

Syllabus

Module 4 Fundamental of Computer Architectures:

- Structure and Function, basic operational concepts of bus structures Memory locations and addresses functions. Types of computer units (CPU, Memory, I/O, System Bus), Von Neumann & Harvard architecture, Key characteristics of RISC & CISC.
- ALU: Multiplication – Block diagram, Hardware implementation of unsigned binary and signed number, Booth's Algorithm, Division Algorithms (Unsigned Binary).
- Processor organization, Register organization- user visible registers, control and status registers- Case Study 8086. Instruction Cycle- The machine cycle and Data flow. Types of operands, Types of operations, Types of instructions, Addressing modes and Instruction Formats- instruction length, allocation of bits, variable length instructions instruction formats - Case Study- 8086. Processor and Instruction Pipelining, Pipeline Performance, Pipeline Hazards - Structural, Data, Control.

Syllabus

List of Assignments:

1. Design and Implement Code Converters using logic gates
2. Design and Implement Combinational Logic Design using MUX/Decoder ICs
3. Design and Implement asynchronous counter using JK- Flip flop.
4. Design and Implement MOD-N asynchronous counter using JK- Flip flop /IC7490.
5. Design and implement Synchronous Counter
6. Design of a sequence detector
7. Write an assembly language program (ALP) to display 2-digit and 4-digit hex numbers using 64- bit assembly language programming
8. Write an assembly language program (ALP) to accept 2-digit and 4-digit hex numbers user input using 64- bit assembly language programming.
9. Write an assembly language program (ALP) to implement addition and subtraction of 8-bit numbers (Using user input, macro and procedure).
10. Write an ALP to perform basic string operations. (length and Reverse)

Fundamental of Computer Architectures

- Computer has two important terminologies associated with it.

- **Computer Architecture**
- **Computer Organization**

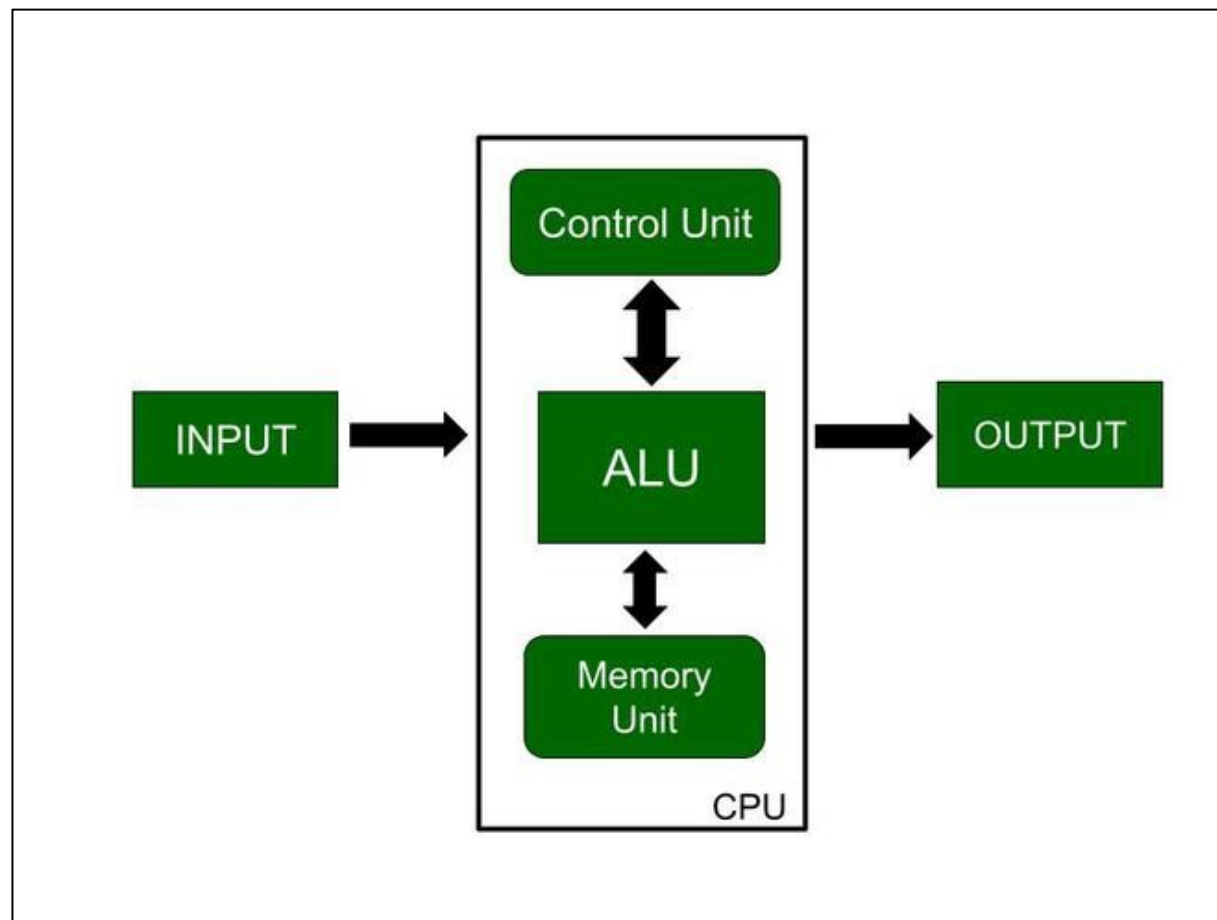
- **Computer Architecture:**

Describes features of a computer family (notably the instructions) and not the specific implementation, just like architecture of a house- **Instruction set, Addressing types.**

- **Computer Organization:**

The internal arrangement of computer, which includes **the design of the processor, memory and input/output circuits.**

Fundamental of Computer Architectures



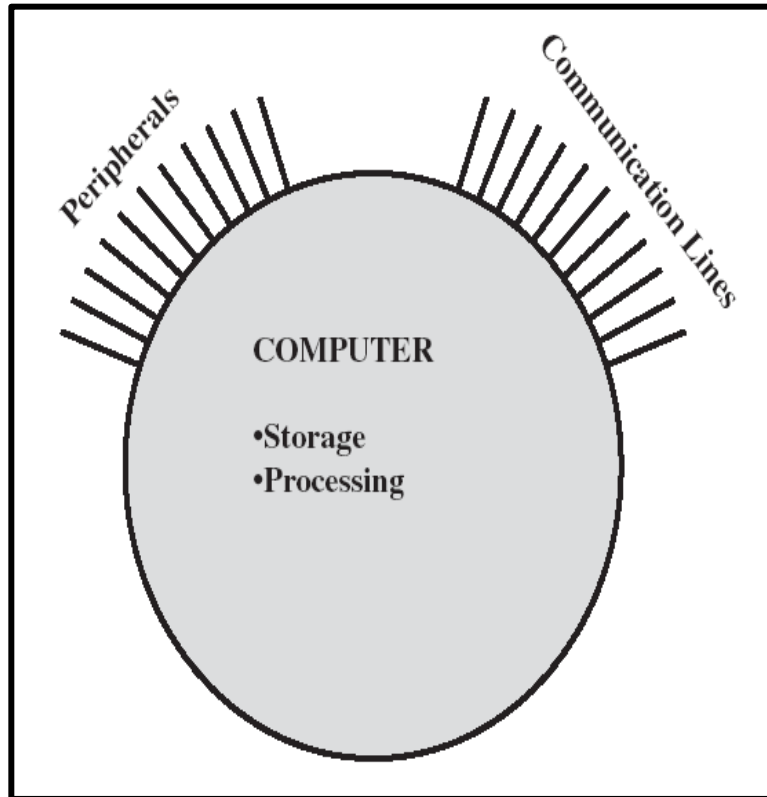
Fundamental of Computer Architectures

- Structure & Function:

- Structure: is the way in which components **relate to each other.** (interrelated)
- Function: is the **operation** of individual components as part of the structure.

Computer Structure

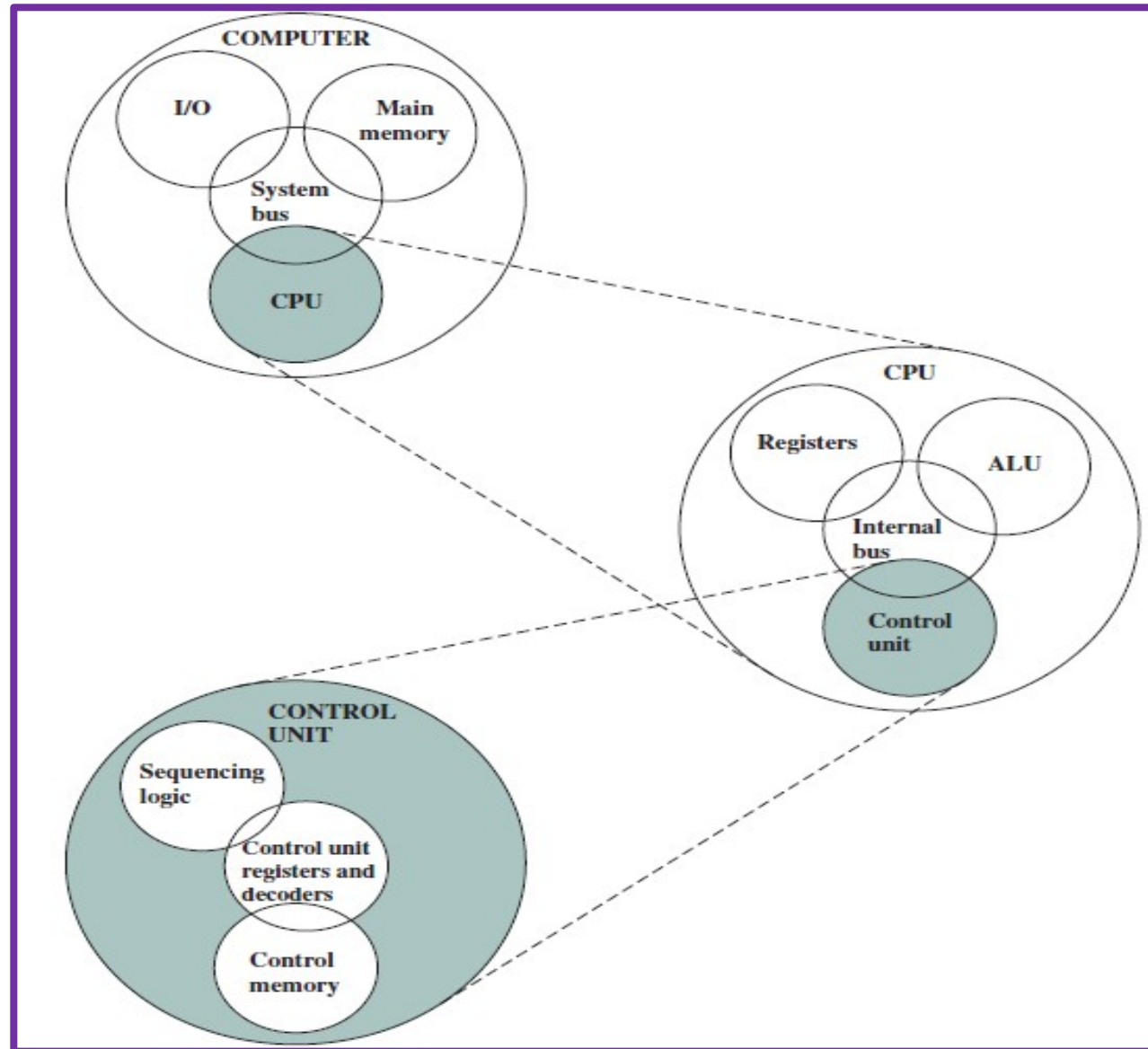
- *Two methods are used in specifying Computer Structure*
- *Top-Down and Down-Top.*



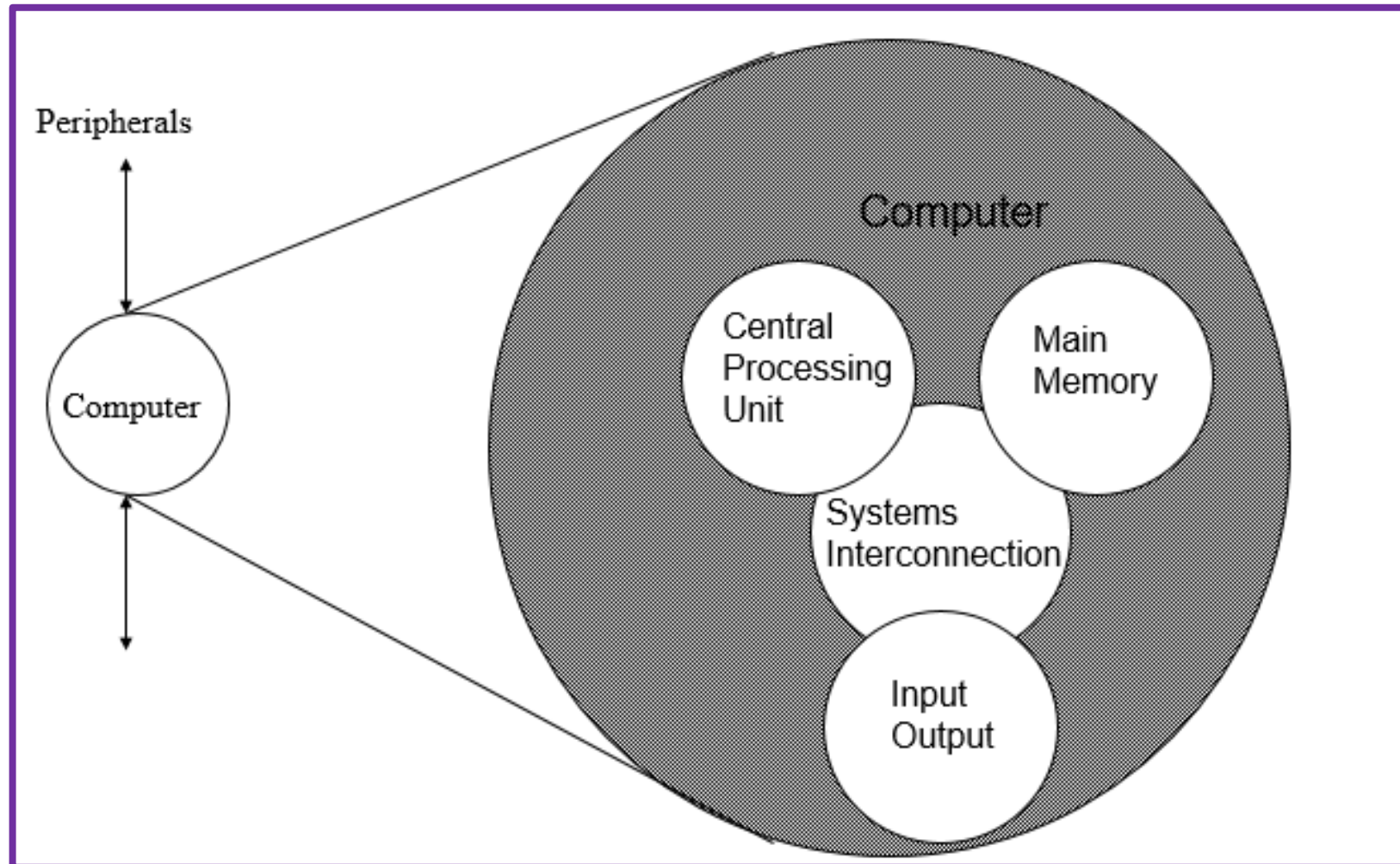
Computer as a Structure includes basic building blocks as:

- **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:** Provided for communication among CPU, main memory, and I/O.

