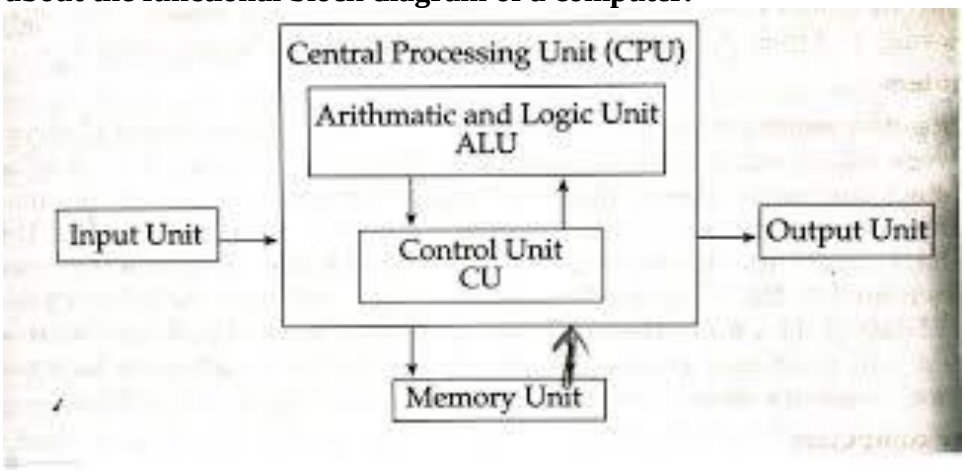
1. **Explain about the functional block diagram of a computer.**



2. **Compare between primary and secondary memory,**

Primary Memory	**Secondary Memory**
Temporary memory that directly interacts with the CPU for quick access to data and instructions.	Permanent memory used for long-term data storage and retrieval.
RAM (Random Access Memory) and ROM (Read-Only Memory).	Hard Disk Drive (HDD), Solid-State Drive (SSD), Optical Discs (CD/DVD), USB Drives.
Mostly volatile (RAM loses data when power is off). ROM is non-volatile.	Non-volatile (Data is retained even when power is off).
Faster Access time is in nanoseconds).	Slower (Access time is in milliseconds).
Smaller (Usually measured in gigabytes, e.g., 8GB or 16GB).	Larger (Measured in terabytes, e.g., 1TB, 2TB).
Stores data and instructions currently in use for quick access by the CPU.	Stores large volumes of data permanently for future use.
Directly accessible by the CPU.	Needs to be loaded into primary memory before the CPU can process it.
More expensive per unit of storage.	Cheaper per unit of storage.
RAM, ROM, Cache Memory.	HDD, SSD, USB Drives, CDs/DVDs.
Running applications and active processes.	Storing files, software, and backups.
Short-term (Only while the computer is on).	Long-term (Persistent storage).

3. **Subtract i) $(45)_{10} - (25)_{10}$ and ii) $(456)_{10} - (650)_{10}$ using 9's complement method and verify also the results with 10's complement method.**

9's complement of 25 -> 99−25=74
45+74 = 119 :: 119 -100 + 1(as there is a carry) = 20

10's complement 74+1 = 75
45+75 = 120
120-100 =20

4. **Classify five differences between arithmetic and shift microoperation.**

Arithmetic Microoperations	**Shift Microoperations**
Operations that perform basic arithmetic e.g.	shift the bits of a register to the left or right.

Arithmetic Microoperations	Shift Microoperations
addition, subtraction, increment, and decrement.	
Addition, Subtraction, Increment, Decrement.	Logical Shift (Left/Right), Arithmetic Shift, Circular Shift (Left/Right).
Alters the actual value of the data by performing arithmetic changes.	Changes the bit position without performing arithmetic calculations.
Used in ALU operations for performing arithmetic tasks.	Used for data movement, sign extension, and efficient arithmetic operations like multiplication/division.

5. Consider a value $(11011)_2$ stored in register. Identify the change in the value of register if 0 is inserted while right shift is executed and after that left shift is executed.

