DMAHOLD or Bus Request, DMA HOLD ACK or BUS GRANT as the case may be.



DMAC functional components

Byte Count Register is an important register which holds and tracks the byte transfer taking place with Memory. CPU sets the value. It is decremented for every byte transferred and the Memory address register is updated to show the next location for data transfer. When the byte count becomes zero, it indicates End of Data Transfer. This status is used for generating an interrupt signal to CPU.

Buffered Data Transfer takes place at the device end. The required control signals are generated. This part is very much similar to I/O Controller.

Generally, DMA has more than one channel to facilitate more I/O devices connection. In such a case, we will have duplicity of Memory Address Register, Data Buffer, Status and Control Registers and Byte Count register. The interface logics remain common.

Advantages of DMAC

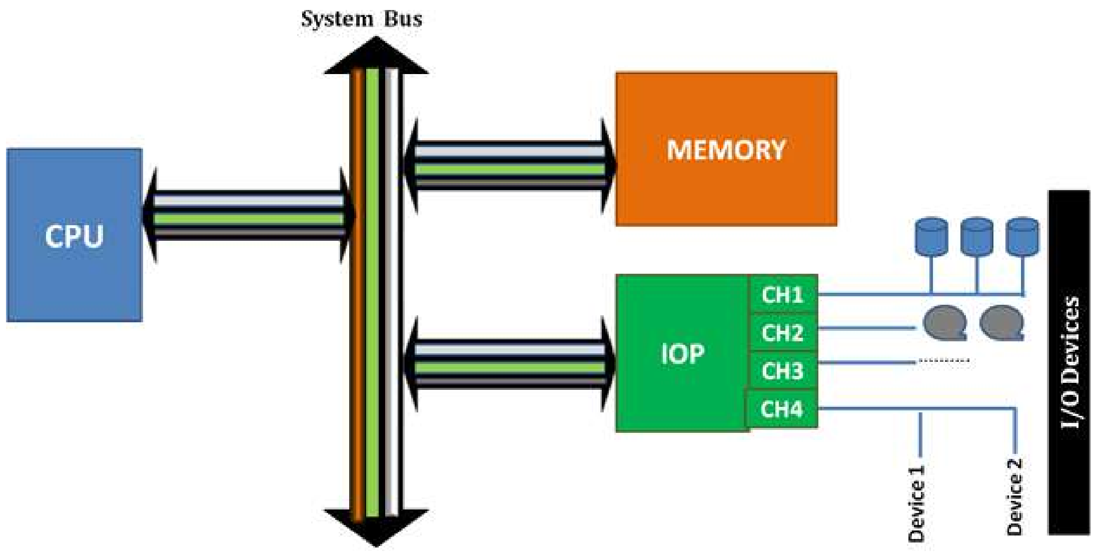
- Certainly speeds up the data transfer to memory with minimal bus holding
- Relieves the CPU for its task

In the case of DMAC, the CPU is involved to initiate every single IO Operation. Although the DMA method is better than the Programmed IO and Interrupt driven IO, overall system performance can be improved if there is a design methodology which will take care of total I/O with simple initiation from CPU. The IO Processor (IOP) design supports this requirement.

I/O Processors

I/O processors (IOP) are an extension to the DMA concept of I/O Operation. As the name implies, I/O processors have some processing capacity for serving the device controllers connected to it. IOPs can decode and execute I/O instruction. These instructions are brought from Main Memory by the IOP. Data transfer happens with Memory directly. Thus an IOP has a fair amount of control over IO Operations with very minimal initiation from CPU.

IOPs are also called ***Peripheral Processing Unit*** (PPU). I/O Channels is a name used in Mainframe to mean IOP.



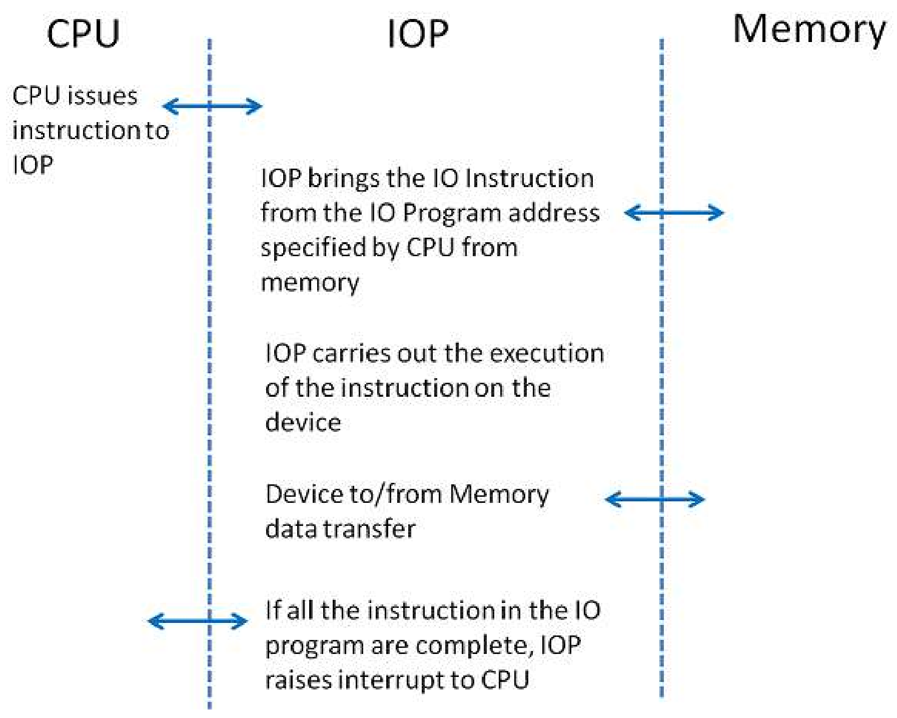
A System configuration with IOP

Characteristics of IOP

- Has Instruction set catered to IO operations
- It can execute a sequence of instructions called "I/O Program" on a device connected to it.
- Ability to decode and execute IO instructions for a variety of devices. Each device has unique commands that can be executed on them.
  - DISK – RESET, SEEK, READ, WRITE, FORMAT, Error Checking, etc
  - TAPE – REWIND, FORWARD, WRITE, READ, BOT, EOT, etc
  - PRINTER – LINE FEED, FORM FEED, WRITE, PRINT, etc
- High-Speed Data transfer to increase the IO Performance
- Variety of devices can be connected
- Multiplexed IOP connects slow speed device controllers
- Selector IOP for high-speed device controllers
- Similar to DMA channels, IOP also has channels to connect Device controllers
- IOP is connected to System Bus
- Possible to have more than one IOP in a system configuration

Functioning of IOP

IOP frees CPU from I/O data transfer operations. The CPU is involved to the extent of informing the address of the I/O program in memory. And the IOP is expected to get the IO Program, decode and execute. After the data transfer is complete, the IOP raises an Interrupt to CPU. The communication steps between CPU, IOP and Memory in completing data transfer is depicted in below .

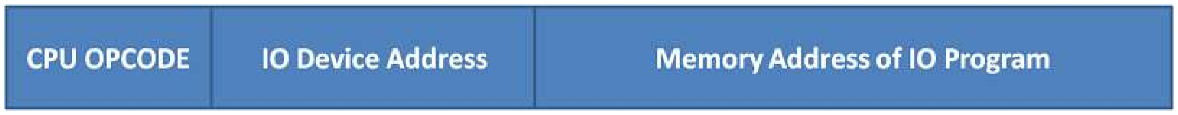

CPU
IOP
Memory
CPU issues instruction to IOP
IOP brings the IO Instruction from the IO Program address specified by CPU from memory
IOP carries out the execution of the instruction on the device
Device to/from Memory data transfer
If all the instruction in the IO program are complete, IOP raises interrupt to CPU

Communication steps in IOP operation

CPU – IOP Instruction

The instruction issued by CPU to IOP has three parts:

- CPU Opcode indicating that the instruction is to instruct IOP
- IO Device Address – it is the target device on which the IO program is to be executed
- IO Program Address – It is a Main Memory Address where the set of commands to be executed on the IO Device is stored.



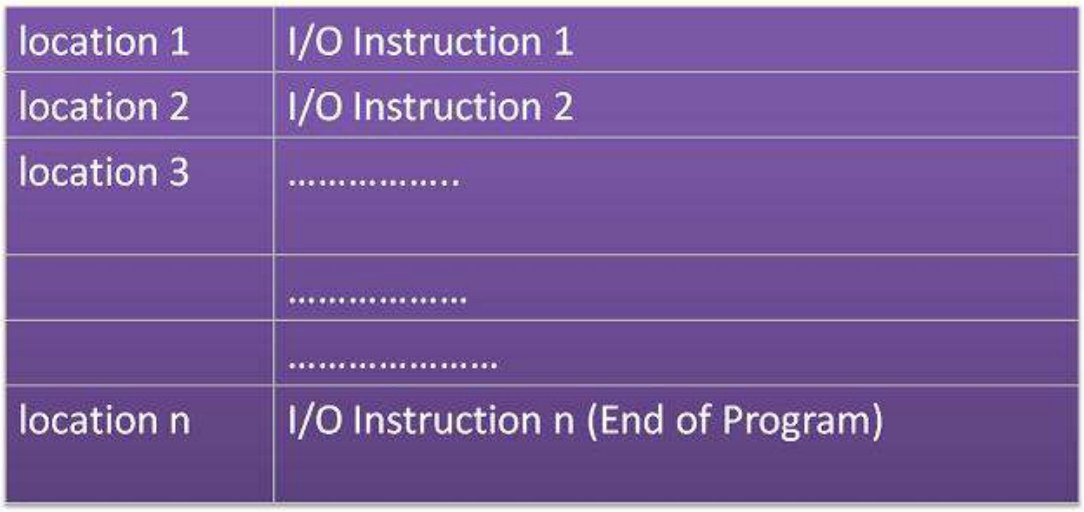
CPU OPCODE	IO Device Address	Memory Address of IO Program

Components of CPU instruction to IOP

From the third part, it is clear that the CPU does not get involved in instructing the IO devices. Corresponding to each IO Device, an IO Program is stored in Memory. The IO program consists of a set of instructions to be executed on the device notified by the CPU Instruction to IOP.

IO Program

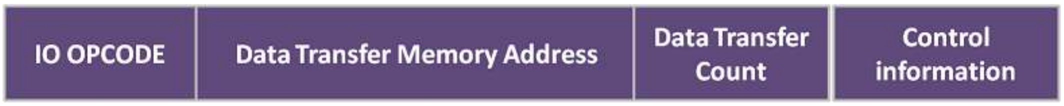
A typical IO program is much like any other program, except that it has IO Instructions to be executed on a select device.



location 1	I/O Instruction 1
location 2	I/O Instruction 2
location 3	…………….
	………………
	…………………
location n	I/O Instruction n (End of Program)

IO Program in Memory

An I/O instruction received by IOP from memory as part of the IO Program has four parts as below:



IO OPCODE	Data Transfer Memory Address	Data Transfer Count	Control information

IOP Instruction from Memory

- IO Opcode – the IO instruction to be executed on the device

- If it is a Data Transfer instruction, the memory address for data transfer. For data transfer, IOP accesses memory via the system bus. It contends on the bus based on the bus arbitration logic design.
- If it is a Data Transfer instruction, the size of the data transfer – Data transfer count
- Control information – how this instruction to be executed in terms off error handling, interrupts etc. and also, whether any more instructions are chained and hence follows this instruction. This is a very useful feature. IT has bits assigned to indicate End of IO Program

Any data transfer operation on DISK is preceeded by a SEEK command to position the heads on the place where the file is located. Command chaining helps.

Also if data transfer is to happen from more than one area of disk, chaining of SEEK + data transfer is done easily with IO program.

Many such examples in the case of printer, tape and other devices can be given.

For CPU it is too costly to get involved in these scenario. Thus, IOP improves the IO and CPU performance.

IOP Instruction Categories

The IOP is likely to receive the following categories of instructions.

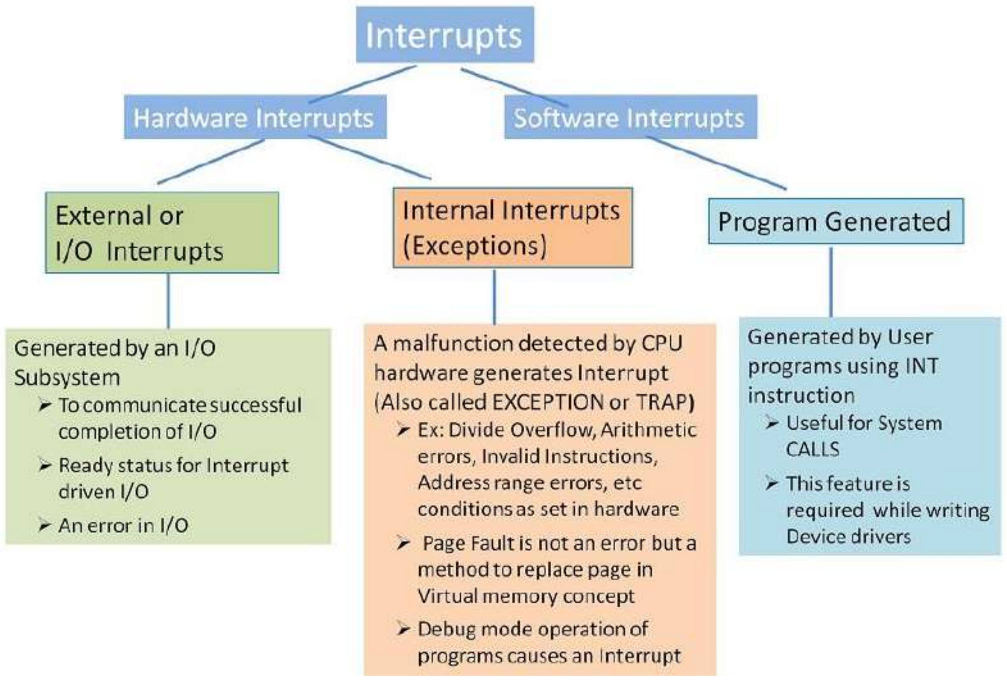
- **Data transfer instructions** - for doing data transfer.
- **Branch instructions** - allow the IOP to fetch the next CCW form a specified memory location instead of from the contiguous location
- **Arithmetic and logical instructions** – facilitate the calculation of memory addresses, IO Device priorities and so on
- **IO Device Control instructions** - These are not data transfer ones; these are transmitted to the device and specify functions peculiar to that device. Ex: seek, rewind, Top of form etc.