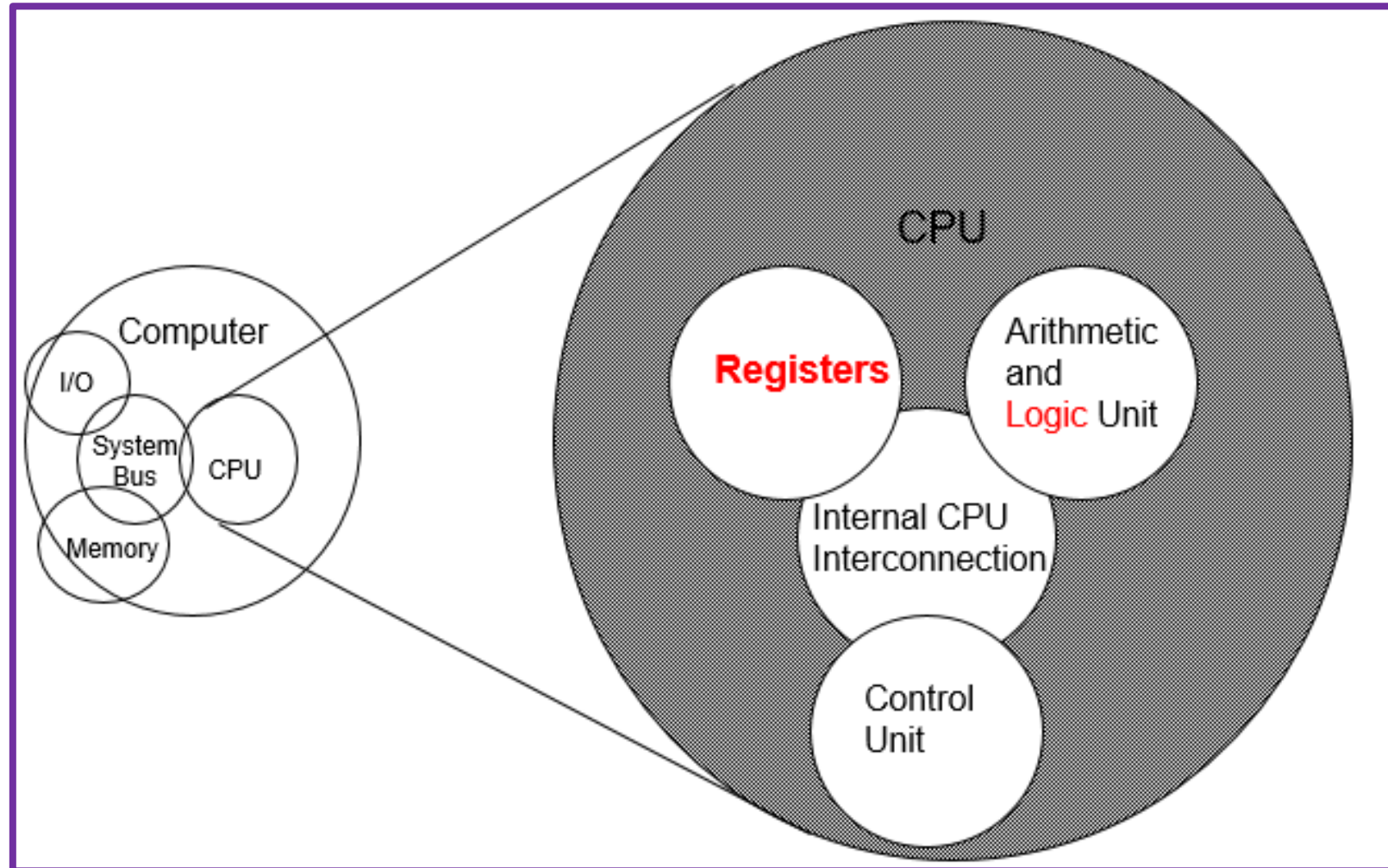
The Computer: Top Down Structure



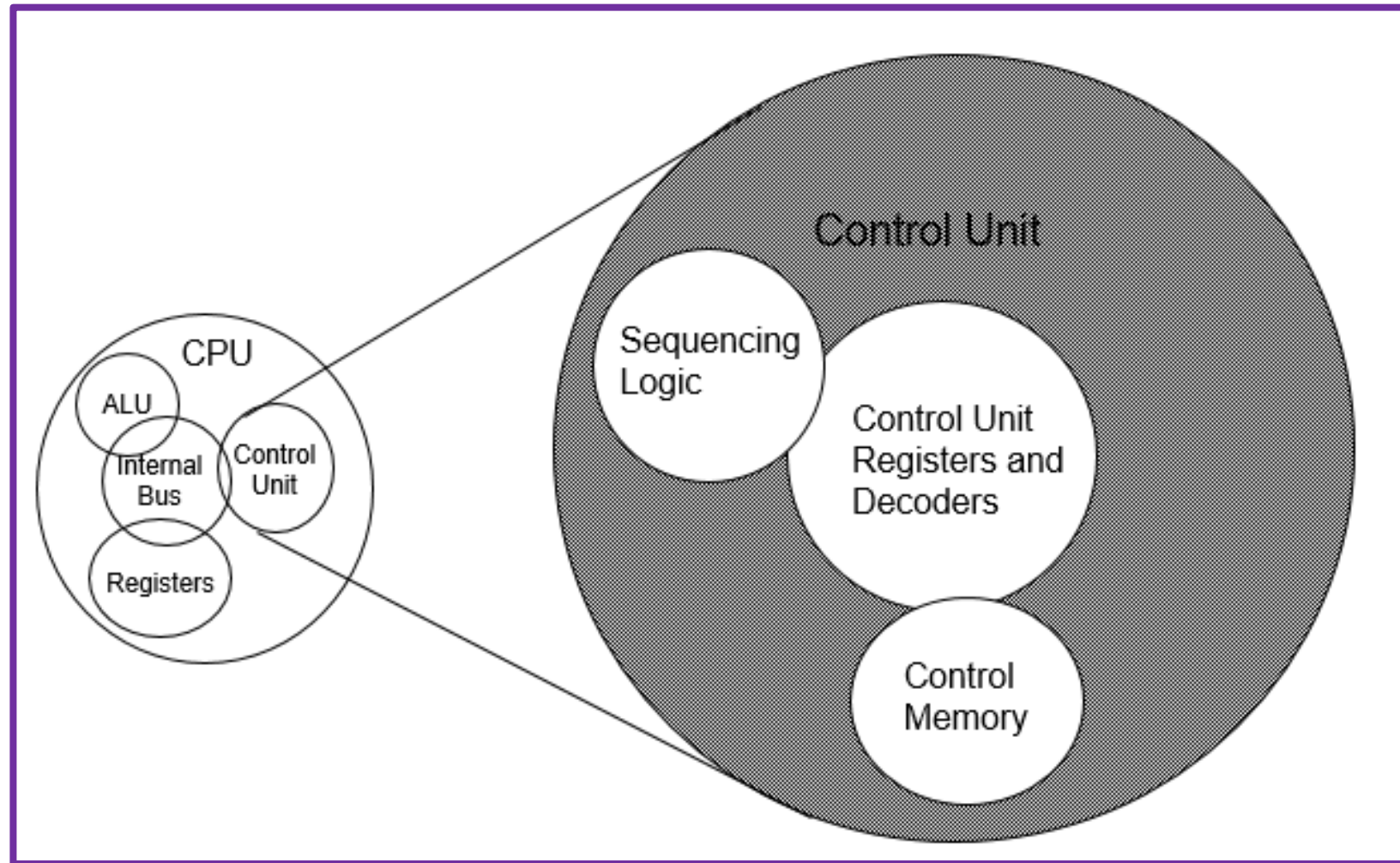
Structure - Top Level



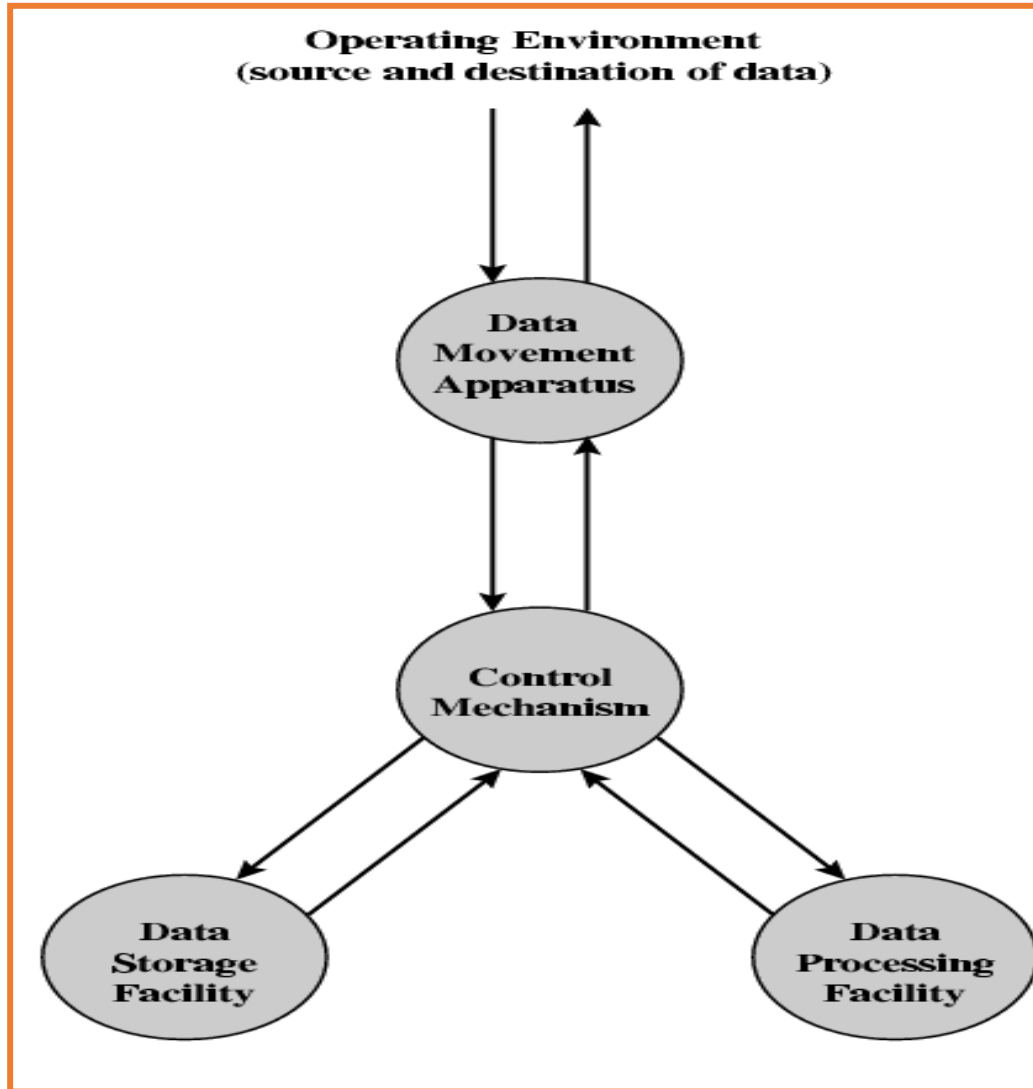
Structure - ALU



Structure – Control Unit

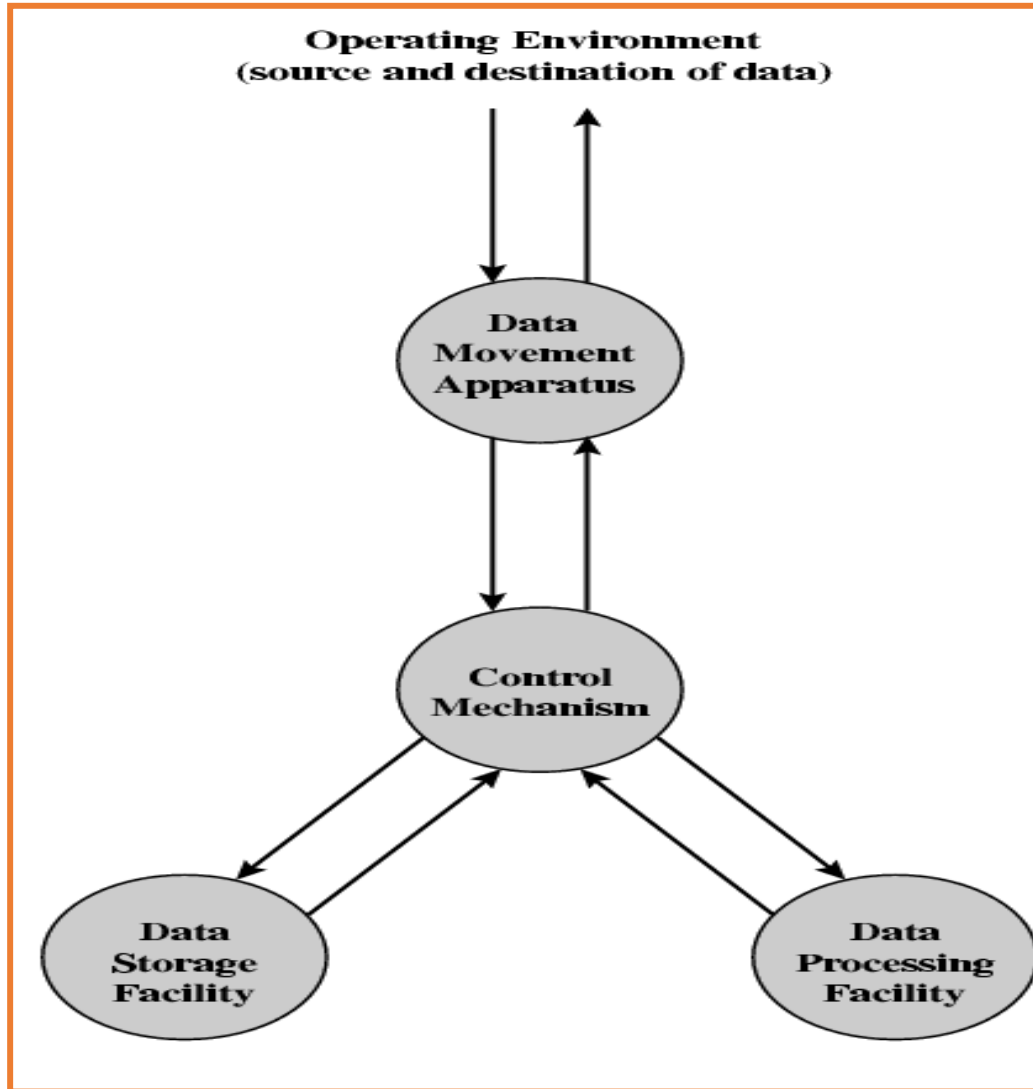


Computer functions



- Computer functions are:
 - Data movement
 - Data storage
 - Data processing
 - Control

Computer functions

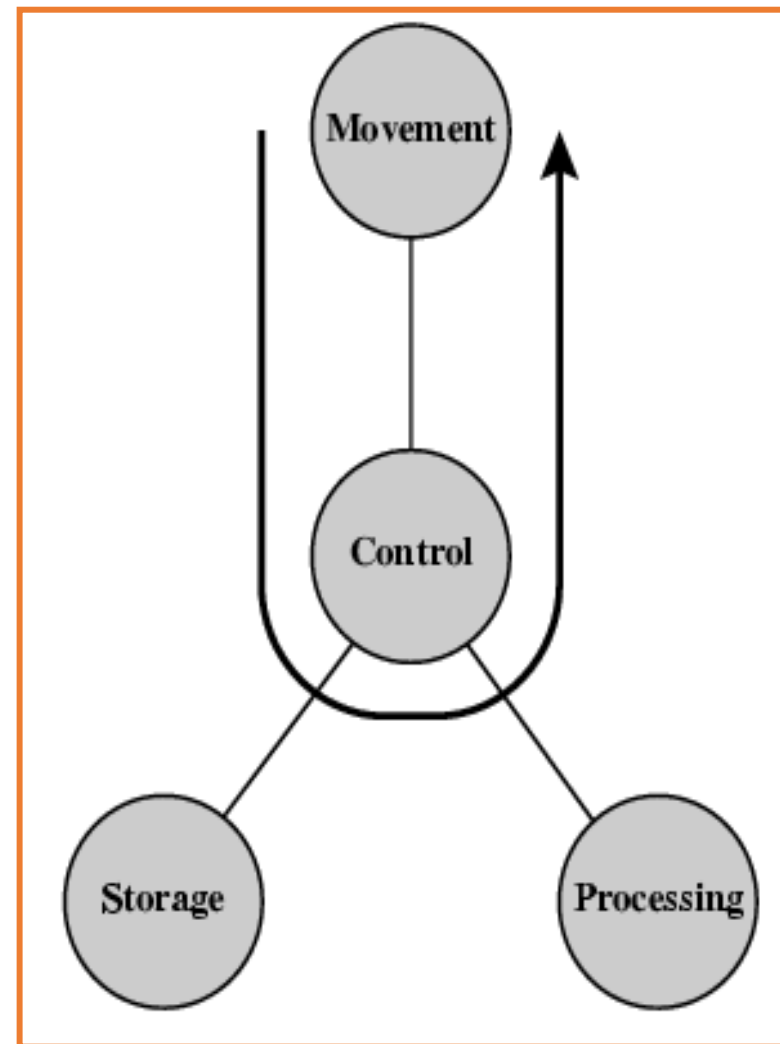


- Computer functions are:
 - Data movement
 - Data storage
 - Data processing
 - Control

Data Movement

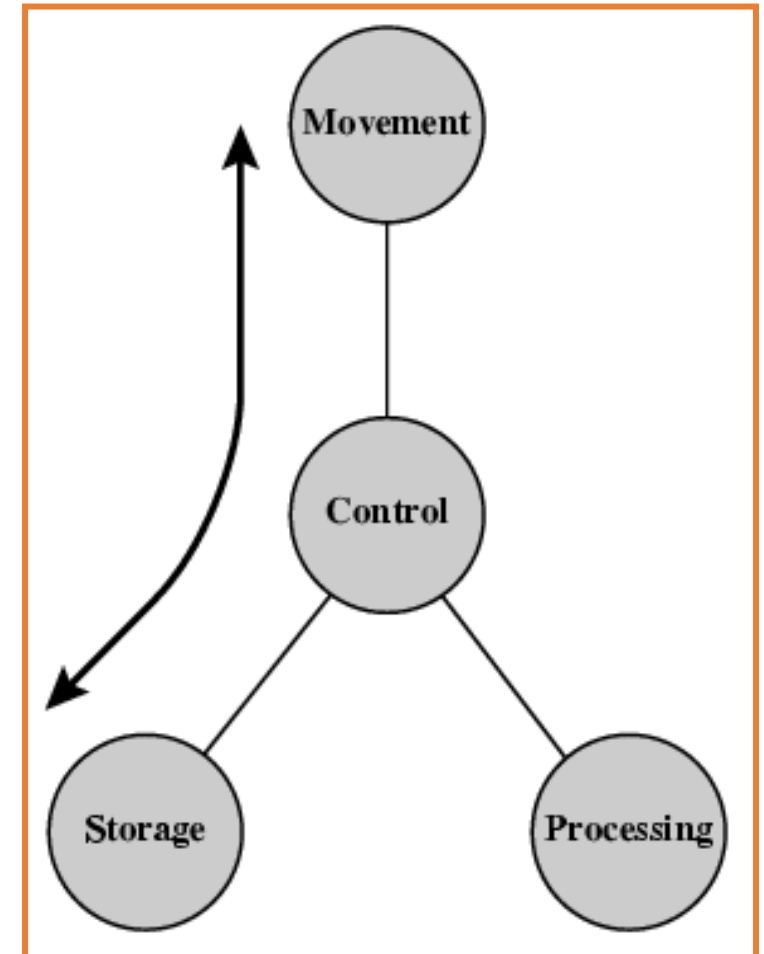
- **Data movement:**

- 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*.



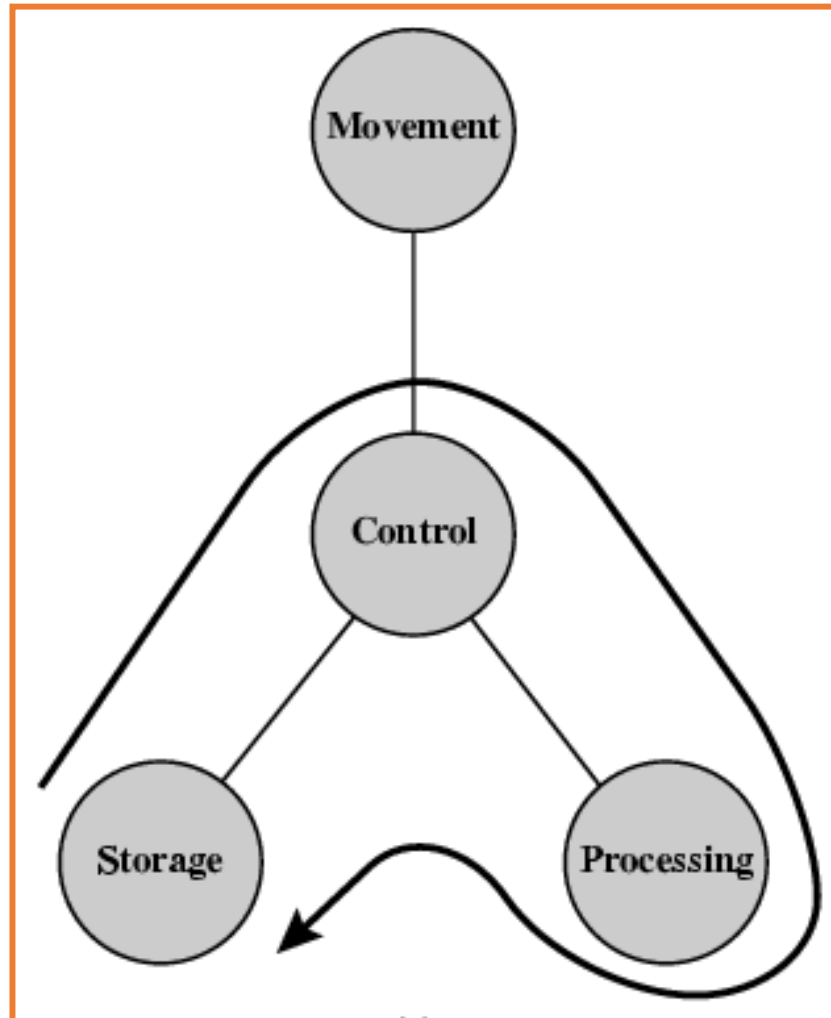
Storage (Read or Write)

- Even if the computer is processing data on the fly (i.e., data come in and get processed, and the results go out immediately), the computer must **temporarily store** at least those pieces of data that are being worked on at any given moment.
- Thus, there is at least a **short-term data storage** function. Equally important, the computer performs a **long-term data storage function**.
- Files of data are stored on the computer for subsequent retrieval and update. **SUB AL,BL**

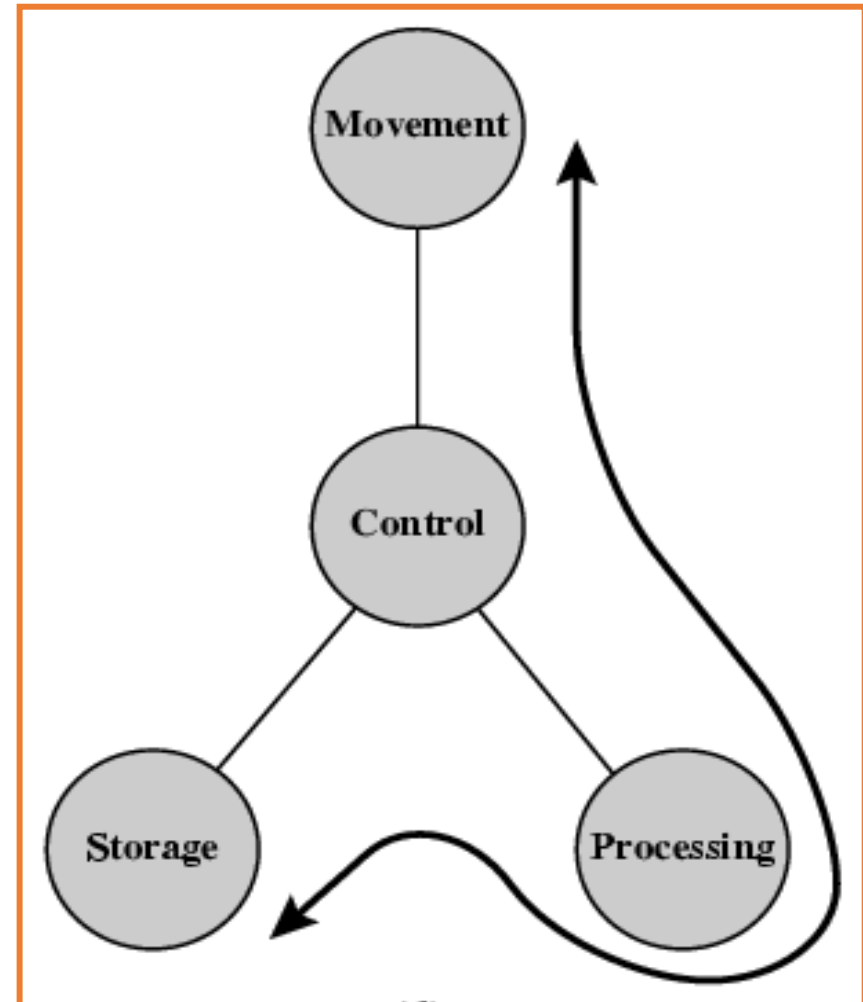


Processing Function

Processing from/to storage



Processing from storage to I/O

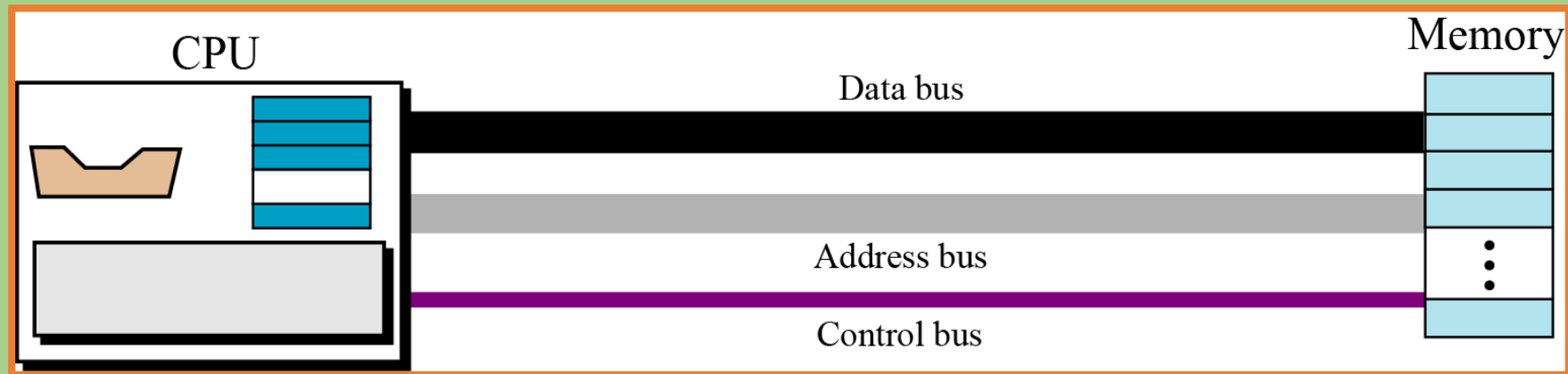


Functions

- 1. Data processing:** Data may take a wide variety of forms, and the range of processing requirements is broad.
- 2. Data storage:** Even if the computer is processing data on the fly , the computer must **temporarily store** 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.
- 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:** Within the computer, a control unit manages the **computer's resources** and orchestrates (coordinates) the performance of its functional parts in response to **instructions**.

