# Overview of Computer Architecture & Organization

# MODULE -2

Introduction of Computer Organization and Architecture. Basic organization of computer and block level description of the functional units. Evolution of Computers, Von Neumann model. Performance measure of Computer Architecture, Amdahl's Law Architecture of 8086 Family, Instruction Set, Addressing Modes, Assembler Directives, Mixed Language Programming, Stack, Procedure, Macro.

## Computer Architecture & Organization

- **Computer architecture** refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program.
- **Computer organization** refers to the operational units and their interconnections that realize the architectural specifications. Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory. Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.

## Computer Architecture vs Computer Organization

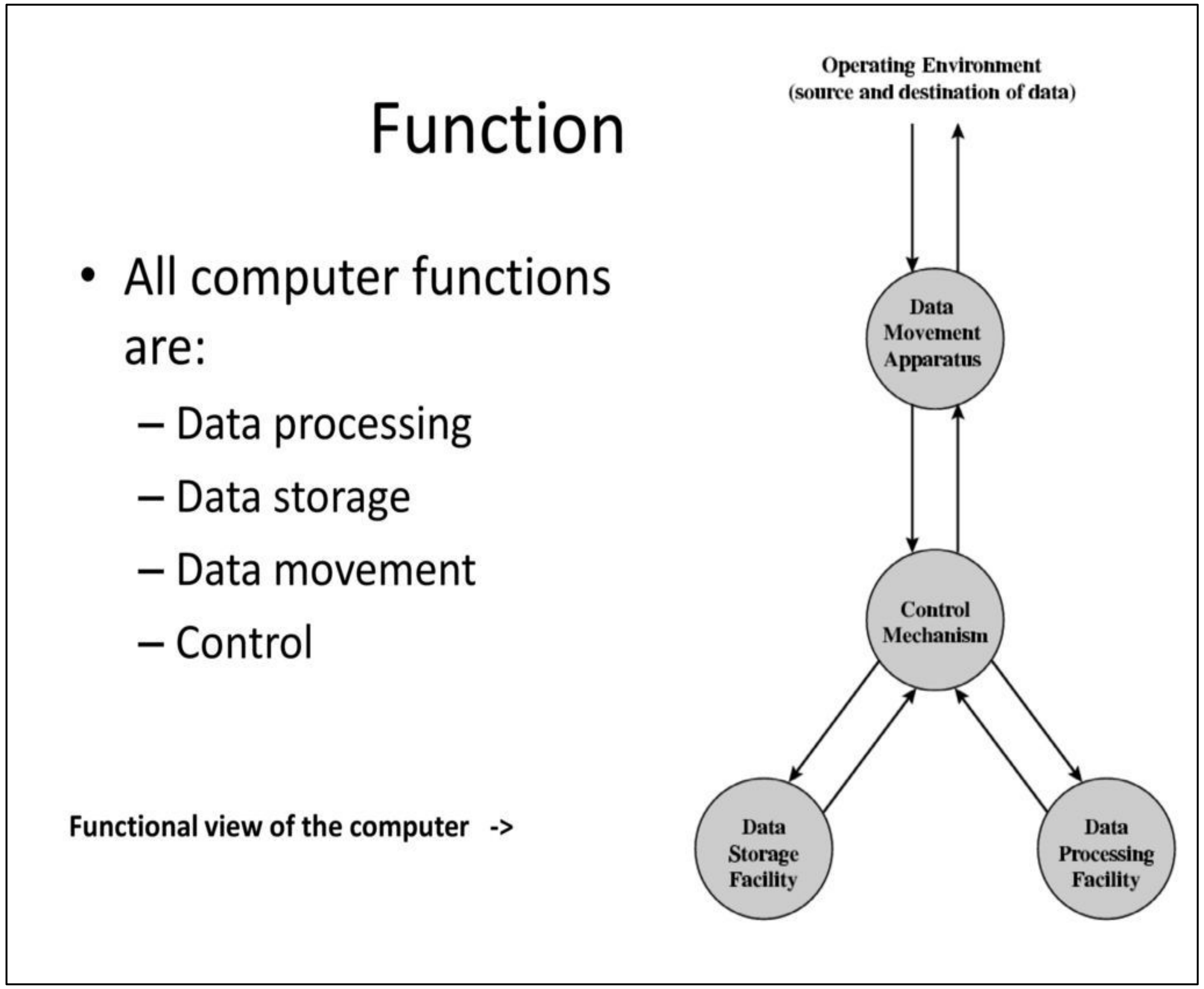
- Computer organization
  - Encompasses all physical aspects of computer systems.
  - E.g., circuit design, control signals, memory types.
  - *How does a computer work?*
- Computer architecture
  - Logical aspects of system implementation as seen by the programmer.
  - E.g., instruction sets, instruction formats, data types, addressing modes.
  - *How do I design a computer?*

## Introduction

- A computer is a complex system; contemporary computers contain millions of elementary electronic components. How, then, can one clearly describe them?
- The key is to recognize the hierarchical nature. A hierarchical system is a set of interrelated subsystems.
- The hierarchical nature of complex systems is essential to both their design and their description. At each level, the designer is concerned with structure and function
- **Structure:** The way in which the components are interrelated.
- **Function:** The operation of each individual component as part of the structure.

## Function

- We begin with the major components of a computer, describing their structure and function, and proceed to successively lower layers of the hierarchy



## Function

- All computer functions are:
  - Data processing
  - Data storage
  - Data movement
  - Control

Functional view of the computer ->

## Types of Functions

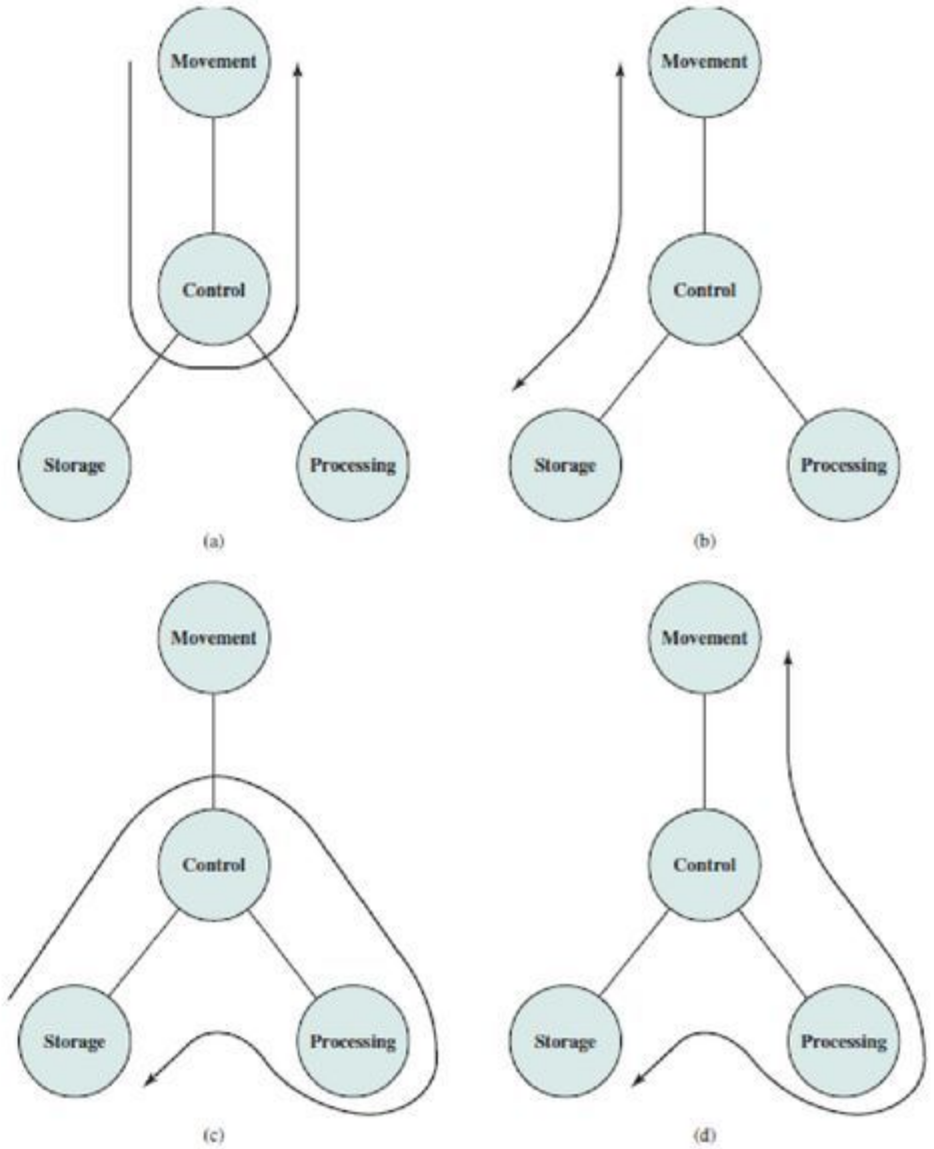
- The computer, of course, must be able to **process data**. The data may take a wide variety of forms, and the range of processing requirements is broad.
- It is also essential that a computer **store data**. Even if the computer is processing data on the fly (i.e., data come in and get processed, and the results go out immediately), the computer must temporarily store at least those pieces of data that are being worked on at any given moment. Thus, there is at least a short-term data storage function. Equally important, the computer performs a **long-term data storage function .**

## Types of Functions

- The computer must be **able to move** data between itself and the outside world. The computer’s operating environment consists of devices that serve as either sources or destinations of data.
- When data are received from or delivered to a device that is directly connected to the computer, the process is known as **input– output (I/O)**, and the device is referred to as a peripheral.
- When data are moved over longer distances, to or from a remote device, the process is known as **data communications**.
- Finally, there must be **control** of these three functions. Ultimately, this control is exercised by the individual(s) who provides the computer with instructions.

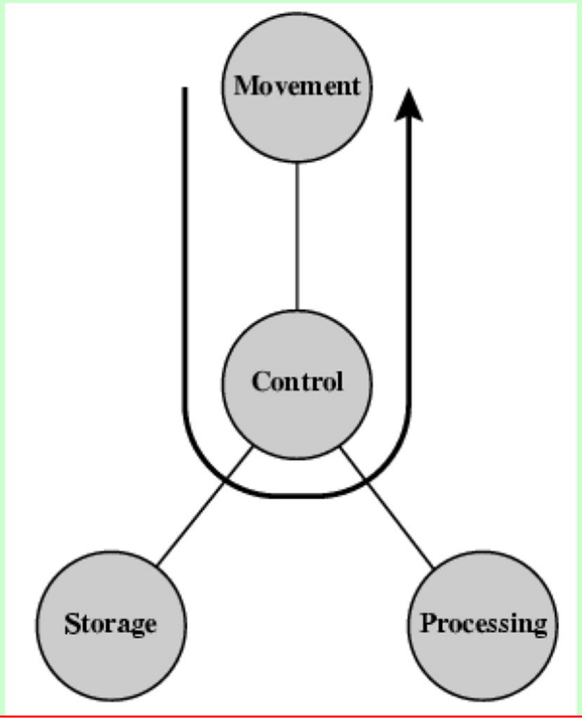
## Possible Computer Operations

- The computer can function as a data movement device (a), simply transferring data from one peripheral or communication line to another.
- It can also function as a data storage device (b), with data transferred from the external environment to computer storage (read) and vice versa (write).
- The final two diagrams show operations involving data processing, on data either in storage (c) or en route between storage and the external environment (d).



# Operations with examples

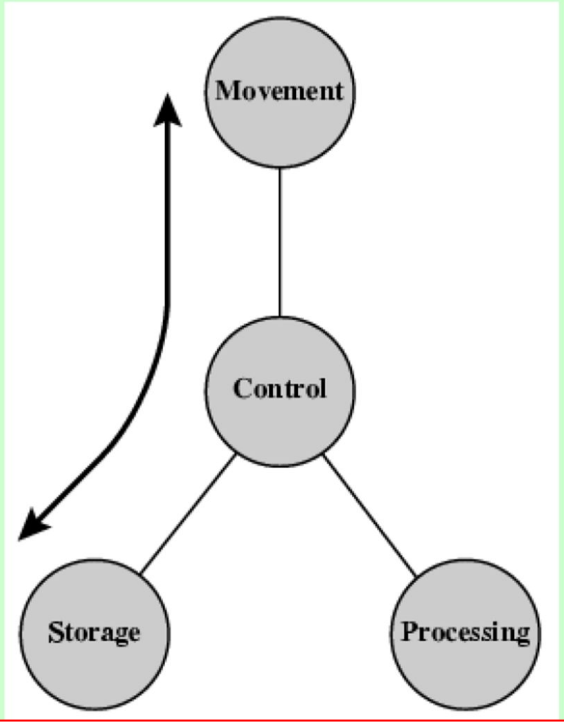
## Operations (a) Data movement



I/O (peripherals directly attached)
Communications/Networking
(communication lines)

Camera attached to a PC, sending the frames to a window on the screen of the same PC.

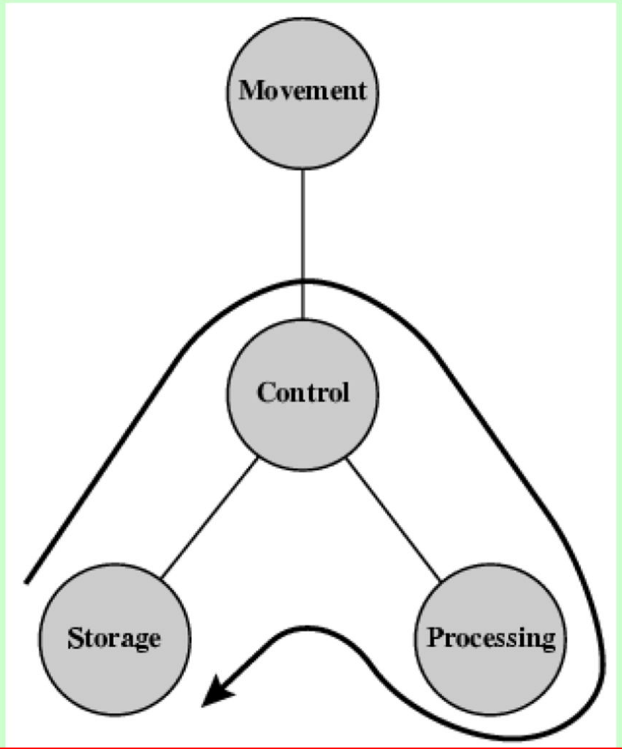
## Operations (b) Storage



Playing an mp3 file stored in memory to earphones attached to the same PC.

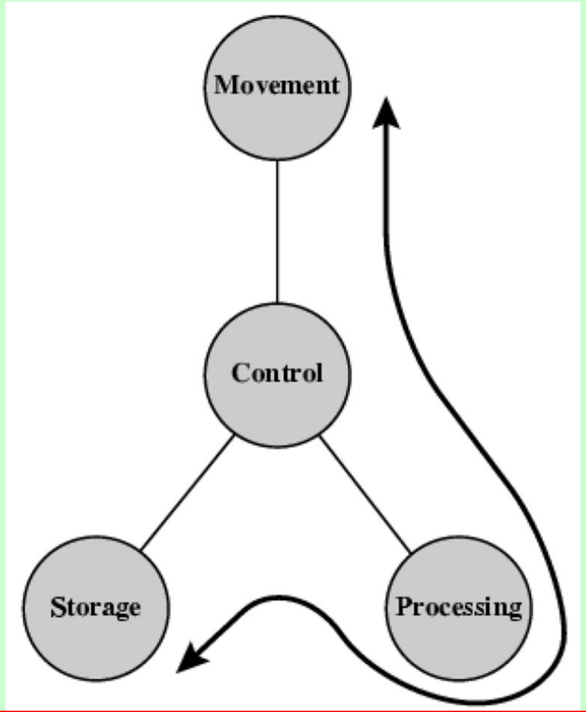
# Operations with examples

**Operation (c) Processing from/to storage**



Any number-crunching application that takes data from memory and stores the result back in memory.

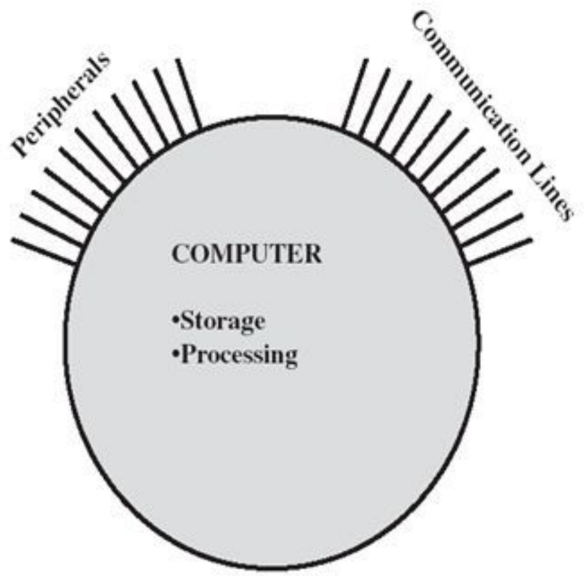
**Operation (d)**
**Processing from storage to I/O**

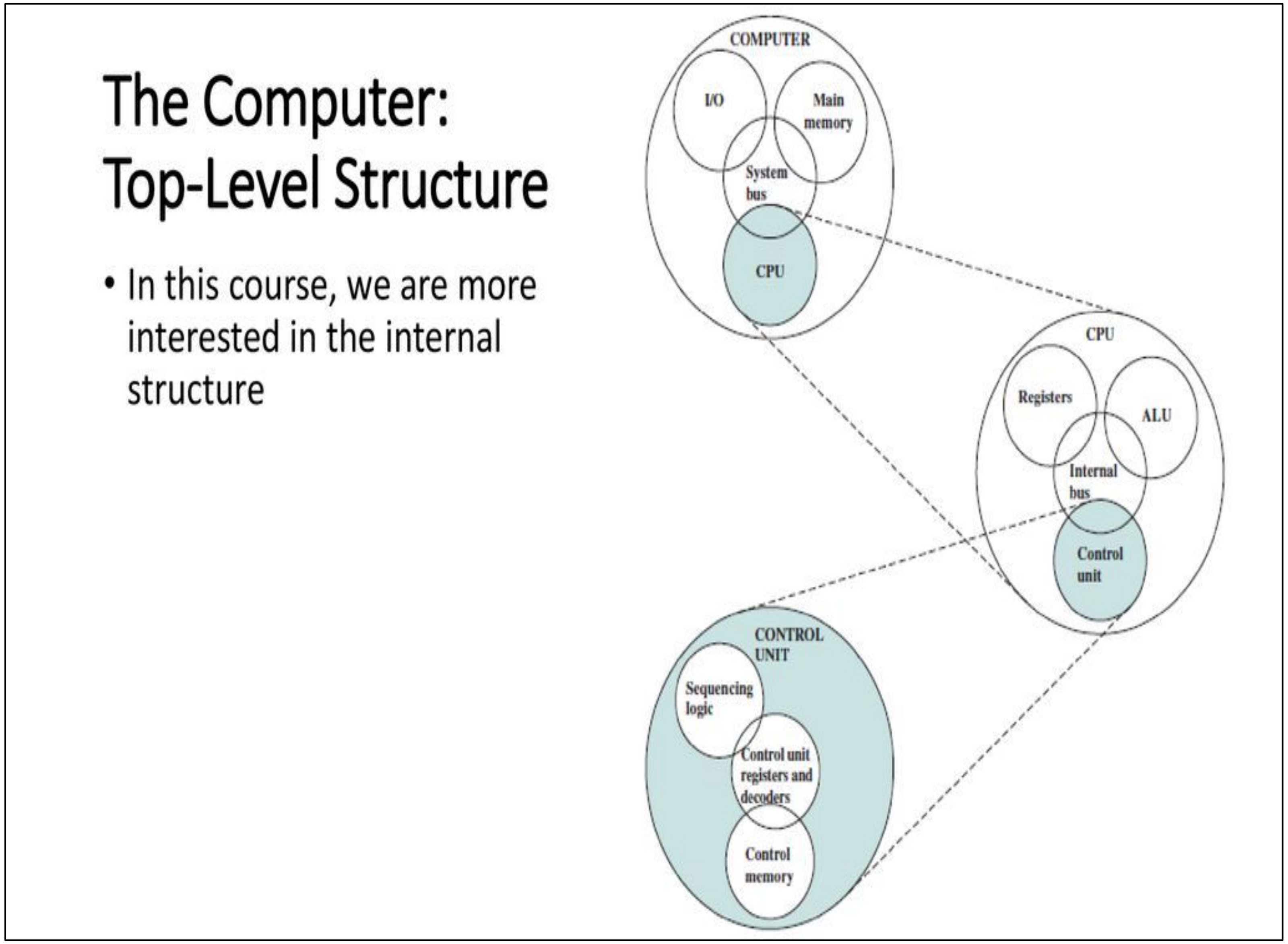


Receiving packets over a network interface, verifying their CRC, then storing them in memory.

