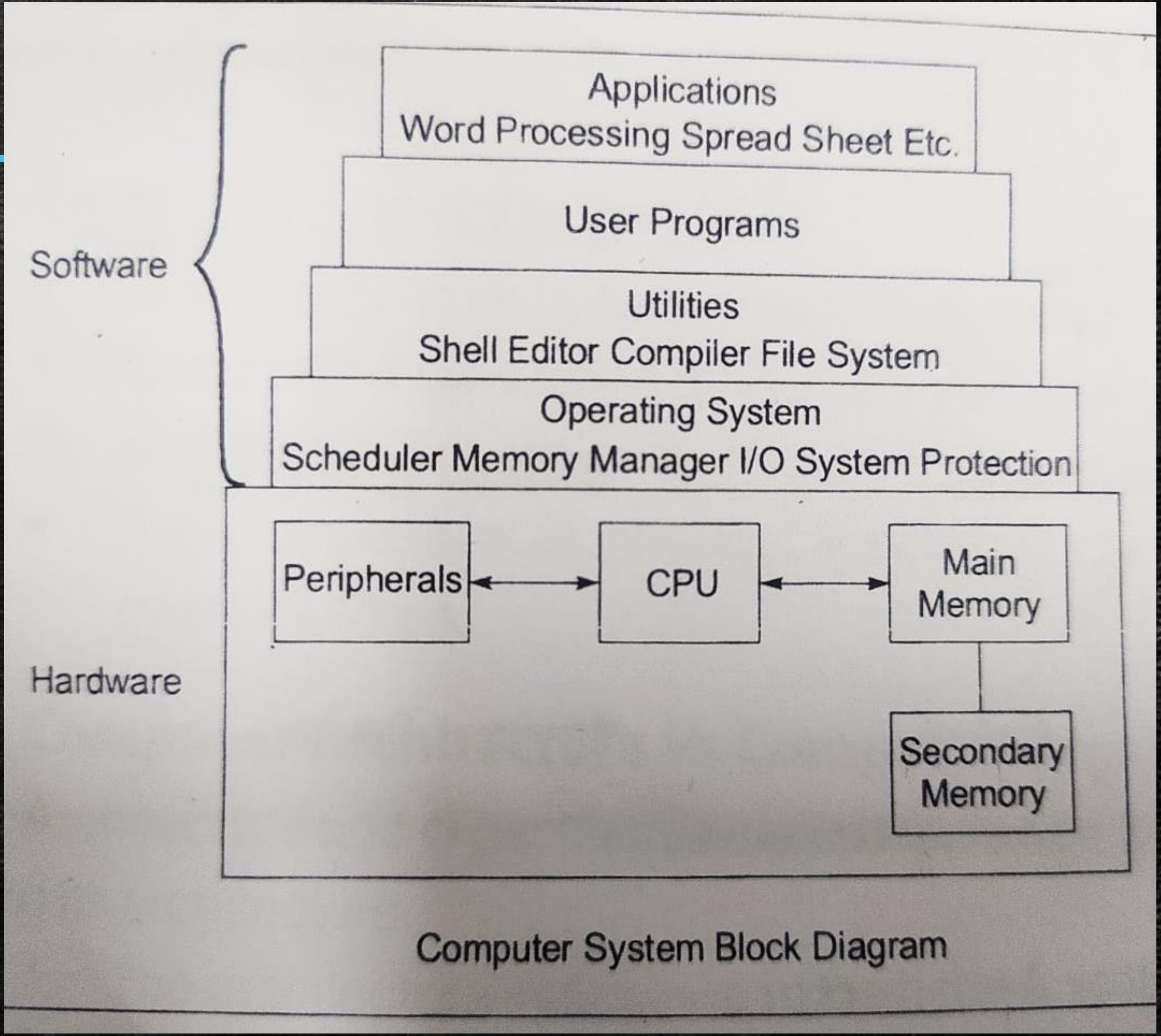
Computer Organization & Architecture

Pushpendra Kumar Pateriya

Computer System

- Hardware
- Software

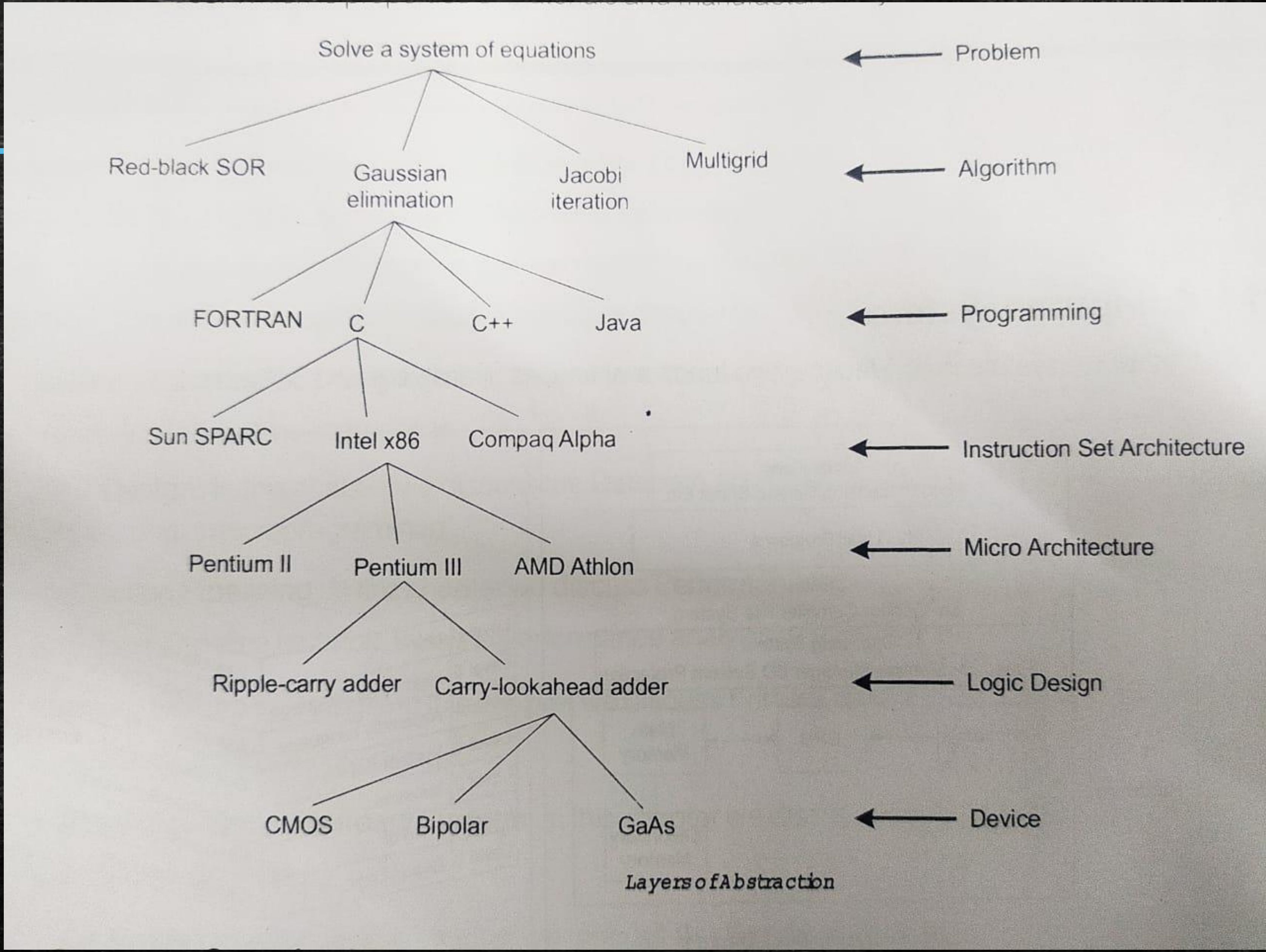
Computer System Block Diagram



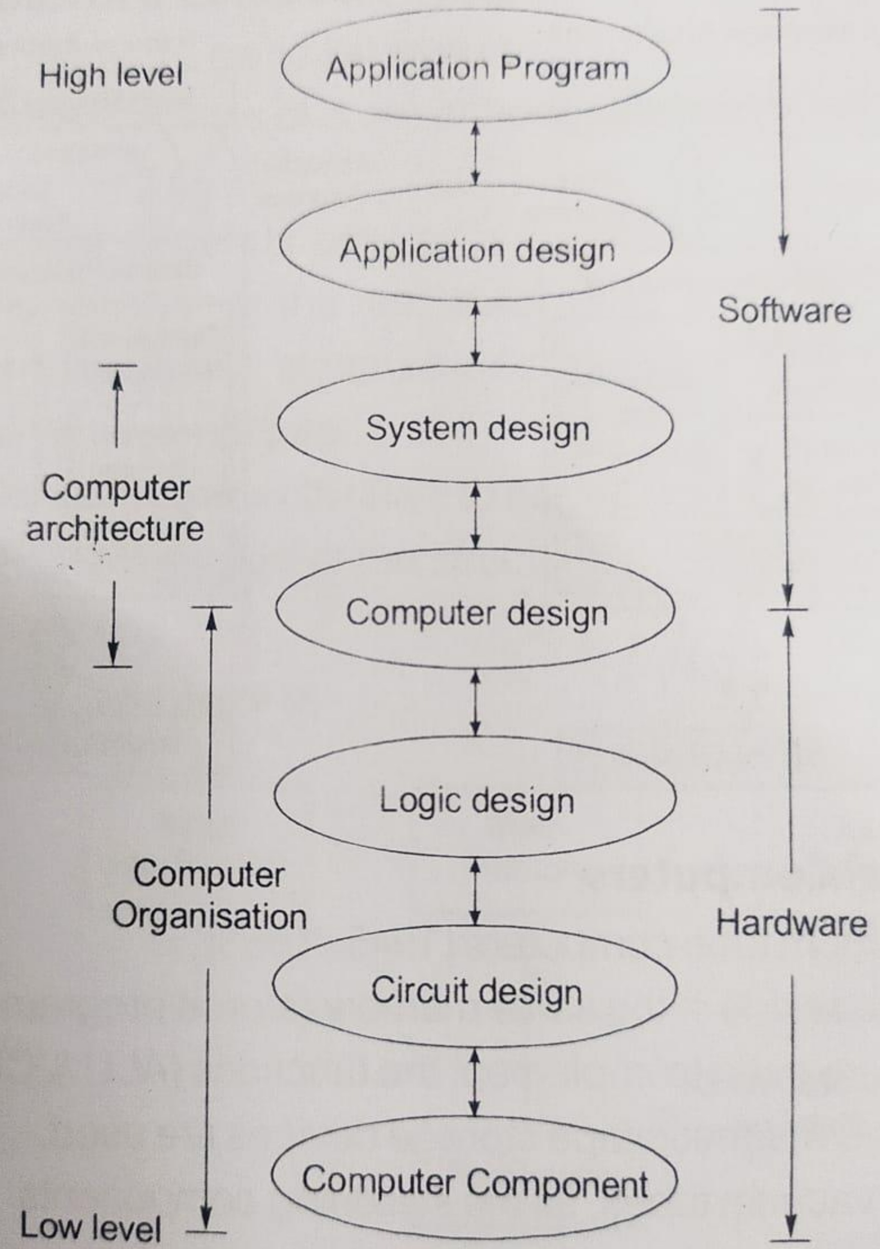
Software
Applications
Word Processing Spread Sheet Etc.
User Programs
Utilities
Shell Editor Compiler File System
Operating System
Scheduler Memory Manager I/O System Protection
Hardware
Peripherals
CPU
Main Memory
Secondary Memory