We have seen the IOP fetching instructions from memory, decoding, controlling, calculating, etc. This is as much a mini CPU. It requires most of the functional logic of CPU like program counter, decoder, flags, etc. Thus the name IO Processor is justified.

CPU Interrupts and Interrupt Handling

Interrupts are Interruption to CPU. CPU is a busy taskmaster. Any subsystem requiring the attention of the CPU generates Interrupt. INTERRUPT (INT) is both a control and status signal to the CPU. Generally, the memory subsystem does not generate Interrupt. The Interruption alters the CPU execution flow. Recognizing and servicing Interrupts is fundamental to any processor design.

Interrupts are generated by I/O subsystem, CPU or Software. Interrupts categorization and details are shown below.


Interrupts
Hardware Interrupts
Software Interrupts
External or I/O Interrupts
Internal Interrupts (Exceptions)
Program Generated
Generated by an I/O Subsystem
To communicate successful completion of I/O
Ready status for Interrupt driven I/O
An error in I/O
A malfunction detected by CPU hardware generates Interrupt (Also called EXCEPTION or TRAP)
Ex: Divide Overflow, Arithmetic errors, Invalid Instructions, Address range errors, etc conditions as set in hardware
Page Fault is not an error but a method to replace page in Virtual memory concept
Debug mode operation of programs causes an Interrupt
Generated by User programs using INT instruction
Useful for System CALLS
This feature is required while writing Device drivers

Interrupts Categorization

An Interrupt handling mechanism has the following characteristics:

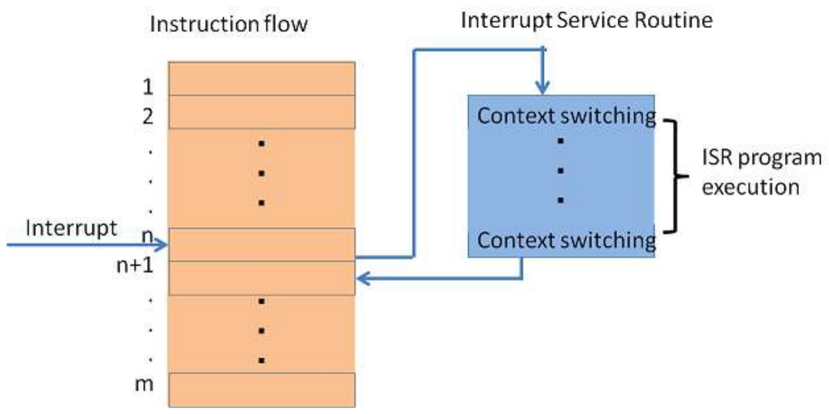
- Able to identify that an interrupt is pending

- Identify the type of interrupt
- Identify the source of Interrupt
- Control the interrupt mechanism
- Identify or calculate the ISR address
- Context switching mechanism

Interrupt Service Routine

Most of the Interrupts are asynchronous. When an Interrupt occurs,

- The execution flow control is transferred to the corresponding **Interrupt Service Routine (ISR)**
- Once the ISR is completed, the original execution flow restarts from the interrupted point as shown in below figure. ISR is also called **Interrupt Handler**.
- Interrupts are recognized and serviced by CPU at the end of the current instruction execution.
- Context switching of the Processor happens while breaking for ISR. This part is common to all kinds of interrupts.
- What happens in the ISR routine is specific to the Interrupt. For example, if divide overflow error, the ISR block may send a msg to the user informing the error and terminate the program. If page fault, page swapping happens. If I/O interrupt, status reading first happens; based on the status further course is taken up.

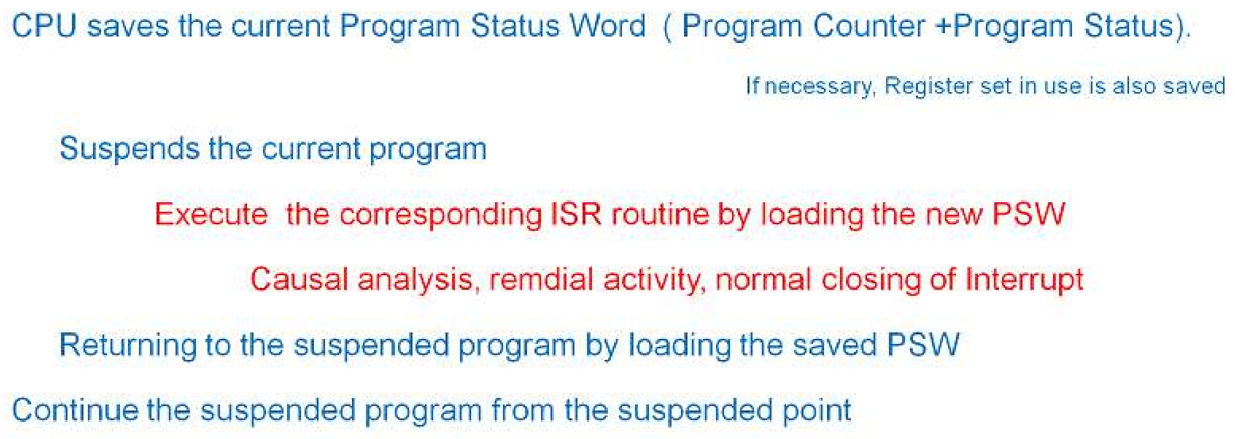


Template of ISR

In the case of I/O interrupt, the cause is analysed in the ISR. Any remedial action to be taken is also part of the ISR. For example, if an interrupt is caused for informing an error in the device, the same needs to be informed to the user and any possible retry to be done for recovery from the error.

Context switching

Essentially, an Interrupt alters the flow of the program execution. Context switching is about the CPU taking necessary steps to store the current status of the CPU so that on return the CPU status is restored for resumption. Context switching helps the CPU to switch processes or tasks and is an essential feature supported by the Operating System.



CPU saves the current Program Status Word (Program Counter +Program Status).

If necessary, Register set in use is also saved

Suspends the current program

Execute the corresponding ISR routine by loading the new PSW

Causal analysis, remdial activity, normal closing of Interrupt

Returning to the suspended program by loading the saved PSW

Continue the suspended program from the suspended point

Identifying the ISR Location

By now you would have guessed the increasing complexity of information regarding Interrupt handling especially the puzzle about the CPU getting ISR location address. The ISR location depends on the source of Interrupt. The interrupting device or process provides the CPU with this information about ISR. This information is provided in the form of type code, or a vector, or an address where the vector can be found. Generally, the type codes are translated into a vector by the processor. Interrupt vectors are either the address of the ISR or pointer in a vector table where the relevant ISR address is found. Although the vector table composition is implementation-specific, an example of a vector table is shown below.

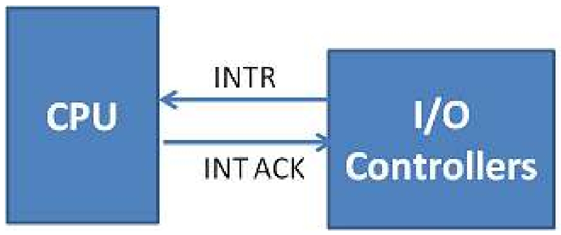
Interrupt Vector	**ISR Address**
1 (Page Fault)	1234
2 (Divide Overflow)	1658
3 (IO Interrupt)	2340
*	*
*	*
*	*
n (Floating Point Error)	5240

Interrupt Identification

In the below figure the I/O Interrupt is conveyed to CPU by asserting the signal INTR. When the CPU recognizes the INTR, it returns the signal INTR ACK as an acknowledgement. From figure, you may understand that the line INTR is a summation of interrupts from all the I/O controllers. The issue is how is the interrupting device identified. The possibilities for a pending interrupt are

- Only one of the I/O controller has raised the Interrupt
- More than one I/O controller has raised the Interrupt

The hardware is designed to handle the second case of as many I/O controllers may raise interrupts asynchronously but conveyed to CPU on a single line.



Interrupt and Interrupt Acknowledgment between I/O and CPU

The Points to be noted in identifying the interrupting device are:

- The CPU services all the interrupts one by one as it finds the chance to service the interrupt.

- Amongst the I/O controllers, Interrupt priority is assigned in the hardware. So the highest priority one gets serviced first and cleared of pending interrupt. This method is called **Daisy Chaining**. Generally, the slow speed device controllers are assigned lower interrupt priority. Starvation may be a possibility. The INTR and INT ACK signals are passed through all the controllers in the order of the pre-assigned priority
- Yet the question of which device is not answered. The identification is done by one of the following methods
    - **Polling** - As part of the ISR by reading the various status registers, the interrupting device is identified. In this case, the I/O ISR always starts from a particular common vector address. Polling is a Software method.
    - **Vectored Interrupts** – Extra signal lines between CPU and I/O are run to indicate the INT CODE (type code) of the interrupt. Every controller is assigned an INT CODE. For example, if there are eight I/O controllers, 3 lines are used so that the INT CODE is presented in binary coded form. This type code, as said earlier is useful in generating the vector for ISR.
- Multiple Interrupts Handling – Two possibilities exist:
    - Simultaneously more than one interrupt could be pending necessitating some priority assignment and identification mechanism. We have just discussed this case.
    - During an ISR, another interrupt could come in. A decision to deal with (as Nested Interrupt) or to defer (Masking the Interrupts) is required. Not all interrupts are mask able.

Internal interrupts have higher priority over I/O interrupts. Internal interrupts are vectored interrupts. In the case of software interrupts too, the instruction code will help identify the ISR vector.

UNIT – 6
I/O INTERFACE & BUS ARCHITECTURE

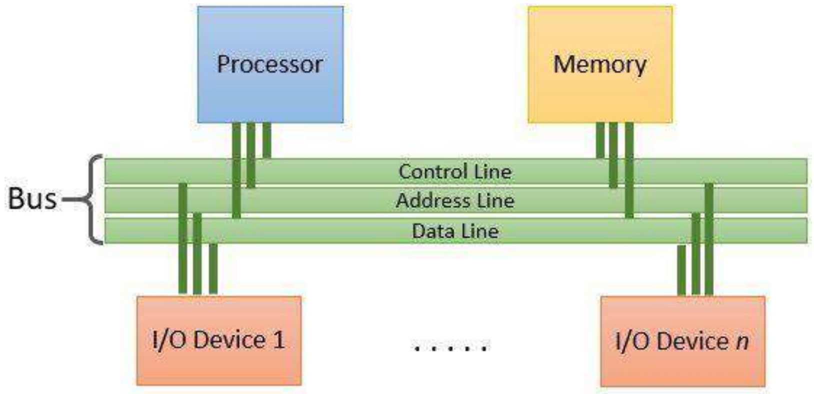
Bus

- Bus is a group of wires that connects different components of the computer. It is used for transmitting data, control signal and memory address from one component to another. A bus can be 8 bit, 16 bit, 32 bit and 64 bit. A 32 bit bus can transmit 32 bit information at a time. A bus can be internal or external.

- The bus in the computer is the *shared transmission medium*. This means multiple components or devices use the same bus structure to transmit the information signals to each other. At a time only one pair of devices can use this bus to communicate with each other successfully. If multiple devices transmit the information signal over the bus at the same time the signals overlap each other and get jumbled.

Bus Structure

A system bus has typically from fifty to hundreds of distinct lines where each line is meant for a certain function. These lines can be categories into three functional groups i.e., data lines, address lines, and control lines.



Bus Structure

1. Data Lines

- Data lines coordinate in transferring the data among the system components. The data lines are collectively called data bus. A data bus may have 32 lines, 64 lines, 128 lines, or even more lines. The number of lines present in the data bus defines the *width* of the data bus.

- Each data line is able to transfer only one bit at a time. So the number of data lines in a data bus determines how many bits it can transfer at a time. The performance of the system also depends on the width of the data bus.

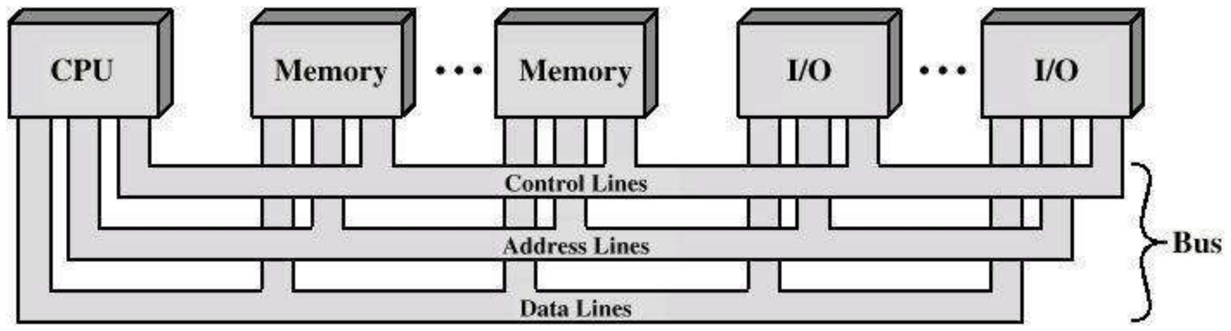
2. Address Lines

- The content of the address lines of the bus determines the source or destination of the data present on the data bus. The number of address lines together is referred to as address bus. The number of address lines in the address bus determines its *width*.
- The width of the address bus determines the memory capacity of the system. The content of address lines is also used for addressing I/O ports.
- The higher-order bits determine the bus module and the lower ordered bits determine the address of memory locations or I/O ports.
- Whenever the processor has to read a word from the memory it simply places the address of the corresponding word on the address line.

3. Control Lines

- The address lines and data lines are shared by all the components of the system so there must some means to control the use and access of data and address lines.
- The control signals placed on the control lines. The control bus lines are used to control access to address and data lines of the bus. The control signal consists of the *command* and *timing information.*

- Here the command in the control signal specify the *operation* that has to be performed. And the timing information over the control signals specify till when the data and address information is valid.



