

Lecture of Module 1

Introduction

Overview

- ▶ **Computer Architecture**
- ▶ **Commuter Organization**
- ▶ **Structure and Function**
- ▶ **Functional units**
- ▶ **Basic Operational Concept**
- ▶ **Registers**
- ▶ **Bus Interconnection**
- ▶ **Bus Structure**
- ▶ **Harvard Architecture**
- ▶ **Von Neumann Architecture**
- ▶ **IAS Architecture**

What is a Computer?



A computer is an Electronic device that can be programmed to process information, data etc. to yield meaningful results.



Computer Architecture

Computer Organization

- Attributes of a system visible to the programmer
- Have a direct impact on the logical execution of a program

- Instruction set, number of bits used to represent various data types, I/O mechanisms, techniques for addressing memory

Computer Architecture

Architectural attributes include:

Organizational attributes include:

Computer Organization

- Hardware details transparent to the programmer, control signals, interfaces between the computer and peripherals, memory technology used

- The operational units and their interconnections that realize the architectural specifications

- **Whether a computer system can execute multiply instructions?**

Architectural Issue

Organizational Issue

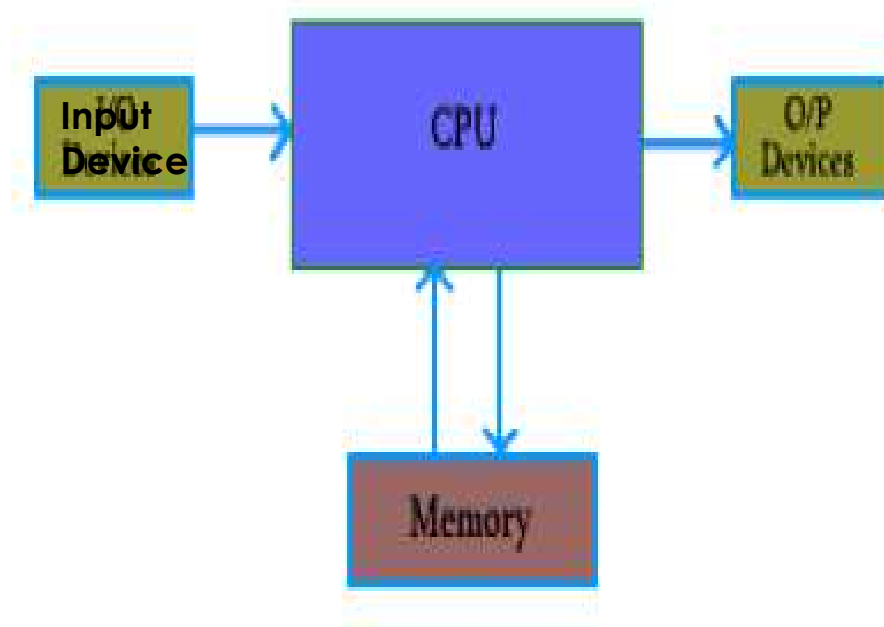
- **Whether to have a special multiply circuit? Or**
- **To have a method that makes repeated use of the add unit?**

Architectural Issue

Organizational decision depends on:

- **The frequency of use of the multiply instruction**
- **The relative speed of the two approaches**
- **The cost and size of the special multiply unit**

Structure and Function



- ▶ Structure
 - ▶ The way in which components relate to each other
- ▶ Function
 - ▶ The operation of individual components as part of the structure



- ▶ Hierarchical system

- ▶ Set of interrelated subsystems

- ▶ Hierarchical nature of complex systems is essential to both their design and their description

- ▶ Designer need only deal with a particular level of the system at a time

- ▶ Concerned with structure and function at each level

- ▶ Structure

- ▶ The way in which components relate to each other

- ▶ Function

- ▶ The operation of individual components as part of the structure

Function

- ▶ There are four basic functions that a computer can perform:
 - ▶ **Data processing**
 - ▶ Data may take a wide variety of forms and the range of processing requirements is broad
 - ▶ **Data storage**
 - ▶ Short-term
 - ▶ Long-term
 - ▶ **Data movement**
 - ▶ Input-output (I/O) - when data are received from or delivered to a device (peripheral) that is directly connected to the computer
 - ▶ Data communications – when data are moved over longer distances, to or from a remote device
 - ▶ **Control**
 - ▶ A control unit manages the computer's resources and cares the performance of its functional parts in response to instructions

Structure

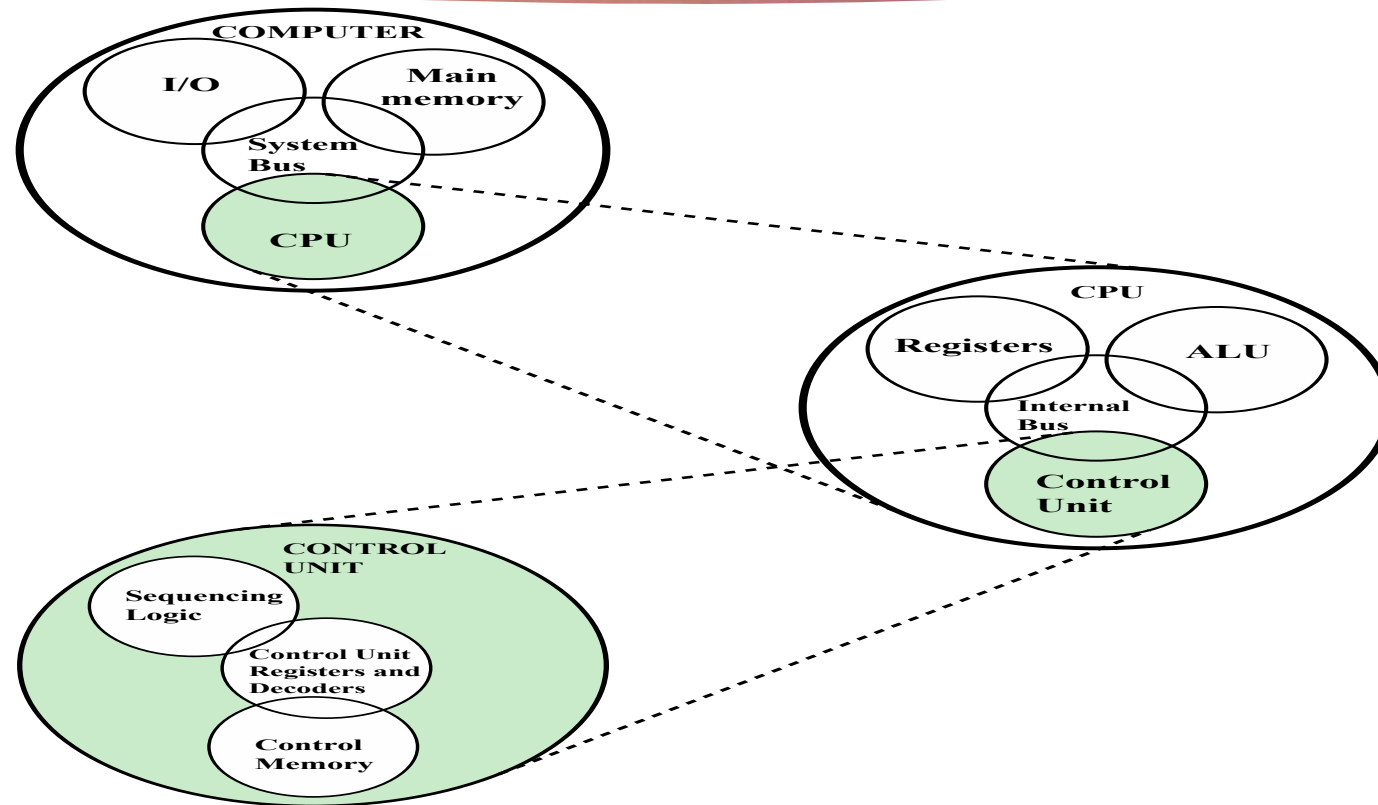



Figure 1.1 A Top-Down View of a Computer

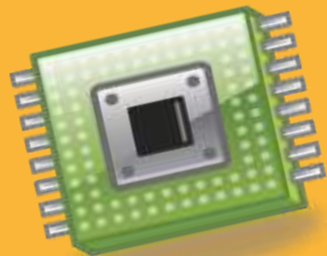
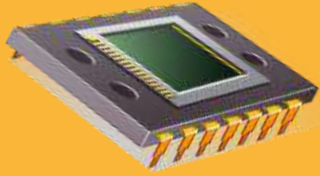


There are four main structural functional units of the computer:

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

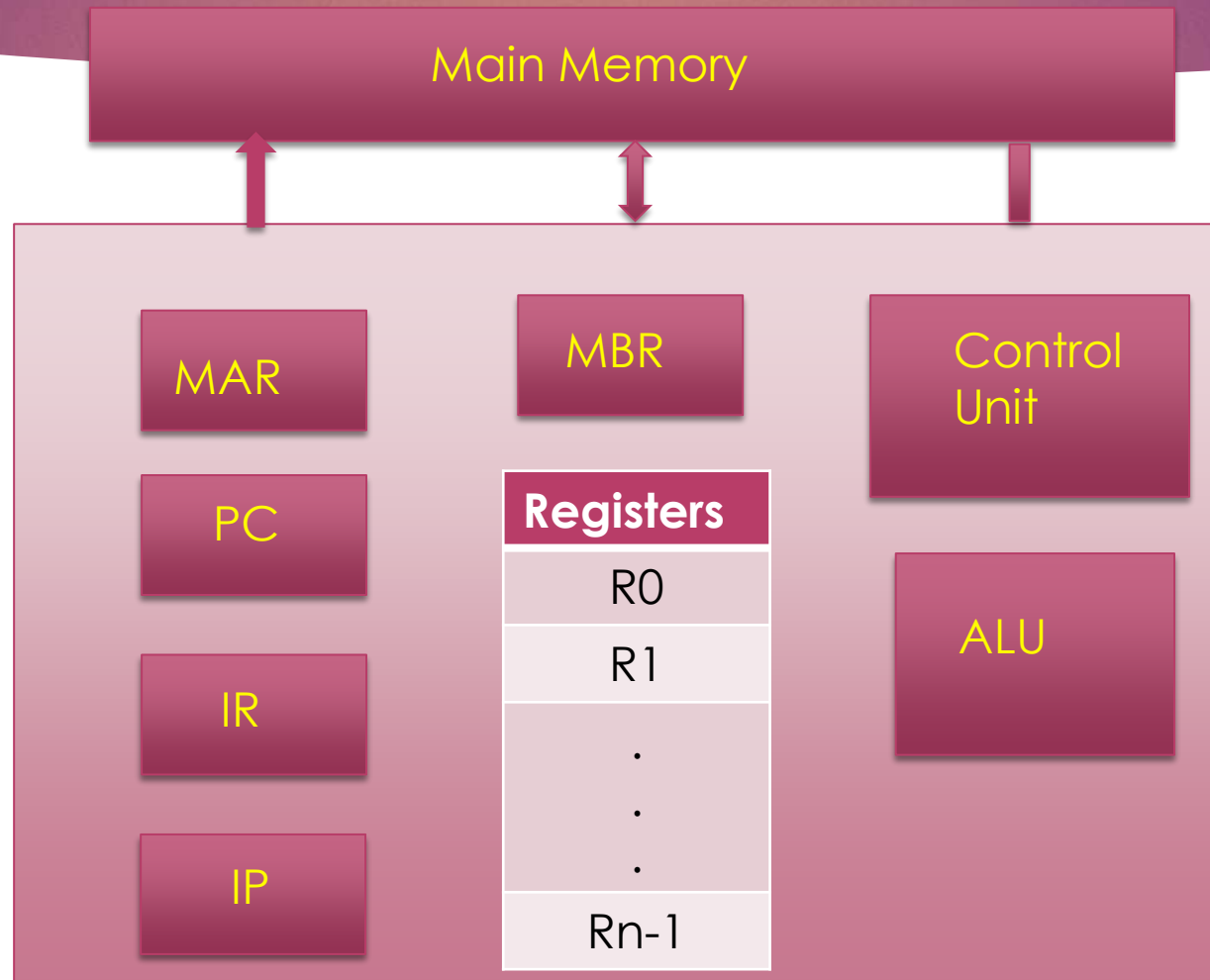
CPU

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

Basic Operational Concept



Registers

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)

- Specifies the address in memory of the word to be written from or read into the MBR

Instruction register (IR)

- Contains the current opcode of the instruction being executed

Instruction Pointer (IP)

- Holds the address of the current instruction on which processing is going on

Program counter (PC)

- Contains the address of the next instruction to be fetched from memory

General Purpose Registers (GPR)

- Arrays of registers present in the processor

Other Registers

- ▶ **Stack Pointer (SP):** It is a memory pointer. It points to a memory location called STACK. The beginning of the Stack is defined by loading the starting address to the Stack pointer.
- ▶ **Base Register:** It stores the base address of the memory. Any address of any data is calculated logically after finding the address which is in the base register
- ▶ **Temporary Register (TR):** Holds temporary data during processing.
- ▶ **Flag Register(FR):** Shows the status of the system during processing. It is affected by ALU operations. It consists of different Flag bits.



Flag Register

- ▶ **CF:** If there is a carry then flag bit is set, otherwise reset.
- ▶ **PF:** If even numbers 1's in the result then flag is set, otherwise reset.
- ▶ **AC:** If carry is generated from D3 during operation and passes to D4 then set, otherwise reset.
- ▶ **ZF:** If the result is zero then flag is set, otherwise reset.
- ▶ **SF:** If D7 is 1 then flag is set, otherwise reset (Signed number).
- ▶ **TF:** Trap is a non-maskable interrupt. When non-maskable interrupt is generated then flag is set.
- ▶ **IF:** If any interrupt is generated then flag is set.
- ▶ **DF:** When memory is accessed from lower location to higher location the flag is set. When memory is accessed from higher location to lower location the flag is reset.
- ▶ **OF:** When any overflow takes place during arithmetic or logical operations in Register, Stack, Queue, Array etc. then flag is set showing the overflow condition of the current operation.

Bus Interconnection

- ▶ If a computer is to achieve a reasonable speed of operation, it must be organized so that all units can handle one full word of data at a given time.
- ▶ When a word of data is transferred between units, all its bits are transmitted in parallel.
- ▶ This requires a considerable number of wires (lines) to establish the necessary connections.
- ▶ **BUS:** A collection of wires that connects several devices to carry the information to or from different units of the system is called as BUS.