Computer System Block Diagram

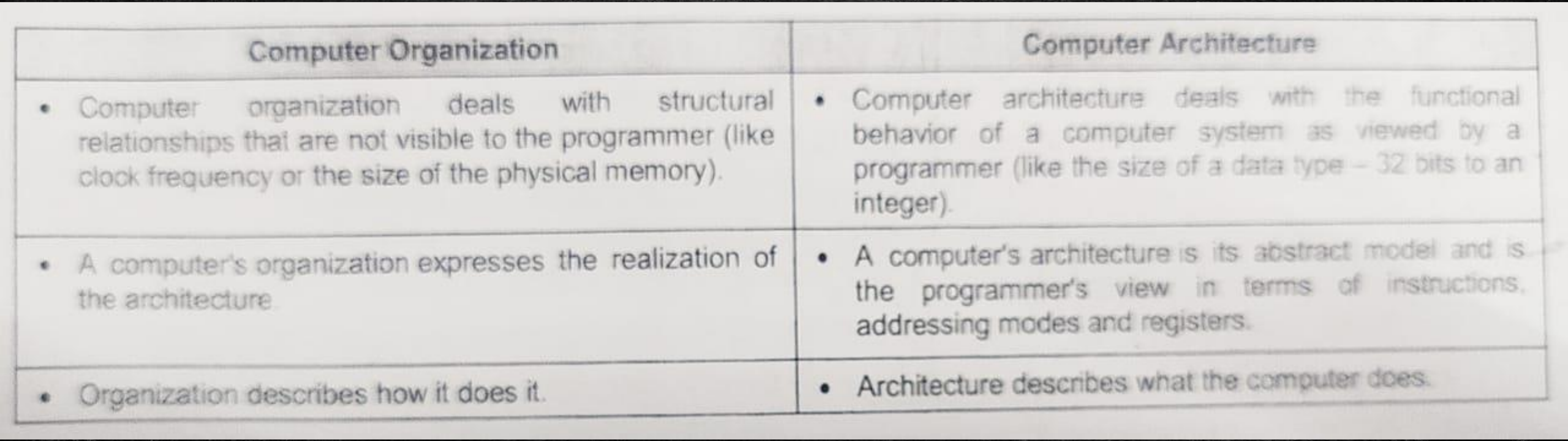
Layers of Abstraction



LayersofAbstraction

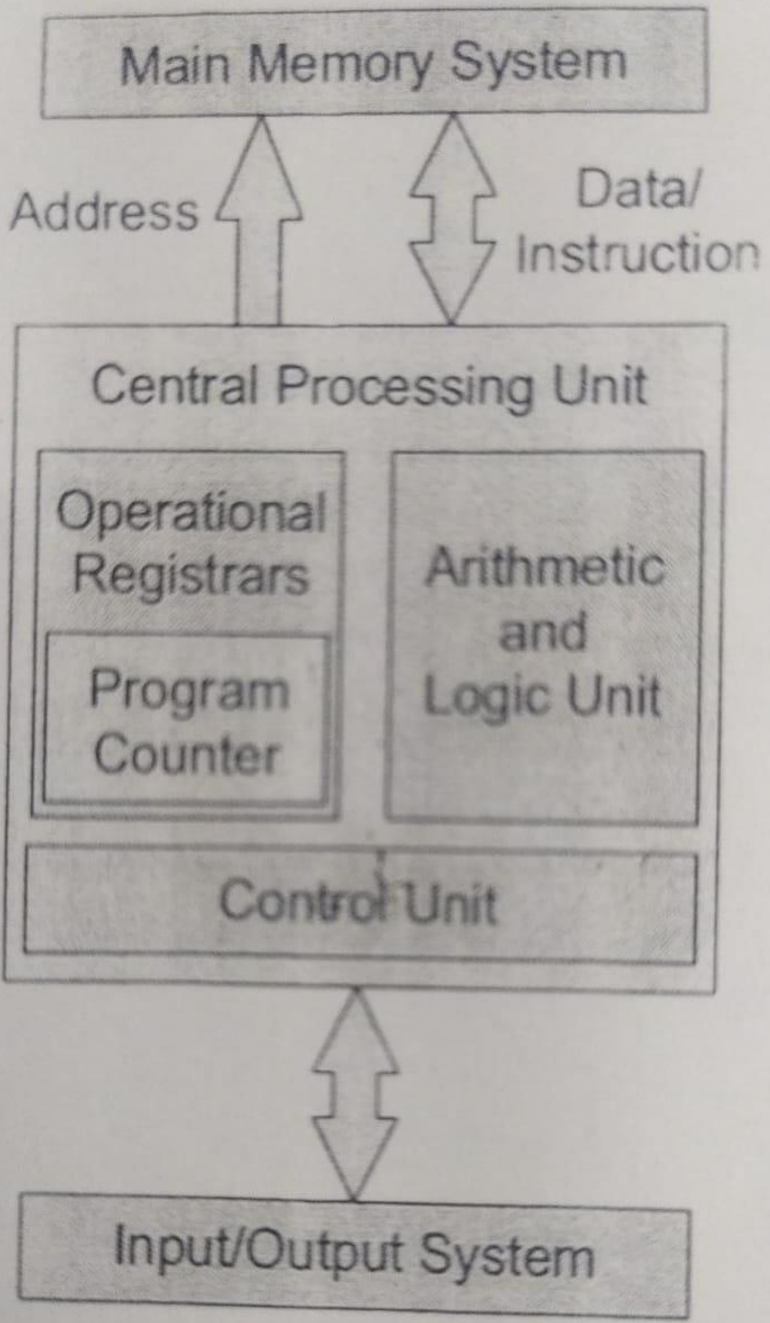


Computer System

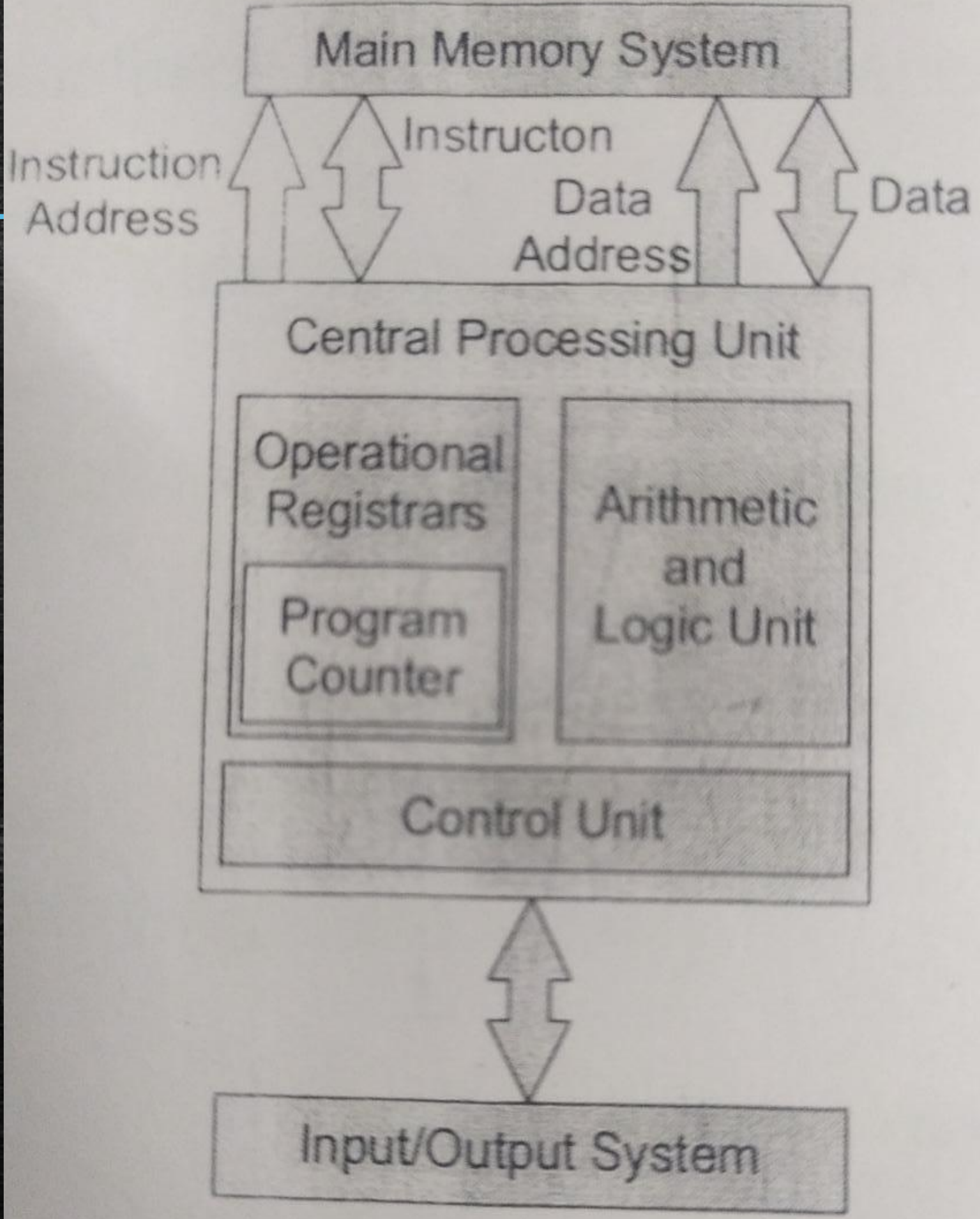


Computer Organization	Computer Architecture
• Computer organization deals with structural relationships that are not visible to the programmer (like clock frequency or the size of the physical memory).	• Computer architecture deals with the functional behavior of a computer system as viewed by a programmer (like the size of a data type – 32 bits to an integer).
• A computer's organization expresses the realization of the architecture.	• A computer's architecture is its abstract model and is the programmer's view in terms of instructions, addressing modes and registers.
• Organization describes how it does it.	• Architecture describes what the computer does.

Von Neumann Architecture



Harvard Architecture



Evolution of Digital Computers

- First Generation: Vacuum tube computers (1945 – 1953)
  - Program and data resides in same memory
  - Magnetic tapes used for storage
  - Assembly level language is used
- Second Generation: Transistorized computers (1954 – 1965)
  - Transistor were used to design ALU & CU
  - High level language is used (FORTRAN)
  - To convert HLL to MLL/LLL compiler were used
  - Separate I/O processor were developed to operate in parallel with CPU (improved performance

- Third Generation: Integrated circuit computers (1965 – 1980)
  - Multiprogramming, pipelining concepts were incorporated
  - DOS
  - Cache and virtual memory concepts were developed.
  - More than one circuit on a single silicon chip became available
- Forth Generation: VLSI computers (1980 – 2000)
  - CPU termed as microprocessor
  - INTEL, MOTOROLA, TEXAX, NATIONAL semiconductors started developing microprocessor
  - Workstations, Microprocessor (PC) & Notebook computers were developed
  - Interconnection of different computers for better communication LAN, MAN, and WAN
  - Computational speed increased 1000 times
  - Specialized computers like Digital Signal Processor were also developed

- Fifth Generation: System-on-chip (SOC) computers (2000 -)
  - E-commerce, E-banking, home office
  - ARM, AMD, INTEL, MOTOROLA
  - High speed processor – GHz speed
  - Because of submicron IC technology lot of added features in small size.

