UNIT 1 — Number Systems, Binary Arithmetic & Logic Gates (CO1)

1.1 Number Systems: Binary, Octal, Hexadecimal

Binary Number System

- Base = **2**, Digits = {0, 1}.
- Each position represents increasing powers of 2.

Octal Number System

- Base = **8**, Digits = {0–7}.
- Useful because **3 binary bits = 1 octal digit**.

Hexadecimal Number System

- Base = **16**, Digits = {0–9, A–F}.
- **4 bits = 1 hex digit**.

1.2 Binary Number Representation

A) Sign Magnitude Representation

- MSB = sign bit
  - 0 → positive
  - 1 → negative
- Remaining bits → magnitude

- Example: 8-bit
  - +13 → 00001101
  - –13 → 10001101

B) 1’s Complement Representation

- Negative numbers are formed by **inverting all bits** of the positive number.
- Example (8-bit):
  - +13 → 00001101
  - –13 → 11110010

Issues:

- Two representations of zero (+0 and –0)

C) 2’s Complement Representation

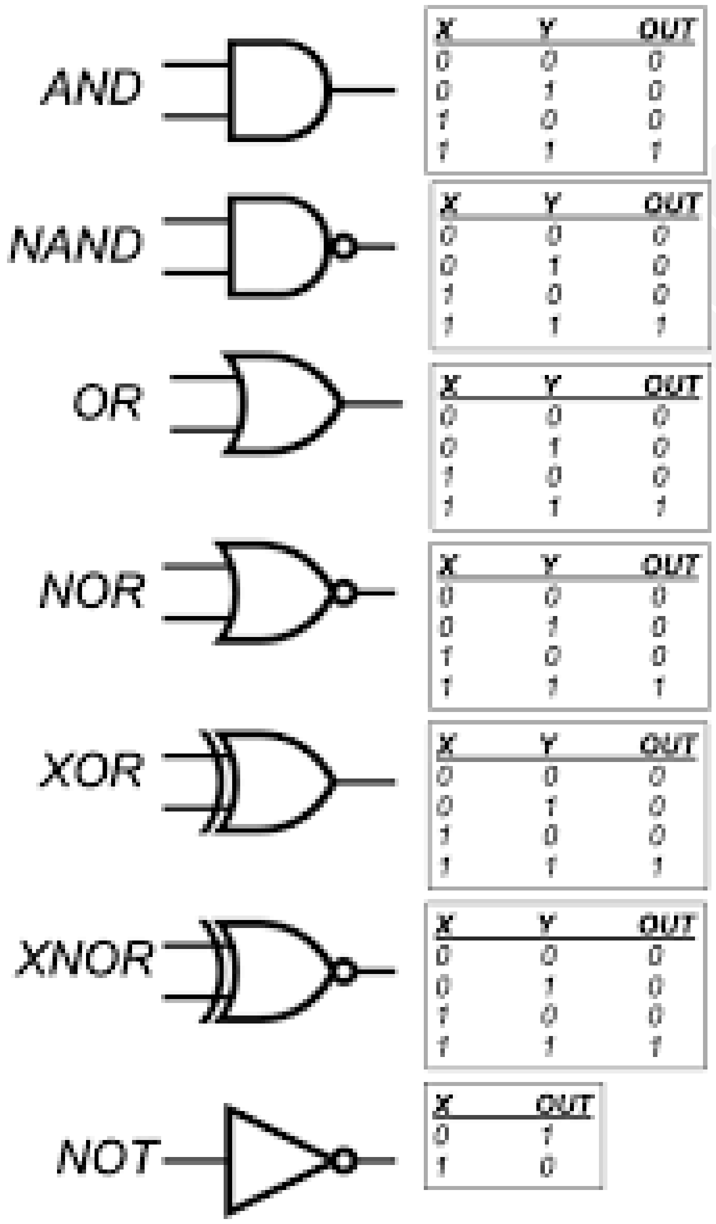
- Most widely used.
- Steps for negative number:
  - Take 1’s complement
  - Add 1
- Example:
  - +13 → 00001101
  - 1’s complement → 11110010
  - Add 1 → 11110011 (–13)

Advantages:

- Single representation for zero
- Easy for addition/subtraction

1.3 Basic Logic Gates

Gate	Operation	Symbol	Boolean Expression
AND	All inputs 1 → output 1	·	Y = A·B
OR	Any input 1 → output 1	+	Y = A+B
NOT	Inverts input	‘	Y = A’



X	Y	OUT
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	OUT
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	OUT
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	OUT
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	OUT
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	OUT
0	0	0
0	1	0
1	0	0
1	1	1