Right shift 01101 :: 13

Left shift 11010 :: 26

6. Categorize the basic instruction formats for I/O, Register and Memory reference instructions.

I/O instructions in Input Output Devices, fields are Opcode, Device Code, Control, Instaructions are IN, OUT, CHECK STATUS,

Register instructions in CPU Registers, fields are Opcode, Register, Control Bits Instructions are INR, CLEAR, COM, ROR

Memory reference instructions in Main Memory, fields are Opcode, Address, Mode, Instruction are LOAD, STORE, ADD, SUB

7. Explain the concept of microinstruction and control memory.

A microinstruction is a low-level instruction that specifies one or more micro-operations required to implement a single machine-level instruction. (Please refer slides)

Control memory is a specialized read-only memory (ROM) or RAM that stores the microinstructions required to control the processor's internal operations. It plays a crucial role in a microprogrammed control unit by storing microprograms, which are collections of microinstructions that define how each machine-level instruction is executed. Each microinstruction generates control signals that guide the processor's internal components, such as the arithmetic logic unit (ALU), registers, and buses, to perform specific micro-operations during each clock cycle. When a machine-level instruction is fetched from the main memory, the control unit decodes it and uses the control memory to access the corresponding microprogram. This microprogram breaks down the complex instruction into a sequence of simpler steps, ensuring precise and efficient execution. Acting as an intermediary between instruction decoding and hardware execution, control memory enables a structured and flexible way to manage a CPU's operations. This approach is particularly useful in microprogrammed processors, as it allows for easier modifications and updates to the instruction set without redesigning the physical hardware.

8. **Compare in five points about Hardwired and Micro programmed control unit.**

-- in unit 3 not covered

9. Demonstrate five points about advantage of RISC over CISC architecture.

--refer slides

10. **Explain about stored program organization. Illustrate with reference to 4096*16 memory, the flowchart of Instruction cycle.**

--

11. A processor with a 2 GHz speed is executing a program of 109 instructions. The following are the execution times observable:
• 30% of instructions take 2 clock cycles
• 40% take 3 clock cycles
• 20% take 5 clock cycles
• 10% take 8 clock cycles
What is the average CPI (Cycles per instruction) of the system? What will be the improvement in the execution time if the speed of the same processor is increased to 3 GHz?

Average CPI=(Fraction of Instructions)×(Cycles per Instruction)

Average CPI=(0.30×2)+(0.40×3)+(0.20×5)+(0.10×8) = 0.6+1.2+1.0+0.8=3.6

Execution Time=Instructions × CPI × Clock Cycle Time

Instructions =109
CPI = 3.6
clock cycle time = 1/ clock speed = $1/(2*10^9)$
Execution Time = 109 X 3.6 X $1/(2*10^9)$

For 3Ghz
Execution Time = 109 X 3.6 X $1/(3*10^9)$

12. **Explain the operation of register stack and memory stack.**

Register stack

The register stack operates on the Last-In, First-Out (LIFO) principle, meaning the most recently added item is the first to be removed. The two main operations performed on the register stack are the push and pop operations. During a push operation, data is stored in the stack, and the stack pointer (SP) is updated to point to the next available register. This allows the CPU to temporarily store intermediate results, addresses, or variables. In a pop operation, the most recently stored value is retrieved from the stack, and the stack pointer is adjusted backward. The Top of Stack (TOS) always points to the last pushed value, making retrieval fast and efficient. However, register stacks are limited by the number of registers available. If the stack exceeds this capacity, an overflow error occurs. Conversely, if a pop operation is attempted on an empty stack, an underflow error occurs.

Memory Stack

The memory stack also functions on the LIFO principle but is located in the main memory (RAM), allowing it to hold larger amounts of data. The stack pointer keeps track of the memory address corresponding to the current top of the stack. When a push operation occurs, the data is stored at the stack pointer location, and the pointer moves to the next memory address. During a pop operation, the last stored value is retrieved, and the pointer moves backward to reflect the new stack top. The memory stack is crucial for handling function calls in programming. When a function is called, the return address (where the program should resume after the function finishes) is pushed onto the stack. Parameters and local variables are also stored here. When the function completes, a return operation pops the stored address, allowing the program to resume execution. While the memory stack can store more data than a register stack, accessing it is slower due to memory latency, and exceeding the allocated memory space results in a stack overflow error.

