Computer Science & IT

Computer Organization & Architecture

Comprehensive Theory

with Solved Examples and Practice Questions

MADE EASY
India's Best Institute for IES, GATE & PSUs

CS

Contents

Computer Organization & Architecture

■■■■

Computer Organization & Architecture

Goal of the Subject

Basic understanding of computer organization includes:

- Understanding roles of processors, main memory, and input/output devices.
- Understanding the concept of programs as sequences of machine instructions.
- Understanding the relationship between assembly language and machine language; development of skill in assembly language programming;
- Understanding the relationship between high-level compiled languages and assembly language.
- Understanding arithmetic and logical operations with integer operands, floating-point number systems and operations.
- Understanding simple data path and control designs for processors, memory organization, including cache structures and virtual memory schemes.

Computer Organization & Architecture

INTRODUCTION

In this book we tried to keep the syllabus of Computer Organization around the GATE syllabus. Each topic required for GATE is crisply covered with illustrative examples and each chapter is provided with Student Assignment at the end of each chapter so that the students get the thorough revision of the topics that he/she had studied. This subject is carefully divided into eight chapters as described below.

1. **Basics of Computer Design:** In this chapter we discuss the Computer System, Data storage in the memory and Machine instructions
2. **CPU Design:** In this chapter we discuss the Datapath and Control unit design first is hardwired control unit second is microprogrammed.
3. **Instruction Pipelining:** In this chapter we discuss Performance, Instruction processing, Pipeline design and issue, Pipeline hazards, Pipeline performance analysis and Speedup.
4. **Memory Hierarchy Design:** In this chapter we discuss Primary memory, Associative memory, Address space and Cache memory design.
5. **Input-Output and Secondary Storage:** In this chapter we discuss Interface design, Input-output mode and Secondary memory.
6. **Data Representation:** In this chapter we discuss the Fixed and floating point formate, IEEE floating - point number representation, Computer arithmetic, Adding 2's complement numbers and Multiplying floating-point numbers.

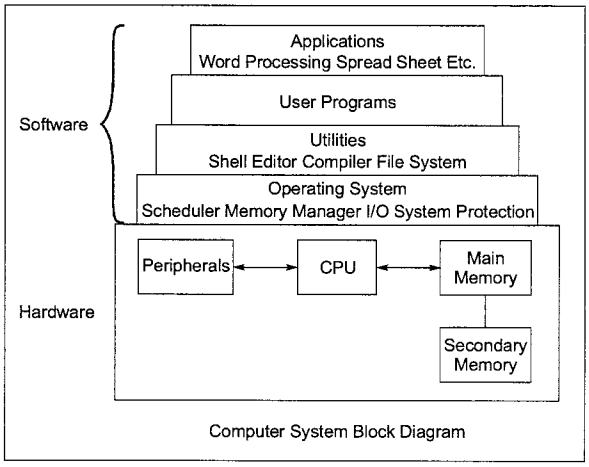
■■■■

CHAPTER 01

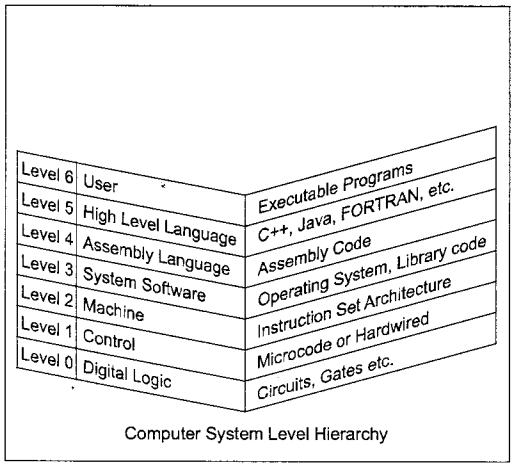
Basics of Computer Design

1.1 Computer System

Computer system is divided into two functional entities: Hardware and Software.



Computer System Block Diagram



Computer System Level Hierarchy

- **Hardware:** Lowest level in a computer are all the electronic circuits and physical devices from which it is built.

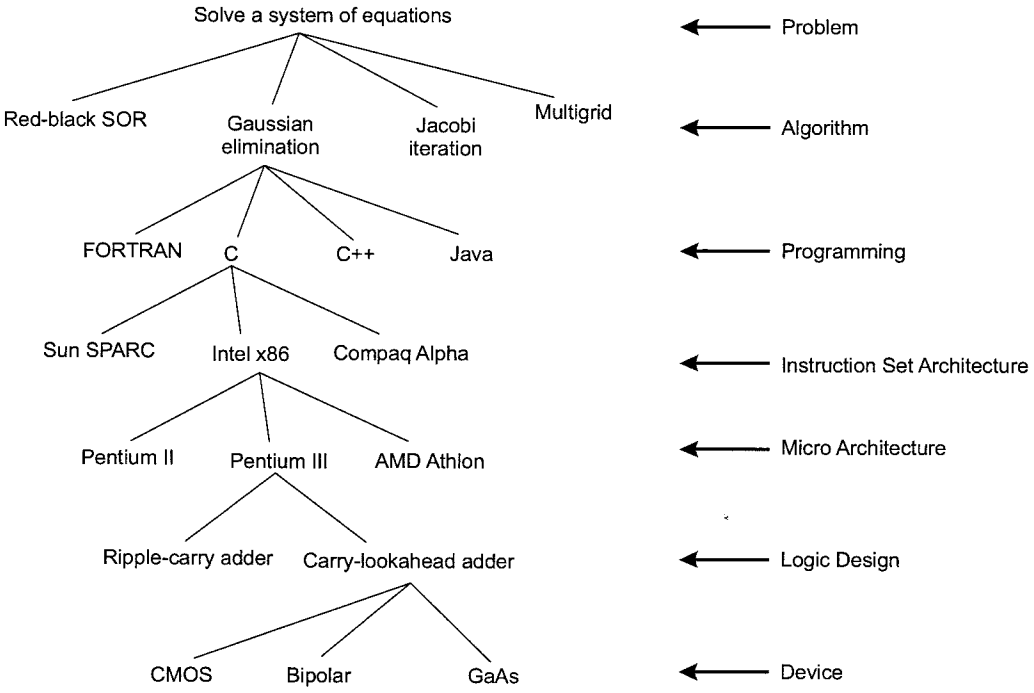
  Hardware consisting of its physical devices (CPU, memory, bus, storage devices, ...)
- **Software:** Sequences of instructions and data that make computers do useful work.

  Software, consisting of the programs it has (Operating system, applications, utilities, ...)

  Program is a sequence of instructions for a particular task.
- Operating system is set of programs included in system software package and Link between hardware and user needs.

1.1.1 Layers of Abstraction

- **Problem Statement:** stated using "natural language". It may be ambiguous are imprecise.
- **Algorithm:** step-by-step procedure, guaranteed to finish. It is definiteness, effective computability, and finiteness.
- **Program:** Express the algorithm using a computer language such as high-level language and low-level language.
- **Instruction Set Architecture (ISA):** It specifies the set of instructions the computer can perform using data types and addressing modes.
- **Micro-architecture:** It is detailed organization of a processor implementation.
- **Logic Circuits:** Combine basic operations to realize micro-architecture.
- **Devices:** Which is properties of materials and manufacturability.



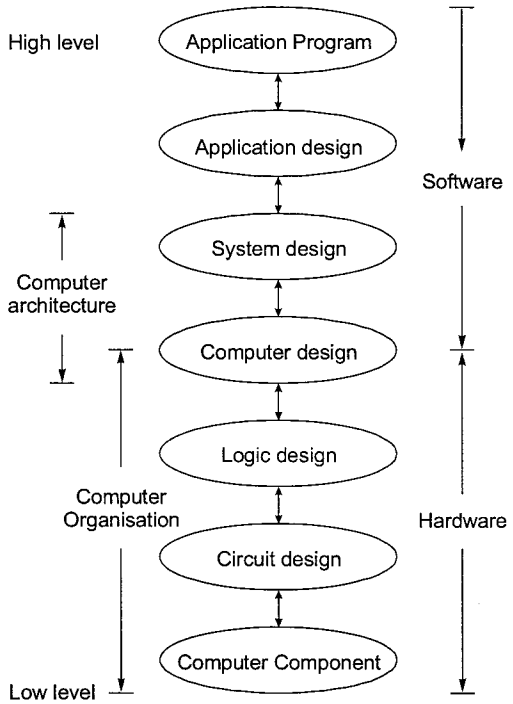
Layers of Abstraction

1.1.2 Computer Organization and Computer Architecture

Computer design: The determination of how to interconnect the components and which components to used based upon some specifications.

1.1.3 Computer Architecture (CA)

- Computer architecture is the conceptual design and fundamental operational structure of a computer system. It is a functional description of requirements and design implementations for the various parts of a computer.
- It is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals.
- It deals with the architectural attributes like physical address memory, CPU and how they should be designed and made to coordinate with each other keeping the goals in mind.



Computer System

1.1.4 Computer Organization (CO)

- Computer architecture comes before computer organization.
- Computer organization is how operational attributes are linked together and contribute to realise the architectural specifications.
- It encompasses all physical aspects of computer systems. e.g. Circuit design, control signals, memory types.

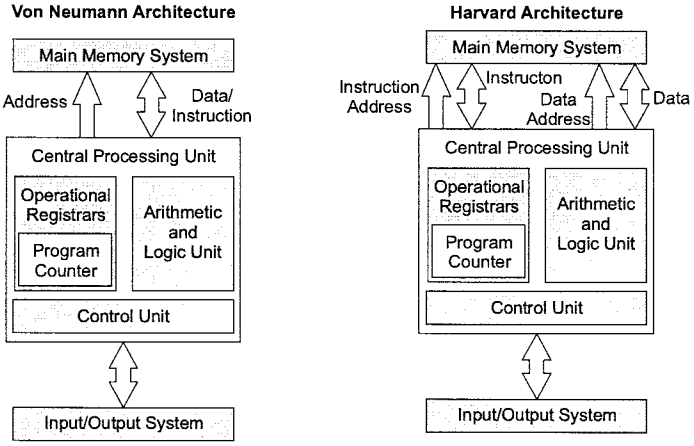
1.1.5 Computer Architecture Vs Computer Organization

Architecture and organization are independent, you can change the organization of a computer without changing its architecture.

1. The architecture indicates its hardware whereas the organization reveals its performance.
2. For designing a computer, its architecture is fixed first and then its organization is decided.

Computer Organization	Computer Architecture
• Computer organization deals with structural relationships that are not visible to the programmer (like clock frequency or the size of the physical memory).	• Computer architecture deals with the functional behavior of a computer system as viewed by a programmer (like the size of a data type – 32 bits to an integer).
• A computer's organization expresses the realization of the architecture.	• A computer's architecture is its abstract model and is the programmer's view in terms of instructions, addressing modes and registers.
• Organization describes how it does it.	• Architecture describes what the computer does.

Von Neumann Architecture Vs Harvard Architecture



1.1.6 Evolution of Digital Computers

First generation: Vacuum tube computers (1945~1953)

- Program and data reside in the same memory (stored program concepts: John von Neumann)
- Vacuum tubes were used to implement the functions (ALU & CU design)
- Magnetic core and magnetic tape storage devices are used.
- Using electronic vacuum tubes, as the switching components.
- Assembly level language is used

Second generation: Transistorized computers (1954~1965)

- Transistor were used to design ALU & CU
- High Level Language is used (FORTRAN)
- To convert HLL to MLL compiler were used
- Separate I/O processor were developed to operate in parallel with CPU, thus improving the performance
- Invention of the transistor which was faster, smaller and required considerably less power to operate

Third generation: Integrated circuit computers (1965~1980)

- IC technology improved
- Improved IC technology helped in designing low cost, high speed processor and memory modules
- Multiprogramming, pipelining concepts were incorporated
- DOS allowed efficient and coordinate operation of computer system with multiple users
- Cache and virtual memory concepts were developed
- More than one circuit on a single silicon chip became available.

Fourth generation: Very large scale integrated (VLSI) computers (1980~2000)

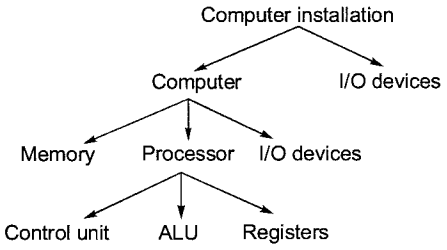
- CPU termed as microprocessor
- INTEL, MOTOROLA, TEXAS,NATIONAL semiconductors started developing microprocessor
- Workstations, microprocessor (PC) & Notebook computers were developed
- Interconnection of different computer for better communication LAN, MAN and WAN
- Computational speed increased by 1000 times
- Specialized processors like Digital Signal Processor were also developed.

Fifth generation: System-on-chip (SOC) computers (2000~)

- E-Commerce, E- banking, home office
- ARM, AMD, INTEL, MOTOROLA
- High speed processor - GHz speed
- Because of submicron IC technology lot of added features in small size.

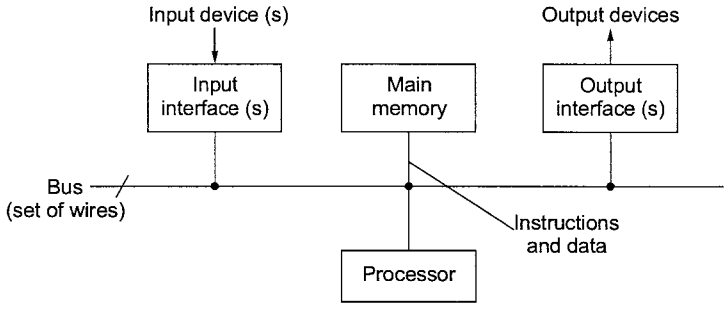
1.1.7 Structure and Function of a Computer System

The designer need only deal with a particular level of the system at a time. At each level, the system consists of a set of *components and their interrelationships.*



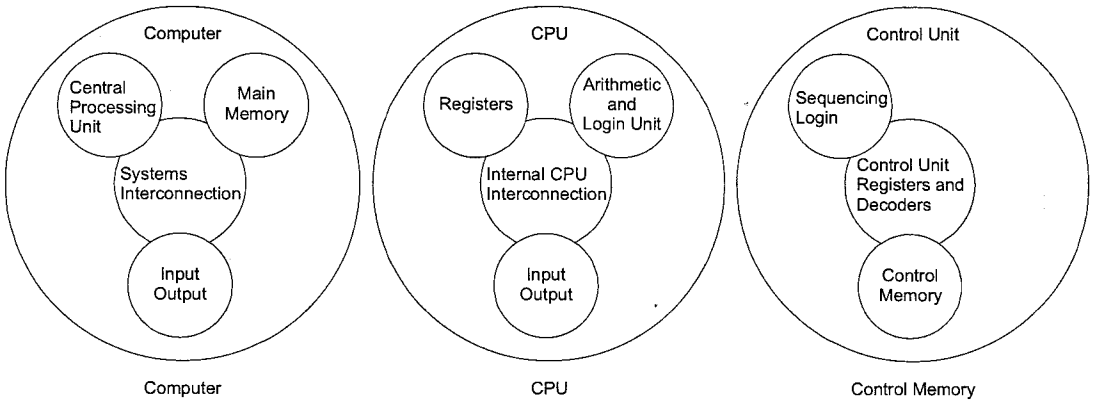
The behavior at each level depends only on a simplified, abstracted characterization of the system at the next lower level. At each level, the designer is concerned with structure and function. Important relationships are explained in the figure.

Structure is the way in which components relate to each other (shown in the following figure). Function is the operation of individual components as part of the structure. Functions are Data processing, Data storage, Data movement and Control.



Internal Structure of Computer

1.1.8 Components of Computer Structure



Computer Structure vs CPU Structure vs Control Unit

1. **Input Unit:** Computers can understand only machine language. Therefore for converting data from human language to machine language we use some special peripheral devices which are called input device.
   Examples: Keyboard, Mouse, Joystick, etc.
2. **Output Unit:** After passing instructions for solving particular problem, the results came out from computer comes in machine language and this is very difficult to convert that results into human language. There are several such peripheral devices which help us to convert the machine language data into human acceptable data. These devices are called output devices.
   Examples: Monitor, Printer, LCD, LED etc.

3. **Memory Unit:** Which is used to store data in computer.
   Memory unit performs the following actions
   (a) Stores data and instructions required for processing.
   (b) Stores the intermediate results obtain during processing.
   (c) Stores final results before sending it to output unit.
   Two class of storage units: (i) Primary Memory (ii) Secondary Memory
   Two types of primary memory are RAM (Random Access Memory) and ROM (Read Only Memory). RAM is used to store data temporarily during the program execution. ROM is used to store data and program which is not going to change.
   Secondary Memory is used for bulk storage or mass storage to store data permanently.
4. **CPU:** It is main unit of the computer system. It is responsible for carrying out computational task. The major structural components of a CPU are:
   (a) *Control Unit (CU):* Controls the operation of the CPU and hence the computer.
   (b) *Arithmetic and Logic Unit (ALU):* Performs computer's data processing functions.
   (c) *Register:* Provides storage internal to the CPU.
   (d) *CPU Interconnection:* communication among the control unit, ALU, and register.

1.1.9 Bus Structure

Bus: It is a group of wires (lines or signals) which carries information form CPU to peripherals or peripherals to CPU. The CPU and Memory are connected by Data Bus, Address Bus and Control Bus.

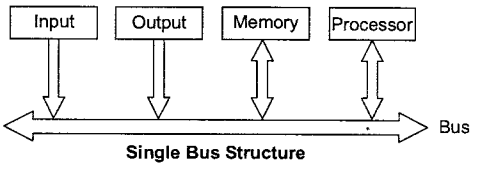
Address Bus: It is unidirectional bus which carries address information bits form processor to peripherals.

Data Bus: It is bidirectional bus which carries data information bit form processor to peripherals and vice-versa.

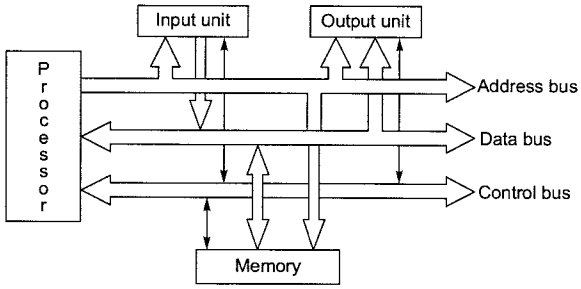
Control Bus: It is bidirectional bus which carries control signals form processor to peripherals and vice-versa.

1.1.10 Types of Bus Structure

- **Single bus structure:** Common bus used (shown in the following figure) to communicate between peripherals and microprocessor.



Single Bus Structure



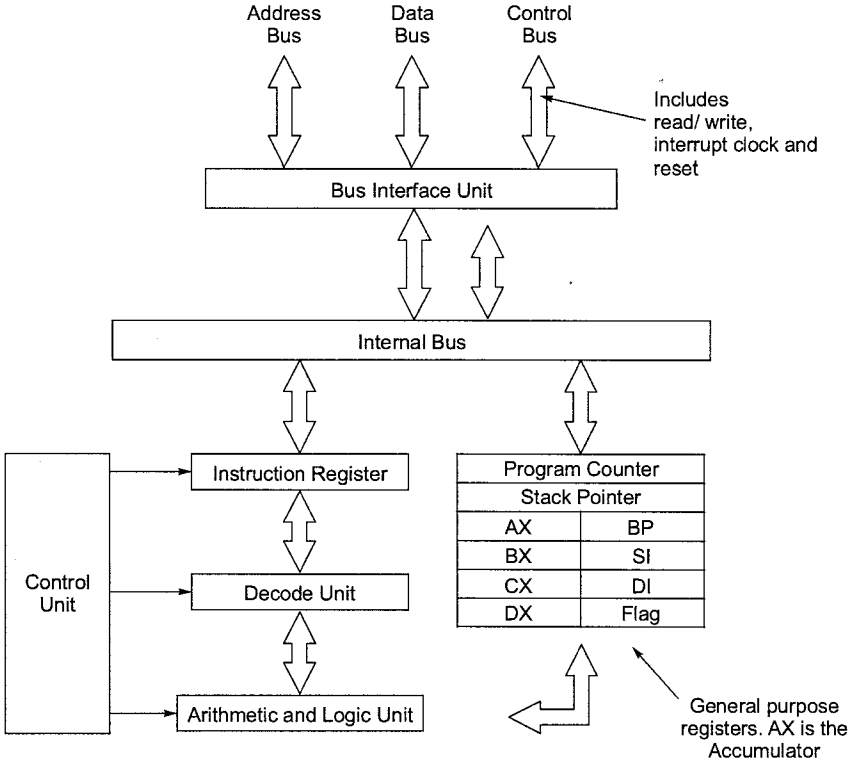
Internal view of Single Bus Structure

- **Two Bus Structure:** One bus can be used to fetch instruction other can be used to fetch data, required for execution. It improves the performance, but cost increases.

1.1.11 CISC and RISC Architectures

CISC (Complex Instruction Set Computers)	RISC (Reduced Instruction Set Computers)
• Large instruction set	• Compact instruction set
• Instruction formats are of different lengths	• Instruction formats are all of the same length
• Instructions perform both elementary and complex operations	• Instructions perform elementary operations
• Control unit is micro-programmed	• Control unit is simple and hardwired
• Not pipelined or less pipelined	• Pipelined
• Single register set	• Multiple register set
• Numerous memory addressing options for operands	• Compiler and IC developed simultaneously
• Emphasis on hardware	• Emphasis on software
• Includes multi-clock complex instructions	• Single-clock, reduced instruction only
• Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	• Register to register: "LOAD" and "STORE" are independent instructions
• Small code sizes, high cycles per second	• Low cycles per second, large code sizes
• Transistors used for storing complex instructions	• Spends more transistors on memory registers
Examples of CISC processors: • VAX • PDP?11 • Motorola 68000 family • Intel x86 architecture based processors.	**Examples** of RISC processors • Apple iPods (custom ARM7TDMI SoC) • Apple iPhone (Samsung ARM1176JZF) • Nintendo Game Boy Advance (ARM7) • Sony Network Walkman (Sony in?house ARM based chip)

1.1.12 General CPU Architecture (8086 Microprocessor)



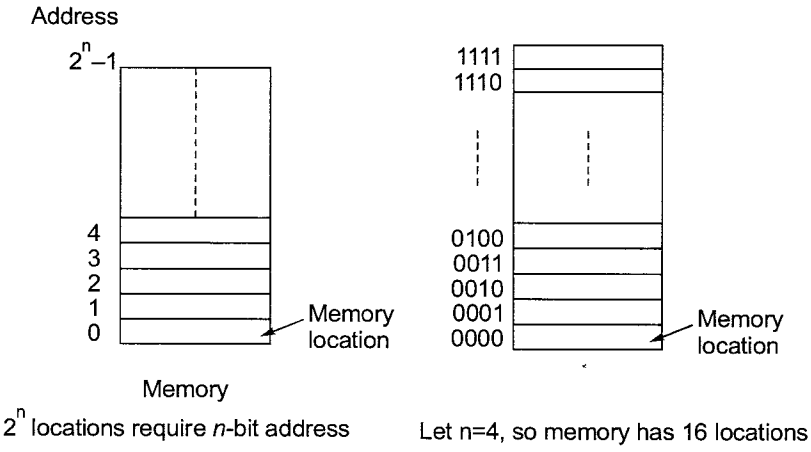
1.1.13 Issues of Computer Design

- Cannot assume infinite speed and memory.
- Speed mismatch between memory and processor
- Handle bugs and errors
- Multiple processors, processes, threads
- Shared memory
- Disk access
- Better performance with reduced power

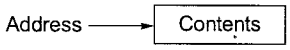
1.2 Data Storage in the Memory

How to address main memory?

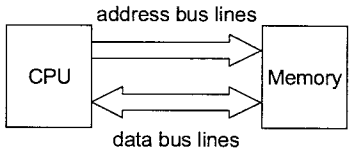
Main memory is a set of storage locations. Each location of memory has a unique address (a binary number starting from zero). Each "addressable" location holds a fixed number of bits. Any location can be accessed at high speed in any order (random access memory).



Memory consists of addressable locations. A memory location has 2 components: Address and Contents.



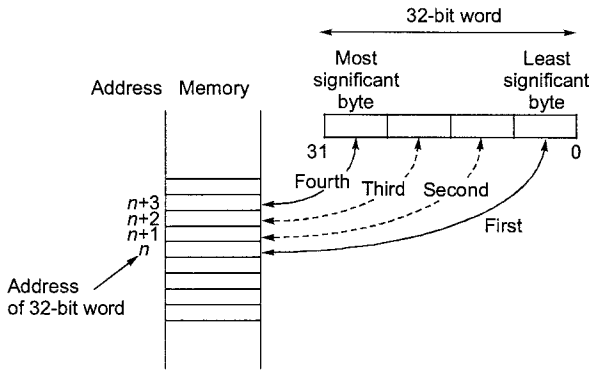
Data transfer between CPU and memory involves address bus and data bus



There are two ways stored information can be organized.

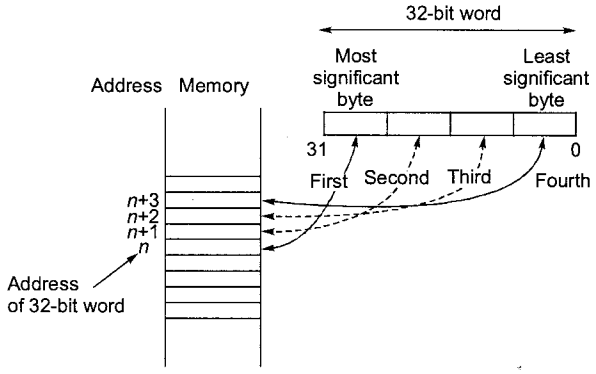
1. **Little Endian (little end first):** First location can hold least significant byte.
   Example: Intel uses little endian

Little endian format (little end first)



Assume 4 byte words are used in memory

2. **Big Endian (big end first):** First location can hold most significant byte.
   Example: Mac/SUN use big endian
   Big endian format (big end first)



Assume word size is 4 bytes in memory

1.3 Machine Instructions

A set of binary codes that are recognised and executed directly by a processor is called a machine code. An individual machine code is called a machine instruction.

Main memory holds machine instructions and Data. Each instruction/data require one or more memory locations depends on size of instruction/data.

The set of all codes recognized by a particular processor is known as its *instruction set*. Important things that make up memory instruction set architectures,

1. Memory Models (how does the memory look to the CPU)
   (a) Addressable cell size. Most commonly is 8 bits (= 1 byte) called byte addressable
   (b) Alignment
   (c) Address Space
   (d) Endianness (Little or Big endian)
2. Registers
   (a) General Purpose Registers(to store temp results...)
   (b) Special Purpose Registers: Program Counter (PC), Stack Pointer (SP), Input/Output Registers, Status Registers
3. Data Types
4. Instructions

Encoding Machine Instructions: Machine instruction is encoded in a binary pattern that specifies:

(*i*) Operation,

(*ii*) Operands/addresses used for the operation, and

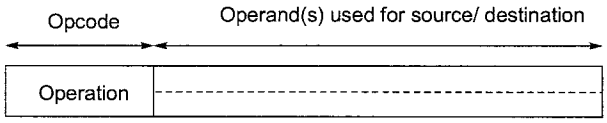
(*iii*) Where the result should be placed if any

Fields of an Instruction:

(*i*) **Opcode:** The operation itself is usually represented by a code called the opcode.

(*ii*) **Address:** Other than opcode, all the other parts of an instruction are called address or operand fields.

(*iii*) **Mode:** It specifies the way the operand or effective address is determined. It may be implicit or explicit in the instruction.



Machine Instruction

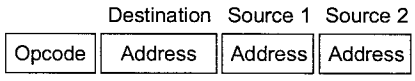
1.3.1 Instruction Formats

Based on number of address fields used in the instruction, machine instruction has the following formats:

1. Three Address Instruction
2. Two Address Instruction
3. One Address Instruction
4. Zero Address Instruction

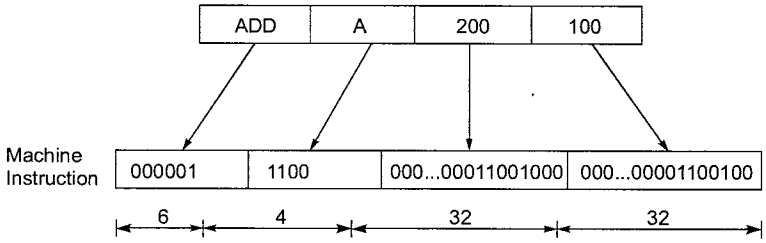
1.3.2 Three-Address Instructions

3-Address Instruction has one opcode and three address fields.



3- Address Instruction Format

Three address instruction can be encoded as following:



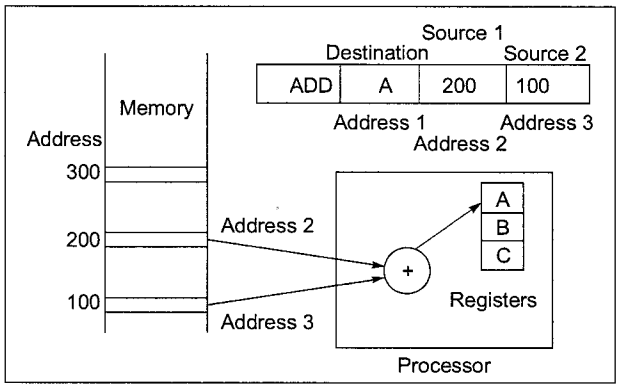
Assume 6 bits used for Opcode, 4 bits to specify a register and 32 bits for Memory address

Assume 6 bits used for Opcode, 4 bits to specify a register and 32 bits for memory address.

Example: ADD A, [200], [100]

Where [ ] means "contents of memory", a common notation.

ADD is the op-code *mnemonic* for Addition



Disadvantages of three address instructions:

- It has long instruction length.
- There may be three memory accesses needed for an instruction.

***Example*-1.1** Write the three address instructions for the following statement.

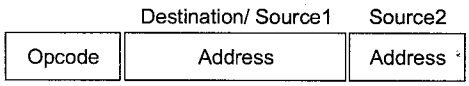
$$X = (A + B) \times (C + D)$$

Solution:

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

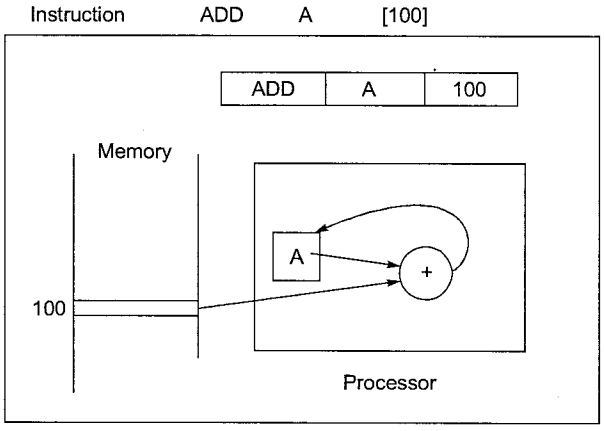
1.3.3 Two Address Instruction

2-Address Instruction has one opcode and two address fields.



Two Address Instruction Format

Example: Instruction ADD A, [100]



Disadvantage of two address instructions: It may needs three memory accesses (using all operands as memory addresses for arithmetic operations, ADD [200] [100]).

Example-1.2 Write the two address instructions for the following statement:

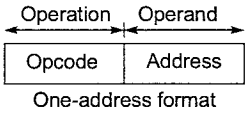
$$X = (A + B) \times (C + D)$$

Solution:

```
MOV R1, A   ;   R1 ← M[A]
ADD R1, B   ;   R1 ← R1 + M[B]
MOV R2, C   ;   R2 ← M[C]
ADD R2, D   ;   R2 ← R2 + D
MUL R1, R2  ;   R1 ← R1 × R2
MOV X, R1   ;   M[X] ← R1
```

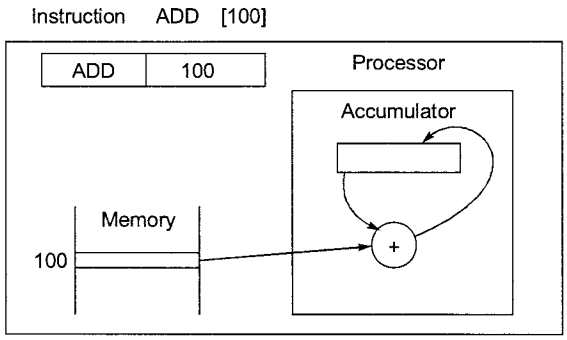
1.3.4 One address Instruction

One address instruction has only two fields. One for opcode and other filed for operand.



One-address format

Example:



Advantages:

- Shorter instruction
- Eliminates two memory accesses
- Faster accessing location inside processor than memory

Disadvantages:

- Only one location for one operand and result
- Still it may need one memory access

Example-1.3 Write the one address instructions for the following statement:

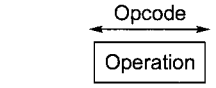
$$X = (A + B) \times (C + D)$$

Solution:

```
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
```

1.3.5 Zero Address Instruction

Zero address instruction has only opcode filed. It has no address field.

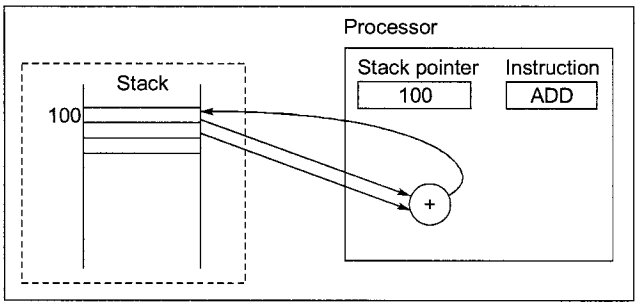


Zero Address Instruction Format

It is possible to eliminate all addresses by using specific locations. Usually the locations for the operands/result are the top two locations of a *stack* (a last-in first-out queue) in memory or implemented with registers within the processor.

Stack pointer is a register within the processor used to hold, the address of the top location. Zero-address format may be useful to compilers for producing code for arithmetic expressions (using reverse Polish notation) with the help of stack.

Example: Instruction: ADD ; it adds top two elements of the stack and stores result at top of stack



Example - 1.4 Write the zero address instructions for the following statement.

$$X = (A + B) \times (C + D)$$

Solution:

```
LOAD  A ;  AC ← M[A]
PUSH  A ;  TOS ← A
PUSH  B ;  TOS ← B
ADD     ;  TOS ← (A +B)
PUSH  C ;  TOS ← C
PUSH  D ;  TOS ← D
ADD     ;  TOS ← (C + D)
MUL     ;  TOS ← (C + D) × (A + B)
POP   X ;  M[X] ← TOS
```

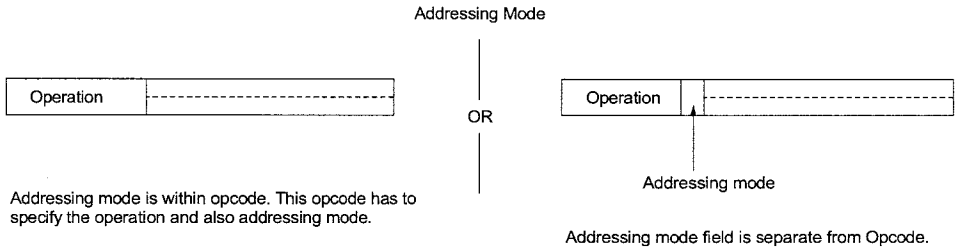
A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

1.3.6 Addressing Modes

The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes. It is the method used to identify the location of an operand.

Addressing: "*The general subject of specifying where the operands are*" is called addressing.

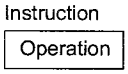
- Addressing mode may present within operation field (opcode) or in a separate ***mode*** field.
- The decoding step in the instruction cycle determines the operation to be performed, the addressing mode of the instruction, and the location of the operands.



Effective address: The actual address of the operand after all address computations of the addressing mode have been performed. Operand may present in register, memory, or within the instruction. Effective address of operand is computed using the addressing mode.

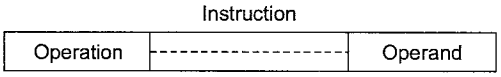
1.3.7 Types of Addressing Modes

1. **Implied Mode:** Operands are specified implicitly in the definition of the instruction.



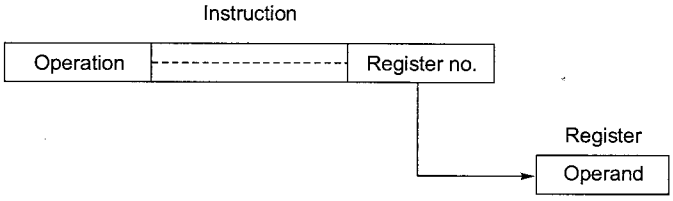
Example: Zero-address instructions (stack operations ADD and SUB).

2. **Immediate Mode:** The actual operand (data) is specified in the instruction.



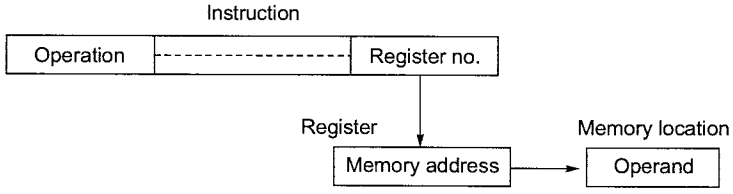
Example (Assembly): MOV R1, 123
The operand 123 is held in the instruction.
Example (High Level): $x = 123$;

3. **Register Mode:** The instruction specifies a register that has operand.



Example (Assembly): MOV R1, R2
Instruction has register R2 and R2 has operand.
Example (High Level): $x = y$;

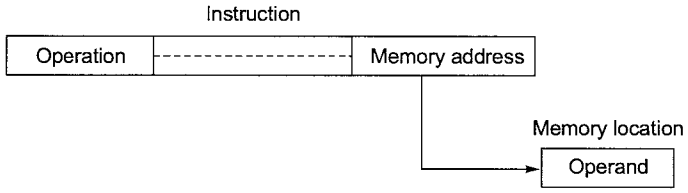
4. **Register Indirect Mode:** The instruction specifies a register that contains the address of the operand.



The operand is held in memory. The address of the operand location is held in a register which is specified in instruction.
Example (Assembly): ADD R1, (R2)
Instruction has register R2 and R2 has memory address of operand.

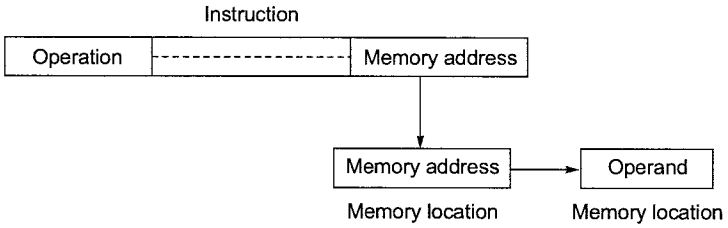
Example (High Level): int*x*;
int y, z = 0;
x = &y;
**x* = 123;
z = z + **x*; // z is similar to R1, *x* is similar to R2.

5. **Direct (Absolute) Address Mode:** The instruction has memory address where the operand is located.



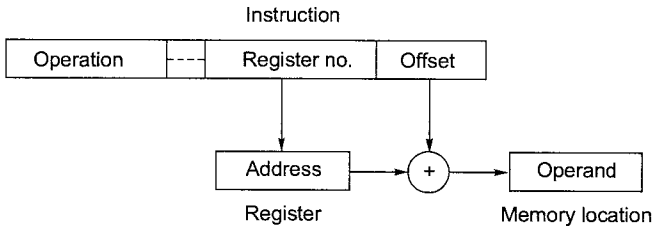
Example: MOV R1, [1000]
The operand is in memory and the memory address 1000 of the operand is held in instruction.

6. **Indirect Address Mode:** The instruction has the address that specifies the effective address of the operand.



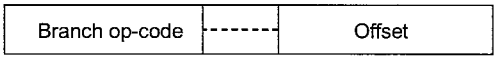
Example (Assembly): ADD R1, (1000)
Instruction has memory address 1000. At location 1000, the memory address of operand is available.
Example (High Level): int **x*;
int y, z = 0;
x = &y;
**x* = 123;
z = z + **x*; // z is similar to R1, *x* is similar to 1000.

7. **Indexed Addressing Mode:** The effective address is the sum of an index register and the offset field(constant). It is similar to register indirect addressing except an offset held in the instruction is added to register contents to compute the effective address. The indexed addressing mode is useful in dealing with lists and arrays.



Example (Assembly): MOV R1, 2(R2); R1 stores operand from memory address: 2 + (R2). R2 has memory address and offset is 2. By adding constant 2 and content of R2 gives the effective address of operand.
Example (High-level): int *x*[10], y; y = *x*[2]; // same as y = *(*x* + 2); where y is similar to R1 and *x* is similar to R2.

8. **Base with Index Address Mode:** The effective address is the sum of contents of base register and index register.
   Example (Assembly): MOV R1, (R2, R3). Register R2 contain the base address and R3 contain the index. The effective address is the sum of the contents of registers R2 and R3.
   Example (High-level): int x[10], i = 2 , z; y = $x[i]$; // where y, x and i are similar to R1, R2 and R3 respectively.
9. **Base with Index Address Mode:** It uses two registers R_i and R_j plus a constant X. The effective address is the sum of the constant X and the contents of registers R_i and R_j.
   Example (Assembly): MOV R1, 2(R2, R3)
   Register R2 contain the base address and R3 contain the index. The effective address is the sum of the constant value 2, the contents of registers R2 and R3.
   Example (High-level): int x[10], i = 2, z; y = $x[i + 3]$; // where y, x and i are similar to R1, R2 and R3 respectively.
10. **Relative Address Mode:** The effective address is the sum of the address field(offset) and the content of the Program Counter (PC). Specify target location as number of instructions from branch/ jump instruction . Also makes code *relocatable.* (i.e. code can be loaded anywhere in memory without altering branch/jump instructions.) The number of locations to the target is held in the instruction as an offset.



   Example (Assembly): Assume PC holding 100 while executing the following instruction. JUMP 123; control will transfer to 110 by adding 100 with offset filed 23. In high-level (C) programming, using goto relative addressing can be done.
11. **Auto Increment and Auto Decrement Modes:**
    (*i*) *Post Auto increment mode:* The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. It is denoted as $(R_i)^+$.
    Example: MOV R1, $(R2)^+$
    (*ii*) *Pre Auto Increment mode:* Before accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. Then effective address of the operand is the incremented contents of a register. It is denoted as $+(R_i)$.
    Example: MOV R1, +(R1)
    (*iii*) *Post Auto Decrement mode:* The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically decremented to point to the next item in a list. It is denoted as $(R_i)^-$.
    Example: MOV R1, $(R2)^-$
    (*iv*) *Pre Auto Decrement mode:* Before accessing the operand, the contents of this register are automatically decremented to point to the next item in a list. Then effective address of the operand is the incremented contents of a register. It is denoted as $-(R_i)$.
    Example: MOV R1, –(R1)

Table for Effective Address calculation of operands using Addressing Modes

Name	Example	Operation	Syntax (Assembly)	Effective Address Computation	Usage
Immediate	ADD R_4, 3	R_4: = R_4 + 3	# Value	Operand = Value	Constants
Register	ADD R_4, R_3	R_4: = R_4 + R_3	R_i	EA = R_i	Temporary variables
Register Indirect	ADD R_4, [R_1]	R_4: = R_4 + M[R_1]	(R_i)	EA = (R_i)	Pointers
Direct (Absolute)	ADD R_4, [100]	R_4: = R_4 + M[100]	LOC	EA = LOC	Global variables
Memory Indirect	ADD R_4, @ [100]	R_4: = R_4 + M[M [100]]	(LOC)	EA = (LOC)	Pointers
Displacement (Based)	ADD R_4, [R_1+100]	R_4: = R_4 + M[R_1 + 100]	X(R_i)	EA = (R_i)+X	Local Variables
Based with Index	ADD R_4, [R_1+R_2]	R_4: = R_4 + M [R_1 + R_2]	(R_i, R_j)	EA = (R_i)+(R_j)	Arrays
Based with index and offset	ADD R_4, [R_1+R_2+8]	R_4: = R_4 + M [R_1 + R_2 + 8]	X(R_i, R_j)	EA = (R_i)+(R_j)+X	Arrays
Relative	ADD R_1, [R_2+PC]	R_4: = R_4 + M [R_2 + PC]	X(PC)	EA = (PC)+X	Static local variables
Auto-increment (post)	ADD R_4, [R_2]+	R_4: = R_4+M[R_2] R_2: = R_2 + d	(R_i)+	EA = (R_i); Increament R_i	Stack operations
Auto-increment (pre)	ADD R_4, +[R_2]	R_2: = R_2 + d R_4: = R_4+M[R_2]	+(R_i)	Increament R_i EA = (R_i);	Stack operations
Auto-decrement (post)	ADD R_4, [R_2]–	R_4: = R_4 + M[R_2] R_2: = R_2 – d	(R_i) –	EA = (R_i); Decrement R_i	Stack operations
Auto-decrement (pre)	ADD R_4, –[R_2]	R_2: = R_2 – d R_4: = R_4 + M[R_2]	–(R_i)	Decrement R_i EA = (R_i);	Stack operations

Example - 1.5 An address field in an instruction contains decimal value 14. Where is the corresponding operand located for:

(a) immediate addressing?
(b) direct addressing?
(c) indirect addressing?
(d) register addressing?
(e) register indirect addressing?

Solution:

Instruction

Opcode	Address 14

(a) 14 (The address field)
(b) Memory location 14
(c) The memory location whose address is in memory location 14
(d) Register 14
(e) The memory location whose address is in register 14

Example - 1.6 Given the following memory values and a one address machine with an accumulator:

Word 20 contains 40
Word 30 contains 50
Word 40 contains 60
Word 50 contains 70

What values do the following instructions load into the accumulator?

(a) Load IMMEDIATE 20
(b) Load DIRECT 20
(c) Load INDIRECT 20
(d) Load IMMEDIATE 30
(e) Load DIRECT 30
(f) Load INDIRECT 30

Solution:

(a) 20
(b) 40
(c) 60
(d) 30
(e) 50
(f) 70

***Example*-1.7** Let the address stored in the program counter be designated by the symbol X1. The instruction stored in X1 has an address part (operand reference) X2. The operand needed to execute the instruction is stored in the memory word with address X3. An index register contains the value X4. What is the relationship between these various quantities if the addressing mode of the instruction is

(a) Direct
(b) Indirect
(c) PC relative
(d) Indexed

Solution:

(a) X3 = X2
(b) X3 = (X2)
(c) X3 = X1 + X2 + 1
(d) X3 = X2 + X4

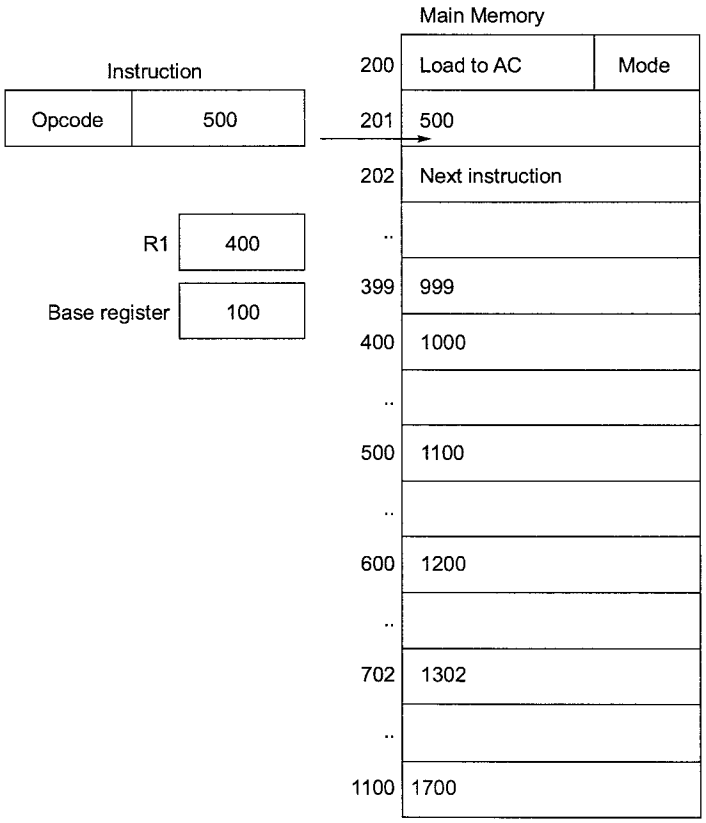
***Example*-1.8** Consider a 16 bit processor in which the following appears in main memory, starting at location 200:

The first part of the first word indicates that this instruction loads a value into an accumulator. The mode field specifies an addressing mode and, if appropriate, indicates a source, register, assuming that when used, the source register is R1, which has a value of 400. There is also a base register that contains the value of 100. The value of 500 in location 201 may be part of the address calculation. Assume that location 399 contains the value 999, location 400 contains the value 1000 and so on. Determine the effective address and the operand to be loaded for the following addressing modes:

200	Load to AC	Mode
201	500	
202	Next instruction	

(a) Direct
(b) Immediate
(c) Indirect
(d) PC relative
(e) Displacement
(f) Register
(g) Register indirect
(h) Autoindexing with increment, using R1

Solution:



	EA	Operand
a	500	1100
b	201	500
c	1100	1700
d	201 + 1 + 500 = 702	1302
e	500 + 100 = 600	1200
f	R1	400
g	400	1000
h	400	1000

R1 401

R1 is incremented after the execution of the instruction.