## Structure

- The Computer
  - CPU
    - Controls the operation of the computer and performs its data processing functions.
  - Main memory
    - Stores data
  - I/O
    - Moves data between the computer and its external environment
  - System interconnection
    - Provides for communication among CPU, main memory, and I/O





## The Computer: Top-Level Structure

- In this course, we are more interested in the internal structure

However, for our purposes, the most interesting and in some ways the most complex component is the CPU. Its major structural components are as follows:

- Control unit: Controls the operation of the CPU and hence the computer.
- Arithmetic and logic unit (ALU): Performs the computer's data processing functions.
- Registers: Provides storage internal to the CPU.
- CPU interconnection: Some mechanism that provides for communication among the control unit, ALU, and registers.

.

## Computer Evolution and Performance measure

## ENIAC – a vacuum tube computer running on decimal arithmetic

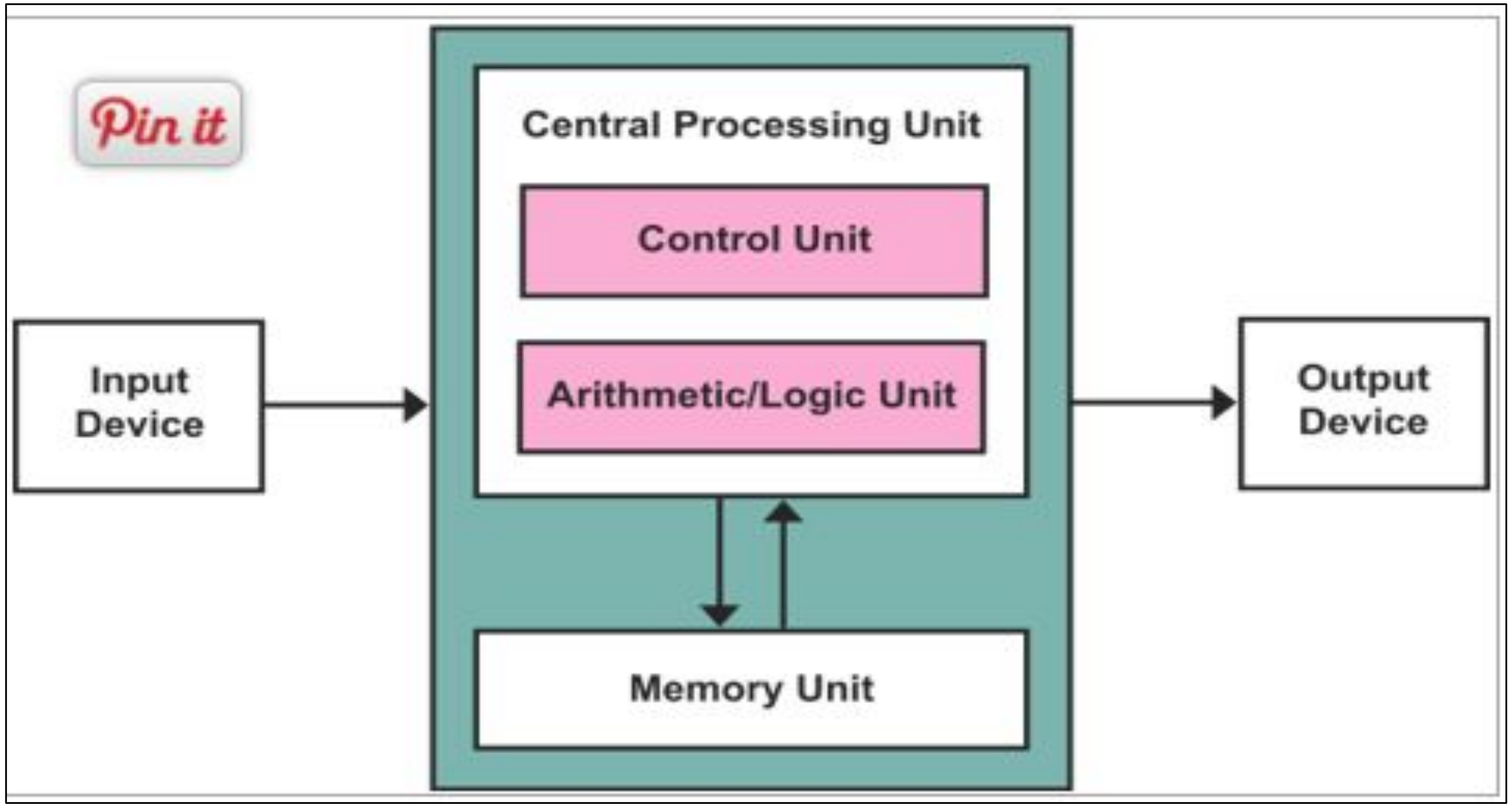
- Electronic Numerical Integrator And Computer
- Eckert and Mauchly
- University of Pennsylvania
- Trajectory tables for weapons
- Started 1943
- Finished 1946
  - Too late for war effort
- Used until 1955

## John von Neumann/AllanTuring

- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Princeton Institute for Advanced Studies
    - IAS
- Completed 1952

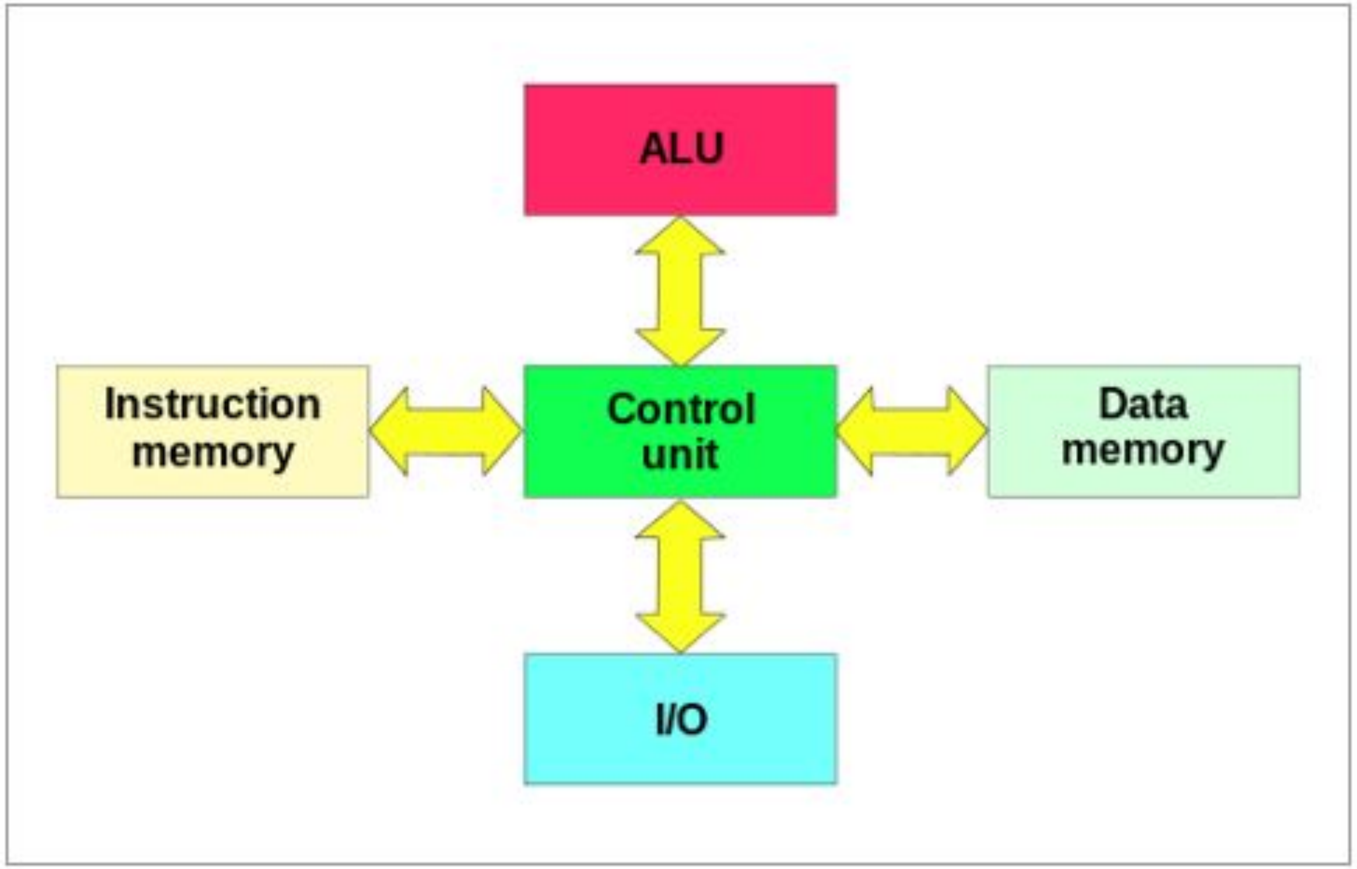
## What is Von Neumann Architecture?

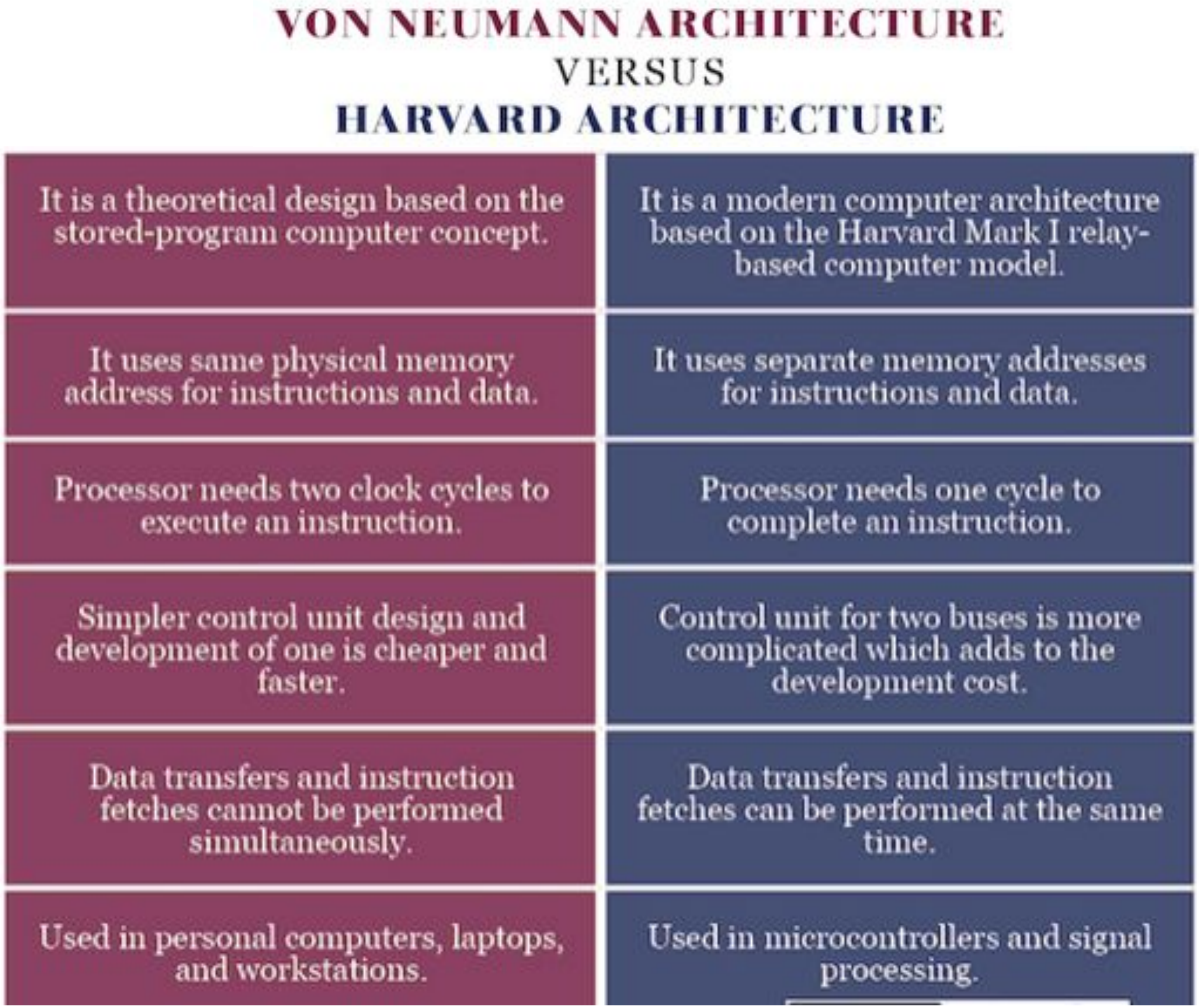
It's a theoretical design based on the concept of stored-program computers where program data and instruction data are stored in the same memory.



## What is Harvard Architecture?

It is a computer architecture with physically separate storage and signal pathways for program data and instructions.





## VON NEUMANN ARCHITECTURE VERSUS HARVARD ARCHITECTURE

| | |
|---|---|
| It is a theoretical design based on the stored-program computer concept. | It is a modern computer architecture based on the Harvard Mark I relay-based computer model. |
| It uses same physical memory address for instructions and data. | It uses separate memory addresses for instructions and data. |
| Processor needs two clock cycles to execute an instruction. | Processor needs one cycle to complete an instruction. |
| Simpler control unit design and development of one is cheaper and faster. | Control unit for two buses is more complicated which adds to the development cost. |
| Data transfers and instruction fetches cannot be performed simultaneously. | Data transfers and instruction fetches can be performed at the same time. |
| Used in personal computers, laptops, and workstations. | Used in microcontrollers and signal processing. |

## Commercial Computers

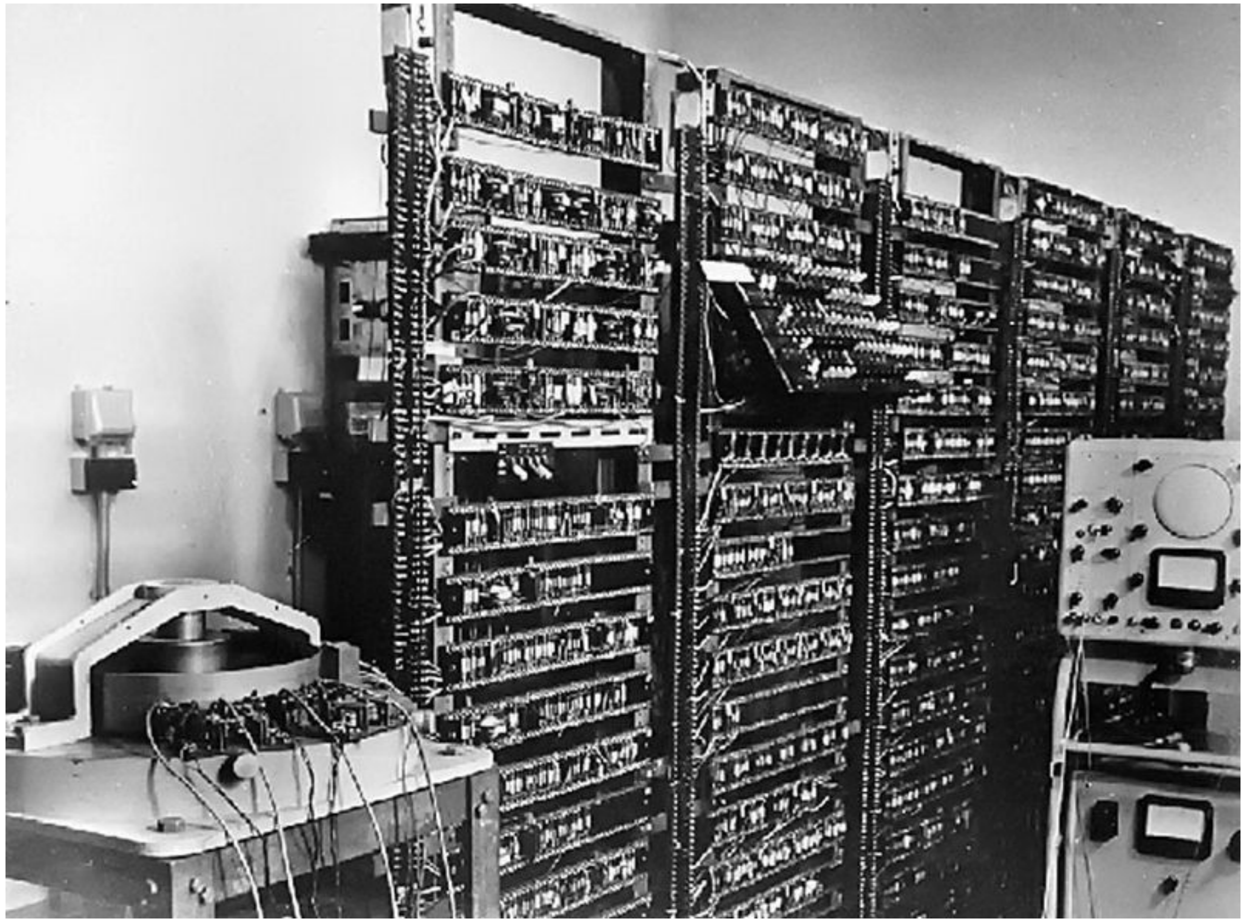
- 1947 - Eckert-Mauchly Computer Corporation
- UNIVAC I (Universal Automatic Computer)
- US Bureau of Census 1950 calculations
- Became part of Sperry-Rand Corporation
- Late 1950s - UNIVAC II
    - Faster
    - More memory

## IBM

- Punched-card processing equipment
- 1953 - the 701
  - IBM’s first stored program computer
  - Scientific calculations
- 1955 - the 702
  - Business applications
- Lead to 700/7000 series

## Punched-card processing equipment

## IBM’s first stored program computer

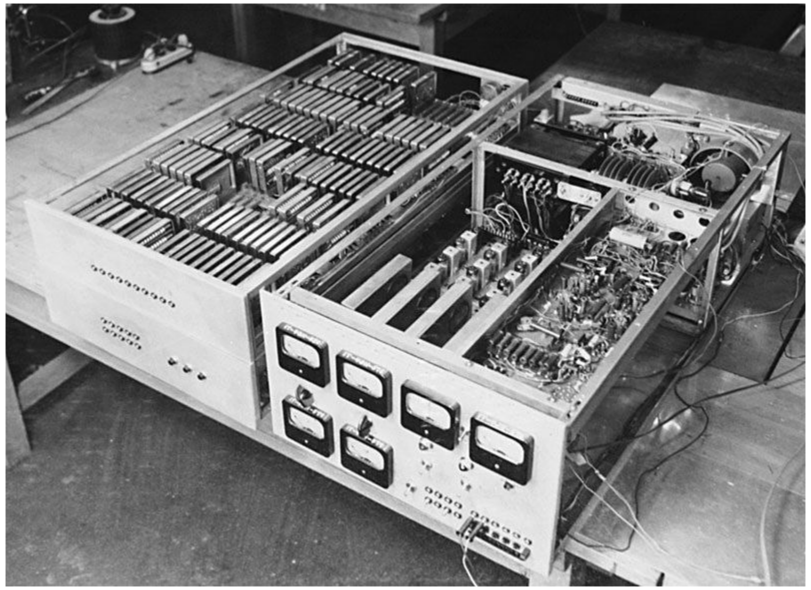


## Transistors

- Replaced vacuum tubes
- Smaller
- Cheaper
- Less heat dissipation
- Solid State device
- Made from Silicon (Sand)
- Invented 1947 at Bell Labs
- William Shockley et al.