13. Discuss about registers in computers and show with a diagram, how they are connected through a common bus.

--refer slides

14. **Discuss about register stack and memory stack. Use a neat diagram to show the PUSH and POP operation in terms of microoperations. Implement (A+B)*[C*(D+E)+F] in RPN.**
-- refer slide COA-5 slide 4-8

(A+B)*[C*(D+E)+F]
reverse polish is postfix notation
step 1 (AB+)*[C*(DE+)+F
step 2 X*[C*Y+F] // (AB+) as X and (DE+) as Y
step 3 X*[(CY*)+F]
step 4 X*[Z +F] // (CY*) as Z
step 5 X*[ZF+]
step 6 X*M // (ZF+) as M
step 7 XM*

AB+ZF+*
AB+CY*F+*
AB+CDE+*F+* → this is the answer

15. **Discuss about arithmetic microoperation and describe how A-B can be performed.**
-- refer slide COA-3 slider no 9

16. The outputs of four registers R0, R1, R2 and R3 relate to a 4:1 Mux to the input of destination register R4. Each register is 8 bit long and the required transfer take place using timing interval T0 to T3.

T0: R4,←R0
T1:R4 ←R1
T2:R4 ←R2
T1:R4 ←R3

Demonstrate a hardware to implement this register transfer
--refer slide COA-3 slide 7

17. **Implement a hardware for logic micro-operation for A and B, A or B, A xor B and A complement.**
-- refer slide COA-3

18. Compare between Memory reference and register reference instructions.

Memory Reference Instructions	Register Reference Instructions
Instructions that access data from memory locations.	Instructions that manipulate data within CPU registers.
Operands are stored in main memory (RAM).	Operands are located in processor registers.
Slower due to external memory access time.	Faster due to quick access within the CPU.
Instruction like LDA (Load Accumulator), STA (Store Accumulator), ADD (Add from memory).	Instructions are CLA (Clear Accumulator), CMA (Complement Accumulator), INC (Increment).
Requires memory addresses for operand access.	Requires register names or codes for access.
Requires more clock cycles (due to memory read/write).	Requires fewer clock cycles (register operations are faster).
More complex due to interaction with memory bus.	Less complex as it only involves internal registers.
Used for loading, storing, and manipulating data in	Used for fast arithmetic, logic, and data transfer

Memory Reference Instructions	Register Reference Instructions
memory.	between registers.
Contains opcode, address, and mode bits.	Typically involves opcode and register codes.
Used in memory-intensive tasks like data storage and retrieval.	Used in arithmetic operations and CPU control tasks.

19. Show the status of the stack after each step when you perform the following operation 5*2+(3*4-4)+(2+4*4) on a stack based computer.

To make it first do as RPN (reverse polish notation) or postfix
5*2+(3*4-4)+(2+4*4)
step 1: (52*)+((34*)-4)+((2+44*))
step 2: X+(Y-4)+(2+Z) // X= 52*; Y= 34*; Z= 44*
Step 3: X+(Y4-)+(2Z+)
step 4: X+A+B // A= Y4-; B=2Z+
step 5: (XA+)+B
step 6: M+B // M = XA+
step 7 MB+

XA+B+
52*Y4-+2Z++
52*34*4-+244*++

Step	Operation	Stack Status
1	Push 5	[5]
2	Push 2	[5, 2]
3	× (Multiply)	[10]
4	Push 3	[10, 3]
5	Push 4	[10, 3, 4]
6	× (Multiply)	[10, 12]
7	Push 4	[10, 12, 4]
8	− (Subtract)	[10, 8] (12 – 4)
9	+ (Add)	[18] (10 + 8)
10	Push 2	[18, 2]
11	Push 4	[18, 2, 4]
12	Push 4	[18, 2, 4, 4]
13	× (Multiply)	[18, 2, 16] (4 × 4)
14	+ (Add)	[18, 18] (2 + 16)
15	+ (Add)	[36] (18 + 18)

20. Explain CPU organization in terms of Single accumulator organization, General register organization and Stack organization.