Bus Structure

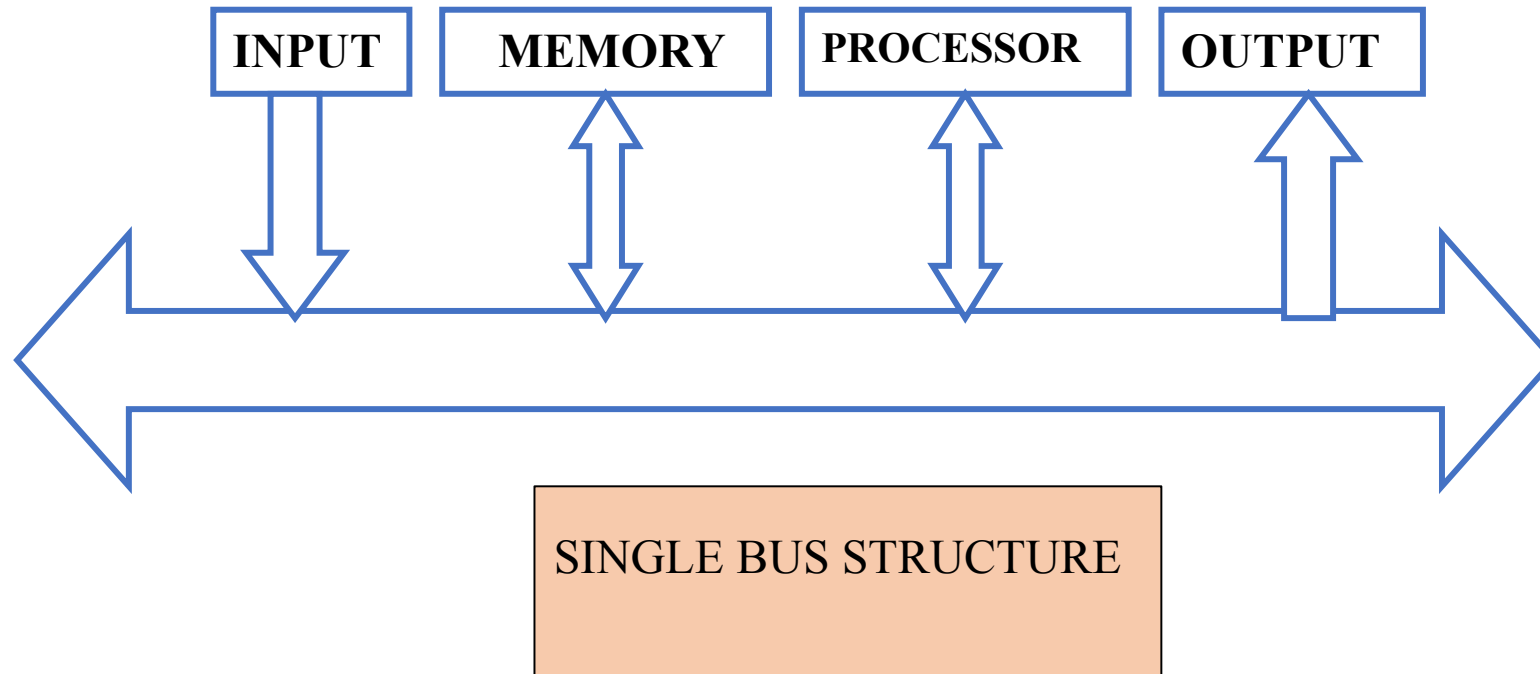
- The CPU and memory are normally connected by three groups of connections, each called a **bus**: *data bus*, *address bus* and *control bus*



Connecting CPU and memory using three buses

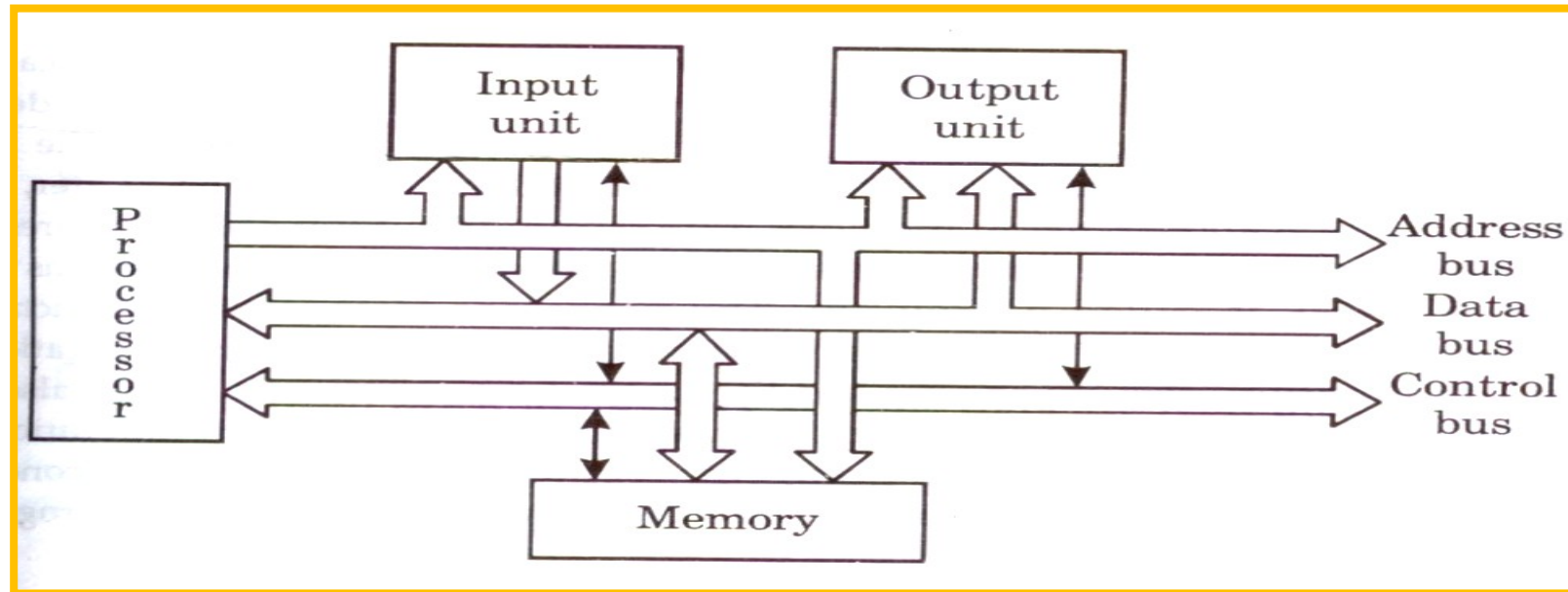
Bus Structure

- Group of wires which carries information from CPU to peripherals or vice – versa
- **Single bus structure:** Common bus used to communicate between peripherals and microprocessor



Bus Structure

- To improve performance **multi-bus** structure can be used
- In two – bus structure : One bus can be used to fetch instruction other can be used to fetch data, required for execution.
- Thus improving the performance ,but cost increases



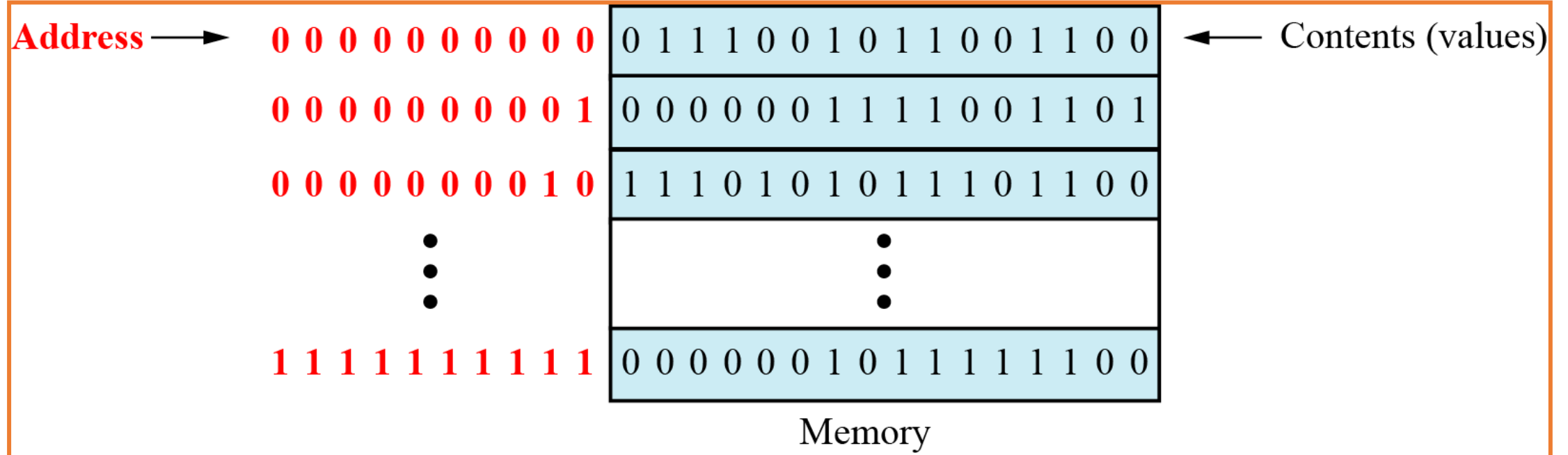
Bus Structure

- **Address bus** : unidirectional : group of wires which carries address information bits from processor to peripherals (16,20,24 or more parallel signal lines)
- **Data bus**: bidirectional : group of wires which carries data information bit from processor to peripherals and vice – versa
- **Control bus**: bidirectional: group of wires which carries control signals from processor to peripherals and vice – versa

Memory Locations and Addresses

- **Main memory** is the second major subsystem in a computer. It consists of a collection of storage locations, each with a unique identifier, called an **address**.
- Data is transferred to and from memory in groups of bits called **words**. A word can be a group of 8 bits, 16 bits, 32 bits or 64 bits (and growing).
- If the word is 8 bits, it is referred to as a **byte**. The term “byte” is so common in computer science that sometimes a 16-bit word is referred to as a 2-byte word, or a 32-bit word is referred to as a 4-byte word.

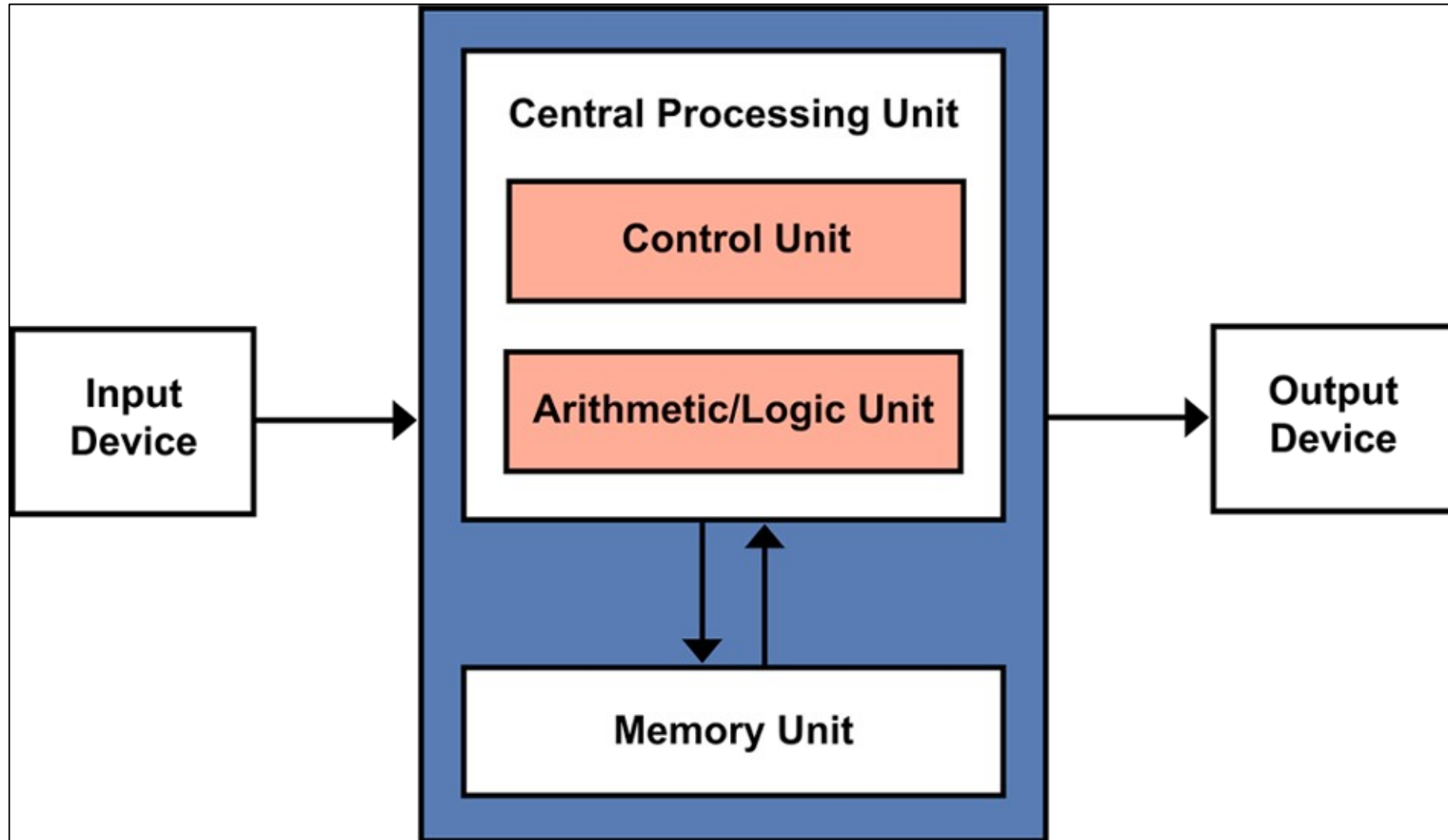
Memory Locations and Addresses



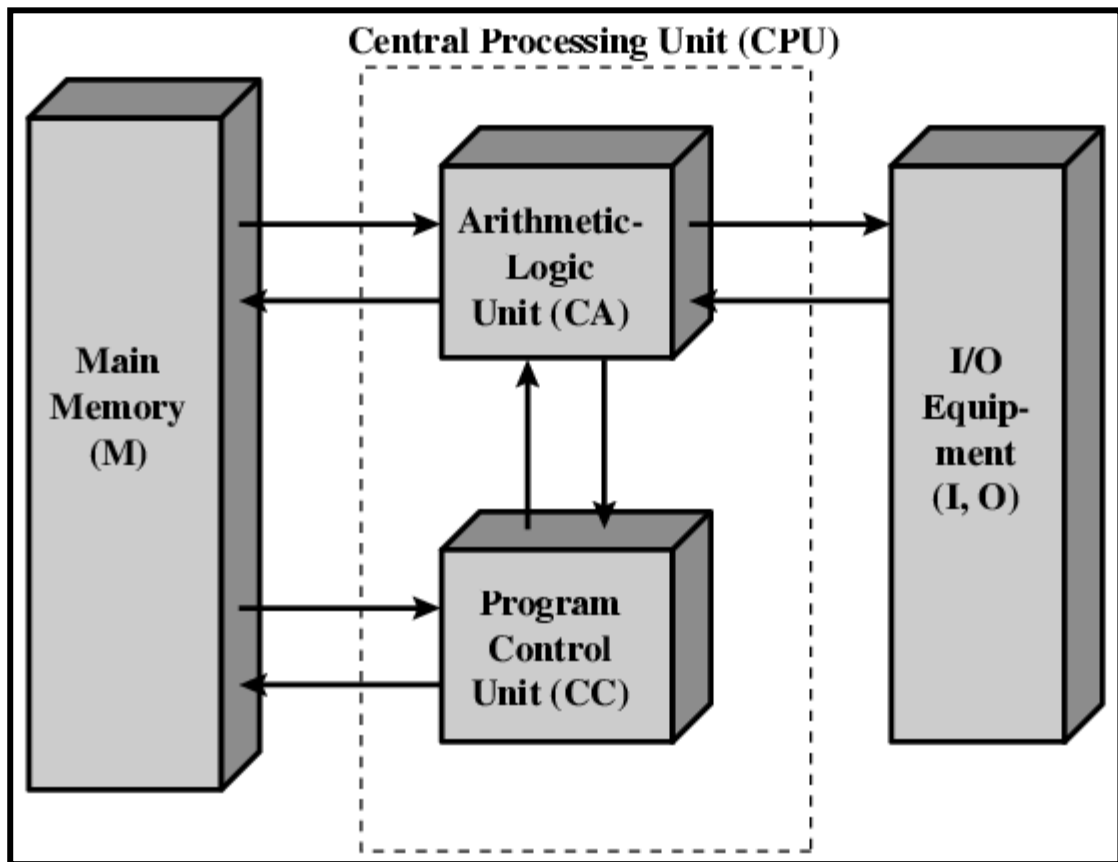
Address space

- To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address.
- The total number of uniquely identifiable locations in memory is called the **address space**.
- For example, a memory with 64 kilobytes (16 address line required) and a word size of 1 byte has an address space that ranges from 0 to 65,535.

Computer Units



Von Neumann Machine



Historically there have been 2 types of Computers:

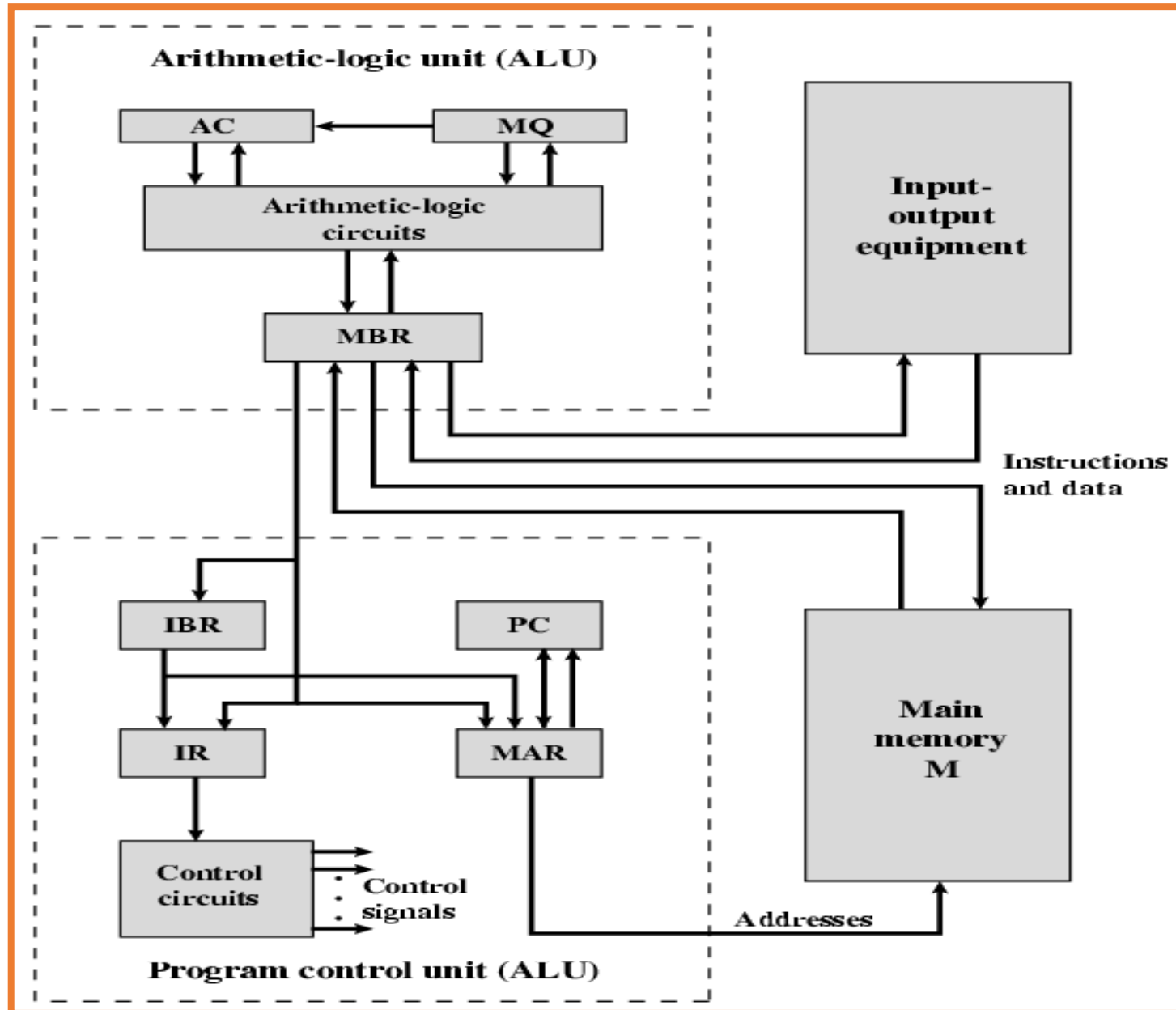
1. **Fixed Program Computers** – Their function is very specific and they couldn't be programmed, e.g. Calculators. **ADD AL,BL**
2. **Stored Program Computers** – These can be programmed to carry out many different tasks, applications are stored on them, hence the name.