A Typical Bus Interconnection

The control signal can be classified into:

- **Memory Write:** This command causes the data on the data bus to be placed over the addressed memory location.
- **Memory Read:** This command causes the data on the addressed memory location to be placed on the data bus.
- **I/O Write:** The command over this control line causes the data on the data bus to be placed over the addressed I/O port.
- **I/O Read:** The command over this control line causes the data from the addressed I/O port to be placed over the data bus.
- **Transfer ACK:** This control line indicates the data has been received from the data bus or is placed over the data bus.
- **Bus Request:** This control line indicates that the component has requested control over the bus.
- **Bus Grant:** This control line indicates that the bus has been granted to the requesting component.
- **Interrupt Request:** This control line indicates that interrupts are pending.
- **Interrupt ACK:** This control line provides acknowledgment when the pending interrupt is serviced.
- **Clock:** This control line is used to synchronize the operations.

- **Reset:** The bit information issued over this control line initializes all the modules.

If a component connected to the bus wishes to send data to another connected component, it first has to acquire control over the bus, and then it can transfer the data to another component over the bus. The same happen when a component request data from another component.

During the transfer of data between two components, one component act as a master and other act as a slave. The device initiating the data transfer is referred to as *master* and usually, it is a processor, or sometimes it may be some other device or component. The component addressed by the master component is referred to as a *slave*.

Multiple bus architecture

Most computer systems employ multiple bus architecture, which can use different buses depending on the data rate of the connected device.

Two typical bus architectures based on the above considerations are:

- Traditional bus architecture
- High performance bus architecture

Traditional bus architecture

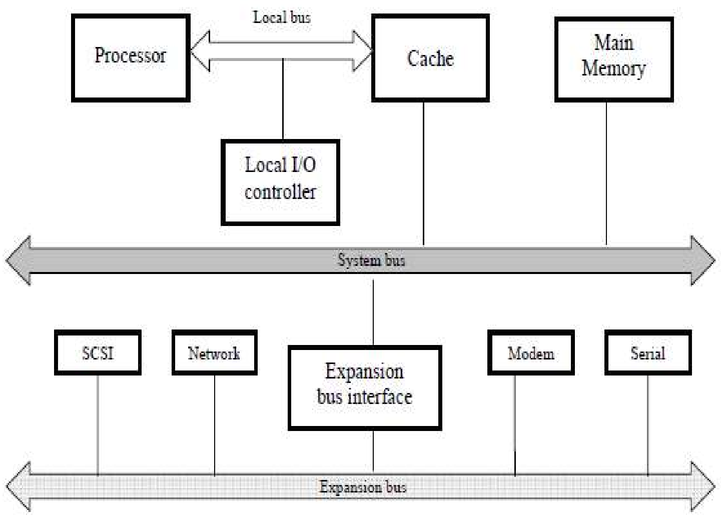


Figure 1.3.3 Traditional bus configuration

- The above figure shows the traditional bus architecture of interconnecting different modules of the computer.
- This consists of three buses: local bus, system bus & expansion bus.
- The local bus connects the processor to the cache memory as well as one or more local devices.
- The cache controller also connects to a system bus to which all main memory modules are attached. The use of cache structure insulates the processor from accessing the main memory frequently.
- This way the main memory can be connected to the system bus instead of the local bus & I/O transfers to and from the main memory will not interfere with the processor activity.
- The I/O controllers can be directly connected onto the system bus, but a more efficient way is to connect them through an expansion bus.
- This arrangement not only allows the system to support a wide variety of I/O devices, but also insulates the processor to memory traffic from I/O traffic.
- The expansion bus can be utilized to attach different I/O devices such as SCSI (small computer system interface), LAN (local area network such as 10 mbps Ethernet), and serial interface to support printers or scanners.

High performance bus architecture

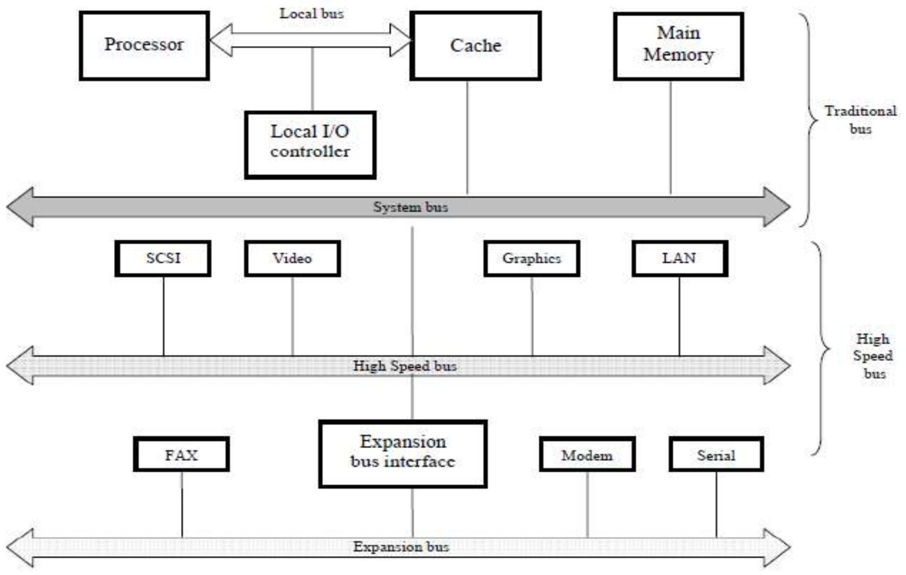


Figure 1.3.4 High speed bus configuration

- The above figure shows the high performance integrated bus architecture. It is similar to the traditional bus architecture, this too contains a local bus that connects the processor to a cache controller.
- This in turn is connected to the main memory through the system bus.
- The cache controller is integrated into a bridge or buffering device that connects to the high speed bus. This bus supports connections to high speed LANs, such as fast Ethernet at 100 mbps, video & graphics work station controllers & interface controllers to local peripheral controllers like SCSI & Fire Wire.
- The Fire Wire is a high speed bus specially designed for high capacity I/O devices.
- The lower speed devices are still supported by the expansion bus with an interface buffering traffic between the expansion & high speed buses.

This way the high speed devices are more closely integrated with the processor through the high speed bus & at the same time leaving the processor independent. Additionally the speed differences between the processor, the high speed bus & the single line connections are easily managed without pulling down the overall performance. Also the processor & high speed bus become independent, making design changes easy.

Basic Parameters Of Bus Design

Basic Parameters Of Bus Design Table	
Type of bus	• Dedicated • Multiplexed
Width of bus	• Data • Address • Control
Method of arbitration	• Centralized • Distributed
Method of timing	• Synchronous • Asynchronous
Type of data transfer	• Read/write • Read-modify-write • Read-after-write • Block transfer

1) Bus Types

Bus lines can be classified into two generic types: dedicated & multiplexed.

❖ **Dedicated**

- A line is permanently assigned either to one function.
- An example of functional dedication is the use of separate dedicated address and data line.

❖ **Multiplexed**

- Using the same lines for multiple purpose.
- Eg:- Address and data information may be transmitted over the same set of lines.
- At the beginning of the data transfer the address is placed on the bus and the address valid line is activated.
- The address is then remove from the same bus line is used for data transfer.
- Multiple buses are also called **Physical Dedication, which** connects each module by a subset of bus lines.

2) <u>Method of Arbitration</u>

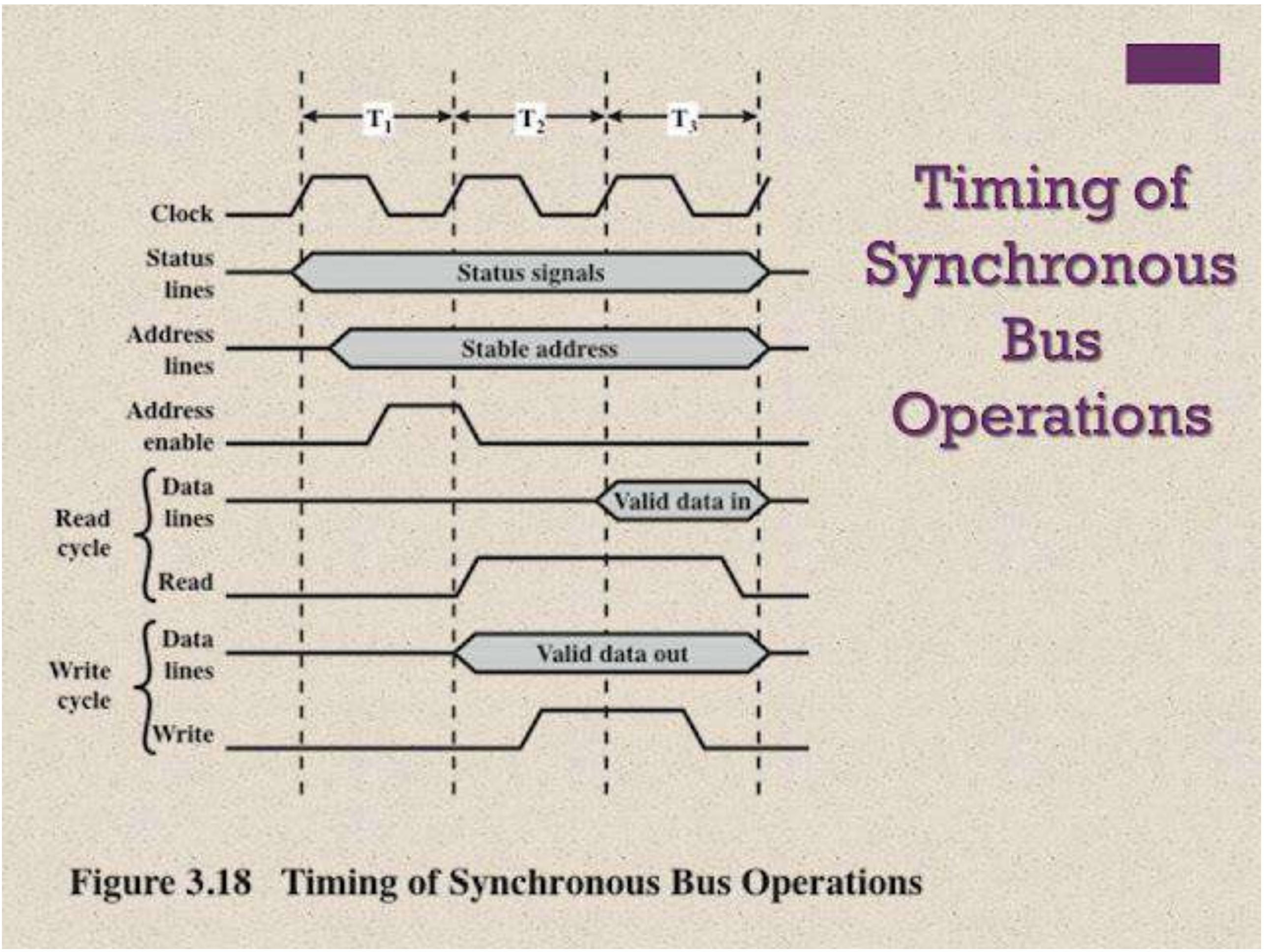
As there are the many modules connected to the buses, more than one module may need control of the bus.

- ❖ **Centralized**
  - A single hardware device called the **bus controller or arbiter** is responsible for allocating time on the bus.
  - The device may be a separate or a part of a processor.
- ❖ **Distributed**
  - There is no centralized controllers.
  - Each module contains assess control logic and the modules act together to share the bus.

In both centralized & distributed schemes, one device, the processor or an I/O module, is designated as master. The master will then initiate a data transfer to/ from some other device, which will act as the slave.

3) <u>Timing</u>

- ❖ **Synchronous Timing**
  - In this method of timing, a clock controls the occurrence of events on the bus.
  - Bus includes a clock line upon which a clock transmits a regular sequence of alternating 1's and 0's of equal duration.
  - A single 1-0 transition is referred to as a clock cycle or bus cycle.
  - All other devices on the bus can read the clock line.
  - All events start at the beginning of a clock cycle.
  - For read operation, a read command is issued at the start of the second cycle. The addressed memory module places the data on the data lines after a delay on one clock cycle.
  - For write operation, the processor puts the data on the data lines at the start of the second cycle & once the data lines stabilize, issues a write command.
  - The memory module copies the information available on the data lines in the third clock cycle.



Timing of Synchronous Bus Operations

Figure 3.18 Timing of Synchronous Bus Operations

❖ **Asynchronous Timing**

- The occurrence of one event on a bus follows and depends on the occurrence of a previous event.
- In this method, the processor places the address & the status signals on the bus. After these signals stabilize, it issues a read command, signifying the presence of valid address & control signals.
- The addressed memory responds by placing the data on the data lines. After the data lines stabilize, the memory module asserts the acknowledge line to signal to the processor that the data are available.