1.3.8 Types of Machine Instructions

Based on operation performed, machine instructions can be divided into the following:

1. Data transfer Instructions
2. Data Manipulation Instructions (Computation): Arithmetic and Logical Instructions.
3. Program Control Instructions.

Data Transfer Instructions

Instructions that transfers data from one location(Register/Memory) to another location(Register/Memory) without changing the data. Data transfer operations supported by many processors are Load, Store, Move, Input, Output, Push, Pop, Exchange, etc.

- **LOAD:** Data transfers from memory to register.
- **STORE:** Data transfers from register to memory.
- **MOVE:** Data transfers from register to register.
- **IN:** Transfers data from input device to register.
- **OUT:** Transfers data from register to output device.
- **PUSH:** Gets data from memory or register on to the top of stack.
- **POP:** Gets data from top of stack to memory or register.
- **XCHG:** Exchanges the data between memory and registers.

Data Manipulation Instructions

- **Arithmetic Instructions:** Performs an arithmetic operations such as addition, subtraction, multiplication division, Increment, Decrement, etc.
  Example: ADD, SUB, MUL, DIV, INC, DEC, etc.
- **Logical Instructions:** Performs bit-wise logical operation such as AND, OR, exclusive-OR, NOT, shift, rotate, etc.
  Example: AND, OR, NOT, XOR, SHL, SHR, ROR, ROL, etc.
- **Arithmetic and Logical Instructions:** Performs operations such as arithmetic shift left, arithmetic shift right, etc.
  Example: SAL, SAR, etc.

Program Control Instructions

1. **Compare and Test Instructions:** To compare and test.status flags we need support of condition code register.
   Condition code register(CCR): Condition Code Register has following flag bits:
   (*i*) **5 status bits:** To indicate a situation after the arithmetic or logical operation.

   Carry borrow (C)
   Overflow (V)
   Zero (Z)
   Negative (N)
   Half Carry (H)
   ⇒ Used CPU's branch instructions to make decisions

   (*ii*) 2 interrupt masking bits

(*iii*) 1 stop disable bit

Bit Name	Function
S	Stop disable flag
X	XTRQ interrupt mask
H	Half carry from lower 4 bit
I	Interrupt mask
N	Negative
Z	Zero
V	Overflow
C	Carry

Status Bits:

(*i*) **Carry Flag (C):** It indicates a carry from an addition or a borrow from a subtraction.
For addition, C = 1 if carry out exists, otherwise C = 0.
For subtraction, C = 1 if borrow exists, otherwise C = 0.

(*ii*) **Overflow Flag (V):** CPU sets V flag to a logic '1' when the result of an operation is beyond of the range that can be represented.
It also called as 2's complement overflow. CPU clears flag when the number is within range.

(*iii*) **Negative Flag (N):** It is also called as sign flag. Indicates whether previous arithmetic result is negative or positive. N = 1 for negative result, and N = 0 for positive result. N^{th} bit indicates state of MSB of a result. If MSB = 1 then N = 1, otherwise N = 0.

(*iv*) **Half Carry (H):** Used only for BCD (Binary Coded Decimal)operations.

(*iv*) **Zero Flag (Z):** Z bit is set to a logic '1' when the result is '0', otherwise Z=0. Note that zero is a positive number.

NOTE: N and Z flags caused by an arithmetic or a logic operation, V and C flags caused by an arithmetic operation.

Compare Instruction: Compare instruction is specifically provided, which is similar to a subtract instruction except the result is not stored anywhere, but flags are set according to the result.

Relation	x – y
<	Negative
>	Positive and not zero
>=	Positive or zero
<=	Negative or zero
==	Zero
!=	not zero

Example: CMP R1, R2;
Above instruction first does (R1 – R2). It sets CCR flags by checking the result of (R1 – R2).

2. **Unconditional branch Instruction:** It causes an unconditional change of execution sequence to a new location.

Example:

```
      JUMP   L2
      MOV    R3, R1     goto L2
L2:   ...
```

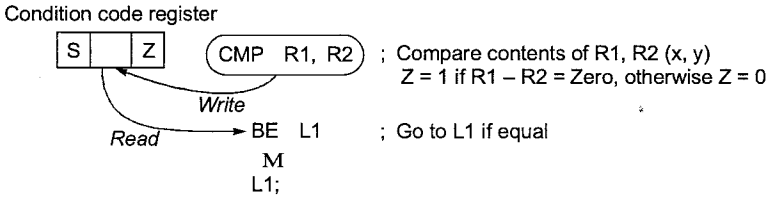
3. **Conditional Branch Instruction:** A *conditional branch instruction* used to examine the values stored in the condition code register to determine whether the specific condition exists and to branch if it does.

Opcode	Condition	High level language notation
BL	Branch if less than	<
BG	Branch if greater than	>
BGE	Branch if greater or equal to	>=
BLE	Branch if less or equal to	<=
BE	Branch if equal	==
BNE	Branch if not equal	!=

Example:

High Level Code	Assembly Code
if (x == y) goto L1; ⋮ L1:	BE R1, R2, L1 ⋮ L1: Compiler allocates R1 for x and R2 for y.

Example: Consider the following high level code.
If (x = = y) goto L1;
The above code can be implemented with the following assembly code using conditional branch and compare instructions. Suppose x held in R1 and y held in R2. The if statement: Could be implemented by sequence of two instructions, CMP and BE



Example:

High Level Code	Equivalent Assembly Code

for (i = 1; i < 10; i++)
a = a*1;
Λ
	The variable i is held in register R1, and the variable x is held in register R2.
	MOV R1, 1 ; i = 0
	L2: CMP R1, 10 ; Compare R1 with 10
	BGE L1 ; Go to L1 if end of loop
	MUL R2, R2, R1 ; a = a*i
	ADD R1, R1, 1 ; increment i
	JUMP L2
	L1: Λ

4. **Subroutines:** A subroutine is a program fragment that lives in user space, performs a well-defined task. It is invoked (called) by another user program and returns control to the calling program when finished.
Subroutine is used for
- Frequently executed code segments
- Library routines

CALL and RET instructions are used to execute subroutines. CALL is used from the main code where subroutine invoked and RET is used to return from the subroutine to the main code.

Example: Compute the value R1 – R3 using the routine that negates the value of R3.

Subroutine	**Negate the Value in R0**				
	2sComp	NOT	R0,	R0	; flip bits
		ADD	R0	R0, #1	; add one
		RET			; return to caller
Main Program	ADD	R0, R3, #0		R0	; copy R3 to R0
	JSR	2sComp		R0, #1	; negate
	ADD	R4, R1, R0			; add to R1
	Λ				

5. Halting and Interrupt Instructions:
   - **NOP Instruction:** NOP is no operation, causes no change in the processor state other than an advancement of the program counter. It can be used to synchronize timing.
   - **HALT:** It brings the processor to an orderly halt, remaining in an idle state until restarted by interrupt, trace, reset, or external action.
   - **Interrupt Instructions:** Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced. Generally, a particular task is assigned to that interrupt signal. In the microprocessor based system the interrupts are used for data transfer between the peripheral devices and the microprocessor.

     Example: TRAP, INTR, etc.

     RESET: It reset the processor. This may include any or all of setting registers to an initial value, setting the program counter to a standard starting location (restarting the computer), clearing or setting interrupts, and sending a reset signal to external devices.

     TRAP: It is non-maskable edge and level triggered interrupt. TRAP has the highest priority and vectored interrupt. In case of sudden power failure, it executes a ISR and send the data from main memory to backup memory. TRAP can be delayed using HOLD signal. This interrupt transfers the microprocessor's control to location 0024H. TRAP interrupts can only be masked by resetting the microprocessor. There is no other way to mask it.

     INTR: It is level triggered and maskable interrupt. On receiving the instruction, CPU save the address of next instruction on stack and execute received instruction. It has the lowest priority. It can be disabled by resetting the microprocessor or by DI and SIM instruction.

1.3.9 Instruction Stream Vs Data Stream

There are four types of Devices bases on instruction and data streams.

1. Single Instruction Single Data
2. Multiple Instruction Single Data
3. Single Instruction Multiple Data
4. Multiple Instruction Multiple Data

Single/multiple instruction streams: How many types of instructions may be performed at once?

Single/multiple data streams: How many data streams may be operated on at once?

Instruction Stream \ Date Stream	Single	Multiple
Single	SISD Uniprocessors	SISD Pipelined Vector Processors
Multiple	SISD Systolic arrays	SISD Multiprocessors Multi-core Multicomputers

Most modern parallel computers (and personal computers) are MIMD.

MISD never used to large extent.

Summary

- **Computer:** A device that accepts input, processes data, stores data, and produces output, all according to a series of stored instructions.
- **Hardware:** Includes the electronic and mechanical devices that process the data; refers to the computer as well as peripheral devices.
- **Software:** A computer program that tells the computer how to perform particular tasks.
- **Computer organization:** Interconnection of hardware to form the computer system
- **Computer architecture:** the structure and behaviour of the computer perceived by the user.
- **Input:** Whatever is put into a computer system.
- **Data:** Refers to the symbols that represent facts, objects, or ideas.
- **Information:** The results of the computer storing data as bits and bytes; the words, numbers, sounds, and graphics.
- **Output:** Consists of the processing results produced by a computer.
- **Main Memory:** Area of the computer that temporarily holds data waiting to be processed, stored, or output. *Example:* Cache and Main memory
- **Secondary Storage:** Area of the computer that holds data on a permanent basis when it is not immediately needed for processing. Example: Disk, Floppy, etc.
- ***Instruction set:*** *It is a list of all the instructions, that a processor can execute.*
- **Addressing Modes:** Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced)
- **Implied Mode:** Address of the operands are specified implicitly in the definition of the instruction. No need to specify address in the instruction.
- **Immediate Mode:** Instead of specifying the address of the operand, operand itself is specified.
- **Register Mode:** Address specified in the instruction is the register address. Designated operand need to be in a register.
- **Register Indirect Mode:** Instruction specifies a register which contains the memory address of the operand.
- **Direct Address Mode:** Instruction specifies the memory address which can be used directly to access the memory.
- **Indirect Addressing Mode:** The address field of an instruction specifies the address of a memory location that contains the address of the operand

Student's Assignments

Q.1 What does CISC and RISC means?
(a) common instruction set controller and rare instruction set controller
(b) complex instruction set controller and reduced instruction set controller
(c) compiled instruction set source code and recompiled instruction source code
(d) none of the above

Q.2 A 32-bit address bus allows access to a memory of capacity
(a) 64 MB (b) 16 MB
(c) 1 GB (d) 4 GB

Q.3 The system bus is made up of
(a) data bus
(b) data bus and address bus
(c) data bus and control bus
(d) data bus, control bus and address bus

Q.4 Which of the following is not involved in a memory write operation?
(a) MAR (b) PC
(c) MDR (d) data bus

Q.5 The read/write line
(a) belongs to the data bus
(b) belongs to the control bus
(c) belongs to the address bus
(d) CPU bus

Q.6 _____ is a piece of hardware that executes a set of machine-language instructions.
(a) controller (b) bus
(c) processor (d) motherboard

Q.7 Given below are some statements associated with the registers of a CPU. Identify the false statement.
(a) The program counter holds the memory address of the instruction in execution.
(b) Only opcode is transferred to the control unit.
(c) An instruction in the instruction register consists of the opcode and the operand.
(d) The value of the program counter is incremented by 1 once its value has been read to the memory address register.

Q.8 In Flynn's classification of computers, the vector and array classes of machines belong to
(a) Single instruction/single data category
(b) Single instruction/multiple data category
(c) Multiple instruction/single data category
(d) Multiple instruction/multiple data category

Q.9 The following are four statements regarding what a CPU with only a set of 32 bit registers can perform.
1. Hold and operate on 32 bit integers
2. Hold and operate on 16 bit integers
3. Hold and operate on 64 bit floating point arithmetic
4. Hold and operate on 16 bit UNICODE characters

Which of the following is true about such a CPU?
(a) all are true (b) 1,2 and 3 only
(c) 1,2 and 4 only (d) 1,3 and 4 only

Q.10 The following are four statements about Reduced Instruction Set Computer (RISC) architectures.
1. The typical RISC machine instruction set is small, and is usually a subset of a CISC instruction set.
2. No arithmetic or logical instruction can refer to the memory directly.
3. A comparatively large number of user registers are available.
4. Instructions can be easily decoded through hard-wired control units.

Which of the above statements is true?
(a) 1 and 3 only
(b) 1,3 and 4 only
(c) 1, 2 and 3 only
(d) All of these

Q.11 The word length of a CPU is defined as
(a) the maximum addressable memory size
(b) the width of a CPU register (integer or float point)
(c) the width of the address bus
(d) the number of general purpose CPU registers

Q.12 Which of the following statements is false about CISC architectures?

(a) CISC machine instructions may include complex addressing modes, which require many clock cycles to carry out.

(b) CISC control units are typically micro-programmed, allowing the instruction set to be more flexible.

(c) In the CISC instruction set, all arithmetic/logic instructions must be register based.

(d) CISC architectures may perform better in network centric applications than RISC.

Q.13 Which one is required while establishing the communication link between CPU and peripherals?

(a) synchronization mechanism

(b) conversion of signal values

(c) operating modes

(d) all of the above

Q.14 Consider the following register-transfer language

$$R_3 \longleftarrow R_3 + M[R_1 + R_2]$$

where R_1, R_2 are the CPU index and base registers and M is a memory location in primary memory. Which addressing mode is suitable for above register-transfer language?

(a) immediate (b) indexed

(c) direct (d) displacement

Q.15 Consider a high-level language statement $i--$ then which addressing mode is suitable for it?

(a) autoincrement (b) indexed

(c) displacement (d) autodecrement

Q.16 In a 16-bit instruction code format 3 bit operation code, 12 bit address and 1 bit is assigned for address mode designation. How much data memory space is available

(a) 4 MB (b) 4 KB

(c) 2 KB (d) 2 GB

Q.17 Match **List-I** with **List-II** and select the correct answer using the codes given below the lists:

List-I

A. $Regs[R_4] \leftarrow Regs[R_4] + Regs[R_3]$

B. $Regs[R_4] \leftarrow Regs[R_4] + 3$

C. $Regs[R_4] \leftarrow Regs[R_4] + Mem[Regs[R_1]]$

List-II

1. immediate
2. register
3. register indirect

Codes:

	A	B	C
(a)	3	2	1
(b)	2	1	3
(c)	1	2	3
(d)	None of these		

Q.18 Addressing mode is _____.

(a) explicitly specified

(b) implied by the instruction

(c) both (a) and (b)

(d) Neither (a) nor (b)

Q.19 Relative Addressing Mode is used to write Position-Independent code because

(a) The Code in this mode is easy to atomize

(b) The Code in this mode is easy to make resident

(c) The Code in this mode is easy to relocate in the memory

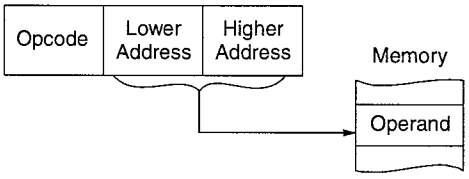
(d) Code executes faster in this mode

Q.20 Consider an (n + k) bit instruction with a k-bit opcode and single n-bit address. Then this instruction allow _____ operations and _____ addressable memory cells.

(a) 2^k, 2^{n+k} (b) 2^{n+k}, 2^{n+k}

(c) 2^k, 2^n (d) 2^{n+k}, 2^{n+1}

Q.21 The following diagram shows, which addressing mode?



(a) immediate addressing mode

(b) indirect addressing mode

(c) extended addressing mode

(d) none of the above

Q.22 The register which holds the address of the location to or from which data are to be transferred is known as
(a) index register
(b) instruction register
(c) memory address register
(d) memory data register

Q.23 Which of the following data transfer mode takes relatively more time?
(a) DMA (b) interrupt initiated I/O
(c) programmed I/O (d) isolated I/O

Q.24 Halt operation comes under _____ .
(a) data transfer (b) control transfer
(c) conversion (d) I/O transfer

Q.25 In four-address instruction format, the number of bytes required to encode an instruction is (assume each address requires 24 bits, and 1 byte is required for operation code)
(a) 9 (b) 13
(c) 14 (d) 12

Q.26 A CPU has an arithmetic unit that adds bytes and then sets its V, C and Z flag bits as follows. The V-bit is set if arithmetic overflow occurs (in 2's complement arithmetic). The C-bit is set if a carry-out is generated from the most significant bit during an operation. The Z-bit is set if the result is zero. What are the values of the V, C and Z flag bit after 8-bit byte 1100 1100 and 1000 1111 are added?

	V	C	Z
(a)	0	0	0
(b)	1	1	0
(c)	1	1	1
(d)	0	1	0

Q.27 Consider the following sequence of instructions intended for execution on a stack machine. Each arithmetic operation pops the second operand, then pops the first operand, operates on them, and then pushes the result back onto the stack

```
Push    b
Push    x
add
Pop     c
Push    c
Push    y
add
Push    c
sub
Pop     z
```

Which of the following statements is/are true?
1. If push and pop instructions each require 5 bytes of storage, and arithmetic operations each require 1 byte of storage then the instruction sequence as a whole requires a total of 40 bytes of storage.
2. At the end of execution, z contains the same value as y.
3. At the end of execution, the stack is empty.

(a) 1 only (b) 2 only
(c) 2 and 3 only (d) 1 and 3 only

Q.28 Match **List-I** with **List-II** and select the correct answer using the codes given below the lists:

List-I	List-II
A. MOV X, R1	1. Three-address instruction
B. STORE X	2. Zero-address instruction
C. POP X.	3. One-address instruction
	4. Two-address instruction

Codes:

	A	B	C
(a)	4	3	2
(b)	3	2	1
(c)	2	3	4
(d)	None of these		

Q.29 The register which keeps track of the execution of a program and which contains the memory address of the instruction currently being executed is known as _____
(a) Index-Register
(b) Memory address register
(c) Program counter
(d) Instruction registers

Q.30 Match **List-I** with **List-II** and select the correct answer using the codes given below the lists:

List-I	List-II
A. 0-address instruction	1. T = TOP (T – 1)
B. 1-address instruction	2. Y = Y + X
C. 2-address instruction	3. Y = A – B
D. 3-address instruction	4. ACC = ACC – X

Codes:

	A	B	C	D
(a)	1	2	3	4
(b)	3	2	4	1
(c)	2	3	1	4
(d)	1	4	2	3

Q.31 An IOSA defines 2 different instruction formats. In the first format, 6 bits are used for the opcode, among these opcode bit patterns, three are used as special patterns. When any of these three patterns appear in the opcode field, the instruction is encoded using the second format, and the actual opcode is present in another 6-bit field. How many unique opcodes can this ISA support?

(a) 64 (b) 253
(c) 254 (d) 189

Q.32 A certain processor executes the following set of machine instructions sequentially.

MOV R_0, # 0
MOV R_1, 100 (R_0)
ADD R_1, 200 (R_0)
MOV 100 (R_0), R_1

Assuming that memory location 100 contains the value 35 (Hex), and the memory location 200 contains the value A4 (Hex), what could be said about the final result?

(a) Memory location 100 contains value A4
(b) Memory location 100 contains value DA
(c) Memory location 100 contains value D9
(d) Memory location 200 contains value 35

Q.33 A stack-based processor executes the following set of machine instructions sequentially.

PUSH 100
PUSH 200
ADD
POP 300

Assuming that:

1. Memory location 100 contains the value 53 (Hex) and memory location 200 contains the value 4C (Hex),
2. The stack is byte organized and the stack pointer is at 00FF, and that
3. All PUSH and POP instructions have a memory operand.

Which of the following could the final result be?

(a) Memory location 300 contains the value 9F
(b) Memory location 00FD contains the value 9F
(c) Memory location 00FF contains a value 100
(d) Memory location 00FE contains a value 200

Answer Key:

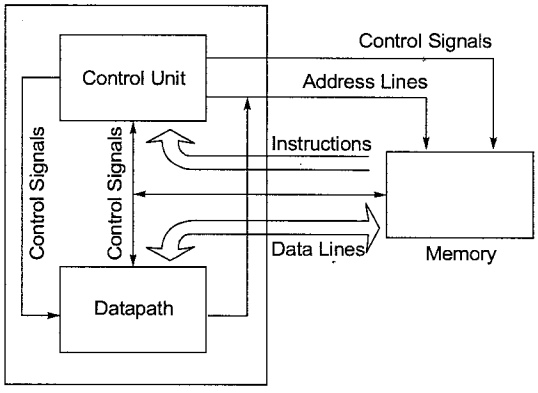
1. (d)	**2.** (d)	**3.** (d)	**4.** (b)	**5.** (b)
6. (c)	**7.** (a)	**8.** (b)	**9.** (c)	**10.** (d)
11. (b)	**12.** (c)	**13.** (d)	**14.** (b)	**15.** (d)
16. (b)	**17.** (b)	**18.** (c)	**19.** (b)	**20.** (c)
21. (c)	**22.** (c)	**23.** (c)	**24.** (b)	**25.** (b)
26. (b)	**27.** (c)	**28.** (a)	**29.** (b)	**30.** (d)
31. (b)	**32.** (c)	**33.** (a)		

CHAPTER 02

CPU Design

2.1 Introduction

Basic processing unit has two units Datapath and Control unit. Datapath includes the register section and the arithmetic/logic unit (ALU). An Arithmetic and Logic Unit (ALU) is a combinational circuit that performs LOGIC and ARITHMETIC Operations. The ALU does the actual computation or processing of data. The basic operations are implemented in hardware level.



Basic Processing Unit (Data Path/Control unit)

Datapath

It is combination of functional units and registers . The layout of buswide logic that operates on data signals is called a datapath.

- Datapath is the component of the processor that performs arithmetic operations.
- Functional units and Registers are called as elements of datapath.
- Functional units are ALU, multipliers, dividers, etc.
- Registers are program counter, shifters, storage registers, etc.

Control Unit

The Control Unit controls the movement of data and instruction into and out of the CPU and controls the operation of the ALU.

Types of CPU Organizations

1. Single Accumulator CPU: *Example:* ADD X ; AC ← AC + M[X]
2. General Register CPU: *Example:* ADD R1, R2 ; R1 ← R1 + R2
3. Stack CPU: *Example:* PUSH X
   ADD: It is zero address instruction, which pops two numbers from the top of stack and adds those two numbers then pushes the result onto stack.
4. Combination of CPU organizations: Combination of any of three CPU organizations.

2.1.1 ALU Design

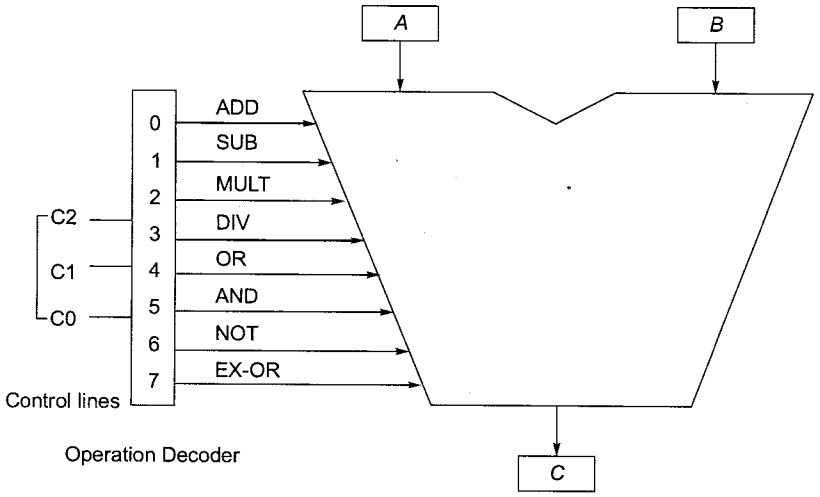
Consider an ALU is having four arithmetic operations (Addition, subtraction, multiplication and division) and four logical operations (OR, AND, NOT & EX-OR). We need three control lines to identify any one of these eight operations. Truth table for 8 operations:

C_1	C_0	Arithmetic $C_2 = 0$	Logical $C_2 = 1$
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	EX-OR

Control line $C_2 = 0$ identifies as arithmetic operation, and $C_2 = 1$ identifies the logical operation. Control lines C_0 and C_1 are used to identify any one of the four operations in a group.

2.1.2 3 to 8 Decoder Circuit for 8 Operations

A 3 to 8 decoder can be used to decode the instruction to implement those 8 operations.



The input data are stored in registers A and B, and according to the operation specified in the control lines, the ALU perform the operation and put the result in register C.

C = A *op* B *where op is any of 8 operations.*

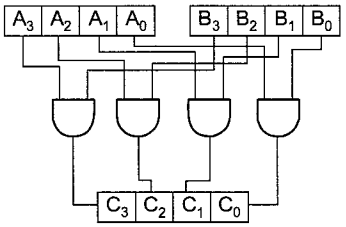
NOT: Complements the input.

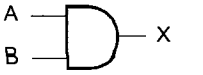
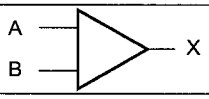
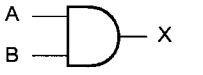
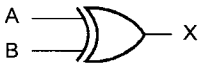
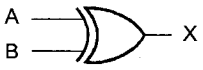
AND: The output is high if both the inputs are high.

OR: The output is high if any one of the input is high.
EX-OR: The output is high if either of the input is high.

2.1.3 4-bit AND Implementation

The following circuit perform the AND operation on two 4-bit number.



Name	Graphic Symbol	Algebraic Function	Truth Table
AND		X = A · B	A B X 0 0 0 0 1 0 1 0 0 1 1 1
OR		X = A + B	A B X 0 0 0 0 1 1 1 0 1 1 1 1
Inverter		X = A′	A X 0 1 1 1
Buffer		X = A	A X 0 0 1 1
NAND		X = (AB)′	A B X 0 0 1 0 1 1 1 0 1 1 1 0
NOR		X = (A+B)′	A B X 0 0 1 0 1 0 1 0 0 1 1 0
Exclusive-OR (XOR)		X = A ⊕ B	A B X 0 0 0 0 1 1 1 0 1 1 1 0
Exclusive-NOR or equivalence		X = (A ⊕ B)′	A B X 0 0 1 0 1 0 1 0 0 1 1 1

2.2 Datapath

System Bus

The CPU is connected to the rest of the system through system bus. Through system bus, data or information gets transferred between the CPU and the other component of the system. The system bus may have three components:

1. **Data Bus:** Data bus is used to transfer the data between main memory and CPU. A single memory unit is used for both instructions and data. In CPU instructions and data is stored in separate registers.
2. **Address Bus:** Address bus is used to access a particular memory location by putting the address of the memory location.
3. **Control Bus:** Control bus is used to provide the different control signals generated by CPU to different parts of the system. As for example, memory read is a signal generated by CPU to indicate that a memory read operation has to be performed. Through control bus this signal is transferred to memory module to indicate the required operation.

Registers in Datapath Design

There are mainly two types of registers: (1) User visible Registers and (2) Control and Status Registers.

User-Visible Registers

These enables the machine - or assembly-language programmer to minimize main memory reference by optimizing use of registers. The user-visible registers can be categorized as follows:

- **General-purpose registers** can be assigned to a variety of functions by the programmer. In some cases, general- purpose registers can be used for addressing functions (e.g., register indirect, displacement). In other cases, there is a partial or clean separation between data registers and address registers.
- **Data registers** may be used to hold only data and cannot be employed in the calculation of an operand address.
- **Address registers** are used for general purpose, or they may be devoted to a particular addressing mode. The following address registers may be used in the system.
- **Segment pointer:** In a machine with segment addressing, a segment register holds the address of the base of the segment. There may be multiple registers, one for the code segment and one for the data segment.
- **Index registers:** These are used for indexed addressing and may be auto-indexed.
- **Stack pointer:** If there is user visible stack addressing, then typically the stack is in memory and there is a dedicated register that points to the top of the stack.
- **Condition Codes** (also referred to as flags) are bits set by the CPU hardware as the result of the operations.

  For example, an arithmetic operation may produce a positive, negative, zero or overflow result. In addition to the result itself begin stored in a register or memory, a condition code is also set. The code may be subsequently be tested as part of a condition branch operation. Condition code bits are collected into one or more registers.

Control and Status Registers

These are used by the control unit to control the operation of the CPU. *Operating system programs may* also use these in privileged mode to control the execution of program. Processor status word is used as synonym for control and status registers.

Processor Status Word (PSW): All CPU designs include a register or set of registers, often known as the processor status word (PSW), that contains status information. The PSW typically contains condition codes plus other status information. Common fields or flags include the following:

(a) **Sign:** Contains the sign bit of the result of the last arithmetic operation.

(b) **Zero:** Set when the result is zero.

(c) **Carry:** Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high order bit.

(d) **Equal:** Set if a logical compare result is equal.

(e) **Overflow:** Used to indicate arithmetic overflow.

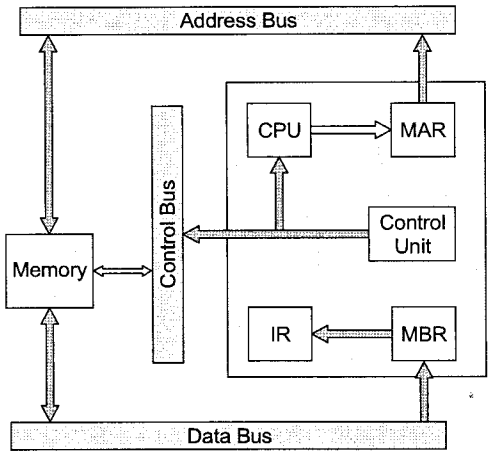
(f) **Interrupt enable/disable:** Used to enable or disable interrupts.

(g) **Supervisor:** Indicate whether the CPU is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

Apart from these, a number of other registers related to status and control might be found in a particular CPU design. In addition to the PSW, there may be a pointer to a block of memory containing additional status information (e.g. process control blocks).

Essential Registers for Instruction Execution

The following figure shows simple design of CPU datapath and control.



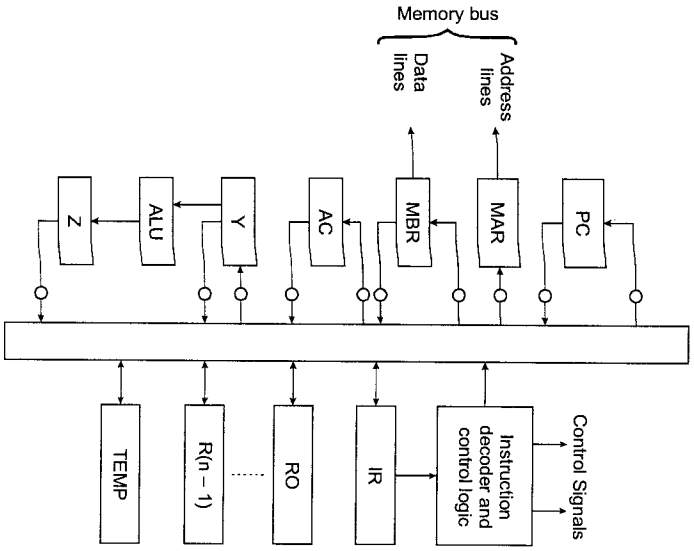
- **Program Counter (PC):** Contains the address of an instruction to be fetched. Typically, the PC is updated by the CPU after each instruction fetched so that it always points to the next instruction to be executed. A branch or skip instruction will also modify the contents of the PC.
- **Instruction Register (IR):** Contains the instruction most recently fetched. The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed.
- **Memory Address Register (MAR):** Contain address of a location of main memory from where information has to be fetched or information has to be stored. Contents of MAR is directly connected to address bus.
- **Memory Buffer Register (MBR or MDR):** Contains a word of data to be written to memory or the word most recently read. Contents of MBR is directly connected to the data bus. It is also known as Memory Data Register (MDR).

Apart from these specific registers, we may have some temporary registers which are not visible to the user. As such, there may be temporary buffering registers at the boundary to the ALU; these registers serve as input and output registers for the ALU and exchange data with the MBR and user visible registers.

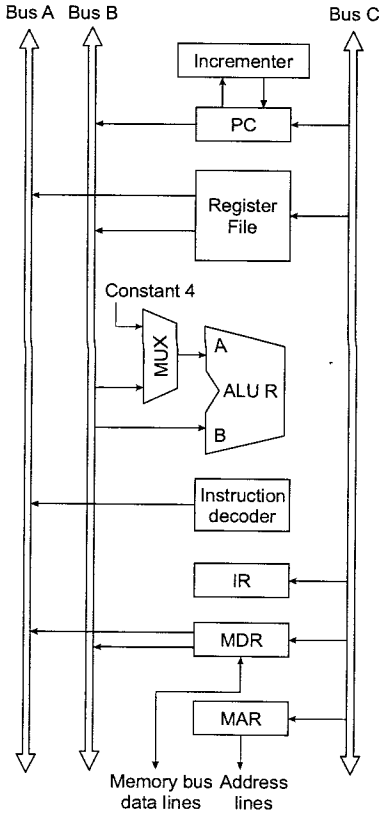
2.2.1 Bus Organizations of Datapath

Single Bus Organization (CPU Architecture)

CPU always stores the results of most calculations in one special register called Accumulator.



Three-bus Organization of Datapath (CPU Architecture)



2.2.2 Implementations of Datapath

1. **A single cycle datapath instruction implementation:** All micro-operations of an instruction are to be carried out in a single system clock cycle.
2. **Multi cycle data path instruction implementation:** In this multi-cycle design, we assume that the clock cycle can accommodate at most one of the following operations:
   (a) A memory access,
   (b) A register file access (two reads or one write), or
   (c) An ALU operation.

Hence, any data produced by one of these three functional units (the memory, the register file, or the ALU) must be saved, into a temporary register for use on a later cycle. If it were not saved then the possibility of a timing race could occur, leading to the use of an incorrect value.

***Example*-2.1** Suppose every instruction is one word long, as well as every address. How may memory accesses require the following instructions?

ADD, r_1, r_2, r_3 ; Register addressing
ADD r_1, r_2, (r_3) ; Direct addressing
ADD r1, r2, @ r_3; Memory indirect addressing

Solution:

ADD r_1, r_2, r_3 → require only one memory access, reading the instruction.
ADD r_1, r_2, (r_3) → require two memory access, the first to read the instruction and the other one to read the value from memory location.
Add r_1, r_2, @ r_3 → require three memory address, first to read the instruction second to get M[r_3], and third one to get M[m[r_3]].

***Example*-2.2** A 16 bit register contains the binary configuration:

0000 0000 0001 0010

that is the register contain 18_{10} in an unsigned representation. Show the register's content after left shifting by one, two and three positions respectively, and the decimal representation of the number?

Solution:

One bit left shift: 0000 0000 0010 0100
It is the binary representation of 36.

$$36 = 18 * 2$$

Two bit left shift: 0000 0000 0100 1000
It is the binary representation of 72.

$$72 = 18 * 4$$

Three bit left shift: 0000 0000 1001 0000
It is the binary representation of 144.

$$144 = 18 * 8$$

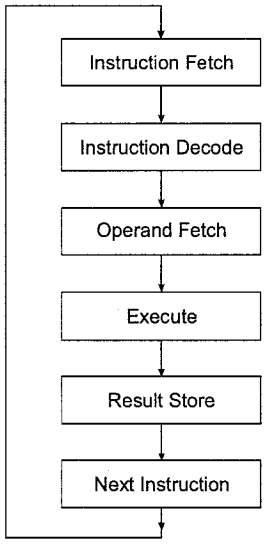
Example-2.3 An array of two integer (each integer = 32 bits) is placed in memory starting with address100. Show how to increment each element of the array using register addressing mode.

Solution:

```
Load r1, 100      # the base
Load r2, 1        # 1 will be used for increment
Add r3, r2, (r1)
Store (r1), r3
Add r1, r1, 4     # the next array element is at 104
Store (r1), r3
```

2.3 Control Unit

A CPU contains three main sections: the register section, the arithmetic/ logic unit (ALU), and the control unit. These sections work together to perform the sequences of micro-operations needed to perform the fetch, decode, and execution of every instruction in the CPU's instruction set (shown in the following Figure). The operation or task that must perform by CPU to execute an instruction are:



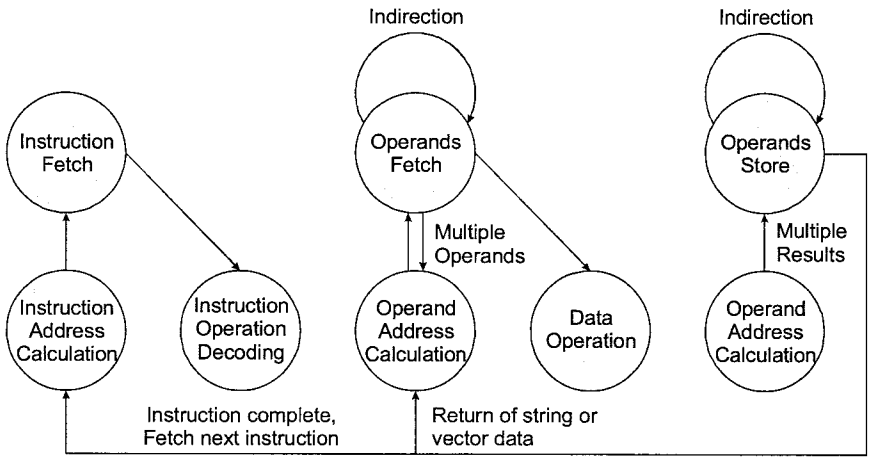
Instruction Execution Cycle

1. **Fetch Instruction:** An instruction, stored in the memory, is fetched into the control unit by supplying the memory with the address of the instruction.
2. **Decode (Interpret Instruction):** The control unit decodes the instruction in order to find the sequence of operation necessary to execute it.
3. **Memory (Operand Fetch/Data Fetch):** Any data necessary for an instruction is fetched from the memory by the control unit and stored in the datapath.
4. **Execute (Process data):** The operation (Arithmetic or Logical) is performed within the datapath.
5. **Write (result store):** The result of the operation is possibly written to the memory.

Instruction Cycle with States

For any given instruction cycle, some states may be null and other may be visited more than once.

- **Instruction Address Calculation:** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number of the address of the previous instruction.
- **Instruction Fetch:** Read instruction from the memory location into the processor.
- **Instruction Operation Decoding:** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **Operand Address Calculation:** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- **Operand Fetch:** Fetch the operand from memory or read it from I/O.
- **Data Operation:** Perform the operation indicated in the instruction.
- **Operand Store:** Write the result into memory or out to I/O.



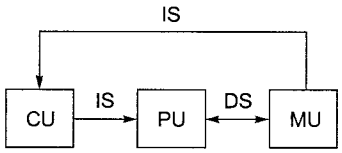
State Diagram for Instruction Cycle

NOTE

In some machines, a single instruction can specify an operation to be performed on a vector (one-dimensional array) of numbers or a string of characters. This would involve repetitive operand fetch and/or store operation for a particular opcode (opcode is fetched only once).

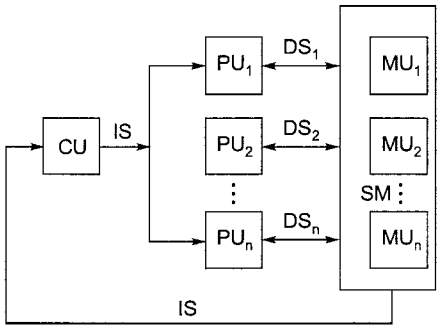
Flynn's Classification

1. **Single Instruction Stream, Single Data Stream (SISD):** A computer with a single processor is called a Single Instruction Stream, Single Data Stream (SISD) Computer. It represents the organization of a single computer containing a control unit, a processor unit, and a memory unit. Instructions are executed sequentially and the system may or may not have internal parallel processing. Parallel processing may be achieved by means of a pipeline processing.
   In such a computer a single stream of instructions and a single stream of data are accessed by the processing elements from the main memory, processed and the results are stored back in the main memory. SISD computer organization is shown in figure below.

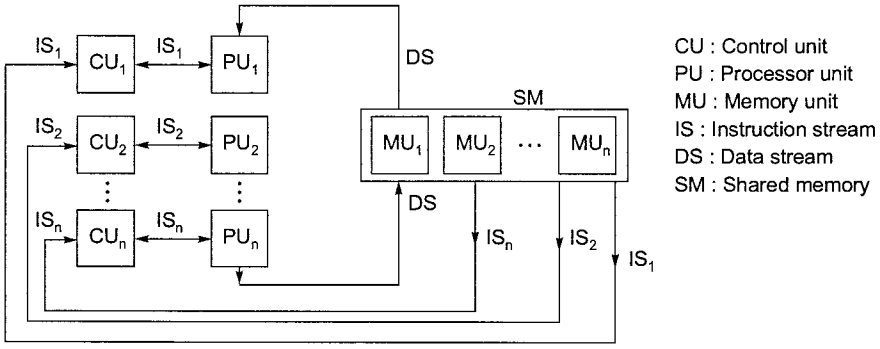


CU : Control unit
PU : Processor unit
MU : Memory unit
IS : Instruction stream
DS : Data stream

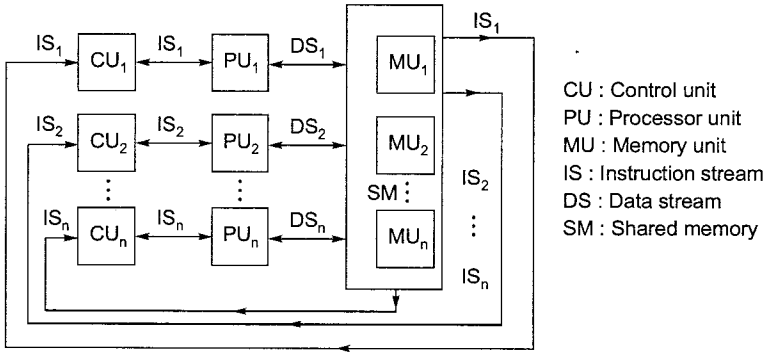
2. **Single Instruction Stream, Multiple Data Stream (SIMD):** It represents an organization of computer which has multiple processors under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of the data. SIMD computers are used to solve many problems in science which require identical operations to be applied to different data set synchronously. Examples are added a set of matrices simultaneously, such as $\sum_i \sum_k (a_{ik} + a_{ik})$. Such computers are known as array processors. SIMD computer organization is shown in figure below.



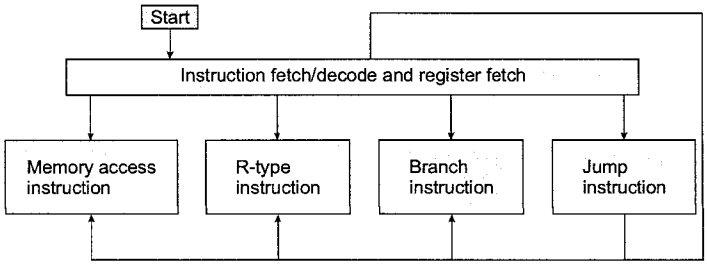
3. **Multiple Instruction Stream, Single Data Stream (MISD):** It refers to the computer in which several instructions manipulate the same data stream concurrently. In the structure different processing element run different programs on the same data. This type of processor may be generalized using a 2-dimensional arrangement of processing element. Such a structure is known as systolic processor. MISD computer organization is shown in figure below.



4. **Multiple Instruction Stream, Multiple Data Stream (MIMD):** MIMD computers are the general purpose parallel computers. Its organization refers to a computer system capable of processing several programs at a same time. MIMD systems include all multiprocessing systems. MIMD computer organization is shown in figure below.]



2.3.1 Multi Cycle Datapath and Control



High level view of finite state machine control

1. **Fetch Instruction:** An instruction, stored in the memory, is fetched into the control unit by supplying the memory with the address of the instruction.
   - Assume each word is 4 bytes and each instruction is stored in a word, and that the memory is byte addressable.
   - PC (Program Counter) contains address of next instruction.

     IR ⇐ [[PC]]

     PC ⇐ [PC] + 4
2. **Decode (Interpret Instruction):** The control unit decodes the instruction in order to find the sequence of operation necessary to execute it.
   - First, read the **opcode** to determine instruction type and field lengths.
   - Second, read in data from all necessary registers.

   Example:

   (*i*) A <= Reg [IR [25 : 21]]; // Reg. rs in opcode (instruction part)

   B <= Reg [IR [20 : 16]]; // Reg. rt in opcode (instruction part)

   (*ii*) ALU_{Out} <= PC + (sign-extend (IR [15-0]) << 2); // Branch target address(instruction part). The output of the ALU, which is the value PC + 4 during instruction fetch. This value should be stored directly in to the PC.
3. **Execution, Memory Address Computation, or Branch Completion**

   Example:

   (*i*) Memory reference: ALU_{Out} <= A + sign-extend (IR [15 : 0]);

   (*ii*) Arithmetic-logical instruction (R-type): ALU_{Out} <= A op B;

   (*iii*) **Branch:** if (A == B) PC <= ALU_{Out}; // conditional

   A branch instruction would calculate: Program counter = Register + Branch address;

   (*iv*) **Jump:** PC <= PC [31 : 28]. (IR [25 : 0] << 2); //// unconditional

   // x.y is concatenation of bit fields x and y

   The lower 26 bits of the Instruction register (IR) shifted left by two and concatenated with the upper 4 bits of the incremented PC, which is the source when the instruction is a jump.

NOTE

- The PC is written both unconditionally and conditionally.
- During a normal increment and for jumps, the PC is written unconditionally.
- If the instruction is a conditional branch, the incremented PC is replaced with the value in ALU_{Out} only if the two designated registers are equal.

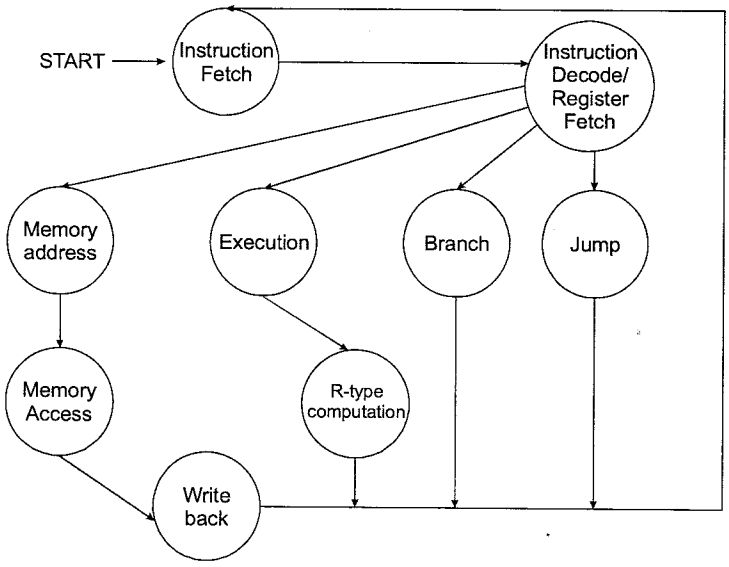
4. **Memory Access or R-type Instruction Completion Step:** Actually only the load and store instructions do anything during this stage; for the other instructions, they remain idle during this stage.

   Example:

   (*i*) Memory reference: MDR <= Memory [ALUOut]; // Load
   or Memory [ALU_{Out}] <= B; // Store from register to Memory.

