Department of Computer Engineering

R. C. PATEL
INSTITUTE OF TECHNOLOGY
An Autonomous Institute

Vision of the Institute

To achieve excellence in engineering education with strong ethical values.

Mission of the Institute

To impart high quality Technical Education through:

- Innovative and Interactive learning process and high quality, internationally recognized instructional programs.
- Fostering a scientific temper among students by the means of a liaison with the Academia, Industries and Government.
- Preparing students from diverse backgrounds to have aptitude for research and spirit of Professionalism.
- Inculcating in students a respect for fellow human beings and responsibility towards the society.

Vision of the Department

To provide prominent computer engineering education with socio-moral values.

Mission of the Department

M1 To provide state-of-the-art ICT based teaching-learning process.

M2 To groom the students to become professionally sound computer engineers to meet growing needs of industry and society.

M3 To make the students responsible human being by inculcating ethical values.

Program Educational Objectives (PEOs) of the Department

PEO1 To provide the foundation of lifelong learning skills for advancing their careers being a professional, entrepreneur and leader.

PEO2 To develop computer professionals to fulfill Industry expectations.

PEO3 To foster ethical and social values to be socially responsible human being.

Sub- Processor Organization and Architecture (POA)

Teaching Scheme	Examination Scheme
Lectures : 03 Hrs./week	Term Test : 15 Marks
Credits : 03	Teacher Assessment : 20 Marks
	End Sem Exam : 65 Marks
	Total Marks : 100 Marks

Sub- Processor Organization and Architecture (POA)

Course Objectives:

1. To have a thorough understanding of the basic structure and operations of a computer system.
2. To study the hierarchical memory system including cache memories and virtual memory.
3. To prepare students for higher processor architectures and embedded systems.
4. To apply innovative solutions and make progress in the knowledge to exploit the new paradigms of computing, particularly in distributed environments.

Sub- Processor Organization and Architecture (POA)

Course Outcomes:

COs	Course Outcomes	Blooms Level	Blooms Description
CO1	Understand the arithmetic and logic algorithms for processors	L2	Understand
CO2	Understand the concepts of memory organization and mapping techniques.	L2	Understand
CO3	Explain, Interpret and implement the instructions and addressing modes of 8086 microprocessor and write assembly and mixed language programs.	L3,L4	Apply, Analyze
CO4	Understand the architecture and concepts of an 8051 microcontroller.	L2	Understand
CO5	Understand advanced trends and technologies in processor architectures.	L2	Understand

Sub- Processor Organization and Architecture (POA)

Unit No.	Unit Name	No. of Hrs
Unit-I	Introduction to Computer Architecture & Organization	06 Hrs
Unit-II	Memory Organization	08 Hrs
Unit-III	Intel 8086 Architecture and Addressing Modes	06 Hrs.
Unit-IV	8086 Instruction set, Interrupts and Programming	10 Hrs.
Unit-V	8051 Microcontroller	05 Hrs.
Unit-VI	Intel Pentium Processor	06 Hrs.

Sub- Processor Organization and Architecture (POA)

Unit-I Introduction to Computer Architecture & Organization 06 Hrs.

Introduction, Basic Organization of Computer Architecture, Von Neumann Model and Harvard Architecture, Data Representation and Arithmetic Algorithms- Addition, Subtraction, Multiplication - Unsigned Multiplication, Booth's Algorithm (Signed Multiplication), Division of Integers - Restoring Division, Non-Restoring Division.

Unit-II Memory Organization 08Hrs.

Types of RAM (SRAM, DRAM, SDRAM, DDR, SSD) and ROM, Characteristics of Memory, Memory Hierarchy- Cost and Performance Measurement.

Virtual Memory: Concept, Segmentation and Paging.

Address Translation Mechanism, Interleaved and Associative Memory, Cache Memory Concepts, Cache Coherency.

Sub- Processor Organization and Architecture (POA)

Unit-III Intel 8086 Architecture and Addressing Modes 06 Hrs.

Major Features of 8086 Processor, 8086 CPU Architecture and Pipelined Operations, Programmer's Model and 8086 Pin Description, 8086 Addressing Modes.

Unit-IV 8086 Instruction set, Interrupts and Programming 10 Hrs.

Instruction Set of 8086 Microprocessor, Assembler Directives, Procedure and Macros.

Interrupts in 8086 Microprocessor: Dedicated Interrupts, Software Interrupts, DOS Interrupts (Programming Examples), Assembly Language Programming for 8086 Microprocessor, Mixed Mode Programming for 8086.

Sub- Processor Organization and Architecture (POA)

Unit-V 8051 Microcontroller — 05 Hrs.

Architecture of 8051 Microcontroller, Addressing Modes for 8051.

Instruction Set for 8051 Microcontroller: Data Transfer, Arithmetic and Logical.

Interrupts in 8051 Microcontroller.

Unit-VI Intel Pentium Processor — 06 Hrs.

Features of Intel Pentium Processor, Pentium Superscalar Architecture, Pipelining, Branch Prediction, Instruction and Data Cache Concept.

Sub- Processor Organization and Architecture

Reference Books:

1. William Stallings- "Computer Organization and Architecture: Designing for Performance", 11 Pearson Publication, 10th Edition, 2013.
2. John P. Hayes- "Computer Architecture and Organization", McGraw-Hill, 1988.
3. John Uffenbeck – "8086/8088 Family: Design Programming and Interfacing", PHI.
4. Douglas Hall- "Microprocessor and Interfacing", Tata McGraw Hill.
5. M. A. Mazidi, J. C. Mazidi, Rolin D. McKinlay- "The 8051 Microcontroller and Embedded Systems Using Assembly and C", Pearson Education, 2nd Edition.
6. Kenneth J. Ayala- "The 8051 Microcontroller", Cengage Learning India Pvt. Ltd, 3rd Edition.
7. James L. Antonakos- "The Intel Microprocessor Family: Hardware and Software Principles and Applications", Cengage Learning.

Sub- Processor Organization and Architecture

Evaluation Scheme:

Theory :

Continuous Assessment (A):

Subject teacher will declare Teacher Assessment criteria at the start of semester.

Continuous Assessment (B):

1. Two term tests of 15 marks each will be conducted during the semester.
2. Average of the marks scored in both the tests will be considered for final grading.

End Semester Examination (C):

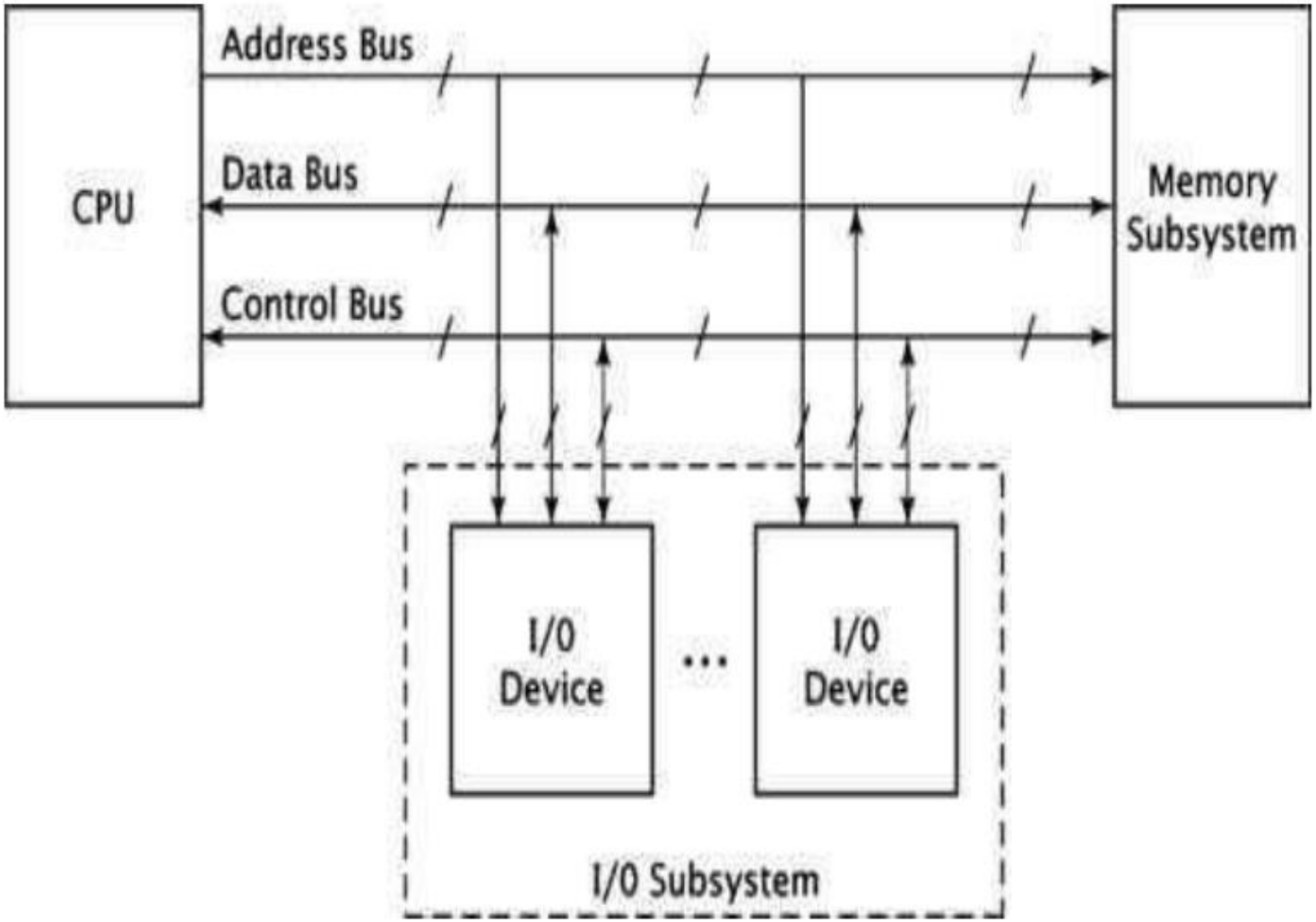
1. Question paper based on the entire syllabus, summing up to 65 marks.
2. Total duration allotted for writing the paper is 3 hrs.

Unit I – Introduction to Computer Architecture & Organization 6 Hrs.

Basic Computer Organization

- The basic computer organization has following main components:
  - CPU
  - Memory subsystem
  - I/O subsystem
  - System buses

Basic Computer Organization



CPU Organization

- Central processing unit(CPU) is the electronic circuitry within a computer.
- It consists of ALU, Registers and Control unit.
- ALU performs the arithmetic and logical operations like addition, subtraction, multiplication, etc.
- Registers are the temporary storage used to store the intermediate data during instruction execution.
- Control unit is used to generate the control signals for various operations.

Memory Subsystem

- The function of the memory unit is to store programs and data.
- There are two classes of storage, called primary and secondary.
- Primary Memory also called main memory, is a fast memory that operates at electronic speeds.
- Programs must be stored in this memory while they are being executed.
- Although primary memory is essential, it tends to be expensive and does not retain information when power is turned off.
- Thus additional, less expensive, permanent secondary storage is used when large amounts of data and many programs have to be stored.

I/O Subsystem

- The I/O subsystem consists of input unit and output unit.
- The input unit accepts coded information from human operators using devices such as keyboards, or from other computers over digital communication lines.
- Output unit function is to send processed results to the outside world.
- A familiar example of such a device is a printer

System bus

- The system bus has three buses,
  - Address bus : The uppermost bus is the **address bus**.
  - When the CPU reads data or instructions from or writes data to memory, it must specify the address of the memory location it wishes to access.

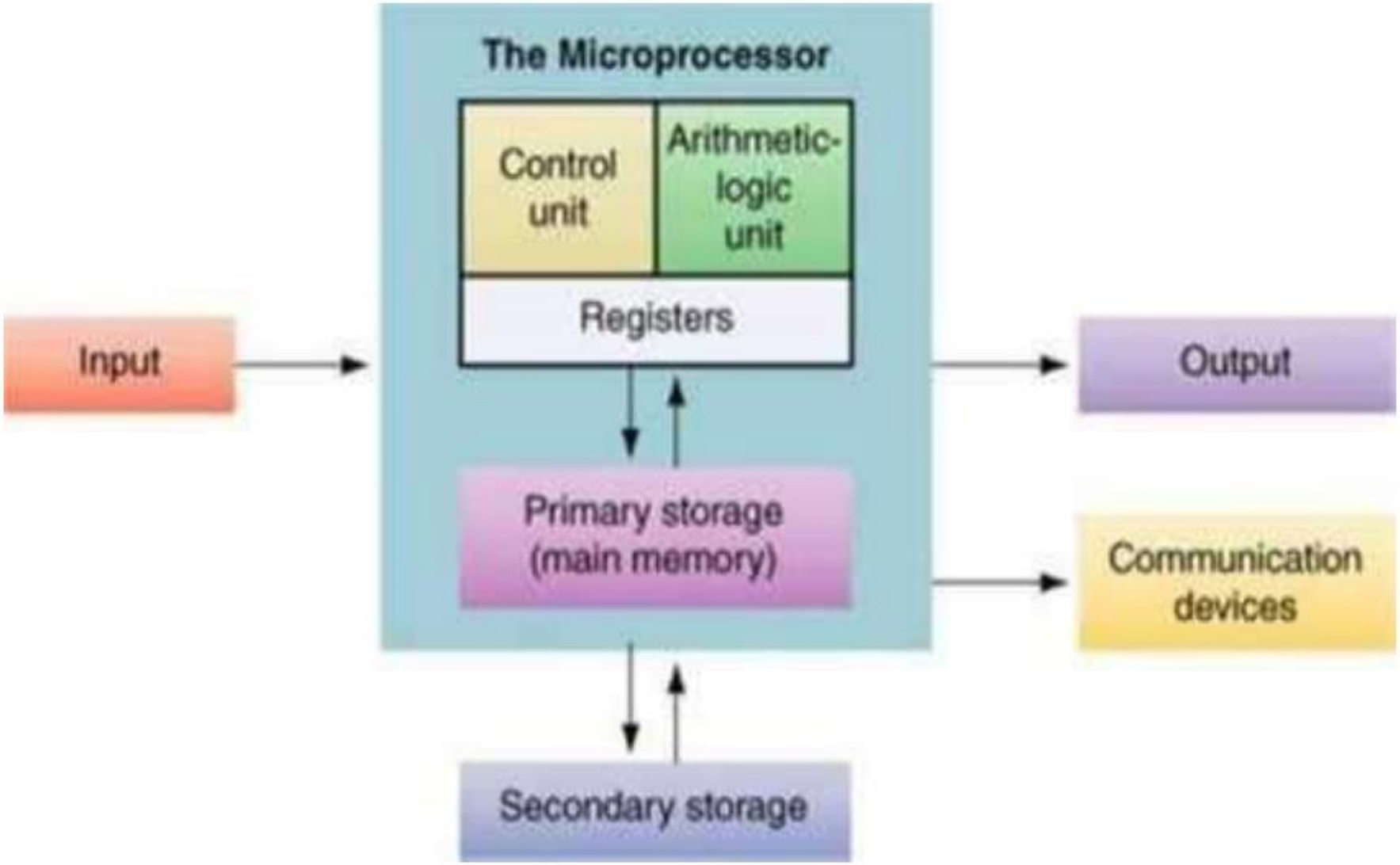
Basic Computer Organization

- Data bus : Data is transferred via the **data bus**.
- When CPU fetches data from memory it first outputs the memory address onto its address bus.
- Then memory outputs the data onto the data bus.
- Memory then reads and stores the data at the proper locations.

Basic Computer Organization

- Control bus : **Control bus** carries the control signal.
- Control signal is the collection of individual control signals.
- These signals indicate whether data is to be read into or written out of the CPU.

CPU Organization



Organization and Architecture

- **Computer architecture** refers to those attributes of a system visible to a programmer or those attributes that have a direct impact on the logical execution of a program.
- It includes instruction sets, number of bits used for data, addressing techniques, etc.
- A term that is often used interchangeably with computer architecture is instruction set architecture (ISA).

Organization and Architecture

- **Computer organization** refers to the operational units and their interconnections that realize the architectural specifications.
- 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.
- For example, it is an architectural design issue whether a computer will have a multiply instruction. It is an organizational issue whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system.