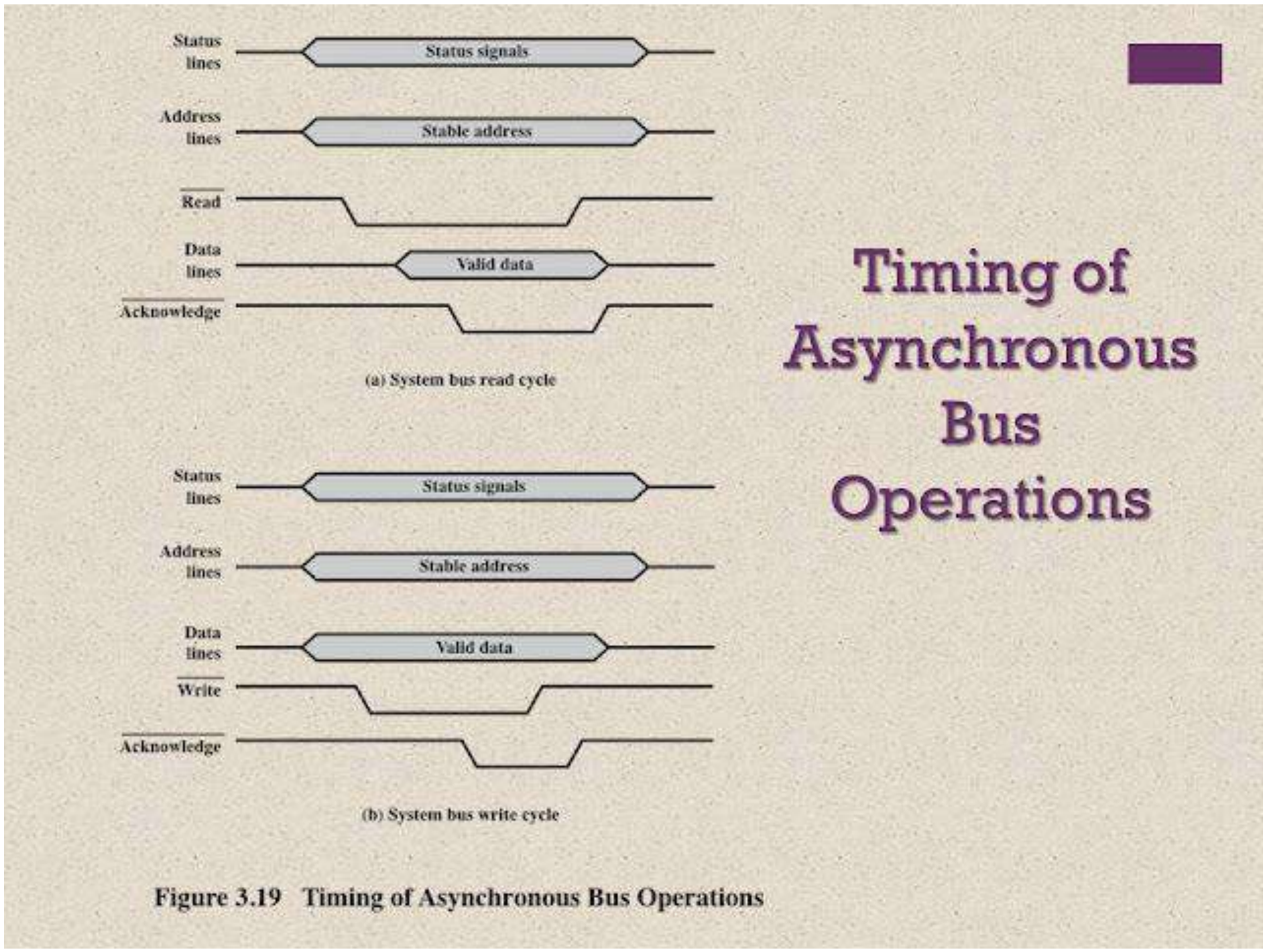


Figure 3.19 Timing of Asynchronous Bus Operations

4) Bus Width

- The width of data bus has an impact on the data bus has an impact on the system performance.
- The wider data bus, the greater number of bit transferred at one time.
- The wider address bus, the greater range of location that can be referenced.

5) Data Transfer Type

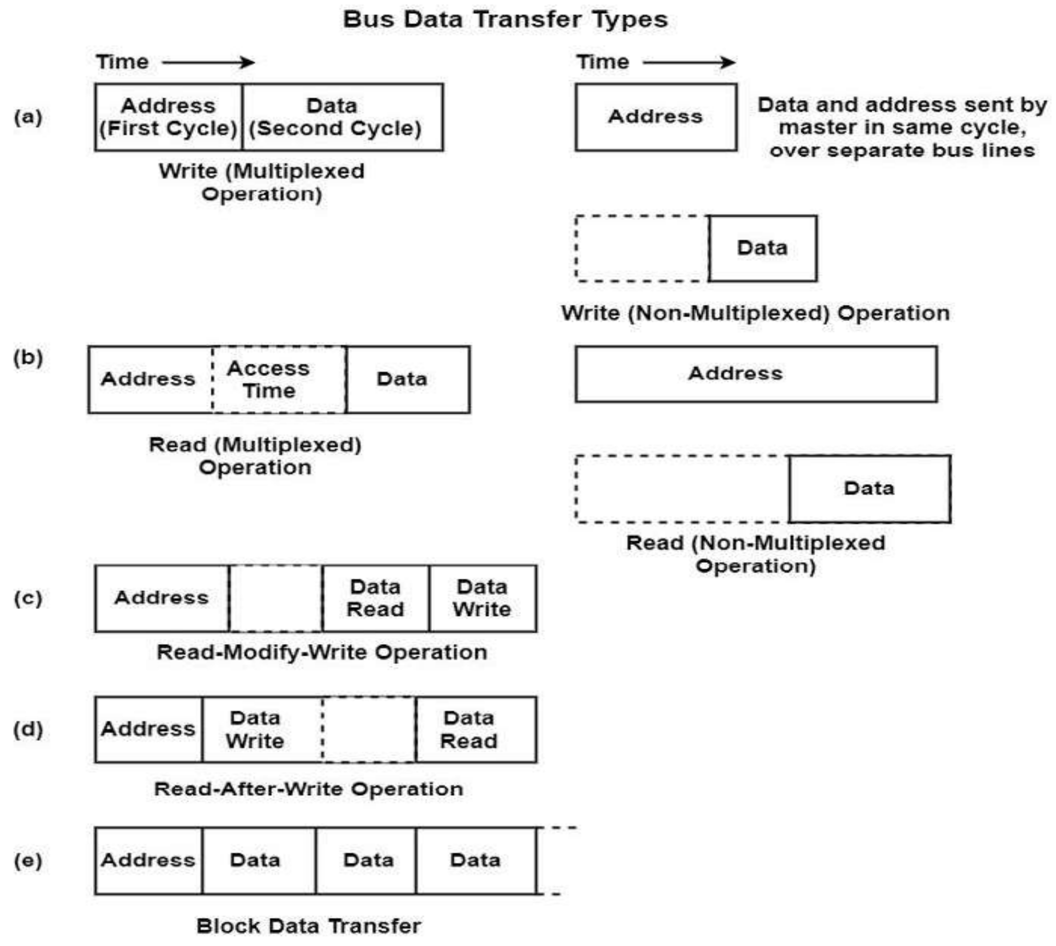
A bus provides several data transfer types as shown in the figure.

- All buses provide both **write** (master to slave) and **read** (slave to master) assigns.
- **Read-Modify-Write:** A **read-modify-write** is simply a read followed immediately by a write to the same address.

- **Read-After-Write : A read-after-write** is also an invisible operation consisting of a write followed immediately by a read from the same address (for error checking purposes).

Some bus systems support a **Block Data Transfer** operation in which

- One address cycle is followed by n data cycles.
- The First data item is transferred to or from the specified address.
- The Remaining data items are transferred to or from the subsequent addresses.


Bus Data Transfer Types
Time
Time
(a)
Address (First Cycle)
Data (Second Cycle)
Write (Multiplexed Operation)
Address
Data and address sent by master in same cycle, over separate bus lines
Data
Write (Non-Multiplexed) Operation
(b)
Address
Access Time
Data
Read (Multiplexed) Operation
Address
Data
Read (Non-Multiplexed Operation)
(c)
Address
Data Read
Data Write
Read-Modify-Write Operation
(d)
Address
Data Write
Data Read
Read-After-Write Operation
(e)
Address
Data
Data
Data
Block Data Transfer

<u>SCSI (Small Computer System Interface)</u>

- SCSI is an acronym for Small Computer Systems Interface. It is an ANSI standard that has become one of the leading I/O buses in the computer industry.

- The SCSI bus is a parallel bus, which comes in a number of variants. The oldest and most used is an 8 bit wide bus, with single-ended signals, carried on 50 wires.
- Modern designs also use 16 bit wide buses, with differential signals. Because of these various options, the SCSI connector may have 50, 68, or 80 pins. The maximum transfer rate in commercial devices that are currently available varies from 5 megabytes/s to 160 megabytes/s.
- The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa.
- A controller connected to a SCSI bus is one of two types - an initiator or a target. An initiator has the ability to select a particular target and to send commands specifying the operations to be performed.
- The disk controller operates as a target. It carries out the commands it receives from the initiator.
- Data transfers on the SCSI bus are always controlled by the target controller. To send a command to a target, an initiator requests control of the bus.
- Then the controller starts a data transfer operation to receive a command from the initiator.
- The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

    1. The SCSI controller, acting as an initiator, contends for control of the bus.
    2. When the 'initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
    3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
    4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.

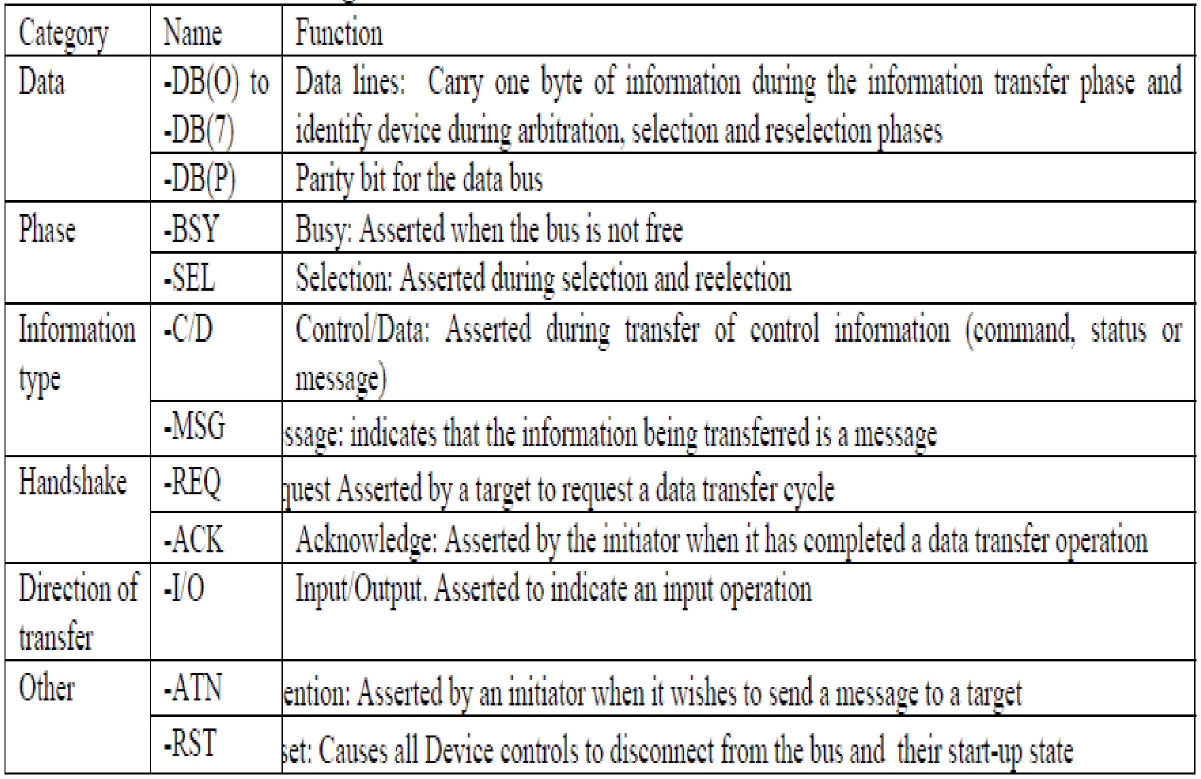
5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it res elects the initiator controller, thus restoring the suspended connection.
6. The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.
7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator, as before. At the end of this transfer, the logical connection between the two controllers is terminated.
8. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
9. The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

Bus signals

The main phases involved in the operation of the SCSI bus are:

- Arbitration
- Selection
- Information Transfer
- Reselection

Table 5.2 The SCSI bus signal



Category	Name	Function
Data	-DB(O) to -DB(7)	Data lines: Carry one byte of information during the information transfer phase and identify device during arbitration, selection and reselection phases
	-DB(P)	Parity bit for the data bus
Phase	-BSY	Busy: Asserted when the bus is not free
	-SEL	Selection: Asserted during selection and reelection
Information type	-C/D	Control/Data: Asserted during transfer of control information (command, status or message)
	-MSG	ssage: indicates that the information being transferred is a message
Handshake	-REQ	quest Asserted by a target to request a data transfer cycle
	-ACK	Acknowledge: Asserted by the initiator when it has completed a data transfer operation
Direction of transfer	-I/O	Input/Output. Asserted to indicate an input operation
Other	-ATN	ention: Asserted by an initiator when it wishes to send a message to a target
	-RST	set: Causes all Device controls to disconnect from the bus and their start-up state

Arbitration

The bus is free when the -BSY signal is in the inactive (high voltage) state. Any controller can request the use of the bus while it is in this state. Since two or more controllers may generate such a request at the same time, an arbitration scheme must be implemented. A controller requests the bus by asserting the -BSY signal and by asserting its associated data line to identify itself. Each controller in the bus is assigned a fixed priority, with controller 7 having highest priority. When -BSY becomes active, all controllers that are requesting the bus examine the data lines and determine whether a higher priority device is requesting the bus at the same time. The controller using the highest numbered line realizes that it has won the arbitration.

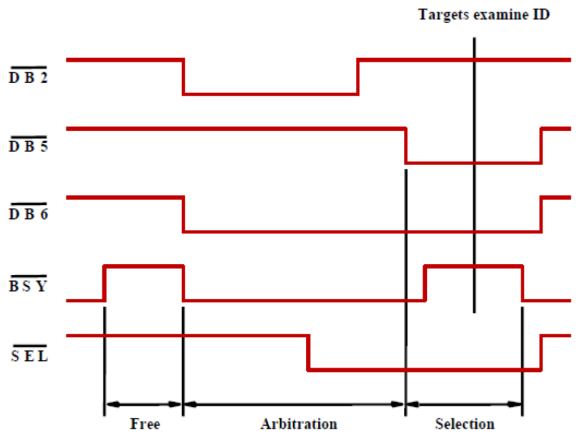


Figure 5.32 Arbitration and selection on the SCSI bus. Device6 wins arbitration and selects device2.

Selection

Having won arbitration, controller 6 continues to assert -BSY and -DB6 (its address). It indicates that it wishes to select controller 5 by asserting the -SEL and then the -DB5 line. The selected target controller responds by asserting -BSY. This informs the initiator that the connection it is requesting has been established, so that it may remove the address information from data lines. The selection process is now complete, and the target controller is asserting -BSY.

Information Transfer

The information transferred between two controllers may consist of commands from the initiator to the target, status responses from the target to the initiator, or data being transferred to or from the I/O device. Handshake signaling is used to control information transfers. The target asserts -I/O during an input operation (target to initiator) and it asserts -C/D to indicate that information is a command or status response. At the end of the transfer, the target controller releases -BSY signal, thus freeing the bus for use by other devices.

Reselection

When a logical connection is suspended and the target is ready to restore it, the target must first gain control of the bus. It starts an arbitration cycle, and after winning arbitration, it selects the initiator. The initiator is now asserting -BSY. The initiator waits for short period after being selected to make sure that target has asserted -BSY, and then releases the -BSY line. The connection between the two controllers has now been reestablished, with the target in control of the bus as required for data transfer to proceed. The SCSI standard defines the structure and contents of various types of packets that the controllers exchange to handle different situations.