   (*ii*) Arithmetic-logical instruction (R-type): Reg [IR [15:11]] <= ALUOut; // register destination in opcode

5. **Memory Read Completion Step (Register Write):**
   - Most instructions write the result of some computation into a register.
     Examples: Arithmetic, Logical, Shifts, Loads, etc.
     Load: Reg[IR[20:16]] <= MDR;
   - What about stores, branches, jumps?
   - They do not write anything into a register at the end.
   - These remain idle during this fifth stage.



High level view of finite state machine control

***Example*-2.4** Find the stages of datapath and control (Execution sequence) for ADD R1, R2, R3 ; R3 <= R1 + R2

Solution:

Instruction: ADD R3, R1, R2 ;

- **Stage 1:** Fetch the instruction, and increment PC.
- **Stage 2:** Decode to find that it is an ADD instruction, then read registers R1 and R2.
- **Stage 3:** Add the two values retrieved in stage 2.
- **Stage 4:** Idle (nothing to write to memory).
- **Stage 5:** Write result of stage 3 into register R3.

Example - 2.5 Find the stages of datapath and control (Execution sequence) for MOV 20 (R1), R2; [20 + R1] <= R2.

Solution:

Instruction: MOV 20(R1), R2;

- **Stage 1 (Fetch):** Fetch the instruction and increment PC.
- **Stage 2 (Decode):** Decode to find it is MOV, then read registers R1 and R2.
- **Stage 3 (Execute):** Add 20 to value in register R1.
- **Stage 4 (Memory Access):** Write value in register R2 into memory address computed in stage 3.
- **Stage 5 (Register Write):** Go idle (nothing to write into a register).

Example - 2.6 Find the stages of datapath and control (Execution sequence) for MOV R2, 20 (R1); R2 = [20 + R1] which reads a word from memory address (Base-indexed).

Solution:

Instruction: MOV R2, 20(R1);

- **Stage 1 (Fetch):** Fetch the instruction and increment PC.
- **Stage 2 (Decode):** Decode to find it is MOV, then read registers R1 and R2.
- **Stage 3 (Execute):** Add 20 to value in register R1.
- **Stage 4 (Memory Access):** Read value from memory address compute in stage 3.
- **Stage 5 (Register Write):** Write value found in stage 4 into register R3.

Remember

- Hardware Needed for Datapath and Control (CPU execution)
- **PC:** Which keeps track of address of the next instruction.
- **General Purpose Registers:** Used in stages 2 (read/fetch) and 5 (register write).
- **Memory:** Used in stage1 (fetch) and stage4 (R/W -Memory access).
- **ALU:** Which is used in computation.
- **System Bus:** Used for address, data and control signals

Control Unit

Control unit generates the signals for sequencing the operations in the datapath. It performs the task by repeatedly cycling through fetch-execute cycle steps:

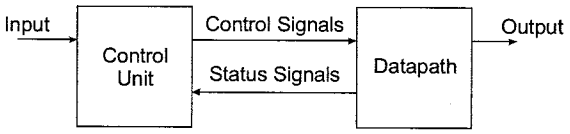
- Read the instruction that's pointed to by the PC from memory and move it into the IR.
- Increment the PC.
- Decode the instruction in the IR.
- If the instruction has to read an operand from memory, calculate the operand's address (*effective address*) and read the operand from memory.
- Execute the current instruction from the IR.

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence.

2.3.2 Functions of Control Unit

1. **Fetch and instruction sequencing:** Generates control signal to fetch instruction from memory and the sequence of operations involved in processing an instruction.
2. **Instruction interpretation and execution:**
   - Interpreting the operand addressing mode implied in the operation code and fetching the operands.
   - Sequencing the successive micro operations on the data path to execute the operation code specified in the instruction.
3. **Interrupt processing:** Process unmasked interrupts in the interrupt cycle as follows:
   - Suspend execution of current program
   - Save context
   - Set PC to start address of interrupt handler routine
   - Process interrupt
   - Restore context and continue interrupted program

2.3.3 Control Unit and Datapath



Control Unit: It generates the signals for sequencing the micro-operations in the datapath based on status and input signals.

Datapath: It implements the micro-operations under control of the control unit using its functional units (registers, ALU, MUXex, Buses, etc.)

2.3.4 Microoperations and Control Signals

Microoperation: Each micro-operation is executed in one clock cycle. Cycle time determined by the longest micro-operation. Micro-operations for (different) instructions (for non-conflicting, independent : no write to/read from with same register at the same time) can be executed simultaneously and must have proper sequence as shown in the following.

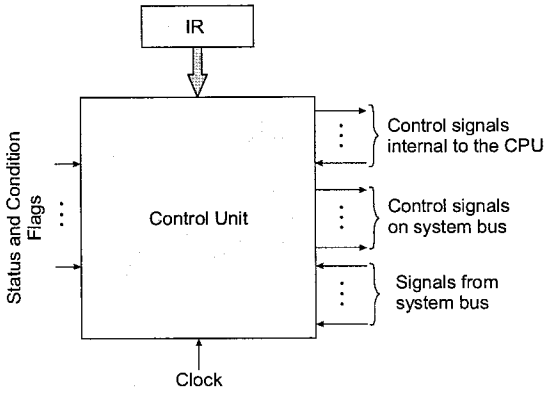
Microoperations	Instruction Cycles
s1 : MAR ← (PC), READ **s2** : MBR ← (Memory) **s3** : PC ← (PC) + 1 **s4** : IR ← (MBR)	**t1** : MAR ← (PC), READ **t2** : MBR ← (Memory) PC ← (PC) + 1 **t3** : IR ← (MBR)
Number of Microoperations = 4	**Number of Cycles = 3**

2.3.5 Types of Microoperations

- Transfer data from one register to another,
- Transfer data from register to external (memory or I/O),
- Transfer data from external to register,
- ALU or logical operation between registers.

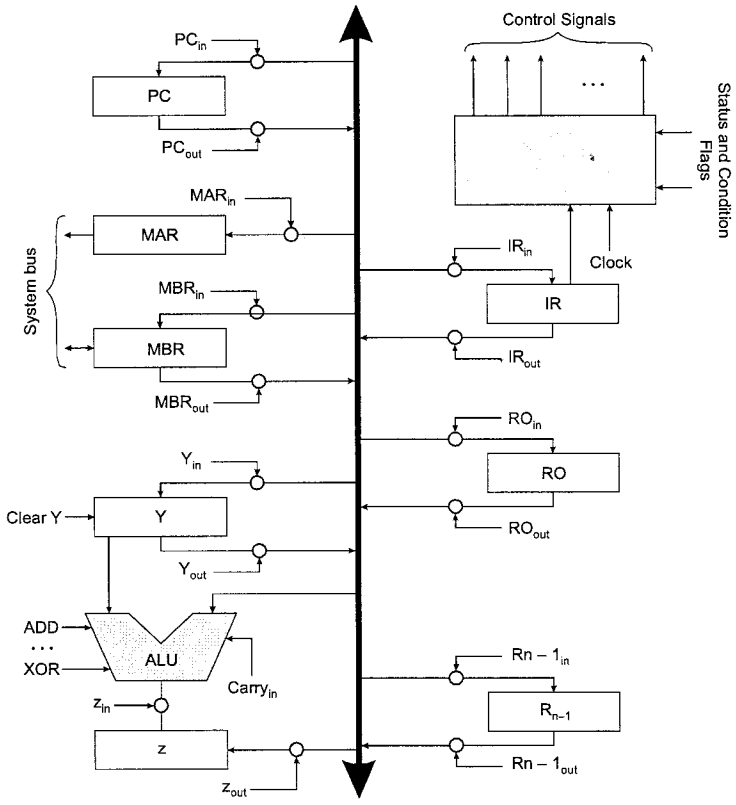
Control Signal

A signal generated by control unit called as control signal. The control unit is driven by the processor clock. The signals to be generated at a certain moment depend on the following:



- Actual step to be executed
- Condition and status flags of the processor
- Actual instruction executed
- External signals received on the system bus (e.g. interrupt signals)

In order to allow the execution of a micro-operation, one or several control signals (shown in the following figure) have to be issued; they allow the corresponding data transfer and/or computation to be performed.



Control Signals and Data Path

Examples:

(a) Control signals for transferring content of register R0 to R1: $R0_{out}$, $R1_{in}$

(b) Control signals for adding content of Y to that of R0 (result in Z): $R0_{out}$, Add, Z_{in}

(c) Control signals for reading a memory location; address in R3: $R3_{out}$, MAR_{in}, Read

NOTE: The CPU executes an instruction as a sequence of control steps. In each control step one or several micro-operations are executed.

Example-2.7 Write the control system (microoperation) for the following instruction:

Instruction: ADD R1, R3; R1 ← R1 + R3

Solution:

Control steps and Control signals for instruction ADD R1, R3:

Fetch Instruction PC ← PC + 1

1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
2. Z_{out}, PC_{in}
3. MBR_{out}, IR_{in}
4. $R1_{out}$, Y_{in}
5. $R3_{out}$, Add_{out}, Z_{in}
6. Z_{out}, $R1_{out}$, End

Example-2.8 Write the control system (microoperations) for the following instruction:

Instruction: ADD R1, (R3); R1 ← R1 + (R3)

Solution:

Control steps and Control signals for instruction ADD R1, (R3):

Fetch Instruction PC ← PC + 1

1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
2. Z_{out}, PC_{in}
3. MBR_{out}, IR_{in}
4. $R1_{out}$, MAR_{in}, Read ← Indirect addressing
5. $R1_{out}$, Y_{in}
6. MBR_{out}, Add, Z_{in}
7. Z_{out}, $R1_{in}$, End

Example-2.9 Write the control system (microoperation) for the following instruction:

Instruction: BR Target address; Uconditional brach with relative addressing

Solution:

Control steps and Control signals for instruction BR Target:

Fetch Instruction PC ← PC + 1

1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
2. Z_{out}, PC_{in}
3. MBR_{out}, IR_{in}
4. PC_{out}, Y_{in}
5. (Displacement-field) R_{out}, Add, Z_{in}
6. Z_{out}, PC_{in}, End

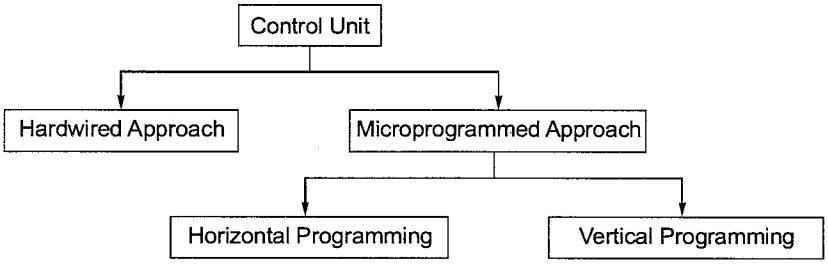
NOTE

Differences of Indirect, Interrupt and Execute cycles: Indirect cycle, interrupt cycles are not changed but execute cycle may differ for each instruction.

Indirect Cycle	Interrupt Cycle	Execute Cycle
t1: MAR ← ($IR_{address}$)	t1: MBR ← (PC)	t1: MAR ← ($IR_{address}$)
t2: MBR ← (mem)	t2: MAR ← Save-address	t2: MBR ← (mem)
t3: IR ← (MBR)	t3: mem ← (MBR)	t3: R1 ← (MBR)

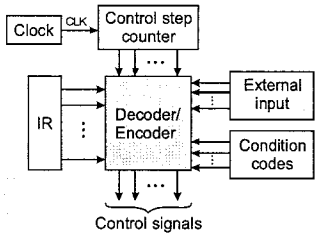
2.3.6 Control Unit Implementation

The main objective of control unit is to generate the control signal in proper sequence. Control unit is implemented in one of two ways either Hardwired control or Micro-programmed control.



2.3.7 Hardwired Control Unit

Hardwired Control unit is made up of sequential and combinational circuits (Hardware) to generate the control signals.



Control Unit Organization

Decoder/Encoder Block

It is a combinational circuit that generates the required control signals depending on the state of all its input.

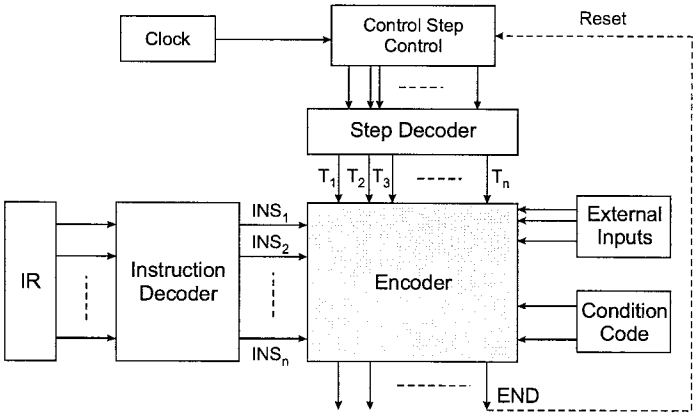
(*i*) **Step Decoder:** The decoder part of decoder/encoder part provide a separate signal line for each control step, or time slot in the control sequence.

(*ii*) **Instruction Decoder:** The output of the instructor decoder consists of a separate line for each machine instruction loaded in the IR, one of the output line INS_1 to INS_m is set to 1 and all other lines are set to 0.

(*iii*) **Encoder:** It is required to generate many control signals by the control unit. These are basically coming out from the encoder circuit of the control signal generator.

The control signals are: PC_{in}, PC_{out}, Z_{in}, Z_{out}, MAR_{in}, ADD, END, etc.

The encoder sends a reset signal after the end of an instruction and a stop signal to the sequencer after the last sequence. The encoder also sends count start signal to let the clock increment the counter during processing of an instruction.



Detailed view of Control Unit Organization

External Inputs

External interrupts can be from an external interrupting source and an external device requesting access to external buses. The external inputs represent the state of the CPU and various control lines connected to it, such as MFC status signal.

Condition/Status Flags

These are as per conditions/status flags set in earlier instructions. The condition codes/status flags indicates the state of the CPU. These includes the status flags like carry, overflow, zero, etc.

Control Signals

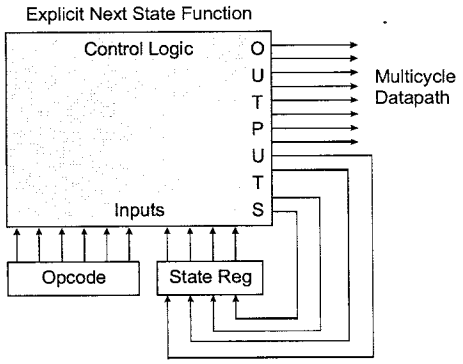
Required control signals are uniquely determined by the following information:

- Contents of the control counter
- Contents of the instruction register
- Contents of the condition code and other status flags

Three sets of control signals that are outputs in various states:

1. **Function select control signals:** To select function signals f1, f2, ... fy – 2, fy – 1
   Example: **f1:** Data transfer, **f2:** Add
2. **Storage unit control signals:** To select storage units r1, r2, ..., rz – 2, rz – 1
   Example:
   r1 : Storing unit PC output control (PC_{out})
   r2 : Storing unit PC input control (PC_{in})
   r3 : Storing unit MAR output control (MAR_{out})
   r4 : Storing unit MAR input control (MAR_{in})
3. **Data route select control signals:** To select data route signals d1, d2, ..., dr – 2, dr – 1.
   Example:
   d1: Data route internal bus control
   d2: Data route external address bus output control
   d3: Data route external data bus output control
   d4: Data route external data bus input control

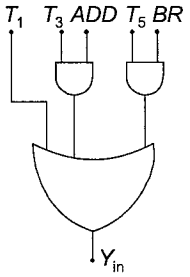
Let c0, c1, ..., cn be the sequence/step counter output from the instance activation of the count output of encoder. Then PC → MAR implements by a control signal r4 ← c0.f1.d1.r1.



The advantage of hardwired control is that is very fast. The disadvantage is that the instruction set and the control logic are directly tied together by special circuits that are complex and difficult to design or modify.

The control signals are expressed as Sum-of-Product (SOP) expression and they are directly realized on the independent hardware.

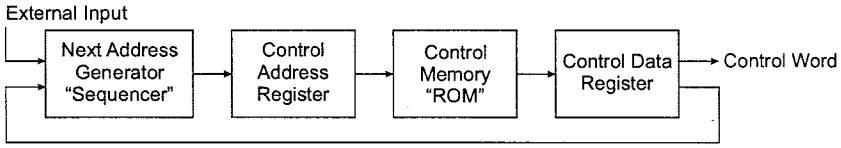
Example: $Y_{in} = T_1 + T_3 \times ADD + T_5 \times BR + ...$
Here Y_{in} is enabled during T_1 for all instructions, during T_3 for *ADD* and so on...
It can be realised as:



2.3.8 Microprogrammed Control Unit

A control unit whose binary control variables are stored in memory is called a micro-programmed control unit. A control memory (Control storage) on the processor contains micro-programs that activate the necessary control signals whereas hardwired control unit generate control signals by sequential and combinational circuits.

- **Microinstruction:** The microinstruction specifies one or more micro-operations for the system. It Contains a control word and a sequencing word.
- **Microprogram:** A sequence of microinstructions called a microprogram. Program stored in memory that generates all the required control signals to execute the instruction set correctly.



Control Memory (Control Storage): It is Memory unit in the micro-programmed control unit to store the micro-program. Each word in control memory contains within it a microinstruction.

Control Address Register: It specifies the address of the microinstruction in control memory.

Control Data Register: It holds the microinstruction read from memory.

The location of the next microinstruction may be the one next in sequence, or it may be locate somewhere else in the control memory. For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.

The next address may also be a function of external input conditions. While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

Sequencing Word: Information needed to decide the next microinstruction address.

Next Address Generator (Sequencer or Microprogram Sequenceer): It determines the microinstruction Address to be executed in the next clock cycle.

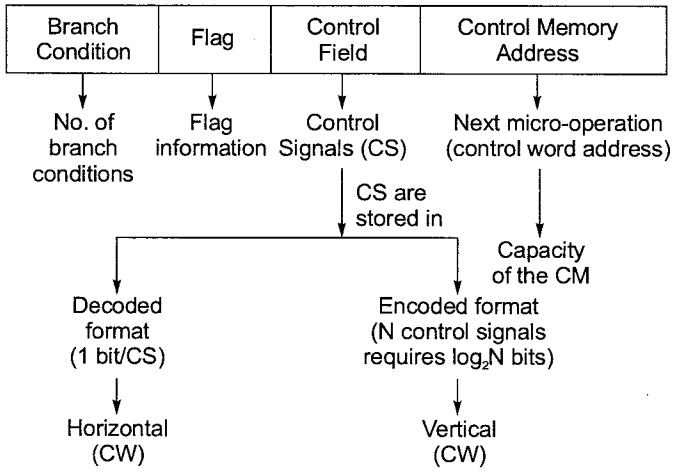
The address of the next microinstruction can be specified several ways, depending on the sequencer inputs as follows:

(*i*) Incrementing the control address register by one,
(*ii*) Loading into the control address register an address from control memory,
(*iii*) Transferring an external address, or
(*iv*) Loading an initial address to start the control operations.

Determining the address of the next microinstruction depends on one of the following:

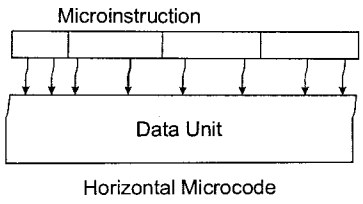
- In-line Sequencing
- Branch
- Conditional Branch
- Subroutine
- Loop
- Instruction OP-code mapping

Control word: The control variables at any given time can be represented by a string of 1's and 0's called a control word. It has all the control information required for one clock cycle. Which can be programmed to perform various operations on the component of the system.

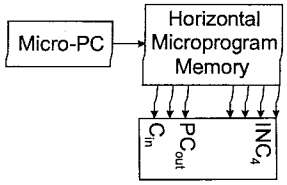


Types of Microinstructions

1. Horizontal Microprogramming:
   - There are no intermediate decoders and the control word bits are directly connected to their destination.
   - Each bit in the control word is directly connected to some control signal.
   - The total number of bits in the control word is equal to the total number of control signals in the CPU.
   - Each Microinstruction specifies many different microoperations to be performed in parallel.
   - All control signals directly in micro-code
   - Due to lot of signals, many bits in micro-instruction.

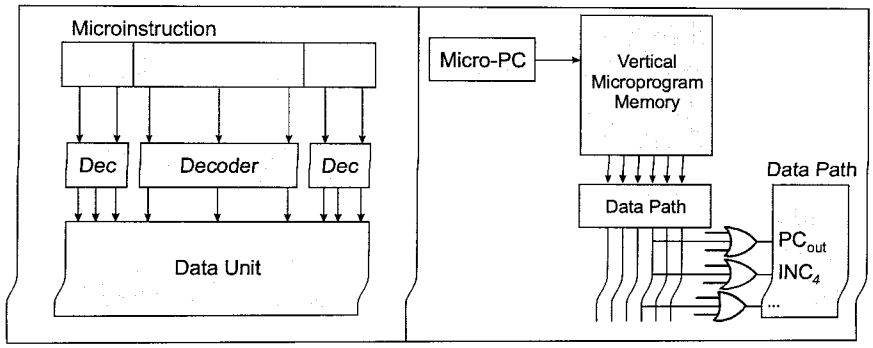


Horizontal Microcode



Horizontal Microprogramming

2. Vertical Microprogramming:

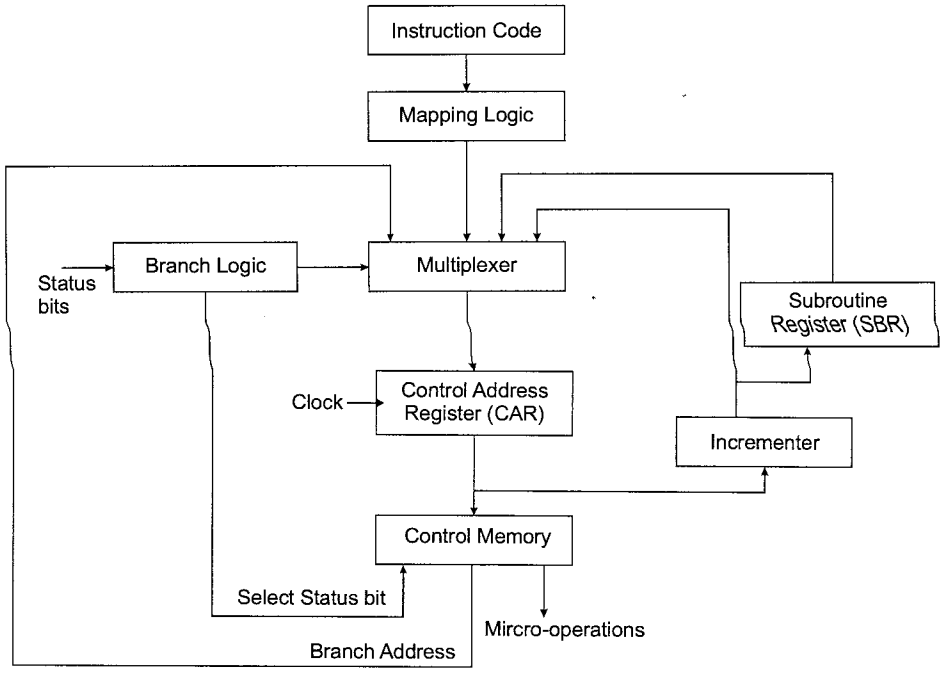


Vertical Microcode **Vertical Microprogramming**

- Vertical microcode schemes employ an extra level of decoding to reduce the control word width.
- From an n bit control word we may have 2^n bit signal values.
- It takes less space but may be slower
- Actions need to be decoded to signals at execution time
- Each Microinstruction specifies single or few microoperations to be performed.

Implementation of Sequencer

The following figure shows all needed hardware for selecting the next microinstruction address. The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the address is obtained.



Micro Instructions Sequencer

- There are four different paths from which the *Control Address Register* receives the address with the help of *Multiplexer*.
- *Branching* is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- *An external address* is transferred into control memory via a **mapping logic** circuit.
- *Incrementer* increments the content of the control address register by one, to select the next microinstruction in sequence.
- The return address for a subroutine is stored in a special register called *Subroutine Register* whose value is then used when the microprogram wishes to return from the subroutine.

2.3.9 Difference between Hardwired and Micro Programmed Control Unit

	Hardwired Control	Micro-program Control
Initial represents:	Finite State Diagram (Moore Machine)	Microprogram
	⇩	⇩
Sequencing control:	Explicit Next State Function	Microprogram Counter
	⇩	⇩
Logic represent:	Logic Equations	Truth Tables
	⇩	⇩
Implementation:	PLA (Programmable Logic Array)	Read Only Memory

Advantage of Micro-programmed Control Unit

Once the hardware configuration is established, there should be no need for further hardware or wiring changes. If we want to establish are different control sequence for the system, all we need to do is specify different set microinstructions for control memory.

The hardware configuration should not be changed for different operations, the only thing that must be changed is the micro-program residing in control memory.

Dynamic Microprogramming

Computer system whose control unit is implemented with a microprogram in WCS. Microprogram can be changed by a systems programmer or a user. Dynamic microprogramming permits a microprogram to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic micro-programming employ a writable control memory. This type of memory can be used for writing (to change the microprogram) but is used mostly for reading.

Summary

- **ALU:** It is a combinational circuit that performs logic and arithmetic operations.
- **Datapath:** It is combination of functional units and registers.
- **Half Adder(x, y):** Half adder is a combinational circuit which is capable of adding two bits.
  Sum = x'y + xy' and Carry C = xy
- **Full Adder(x, y, z):** Full adder is a combinational circuit which is capable of adding three bits.
  Sum S = x'y'z + x'yz' + xy'z' + xyz = z ⊕ (x ⊕ y) and Carry C = xy + xz + yz
- In one bus data path, CPU registers and ALU use same bus for all incoming and outgoing data.
- In two bus data path, general purpose registers are connected to two buses. Data can be transferred from two different registers to the input point of the ALU at the same time.
- In three bus data path, two buses are used as source (moves data out of registers) while the third is used as destination (Moves data into registers).
- Source bus is also called as in-bus and destination bus is also called as out-bus.
- **PC:** It is used to hold the starting instruction address and immediately point the next instruction's address.
- **IR:** It is used to hold the currently fetched instruction to decode. As instruction format it predefined in this register therefore it is not a user accessible register.
- **Accumulator:** It is used to hold the one of the ALU input and output.
- **MAR (Memory Address Register):** It is used to carry the address. This register is directly connected to the address lines of the system bus.
- **MBR (Memory Buffer Register) or MDR (Memory Data Register):** It is used to carry the binary sequence. This register is directly connected to the data lines of the system bus.
- **Functions of Control Unit:** Sequencing (Causing the CPU to step through a series of micro-operations), Execution (Causing the performance of each micro-operation) and Use of Control Signals to accomplish the task.
- **Memory-Function-Complete (MFC):** It is a control signal. The memory set this signal to 1 to indicate that the contents of the specified memory location are available in memory data bus.
- The control unit is in charge of coordinating the activities inside the CPU and the interaction with the outside. It is doing this by issuing in each clock cycle the appropriate control signals.
- A set of control signals activates the micro-operations which have to be executed in a given control step.
- Control units can be implemented as hardwired or micro-programmed.
- A hardwired control unit is a combinatorial circuit which gets a set of inputs and transforms them into a set of control signals.
- A micro-programmed control unit is implemented like another CPU inside the CPU. It executes micro-programmes stored in the control store.
- Each instruction of the micro-program practically represents the set of signals which the control unit has to issue in the respective control step.

- Microinstruction contains a control word that specifies one or more microoperations for the data processor.
- The microroutines contain, beside controlwords, also branches which have to be interpreted by the microprogrammed controller.
- Hardwired controllers are faster than micro-programmed ones. They are used in all RISCs. If the instruction set is complex, hardwired controllers become too complicated. So CISCs are implemented with micro-programmed controllers. Microprogram also called as firmware.
- Vertical Microprogrammed organization: highly encoded schemes that use compact codes to specify only a small number of control functions in each microinstruction.
- Horizontal Microprogrammed organization: minimally encoded scheme in which many resources can be controlled with a single microinstructions.

Student's Assignment

Q.1 The computer performs all mathematical and logical operations inside its
(a) memory unit
(b) central processing unit
(c) output unit
(d) visual display unit

Q.2 Booth's algorithm is used in signed-data
(a) addition (b) subtraction
(c) multiplication (d) division

Q.3 Both's coding in 8-bits for the decimal number –57 is _____
(a) 0 – 100 + 1000
(b) 0 – 100 + 100 – 1
(c) 0 – 1 + 100 – 10 + 1
(d) 00 – 10 + 100 – 1

Q.4 Consider the following:
1. Input device
2. Arithmetic and logic unit
3. Control unit
4. Auxiliary memory
5. Main memory
6. Active hub

Which of these form part of CPU?
(a) 1, 4 and 6 (b) 2, 3 and 6
(c) 2, 4 and 5 (d) 2, 3 and 5

Q.5 In a microprocessor when a CPU is interrupted, it
(a) Stops execution of instructions
(b) Acknowledges interrupt and branches of subroutine
(c) Acknowledges interrupt and continues
(d) Acknowledges interrupt and waits for the next instruction from the interrupting device

Q.6 The content of which of the following determines the state of the CPU at the end of the execute cycle (when the interrupt is recognized)?
1. Program counter
2. Processor register
3. Certain status conditions

Select the correct answer using the codes given below:
(a) 1 and 2 (b) 2 and 3
(c) 1 and 3 (d) 1, 2 and 3

Q.7 **Statement (I):** The data path contains all the circuits to process data within the CPU with the help of which data is suitably transformed.
Statement (II): It is the responsibility of the control path to generate control and timing signals as required by the opcode.
(a) Both Statement (I) and Statement (II) are individually true and Statement (II) is the correct explanation of Statement (I).
(b) Both Statement (I) and Statement (II) are individually true but Statement (II) is not the correct explanation of Statement (I).
(c) Statement (I) is true but Statement (II) is false.
(d) Statement (I) is false but Statement (II) is true.

Q.8 An instruction cycle refers to
(a) Fetching an instruction
(b) Clock speed
(c) Fetching, decoding and executing an instruction
(d) Executing an instruction

Q.9 What is the control unit's function in the CPU?
(a) to decode program instructions
(b) to transfer data to primary storage
(c) to perform logical operations
(d) to store program instructions

Q.10 Which of the following affects processing power?
(a) data bus capacity
(b) addressing scheme
(c) clock speed
(d) all of the above

Q.11 A _____ is required to translate such microprogram into executable programs that can be stored in the control memory in microprogramming.
(a) microassembler
(b) microcompiler
(c) microprogrammed CPU
(d) microprogrammed counter

Q.12 Microinstruction length is determined by _____.
1. The maximum number of simultaneous micro operations that must be specified.
2. The way in which the control information is represented or encoded.
3. The way in which the next microinstruction address is specified.

(a) 1 and 2 (b) 2 and 3
(c) 1 and 3 (d) all of the above

Q.13 Consider a CPU has 8 general-purpose registers R_0, R_1, ..., R_7 and supports the following operations.

ADD R_a,R_b,R_c ; $R_c \leftarrow R_a + R_b$
MUL R_a,R_b,R_c ; $R_c \leftarrow R_a \times R_b$

An operation normally takes one clock cycles, an operation takes two clock cycles if it produces a result required by the immediately following operations. Consider the expression XY + XYZ + YZ, where variables X, Y and Z are initially located in the registers R_0, R_1 and R_2. If contents these registers must not be modified, what is the minimum number of clock cycles required for an operation sequence that computes the value of XY + XYZ + YZ?
(a) 4 (b) 5
(c) 6 (d) 7

Common Data Q.14 & Q.15

Consider an accumulator-based CPU supports only single address instruction. The CPU supports the following instructions.
LOAD A : Load A from memory into accumulator AC.
STORE A : Store contents of accumulator AC in memory location M.
ADD B : Load B into data register (DR) and add to accumulator.
MOVE A, B : Move content of B to A.
SUB A, B : Subtract B from A.

Q.14 Assume CPU uses the memory referencing and each instruction LOAD, STORE and ADD takes one clock cycle. To compute Z = X + Y CPU takes how many minimum number of clock cycles?
(a) 2 (b) 3
(c) 4 (d) 5

Q.15 How many minimum clock cycles are needed when accumulator becomes O. If each instruction takes one clock cycles?
(a) 0 (b) 1
(c) 2 (d) 3

Q.16 A bi-directional bus between a memory and CPU, possibly has three states. While the bus behaves like a open circuit, the output does not carry a signal in other state it carry signal in both direction. Which one of the following 3rd state?
(a) 0 (b) 1
(c) High impedance (d) (a) and (c) both

Q.17 The following are the some of the sequences of operations in instruction cycle, which one is correct sequence?
(a)

PC → Address register
Data from memory → Data register
Data register → IR
PC + 1 → PC

(b)
Address register → PC
Data register → Data from memory
Data register → IR
PC + 1 → PC

(c)
Data from memory → Data register
PC → Address register
Data register → IR
PC + 1 → PC

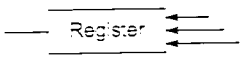
(d) None of these

Q.18 Horizontal microinstruction have which of the following attributes?
1. Short formats
2. Limited ability to express parallel micro operations
3. Considerable encoding of the control information

(a) 1 and 2 (b) 2 and 3
(c) 1, 2 and 3 (d) None of these

Q.19 Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted, way of achieving this _____.
(a) Is by means of a strobe pulse from one of the unit
(b) Is by means of handshaking
(c) Both (a) and (b)
(d) *None of these*

Q.20 Consider the following figure



The above circuit represents _____.
1. *Hard-wired control unit*
2. Micro programmed control unit

(a) Only 1 (b) Only 2
(c) Both 1 and 2 (d) Either 1 or 2

Q.21 The sequence of events that happen during a typical fetch operation is
(a) PC → Mar → Memory → MDR → IR, PC ← PC+1
(b) PC → Memory → MDR → IR, PC ← PC+1
(c) PC → Memory → IR, PC ← PC+1
(d) PC → Mar → Memory → IR, PC ← PC+1

Q.22 Determine the width of Micro-instruction having following Control signal field, in a Vertical Micro-programmed Control Unit
1. Next Address field of 7 Bits
2. ALU Function field selecting 1 out of 13 ALU Function
3. Register-in field selecting 1 out of 13 ALU Function
4. Register-out field selecting 1 out of 8 Registers
5. Shifter field selecting no shift, right shift or left shift
6. Auxiliary control field of 4 bits

(a) 22 (b) 23
(c) 24 (d) 25

Q.23 Hardwired control units are faster than Micro-programmed control unit because
(a) They do not consist of slower memory elements
(b) They do not have slower element such as Gates and Flip-flops
(c) They are made using faster VLSI design technology
(d) They contain high speed digital components

Answer Key:

1. (b) **2.** (c) **3.** (a) **4.** (d) **5.** (d)

11. a **12.** c **13.** c **14.** (b) **15.** (d)
16. c **17.** (a) **18.** (d) **19.** (c) **20.** (c)
21. (a) **22.** (b) **23.** (a)

CHAPTER 03

Instruction Pipelining

3.1 Performance

Performance is the time taken by the system to execute a program. Performance affected by the following parameters:

- Clock speed
- Type and number of instructions available
- Average time required to execute an instruction
- Memory access time
- Power dissipation in the system
- Number of I/O devices and types of I/O devices connected
- The data transfer capacity of the bus

Remember

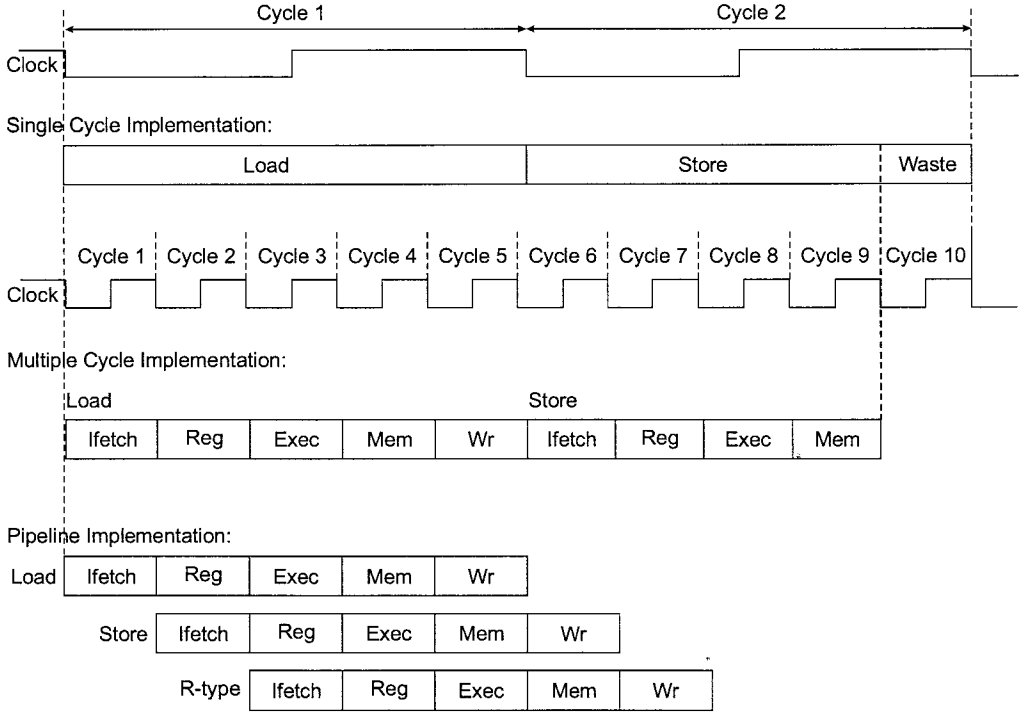
- **CPI:** Cycles per instruction
- **IPC:** Instructions per cycle.
- IPC = 1/CPI
- CPU clock cycles = Number of instructions × Average clock cycles per Instruction.
- CPU clock cycles = $\sum_{i=1}^{n} N_i \times CPI_i$
- CPU execution time = CPU clock cycles × Clock cycle time
- CPU execution time = CPU clock cycles / Clock rate
- CPU execution time = Instruction Count × CPI × Clock Cycle Time
- CPU execution time = $\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycle}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$
- CPU execution time = Seconds / Program
- Speedup = (Earlier execution time) / (Current execution time).

3.2 Instruction Processing

Single Cycle	Multiple Cycle	Pipeline
• Cycle time long enough for longest instruction	• Cycle time long enough for longest stage	• Cycle time long enough for longest stage
• Entire cycle acts as a single stage	• Shorter stages waste time	• Shorter stages waste time
• Shorter instruction waste time	• Shorter instructions can take fewer cycles	• No additional benefit from shorter instructions
• No overlap	• No overlap	• Overlap instruction execution

5-stage instruction cycle has Instruction fetch, Decode (interpret) instruction, Fetch operands, perform operation, and store results.

Example: Suppose two instructions Load and Store are executed. The following figure explains the instruction execution with single cycle, multiple cycles or pipeline.



Where **Ifetch (Instruction Fetch):** Fetch the instruction from the Instruction Memory
Reg/Dec: Registers Fetch and Instruction Decode
Exec: Calculate the memory address
Mem: Read the data from the Data Memory
Wr: Write the data back to the register file

Pipeline increases the instruction throughput (but not execution time of an individual instruction). An individual instruction can be slower: Additional pipeline control, Imbalance among pipeline stages.

Example: Suppose we execute 100 instructions:

Single Cycle Machine: 45 ns/cycle x 1 CPI x 100 inst = 4500 ns

Multi-cycle Machine: 10 ns/cycle x 4.2 CPI (due to inst mix) x 100 inst = 4200 ns

Ideal 5 stages pipelined machine:10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns

3.3 Differences between Datapaths

	Single Cycle	Multiple Cycle	Pipeline
Instructions Subdivided?	No	Yes, into arbitrary number of steps	Yes, into one step per pipeline stage
Clock Cycle Time	Long (long enough for the slowest instruction)	Short (long enough for the slowest instruction step)	Short (long enough for the slowest pipeline stage)
CPI	1 clock cycle per instruction (by definition)	Variable number of clock cycles per instruction	Fixed number of clock cycles per instruction, one for each pipeline stage (under ideal conditions)
Number of Instructions Executing at the Same Time	1	1	As many instructions as pipeline stages (under ideal conditions)
Control Unit	Generates signals for entire instruction	Generates signals for instruction's current step, and keeps track of the current step	Generates signals for entire instruction; these signals are propagated from one pipeline stage to another via the pipeline registers
Duplicate Hardware?	Yes, since we can use a functional unit (FU) for at most one subtask per instruction	No, since the instruction generally is broken into single-FU steps	Yes, so that there are no restrictions on which instructions can be in the pipeline simultaneously
Extra Registers?	No	Yes, to hold the results of one step for use in the next step	Yes, to provide the results of one pipeline stage to the next pipeline stage
Performance	Baseline	Slightly slower to moderately faster than single cycle, the latter when the instructions steps are well balanced and a significant fraction of the instructions take less than the maximum number of clock cycles	Moderately faster to significantly faster than single cycle, the latter if the pipeline stages are well balanced and the pipeline handles hazards well

3.4 Pipeline Design and Issue

3.4.1 Pipeline Datapath

Every stage must be completed in one clock cycle to avoid stalls. Values must be latched to ensure correct execution of instructions. The PC multiplexer has moved to the IF stage to prevent two instructions from updating the PC simultaneously (in case of branch instruction).

3.4.2 CPU with 5-stage Pipeline Hardware

Let us consider the following decomposition of the instruction execution into five stages:

1. **Fetch Instruction (IF):** Read the next expected instruction into a buffer.
2. **Instruction Decoding (ID):**
   - Decode Instruction: Determine the opcode and the operand specifiers.
   - Calculate Operand: Calculate the effective address of each source operand.
   - Fetch Operands: Fetch each operand from memory.
3. **Execute Instruction (EX):** Perform the indicated operation.

4. **Memory access/branch completion cycle (MEM):** Access the memory
5. **Write Operand (WO):** Store the result.

Instruction fetch cycle (IF): $IR \leftarrow Mem[PC];$

$PC \leftarrow PC + 4$

Instruction decode/register fetch cycle (ID):

$A \leftarrow Regs[IR_{6...10}];$

$B \leftarrow Regs[IR_{11...15}];$

$Immediate \leftarrow (IR_{16})^{16} \#\# (IR_{16...31})$

Execution / effective address cycle (EX):

Memory ref: $ALU_{output} \leftarrow A + Immediate;$

Reg-Reg ALU: $ALU_{output} \leftarrow A \text{ func } B;$

Reg-Imm ALU: $ALU_{output} \leftarrow A \text{ op } Immediate;$

Branch: $ALU_{output} \leftarrow PC + Immediate;\ Cond * (A \text{ op } 0)$

Memory access / branch completion cycle (MEM):

Memory ref: $LMD \leftarrow Mem[ALU_{output}]$ or $Mem(ALU_{output}) \leftarrow B$

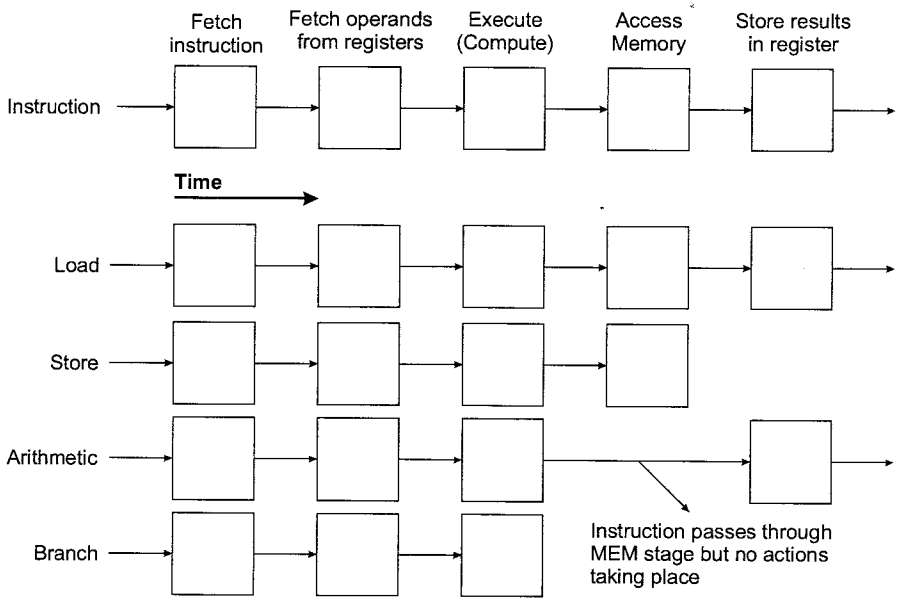
Branch: if (cond) $PC \leftarrow ALU_{output};$

Write-back cycle (WB):

Reg-Reg ALU: $Regs[R_{16...20}] \leftarrow ALU_{output};$

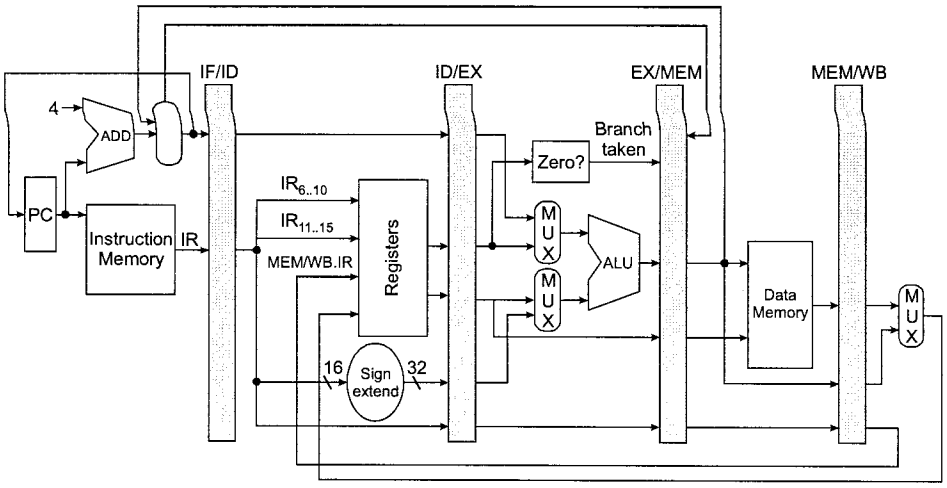
Reg-Imm ALU: $Regs[R_{11...15}] \leftarrow ALU_{output};$

Reg-Reg ALU: $Regs[R_{11...15}] \leftarrow LMD;$



Stages of Instruction Execution

3.4.3 Pipeline Stage Interface



Interface for Pipelining

Stage	Any Instruction		
IF	IF/ID.IR ← MEM [PC] IF/ID.PC ← if (EX/MEM.opcode == branch & EX/mem.cond) {EX/MEM.ALUoutput} else {(PC + 4)};		
ID	ID/EX.A = Regs [IF/ID.$IR_{6...10}$]; ID/EX.B ← Regs [IF/ID.$IR_{11...15}$]; ID/EX.A = Regs [IF/ID.PC; ID/EX.IR ← IF/ID.IR; ID/EX.Immediate ← $(IF/ID.IR_{16})^{16}$ ## IF/ID.$IR_{16...31}$];		
	ALU	**Load or Store**	**Branch**
EX	EXMEM.IR = ID/EX.IR; EXMEM.ALU_{output} ← ID/EX.A func ID/EX.B; **OR** EXMEM.ALU_{output} ← ID/EX.A OP ID/EX.Imm; EX/MEM.cond ← 0;	EX/MEM.IR ← ID/EX.IR; EX/MEM. ALU_{output} ← ID/EX.A + ID/EX.Imm; EX/MEM.cond ← 0; EX/MEM.B ← IDX/EX.B;	EX/MEM. ALU_{output} ← ID/EX.PC + ID/EX.Imm; EX/MEM.cond ← (ID/EX_A OP 0)
MEM	MEM/WB.IR ← EX/MEM.IR; MEM/WB. ALU_{output} ← EXMEM. ALU_{output}	MEM/WB.IR ← EX/MEM.IR; MEM/WB.LMD ← MEM [EX/MEM. ALU_{output}]; **OR** MEM [EX/MEM. ALU_{output}] ← EX/MEM. B;	
WB	Regs [MEM/WB.$IR_{16...20}$] ← EM/WB. ALU_{output}; **OR** Regs [MEM/WB.$IR_{16...20}$] ← MEM/WB. ALU_{output};	For load only: Regs [MEM/WB.WB.$IR_{11...15}$] ← MEM/WB.LMD;	

3.4.4 Performance Analysis with Pipeline

$$\text{Cycle time } \tau = \text{Max}(\tau_i) + d = \tau_m + d;\ 1 \le i \le k$$

where τ_m = Maximum stage delay (delay through stage which experience the largest delay)

k = Number of stages in the instruction pipeline.

d = Time delay of a latch, needed to advance signals and data from one stage to the next.