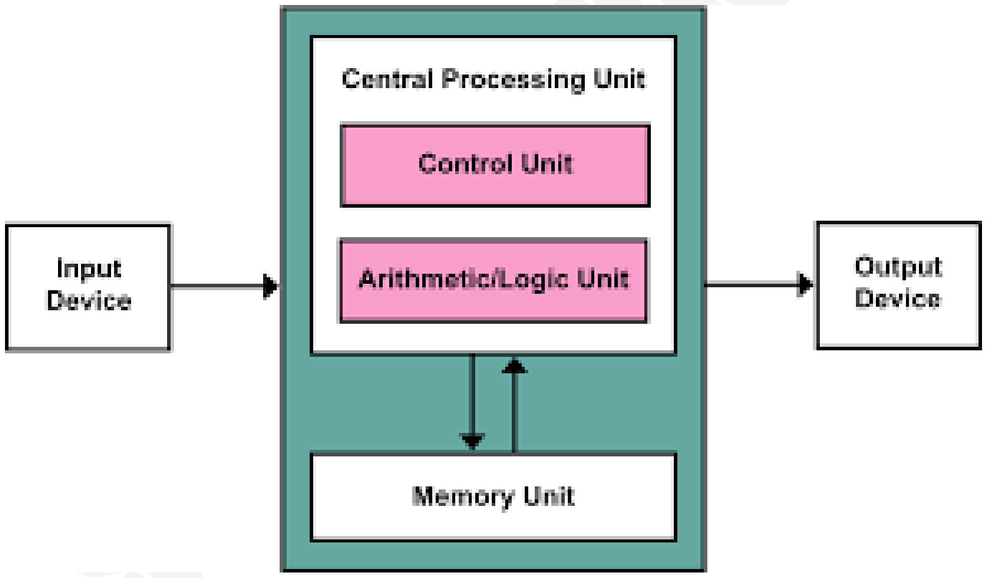
X	OUT
0	1
1	0

1.4 Van Neumann Architecture

4.1 Basic Organization of a Computer

A computer consists of:

1. **Input Unit**
2. **Central Processing Unit (CPU)**
    - ALU
    - Control Unit
3. **Memory Unit**
4. **Output Unit**



- **Stored Program Concept**
  Both instructions and data reside in the **same memory**.
- **Sequential instruction execution**
  Unless a branch/jump occurs.
- **Single bus system** shared for data & instructions.

UNIT 2 — ALU Operations, Booth's Algorithm, Division Algorithms & IEEE 754

1.1 Introduction

The **Arithmetic Logic Unit (ALU)** is the subsystem of the CPU responsible for executing:

- Arithmetic operations
- Logical operations
- Shift/rotate operations
- Comparison operations
- Boolean functions

The ALU has:

- Two operand inputs (A, B)
- Control lines (select operation)
- Output
- Status flags (Zero, Carry, Sign, Overflow)

2.1 Binary Addition

Binary Addition Rules

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
   10101
 + 11010
 ----------
  101111
```

2.2 Binary Subtraction

Binary subtraction is easier using **2’s complement**.

Steps:

1. Take 2’s complement of the subtrahend
2. Add to the minuend
3. If carry out → discard
4. If no carry → answer is negative, take 2’s complement

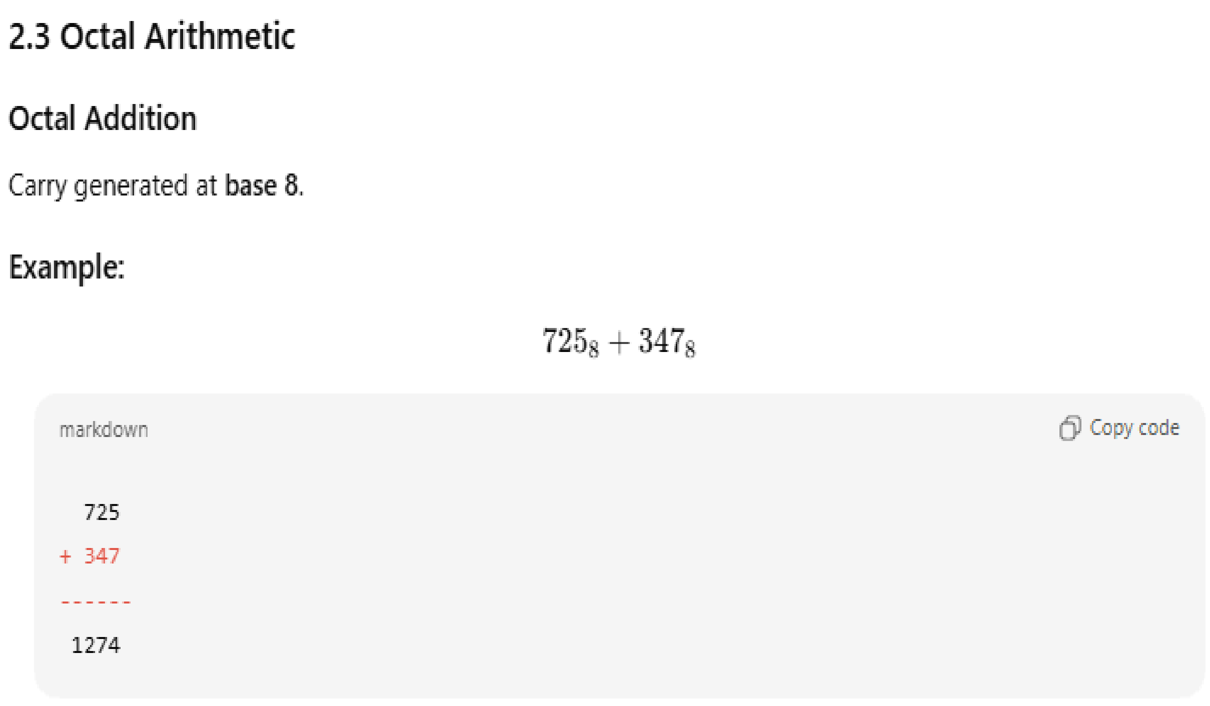
Example 2: 25 – 18

25 = 11001
18 = 10010

1. Take 2’s complement of 10010 → 01101 + 1 = 01110

2. Add:

```
  11001
+ 01110
 --------
 100111
```



2.3 Octal Arithmetic

Octal Addition

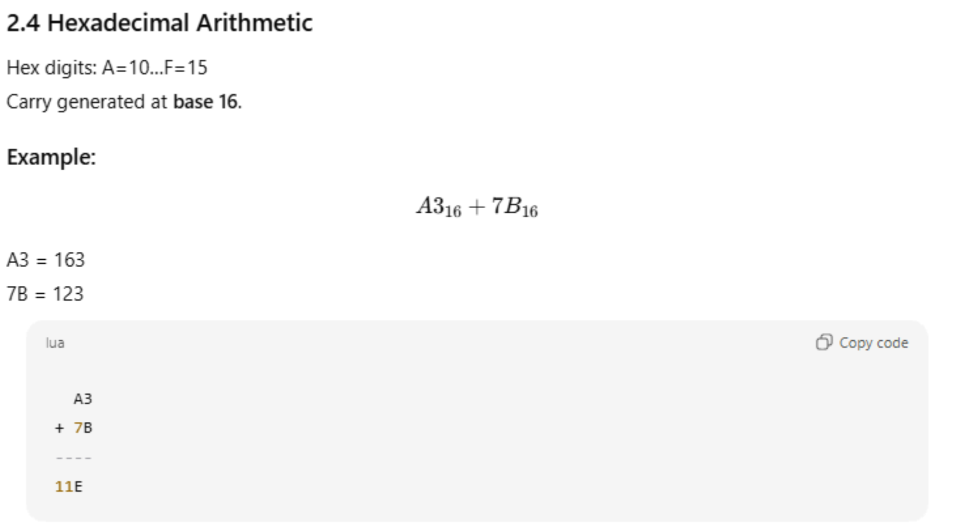
Carry generated at **base 8**.

Example:

$$725_8 + 347_8$$

markdown

```
  725
+ 347
------
 1274
```



2.4 Hexadecimal Arithmetic

Hex digits: A=10…F=15
Carry generated at **base 16**.

Example:

$$A3_{16} + 7B_{16}$$

A3 = 163
7B = 123

lua