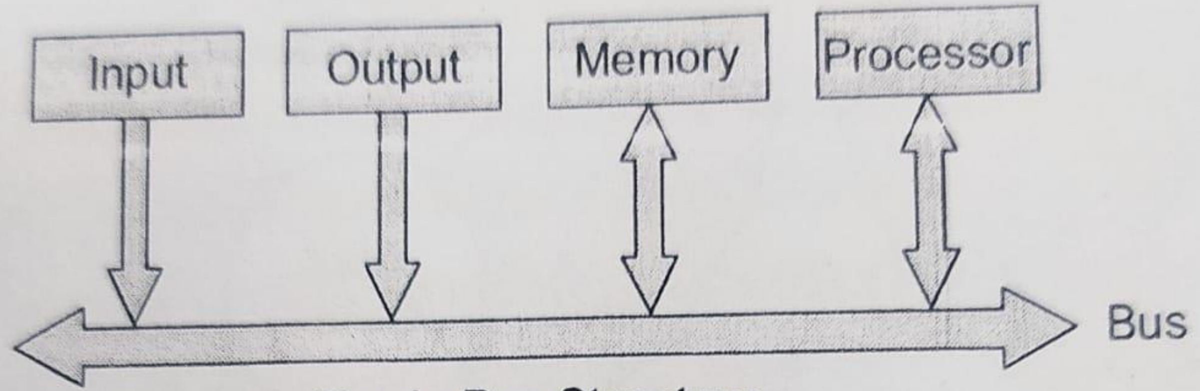
Components of Computer Structure

- Input Unit:
  - Convert data from human language to machine language (input device)
  - Keyboard, mouse, Joystick, etc
- Output Unit
  - Monitor, Printer, LCD, LED, etc
- Memory Unit
  - Stores data and instructions required for processing
  - Stores the intermediate results obtain during processing
  - Stores final results before sending it to output unit

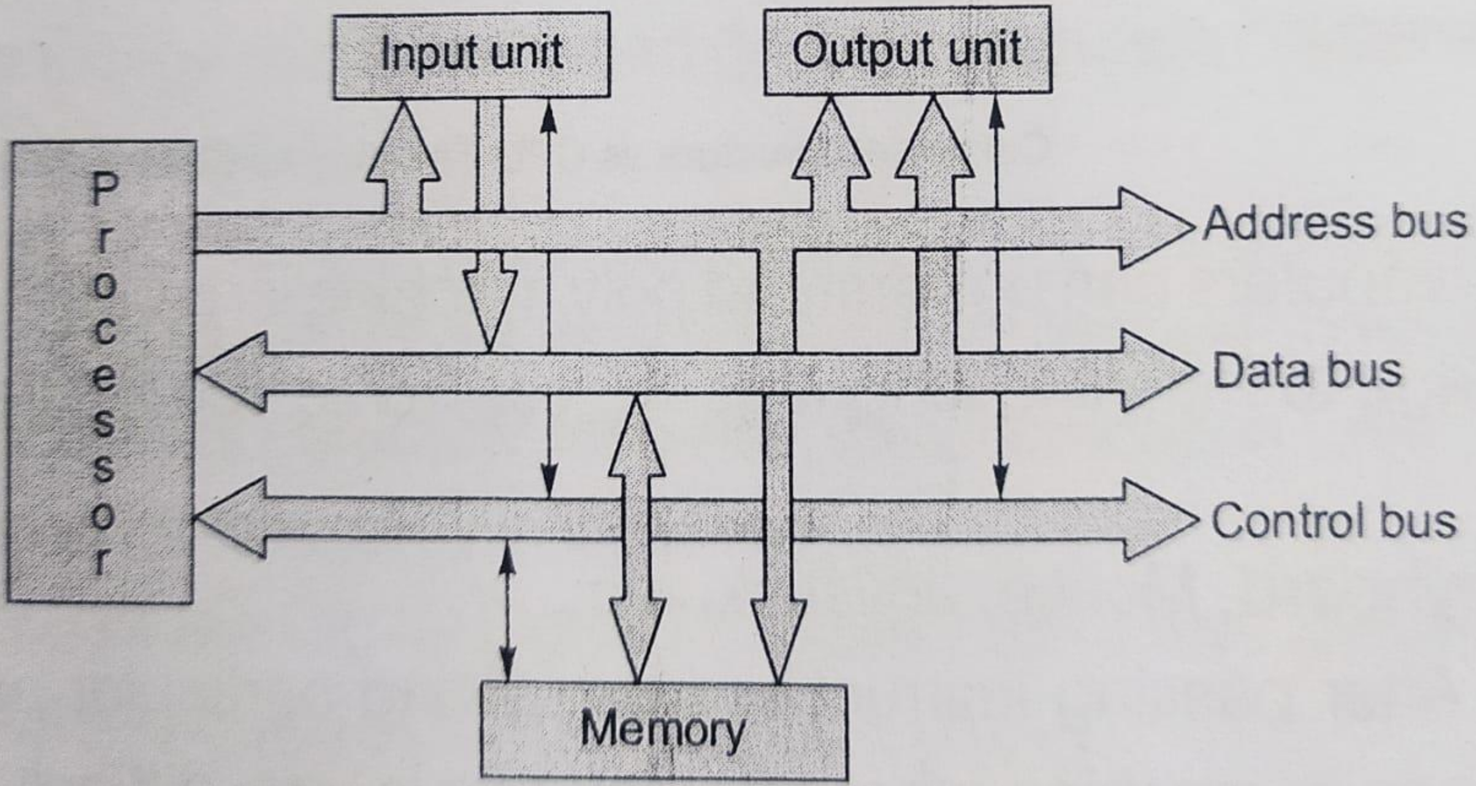
- CPU
  - Major structural components of CPU
    - Control unit (CU): Controls the operations of the CPU and hence the computer
    - Arithmetic and Logic Unit (ALU): Performs computer's data processing functions
    - Register: Provide storage internal to the CPU
    - CPU Interconnection: Communication among the control unit, ALU, and registers