In general, the time delay d is equivalent to a clock pulse and $\tau_m >> d$. The total time required τ_k to execute all n instructions is

$$T_k = [k + (n - 1)]\tau$$

A total of k cycles are required to complete the execution of the first instruction, and the remaining (n–1) instructions require (n–1) cycles.

The time required to execute n instructions without pipeline is $T_1 = nk\tau$ because to execute one instruction it will take $n\tau$ cycle.

The speed up factor for the instruction pipeline compared to execution without the pipeline is defined as:

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k+(n-1)]\tau} = \frac{nk}{k+(n-1)} = \frac{nk}{(k-1)+n}$$

If the number of instruction executed is much more higher than the number of stages in the pipeline then the n tends to α, $\Rightarrow S_k = k$.

NOTE: The speed up factor is a function of the number of stages (k) in the instruction pipeline.

3.4.5 Problems in Instruction Pipelining

Several difficulties prevent instruction pipelining from being as simple as the above description suggests. The principal problems are:

- **Timing Variations:** Not all stages take the same amount of time. This means that the speed gain of a pipeline will be determined by its slowest stage. This problem is particularly acute in instruction processing, since different instructions have different operand requirements and sometimes vastly different processing time. Moreover, synchronization mechanisms are required to ensure that data is passed from stage to stage only when both stages are ready.
- **Data Hazards:** When several instructions are in partial execution, a problem arises if they reference the same data. We must ensure that a later instruction does not attempt to access data sooner than a preceding instruction, if this will lead to incorrect results. For example, instruction N+1 must not be permitted to fetch an operand that is yet to be stored into by instruction N.
- **Branching:** In order to fetch the "next" instruction, we must know which one is required. If the present instruction is a conditional branch, the next instruction may not be known until the current one is processed.
- **Interrupts:** Interrupts insert unplanned "extra" instructions into the instruction stream. The interrupt must take effect between instructions, that is, when one instruction has completed and the next has not yet begun. With pipelining, the next instruction has usually begun before the current one has completed.

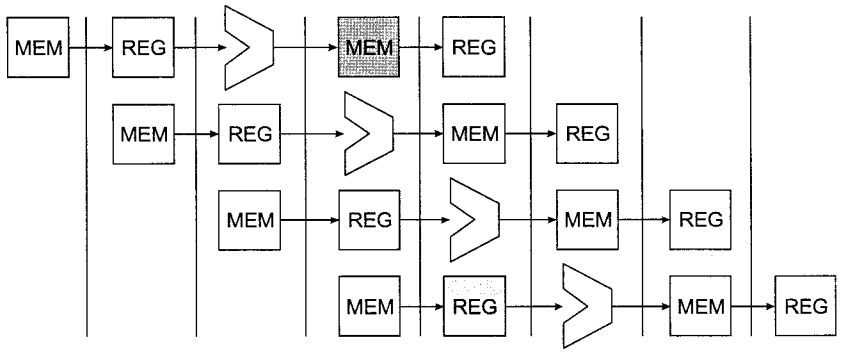
All of these problems must be solved in the context of our need for high speed performance. If we cannot achieve sufficient speed gain, pipelining may not be worth the cost.

3.5 Pipeline Hazards

Any condition that causes the pipeline to stall is called a *harzard*. Mainly there are three types of Hazards: (1) Structural hazards, (2) Data hazards and (3) Control hazards.

3.5.1 Structural Hazard

It occur when different instructions collide while trying to access the same piece of hardware in the same segment of a pipeline (Structural dependency shown in the following Figure).



Example, load and stores use the same memory port as IF. Attempt to use a resource two different ways at same time. Reason is single memory for instruction and data.

Handling Structural Hazards

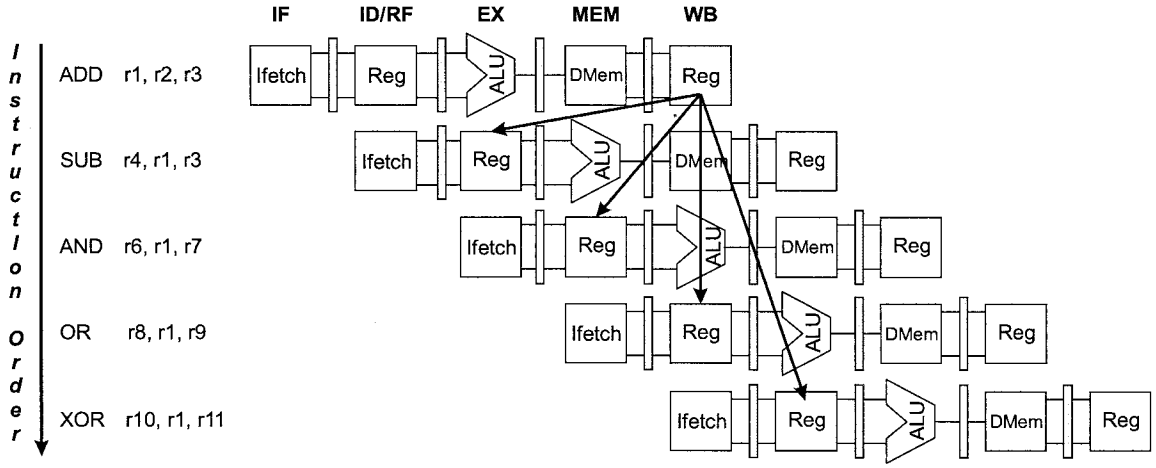
Design a special-purpose memory module: It permits writing (reading) on the first (resp. second) half of the clock cycle. However, this requires special (expensive) hardware.

Use Instruction and Data Cache: Use two fast caches, one for instructions, and one for data, that access a large, slower main memory in which instructions and data are both stored.

3.5.2 Data Hazard

It occur when an instruction depends on the result of a previous instruction still in the pipeline, which result has not yet been computed. When two different instructions use the same storage location It must *appear* as if they executed in sequential order.

- Instruction depends on result of prior instruction still in the pipeline.
- It can occur among the operands in the instruction at the pipeline stages.
- It occur when instructions read or write registers that are used by other instructions.



There are four types of data dependencies: Read After Write, Write After Read, Write After Write, and Read After Read.

Read After Write (RAW / True Data Dependency / Flow Dependency)

It occurs when value produced by an instruction is required by a subsequent instruction.

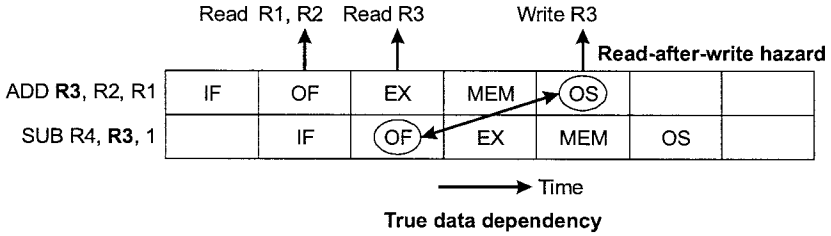
Example-1: Consider the following two instructions.

ADD R3, R2, R1;

SUB R4, R3, 1;

It reads R3 which is written by first instruction (shown in the following figure).

So RAW dependency and also it is Hazard.



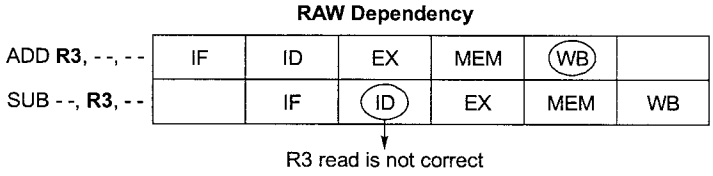
True data dependency

Example-2: Consider the following two instructions.

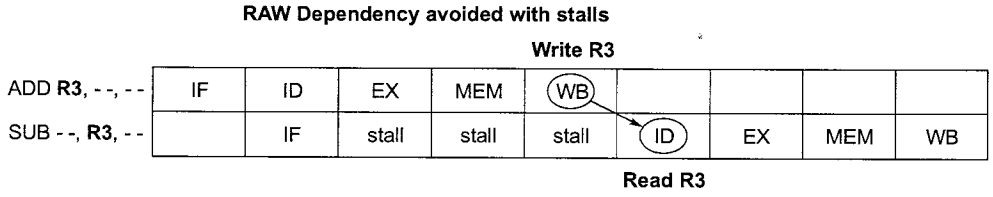
ADD R3, --, --;

SUB --, R3, --;

These two instructions contain RAW dependency as shown in below figure.

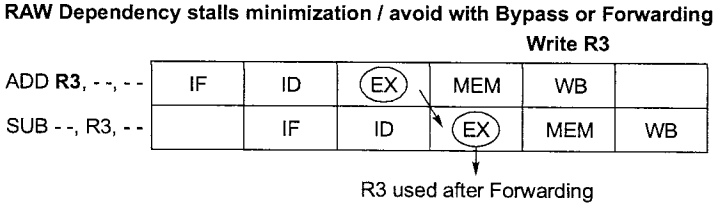


Simple Solution for Example-2: (Use stall cycles)



Stall (Bubble): The instruction at the later stage waits for the front one to complete and thus clock cycle(s) is not utilized during that period Stall. It is same as NOP (no operation) instruction.

Another Solution for Example-2: (Use Bypass/Forward/Short-Circuit)



Write After Read (WAR / Antidependency)

Hazards occur when the output register of an instruction is used for write after read by a previous instruction.

Example: ADD R1, R2, R3;

SUB R2, R5, R6; //It writes data into R2 after reading from the previous instruction.

Write After Write (WAW, Output Dependency)

Hazard occur when the output register of an instruction is used for write after written by a previous instruction

Example: ADD R1, R2, R3;
SUB R1, R5, R6;

NOTE

- If a processor executes instructions in the order that they appear in the program and uses the same pipeline for all instructions, WAR and WAW hazards do not cause the delays because of the way instructions flow through the pipeline.
- For WAR, the register read stage of the pipeline occurs before the write back stage.
- For WAW, the output register of an instruction written in the write back stage of the *pipeline.*

Problems with WAR and WAW dependencies:

- WAW and WAR hazards can cause problems, because it is possible for a low-latency instruction to complete before a longer-latency instruction that appeared earlier in the program. These processors must keep track of name dependencies between instructions and stall the pipeline as necessary to resolve these hazards.
- An issue in out-of-order processor: Processor allows instructions to execute in different orders than the original program to improve performance. Which causes the hazard when those order is changed.

Read After Read (no Hazard)

It occurs when two instructions both read from the same register. Don't cause a problem for the processor because reading a register doesn't change the register's value.

Example: ADD R1, R2, R3;
SUB R4, R5, R3; // This instruction also reads R3. So RAR dependency but no hazard.

3.5.3 Berstein's Conditions

It is a method to detect the possibility of data hazards in the code sequence for some pipelining.

Let $O(i)$ indicate the set of (output) locations altered by instruction i, $I(i)$ indicate the set of (input) locations read by instruction i, and ϕ indicates an empty set.

A potential hazard exists between instruction i and a subsequent instruction j when at least one of the following conditions fails:

For RAW : $O(i)) \cap I(j) = \phi$
For WAR : $I(i)) \cap O(j) = \phi$
For WAW : $O(i)) \cap O(j) = \phi$

Example - 3.1 For the following code sequence check the existence of Data Hazards.

1. ADD R3, R2, R1 ; R3 = R2 + R1
2. SUB R5, R1, 1 ; R5 = R1 – 1

Solution:

$O(1) = \{R3\}, O(2) = \{R5\}, I(1) = \{R1, R2\}$ and $I(2) = \{R1\}$

The conditions:
$\{R3\} \cap \{R1\} = \phi$ (RAW)
$\{R2, R1\} \cap \{R5\} = \phi$ (WAR)
$\{R3\} \cap \{R5\} = \phi$ (WAW)

All conditions are satisfied hence there are no data hazards in the code.

Handling Data Hazards

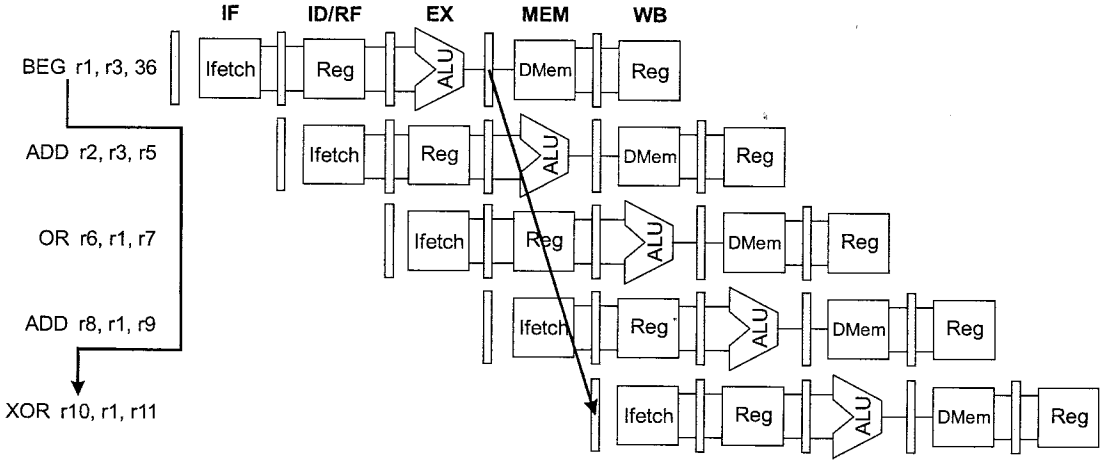
Forwarding: In order to resolve a dependency, one adds special circuitry to the pipeline that is comprised of wires and switches with which one forwards or transmits the desired value to the pipeline segment that needs that value for computation. Although this adds hardware and control circuitry, the method works because it takes far less time for the required value(s) to travel through a wire than it does for a pipeline segment to compute its result.

Code Re-Ordering: The compiler reorders statements in the source code, or the assembler reorders object code, to place one or more statements between the current instruction and the instruction in which the required operand was computed as a result. This requires an "intelligent" compiler or assembler, which must have detailed information about the structure and timing of the pipeline on which the data hazard would occur. We call this type of software a hardware-dependent compiler.

Stall Insertion: It is possible to insert one or more stalls (no-op instructions) into the pipeline, which delays the execution of the current instruction until the required operand is written to the register file. This decreases pipeline efficiency and throughput, which is contrary to the goals of pipeline processor design. Stalls are an expedient method of last resort that can be used when compiler action or forwarding fails or might not be supported in hardware or software design.

3.5.4 Control Hazard

It can result from branch instructions. The branch target address might not be ready in time for the branch to be taken, which results in *stalls* (dead segments) in the pipeline that have to be inserted as local wait events, until processing can resume after the branch target is executed. Control hazards are the most difficult types of hazards arising from normal operation of a program.



Types of Control Hazards

1. Jump to the branch target address if the instruction succeeds, or
2. Execute the instruction after the branch (at PC+4 of instruction memory) if the branch fails.

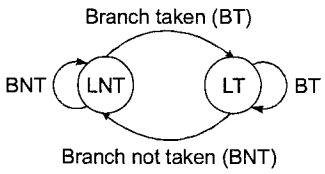
The problem with the branch instruction is that we usually do not know which result will occur (i.e., whether or not the branch will be taken) until the branch condition is computed. Often, the branch condition depends on the result of the preceding instruction, so we cannot pre-compute the branch condition to find out whether or not the branch will be taken. The following four strategies are employed in resolving control dependencies due to branch instructions.

1. **Assume branch not taken:** We can insert stalls until we find out whether or not the branch is taken. However, this slows pipeline execution unacceptably.

   A common alternative to stalling is to continue execution of the instruction stream as though the branch was not taken. The intervening instructions between the branch and its target are then executed.

   If the branch is not taken, this is not a harmful or disruptive technique. However, if the branch is taken, then we must discard the results of the instructions executed after the branch statement. This is done by flushing the IF, ID, and EX stages of the pipeline for the discarded instructions. Execution continues uninterrupted after the branch target.

   The cost of this technique is approximately equal to the cost of discarding instructions. For example, if branches are not taken 50 percent of the time, and the cost of discarding results is negligible, then this technique reduces by 50 percent the cost of control hazards.



2. **Reducing branch delay:** In the pipeline architecture, we currently assume that the branch condition is evaluated in Stage 3 of the pipeline (EX).

   If we move the branch evaluation up one stage, and put special circuitry in the ID (Decode, Stage #2), then we can evaluate the branch condition for the BEQ instruction. This would allow us to take the branch in EX instead of MEM, since we would have ready for Stage 3 (EX) the *Zero* output of the comparator that would normally be computed in EX.

   The hardware needed to compute equality is merely a series of parallel *XNOR* gates *AND-ed* together, then inverted.

   The advantage of this technique is that only one instruction needs to be discarded rather than two, as in the previous section. This reduces hardware cost and time required to flush the pipeline, since only the IF and ID stages would need to be flushed, instead of the three stages.

3. **Dynamic branch prediction:** It would be useful to be able to predict whether or not a majority of the branches are taken or not taken. This can be done in software, using intelligent compilers, and can also be done at runtime. We concentrate on the software-intensive techniques first, since they are less expensive to implement (being closer to the compiler, which is easier to modify than the hardware).

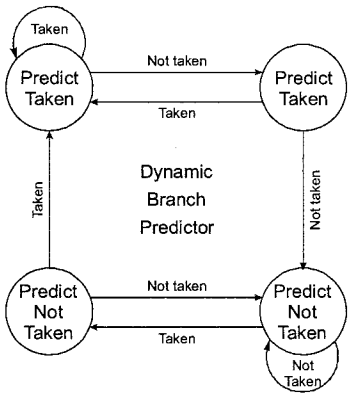
   The most advantageous situation is one where the branch condition does not depend on instructions immediately preceding it, as shown in the following code fragment:

   - ADD $5, $5, $6 # One of the registers for BEQ comparison is modified
   - SUB $4, $3, $6 # Nothing important to the branch here
   - AND $7, $8, $6 # Nothing important to the branch here
   - AND $9, $6, $6 # Nothing important to the branch here
   - BEQ $5, $6, target

   Here, the branch compares Registers 5 and 6, which are last modified in the add instruction. We can therefore precompute the branch condition as sub r $5, $6, where *r* denotes a destination register. If r = 0, then we know the branch will be taken, and the runtime module (pipeline loader) can schedule the jump to the branch target address with full confidence that the branch will be taken.

   Another approach is to keep a history of branch statements, and to record what addresses these statements branch. Since the vast majority of branches are used as tests of loop indices, then we know that the branch will almost always jump to the loopback point.

If the branch fails, then we know the loop is finished, and this happens only once per loop. Since most loops are designed to have many iterations, branch failure occurs less frequently in loops than does taking the branch. The branch prediction decision is always the same every time a given instruction is executed, this is called *static branch prediction.*



The prediction decision may changed depending on execution history is called *dynamic branch prediction.* In dynamic branch prediction schemes, the processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decisions every time that instruction is executed.

The simplest form, the execution history used in predicting the outcome of a given instruction is the result of the most recent execution of that instruction. The processor assumes that the next time the instruction is executed, the result is likely to be the same.

Hence, the algorithm may be described by a two state machine (as shown in above figure). The two states are LT: Branch is likely to be taken, and LNT: Branch is likely not to be taken.

3.6 Pipeline Performance Analysis

3.6.1 CPI of a Pipeline Processor

Suppose an N-segment pipeline processes M instructions without stalls or penalties. We know that it takes (N – 1) cycles to load (setup) the pipeline, and M cycles to complete the instructions.

- The number of cycles (N_{cyc}) = N + M – 1.
- The cycles per instruction (CPI) = N_{cyc} / M = 1 + (N – 1)/M.

3.6.2 Effect of Stalls on CPI

Suppose that we have a N-segment pipeline processing M instructions, and we must insert K stalls to resolve data dependencies. The pipeline has setup penalty of N – 1 cycles, as before, a stall penalty of K cycles, and a processing cost (as before) of M cycles to process the M instructions.

- The number of cycles (N_{cyc}) = N + M + K – 1.
- The cycles per instruction (CPI) = N_{cyc} / M = 1 + (N + K – 1)/M.

NOTE

Suppose that an N-segment pipeline executes M instructions, and that a fraction f_{stall} of the instructions require the insertion of K stalls per instruction to resolve data dependencies.

- The total number of stalls = $f_{stall} \times M \times K$.
- The number of cycles (N_{cyc}) = N + M + ($f_{stall} \times M \times K$) – 1.
- The cycles per instruction (CPI) = N_{cyc} / M = 1 + ($f_{stall} \times K$) + (N – 1) / M.

So, the CPI penalty due to the combined effects of setup cost and stalls now increases to fK + (N – 1) / M.

3.6.3 Effect of Exceptions

Assume that we have M instructions executing on an N-segment pipeline with no stalls, but that a fraction f_{ex} of the instructions raise an exception in the EX stage. Further assume that each exception requires the following:

(a) The pipeline segments before the EX stage be flushed,
(b) That the exception be handled, requiring an average of H cycles per exception, then
(c) The instruction causing the exception and its following instructions be reloaded into the pipeline.

Therefore $f_{ex} \times M$ instructions will cause exceptions.

In the pipeline, each of these instructions causes three instructions to be flushed out of the pipe (IF, ID, and EX stages), which incurs a penalty of four cycles (one cycle to flush, and three to reload) plus H cycles to handle the exception. Therefore pipeline performance equations become:

$$\begin{aligned} N_{cyc} &= N - 1 + (1 - f_{ex}) \times M + (f_{ex} \times M \times (H + 4)) \\ &= M + [N - 1 - M + (1 - f_{ex}) \times M + (f_{ex} \times M \times (H + 4))] \\ CPI &= N_{cyc} / M = 1 + [1 - f_{ex} + (f_{ex} \times (H + 4)) - 1 + (N - 1)/M] \\ &= 1 + [(f_{ex} \times (H + 3)) + (N - 1) / M] \end{aligned}$$

3.6.4 Effect of Branches

Branches present a more complex picture in pipeline performance analysis. There are three ways of dealing with a branch:

1. Assume the branch is not taken, and if the branch is taken, flush the instructions in the pipe after the branch, then insert the instruction pointed to by the BTA;
2. Converse of above statement and
3. Use a delayed branch with a branch delay slot and re-ordering of code (assuming that this can be done).

The first two cases are symmetric. Assume that an error in branch prediction (i.e., taking the branch when you expected not to, and conversely) requires L instruction to be flushed from the pipeline (one cycle for flushing plus L-1 "dead" cycles, since the branch target can be inserted in the IF stage). Thus, the cost of each branch prediction error is L cycles. Further assume that a fraction f_{br} of the instructions are branches and f_{be} of these instructions result in branch prediction errors. The penalty in cycles for branch prediction errors is:

$$branch\_penalty = f_{br} \times f_{be} \times M \text{ instructions} \times L \text{ cycles per instruction.}$$

The pipeline performance equations become:

$$\begin{aligned} N_{cyc} &= N - 1 + (1 - f_{br} \times f_{be}) \times M + (f_{br} \times f_{be} \times M \times L), \\ &= M + [N - 1 - M + (1 - f_{br} \times f_{be}) \times M + (f_{br} \times f_{be} \times M \times L), \\ CPI &= N_{cyc} / M = 1 + [(1 - f_{br} \times f_{be}) + (f_{br} \times f_{be} \times L) - 1 + (N - 1)/M] \\ &= N_{cyc} / M = 1 + [(f_{br} \times f_{be} \times (L - 1)) + (N - 1)/M] \end{aligned}$$

In the case of the branch delay slot, we assume that the branch target address is computed and the branch condition is evaluated at the ID stage. Thus, if the branch prediction is correct, there is no penalty.

Depending on the method by which the pipeline evaluates the branch and fetches (or pre-fetches) the branch target, a maximum of two cycles penalty (one cycle for flushing, one cycle for fetching and inserting the branch target) is incurred for insertion of a stall in the case of a branch prediction error. The pipeline performance equations become:

$$\begin{aligned} N_{cyc} &= N - 1 + (1 - f_{br} \times f_{be}) \times M + (f_{br} \times f_{be} \times 2M), \\ CPI &= N_{cyc} / M = 1 + [f_{br} \times f_{be} + (N - 1)/M]. \end{aligned}$$

Since $f_{br} << 1$ is usual, and f_{be} is, on average, assumed to be no worse than 0.5, the product $f_{br} \times f_{be}$, which represents the additional branch penalty for CPI in the presence of delayed branch and BDS, is generally small.

Remember

Performance effect shown with important techniques in the following Table

Technique	Reduces
Dynamic scheduling	Data hazard stalls
Dynamic branch prediction	Control stalls
Issuing multiple instructions per cycle	Ideal CPI
Speculation	Data and control stalls
Dynamic memory disambiguation	Data hazard stalls involving memory
Loop unrolling	Control hazard stalls
Basic compiler pipeline scheduling	Data hazard stalls
Compiler dependence analysis	Ideal CPI and data hazard stalls
Software pipelining and trace scheduling	Ideal CPI and data hazard stalls
Compiler speculation	Ideal CPI, data and control stalls

3.7 Speedup

- Ideal $CPI_{pipeline} = 1$
- $CPI_{pipeline}$ = Ideal $CPI_{pipeline}$ + Pipelined stall cycles per instruction
  = 1 + Pipelined stall cycles per instruction
- $$\text{Speedup} = \frac{\text{Average\_instruction\_time}_{unpipelined}}{\text{Average\_instruction\_time}_{pipelined}}$$

  $$= \frac{CPI_{unpipelined}}{CPI_{pipelined}} \times \frac{\text{Clock\_cycle}_{unpipelined}}{\text{Clock\_cycle}_{pipelined}}$$

  $$= \frac{CPI_{unpipelined}}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock\_cycle}_{unpipelined}}{\text{Clock\_cycle}_{pipelined}}$$

  $$= \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$
- Computation of speedup with branch penalty:

  $$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} = \frac{\text{Pipeline depth}}{1 + \text{Branch Frequency} \times \text{Branch penalty}}$$
- If **"A is n times faster than B"**:

  $$n = \frac{\text{Execution\_time}_B}{\text{Execution\_time}_A} = \frac{1/\text{Performance}_B}{1/\text{Performance}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$
- If **"A is n% faster than B"**:

  $$n = \left(\frac{\text{Performance}_A - \text{Performance}_B}{\text{Execution}_B}\right) \times 100 = \left(\frac{\text{Execution\_Time}_B - \text{Execution\_Time}_A}{\text{Execution\_Time}_A}\right) \times 100$$

Example-3.2 If machine A runs a program in 10 seconds and machine B runs the same program in 15 seconds, which of the following statements is true?

(a) A is 50% faster than B (b) A is 33% faster than B

Solution:

Machine A is n% faster than machine B can be expressed as:

$$\frac{Execution\_Time_B}{Execution\_Time_A} = 1+\frac{n}{100}$$

$$\Rightarrow \quad n = \left\{\frac{Execution\_Time_B - Execution\_Time_A}{Execution\_Time_A}\right\}\times 100 = \frac{15-10}{10}\times 100 = 50$$

Therefore A is 50% faster than B.

Example-3.3 *CPU clock frequency is 1 MHz (clock cycle time is 10^{-6} sec.) Program takes* 4.5 million cycles to execute. What is the CPU time?

Solution:

CPU time = 4,500,000 × 10^{-6} = 4.5 seconds

Example-3.4 CPU clock frequency is 500 MHz (clock cycle time is 10^{-6} sec.) Program takes 45 million cycles to execute. What is the CPU time?

Solution:

CPU time = 4,500,000 × (1/500,000,000) = 0.09 seconds

Example-3.5 A benchmark has 100 instructions with the following information:

(a) 25 instructions are loads/stores (each takes 2 cycles)

(b) 50 instructions are adds (each takes 1 cycles)

(c) 25 instructions are square root (each takes 100 cycles)

What is the CPI?

Solution:

CPI = (25 × 2) + (50 × 1) + (25 × 100) / 100 = 2600/100 = 26.0

Example-3.6 Calculating average CPI for the following table.

OP	Freq	CPI_i
ALU	50%	1
Load	20%	2
Store	10%	2
Branch	20%	2
	100%	

Solution:

OP	Freq	CPI_i	$CPI_i \times F_i$	(% time)
ALU	50%	1	0.5	(33%)
Load	20%	2	0.4	(27%)
Store	10%	2	0.2	(13%)
Branch	20%	2	0.4	(27%)
	100%		1.5	

Remember

Amdahl's Law: The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's Law.

- Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.
- Amdahl's Law defines the speedup that can be gained by using a particular feature.
- Suppose that we can make an enhancement to a machine that will improve performance when it is used.
- **Speedup:**

$$\text{Speedup} = \frac{\text{Performance\_for\_entire\_task\_using\_the\_enhancement\_when\_possible}}{\text{Performance\_for\_entire\_task\_without\_using\_the\_enhancement}}$$

$$= \frac{\text{Performance}_{new}}{\text{Performance}_{old}}$$

$$= \frac{\text{Entire\_task\_Performance\_using\_the\_enhancement\_when\_possible}}{\text{Entire\_task\_Performance\_task\_without\_using\_the\_enhancement}}$$

$$= \frac{\text{Execution Time}_{old}}{\text{Execution Time}_{new}}$$

***Example*-3.7** Program runs for 100 seconds on a uniprocessor. 10% of the program can be parallelized on a multiprocessor (F = 0.1). Assume a multiprocessor with 10 processors (n = 10). Compute the speedup.

Solution:

$$\text{Speedup} = \frac{1}{(1-0.1)+\frac{0.1}{10}} = \frac{1}{0.01+0.9} = \frac{1}{0.91} = 1.1$$

***Example*-3.8** Floating point instructions improved to run 2X; but only 10% of actual instructions are FP. Assume a multiprocessor with 10 processors (n = 10). Compute overall speedup.

Solution:

$$\text{Execution\_Time}_{new} = \text{Execution\_Time}_{old} \times \left(0.9 + \frac{0.1}{2}\right) = 0.95 \times \text{Execution\_Time}_{old}$$

$$\text{Speedup}_{overall} = \frac{1}{0.95} = 1.053$$

***Example*-3.9** Suppose we could improve the speed of the CPU in our machine by a factor of five (without affecting I/O performance) for five times the cost. Also assume that the CPU is used 50% of the time, and the rest of the time the CPU is waiting for I/O. If the CPU speed by a factor of five a good investment from the cost/performance viewpoint?

Solution:

$$\text{Speedup} = \frac{1}{0.5+\frac{0.5}{5}} = \frac{1}{0.6} = 1.67$$

The new machine will cost 2.33 times the original machine: $\frac{2}{3}\times 1+\frac{1}{3}\times 5 = 2.33$. Since the cost increase is larger than the performance improvement, this change does not improve cost/performance.

Example-3.10 Given the following information, what is the clock cycle time (in ns) of the machine and how many nanoseconds does it take for an instruction to complete?

Number of pipeline stages = 8
Critical path = 15

Solution:

Critical path is the longest path through a pipeline stage.
For a pipelined machine, the critical path defines the cycle time.
Therefore, the clock cycle time is 15ns.
One instruction takes 8 stages × 15 ns = 120 ns.

Example-3.11 Using the machine specified from the previous problem. (a) What is the minimum (best) CPI (assume a MIPS 5-state pipeline)? (b) What is the machine's CPI if 20% of all instructions are loads and 5% of the instructions following a load depends on the result of the load (assume all other instructions have no dependencies)?

Solution:

The best CPI is 1.0

$$20\% \times 5 = 1\%$$

1% of the instructions stall for one cycle.

$$\text{CPI} = 99\% \times 1.0 \text{ cycles} + 1\% \times 2.0 \text{ cycles}$$
$$= 1.01 \text{ cycles per instruction.}$$

Example-3.12 An instruction requires four stages to execute: state 1 (instruction fetch) requires 30 ns, stage 2 (instruction decode) = 9 ns, stage 3 (instruction execute) = 20 ns and stage 4 (store results) = 10 ns. An instruction must proceed through the stages in sequence. What is the minimum asynchronous time for any single instruction to complete:

Solution:

Asynchronous time = IF + ID + IE + SR = 30 + 9 + 20 + 10 = 69 ns

Example-3.13 For the above question, how many stages should we have and at what rate should we clock the pipeline?

Solution:

We have 4 natural stages given and no information on how we might be able to further subdivide them, so we use 4 stages in our pipeline. We have a choice of what clock rate to use. The simplest choice would be to use a clock cycle that accommodates the longest stage in our pipe – 30 ns. This would allow use to initiate a new instruction every 30 ns with a latency through the pipe of 30 ns × 4 stages = 120 ns. We could also pick a finer clock cycle that more closely matches the shortest stage (9 ns) but is integrally divisible into the other stages. A clock of 10 ns would be a good match and would require three clocks for the first stage, 1 clock for the second, 2 clocks for the third and 1 clock for the fourth. This would allow us to initiate a new instruction every 30 ns but provide a latency of 70 ns rather than 120. Either 30 ns or 10 ns is acceptable.

Example-3.14 What is the speedup of the pipeline in question 2?

Solution:

Speedup per Stone's preferred definition is (30 + 9 + 20 + 10)/30 = 2.3

Speedup per best clocked definition is (30 + 10 + 20 + 10)/30 = 2.33

Example-3.15 Suppose the branch frequencies follow:

Type of Branch	Frequency
Conditional branches	15%
Jumps and Calls	1%
Conditional branches	60% are taken

- Branches are resolved at end of second cycle for unconditional branches.
- And at end of third cycle for conditional branches.

How much faster would be the machine without any branch hazard?

Solution:

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stalls}}$$

Jumps stalls:

- See next figure. It shows stalls caused from this type of jump.
- In CC2, pipeline fetches instruction $i + 1$. By end of CC2 branch is resolved and so at CC3 the new instruction will be fetched.
- So we have 1 CC stall.

	Clock cycle					
Instruction	1	2	3	4	5	6
Jump	IF	ID	EX	WB		
$i + 1$		IF	IF	ID	EX	...
$i + 2$			stall	IF	ID	...
$i + 3$				stall	IF	...

Taken conditional branch:

- See next figure.
- Delayed for 2 CC stalls. Branch resolves in CC3. So in CC4 pipeline will fetch the target instruction.
- So we have 2 CC stalls.

	Clock cycle					
Instruction	1	2	3	4	5	6
Taken branch	IF	ID	EX	WB		
$i + 1$		IF	stall	IF	ID	...
$i + 2$			stall	stall	IF	...
$i + 3$				stall	stall	...

Figure: *Effects of a taken conditional branch on the pipeline*

Non-taken conditional branch:

- See next figure.
- In CC2 fetch cycle issues.
- In CC3 branch resolves so decision comes to finish the same instruction by issuing ID cycle.
- So we have 1 CC stall.

	Clock cycle					
Instruction	1	2	3	4	5	6
Non-taken branch	IF	ID	EX	WB		
$i + 1$		IF	stall	ID	EX	...
$i + 2$			stall	IF	ID	...
$i + 3$				stall	IF	...

Figure : *Effects of a not-taken conditional branch on the pipeline*

Pipeline stall computation:

Next table will summarize previous calculations.

Control flow type	Frequency (per instruction)	Stalls (cycles)
Jumps and calls	1%	1
Conditional (taken)	15% × 60% = 9%	2
Conditional (not taken)	15% × 40 = 6%	1

Figure : *A symmetry of the behaviour of control flow instruction*

$$\text{Pipeline stall}_{real} = (1 \times 1\%) + (2 \times 9\%) + (1 \times 6\%) = 0.24$$

$$\text{Pipeline speed up}_{real} = \frac{4}{(1+0.24)} = 3.23$$

$$\text{Pipeline speedup} = \frac{4}{3.23} = 1.24$$

Summary

- **Disadvantages of the Single Cycle Processor:** Long cycle time (Cycle time is too long for all instructions except the Load).
- **Multiple Clock Cycle Processor:** Divide the instructions into smaller steps, Execute each step (instead of the entire instruction) in one cycle.
- **Pipeline Processor:** It is enhancement of the multiple clock cycle processor. Each functional unit can only be used once per instruction.
- Pipelining doesn't help latency of single task, it helps throughput of entire workload. It Improves instruction throughput rather instruction latency.
- Pipeline rate limited by slowest pipeline stage.
- Multiple tasks operating simultaneously using different resources.
- Potential speedup is equal to Number pipe stages.
- Unbalanced lengths of pipe stages reduces speedup.
- **Structural hazards:** Different instructions in different stages (or the same stage) conflicting for the same resource.
- **Data hazards:** An instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction.
- **Control hazard:** Fetch cannot continue because it does not know the outcome of an earlier branch- special case of a data hazard - separate category because they are treated in different ways.

Student's Assignment

Q.1 Pipelining improves CPU performance due to
(a) reduced memory access time
(b) increased clock speed
(c) the introduction of parallelism
(d) additional functional units

Q.2 A 5 stage pipeline with the stages taking 1, 1, 3, 1, 1, units of time has a throughput of
(a) 1/3 (b) 1/7
(c) 7 (d) 3

Q.3 Given a 5 stage pipeline with stages taking 1, 2, 3, 1, 1 units of time, the clock period of the pipeline is
(a) 8 (b) 1/8
(c) 1/3 (d) 3

Q.4 Which of following registers processor used for fetch and execute operations?
1. Program counter 2. Instruction register
3. Address register
(a) 1 and 3 (b) 1 and 2
(c) 2 and 3 (d) 1, 2 and 3

Q.5 Which of the following statements is false with regard to instruction pipelining?
(a) The basic goal of instruction pipelining is to achieve a CPI of 1.
(b) Instruction set architectures having simple registry/register addressing modes can be easily pipelined than those having complex addressing modes.
(c) By the use of hardware special features and compiler design techniques, pipeline hazards can be removed.
(d) The basic goal of instruction pipelining is to achieve a CPI of more than 1.

Common Data for Q.6 & Q.7

Consider an unpipelined processor assume that it has a 1 ns clock cycles and that it uses 4 cycles for ALU operation and branches and 5 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20% and 40% respectively. Suppose due to clock skew and setup, pipelining the

processor adds 0.2 ns of overhead to the clock. Ignore any latency impact.

Q.6 What is the average instruction execution time for unpipelined processor?
(a) 3.4 ns (b) 4.4 ns
(c) 3.1 ns (d) 1.2 ns

Q.7 What speedup gain after pipelined the processor?
(a) 3.4 ns (b) 3.8 ns
(c) 3.7 ns (d) 1.2 ns

Linked Data For Q.8 & Q.9

Q.8 Consider a case where (k) 4-segment pipeline with a clock cycle time (tp) 20 ns in each sub operation to execute (n) 100 tasks. Assume that a non pipeline unit that can perform the same operation.

Pipelined system will take how much time to complete the task?
(a) 2060 ns (b) 2000 ns
(c) 20 ns (d) 1901 ns

Q.9 Find out the speed-up ratio between pipelined and non-pipelined system?
(a) 3 (b) 4
(c) 3.88 (d) 400

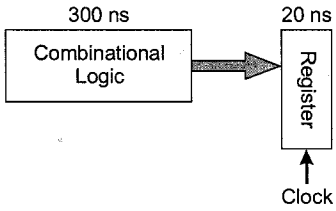
Q.10 Pipelined operation is interrupted whenever two operations being attempted in parallel need same hardware component. Such a conflict can occur if _____.
(a) The execution phase of an instruction requires access to the main memory, which also contains the next instruction that should be fetched at the same time
(b) The prefetch phase of an instruction requires access to the main memory, which also contains the next instruction that should be fetched at the same time
(c) The decode phase of an instruction requires access to the main memory, which also contains the next instruction that should be fetched at the same time
(d) None of these

Q.11 Consider the unpipelined machine with 10 ns clock cycles. It uses four cycles for ALU operations and branches, whereas five cycles for memory operations. Assume that the relative frequencies of there operations are 40%, 20%, 40% respectively.
Suppose that due to clock skew and setup, pipelining the machine adds 1 ns overhead to the clock. How much speed up in the instruction execution rate will we gain from a pipeline?
(a) 5 times (b) 8 times
(c) 4 times (d) 4.5 times

Common Data for Q.12 & Q.13

Using a sequential implementation, it takes a total of 320 ns for each instruction, 300 ns for the combinational logic to complete, and 20 ns to store the result (in a register).

This means that a throughput will be about 3.12 millions instructions/second. Assuming you switch to a 3 stage pipeline by splitting the combinational logic into three equal parts and all registers take 20 ns to store results.



Q.12 How long will it take for a single instruction to execute in the pipelined implementation?
(a) 300 ns (b) 360 ns
(c) 100 ns (d) 380 ns

Q.13 By assuming the pipeline never starts, what will the improvement in throughput be?
(a) 2 Instruction/300 ns
(b) 2 Instruction/360 ns
(c) 3 Instruction/360 ns
(d) 3 Instruction/300 ns

Q.14 **Assertion (A):** Reduced Instruction Set Computers (RISC) use pipelined control unit.
Reason (R): Pipelining reduces memory requirements of programs.

(a) Both A and R are true and R is the correct explanation of A
(b) Both A and R are true but R is NOT the correct explanation of A
(c) A is true but R is false
(d) A is false but R is true

Q.15 Match **List-I** with **List-II** and select the correct answer using the codes given below the lists:

List-I

A. Pipelined ALU
B. Simpler compiler
C. Separate data and instruction caches
D. Lesser cycles per instruction

List-II

1. RISC
2. CISC
3. Mixed
4. Mixed RISC-CISC

Codes:

	A	B	C	D
(a)	3	2	3	1
(b)	1	2	3	3
(c)	3	3	2	1
(d)	3	3	3	1

Q.16 Consider the unpipelined machine with 12 ns clock cycles. It uses four cycles for ALU operations and branches, whereas five cycles for memory operations. Assume that the relative frequencies of there operations are 30%, 20%, 20% respectively.

Suppose that due to clock skew and setup, pipelining the machine adds 1 ns overhead to the clock. How much speed up in the instruction execution rate will we gain from a pipeline?

(a) 2.0 times (b) 3.1 times
(c) 2.8 times (d) 3.5 times

Answer Key:

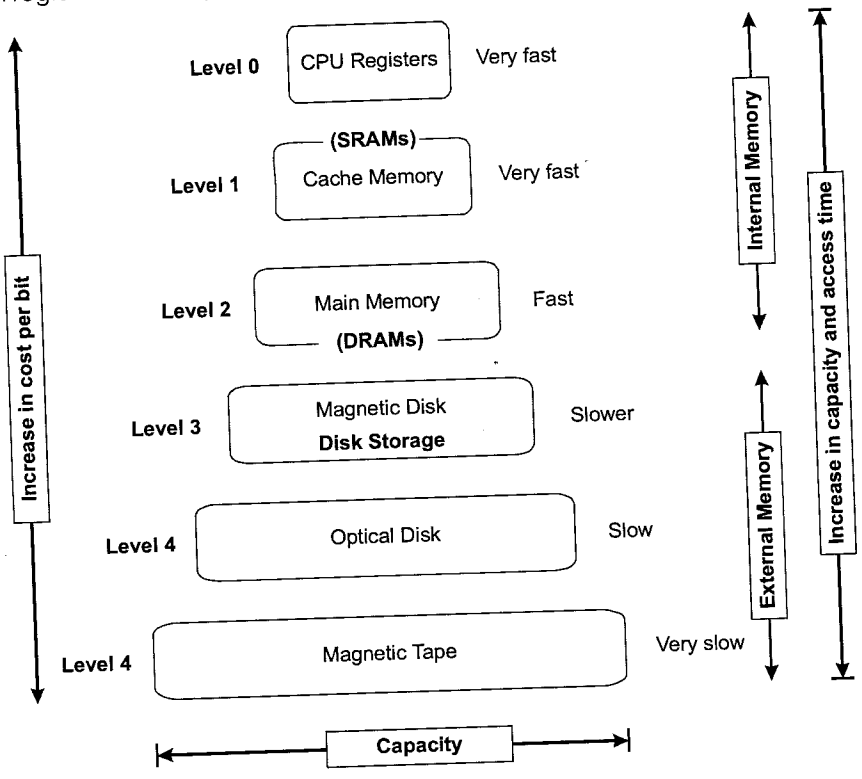
1. (c)	**2.** (a)	**3.** (d)	**4.** (b)	**5.** (d)
6. (b)	**7.** (c)	**8.** (a)	**9.** (c)	**10.** (a)
11. (c)	**12.** (b)	**13.** (c)	**14.** (a)	**15.** (a)
16. (c)				

CHAPTER 04

Memory Hierarchy Design

4.1 Introduction

The memory hierarchy was developed based on a program behavior known as locality of references. Memory references are generated by the CPU for either instruction or data access. These accesses tend to be clustered in certain regions in time, space, and ordering.



4.1.1 Types of Memory based on Access

1. **Serial Access Memory:** The system must search the storage device from the beginning of the memory address until it finds the required piece of data. Memory device which supports such access is called a Sequential Access Memory or Serial Access Memory.
   Example: Magnetic tape.
2. **Direct Access Memory:** Direct access memory or Random Access Memory, refers to condition in which a system can go directly to the information that the user wants. Memory device which supports such access is called a Direct Access Memory.
   Example: Magnetic disk and optical disks.

4.1.2 Memory Access Methods

1. ***Sequential Access:*** *In this method, the memory is accessed in a specific linear sequential manner.* For example, if fourth record (collection of data) stored in a sequential access memory needs to be accessed, the first three records must be skipped. Thus, the access time in this type of memory depends on the location of the data. Magnetic disks, magnetic tapes and optical memories the CD-ROM use this method.
2. **Random Access:** In this mode of access, any location of the memory can be accessed randomly. In other words, the access to any location is not related with its physical location and is independent of other locations. For random access, a separate mechanism is therefore each location. Semiconductor memories (RAM, ROM) are this type.
3. **Direct Access:** This method is basically the combination of previous two methods. Memory devices such as magnetic hard disks contain many rotating storage tracks. If each track has its own read/write head, the tracks can be accessed randomly, but access within each track is sequential. In this case the access is semi-random or direct. The access time depends on both the memory organization and the characteristic of storage technology.
4. **Associative Access:** This is a special type of random access method that enables one to make a comparison of desired bit locations within a word for a specific match and to do this for all words simultaneously. Thus, based on a portion of a word's content, word is retrieved rather than its address. Cache memory uses this type of access mode.

4.2 Primary Memory

The memory stores the instructions and data for an executing program. Memory is characterized by the smallest addressable unit as one of the following.

- **Byte addressable:** Smallest unit is an 8-bit byte.
- **Word addressable:** Smallest unit is a word, usually 16 or 32 bits in length.

Most modern computers are byte addressable, facilitating access to character data. Logically, computer memory should be considered as an array. The index into this array is called the **address** or **"memory address"**.

There are two types of primary memories: RAM and ROM

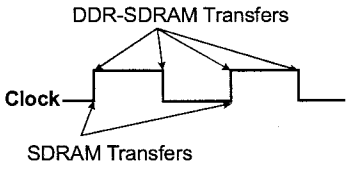
4.2.1 Random Access Memory (RAM)

RAM is Read/write memory, Random access, and data is temporarily stored. RAM is further classified into two types:

1. **DRAM (Dynamic Random Access Memory):** It tends to lose its contents, even when powered. Special "refresh circuitry" must be provided.

SDRAM (Synchronous DRAM): It is DRAM that is designed to work with a **Synchronous Bus**, one with a clock signal. The memory bus clock is driven by the CPU system clock, but it is always slower.

In **SDRAM**, the memory transfers take place on a timing dictated by the memory bus clock rate. This memory bus clock is always based on the system clock. In "plain" SDRAM, the transfers all take place on the rising edge of the memory bus clock. In **DDR SDRAM** (Double Data Rate Synchronous DRAM), the transfers take place on both the rising and falling clock edges.



"Plain" SDRAM makes a transfer every cycle of the memory bus. DDR–SDRAM makes two transfers for every cycle of the memory bus, one on the rising edge of the clock cycle and another one on the falling edge of the clock cycle.