```
  A3
+ 7B
----
11E
```

3.1 Booth’s Multiplication Algorithm

Most famous algorithm for **signed binary multiplication**.

Purpose

- Fast multiplication
- Handles **positive & negative numbers**

- Minimizes addition/subtraction operations

3.1.1 Working Principle

Booth examines the multiplier bits in **pairs**:

Pair (Q0 Q-1)	Operation
00	No operation
01	Add Multiplicand to A
10	Subtract Multiplicand from A
11	No operation

3.1.2 Booth Algorithm Steps

Let:

- M = multiplicand
- Q = multiplier
- A = accumulator
- Q-1 = extra bit
- n = number of bits

Steps:

1. Initialize A = 0, Q-1 = 0
2. Check pair (Q0, Q-1) → apply rules
3. Perform add/subtract
4. Arithmetic Right Shift (A, Q, Q-1)
5. Repeat n times

Booth's Algorithm With Example | booths | booths algo

4. Division Algorithms

Two types of hardware division algorithms:

4.1 Restoring Division Algorithm

Concept

After subtraction:

- If remainder < 0 → **restore** previous value
- Put quotient bit = 0
- Else quotient bit = 1

4.1.1 Steps

1. Initialize Remainder = 0
2. Left shift (R, Q)
3. R = R – Divisor
4. If R < 0 → restore and set Q0 = 0
5. Else set Q0 = 1
6. Repeat n times

4.1.2 Example (Restoring Division)

Divide: 13 / 3

13 = 1101
3 = 0011

Perform shifts and subtractions — result:

Quotient = **0100 = 4**
Remainder = **1**

---

5. Non-Restoring Division Algorithm

Concept

Avoids the restoring step to make division faster.

Rules:

- If previous remainder ≥ 0 → subtract
- If previous remainder < 0 → add divisor

Final Step:

If remainder < 0 → add divisor

5.1 Non-Restoring Example

Divide 13/3:

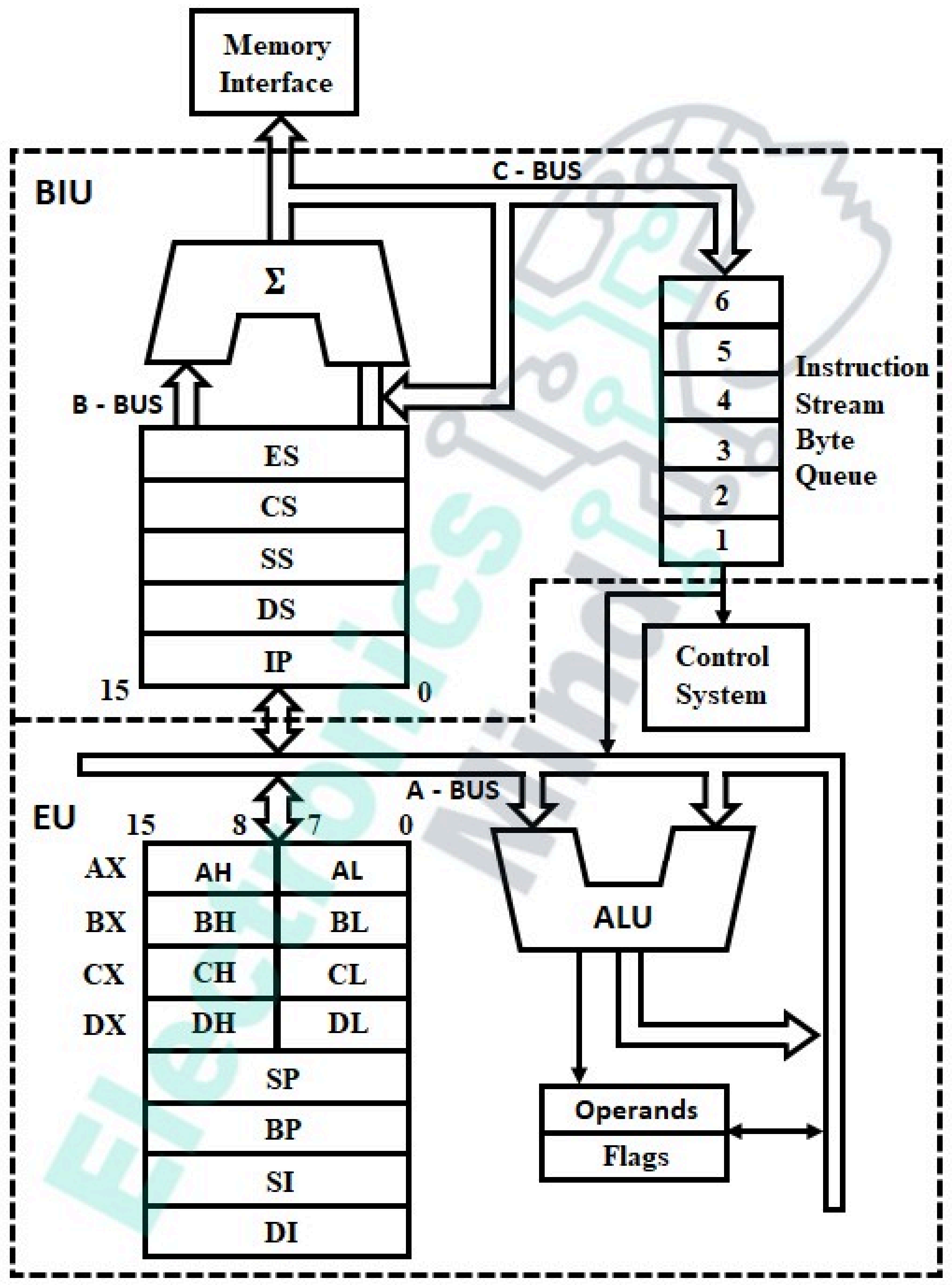
Follow rules →
 Quotient = 0100
 Remainder = 1

Produces same result.

3. Binary Division method (Restoring and Non-restoring Division Algorithm)

3.1 8086 Processor

3.1.1 Architecture of 8086 Processor



8086 Internal Architecture

8086 is a **16-bit CISC microprocessor** with a **20-bit address bus**, meaning it can access **2^{20} = 1 MB memory**.

Block Diagram of 8086 (Explained)

8086 is divided into **two main units**:

A. Bus Interface Unit (BIU)

Responsible for interaction with memory & I/O.

Functions of BIU

- Fetches instructions from memory
- Computes physical addresses
- Handles instruction queue (Pipeline)
- Drives address & data buses

Components of BIU

1. **Segment Registers**
   - **CS** – Code Segment
   - **DS** – Data Segment
   - **SS** – Stack Segment
   - **ES** – Extra Segment
   -
2. **Instruction Pointer (IP)**
   - Holds offset address of next instruction to execute.

3. **Instruction Queue (6-byte)**
   - Prefetches instructions → forms a **pipeline** to improve speed.

4. **Address Generation Circuit**
   - Computes physical address using:
     Physical Address = (Segment Register × 16) + Offset

B. Execution Unit (EU)

Responsible for *decoding* and *executing* instructions.

Components of EU

1. **General Purpose Registers**
   - AX = Accumulator
   - BX = Base
   - CX = Count
   - DX = Data

2. **Pointer Registers**
   - SP (Stack Pointer)

    - BP (Base Pointer)

3. **Index Registers**
    - SI (Source Index)
    - DI (Destination Index)

4. **ALU – Arithmetic Logic Unit**
    - performs operations: ADD, SUB, INC, DEC, AND, OR, XOR etc.

5. **Flag Register (Status Flags)**
    - CF – Carry
    - PF – Parity
    - AF – Auxiliary carry
    - ZF – Zero
    - SF – Sign
    - OF – Overflow

Instruction Cycle

Instruction cycle includes:

1. **Fetch**
    - BIU fetches from memory → instruction queue.
2. **Decode**
    - EU decodes instruction.
3. **Execute**
    - EU uses ALU to perform action.
4. **Write Back**
    - Result stored in register/memory.
5. **Next Instruction**
    - IP is updated.

8086 pipeline:
While **EU executes**, **BIU fetches next instruction**.

Addressing Modes of 8086 (One-Line + Example)

1. Immediate Addressing

Operand is a constant value inside the instruction.