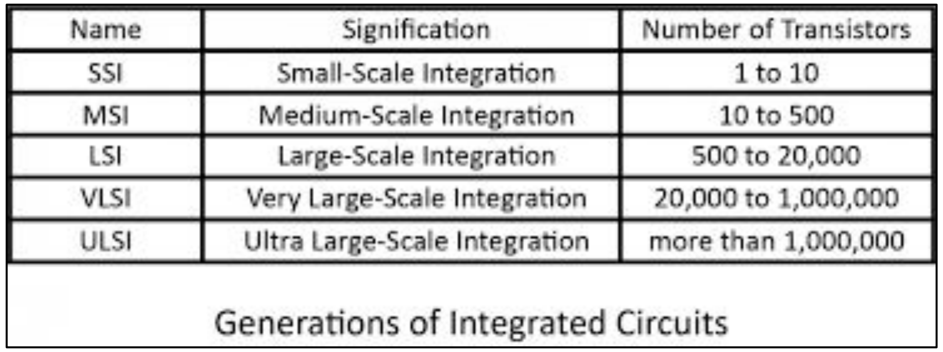
## Transistor Based Computers

- Second generation machines
- NCR & RCA produced small transistor machines
- IBM 7000
- DEC - 1957
  - Produced PDP-1



## Generations of Computer

- Vacuum tube - 1946-1957
- Transistor - 1958-1964
- Small scale integration - 1965 on
  - Up to 100 devices on a chip
- Medium scale integration - to 1971
  - 100-3,000 devices on a chip
- Large scale integration - 1971-1977
  - 3,000 - 100,000 devices on a chip
- Very large-scale integration - 1978 -1991
  - 100,000 - 100,000,000 devices on a chip
- Ultra large-scale integration – 1991 -
  - Over 100,000,000 devices on a chip



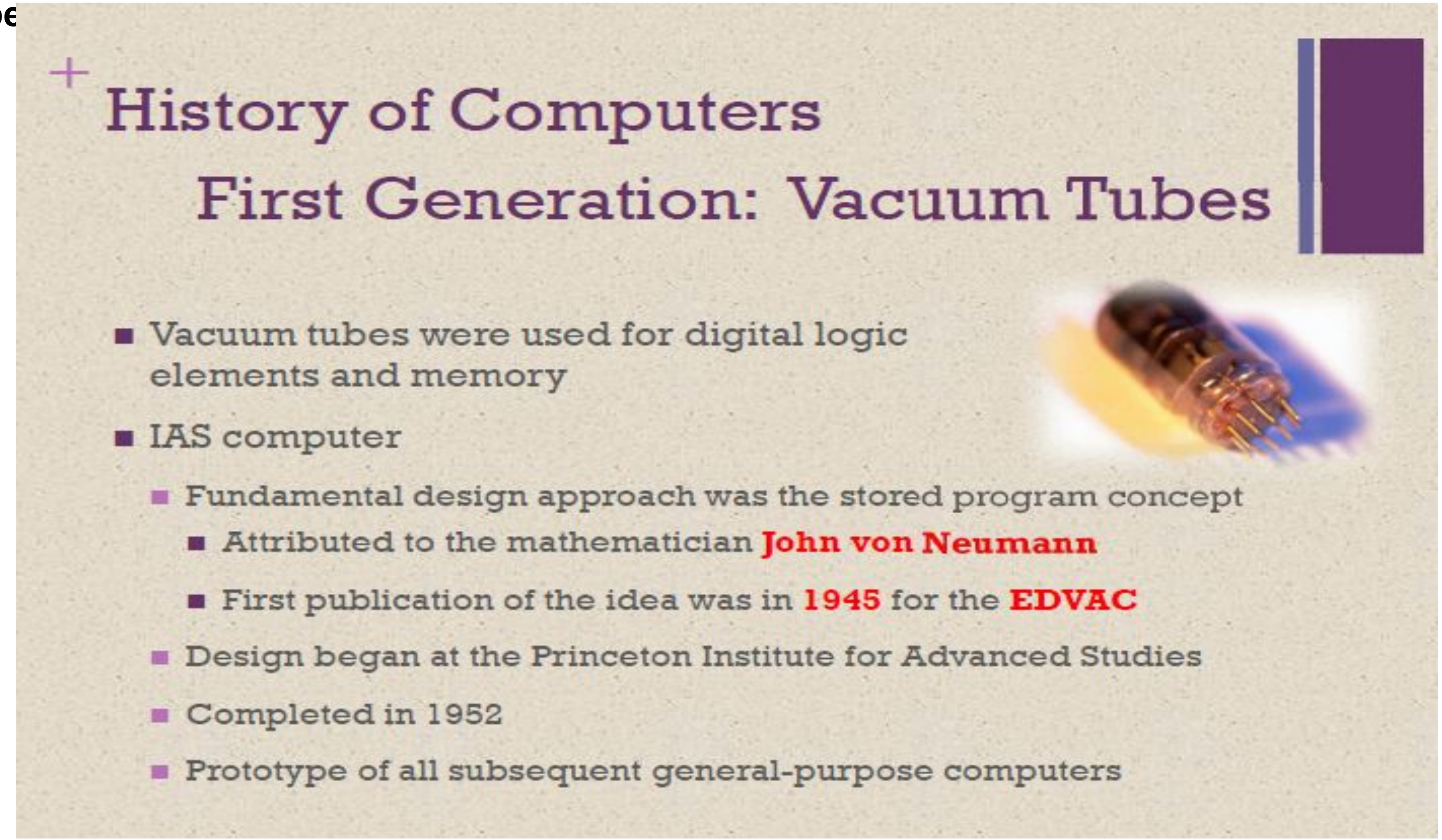
| Name | Signification | Number of Transistors |
|---|---|---|
| SSI | Small-Scale Integration | 1 to 10 |
| MSI | Medium-Scale Integration | 10 to 500 |
| LSI | Large-Scale Integration | 500 to 20,000 |
| VLSI | Very Large-Scale Integration | 20,000 to 1,000,000 |
| ULSI | Ultra Large-Scale Integration | more than 1,000,000 |

Generations of Integrated Circuits

**Brief History of Computers**

- **The First Generation: Vacuum Tube**

IAS (Institute for Advanced Study)

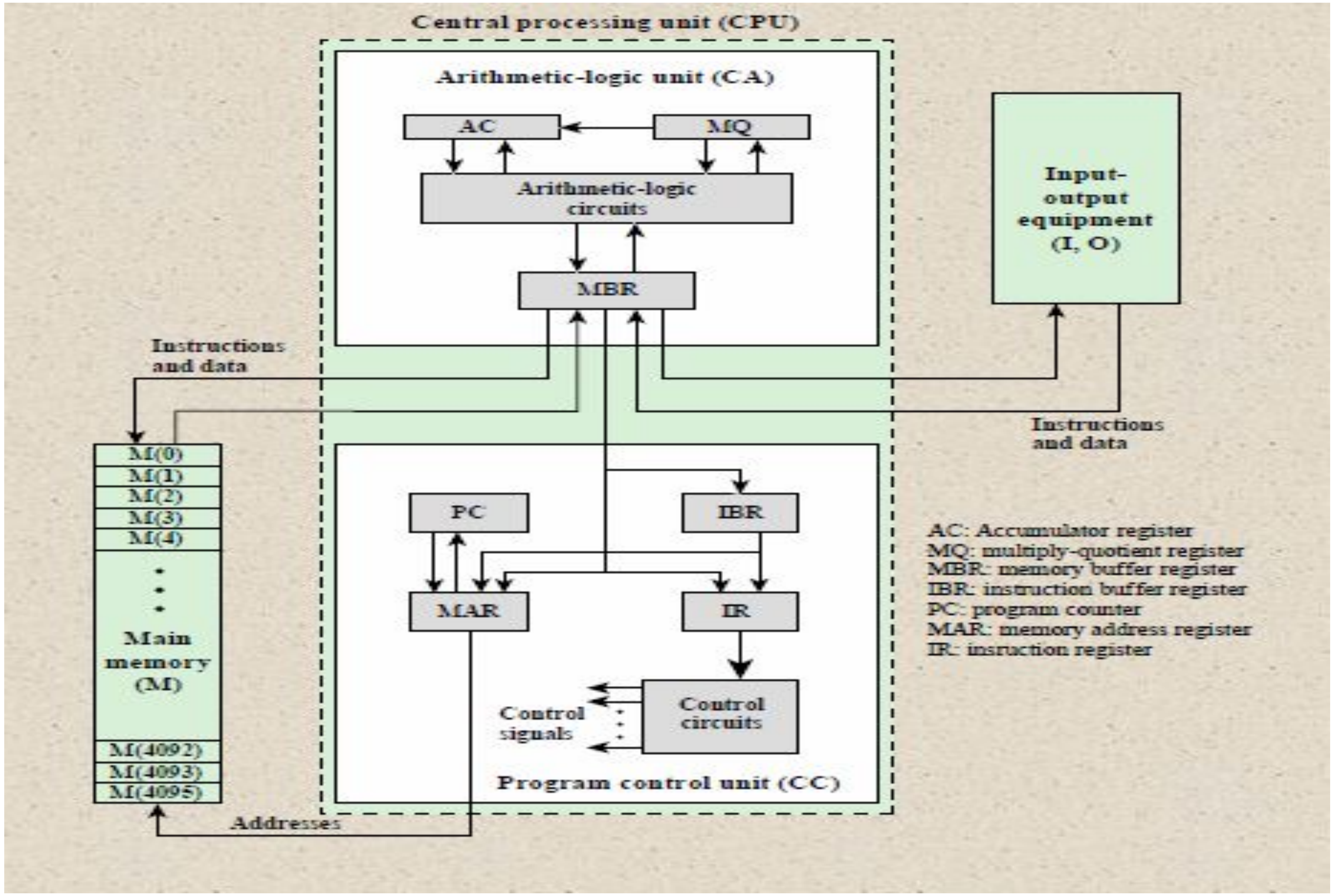


## History of Computers
## First Generation: Vacuum Tubes

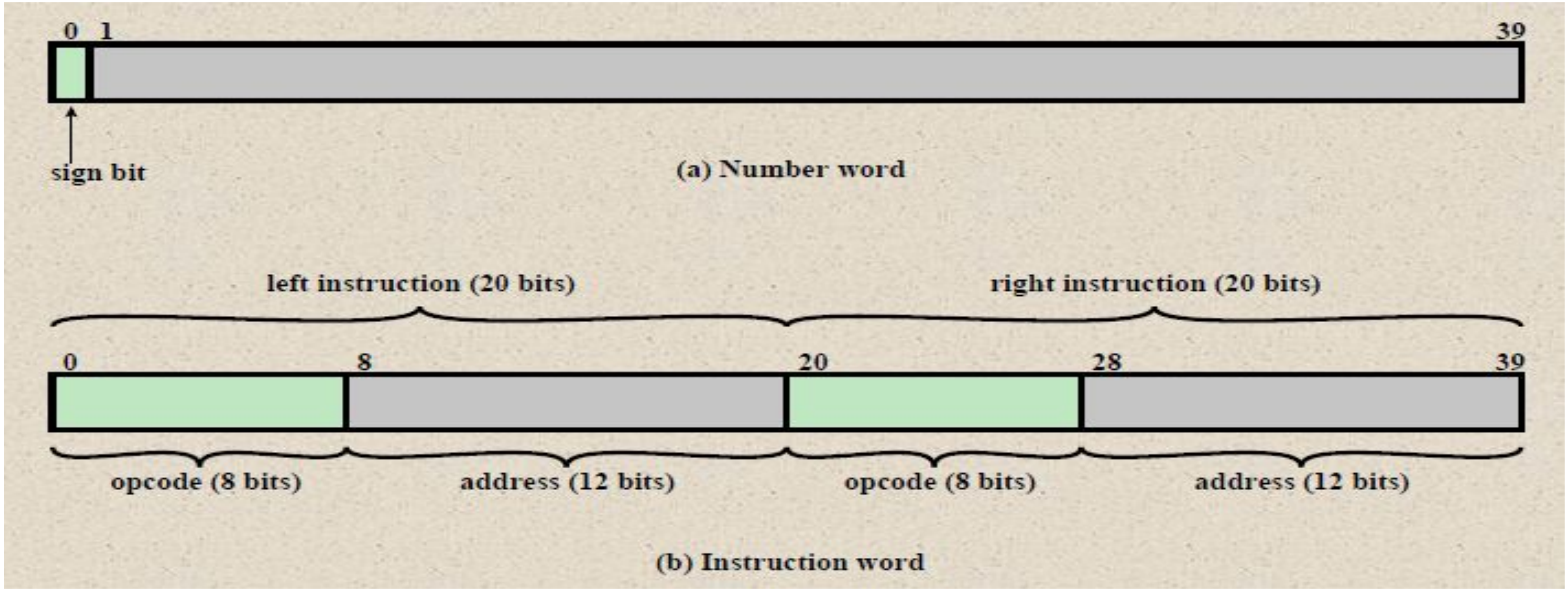
- Vacuum tubes were used for digital logic elements and memory
- IAS computer
  - Fundamental design approach was the stored program concept
    - Attributed to the mathematician **John von Neumann**
    - First publication of the idea was in **1945** for the **EDVAC**
  - Design began at the Princeton Institute for Advanced Studies
  - Completed in 1952
  - Prototype of all subsequent general-purpose computers

Electronic Discrete Variable Automatic Computer

IAS Structure


Central processing unit (CPU)
Arithmetic-logic unit (CA)
AC
MQ
Arithmetic-logic circuits
MBR
Input-output equipment (I, O)
Instructions and data
Instructions and data
M(0)
M(1)
M(2)
M(3)
M(4)
Main memory (M)
M(4092)
M(4093)
M(4095)
PC
IBR
MAR
IR
Control circuits
Control signals
Program control unit (CC)
Addresses
AC: Accumulator register
MQ: multiply-quotient register
MBR: memory buffer register
IBR: instruction buffer register
PC: program counter
MAR: memory address register
IR: insruction register

IAS Memory format



## History of Computers Second Generation: Transistors

- Smaller
- Cheaper
- Dissipates less heat than a vacuum tube
- Is a *solid state device* made from silicon
- Was invented at Bell Labs in 1947
- It was not until the late 1950's that fully transistorized computers were commercially available

## Table 1.2 Computer Generations

| Generation | Approximate Dates | Technology | Typical Speed (operations per second) |
|---|---|---|---|
| 1 | 1946–1957 | Vacuum tube | 40,000 |
| 2 | 1957–1964 | Transistor | 200,000 |
| 3 | 1965–1971 | Small and medium scale integration | 1,000,000 |
| 4 | 1972–1977 | Large scale integration | 10,000,000 |
| 5 | 1978–1991 | Very large scale integration | 100,000,000 |
| 6 | 1991- | Ultra large scale integration | >1,000,000,000 |

## Second Generation Computers

- Introduced:
  - More complex arithmetic and logic units and control units
  - The use of high-level programming languages
  - Provision of *system software* which provided the ability to:
    - Load programs
    - Move data to peripherals
    - Libraries perform common computations