Von Neumann Machine is a general purpose computer with store program concept

Von Neumann Machine

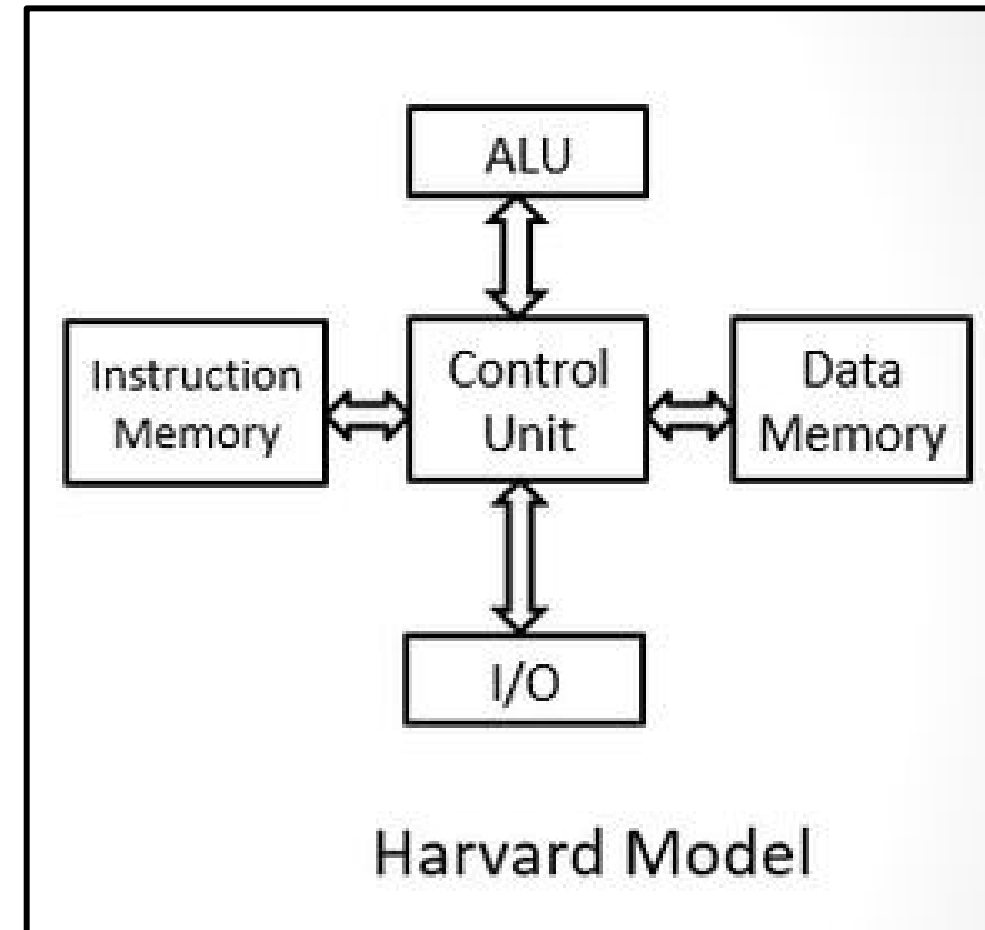
- The modern computers are based on a **stored-program concept** introduced by **John Von Neumann**. In this stored-program concept, programs and data are stored in a **separate storage unit called memories** and are treated the same.
- Stored Program concept
- Main memory storing **programs (code)** and data
- ALU operating on **binary data**
- **Control unit** interpreting instructions from memory and executing
- Input and output equipment operated by **control unit**
- It is also known as **IAS** computer:
- Princeton **Institute for Advanced Studies (IAS)**
- Completed 1952

IAS Machine (Institute for Advanced Studies)



Harvard Architecture

- Harvard Architecture introduce the separate memory concept.
- Data and instruction memory blocks becomes separate entity.
- Provides dedicated bus for connecting it.
- Separate connection allows to access data and instruction simultaneously.
- It overcomes the bottleneck of Von Neumann Machine with single memory block.



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): 12 bits

- 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):

- Holds temporarily the right hand instruction from a word in memory.

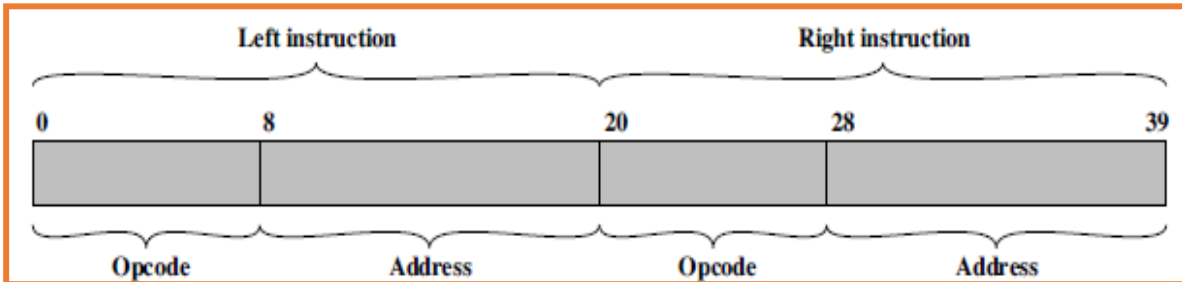
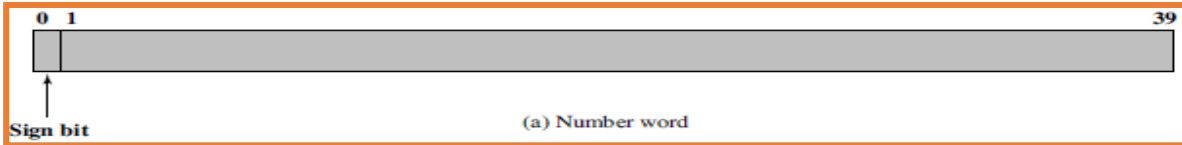
PC:

- Connected to the internal address bus, the Program Counter holds the address in memory of the next program instruction. Notice that it doesn't connect directly to the memory, but must go via the MAR.

Accumulator (AC) and Multiplier Quotient (MQ):

- Hold temporary operands and result of ALU operation.
- **Ex. 40bit *40 bit=80 bit then AC=MSB (40 bit) and MQ=LSB (40 bit)**

IAS Machine Instruction Format

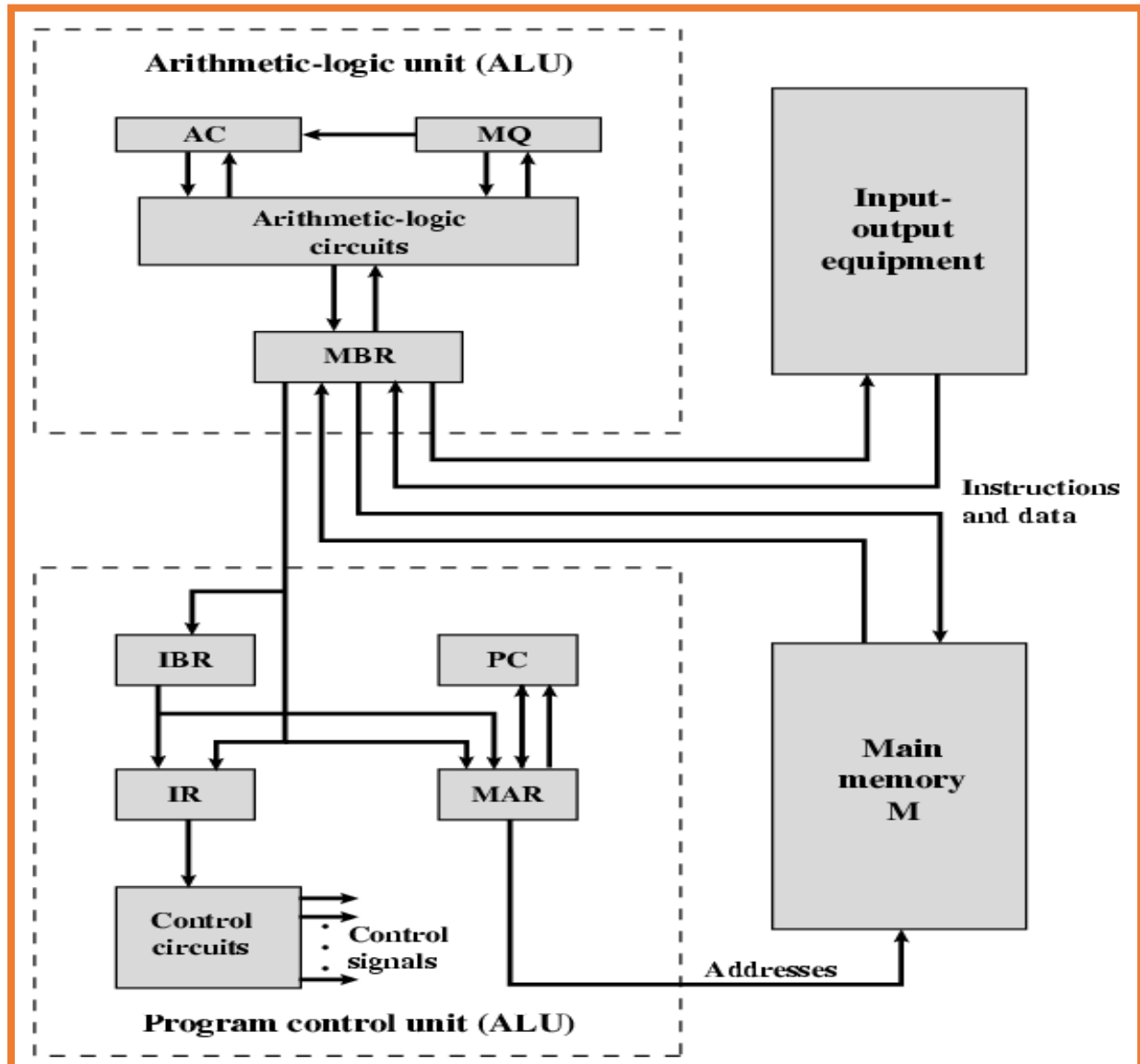


Instruction word

- **Opcode** is an instruction that tells processor what to do with the variable or data written besides it. Ex MOV: 8BH; LOAD:3AH; STOR:32H; ADD:86H
- **Operand** is a variable that stores data (and data can be a memory address or any data that we want to process).
- e.g. **MOV** AL,BL
- here instruction MOV is an opcode. AL & BL are operands.

*Opcode is processor's Operational Code which is designed during the manufacturing of the chip. Which means you can not change the opcode unless you change the **HARDWARE** design of the processor.*

IAS working Steps



Memory:

1. **LOAD M(X) 500, ADD M(X) 501**
2. **STOR M(X) 500, (Other Instruction) 500=7**

500 3

501 4

Ref.: <https://www.youtube.com/watch?v=mVbxrQE4f90>

IAS working Steps

Demonstration of IAS Machine

MEMORY

1. LOAD M(X) 500, ADD M(X) 501

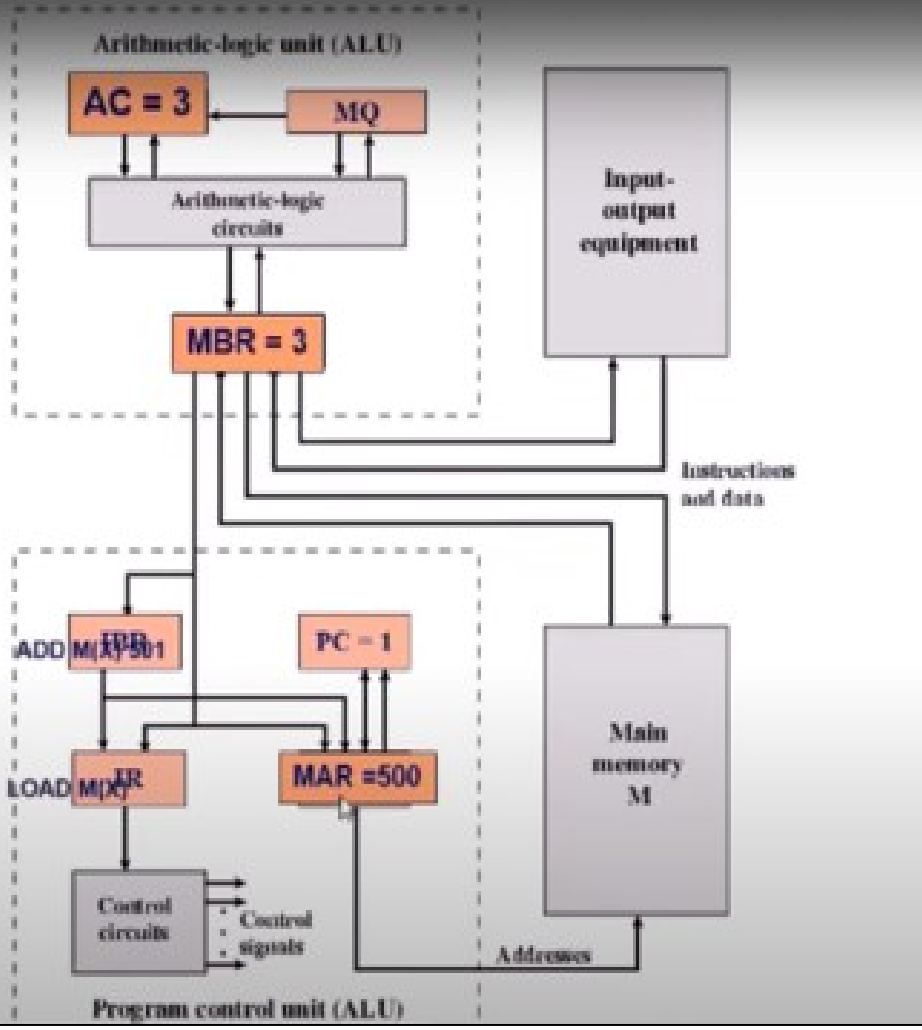
2. STOR M(X) 500, (Other Ins)

.....

500. 3

501. 4

PC	1
MAR	500
MBR	3
IR	LOAD M(X)
IBR	ADD M(X) 501
AC	3



Memory:

1. **LOAD** M(X) 500, **ADD** ~~M(X) 501~~

2. **STOR** ~~M(X) 500, (Other~~ **Instruction)**

500 3

501 4

IAS working Steps

Demonstration of IAS Machine

MEMORY

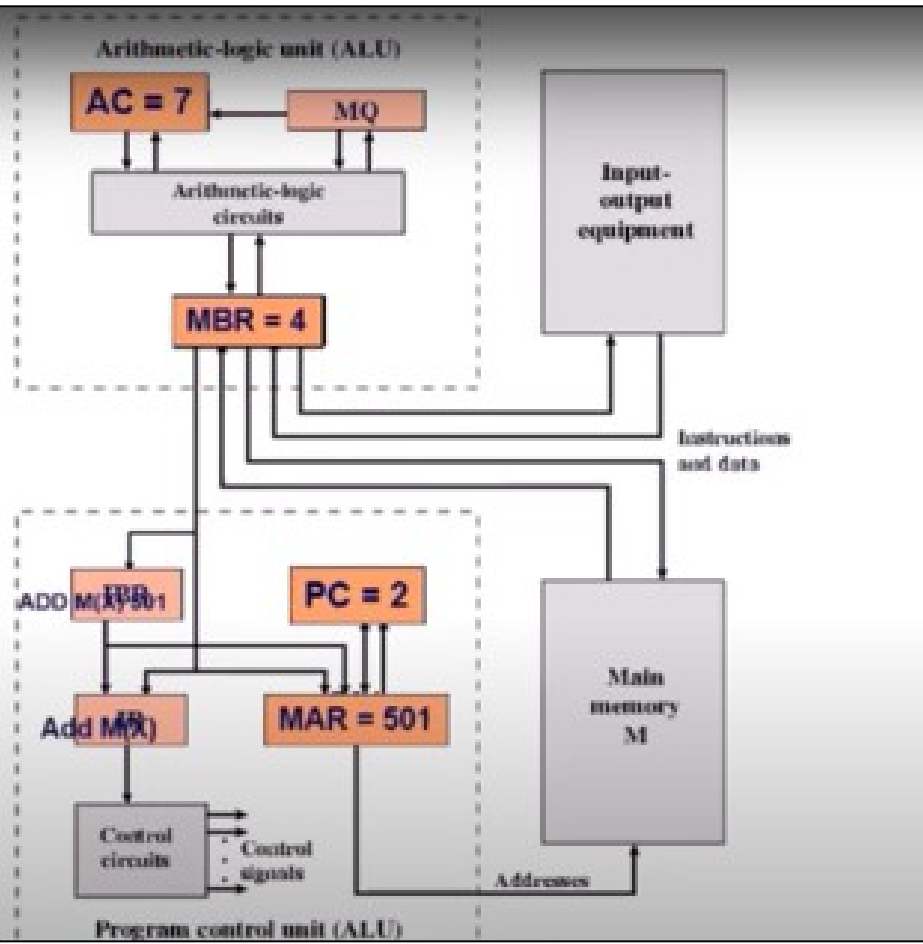
1. LOAD M(X) 500, ADD M(X) 501

2. STOR M(X) 500, (Other Ins)

500. 3

501. 4

PC	2
MAR	501
MBR	4
IR	ADD M(X)
IBR	ADD M(X) 501
AC	7

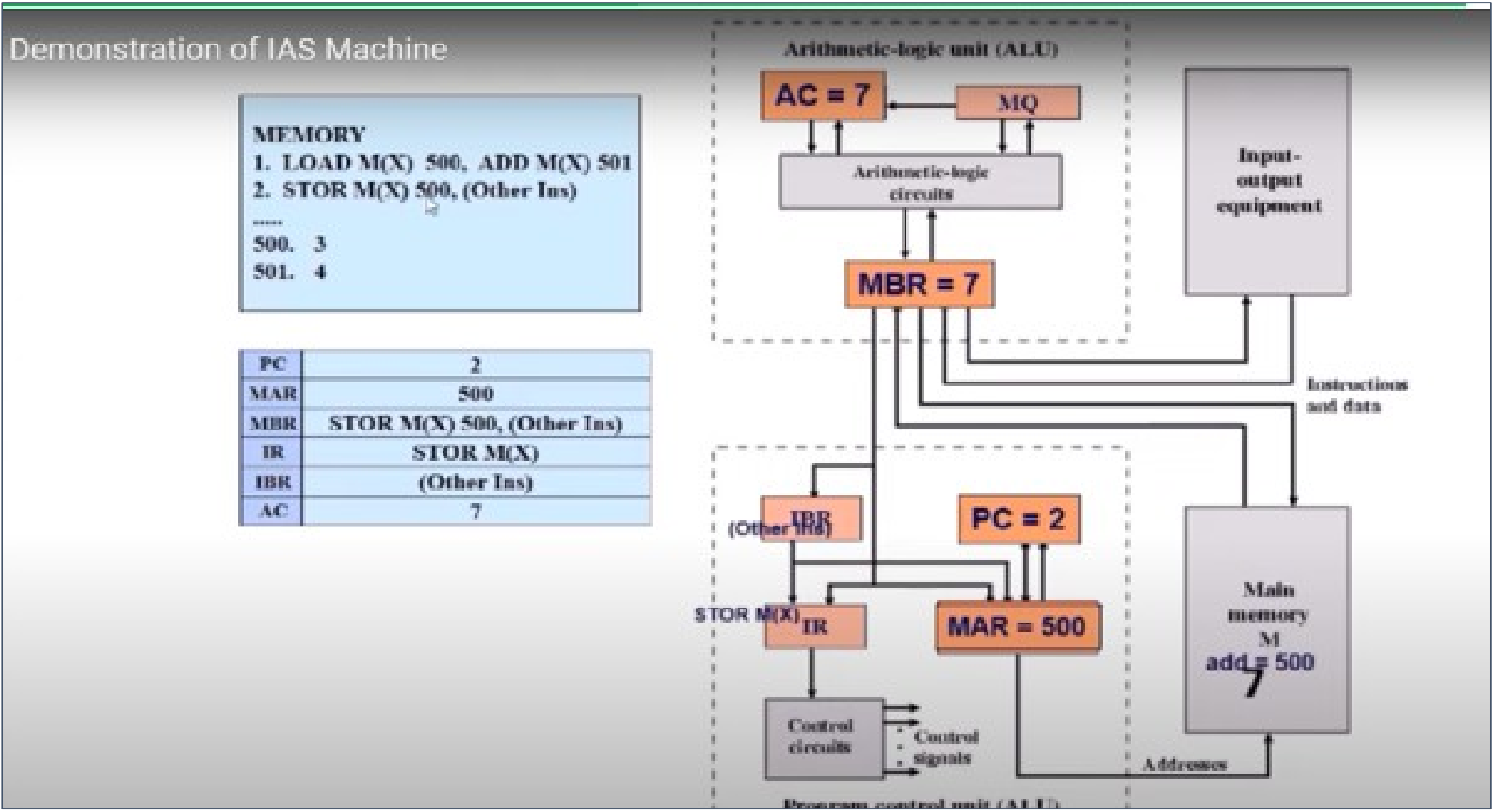


Memory:

1. ~~LOAD M(X) 500,~~
~~ADD M(X) 501~~

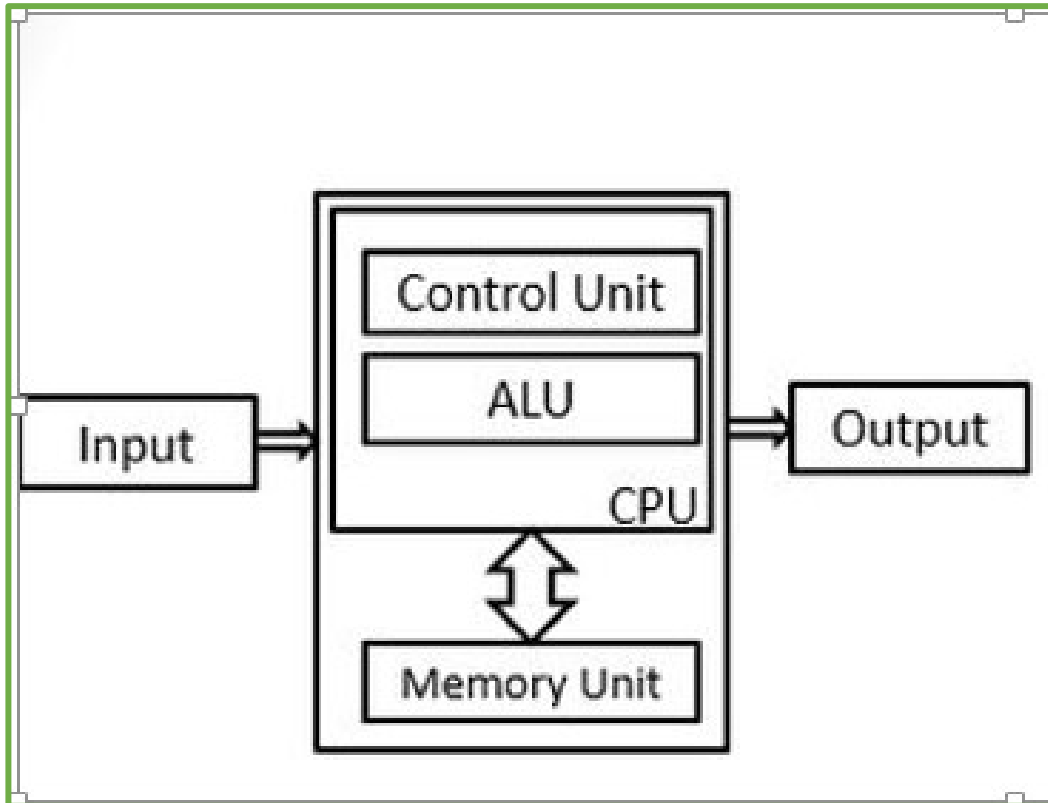
2. ~~STOR M(X) 500,~~
~~(Other Instruction)~~
5003
5014s

IAS working Steps

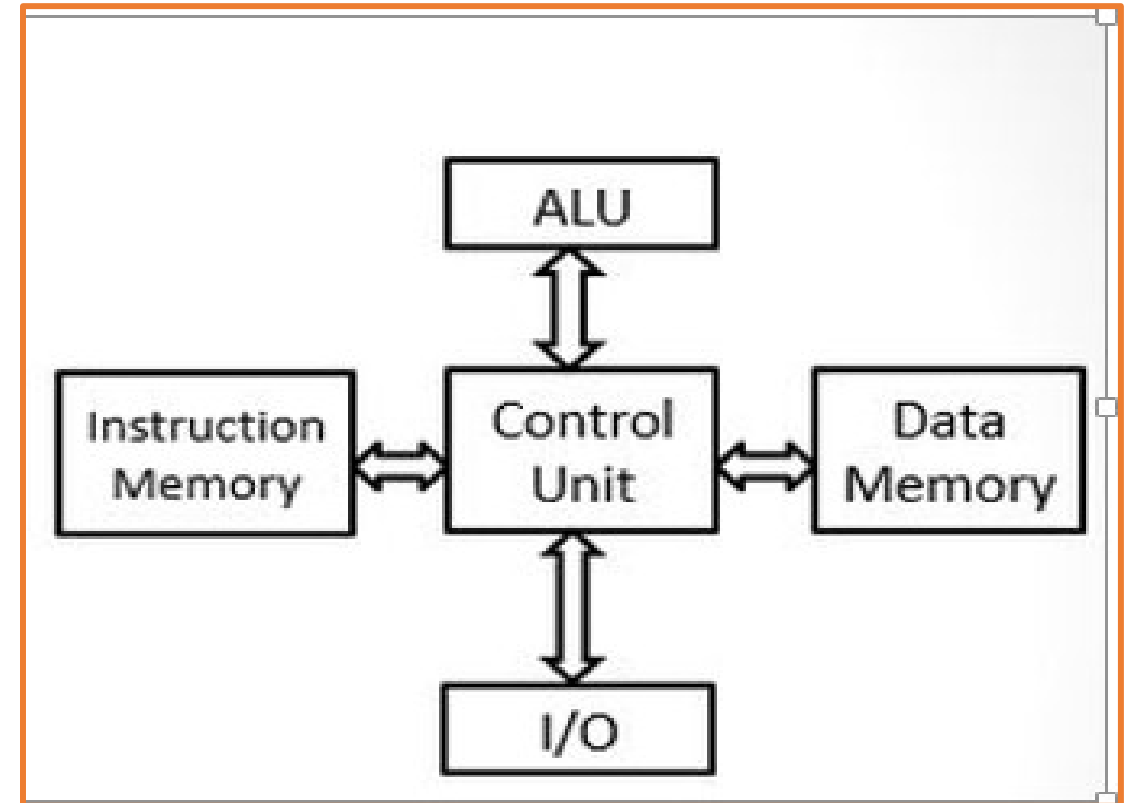


Memory:
2. STOR M(X) 500,
(Other Instruction)
5003
5014

Von Neumann Architecture



Harvard Architecture



Comparison of Von Neumann and Harvard

Von Neumann Architecture	Harvard Architecture
It is named after the mathematician and early computer scientist John Von Neumann.	The name is originated from “ Harvard Mark I ” a relay based old computer.
It required only one memory for their instruction and data.	It required two memories for their instruction and data.
Design of the von Neumann architecture is simple .	Design of Harvard architecture is complicated .
Von Neumann architecture is required only one bus for instruction and data.	Harvard architecture is required separate bus for instruction and data.
Processor needs two clock cycles to complete an instruction. ADD AL,BL	Processor can complete an instruction in one cycle
Low performance as compared to Harvard architecture.	Easier to pipeline , so high performance can be achieve.
It is cheaper .	Comparatively high cost .

RISC and CISC Machines

Reduced Instruction Set Architecture

- Focus on hardware simplification
- Uses basic instruction
- Reduce the cycles per instruction at the cost of the number of instructions per program.

- **Complex Instruction Set Architecture**

- Try to perform functions like load, evaluate and store operation with single instruction.
Example – Multiply command
- Approach attempts to minimize the number of instructions per program but at the cost of an increase in the number of cycles per instruction.

RISC and CISC Machines

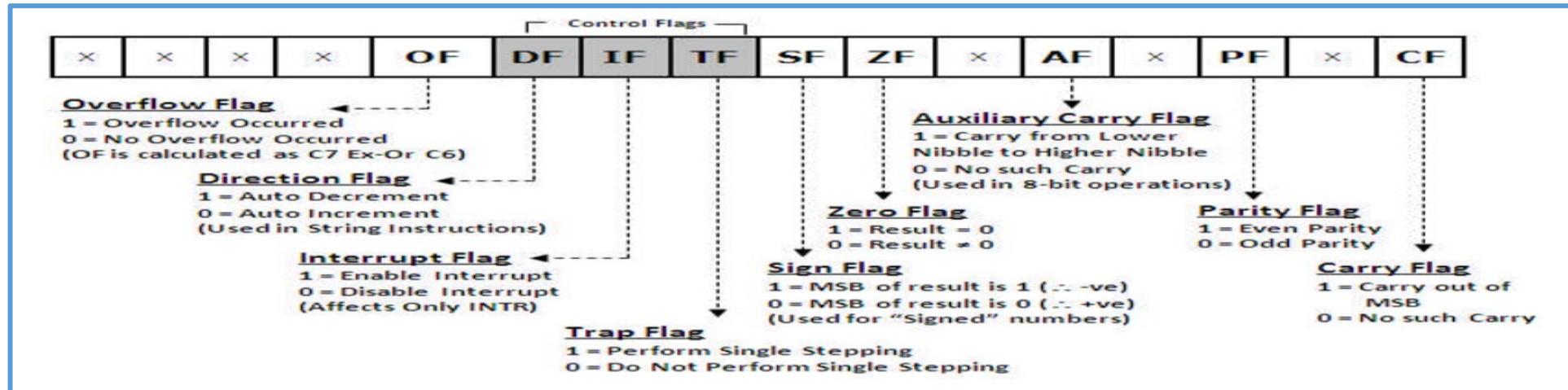
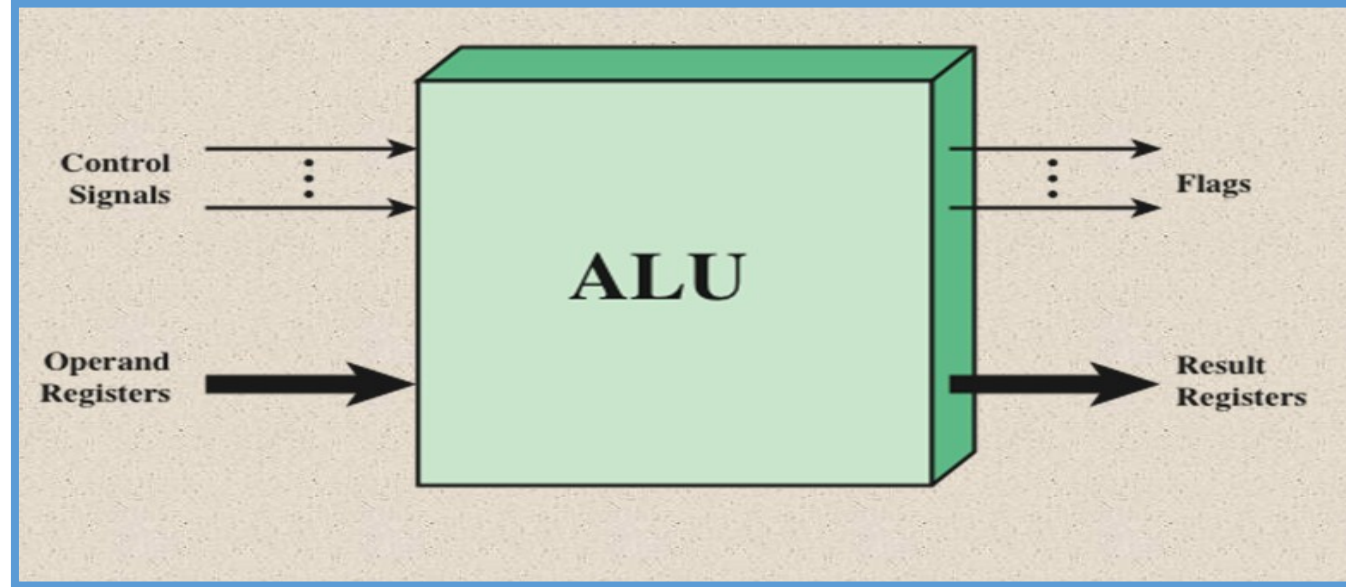
BASIS FOR COMPARISON	RISC (Reduced Instruction Set Computer)	CISC (Complex Instruction Set Computer)
Emphasis on	Software	Hardware
Includes	Single clock	Multi-clock
Instruction-set size	Small	Large
Instruction formats	fixed (32-bit) format	Varying formats (16-64 bits each instruction).
Addressing modes used	Limited to 3-5	12-24
General purpose registers used	32-192	8-24 al bl, cl
Memory inferences	Register to register	Memory to memory
Cache design	Split data cache and instruction cache.	Unified cache for instructions and data.
Clock rate	50-150 MHz	33-50 MHz
CPU Control	Hardwired without control memory.	Microcoded using control memory (ROM).

RISC and CISC Machines

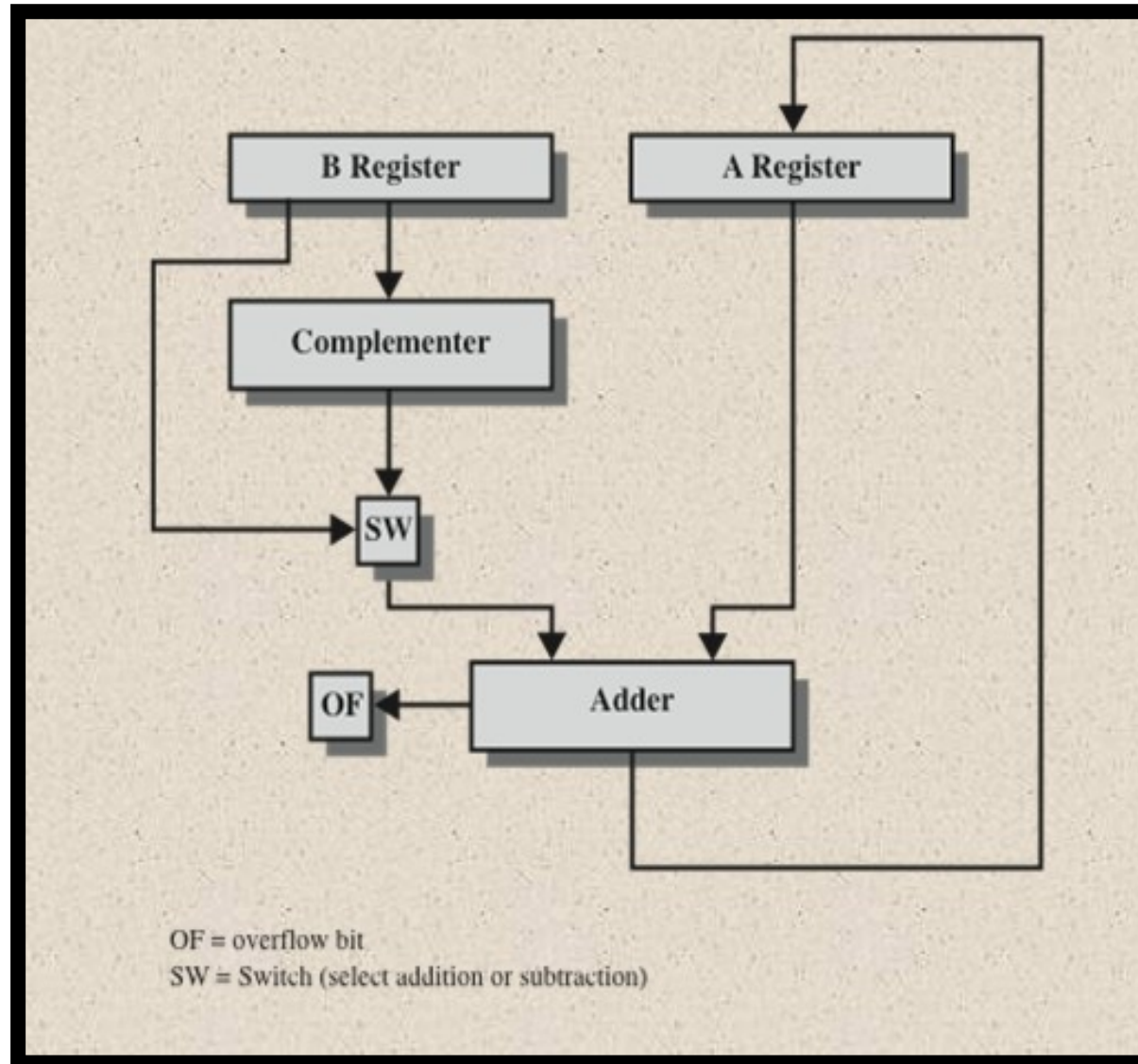
BASIS FOR COMPARISON	RISC (Reduced Instruction Set Computer)	CISC (Complex Instruction Set Computer)
Pipeline	High Pipelined	Not or less Pipelined
Data Transfer	Register to Register	Memory to Memory
Instruction refer to the Memory	Only load and store	All
Used in	Portable devices, Apple iPod, Smart phone, Tablets	Desktop, Laptops
Set of instructions	Highly optimized	Less optimized
RAM is required to store instruction	High	Very less
Instruction decoding logic	Simple	Complex
Program length	Large	Small
Example: Multiply 2 Numbers	LW A, 203 ;Load LW B, 502 ;Load MULT A,B SW 203,A ;Store	MULT 203,502