Three types Bus

- **Data Bus**
- **Address Bus**
- **Control Bus**

The interconnection structure must support the following types of transfers:

Memory
to
processor

Processor reads an instruction or a unit of data from memory

Processor
to
memory

Processor writes a unit of data to memory

I/O to
processor

Processor reads data from an I/O device via an I/O module

Processor
to I/O

Processor sends data to the I/O device

I/O to or
from
memory

An I/O module is allowed to exchange data directly with memory without going through the processor using direct memory access

Bus Interconnection

A communication pathway connecting two or more devices

- Key characteristic is that it is a shared transmission medium

Signals transmitted by any one device are available for reception by all other devices attached to the bus

- If two devices transmit during the same time period their signals will overlap and become garbled

Typically consists of multiple communication lines

- Each line is capable of transmitting signals representing binary 1 and binary 0

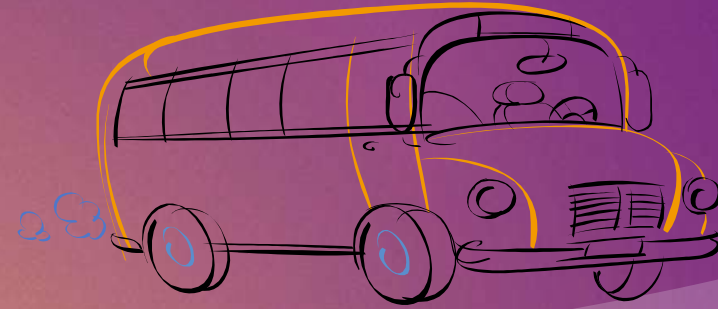
Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy

System bus

- A bus that connects major computer components (processor, memory, I/O)

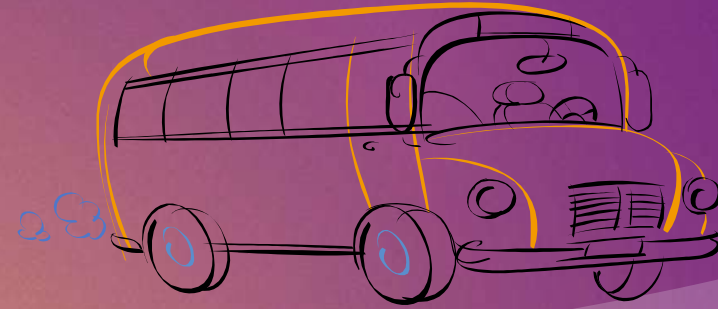
The most common computer interconnection structures are based on the use of one or more system buses

Data Bus



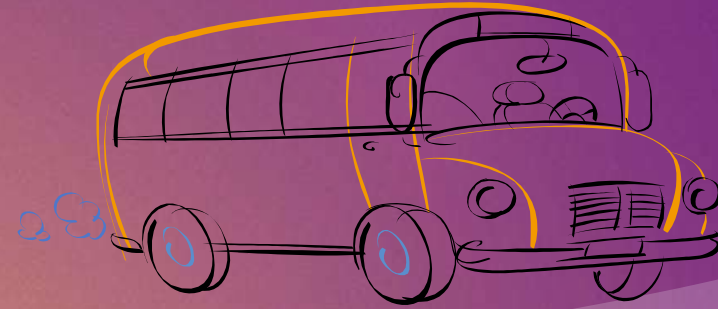
- ▶ Data lines that provide a path for moving data among system modules
- ▶ Number of wires depends on type of data transfer and word length
- ▶ May consist of 32, 64, 128, or more separate lines for parallel communication
- ▶ One line is required for serial communication
- ▶ The number of lines is referred to as the *width* of the data bus
- ▶ The number of lines determines how many bits can be transferred at a time (word length)
- ▶ The width of the data bus is a key factor in determining overall system performance
- ▶ **Direction is Bidirectional**

Address Bus



- ▶ Used to designate the source or destination of the data
 - ▶ If the processor wishes to read or write a word of data from or to memory it puts the address of the desired word on the address lines
- ▶ Width determines the maximum possible memory capacity of the system
- ▶ Also used to address I/O ports
 - ▶ Used to select a I/O port
- ▶ **Direction is unidirectional**

Control Bus



- ▶ All the functions of the system must be synchronized and controlled
- ▶ This is the function of control unit which provides control signals through buses
- ▶ Used to control the access and the use of the data and address lines
- ▶ Because the data and address lines are shared by all components there must be a means of controlling their use
- ▶ Control signals transmit both command and timing information among system modules
- ▶ Timing signals indicate the validity of data and address information
- ▶ Command signals specify operations to be performed
- ▶ Each line of the bus indicates a particular control signal
- ▶ A particular control line may be unidirectional or bidirectional but collectively as a bus no concept of direction

Bus Structure

- ▶ According to the connection mechanism of different functional units the Bus structures are of two types

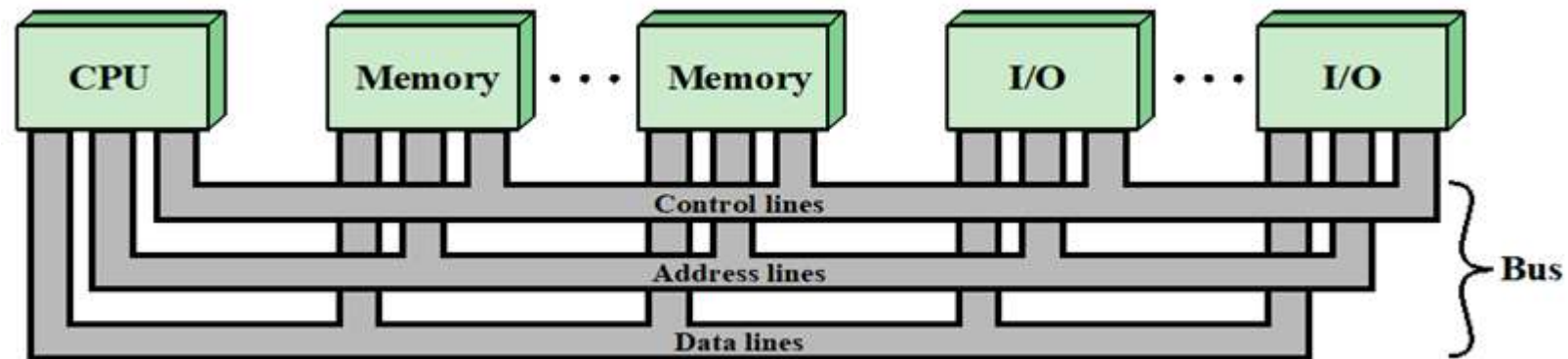
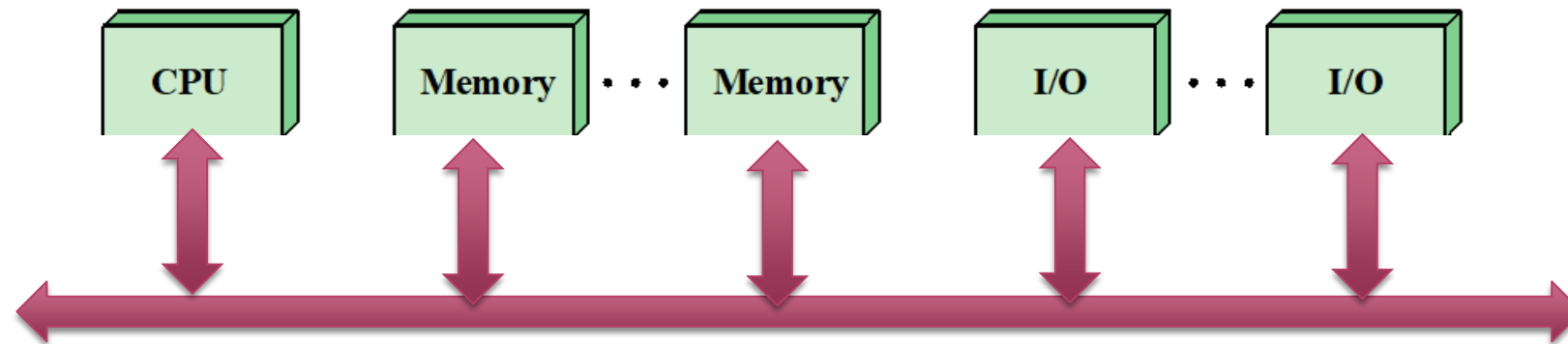
Single Bus structure

Multi-Bus structure

Single Bus structure:

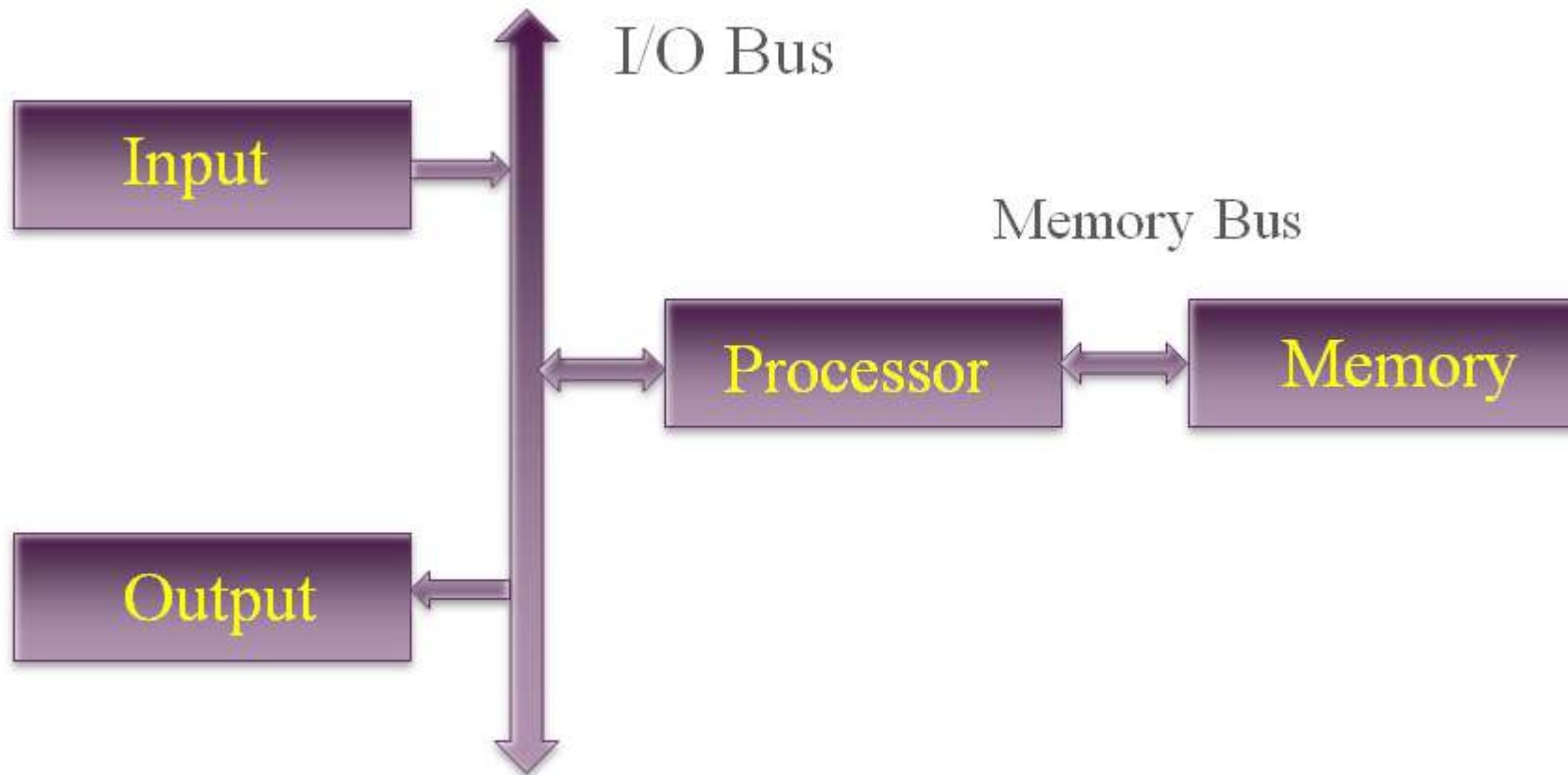
- ▶ All units are connected to single I/O bus
- ▶ At any given time two units can actively use the bus
- ▶ Bus control is used to arbitrate multiple requests for use of bus
- ▶ Flexibility for attaching peripheral devices
- ▶ Low hardware complexity
- ▶ Low cost
- ▶ But, slower data transfer

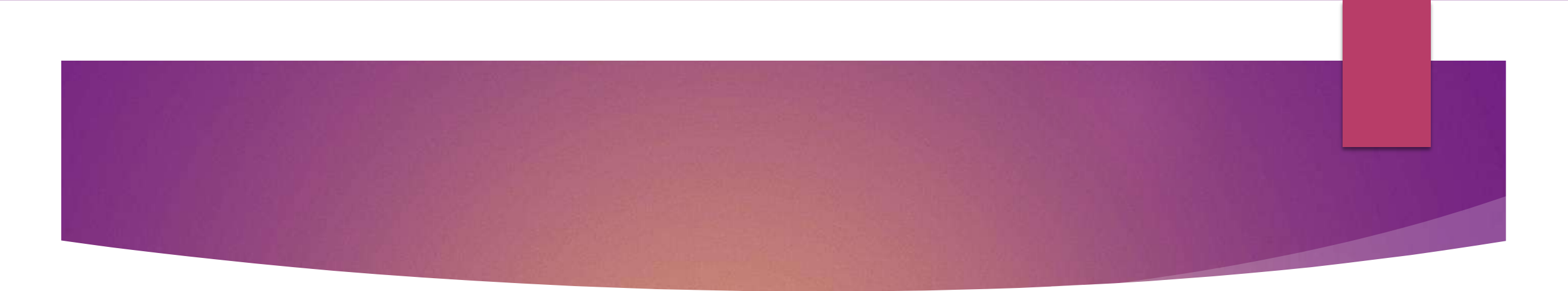
Single Bus structure

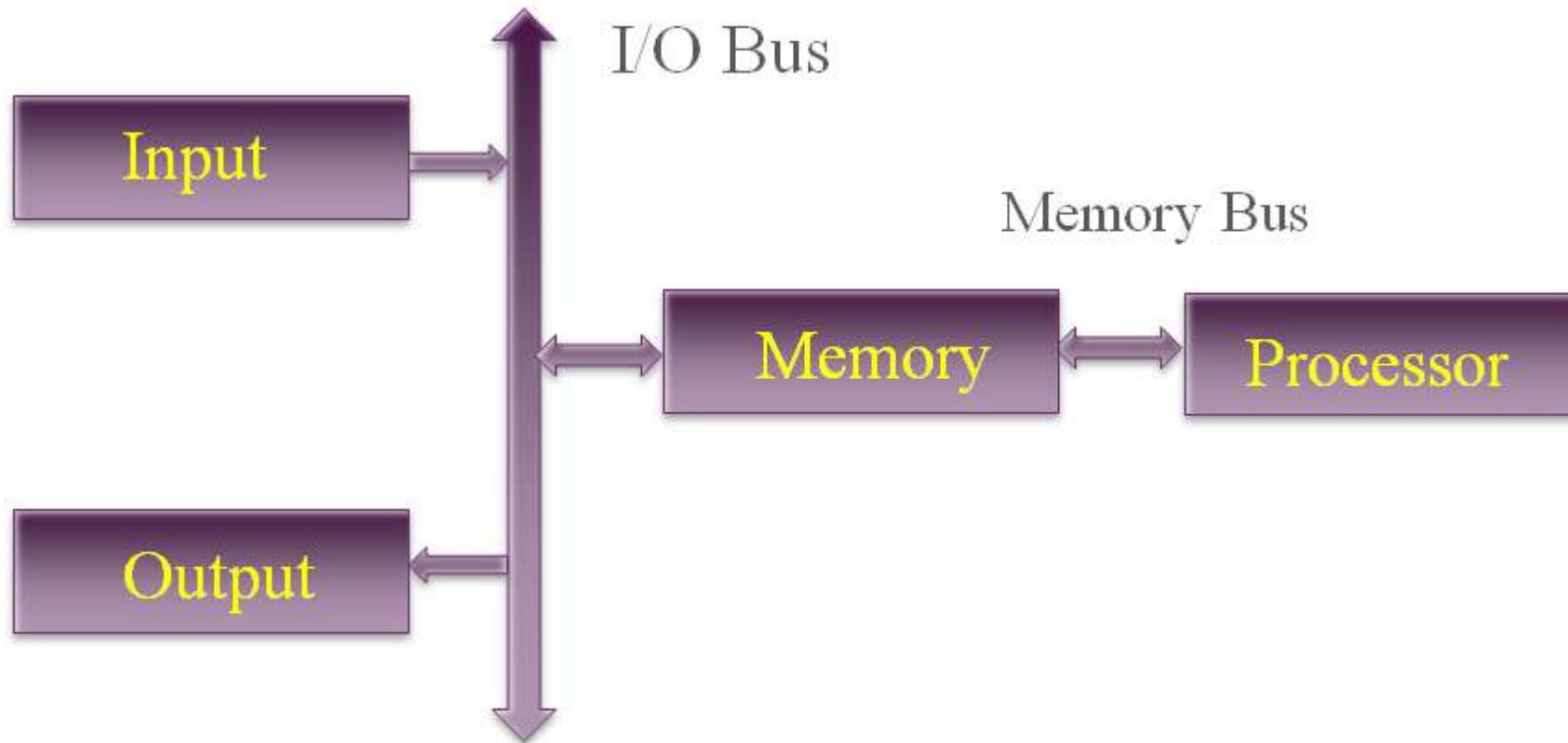


Bus Interconnection Scheme

Multi Bus structure:



- 
- ▶ It is a simplest Multi bus (two bus) computer
 - ▶ The processor interacts with memory through **memory bus**
 - ▶ Input and output functions are handled over an **I/O bus**
 - ▶ Data passes to memory for processing through the processor
 - ▶ I/O transfers are usually under direct control of the processor
 - ▶ Processor initiates the transfer and monitors their progress until completion
 - ▶ In this architecture the processor sit ideally after initiating the I/O operations till completion
 - ▶ Wastage of CPU time which degrades the performance
 - ▶ So, another multi bus architecture has been developed to enhance the performance of the system



- ▶ It is another Multi bus (two bus) architecture
- ▶ Here, the position of memory and processor interchanged
- ▶ I/O transfers are performed directly to or from memory
- ▶ But, memory can not control the I/O transfer
- ▶ So, a control circuitry as part of the I/O equipment is necessary
- ▶ That control circuitry is a special purpose processor called as **Peripheral Processor** or **Secondary Processor** or **I/O Channel** which controls the I/O transfer
- ▶ The main processor initiates I/O transfer by passing required information to the I/O channel
- ▶ The I/O channel then takes over and controls the actual transfer of data
- ▶ During I/O operations now the main processor is free and it can perform other CPU operations
- ▶ So, the performance enhanced

Design of practical Computer

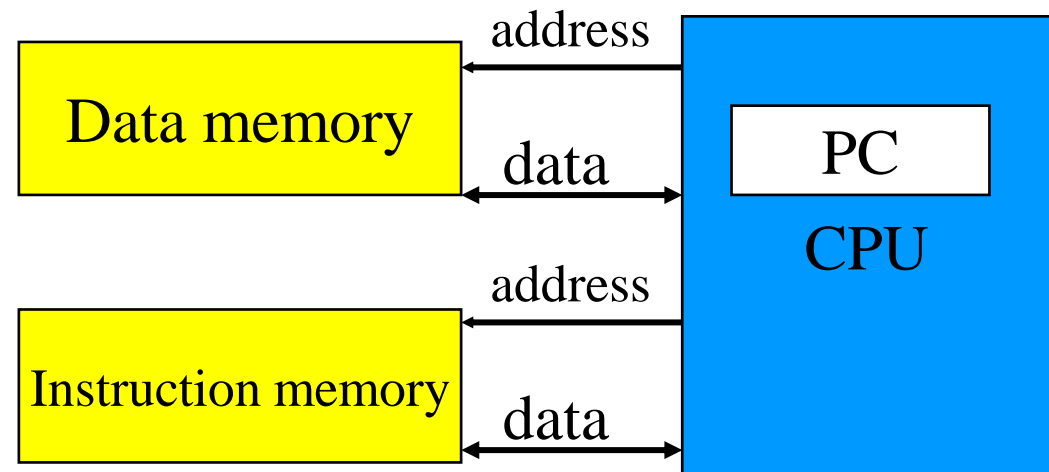
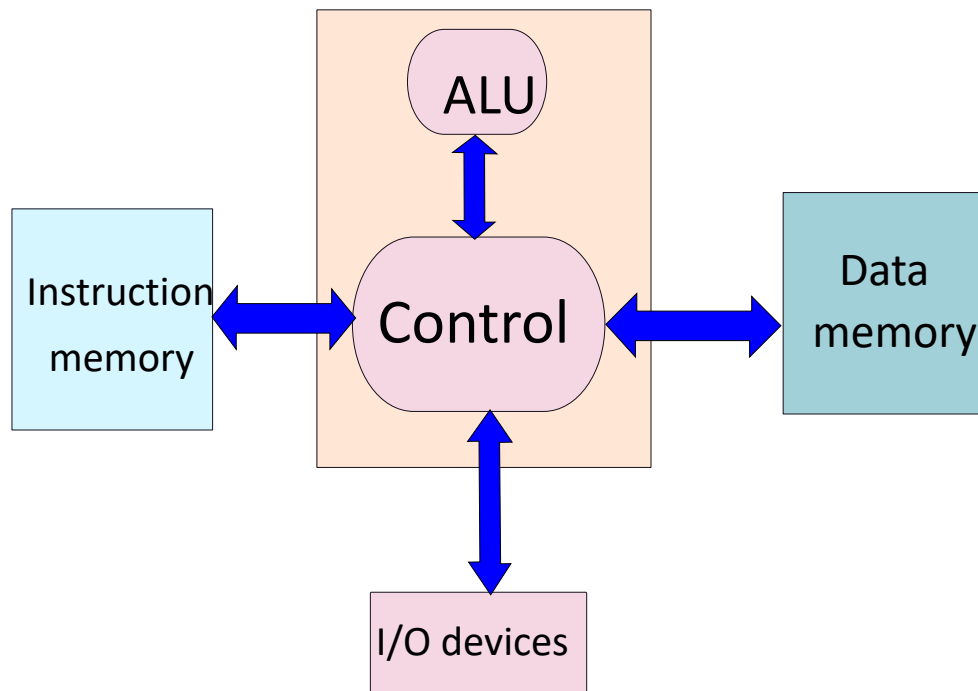
Two proposed architectural design

Harvard Architecture

Von Neumann Architecture

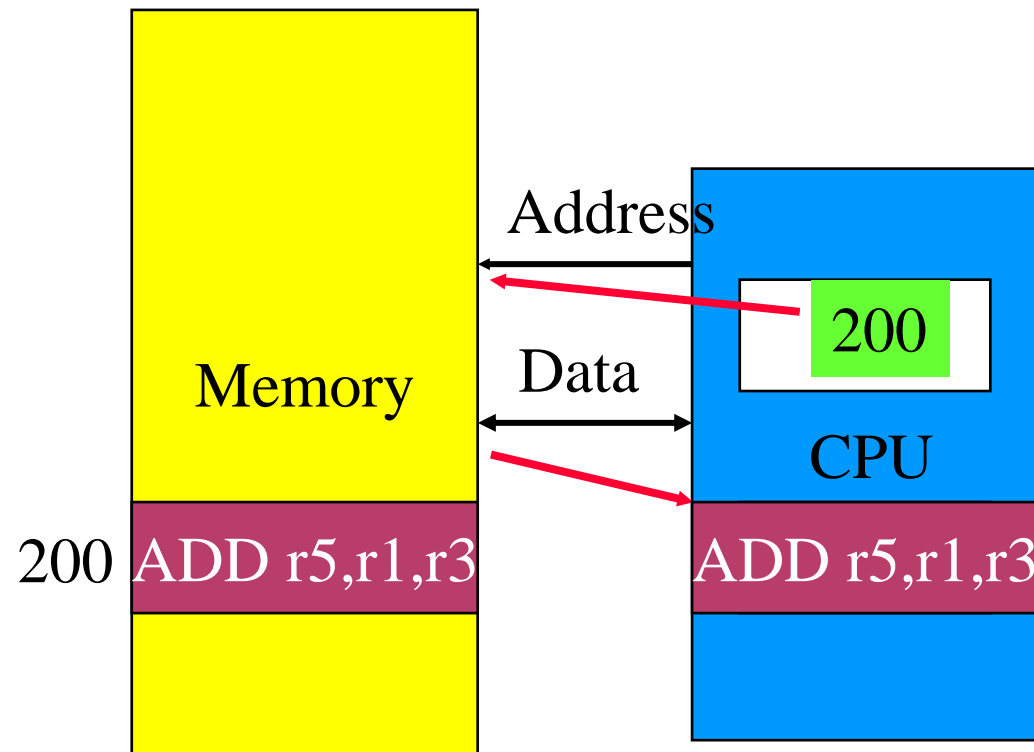
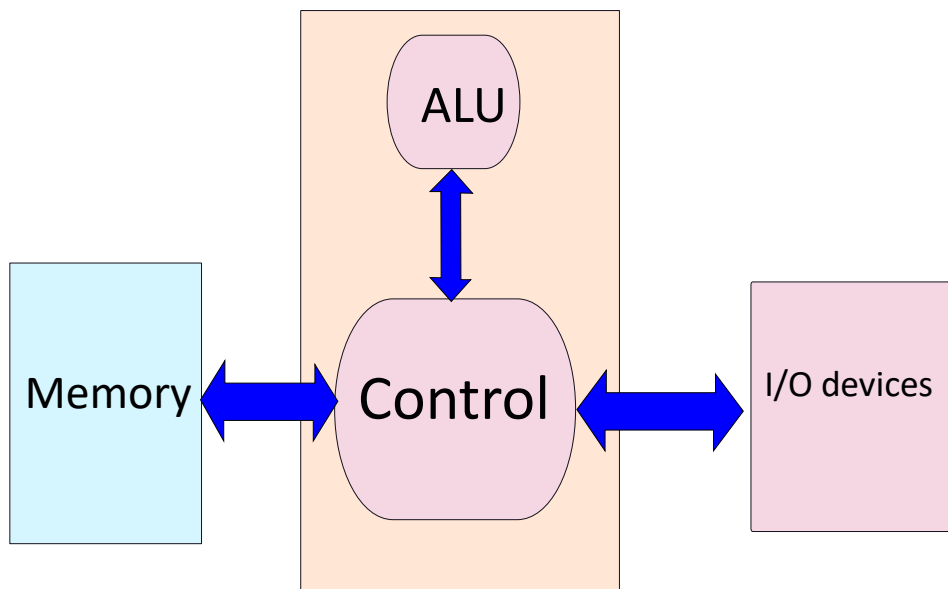
Harvard Architecture

- ▶ Separate data and instruction memories



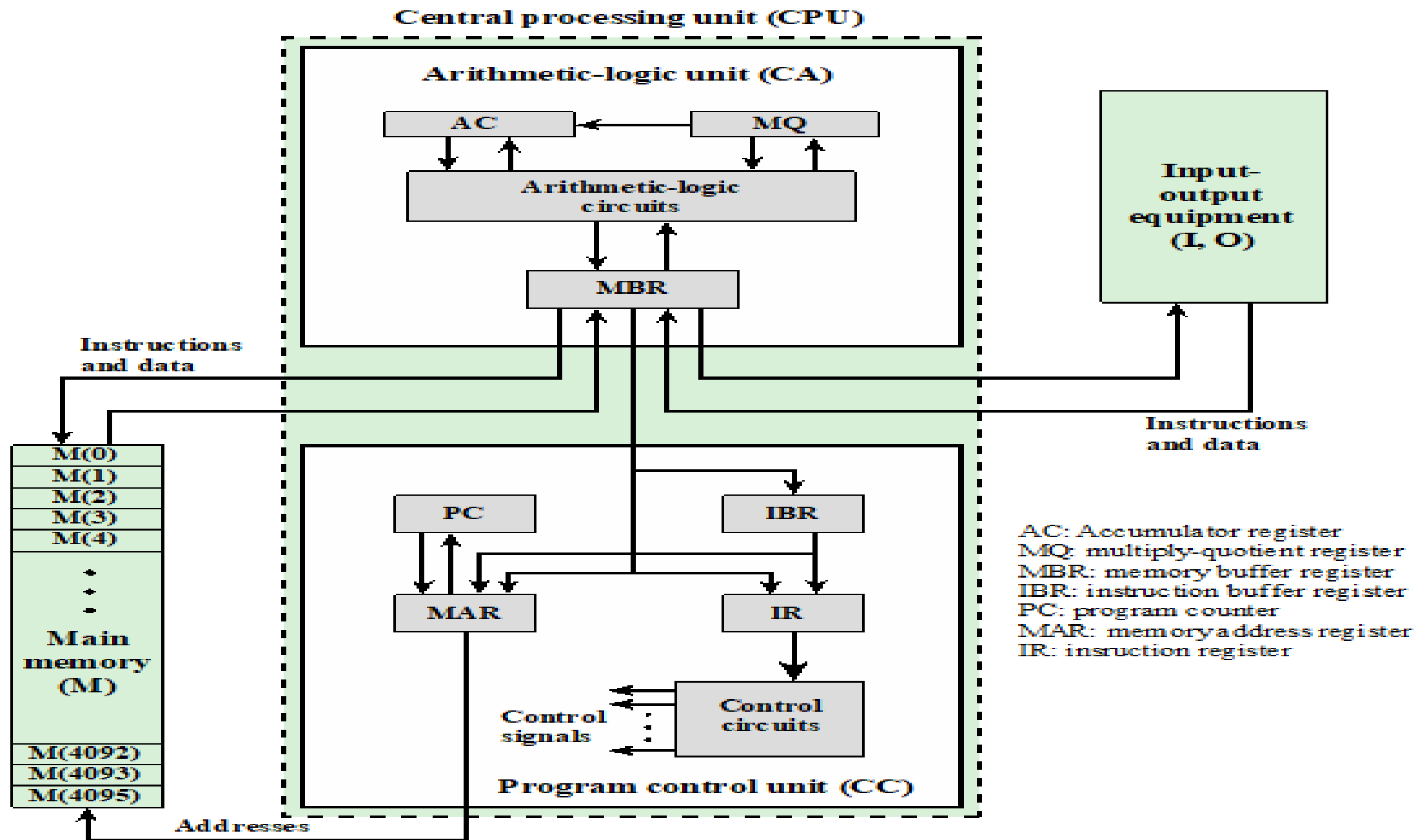
Von Neumann Architecture

- ▶ Contemporary computer designs are based on concepts developed by John Von Neumann at the Institute for Advanced Studies, Princeton
- ▶ Referred to as the *Von Neumann Architecture* and is based on three key concepts:
 - ▶ **Stored Program Architecture**: Data and instructions are stored in a single read-write memory
 - ▶ The contents of this memory are addressable by location, without regard to the type of data contained there
 - ▶ Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next
 - ▶ Example is **IAS Computer** developed by John Von Neumann and group at the **Institute for Advanced Studies**, Princeton