USB (Universal Serial Bus)

- Universal Serial Bus (USB) is a simple and low cost mechanism to connect the devices such as keyboards, mouse, cameras, speakers, printer and display devices to the computer.
- The USB supports two speeds of operation, called low-speed (1.5 megabits/s) and full speed (12 megabits/s). The most recent revision of the bus specification (USB 2.0) introduced a third speed of operation, called high-speed (480 megabits/s).
- The USB is quickly gaining acceptance in the market place, and with the addition of the high-speed capability it may well become the interconnection method of choice for most computer devices.
- The USB has been designed to meet several key objectives:

    1. Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer
    2. Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections
    3. Enhance user convenience through a "plug-and-play" mode of operation

Port limitation:

- Only a few ports are provided in a typical computer. To add new ports, a user must open the computer box to gain access to the internal expansion bus and install a new interface card.
- The user may also need to know how to configure the device and the software.

- An objective of the USB is to make it possible to add many devices to a computer system at any time, without opening the computer box.

Device Characteristics:

- The different kinds of devices may be connected to a computer cover a wide range of functionality.
- The speed, volume, and timing constraints associated with data transfers to and from such devices vary significantly.
- In the case of a keyboard, one byte of data is generated every time a key is pressed, which may happen at any time. These data should be transferred to the computer promptly.
- Since the event of pressing a key is not synchronized to any other event in a computer system, the data generated by the keyboard are called asynchronous.
- Furthermore, the rate at which the data are generated is quite low. It is limited by the speed of the human operator to about 100 bytes per second, which is less than 1000 bits per second.

Plug-and-play:

- The plug-and-play feature means that a new device, such as an additional speaker, can be connected at any time while the system is operating.
- The system should detect the existence of this new device automatically, identify the appropriate device-driver soft- ware and any other facilities needed to service that device, and establish the appropriate addresses and logical connections to enable them to communicate.
- The plug-and-play requirement has many implications at all levels in the system, from the hardware to the operating system and the applications software.
- One of the primary objectives of the design of the USB has been to provide a plug-and-play capability.

USB Architecture:

- A serial transmission format has been chosen for the USB because a serial bus satisfies the low-cost and flexibility requirements. Clock and data information are encoded together and transmitted as a single signal. Hence, there are no limitations on clock frequency or distance arising from data skew. Therefore, it is possible to provide a high data transfer bandwidth by using a high clock frequency. As pointed out earlier, the USB offers three bit rates, ranging from 1.5 to 480 megabits/s, to suit the needs of different I/O devices.
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure shown in Figure 5.33.
- Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root hub connects the entire tree to the host computer. The leaves of the tree are the I/O devices being served (for example, keyboard, speaker, or digital TV), which are called functions in USB terminology.

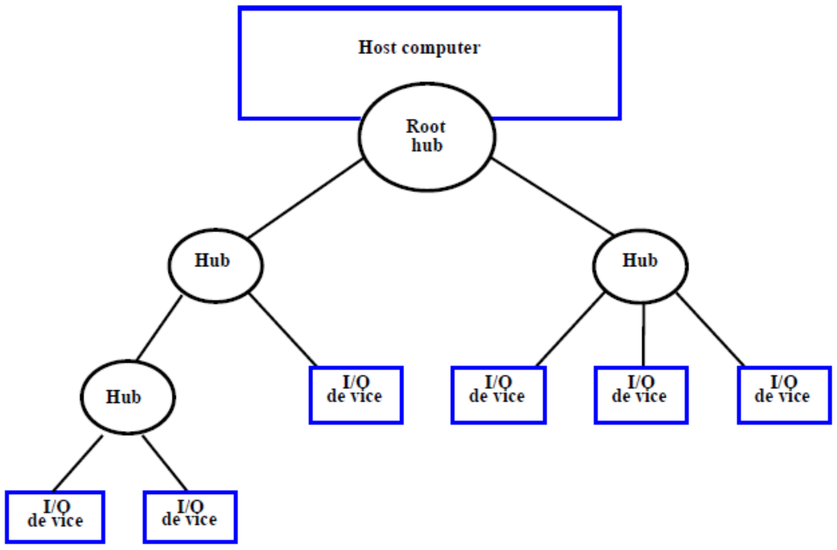


Figure 5.33 Universal Serial Bus tree structure.

- The tree structure enables many devices to be connected while using only simple point –to-point serial links. Each hub has a number of ports where devices may be connected, including other hubs.

- The USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.
- The tree makes it possible to connect a large no. of devices to a computer through a few ports (the root hub) & each i/o devices is connected through a serial point-to-point connection. This is an important consideration in facilitating the plug & play feature.
- The USB operates strictly on the basis of polling. A device may send a message only in response to a poll message from the host. Hence, upstream messages do not encounter conflicts or interfere with each other, as no two devices can send messages at the same time. This restriction allows hubs to be simple, Low-cost devices.

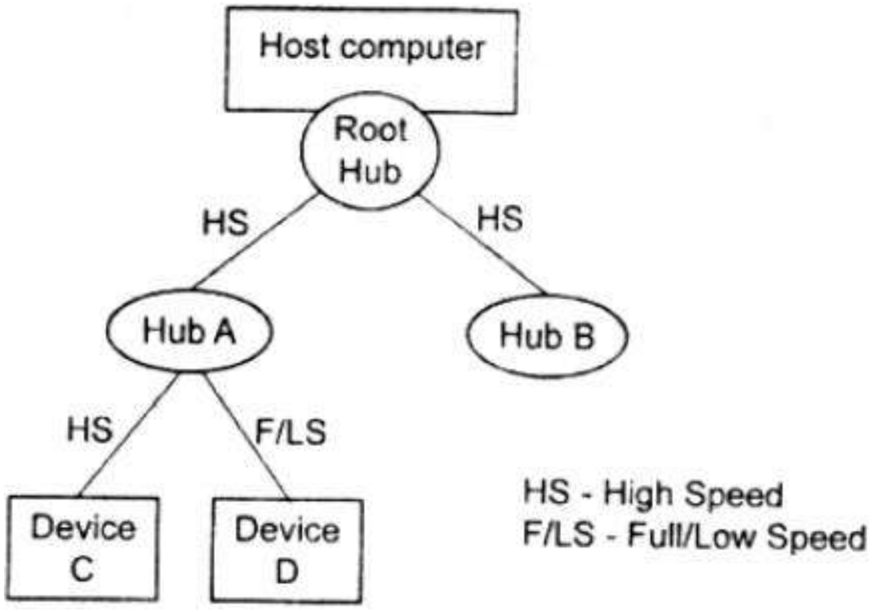


Figure 5.34 USB Split bus operations.

- Consider the situation in Figure 5.34. Hub A is connected to the root hub by a high-speed link. This hub serves one high-speed device, C, and one low-speed device, D. Normally, a message to device D would be sent at low speed from the root hub. At 1.5 megabits/s, even a short message takes several tens of microseconds.

- For the duration of this message, no other data transfers can take place, thus reducing the effectiveness of the high-speed links and introducing unacceptable delays for high-speed devices.
- The purpose of the USB software is to provide bidirectional communication links between application software and I/O devices. These links are called pipes. Any data entering at one end of a pipe is delivered at the other end. Issues such as addressing, timing, or error detection and recovery are handled by the USB protocols.

Addressing:

- I/O devices are normally identified by assigning them a unique memory address. In fact, a device usually has several addressable locations to enable the software to send and receive control and status information and to transfer data.
- When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device. The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the USB tree.
- Each device on the USB, whether it is a hub or an I/O device, is assigned a 7 -bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.
- A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily. When a device is first connected to a hub, or when it is powered on, it has the address 0.
- The hardware of the hub to which this device is connected is capable of detecting that the device has been connected, and it records this fact as part of its own status information.
- Periodically, the host polls each hub to collect status information and learn about new devices that may have been added or disconnected.

USB Protocol:

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information.
- The information transferred on the USB can be divided into two broad categories: control and data.
- Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error. Data packets carry information that is delivered to a device.
- For example, input and output data are transferred inside data packets.
- A packet consists of one or more fields containing different kinds of information. The first field of any packet is called the packet identifier, PID, which identifies the type of that packet. There are 4 bits of information in this field, but they are transmitted twice. The first time they are sent with their true values, and the second time with each bit complemented, as shown in Figure 5.35(a).
- This enables the receiving device to verify that the PID byte has been received correctly.

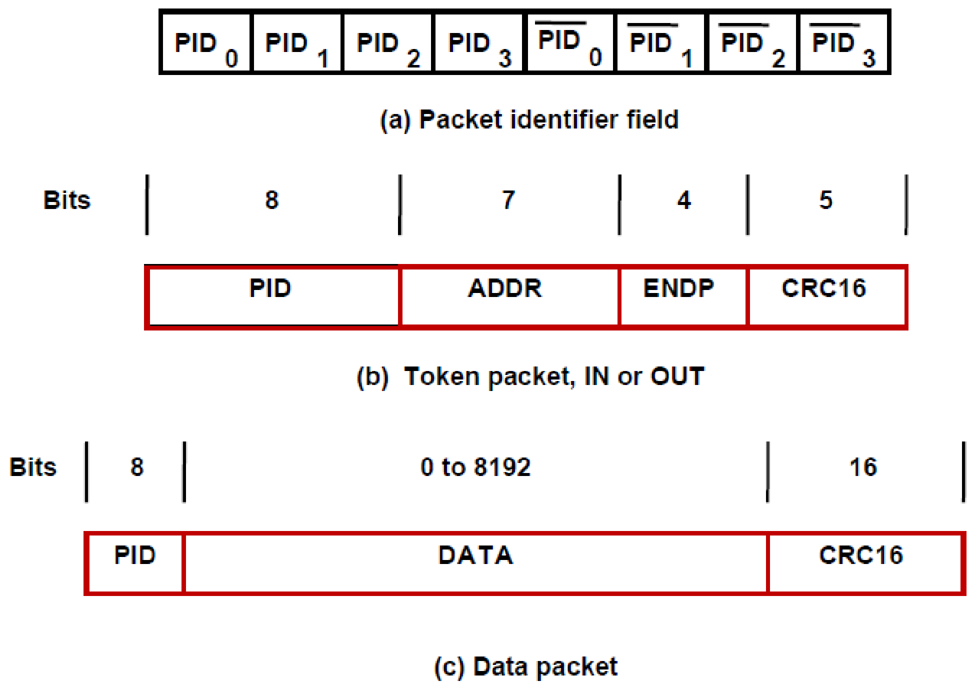

PID_0 PID_1 PID_2 PID_3 $\overline{PID}_0$ $\overline{PID}_1$ $\overline{PID}_2$ $\overline{PID}_3$
(a) Packet identifier field
Bits 8 7 4 5
PID ADDR ENDP CRC16
(b) Token packet, IN or OUT
Bits 8 0 to 8192 16
PID DATA CRC16
(c) Data packet

Figure 5.35. USB packet format.

- The four PID bits identify one of 16 different packet types. Some control packets, such as ACK (Acknowledge), consist only of the PID byte. Control packets used for controlling data transfer operations are called token packets.
- They have the format shown in Figure 5.35(b). A token packet starts with the PID field, using one of two PID values to distinguish between an IN packet and an OUT packet, which control input and output transfers, respectively.
- The PID field is followed by the 7 -bit address of a device and the 4-bit endpoint number within that device. The packet ends with 5 bits for error checking, using a method called cyclic redundancy check (CRC). The CRC bits are computed based on the contents of the address and endpoint fields.
- By performing an inverse computation, the receiving device can determine whether the packet has been received correctly.
- Data packets, which carry input and output data, have the format shown in Figure 4.45c. The packet identifier field is followed by up to 8192 bits of data, then 16 error-checking bits. Three different PID patterns are used to identify data packets, so that data packets may be numbered 0, 1, or 2.
- Note that data packets do not carry a device address or an endpoint number. This information is included in the IN or OUT token packet that initiates the transfer.
- Consider an output device connected to a USB hub, which in turn is connected to a host computer. An example of an output operation is shown in Figure 5.36.
- The host computer sends a token packet of type OUT to the hub, followed by a data packet containing the output data. The PID field of the data packet identifies it as data packet number o.
- The hub verifies that the transmission has been error free by checking the error control bits, and then sends an acknowledgment packet (ACK) back to the host. The hub forwards the token and data packets downstream. All I/O devices receive this sequence of packets, but only the device that recognizes its address in the token packet accepts the data in the packet that follows.
- After verifying that transmission has been error free, it sends an ACK packet to the hub.

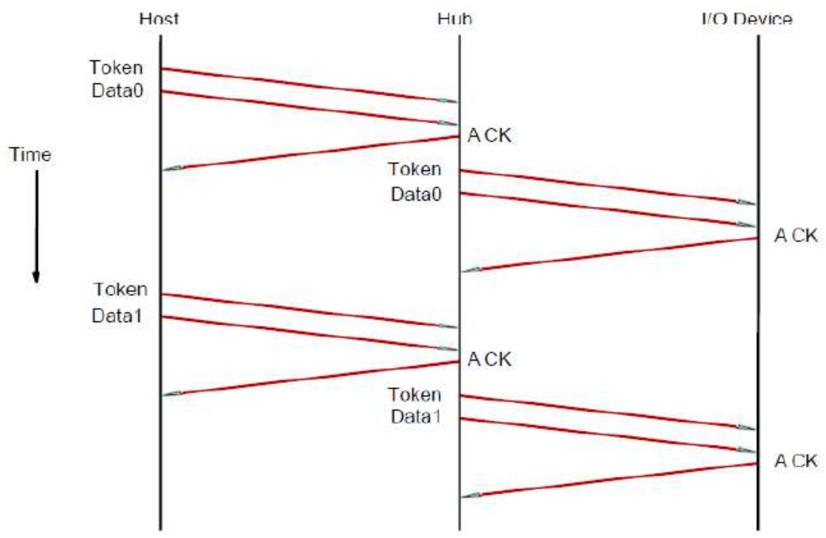


Figure 5.36 An output transfer

- Successive data packets on a full-speed or low-speed pipe carry the numbers 0 and 1, alternately. This simplifies recovery from transmission errors. If a token, data, or acknowledgment packet is lost as a result of a transmission error, the sender resends the entire sequence.
- By checking the data packet number in the PID field, the receiver can detect and discard duplicate packets. High-speed data packets are sequentially numbered 0, 1, 2, 0, and so on. Input operations follow a similar procedure.