2. **SRAM (Static Random Access Memory):** It will keep its contents as long as it is powered. Compared to DRAM, SRAM is faster, more expensive, physically larger (fewer memory bits per square millimeter).

Comparison of DRAM and SRAM:

	DRAM	SRAM
Types of device	Analog	Digital
Electricity supply	Need	No need
Speed	Slow	Fast
Construction	Simple	Complicated
Capacity	More	Less
Cost	Cheap	Less
Application	Main memory	Cache memory

4.2.2 Read Only Memory (ROM)

ROM is Read only memory, Random access, and Data is permanently stored. ROM is further classified as following:

1. **MROM (Masked ROM):** The contents of the memory are set at manufacture and cannot be changed without destroying the chip.
2. **PROM (Programmable ROM):** The contents of the chip are set by a special device called a "PROM Programmer". Once programmed the contents are fixed.
3. **EPROM (Erasable and Programmable ROM):** It is same as a PROM, but that the contents can be erased using UV light and reprogrammed by the PROM Programmer.
4. **EEPROM (Electrically EPROM):** The contents can be erased electrically and reprogrammed by the PROM Programmer.

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-Access Memory (RAM)	Read-Write Memory	Electrically, byte-level	Electrically	Volatile
Read-Only Memory (ROM)	Read-Only Memory	Not possible	Masks	Non-volatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-Mostly Memory	UV light, Chip-level		
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

4.2.3 RAM Vs ROM

	RAM	ROM
Accessibility	The information stored in the RAM is easily accessed because it communicates directly with the processor	The processor cannot directly access the information. Hence, the information will be transferred into the RAM and then it gets executed by the processor to access the ROM information.
Volatility	Volatile in nature, Data is stored as long as the power supply is switched on. Data will erased if the computer crashes or is turned off.	Non-volatile in nature. Data is stored even the power supply is switched off. Data is retain even if the computer crashes or is turned off.
Storage	Data is temporary It is only there as long as the computer is on and it can be changed	Data is permanent It can never be changed Contents are remain same
Speed	The accessing speed of RAM is faster, it assist the processor to boost up the speed	Speed is slower compared to RAM, ROM cannot boost up the processor speed
Data Preserving	Electricity supply is needed in RAM to flow to preserving information	Electricity supply is not needed in ROM to flow to preserving information
Structure	The RAM is an chip, which is in the rectangle form and is inserted over the mother board of the computer	ROMs are generally the optical drivers, which are made of magnetic tapes.
Cost	The price of RAMs are comparatively high	The price of ROM's are comparatively low
Chip size	Physically size of RAM chip is larger than ROM chip	Physically size of ROM chip is smaller than RAM chip.
Category	Read-write memory Data can be written to or read from.	Read-only memory Data can only be read User cannot make any changes to the information

4.2.4 Memory Registers

There are two types of memory registers:

- **MAR (Memory Address Register):** It specifies the memory address of instruction or data item.
- **MBR (Memory Buffer Register):** This holds the data read from memory or to be written to memory. MAR also called MDR (Memory Data Register).

Memory Control Signals: R/W (Read / Write) Memory signal can do three actions:

- **READ:** Copy contents of an addressed word into the MBR.
- **WRITE:** Copy contents of the MBR into an addressed word.
- **NOTHING:** The memory is expected to retain the contents written into it until those contents have been rewritten.

4.2.5 Characteristic of Memory

- **Capacity:** It is the global volume of information (in bits) that the memory can store.
- **Access time:** It is the time interval between the read/write request and the availability of the data.
- **Cycle time:** It is the minimum time interval between two successive accesses.
- **Throughput:** It is the volume of information exchanged per unit of time, expressed in bits per second.
- **Non-volatility:** It characterizes the ability of a memory to store data when it is not being supplied with electricity.

4.3 Associative Memory

- It is also known as content addressable memory(CAM) or associative storage or associative array.
- It is a special type of memory that is optimized for performing searches through data, as opposed to providing a simple direct access to the data based on an address.
- It is a hardware search engines, a special type of computer memory used in certain very high searching applications.
- It is composed of conventional semiconductor memory (usually SRAM) with added comparison circuitry that enable a search operation to complete in a single clock cycle.

Where is associative memory used?

We are only using this associative memory in memory allocation format and it is widely used in database management systems, etc.

Advantages of Associative Memory: This is suitable for parallel searches. It is also used where search time needs to be short. Associative memory is often used to speed up databases, in neural networks and in the page tables used by the virtual memory of modern computers.

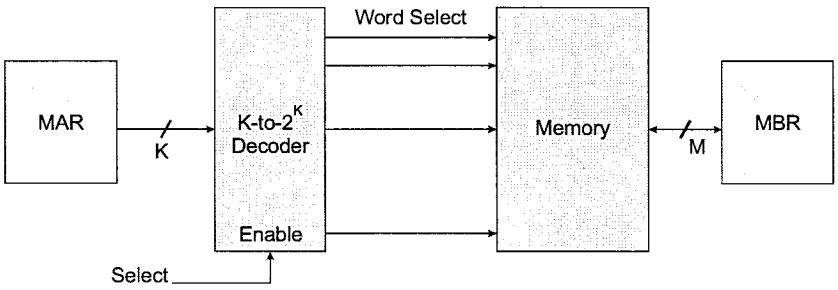
Disadvantages of Associative Memory: It is expensive than RAM, as each cell must have storage capability and logical circuits for matching its content with external argument.

Associative Memory Vs Random Access Memory (RAM)

- In RAM, the user supplies a memory address and the RAM returns the data word stored at that address.
- In associative memory, the user supplies a data word and the associative memory searches its entire memory to see if that data word is stored anywhere in it.
- If the data word is found, the associative memory returns a list of one or more storage addresses where the word was found.
- Hardware of associative memory allows operations to occur in a single-clock cycle, as opposed to the much greater time required for an algorithmic based search through traditional RAM.
- It is a special type of memory that is optimized for performing searches through data, as opposed to providing a simple direct access to the data based on an address as in RAM.

4.4 Address Space

The **memory size** is defined in terms of the amount of primary memory actually installed. The **address space**, determined by the size of the MAR, indicates the range of addresses that actually can be generated.



Logical view of memory access with MAR and MBR

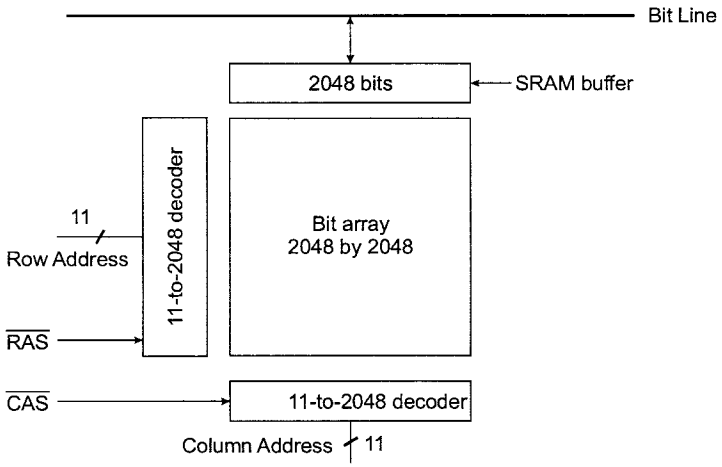
NOTE: An N–bit MAR can address 2^N distinct memory locations, $0...2^N - 1$.

4.4.1 Memory Chip Organization

Consider a 4 Megabit memory chip, in which each bit is directly addressable.

$$4M = 2^{22} = 2^{11} \times 2^{11}, \text{ and } 2^{11} = 2{,}048$$

The linear view of memory has a 22–to–2^{22} decoder (22–to–4,194,304 decoder). If we organize the memory as a two–dimensional grid of bits, then the design calls for two 11–to–2048 decoders (shown in Figure).

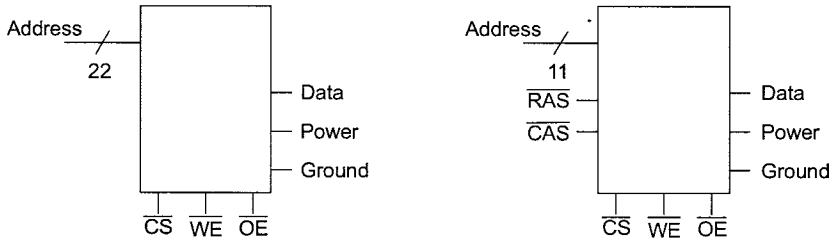


Two Dimensional Memory

Two dimensional array approach can be used to create a faster memory. This is done by adding a SRAM (Static RAM) buffer onto the chip. Consider the 4Mb (four megabit) chip with a 2Kb SRAM buffer as shown in the above Figure. RAS signal is used for row address selection and CAS is used for column address selection.

For reading the chip, a Row Address is passed to the chip, followed by a number of column addresses. When the row address is received, the entire row is copied into the SRAM buffer. Subsequent column reads come from that buffer.

***Example*-4.1** Consider the following linear memory with 22 address lines (shown left) and two–dimensional memory with 11 address lines for both row and column addresses (shown right). How many pins needed for each configuration? Both figures are having three control signals (CS, WE, OE).



Solution:

Linear Memory

Address Lines: 22 pins,Power & Ground : 2 pins, Data: 1 pin, and Control: 3 pins.

Total number of pins required: 22 + 2 + 1 + 3 = 28

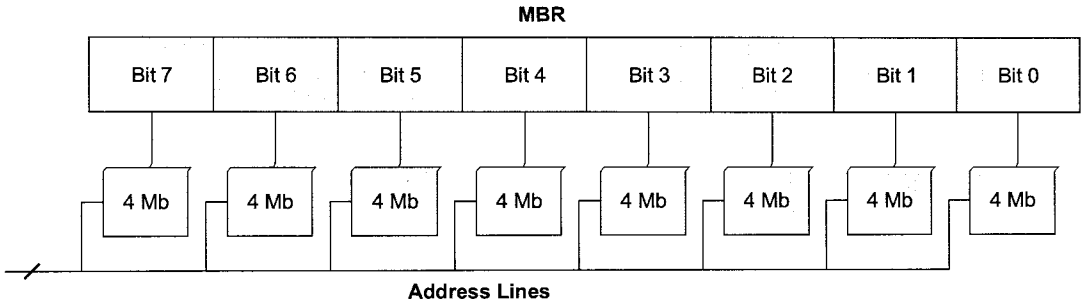
Two-Dimensional Memory

Address Lines: 11 pins, Row/Column: 2 pins, Power & Ground : 2 pins, Data: 1 pin, and Control: 3 pins.

Total number of pins required: 11 + 2 + 2 + 1 + 3 = 19

4.4.2 Memory Interleaving

Suppose a 64MB memory made up of the 4Mb chips as shown in the below.



We organize the memory into 4MB banks, each having eight of the 4MB chips. The memory thus has 16 banks, each of 4MB. 64 MB memory = 2^{26}, so 26 bits used for addressing.

$16 = 2^4$, so 4 bits of address to select the bank, and $4M = 2^{22}$, so 22 bits of address to each chip.

NOTE

In general, an N–bit address, with N = L + M, is broken into two parts.

- L-bit bank select, used to activate one of the 2^L banks of memory, and
- M-bit address that is sent to each of the memory banks.

When one of the memory banks is active, the other ($2^L - 1$) are inactive. All banks receive the M-bit address, but the inactive ones do not respond to it.

Memory interleaving is classified into two types:

1. **High Order Interleaving (Memory Banking):** In high-order interleaving, the most significant bits of the address select the memory chip. The low–order bits are sent as addresses to each chip.

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

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.

2. **Low Order Interleaving:** In low order interleaving, the least significant bits select the memory bank (module).

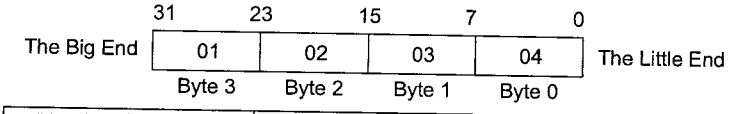
Address bits	25 - 4	3 - 0
Use	Address to the chip	Bank Select

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

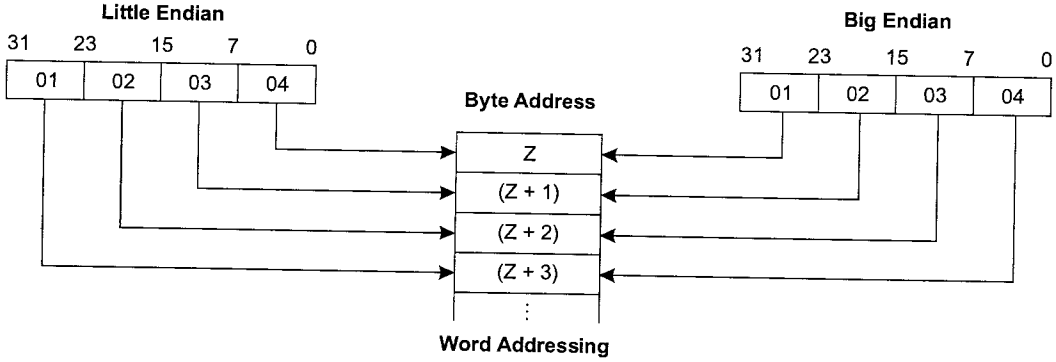
In low order interleaving, consecutive memory addresses are in different memory modules. This allows memory access at much faster rates that allowed by the cycle time.

4.4.3 Word Addressing in a Byte Addressable Machine

A 32-bit word at address Z contains bytes at addresses : Z, Z + 1, Z + 2, and Z + 3.



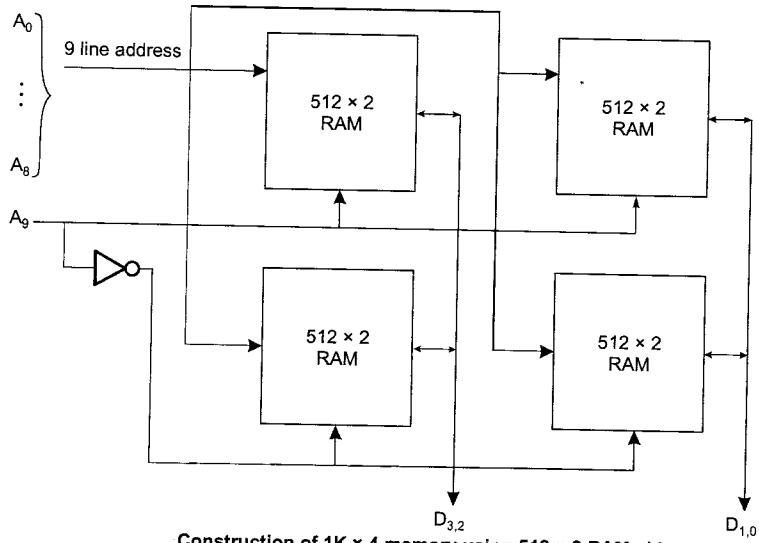
Address	Big-Endian	Little-Endian
Z	01	04
Z + 1	02	03
Z + 2	03	02
Z + 3	04	01



NOTE: Suppose the required large RAM memory size is K × L and the small size Ram chip capacity is m × n, then the number of small size chips required can be calculated as:
The number of chips each of size m × n = s = $\lceil (K \times L)/(m \times n) \rceil$

Construction of 1024 × 4 RAM using 512 × 2 RAM Chips

Suppose a large memory of 1K × 4 is to be constructed using 512 × 2 RAM chips. For small size RAM chip of 512 × 2, number of address lines required is 9 and number of data lines is 2. For large memory of 1 K × 4, number of address lines required is 10 and number of data lines is 4.



Construction of 1K × 4 memory using 512 × 2 RAM chips

The number of rows = 1K/512 = 1024/512 = 2

The number of columns = 4/2 = 2

Hence, the number of small size RAMs required = 2 × 2 = 4.

The interconnection diagram is shown in Figure. Here in the diagram, only two rows are present. So, the first row is selected (activated) by A_9 line of the address bus directly and the second row is selected by its complement bit information. In other words, if A_9 line contains logic 1, then first row of chips will be selected and otherwise the second row will be selected.

***Example*-4.2** **Suppose a DRAM memory has 4 K rows in its array of bit cells, its refreshing period is 64 ms and 4 clock cycles are needed to access each row. What is the time needed to refresh the memory if clock rate is 133 MHz? What fraction of the memory's time is spent performing refreshes?**

Solution:

In DRAM memory, number of row of cells in memory is 4K = 4096 and 4 clock cycles are needed to access each row. Therefore, number of cycles needed to refresh all rows = 4096 × 4 = 16384 cycles.

Since clock rate is 133 MHz.

The time needed to refresh all rows = $16384/(133 \times 10^6)$ seconds

$= 123 \times 10^{-6}$ seconds

$= 0.123$ ms [1 ms = 10^{-3} sec]

Thus, the refreshing process occupies 0.123 ms in each 64 ms time interval.

Therefore, refresh overhead is 0.123/64 = 0.002.

Hence, only 0.2% of the memory's time is spent performing refreshes.

***Example*-4.3** **How many 256 × 4 RAM chips are needed to provide a memory capacity of 2048 bytes?**

Solution:

The given RAM memory size is 256 × 4. This memory chip required 8 (because $256 = 2^8$) address lines and 4 data lines. Size of memory to be constructed is 2048 bytes, which is equivalent to 2048 × 8.

Thus, it requires 11 (because $2048 = 2^{11}$) address lines and 8 data lines.

The number of rows required = 2048/256 = 8

The number of columns required = 8/4 = 2

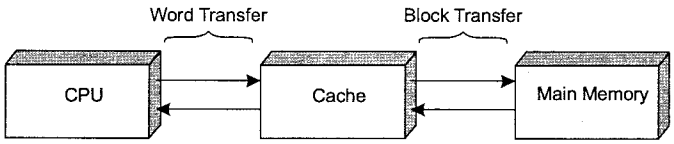
Total number of RAMs each of size 256 × 4 required = 8 × 2 = 16.

4.5 Cache Memory Design

4.5.1 Cache Memory

Cache is a random access memory used by the central processing unit (CPU) to reduce the average time to access memory. Cache memory stores instructions that are repeatedly required to run programs, for improving overall system speed as cache memory is designed to accelerate the memory function. The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations.

The advantage of cache memory is that the CPU does not have to use the motherboard 's system bus for data transfer, enabling process of data transfer is processed much more faster by avoiding bottleneck created by system bus.



When the computer read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory.

- **Principle of Locality:** Program access a relatively small portion of the address space at any instant of time.
  (a) *Temporal Locality (Locality in Time):* If an item is referenced, it will tend to be referenced again soon.
  (b) *Spatial Locality (Locality in Space):* If an item is referenced, items whose addresses are close by tend to be referenced soon.
- **Cache Hit:** If data is present in the cache then it is called as cache hit.
- **Hit Rate:** It is the fraction of memory access found in the cache.
- **Hit Time:** It is the time to access the cache which consists of Cache access time and time to determine hit.
- **Cache Miss:** If data is not present in the cache then it is called as cache miss.
  Miss Rate = 1 – (Hit Rate)
  Miss Penalty = Time to replace a block in the cache + Time to deliver the block to the processor.
- **Average Access Time:** = Hit Time × (1 – Miss Rate) + Miss Penalty × Miss Rate

Remember

Cache Memory: Cache memory is a very high speed semiconductor memory. It acts as a buffer between the CPU and main memory. It is used to hold those parts of data and program which are most frequently used by CPU.

- **Advantages:** Cache memory is faster than main memory. It consumes less access time as compared to main memory. It stores the program that can be executed within a short period of time. It stores data for temporary use.
- **Disadvantages:** Cache memory has limited capacity. It is very expensive. Auxiliary Memory (Secondary Memory). Auxiliary memory is much larger in size than main memory but is slower. It normally stores system programs, instruction and data files. Secondary memories cannot be accessed directly by a processor. First the data/information of auxiliary memory is transferred to the main memory and then that information can be accessed by the CPU.

Elements of Cache Design

- **Cache Size:** It is the amount of main memory data that cache can hold.
- **Block Size:** It is the number of words (bytes) grouped into a single unit called a block.
- **Mapping Function:** Direct mapped, Associative and Set associative mapping.
- **Replacement Algorithms:** Least Recently Used Algorithm and FIFO.
- **Write Policy:** Write Through and Write Back.

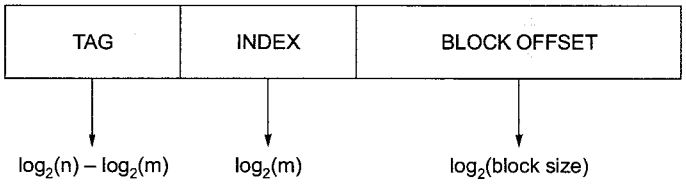
NOTE

Cache treats main memory as a set of blocks. As the cache size is much smaller than main memory so the number of cache lines are very less than the number of main memory blocks. So a procedure is needed for mapping main memory blocks into cache lines. Cache mapping scheme affects cost and performance.

There are three methods in block placement: Direct Mapped Cache, Fully Associative Mapped Cache and Set-Associative Mapped Cache.

4.5.2 Direct Mapped Cache (1-way Set Associative Cache)

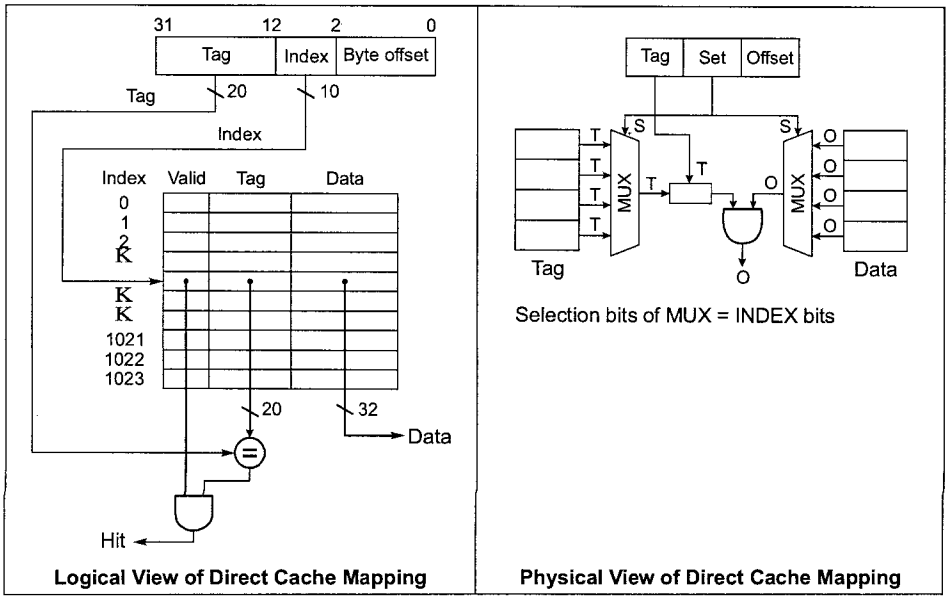
The memory address is divided into three parts in direct mapping: **TAG**, **INDEX** (also called as SET/ BLOCK/Set Index/Block Index/Line), and **BLOCK OFFSET** (also called as offset/word).



Let the main memory contains n blocks (which require $\log_2 n$ bits of physical address) and cache contains m blocks. A given memory block can be mapped into one and only cache line (block). Here, n/m different blocks of memory can be mapped (at different times) to a cache block. Each cache block has a tag saying which block of memory is currently present in it, each cache block also contain a valid bit to ensure whether a memory block is in the cache block currently.

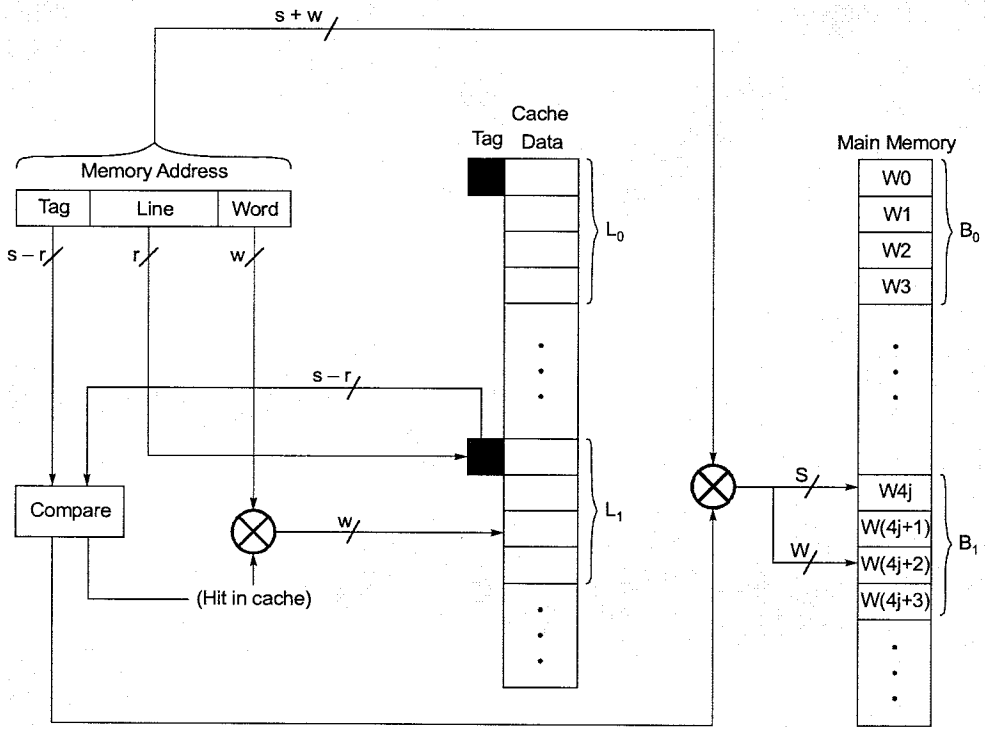
- Number of bits in the tag = $\log_2 (n/m)$
- Number of sets (or blocks) in the Cache = m
- Number of bits to identify the correct Block = $\log_2 (m)$
- Number of Sets in cache = Number of Blocks in cache
- Each set contain only one block, so in direct cache mapping set is also called as block.
- INDEX is used to select the memory block.
- TAG is used to select the cache block from main memory set
- Select location within block using block offset.
- TAG + INDEX = Block Address

Cache Line Number = (Main memory Block number) MOD (Number of Cache lines)



Remember

- The address is broken into three parts: (s-r) MSB bits represent the tag to be stored in a line of the cache corresponding to the block stored in the line; r bits in the middle identifying which line the block is always stored in; and the w LSB bits identifying each word within the block.



- The number of addressable units = 2^{s+w} words or bytes
- The block size (cache line width not including tag) = 2^w words or bytes
- The number of blocks in main memory = 2^s (i.e., all the bits that are not in w)
- The number of lines in cache = m = 2^r
- The size of the tag stored in each line of the cache = (s – r) bits
- **Cache location = (block address) MOD (Number of blocks in cache)**

***Example*-4.4** A 32 bits byte address Direct-mapped cache defined as:

Cache size = 2^n block, n bits used for index

Block size = 2^m block, m bits for word between block, 2 used for byte part of address

Size of tag field : 32 – (m + n + 2)

One valid bit field is used in cache.

Find the direct mapped cache size (in bits).

Solution:

Total number of bits in direct-mapped cache = 2^n x (block size + tag size + valid field size)

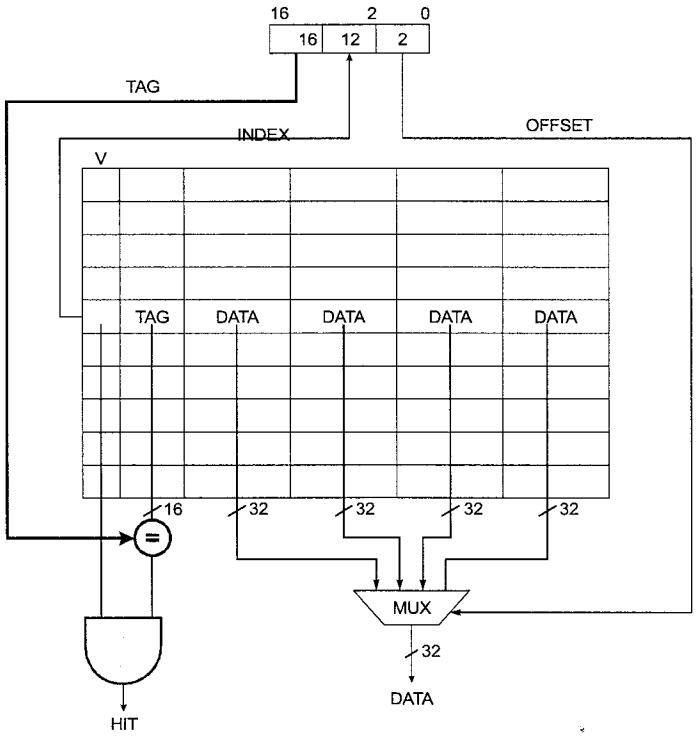
Block size = 2^m words

1 bit valid field is needed.

$$\text{Cache size} = 2^n \times (2^m \times 32 + 32 - n - m - 2) + 1) \text{ bits}$$
$$= 2^n \times (2^m \times 32 + 31 - n - m) \text{ bits}$$

Example-4.5 A 32-bit address direct cache mapping with 2048 blocks in cache and each block has 16-bits in byte addressable memory. Design the direct cache mapping with indicating all the fields of address.

Solution:



Example-4.6 How many total bits are required for a direct-mapped cache with 16 kB of data, 1 bit field for valid and 4-word blocks, assuming a 32-bit address?

Solution:

16 kB = 2^{12} words of data, 4 words block size (= 2^2).

So, there are 1024 blocks (= 2^{10})

$$\text{Tag} = 32 - 10 - 2 - 2 \text{ bits}$$

$$\text{Cache Entry size} = 128 \text{ bits of data} + \text{tag bits} + \text{valid bit.}$$

$$\text{Cache size} = 2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1$$

$$= 2^{10} \times 147 = 147 \text{ K bits}$$

Disadvantage of the Direct Mapped Cache

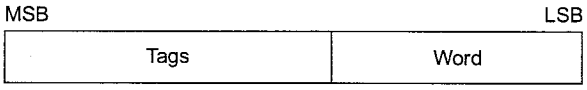
It is easy to build, but suffer the most from thrashing due to the 'conflict misses' giving more miss penalty. There is no choice of which cache entry's contents to evict, so no replacement policy. They may collide if two locations map to the same entry.

NOTE: Larger blocks exploit spatial locality to lower miss rate. (Increasing block size normally decreases miss rate)

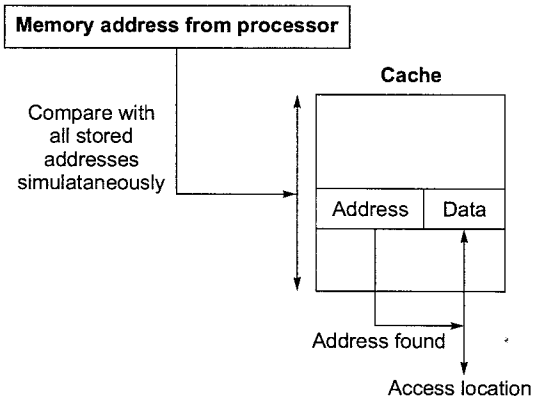
4.5.3 Fully Associative Cache

- It is also called as m-way set associative cache, where m is number of blocks and all blocks fit into one set.
- Instead of using a cache index, compare the tags of all cache entries in parallel
- Because no bit field in the address specifies a line number the cache size is not determined by the address size.
- Associative-mapped memory is also called "content-addressable memory".
- Items are found not by their address but by their content
- Used extensively in routers and other network devices

Any main memory block can mapped into any cache line. Main memory address is divided into two groups which are TAG and WORD (Offset) bits. Words are low-order bits and identifies the location of a word within a block.

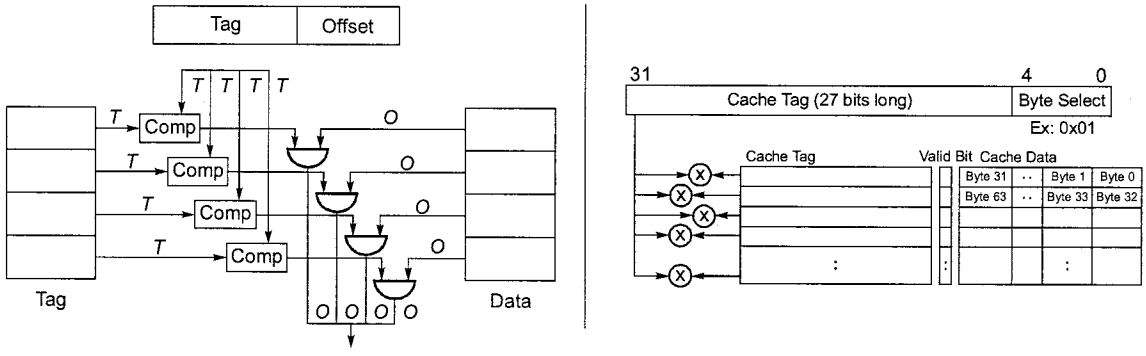


TAG bits are high-order bits which identifies the block.



Logical View of Associative Cache

If a miss occur CPU bring the block from the main memory to the cache, if there is no free block in the corresponding set it replaces a block and put the new one. CPU uses different replacement policies to decide which block is to replace.



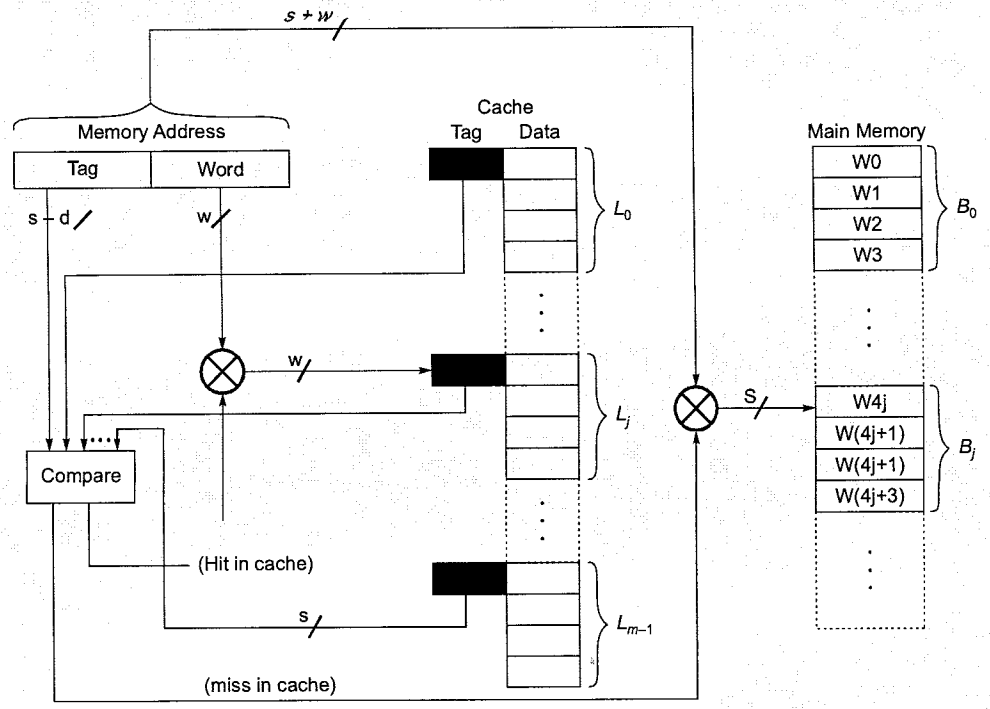
Physical View of Associative Cache

Advantage: Associative mapping allows flexibility in choice of replacement blocks when cache is full.

Disadvantages: No replacement policy has been implemented in the experiment (**Design issue**). High cost for implementing parallel tag comparison, but suffer the most from thrashing due to the 'conflict misses' giving more miss penalty.

Remember

- The address is broken into two parts: a tag used to identify which block is stored in which line of the cache (s bits) and a fixed number of LSB bits identifying the word within the block (w bits).



- Address length = (s + w) bits where w = log(block size)
- The number of addressable units = 2^{s+w} words or bytes
- The block size (cache line width not including tag) = 2^w words or bytes
- The number of blocks in main memory = 2^s (i.e., all the bits that are not in w)
  $= 2^{s+w}/2^w$
- The number of lines in cache is not dependent on any part of the memory address (undetermined)
- The size of the tag stored in each line of the cache = s bits
- Cache location = any

4.5.4 Set Associative Cache (k-way Set Associative Cache)

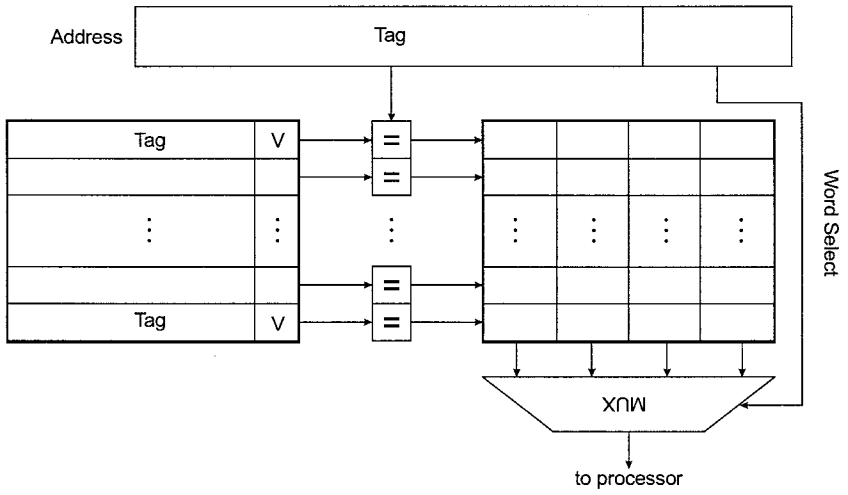
A compromise that provides strengths of both direct and associative approaches. Cache is divided into a number of sets of lines(blocks), where each set contains a fixed number of lines.

k-direct mapped caches operates in parallel. Tags are much smaller than fully associative memories. Comparators are used for simultaneous lookup and less expensive.

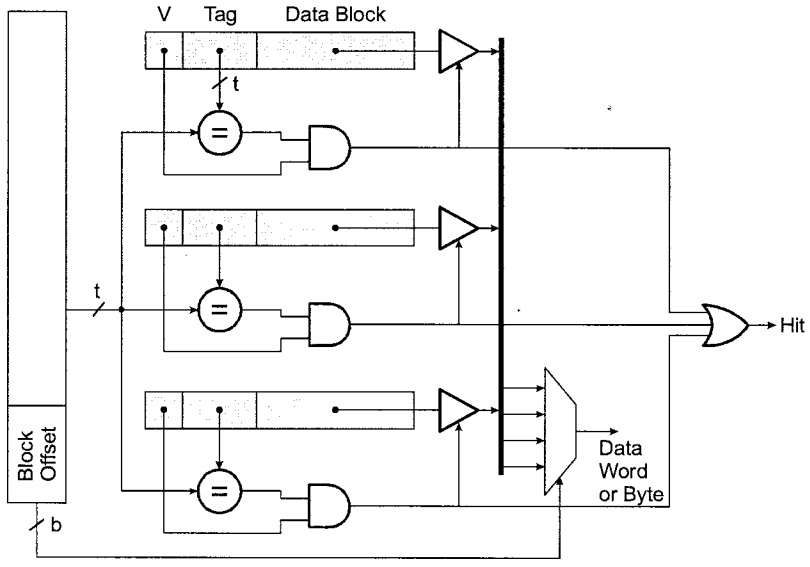
Main memory address is divided into three groups which are TAG, SET, and (OFFSET) bits.

TAG	SET	OFFSET

- k-entries for each Cache Index
- Cache Index selects a "set" from the cache (Index also called as Set Index or Set)
- Data is selected based on the tag result
- A given block maps to any line in a given set determined by that block's address
- **Set selection = (block address) MOD (Number of sets in cache)**



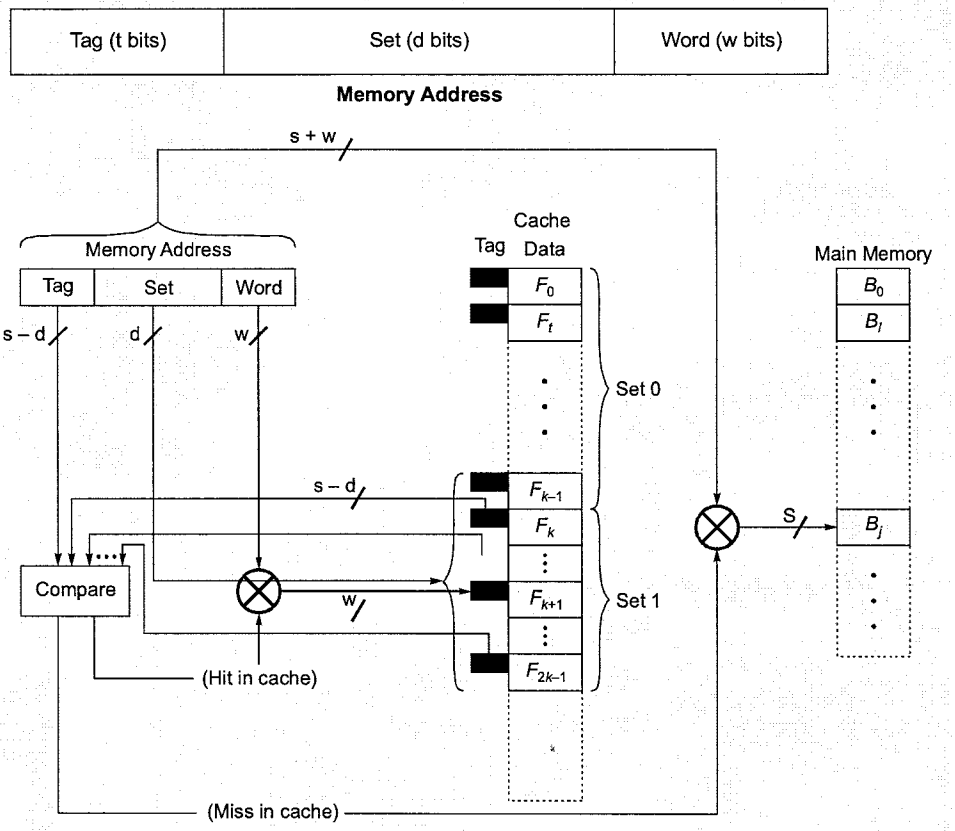
Logical View of Set Associative Cache Memory



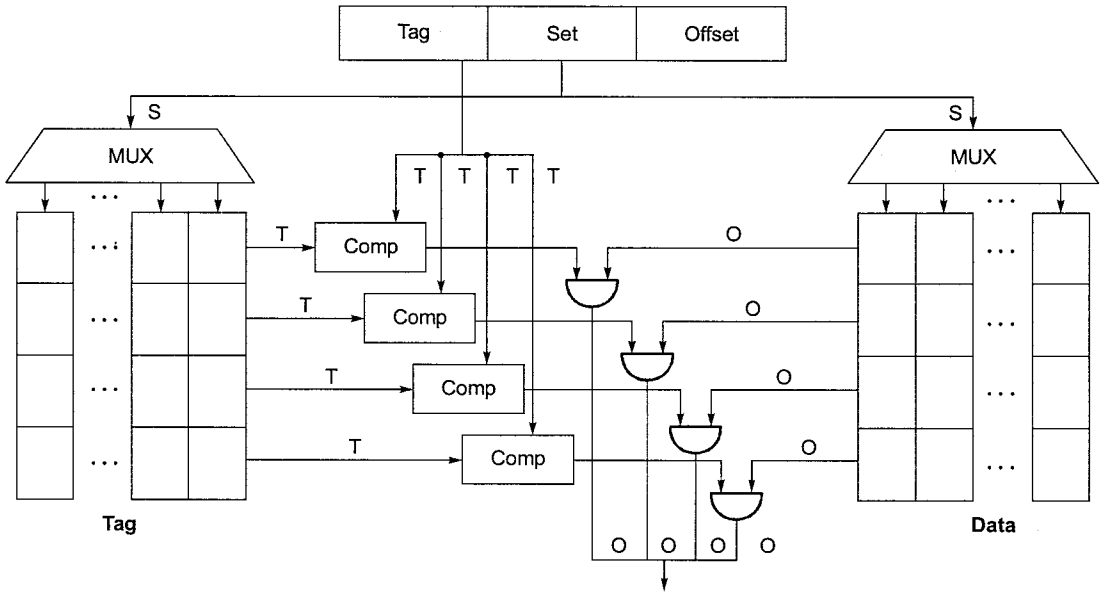
Remember

For a k-way set associative cache with v sets (each set contains k lines):

- Address length = t + d + w = (s + w) bits where w = log (block size)
- The number of addressable units = 2^{s+w} words = 2^{t+d+w}
- Number of tag bits (t) = s-d



- Address length = (t + d + w) bits where w = log (block size) and d = log (v)
- Number of addressable units = 2^{t+d+w} words
- Size of tag = t bits
- Block size = line size = 2^w words
- Number of blocks in main memory = $2^t + d$
- Number of sets = v = 2^d
- Number of lines in set = k
- Number of lines in cache = k × v = k × 2^d
- Set number = Memory block number % number of sets
  i = j modulo v, (where i is set number and j is main memory block number)
- A Block j can be mapped only into one of lines of set i.



Physical View of Set Associative Cache Memory

Advantages of set-associative cache: This cache memory has highest hit ratio compared to other two cache memories.

Disadvantages of set-associative cache: This is the most expensive memory. The cost increases as set size increases.

***Example*-4.7** Consider a system with 4-way set associative cache of 256 KB. The cache line size is 8 words (32 bits per word). The smallest addressable unit is a byte, and memory addresses are 64 bits long.

(a) How many bits are used for TAG and INDEX fields of cache mapping?

(b) What memory addresses can map to set 289 of the cache?

(c) What percentage of the cache memory is used for tag bits?

Solution:

(a) Block size = 8 words (32 bytes).

Therefore, the number of bytes required to specify the block offset is $\log_2 32 = 5$ bits.

Cache size is 256 KB.

The number of sets = 256 KB/(32 × 4) = 2048 sets.

Therefore, the index field would require 11 bits.

The remaining 64 – 11 – 5 = 48 bits are used for the tag field.

(b) Memory locations with index bits 00100100001 will map to set 289.

(c) For each cache line (block), we have 1 tag entry.

The size of the cache line is 32 × 8 = 256 bits.

Number of TAG bits = 48

Total cache memory used for each line including TAG = 48 + 256

Therefore, the percentage of cache memory used for tag bits is 48 / (48 + 256) = 15.8%.

Example - 4.8 A computer uses 32-bit byte addressing. The computer uses a 2-way associative cache with a capacity of 32KB. Each cache block contains 16 bytes. Calculate the number of bits in the TAG, SET, and OFFSET fields of a main memory address.

Solution:

Cache block size = 16 bytes

So the OFFSET field must contain 4 bits ($2^4 = 16$).

Each set contains 2 cache blocks (2-way associative) so a set contains 32 bytes.

There are 32KB bytes in the entire cache, so there are 32KB/32B = 1K sets.

Thus the set field contains 10 bits ($2^{10} = 1K$).

Finally, the TAG field contains the remaining 18 bits (32 – 4 – 10).

18	10	4
TAG	SET	OFFSET

Comparisons of Cache Mappings

Finding a Block in Cache

Associativity	Location Method	Comparisons Required
Direct mapped	Index	1
Set associative	Index the set, search among elements	Degree of associativity
Full	Search all cache entries	Number of cache blocks
	Separate lookup table	0

Hit Ratio and Performance

Cache Type	Hit Ratio	Search Speed
Direct mapped	Good	**Best**
Fully associative	**Best**	Moderate
N-Way set Associative, N>1	Very Good, Better as N Increases	Good, Worse as N Increases

NOTE

- The Main memory is not physically partitioned in the given way, but this is the view of Main memory that the cache sees.
- Both Main Memory and cache memory divided into blocks of same size.
- A cache contains m lines and k-way associative for some k that divides m.
- When k = 1, the cached is called direct mapped.
- When k = m, the cached is called fully associative. In this case there is only one set so there are no set bits in the address decomposition.

4.5.5 Handling Cache Operations

Handling Cache Miss

When a data is not available in the cache, cache miss is produced. CPU must detect a miss and process the miss by fetching requested data from memory.