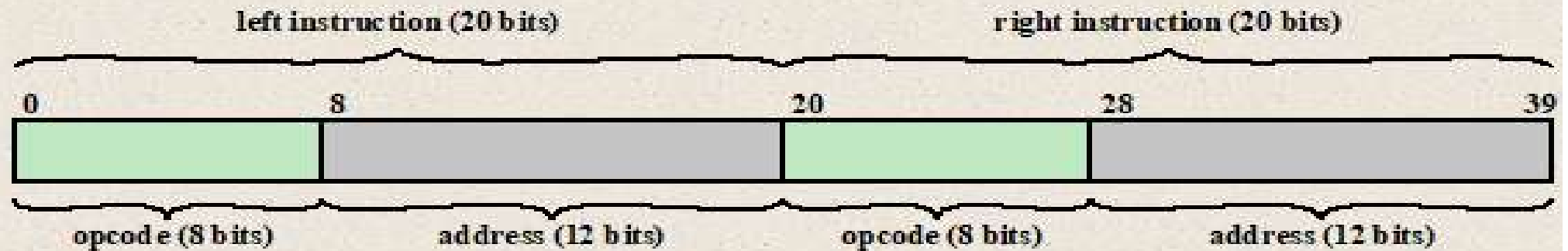
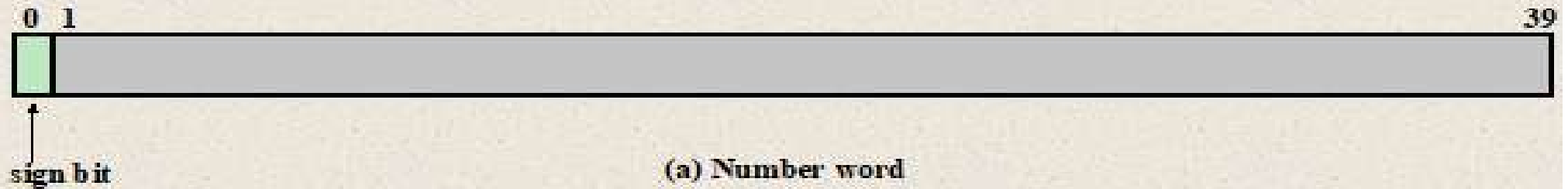


IAS Computer

- ▶ 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
 - ▶ Design began at the Institute for Advanced Studies, Princeton
 - ▶ Completed in 1952
 - ▶ Prototype of all subsequent general-purpose computers



IAS Instruction format



Registers

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)

- Specifies the address in memory of the word to be written from or read into the MBR

Instruction register (IR)

- Contains the 8-bit opcode instruction being executed

Instruction buffer register (IBR)

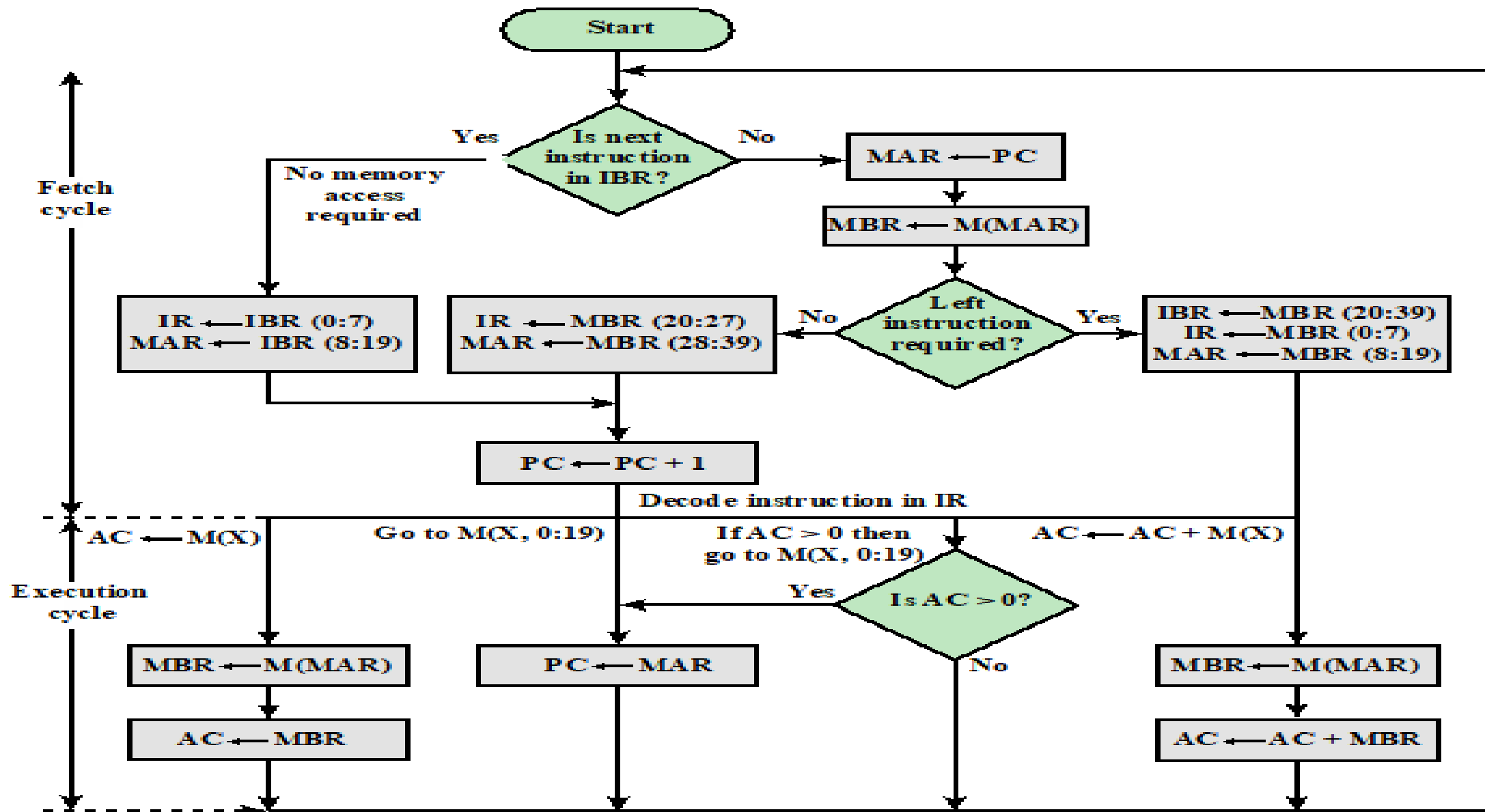
- Employed to temporarily hold temporarily the right-hand instruction from a word in memory

Program counter (PC)

- Contains the address of the next instruction pair to be fetched from memory

Accumulator (AC) and multiplier quotient (MQ)

- Hold operands and results of ALU operations. The most significant 40 bits are stored in the AC and the least significant in the MQ

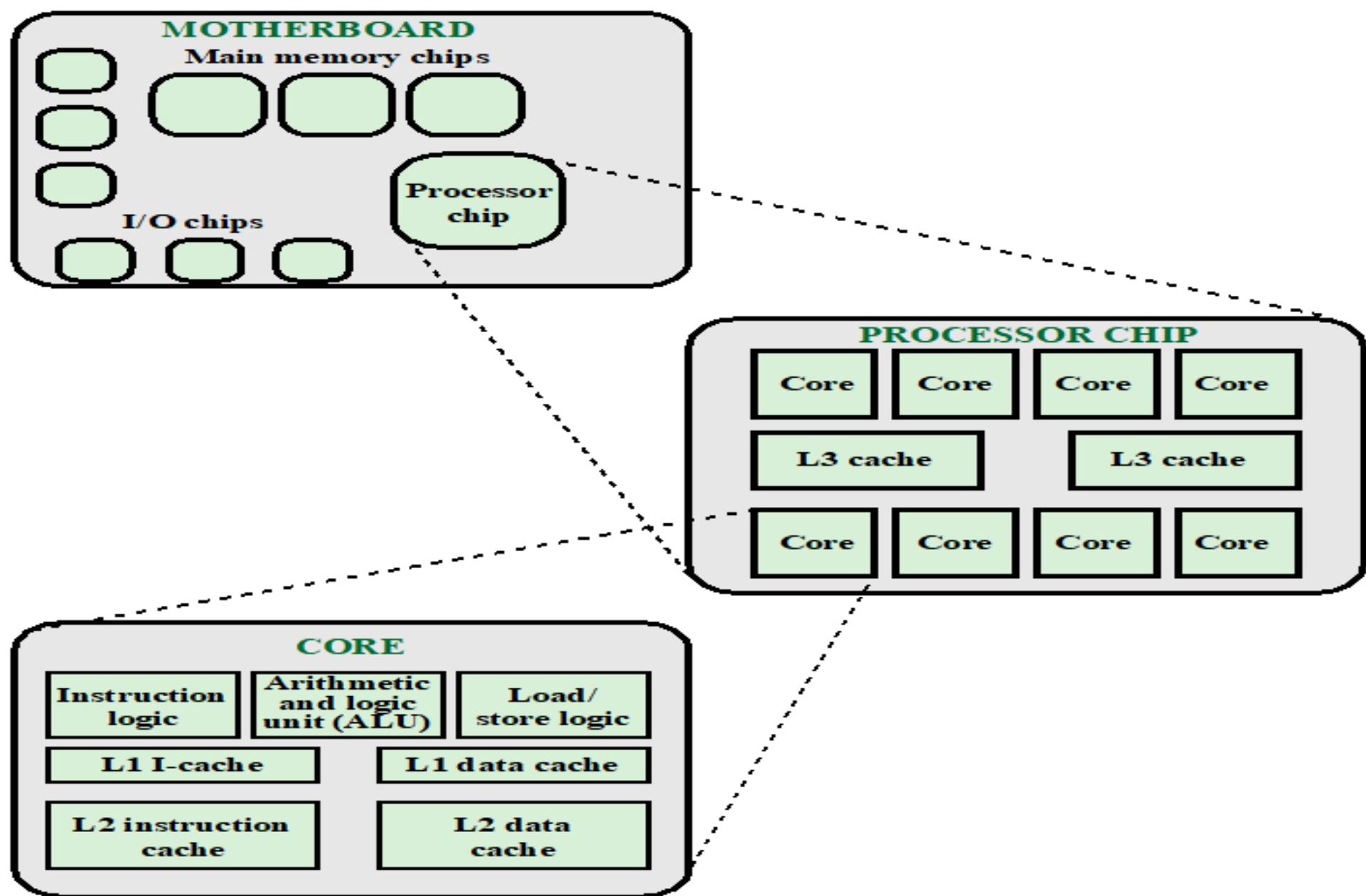


$M(X)$ = contents of memory location whose address is X
(i:j) = bits i through j

Partial Flowchart of IAS Operation

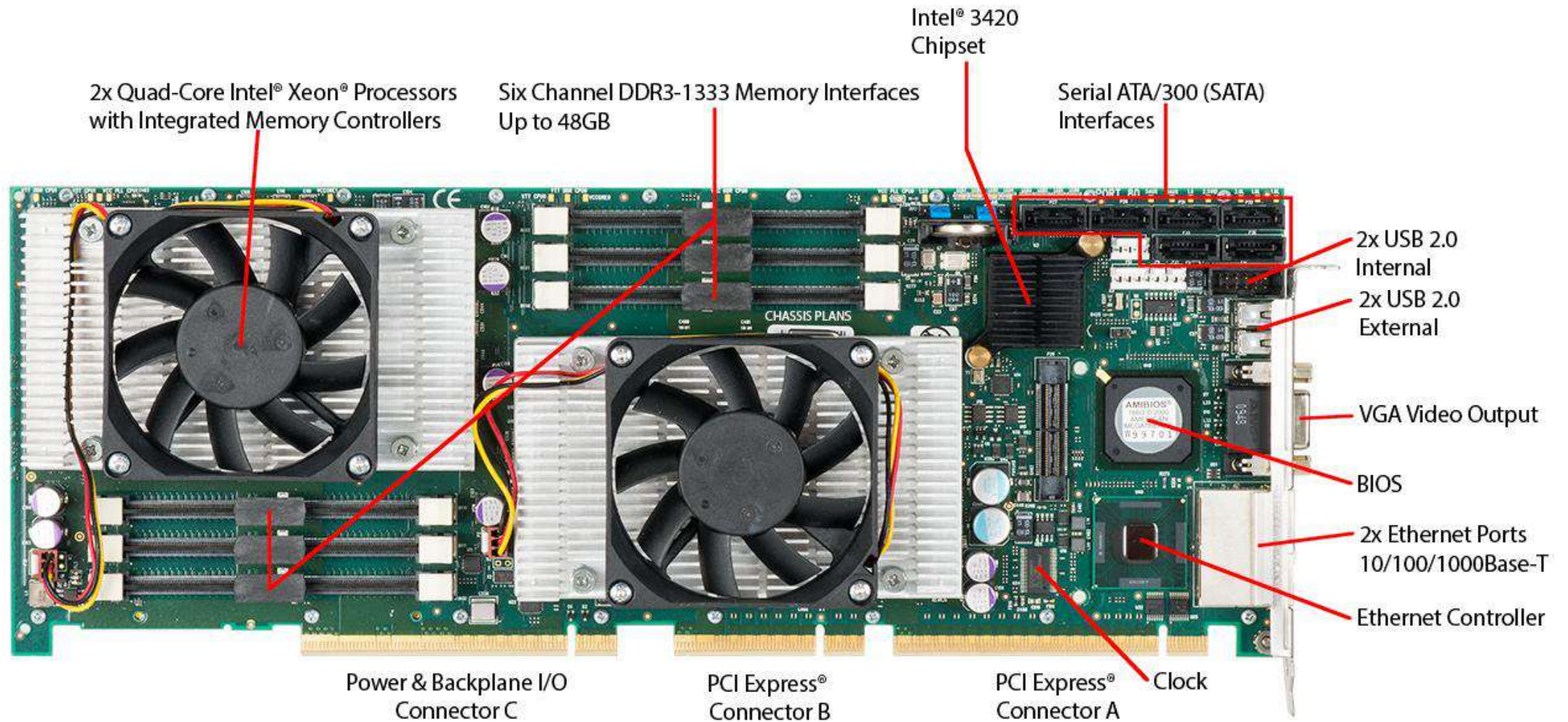
IAS Instruction Set

Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD -M(X)	Transfer -M(X) to the accumulator
	00000011	LOAD M(X)	Transfer absolute value of M(X) to the accumulator
Unconditional branch	00000100	LOAD - M(X)	Transfer - M(X) to the accumulator
	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
Conditional branch	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
Arithmetic		<i>JU</i>	<i>If number in the accumulator is nonnegative, take next instruction from right half of M(X)</i>
		<i>MP</i> + <i>M(X</i> <i>,20:</i> <i>39)</i>	
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD M(X)	Add M(X) to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB M(X)	Subtract M(X) from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; i.e., shift left one bit position
00010101	RSH	Divide accumulator by 2; i.e., shift right one position	
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC



Simplified View of Major Elements of a Multicore Computer

Motherboard with Two Intel Quad-Core Xeon Processors





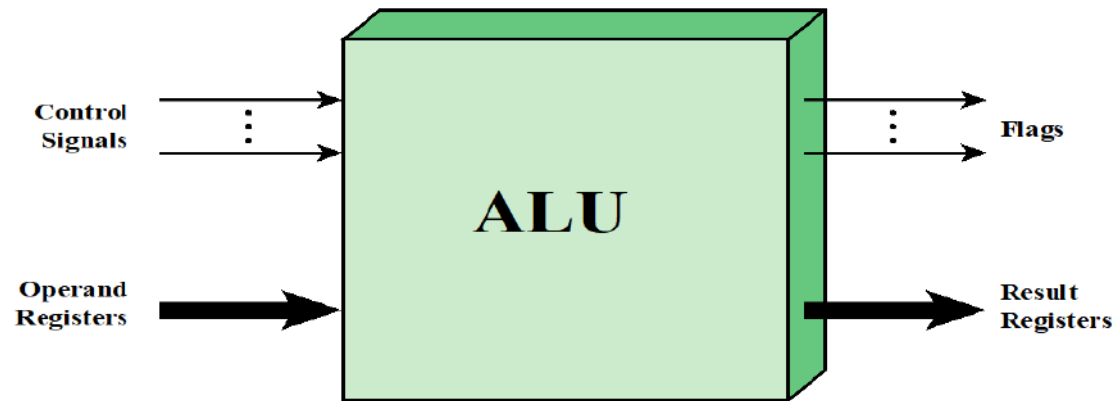
Computer Arithmetic

Overview

- ▶ **Arithmetic Operations**
- ▶ **Binary Arithmetic**
- ▶ **Signed Binary Numbers**
- ▶ **Decimal Arithmetic operation**
- ▶ **Floating point representation**
- ▶ **Floating point Arithmetic**
- ▶ **General Multiplication**
- ▶ **Booth Multiplication**
- ▶ **Array Multiplier**
- ▶ **Division**

Arithmetic & Logic Unit (ALU)

- ▶ Part of the computer that actually performs arithmetic and logical operations on data
- ▶ All of the other elements of the computer system are there mainly to bring data into the ALU for it to process and then to take the results back out
- ▶ Based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations



ALU Inputs and Outputs

Arithmetic Operations

Addition

- ▶ Follow same rules as in decimal addition, with the difference that when sum is 2 indicates a carry (not a 10)
- ▶ Learn new carry rules
 - ▶ $0+0 = \text{sum } 0 \text{ carry } 0$
 - ▶ $0+1 = 1+0 = \text{sum } 1 \text{ carry } 0$
 - ▶ $1+1 = \text{sum } 0 \text{ carry } 1$
 - ▶ $1+1+1 = \text{sum } 1 \text{ carry } 1$

Carry	1	1	1	1	1	0
Augend	0	0	1	0	0	1
Addend	0	1	1	1	1	1
Result	1	0	1	0	0	0

$$\begin{array}{r} 111 \\ 0101 \\ +1011 \\ \hline 10000 \end{array}$$

Carry Values

Subtraction

- ▶ Learn new borrow rules
 - ▶ $0-0 = 1-1 = 0$ borrow 0
 - ▶ $1-0 = 1$ borrow 0
 - ▶ $0-1 = 1$ borrow 1

The rules of the decimal base applies to binary as well. To be able to calculate $0-1$, we have to “borrow one” from the next left digit.

$$\begin{array}{r} 12 \\ 0202 \\ 1010 \\ -0111 \\ \hline 0011 \end{array}$$

Binary Subtraction

- ▶ 1's Complement Method
- ▶ 2's Complement Method

1's Complement Method

Example: $1010100 - 1000100$

1's complement of 1000100 is 0111011

$$\begin{array}{r}
 1010100 \\
 + 0111011 \\
 \hline
 10001111 \\
 \hline
 0010000
 \end{array}$$

+1

If Carry, result is positive.
Add carry to the partial result

Example: $1000100 - 1010100$

1's complement of 1010100 is 0101011

$$\begin{array}{r}
 1000100 \\
 + 0101011 \\
 \hline
 1101111 \\
 = -0010000
 \end{array}$$

If no Carry, result is negative.
Magnitude is 1's complement of the result

Binary Subtraction

- ▶ 1's Complement Method
- ▶ 2's Complement Method

2's Complement Method

Example: $1010100 - 1000100$

2's complement of 1000100 is 0111100

If Carry, result is positive.
Discard the carry

$$\begin{array}{r} 1010100 \\ + 0111100 \\ \hline 10010000 \end{array}$$

$$0010000$$

Example: $1000100 - 1010100$

2's complement of 1010100 is 0101100

If no Carry, result is negative.
Magnitude is 2's complement of the result

$$\begin{array}{r} 1000100 \\ + 0101100 \\ \hline 1110000 \\ = -0010000 \end{array}$$

Signed Binary Numbers

- ▶ When a signed binary number is positive
 - The MSB is '0' which is the sign bit and rest bits represents the magnitude
- ▶ When a signed binary number is negative
 - The MSB is '1' which is the sign bit and rest of the bits may be represented by three different ways
 - ❖ Signed magnitude representation
 - ❖ Signed 1's complement representation
 - ❖ Signed 2's complement representation

Signed Binary Numbers

	<u>-9</u>	<u>+9</u>
Signed magnitude representation	1 1001	0 1001
Signed 1's complement representation	1 0110	0 1001
Signed 2's complement representation	1 0111	0 1001
	<u>-0</u>	<u>+0</u>
Signed magnitude representation	1 0000	0 0000
Signed 1's complement representation	1 1111	0 0000
Signed 2's complement representation	-None-	0 0000

Range of Binary Number

Binary Number of n bits

- ▶ General binary number: $(2^n - 1)$
- ▶ Signed magnitude binary number: $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$
- ▶ Signed 1's complement binary number: $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$
- ▶ Signed 2's complement binary number: $-(2^{n-1})$ to $+(2^{n-1} - 1)$

Signed Binary Number Arithmetic

- ▶ Add or Subtract two signed binary number including its sign bit either signed 1's complement method or signed 2's complement method
- ▶ The 1's complement and 2's complement rules of general binary number is applicable to this
- It is important to decide how many bits we will use to represent the number
- Example: Representing +5 and -5 on 8 bits:
 - +5: 00000101
 - -5: 10000101
- *So, the very first step we have to decide on the number of bits to represent number*

Decimal Subtraction

- ▶ 9's Complement Method
- ▶ 10's Complement Method

9's Complement Method

Example: $72532 - 3250$

9's complement of 03250 is

$$99999 - 03250 = 96749$$

If Carry, result is positive.
Add carry to the partial result

$$\begin{array}{r} 72532 \\ + 96749 \\ \hline 169281 \\ \rightarrow +1 \\ \hline 69282 \end{array}$$

Example: $3250 - 72532$

9's complement of 72532 is

$$99999 - 72532 = 27467$$

If no Carry, result is negative.
Magnitude is 9's complement of the result

$$\begin{array}{r} 03250 \\ + 27467 \\ \hline 30717 \\ \rightarrow \\ = -69282 \end{array}$$

Decimal Subtraction

- ▶ 9's Complement Method
- ▶ 10's Complement Method

Example: $72532 - 3250$

10's complement of 03250 is

$$100000 - 03250 = 96750$$

10's Complement Method

If Carry, result is positive.
Discard the carry

$$\begin{array}{r} 72532 \\ + 96750 \\ \hline \end{array}$$

$$\mathbf{1}69282$$

Result is 69282

Example: $3250 - 72532$

10's complement of 72532 is

$$100000 - 72532 = 27468$$

If no Carry, result is negative.
Magnitude is 10's complement of the result

$$\begin{array}{r} 03250 \\ + 27468 \\ \hline \end{array}$$

$$30718$$

$$= -69282$$

BCD Addition Rules

BCD addition

Add two numbers as same as binary addition

Case 1: If the result is less than or equals to 9 and carry is zero then it is valid BCD.

Case 2: If result is greater than 9 and carry is zero then add 6 in four bit combination.

Case 3: If result is less than or equals to 9 but carry is 1 then add 6 in four bit combination.

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

Comparing Binary and BCD Sums

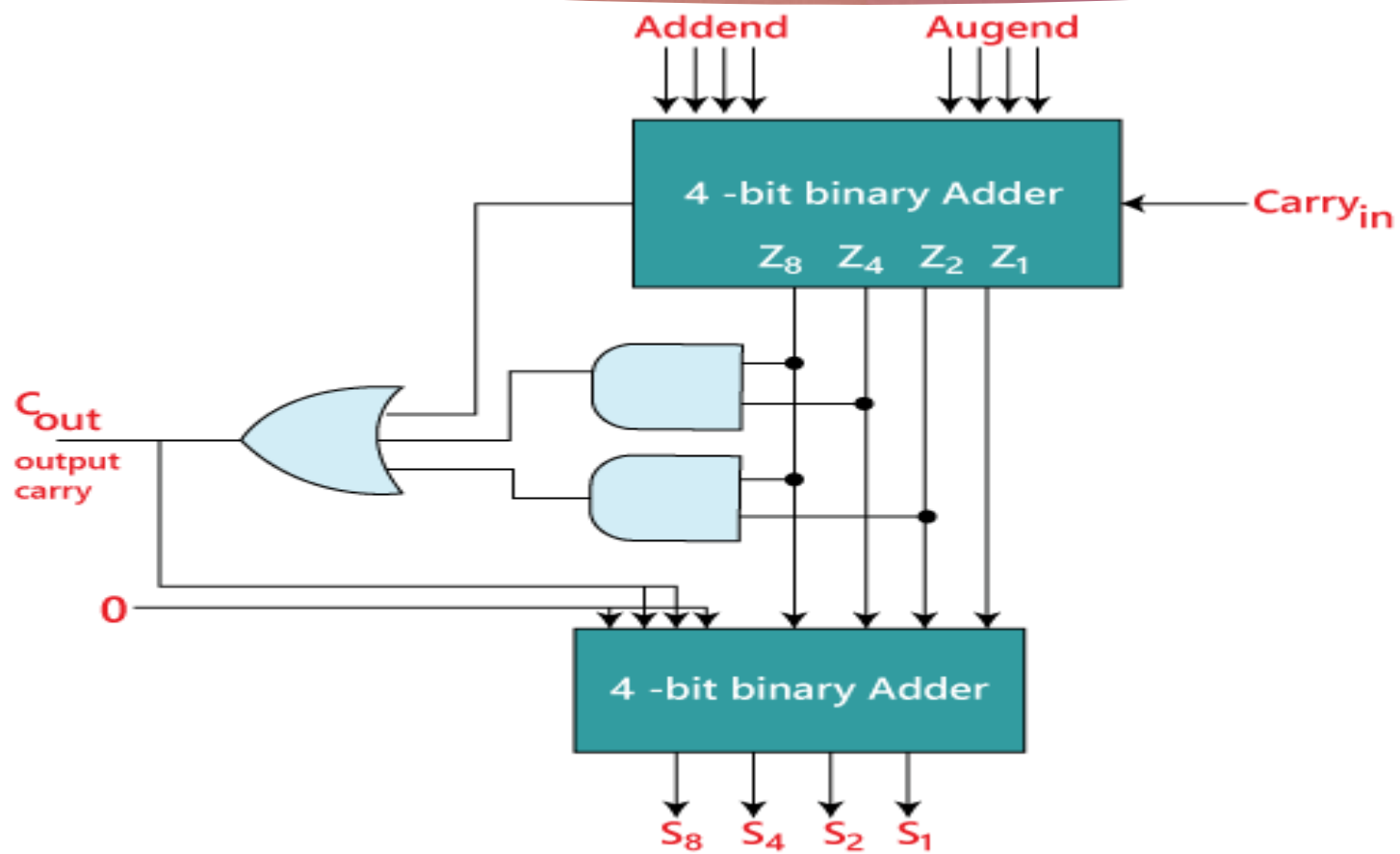
Binary Sum						BCD Sum						Decimal
K	Z ₈	Z ₄	Z ₂	Z ₁		C	S ₈	S ₄	S ₂	S ₁		
0	0	0	0	0	S A M E C O D E	0	0	0	0	0		0
0	0	0	0	1		0	0	0	0	1		1
0	0	0	1	0		0	0	0	1	0		2
.
.
.
.
0	1	0	0	0		0	1	0	0	0		8
0	1	0	0	1		0	1	0	0	1		9
10 to 19 Binary and BCD codes are not the same												
0	1	0	1	0		1	0	0	0	0		10
0	1	0	1	1		1	0	0	0	1		11
0	1	1	0	0		1	0	0	1	0		12
0	1	1	0	1		1	0	0	1	1		13
0	1	1	1	0		1	0	1	0	0		14
0	1	1	1	1		1	0	1	0	1		15
1	0	0	0	0		1	0	1	1	0		16
1	0	0	0	1		1	0	1	1	1		17
1	0	0	1	0		1	1	0	0	0		18
1	0	0	1	1		1	1	0	0	1		19

BCD Adder

- ▶ In the previous table Decimal sum from **0 to 9**, the Binary sum same as BCD sum. So, no conversion is needed.
- ▶ Apply correction if the Decimal sum is between **10-19**.
 - ❖ The correction is needed (Decimal sum **16-19**) when the binary sum has an output carry $K = 1$
 - ❖ The correction is needed (Decimal sum **10-15**) when $Z_8 = 1$ and either $Z_4 = 1$ or $Z_2 = 1$.
- ▶ So, the condition for a correction and an output carry can be expressed by the Boolean function:

$$C = K + Z_8Z_4 + Z_8Z_2$$

- ▶ When $C = 1$, it is necessary to add 0110 to the binary sum to get BCD sum and provide an output carry for the next stage.



Block diagram of a BCD adder

Cascading of BCD Adders

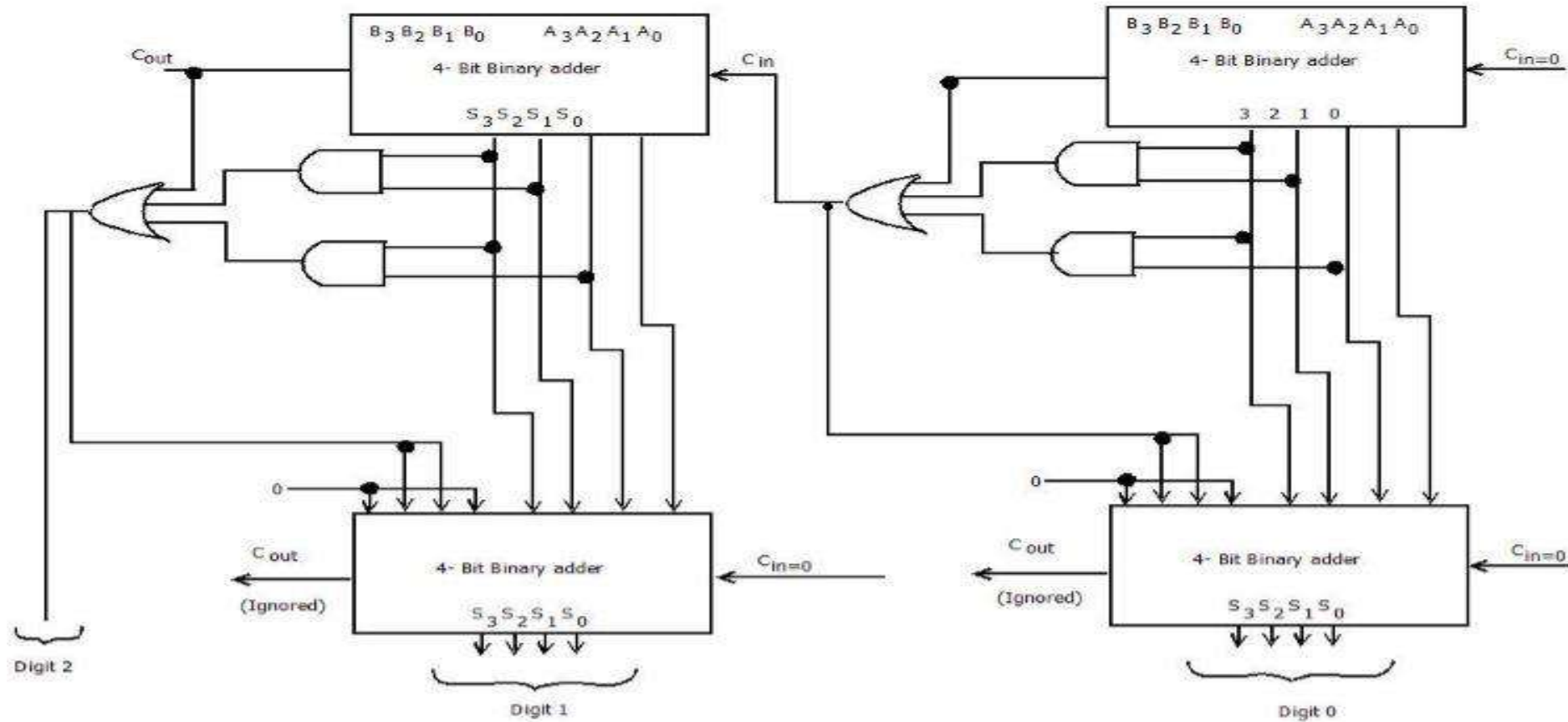


Fig : 8 - Bit BCD Adder

BCD Subtraction Rules

Let two BCD numbers are A and B.
B to be subtracted from A.