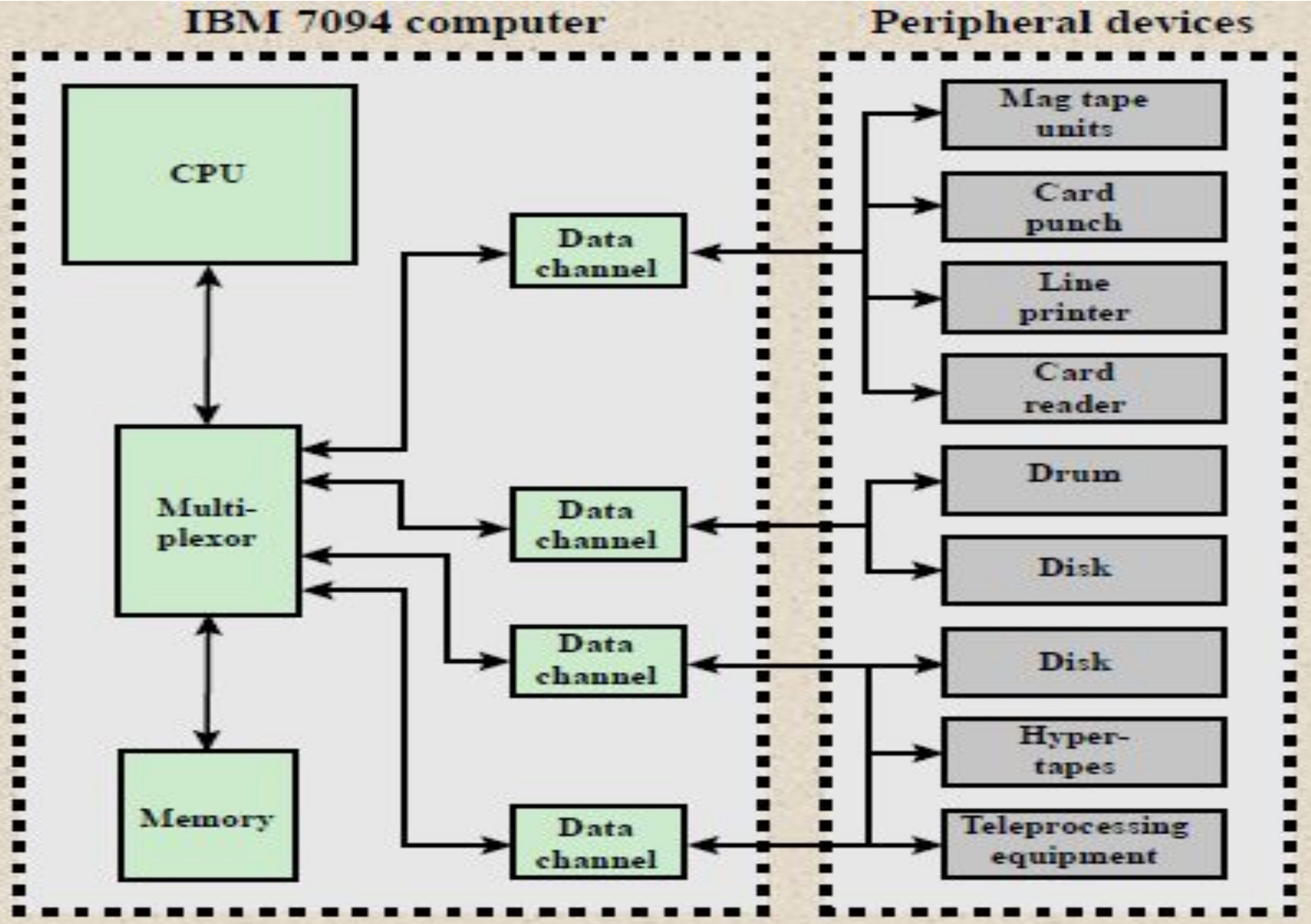


Figure 1.9 An IBM 7094 Configuration

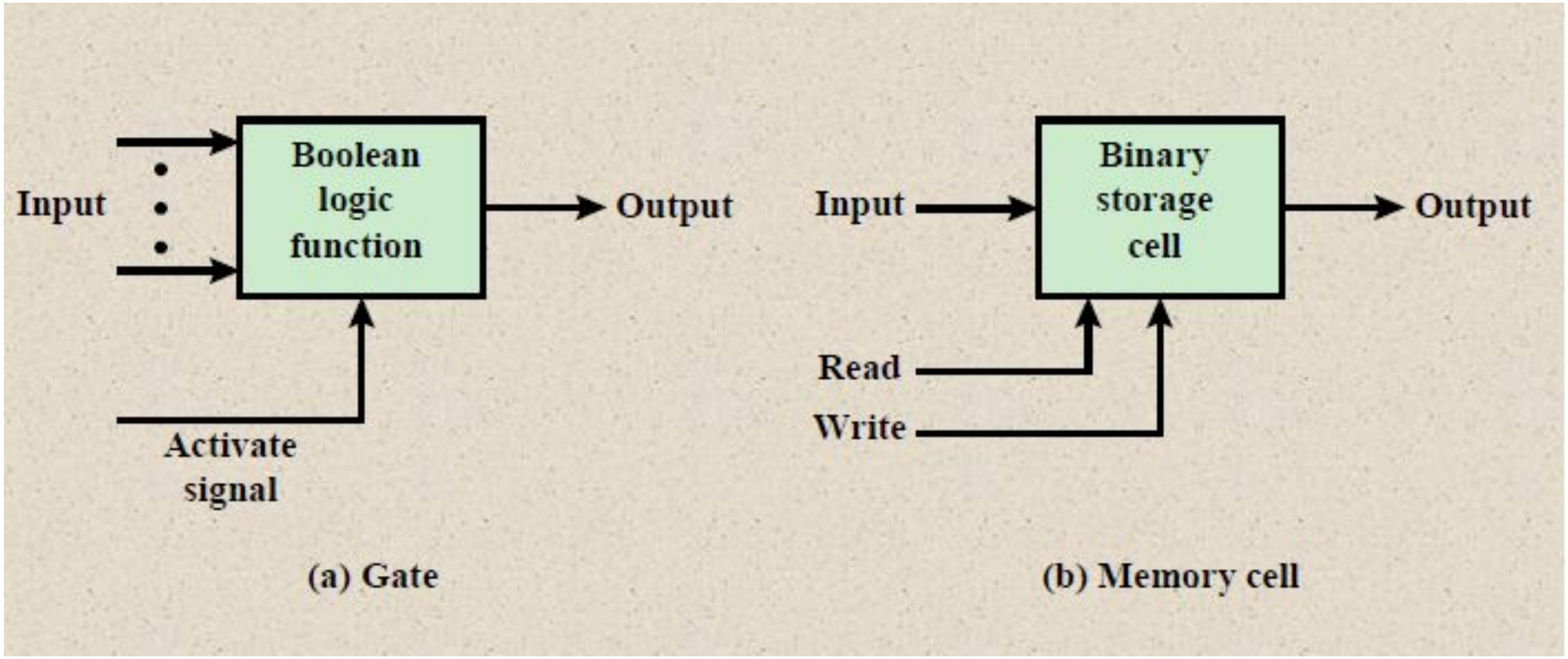
**Data Channel is a Specialized I/O Computer**

## History of Computers

## Third Generation: Integrated Circuits

- 1958 – the invention of the integrated circuit
- *Discrete component*
  - Single, self-contained transistor
  - Manufactured separately, packaged in their own containers, and soldered or wired together onto masonite-like circuit boards
  - Manufacturing process was expensive and cumbersome
- The two most important members of the third generation were the **IBM System/360** and the **DEC PDP-8**

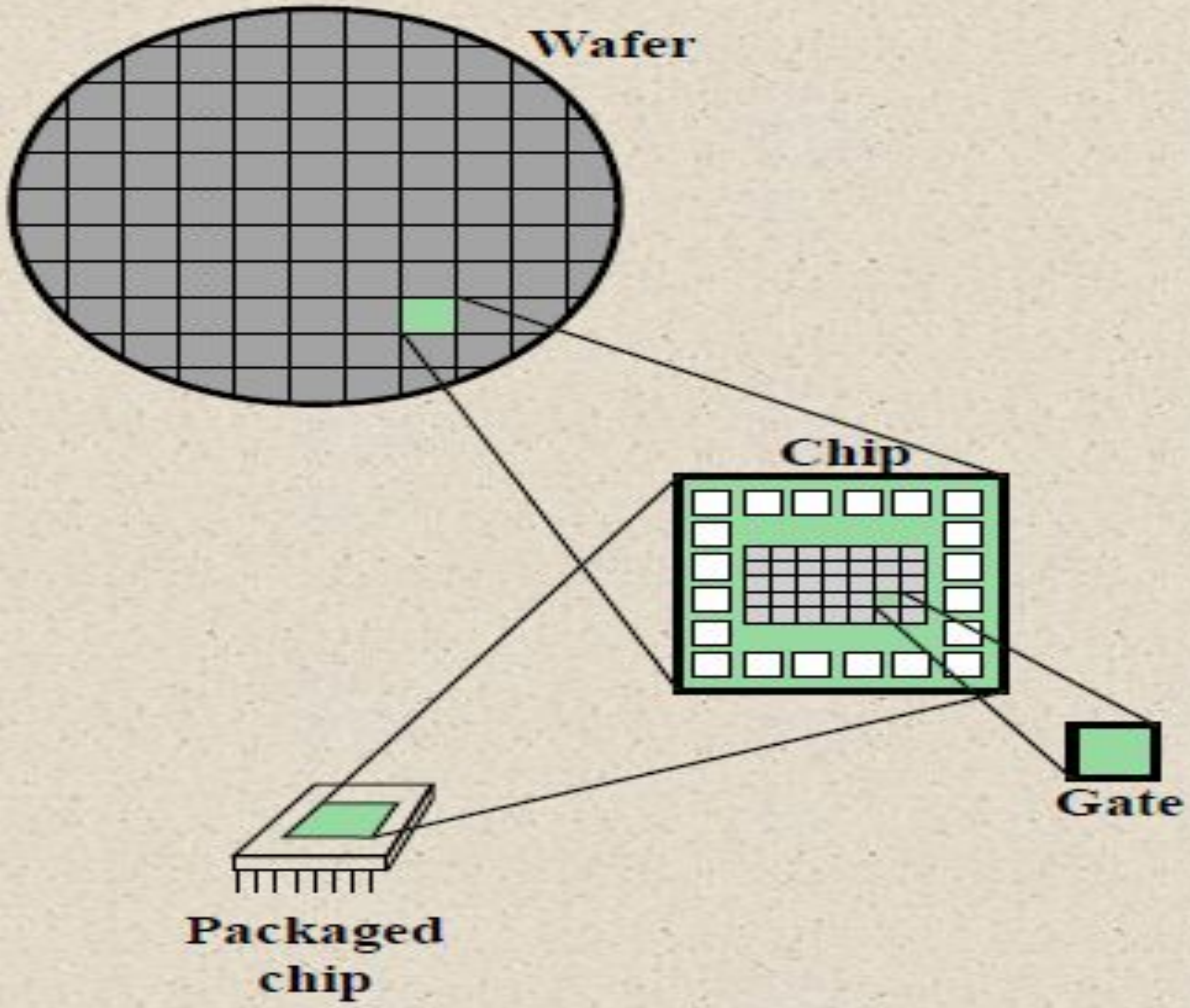
## Fundamental Computer Elements



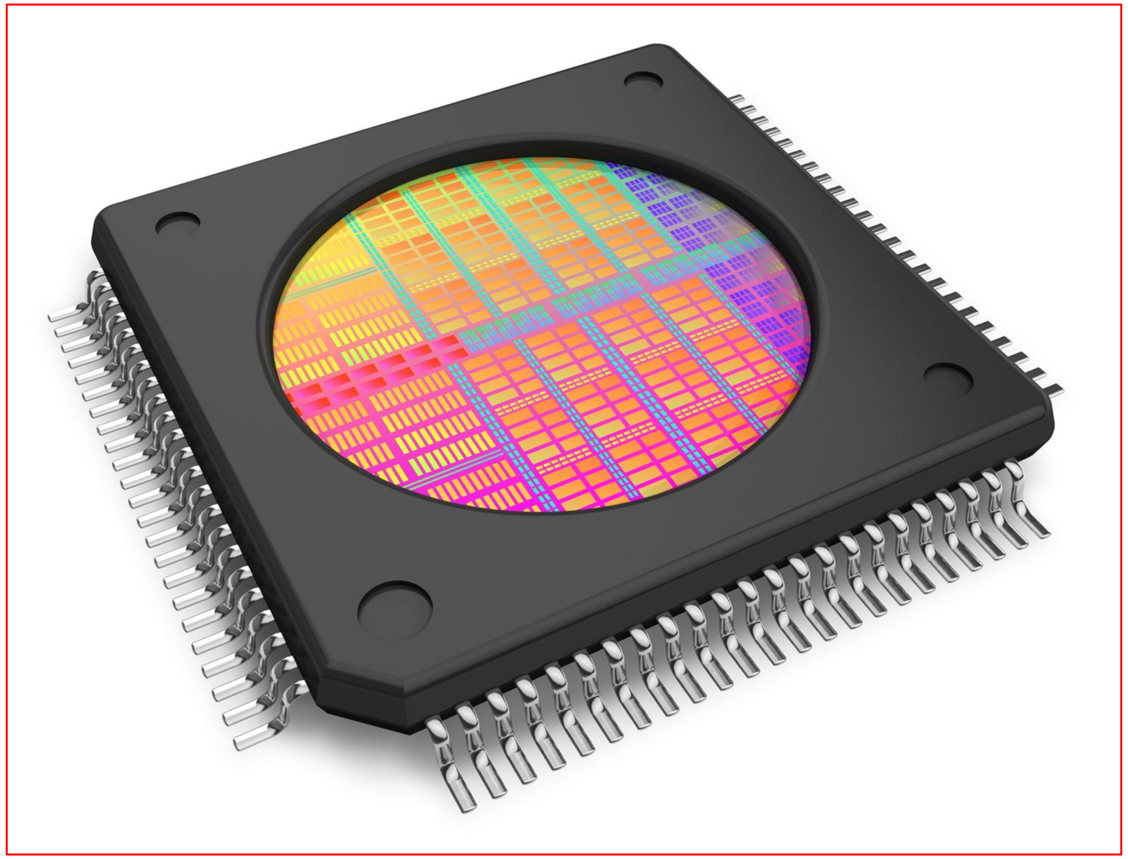
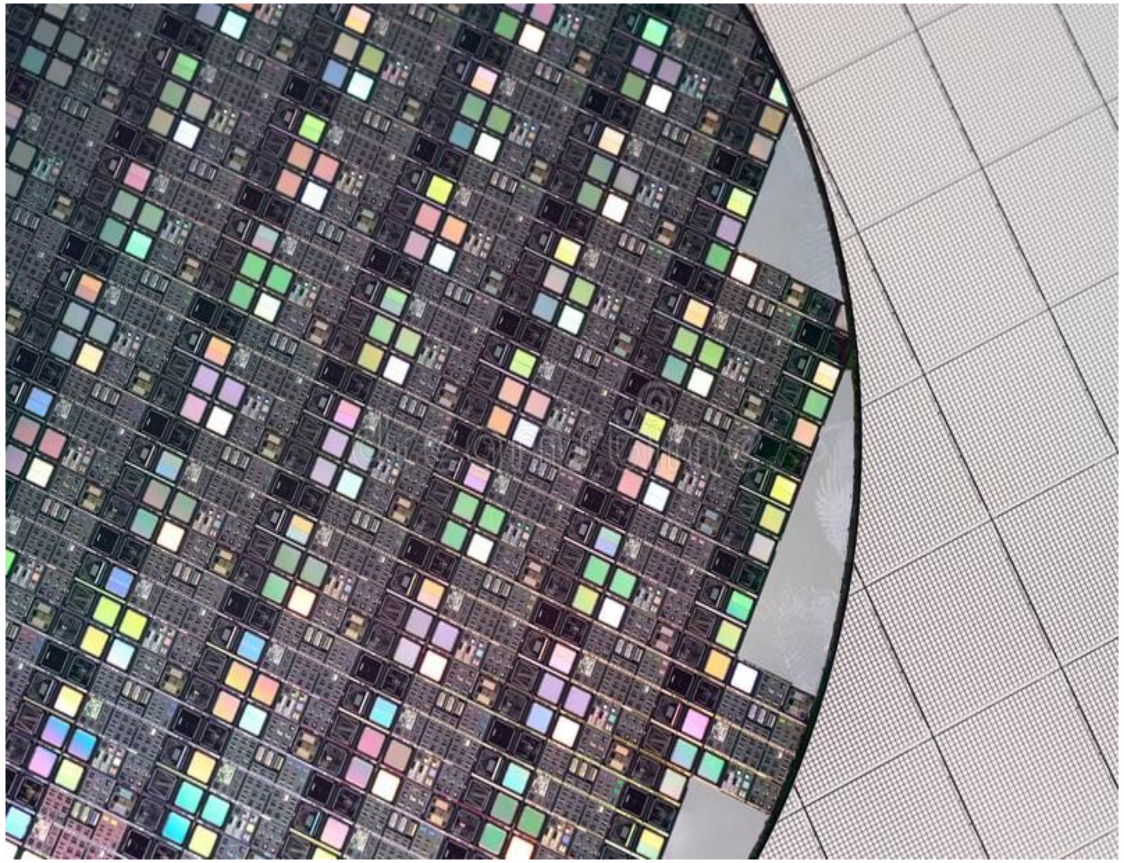
## Integrated Circuits

- Data storage – provided by memory cells
- Data processing – provided by gates
- Data movement – the paths among components are used to move data from memory to memory and from memory through gates to memory
- Control – the paths among components can carry control signals

- A computer consists of gates, memory cells, and interconnections among these elements
- The gates and memory cells are constructed of simple digital electronic components
- Exploits the fact that such components as transistors, resistors, and conductors can be fabricated from a semiconductor such as silicon
- Many transistors can be produced at the same time on a single wafer of silicon
- Transistors can be connected with a processor metallization to form circuits



Relationship Among Wafer, Chip, and Gate

## IBM System/360

- Announced in 1964
- Product line was incompatible with older IBM machines
- Was the success of the decade and cemented IBM as the overwhelmingly dominant computer vendor
- The architecture remains to this day the architecture of IBM's mainframe computers
- Was the industry's first planned family of computers
  - Models were compatible in the sense that a program written for one model should be capable of being executed by another model in the series

## Family Characteristics

- Similar or identical instruction set
- Similar or identical operating system
- Increasing speed
- Increasing number of I/O ports
- Increasing memory size
- Increasing cost

## Later Generations

**LSI**
Large Scale Integration

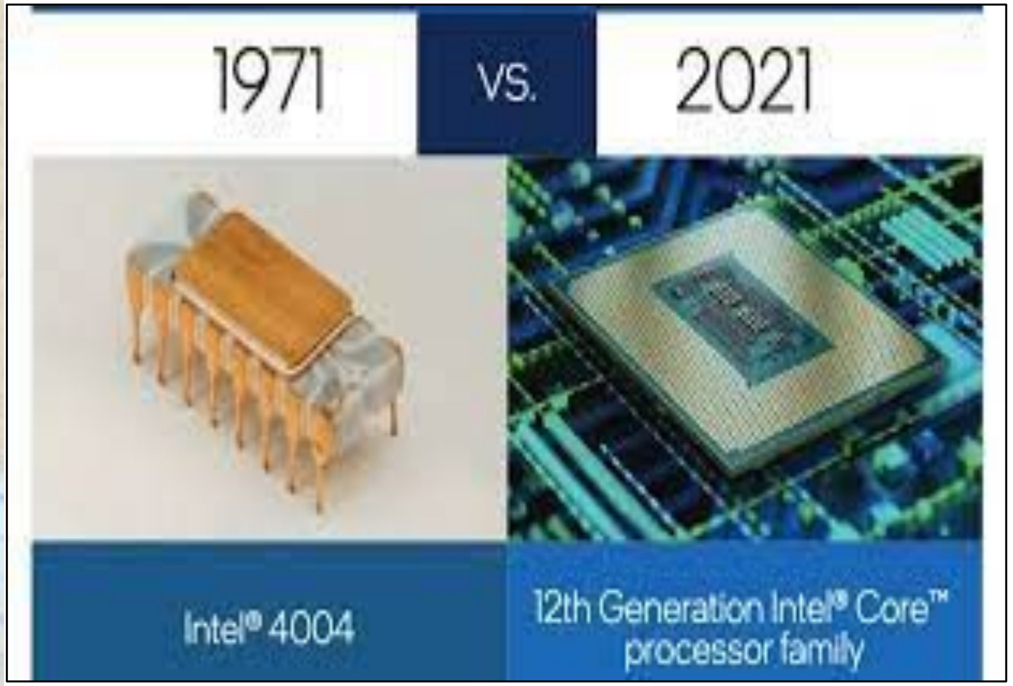
**VLSI**
Very Large Scale Integration

Semiconductor Memory
Microprocessors

**ULSI**
Ultra Large Scale Integration

## Microprocessors

- The density of elements on processor chips continued to rise
  - More and more elements were placed on each chip so that fewer and fewer chips were needed to construct a single computer processor
- 1971 Intel developed 4004
  - First chip to contain all of the components of a CPU on a single chip
  - Birth of microprocessor
- 1972 Intel developed 8008
  - First 8-bit microprocessor
- 1974 Intel developed 8080
  - First general purpose microprocessor
  - Faster, has a richer instruction set, has a large addressing capability



## Evolution of Intel Microprocessors

| | 4004 | 8008 | 8080 | 8086 | 8088 |
|---|---|---|---|---|---|
| Introduced | 1971 | 1972 | 1974 | 1978 | 1979 |
| Clock speeds | 108 kHz | 108 kHz | 2 MHz | 5 MHz, 8 MHz, 10 MHz | 5 MHz, 8 MHz |
| Bus width | 4 bits | 8 bits | 8 bits | 16 bits | 8 bits |
| Number of transistors | 2,300 | 3,500 | 6,000 | 29,000 | 29,000 |
| Feature size (µm) | 10 | 8 | 6 | 3 | 6 |
| Addressable memory | 640 Bytes | 16 KB | 64 KB | 1 MB | 1 MB |

(a) 1970s Processors

# Evolution of Intel Microprocessors

| | 80286 | 386TM DX | 386TM SX | 486TM DX CPU |
|---|---|---|---|---|
| Introduced | 1982 | 1985 | 1988 | 1989 |
| Clock speeds | 6 MHz - 12.5 MHz | 16 MHz - 33 MHz | 16 MHz - 33 MHz | 25 MHz - 50 MHz |
| Bus width | 16 bits | 32 bits | 16 bits | 32 bits |
| Number of transistors | 134,000 | 275,000 | 275,000 | 1.2 million |
| Feature size (µm) | 1.5 | 1 | 1 | 0.8 - 1 |
| Addressable memory | 16 MB | 4 GB | 16 MB | 4 GB |
| Virtual memory | 1 GB | 64 TB | 64 TB | 64 TB |
| Cache | — | — | — | 8 kB |