Organization and Architecture

- Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in organization.
- Consequently, the different models in the family have different price and performance characteristics.
- Furthermore, a particular architecture may span many years and encompass a number of different computer models, its organization changing with changing technology.

Computer Architecture	Computer Organization
Computer Architecture is the way hardware and software interact with one another to form a computer system, (what it does)	Computer Organization is related to the structure and behaviour of a computer system (how it does)
Computer Architecture acts as an interface between hardware and software	Computer Organization handles the connection components of the system
Firstly, computer architecture is considered while designing a computer	Computer Organization is built after considering computer architecture
It provides us the functionalities of the computer system	It helps us to understand how all the units in the system are interconnected
All the high-level design issue are handled by the computer architecture	All the low-level design issue are handled by the computer organization
It comprises logical units (instruction set, registers, addressing modes, data types)	It comprises physical units (peripherals, adders, circuit design)
Instruction set architecture also means computer architecture	Microarchitecture also means computer organization

Structure and Function

- A computer is a complex system; present computers contain millions of electronic components.
- The key is to recognize the hierarchical nature of most complex systems, including the computer .
- A hierarchical system is a set of interrelated subsystems, each in turn, hierarchical in structure until we reach some lowest level of elementary subsystem.
- 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.

Structure and Function

- 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:
- ■ Structure: The way in which the components are interrelated.
- ■ Function: The operation of each individual component as part of the structure.

Function

- There are only four basic functions that a computer can perform:
- 1) Data processing: Data may take a wide variety of forms, and the range of processing requirements is broad.
- 2) Data storage:
  - 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.
  - Files of data are stored on the computer for subsequent retrieval and update.

Function

- 3) Data movement:
  - 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.
- 4) Control: A control unit manages the computer's resources and Manages the performance of its functional parts in response to instructions.

Structure

- We now look in a general way at the internal structure of a computer.
- We begin with a traditional computer with a single processor that employs a microprogrammed control unit, then examine a typical multicore structure.
  - ***simple single- processor computer***
  - Figure provides a hierarchical view of the internal structure of a traditional single-processor computer. There are four main structural components:

Structure

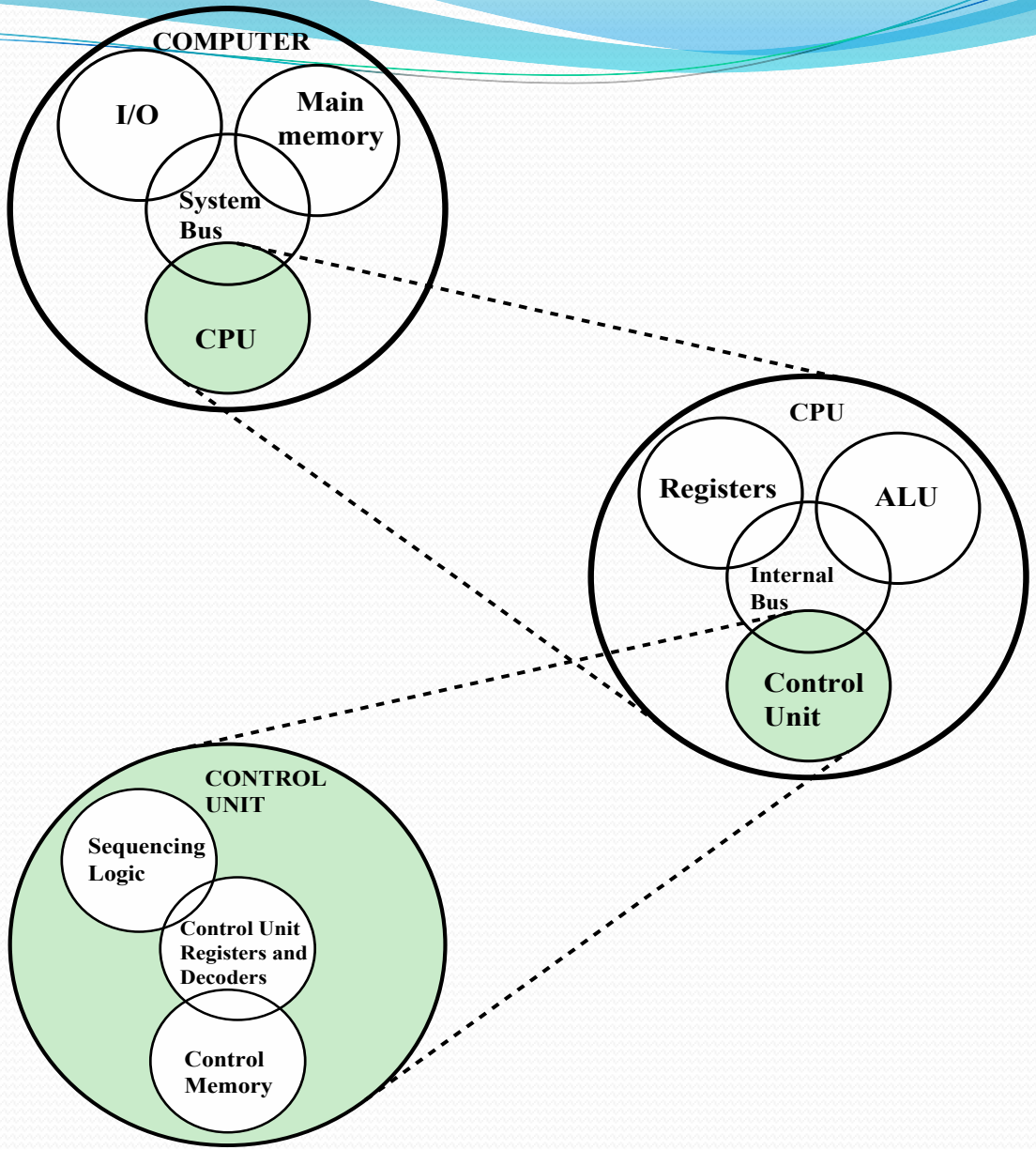


Figure 1.1 A Top-Down View of a Computer

Structure

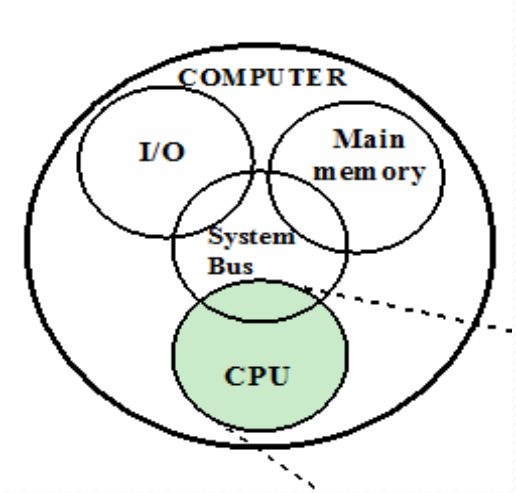
- There are four main structural components:

✦ CPU – controls the operation of the computer and performs its data processing functions.

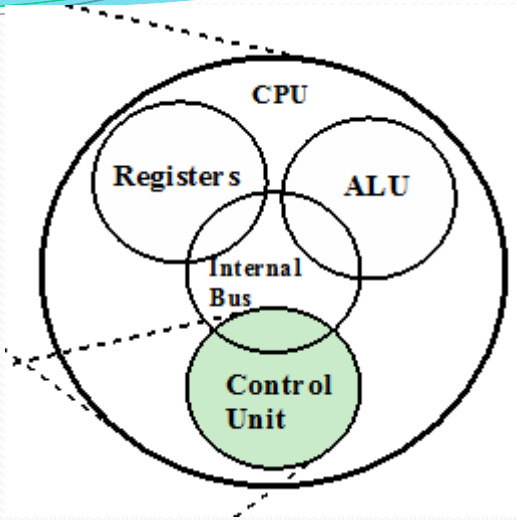
✦ Main Memory – stores data.

✦ I/O – moves data between the computer and its external environment.

✦ System Interconnection – some mechanism that provides for communication among CPU, main memory, and I/O.



CPU : Major structural components:



- Control Unit
  - Controls the operation of the CPU and hence the computer.
- Arithmetic and Logic Unit (ALU)
  - Performs the computer's data processing function.
- Registers
  - Provide storage internal to the CPU.
- CPU Interconnection
  - Some mechanism that provides for communication among the control unit, ALU, and registers.

Multicore Computer Structure

- ***multicore computer structure*** are the computers generally have multiple processors.
- When these processors all reside on a single chip, the term *multicore computer* is used, and each processing unit (consisting of a control unit, ALU, registers, and perhaps cache) is called a *core*.

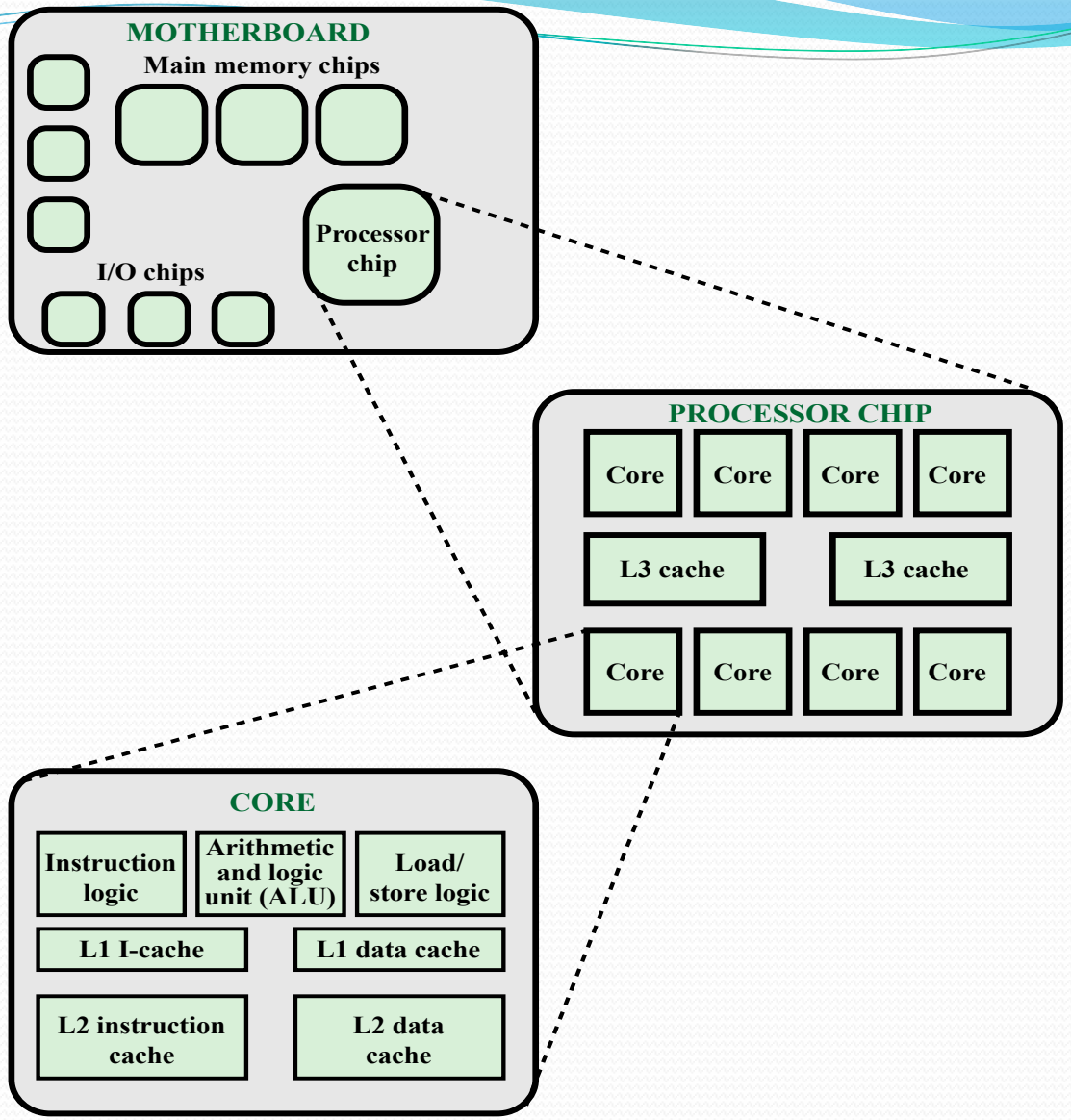


Figure 1.2 Simplified View of Major Elements of a Multicore Computer

Multicore Computer Structure

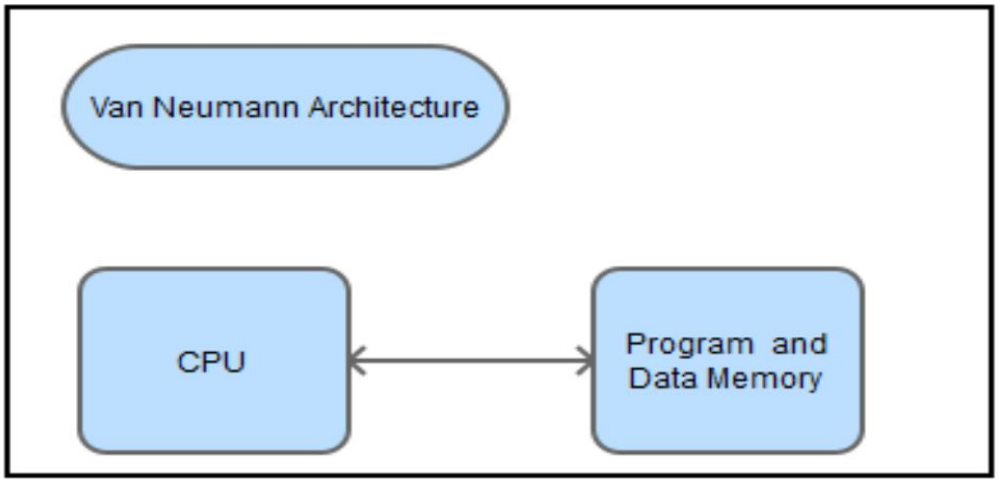
- **Central processing unit (CPU):**
  - That portion of a computer that fetches and executes instructions.
  - It consists of an ALU, a control unit, and registers.
  - In a system with a single processing unit, it is often simply referred to as a *processor*.
- **Core:**
  - An individual processing unit on a processor chip.
  - A core may be equivalent in functionality to a CPU on a single- CPU system.
  - Other specialized processing units, such as one optimized for vector and matrix operations, are also referred to as cores.
- **Processor:**
  - A physical piece of silicon containing one or more cores.
  - The processor is the computer component that interprets and executes instructions.
  - If a processor contains multiple cores, it is referred to as a **multicore processor**.

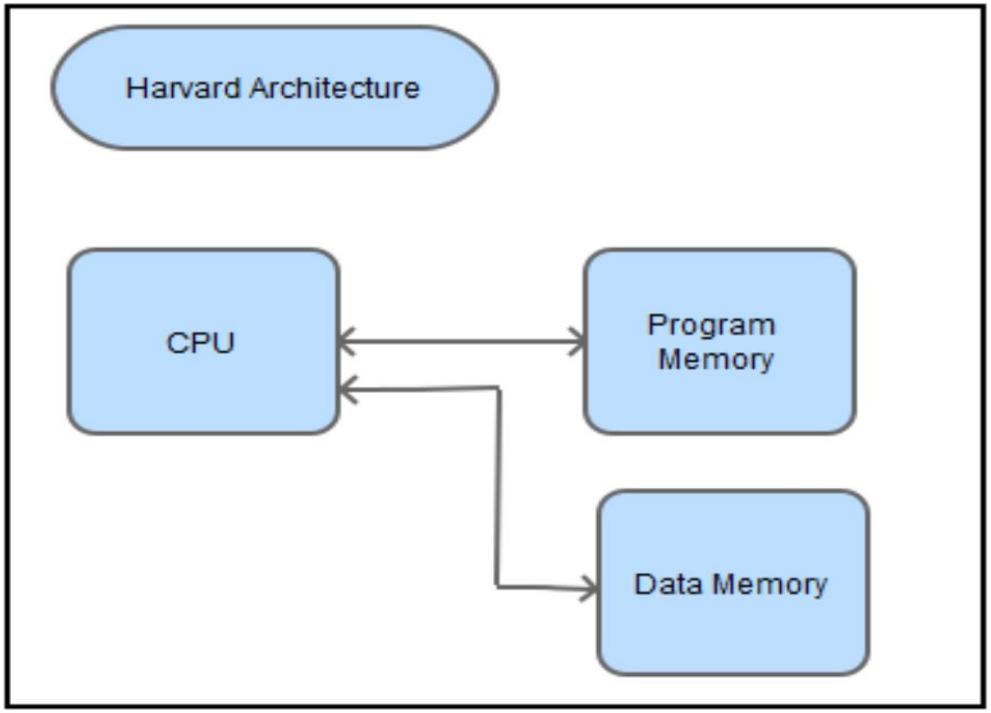
Multicore Computer Structure

- Another prominent feature of present computers is the use of multiple layers of memory, called *cache memory*, between the processor and main memory.
- A cache memory is smaller and faster than main memory and is used to speed up memory access, by placing in the cache data from main memory, that is likely to be used in the near future.
- A greater performance improvement may be obtained by using multiple levels of cache, with level 1 (L1) closest to the core and additional levels (L2, L3, and so on) progressively farther from the core.