Types of Cache Misses

1. **Compulsory Miss (cold start misses or first reference misses):** Miss occurs when the first access to a block happens. In this miss, the block must be brought into the cache.
2. **Capacity Miss:** It occurs when program working set is much larger than the cache capacity. Blocks are being discarded from cache because cache cannot contain all blocks needed for program execution.
3. **Conflict Miss (collision misses or interference misses):** In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame.
4. **Coherence Miss (Invalidation):** It occurs when other external processors (e.g., I/O) updates memory.

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Coherence Miss	Same	Same	Same

Cache Block Replacement Policy Techniques

1. **Random block replacement:** Replace the block chosen at random.
2. **LRU (Least-recently used):** Replace block not accessed for longest time.
3. **FIFO (first-in-first-out):** Insert block into queue when accessed and choose block to replace by deleting from queue.

Cache miss handling is done in collaboration with processor control unit, with a separate controller that initiates memory access and refill the cache. When cache miss is being processed, pipeline stall is created as opposed to an interrupt, which require saving the state of all register. For cache miss, the entire processor is stalled, essentially freezing the content of temporary and programmer-visible register while waiting for memory.

4.5.6 Handling Writes

Cache read is much easier to handle than cache write. Keep data in the cache and memory consistent for a write operation, for this Caches may be implemented as either write-back or write-through systems.

1. Write-Through Cache

- Every write to cache causes a write to main memory. In this system ensures that the contents of the level and the next level down are always the same in memory hierarchy.
- After block is fetched from memory and placed into cache, word that caused the miss can be overwrite into cache block. Words also writes to main memory using full address.
- Disadvantage: Write-through take long time and slow down processor considerably. This problem can be solved by using write buffer to stores the data while it is waiting to be written to memory.
- It is not necessary to record which lines have been written, because the data in a write-through cache is always consistent with the contents of the next level (memory). Evicting a line can be done by writing the new line over the old one, reducing the time to bring a line into the cache.

2. Write-Back Cache

- Write-back levels keep stored data in the cache. When a block that has been written is evicted from the cache, the contents of the block are written back (copied) into the main memory.

- The cache tracks which locations have been written over(these location will be marked *dirty*). The data then written back to main memory when it is evicted from cache. This is because a miss in write-back cache sometimes required two memory accesses to service: write dirty location to memory, then read new location from memory.
- **Advantages:** Main memory only written when "dirty" block replaced. Extra dirty bit for each block set when cache block written to.Reduces number of slow main memory writes.
- **Disadvantages:** Write-back is more complex to implement than write-through.

Handling write Misses of Cache

- Write allocate: When there is a write miss, the block is loaded in the cache and the write is reattempted.
- No-write allocate: When there is a write miss, only main memory is changed (block not loaded in the cache).

Cache Coherency

Keeping the data in the cache and in the memory consistent is called cache coherency. For a single processor system, Write Back or Write Through achieves this. For multiprocessor machines sharing memory, things become more difficult. When one processor writes, not only the memory, but all the other processor caches must be updated. There are two main ways of achieving this:

- Snoopy Cache - look at the address bus (snoop) to see what the other caches are doing.
- Directory - Maintain a global map of the cache entries.

4.5.7 Types of Caches

- L1 Cache: Cache built into the CPU itself is Level 1 (L1) cache: L1 cache holds most recent data, so when the data is needed again, the microprocessor check this cache first, so it don't need to go to main memory or Level 2.

  The main motivations behind this concept is "locality of reference", which a location just accessed by the CPU has a higher probability being visited again in short term.
- L2 Cache: It resides on a separate chip next to the CPU is Level 2 (L2) cache: L2 cache stores recent used data that are not in L1 cache. *Some CPUs have both L1 and L2 cache built-in and designate the separate cache chip as Level 3 (L3) cache.*

  Cache that is built into the CPU is faster than separate cache. Separate cache is faster than RAM. Built in cache is run at the speed of the microprocessor.
- Disk cache: It contain most recently read in data from the hard disk. This cache is slower than RAM.
- Instruction cache vs Data cache: Instruction or I-cache stores instructions only, while a data or D-cache stores data only. Separating the stored data in this way recognizes the different access behaviour patterns of instructions and data. For example, programs tend to involve few write accesses, and they often exhibit more temporal and spatial locality than the data they process.
- Unified cache vs Split cache: A cache that stores both instructions and data is referred to as unified cache. A split cache, on the other hand, consists of two associated but largely independent units: An I-cache for instructions and a D-cache for data. While a unified cache is simpler, a split cache makes it possible access programs and data concurrently. A split cache can also be designed to manage its I-and D-cache components differently.

4.5.8 Performance

Cache Memory Performance

The performance of the cache memory is measured in terms of a quantity called hit ratio. When the CPU refers to memory and find the word in cache, it is said that a hit occurred. If the word is not found in cache, then the CPU refers to the main memory for the desired word and it is referred to as a miss to cache.

- **Hit Ratio (h):**

$$\text{Hit ratio (h)} = \frac{\text{Number of hits}}{\text{Total CPU references to memory}} = \frac{\text{Number of hits}}{\text{Number of hits + Number of misses}}$$

The hit ratio is nothing but a probability of getting hits out of some number of memory references made by CPU. So its range is $0 \le h \le 1$.

- **Average Access Time (t_{avg}):** Let, t_c, h and t_m denote the cache access time, hit ratio in cache and the main memory access time, respectively.

$$t_{avg} = h \times t_c + (1 - h) \times (t_c + t_m) = t_c + (1 - h) \times t_m$$

Average memory access time = Hit time + Miss rate × Miss Penalty

Miss rate is the fraction of accesses that are not in the cache (i.e., (1 – h))

Miss penalty is the additional clock cycles to service the miss, the extra time needed to bring the desired information into the cache from main memory in case of a miss in cache is called the miss penalty.

CPU Performance (using Write back and Write through Cache Design)

CPU time divide into clock cycle that spends for executing program, and clock cycles that spends for waiting for memory system. Cache hits are part of normal CPU execution cycle.

CPU time = (CPU execution clock cycles + memory-stall clock cycles) × Clock cycle time

Assume that memory-stall clock cycles initially come from caches misses.

Memory-stall clock cycles (for write-back cache):

- Memory-stall clock cycles = Read-stall cycles + write-stall cycles
- $\text{Read-write cycle} = \frac{\text{Reads}}{\text{Programs}} \times \text{Read miss rate} \times \text{Read miss penalty}$
- $\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} + \text{Write buffer stalls}\right)$

Memory-stall clock cycles (for write-through cache):

- Assume write buffer stalls are negligible. Every access (read or write) treated similar.
- $\text{Memory-stall clock cycles} = \frac{\text{Memory access}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalties}$
- $\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalties}$

Example - 4.9 Assume miss rate of an instruction cache is 2% and miss rate of data cache is 4%. If a processor have CPI of 2 without any memory stalls and miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume frequency of all loads and stores is 36%.

Solution:

$$\text{Instruction miss cycles} = I \times 2\% \times 100 = 2.00 \times I$$

$$\text{Data miss cycles} = I \times 36\% \times 4\% \times 100 = 1.44 \times I$$

Total number of memory stall cycles = Instruction miss cycle + data miss cycle

$$= 2.00\,I + 1.44\,I = 3.44\,I$$

$$\text{Total CPI} = 2 + 3.44 = 5.44$$

Since there is no change in instruction count or clock rate, hence, the ratio of CPU execution is:

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times \text{CPI}_{\text{stall}}}{I \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}} = \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2} = 2.72$$

The performance with the perfect cache is better by $\frac{5.44}{2} = 2.72$.

4.5.9 Measuring and Improving Cache Performance

Techniques used to minimize the average memory access time (AMAT):

- Reducing hit time
- Reducing miss penalty
- Reducing miss rate
- Reducing miss penalty × Miss rate

Techniques for Reducing Hit Time:

- Small and simple caches
- Pipelined cache access
- Trace caches
- Avoid time loss in address translation
  - (*i*) Virtually indexed, physically tagged cache.
  - (*ii*) Virtually addressed cache.

Techniques for Reducing Miss Penalty:

- Use Multi level caches
- Critical word first and early restart
- Giving priority to read misses over write
- Merging write buffer
- Victim caches

Techniques for Reducing Miss Rate:

- Increase Block Size (Large Block Size):
  - (*i*) It reduces compulsory misses
  - (*ii*) It may increase conflict misses
  - (*iii*) It may increase capacity misses if cache is small.

- Higher Associativity:
  (*i*) It will reduce the cache miss rate of set associative cache.
  (*ii*) It will increase the cost of set associative cache.
- Use victim Cache (pseudo associative cache/buffer):
  (*i*) Victim Cache is small fully associative cache placed between a cache and refill path.
  (*ii*) Which holds only victim blocks that are discarded recently from a cache because of miss.
  (*iii*) It reduce conflict misses.
- Large Cache:
  (*i*) It reduce capacity miss rates.
  (*ii*) It also reduce conflict miss rates.
- Compiler Optimizations: It can reduce miss rates without changing the hardware. Some of the optimizations are loop merging (fusion) for improving temporal locality, loop interchange and Array merging optimizations for improving spatial locality.

Techniques for Reducing (Miss Rate × Miss Penalty):

- Nonblocking cache
- Hardware pre-fetching
- Compiler controlled pre-fetching

Example-4.10 A hierarchical cache-main memory subsystem has the following specifications: (*i*) Cache access time of 50 nsec (*ii*) Main memory access time of 500 nsec (*iii*) 80% of memory request are for read (*iv*) Hit ratio of 0.9 for read access and the write-through scheme is used.

Calculate the following:

(a) Average access time of he memory system considering only memory read cycle.

(b) Average access time of the system both for read and write requests.

Solution:

Given, Cache access time t_c = 50 nsec

Main memory access time t_m = 500 nsec

Probability of read p_r = 0.8

Hit ratio for read access h_r = 0.9

Writing scheme: Write-through

(a) Consider only memory read cycle

The average access time $t_{av.r} = h_r \times t_c + (1 - h_r) \times (t_c + t_m)$

$= 0.9 \times 50 + (1 - 0.9) \times 550 = 100$ nsec.

(b) For both read and write cycles

The average access time $= p_r \times t_{av.r} + (1 - p_r) \times t_m$

Since in write-through method, access time for write cycle will be the main memory access time.

$= 0.8 \times 100 + (1 - 0.8) \times 500 = 180$ nsec.

Example-4.11 For a cache memory of size 32 KB, how many cache lines (blocks) does the cache hold for block sizes of 32 or 64 bytes?

Solution:

The number of blocks in cache = Size of cache/block size.

Thus, for block size of 32 bytes, the number of blocks = (32 × 1024)/32 = 1024

Similarly, for block size of 64 bytes, the number of blocks = (32 × 1024)/64 = 512

Example-4.12 A computer has a main memory of 64 K × 16 and a cache memory of 1 K words. The cache uses direct mapping with a block size of four words.

(a) How many bits are there in the tag, index, block and word fields of the address format?

(b) How many bits are there in each word of cache?

(c) How many blocks can the cache accommodate?

Solution:

(a) The main memory size = 64 K × 16

Therefore, the CPU must generate the address of 16 bit (since 64 K = 2^{16})

The cache memory size = 1K

Therefore, the size of index field of cache = 10 bit (1 K = 2^{10})

The tag-field uses 16 – 10 = 6 bits

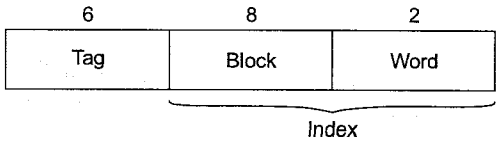
The size of each cache block = 4 words

Thus, the number of blocks in cache = 1024/4 = 256

Therefore the number of bits required to select each block = 8 (since 256 = 2^8)

The number of bits required to select a word in a block = 2, because there are 4 words in each block.

Thus, the address format is as follows:



(b) The main memory size = 64K × 16

Therefore, the number of bits in each word in cache = 16

(c) From part (a); the number of blocks in cache = 256.

Example-4.13 A three level memory system having cache access time of 15 ns and disk access time of 80 ns has a cache hit ratio of 0.96 and main memory hit ratio of 0.9. What should be the main memory access time to achieve effective access time of 25 ns?

Solution:

Given, Cache access time, t_c = 15 ns

Disk (secondary) memory access time, t_8 = 80 ns

Hit ratio for cache, h_c = 0.96

Hit ratio for main memory, h_m = 0.9

The average access time, t_{av} = 25 ns

Let, the main memory access time is t_x unit.

Now, we know, the average access time of the memory system.

$T_{av} = h_c \times t_c \times h_m \times (1 - h_c) \times (t_c + t_m) + (1 - h_c) \times (1 - h_m) \times (t_c + t_m + t_8)$

That is, $25 = 0.96 \times 15 \times 0.9 \times 0.04 \times (15 + t_m) + 0.04 \times 0.1 \times (15 , t_m + 80)$

By simplifying, we get t_m = 27

Hence, the main memory access time must be 27 ns to achieve the effective access time of 25 ns.

Example-4.14 Given the following, determine size of the sub-fields (in bits) in the address for direct mapping, associative and set associative mapping cache schemes: We have 256 MB main memory and 1 MB cache memory. The address space of this processor is 256 MB. The block size is 128 bytes. There are 8 blocks in a cache set.

Solution:

Given, The capcity of main memory = 256 MB

The capcity of cache memory = 1 MB

Block size = 128 bytes

A set contains 8 blocks

Since, the address space of the processor is 256 MB

The processor generates address of 28 bit to access a byte (word)

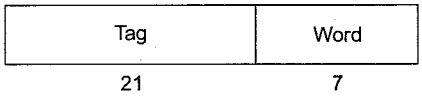
The number of blocks main memory contains = 256 MB/128 bytes = 2^{21}

Therefore, number of bits required to specify one block in main memory = 21

Since the block size is 128 bytes.

The number of bits required to access each word (byte) = 7

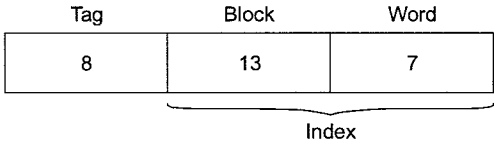
For associative cache, the address format:



The number of blocks cache memory contains = 1 MB / 128 bytes = 2^{13}

Therefore, number of bits required to specify one block in cache memory = 13.

For direct cache, the address format is:



Example-4.15 What is the bandwidth of a memory system that transfers 128 bit of data per reference, has a speed 20 ns per operation?

Solution:

Given the speed of 20 ns, one memory reference can initiate in every 20 ns and each memory reference fetches 128 bit (i.e., 16 bytes) of data. Therefore, the bandwidth of the memory system is 16 bytes/20 ns = $(16 \times 10^9)/20$ bytes/second = 8×10^8 bytes/second.

Example-4.16 A computer has direct mapped cache with 16 one-word blocks. The cache is initially empty. What is the observed hit ratio when the CPU generates the following word address sequence:

1, 4, 8, 5, 20, 17, 19, 5, 6, 9, 11, 4, 43, 5, 6, 9, 17?

Solution:

The direct mapping is expressed as I = J mod K

Where I = Cache block number

J = Main memory block number

K = Number of blocks in cache.

The processor generates the addresses for words in main memory. In our problem. K = 16 with one word per block and main memory block sequence is: 1, 4, 8, 5, 20, 17, 19, 5, 6, 9, 11, 4, 43, 5, 6, 9, 17. Thus, the corresponding cache block sequence and its word is as (block number, word address): (1, 1), (4, 4), (8, 8), (5, 5), (4, 20), (1, 17), (3, 19), (8, 56), (9, 9), (11, 11), (4, 4), (11, 43), (5, 5), (6, 6), (9, 9), (1, 17). Initially, cache is empty.

(Cache block number, word address)	(1, 1)	(4, 4)	(8, 8)	(5, 5)	(4, 20)	(1, 17)	(3, 19)	(8, 56)
Hit (H) / Miss (M)	M	M	M	M	M and replace	M and replace	M	M and replace
(Cache block number, word address)	(9, 9)	(11, 11)	(4, 4)	(11, 43)	(5, 5)	(6, 6)	(9, 9)	(1, 17)
Hit (H) / Miss (M)	M	M	M	M and replace	H	M	H	H

Therefore, hit ratio = 3/16.

***Example*-4.17** Consider a two level memory hierarchy of the form (M_1, M_2) where M_1 is connected directly to the CPU. Determine the average cost per bit c and average access time t_A for the data given below:

Level (i)	Capacity (S_i)	Cost (c_i)	Access time (t_{Ai})	Hit ratio (H)
M_1 (Cache)	1024	0.1000	10^{-8}	0.9000
M_2 (Main)	2^{16}	0.0100	10^{-6}	-

Solution:

$$\text{Average cost (c)} = \frac{c_1 S_1 + c_2 S_2}{S_1 + S_1} = \frac{0.1 \times 1024 + 0.01 \times 2^{16}}{2014 + 2^{16}} = 0.011384615$$

$$\text{Average access time } t_A = Ht_{A1} + (1 - H)\, t_{A2}$$
$$= 0.9000 \times 10^{-8} + (1 - 0.9000)\, 10^{-6} = 1.09 \times 10^{-7}$$

***Example*-4.18** In a two level virtual memory, $t_{A1} = 10^{-7}$ s and $t_{A2} = 10^{-2}$ s. What must be the hit ratio H be in order for the access efficiency to be atleast 90 percent of its maximum possible value.

Solution:

$t_{A1} = 10^{-7}$ e = 90% = 0.9
$t_{A2} = 10^{-2}$ Hit ratio H = ?

$$e = \frac{1}{r + (1 - r)H}$$

$$r = \frac{t_{A2}}{t_{A1}} = \frac{10^{-2}}{10^{-7}} = 10^5$$

$$0.9 = \frac{1}{10^5 + (1 - 10^5)H}$$

$$H = 0.999999$$

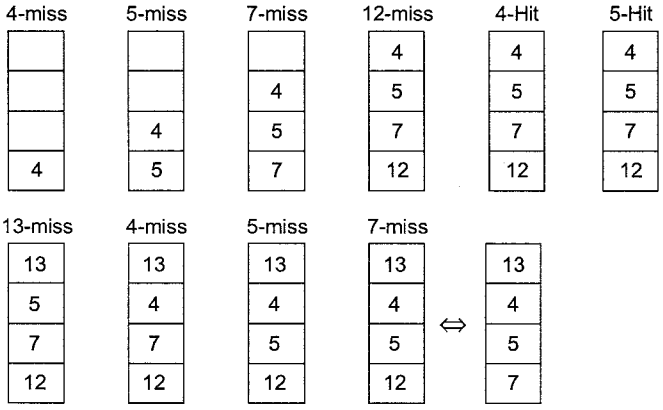
Example-4.19 Consider a 4 block cache memory (Initially empty) with the following main memory block references 4, 5, 7, 12, 4, 5, 13, 4, 5, 7. Identify the hit ratio by using

(a) FIFO
(b) LRU
(c) Direct mapped cycle
(d) 2-way set associative with LRU

Solution:

(a) FIFO:

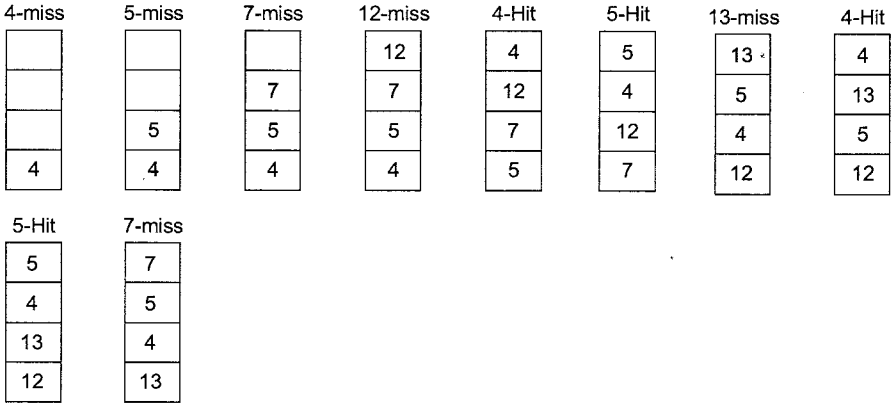
In FIFO, replace the cache block which is having the longest time stamp.



$$\text{Hit ratio} = \frac{2}{10} = 0.2$$

(b) LRU:

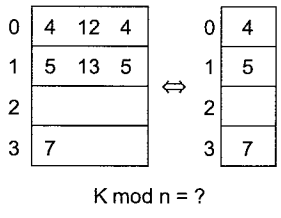
In LRU, replace the cache block which is having the less number of references with longest time stamp.



$$\text{Hit ratio} = \frac{4}{10} = 0.4$$

(c) Direct mapped cache:

4-miss; 4 mod 4 = 0
5-miss; 5 mod 4 = 1
7-miss; 7 mod 4 = 3
12-miss; 12 mod 4 = 0
4-miss; 4 mod 4 = 0



K mod n = ?

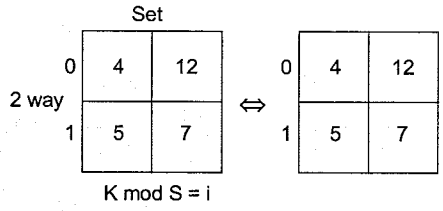
5-Hit; 5 mode 4 = 1
13-miss; 13 mod 4 = 1
4-Hit; 4 mod 4 = 0
5-miss; 5 mod 4 = 1
7-Hit; 7 mod 4 = 3

$$\text{Hit ratio} = \frac{3}{10} = 0.3$$

(d) 2-way set associative with LRU

$$S = \frac{n}{\text{p-way}} \Rightarrow \frac{4}{2} = 2$$

4-miss; 4 mod 2 = 0
5-miss; 5 mod 2 = 1
7-miss; 7 mod 2 = 1
12-miss; 12 mod 2 = 0
4-hit; 4 mod 2 = 0
5-hit: 5 mod 2 = 1
13-miss; 13 mod 2 = 1
4-hit; 4 mod 2 = 0
5-hit; 5 mod 2 = 1
7-miss, 7 mod 2 = 1



$$\text{Hit ratio} = \frac{4}{10} = 0.4$$

Summary

- Registers are the fastest type of memory, which are located internal to a processor. These elements are primarily used for temporary storage of operands.
- Cache is a very fast type of memory that can be external or internal to a given processor. Cache is used for temporary storage of blocks of data obtained from main memory (read operation) or created by the processor and eventually written to main memory (write operation).
- Main Memory is modelled as a large, linear array of storage elements that is partitioned into static and dynamic storage.
- Main memory is used primarily for storage of data that a program produces during its execution, as well as for instruction storage.
- Disk Storage is much slower than main memory, but also has much higher capacity. Because of the relatively long search times, we prefer not to find data primarily in disk storage, but to page the disk data into main memory, where it can be searched much faster.
- Archival Storage is offline storage such as a CD-ROM jukebox or magnetic tapes. This type of storage has a very long access time, in comparison with disk storage, and is also designed to be much less volatile than disk data.
- **Memory Access Time:** Defined in terms of reading from memory. It is the time between the address becoming stable in the MAR and the data becoming available in the MBR.
- **Memory Cycle Time:** It is defined as the minimum time between two independent memory accesses.
- **Direct mapped Cache:** A block with a given address can only be placed in a single location in the cache.
- ***Fully associative Cache:*** *A block can be placed anywhere in the cache.*
- **Set associative Cache:** A block can be placed anywhere within a set of locations in the cache.

  Assuming a cache with $B = 2^n$ blocks, and a block with block address A:
- **Direct mapped:** The index is A mod B (the last n bits are used for the block location)
- **Fully associative:** The block can go anywhere in the cache
- **k-way Set associative:** With m sets, block B can mapped to set A mod K (the last k bits tell you the set nb)
- **Cache Block Placement:** Where can a block be placed in the cache?
- **Cache Block Identification:** How is a block found if it is in the cache?
- **Cache Block Replacement:** Which block should be replaced on a cache miss?
- **Cache Write Strategy:** What happens on a write?

Student's Assignment

Q.1 Spatial locality refers to the problem that once a location is referenced
(a) it will not be referenced again
(b) it will be referenced again
(c) a nearby location will be referenced soon
(d) None of these

Q.2 The principle of locality justifies the use of
(a) interrupts (b) threads
(c) DMA (d) cache memory

Q.3 A system which has lot of crashes, data should be written to the disk using
(a) write-through (b) write-back
(c) both of the above (d) none of these

Q.4 Which memory unit has lowest access time?
(a) cache (b) registers
(c) magnetic disk (d) main memory

Q.5 Cache memory enhances
(a) memory capacity
(b) memory access time
(c) effective memory access time
(d) secondary storage access time

Q.6 Cache memory
(a) has greater capacity than RAM
(b) is faster to access than CPU registers
(c) is permanent storage
(d) faster to access than RAM

Q.7 The average memory access time for a machine with a cache hit rate of 90% where the cache access time is 10 ns and the memory access time is 100 ns is
(a) 55 ns (b) 45 ns
(c) 90 ns (d) 19 ns

Q.8 Which of the following lists memory types from highest to lowest speed?
(a) secondary storage, main memory, cache, registers
(b) registers, cache, secondary storage, main memory
(c) registers, cache, main memory, secondary storage
(d) cache, registers, main memory, secondary storage

Q.9 _____ improve system performance by temporarily storing data during transfers between devices or processes that operate at different speeds.
(a) caches (b) controllers
(c) buffers (d) registers

Q.10 According to temporal locality, processes are likely to reference pages that _____.
(a) have been referenced recently
(b) are located at address near recently referenced pages in memory
(c) have been preloaded into memory
(d) none of these

Q.11 Given below are some statements associated with cache memory. Identify the correct statement.
(a) The Level 1 cache is always faster than the Level 2 cache.
(b) The Level 2 cache is used to mitigate the dynamic slowdown every time a Level 1 cache miss occurs.
(c) Level 2 cache comes as on board only.
(d) In modern day computers, the Level 2 cache is considered an internal cache.

Q.12 Given below are some statements associated with computer memory. Identify the correct statement.
(a) Dynamic random access memory is faster than static random access memory.
(b) Main memory is used by the processor to store primary active programs and data.
(c) The cache memory is of dynamic random access memory.
(d) The production cost of dynamic random access memory is higher than that of static random access memory.

Q.13 In caching system, the memory reference made in any short time interval tend to use only a small fraction of the total memory is called _____.
(a) checker boarding (b) locality principle
(c) memory interleaving (d) None of these

Q.14 Which of the following statements is false about dynamic RAM?
(a) Power consumption of dynamic RAMs is more than that of static RAMs.
(b) Gate density of dynamic RAMs is higher than that of static RAMs.
(c) Dynamic RAMs require a memory refresh cycle after each memory read cycle.
(d) Main memory of a general purpose computer system is usually built using dynamic RAMs.

Q.15 Which of the following statements is false with regard to fully associative and direct mapped cache organizations?
(a) Direct mapped caches do not need a cache block replacement policy, whereas fully associative caches do.
(b) In a fully associative cache, the full address is stored alongside the data block.
(c) A direct mapped cache is organized according to an 'index', and the 'tag' part of the memory address is stored alongside the data block.
(d) Direct mapped caches may produce more misses if programs refer to memory words that occupy a single tag value.

Q.16 In a two level memory hierarchy, the access time of the cache memory is 12 nanoseconds and the access time of the main memory is 1.5 microseconds. The hit ratio is 0.98. What is the average access time of the two level memory system?
(a) 13.5 nsec (b) 42 nsec
(c) 7.56 nsec (d) 41.76 nsec

Q.17 Determine True(T)/False(F) of the following statements.
1. Dynamic RAM consists of internal flip-flops.
2. Static RAM consists of capacitors.
3. Static RAM is easier to use than dynamic RAM.
4. Static RAM has shorter read and write cycles than that of dynamic RAM.

(a) F F T F (b) T T T T
(c) F F T T (d) T T F F

Q.18 Consider the following organization of main memory and cache memory.
Main memory: 64K × 16
Cache memory: 256 × 16
Memory is word-addressable and block size is of 8 words. Determine the size of tag field if direct mapping is used for transforming data from main memory to cache memory.
(a) 5 bits (b) 6 bits
(c) 7 bits (d) 8 bits

Q.19 The minimum time delay between the initiations of two independent memory operations is called
(a) access time (b) cycle time
(c) transfer rate (d) latency time

Q.20 Match **List-I** with **List-II** using memory hierarchy from top to down and select the correct answer using the codes given below the lists:

List-I	List-II
A. Size	1. Increases
B. Cost/bit	2. Decreases
C. Access time	3. No change
D. Frequency of access	

Codes:

	A	B	C	D
(a)	1	1	2	2
(b)	2	1	2	1
(c)	2	2	1	1
(d)	1	2	1	2

Q.21 A computer system has 4K-word cache organized in a block-set-associative manner, with 4 blocks per set, 64 words per block. Memory is word addressable. The number of bits in the SET and WORD fields of the main memory address format is
(a) 15, 4 (b) 6, 4
(c) 7, 6 (d) 4, 6

Q.22 If the memory clip size is 256 × 1 bits, then how many clips are required to make up 1 KB of memory?

(a) 8 (b) 16
(c) 24 (d) 32

Q.23 Three memory chips are of size 1 kB, 2 kB and 4 kB. Their address bus is 10 bits. What are the data bus sizes of the chips?

(a) 8 bits, 16 bits and 24 bits respectively
(b) 8 bits, 16 bits and 32 bits respectively
(c) 8 bits, 16 bits and 64 bits respectively
(d) 8 bits, 16 bits and 128 bits respectively

Q.24 A memory system of size 16 Kbytes is required to be designed using memory chips which have 12 address lines and 4 data lines each. What is the number of such chips required to design the memory system?

(a) 2 (b) 4
(c) 8 (d) 16

Q.25 A memory system of size 32 kbytes is required to be designed using memory chips which have 12 address lines and 4 data lines each. What is the number of such chips required to design the memory system?

(a) 4 (b) 8
(c) 16 (d) 32

Q.26 A 3 × 8 decoder with two enable inputs is to be used to address 8 blocks of memory. What will be the size of each memory block when addressed from a sixteen bit bus with two MSBs used to enable the decoder ?

(a) 2 K (b) 4 K
(c) 16 K (d) 64 K

Q.27 Match **List-I** (Type of Memory) with **List-II** (Used as) and select the correct answer using the code given below the lists:

List-I
A. SRAM
B. DRAM
C. Parallel Access Registers
D. ROM

List-II
1. Cache memory
2. Main memory
3. BIOS memory
4. CPU registers

Code:

	A	B	C	D
(a)	1	4	2	3
(b)	3	4	2	1
(c)	1	2	4	3
(d)	3	2	4	1

Q.28 Which of the following are the memory performance parameters ?

1. Access time and latency
2. Block size and Block access time
3. Cycle time and Bandwidth

Select the correct answer from the codes given below:

(a) 1 only (b) 1 and 2 only
(c) 2 and 3 only (d) 1, 2 and 3

Answer Key:

1. (c)	**2.** (b)	**3.** (a)	**4.** (b)	**5.** (c)
6. (d)	**7.** (d)	**8.** (c)	**9.** (c)	**10.** (a)
11. (b)	**12.** (b)	**13.** (b)	**14.** (a)	**15.** (d)
16. (d)	**17.** (c)	**18.** (d)	**19.** (b)	**20.** (d)
21. (d)	**22.** (d)	**23.** (b)	**24.** (c)	**25.** (c)
26. (a)	**27.** (c)	**28.** (d)		

CHAPTER 05

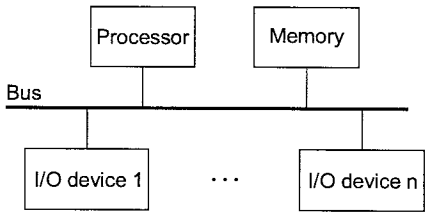
Input-Output and Secondary Storage

5.1 Interface Design

The computer system's I/O architecture is its interface to the outside world. This architecture is designed to provide a systematic means of controlling interaction with the outside world and to provide the operating system with the information it needs to manage I/O activity effectively.

5.1.1 I/O Access Structure

Most modern computers use single bus arrangement for connecting I/O devices to CPU and Memory Single-bus structure enables all the devices connected to it to exchange information (as shown in the following Figure).



Single bus structure for I/O Access

The bus consists of three sets of lines used to carry address, data, and control signals.

- Processor places a particular address on address lines. Each I/O device is assigned a unique set of addresses.
- Device which recognizes this address responds to the commands issued on the Control lines.
- Processor requests for either Read / Write: The data will be placed on Data lines.

Remember

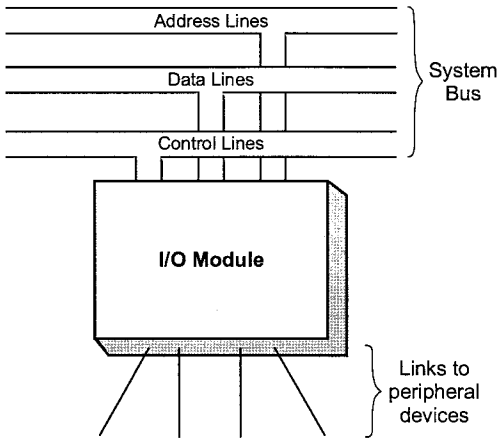
Why I/O Access is Important?

- There are a wide variety of peripherals each with varying methods of operation.
- Delivering different amounts of data, at different speeds, in different formats.
- Data transfer rates are often slower than the processor and/or memory
- Data transfer rates may be faster than that of the processor and/or memory
- Peripheral often use different data formats and word lengths (vary from CPU)
- Many are not connected directly to system or expansion bus.

5.1.2 I/O Modules

Peripheral communications are handled with I/O modules. I/O module has the following two function:

- Interface to the processor and memory via the system bus or control links
- Interface to one or more peripheral devices via data links

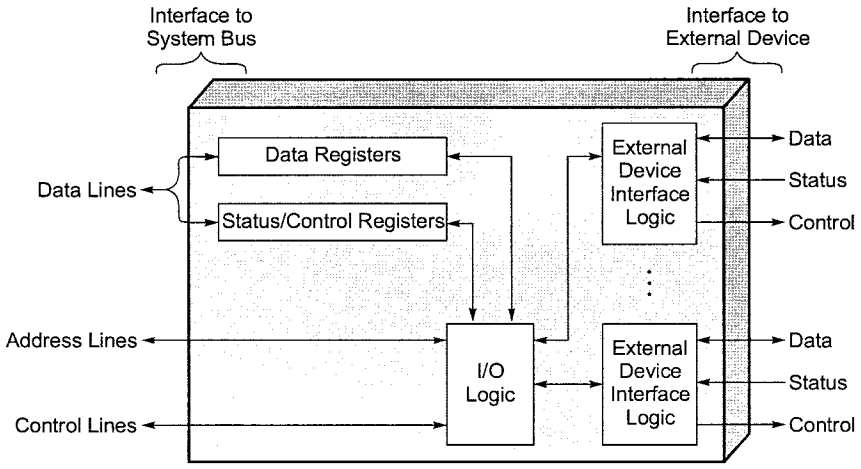


5.1.3 Functions of I/O Module

1. **Control and timing:** Coordination of traffic between internal resources and external devices can be done by the following sequence of steps:
   - Processor checks I/O module for external device status
   - I/O module returns status
   - If device ready, processor gives I/O module command to request data transfer
   - I/O module gets a unit of data from device
   - Data transferred from the I/O module to the processor
2. **Processor communication:** It involves the following.
   - **Command decoding:** I/O module accepts commands from the processor sent as signals on the control bus
   - **Data:** data exchanged between the processor and I/O module over the data bus
   - **Status reporting:** common status signals BUSY and READY are used because peripherals are slow
   - **Address recognition:** I/O module must recognize a unique address for each peripheral that it controls.
3. **Device (I/O Module) communication:**
   - *Device communication:* Commands, status information, and data
   - *Data buffering:* Data comes from main memory in rapid burst and must be buffered by the I/O module and then sent to the device at the device's rate
   - *Error detection:* Responsible for reporting errors to the processor
   - Handles the speed mismatch between memory and the device

5.1.4 Structure of I/O Module

Block Diagram of an I/O Module



Module connects to the computer through a set of signal lines (system bus). I/O module may hide device details from the processor so the processor only functions in terms of simple read and write operations. I/O module may leave much of the work of controlling a device visible to the processor.

- **Data (Buffer) registers:** Data transferred to and from the module are buffered with data registers. DataIn and DataOut Registers may used for transfer.
- **Status/Control Registers:** Status and control provided through status or control registers.
- **I/O Logic:** Module logic interacts with processor via a set of control signal lines. Processor uses control signal lines to issue commands to the I/O module logic.
- **External Device Interface Logic:** Module must recognize and generate addresses for devices it controls. Module contains logic for device interfaces to the devices it controls.

Examples of I/O Modules

- **I/O channel or I/O processor:** I/O module that takes on most of the detailed processing burden. It is used on mainframe computers.
- **I/O controller of device controller:** Primitive I/O module that requires detailed control. It is used on microcomputers.

5.1.5 Types of Interrupts

Hardware Interrupts

This interrupt present at the hardware pins of the processor hardware interrupt may be an external interrupts or internal interrupts.

(a) External interrupts generated by externally connected component likes:
Power source, Various I/O devices

(b) Internal interrupts generated by the internal components of the processor that is:
Stack overflow, invalid opcode, divide byzero, temperature sensors etc.

Software Interrupts

It is an instruction defined in the instruction set of processor.

Maskable Interrupts

This interrupt is enable or disable based on the signals. All the low priority I/O device connected to the maskable interrupt pins.

Non-Maskable Interrupts

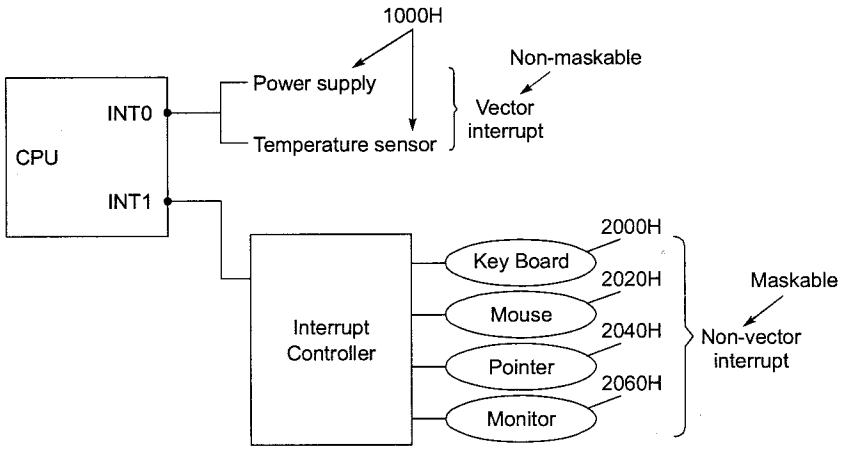
This interrupt is always present in enable state so high priority components are connected to the non-maskable interrupt pins.

Vector Interrupt

This interrupt contain fixed vector points in the memory. (Static, vector address). (This is for high interrupt)

Non-Vector Interrupt

This interrupt contain dynamic vector points in which source supplies the vector address. In above configuration, when the CPU encounter the interrupts at INTO pin, then the program counter is always loaded with the fixed (1000H) address. So INTO acts as the vector interrupts. When the CPU recognize the interrupt at INT1 pin then the source will supply the vector address. Therefore INT1 act as the non-vector interrupt.

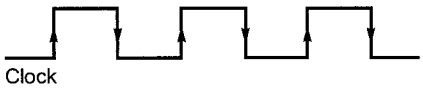


Level Triggered Interrupt

This interrupt is enabled based on the levels of clock signal.

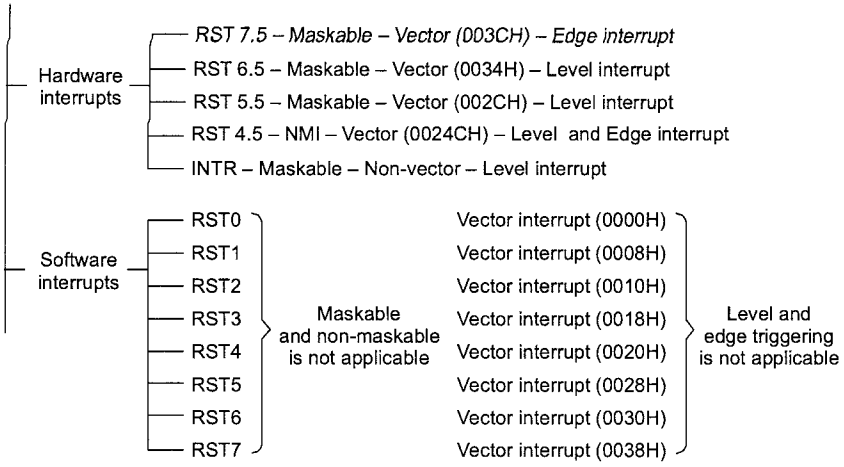
Edge Triggered Interrupt

This interrupt is enable based on the raising edge or falling edge transition of the clock pulse.



5.1.6 Interrupt Structure

To analyse the various interrupt implementation process. Let us consider 8085 interrupt structures



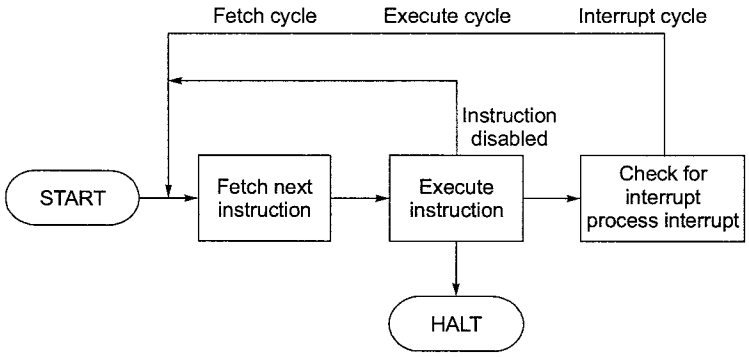
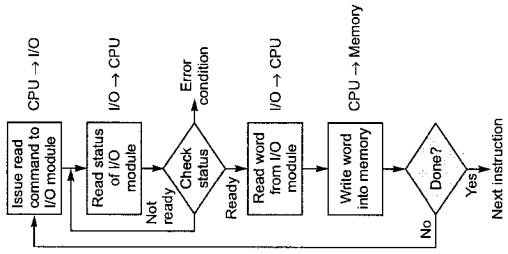


Figure : *Instruction cycle with interrupts*

5.2 Input-Output Mode

5.2.1 Input-Output Technics

Programmed I/O refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device.



Programmed I/O basically works in the following way:

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits in status register (I/O module does not inform and interrupt CPU directly)
- CPU checks status bits periodically
- CPU may wait or come back later

When CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time, while waiting, must repeatedly check the status of the I/O module, and this process is known as **Polling**. As a result, the level of the performance of the entire system is severely degraded.

I/O Commands

The processor issues an address, specifying I/O module and device, and an I/O command.
The commands are:

- **Control:** activate a peripheral and tell it what to do.
- **Test:** test various status conditions associated with an I/O module and its peripherals.
- **Read:** causes the I/O module to obtain an item of data from the peripheral and place it into an internal register.
- **Write:** causes the I/O module to take a unit of data from the data bus and transmit it to the peripheral.

I/O Instructions

CPU views I/O data transfer in a similar manner as memory access. Each device is given a unique identifier or address. Processor issues commands containing device address. I/O module must check address lines to see if the command is for itself. I/O Mapping can be done in either memory-mapped I/O or isolated I/O.

I/O Mapping Techniques

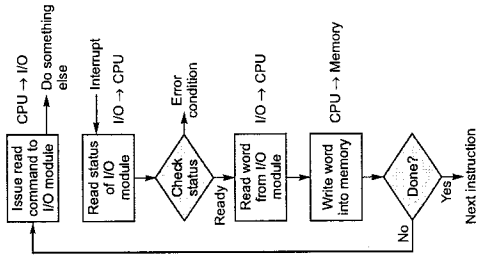
1. **Memory-mapped I/O:** I/O devices and the memory share the same address space.
   - I/O module registers treated as memory addresses.
   - Same machine instructions used to access both memory and I/O devices.
   - It allows for more efficient programming.
   - Single read line and single write lines needed.
2. **Isolated I/O:** Separate address space for both memory and I/O devices.
   - Separate memory and I/O select lines needed.
   - Limited set of I/O instructions.
   - Special instructions for I/O is needed.

In programmed control I/O, CPU has direct control over I/O (Sensing status, Read/write commands, Transferring data). CPU waits for I/O module to complete operation. This technique wastes CPU time by polling.

Interrupt-Driven I/O

The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work. Data transfer from I/O to CPU has following operations:

- CPU issues read command
- I/O module gets data from peripheral while the CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data



I/O Module View:

- I/O module receives a READ command form the processor
- I/O module reads data from desired peripheral into data register
- I/O module interrupts the processor
- I/O module waits until data is requested by the processor
- I/O module places data on the data bus when requested

Processor View:

- The processor issues a READ command
- The processor performs some other useful work
- The processor checks for interrupts at the end of the instruction cycle
- The processor saves the current context when interrupted by the I/O module
- The processor read the data from the I/O module and stores it in memory
- The processor the restores the saved context and resumes execution

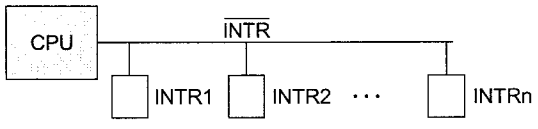
When I/O Device is ready, it sends the INTERRUPT signal (INTR) to processor via a dedicated controller line. Using interrupt we are ideally eliminating WAIT period. (Avoids repeated checking of device by CPU). In response to the interrupt, the processor executes the Interrupt Service Routine (ISR). All the registers , flags, program counter values are saved by the processor before running ISR. The time required to save status and restore contribute to execution overhead "Interrupt Latency".

Interrupt-Acknowledge Signal (INTA): I/O device interface accomplishes this by execution of an instruction in the ISR that accesses a status or data register in the device interface; implicitly informs the device that its interrupt request (IRQ) has been recognized. IRQ signal is then removed by device.

Handling Multiple Interrupts

Design issues: *How does the processor determine which device issued the interrupt?*
How are multiple interrupts dealt with Device identification?

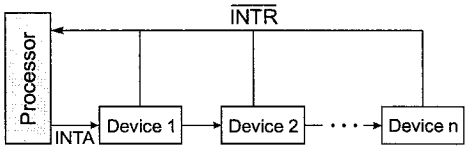
- **Multiple interrupt lines:** Different line for each module but lines can be shared between devices. It limits number of devices. Each line may use one of the following techniques. CPU picks line with highest priority.
- **Software polling:** The IRQ (interrupt request) bit in the status register is set when a device is requesting an interrupt. The ISR polls the I/O devices connected to the bus. The first device encountered with the IRQ bit set is serviced and the subroutine is invoked.
  Easy to implement, but too much time spent on checking the IRQ bits of all devices, though some devices may not be requesting service. Polling order determines priority but disadvantage is the time spent interrogating all the devices.
- **Vectored Interrupts:** A device requesting an interrupt may identify by CPU directly itself and then CPU starts executing the corresponding ISR. An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device.



Mechanism with Common INTR signal

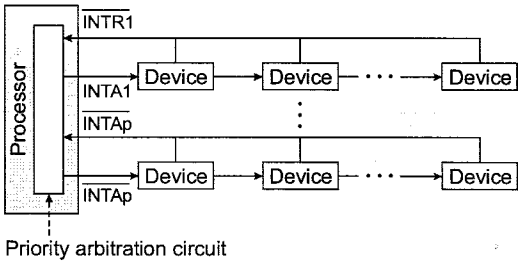
Requesting I/O module places a word of data on the data lines which is called as vector that uniquely identifies the I/O module called vectored interrupt. Disadvantage is common interrupt request signal (INTR) cannot identify the simultaneous requests by devices (shown in the above Figure).

- **Priority Implementation of Vectored Interrupts:** Consider the problem of simultaneous arrivals of interrupt requests from two or more devices. The processor must have some means of deciding which request to service first.
  1. **Daisy Chain:** Daisy chain vectored interrupts: It has common interrupt request line (INTR) but all the devices are connected in chain form based on their priority (shown in the following figure).