(b) 1980s Processors

## Evolution of Intel Microprocessors

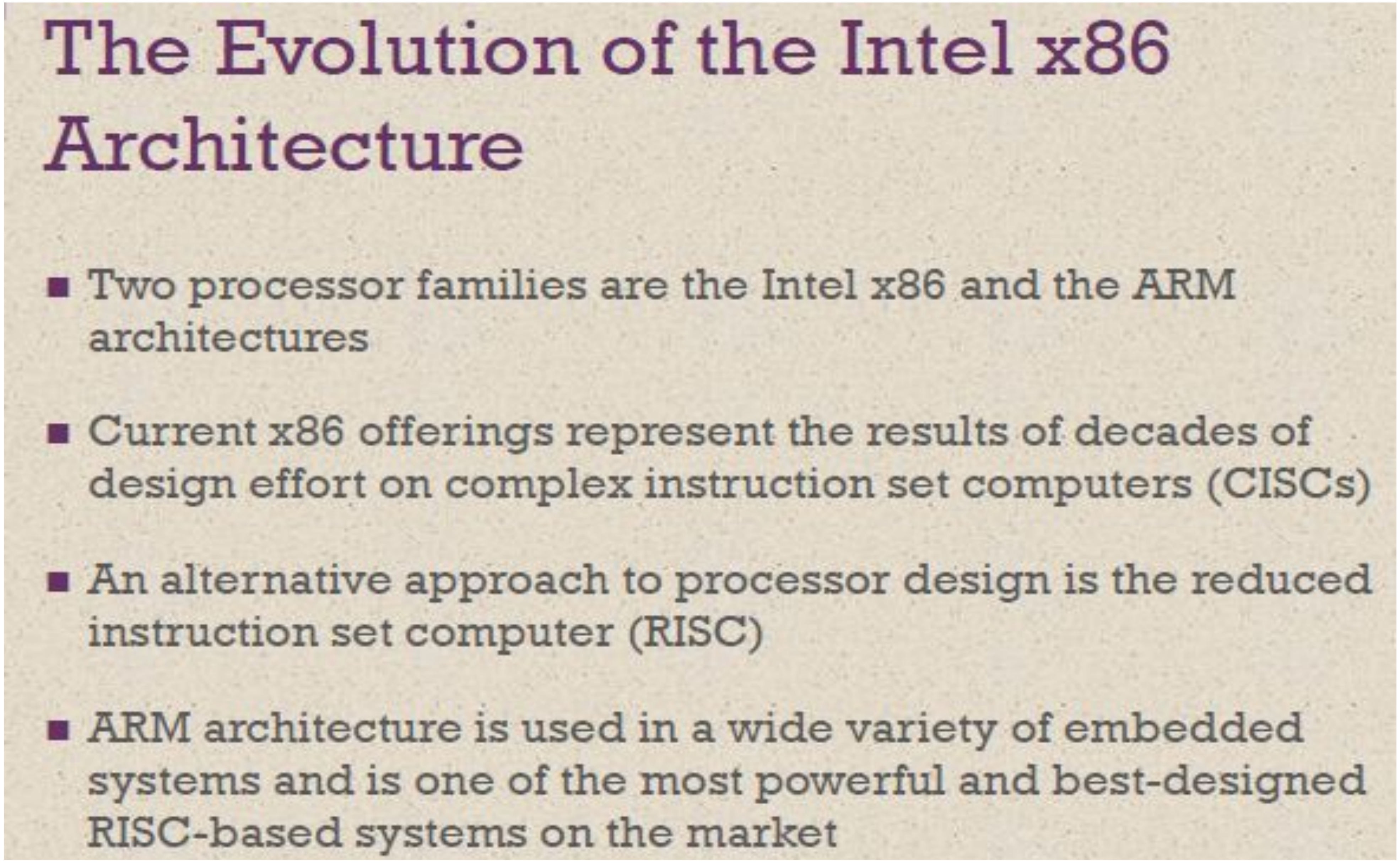
| | 486TM SX | Pentium | Pentium Pro | Pentium II |
|---|---|---|---|---|
| Introduced | 1991 | 1993 | 1995 | 1997 |
| Clock speeds | 16 MHz - 33 MHz | 60 MHz - 166 MHz, | 150 MHz - 200 MHz | 200 MHz - 300 MHz |
| Bus width | 32 bits | 32 bits | 64 bits | 64 bits |
| Number of transistors | 1.185 million | 3.1 million | 5.5 million | 7.5 million |
| Feature size (µm) | 1 | 0.8 | 0.6 | 0.35 |
| Addressable memory | 4 GB | 4 GB | 64 GB | 64 GB |
| Virtual memory | 64 TB | 64 TB | 64 TB | 64 TB |
| Cache | 8 kB | 8 kB | 512 kB L1 and 1 MB L2 | 512 kB L2 |

(c) 1990s Processors

# Evolution of Intel Microprocessors

| | Pentium III | Pentium 4 | Core 2 Duo | Core i7 EE 4960X |
|---|---|---|---|---|
| Introduced | 1999 | 2000 | 2006 | 2013 |
| Clock speeds | 450 - 660 MHz | 1.3 - 1.8 GHz | 1.06 - 1.2 GHz | 4 GHz |
| Bus wid th | 64 bits | 64 bits | 64 bits | 64 bits |
| Number of transistors | 9.5 million | 42 million | 167 million | 1.86 billion |
| Feature size (nm) | 250 | 180 | 65 | 22 |
| Addressable memory | 64 GB | 64 GB | 64 GB | 64 GB |
| Virtual memory | 64 TB | 64 TB | 64 TB | 64 TB |
| Cache | 512 kB L2 | 256 kB L2 | 2 MB L2 | 1.5 MB L2/15 MB L3 |
| Number of cores | 1 | 1 | 2 | 6 |

(d) Recent Processors



## The Evolution of the Intel x86 Architecture

- Two processor families are the Intel x86 and the ARM architectures
- Current x86 offerings represent the results of decades of design effort on complex instruction set computers (CISCs)
- An alternative approach to processor design is the reduced instruction set computer (RISC)
- ARM architecture is used in a wide variety of embedded systems and is one of the most powerful and best-designed RISC-based systems on the market

## Highlights of the Evolution of the Intel Product Line:

### 8080

- World's first general-purpose microprocessor
- 8-bit machine, 8-bit data path to memory
- Was used in the first personal computer (Altair)

### 8086

- A more powerful 16-bit machine
- Has an instruction cache, or queue, that prefetches a few instructions before they are executed
- The first appearance of the x86 architecture
- The 8088 was a variant of this processor and used in IBM's first personal computer (securing the success of Intel

### 80286

- Extension of the 8086 enabling addressing a 16-MB memory instead of just 1MB

### 80386

- Intel's first 32-bit machine
- First Intel processor to support multitasking

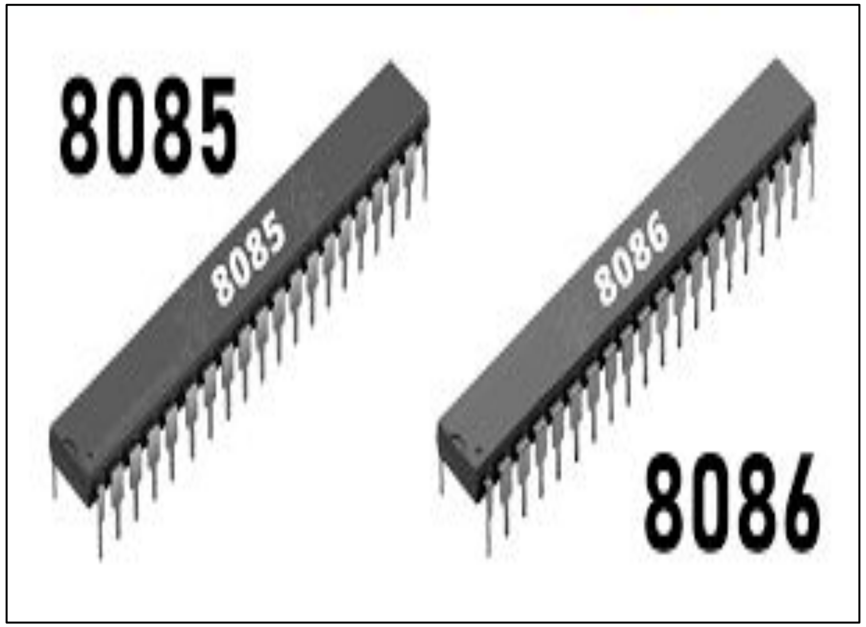
### 80486

- Introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining
- Also offered a built-in math coprocessor

## Amdahl's law

- **Amdahl's law**, named after computer architect Gene Amdahl, is used to find the maximum expected improvement to an overall system when only part of the system is improved.
- Amdhal's law can be interpreted more technically, but in simplest terms it means that it is the algorithm that decides the speedup not the number of processors.

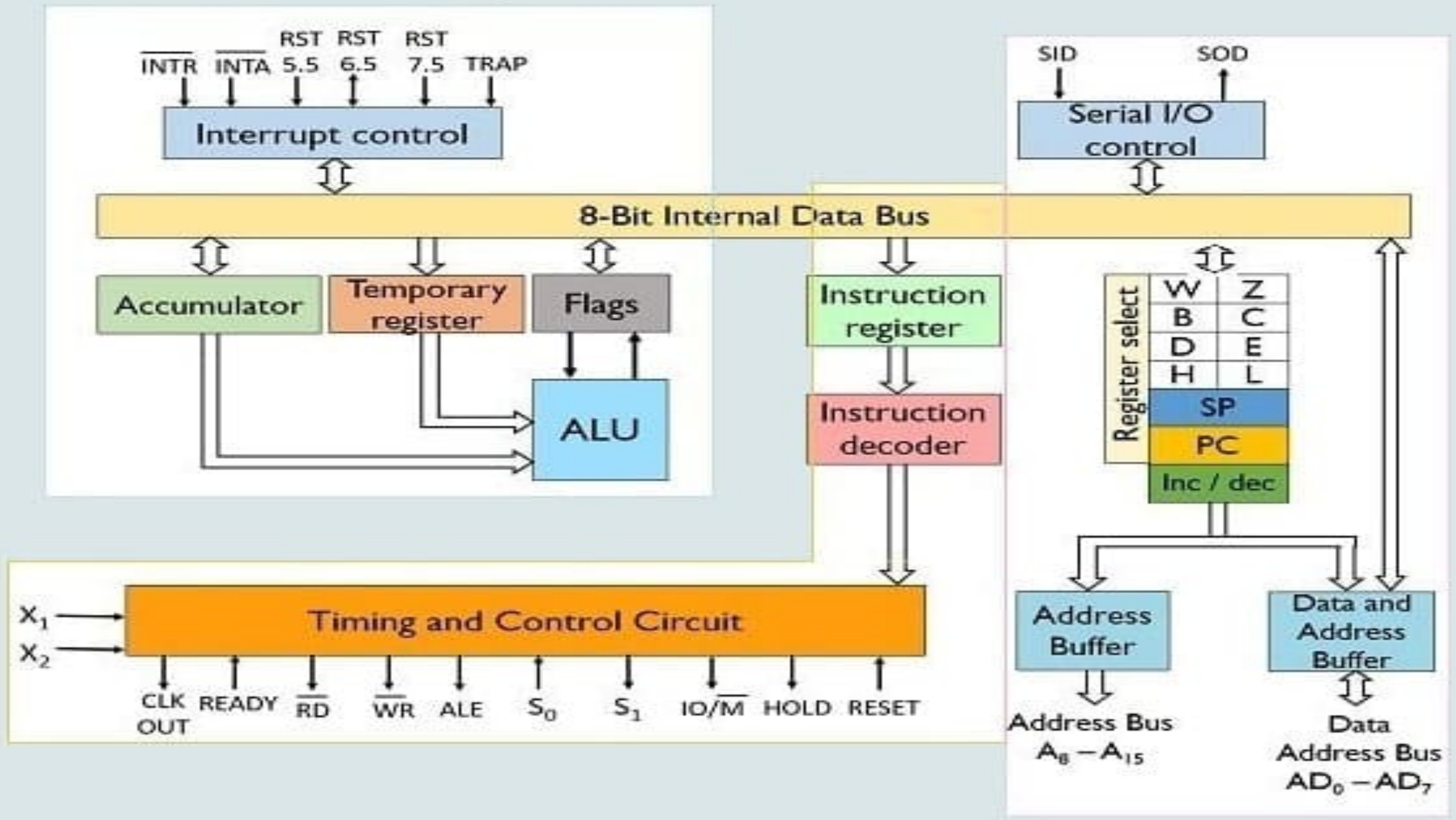
## Microprocessor



A microprocessor is a component that performs the instructions and tasks involved in computer processing. In a computer system, the microprocessor is the central unit that executes and manages the logical instructions passed to it.

A microprocessor may also be called a processor or central processing unit, but it is more advanced in terms of architectural design and is built over a silicon microchip.
**A laptop** is the best example where a microprocessor is used. Some other home items that contain microprocessors are: microwaves, toasters, televisions, VCRs, DVD players, ovens, stoves, clothes washers, stereo systems, home computers, alarm clocks, hand-held game devices, thermostats, video game systems, bread machines, dishwashers and home lighting systems

# Block Diagram of 8085 Microprocessor



## 8086

A Microprocessor is an **Integrated Circuit with all the functions of a CPU**. However, it cannot be used stand-alone since unlike a microcontroller it has no memory or peripherals.

8086 **does not have a RAM or ROM inside it**. However, it has internal registers for storing intermediate results and interfaces with memory located outside it through the System Bus.

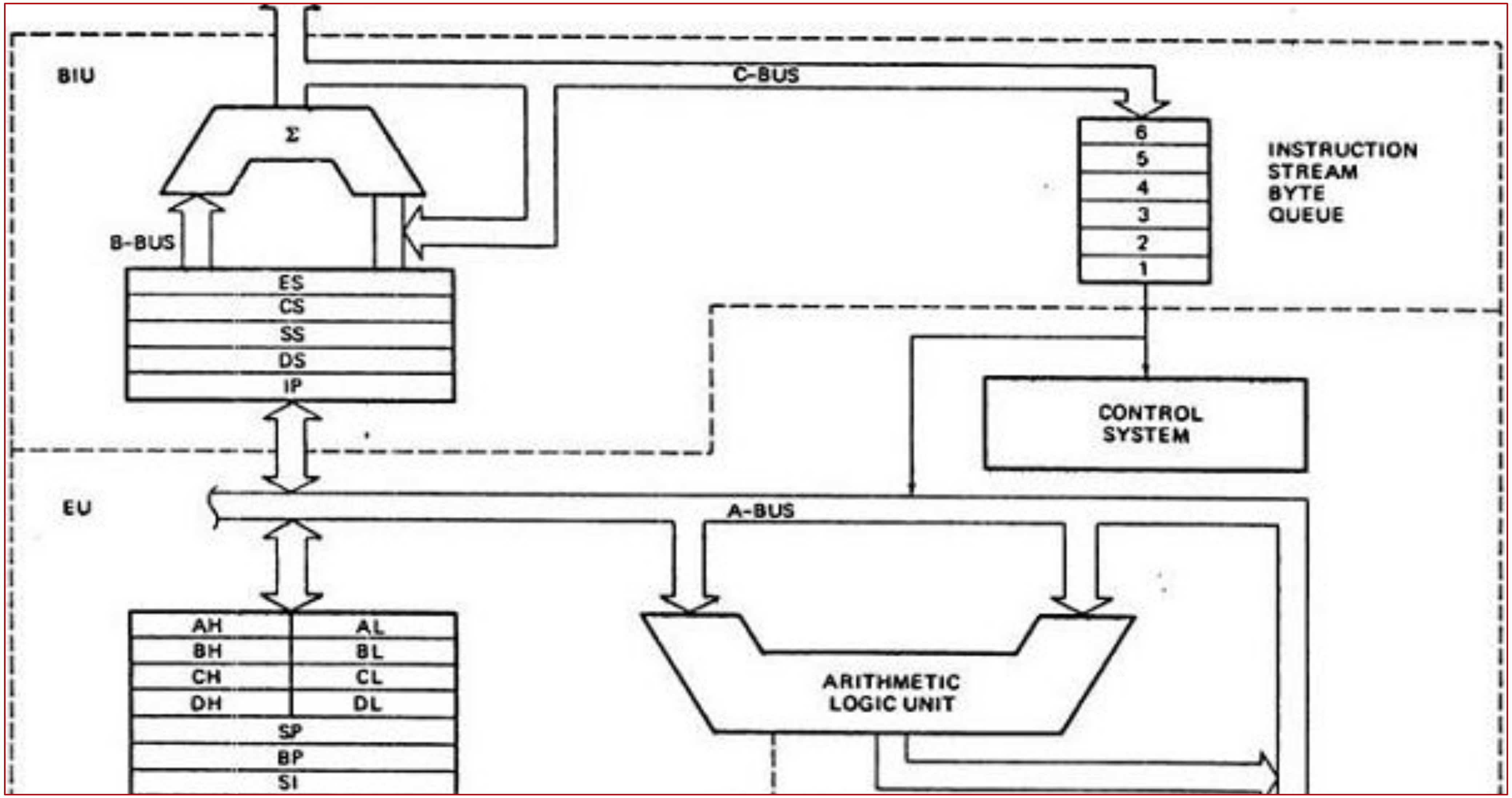
In the case of 8086, it is a **16-bit** Integer processor in a **40-pin**, Dual Inline Packaged IC.

8086 provides the programmer with 14 internal registers, each of 16 bits or 2 bytes wide. The main advantage of the 8086 microprocessor is that it **supports Pipelining.**

Data bus - **carries the data between the processor and other components**

The **opcode** is the instruction that is executed by the CPU and the **operand** is the data or memory location used to execute that instruction.

# 8086

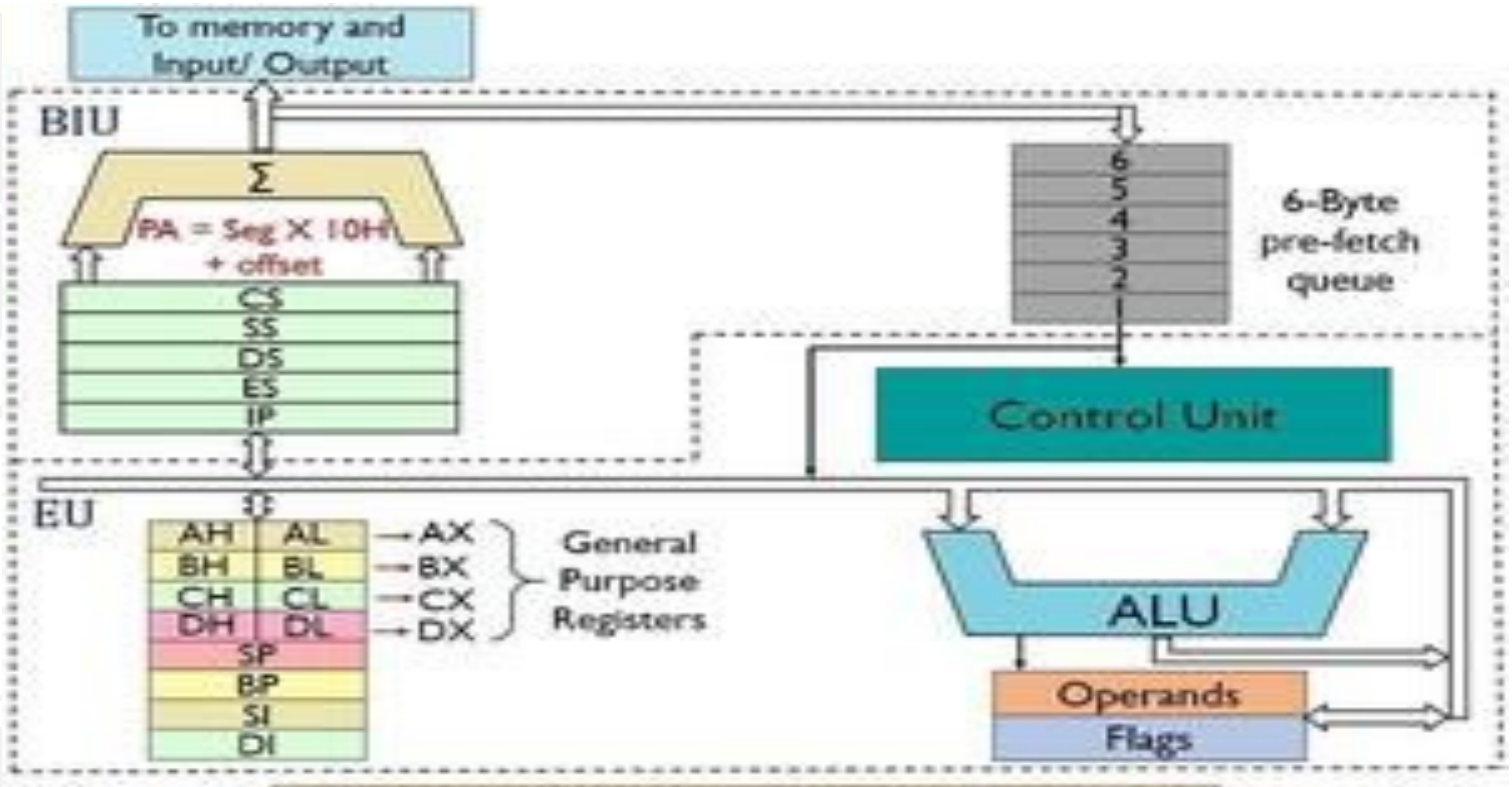

BIU
C-BUS
Σ
B-BUS
ES
CS
SS
DS
IP
6
5
4
3
2
1
INSTRUCTION STREAM BYTE QUEUE
CONTROL SYSTEM
EU
A-BUS
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
SI
ARITHMETIC LOGIC UNIT

## Microprocessor

- **8085 Microprocessor**
- 8085 Microprocessor is a predecessor of version 8086 Microprocessor, designed by Intel in 1976 with the help of NMOS technology.
- It includes a data bus of 8 bits, and 16 bits of the address bus, having a +5V voltage supply, and operates at 3.2 MHz single segment CLK.
- It has an internal clock generator and functions on a clock cycle having a duty cycle of 50%.
- 246 total operational codes and 80 instructions are present in the 8085 Microprocessor.