Single Accumulator Organization
In Single Accumulator Organization, there is a dedicated register called the accumulator that is used for all arithmetic and logical operations. Whenever the CPU performs an operation, one operand is typically fetched from memory, and the result is stored back into the accumulator. For example, to perform the addition of two numbers (A + B), the CPU first loads A into the accumulator, then adds B to it, and finally stores the result back into memory if needed.

This type of CPU organization is simple and requires fewer registers, making it cost-effective and easier to design. It was widely used in early microprocessors like the Intel 8085. However, because only one register is available for computation, frequent loading and storing from memory make this design slower for complex operations.

General Register Organization
General Register Organization uses multiple general-purpose registers (e.g., R0, R1, R2, etc.) instead of a single accumulator. This allows the CPU to store multiple operands, intermediate results, and output data within the processor itself. For example, to perform the addition of A + B, the CPU loads A into one register (R1), B into another register (R2), performs the addition, and stores the result in a third register (R3).
This architecture is faster because it reduces the need to access memory repeatedly, as data is stored directly in the registers. It also supports more complex operations and addressing modes. Modern CPUs like Intel x86 and ARM are based on general register organization. However, the increased number of registers requires more hardware complexity and larger instruction formats to specify which registers to use.

Stack Organization
In Stack Organization, the CPU uses a special area of memory called the stack to store operands and intermediate results. The stack operates on the Last-In, First-Out (LIFO) principle, meaning the last value pushed onto the stack is the first one retrieved. Instructions in this architecture implicitly reference the top of the stack, so operands do not need to be specified in the instruction. For example, to compute A + B, the CPU pushes both A and B onto the stack, adds them, and then pushes the result back.
Stack-based CPUs are particularly useful for handling function calls and recursion, as they can easily manage return addresses and parameters. This architecture simplifies the instruction set because there is no need to explicitly mention operand locations. However, stack operations can be slower due to the overhead of pushing and popping values, and there is a risk of stack overflow if the stack exceeds its allocated space.

21. Consider a memory of 64K * 20 and the instruction has three parts which are the operand, opcode and a mode bit to specify direct or indirect address. Use reference of this memory to find out the instruction format followed by the size of a) Address register b) Data register c) Instruction register.

64K*20 = 64*1024= 65536
each store 20 bits

instruction has operand, opcode (address bit) and a mode bit

Data register need 20 bit

Memory 65536 location so number of bits for address $\log_2(65536)=16$ bits
Instruction size=Opcode bits+Address bits (16)+Mode bit (1)
20 = Opcode bits+16+1
Opcode bits = 3
so a) Address register: 16 b) Data register:20 c) Instruction register :20.

22. Compare between Von Neumann and Harvard architecture

--refer slide COA-1

23. What do you mean by common bus system and why it is used?
--Refer Slide COA-3 slide 7

24. **Implement an instruction cycle flowchart to decode the instruction types for a memory size of 8192 *18.**
-- instruction cycle steps
Step1: Fetch
Step2: Decode
8192 = 2^{13}
instruction(18) = Opcode bits + Address bits (13)+Mode bit (1)
opcode 4 bit
0-3 opcode
4-16 address
17 mode

Step 3: Indirect/ direct address
Step 4: Execute
Step 5: any interrupt
Step 6: update program counter

25. **Discuss about direct and indirect addressing.**
--refer slide COA-5 slide 38,39

26. **Illustrate this equation X=(A+B)*(C+D) with One address instruction, Two address instruction, Three address instruction, Zero address instruction and RISC instruction.**
--refer side COA-5

27. **Explain the instruction format for one address, two address and three address instruction.**
--refer Slide COA-5

28. **Explain indexed and based register addressing mode.**
--refer Slide COA-5

29. **Explain with examples the concept of any four data transfer, data manipulation and program control instruction.**

Data Transfer: MOV, STORE, LOAD
data manipulation: ADD, MUL, SHIFT
program control: JMP, CALL

30. **Illustrate with a neat flowchart to explain how the control unit determines the instruction after decoding an instruction.**

--slide COA5 slide 36,37