Von Neumann Model and Harvard Architecture

- When discussing how memory is accessed at the CPU level, there are two designs to consider.
- The first is a Von Neumann architecture, and the second is a Harvard architecture.
- The major difference between the two architectures is that in a Von Neumann architecture memory is capable of storing all program elements, data and instructions;
- In a Harvard architecture the memory is divided into two memories, one for data and one for instructions.

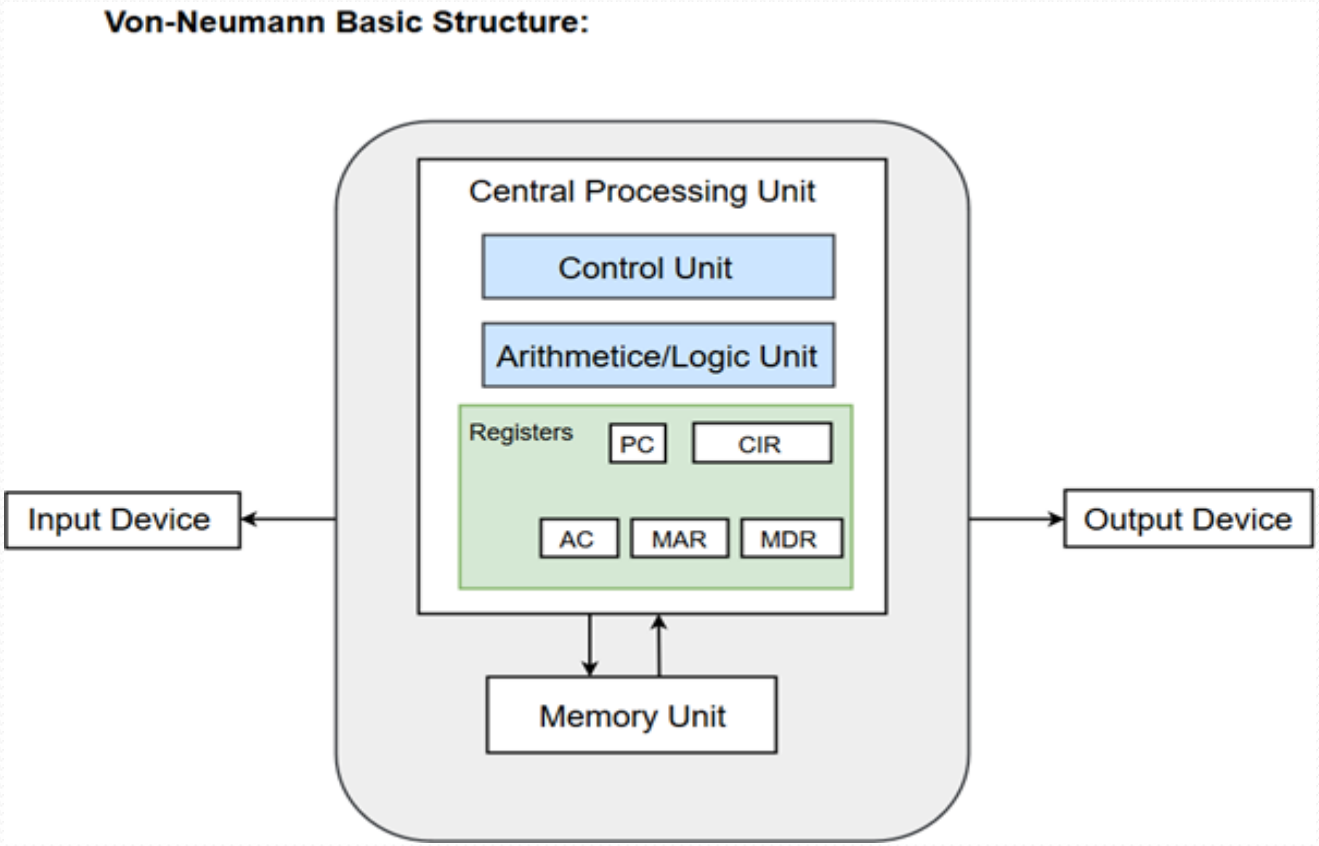

Van Neumann Architecture
CPU
Program and
Data Memory


Harvard Architecture
CPU
Program
Memory
Data Memory

Von Neumann Model

- Von-Neumann proposed his computer architecture design in 1945 which was later known as Von-Neumann Architecture.
- It consisted of a Control Unit, Arithmetic, and Logical Memory Unit (ALU), Registers and Inputs/Outputs.
- Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory.
- This design is still used in most computers produced today.

Von Neumann Model



Von Neumann Model

- Components of Von-Neumann Model:
  - Central Processing Unit
  - Buses
  - Memory Unit

Von Neumann Model

- Central Processing Unit:
  - The part of the Computer that performs the bulk of data processing operations is called the Central Processing Unit and is referred to as the CPU.
  - The Central Processing Unit can also be defined as an electric circuit responsible for executing the instructions of a computer program.
  - The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer.
  - The major components of CPU are Arithmetic and Logic Unit (ALU), Control Unit (CU) and a variety of registers.

Von Neumann Model

- Arithmetic and Logic Unit (ALU):
  - The Arithmetic and Logic Unit (ALU) performs the required micro-operations for executing the instructions. In simple words, ALU allows arithmetic (add, subtract, etc.) and logic (AND, OR, NOT, etc.) operations to be carried out.
- Control Unit:
  - The Control Unit of a computer system controls the operations of components like ALU, memory and input/output devices.
  - The Control Unit consists of a program counter that contains the address of the instructions to be fetched and an instruction register into which instructions are fetched from memory for execution.
- Registers:
  - Registers refer to high-speed storage areas in the CPU. The data processed by the CPU are fetched from the registers.

Von Neumann Model

MAR (Memory Address Register)	This register holds the memory location of the data that needs to be accessed.
MDR (Memory Data Register)	This register holds the data that is being transferred to or from memory.
AC (Accumulator)	This register holds the intermediate arithmetic and logic results.
PC (Program Counter)	This register contains the address of the next instruction to be executed.
CIR (Current Instruction Register)	This register contains the current instruction during processing.

Von Neumann Model

- Buses
  - Buses are the means by which information is shared between the registers in a multiple-register configuration system.
  - A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
  - Control signals determine which register is selected by the bus during each particular register transfer.
  - Von-Neumann Architecture comprised of three major bus systems for data transfer.

Von Neumann Model

Bus	Description
Address Bus	Address Bus carries the address of data (but not the data) between the processor and the memory.
Data Bus	Data Bus carries data between the processor, the memory unit and the input/output devices.
Control Bus	Control Bus carries signals/commands from the CPU.

Von Neumann Model

- Memory Unit:
- A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the storage.
- The memory stores binary information in groups of bits called words.
- The internal structure of a memory unit is specified by the number of words it contains and the number of bits in each word.
- **Two major types of memories are used in computer systems:**
  - RAM (Random Access Memory)
  - ROM (Read-Only Memory)

Von Neumann Model Limitations

- A program is stored in the computer's memory along with the data to be processed
- This can lead to a condition called the von Neumann bottleneck, it places a limitation on how fast the processor can run.
- Instructions and data must share the same path to the CPU from memory, so if the CPU is writing a data value out to memory, it cannot fetch the next instruction to be executed.
- It must wait until the data has been written before proceeding and vice versa.

Expanded Structure of IAS Computer

Memory buffer register (MBR): Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.

- **Memory address register (MAR):** It contains the address of a location of main memory from where information has to be fetched or information has to be stored.

- **Instruction register (IR):** Contains the 8-bit opcode instruction being executed.

- **Instruction buffer register (IBR):** Employed to hold temporarily the right-hand instruction from a word in memory.

- **Program counter (PC):** Contains the address of the next instruction-pair to be fetched from memory.

- **Accumulator (AC) and multiplier quotient (MQ):** Employed to hold temporarily operands and results of ALU operations.

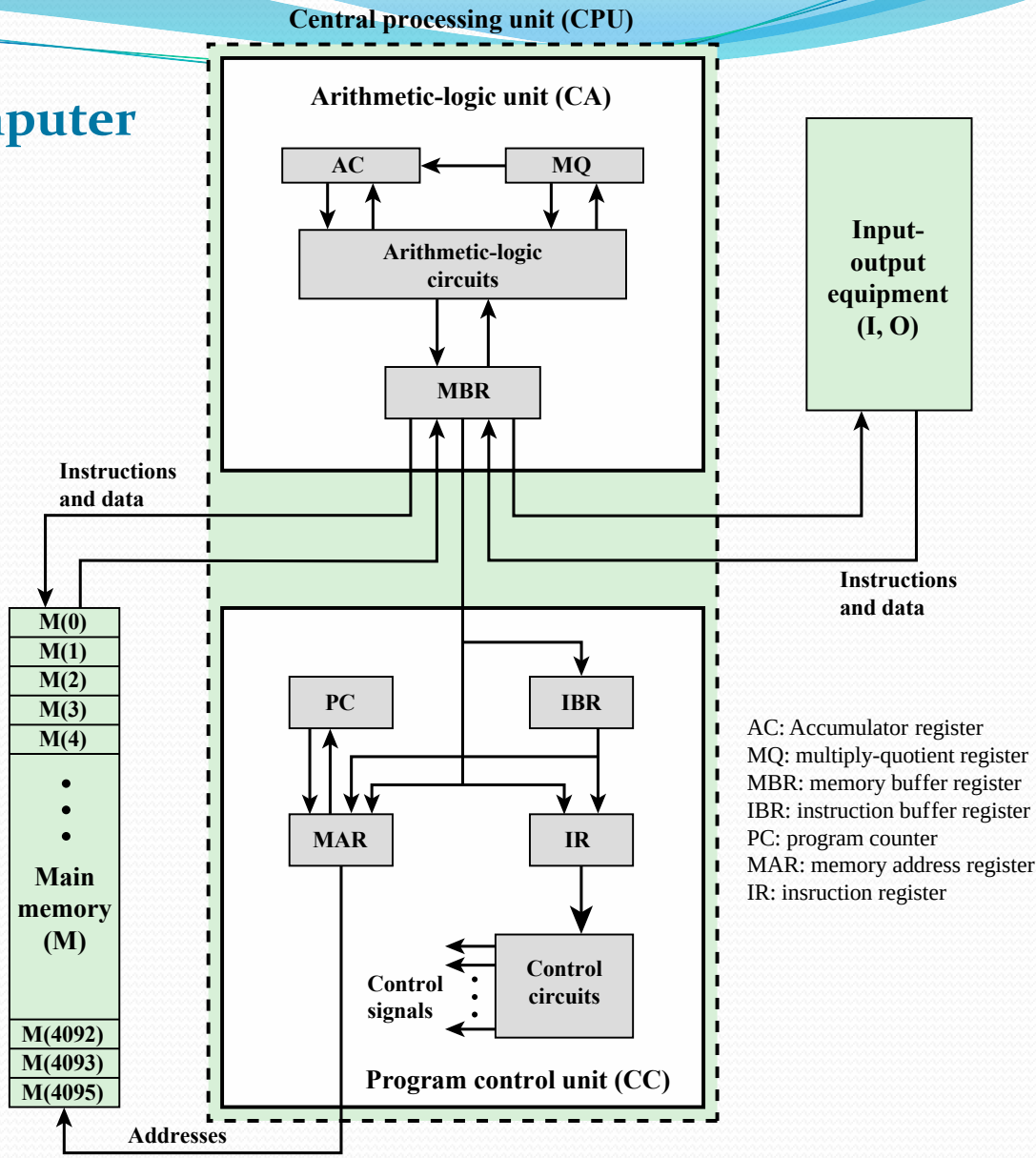
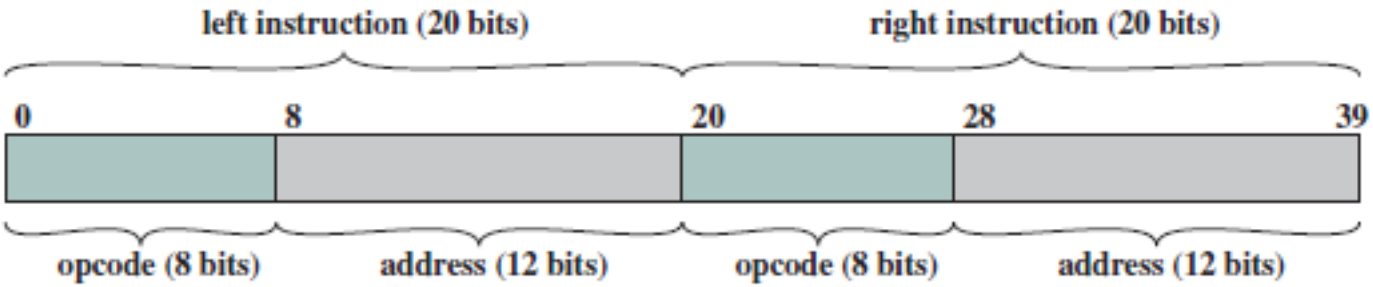


Figure 1.6 IAS Structure

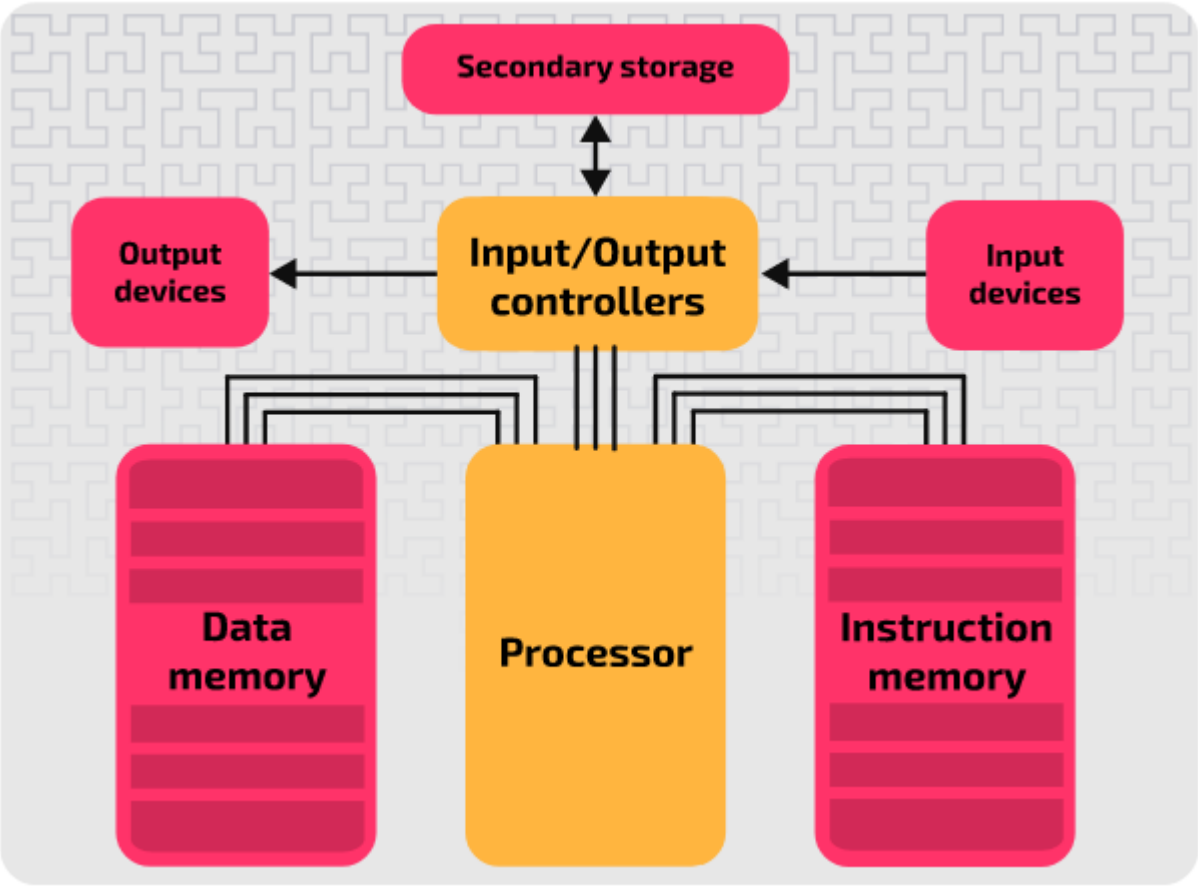
IAS Instruction Word



Harvard Architecture

- **Harvard Architecture** is the computer architecture that contains separate storage and separate buses (signal path) for instruction and data.
- It was basically developed to overcome the bottleneck of Von Neumann Architecture.
- The main advantage of having separate buses for instruction and data is that the CPU can access instructions and read/write data at the same time.
- The processor is connected to the 'instructions memory' using a dedicated set of address and data buses, and is connected to the 'data memory' using a **different** set of address and data buses.
- The Harvard architecture is used extensively in embedded systems, for example in digital signal processing (DSP) systems.