Bus Structure

- Bus: it is a group of wires which carries information from CPU to peripherals or peripherals to CPU.
- The CPU and Memory are connected by Data Bus, Address Bus, and Control Bus
  - Address Bus:
    - Unidirectional
    - Carries address information bits from processor to peripherals
  - Data Bus
    - Bidirectional bus
    - Carries data information bit from processor to peripherals and vice-versa
  - Control Bus
    - Bidirectional bus
    - Carries control signals from processor to peripherals and vice-versa



Single Bus Structure

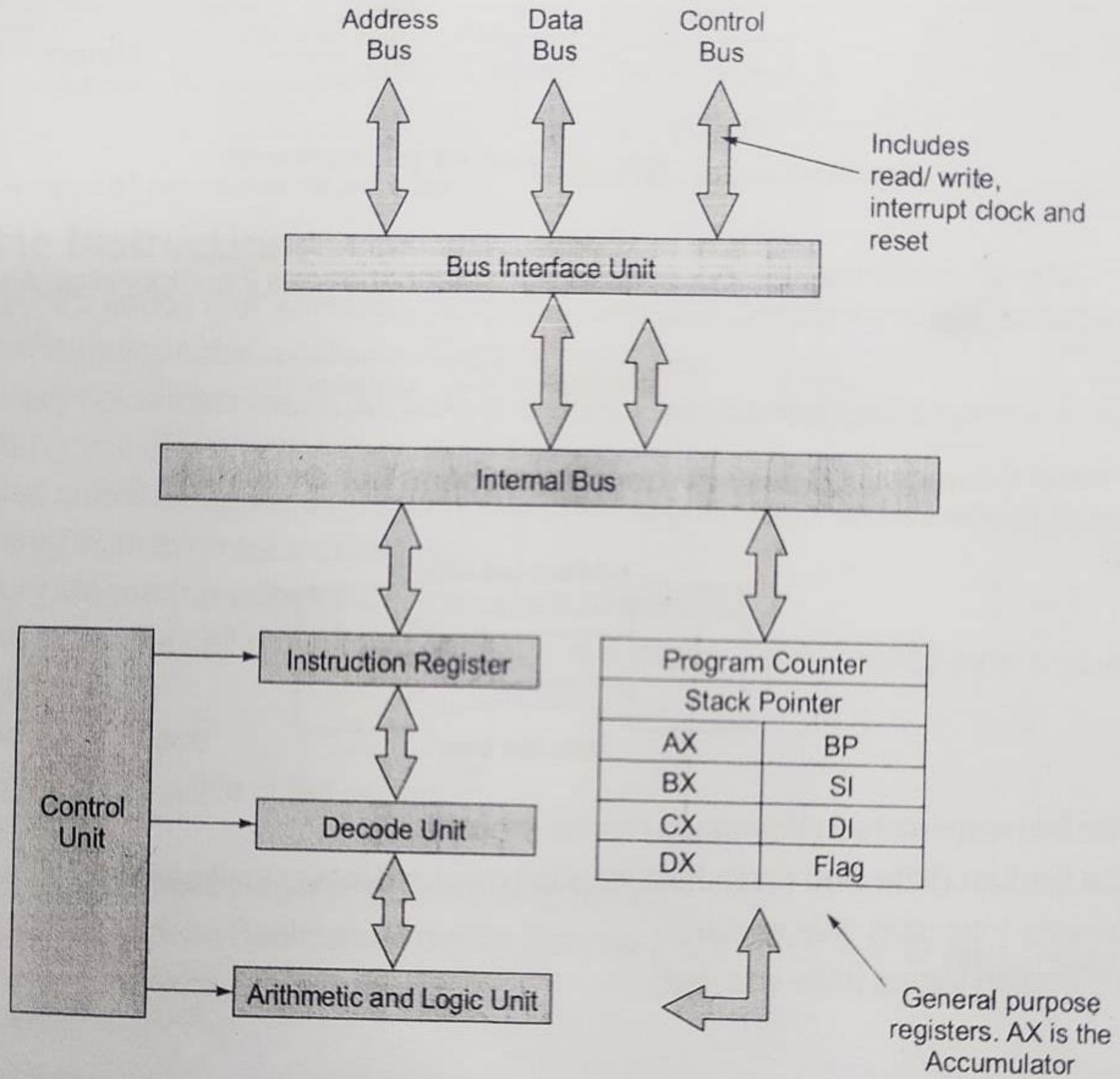


Internal view of Single Bus Structure

CISC and RISC Architectures

CISC	RISC
Large instruction set	Compact instruction set
Instruction formats are of different lengths	Instruction formats are of same lengths
Instructions perform both elementary and complex operations	Instructions perform elementary
Control unit is micro programmed	Control unit is simple and hardwired
Not pipelined or less pipelined	Pipelined
Single register set	Multiple register set
Numerous memory addressing options for operands	Less addressing modes
Include multi-clock complex instructions	Single clock, reduced instruction only
Memory-to-Memory: LOAD and STORE incorporated in instructions	Register-to-Register: LOAD and STORE independent instructions
Small code size, high cycles per second	Large code size, low cycles per second
Examples: VAX, PDP?11, Motorola 68000 family, Intel x86 architecture based processors	Examples: Apple iPods, Apple iPhone, ARM7, ARM based chip

General CPU Architecture (8086 Microprocessor)



Issues of Computer Design

- Speed and memory
- Speed match between memory and processor
- Handle bugs and errors
- Shared memory
- Disk access
- Better performance with reduced power

Data Storage in Memory

- Little Endian (Little end first)
- Big Endian (big end first)

Machine Instructions

- A set of binary codes that are recognized and executed directly by a processor is called a machine code. An individual machine code is called a machine instruction.
- Main memory holds instructions and Data.
- Each instruction/data require one or more memory locations depends on size of instruction/data
- The set of all codes recognized by a particular processor is known as instruction set.