Electrical characteristics:

The cables used for USB connections consist of four wires. Two are used to carry power, 5 V and Ground. Thus, a hub or an I/O device may be powered directly from the bus, or it may have its own external power connection. The other two wires are used to carry data.

UNIT – 7
PARALLEL PROCESSING

Parallel Processing

Parallel processing can be described as a class of techniques which enables the system to achieve simultaneous data-processing tasks to increase the computational speed of a computer system.

A parallel processing system can carry out simultaneous data-processing to achieve faster execution time. For instance, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.

The primary purpose of parallel processing is to enhance the computer processing capability and increase its throughput, i.e. the amount of processing that can be accomplished during a given interval of time.

Parallel processing mechanism:-

A number of parallel processing mechanisms have been developed in uniprocessor computers. We identify them in the following six categories.

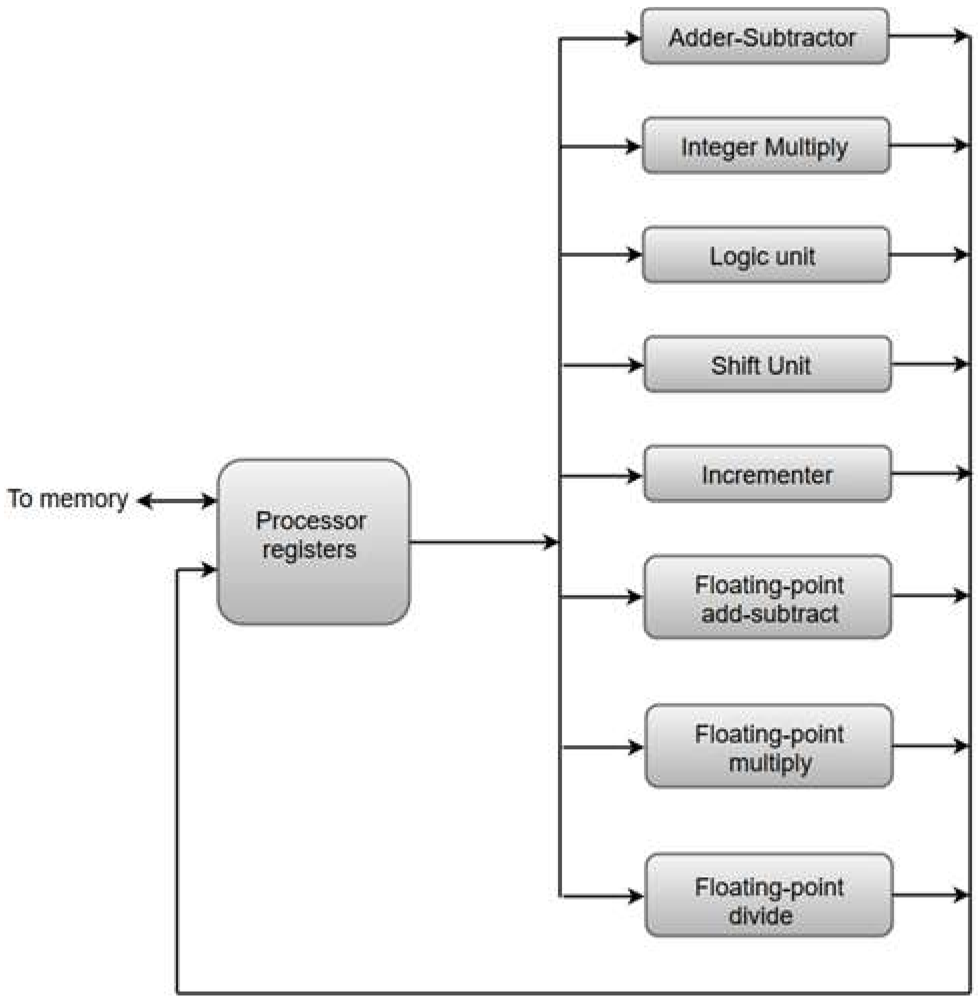
- Multiplicity of functional units.
- Parallelism and pipeline within the CPU
- Overlapped CPU and I/O
- Use of hierarchical memory system
- Balancing of subsystem bandwidths.
- Multiprogramming and time sharing.

A parallel processing system can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. The data can be distributed among various multiple functional units.

Parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish between parallel and serial operations by the type of registers used. At the higher level of complexity, parallel processing can be achieved by using multiple functional units that perform many operations simultaneously.

The following diagram shows one possible way of separating the execution unit into eight functional units operating in parallel.

The operation performed in each functional unit is indicated in each block if the diagram:



- The adder and integer multiplier performs the arithmetic operation with integer numbers.
- The floating-point operations are separated into three circuits operating in parallel.

- The logic, shift, and increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented.

Data Transfer Modes of a Computer System

According to the data transfer mode, computer can be divided into 4 major groups:

1. SISD
2. SIMD
3. MISD
4. MIMD

SISD (Single Instruction Stream, Single Data Stream)

- It represents the organization of a single computer containing a control unit, processor unit and a memory unit. Instructions are executed sequentially.
- It can be achieved by pipelining or multiple functional units.

SIMD (Single Instruction Stream, Multiple Data Stream)

- It represents an organization that includes multiple processing units under the control of a common control unit.
- All processors receive the same instruction from control unit but operate on different parts of the data.
- They are highly specialized computers.
- They are basically used for numerical problems that are expressed in the form of vector or matrix. But they are not suitable for other types of computations

MISD (Multiple Instruction Stream, Single Data Stream)

- It consists of a single computer containing multiple processors connected with multiple control units and a common memory unit.
- It is capable of processing several instructions over single data stream simultaneously.
- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD (Multiple Instruction Stream, Multiple Data Stream

- It represents the organization which is capable of processing several programs at same time.
- It is the organization of a single computer containing multiple processors connected with multiple control units and a shared memory unit.
- The shared memory unit contains multiple modules to communicate with all processors simultaneously.
- Multiprocessors and multicomputer are the examples of MIMD. It fulfills the demand of large scale computations.

PIPELINING

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as **pipeline processing**.

Pipelining is a technique where multiple instructions are overlapped during execution.

Pipelining organizes the execution of the **multiple instructions simultaneously**. Pipelining improves the **throughput** of the system. In pipelining the instruction is divided into the subtasks. Each subtask performs the dedicated task.

The instruction is divided into 5 subtasks: **instruction fetch**, **instruction decode**, **operand fetch**, **instruction execution** and **operand store**. The instruction fetch subtask will only perform the instruction fetching operation, instruction decode subtask will only be decoding the fetched instruction and so on the other subtasks will do.

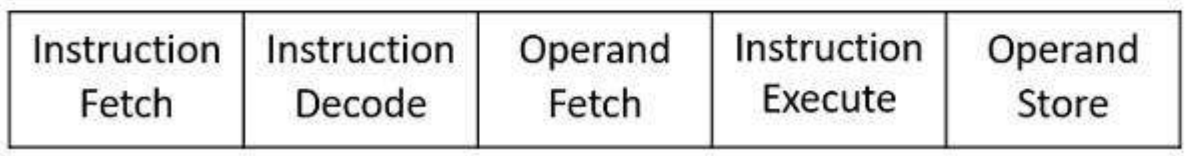
For **example**, take a car manufacturing plant. At the first stage, the automobile chassis is prepared, in the next stage workers add body to the chassis, further, the engine is installed, then painting work is done and so on.

The group of workers after working on the chassis of the first car don't sit idle. They start working on the chassis of the next car. And the next group take the chassis of the car and add body to it. The same thing is repeated at every stage, after finishing the work on the current car body they take on next car body which is the output of the previous stage.

Here, though the first car is completed in several hours or days, due to the assembly line arrangement it becomes possible to have a new car at the end of an assembly line in every clock cycle.

Similarly, the concept of **pipelining** works. The output of the first pipeline becomes the input for the next pipeline. It is like a set of data processing unit connected in series to utilize processor up to its maximum.

An instruction in a process is divided into 5 subtasks likely,



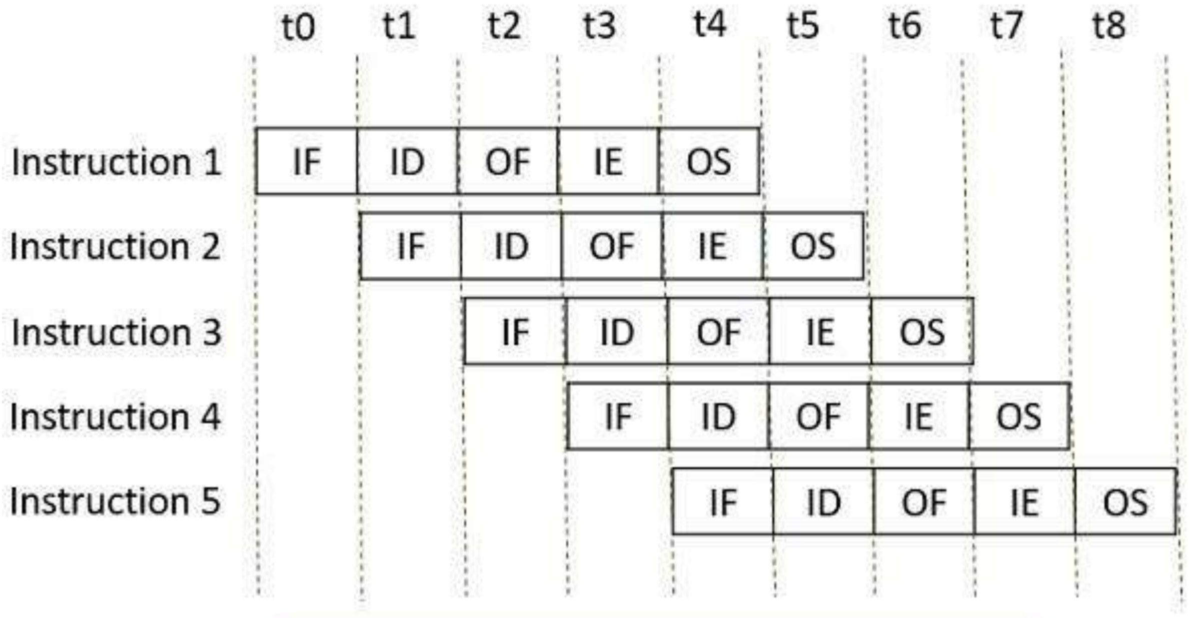
Instruction Fetch	Instruction Decode	Operand Fetch	Instruction Execute	Operand Store

1. In the first subtask, the instruction is fetched.
2. The fetched instruction is decoded in the second stage.
3. In the third stage, the operands of the instruction are fetched.
4. In the fourth, arithmetic and logical operation are performed on the operands to execute the instruction.

5. In the fifth stage, the result is stored in memory.

Now, understanding the division of the instruction into subtasks. Let us understand, how the n number of instructions in a process, are pipelined.

Look at the figure below the 5 instructions are pipelined. The first instruction gets completed in 5 clock cycle. After the completion of first instruction, in every new clock cycle, a new instruction completes its execution.



Pipelining of 5 Instructions

Observe that when the Instruction fetch operation of the first instruction is completed in the next clock cycle the instruction fetch of second instruction gets started. This way the hardware never sits idle it is always busy in performing some or other operation. But, no two instructions can **execute** their **same stage** at the **same clock cycle**.

Types of Pipelining

In 1977 Handler and Ramamoorthy classified pipeline processors depending on their functionality.

1. Arithmetic Pipelining

- It is designed to perform high-speed floating-point addition, multiplication and division.
- Here, the multiple arithmetic logic units are built in the system to perform the parallel arithmetic computation in various data format.
- Examples of the arithmetic pipelined processor are Star-100, TI-ASC, Cray-1, Cyber-205.

For example: The input to the Floating Point Adder pipeline is:

```
X = A*2^a

Y = B*2^b
```

Here A and B are mantissas (significant digit of floating point numbers), while **a** and **b** are exponents.
The floating point addition and subtraction is done in 4 parts:

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

Registers are used for storing the intermediate results between the above operations.

2. Instruction Pipelining

- Here, the number of instruction are pipelined and the execution of current instruction is overlapped by the execution of the subsequent instruction. It is also called **instruction lookahead**.

3. Processor Pipelining

- Here, the processors are pipelined to process the **same data stream**.
- The data stream is processed by the first processor and the result is stored in the memory block. The result in the memory block is accessed by the second processor.
- The second processor reprocesses the result obtained by the first processor and the passes the refined result to the third processor and so on.

4. Unifunction Vs. Multifunction Pipelining

- The pipeline performing the precise function every time is unifunctional pipeline.
- On the other hand, the pipeline performing multiple functions at a different time or multiple functions at the same time is multifunction pipeline.

5. Static vs Dynamic Pipelining

- The static pipeline performs a fixed-function each time. The static pipeline is unifunctional.
- The static pipeline executes the same type of instructions continuously. Frequent change in the type of instruction may vary the performance of the pipelining.
- Dynamic pipeline performs several functions simultaneously. It is a multifunction pipelining.

6. Scalar vs Vector Pipelining

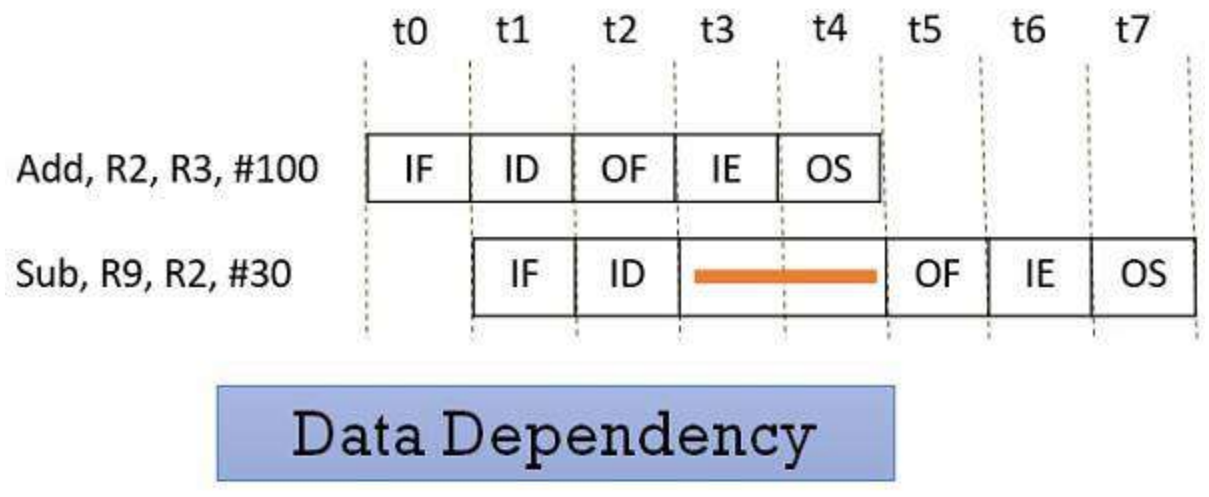
- Scalar pipelining processes the instructions with scalar operands.
- The vector pipeline processes the instruction with vector operands.