Harvard Architecture

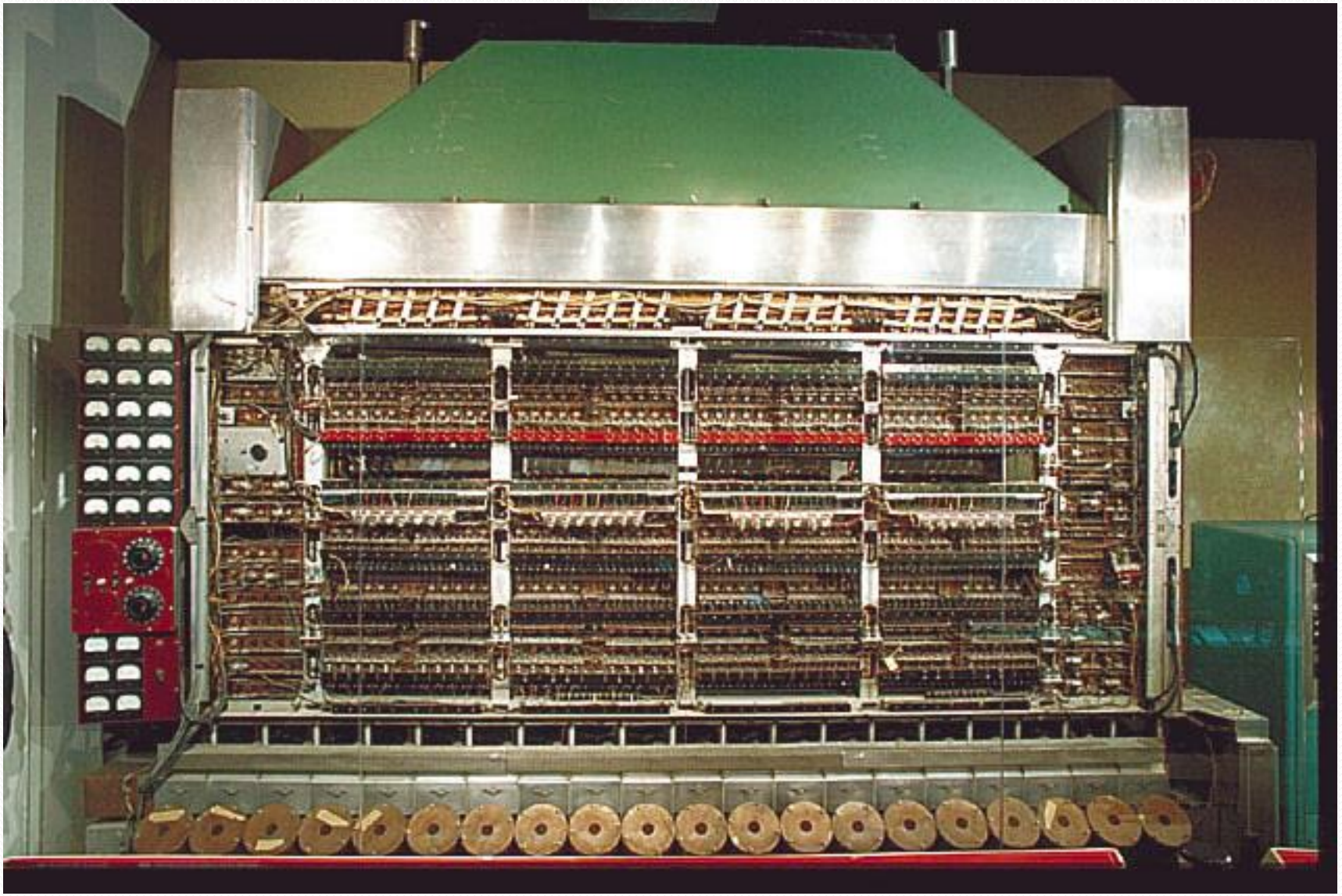


Harvard Architecture

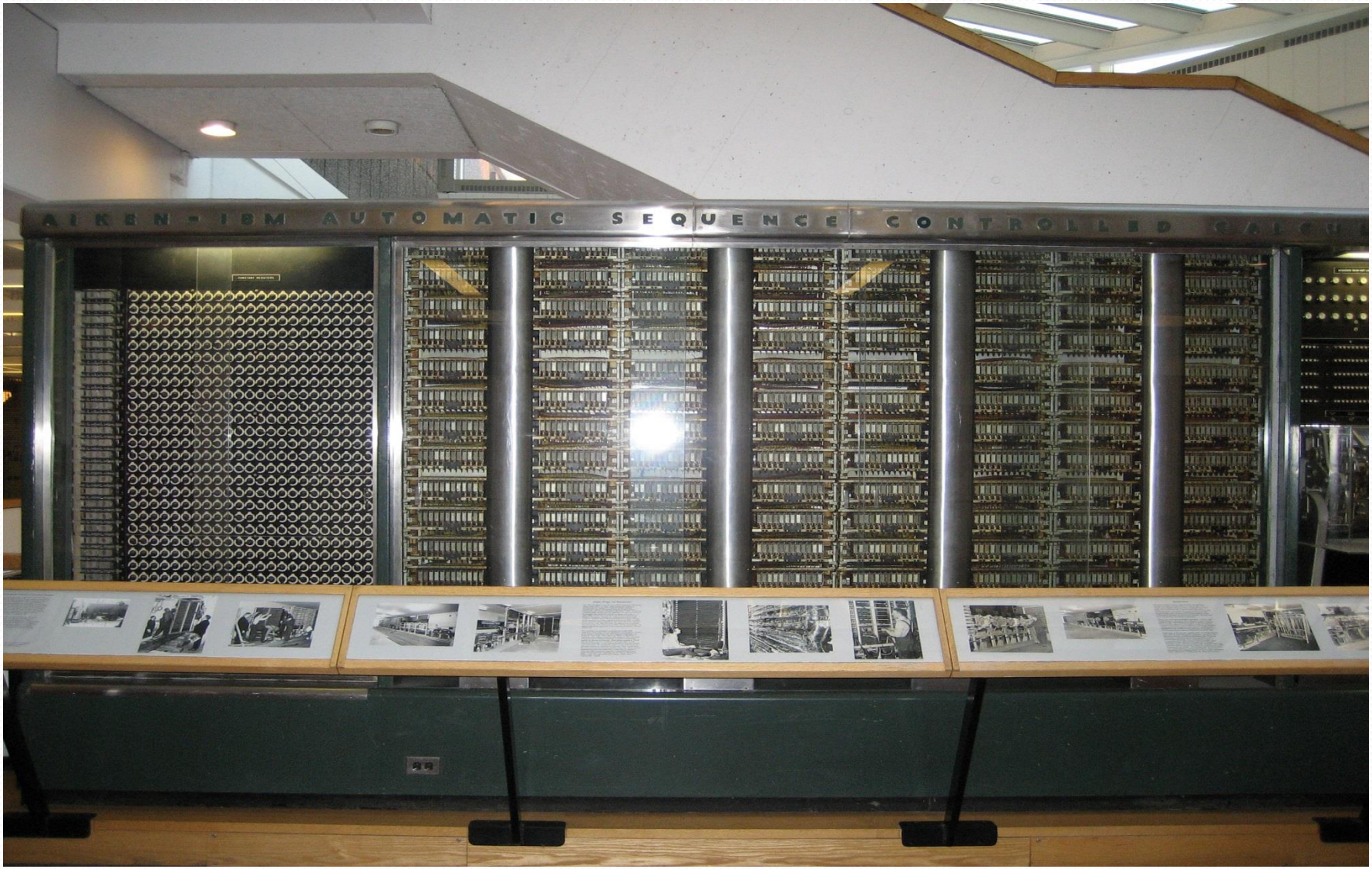
- Buses :
- Buses are used as signal pathways.
- In Harvard architecture, there are separate buses for both instruction and data.
- Types of Buses:
  - **Data Bus:** It carries data among the main memory system, processor, and I/O devices.
  - **Data Address Bus**: It carries the address of data from the processor to the main memory system.
  - **Instruction Bus**: It carries instructions among the main memory system, processor, and I/O devices.
  - **Instruction Address Bus**: It carries the address of instructions from the processor to the main memory system.

Von-Neumann Architecture	Harvard Architecture
The Von Neumann Architecture is an ancient type of computer architecture that follows the concept of a stored-program computer.	Harvard Architecture is a modern type of computer architecture that follows the concept of the relay-based model by Harvard Mark I.
Single memory to be shared by both code and data.	Separate memories for code and data.
Processor needs to fetch code in a separate clock cycle and data in another clock cycle. So it requires two clock cycles.	Single clock cycle is sufficient, as separate buses are used to access code and data.
The speed of execution of the Von Neumann Architecture is comparatively slower.	The overall speed of execution of Harvard Architecture is comparatively faster.
Simple in design.	Complex in design.
This method comes to play in the case of small computers and personal computers.	This architecture is best for signal processing as well as microcontrollers.
This architecture basically requires less space.	This architecture comparatively requires more space.
It is comparatively cheaper in cost than Harvard Architecture.	It is comparatively more expensive than the Von Neumann Architecture.

Von Neumann/ IAS Computer



Harvard Computer



Data Representation

- In the binary number system arbitrary numbers can be represented with:
  - The digits zero and one
  - The minus sign (for negative numbers)
  - The period, or ***radix point*** (for numbers with a fractional component)
  - Eg: -1101.0101_2
- For purposes of computer storage and processing we do not have the benefit of special symbols for the minus sign and radix point
- Only binary digits (0,1) may be used to represent numbers

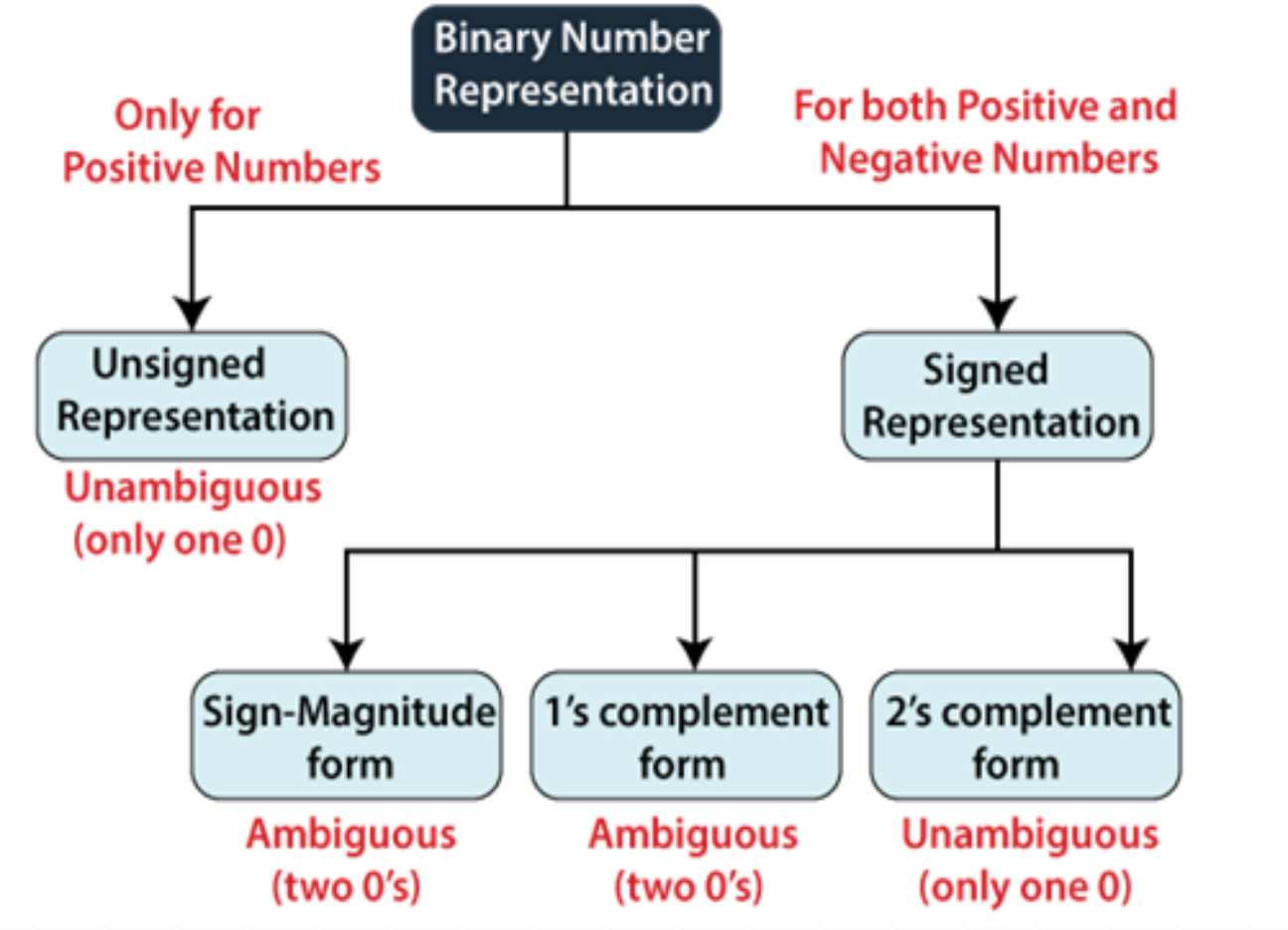
Representation of Binary Numbers:

- The binary numbers are represented in both ways, i.e.,
  - signed and
  - unsigned
- The positive numbers are represented in both ways- signed and unsigned, but the negative numbers can only be described in a signed way.
- The difference between unsigned and signed numbers is that unsigned numbers do not use any sign bit for positive and negative numbers identification, but the signed number used.

Representation of Binary Numbers:

Decimal	Unsigned Only positive	Signed Positive	Signed Negative
0	000	0000	1000
1	001	0001	1001
2	010	0010	1010
3	011	0011	1011
4	100	0100	1100
5	101	0101	1101
6	110	0110	1110
7	111	0111	1111

Representation of Binary Numbers:



1) Unsigned Numbers

- As we already know, the unsigned numbers don't have any sign for representing negative numbers.
- So the unsigned numbers are always positive.
- By default, the decimal number representation is positive.
- We always assume a positive sign in front of each decimal digit.
- There is no sign bit in unsigned binary numbers so it can only represent its magnitude.
- zero is also an unsigned binary number, there is only one zero (0) in this representation, which is always positive.