RULES:

- Add 9's Complement of B to A
- If result > 9 , Correct by adding 0110
- If carry is generated at most significant position then the result is positive and the End around carry must be added
- If carry is not generated at most significant position then the result is negative and the result is 9's complement of original result

BCD number (d)	Binary equivalent of BCD number				9's complement of BCD (9 - d)	Binary equivalent of 9's complement number			
	w	x	y	z		C ₃	C ₂	C ₁	C ₀
0	0	0	0	0	9	1	0	0	1
1	0	0	0	1	8	1	0	0	0
2	0	0	1	0	7	0	1	1	1
3	0	0	1	1	6	0	1	1	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	4	0	1	0	0
6	0	1	1	0	3	0	0	1	1
7	0	1	1	1	2	0	0	1	0
8	1	0	0	0	1	0	0	0	1
9	1	0	0	1	0	0	0	0	0

Example

Regular Subtraction

(a)

$$\begin{array}{r} 8 \\ - 2 \\ \hline 6 \end{array}$$

(b)

$$\begin{array}{r} 28 \\ - 13 \\ \hline 15 \end{array}$$

(c)

$$\begin{array}{r} 18 \\ - 24 \\ \hline -6 \end{array}$$

9's Complement Subtraction

(a)

$$\begin{array}{r} 8 \\ + 7 \text{ 9's complement of 2} \\ \hline 5 \\ \text{\textcircled{1}} \downarrow + 1 \text{ Add carry to result} \\ \hline 6 \end{array}$$

(b)

$$\begin{array}{r} 28 \\ + 86 \text{ 9's complement of 13} \\ \hline 14 \\ \text{\textcircled{1}} \downarrow + 1 \text{ Add carry to the result} \\ \hline 15 \end{array}$$

(c)

$$\begin{array}{r} 18 \\ + 75 \text{ 9's complement of 24} \\ \hline 93 \text{ 9's complement of result} \\ \downarrow \text{(No carry indicates that the} \\ \text{answer is negative and in} \\ \text{complement form)} \\ -06 \end{array}$$

E.G. $8 - 3 = 8 + [9\text{'s COMP. OF } 3]$
 $= 8 + 6$

$$\begin{array}{r} 1000 \\ 0110 \\ \hline 1110 \leftarrow \text{INVALID } (>9) \\ 0110 \leftarrow \text{CORRECTION} \\ \text{\textcircled{1}} 0100 \\ \text{\textcircled{1}} \rightarrow 1 \leftarrow \text{END AROUND CARRY} \\ \hline 0101 = 5 \end{array}$$

(b) $3 - 8 = -5$

$$\begin{array}{r} 0011 \\ 0001 \\ \hline 0100 \end{array}$$

NO CARRY >>> NEGATIVE
 9's COMP. OF 0100 = 0101 = -5

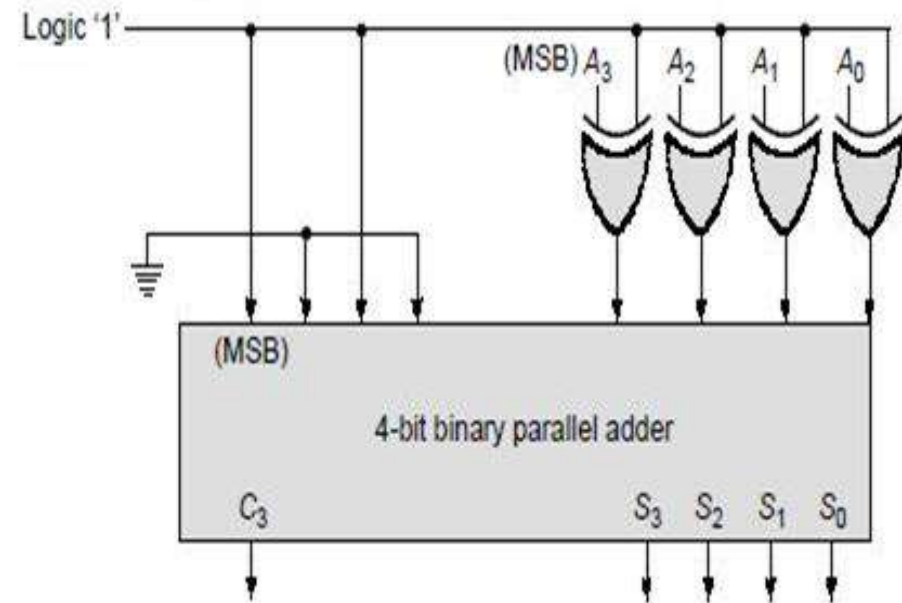
(c) $87 - 39 >>> 87 + [9\text{'s COMP OF } 39]$

$$\begin{array}{r} 87 \quad 1000 \quad 0111 \\ 60 \quad 0110 \quad 0000 \\ \hline \text{INVALID} \rightarrow 1110 \quad 0111 \\ 0110 \\ \text{\textcircled{1}} 0100 \quad 0111 \\ \text{\textcircled{1}} \rightarrow 1 \\ \hline 0100 \quad 1000 \\ = \quad 4 \quad 8 \end{array}$$

9's Complement Circuit

- 9's complement of 2 is 7
- Binary equivalent of 2 is 0010
- 1's complement of 0010 is 1101
- Then,
$$\begin{array}{r} 1101 \\ + 1010 \\ \hline = 0111 \end{array}$$
 which is Binary equivalent of 7
- If carry discard it.

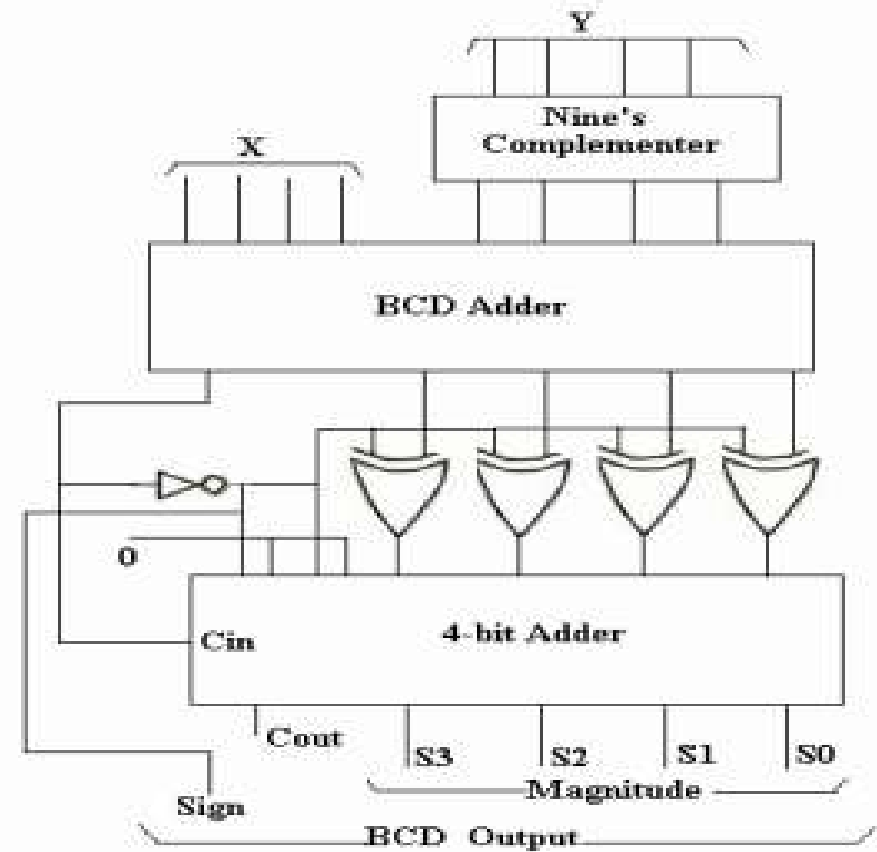
- 9's complement of 3 is 6
- Binary equivalent of 3 is 0011
- 1's complement of 0011 is 1100
- Then,
$$\begin{array}{r} 1100 \\ + 1010 \\ \hline = 0110 \end{array}$$
 which is Binary equivalent of 6
- **If carry discard it.**



BCD Subtractor Circuit

RULES:

- Add 9's Complement of B to A
- If result > 9 , Correct by adding 0110
- If carry is generated at most significant position then the result is positive and the End around carry must be added
- If carry is not generated at most significant position then the result is negative and the result is 9's complement of original result



Floating Point Number

- ▶ Floating point number can be represented as

$$m \times r^e$$

- ▶ m is mantissa, e is exponent and r is radix
- ▶ Let the decimal number is 6132.789, which can be represented as 0.6132789×10^4
- ▶ Let the binary number is 1001.110, which can be represented as 0.1001110×2^4 or can be represented as 1.001110×2^3

Floating Point Arithmetic

► Addition/Subtraction

- Align the radix point first to make the exponent equal before addition or subtraction
- Add or Subtract mantissa
- Normalize the result by adjusting the exponent
- $(A \times E^n) \pm (B \times E^n) = (A \pm B) E^n$

► Multiplication

- $(A \times E^m) \times (B \times E^n) = (A \times B) E^{m+n}$

► Division

- $(A \times E^m) \div (B \times E^n) = (A \div B) E^{m-n}$

Addition (Decimal FP)

- Consider a 4-digit decimal example
 - $9.999 \times 10^1 + 1.610 \times 10^{-1}$
- **1. Align decimal points**
 - Adjust exponent
 - $9.999 \times 10^1 + 0.016 \times 10^1$
- **2. Add mantissa**
 - $9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$
- **3. Normalize result & check for over/underflow**
 - 1.0015×10^2
- **4. Round and renormalize if necessary**
 - 1.002×10^2

Addition (Binary FP)

- Now consider a 4-digit binary example
 - $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + - 0.4375)
- **1. Align binary points**
 - Adjust exponent
 - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- **2. Add mantissa**
 - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- **3. Normalize result & check for over/underflow**
 - $1.000_2 \times 2^{-4}$
- **4. Round and renormalize if necessary**
 - $1.000_2 \times 2^{-4}$ (no change) = 0.0625

Multiplication (Decimal FP)

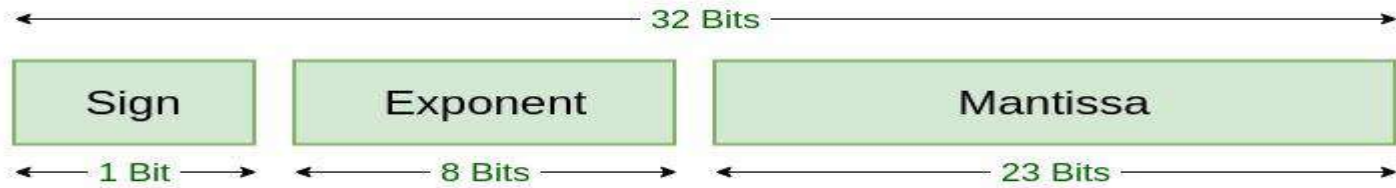
- Consider a 4-digit decimal example
 - $1.110 \times 10^{10} \times 9.200 \times 10^{-5}$
- **1. Add exponents**
 - For biased exponents, subtract bias from sum
 - New exponent = $10 + -5 = 5$
- **2. Multiply mantissa**
 - $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^5$
- **3. Normalize result & check for over/underflow**
 - 1.0212×10^6
- **4. Round and renormalize if necessary**
 - 1.021×10^6
- **5. Determine sign of result from signs of operands**
 - $+1.021 \times 10^6$

Multiplication (Binary FP)

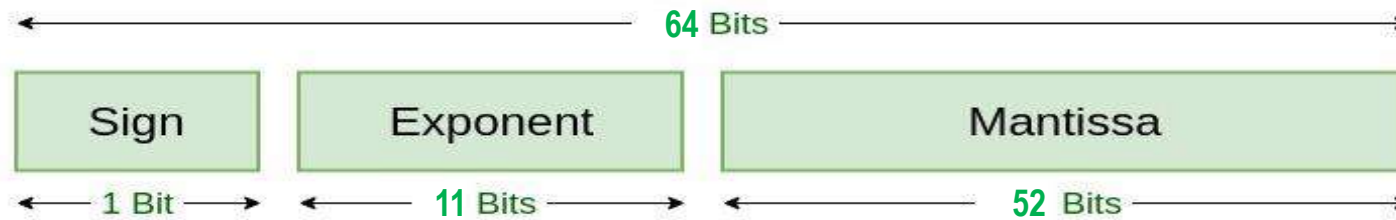
- Now consider a 4-digit binary example
 - $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5×-0.4375)
- **1. Add exponents**
 - Unbiased: $-1 + -2 = -3$
 - Biased: $(-1 + -2 + 127) = -3 + 127$
- **2. Multiply mantissa**
 - $1.000_2 \times 1.110_2 = 1.110_2 \Rightarrow 1.110_2 \times 2^{-3}$
- **3. Normalize result & check for over/underflow**
 - $1.110_2 \times 2^{-3}$
- **4. Round and renormalize if necessary**
 - $1.110_2 \times 2^{-3}$ (no change)
- **5. Determine sign: + ve \times - ve = -ve**
 - $-1.110_2 \times 2^{-3} = -0.21875$

Floating Point Standard

- The IEEE Standard for Floating-Point (IEEE 754) is a technical standard for floating-point representation which was defined in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**.
- Developed in response to divergence of representations
 - Portability issues for scientific code
- Now almost universally adopted
- Two representations
 - Single precision (32-bit)
 - Double precision (64-bit)




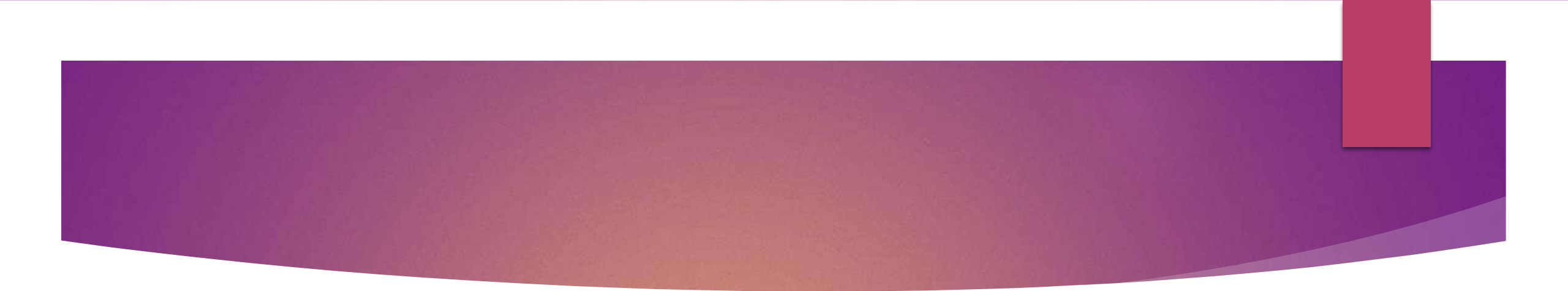
Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

- Normalize significand: $1.0 \leq |\text{significand}| < 2.0$
 - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
 - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
 - Ensures exponent is unsigned
 - Single precision: Bias = 127
 - Double precision: Bias = 1023

- 
- ▶ In the CPU, a 32-bit floating point number is represented using IEEE single precision standard format as follows:
 - ▶ S | EXPONENT | MANTISSA
 - ▶ where S is one bit, the EXPONENT is 8 bits, and the MANTISSA is 23 bits.
 - The *mantissa* represents the leading significant bits in the number.
 - The *exponent* is used to adjust the position of the binary point (like "decimal" point)
 - ▶ The mantissa is said to be normalized when it is expressed as a value between 1 and 2. i.e., the mantissa would be in the form 1.xxxx.