Example: `MOV AL, 25H`

---

2. Register Addressing

Operand is stored in a register.
Example: `MOV AX, BX`

---

3. Direct Addressing

Memory address is directly given in the instruction.
Example: `MOV AX, [5000H]`

---

4. Register Indirect Addressing

Memory address is taken from a register (BX, BP, SI, DI).
Example: `MOV AX, [BX]`

---

5. Based Addressing

Base register + displacement forms the memory address.
Example: `MOV AX, [BX + 04H]`

---

6. Indexed Addressing

Index register + displacement gives the memory address.
Example: `MOV AX, [SI + 05H]`

---

7. Based Indexed Addressing

Base register + index register used together for memory address.
Example: `MOV AX, [BX + SI]`

---

8. Based Indexed with Displacement

Base register + index register + displacement.
Example: `MOV AX, [BX + SI + 10H]`

---

9. Relative Addressing

Uses IP + displacement for jumps.
Example: `JMP +5`

---

10. Immediate to Memory

Immediate data sent directly to memory location.
Example: `MOV [5000H], 25H`

---

11. Memory Indirect (through pointer)

Memory contains an address pointing to another memory location.
Example: `MOV AX, [ [BX] ]` *(conceptual)*

---

12. Implicit Addressing

Operand is implied by the instruction.
Example: `CLC` (Clear Carry Flag)

CISC Architecture

Used in 8086, Intel processors.

Features

- Complex & large instruction set
- Supports multiple addressing modes
- Microprogrammed control unit
- Slower decoding
- Larger instructions
- Example: x86

RISC Architecture

Used in ARM, MIPS, SPARC.

Features

- Small instruction set
- Only simple instructions
- Hardwired control unit
- Fast instruction cycle
- Load–store architecture
- Fixed-length instructions

3.3.3 Key Differences

CISC	RISC
Many instructions	Few instructions
Complex instructions	Simple instructions
Many addressing modes	Few addressing modes
Microprogrammed control	Hardwired control
Slow execution	Fast execution
Variable length	Fixed length

RISC/CISC Design Issues

CISC Issues

- Complexity increases delay
- High power consumption

RISC Issues

- Requires more registers
- Larger program size

UNIT 4 – MEMORY SYSTEMS ORGANIZATION (DETAILED NOTES)

4.1 Introduction to Memory & Memory Parameters

Memory stores data and instructions required by the processor.

Key Memory Parameters

1. **Capacity**
    - Total number of bits/bytes that can be stored.
    - Example: 4GB, 8KB, 1MB
2. **Access Time (t_a)**
    - Time to read/write data after supplying address.
    - RAM ≈ 50–100 ns
    - Cache ≈ 1–5 ns
3. **Cycle Time (t_C)**
    - Minimum time between two consecutive accesses.
4. **Bandwidth**
    - Amount of data transferred per second.
5. **Latency**
    - Delay before data transfer begins.
6. **Memory Word Size**
    - Number of bits per memory location (8, 16, 32 bits).

4.2 Classification of Primary & Secondary Memory

Primary Memory (Main Memory)

- Directly accessible by CPU
- Fast but expensive
- Volatile

Examples:

- RAM (DRAM, SRAM)
- Cache
- Registers

Secondary Memory

- Not directly accessed by CPU
- Non-volatile
- Used for long-term storage

Examples:

- HDD, SSD
- USB, CD/DVD

4.3 Types of RAM and ROM

4.3.1 RAM (Random Access Memory)

Volatile memory used for temporary program/data storage.

Types of RAM

A. SRAM – Static RAM

- Uses flip-flops
- Very fast
- Used in cache memory
- Costly, low density

B. DRAM – Dynamic RAM

- Uses capacitors
- Slower than SRAM
- Needs refreshing
- Used in main memory

4.3.2 ROM (Read Only Memory)

Non-volatile memory used for firmware.

Types of ROM

1. **Mask ROM** – Permanently written at manufacture
2. **PROM** – Programmable once
3. **EPROM** – Erasable using UV light
4. **EEPROM** – Electrically erasable
5. **Flash Memory** – Modern form of EEPROM

4.4 Memory Hierarchy & Characteristics

Memory hierarchy places memory types based on:

- speed
- cost
- size

Hierarchy (Top → Bottom)