Memory Models (how does the memory look to the CPU)

- Addressable cell size
- Alignment
- Address space
- Endianness (Little/Big)

Registers

- General Purpose
- Special Purpose: Program Counter (PC), Stack Pointer (SP), Input/Output Registers, Status Register

Data Types

Instructions

Encoding Machine instructions

- Machine Instructions are encoded in a binary pattern that specifies

1. Operation
2. Operation/addresses used for the operation
3. Where the result should be placed if any

Fields of an instruction

- OPCODE
- Operand/Address Field
- Mode

Instruction formats

- Three address instruction
- Two address instruction
- One address instruction
- Zero address instruction

Tree Address Instructions

OPCODE	Address	Address	Address

ADD	A	200	100

000001	1100	000…00011001000	000…0001100100

Example: ADD A, [200], [100]

Disadvantages

- Long instruction length
- May be three memory accesses needed for an instruction

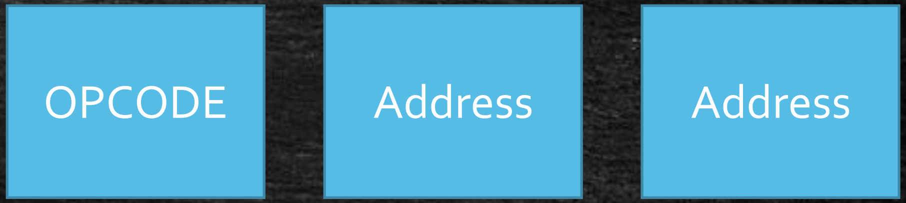
Write the tree address instructions for the following statement

- X = (A + B) X (C + D)
- Solution:

▪ ADD R1, A, B	R1 $\leftarrow$ M[A] + M[B]
▪ ADD R2, C, D	R2 $\leftarrow$ M[C] + M[D]
▪ MUL X, R1, R2	M[X] $\leftarrow$ R1 x R2

Two Address Instruction

- One OPCODE and Two Address Fields
- Example: ADD A, [100]



Disadvantages

- It may needs three memory accesses, eg: ADD [200] [100]

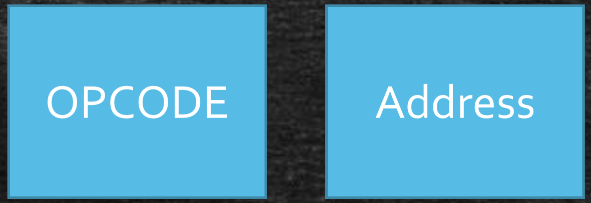
Write the two address instructions for the following statement

- X = (A + B) X (C + D)
- Solution:

MOV R1, A	R1 ← M[A]
ADD R1, B	R1 ← R1 + M[B]
MOV R2, C	R2 ← M[C]
ADD R2, D	R2 ← R2 + M[D]
MUL R1, R2	R1 ← R1 X R2
MOV X, R1	M[X] ← R1

One address instructions

- Example: ADD [100]



Advantages

- Shorter instruction
- Eliminates two memory accesses
- Faster accessing location inside processor than memory

Disadvantages

- Only one location for one operand and result
- Still it may need one memory access

Write the one address instructions for the following

- X = (A + B) X (C + D)

Solution:

LOAD A;	AC ← M[A]
PUSH A;	TOS ← A
PUSH B;	TOS ← B
ADD ;	TOS ← (A + B)
PUSH C;	TOS ← C
PUSH D;	TOS ← D
ADD ;	TOS ← (C + D)
MUL ;	TOS ← (C+D) x (A+B)
POP X;	M[X] ← TOS

Zero Address Instruction

OPCODE

- Example: Add ;
- it adds top two elements of the stack and stores result at top of stack

Write the zero address instructions for the following statement:

- X = (A + B) X (C + D)

Solution:

LOAD A;	AC ← M[A]
PUSH A;	TOS ← A
PUSH B;	TOS ← B
ADD ;	TOS ← (A + B)
PUSH C;	TOS ← C
PUSH D;	TOS ← D
ADD ;	TOS ← (C + D)
MUL ;	TOS ← (C+D) x (A+B)
POP X;	M[X] ← TOS

Addressing modes

Types of Addressing Modes

- Implied Mode
- Immediate Mode
- Register Mode
- Register Indirect
- Direct (Absolute) Address Mode
- Indirect Address Mode
- Indexed Addressing Mode
- Base with Index Address Mode
- Relative Address Mode
- Auto Increment and Auto Decrement Modes

Implied Mode

- The actual specified implicitly in the definition of the instruction
- Example: Zero-address instruction (stack operations ADD and SUB)

Immediate Mode

- The actual operand (data) is specified in the instruction
- Example: MOV R1, 123
- The operand 123 is held in the instruction
- Example (High Level): x = 123

Register Mode

- The instruction specifies a register that has operand
- Example (Assembly): MOV R1, R2
- Instruction has register R1 and R2 operand
- Example (High Level): x = y

Register Indirect Mode

- The instruction specifies a register that contains the address of the operand
- The operand is held in memory. The address of the operand location is held in a register which is specified in instruction.
- Example (Assembly): ADD R1, (R2)
- Instruction has register R2 and R2 has memory address of operand.
- Example (High Level): int *x;

Direct (Absolute) Address Mode

- Example: MOV R1, [1000]
- The operand is in memory and the memory address 1000 of the operand is held in instruction.

Indirect Address Mode

- Example (Assembly): ADD R1, (1000)
- Instruction has memory address 1000. At location 1000, the memory address of operand is available.
- Example (high Level):
  - int *x;
  - int y, z=0;
  - x = &y;
  - *x = 123;
  - z = z + *x; // z is similar to R1, x is similar to 1000.