1) Unsigned Numbers

- Because of one unique binary equivalent form of a number in unsigned number representation, it is known as unambiguous representation technique.
- The range of the unsigned binary numbers starts from 0 to (2^n-1).

Decimal	Operation	Result	Remainder
12	12/2	6	0
6	6/2	3	0
3	3/2	1	1
1	1/2	0	1

- So the binary number of $(12)_{10}$ is $(1100)_2$, of 4 bit magnitude.
- The range of 4 bit number is $(2^4 -1) = 15$

  i.e. 0000 to 1111 (0 to 15)

2) Signed Numbers

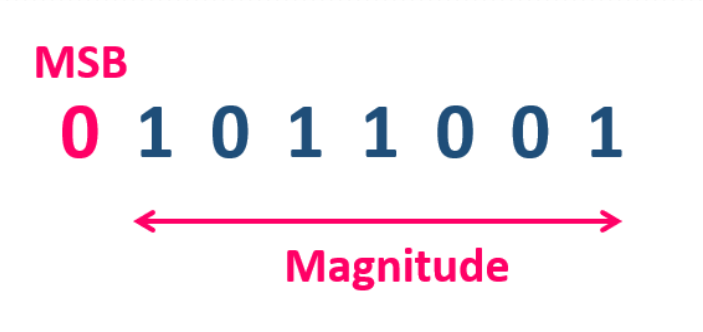
- The signed numbers have a sign bit so that it can differentiate positive and negative integer numbers.
- The signed binary number technique has both the sign bit and the magnitude of the number.
- For representing the negative decimal number, the corresponding symbol in front of the binary number will be added.
- The signed numbers are represented in three ways.
  - **Sign-Magnitude form**
  - **1's complement form**
  - **2's complement form**

2) Signed Numbers

- In Sign magnitude and 1's Complement form:
  - The signed bit makes two possible representations of zero (positive (0) and negative (1)), which is an ambiguous representation.
- While in 2's complement representation in which no double representation of zero is possible, which makes it unambiguous representation.

a) Sign-Magnitude form:

- For n bit binary number, 1 bit is reserved for sign symbol.
- If the value of the sign bit is 0, then the given number will be positive, else if the value of the sign bit is 1, then the given number will be negative.
- The Remaining (n-1) bits represent the magnitude of the number.



a) Sign-Magnitude form:

- Since the magnitude of number zero (0) is always 0, so there can be two representations of number zero (0), positive (+0), and negative (-0), which depends on the value of sign bit.
- Hence these representations are ambiguous generally because of two representations of number zero (0).

+18	= 00010010	
−18	= 10010010	(sign magnitude)

$+0_{10}$	= 00000000	
-0_{10}	= 10000000	(sign magnitude)

a) **Sign-Magnitude form:**

- Another drawbacks of sign-magnitude representation is that addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation.
- Because of these drawbacks, sign-magnitude representation is rarely used in implementing the integer portion of the ALU.
- Instead, the most common scheme is twos complement representation

(b) 1's complement form:

- Since 1's complement of a number is obtained by inverting each bit of a given number.
- So, we represent positive numbers in binary form and negative numbers in 1's complement form.
- There is an extra bit for sign representation.
- If the value of the sign bit is 0, then the number is positive and you can directly represent it in simple binary form.
- but if the value of the sign bit 1, the number is negative and you have to take 1's complement of given binary number.
  - -5 is represented using the following steps:
  - (i) +5 = 0 0101
  - (ii) Take 1's complement of 0 0101 and that is 1 1010.
  - MSB is 1 which indicates that number is negative.

(b) 1's complement form:

- You can get the negative number by 1's complement of a positive number and positive number by using 1's complement of a negative number.
- Therefore, in this representation, zero (0) can have two representations, that's why 1's complement form is also ambiguous.
- And zero (0) has two representations,

+0 = 0 00000

-0 = 1 11111

(b) 2's complement form:

- By inverting each bit of a number and adding plus 1 to its least significant bit, we can obtain the 2's complement of a number.
- The negative numbers can also be represented in the form of 2's complement.
- In this form, the binary number also has an extra bit for sign representation as a sign-magnitude form.
- Therefore, in this representation, zero (0) can have only 1 representations, that's why 2's complement form is unambiguous.
- E.g.:

+0 = 0 00000

1 11111 **take 1's complement**

+ 1 **add 1**

---

1 0 00000

ignore

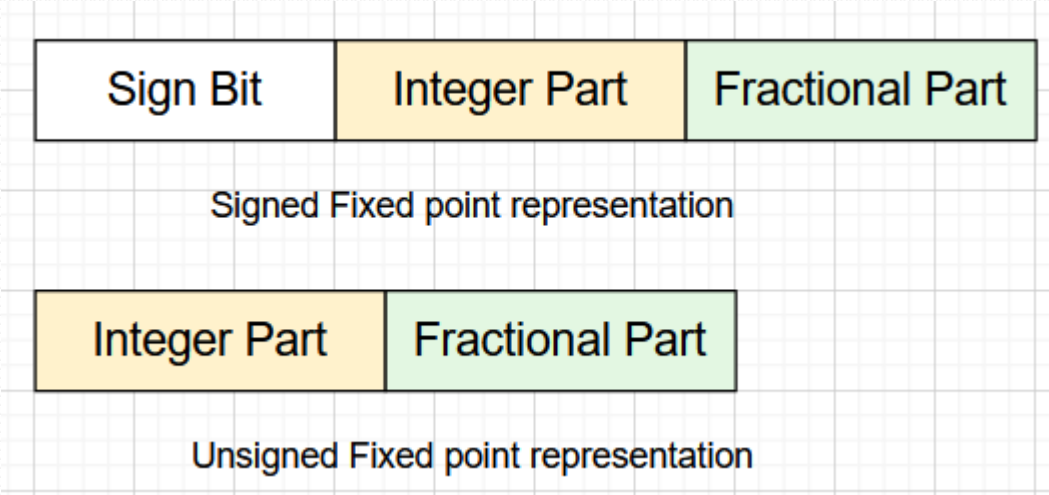
Storing Real Number

- There are two major approaches to represent real numbers (i.e., numbers with fractional component) in modern computing.
- These are
  - (i) Fixed-Point Representation and
  - (ii) Floating Point Representation.
- In fixed-point Representation, there are a fixed number of digits after the decimal point.
- In floating-point Representation allows for a varying number of digits after the decimal point.

a) Fixed Point Representation

- Fixed point representation can convert data into binary form, and then the data is processed, stored, and used by the computer.
- It has a fixed number of bits for the integral and fractional parts.
- There are three parts of the fixed-point number representation:
  - **Sign bit**,
  - **Integral part,** and
  - **Fractional part.**

a) Fixed Point Representation



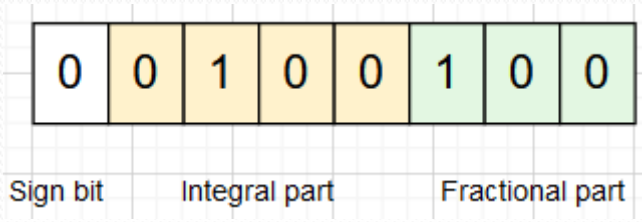
- **Sign bit:-** The fixed-point number representation in binary uses a sign bit. The negative number has a sign bit 1, while a positive number has a bit 0.
- **Integral Part:-** The integral part in fixed-point numbers is of different lengths at different places. It depends on the register's size; for an 8-bit register, the integral part is 4 bits.
- **Fractional part:-** The Fractional part is of different lengths at different places. It depends on the registers; for an 8-bit register, the fractional part is 3 bits.

a) Fixed Point Representation

- Size of Sign Bit, Integer Part, and Fractional Part for different registers are displayed below:

Register	Sign Bit	Integer Part	Fraction Part
8-bit register	1 bit	4 bits	3 bits
16-bit register	1 bit	9 bits	6 bits
32-bit register	1 bit	15 bits	16 bits

- The number **considered is 4.5**
- **Step 1:** We will convert the number 4.5 to binary form. **4.5 = 100.1**
- **Step 2:** Represent the binary number in fixed-point notation with the following format.



b) Floating Point Representation

- The floating-point number representation has three parts:
  - 1. Mantissa
  - 2. Base
  - 3. Exponent (characteristic)

Number	Mantissa	Base	Exponent
3×10^6	3	10	6
110×2^8	110	2	8
6132.784	0.6132784	10	4
34.58	0.3458	10	2

b) Floating Point Representation

- A floating-point number is typically expressed in the scientific notation, with a *fraction* (f), and an *exponent* (e) of a certain *radix* (r), in the form of $\mathbf{f \times r^e}$.
- Decimal numbers use radix of **10 ($\mathbf{f \times 10^e}$);** while binary numbers use radix of **2 ($\mathbf{f \times 2^e}$).**
- Representation of floating point number is not unique.
- For example, the number **55.66** can be represented as $\mathbf{5.566 \times 10^1}$**,** $\mathbf{0.5566 \times 10^2}$**,** $\mathbf{0.05566 \times 10^3}$, and so on.
- The fractional part can be *normalized*. In the normalized form, there is only a single non-zero digit before the radix point.
- For example, decimal number **123.4567** can be normalized as $\mathbf{1.234567 \times 10^2}$; binary number **1010.1011** can be normalized as $\mathbf{1.0101011 \times 2^3}$**.**

b) Floating Point Representation

- 1.0101011×2^3
- Exponent =3
- Add 127 to Exponent that is (3+127)=130 in binary form = 10000010

Number = 1.0101011

Exponent = 10000010

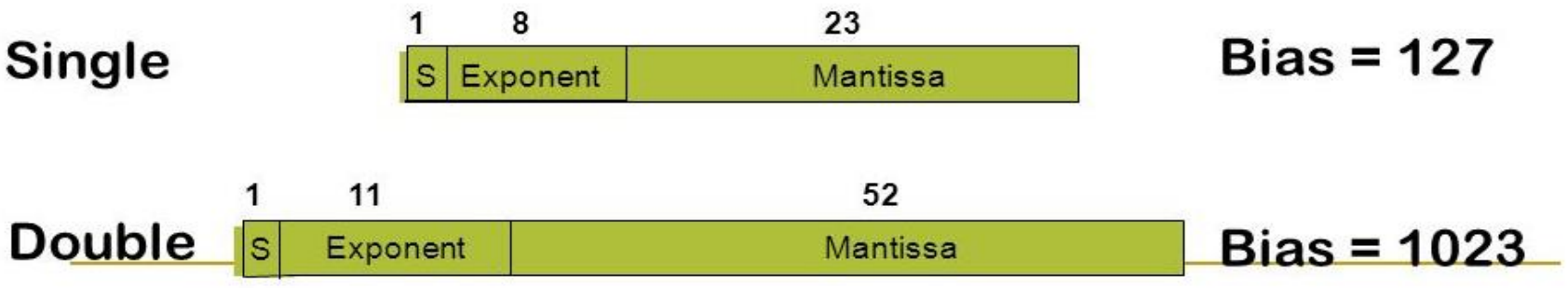
In single precision format:

Sign = 0 (as number is positive)

Exponent = 10000010

Mantissa = 0101011 (preceding one is omitted)

0	1	0	0	0	0	0	1	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Character Representation

- The most common encoding scheme for characters is ASCII (American Standard Code for Information Interchange).
- Alphanumeric characters, operators, punctuation symbols, and control characters are represented by 7-bit codes.
- It is convenient to use an 8-bit to represent and store a character.
- The code occupies the low-order seven bits.
- The high-order bit is usually set to 0.

Character Representation

ASCII	Char	Hex	Bin	ASCII	Char	Hex	Bin
65	A	41	0100 0001	97	a	61	0110 0001
66	B	42	0100 0010	98	b	62	0110 0010
67	C	43	0100 0011	99	c	63	0110 0011
68	D	44	0100 0100	100	d	64	0110 0100
69	E	45	0100 0101	101	e	65	0110 0101
70	F	46	0100 0110	102	f	66	0110 0110
71	G	47	0100 0111	103	g	67	0110 0111
72	H	48	0100 1000	104	h	68	0110 1000
73	I	49	0100 1001	105	i	69	0110 1001
74	J	4A	0100 1010	106	j	6A	0110 1010
75	K	4B	0100 1011	107	k	6B	0110 1011
76	L	4C	0100 1100	108	l	6C	0110 1100
77	M	4D	0100 1101	109	m	6D	0110 1101
78	N	4E	0100 1110	110	n	6E	0110 1110
79	O	4F	0100 1111	111	o	6F	0110 1111
80	P	50	0101 0000	112	p	70	0111 0000
81	Q	51	0101 0001	113	q	71	0111 0001
82	R	52	0101 0010	114	r	72	0111 0010

Arithmetic Algorithms: Addition and Subtraction

- There are three ways of representing negative fixed-point binary numbers:
  - 1. Signed-magnitude
  - 2. Signed-1's complement
  - 3. Signed-2's complement
- Most computers use the signed-2's complement representation when performing arithmetic operations with integers.
- In this section we develop the addition and subtraction algorithms for data represented in signed- magnitude and again for data represented in signed-2's complement.

Addition and Subtraction with Signed-Magnitude Data

- Consider two numbers having magnitude A and B.
- When the signed numbers are added or subtracted, there can be 8 different conditions depending on the sign and the operation performed.
- These conditions are listed in the first column of Table.
- The other columns in the table show the actual operation to be performed with the magnitude of the numbers.
- The last column is needed to prevent a negative zero.
- In other words, when two equal numbers are subtracted, the result should be +0 not -0.

Addition and Subtraction with Signed-Magnitude Data

Operation	Add Magnitudes	Subtract Magnitudes		
		A>B	A<B	A=B
(+ A) + (+ B)	+ (A + B)			
(+ A) + (- B)		+ (A - B)	- (B - A)	+ (A - B)
(- A) + (+ B)		- (A - B)	+ (B - A)	+ (A - B)
(- A) + (- B)	- (A + B)			
(+ A) - (+ B)		+ (A - B)	- (B - A)	+ (A - B)
(+ A) - (- B)	+ (A + B)			
(- A) - (+ B)	- (A + B)			
(- A) - (- B)		- (A - B)	+ (B - A)	+ (A - B)

Addition Algorithm:

- **Addition Algorithm**

- When the sign of A and B are **identical**, add the magnitudes and attach the sign of A to the result.