- **8086 Microprocessor**
- 8086 Microprocessor is an advanced version of the 8085 Microprocessor, designed by Intel in 1976.
- The number 8086 denotes the IC number of this microprocessor.
- It is a 16-bit microprocessor.
- It has 16 bits of the data bus, which is why it can read or write either 16 bits or 8 bits of data at a time.
- It has 20 bits of address lines that can access 220 address locations.
- It works in 2 modes-
- Maximum mode  and minimum mode

| 8085 microprocessor | 8086 microprocessor |
|---|---|
| The data bus is 8 bits. | The data bus is 16 bits. |
| The address bus is 16 bits. | The address bus is 20 bits. |
| The memory capacity is 64 KB. Also, 8085 Can Perform Operation Up to $2^8$ i.e. 256 numbers. A number greater than this is to be taken multiple times in an 8-bit data bus. | The memory capacity is 1 MB. Also, 8086 Can Perform operations up to $2^{16}$ i.e. 65,536 numbers. |
| The input/output port addresses are 8 bits. | The input/output port addresses are 16 bits. |
| The operating frequency is 3.2 MHz. | The operating frequency is 5 MHz, 8 MHz, and 10 MHz. |
| 8085 MP has a Single Mode Of Operation. | 8086 MP has Two Modes Of Operation. 1. Minimum Mode = Single CPU PROCESSOR 2. Maximum Mode = Multiple CPU PROCESSOR. |
| It does not have multiplication and division instructions. | It has multiplication and division instructions. |
| It does not support pipelining. | It supports pipe-lining as it has two independent units Execution Unit (EU) and Bus Interface Unit (BIU). |
| It does not support an instruction queue. | It supports an instruction queue. |
| Memory space is not segmented. | Memory space is segmented. |
| It consists of 5 flags(Sign Flag, Zero Flag, Auxiliary Carry Flag, Parity Flag, and Carry Flag). | It consists of 9 flags(Overflow Flag, Direction Flag, Interrupt Flag, Trap Flag, Sign Flag, Zero Flag, Auxiliary Carry Flag, Parity Flag, and Carry Flag). |
| It is a low-cost Microprocessor | It is a comparatively High-cost Microprocessor |



Block Diagram of 8086 Microprocessor

## Bus

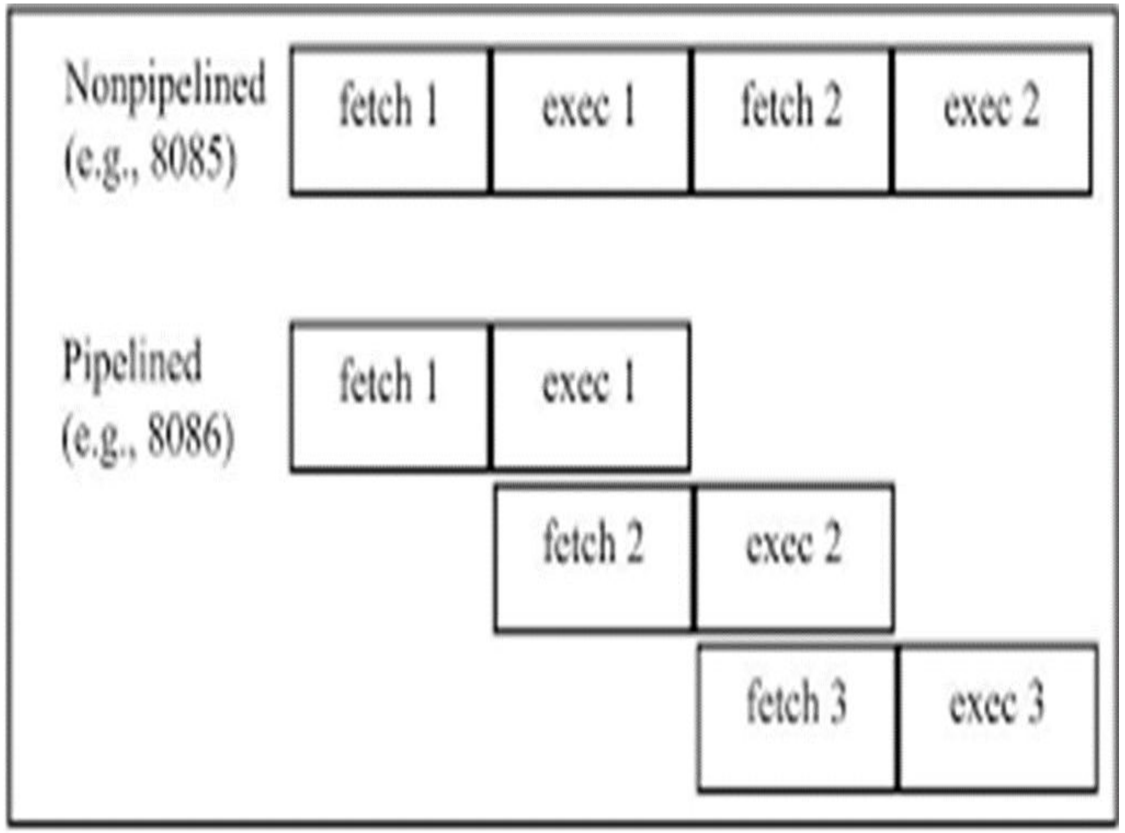
- Function of bus  : to transfer data and instructions between the microprocessor and other components in a computer system.
- There are 3 types of Bus
- 1.Address Bus: The address bus is used to send the memory address of the instruction or data being read or written. The address bus is 16 bits wide, allowing the 8086 to address up to 64 kilobytes of memory.
- 2.Data Bus: The data bus is used to transfer data between the microprocessor and memory. The data bus is 16 bits wide, allowing the 8086 to transfer 16-bit data words at a time.
- 3.Control Bus: The control bus is used to transfer control signals between the microprocessor and other components in the computer system. The control bus is used to send signals such as read, write, and interrupt requests, and to transfer status information between the microprocessor and other components.

## What is pipelining?

- The greater performance of the cpu is achieved by instruction pipelining.
- 8086 microprocesor has two blocks

➢ BIU(BUS INTERFACE UNIT)
➢ EU(EXECUTION UNIT)

- The BIU performs all bus operations such as instruction fetching,reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.
- EU executes instructions from the instruction system byte queue.
- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining.



## Architecture of 8086

The internal architecture of Intel 8086 is divided into 2 units: The Bus Interface Unit (BIU), and The Execution Unit (EU). These are explained as following below.

1. The Bus Interface Unit (BIU):
It provides the interface of 8086 to external memory and I/O devices via the System Bus. It performs various machine cycles such as memory read, I/O read, etc. to transfer data between memory and I/O devices.

BIU performs the following functions are as follows:

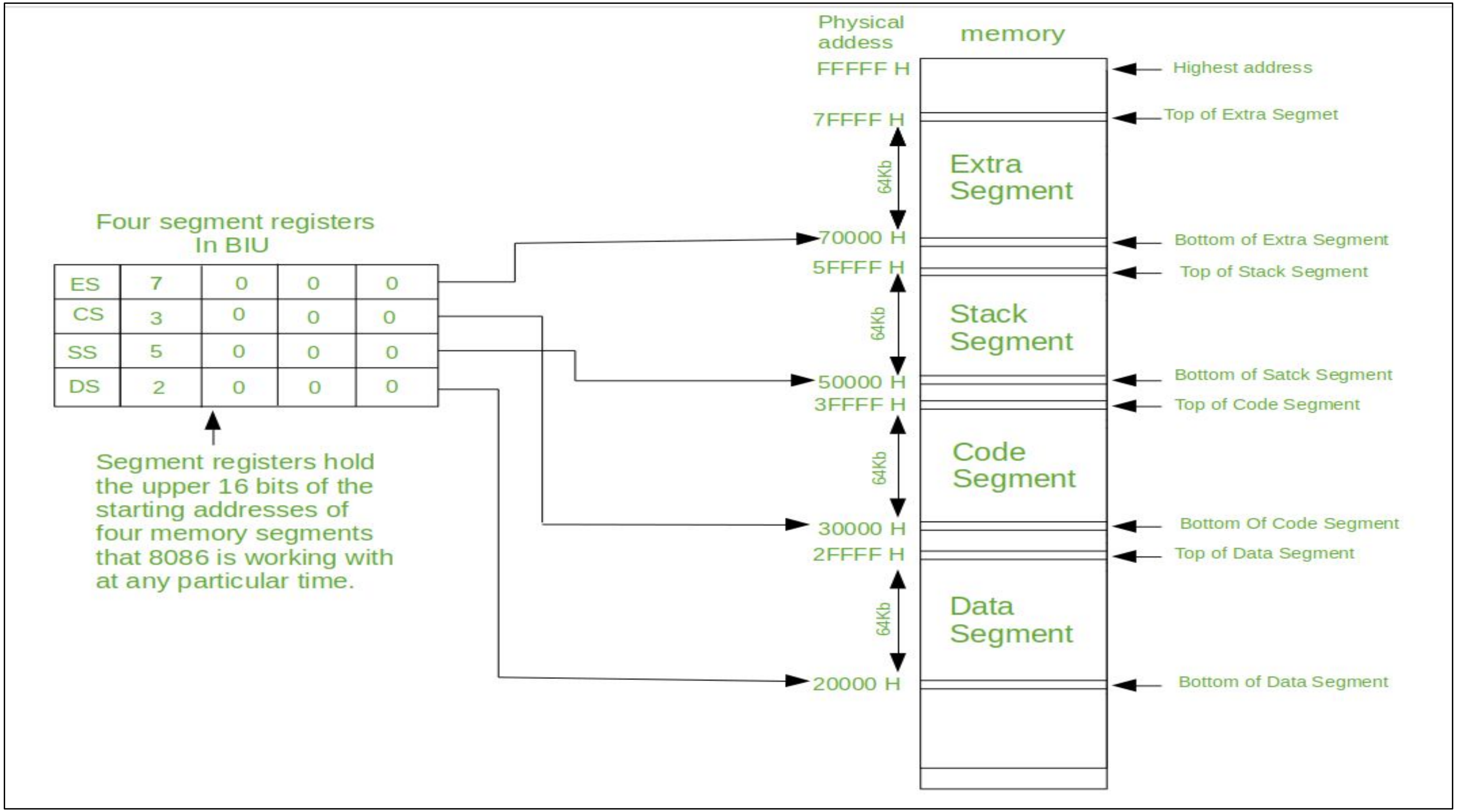
It generates the 20-bit physical address for memory access.
It fetches instructions from the memory.
It transfers data to and from the memory and I/O.
Maintains the 6-byte pre-fetch instruction queue(**supports pipelining**).
BIU mainly contains the 4 Segment registers, the Instruction Pointer, a pre-fetch queue, and an Address Generation Circuit.

## Instruction Pointer (IP):

- It is a 16-bit register. It holds offset of the next instructions in the Code Segment.
- IP is incremented after every instruction byte is fetched.
- IP gets a new value whenever a branch instruction occurs.
- CS is multiplied by 10H to give the 20-bit physical address of the Code Segment.
- The address of the next instruction is calculated by using the formula CS x 10H + IP.
- Example:

- CS = 4321H IP = 1000H
- then CS x 10H = 43210H + offset = 44210H
- Here Offset = Instruction Pointer(IP)
- This is the address of the next instruction.

- **Code Segment register: (16 Bit register):** CS holds the base address for the Code Segment. All programs are stored in the Code Segment and accessed via the IP.
- **Data Segment register: (16 Bit register):** DS holds the base address for the Data Segment.
- **Stack Segment register: (16 Bit register):** SS holds the base address for the Stack Segment.
- **Extra Segment register: (16 Bit register):** ES holds the base address for the Extra Segment.

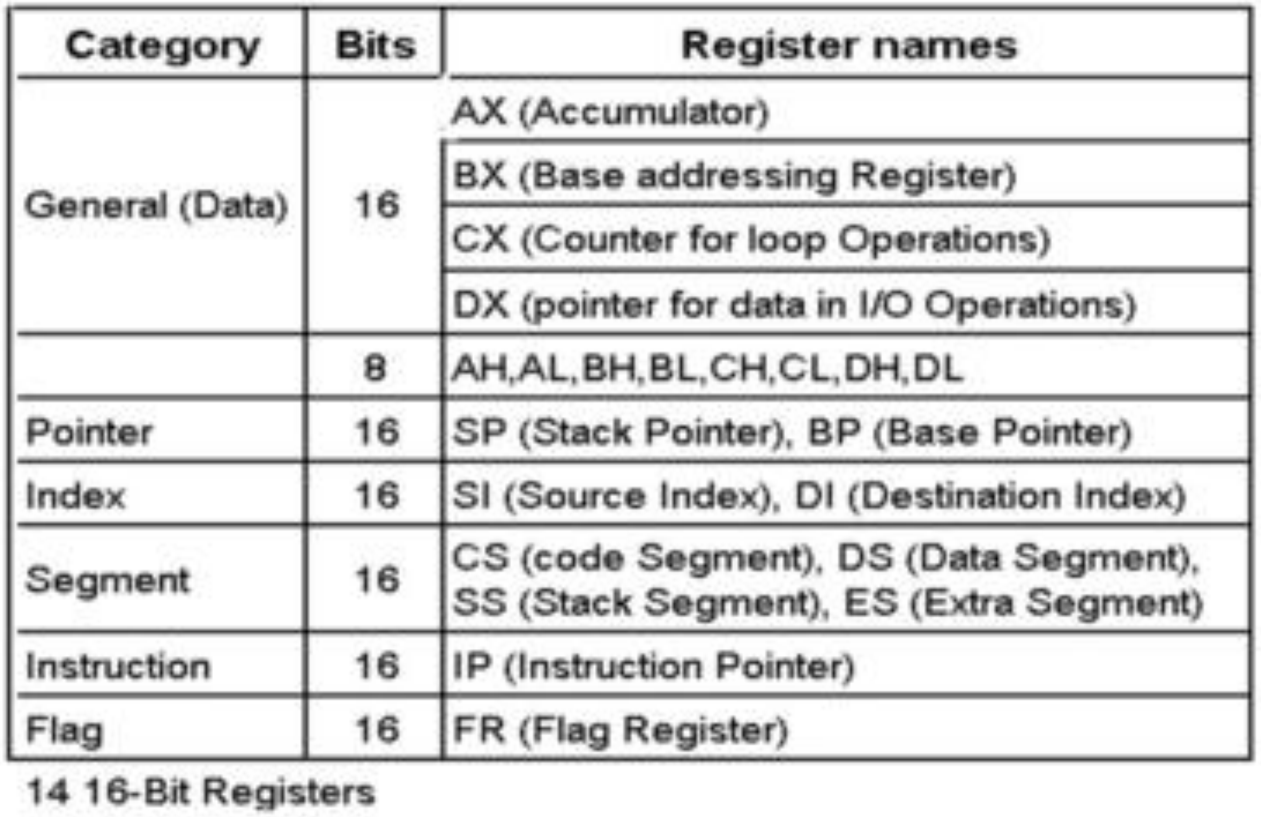

Four segment registers
In BIU
ES 7 0 0 0
CS 3 0 0 0
SS 5 0 0 0
DS 2 0 0 0
Segment registers hold the upper 16 bits of the starting addresses of four memory segments that 8086 is working with at any particular time.
Physical addess
memory
FFFFF H
7FFFF H
70000 H
5FFFF H
50000 H
3FFFF H
30000 H
2FFFF H
20000 H
64Kb
Extra Segment
Stack Segment
Code Segment
Data Segment
Highest address
Top of Extra Segmet
Bottom of Extra Segment
Top of Stack Segment
Bottom of Satck Segment
Top of Code Segment
Bottom Of Code Segment
Top of Data Segment
Bottom of Data Segment

## The Execution Unit (EU):

- The main components of the EU are General purpose registers, the ALU, Special purpose registers, the Instruction Register and Instruction Decoder, and the Flag/Status Register.

- Fetches instructions from the Queue in BIU, decodes, and executes arithmetic and logic operations using the ALU.
- Sends control signals for internal data transfer operations within the microprocessor.(Control Unit)
- Sends request signals to the BIU to access the external module.
- It operates with respect to T-states (clock cycles) and not machine cycles.