- 
- ▶ The leading integer of the binary representation is not stored. Since it is always a 1, it can be easily restored
 - ▶ The "S" bit is used as a sign bit and indicates whether the value represented is positive or negative
 - ▶ 0 for positive, 1 for negative
 - ▶ If a number is smaller than 1, normalizing the mantissa will produce a negative exponent
 - ▶ But 127 is added to all exponents in the floating point representation, allowing all exponents to be represented by a positive number

Single Precision

- ▶ **Example 1.** Represent the decimal value 2.5 in 32-bit floating point format.

$$2.5 = 10.1b$$

- ▶ In normalized form, this is: 1.01×2^1

- ▶ The mantissa: $M = 010000000000000000000000$

(23 bits without the leading 1)

- ▶ The exponent: $E = 1 + 127 = 128 = 10000000b$

- ▶ The sign: $S = 0$ (the value stored is positive)

- ▶ So, $2.5 = 0 \ 10000000 \ 010000000000000000000000$

Sign Exponent Mantissa

▶ **Example 2:** Represent the number - 0.00010011b in floating point form.

▶ $0.00010011b = 1.0011 \times 2^{-4}$ in normalized form

▶ Mantissa: $M = 001100000000000000000000$

▶ Exponent: $E = -4 + 127 = 123 = 01111011b$

▶ $S = 1$ (as the number is negative)

▶ Result: **1 01111011 001100000000000000000000**

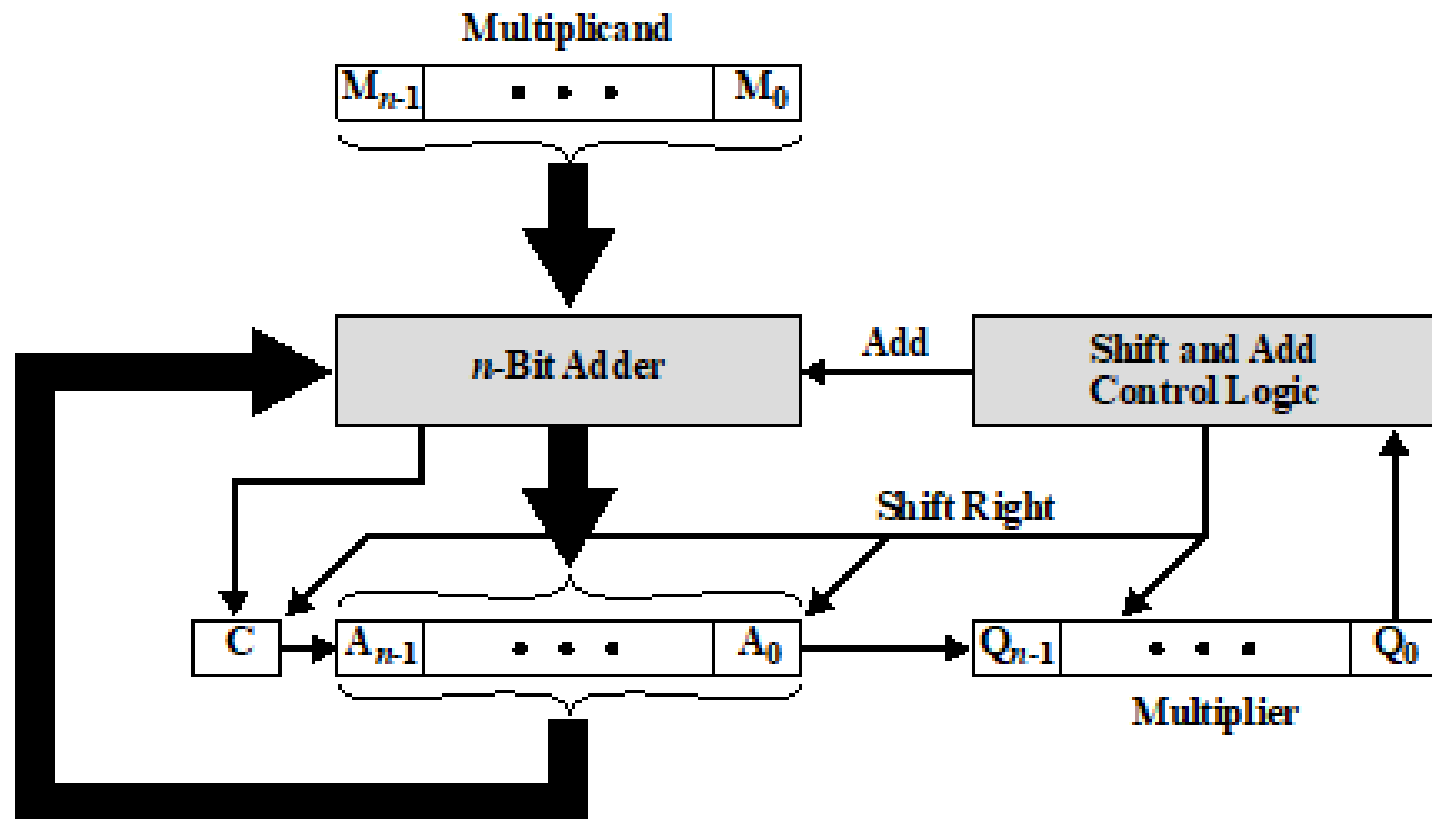
Sign Exponent Mantissa

Multiplication

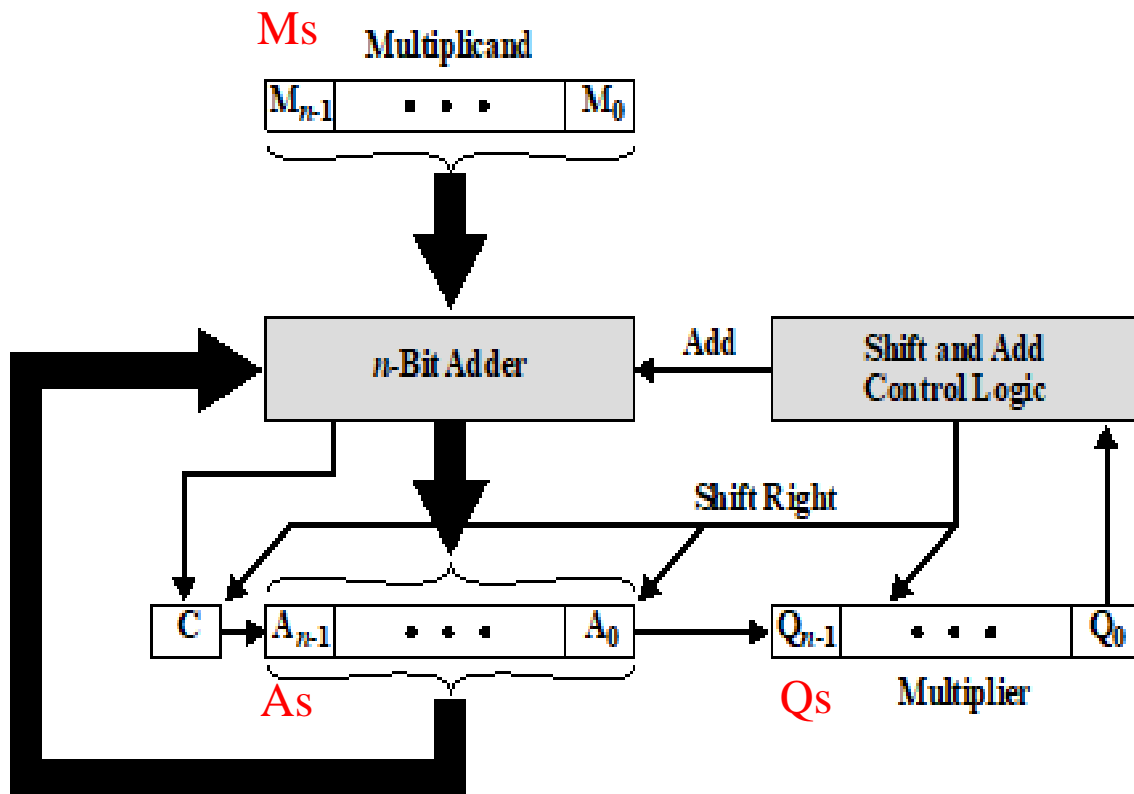
1011	Multiplicand (11)
× 1101	Multiplier (13)
<hr/>	
1011	} Partial products
0000	
1011	
1011	
<hr/>	
10001111	Product (143)

Multiplication of Unsigned Binary Integers

Hardware Diagram



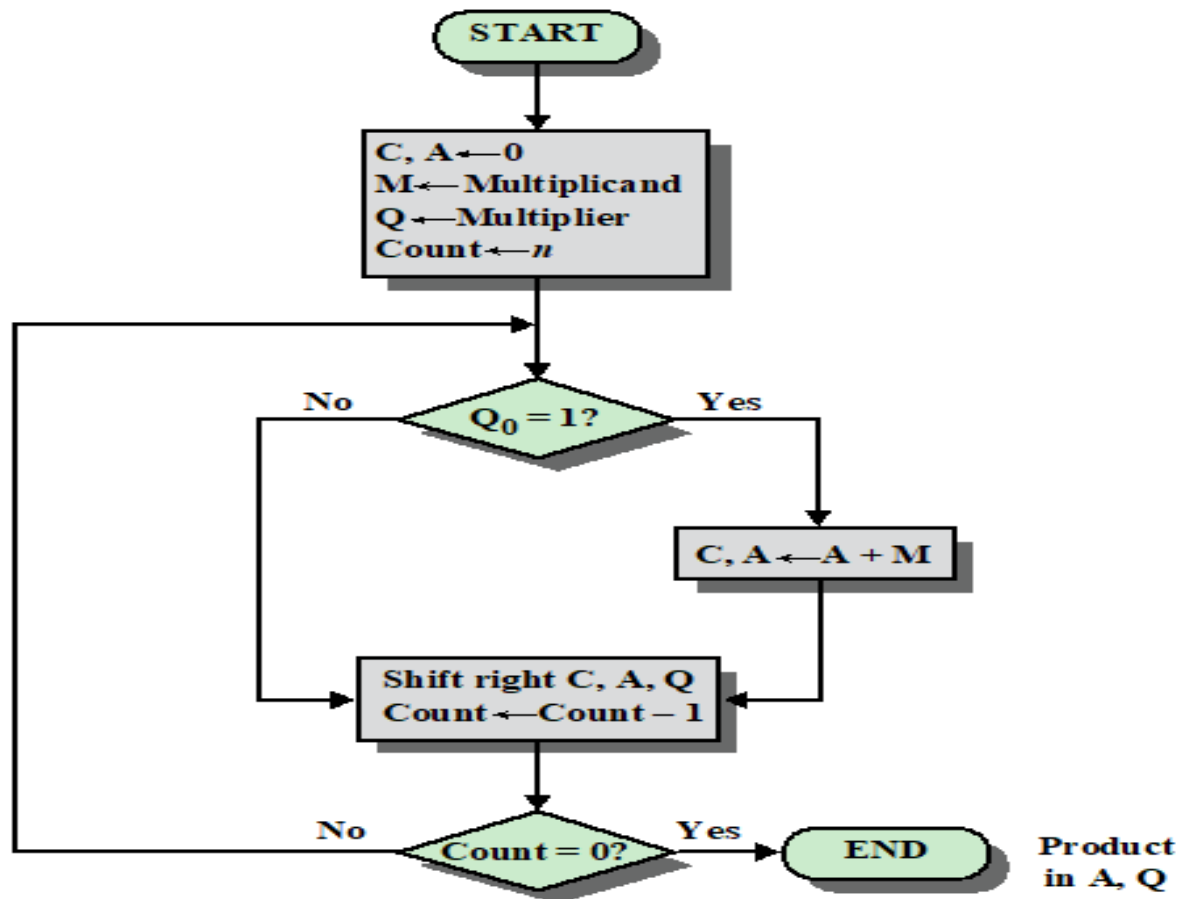
Hardware Diagram



The sign of the product is determined from the signs of the Multiplicand and Multiplier.

- If they are alike, Sign of the product is Positive.
- If they are unlike, Sign of the product is Negative
- So, As will be equal to Ms XOR with Qs

General 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	

Booth Multiplication

- ▶ **Booth's multiplication algorithm** is a multiplication algorithm that multiplies two signed binary numbers in 2's complement notation.
- ▶ The algorithm was invented by Andrew Donald Booth in 1950.
- ▶ It is used to speed up the performance of the multiplication process. It is very efficient too.
- ▶ If string of 0's or string of 1's are there in the multiplier that requires no operation only shift.
- ▶ Consider a general multiplier consisting of a block of 1s surrounded by 0s. For example, 00111110. The product is given by:
$$M \times 00111110 = M \times (2^5 + 2^4 + 2^3 + 2^2 + 2^1) = M \times 62$$
 where, M is the multiplicand
- ▶ The number of operations can be reduced to two by rewriting the same as
$$M \times 00111110 = M \times (2^6 - 2^1) = M \times 62$$
- ▶ This one is Booth Multiplication.

Example

- ▶ Let the multiplication is $M \times +14$

In signed 2's complement representation $+14 = 0\ 000\ 1110$

Which is $M \times 0\ 000\ 1110 = M \times (2^4 - 2^1) = M \times (16 - 2) = M \times +14$

- ▶ Let the multiplication is $M \times -14$

In signed 2's complement representation $-14 = 1\ 111\ 0010$

Which is $M \times 1\ 111\ 0010 = M \times (-2^4 + 2^2 - 2^1) = M \times (-16 + 4 - 2) = M \times -14$

Algorithm

- ▶ As in all multiplication schemes, Booth algorithm also requires examination **of the multiplier bits** from LSB to MSB and shifting of the partial product.
- ▶ Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to following rules:
 - The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier
 - The multiplicand is added to the partial product upon encountering the first 0 (provided that there is a previous '1') in a string of 0's in the multiplier.
 - The partial product does not change when the multiplier bit is identical to the previous multiplier bit, that is string of 0s or string of 1s.