Arithmetic and Logical Unit

ALU Inputs and Outputs



Block Diagram: Hardware for Addition and Subtraction



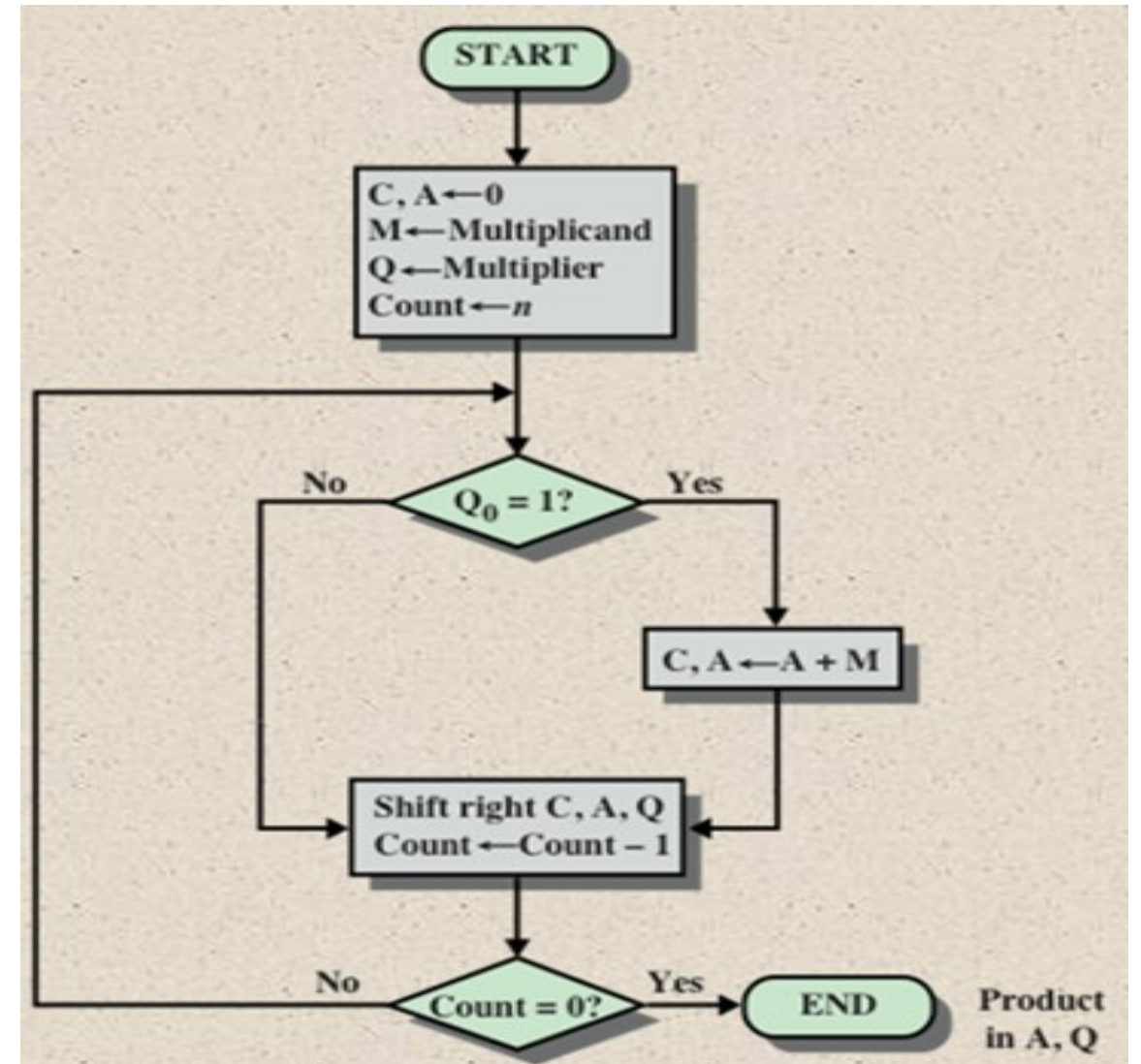
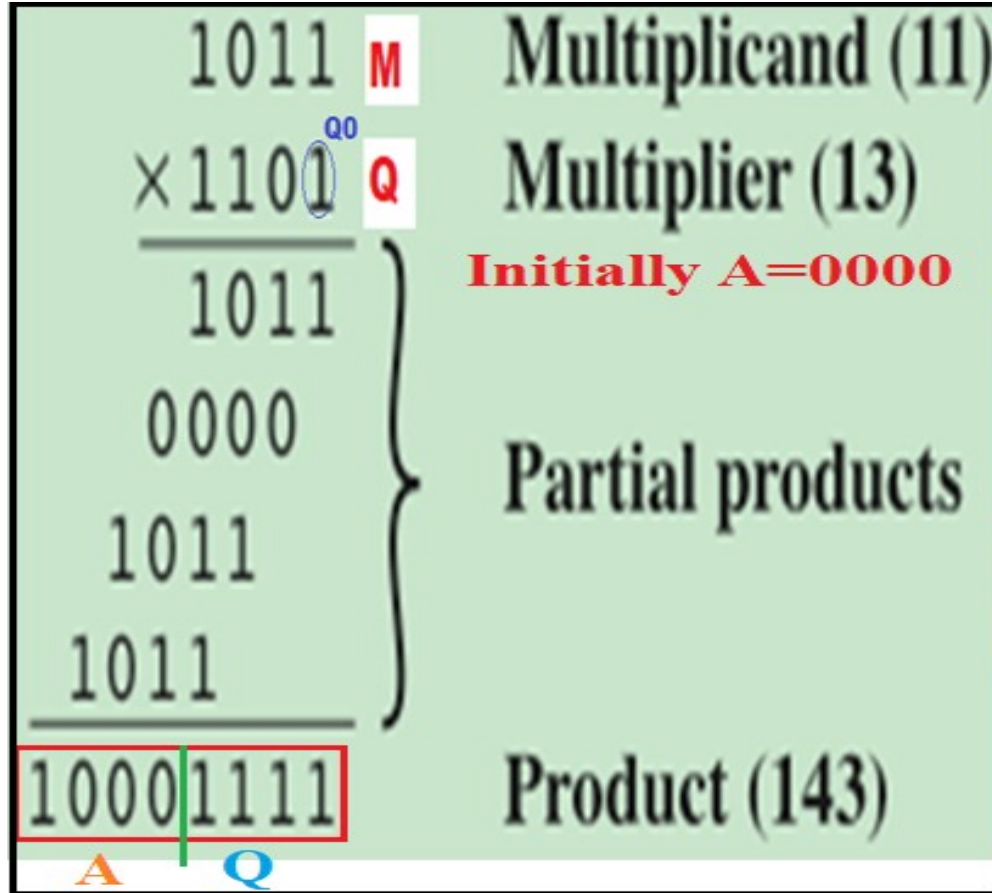
$$\begin{array}{r} 0011 = 3 \\ +0100 = 4 \\ \hline 0111 = 7 \\ (c) (+3) + (+4) \end{array}$$

$$\begin{array}{r} 0101 = 5 \\ +1110 = -2 \\ \hline 10011 = 3 \end{array}$$

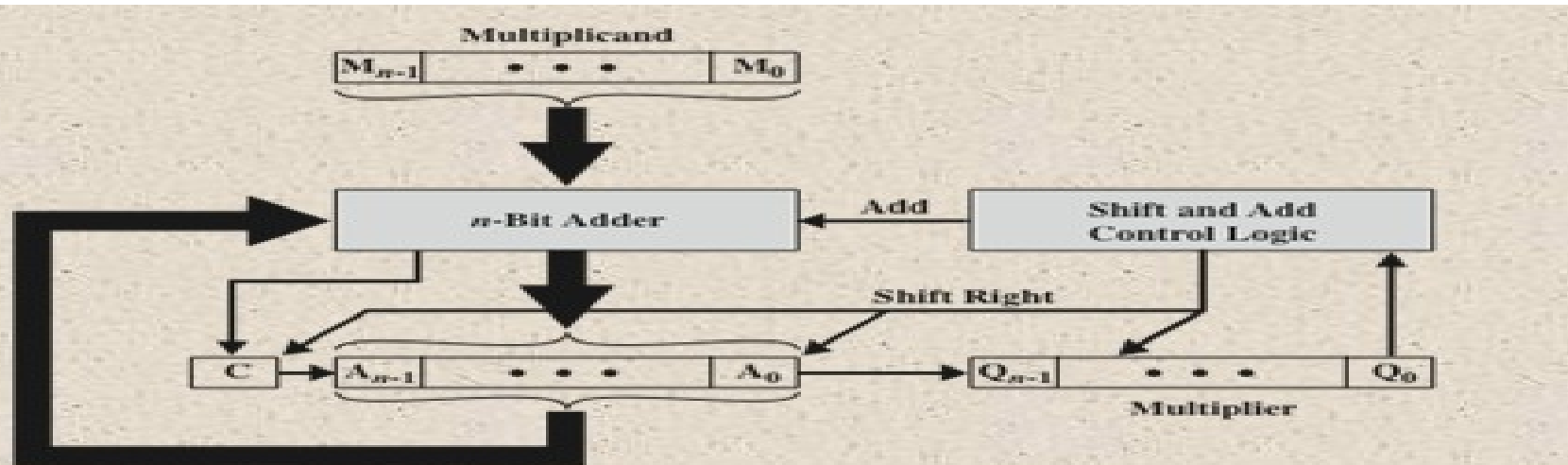
$$\begin{array}{r} 1001 = -7 \\ +0101 = 5 \\ \hline 1110 = -2 \\ (a) (-7) + (+5) \end{array}$$

$$\begin{array}{r} 1100 = -4 \\ +1111 = -1 \\ \hline 11011 = -5 \\ (d) (-4) + (-1) \end{array}$$

Multiplication (Unsigned Binary Integers $11 \times 13 = 143$)



Hardware Implementation of Unsigned Binary Multiplication Operation



(a) Block Diagram

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

Booth's Algorithm (Signed Number Multiplication)

- Used to reduce number of operations
- For signed multiplication
- Generate $2n$ bit product
- Treats positive and negative numbers uniformly (called 2s compliment method)
- Recoding of Multiplier is performed
- Rules of recording – Use implied zero at LSB and start scanning LSB to MSB

Rules for recoded number	
0-1	-1
1-0	1
0-0	0
1-1	0

1. 001110 0 = Recoded 0 1 0 0 -1 0
2. 101100 0 = Recorded -11 0 -1 0 0

Booth's Algorithm

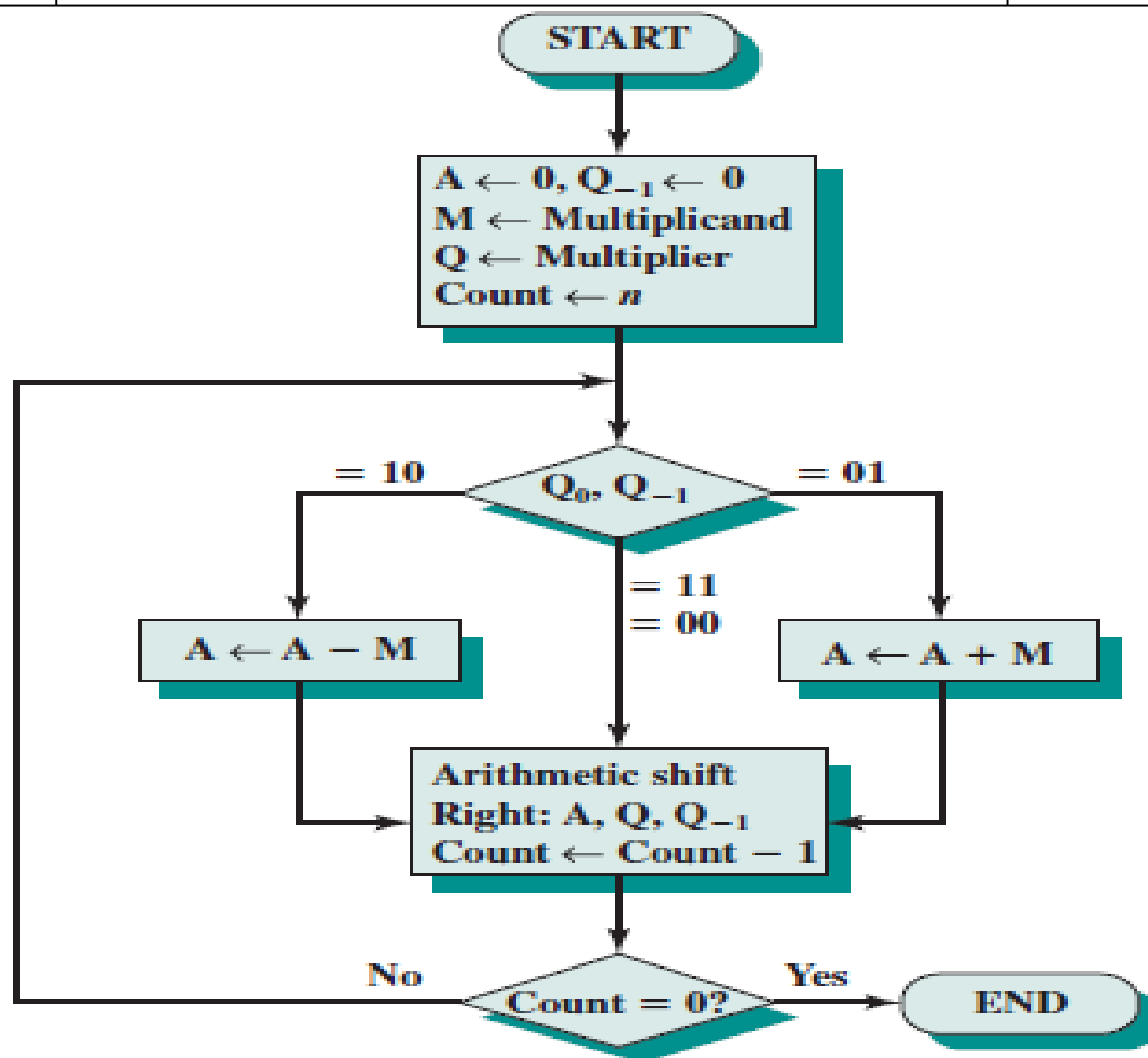


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

Booths Algorithm using Tabular Method.

Example Consider the multiplication of the two positive numbers $M = 0111$ (7) and $Q = 0011$ (3) and assuming that $n = 4$. The steps needed are tabulated below.

M	A	Q	$Q(-1)$		
0111	0000	0011	0	Initial value	
0111	1001	0011	0	$A = A - M$	
0111	1100	1001	1	ASR	End cycle #1

0111	1110	0100	1	ASR	End cycle #2

0111	0101	0100	1	$A = A + M$	
0111	0010	1010	0	ASR	End cycle #3

0111	0001	0101	1	ASR	End cycle #4
	<div style="text-align: center;"> } </div>				
	+21 (correct result)				

Booth's Algorithm

Booth's Algorithm (Signed Number) 19*12=228														
Multiplicand	19	Binary=	010011											
Two's Complement of Multiplicand = 101101														
Multiplier	12	Binary-	001100											
Recoded Multiplier is = 0 0 1 1 0 0 0				0 - Implied Zero						0	1	0	0	1 1
Recoded Multiplier is = 0 +1 0 -1 0 0										0	1	0	-1	0 0
Sign Extension				0	0	0	0	0	0	0	0	0	0	0
				0	0	0	0	0	0	0	0	0	0	+
Two's Complement of Multiplicand				1	1	1	1	1	0	1	1	0	1	+
				0	0	0	0	0	0	0	0	0	+	+
				0	0	0	1	0	0	1	1	+	+	+
				0	0	0	0	0	0	0	+	+	+	+
Carry				1	1	1			1	1				
Ignore final carry				0	0	0	0	1	1	1	0	0	1	0
				2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹
				0	0	0	0	128	64	32	0	0	4	0
														228

Booth's Algorithm

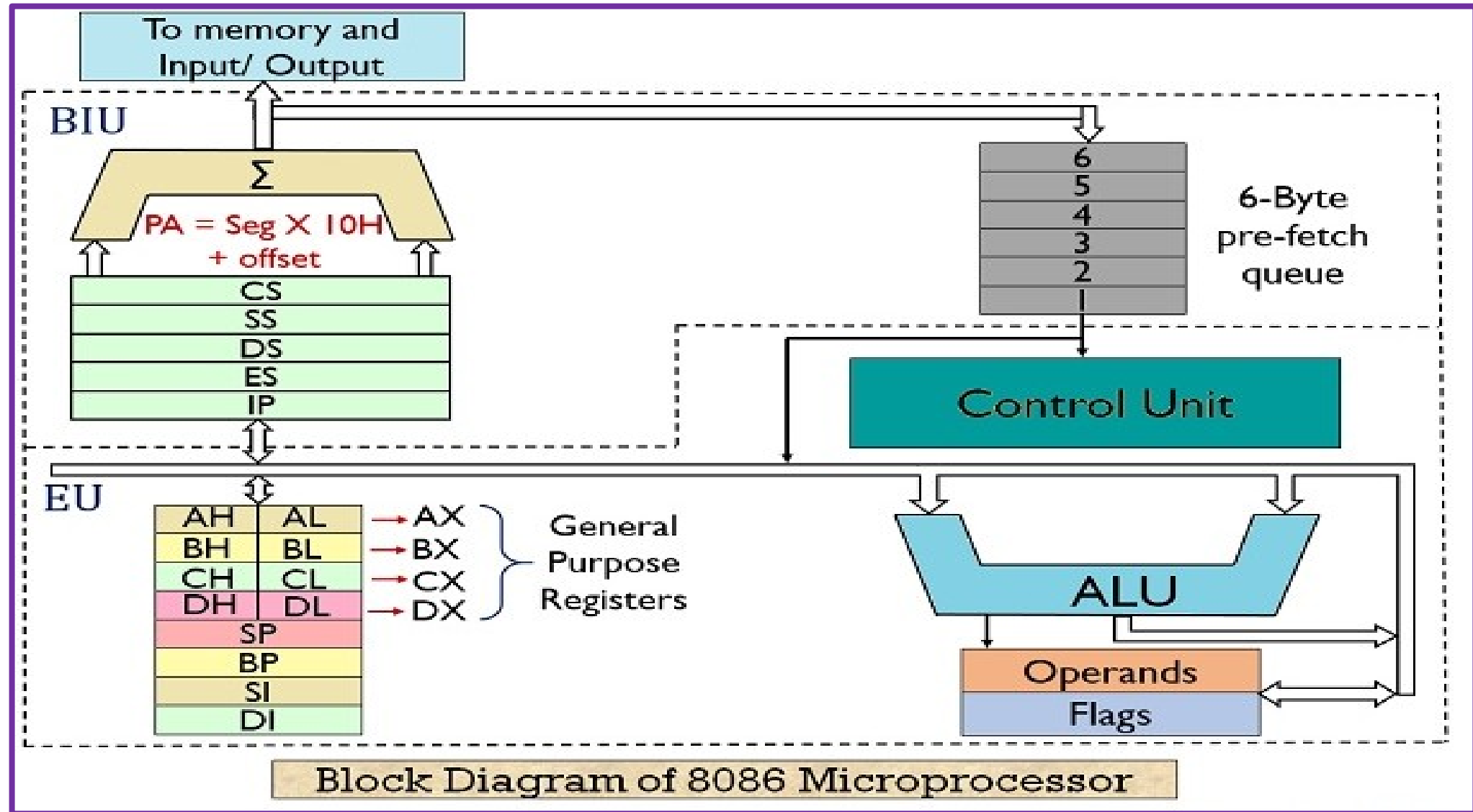
Booth's Algorithm (Signed Number) +14*(-5)=-70										
Multiply = 14*(-5)										
Multiplicand	14	Binary - 01110								
Two's Complement of Multiplicand = 10010										
Multiplier	-5	Binary- 00101								
Two's Complement of -5= 11011										
Recoded Multiplier is = 11011 0 0 - Implied Zero										
Recoded Multiplier is = 0 -1 10 -1										
Sign Extension										
Two's Complement of Multiplicand										
	1	1	1	1	1	1	0	0	1	0
	0	0	0	0	0	0	0	0	0	+
	0	0	0	0	1	1	1	0	+	+
Two's Complement of Multiplicand										
	1	1	1	0	0	1	0	+	+	+
	0	1	0	0	0	0	+	+	+	+
Carry										
	1	1	1	1	1					
	1	1	1	0	1	1	1	0	1	0
Take Two's Complement										
	0	0	0	1	0	0	0	1	1	0
	2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
	0	0	0	64	0	0	0	4	2	0
70										

Booth's Algorithm

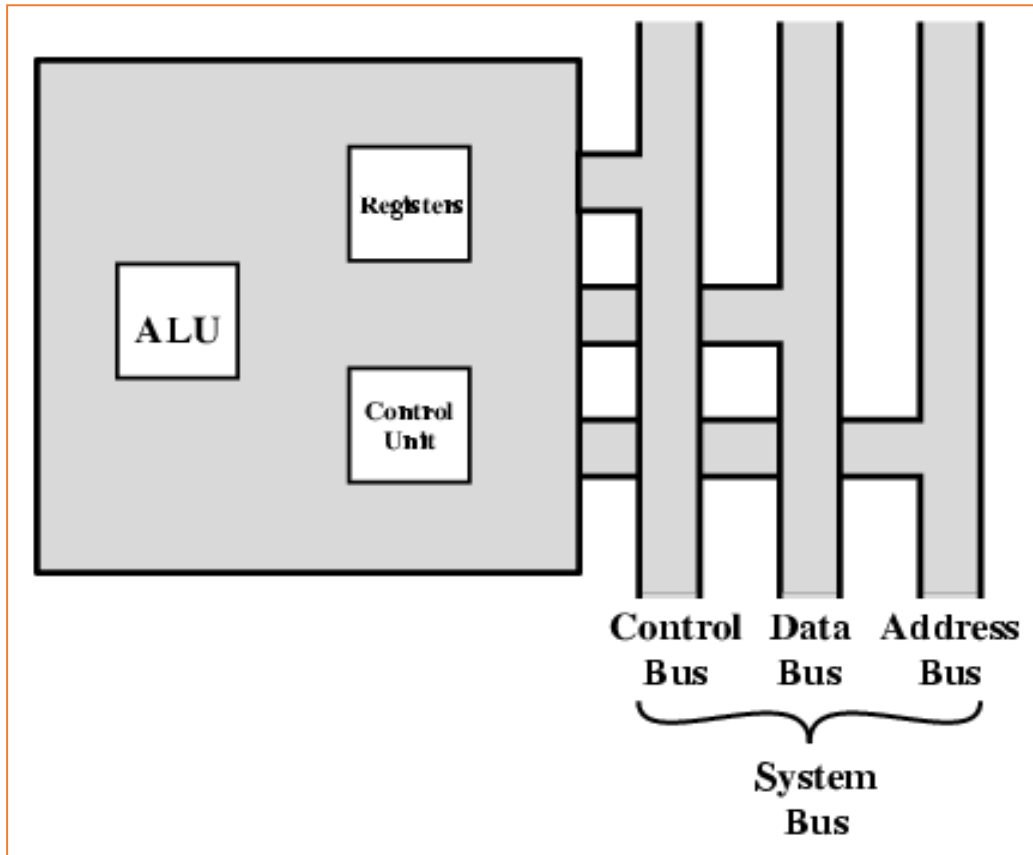
Booth's Algorithm (Signed Number) : (-13)*(-20)=260															
Multiplicand	-13	Binary - 001101													
Two's Complement of Multiplicand = 110011															
Two's Complement of 010011 = 001101															
Multiplier	-20	Binary- 010100													
Two's Complement of 20= 101100															
Recoded Multiplier is = 101100 0			0 - Implied Zero					1	1	0	0	1	1	-13	
Recoded Multiplier is = -1 1 0 -1 0 0								-1	1	0	-1	0	0	-20	
Sign Extension															
			0	0	0	0	0	0	0	0	0	0	0		
			0	0	0	0	0	0	0	0	0	0	0	+	
		Two's Complement of Multiplicand	0	0	0	0	0	0	1	1	0	1	+	+	
			0	0	0	0	0	0	0	0	0	+	+	+	
			1	1	1	1	0	0	1	1	+	+	+	+	
			0	0	0	1	1	0	1	+	+	+	+	+	
Carry			1	1	1	1	1	0	1						
Discard Carry			1	0	0	0	1	0	0	0	0	0	0	0	
						2 ⁹	2 ⁸	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
						0	256	0	0	0	0	0	4	0	0
															260