# Registers of the 8086/80286

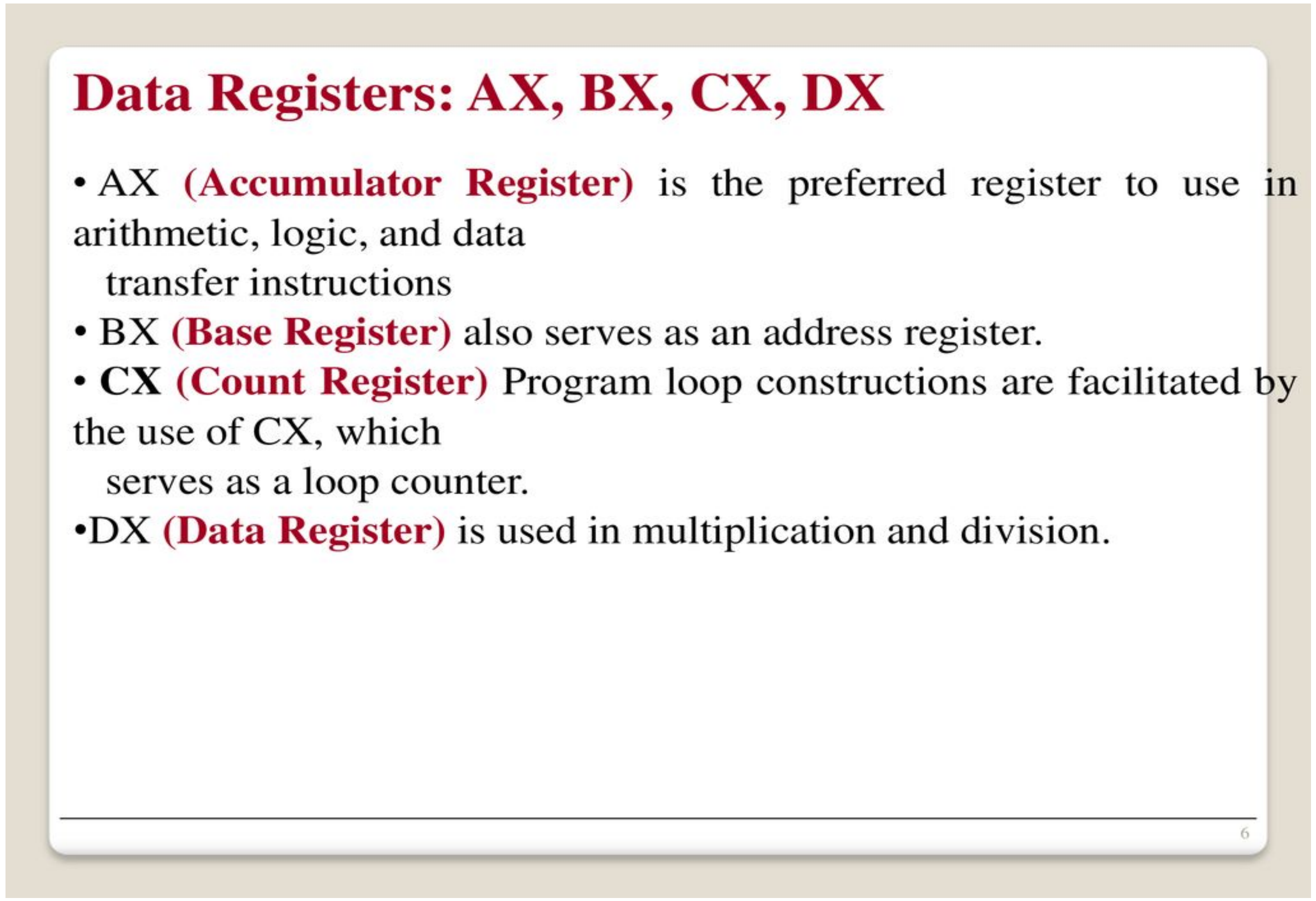


| Category | Bits | Register names |
|---|---|---|
| General (Data) | 16 | AX (Accumulator) |
| | | BX (Base addressing Register) |
| | | CX (Counter for loop Operations) |
| | | DX (pointer for data in I/O Operations) |
| | 8 | AH,AL,BH,BL,CH,CL,DH,DL |
| Pointer | 16 | SP (Stack Pointer), BP (Base Pointer) |
| Index | 16 | SI (Source Index), DI (Destination Index) |
| Segment | 16 | CS (code Segment), DS (Data Segment), SS (Stack Segment), ES (Extra Segment) |
| Instruction | 16 | IP (Instruction Pointer) |
| Flag | 16 | FR (Flag Register) |

14 16-Bit Registers

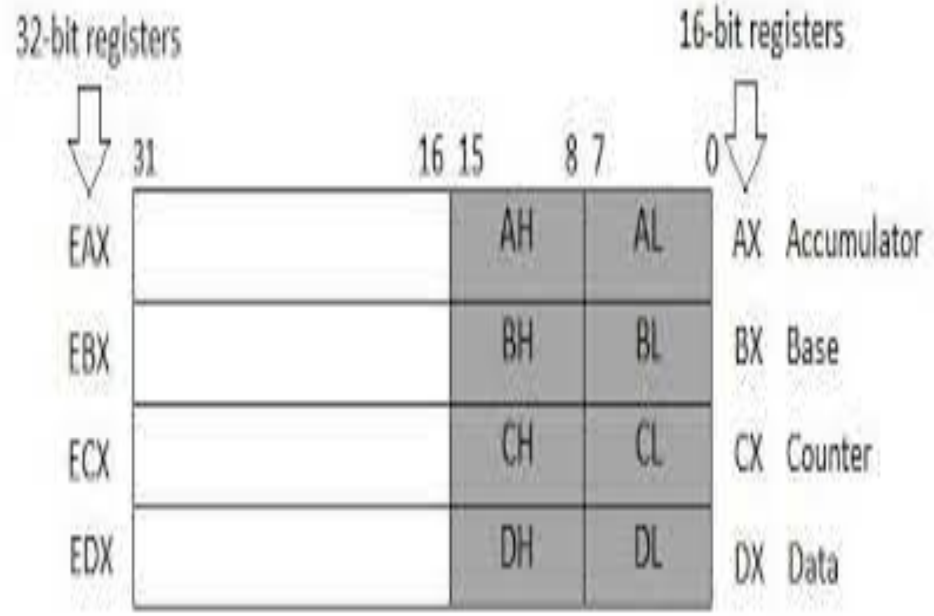
## GENERAL PURPOSE REGISTERS

- 8086 has four 16-bit general purpose registers AX, BX, CX, and DX which store intermediate values during execution.
- Each of these has two 8-bit parts (higher and lower).
- **AX register**: (Combination of AL and AH Registers)
- It holds operands and results during multiplication and division operations. Also, an accumulator during String operations.
- **BX register:** (Combination of BL and BH Registers)
- It holds the memory address (offset address) in indirect addressing modes.
- **CX register:** (Combination of CL and CH Registers)
- It holds the count for instructions like a loop, rotates, shifts and string operations.
- **DX register:** (Combination of DL and DH Registers)
- It is used with AX to hold 32-bit values during multiplication and division.



## Data Registers: AX, BX, CX, DX

• AX **(Accumulator Register)** is the preferred register to use in arithmetic, logic, and data
 transfer instructions
• BX **(Base Register)** also serves as an address register.
• **CX** **(Count Register)** Program loop constructions are facilitated by the use of CX, which
 serves as a loop counter.
•DX **(Data Register)** is used in multiplication and division.



- **Arithmetic Logic Unit** (16-bit): Performs 8 and 16-bit arithmetic and logic operations.
- **Special purpose registers** (16-bit): Special purpose registers are also called as Offset registers. Which points to specific memory locations under each segment.
- We can understand the concept of segments as Textbook pages.
- Suppose there are 10 chapters in one textbook and each chapter takes exactly 100 pages. So the book will contain 1000 pages.
- Now suppose we want to access page number 575 from the book then 500 will be the segment base address which can be anything in the context of microprocessors like Code, Data, Stack, and Extra Segment. So, 500 will be segment registers that are present in Bus Interface Unit (BIU).
- And 500 + 75 is called an offset register through which we can reach on specific page number under a specific segment.

## Execution unit-registers

- Stack Pointer: Points to Stack top. Stack is in Stack Segment, used during instructions like PUSH, POP, CALL, RET etc.
- Base Pointer: BP can hold the offset addresses of any location in the stack segment. It is used to access random locations of the stack.
- Source Index: It holds offset address in Data Segment during string operations.
- Destination Index: It holds offset address in Extra Segment during string operations.

- **Instruction Register and Instruction Decoder:**
- The EU fetches an opcode from the queue into the instruction register. The instruction decoder decodes it and sends the information to the control circuit for execution.
- Flag/Status register (16 bits): It has 9 flags (6 status and 3 control ) that help change or recognize the state of the microprocessor.
- **6 Status flags:**

 Carry flag(CF)

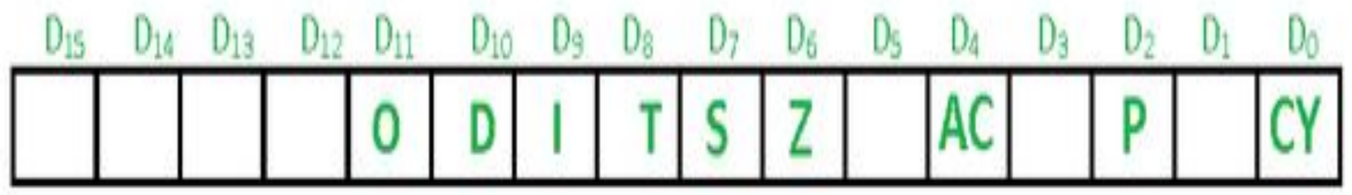
 Parity flag(PF)

 Auxiliary carry flag(AF)

 Zero flag(Z)

 Sign flag(S)

 Overflow flag (O)



## Flags

**Conditional/status flags :**
They indicate the status of result of ALU .
1.Carry flag: Becomes 1 when carry is generated at MSB
2.Parity flag: Indicates parity of the result of ALU.
3. Auxiliary carry flag : Indicates overflow at 4th bit in case of BCD arithmetic.
4.Sign flag: Equal to MSB of result.
5. Zero flag: Is set when the result of an ALU operation is zero.
6.overflow flag: Indicates overflow into signed bit. Significant for signed numbers.

- Status flags are updated after every arithmetic and logic operation.
- **3 Control flags:**

These flags are not implicitly set by program execution but explicitly set by the
programmer as per requirement.
1.Trap flag: Used for debugging. Setting this flag enables single step trap for debugging purpose.
2.Interrupt flag: The processor services interrupts only if this flag is set. One could call this as Interrupt Enable.
3.Direction flag: This is significant in case of string operations. If directional flag is set, the access of the string data happens from higher memory location towards lower memory location and vice versa when flag is cleared, i.e. made zero.

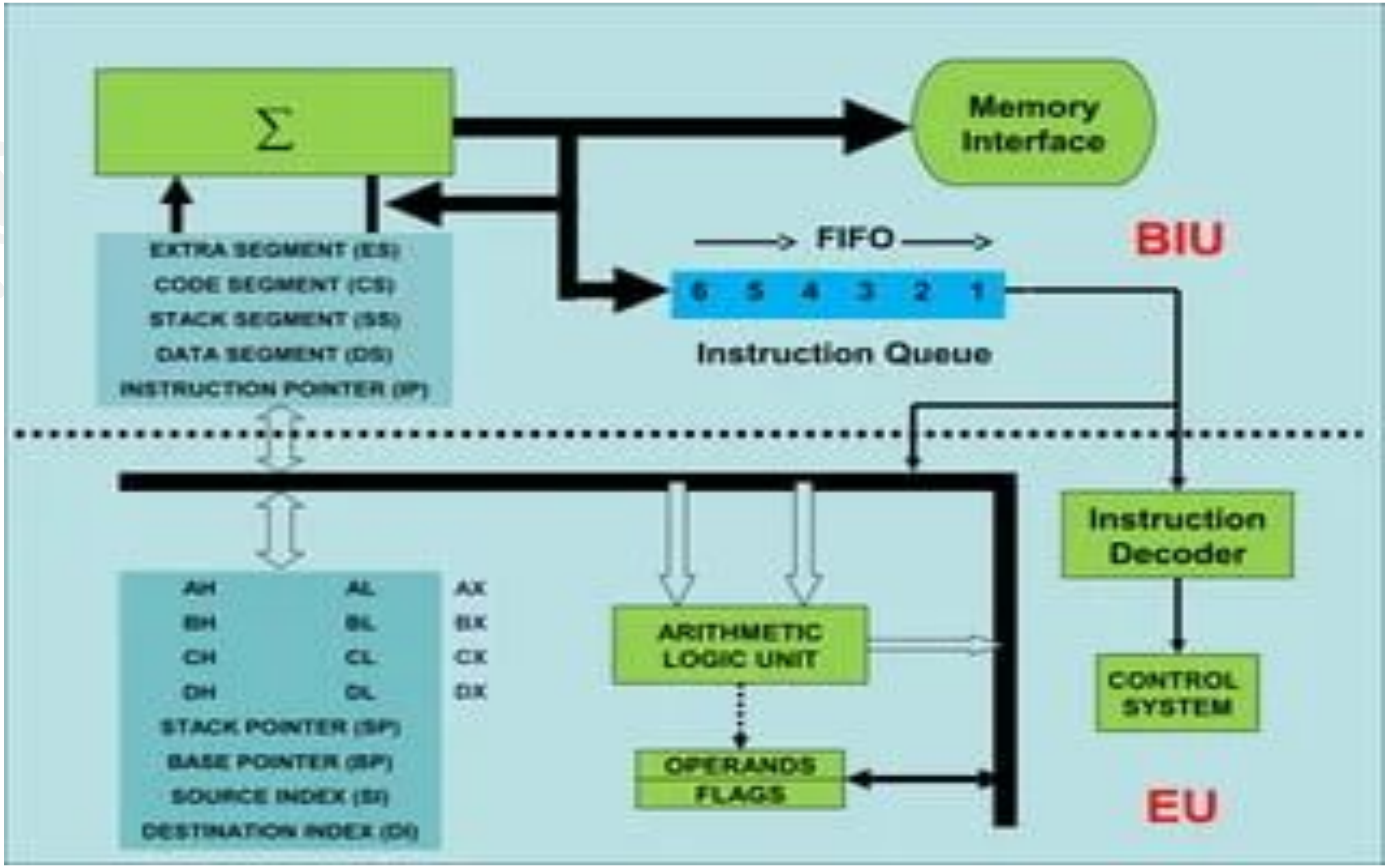
## Decode unit

- The Decode Unit in the 8086 microprocessor is a component that decodes the instructions that have been fetched from memory. The decode unit takes the machine code instructions and translates them into micro-operations that can be executed by the microprocessor’s execution unit.
- The Decode Unit is responsible for decoding instructions, performing register-to-register operations, and performing memory-to-register operations. It also decodes conditional jumps, calls, and returns, and performs data transfers between memory and registers.

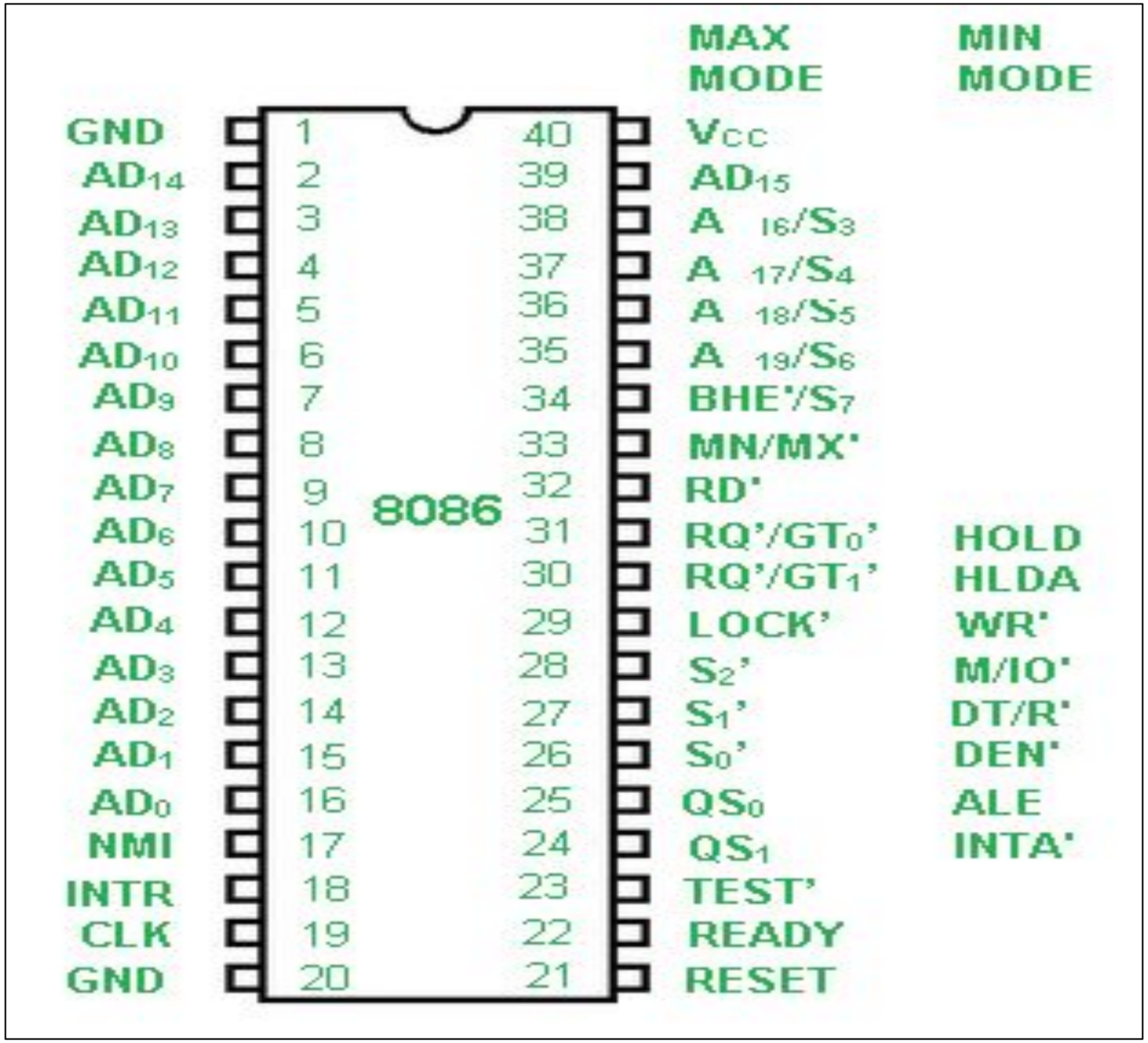
- The Decode Unit helps to improve the performance of the 8086 microprocessor by allowing it to execute instructions quickly and accurately. This improved performance helps to ensure that the 8086 remains competitive in its performance and capabilities, even as technology continues to advance.

## Control unit

- The Control Unit in the 8086 microprocessor is a component that manages the overall operation of the microprocessor. The control unit is responsible for controlling the flow of instructions through the microprocessor and coordinating the activities of the other components, including the Decode Unit, Execution Unit, and Prefetch Unit.
- The Control Unit acts as the central coordinator for the microprocessor, directing the flow of data and instructions and ensuring that the microprocessor operates correctly. It also monitors the state of the microprocessor, ensuring that the correct sequence of operations is followed.
- The Control Unit is responsible for fetching instructions from memory, decoding them, executing them, and updating the microprocessor’s state. It also handles interrupt requests and performs system management tasks, such as power management and error handling.