```
Registers (fastest, lowest capacity)
↓
```

```
Cache Memory (L1, L2, L3)
↓
Main Memory (RAM)
↓
Secondary Storage (SSD/HDD)
↓
Archival Storage (Tape, Cloud)
```

Characteristics

- **Speed** ↓ as we go down
- **Capacity** ↑
- **Cost per bit** ↓

Underlying principle: **locality of reference**

- **Temporal locality:** recently accessed data is used again
- **Spatial locality:** nearby addresses accessed together

4.5 Virtual Memory

Virtual Memory allows programs larger than main memory to run.

4.5.1 Paging

- Memory is divided into **fixed-size pages** (4 KB typical)
- Logical address → Page Number + Offset
- Page Table maps logical pages to physical frames
- Removes external fragmentation

4.5.2 Segmentation

- Memory is divided into **variable-sized segments**
- Segments represent logical units:
    - code

- data
- stack

Address = Segment Number + Offset

Used where logical program structure is important.

4.6 Cache Memory

Cache is high-speed memory between CPU & RAM.

4.6.1 Cache Levels

- **L1 Cache** – smallest & fastest
- **L2 Cache** – larger, slower
- **L3 Cache** – shared among cores, large

4.6.2 Cache Mapping Techniques

Cache uses mapping to place data from RAM:

1. Direct Mapping

- Each block of RAM → mapped to exactly one cache line
- Simple & cheap
- Conflict misses are common

Formula:
Cache Line = (Main Memory Block No.) mod (Number of Cache Lines)

---

2. Associative Mapping

- Any block can be placed anywhere in cache
- Fully flexible
- But expensive (needs parallel search)

---

3. Set Associative Mapping

- Compromise of above two
- Cache is divided into sets (2-way, 4-way, etc.)
- Block can go into any line of its set
- Most widely used

4.7 Cache Coherency

In multiprocessor systems, copies of data may exist in different caches.

Cache Coherency Problem:
When one core modifies data, other caches must update or invalidate their copies.

4.8 MESI Protocol (Self-Study Topic)

MESI maintains cache consistency using 4 states:

1. **Modified (M)** – Updated in cache, not in RAM
2. **Exclusive (E)** – Present only in this cache, same as RAM
3. **Shared (S)** – Present in multiple caches
4. **Invalid (I)** – Data invalid or outdated

Widely used in Intel Pentium processors.

4.9 Interleaved & Associative Memory

A. Interleaved Memory

Divides memory into multiple banks to increase speed.

- Parallel access possible

- Reduces wait time

B. Associative Memory (Content-Addressable Memory – CAM)

- Accessed by **content**, not by address
- Used in TLB (Translation Lookaside Buffer)

UNIT 5 – I/O ORGANIZATION (DETAILED NOTES)

5.1 Introduction to I/O Systems

I/O (Input/Output) systems allow the CPU to communicate with external devices such as:

- Keyboard
- Mouse
- Hard Disk
- Display
- Network Devices

Since I/O devices are slower than CPU, special mechanisms are required for communication.

5.2 Buses

A **bus** is a collection of wires that transmit data, addresses, and control signals between components.

5.2.1 Types of Buses

1. Data Bus

- Carries actual data between CPU, memory, and I/O
- Bi-directional
- Size determines data transfer width (8, 16, 32, 64 bits)

2. Address Bus

- Carries address of memory or I/O location
- Uni-directional
- Width determines maximum addressable memory
  (Ex: 32-bit address bus → 4GB)

3. Control Bus

Carries control & timing signals such as:

- Read
- Write
- Interrupt request
- Acknowledge
- Clock
- Memory/I/O select

5.2.2 Bus Arbitration

When multiple devices want to use the bus at the same time, arbitration decides which device gets control.

Methods of Bus Arbitration

---

1. Daisy Chain Arbitration

- Devices connected in chain form
- Priority decreases from head to tail
- Simple but priority fixed

---

2. Centralized Arbitration

A controller decides the priority.

Types:

- **Parallel Priority** (faster)
- **Polling** (slower)

---

3. Distributed Arbitration

No central controller; all devices negotiate using a bus request protocol.
Used in advanced multiprocessor systems.

5.2.3 Bus Standards & Comparative Study

Common standards:

Standard	Use
PCI	Modern PC expansion bus
PCI Express (PCIe)	High-speed serial bus
USB	Universal peripheral connection
SATA	Hard disk/SSD connection
ISA	Old PC bus

Key Comparison Factors:

- Speed
- Device support
- Parallel vs serial
- Hot-plugging ability
- Cost

PCIe is currently the fastest and most used standard.

5.3 I/O Interface

I/O Interface is the hardware required to connect I/O devices with the CPU.

Functions

- Device communication
- Data buffering
- Address decoding
- Handshaking
- Command execution

Types of Interfaces

1. Memory-Mapped I/O

- I/O devices share the same address space as memory