CPU send interrupt acknowledge (INTA) to initiate transfer through devices. Daisy chain order of the modules determines priority. Low priority device may have starvation.

  2. **Bus arbitration:** It is implemented with the combination of Daisy chain scheme and with individual interrupt request and interrupt-acknowledge signals (as shown in the following figure). Each group has different priority levels and within each group devices are connected in daisy chain fashion. Arbitration scheme determines priority.



- I/O module first gains control of the bus
- I/O module sends interrupt request
- The processor acknowledges the interrupt request using arbitration circuit.
- I/O module places its vector of the data lines

Disadvantages of Interrupt Driven I/O

Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory. I/O transfer rate limited to speed that processor can test and service devices.

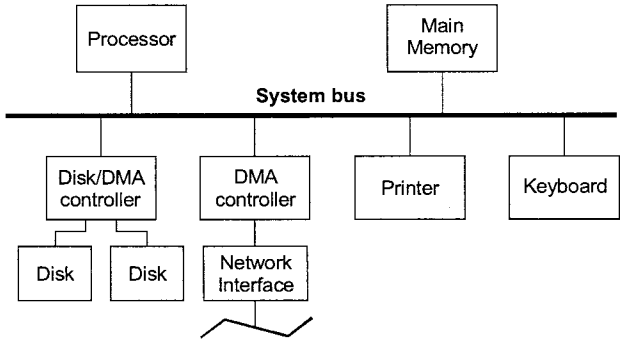
NOTE: Both Programmed and Interrupt driven I/O need CPU intervention to transfer the data, which can be avoided using DMA intervention.

5.2.2 Direct Memory Access

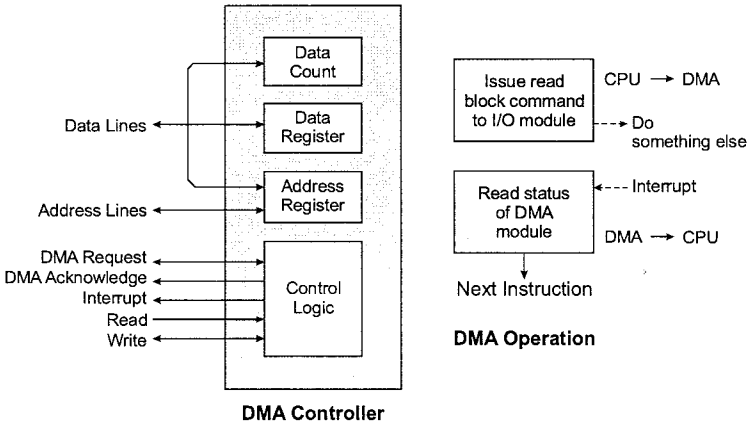
- CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module controls exchange of data between main memory and the I/O device.
- CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.
- DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses.

DMA Controller

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus (shown in the following figure).



The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

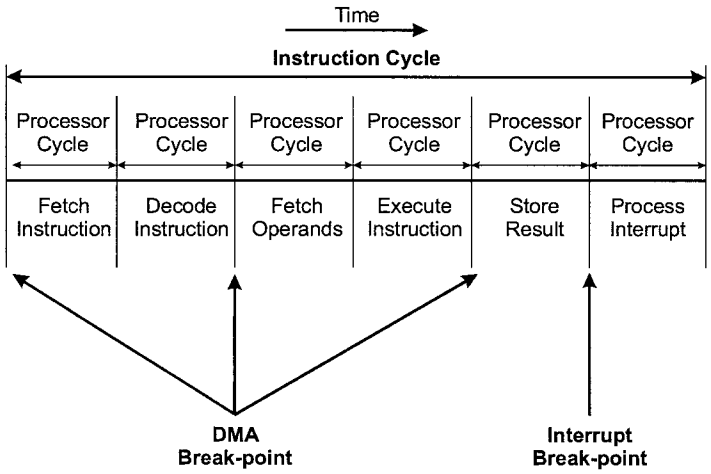


DMA Data Transfer (DMA Operation): Data transfer using DMA takes place in the following three steps:

1. When the processor has to transfer data it issues a command to the DMA controller with the following information:
   - The starting address of the memory block
   - Address of I/O device
   - *The word count:* Size of data to be transferred.
   - Control to specify the mode of transfer such as read or write
   - A control to start the DMA transfer

   After this, the processor becomes free and it may be able to perform other tasks.
2. In this step the entire block of data is transferred directly to or from memory by the DMA controller.
3. In this, at the end of the transfer, the DMA controller informs the processor by sending an interrupt signal.

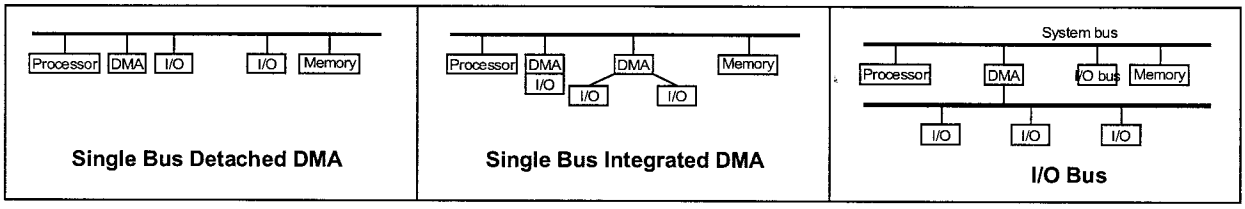
DMA and Interrupt Breakpoints during Instruction Cycle:



Instructions Cycle Vs CPU (Processor) Cycle

The above figure shows that the CPU suspends or pauses for one bus cycle when it needs a bus cycle, transfers the data and then returns the control back to the CPU. DMA requests can be handled during DMA breakpoints. The processor is suspended just before it needs to use the bus. The DMA module transfers one word and returns control to the processor. Since this is not an interrupt the processor does not have to save context. The processor executes more slowly, but this is still far more efficient that either programmed or interrupt-driven I/O.

DMA Configurations



DMA Configurations

Single Bus Detached DMA	Single Bus Integrated DMA	DMA with separate I/O Bus
It support only one device	It can support more than one device	It supports all DMA enabled devices
Each transfer uses bus twice (I/O to DMA, DMA to memory)	Each transfer uses bus once (DMA to memory)	Each transfer uses bus once (DMA to memory)
Processor suspended twice	Processor suspended once	Processor suspended once

NOTE: More than one DMA controller may be used to handle I/O devices. Bus arbitration is used to resolve the simultaneous access by more than one DMA.

Types of DMA Transfers

1. **One step DMA transfer:** DMA controllers can transfer data in a two-step process by reading a value from one port or address in one bus cycle and writing that value to another port or address in a second bus cycle.

2. **Two step DMA transfer:** It is also possible for the DMA controller to carry out read and write operations simultaneously. In this case the data is transferred directly between the I/O device and memory in the same bus cycle.

Types of DMA Transfer Modes

1. **Burst or Block transfer DMA:** In Burst mode, once the DMA controller takes the control of the system bus, it transfers all bytes of data block (without interruption) before releasing control of the system bus back to the processor.
   - Size of block is predetermined in the initialization process by the CPU.
   - Burst mode is useful for loading programs or data files into memory, because CPU needs these data to continue its work.

   Burst Mode Operations: Transferring a block from I/O to CPU using DMA Burst mode has the following sequence of operations:
   - DMA controller will send HOLD (for I/O to CPU transfer)
   - CPU suspends its operations and releases the system bus and then generates HLDA (Acknowledgement for HOLD)
   - Now DMA controller acquires bus system and starts transferring the data.
   - For each byte of data transfer, count register is decremented by one. If count register is reaches to zero then stops transfer.
   - DMA controller drops HOLD, CPU gains control of system bus and resumes execution.

   Disadvantage: CPU may be inactive for long time. CPU does no operation during DMA bust mode data transfer.
2. **Cycle Stealing (Single or byte or single byte) transfer DMA:** In Cycle stealing mode, once the DMA controller takes the control of the system bus, it transfers only one byte of data (without interruption) before releasing control of the system bus back to the processor.
   - DMA Controller "steals" memory cycles from processor, though processor originates most memory access.
   - To transfer one block from I/O, DMA issues requests continuously for each byte until it has transferred the entire block.

   Cycle Stealing Mode Operations: Transferring a block from I/O to CPU using DMA Burst mode has the following sequence of operations:
   - DMA controller will send HOLD (for I/O to CPU transfer)
   - CPU suspends its operations and releases the system bus and then generates HLDA (Acknowledgement for HOLD)
   - Now DMA controller acquires bus system and transfers one byte.
   - DMA controller drops HOLD, CPU gains control of system bus and resumes execution.
   - If count register is greater than zero and next byte is ready to transfer then repeat above steps from starting until count register reaches to zero.

   Advantage: CPU need not inactive for long time.

   Disadvantage: CPU is idle during DMA transfer. i.e. CPU stops the execution when the DMA has control over system bus.
3. **Demand (Continuous/Complete data) Transfer DMA:** In Demand transfer mode, once the DMA controller takes the control of the system bus, it transfers entire data (without interruption) before releasing control of the system bus back to the processor.

   It can transfer all bytes of data transferred without interruption.

   It is similar to the Burst mode transfer, but demand transfer DMA will take the control until all bytes of data transferred instead of data block transfer.

4. **Transparent (Hidden) DMA:** In Transparent mode, DMA controller monitors the CPU and system bus. If CPU is not using the system bus during any cycle then DMA controller uses the system bus.
   - CPU may access the memory in the instruction fetch, operand fetch and operand write cycles.
   - In Decode and Execution cycles, the CPU can operate without accessing the memory in parallel with DMA controller.

   Advantage: In Transparent mode, CPU is not slowed down during DMA transfer. i.e. CPU does useful work during DMA transfer. Whereas in Burst, Cycle stealing and Demand modes, CPU is interrupted to use the system bus by DMA controller.

   Disadvantages: DMA takes longer than other modes to transfer the data block. Hardware needed to determine when the CPU is not using the system bus.

Remember

- The choice of operating mode depends on the speed at which data is arriving relative to the bus bandwidth and whether a particular application will allow the CPU to be locked off the bus for the duration of one transfer.
- The choice of operating mode depends on the speed at which data is arriving relative to the bus bandwidth and whether a particular application will allow the CPU to be locked off the bus for the duration of one transfer.
- Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

Advantage of DMA: Higher transfer rates (approaching that of the memory) can be achieved.
Disadvantage of DMA: A DMA Controller, or a DMAC, is needed, making the system complex and expensive.

***Example*-5.1** An I/O device transfers data at a rate of 10MB/s over a 100MB/s bus. The data is transferred in 4KB blocks. If the processor operates at 500MHz, and it takes a total of 5000 cycles to handle each DMA request, find the fraction of CPU time handling the data transfer with and without DMA.

Solution:

Without DMA: The processor here copies the data into memory as it is sent over the bus.
Since the I/O device sends data at a rate of 10MB/s over the 100MB/s bus, 10 % of each second is spent transferring data. Therefore 10% of the CPU time is spent copying data to memory.
With DMA: Time required in handling each DMA request is 5000 cycles.
Since 2500 DMA requests are issued (10 MB / 4 KB) the total time taken is 12,500,000 cycles.
As the CPU clock is 500MHZ, the fraction of CPU time spent is $12,500,000/(500 \times 10^6)$ or 2.5%.

***Example*-5.2** A hard drive with a maximum transfer rate of 1Mbyte/sec is connected to a 32-bit, 10MIPS CPU operating at a clock frequency of 100 MHz. Assume that the I/O interface is DMA based and it takes 500 clock cycles for the CPU to set-up the DMA controller. Also assume that the interrupt handling process at the end of the DMA transfer takes an additional 300 CPU clock cycles. If the data transfer is done using 2 KB blocks, calculate the percentage of the CPU time consumed in handling the hard drive.

Note: The DMA controller is the only device accessing the memory. If the CPU also tries to access memory, then either the DMAC or the CPU will have to wait while the other one is actively accessing the memory.

Solution:

Since the hard drive transfers at 1MB/sec, and each block size is 2KB. So there are 1000/2 = 500 blocks transferred/sec.

Every DMA transfer uses 500 + 300 = 800 CPU cycles.

This gives us 800x500 = 400,000 = 400×10^3 cycles/sec

For the 100 MHz CPU, this corresponds to $(400 \times 10^3) / (100 \times 10^6) = 4 \times 10^{-3} = 0.4\%$

This would be the case when the hard drive is transferring data all the time.

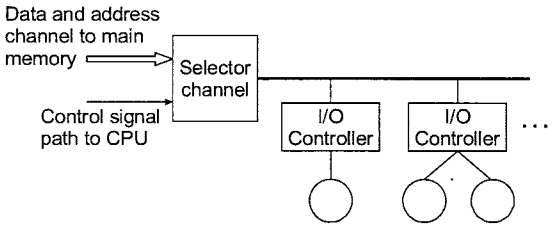
In actual situation, drive will not be active all the time, and this number will be much smaller than 0.4%.

5.2.3 I/O Channels

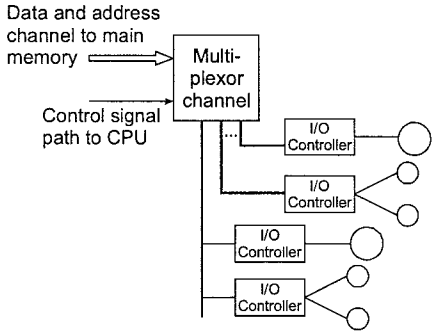
- I/O Channel is an extension of the DMA concept
- It has ability to execute I/O instructions using special-purpose processor on I/O channel and complete control over I/O operations.
- Processor does not execute I/O instructions itself. Processor initiates I/O transfer by instructing the I/O channel to execute a program in memory.
- Program specifies: Device or devices, Area or areas of memory, Priority, and Error condition actions

Types of I/O Channels

1. **Selector channel:**
   - Controls multiple high-speed devices.
   - Dedicated to the transfer of data with one of the devices.
   - Each device handled by a controller, or I/O module.
   - I/O channel controls these I/O controllers (shown in the following Figure).



2. **Multiplexer channel:** It is DMA controller that can handle multiple devices at the same time (as shown in the following Figure). i.e. It can do block transfers for several devices at once.

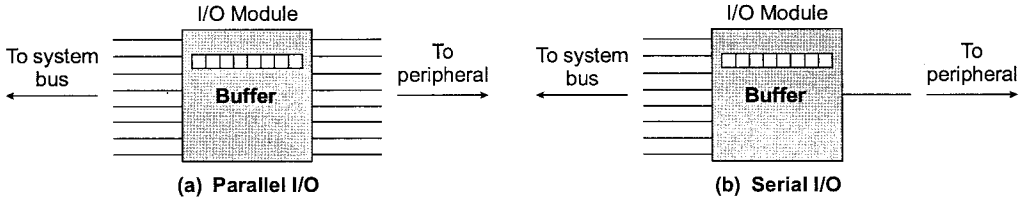


There are two types of multiplexors:

1. **Byte multiplexer:** It is used for low-speed devices. It accepts or transmits characters. Interleaves bytes from several devices.
2. **Block multiplexer:** Block multiplexer accepts or transmits block of characters. Interleaves blocks of bytes from several devices. Used for high-speed devices.

Type of Interfaces

1. **Parallel interface:** Multiple bits are transferred simultaneously.
2. **Serial interface:** Bits transferred one at a time.



I/O Module Vs I/O Device

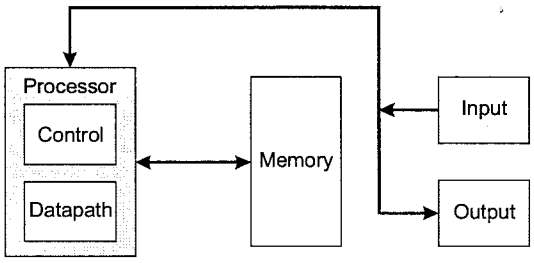
The following sequence of operations performed for a write operation:

1. I/O module sends control signal for requesting permission to send data
2. I/O device acknowledges the request
3. I/O module transfer data
4. I/O device acknowledges receipt of data

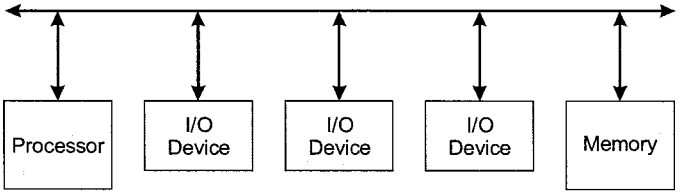
5.2.4 Buses

Bus: Bus is connecting I/O to Processor and Memory

- A bus is a shared communication link.
- It uses one set of wires to connect multiple subsystems (shown in the following Figure).



Sometimes shared bus with memory and sometimes a separate I/O bus is used as shown below.



Advantages of Bus: Versatility, New devices can be added easily, Peripherals can be moved between computer systems that use the same bus standard, Low cost, and A single set of wires is shared in multiple ways.

Disadvantages of Bus:

- It creates a communication bottleneck
- The bandwidth of that bus can limit the maximum I/O throughput
- The maximum bus speed is largely limited by the length of the bus and the number of devices on the bus
- The need to support a range of devices with: (a) Widely varying latencies (b) Widely varying data transfer rates

Synchronous and Asynchronous Bus

Synchronous Bus:

- Includes a clock in the control lines
- A fixed protocol for communication that is relative to the clock

Advantage

Involves very little logic and can run very fast.

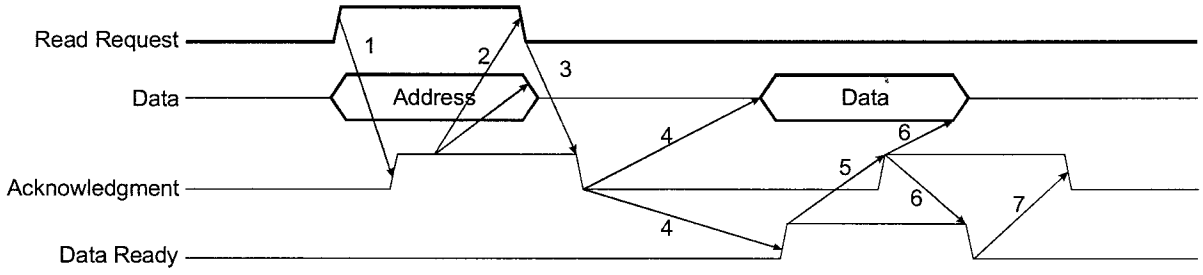
Disadvantages

Every device on the bus must run at the same clock rate. To avoid clock skew, they cannot be long if they are fast.

Asynchronous Bus:

- It is not clocked
- It can accommodate a wide range of devices
- It can be lengthened without worrying about clock skew
- It requires a handshaking protocol

A Handshaking Protocol



Three control lines are used in the above Handshaking diagram:

1. **Read Request:** Memory address is put on the data lines at the same time
2. **Data Ready:** Indicate the data word is now ready on the data lines data is put on the data lines at the same time.
3. **Acknowledgement:** Acknowledge the Read Request or the Data Ready of the other party.

Increasing the Bus Bandwidth

Data bus width: By increasing the width of the data bus, transfers of multiple words require fewer bus cycles. *Cost:* More bus lines needed.

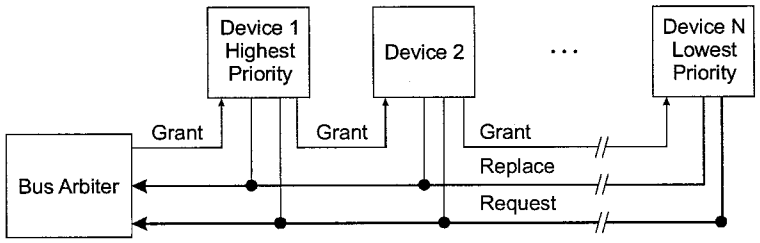
Block transfers: Allow the bus to transfer multiple words in back-to-back bus cycles. Only one address needs to be sent at the beginning. The bus is not released until the last word is transferred. *Cost:* (a) Increased complexity (b) Decreased response time for request.

Bus Arbitration

Any device that can control the bus is called a bus master

- Bus arbitration scheme:
- A bus master wanting to use the bus asserts the bus request
- A bus master cannot use the bus until its request is granted
- A bus master must signal to the arbiter after it has finished using the bus
- Bus arbitration schemes usually try to balance two factors:
- **Bus priority:** The highest priority device should be serviced first
- **Fairness:** Even the lowest priority device should never be completely locked out from the bus

The Daisy Chain Bus Arbitrations Scheme:



Advantage: Simple technique.

Disadvantages: Cannot assure fairness, a low-priority device may be locked out indefinitely. The use of the daisy chain grant signal also limits the bus speed.

5.2.5 I/O Device Commands

Two methods are used to address the device: Special I/O instructions and Memory-mapped I/O.

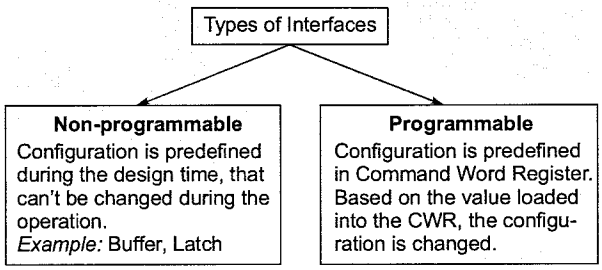
- **Special I/O instructions:** Specifies both the device number and the command word.
  (a) *Device number:* The processor communicates this via a set of wires normally included as part of the I/O bus.
  (b) *Command word:* This is usually send on the bus.
- **Memory-mapped I/O:** Portions of the address space are assigned to I/O device. Read and writes to those addresses are interpreted as commands to the I/O devices.

5.2.6 Notifying the OS

- The OS needs to know when:
- The I/O device has completed an operation
- The I/O operation has encountered an error
- This can be accomplished in two different ways:
  (a) **Polling:** The I/O device put information in a status register. Operating System periodically check the status register.
  (b) **I/O Interrupt:** Whenever an I/O device needs attention from the processor, it interrupts the processor.

Remember

- Types of I/O interfaces:



- During the DMA operation, the CPU is in two states: Busy State and Block/Hold State.
- Until preparing the data, the CPU is busy with other executions. While transferring the data, the CPU is in blocked state.

  Let 'X' = Preparation time (Depends on I/O speed), 'Y' = Transfer time (Depends on main memory speed).

  Then, Percentage of time CPU is blocked = $\left(\frac{Y}{X+Y}\right)\times 100$

  Percentage of time CPU is busy = $\left(\frac{X}{X+Y}\right)\times 100$

5.3 Secondary Memory

5.3.1 Secondary Storage

Storage: It is defined as all various media on which a computer can store programs and information.

Secondary Storage: It is also known as External memory / Storage memory / Auxiliary memory / Backup memory. It is used for storing data and instructions permanently and non-volatile. Example: Hard disk, CD, DVD, Pen drive, etc.

The Criteria for rating Secondary Storage Devices are:

- **Storage Capacity:** This is determined by the type of program or amount of data to be stored.
- **Access Speed:** This refers to the average time taken to locate data on a sec. storage device.
- **Transfer Rate:** This refers to the speed and time taken for data to be transferred from a secondary device to main memory in order for the data to be executed (processed).
- **Size:** This is determined by the expected amount of data to be stored.
- **Cost:** This is directly related to the other 4 factors. E.g. " floppy disk is cheaper than CD which is cheaper than a memory stick."

Secondary Storage devices may be classified by either its Access method or by its Technology.

- **Access Method:** It refers to the way in which the data is retrieved i.e. sequentially or directly.
- **Technology:** It refers to technology/ media used to retrieved the data i.e. magnetically or optically.

Characteristics of Auxiliary Memory

- **Non-volatile memory:** Data is not lost when power is cut off.
- **Capacity:** Secondary storage can store large volumes of data in sets of multiple disks.

- **Cost:** It is much lesser expensive to store data on a tape or disk than primary memory.
- **Reusable:** The data stays in the secondary storage on permanent basis until it is not overwritten or deleted by the user.
- **Reliable:** Data in secondary storage is safe because of high physical stability of secondary storage *device.*
- **Convenience:** With the help of a computer software, authorised people can locate and access the data quickly.

Difference between Primary Memory and Secondary Memory

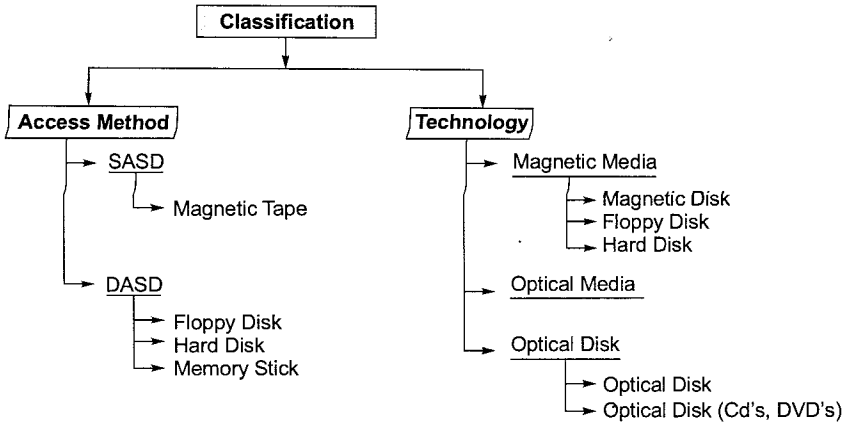
Primary memory	Secondary memory
Fast	Slow
Expensive	Cheap
Low capacity	Large capacity
Works directly with the processor	Not connected directly to the processor

Device Interface

SATA, SCSI, IDE, are all devices in the motherboard of a computer which provide an interface for devices such as internal hard disk. Drives and mass storage devices to connect to the computer.

- **IDE:** Intelligent Drive Electronics or Integrated Drive Electronics. An IDE interface is an interface for mass storage in which the controller is integrated into the disk or CD-ROM drive.
- **SATA:** Serial Advanced Technology Attachment. It is a computer bus primarily designed for transfer of data between a computer and mass storage devices (hard disk drives and optical drives).
- **SCSI:** Small Computer System Interface.

Classification of Storage Devices

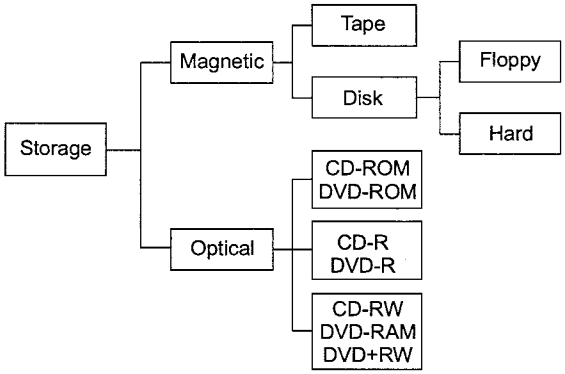


Access Methods

- **Sequential Access Storage:** This refers to reading or writing data consecutively; as the computer has to search the storage medium from beginning to end in order to find the required data.
- **Direct Access Storage (Random):** This is also called Random Access which means data can be located immediately without having to search consecutively through the storage medium.

Technology

- **Magnetic Media:** It reads data from the storage medium by magnetizing the iron particles on the medium. It is also cheap.
- **Optical Media:** It uses high powered laser beams to burn microscopic sports represented in a disk's surface. It is expensive.
- **Magnetic Disks:** Hard Disks (High capacity, low cost, per bit). Floppy Disks (Low capacity, slow, cheap).
- **Optical Disks:** CD-ROM = Compact Disk, Read-only memory (Read-only / write once, holds a lot of data, cheap reproduction).
- **Serial Devices:** Magnetic tapes (very fast sequential access)



Classification of Secondary Memory

5.3.2 Magnetic Memory

Magnetic memory uses the property of magnet for storing data. Magnetic Drum, Magnetic tape and Magnetic disks are types of magnetic memory.

Magnetic tape

Magnetic tape contains thin plastic ribbon. In magnetic tape only one side of the ribbon is used for storing data. The data storing side is coated by magnetic oxide. It is a sequential access memory. So, the data read/write speed is slower. It is mainly used for storing audio, video and back-up data.

It is highly reliable. It requires magnetic tape drive for reading and writing data. It has the storage capacity of 100 MB - 200 GB. The width of the ribbon also varies from 4 mm - 1 inch.

Advantages of Magnetic Tape

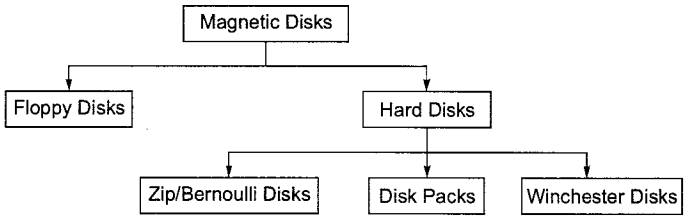
- Provides archival or backup storage
- They are reusable, light, compact and easy to store on racks
- They are inexpensive
- They can be used for large files or copying from disk files.

Disadvantages of Magnetic Tape

- Allows only sequential data access
- Stored data cannot be easily updated
- They must be stored in a suitable environment, vulnerable humidity and dust.

Magnetic Disk

Magnetic disk contains a circular disc made of metal or plastic. Both side of the disc is usually used for storing data. The disc is coated by magnetic oxide. The disc is divided into multiple concentric circles known as track. Tracks are further divided into small area known as sectors. Data are stored in sectors.



Example: Hard disk, floppy disk, zip disk, super disk, Winchesterdisk, jaz disk.

Hard Disk

- It is used as main storage device of the computer. It uses 2-4 metallic disk (platter).
- The disk is usually made of aluminium.
- Both sides of the disk is used for storing data except the upper side of the uppermost disk and lower side of the lowermost disk.
- The data storing surface is coated by magnetic oxide.
- Each data storing surface contains separate read/write head. During data read/write process platter rotates at the rate of 3600-15000rpm.
- Hard disk is also known as hard drive because both data storing disk and data read/write components are combined together.
- It has the storage capacity of few mega bytes to tera byte.

Advantages:

- Stores and retrieves data at high speeds
- More storage capacity than many other storage media
- Storage is reliable
- Provides random data access

Disadvantages:

- Very delicate
- Not portable: Cannot be easily moved from place to place without complications
- Head crash can occur

Floppy Disk

It contains single plastic disk. Initially it was used as a main storage device but nowadays it is used for carrying data from one computer to another.

It requires floppy drive for its operation.

It is not reliable as hard disk because the data storing surface is exposed. So, it may be affected by dust particles and magnetic field.

Advantages:

- They allow to copy files from one computer to another
- They are quite cheap
- Widely used
- Light/weight portable
- Provides random data access

Disadvantages:

- They have a limited capacity (graphics files often don't fit on a disk)
- They are less care should be taken to prevent loss data
- Vulnerable to dirt
- Susceptible to viruses
- Data access relatively slow
- Data is easily damaged/corrupted

Comparison between Magnetic Disk and Magnetic Tape

Attribute	Magnetic Tape	Magnetic Disk
Access	Sequential	Direct or Sequential
Access time	More	Less
Environmental restrictions	More	Less
Data trnasfer rate	Comparatively less	More
Reliability	Less	More
Portability	More	Less
Cost	Less	High
Application	Used for backups	Used as a secondary storage device in computer system

5.3.3 Optical Memory

Optical memory uses light beam for its operation. It is developed in fourth generation of computer. It is mainly used for storing audio/video, backup as well as for carrying data. It requires optical drive for its operation. Its read/write speed is slower compared to hard disk and flash memories. *Examples:* CD, DVD, BD.

CD (Compact Disk)

CD contains hard circular plastic, single side of this plastic is coated by aluminium alloy. This alloy stores data. It is protected by additional thin plastic covering. CD required CD drive for its operation. It has storage capacity of 700 MB or approximately 90 minutes of standard audio.

Types of CDs

- CD-R: It is a blank CD in which data can be stored once. After storing data it is converted into CD-ROM.
- CD-ROM: It cannot be erased or updated
- CD-RW: It can be erased and used for multiple times.

Advantages of CDs:

- Widely used
- Stored more data in less space
- Storage is extremely reliable
- Very durable
- Provides random data access
- It is entirely unaffected by magnetic fields

Disadvantages of CDs:

- Allows slower data access than a drive
- It is not easy to copy on optical disk
- Optical disk technology is expensive but prices are falling.

CD-ROM

A CD-ROM is a metal disc embedded into a plastic protective housing. Each disc has to be 'mastered' this is the process of creating the CD and placing the data on it. CDs are WORM (Write Once, Read Many) media, this refers to the fact once they have been mastered, there is no way to change the data on them.

Reading from a CD-ROM

1. A single track runs in a spiral pattern from the center of the disc to the outside, this track is made of pits and lands to represent the once and zeroes of binary data.
2. A low-powered laser is shone on the metallic surface and the reflection is captured in a photodiode sensor, the lands reflect differently to the pits, meaning it can tell the difference between a 1 and a 0.
3. The disc spins and the laser follows the track.
4. The binary data (the 1s and 0s) are put together and the CD-ROM has been read.

Advantages:

- Cheap
- Data cannot be written over by the consumer

Disadvantages:

- Slow seek time
- Data degrades with time, discs from 20 years ago might not work
- Can only be written to with a very high powered laser, which is not usually available in home computers
- Data cannot be written over

Comparison between CD ROM and Magnetic Disks

CD-ROM	Magnetic Disks
CLV = Constant Linear Velocity	CAV = Constant Angular Velocity
Sectors organized along a spiral	Sectors organized in concentric track
Sectors have same linear length (data packed at its maximum density permitted)	Sectors have same angular length (data written less densely in the outer traks)
Advantage: takes advantage of all storage space available	**Advantage:** operates on constant speed, timing marks to delimit tracks
Disadvantage: has to change rotational speed when seeking (slower towards the outside)	**Disadvantage:** it doesn't use up all storage available

CD-R

The CD-R is made of a reflective metal disk with a layer of (usually green, opaque) dye on top

Writing to a CD-R

1. A single track runs in a spiral pattern from the center of the disc to the side.
2. A high powered laser is shone onto the CD-R, changing the transparency (permanently) of the dye above. The transparent and opaque parts represent binary 1s and 0s.
3. The disc spins and the laser follows the track, putting the binary data onto the CD-R in a spiral track.
4. The data has been written.

Reading from a CD-R

1. A single track runs in a spiral pattern from the center of the disc to the outside, this track is made of pits and lands to represent the ones and zeroes of binary data.
2. A low-powered laser is shone on the surface and the reflection is captured in a photodiode sensor. The opaque dye will reflect differently to the transparent dye (which would just reflect the metal underneath it), meaning it can tell the difference between a 1 and 0.
3. The disc spins and the laser follows the track.
4. The binary data (the 1's and 0ss) are put together and the CD-R has been read.

Advantages:

- Cheap
- Can be written to using a conventional home computer

Disadvantages

- Slow seek time
- Data degrades with time, discs from 20 years ago might not wor!
- Data cannot be written over

CD-RW

The CD-RW is made of a reflective metal disk with a layer of a special (phase change) metal on top.

Writing to a CD-RW

1. A single track runs in a spiral pattern from the center of the disc to the outside.
2. A high powered laser is shone onto the CD-ROM. Depending on whether this is very high powered or heats at a slightly lower temperature, the top layer of metal cools differently. These will result in different amounts of reflectivity, which represent the 1s and 0s.
3. The disc spins and the laser follows the track, putting the binary data onto the CD in a spiral track.

Reading from a CD-RW

1. A single-track runs in a spiral pattern from the center of the disc to the outside, this track is made of pits and lands to represent the ones and zeroes of binary data.
2. A low powered laser is shone on the surface and the reflection is captured in a photodiode sensor. The different ways the metals has cooled reflect different amounts, meaning it can tell the difference between a 1 and 0.
3. The disc spins and the laser follows the track.
4. The binary data (the 1s and 0s) are put together and the CD-ROM has been read.

Advantages:

- It is cheaper.
- It can be written to using a conventional home computer.

Disadvantages:

- Slow seek time
- Data degrades with time, discs from 20 years ago might not work!
- Data can be changed after writing.
- Not all CD players (mostly older ones) can read CE-RWs, as opposed to CD-ROMs and CD-Rs.

DVD (Digital Versatile Disk)

Its shape and size is similar to CD but the difference in storage capacity is due to different chemical component and data is compressed before storing.

It requires DVD drive for its operation. It has the storage capacity of 4.7GB to 17GB. Read/write speed of DVD is slower than that of CD. Types of DVD are: DVD-R, DVD-ROM and DVD-RW

DVD can also be classified as:

- Single sided single layered DVD (4.7 GB)
- Single sided dual layered DVD (7-8 GB)

- Dual sided single layered DVD (9 GB)
- Dual sided dual layered DVD (17 GB)

Advantages:

- They can store large amounts of data.
- They are dependable.
- Provides random data access.

Disadvantages:

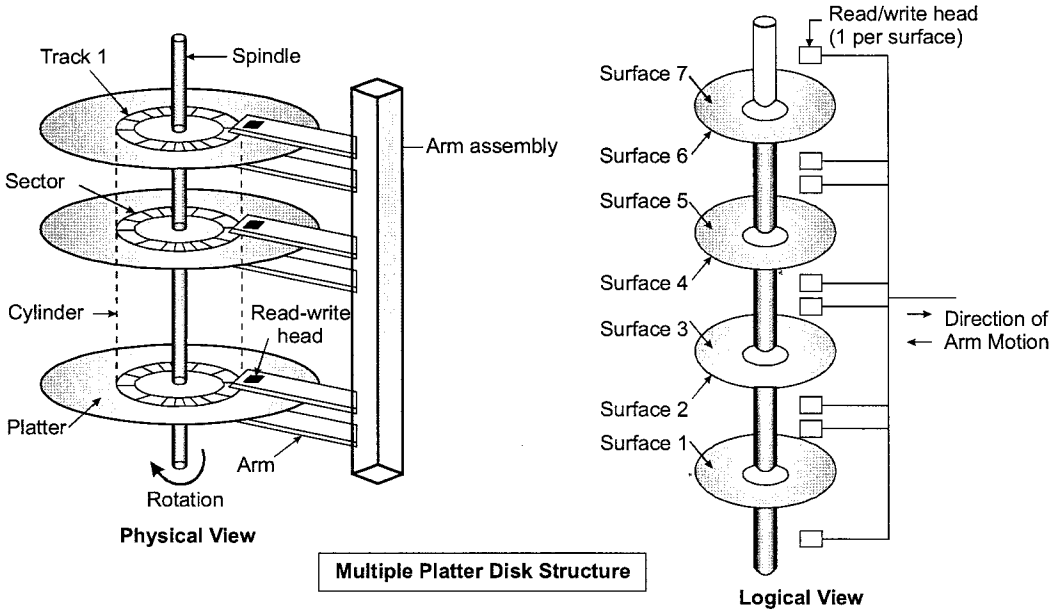
- The drives are relatively expensive.
- They are not as widely used as the other storage devices.

BD (Bluray Disk)

It requires BD drive for its operation. Its shape and size is similar to CD and DVD. It has the storage capacity of 25 GB-50 GB. Types of BD are: BD-R, BD-ROM and BD-RW.

5.3.4 Structure of a Disk (Disk Structure)

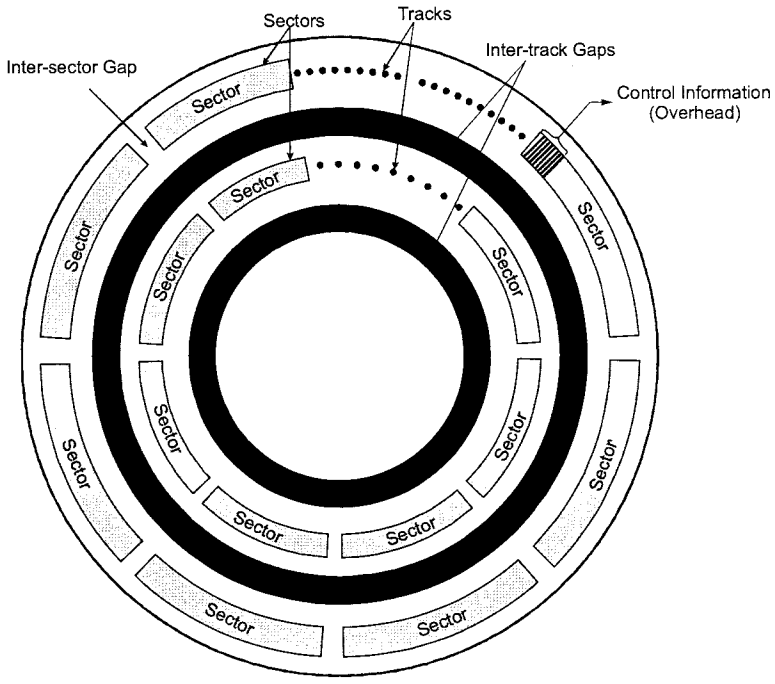
The surface of a platter is organised as a number of concentric tracks. Each track is divided into sectors. The information held in one sector called as a block, is the unit of transfer between the disk and primary memory. The operating system determines where the blocks for each file are placed.



Multiple Platter Disk Structure

- **Track:** It is a circular ring on one side of the disk. Each track has a number.
- **Sector:** It is a wedge shaped piece of a disk. Each sector is numbered.
- **Cylinder:** It is a set of matching tracks in a vertical plane through a disk pack.
- **Read/ Write Head:** A device that reads data from and writes data to a magnetic disk. For writing the surface of the disk is moved past the read/write head.
- **Access Arm (Moving Arm):** This is a mechanical arm that moves the read/write head across the surface of the disk. It is directed by the OS to move the read/write head to a specific track or the disk.

- **Platter:** Metallic disks where one or both sides of the platter are magnetized, allowing data to be stored. The platter spins thousands of times a second around the spindle. There may be several platters, with data stored across them.



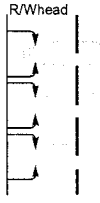
Sectors and Tracks on Surface

The basic unit of transfer from the disk is a "sector". The disk space without control information is called formatted disk space. The disk is a semi random access memory and data is distributed across circular tracks and sectors. Two types of disk construction:

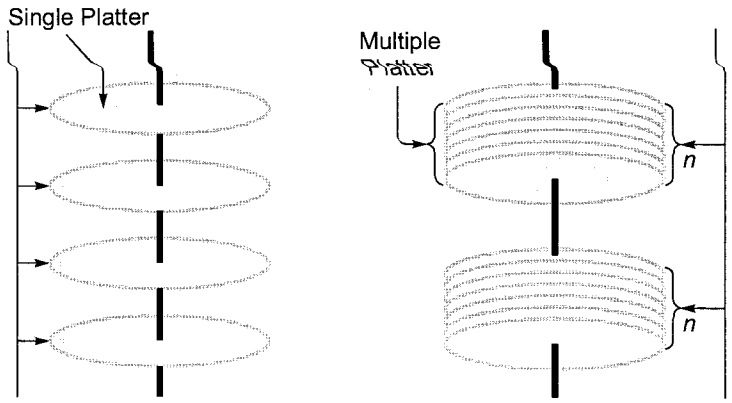
1. **Constant Track Capacity Disk:** In this case variable recording density is used across all the tracks and the disk has to move with angular velocity.
2. **Variable Track Capacity Disk:** In this case constant recording density is used.

Characteristics of Disks

- Single sided/Double sided disks: By default double sided disks are used, in which recording is done on both the surfaces.
- Fixed Read/Write (R/W) head or Movable R/W head:

Fixed R/W Head	Movable R/W Head
	
(a) One R/W head per surface (b) Each rotation covers 'n' tracks n = no. of surfaces (c) Hardware increases and hence cost is more	(a) One R/W head for all surface (b) Each rotation covers 1 track (c) Less hardware required but access time increases

- Single platter (or) Multiple platter disks



- **Fixed Disk:** It is a high speed, high capacity disk drive that is housed in their own cabinets. They are not removable or portable. They have greater capacity than external disk and are more reliable.
- **External Disk:** These are detachable which their own power supply and are not built into the system cabinet. They can store GB's of data.

NOTE

- Number of cylinders = Number of tracks in a surface
- Single Track capacity = Number of sector per track × Number of bytes per sector
- Cylinder capacity = Number of surfaces × Single track capacity
- Disk capacity = Number of cylinders × Cylinder capacity = Track × Sectors/Track × Bytes/Sector
- Disk Capacity$_{formatted}$ = Tracks × Sectors/Track × (Bytes/Sector – Format overhead)

Disk Access Time

- The time taken to access a particular block consists of seek time, rotational delay and transfer time.
- Each rotation of the disk covers 1 track.
- The data transfer rate depends on rpm.
- **Seek Time (t_s):** The time to move Read/Write (R/W) head to the desired track.
- **Rotational Latency (t_r):** The time to move Read/Write (R/W) head to the desired sector beginning.

Rotational latency = Half of rotation time

$$t_r = \frac{1}{2} \times \text{rotation time}$$

- The data transfer rate of the disc ($t_{\text{data transfer}}$) = Number of Bytes/sec

Time Component	Action
Seek time	Time to move the read/write arm to the correct cylinder
Rotational delay (or latency)	Time it takes for the disk to rotate so that the desired sector is under the read/write head
Transfer time	Once the read/write head is positioned over the data, this is the time it takes for transferring data.

- Disk access time = Seek time + Rotational latency + Transfer time + Control overhead

Access time of the disc: $t_{avg} = t_s + t_r + t_{\text{data transfer}} + t_{overhead}$

Advantages and Disadvantages of Disk

Advantages:

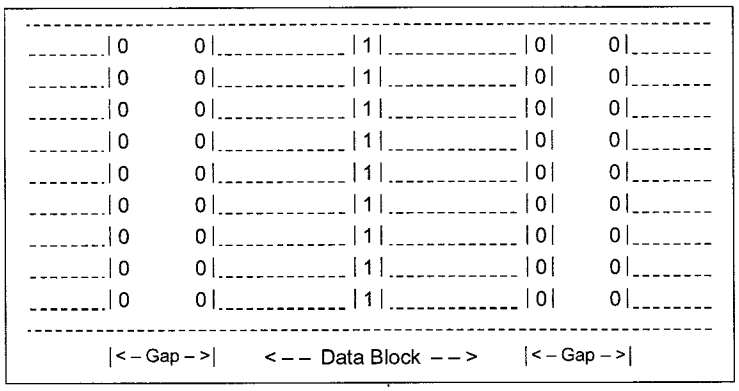
- Fast seek times
- Random access
- High capacities possible
- Low cost per megabyte

Disadvantage: Very susceptible to damage from physical shocks

5.3.5 Data Organization on Nine-Track Tapes

In a tape, the **logical position** of a byte within a file is the same as its **physical position** in the file (sequential access).

Nine-track tape:



- Data blocks are separated by interblock GAPS.
- 9 parallel tracks (each is a sequence of bits)
- A frame is a 1 bit slice of the tape corresponding to 9 bits (one in each track) which correspond to 1 byte plus a parity bit.

Estimating Tape Length Requirements

b = Length of data block (in inches)

g = Length of interblock gap (in inches)

n = Number of data blocks

Tape length = Space requirement (s) = n × (b + g)

Effective Recording Density (ERD)

ERD = Number of bytes per block/Number of inches to store a block.

***Example*-5.3** A disk pack has 19 surfaces. Storage area on each surface has an inner diameter of 22 cm and outer diameter of 33 cm. Maximum storage density on any track is 2000 bits/cm and minimum spacing between tracks is 0.25 cm. (a) What is the storage capacity of the pack? (b) What is the data transfer rate in bytes per second at a rotational speed of 3600 rpm?

Solution:

Given, number of surfaces = 19

Inner track diameter = 22 cm

Outer track diameter = 33 cm
So, total track with = (33 – 22)/2 cm = 5.5 cm
Track separation = 0.25 mm
Thus, number of tracks/surface = (5.5 × 10)/0.25 = 220
Minimum track circumference = 22 × P cm
Maximum track storage density = 2000 bits/cm, which will be on innermost track.
So, data storage capacity/track = 22 × P × 2000 bits= 138.23 Kbits
Disk speed = 3600 rpm
So, rotation time = 1/3600 minute = 16.67 msec (1 msec = 10^3 sec)

(a) Storage capacity = 19 × 220 × 138.23 Kbits = 577.8 Mbits = 72.225 Mbits

(b) Data transfer rate = 138.23 Kbits/16.67 msec = 8.2938 Mbits/sec

This is the maximum data transfer rate excluding seek time and rotational latency.

Example-5.4 Consider the following file and disk characteristics.