Pipelining Hazards

Whenever a pipeline has to stall due to some reason it is called pipeline hazards. Below we have discussed four pipelining hazards.

1. Data Dependency

Consider the following two instructions and their pipeline execution:



In the figure above, you can see that result of the **Add** instruction is stored in the register **R2** and we know that the final result is stored at the end of the execution of the instruction which will happen at the clock cycle **t4**.

But the **Sub** instruction need the value of the register **R2** at the cycle **t3**. So the Sub instruction has to **stall** two clock cycles. If it doesn’t stall it will generate an incorrect result. Thus depending of one instruction on other instruction for data is **data dependency**.

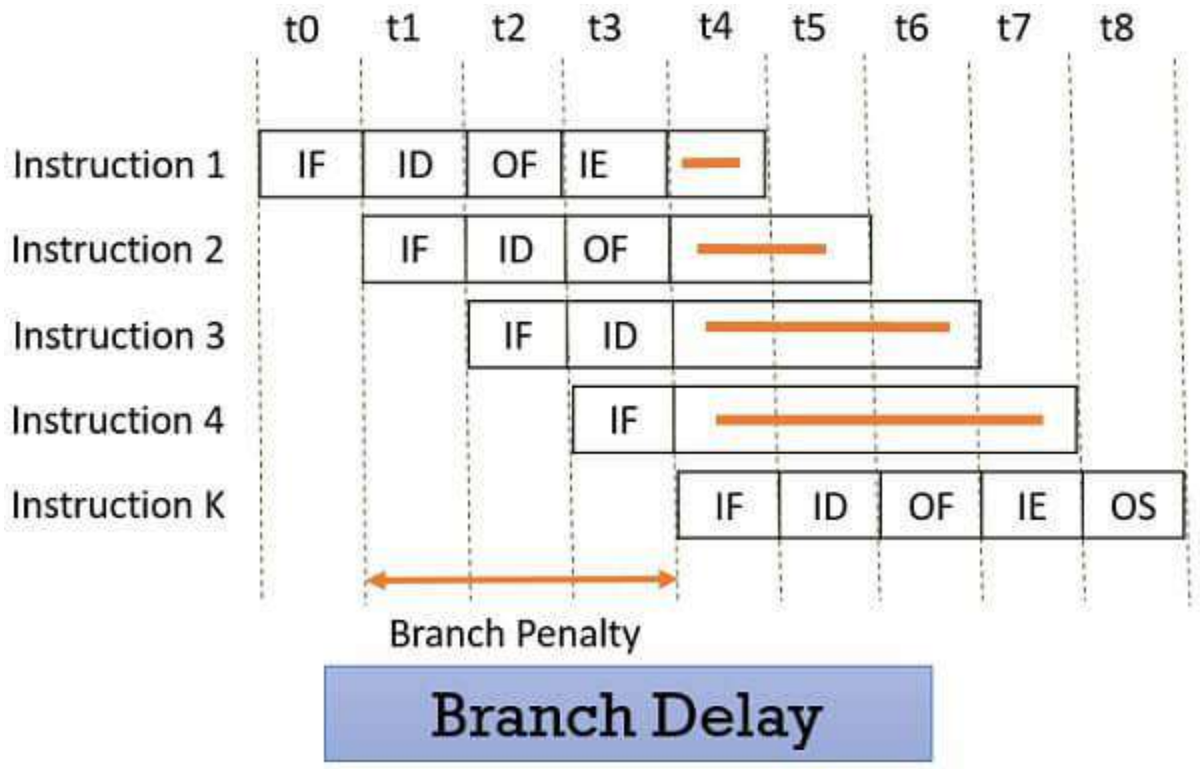
2. Memory Delay

When an instruction or data is required, it is first searched in the cache memory if not found then it is a **cache miss**. The data is further searched in the memory which may take ten or more cycles. So, for that number of cycle the pipeline has to stall and this is a **memory delay** hazard. The cache miss, also results in the delay of all the subsequent instructions.

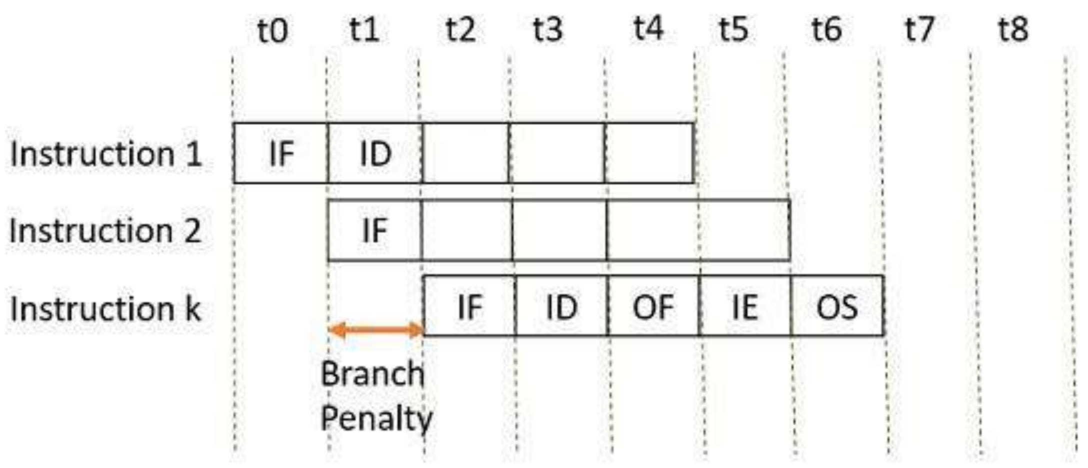
3. Branch Delay

Suppose the four instructions are pipelined I_1, I_2, I_3, I_4 in a sequence. The instruction I_1 is a branch instruction and its target instruction is I_k. Now, processing starts and instruction I_1 is fetched, decoded and the target address is computed at the 4th stage in cycle t3.

But till then the instructions I_2, I_3, I_4 are fetched in cycle 1, 2 & 3 before the target branch address is computed. As I_1 is found to be a branch instruction, the instructions I_2, I_3, I_4 has to be discarded because the instruction I_k has to be processed next to I_1. So, this delay of three cycles 1, 2, 3 is a **branch delay**.



Prefetching the target branch address will reduce the branch delay. Like if the target branch is identified at the decode stage then the branch delay will reduce to 1 clock cycle.



Branch Delay when Target Branch is Determined at Decode stage

4. Resource Limitation

If the two instructions request for accessing the same resource in the same clock cycle, then one of the instruction has to stall and let the other instruction to use the resource. This stalling is due to **resource limitation**. However, it can be prevented by adding more hardware.

Advantages

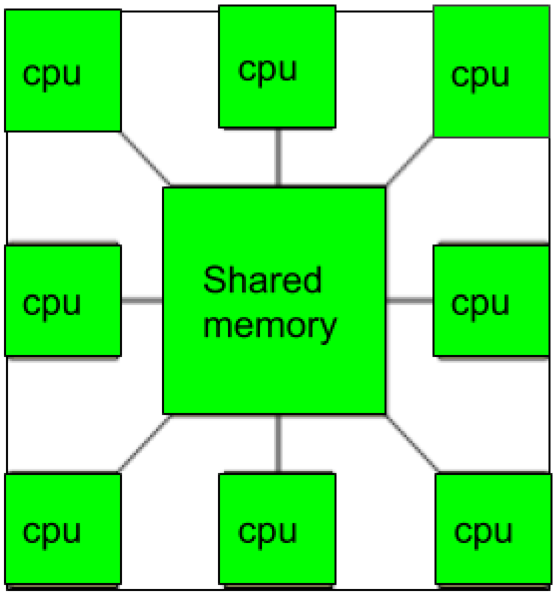
- Pipelining improves the throughput of the system.
- In every clock cycle, a new instruction finishes its execution.
- Allow multiple instructions to be executed concurrently.

Disadvantages

- The design of pipelined processor is complex and costly to manufacture.
- The instruction latency is more.

Multiprocessor

- A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM.
- The main objective of using a multiprocessor is to boost the system's execution speed, with other objectives being fault tolerance and application matching.
- There are two types of multiprocessors, one is called shared memory multiprocessor and another is distributed memory multiprocessor.
- In shared memory multiprocessors, all the CPUs shares the common memory but in a distributed memory multiprocessor, every CPU has its own private memory.



Applications of Multiprocessor

1. As a uniprocessor, such as single instruction, single data stream (SISD).
2. As a multiprocessor, such as single instruction, multiple data stream (SIMD), which is usually used for vector processing.
3. Multiple series of instructions in a single perspective, such as multiple instruction, single data stream (MISD), which is used for describing hyper-threading or pipelined processors.

4. Inside a single system for executing multiple, individual series of instructions in multiple perspectives, such as multiple instruction, multiple data stream (MIMD).

Benefits of using a Multiprocessor

- Enhanced performance.
- Multiple applications.
- Multi-tasking inside an application.
- High throughput and responsiveness.
- Hardware sharing among CPUs.

Types of Multiprocessors

There are mainly two types of multiprocessors i.e. symmetric and asymmetric multiprocessors.

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.

Communication architecture of a multiprocessor:

- Message Passing
    - Independent address space for every processor
    - Processor communication by means of message passing
    - Processors include private memories
    - Concentrates attention on high-priced, non-local operations
- Shared Memory
    - Processor communication is done by means of a shared address space
    - Processor communication is done by means of shared memory read/write
    - Convenient on small-scale devices
    - Lower latency
    - Non-uniform memory access (NUMA) or symmetric multiprocessing (SMP)

Disadvantages of Multiprocessor Systems

- **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**

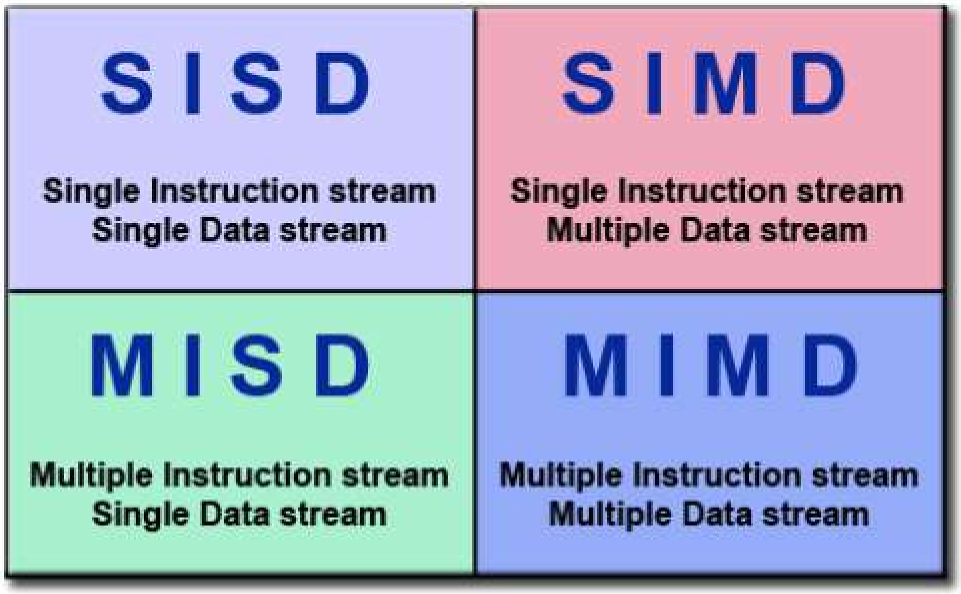
  There are multiple processors in a multiprocessor system that share peripherals, memory etc. So, it is much more complicated to schedule processes and impart resources to processes than in single processor systems. Hence, a more complex and complicated operating system is required in multiprocessor systems.

- **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.

Flynn's Classical Taxonomy

- M.J. Flynn proposed a classification for the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.
- The sequence of instructions read from memory constitutes an **instruction stream**.
- The operations performed on the data in the processor constitute a **data stream**.
- There are different ways to classify parallel computers.
- One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy.
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of Instruction Stream and Data Stream. Each of these dimensions can have only one of two possible states: Single or Multiple.