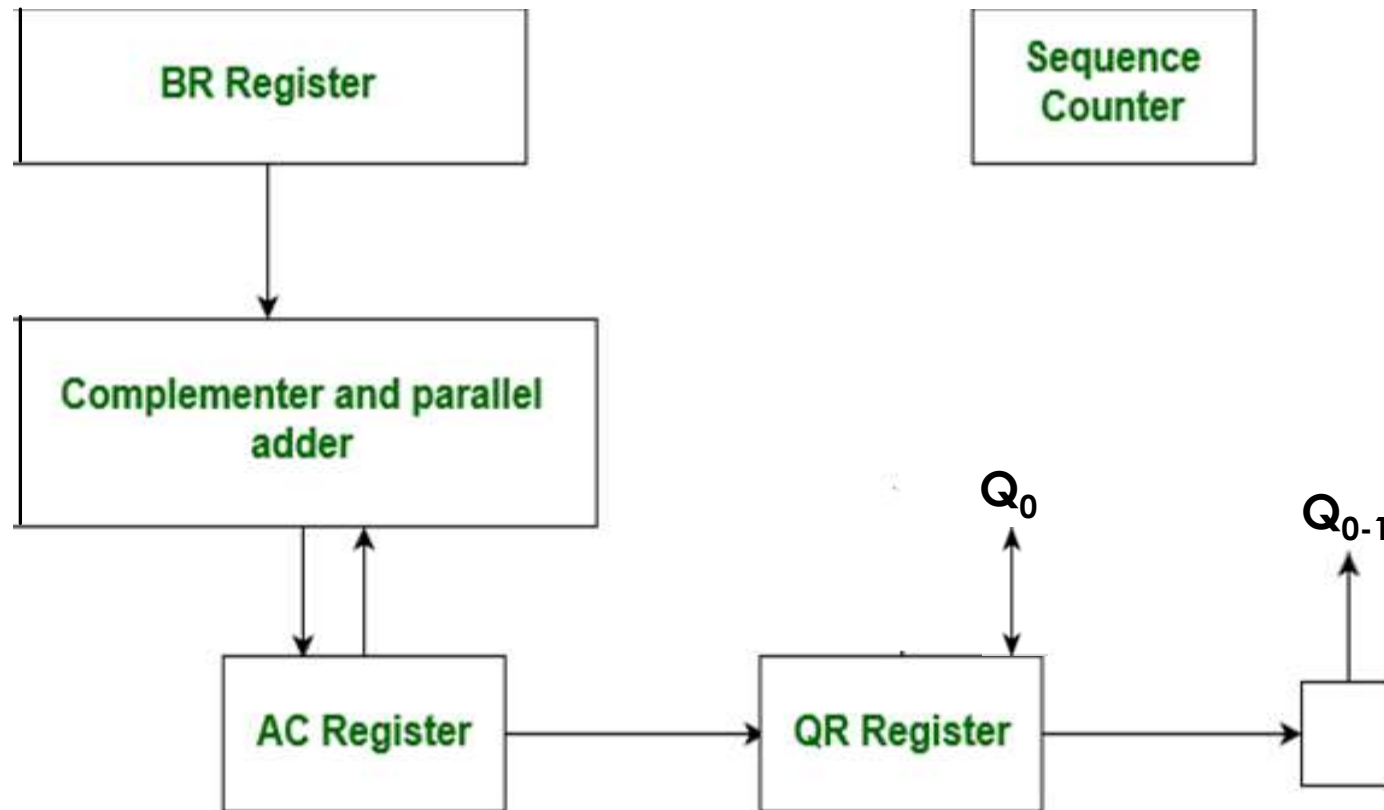
Arithmetic Shift Right

- ▶ In Booth Multiplication Algorithm Shift Right is **Arithmetic shift right**

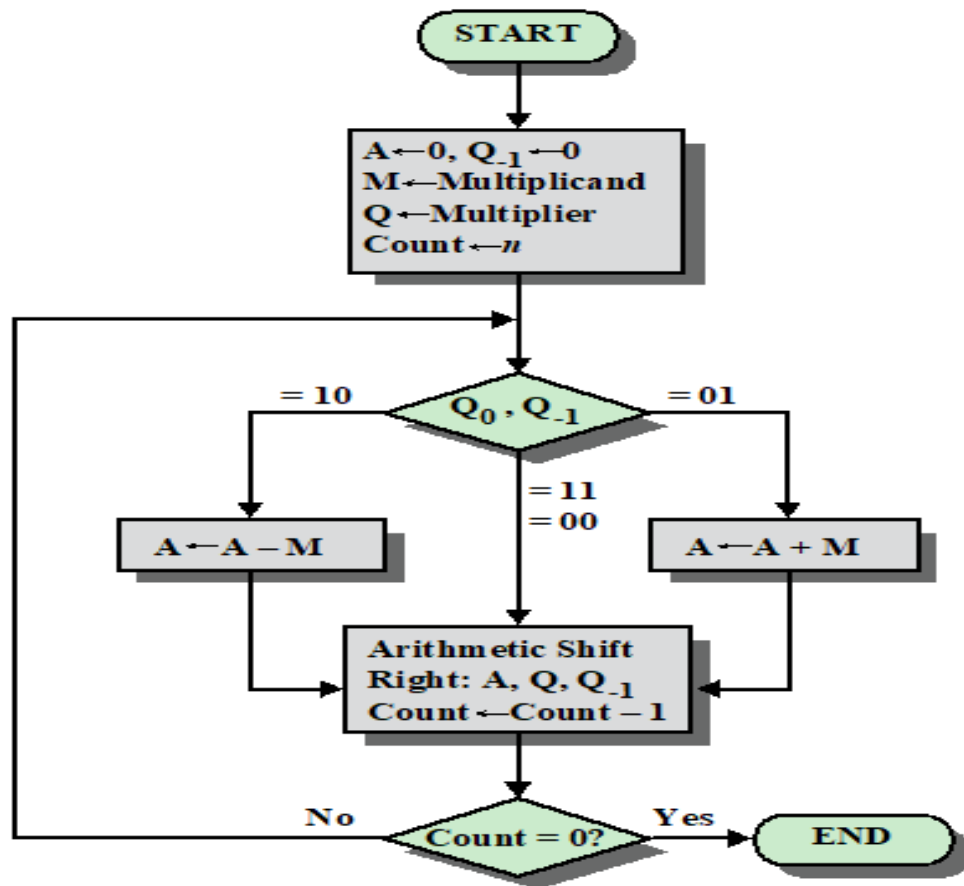
Example:

- ▶ Let the number is 1001
- ▶ Shift right is 0100
- ▶ But, Arithmetic shift right is 1100
- ▶ Let the number is 0101
- ▶ Arithmetic shift right of this number is 0010

Hardware Diagram



Booth Multiplication Algorithm

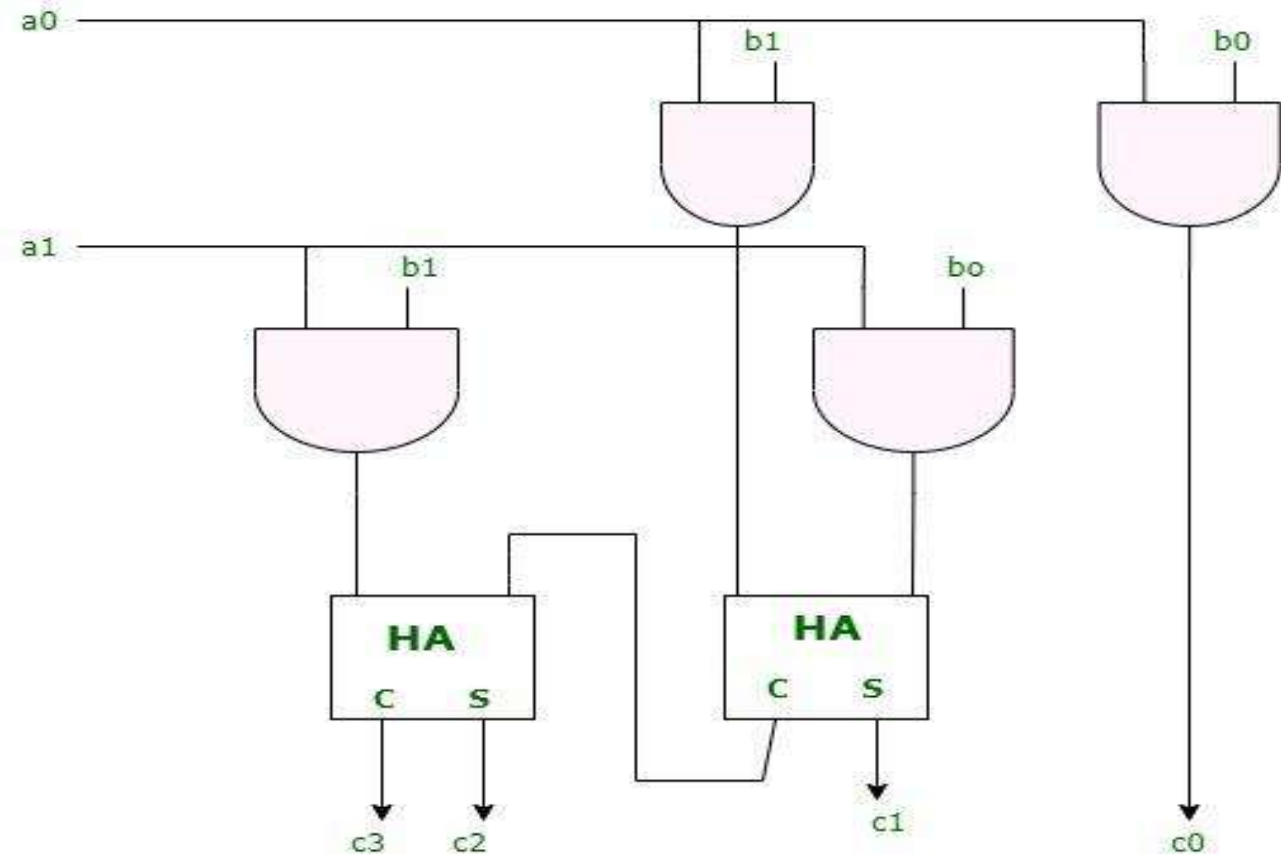


A	Q	Q ₋₁	M	
0000	0011	0	0111	Initial Values
1001	0011	0	0111	A ← A - M } First Cycle
1100	1001	1	0111	
1110	0100	1	0111	Shift } Second Cycle
0101	0100	1	0111	
0010	1010	0	0111	Shift
0001	0101	0	0111	Shift } Fourth Cycle

Example of Booth's Algorithm (7× 3)

Multiplier Design

$$\begin{array}{r} a1\ a0 \\ \times\ b1\ b0 \\ \hline a1b0\ a0b0 \\ a1b1\ a0b1 \\ \hline \end{array}$$



Array Multiplier

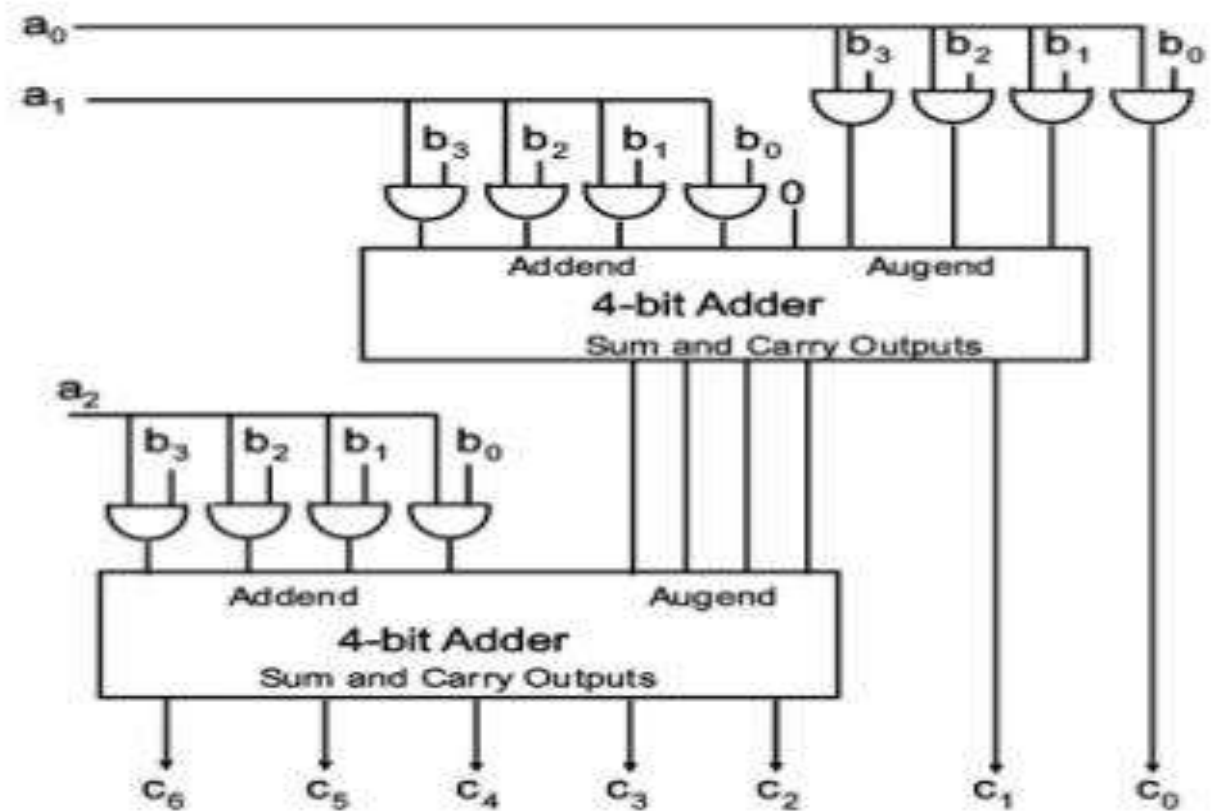
$$\begin{array}{r}
 b_3 \ b_2 \ b_1 \ b_0 \\
 \times a_2 \ a_1 \ a_0 \\
 \hline
 a_0 b_3 \ a_0 b_2 \ a_0 b_1 \ a_0 b_0 \\
 a_1 b_3 \ a_1 b_2 \ a_1 b_1 \ a_1 b_0 \\
 a_2 b_3 \ a_2 b_2 \ a_2 b_1 \ a_2 b_0 \\
 \hline
 c_6 \ c_5 \ c_4 \ c_3 \ c_2 \ c_1 \ c_0
 \end{array}$$

J = Multiplicand

K = Multiplier

AND gate required = JK nos.

(K-1) nos. of J-bit Adder required



Array Multiplier

$$\begin{array}{r}
 m_3 \ m_2 \ m_1 \ m_0 \\
 \times \quad q_2 \ q_1 \ q_0 \\
 \hline
 m_3q_0 \ m_2q_0 \ m_1q_0 \ m_0q_0 \\
 m_3q_1 \ m_2q_1 \ m_1q_1 \ m_0q_1 \\
 m_3q_2 \ m_2q_2 \ m_1q_2 \ m_0q_2 \\
 \hline
 P_6 \ P_5 \ P_4 \ P_3 \ P_2 \ P_1 \ P_0
 \end{array}$$