- CPU uses memory instructions to access I/O
- Larger address space needed

Example:
`MOV AL, [5000H]` could refer to a device port.

---

2. I/O-Mapped (Isolated) I/O

- Separate address space for I/O
- Special instructions used: **IN**, **OUT**

Example:
`IN AL, 60H` (read from keyboard port)

5.4 I/O Channels, I/O Modules & I/O Processors

I/O Module

An electronic circuit between CPU and I/O device.

Functions

- Data conversion
- Error detection
- Device control
- Timing
- Buffering

I/O Channel

A special processor dedicated to I/O operations.

Types

- **Selector Channel** – dedicated high-speed devices
- **Multiplexer Channel** – interleaves slow devices

I/O Processor (IOP)

Independent processor handling I/O tasks without CPU intervention.

Features

- Has own instruction set
- Parallel execution with CPU
- Used in mainframes and high-speed servers

5.5 Data Transfer Techniques

This is one of the most important exam topics.

---

1. Programmed I/O

CPU actively waits for I/O device (busy waiting).

Steps

1. CPU checks device status
2. If ready → CPU reads/writes data
3. CPU repeats for every transfer

Advantages

- Simple to implement

Disadvantages

- CPU is heavily wasted
- Very slow

---

2. Interrupt Driven I/O

Device interrupts the CPU when ready.

Steps

1. CPU gives command
2. CPU continues work
3. When device ready → interrupts CPU
4. CPU transfers data using ISR (Interrupt Service Routine)

Advantages

- CPU is free to execute other tasks
- Efficient for moderate speed devices

3. Direct Memory Access (DMA)

Special hardware (DMA controller) transfers data directly between memory and device.

CPU not involved after starting DMA.

Steps

1. CPU gives DMA controller the source, destination, size
2. DMA takes control of bus
3. Transfers data directly
4. Signals CPU via interrupt when done

Advantages

- Very fast
- CPU fully free
- Used for disk, graphics, network transfers

DMA Modes

- **Burst mode**
- **Cycle stealing**
- **Transparent DMA**

UNIT 6 – PARALLEL PROCESSING (DETAILED NOTES)

(Advanced Processor Models, Pipelining, Superscalar, Hazards, Amdahl’s Law, Flynn’s Classification, etc.)

1. ADVANCED PROCESSOR MODELS – 80386DX MODES

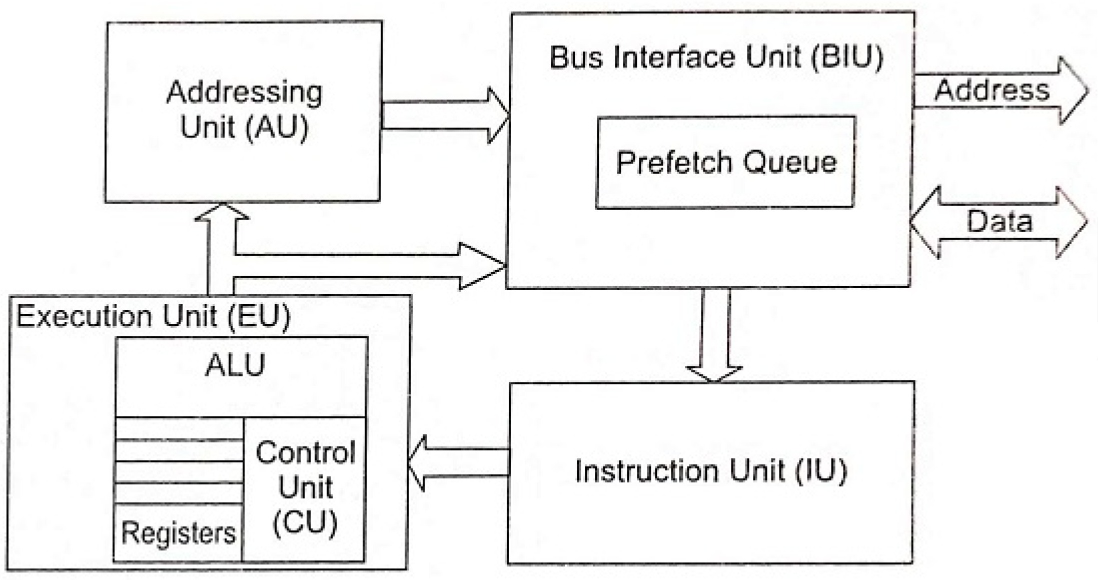


Fig. 11.31 *The simplified block diagram of 80386 processor*

The **Intel 80386DX** processor supports **three operating modes**, each providing different levels of memory protection, multitasking ability, and privilege.

1.1 Real Mode

- Works like an 8086 processor.
- Uses **20-bit addressing** → can access **1 MB memory**.
- No memory protection, no multitasking.
- Segmentation is simple; segments can overlap.

Use: Booting, running DOS programs.

1.2 Protected Mode

- Main operating mode of 80386.

- Supports **32-bit addressing** → access to **4 GB physical memory**.
- Provides **segmentation + paging** for memory protection.
- Supports **multitasking** and privilege levels (Ring 0–3).

Use: Modern operating systems (Windows, Linux).

1.3 Virtual 8086 Mode

- Allows the processor to run multiple **8086 (Real Mode) programs** in protected mode.
- Each program runs as a **virtual DOS machine**.
- Paging is used to isolate each virtual machine.

Use: Running DOS apps under Windows/Linux environments.

---

2. PIPELINED ARCHITECTURE

Pipelining splits instruction execution into multiple stages so different instructions can be processed simultaneously.

Analogy: Assembly line in a factory.

2.1 Pipeline Stages (Typical 5-Stage RISC Pipeline)

Stage	Name	Description
IF	Instruction Fetch	Fetch instruction from memory
ID	Instruction Decode	Decode instruction & read registers
EX	Execute	Perform ALU operation
MEM	Memory Access	Read/write data from memory
WB	Write Back	Write result to register

This improves **instruction throughput** but NOT the execution time of a single instruction.

---

3. SUPERSCALAR ARCHITECTURE

A **superscalar processor** issues and executes **multiple instructions per clock cycle** using multiple execution units.

Features:

- Instruction-level parallelism (ILP).
- Multiple pipelines.
- Dynamic scheduling.
- Out-of-order execution.

Example: Pentium Processor, modern Intel & AMD CPUs.

---

4. PIPELINE HAZARDS

Hazards reduce the efficiency of pipelines by causing delays (stalls).

4.1 Structural Hazards

- Hardware resource conflict.
- Two instructions need the same hardware at the same time.

Example: Single memory module for data and instructions → conflict.

4.2 Data Hazards

Caused by dependency between instructions.

Types:

- **RAW (Read After Write)** → true dependency
- **WAR (Write After Read)**
- **WAW (Write After Write)**

Example:

```
I1: R1 = R2 + R3
I2: R4 = R1 + 5   ← depends on result of I1
```

4.3 Control Hazards

- Occur due to **branch or jump instructions**.
- Next instruction is unknown until the branch is resolved.

---

5. MITIGATION TECHNIQUES

5.1 Branch Prediction

Predicts whether a branch will be taken or not.

Types:

- Static prediction (fixed rule)
- Dynamic prediction (history-based)
- 1-bit/2-bit predictors
- Branch Target Buffer (BTB)

5.2 Data Forwarding (Bypassing)

ALU result is forwarded directly to the next instruction **before** writing it to the register file.

Prevents RAW hazards.

5.3 Pipeline Stalling

Inserts NOP cycles to wait for data availability.
Used only when forwarding cannot fix the hazard.

---

6. AMDAHL'S LAW

Defines the theoretical speedup of a system when a portion of it is enhanced (parallelized).

Formula:

$$\text{Speedup} = \frac{1}{(1-p) + \frac{p}{s}}$$

Where:

- **p** = portion of program that can be parallelized
- **s** = speedup of the improved part

Implication:
Even with infinite processors, speedup is limited by the **sequential part**.

---

7. INTRODUCTION TO PARALLEL PROCESSING

Parallel processing refers to **simultaneous processing** of multiple tasks to improve performance.

Benefits:

- Higher speed
- Better throughput
- Efficient use of multi-core CPUs
- Large data processing (AI/ML, simulations)

Types of Parallelism:

- **Instruction-level** (ILP) – superscalar, pipelining
- **Data-level** (DLP) – vectors, GPUs
- **Thread-level** (TLP) – multicore CPUs
- **Task-level** – distributed systems

---

8. FLYNN'S CLASSIFICATION (Parallel Architectures)

Flynn classifies computers based on **number of instruction streams** and **data streams**.

Type	Full Form	Meaning
SISD	Single Instruction, Single Data	Traditional sequential computer
SIMD	Single Instruction, Multiple Data	Same instruction on multiple data (Vector CPUs, GPUs)
MISD	Multiple Instruction, Single Data	Rarely used; fault-tolerant systems
MIMD	Multiple Instruction, Multiple Data	Multicore processors, Multiprocessor systems

Most modern computers are **MIMD**.

---

9. SELF-STUDY TOPICS (NOTES)

(Short summaries as per exam expectations)

9.1 Superscalar Architecture – Pentium Case Study

- Pentium uses **dual pipelines (U-pipe and V-pipe)** → can execute 2 instructions per cycle.
- Supports **branch prediction**, **out-of-order execution**, **register renaming**.
- First Intel chip to implement superscalar fully.

9.2 GPGPU Architecture

(GPU used for general-purpose computing)

- Based on **SIMD** and **massive parallelism**.
- Thousands of lightweight cores.
- Great for AI, ML, matrix operations, image processing.