- When the signs of A and B are **different**, compare the magnitudes and subtract the smaller number from the larger.
  - Choose the sign of result to be same as A if A>B
  - or the complement of sign of A if A<B
  - if A=B subtract B from A and make the sign of result positive

(+5) + (+3) = +8

(+5) + (-3) = 2

(-5) + (+3) = -2

(-5) + (-3) = -8

Operation	Add Magnitudes	Subtract Magnitudes		
		A>B	A<B	A=B
(+ A) + (+ B)	+ (A + B)			
(+ A) + (- B)		+ (A - B)	- (B - A)	+ (A - B)
(- A) + (+ B)		- (A - B)	+ (B - A)	+ (A - B)
(- A) + (- B)	- (A + B)			

Subtraction Algorithm:

- **Subtraction Algorithm**
  - When the sign of A and B are **different** , add the magnitudes and attach the sign of A to the result.
  - When the signs of A and B are **identical**, compare the magnitudes and subtract the smaller number from the larger.
    - Choose the sign of result to be same as A if A>B
    - or the complement of sign of A if A<B
    - if A=B subtract B from A and make the sign of result positive

(+5) - (+3) = +2

(+5) - (-3) = +8

(-5) - (+3) = -8

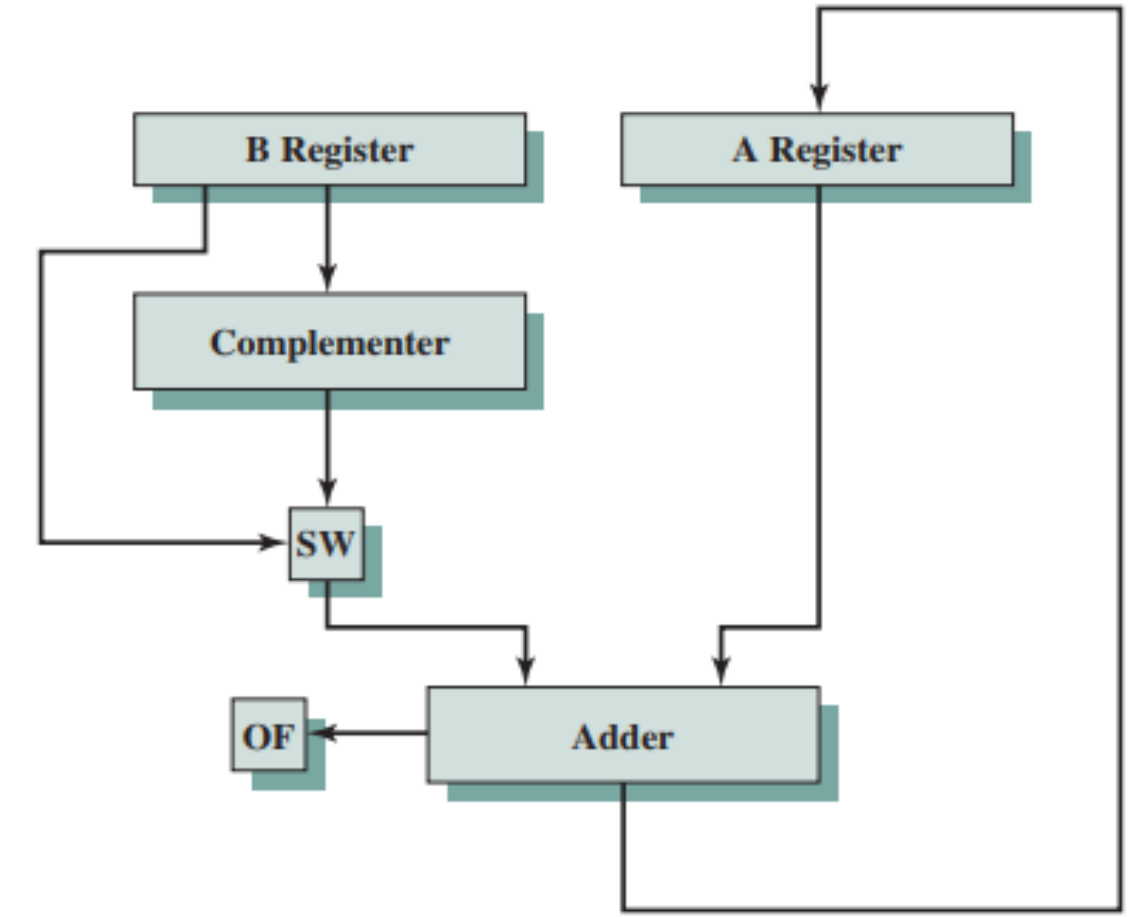
(-5) - (-3) = -2

Operation	Add Magnitudes	Subtract Magnitudes		
		A>B	A<B	A=B
(+ A) - (+ B)		+ (A - B)	- (B - A)	+ (A - B)
(+ A) - (- B)	+ (A + B)			
(- A) - (+ B)	- (A + B)			
(- A) - (- B)		- (A - B)	+ (B - A)	+ (A - B)

Hardware Implementation for Addition and Subtraction

- Figure shows the data paths and hardware elements needed to accomplish addition and subtraction.
- The central element is a binary adder, which is presented with two numbers for addition and produces a sum and an overflow indication.
- The binary adder treats the two numbers as unsigned integers.
- For addition, the two numbers are presented to the adder from two registers, as A and B registers.
- The result may be stored in one of these registers or in a third.
- The overflow indication is stored in a 1-bit overflow flag (0 = no overflow; 1 = overflow).
- For subtraction, the subtrahend (B register) is passed through a twos complementor so that its twos complement is presented to the adder.

Hardware Implementation for Addition and Subtraction

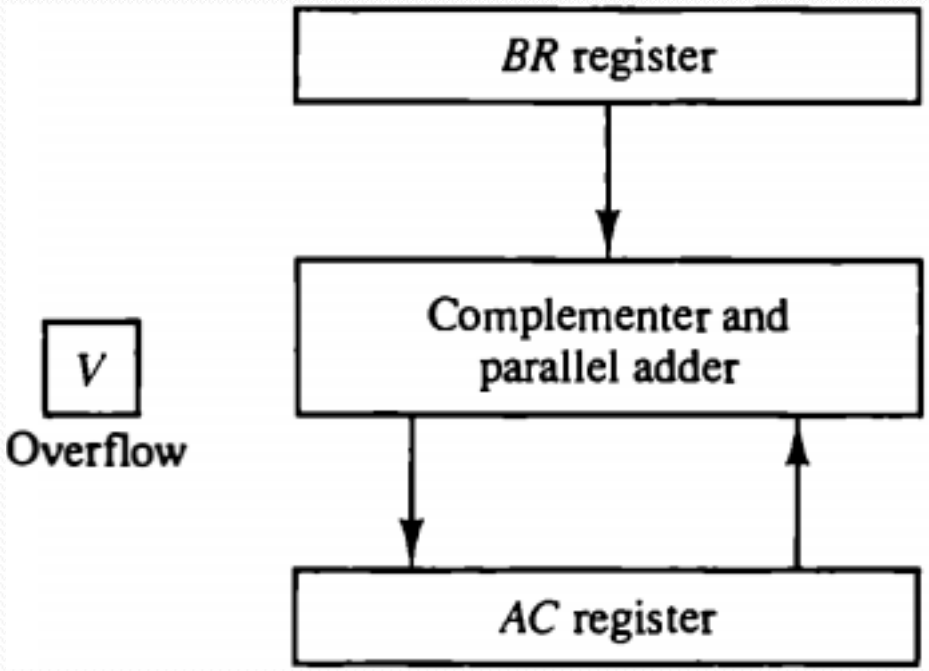


OF = Overflow bit
SW = Switch (select addition or subtraction)

Addition and Subtraction with Signed-2's Complement Data

- The addition of two numbers in signed-2's complement form consists of adding the numbers with the sign bits treated the same as the other bits of the number.
- A carry-out of the sign-bit position is discarded.
- The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend.
- When two numbers of *n* digits each are added and the sum occupies n + 1 digits, we say that an overflow occurred.
- The register configuration for the hardware implementation is shown in Figure.

Addition and Subtraction with Signed-2's Complement Data



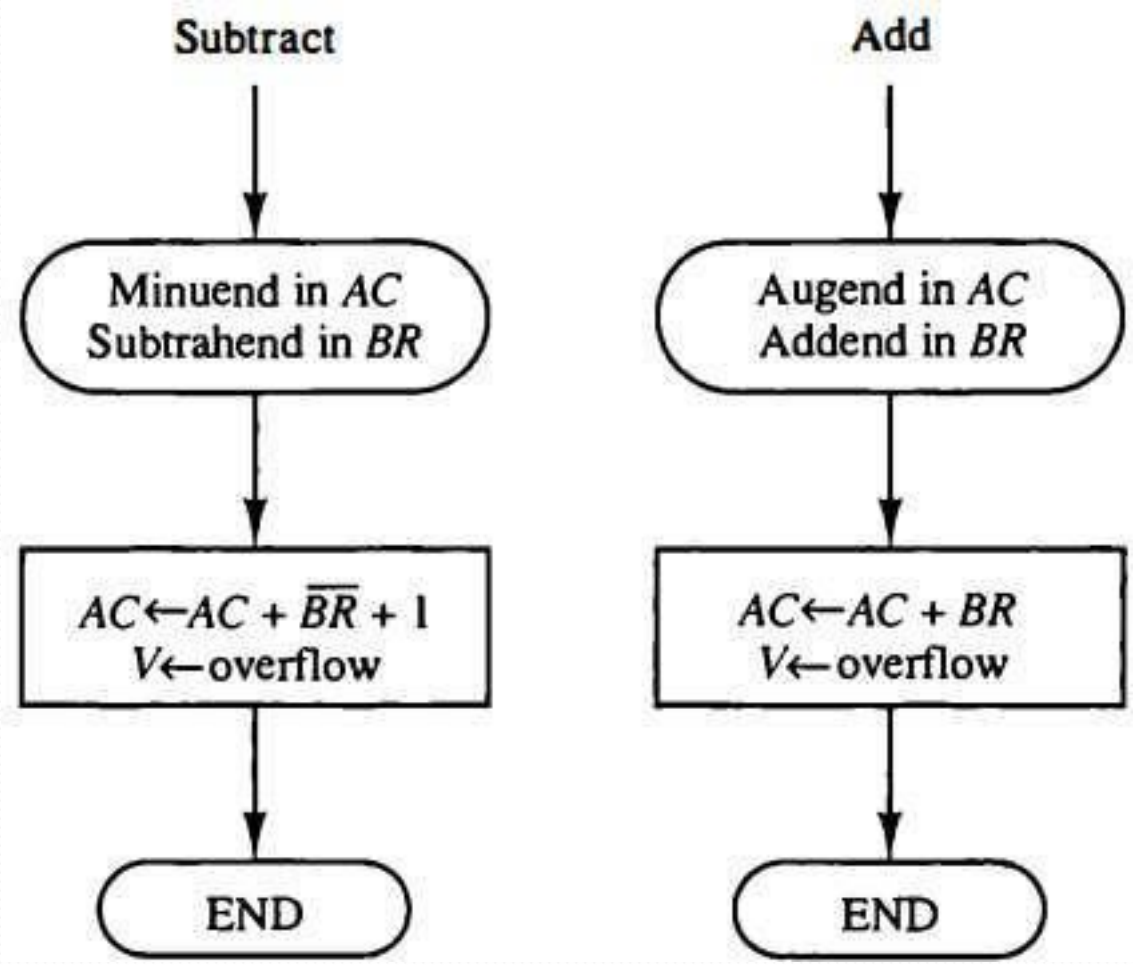
- The sign bits are not separated from the rest of the registers.
- We name the A register AC and the B register BR.
- The leftmost bit in AC and BR represent the sign bits of the numbers.
- The overflow flip-flop V is set to 1 if there is an overflow.

Addition and Subtraction with Signed-2's Complement Data

- The algorithm for adding and subtracting two binary numbers in signed-2's complement representation is shown in the flowchart of Figure.
- The sum is obtained by adding the contents of AC and BR (including their sign bits).
- The overflow bit V is set to 1 if carry is generated and it is ignored.
- The subtraction operation is accomplished by adding the content of AC to the 2's complement of BR.
  - If carry is generated then result is positive and ignore the carry.
  - If carry is not generated then result is negative that is in 2's complement form. Then take 2's complement of result.
  - E.g: 1) 35 (00100011) – 33 (00100001)

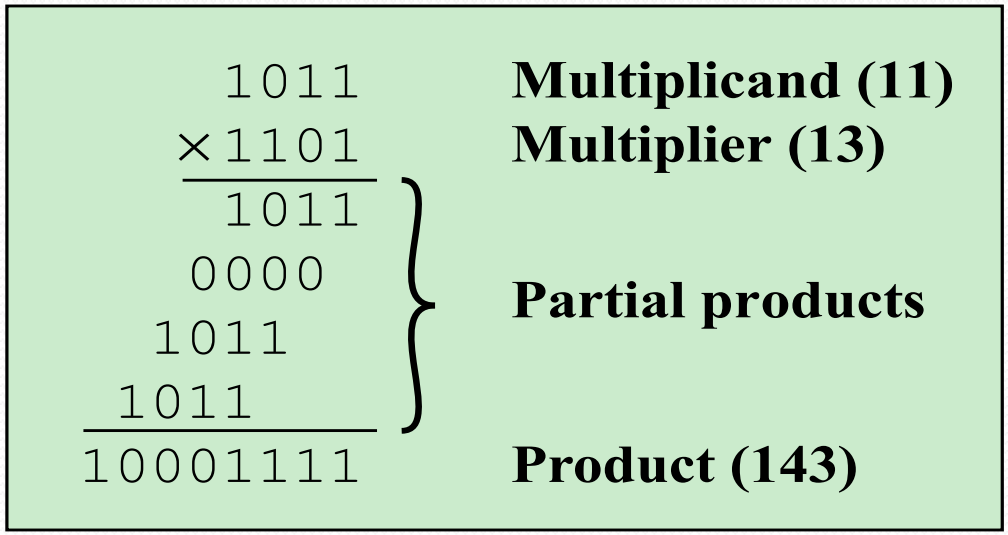
    2) 33 (00100001) – 35 (00100011)

Addition and Subtraction with Signed-2's Complement Data



Un Signed Multiplication

- Figure 10.7 illustrates the multiplication of unsigned binary integers, as might be carried out using paper and pencil.
- Several important observations can be made:



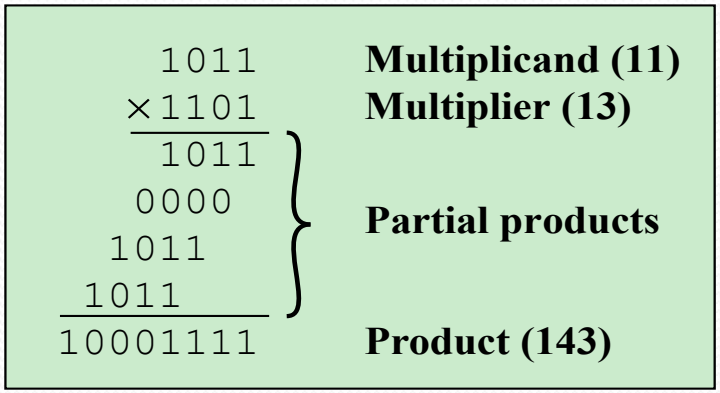
Un Signed Multiplication

1. Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.

2. The partial products are easily defined. When the multiplier bit is 0, the partial product is 0. When the multiplier is 1, the partial product is the multiplicand.

3. The total product is produced by summing the partial products. For this operation, each successive partial product is shifted one position to the left relative to the preceding partial product.

4. The multiplication of two *n*-bit binary integers results in a product of up to 2*n* bits in length (e.g., 11 * 11 = 1001).

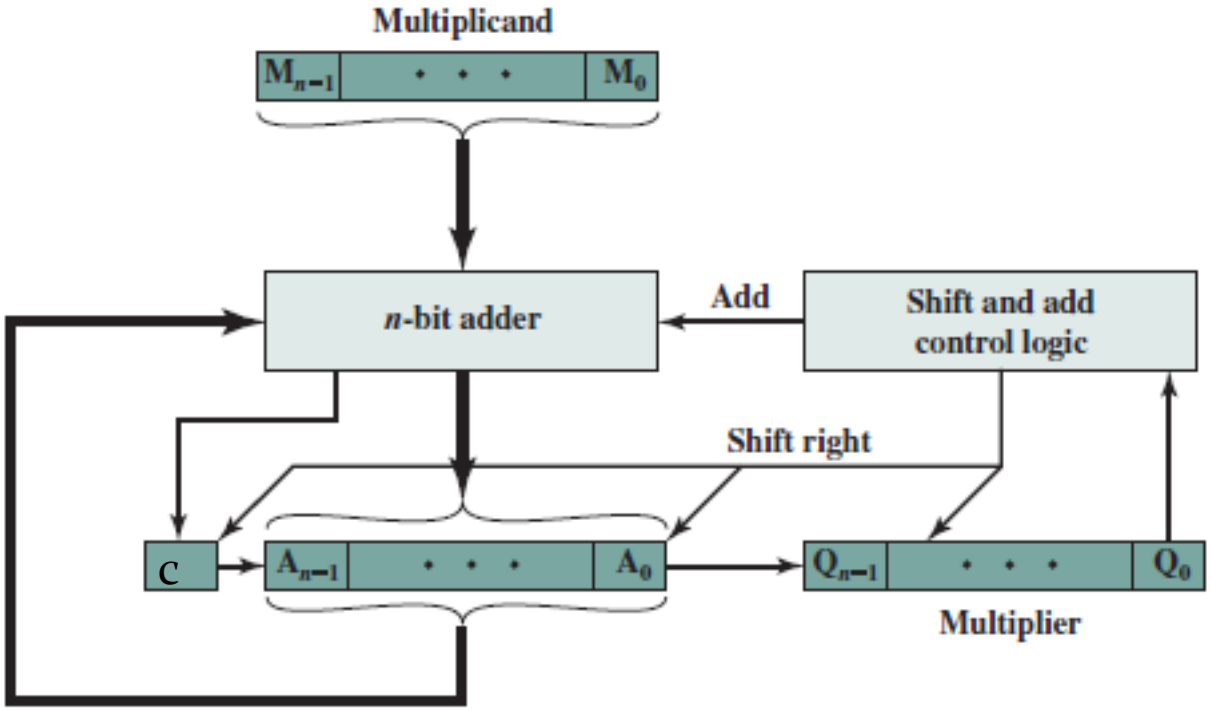


Hardware Implementation of Unsigned Binary Multiplication

- Compared with the pencil-and-paper approach, there are several things we can do to make computerized multiplication more efficient.
- First, we can perform a running addition on the partial products rather than waiting until the end.
- This eliminates the need for storage of all the partial products; fewer registers are needed.
- Second, we can save some time on the generation of partial products.
- For each 1 on the multiplier, an add and a shift operation are required; but for each 0, only a shift is required.