Indexed Addressing Mode

- The effective address is the sum of an index register and the offset field (constant). It is similar to register indirect addressing except an offset held in the instruction is added to register contents to compute the effective address. The indexed addressing mode is useful in dealing with lists and arrays.
- Example: MOV R1, 2(R2);
  - R1 stores operand from memory address: 2 + (R2)
  - R2 has memory address and offset is 2.
  - By adding 2 and content of R2 gives the effective address of operand
- Example (High Level): int x[10], y; y= x[2];

Base with Index Address Mode

- The effective address is the sum of contents of based register and index register.
- Example(Assembly): MOV R1, (R2, R3)
  - Register R2 contain the base address and R3 contain the index.
  - The effective address is the sum of the contents of registers R2 and R3.
  - Example (High Level): int x[10], i=2, y; y=x[i]; //where y, x, and I are similar to R1, R2, and R3 respectively.

Base with Index and offset Mode

- It uses two registers Ri and Rj plus a constant X. The effective address is the sum of the constant X and the contents of registers Ri and Rj.
- Example(Assembly): MOV R1, 2(R2, R3)
- Register R2 contains the base address and R3 contain the index. The effective address is the sum of the constant value 2, the contents of register R2 and R3.
- Example(High Level): int x[10], i=2, y; y=z[i+3]; // where y, x, and I are similar to R1, R2, and R3 respectively

Relative Address Mode

- The effective address is the sum of the address field (offset) and the content of the program counter (PC). Specify target location as number of instructions from branch/jump instruction.
- Makes code relocatable (i.e. code can be loaded anywhere in memory without altering branch/jump instructions).
- The number of locations to the target is held in the instruction as an offset.
- Example: Assume PC holding 100 while executing the following instruction. JUMP 123; control will transfer to 110 by adding 100 with offset 23. In high level (C) programming, using goto relative addressing can be done.

Auto increment and auto decrement modes

- Post Auto increment mode:
  - The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. It is denoted as $(R_i)+$
  - Example: MOV R1, (R2)+
- Pre Auto Increment Mode:
  - Before accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. Then effective address of the operand is the incremented contents of a register. It is denoted as $+(R_i)$
  - Example: MOV R1, +(R2)

- Post Auto Decrement mode:
  - The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically decremented to point to the next item in a list. It is denoted as $(R_i)^-$
  - Example: MOV R1, (R2)-
- Pre Auto Decrement Mode:
  - Before accessing the operand, the contents of this register are automatically decremented to point to the next item in a list. Then effective address of the operand is the decremented contents of a register. It is denoted as $-(R_i)$
  - Example: MOV R1, -(R2)

Table for Effective Address calculation of operands using Addressing Modes

Name	Example	Operation	Syntax (Assembly)	Effective Address Computation	Usage
Immediate	ADD R_4, 3	R_4: = R_4 + 3	# Value	Operand = Value	Constants
Register	ADD R_4, R_3	R_4: = R_4 + R_3	R_i	EA = R_i	Temporary variables
Register Indirect	ADD R_4, [R_1]	R_4: = R_4 + M[R_1]	(R_i)	EA = (R_i)	Pointers
Direct (Absolute)	ADD R_4, [100]	R_4: = R_4 + M[100]	LOC	EA = LOC	Global variables
Memory Indirect	ADD R_4, @ [100]	R_4: = R_4 + M[M [100]]	(LOC)	EA = (LOC)	Pointers
Displacement (Based)	ADD R_4, [R_1+100]	R_4: = R_4 + M[R_1 + 100]	X(R_i)	EA = (R_i) + X	Local Variables
Based with Index	ADD R_4, [R_1+R_2]	R_4: = R_4 + M[R_1 + R_2]	(R_i, R_j)	EA = (R_i) + (R_j)	Arrays
Based with index and offset	ADD R_4, [R_1+R_2+8]	R_4: = R_4 + M[R_1 + R_2 + 8]	X(R_i, R_j)	EA = (R_i) + (R_j) + X	Arrays
Relative	ADD R_1, [R_2+PC]	R_4: = R_4 + M[R_2 + PC]	X(PC)	EA = (PC) + X	Static local variables
Auto-increment (post)	ADD R_4, [R_2]+	R_4: = R_4+M[R_2] R_2: = R_2 + d	(R_i)+	EA = (R_i); Increament R_i	Stack operations
Auto-increment (pre)	ADD R_4, +[R_2]	R_2: = R_2 + d R_4: = R_4+M[R_2]	+(R_i)	Increament R_i EA = (R_i);	Stack operations
Auto-decrement (post)	ADD R_4, [R_2]–	R_4: = R_4 + M[R_2] R_2: = R_2 – d	(R_i) –	EA = (R_i); Decrement R_i	Stack operations
Auto-decrement (pre)	ADD R_4, –[R_2]	R_2: = R_2 – d R_4: = R_4 + M[R_2]	–(R_i)	Decrement R_i EA = (R_i);	Stack operations

Question

- An address field in an instruction contains decimal value 14. Where is the corresponding operand located for:

1. Immediate addressing?
2. Direct addressing?
3. Indirect addressing?
4. Register addressing?
5. Register indirect addressing?

Solution

1. 14 (the address field)
2. Memory location 14
3. The memory location whose address is in memory location 14
4. Register 14
5. The memory location whose address is in register 14

Question

- Given the following memory values and a one address machine with an accumulator:
  - Word 20 contains 40
  - Word 30 contains 50
  - Word 40 contains 60
  - Word 50 contains 70

What values do the following instructions load into the accumulator?

1. Load IMMEDIATE 20
2. Load DIRECT 20
3. Load INDIRECT 20
4. Load IMMEDIATE 30
5. Load DIRECT 30
6. Load INDIRECT 30

Solution:

1. 20
2. 40
3. 60
4. 30
5. 50
6. 70