Processor Organization

8086 Microprocessor



Processor Organization: CPU with System Bus

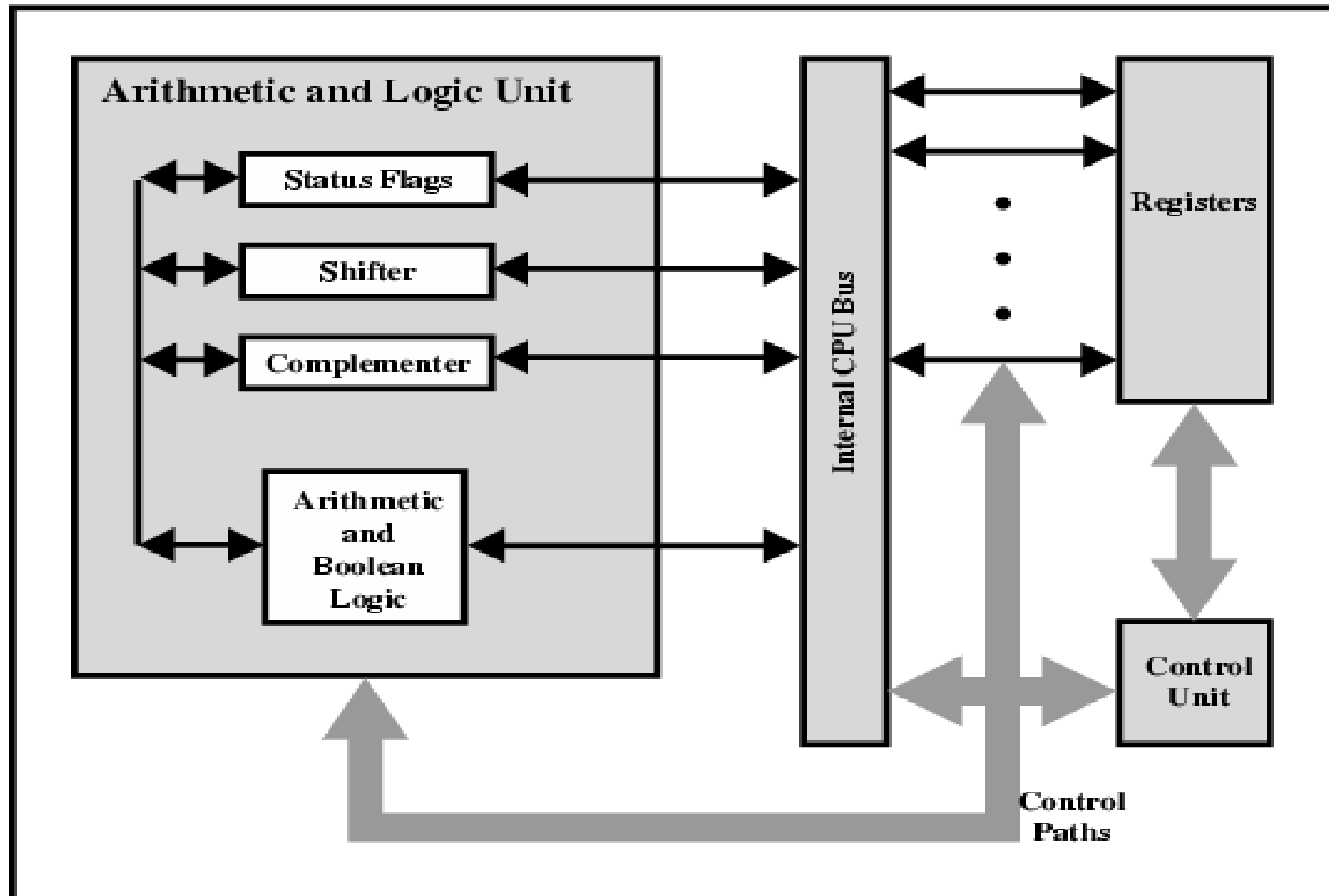


To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:

- **Fetch instruction:** The processor reads an instruction from memory (register, cache, main memory).
- **Interpret instruction:** The instruction is decoded to determine what action is required.

- **Fetch data:** The execution of an instruction may require reading data from memory or an I/O module.
- **Process data:** The execution of an instruction may require performing some arithmetic or logical operation on data.
- **Write data:** The results of an execution may require writing data to memory or an I/O module.

CPU Internal Structure



Register Organization

A computer system employs a memory hierarchy. At higher levels of the hierarchy, memory is faster, smaller, and more expensive (per bit). Within the processor, there is a set of registers that function as a level of memory above main memory and cache in the hierarchy. The registers in the processor perform two roles:

- **User-visible registers:** Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.
- **Control and status registers:** Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

General registers

AX	Accumulator
BX	Base
CX	Count
DX	Data

Pointers and index

SP	Stack ptr
BP	Base ptr
SI	Source index
DI	Dest index

Segment

CS	Code
DS	Data
SS	Stack
ES	Extrat

Program status

Flags
Instr ptr

(b) 8086

Register Organization

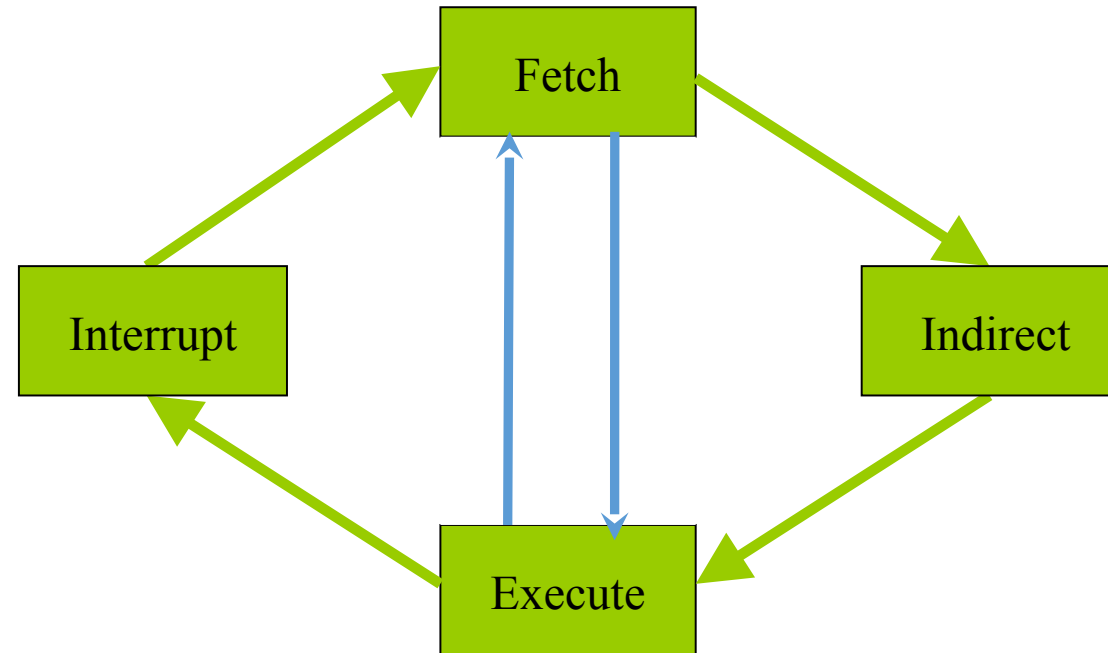
1. User-visible registers: General Purpose , Segment Registers, Index Pointers and Stack Pointer , Condition Codes (Flag Register)



2. Control and status registers (Not Visible to user):

- Program Counter (PC)
- Instruction Decoding Register (IR)
- Memory Address Register (MAR)
- Memory PSW (Program Status Word)
- Buffer Register (MBR)
- Condition Codes + other status information
- The current state of processor is stored in a register called **Processor Status Word (PSW)**.
- Six are status flags and three are control flags.

Instruction Cycle



an instruction cycle includes the following stages:

- **Fetch:** Read the next instruction from memory into the processor.
- **Execute:** Interpret the opcode and perform the indicated operation.
- **Interrupt:** If interrupts are enabled and an interrupt has occurred, save the current process state and service the interrupt.

The Indirect Cycle: if indirect addressing is used, then additional memory accesses are required.

Instruction Cycle

➤ It is the time in which a single instruction is fetched from memory, decoded, and executed.

➤ An **Instruction Cycle** requires the following sub cycle:

1.Fetch : Read next instruction from memory into the processor

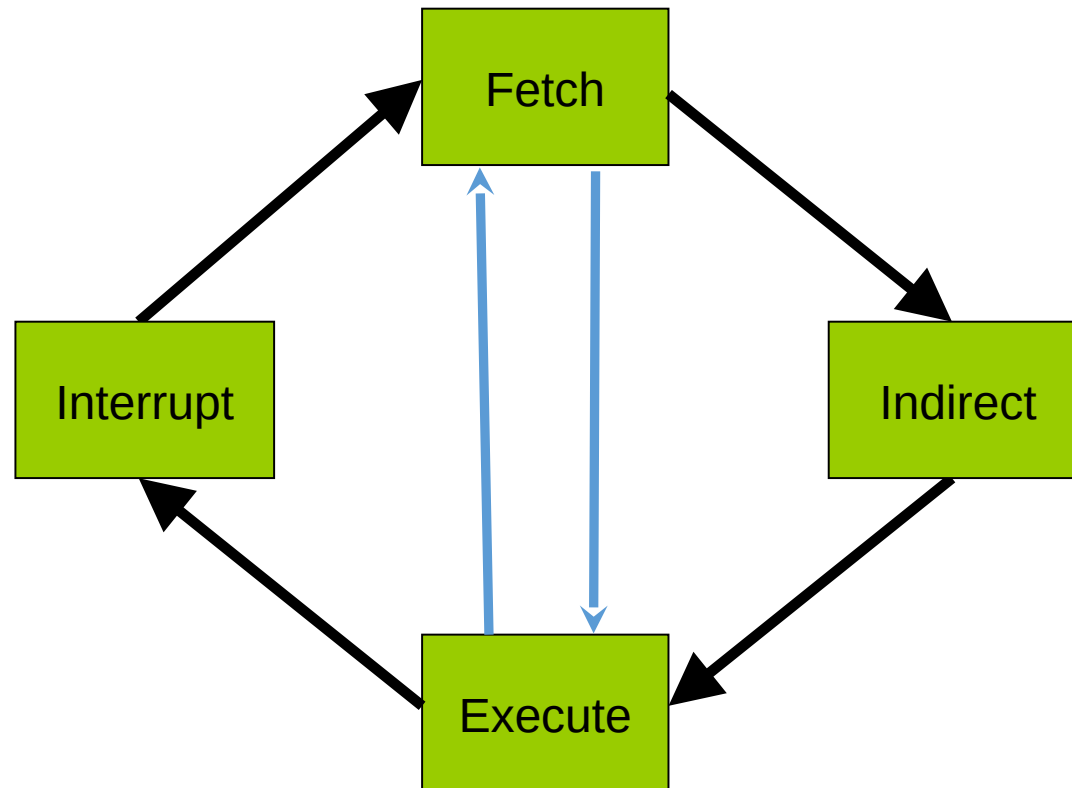
2.Indirect Cycle (Decode Cycle): May require memory access to fetch operands, therefore more memory accesses.

3.Interrupt: Save current instruction and service the interrupt 256 (int 0-int 255) 20=1MB=256*4 CS 2 IP 2 IVT 00000 FFFFFFFH

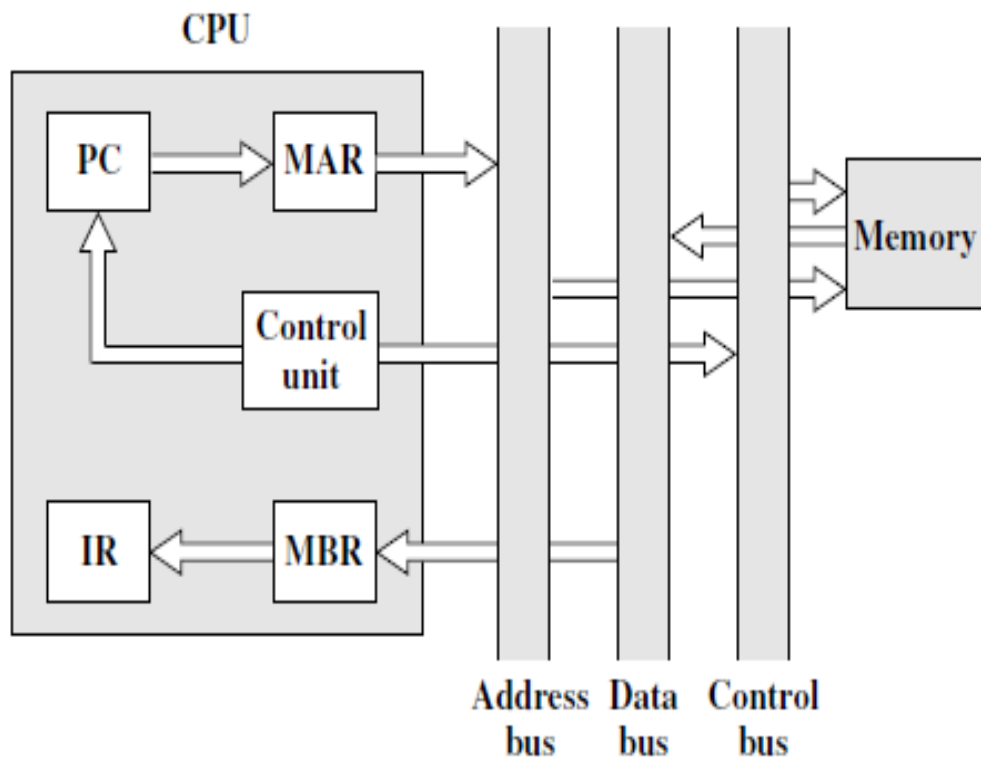
4.Execute: Interpret the opcode and perform the indicated operation

Indirect Cycle

- May require **memory access** to fetch operands.
- **Indirect addressing** requires **more memory accesses**.
- Can be thought of as additional instruction sub cycle.



Data Flow



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Figure 12.6 Data Flow, Fetch Cycle

The exact sequence of events during an instruction cycle depends on the design of the processor. Here indicate in general terms what must happen.

Let us assume that a processor that employs a memory address register (MAR), a memory buffer register (MBR), a program counter (PC), and an instruction register (IR).

During the *fetch cycle*, an instruction is read from memory. As shown in fig.

The flow of data during this cycle. The PC contains the address of the next instruction to be fetched. This address is moved to the MAR and placed on the address bus.

The control unit requests a memory read, and the result is placed on the data bus and copied into the MBR and then moved to the IR. Meanwhile, the PC is incremented by 1, preparatory for the next fetch.

Data Flow

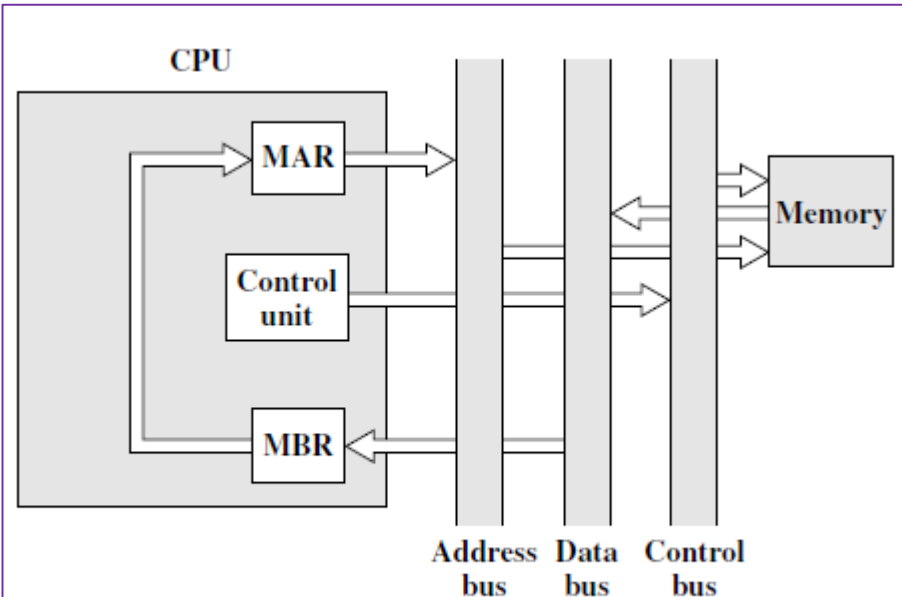


Figure 12.7 Data Flow, Indirect Cycle

Once the fetch cycle is over, the control unit examines the contents of the IR to determine if it contains an operand specifier using indirect addressing. If so, an *indirect cycle* is performed. As shown in Figure, this is a simple cycle. The rightmost

N bits of the MBR, which contain the address reference, are transferred to the MAR. Then the control unit requests a memory read, to get the desired address of the operand into the MBR.

The fetch and indirect cycles are simple and predictable.

The *execute cycle* takes many forms; the form depends on which of the various machine instructions is in the IR. This cycle may involve transferring data among registers, read or write from memory or I/O, and/or the invocation of the ALU.

Addressing Modes

Why Addressing Modes?

- An **addressing mode** specifies **how to calculate** the **effective memory address** of an operand by using information held in registers and/or constants contained within a machine instruction.
- Ex. Immediate addressing Mode:
- **MOV DL, 08H**
- **MOV CX, 4929 H**

Addressing Modes

1. Register Addressing
 2. Immediate Addressing
-

Group I : Addressing modes for register and immediate data

3. Direct Addressing
 4. Register Indirect Addressing
 5. Based Addressing
 6. Indexed Addressing
 7. Based Index Addressing
 8. String Addressing
-

Group II : Addressing modes for memory data

9. Direct I/O port Addressing
 10. Indirect I/O port Addressing
-

Group III : Addressing modes for I/O ports

11. Relative Addressing
-

Group IV : Relative Addressing mode

12. Implied Addressing

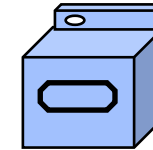
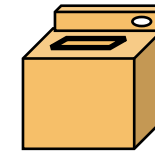
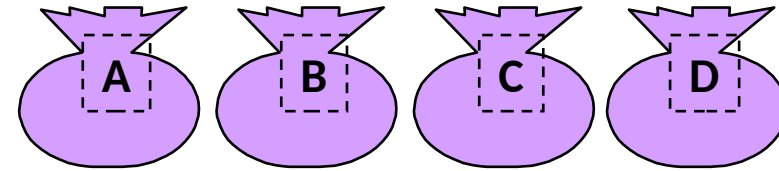
Group V : Implied Addressing mode

Pipelining

What is Pipelining?