File characteristics: file contain fixed-length records.
Number of records = 50,000 records
Size of a record = 256 bytes
Disk characteristics:
Number of bytes per sector = 512
Number of sectors per track = 63
Number of tracks per cylinder = 16
Number of cylinders = 4092
How many cylinders are needed to store the file?

Solution:
Number of records per sector = 2
Number of records per track = 2 × 63
Number of records per cylinder = 16 × 126 = 2016
Number of cylinders = 50000/2016 ≈ 24.8 cylinders

Example-5.5 A high speed tape system accommodates 1200 ft. reel of standard 9 track tape. The tape is moved past the recording head at a rate of 140 inches per second. What must be the linear tape recording density in the order to achieve a data transfer rate of 10^5 bits per seconds?

Solution:
Given, Tape length = 1200 ft.
Tape speed = 140 inches/sec
Data transfer rate = 10^5 bits/sec.
We have, data transfer rate = Tape density × Tape speed
Therefore, the tape density = 10^5/140 bits/inch = 714 bits/inch

Example-5.6 Suppose a 30 GB hard-disk is to be manufactured. If the technology used to manufacture the disks allows 1024 sectors, 2048 sector tracks and 4096 track platters. How many platters are required?

Solution:
The total capacity of each platter = Size of each sector × Number of sectors per track × Number of tracks per platter

$= 1024 \times 2048 \times 4096$ bytes

$= 8 \times 2^{30}$ bytes = 8 GB

Therefore, number of platters required = [Capacity_of_disk/capacity_of_each_platter] = [30/8] = 4

Example-5.7 What is the average time to read or write a 512 byte sector from a disk rotating at 5400 RPMs, if the seek time is 12 ms, the transfer rate is 4 MB per second and the controller overhead is 8 ms?

Solution:

Disk access time = Seek time + Rotational latency + Transfer time + Control overhead

Transfer time = 512/4 MB = 0.122 ms

Disk access time = 12 + 11.11 + 0.122 + 8 ms = 31.232 ms

Example-5.8 A hard disk with one platter rotates at 15,000 rpm and has 1024 and has 1024 tracks, each with 2048 sectors. Disk read-write head starts at track 0. (Tracks are numbered from 0 to 1023). The Disk receives a request to access a random sector on a random track. If the seek time of the disk head is 1 ms for every 100 tracks it crosses.

(a) What is the average seek time?

(b) What is the average rotational latency?

(c) What is the transfer time for a sector?

Solution:

(a) Since, the disk receives a request to access a track at random. Thus, the head may have to move either direction. On an average, the head will have to move 1024/2 = 511.5 tracks.
Since the seek time of the head is 1 ms per 1000 tracks, the average seek time
= 511.5/100 = 5.115 ms.

(b) Since, the platter rotates at 15,000 rpm, each rotation takes 1/15000 min = $(60 \times 10^3)/15000$ = 4 ms
The average rotational latency is half the rotation time, which is = 2 ms.

(c) Each rotation takes 4 ms
Number of sectors per track = 2048
Therefore, each sector has read-write head over it = 4/2048 ms = 0.002 ms (Approx)
Therefore, the transfer time is 0.002 ms (Approx)

Example-5.9 A Winchester magnetic disk unit has densities of 40×10^6 bits per square inch of surface. (a) If the inner diameter of recording area is 4 inches and the outer diameter of 4 inches, what is the average bit density along a track if radial track spacing density is 2000 tracks/inch. (b) What is the data transfer rate in bytes/sec at a rotational speed of 3600 RPM?

Solution:

Inner diameter D_1 = 4 inches

Outer diameter D_0 = 7 inches

Density = 40×10^6 bit/inch^2

$$\text{Total recording area} = \pi \times \left(\frac{D_0}{2}\right)^2 - \pi\left(\frac{D_1}{2}\right)^2 = 25.91814 \text{ inch}^2$$

$$\text{Total disk capacity} = 25.91814 \times 10 \times 10^6 = 103.67 \times 10^3$$

Radia track spacing = 7 – 4 = 3

Number of tracks on the disk = 2000 × 3 = 6000 / inch

$$\text{Average bit density along a track} = \frac{103.67 \times 10^7}{6000} = 172.78 \text{ kbits/track}$$

$$\text{Data transfer rate in bytes /sec} = \frac{\text{Average bit density}}{\text{Along a track}} \times \text{Rotation speed}$$

$$= \frac{172.78K}{8} \times \frac{3600}{60} = 1.2959 \text{ M bytes/sec}$$

Example-5.10 For 120 track tape with a storage density per track of 100 kb/in and tape speed of 50 in/s, calculate the maximum data transfer rate. If tape length is 450 feets, calculate the storage capacity of the tape.

Solution:

$$\text{Maximum data transfer rate D} = \text{Storage density / track} \times \text{Tape speed} \times \frac{\text{Number of tracks}}{8}$$

$$= 100 \times 10^3 \times 50 \times 120/8 = 75 \text{ Mbytes/sec}$$

The storage capacity of the tape is given by

$$S = \frac{d \times \text{Tape length in inches}}{\text{Tape speed}} = \frac{75 \text{ Mbytes/sec} \times (450 \times 12)}{50 \text{ in/sec}}$$

$$= 8.1 \text{ Gbyte}$$

Example-5.11 A magnetic-tape system accommodates 2400 ft reels of standard nine-track tape. The tape is moved post the recording head at rate of 200 in/sec.

(a) What must the linear tape-recording density be in order to achieve data transfer rate of 10 Mbits/sec.

(b) Suppose that the data on the tape is organized into blocks each containing 32 kbytes. A gap of 0.3 in separates each block. How many bytes may be stored on the tape?

Solution:

(a) Tape speed = 200 in/sec

Data transfer rate = 10 Mbits/sec

The tape recording density tot achieve data transfer rate of 10 Mbytes/sec is given by

$$\text{Tape recording density} = \frac{10 \text{ Mbits/sec}}{20 \text{in/sec}} \times \frac{1}{2} = 6.25 \text{ k bytes/in}$$

(b) The block length, $B_L = \frac{\text{Block storage capacity}}{\text{Storage density}} = \frac{32 \text{ kbits}}{6.25 \text{ kbytes/in}} = 5.12 \text{ in}$

$$\text{The number of block on the tape} = \frac{\text{Tape legnth}}{B_L + G_L} = \frac{2400 \times 12}{5.12 + 0.3} = 5313.6 = 5313$$

Storage capacity of tape S = Number of blocks × Storage capacity of block

= 5313 × 32 kytes = 170.016 Mbytes

Example-5.12 Consider the following disk characteristics:

Average seek time = 8 msec
Average rotational delay = 3 msec
Maximum rotational delay = 6 msecs
Spindle speed = 10,000 rpm
Sectors per track = 170 sectors
Sector size = 512 bytes.

What is the average time to read one sector?

Solution:

Transfer time = Revolution time / Number of sectors per track
= (1/10,000) min /170
= (1/10000 × 60)/170 secs = 6/170,000 secs
= 6/170 msecs ≈ 0.035 msecs

Average total time = Average seek + Average rotational delay + Transfer time
= 8 +3 + 0.035 = 11.035 msec.

Example-5.13 Suppose that there is a typical disk with an average seek time of 20 ms, transfer data at the rate of 1MB/s and 512 byte sectors with 32 sectors per track. We wish to read a file consisting of 256 sectors for a total of 128 K bytes. The file occupies 8 adjacent (contiguous/sequential) tracks (8 × 32 = 256). (a) Find the time to read entire file and (b) Find the time to read entire file with a condition that file occupies random sectors.

Solution:

(a) Time to read first track:

Average seek time = 20 ms
Rotational delay = 8.3 ms
To read 32 sectors = 16.7 ms (one revolution takes 16.7 ms)

Also suppose the remaining tracks can be read with essentially no seek time, the only additional delay for all the remaining tracks will the be rotational delay that is 8.3 × 16.7 = 25 ms. Therefore, to read the entire file;

Total time = Time for first track + time for remaining 7 tracks
= (20 + 8.3 + 16.7) + 7 * 25 = **0.22 sec**

(b) With the condition that file occupies random sectors.

For each sector:

Average seek time = 20 ms
Rotational delay = 8.3 ms
Time for one sector = 16.7/32 = 0.52 ms
Total of all these = 28.8 ms
Therefore, total time = 256 × 28.8 = **7.37 sec**

Example-5.14 Consider the hard disk with 10601 cylinders, 12 tracks per cylinder, 281 sectors per track on average, 512 bytes per sector, for an overall disk capacity of 18.3 GBytes. The seek time between adjacent cylinders is 0.8 msec; the average seek time is 6.9 msec, the rotation time is 8.33 msec. Assume that the disk is not interleaved so that an entire track can be read in one rotations of the disk.

(a) How long would it take to read the entire disk from beginning to end in order?

(b) Suppose that the block size is 1KB, that each block consists of two consecutive sectors, and that you must read every block on the disk in random order. How long would this take?

Solution:

(a) There are $12 \times 10601 = 1.3 \times 10^5$ tracks, and each can be read in a rotation time of 8.3×10^{-3} sec, for a total of about 10^3 seconds = about 17 minutes. (The seek time, which is always between adjacent cylinders, is negligible).

(b) There are $10601 \times 12 \times 281/2 = 1.8 \times 10^7$ blocks (or alternatively 18.3 GBytes/1024 Bytes per block) on the disk. To move from one block to another takes on average the average time for a seek plus half a rotation = 11.8 msec. Total time is therefore 2.1×10^5 seconds = 2 days, 10.5 hours.

Example-5.15 Suppose that we build a disk subsystem to handle a high rate of I/O by coupling many disks together. Properties of this system are as follows:

- Uses 10 GB disks that rotate at 10,000 RPM, have a data transfer rate of 10 MBytes/s (for each disk), and have an 8 ms average seek time, 32 KByte block size
- Has a SCSI interface with a 2ms controller command time.
- Is limited only by the disks (assume that no other factors affect performance).
- Has a total of 20 disks

Each disk can handle only one request at a time, but each disk in the system can be handling a different request. The data is not striped (all I/O for each request has to go to one disk). What is the average service time to retrieve a single disk block from a random location on a single disk, assuming no queuing time (i.e. the unloaded request time)?

Solution:

$$\text{Time}_{\text{service}} = \text{Time}_{\text{controller}} + \text{Time}_{\text{seek}} + \text{Time}_{\text{rotational}} + \text{Time}_{\text{transfer}}$$

$$\text{Time}_{\text{contoller}} = 2 \text{ ms}$$

$$\text{Time}_{\text{seek}} = 8\text{ms}$$

$$\text{Time}_{\text{rotational}} = 1/2$$

$$\text{Time}_{\text{rotation}} = 1/2 \times [(60\text{s/M})/(10000 \text{ RPM})] = 0.003\text{s} = 3\text{ms}$$

$$\text{Time}_{\text{transfer}} = (32 \times 1024 \text{ bytes})/(10 \times 106 \text{ bytes/sec}) = 0.0032768\text{s} = 3.2768\text{ms}$$

$$\text{Time}_{\text{service}} = 2 + 8 + 3 + 3.28 \text{ ms} = 16.28\text{ms}$$

Example-5.16 Suppose that we have a disk with the following parameters:

- 750 GB in size
- 12000 RPM, data transfer rate of 40 Mbytes/s (40×10^6 byes/sec)
- Average seek time of 8 ms
- ATA controller with 2ms controller initiation time
- A block size of 4 Kbytes (4096 bytes)

What is the average time to read a random block from the disk (assuming no queueing at the controller).

Solution:

$$T_{read} = \text{Controller} + \text{Seek} + \text{Rotational} + \text{Transfer time}$$

$$= 2ms + 8ms + \frac{1}{2}\left(\frac{6000\ \frac{ms}{min}}{12000\ \frac{revolutions}{min}}\right) + \frac{4096\ \text{bytes}}{\left(40\times10^{6}\ \frac{bytes}{s}\right)\times\left(10^{-3}\ \frac{s}{ms}\right)}$$

$$= 12.6024ms$$

***Example*-5.17** Given the same parameters from above, assume that the operating system has exploited locality by grouping related blocks together in the filesystem. As a result, the typical access pattern is not as random as in 2d. It typically retrieves 10 blocks sequentially at a time and spends only 1ms for each seek. What is the average time to read a *single* block now?.

Solution:

Compute the time for 10 block reads (since there is only q seek and 1 rotational for those 10), then divide by 10 for a single block read. Also, the seek is shorter. Since there is no queuing, each request is treated separately - thus leading to q controller timer/requests.

$$T_{read10} = 10 \times \text{Controller} + \text{Seek} + \text{Rotational} + \text{Transfer time}$$

$$= 10\times2ms + 1ms + \frac{1}{2}\left(\frac{6000\ \frac{ms}{min}}{12000\ \frac{revolutions}{min}}\right) + 10\times\frac{4096\ \text{bytes}}{\left(40\times10^{6}\ \frac{bytes}{s}\right)\times\left(10^{-3}\ \frac{s}{ms}\right)}$$

$$= 24.524$$

$$\therefore \quad T_{read} = T_{read10}/10 = 2.4524ms$$

Summary

- **Programmed I/O:** In which I/O occurs under the direct and continuous control of the program requesting the I/O operation.
- **Interrupt-driven I/O:** In which a program issues an I/O command and then continues to execute, until it is interrupted by the I/O hardware to signal the end of the I/O operations.
- **Direct Memory Access:** In which a specialized I/O processor takes over control of an I/O operation to move a large block of data.
- **DMA controller:** It allows data transfer directly between I/O device and Memory, with minimal intervention of CPU. DMA controller acts as a I/O processor, but it is controlled by CPU.
- **Bus master:** Device that initiates data transfers on the bus. The next device can take control of the bus after the current master relinquishes control.
- **Bus Arbitration:** Process by which the next device to become master is selected.
- **DMA Bust Transfer Mode:** In which a block of data is transferred before returning bus control from DMA to the CPU. **DMA Cycle Stealing Transfer Mode:** In which DMA takes the control over the bus for each byte of data to be transferred and then return control to the CPU.
- **DMA Demand Transfer Mode:** In which DMA takes the control over the bus for entire data to be transferred and then return control to the CPU.
- Burst mode is also called as Block Transfer mode.
- Cycle stealing mode is also called as Byte Transfer mode.
- Demand mode is also called as Complete data transfer mode.
- **Magnetic Tape:** This is storage medium that consists of a plastic ribbon that has been coated with material that can be magnetized to represent data.
- **Floppy Disk:** The diskette is a flexible magnetic disk on which data is recorded magnetically. The device is used to read data from a disk or record data on a disk is called a floppy device.
- **Hard Disk (Hard Drive):** A hard disk is non-removable, enclosed magnetic disk drive that reads data from and writes data to hard disk. The disks inside the hard drive are called platters.
- **Seek Time:** Time to move the heads to the right track called the seek time.
- **Latency/Rotational delay:** Time waiting for the sector to come round to the head called the latency.
- **Block Transfer Time:** Time to actually transfer the data is called the block transfer time.
- Disk access time = Seek time + Rotational latency + Transfer time + Control overhead.

Student's Assignment

Q.1 During DMA transfer, DMA controller takes over the buses to manage the transfer
(a) directly from CPU to memory
(b) directly from memory to CPU
(c) indirectly between the I/O device and memory
(d) directly between the I/O device and memory

Q.2 Memory mapped I/O involves
(a) transferring information between memory locations
(b) transferring information between registers and memory
(c) transferring information between the CPU and I/O devices in the same way as between the CPU and memory
(d) transferring information between I/O devices and memory

Q.3 A hardware interrupt is
(a) also called an internal interrupt
(b) also called an external interrupt
(c) an I/O interrupt
(d) a clock interrupt

Q.4 To prevent signals from colliding on the bus, _____ prioritize access to memory by I/O channels and processors.
(a) a register
(b) interrupts
(c) the processor scheduler
(d) a controller

Q.5 Which of the following is not valid class of interrupts?
1. Program
2. Timer
3. I/O
4. Hardware failure

(a) 1 and 3 (b) 1, 2 and 4
(c) 2 and 3 (d) None of these

Q.6 What is signified by the term Direct Memory Access (DMA)?
(a) A less efficient I/O mechanism than 'interrupts' for single byte transfer operations.
(b) A class of multiprocessor computers.
(c) A mechanism used by the I/O system to bypass the memory management functions of an operating system.
(d) A more efficient I/O mechanism than 'programmed I/O' for single byte transfer operations.

Q.7 Which one is not a type of I/O channel?
(a) Multiplexer (b) Selector
(c) Block-multiplexer (d) None of these

Q.8 The CPU initializes the DMA by sending _____.
(a) the starting address of the memory blocks where data is available or where data is to be stored
(b) the word count
(c) control for mode and start the transfer
(d) All of the above

Q.9 Match **List-I** with **List-II** and select the correct answer using the codes given below the lists:

List-I	List-II
A. Stack overflow	1. Software interrupt
B. Supervisor call	2. Internal interrupt
C. Invalid opcode	3. External interrupt
D. Timer	4. Machine check interrupt

Codes:

	A	B	C	D
(a)	2	3	4	1
(b)	2	1	2	3
(c)	3	1	2	4
(d)	3	1	4	2

Q.10 Consider the following situation and fill in the blanks:
The computer starts the tape moving by issuing a command; the processor then monitors the status of the tape by means of a _____. When the tape is in the correct position, the processor issues a _____.
(a) Data input, status, data output command
(b) Data input, status control command
(c) Control, status, data output command
(d) Control, status, data input command

Q.11 In case of vectored interrupt, interrupt vector means
(a) The branch information from the source which interrupts the system
(b) An address that points to a location in memory where the beginning address of the I/O service routine is stored
(c) Both (a) and (b)
(d) None of these

Q.12 Consider the following I/O instruction format for IBM 370 I/O channel

Operation Code	Channel Address	Device Address

Then operation code specifies
1. test I/O
2. test channel
3. store channel identification
4. halt device

(a) Only 1 and 4 (b) Only 2 and 3
(c) 1, 2, 3 and 4 (d) 1, 3 and 4

Q.13 Compare the following by considering data transfer rate
1. I/O processor
2. Data communication processor

(a) 1 = 2 (b) 1 > 2
(c) 1 < 2 (d) None of these

Q.14 The computer can execute 1,000,000 instructions per second. A program running on this computer performs on average a one sector read and one sector write for every 200 instructions that it executes. The disk drive handling the I/O transfers requires 0.00010 seconds each to perform the read and write operations.

Assuming no overlap of these operations, the percent of CPU time spent in the wait state is
(a) 12% (b) 39%
(c) 57% (d) 91%

Q.15 In a general purpose computer system the CPU, the main memory and the cache may be interconnected via one or more shared system bus(es). However, input/output devices (eg. Hard disk, network interfaces) may only be connected to the system bus through an I/O controller.

The following are four statements regarding the requirement for an I/O controller.
1. The capacities of I/O devices are magnitude order larger than that of main memory and hence direct interfacing is impossible.
2. The response times of I/O devices are magnitude order slower than that of CPU and hence direct interfacing is impossible.
3. It is always better to off load the I/O processing to a secondary processor on the I/O controller board then to depend on the primary CPU for I/O processing.
4. The variety of I/O devices in the market requires that a separate I/O controller exist for each device.

What statement(s) best explain the requirement for an I/O controller?
(a) Only 1 is true
(b) Only 3 is true
(c) Only 2 and 3 are true
(d) Only 2, 3 and 4 are true

Q.16 An 8-Bit DMA Device is operating is Cycle Stealing Mode (Single Transfer Mode). Each DMA cycle is of 6 clock states and DMA clock is 2 MHz. Intermediate CPU machine cycle takes 2 $\mu\sigma$, determine the DMA Data Transfer Rate
(a) 100 Kbytes/sec (b) 200 Kbytes/sec
(c) 350 Kbytes/sec (d) None of these

Q.17 What will be average cost per bit for a system with main memory of 1024 cost, 100 units and secondary memory of 4096 cost, 10 units.
(a) 35.7 (b) 28.0
(c) 82.0 (d) insufficient data

Q.18 Consider a Disk I/O transfer, in which 1500 bytes are to be transferred, but number of bytes on a track is 1000, and rotation speed of disk is 1500 rps but the average time required to move the disk arm to the required track is 15 ms, then what will be total access time?
(a) 16.33 ms (b) 15.33 ms
(c) 16.33 μs (d) 15.33 μs

Q.19 A disc drive has a rotational speed of 3600 rpm, an average seek time of 10 ms, 64 sectors per track and 512 bytes of data per sector. What is the average time to access the entire data of a 16 kbytes file stored sequentially on the disk?
(a) 18.85 ms (b) 10 ms
(c) 27.15 ms (d) 9 ms

Q.20 The seek time of a disk is 30 msec. It rotates at the rate of 30 rotations per second. Each track has a capacity of 300 words. The access time is
(a) 47 msec (b) 50 msec
(c) 60 msec (d) 62 msec

Q.21 A disc drive has a average seek time of 10 ms, 32 sectors on each track and 512 bytes per sector. If the average time to read 8 kbytes of continuously stored data is 20 ms, what is the rotational speed of the disc drive?
(a) 3600 rpm (b) 6000 rpm
(c) 3000 rpm (d) 2400 rpm

Q.22 A disc rotates at a speed of 7200 rpm. It has 4000 cylinders, 16 surfaces and 256 sectors per track. What is the average latency time of the disk?
(a) 8.33 ms (b) 4.166 ms
(c) 41.66 ms (d) 8.33 ms

Q.23 A magnetic drum of 8 inch diameter has 100 tracks and storage density of 200 bits/inch. What is its storage capacity?
(a) 8402 bits (b) 202400 bits
(c) 502400 bits (d) 1004800 bits

Q.24 Match **List-I** (Type of Memory) with **List-II** (Access time in μsec/nsec) and select the correct answer using the code given below the lists:

List-I	List-II
A. DRAM	1. 1
B. SRAM	2. 10
C. Hard Disk	3. 100000
D. Magnetic Tape	4. 10000000

Codes:

	A	B	C	D
(a)	1	2	3	4
(b)	1	2	4	3
(c)	2	1	3	4
(d)	2	1	4	3

Q.25 The difference between memory and storage 1's that the memory is _____ and the storage is _____.
(a) Temporary, Permanent
(b) Permanent, Temporary
(c) Slow, Fast
(d) None of these

Q.26 Which of the following holds, the ROM, CPU, RAM and Expansion cards?
(a) Hard disk (b) Floppy disk
(c) Mother board (d) None of these

Q.27 Information retrieval is faster from
(a) Floppy disk (b) Magnetic tape
(c) Hard disk (d) Cassette tape

Q.28 Winchester disk is a
(a) Disk stack (b) Removable disk
(c) Flexible disk (d) None of these

Q.29 Magnetic tapes are good storage media for
(a) backup and low volume data
(b) backup and high voltage data
(c) storing original but low volume data
(d) storing original but high volume data

Q.30 Suppose that disk has the following parameters.
- 200 GB size
- 12000 rotations per minute
- Data transfer rate = 8×10^4 bytes/sec
- Average seek time = 8 ms
- Block size = 1024 bytes

If head is already placed at required blocks, what is the data transfer time to transfer a single block?
(a) 12.8 ms (b) 20.8 ms
(c) 22 ms (d) None of these

Q.31 Consider the following characteristics of disk system.
- Seek time is 6 ms per cylinder
- Disk uses shortest seek time first scheduling
- Disk requests for cylinders 8, 24, 20, 5, 41, 8 in that order come into the driver
- Initially disk arm is at cylinder 20.

What is the total seek time for the above requests?

(a) 57 ms (b) 59 ms
(c) 342 ms (d) 354 ms

Q.32 Consider the following parameters of a disk system.

- Number of cylinders = N_{cyl}
- Number of sectors/track = N_{sec}
- Number of surfaces = N_{surf}
- Number of tracks per surface = N_{track}
- Sector capacity = C_{sec}
- Cylinder capacity = C_{cyl}

Which of the following is correct to compute the disk capacity of the system?

(a) Disk capacity = $N_{surf} \times N_{track} \times N_{sec} \times C_{sec}$
(b) Disk capacity = $N_{cyl} \times C_{cyl}$
(c) Disk capacity = $N_{surf} \times N_{cyl} \times N_{sec} \times C_{sec}$
(d) All of the above

Q.33 A certain moving arm disk storage with on head has following specifications:

- Number of tracks/recording surface = 200
- Disk rotation speed = 2400 rpm
- Track storage capacity = 62500 bits.

The average seek time (assume that the head can move from one track to another only by traversing the entire track) is

(a) 2.5 s (b) 2.9 s
(c) 3.1 s (d) 3.6 s

Answers Key:

1. (d)	**2.** (c)	**3.** (b)	**4.** (d)	**5.** (d)
6. (a)	**7.** (d)	**8.** (d)	**9.** (b)	**10.** (c)
11. (c)	**12.** (c)	**13.** (b)	**14.** (d)	**15.** (d)
16. (b)	**17.** (b)	**18.** (a)	**19.** (c)	**20.** (b)
21. (d)	**22.** (b)	**23.** (c)	**24.** (c)	**25.** (a)
26. (c)	**27.** (c)	**28.** (a)	**29.** (b)	**30.** (a)
31. (d)	**32.** (d)	**33.** (a)		

CHAPTER 06

Data Representation

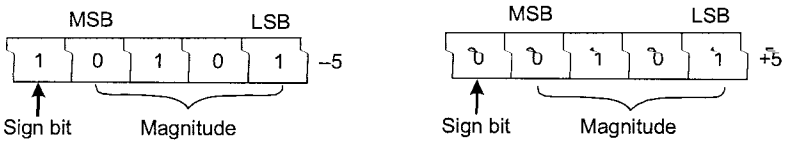
6.1 Fixed and Floating Point Formate

Positive integers, including zero, can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with 1's and 0's, including the sign of a number. As a consequence, it is customary to represent the sign with a bit placed in the leftmost position of the number. The convention is to make the sign bit equal to 0 for positive and to 1 for negative.

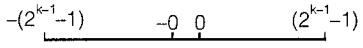
6.1.1 Integer Representation

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

1. **Signed Magnitude Method:** In this method number is divided into two parts: Sign bit and magnitude. If number is positive then sign bit will be 0 and if number is negative then sign bit will be 1. Magnitude is represented with the binary form of the number to be represented.
   Example: Let we are using five bits register. The representation of –5 and +5 will be as follows:



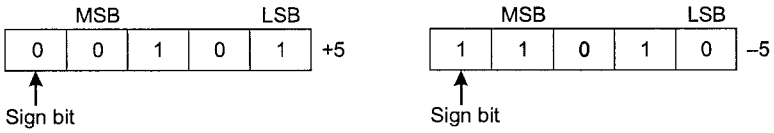
   Range of Numbers: For k bits register, MSB will be sign bit and (k – 1) bits will be magnitude. *Positive largest number that can be stored is* $(2^{k-1} - 1)$ *and negative lowest* number that can be stored is $-(2^{k-1} - 1)$.



NOTE: Drawback of this system is that 0 has two different representation one is –0 and second is +0.

2. **1's Complement Method:** Positive numbers are represented in the same way as they are represented in sign magnitude method. If number is negative then it is represented using 1's complement. First represent the number with positive sign and then take 1's complement of that number.

   Example: Let we are using five bits register. The representation of –5 and +5 will be as follows:



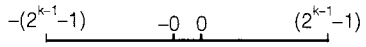
   +5 is represented as it is represented in sign magnitude method. –5 is represented using the following steps:

   (*i*) +5 = 0 0 1 0 1

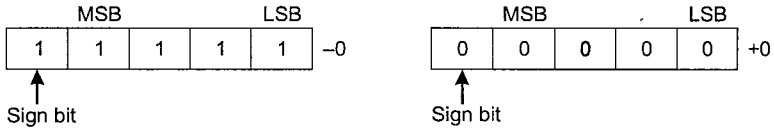
   (*ii*) Take 1's complement of 0 0 1 0 1 and that is 1 1 0 1 0. MSB is 1 which indicates that number is negative.

   MSB is always 1 in case of negative numbers.

   Range of Numbers: For k bits register, positive largest number that can be stored is $(2^{k-1} - 1)$ and negative lowest number that can be stored is $-(2^{k-1} - 1)$.

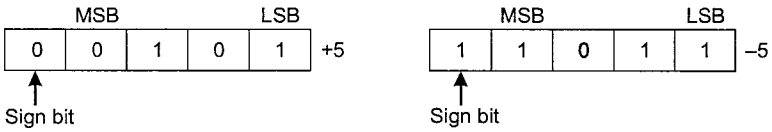


NOTE: Drawback of this system is that 0 has two different representation one is –0 and second is +0 same as in the case of sign magnitude method.



3. **2's Complement Method:** Positive numbers are represented in the same way as they are represented in sign magnitude method. If number is negative then it is represented using 2's complement. First represent the number with positive sign and then take 2's complement of that number.

   Example: Let we are using five bits register. The representation of –5 and +5 will be as follows:



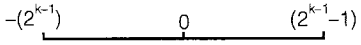
   +5 is represented as it is represented in sign magnitude method. –5 is represented using the following steps:

   (*i*) +5 = 0 0 1 0 1

   (*ii*) take 2's complement of 0 0 1 0 1 and that is 1 1 0 1 1. MSB is 1 which indicates that number is negative.

MSB is always 1 in case of negative numbers.

Range of Numbers: For K bits register, positive largest number that can be stored is $(2^{k-1} - 1)$ and negative lowest number that can be stored is $-(2^{k-1})$.



Advantage of this system is that 0 has only one representation for –0 and +0.

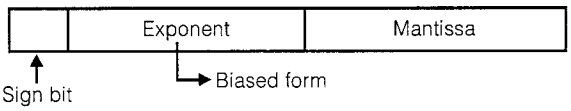
6.1.2 Floating-Point Representation

The floating-point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent. The fixed-point mantissa may be a fraction or an integer. Floating-point is always interpreted to represent a number in the following form:

$$M \times r^e$$

Only the mantissa m and the exponent e are physically represented in the register (including their signs). A floating-point binary number is represented in a similar manner except that it uses base 2 for the exponent.

A floating-point number is said to be normalized if the most significant digit of the mantissa is one. For example, the 8 bit binary number 00011010 is not normalized because of the three leading 0's. The number can be normalized by shifting it three positions to the left and discarding the leading 0's to obtain 11010000. The three shifts multiply the number by $2^3 = 8$. To keep the same value for the floating-point number, the exponent must be subtracted by 3. Normalized numbers provide the maximum possible precision for the floating-point number. A zero cannot be normalized because it does not have a nonzero digit. It is usually represented in floating-point by all 0's in the mantissa and exponent.



Sign bit 0 means number is positive and sign bit 1 means number is negative. Exponent is always stored in biased form. Mantissa part stores the fractional part. For storing the exponent in biased form, bias number is calculated.

If k bits are used to represent exponent then

Bias Number = $(2^{k-1} - 1)$ and

Range of exponent is = $-(2^{k-1} - 1)$ to 2^{k-1}

Thus, if 7 bits are used for storing the exponent then bias number will be $(2^{7-1} - 1) = 63$ and exponent will ranges from –63 to 64.

Note that, here we always store exponent in positive. Biased number is also called excess number. Since exponent is stored in biased form so bias number is added to the actual exponent of the given number. Actual number can be calculated from the contents of the registers by using following formula:

$$\text{Actual Number} = (-1)^s (1 + m) \times 2^{e-\text{Bias}}$$

where s = sign bit

m = mantissa value of register
e = exponent value of register
Bias = bias number

Example: Represent +0.125 if 5 bits are used to represent exponent and 6 bits for mantissa.

Step 1: Calculate bias number

Bias Number = $(2^{k-1} - 1)$ and here k = 5
= $(2^{5-1} - 1)$
= 15

Step 2: Calculate binary number of the given decimal number

$0.125 \times 2 = 0.25$
$0.25 \times 2 = 0.5$
$0.5 \times 2 = 1.0$
$(0.125)_{10} = (0.001)_2$

The above binary is not normalized.

Step 3: Normalized the binary number

$0.001 = 1.0 \times 2^{-3}$

Step 4: Calculate exponent

Since bias number is 15 so it will be added to the exponent part

i.e., $-3 + 15 = 12$

2	12	0
2	6	0
2	3	1
	1	

$= 1100$

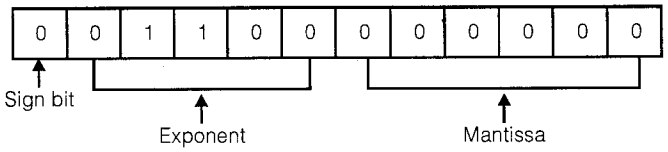
Step 5: Calculate mantissa

Number right to the binary point is 0 so mantissa will be all bits 0.

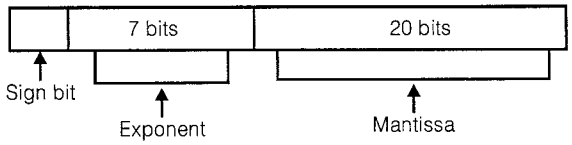
Step 6: Represent the number

Since number is positive so sign bit will be 0.

exponent = 01100
mantissa = 000000



Range of Numbers: Let 7 bits are used to represent exponent and 20 bits are used to represent mantissa.



Bias number = $(2^{7-1} - 1) = 63$

Minimum positive number will be the number with lowest mantissa and exponent i.e.

$$1.000000 \ldots\ldots \underbrace{0000}_{\text{20 times}} \times 2^{-63}$$

i.e., 1×2^{-63} where – 63 is the lowest exponent.

Maximum positive number will be the number with largest mantissa and exponent i.e.

$$1.111111 \ldots\ldots \underbrace{1111}_{\text{20 times}} \times 2^{64}$$

i.e., $(2 - 2^{-20}) \times 2^{64}$ where 64 is the largest exponent.

Similarly, minimum negative number is $-(2 - 2^{-20}) \times 2^{64}$

and maximum negative number is -1×2^{-63}

$$\underbrace{-(2-2^{-20})\times 2^{64} \quad -1\times 2^{-63} \quad 1\times 2^{-63} \quad (2-2^{-20})\times 2^{64}}$$

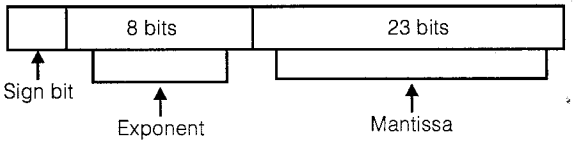
Floating-Point Number Density: As the floating-point number increases, the density of number decreases. So the maximum difference will be between the largest positive number and second largest positive number.

6.2 IEEE Floating-Point Number Representation

According to IEEE standard, floating-point number is represented in two ways:

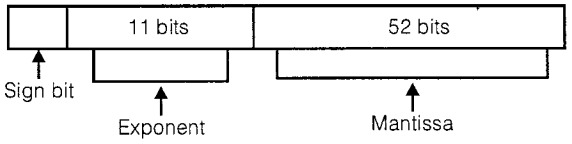
6.2.1 Single Precision

In single precision, 32 bits are used to represent floating-point number. Out of these 32 bits, MSB is used for sign bit, 8 bits for exponent and remaining 32 bits for mantissa. Hence bias number is 127.



6.2.2 Double Precision

In double precision, 64 bits are used to represent floating-point number. Out of these 64 bits, MSB is used for sign bit, 11 bits for exponent and remaining 52 bits for mantissa. Hence bias number is 1023.



In both single and double precision representations, all exponent bits are 0 or all exponent bits are 1 are not used for normal number representation.

Range of exponent will be $-(2^{k-1} - 2)$ to $(2^{k-1} - 1)$. Hence range of numbers in single precision is 2^{-126} to 2^{+127} and range of numbers in double precision is 2^{-1022} to 2^{+1023}.

6.2.3 Special Representation

1. All the exponent bits 0 with all mantissa bits 0 represents zero.
   (a) If sign bit is 0 then it represents + 0.
   (b) If sign bit is 1 then it represents – 0.

2. All the exponent bits 1 with all mantissa bits 0 represents infinity.
   (a) If sign bit is 0 then it represents $+\infty$.
   (b) If sign bit is 1 then it represents $-\infty$.
3. All the exponent bits 0 and mantissa bits non-zero represents denormalized number.
4. All the exponent bits 1 and mantissa bits non-zero represents error.

6.3 Computer Arithmetic

Adding Unsigned Numbers: Adding numbers in binary is pretty much the same as adding in base ten. When adding two k-bit unsigned binary numbers, we apply rules of addition similar to those in base 10. The result is a k-bit unsigned binary number only because hardware doesn't add an additional bit. Thus, adding two 4-bit numbers creates a 4-bit result. If the sum of two k-bit unsigned binary numbers causes a carry out of 1 into column k, there is overflow. Suppose you want to add two unsigned binary 4-bit numbers: $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$.

Here's pseudo-code to add the two numbers:

```
int carry = 0;
for (int i = 0; i < N; i++) {
        int sum = x_i + y_i + carry;
        z_i = sum % 2;
        if (sum >= 2)
                carry = 1;
}
```

Example: Let's add 1101 to 1101 (assume both are 4-bit unsigned binary numbers).

1. Let the rightmost column be column 0, and the leftmost column be column 3.
2. First, add the right most column (column 0). Summing 1 + 1 results in 0, with a carry of 1 to column 1.

$$\begin{array}{r} 1 \\ 1101 \\ +1101 \\ \hline 0 \end{array}$$

3. Then, add the two bits in column 1, plus the carry bit. Summing 0 + 0, plus the carry 1, results in 1, with a carry of 0 to column 2.

$$\begin{array}{r} 01 \\ 1101 \\ +1101 \\ \hline 10 \end{array}$$

4. Then, add the two bits in column 2, plus the carry bit. Summing 1 + 1, plus the carry 0, results in 0, with a carry of 1 to column 3.

$$\begin{array}{r} 101 \\ 1101 \\ +1101 \\ \hline 010 \end{array}$$

5. Finally, add the two bits in column 3, plus the carry bit. Summing 1 + 1, plus the carry 1, results in 1, with a carry of 1 to column 4. We've had to "create" a column 4 to place the fifth bit of the sum.

$$\begin{array}{r} 1\,0\,1 \\ 1\,1\,0\,1 \\ +\,1\,1\,0\,1 \\ \hline \textcircled{1}\boxed{1\,0\,1\,0} \end{array}$$

6. Since we add in hardware, the output usually has the same number of bits as the number of bits being added. Since we were adding two 4-bit unsigned binary numbers, the result should be a 4-bit result. Thus, the additional bit in column 4 (which is circled) is often discarded, or used to detect overflow. The answer is the 4-bit result, drawn inside the rectangle.

Overflow in Unsigned Binary Addition

Overflow can be detected in unsigned binary addition by checking if there is a carry out when summing the leftmost columns (plus the carry bit into the column). The leftmost columns are the most significant bits.

6.4 Adding 2's Complement Numbers

When adding two k-bit 2's complement numbers, we apply the same rules of addition as we did for two k-bit unsigned binary numbers. The result is a k-bit 2's complement number. Provided the result did not overflow, the answer is correct in 2's complement. This is one major reason why we use 2's complement representation for signed integers. Suppose you want to add two 2's complement 4-bit numbers: $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$.

Here's pseudo-code to add the two numbers:

```
int carry = 0;
for (int i = 0; i < N; i++) {
        int sum = x_i + y_i + carry;
        z_i = sum % 2;
        if (sum >= 2)
        carry = 1;
}
```

Example: If you try the same example as in the previous set of notes, you get 1101 (which is -3 in 2's complement) plus 1101 (-3 again), which results in 1010, which is -6 in 2's complement. As before, we throw out the fifth bit. In fact, this example works in 2's complement (i.e., produces a valid answer), but overflows in unsigned binary.

Overflow in 2's Complement Addition

Overflow can be detected in 2's complement addition by exclusive-ORing the carry-in to the most significant column (the leftmost column) and the carry-out.

Adding Floating-Point Numbers

IEEE 754 single precision has so many bits to work with, that it's simply easier to explain how floating-point addition works using a small floating-point representation. Addition is simple. Suppose you want to add two floating point numbers, X and Y. For sake of argument, assume the exponent in Y is less than or equal to the exponent in X. Let the exponent of Y be y and let the exponent of X be x. Here's how to add floating-point numbers.

1. First, convert the two representations to scientific notation. Thus, we explicitly represent the hidden 1.
2. In order to add, we need the exponents of the two numbers to be the same. We do this by rewriting Y. This will result in Y being not normalized, but value is equivalent to the normalized Y.

3. Add x – y to Y's exponent. Shift the radix point of the mantissa (signficand) Y left by x – y to compensate for the change in exponent.
4. Add the two mantissas of X and the adjusted Y together.
5. If the sum in the previous step does not have a single bit of value 1, left of the radix point, then adjust the radix point and exponent until it does.
6. Convert back to the one byte floating point representation.

Example: Let's add the following two numbers:

Variable	Sign	Exponent	Fraction
X	0	1001	110
Y	0	0111	000

Here are the steps:

1. In normalized scientific notation, X is 1.110×2^2, and Y is 1.000×2^0.
2. In order to add, we need the exponents of the two numbers to be the same. We do this by rewriting Y. This will result in Y being not normalized, but value is equivalent to the normalized Y.
3. The difference of the exponent is 2. So, add 2 to Y's exponent, and shift the radix point left by 2. This results in 0.0100×2^2. This is still equivalent to the old value of Y. Call this readjusted value, Y′.
4. We add $(1.110)_2$ to $(0.01)_2$. The sum is: $(10.0)_2$. The exponent is still the exponent of X.
5. In this case, the sum, $(10.0)_2$, has two bits left of the radix point. We need to move the radix point left by 1, and increase the exponent by 1 to compensate. This results in: 1.000×2^3.
6.

Sum	Sign	Exponent	Fraction
X+Y	0	1010	000

Addition of Negative Values

In hardware, you would probably convert the mantissas to two's complement, and perform the addition, while keeping track of the radix point.

Overflow/Underflow

It's possible for a result to overflow (a result that's too large to be represented) or underflow (smaller in magnitude than the smallest denormal, but not zero). Real hardware has rules to handle this.

6.5 Multiplying Floating-Point Numbers

Suppose you want to multiply two floating-point numbers, X and Y. Here's how to multiply floating-point numbers.

1. First, convert the two representations to scientific notation. Thus, we explicitly represent the hidden 1.
2. Let x be the exponent of X. Let y be the exponent of Y. The resulting exponent (call it z) is the sum of the two exponents. z may need to be adjusted after the next step.
3. Multiply the mantissa of X to the mantissa of Y. Call this result m.
4. If m is does not have a single 1 left of the radix point, then adjust the radix point so it does, and adjust the exponent z to compensate.
5. Add the sign bits, mod 2, to get the sign of the resulting multiplication.

6. Convert back to the one byte floating point representation, truncating bits if needed.

Example: Let's multiply the following two numbers:

Variable	Sign	Exponent	Fraction
X	0	1001	010
Y	0	0111	110

Here are the steps:

1. In this case, X is 1.01×2^2 and Y is 1.11×2^0.
2. The resulting exponent is 2 + 0 = 2
3. Multiplying 1.01 by 1.11 results in 10.0011.
4. Now, we have to renormalize 10.0011 to 1.00011 and increase the exponent by 1 to 3 .
5. The sign bit is 0 + 0 = 0.
6. We need to truncate 1.00011×2^3 to 1.000×2^3 and convert.

Product	Sign	Exponent	Fraction
X×Y	0	1010	000

Negative Values

Unlike floating-point addition, negative values are simple to take care of in floating-point multiplication. Treat the sign bit as 1 bit unsigned binary, and add modulo 2. This is the same as XORing the sign bit.

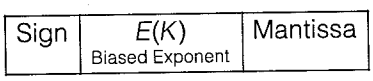
Summary

- Fixed Point Signed Numbers: The fixed point signed numbers are denoted using sign magnitude notation, 1's and 2's complement notation.
  - (*i*) *Signed Magnitude Form :*
    - Sign bit is considered explicitly, hence additional hardware is required for resultant sign of arithmetic.
    - Addition and substraction are performed on seperate hardware.
    - 0 has 2 representations i.e., +0 : 0000 0000
      −0 : 1000 0000
  - (*ii*) *1's Complement Form :*
    - Sign bit is not considered explicitly hence no additional hardware required.
    - 0 has 2 representations i.e., +0 : 0000 0000
      −0 : 1111 1111
    - Non weighted system.
  - (*iii*) *2's Complement Form :*
    - Sign bit is not considered explicitly.
    - Addition and subtractions are performed by using adder only.

$$-B = \bar{B} + 1$$
$$A - B = A + \bar{B} + 1$$

    - 0 has only 1 representation.
    - Only code that assigns negative weight to the sign bit.
- The floating numbers are stored in mantissa exponent form. Any floating point number is represented in the form $\pm m \times r^{\pm e}$.

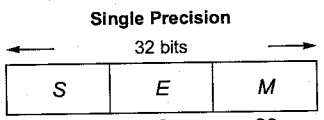
- Any floating point number is represented in "normalised" form. In a normalised floating point number the MSB of mantissa must be non-zero.
- Zero cannot be normalized.
- The value of expression is given by :
  (*i*) Explicit Normalization : $V = (-1)^{sign}\,[0.M]_2 \times 2^{E-Bias}$
  (*ii*) Implicit Normalization : $V = (-1)^{sign}\,[1.M]_2 \times 2^{E-Bias}$
- The biased exponent is an unsigned number which can represent signed exponent of original number.
- True Exponent = Biased Exponent – Bias



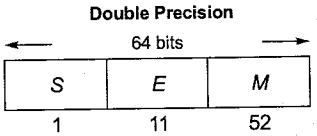
Bias = 2^{K-1}

$0 \le E \le 2^K - 1$, Bias = 2^{K-1}

- In IEEE 754 the base of the system is 2, there is a provision for the values ±0 and ±∞, the floating point number is stored either with single precision or with double precision, certain mantissa exponent combinations does not represent any number representing NAN, and The value represented in single precision.



$0 \le E \le 255$



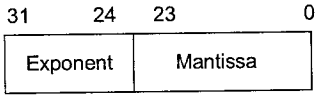
$0 \le E \le 2047$

Student's Assignments

Q.1 In IEEE floating point representation, all the exponent bits are 1 and mantissa bits non-zero. This represents
(a) 0
(b) infinity
(c) denormalized number
(d) Error

Q.2 Which of the following statement is **Incorrect** for the range of *n* bits binary numbers?
(a) Range of unsigned numbers is 0 to $2^n - 1$.
(b) Range of signed numbers is $-2^{n-1} + 1$ to $2^{n-1} - 1$
(c) Range of signed 1's compliment numbers is $-2^{n-1} + 1$ to 2^{n-1}
(d) Range of signed 2's compliment numbers is -2^{n-1} to $2^{n-1} - 1$

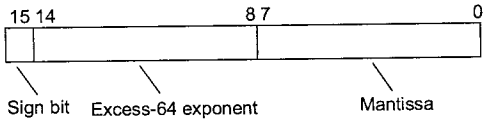
Q.3 Consider the following floating point number representation.



The exponent is in 2's complement representation and mantissa is in the sign magnitude representation. The range of the magnitude of the normalized numbers in this representation is
(a) 0 to 1 (b) 0.5 to 1
(c) 2^{-23} to 0.5 (d) 0.5 to $(1 - 2^{-23})$

Common Data for Q.4 & Q.5

Consider the following floating point format



Mantissa is a pure fraction is sign-magnitude form.

Q.4 The decimal number 0.239×2^{13} has the following hexadecimal representation without normalization and rounding off

(a) 0D 24 (b) 0D 4D
(c) 4D 0D (d) 4D 3D

Q.5 The normalized representation for the above format is specified as follows. The mantissa has an implicit 1 preceding the binary (radix) point. Assume that only 0's are padded in while shifting a field. The normalized representation of the above number (0.239×2^{13}) is

(a) 0A 20 (b) 11 34
(c) 4D D0 (d) 4A E8

Q.6 The following bit pattern represents a floating point number in IEEE 754 single precision format:
1 10000011 10100000000000000000000
The value of the number in decimal form is

(a) –10 (b) –13
(c) –26 (d) None of these

Q.7 In the IEEE floating point representation the hexadecimal value 0x00000000 corresponds to

(a) the normalized value 2^{-127}
(b) the normalized value 2^{-126}
(c) the normalized value +0
(d) the special value +0

Answer Key:

1. (d) **2.** (c) **3.** (d) **4.** (d) **5.** (d)
6. (c) **7.** (d)