Σ
Memory Interface
FIFO
BIU
EXTRA SEGMENT (ES)
CODE SEGMENT (CS)
STACK SEGMENT (SS)
DATA SEGMENT (DS)
INSTRUCTION POINTER (IP)
6 5 4 3 2 1
Instruction Queue
Instruction Decoder
CONTROL SYSTEM
AH AL AX
BH BL BX
CH CL CX
DH DL DX
STACK POINTER (SP)
BASE POINTER (BP)
SOURCE INDEX (SI)
DESTINATION INDEX (DI)
ARITHMETIC LOGIC UNIT
OPERANDS
FLAGS
EU

# 8086 in diagram



|  | **MIN MODE** | **MAX MODE** |
|---|---|---|
| 1 | It is a **uniprocessor mode.** 8086 is the only processor in the circuit. | It is a **multiprocessor mode.** Along with 8086, there can be other processors like 8087 and 8089 in the circuit. |
| 2 | Here **MN/ $\overline{\text{MX}}$** is connected to **Vcc.** | Here **MN/ $\overline{\text{MX}}$** is connected to **Ground.** |
| 3 | **ALE** for the latch is **given by 8086** itself. | As there are multiple processors, **ALE** for the latch is **given by 8288 bus controller.** |
| 4 | **$\overline{\text{DEN}}$** and **DT/ $\overline{\text{R}}$** for the transreceivers **are given by 8086 itself.** | As there are multiple processors, **DEN** and **DT/ $\overline{\text{R}}$** for the transreceivers is **given by 8288 bus controller.** |
| 5 | Direct control signals like **M/ $\overline{\text{IO}}$** , **$\overline{\text{RD}}$ and $\overline{\text{WR}}$ are produced by 8086 itself.** | Instead of control signals, all processors produce status signals **$\overline{S_2}$ , $\overline{S_1}$ and $\overline{S_0}$** |
| 6 | Control signals **M/ $\overline{\text{IO}}$ , $\overline{\text{RD}}$ and $\overline{\text{WR}}$ are decocded by a 3:8 decoder IC 74138.** | Status signals **$\overline{S_2}$ , $\overline{S_1}$ and $\overline{S_0}$ require special decoding** are decoded by **8288 bus controller.** |
| 7 | **$\overline{\text{INTA}}$** for interrupt acknowledgement **is produced by 8086.** | **$\overline{\text{INTA}}$** for interrupt acknowledgement **is produced by 8288 Bus Controller.** |
| 8 | **Bus request are grant** is handled using **HOLD and HLDA** signals. | **Bus request are grant** is handled using **$\overline{\text{RQ}}$ / $\overline{\text{GT}}$** signals. |
| 9 | Since **74138** does not independently generate any signals, it **does not need a CLK.** | Since **8288** independently generates control signals, it **needs a CLK from 8284** clock generator. |
| 10 | The circuit is **simpler but does not support multiprocessing.** | The circuit is **more complex but supports multiprocessing.** |

## Details

- Intel 8086 is a 16-bit HMOS microprocessor.
- It is available in 40 pin DIP chip. It uses a 5V DC supply for its operation.
- The 8086 uses a 20-line address bus.
- It has a 16-line data bus.
- The 20 lines of the address bus operate in multiplexed mode.
- The 16-low order address bus lines have been multiplexed with data and 4 high-order address bus lines have been multiplexed with status signals.
- **AD0-AD15: Address/Data bus.**
- These are low order address bus. They are multiplexed with data. When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A0-A15. When data are transmitted over AD lines the symbol D is used in place of AD, for example D0-D7, D8-D15 or D0-D15.
- A16-A19: **High order address bus**. These are multiplexed with status signals.

- RD’: This is used for read operation. It is an output signal. It is active when low.
- READY : This is the acknowledgement from the memory or slow device that they have completed the data transfer.
- INTR : Interrupt Request. This is triggered input. This is sampled during the last clock cycles of each instruction for determining the availability of the request. If any interrupt request is found pending, the processor enters the interrupt acknowledge cycle.
- NMI : Non maskable interrupt. This is an edge triggered input which results in a type II interrupt. NMI is non-maskable internally by software. A transition made from low(0) to high(1) initiates the interrupt at the end of the current instruction.
- MN/MX’ : Minimum/Maximum. This pin signal indicates what mode the processor will operate in.

| S2 | S1 | S0 | Characteristics |
|---|---|---|---|
| 0 | 0 | 0 | Interrupt acknowledge |
| 0 | 0 | 1 | Read I/O port |
| 0 | 1 | 0 | Write I/O port |
| 0 | 1 | 1 | Halt |
| 1 | 0 | 0 | Code access |
| 1 | 0 | 1 | Read memory |
| 1 | 1 | 0 | Write memory |
| 1 | 1 | 1 | Passive state |

- LOCK’ : Its an active low pin. It indicates that other system bus masters have not been allowed to gain control of the system bus while LOCK’ is active low(0). The LOCK signal will be active until the completion of the next instruction.
- TEST’ : This examined by a ‘WAIT’ instruction. If the TEST pin goes low(0), execution will continue, else the processor remains in an idle state. The input is internally synchronized during each of the clock cycle on leading edge of the clock.
- CLK : Clock Input. The clock input provides the basic timing for processing operation and bus control activity. Its an asymmetric square wave with a 33% duty cycle.
- RESET : This pin requires the microprocessor to terminate its present activity immediately. The signal must be active high(1) for at least four clock cycles.

## Queue Status

- QS1,QS0 : These signals indicate the status of the internal 8086 instruction queue according to the table shown below:

| QS1 | QS0 | Status |
|---|---|---|
| 0 | 0 | No operation |
| 0 | 1 | First byte of op code from queue |
| 1 | 0 | Empty the queue |
| 1 | 1 | Subsequent byte from queue |

- DT/R : Data Transmit/Receive. This pin is required in minimum systems, that want to use an 8286 or 8287 data bus transceiver. The direction of data flow is controlled through the transceiver.

- DEN: Data enable. This pin is provided as an output enable for the 8286/8287 in a minimum system which uses transceiver. DEN is active low(0) during each memory and input-output access and for INTA cycles.

- HOLD/HOLDA: HOLD indicates that another master has been requesting a local bus .This is an active high(1). The microprocessor receiving the HOLD request will issue HLDA (high) as an acknowledgement in the middle of a T4 or T1 clock cycle.

- ALE : Address Latch Enable. ALE is provided by the microprocessor to latch the address into the 8282 or 8283 address latch. It is an active high(1) pulse during T1 of any bus cycle. ALE signal is never floated, is always integer.

## Instruction Set

- The 8086 microprocessor supports 8 types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

- **Data Transfer Instructions**

- These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

- Instruction to transfer a word
- MOV − Used to copy the byte or word from the provided source to the provided destination.

- PPUSH − Used to put a word at the top of the stack.

- POP − Used to get a word from the top of the stack to the provided location.

- PUSHA − Used to put all the registers into the stack.

- POPA − Used to get words from the stack to all registers.

- XCHG − Used to exchange the data from two locations.

- XLAT − Used to translate a byte in AL using a table in the memory.

- **Instructions for input and output port transfer**
- IN − Used to read a byte or word from the provided port to the accumulator.
- OUT − Used to send out a byte or word from the accumulator to the provided port.
- **Instructions to transfer the address**
- LEA − Used to load the address of operand into the provided register.
- LDS − Used to load DS register and other provided register from the memory
- LES − Used to load ES register and other provided register from the memory.

- Instructions to transfer flag registers
- LAHF – Used to load AH with the low byte of the flag register.

- SAHF – Used to store AH register to low byte of the flag register.

- PUSHF – Used to copy the flag register at the top of the stack.

- POPF – Used to copy a word at the top of the stack to the flag register.

## Arithmetic Instructions

- These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.
- Following is the list of instructions under this group −
- Instructions to perform addition
- ADD − Used to add the provided byte to byte/word to word.
- ADC − Used to add with carry.
- INC − Used to increment the provided byte/word by 1.
- AAA − Used to adjust ASCII after addition.
- DAA − Used to adjust the decimal after the addition/subtraction operation.

## Instructions to perform subtraction

- SUB − Used to subtract the byte from byte/word from word.
- SBB − Used to perform subtraction with borrow.
- DEC − Used to decrement the provided byte/word by 1.
- NPG − Used to negate each bit of the provided byte/word and add 1/2’s complement.
- CMP − Used to compare 2 provided byte/word.
- AAS − Used to adjust ASCII codes after subtraction.
- DAS − Used to adjust decimal after subtraction.

- **Instructions to perform multiplication**
- MUL − Used to multiply unsigned byte by byte/word by word.
- IMUL − Used to multiply signed byte by byte/word by word.
- AAM − Used to adjust ASCII codes after multiplication.
- **Instructions to perform division**
- DIV − Used to divide the unsigned word by byte or unsigned double word by word.
- IDIV − Used to divide the signed word by byte or signed double word by word.
- AAD − Used to adjust ASCII codes after division.
- CBW − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- CWD − Used to fill the upper word of the double word with the sign bit of the lower word.

- Instructions to perform logical operation
- NOT − Used to invert each bit of a byte or word.

- AND − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.

- OR − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.

- XOR − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.

- TEST − Used to add operands to update flags, without affecting operands

- Instructions to perform shift operations
- SHL/SAL – Used to shift bits of a byte/word towards left and put zero(S) in LSBs.

- SHR – Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.

- SAR – Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

- Instructions to perform rotate operations
- ROL − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- ROR − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- RCR − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- RCL − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

- Interrupt Instructions
- These instructions are used to call the interrupt during program execution.

- INT – Used to interrupt the program during execution and calling service specified.

- INTO – Used to interrupt the program during execution if OF = 1

- IRET – Used to return from interrupt service to the main program

- Program Execution Transfer Instructions (Branch and Loop Instructions)
- These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

- Instructions to transfer the instruction during an execution without any condition −

- CALL − Used to call a procedure and save their return address to the stack.

- RET − Used to return from the procedure to the main program.

- JMP − Used to jump to the provided address to proceed to the next instruction.

## Addressing Modes

- The different ways in which a source operand is denoted in an instruction is known as addressing modes. There are 8 different addressing modes in 8086 programming −

1) Immediate addressing mode

- The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

- Example
- MOV CX, 4929 H, ADD AX, 2387 H,  MOV AL, FFH

## Addressing Modes(contd)

2.Register addressing mode

- It means that the register is the source of an operand for an instruction.

- Example
- MOV CX, AX   ; copies the contents of the 16-bit AX register into
-              ; the 16-bit CX register),
- ADD BX, AX

## Addressing Modes(contd)

3.Direct addressing mode

- The addressing mode in which the effective address of the memory location is written directly in the instruction.

- Example
- MOV AX, [1592H], MOV AL, [0300H]

## Addressing Modes(contd)

4.Register indirect addressing mode

- This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

- Example
- MOV AX, [BX]  ; Suppose the register BX contains 4895H, then the contents

; 4895H are moved to AX

- ADD CX, {BX}

## Addressing Modes(contd)

5.Based addressing mode

- In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.

- Example
- MOV DX, [BX+04], ADD CL, [BX+08]

## Addressing Modes(contd)

6.Indexed addressing mode

- In this addressing mode, the operand's offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.

- Example
- MOV BX, [SI+16], ADD AL, [DI+16]

## Addressing Modes(contd)

- Based-index addressing mode
- In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.

- Example
- ADD CX, [AX+SI], MOV AX, [AX+DI]

# Addressing Modes(contd)

- Based indexed with displacement mode
- In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

- Example
- MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]

## Assembler Directives(contd)

- ASSEMBLER DIRECTIVES
- Assembler directives are the commands to the assembler that direct the assembly process.
- They indicate how an operand is treated by the assembler and how assembler handles the program.
- They also direct the assembler how program and data should arrange in the memory.
- ALP's are composed of two type of statements.
- (i) The instructions which are translated to machine codes by assembler.
- (ii) The directives that direct the assembler during assembly process, for which no machine code is generated.

## Assembler Directives (contd)

- 1. ASSUME: Assume logical segment name.
- The ASSUME directive is used to inform the assembler the names of the logical segments to be assumed for different
- segments used in the program .In the ALP each segment is given name.
- Syntax: ASSUME segreg:segname,…segreg:segname
- Ex: ASSUME CS:CODE
- ASSUME CS:CODE,DS:DATA,SS:STACK

# Assembler Directives (contd)

- **2. DB: Define Byte**
- The DB directive is used to reserve byte or bytes of memory locations in the available memory.
- Syntax: Name of variable DB initialization value.
- Ex: MARKS DB 35H,30H,35H,40H
- NAME DB “VARDHAMAN”
- **3. DW: Define Word**
- The DW directive serves the same puposes as the DB directive,but it now makes the assembler reserve the number of
- memory words(16-bit) instead of bytes.
- Syntax: variable name DW initialization values.
- Ex: WORDS DW 1234H,4567H,2367H
- WDATA DW 5 Dup(522h)

## Assembler Directives (contd)

- **4. DD: Define Double:**
- The directive DD is used to define a double word (4bytes) variable.
- Syntax: variablename DD 12345678H
- Ex: Data1 DD 12345678H
- **5. DQ: Define Quad Word**
- This directive is used to direct the assembler to reserve 4 words (8 bytes) of memory for the specified variable and
- may initialize it with the specified values.
- Syntax: Name of variable DQ initialize values.
- Ex: Data1 DQ 123456789ABCDEF2H
- **6. DT: Define Ten Bytes**
- The DT directive directs the assembler to define the specified variable requiring 10 bytes for its storage and initialize
- the 10-bytes with the specified values.
- Syntax: Name of variable DT initialize values.
- Ex: Data1 DT 123456789ABCDEF34567H

# Assembler Directives (contd)

- **7. END: End of Program**
- The END directive marks the end of an ALP. The statement after the directive END will be ignored by the assembler.
- **8. ENDP: End of Procedure**
- The ENDP directive is used to indicate the end of procedure. In the AL programming the subroutines are called
- procedures.
- Ex: Procedure Start
- **9. ENDS: End of segment**
- The ENDS directive is used to indicate the end of segment.
- Ex: DATA SEGMENT
- :
- DATA ENDS

## Assembler Directives (contd)

- 10.EVEN: Align on Even memory address
- The EVEN directives updates the location counter to the next even address.
- Ex: EVEN
- Procedure Start
- :
- Start ENDP
- � The above structure shows a procedure START that is to be aligned at an even address.
- 11.EQU: Equate
- The directive EQU is used to assign a label with a value or symbol.
- Ex: LABEL EQU 0500H
- ADDITION EQU ADD

# Assembler Directives (contd)

- 12.EXTRN: External and public
-  The directive EXTRN informs the assembler that the names, procedures and labels declared after this directive have
- been already defined in some other AL modules.
-  While in other module, where names, procedures and labels actually appear, they must be declared public using
- the PUBLIC directive.
- Ex: MODULE1 SEGMENT
- PUBLIC FACT FAR
- MODULE1 ENDS
- MODULE2 SEGMENT
- EXTRN FACT FAR
- MODULE2 END

# Assembler Directives (contd)

- 13.GROUP: Group the related segments
- This directive is used to form logical groups of segments with similar purpose or type.
- Ex: PROGRAM GROUP CODE, DATA, STACK
- *CODE, DATA and STACK segments lie within a 64KB memory segment that is named as PROGRAM.
- 14.LABEL: label
- The label is used to assign name to the current content of the location counterEx: CONTINUE LABEL FAR
- The label CONTINUE can be used for a FAR jump, if the program contains the above statement.

# Assembler Directives (contd)

- **15.LENGTH**: Byte length of a label
- This is used to refer to the length of a data array or a string
- Ex : MOV CX, LENGTH ARRAY
- **16.LOCAL**: The labels, variables, constant or procedures are declared LOCAL in a module are to be used only by the
- particular module.
- Ex : LOCAL a, b, Data1, Array, Routine
- **17.NAME**: logical name of a module
- The name directive is used to assign a name to an assembly language program module. The module may now be refer
- to by its declared name.
- Ex : Name “addition”

# Assembler Directives (contd)

- 18.OFFSET: offset of a label
- When the assembler comes across the OFFSET operator along with a label, it first computing the 16-bit offset address
- of a particular label and replace the string 'OFFSET LABEL' by the computed offset address.
- Ex : MOV SI, offset list
- 19.ORG: origin
- The ORG directive directs the assembler to start the memory allotment for the particular segment, block or code from
- the declared address in the ORG statement.
- Ex: ORG 1000H
- 20.PROC: Procedure
- The PROC directive marks the start of a named procedure in the statement.
- Ex: RESULT PROC NEAR
- ROUTINE PROC FAR

## Assembler Directives (contd)

- 21.PTR: pointer
- The PTR operator is used to declare the type of a label, variable or memory operator.
- Ex : MOV AL, BYTE PTR [SI]
- MOV BX, WORD PTR [2000H]
- 22.SEG: segment of a label
- The SEG operator is used to decide the segment address of the label, variable or procedure.
- Ex : MOV AX, SEG ARRAY
- MOV DS, AX
- 23.SEGMENT: logical segment
- The segment directive marks the starting of a logical segment
- Ex: CODE SEGMENT
- :
- CODE ENDS

## Assembler Directives (contd)

- 24.SHORT: The SHORT operator indicates to the assembler that only one byte is required to code the displacement for
- jump.
- Ex : JMP SHORT LABEL
- 25.TYPE: The TYPE operator directs the assembler to decide the data type of the specified label and replaces the TYPE
- label by the decided data type.
- For word variable, the data type is 2.
- For double word variable, the data type is 4.
- For byte variable, the data type is 1.
- Ex : STRING DW 2345H, 4567H
- MOV AX, TYPE STRING
- AX=0002H

# Assembler Directives (contd)

- 26.GLOBAL: The labels, variables, constants or procedures declared GLOBAL may be used by other modules of the
- program.
- Ex : ROUTINE PROC GLOBAL.
- 27.FAR PTR: This directive indicates the assembler that the label following FAR PTR is not available within the same
- segment and the address of the label is of 32-bits i.e 2-bytes of offset followed by 2-bytes of segment address.
- Ex : JMP FAR PTR LABEL
- 28.NEAR PTR: This directive indicates that the label following NEAR PTR is in the same segment and needs only 16-bit
- i.e 2-byte offset to address it
- Ex : JMP NEAR PTR LABEL
- CALL NEAR PTR ROUTINE

## Procedure and Macro.

- When we need to use a group of instructions several times throughout a program there are two ways we can avoid
- having to write the group of instructions each time we want to use them.
- 1. One way is to write the group of instructions as a separate procedure.
- 2. Another way we can use macros

# Procedure and Macro

- Procedures:
- ▯ The procedure is a group of instructions stored as a separate program in the memory and it is called from the
- main program whenever required using CALL instruction.
- ▯ For calling the procedure we have to store the return address (next instruction address followed by CALL) onto
- the stack.
- ▯ At the end of the procedure RET instruction used to return the execution to the next instruction in the main
- program by retrieving the address from the top of the stack.
- ▯ Machine codes for the procedure instructions put only once in memory.
- ▯ The procedure can be defined anywhere in the program using assembly directives PROC and ENDP.

# Procedure and Macro

- The four major ways of passing parameters to and from a procedure are:
- 1. In registers
- 2. In dedicated memory location accessed by name
- 3 .With pointers passed in registers
- 4. With the stack
- � The type of procedure depends on where the procedure is stored in the memory.
- � If it is in the same code segment where the main program is stored the it is called near procedure otherwise it is
- referred to as far procedure.
- � For near procedure CALL instruction pushes only the IP register contents on the stack, since CS register contents
- remains unchanged for main program.
- � But for Far procedure CALL instruction pushes both IP and CS on the stack.

## Procedure and Macro

- Macros:
- A macro is a group of repetitive instructions in a program which are codified only once and can be used as many
- times as necessary.
- A macro can be defined anywhere in program using the directives MACRO and ENDM
- Each time we call the macro in a program, the assembler will insert the defined group of instructions in place of the
- call.
- The assembler generates machine codes for the group of instructions each time the macro is called.
- Using a macro avoids the overhead time involved in calling and returning from a procedure.

- **Advantage of Procedure and Macros:**
- Procedures:
- **Advantages**
- The machine codes for the group of instructions in the procedure only have to be put once.
- **Disadvantages**
- Need for stack
- Overhead time required to call the procedure and return to the calling program.
- **Macros:**
- **Advantages**
- Macro avoids overhead time involving in calling and returning from a procedure.
- **Disadvantages**
- Generating in line code each time a macro is called is that this will make the program take up more memory
- than using a procedure

| | PROCEDURES | MACROS |
|---|---|---|
| | Accessed by CALL and RET mechanism during program execution | Accessed by name given to macro when defined during assembly |
| | Machine code for instructions only put in memory once | Machine code generated for instructions each time called |
| | Parameters are passed in registers, memory locations or stack | Parameters passed as part of statement which calls macro |
| | Procedures uses stack | Macro does not utilize stack |
| | A procedure can be defined anywhere in program using the directives PROC and ENDP | A macro can be defined anywhere in program using the directives MACRO and ENDM |
| | Procedures takes huge memory for CALL (3 bytes each time CALL is used) instruction | Length of code is very huge if macro’s are called for more number of times |