- **Laundry Example**

1. Washer takes **30 minutes**
2. Dryer takes **40 minutes**
3. Folder takes **20 minutes**



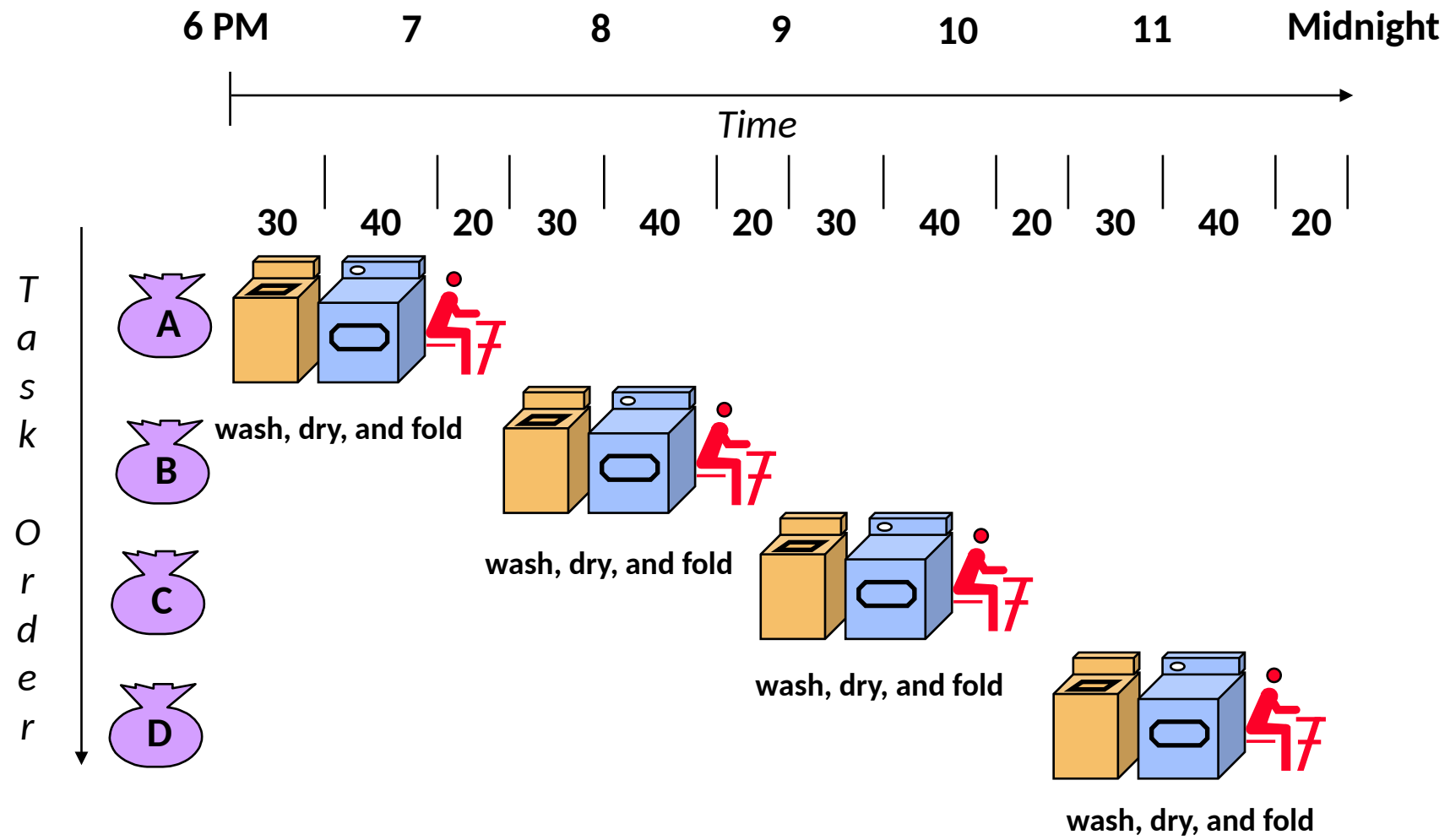
Wash 30 min

Dry 40 min

fold 20 min

Total= 1Hr 30 min
/ 90 min

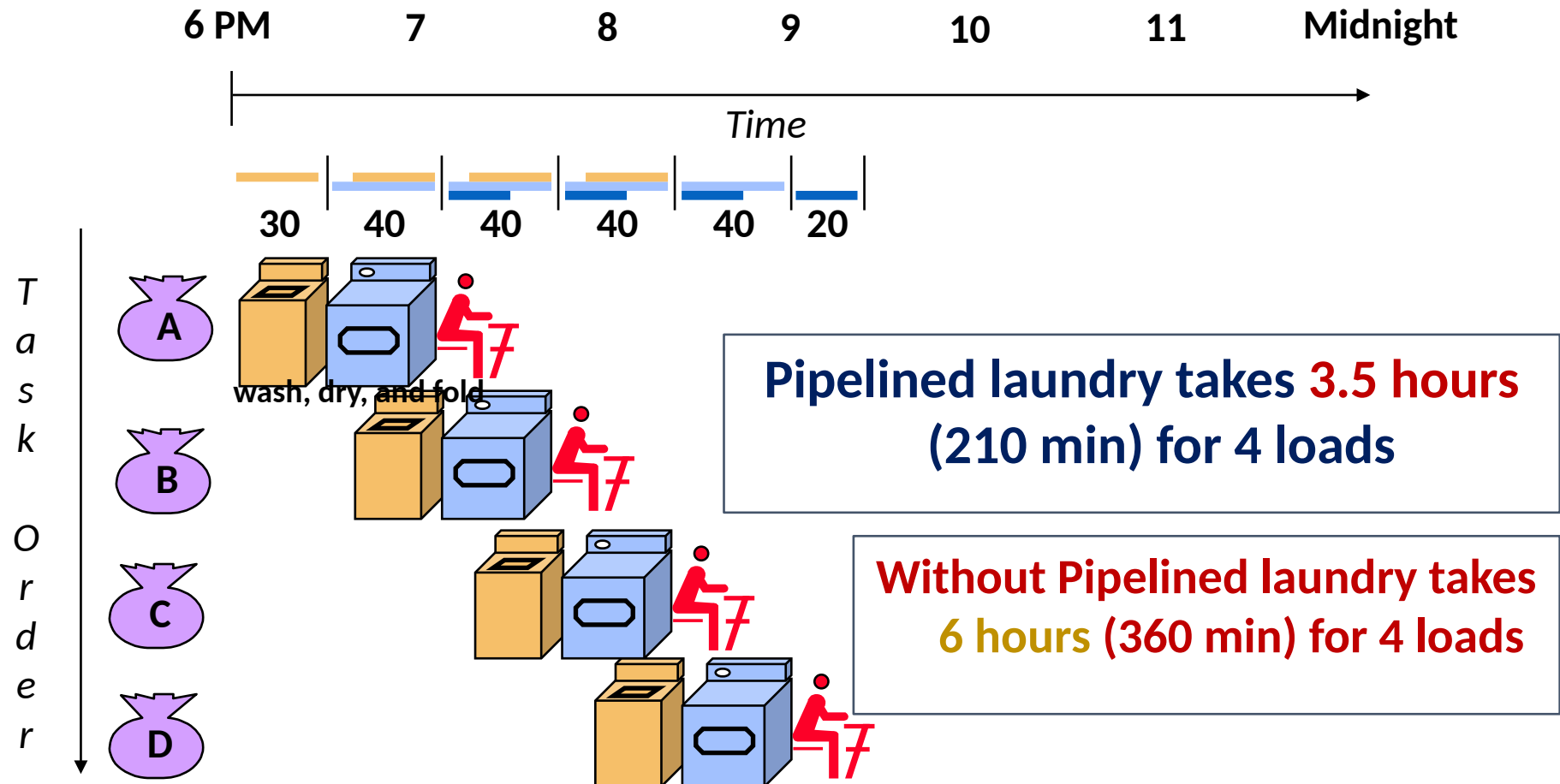
What is Pipelining?



Sequential laundry takes **6 hours for 4 loads (90 min+90 min + 90 min + 90 min= 360 Min)**

If they learned pipelining, how long would laundry take?

What is Pipelining ?



Instruction Pipelining

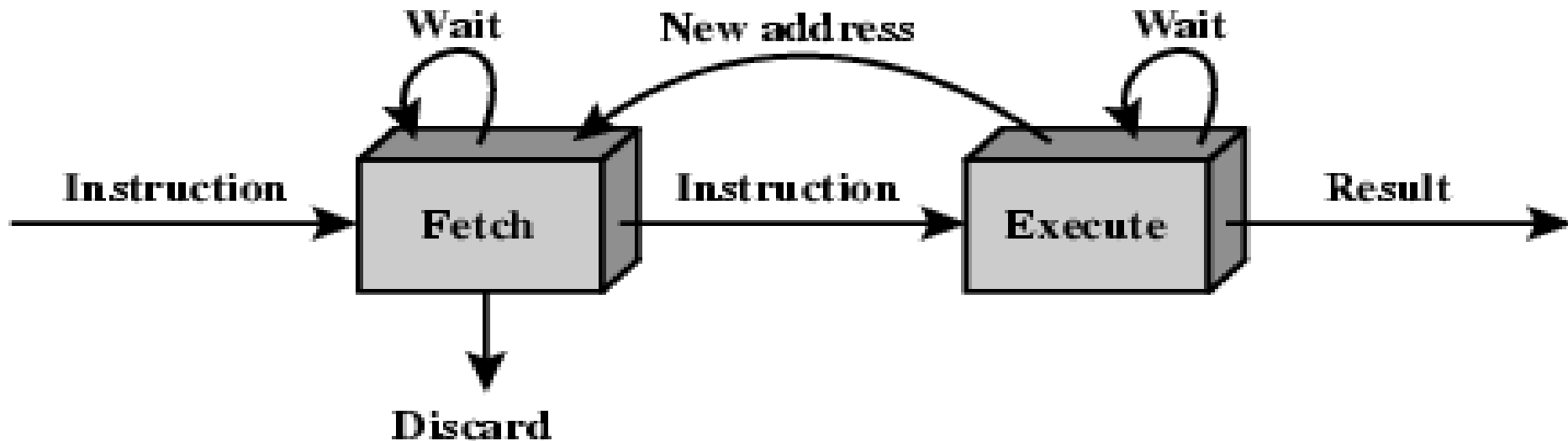
Instruction processing is subdivided:

- **Fetch and Execute instruction**
- **1st Stage – Fetch** an instruction and buffers it.
- **2nd Stage**—Temporarily free until first stage passes it the buffered instruction.
- While the second stage is executing the instruction, the first stage fetches and buffers the next instruction.
- Instruction prefetch or fetch overlap.
- Purpose?→ To speed up instruction execution.

Two-Stage Instruction Pipeline (8086)



(a) Simplified view

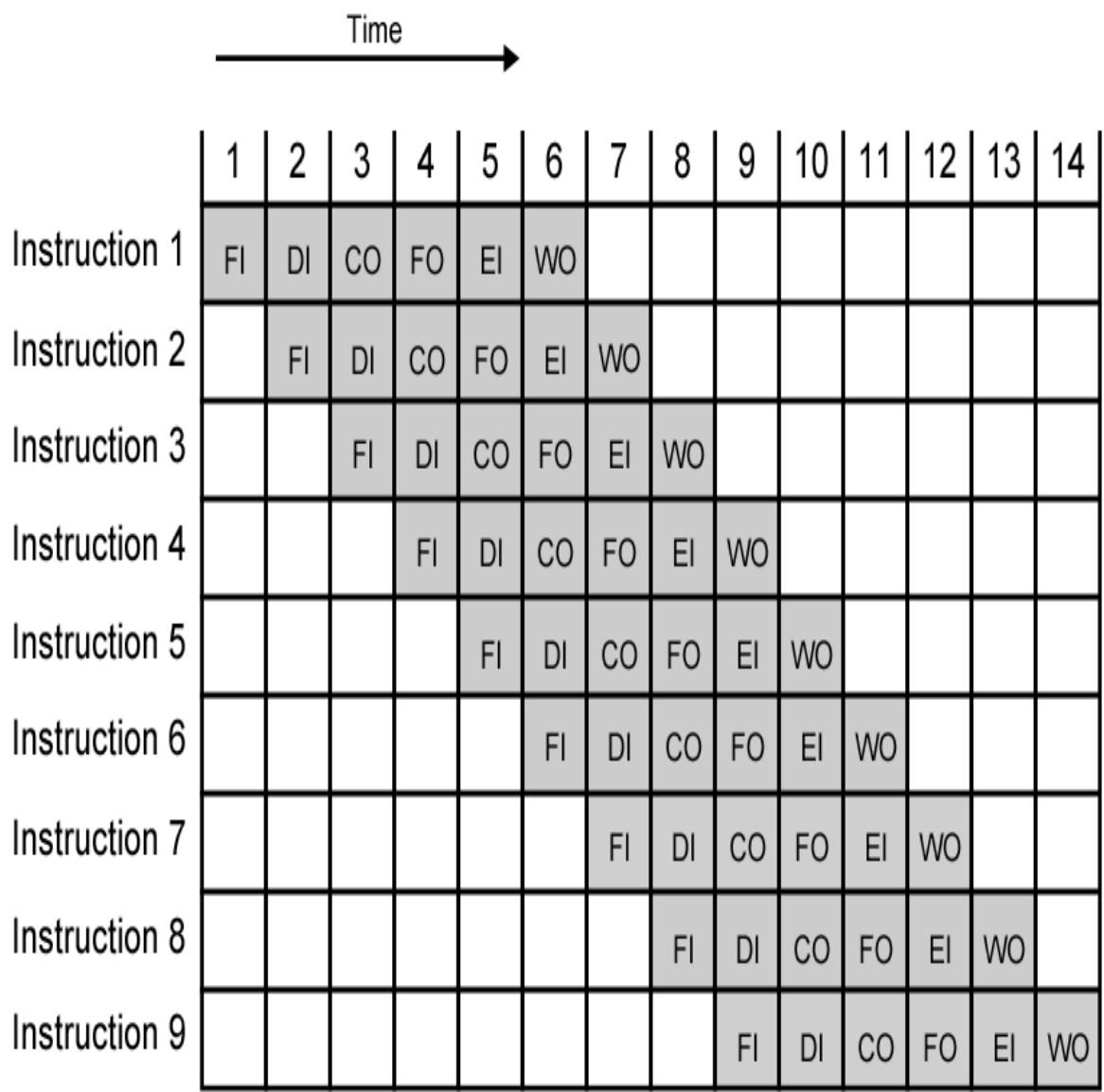


(b) Expanded view

Instruction Processing

1. Fetch instruction (FI)
 2. Decode instruction (DI)
 3. Calculate operands (CO)
 4. Fetch operands (FO)
 5. Execute instruction (EI)
 6. Write operand (WO)
- Successive instructions in a program sequence will overlap in execution.

Timing Diagram for Instruction Pipeline Operation

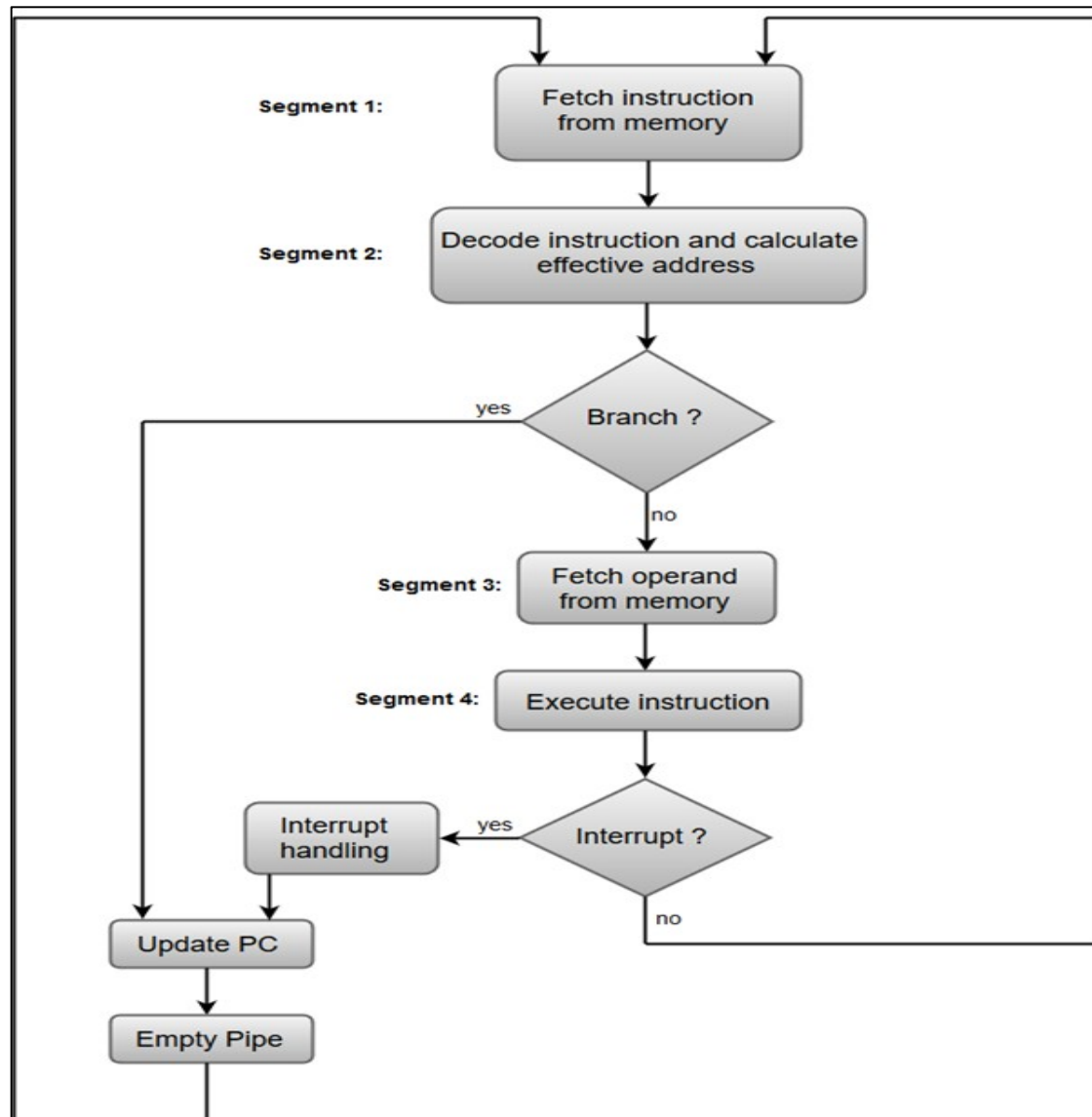


- 1. **Fetch instruction (FI)**
- 2. **Decode instruction (DI)**
- 3. **Calculate operands (CO)**
- 4. **Fetch operands (FO)**
- 5. **Execute instruction (EI)**
- 6. **Write operand (WO)**

14 clk

$6 \times 9 = 54$

Instruction Pipeline



1. **FI:** Fetch instruction
2. **DI:** Decode instruction
3. **CO:** Calculate operands
4. **FO:** Fetch operands
5. **EI:** Execute instruction
6. **WO:** Write operand

MOV AL,BL

30 JNZ 1000h

I15 1000:SUB BL,CL

1500 JZ 2000

2000: ADD AL,BL

Alternative Pipeline description

Time ↓

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I8	I7	I6	I5	I4	I3
9	I9	I8	I7	I6	I5	I4
10		I9	I8	I7	I6	I5
11			I9	I8	I7	I6
12				I9	I8	I7
13					I9	I8
14						I9

(a) No branches

	FI	DI	CO	FO	EI	WO
1	I1					
2	I2	I1				
3	I3	I2	I1			
4	I4	I3	I2	I1		
5	I5	I4	I3	I2	I1	
6	I6	I5	I4	I3	I2	I1
7	I7	I6	I5	I4	I3	I2
8	I15					I3
9	I16	I15				
10		I16	I15			
11			I16	I15		
12				I16	I15	
13					I16	I15
14						I16

(b) With conditional branch

Pipeline Hazards

- Pipeline hazards are situations that **prevent the next instruction** in the instruction stream from executing during its designated clock cycles.
- Any condition that causes a **stall** in the pipeline operations can be called a hazard.
- In the design of pipelined computer processors, a **pipeline stall** is a delay in execution of an instruction in order to resolve a hazard.
- **Structural Hazard**
- **Data Hazard**
- **Control Hazard**

Structural Hazard

Resource (Structural) Hazards - Two (or more) instructions in pipeline need same resource. Executed in serial rather than parallel for part of pipeline Also called *structural hazard*

		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instrucion	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			FI	DI	FO	EI	WO		
	I4				FI	DI	FO	EI	WO	
(a) Five-stage pipeline, ideal case										
		Clock cycle								
		1	2	3	4	5	6	7	8	9
Instrucion	I1	FI	DI	FO	EI	WO				
	I2		FI	DI	FO	EI	WO			
	I3			Idle	FI	DI	FO	EI	WO	
	I4					FI	DI	FO	EI	WO
(b) I1 source operand in memory										

Data Hazard

- **Read after Write (RAW), or true dependency**

- An instruction modifies a register or memory location
- Succeeding instruction reads data in that location
- Hazard if read takes place before write complete

$$\begin{aligned} A &\leftarrow 3 + A \\ B &\leftarrow 4 \times A \end{aligned}$$

- **Write after Read (WAR), or antidependency**

- An instruction reads a register or memory location
- Succeeding instruction writes to location
- Hazard if write completes before read takes place

$$\begin{aligned} &\text{MUL BX,AX} \\ &\text{SUB AX, CX} \end{aligned}$$

- **Write after Write (WAW), or output dependency**

- Two instructions both write to same location
- Hazard if writes take place in reverse of order intended sequence.

Data Hazard

- We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially.
- Hazard occurs
$$A \leftarrow 3 + A$$
$$B \leftarrow 4 \times A$$
- No hazard
$$A \leftarrow 5 \times C$$
$$B \leftarrow 20 + C$$
- When two operations depend on each other, they must be executed sequentially in the correct order.
- Another example:
$$\text{Mul } R2, R3, R4$$
$$\text{Add } R5, R4, R6$$

Data Hazard Diagram

- Conflict in access of an operand location
- Two instructions to be executed in sequence
- Both access a particular memory or register operand
- If in strict sequence, no problem occurs

		Clock cycle									
		1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX		FI	DI	FO	EI	WO					
SUB ECX, EAX			FI	DI	Idle		FO	EI	WO		
I3				FI			DI	FO	EI	WO	
I4							FI	DI	FO	EI	WO

Control Hazard (*Branch Hazard*)

Pipeline makes **wrong decision on branch prediction.**

- Brings instructions into pipeline that must subsequently be **discarded**
(CALL, JUMP, LOOP, JC, JNC etc)
- Dealing with Branches
 - **-Multiple Streams**
 - **-Prefetch Branch Target**
 - **-Loop Buffer**
 - **-Branch Prediction**
 - **-Delayed Branching**