Hardware Implementation of Unsigned Binary Multiplication

- As shown in figure The multiplier and multiplicand are loaded into two registers (Q and M).
- A third register, the A register, is also needed and is initially set to 0.
- There is also a 1-bit C register, initialized to 0, which holds a potential carry bit resulting from addition.



(a) Block diagram

Hardware Implementation of Unsigned Binary Multiplication

- The operation of the multiplier is as follows.
  - Control logic reads the bits of the multiplier one at a time.
  - If Q0 is 1, then the multiplicand is added to the A register and the result is stored in the A register, with the C bit used for overflow.
  - Then all of the bits of the C, A, and Q registers are shifted to the right one bit, so that the C bit goes into An-1, A0 goes into Qn-1, and Q0 is lost.
  - If Q0 is 0, then no addition is performed, just the shift.
  - This process is repeated for each bit of the original multiplier.
  - The resulting 2n-bit product is contained in the A and Q registers.
  - Note that on the second cycle, when the multiplier bit is 0, there is no add operation.

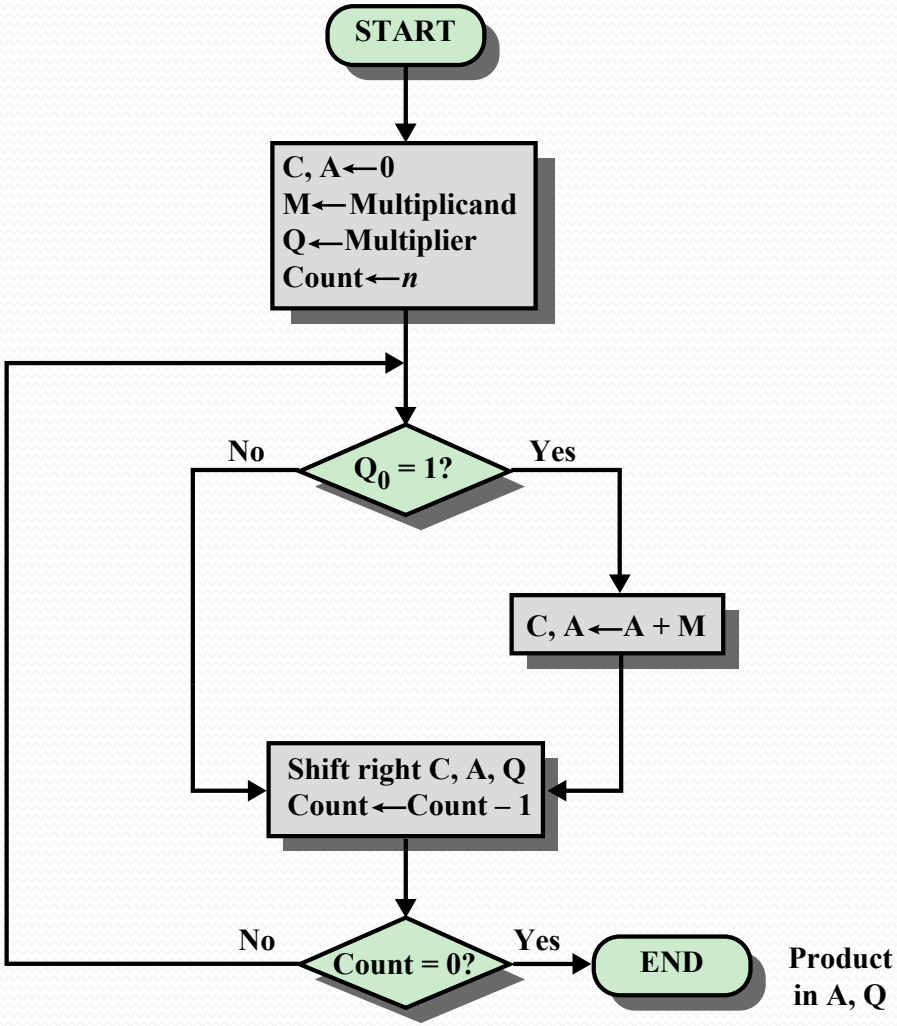
Hardware Implementation of Unsigned Binary Multiplication

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

Q: Multiplier
M: Multiplicand

- Control logic reads the bits of the multiplier one at a time.
- If Qo is 1, then the multiplicand is added to the A register and the result is stored in the A register, with the C bit used for overflow.
- Then all of the bits of the C, A, and Q registers are shifted to the right one bit, so that the C bit goes into A*n*-1, Ao goes into Q*n*-1, and Qo is lost.
- If Qo is 0, then no addition is performed, just the shift.
- This process is repeated for each bit of the original multiplier.
- The resulting 2*n*-bit product is contained in the A and Q registers.
- Note that on the second cycle, when the multiplier bit is 0, there is no add operation.

Flowchart for Unsigned Binary Multiplication



Booth's Algorithm (Signed Multiplication)

- Booth algorithm is used to multiply binary numbers in signed-2's complement form.
- The multiplier and multiplicand are placed in the Q and M registers respectively.
- There is also a 1-bit register placed logically to the right of the least significant bit (Q0) of the Q register and designated Q-1;
- The results of the multiplication will appear in the A and Q registers. A and Q-1 are initialized to 0.
- As before, control logic scans the bits of the multiplier one at a time.
- Now, as each bit is examined, the bit to its right is also examined.

Booth's Algorithm (Signed Multiplication)

- If the two bits are the same (1–1 or 0–0), then all of the bits of the A, Q, and Q-1 registers are shifted to the right 1 bit.
- If the two bits differ, then the multiplicand is added to or subtracted from the A register, depending on whether the two bits are 0–1 or 1–0.
- Following the addition or subtraction, the right shift occurs.
- In either case, the right shift is such that the leftmost bit of A, namely A*n*-1, not only is shifted into A*n*-2, but also remains in A*n*-1.
- This is required to preserve the sign of the number in A and Q.
- It is known as an **arithmetic shift**, because it preserves the sign bit.

Booth's Algorithm (Signed Multiplication)

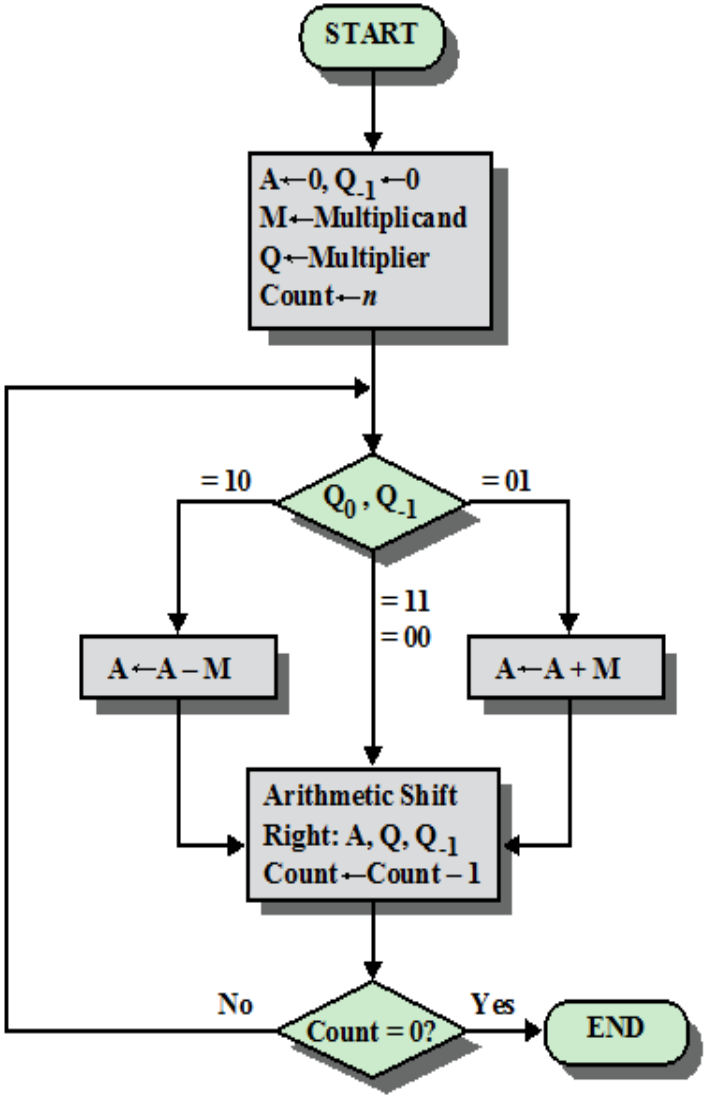
A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A ← A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A ← A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

Example of Booth's Algorithm (7X3)

Booth's Algorithm (Signed Multiplication)

A	Q	Q_{-1}	M	C		
0000	0011	0	0111	0100	Initial Values	
1001	0011	0	0111		A ← A - M	First Cycle
1100	1001	1	0111	0011	Shift	
1110	0100	1	0111	0010	Shift	Second Cycle
0101	0100	1	0111		A ← A + M	Third Cycle
0010	1010	0	0111	0001	Shift	
0001	0101	0	0111	0000	Shift	Fourth Cycle

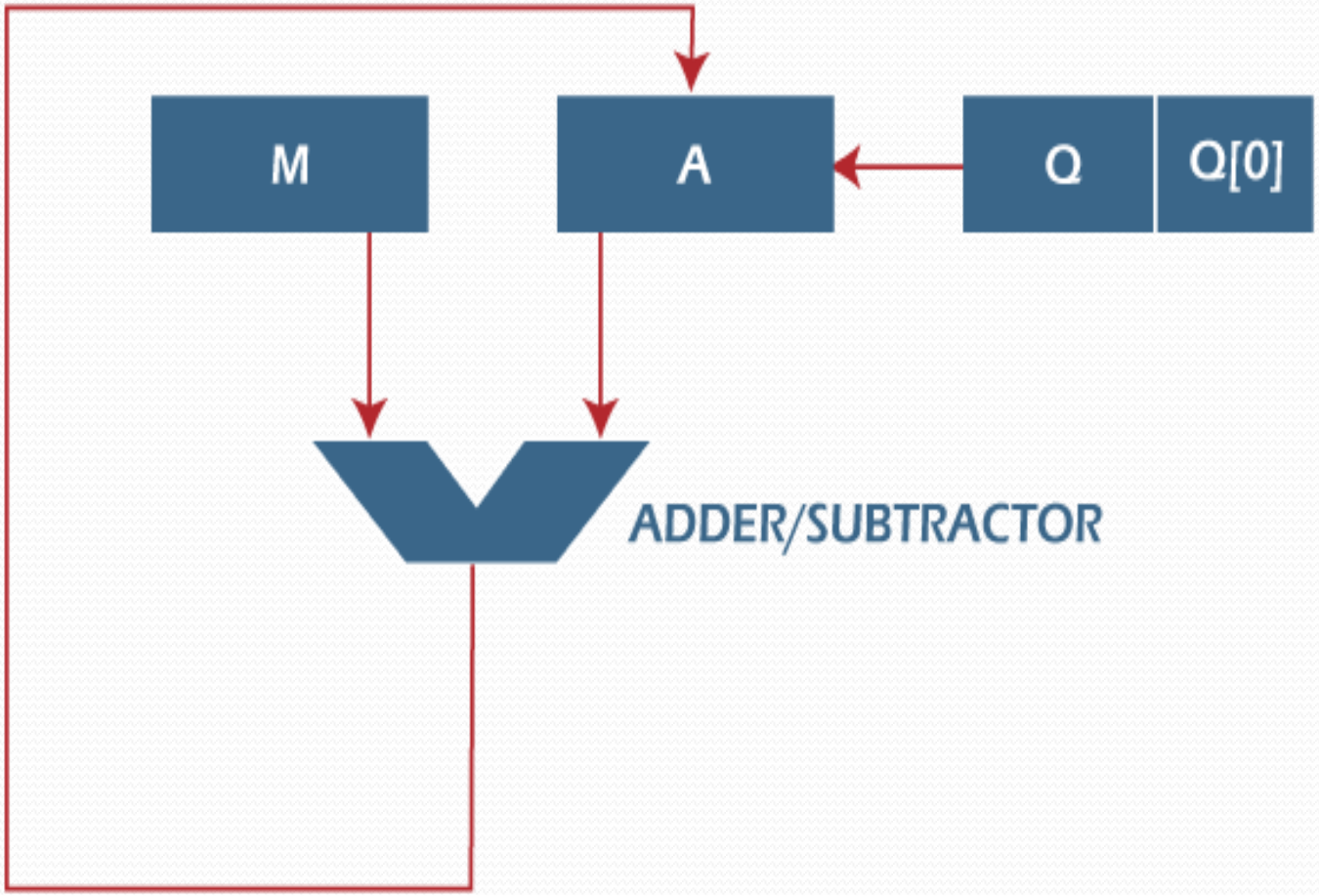
Example of Booth's Algorithm (7X3)

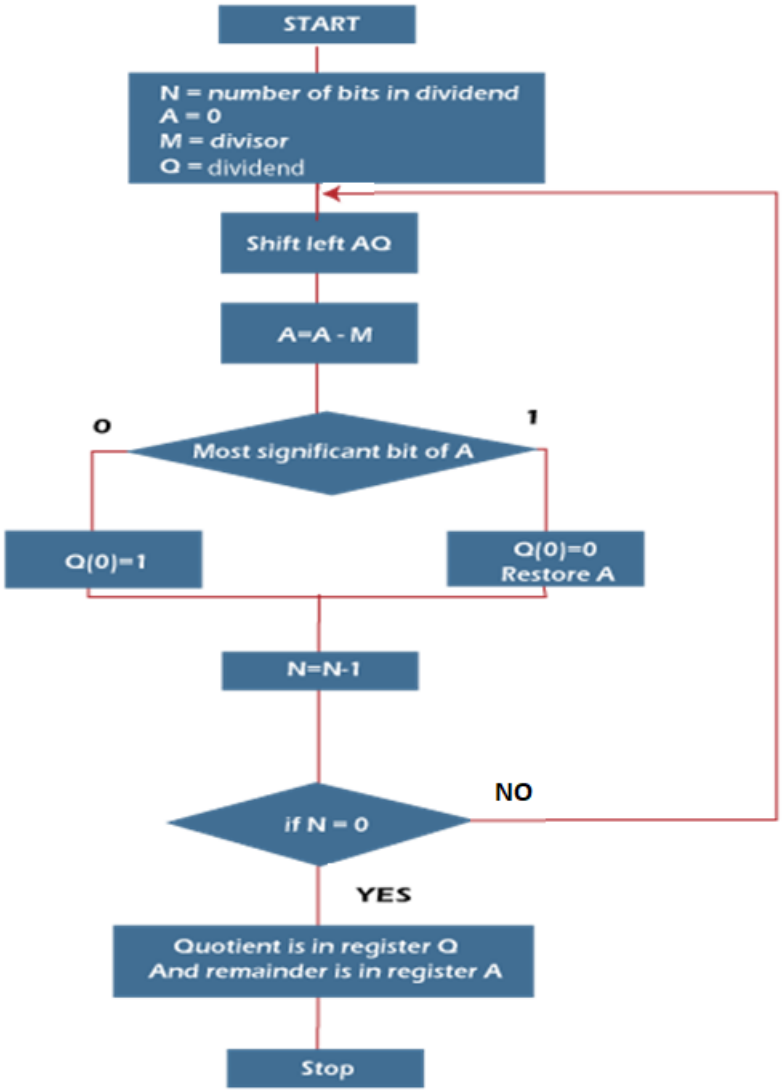


Restoring division Algorithm for Unsigned Integer

- A division algorithm provides a quotient and a remainder when we divide two number.
- We are using restoring term because we know that the value of register A will be restored after each iteration.
- Here, register Q is used to contain the quotient, and register A is used to contain the remainder.
- Here, the divisor will be loaded into the register M, and n-bit divided will be loaded into the register Q.
- 0 is the starting value of a register.
- The values of these types of registers are restored at the time of iteration.
- That's why it is known as restoring.