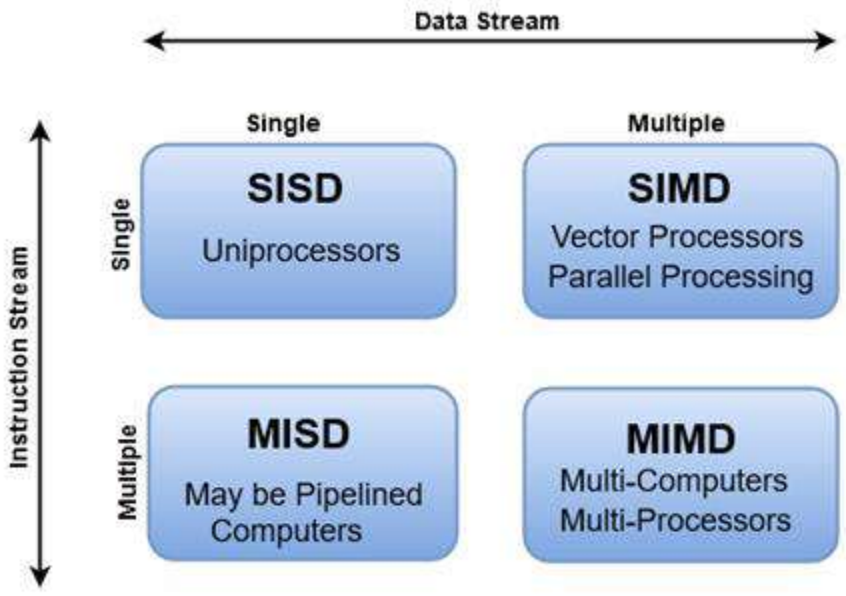

SISD
Single Instruction stream
Single Data stream
SIMD
Single Instruction stream
Multiple Data stream
MISD
Multiple Instruction stream
Single Data stream
MIMD
Multiple Instruction stream
Multiple Data stream

Flynn's classification

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

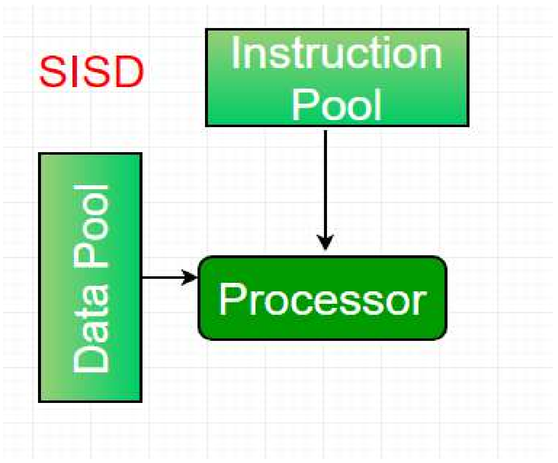
- The matrix below defines the 4 possible classifications according to Flynn:

Flynn's Classification of Computers



Single Instruction, Single Data (SISD):

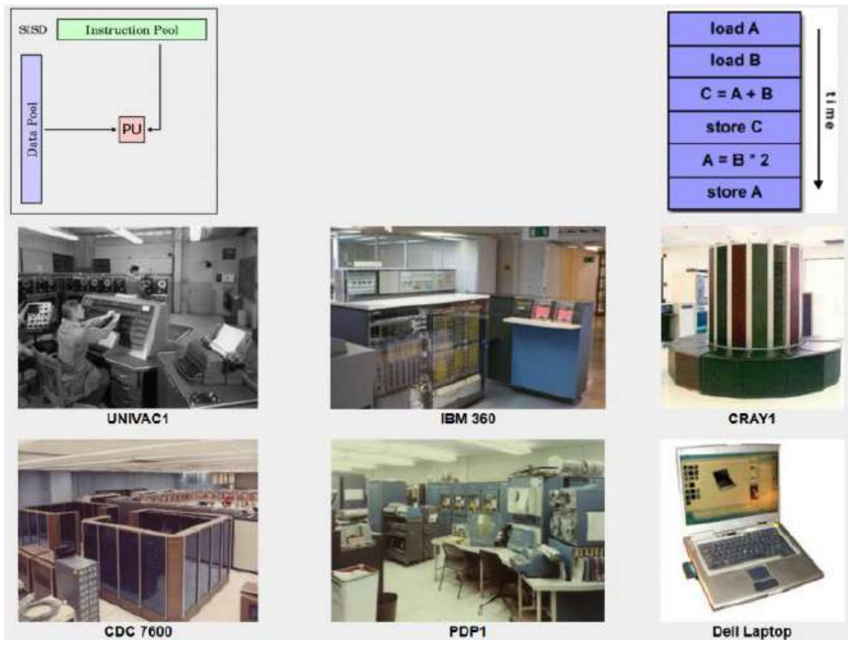
- An SISD computing system is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream.
- In SISD, machine instructions are processed in a sequential manner and computers adopting this model are popularly called sequential computers.
- Most conventional computers have SISD architecture.
- All the instructions and data to be processed have to be stored in primary memory.



- The speed of the processing element in the SISD model is limited (dependent) by the rate at which the computer can transfer information internally.
- Dominant representative SISD systems are IBM PC, workstations.

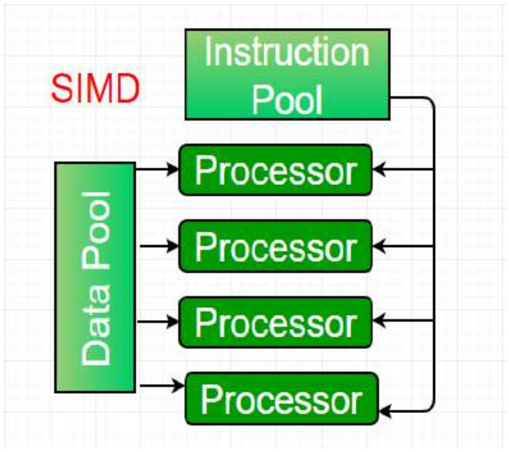
****

- ✓ A serial (non-parallel) computer
- ✓ Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle
- ✓ Single Data: Only one data stream is being used as input during any one clock cycle
- ✓ Deterministic execution
- ✓ This is the oldest type of computer
- ✓ Examples: older generation mainframes, minicomputers, workstations and single processor/core PCs.



Single Instruction, Multiple Data (SIMD):

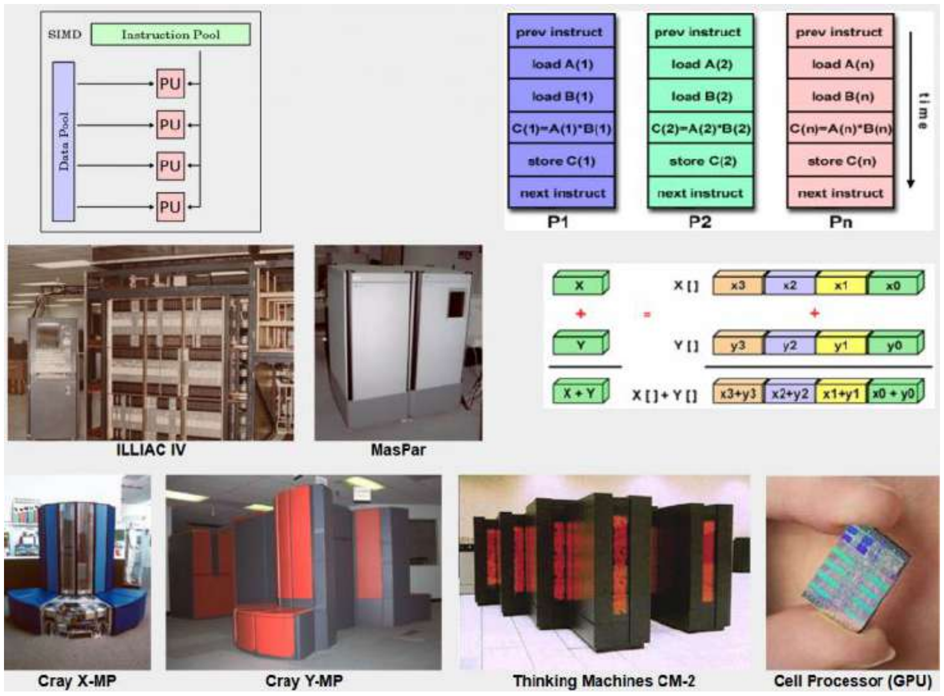
- An SIMD system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams.
- Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations.
- So that the information can be passed to all the processing elements (PEs) organized data elements of vectors can be divided into multiple sets(N-sets for N PE systems) and each PE can process one data set.



- Dominant representative SIMD systems is Cray's vector processing machine.

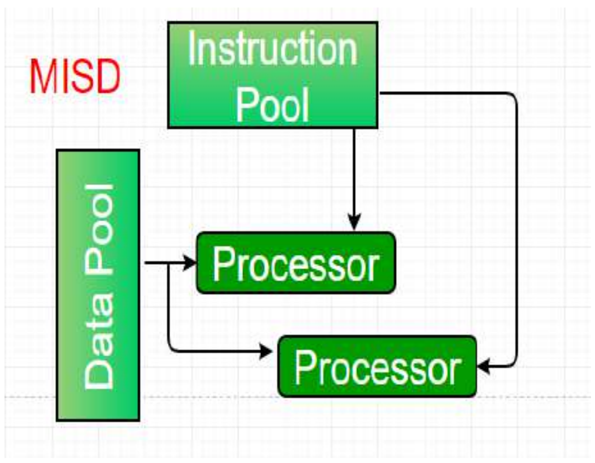
****

- ✓ A type of parallel computer
- ✓ Single Instruction: All processing units execute the same instruction at any given clock cycle
- ✓ Multiple Data: Each processing unit can operate on a different data element
- ✓ Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- ✓ Synchronous (lockstep) and deterministic execution
- ✓ Two varieties: Processor Arrays and Vector Pipelines
- ✓ Examples:
    - o Processor Arrays: Thinking Machines CM-2, MasPar MP-1 & MP-2, ILLIAC IV
    - o Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10
- ✓ Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.



Multiple Instruction, Single Data (MISD):

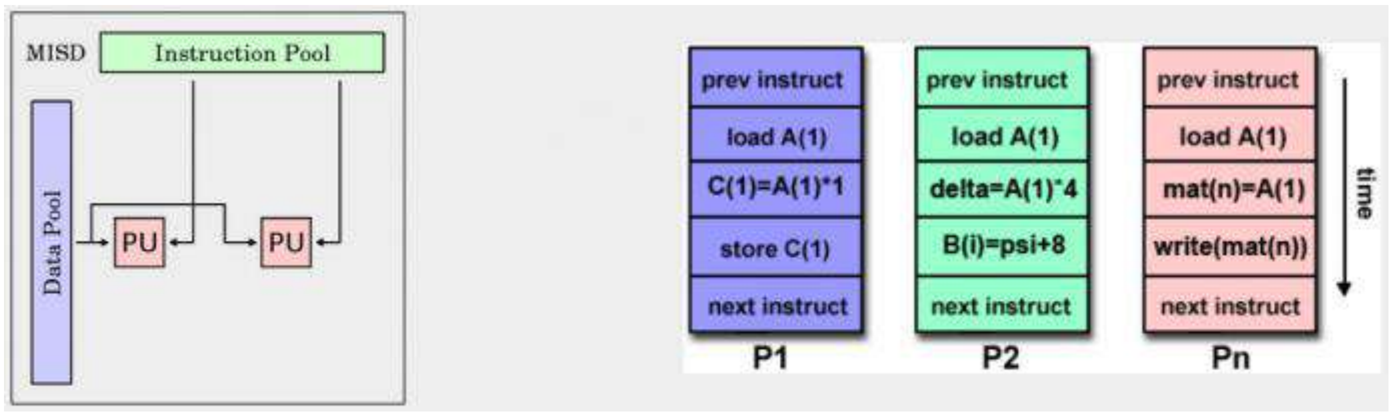
- An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same dataset .



- Example Z = sin(x)+cos(x)+tan(x)
- The system performs different operations on the same data set.
- Machines built using the MISD model are not useful in most of the application, a few machines are built, but none of them are available commercially.

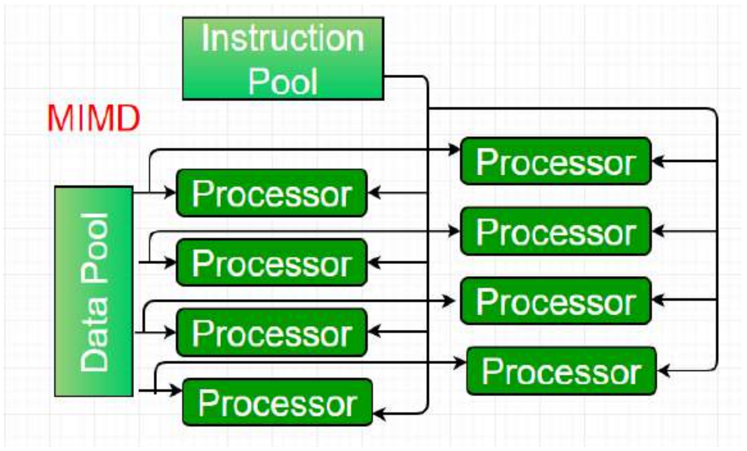
****

- ✓ A type of parallel computer
- ✓ Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.
- ✓ Single Data: A single data stream is fed into multiple processing units.
- ✓ Few (if any) actual examples of this class of parallel computer have ever existed.
- ✓ Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.



Multiple Instruction, Multiple Data (MIMD):

- An MIMD system is a multiprocessor machine which is capable of executing multiple instructions on multiple data sets.
- Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application.
- Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.



- MIMD machines are broadly categorized into **shared-memory MIMD** and **distributed-memory MIMD** based on the way PEs are coupled to the main memory.

<u>Shared Memory MIMD</u>

- In the **shared memory MIMD** model (tightly coupled multiprocessor systems), all the PEs are connected to a single global memory and they all have access to it.
- The communication between PEs in this model takes place through the shared memory, modification of the data stored in the global memory by one PE is visible to all other PEs.
- Dominant representative shared memory MIMD systems are Silicon Graphics machines and Sun/IBM's SMP (Symmetric Multi-Processing).

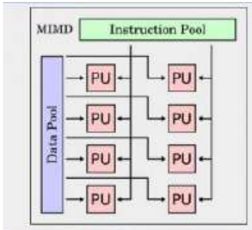
Distributed Memory MIMD

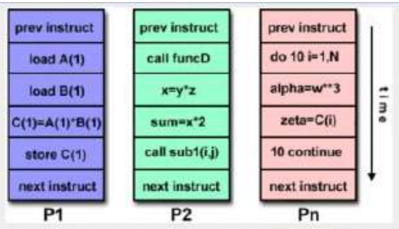
- In **Distributed memory MIMD** machines (loosely coupled multiprocessor systems) all PEs have a local memory.
- The communication between PEs in this model takes place through the interconnection network (the inter process communication channel, or IPC).
- The network connecting PEs can be configured to tree, mesh or in accordance with the requirement.

- The shared-memory MIMD architecture is easier to program but is less tolerant to failures and harder to extend with respect to the distributed memory MIMD model.
- Failures in a shared-memory MIMD affect the entire system, whereas this is not the case of the distributed model, in which each of the PEs can be easily isolated.
- Moreover, shared memory MIMD architectures are less likely to scale because the addition of more PEs leads to memory contention.
- This is a situation that does not happen in the case of distributed memory, in which each PE has its own memory.
- As a result of practical outcomes and user’s requirement , distributed memory MIMD architecture is superior to the other existing models.

****

- ✓ A type of parallel computer
- ✓ Multiple Instruction: Every processor may be executing a different instruction stream
- ✓ Multiple Data: Every processor may be working with a different data stream
- ✓ Execution can be synchronous or asynchronous, deterministic or non-deterministic
- ✓ Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- ✓ Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- ✓ Note: many MIMD architectures also include SIMD execution sub-components





IBM POWER5

HP/Compaq Alphaserver

Intel IA32

AMD Opteron

Cray XT3

IBM BG/L