Restoring division Algorithm for Unsigned Integer





Step 1: Set Register A=0, register M= Divisor, register Q = Dividend, and N = the number of bits in dividend.

Step 2: In this step, register A and register Q will be treated as a single unit, and the value of both the registers will be shifted left.

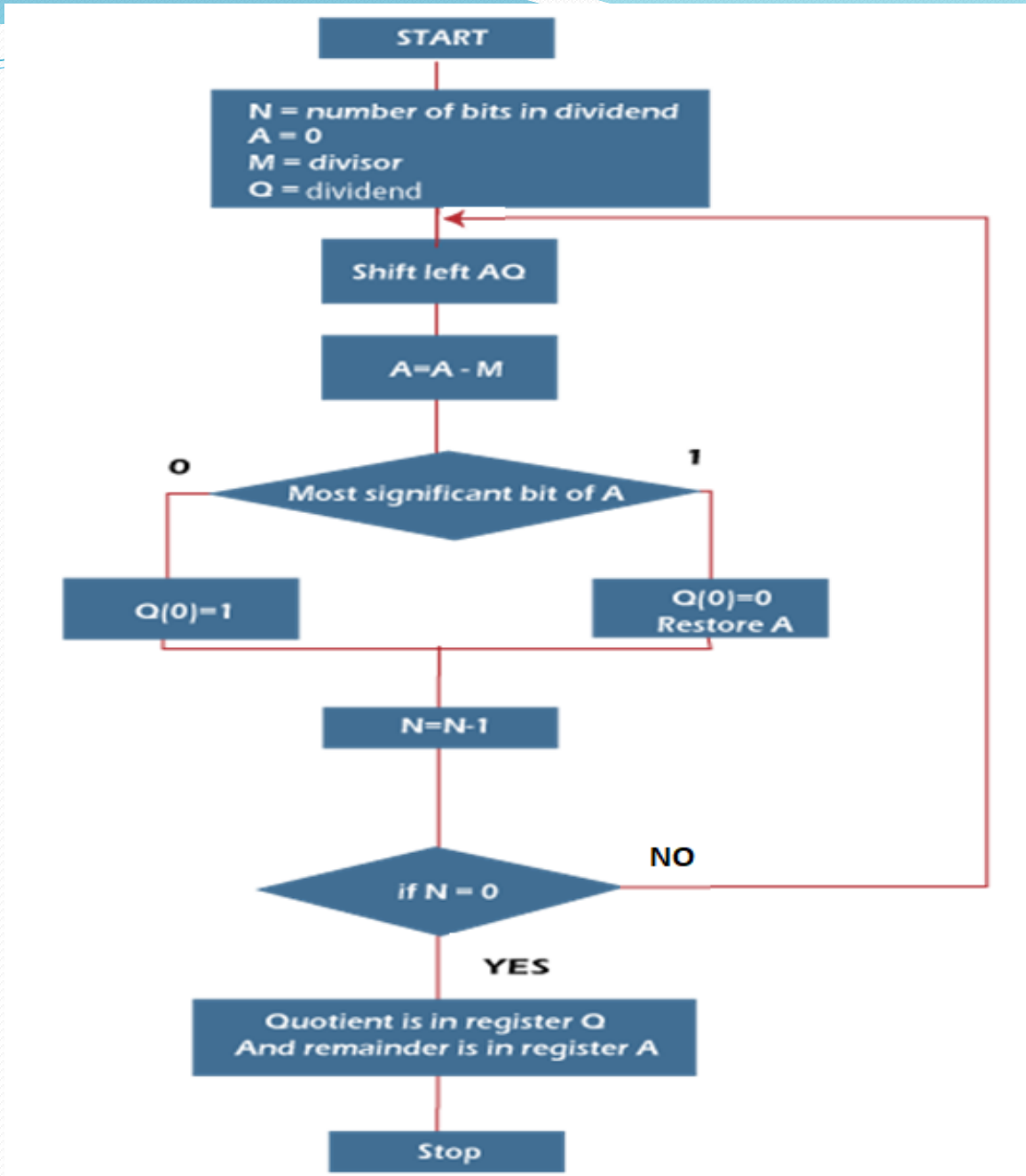
Step 3: After that, Subtract M from register A and store result in register A.

Step 4: Now, check the MSB bit of register A. If this bit of register A is 0, then the LSB bit of register Q will be set with a value 1(Q_0=1). If the MSB bit of A is 1, then the LSB of register Q will be set to with value 0 (Q_0=1), and restore the value of A that means it will restore the value of register A before subtraction with M.

Step 5: After that, the value of N will be decremented. (N= N-1) N is used as a counter.

Step 6: Now, if N = 0, we will break the loop. Otherwise, again go to step 2.

Step 7: This is the last step. In this step, the quotient is contained in the register Q, and the remainder is contained in register A.


START
N = number of bits in dividend
A = 0
M = divisor
Q = dividend
Shift left AQ
A=A - M
0
1
Most significant bit of A
Q(0)=1
Q(0)=0
Restore A
N=N-1
NO
if N = 0
YES
Quotient is in register Q
And remainder is in register A
Stop

N	M	A	Q	Operation
4	00011	00000	1011	Initialize
	00011	00001	011_	Shift left AQ
	00011	11110	011_	A = A - M
	00011	00001	0110	Q[0] = 0 And restore A
3	00011	00010	110_	Shift left AQ
	00011	11111	110_	A = A - M
	00011	00010	1100	Q[0] = 0
2	00011	00101	100_	Shift left AQ
	00011	00010	100_	A = A - M
	00011	00010	1001	Q[0] = 1
1	00011	00101	001_	Shift left AQ
	00011	00010	001_	A = A - M
	00011	00010	0011	Q[0] = 1

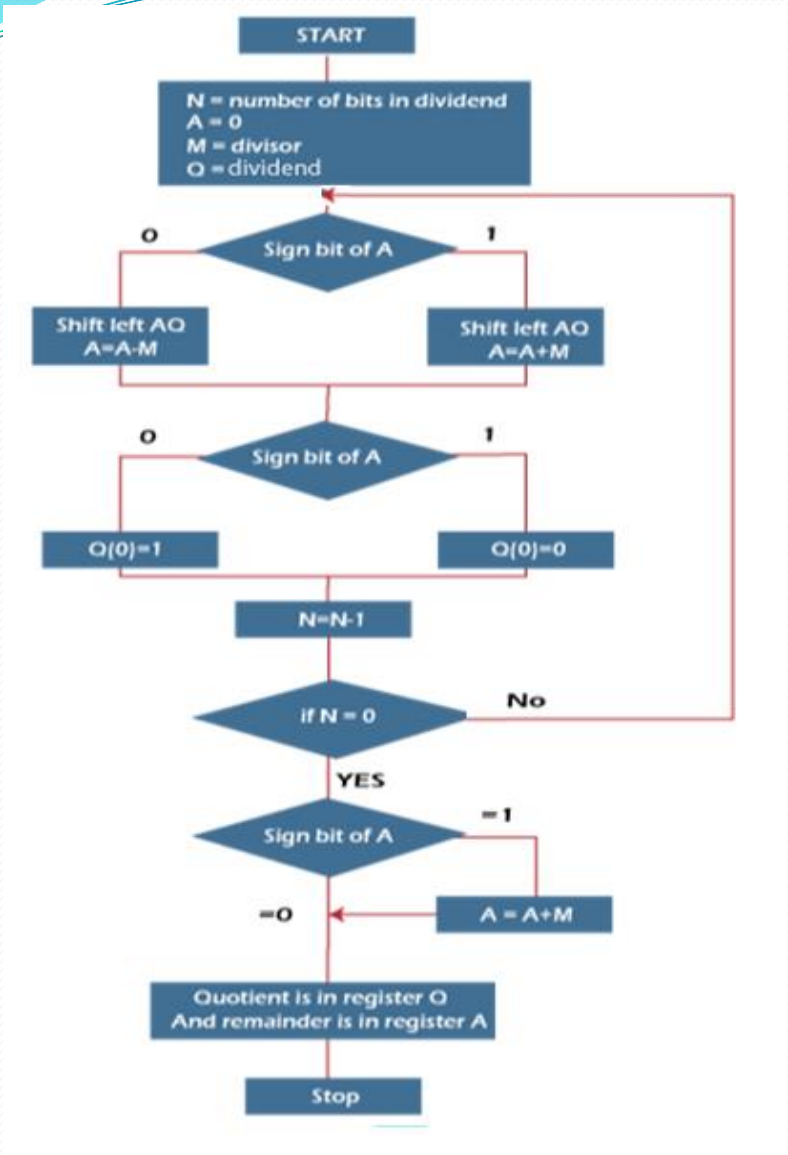
Q= Dividend = 11 (1011)
M= Divisor = 3 (00011)
-M = 11101

After Division
Quotient = Q= 3 (0011)
Remainder = A=2 (0010)

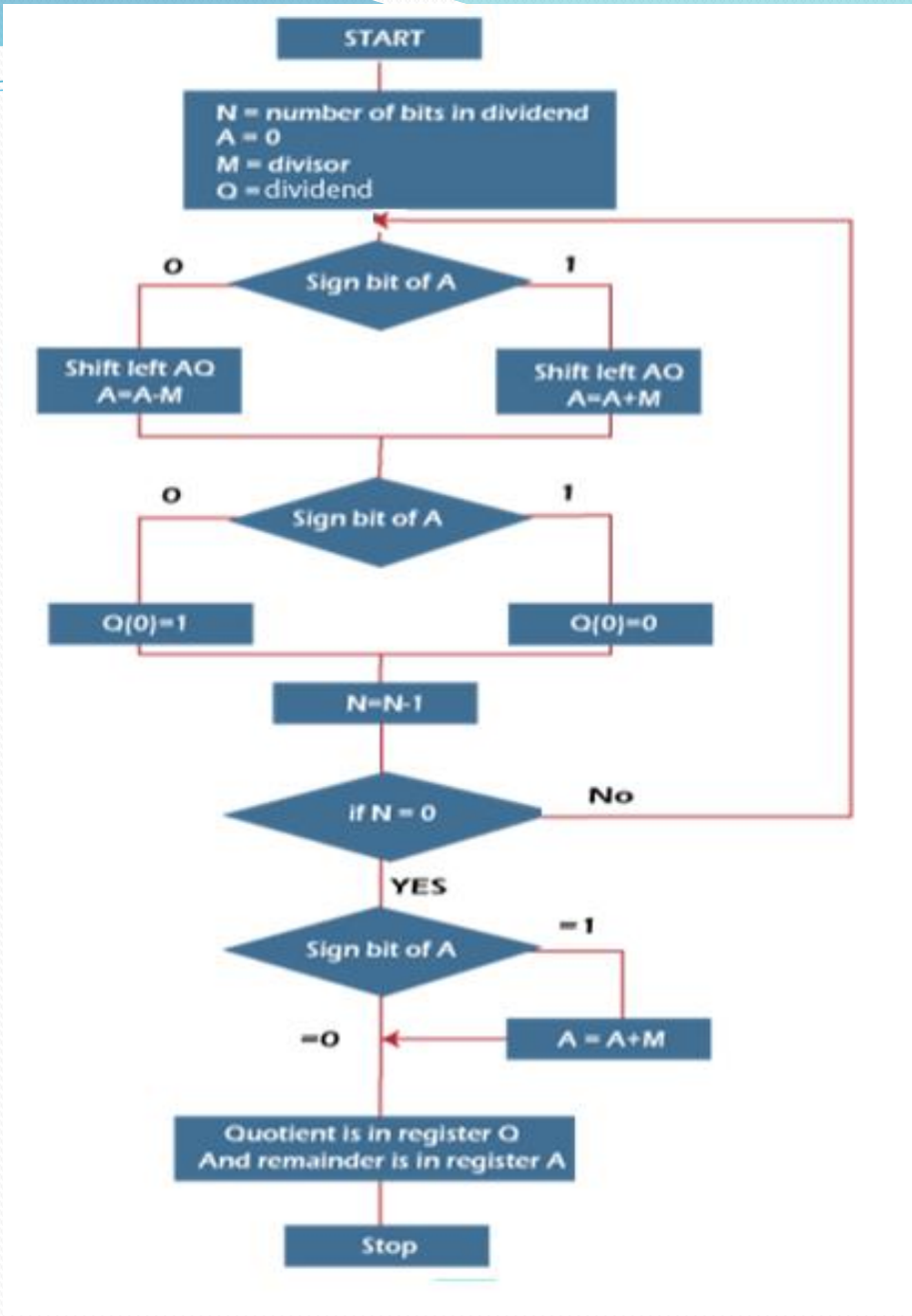
Non Restoring division algorithm

- The non-restoring division algorithm is more complex as compared to the restoring division algorithm.
- But when we implement this algorithm in hardware, it has an advantage, i.e., it contains only one decision and addition/subtraction per quotient bit.
- After performing the subtraction operation, there will not be any restoring steps.
- Due to this, the numbers of operations basically cut down up to half.
- Because of the less operation, the execution of this algorithm will be fast.
- This algorithm basically performs simple operations such as addition, subtraction.

Non Restoring division



Step 1: Initialize reg A = 0, reg M = Divisor, reg Q = Dividend, and N = number of bits in dividend.
Step 2: In this step, check the sign bit of A.
Step 3: If this bit of reg A = 1, then shift AQ through left, and perform A = A + M.
If this bit is 0, then shift AQ into left and perform A = A - M. That means in case of 0, the 2's complement of M is added into register A, and the result is stored into A.
Step 4: Now, we will check the sign bit of A again.
Step 5: If this bit of reg A = 1, then Q[0] will become 0. If this bit is 0, then Q[0] will become 1. Here Q[0] indicates LSB bit of Q.
Step 6: After that, decrement the value of N. Here N is used as a counter.
Step 7: If the value of N = 0, then we will go to the next step 8 . Otherwise, again go to step 2.
Step 8: We will perform A = A + M if the sign bit of register A is 1.
Step 9: This is the last step. In this step, register A contains the remainder, and register Q contains the quotient.


START
N = number of bits in dividend
A = 0
M = divisor
Q = dividend
Sign bit of A
0
1
Shift left AQ
A=A-M
Shift left AQ
A=A+M
Sign bit of A
0
1
Q(0)=1
Q(0)=0
N=N-1
if N = 0
No
YES
Sign bit of A
=1
=0
A = A+M
Quotient is in register Q
And remainder is in register A
Stop

Non Restoring division

N	M	A	Q	Action
4	00011	00000	1011	Begin
	00011	00001	011_	Shift left AQ
	00011	11110	011_	A = A - M
3	00011	11110	0110	Q[0] = 0
	00011	11100	110_	Shift left AQ
	00011	11111	110_	A = A + M
2	00011	11111	1100	Q[0] = 0
	00011	11111	100_	Shift left AQ
	00011	00010	100_	A = A +M
1	00011	00010	1001	Q[0] = 1
	00011	00101	001_	Shift left AQ
	00011	00010	001_	A = A - M
0	00011	00010	0011	Q[0] = 1

Q= Dividend = 11 (1011)
M= Divisor = 3 (00011)
-M = 11101

After Division
Quotient = Q= 3 (0011)
Remainder = A=2 (0010)

References

1. William Stallings- “Computer Organization and Architecture: Designing for Performance”, 11 Pearson Publication, 10th Edition, 2013.
2. John P. Hayes- “Computer Architecture and Organization”, McGraw-Hill, 1988.