UNIT 1 NOTES – Computer Fundamentals & Basics

1. Introduction to Computer Fundamentals

- **Evolution of Computers**
  - *1st Generation (1940–56):* Vacuum tubes, large size, slow, used machine language.
  - *2nd Generation (1956–63):* Transistors replaced vacuum tubes, assembly language.
  - *3rd Generation (1964–71):* Integrated Circuits (ICs), faster, more reliable.
  - *4th Generation (1971–Present):* Microprocessors, VLSI, PCs.
  - *5th Generation (Present & Beyond):* AI, Machine Learning, Quantum Computing.
- **Classification of Computers**
  - **By Size & Power**: Supercomputers, Mainframes, Minicomputers, Microcomputers.
  - **By Purpose**: General-purpose, Special-purpose.
  - **By Data Handling**: Analog, Digital, Hybrid.
- **Applications of Computers**
  - Business, Education, Research, Healthcare, Banking, Communication, Engineering.
- **Components of a Computer System**
  - **Hardware**: Physical parts (CPU, memory, I/O devices).
  - **Software**: System software (OS, utilities), Application software (MS Office, Browsers).
  - **Firmware**: Software embedded in hardware.
  - **Peopleware**: Users who interact with computers.
- **Booting**
  - *Cold Booting*: Starting a computer from OFF state.
  - *Warm Booting*: Restarting without turning OFF power.

---

2. Data Representation in Computers

- **Positional Number Systems**
  - Base-2 (Binary), Base-8 (Octal), Base-10 (Decimal), Base-16 (Hexadecimal).
  - Conversion methods: Repeated division/remainder (for integer part), multiplication method (for fractional part).
- **Signed Integer Representation**
  - **Sign-Magnitude**: MSB represents sign, rest magnitude. (e.g., +5 = 0101, –5 = 1101).
  - **1's Complement**: Invert all bits of number.
  - **2's Complement**: 1's complement + 1 (used in computers for subtraction & arithmetic).

- **Floating-Point Representation (IEEE 754 Standard)**
  - Form: ± mantissa × base^exponent
  - Single precision (32-bit): 1 sign bit + 8 exponent bits + 23 mantissa bits.
  - Double precision (64-bit): 1 sign bit + 11 exponent bits + 52 mantissa bits.
- **Character Codes**
  - **ASCII** (American Standard Code for Information Interchange) – 7/8 bits.
  - **EBCDIC** (Extended Binary Coded Decimal Interchange Code).
  - **UNICODE** – 16-bit/32-bit, supports multiple languages.

---

3. Arithmetic in Computers

- **Fixed-Point Arithmetic**
  - Used for integers.
  - Operations: Addition, Subtraction, Multiplication, Division.
  - Example: Binary addition with carry; subtraction via 2's complement.
- **Floating-Point Arithmetic**
  - Used for real numbers.
  - Addition/Subtraction: Align exponents → Add mantissas → Normalize.
  - Multiplication: Add exponents → Multiply mantissas → Normalize.
  - Division: Subtract exponents → Divide mantissas → Normalize.

---

4. Boolean Algebra and Digital Logic

- **Boolean Algebra**
  - Deals with variables having two values: 0 (False), 1 (True).
  - **Basic Operations**: AND (·), OR (+), NOT (¬).
  - **Laws/Identities**:
    - $A + 0 = A$; $A \cdot 1 = A$
    - $A + A = A$; $A \cdot A = A$
    - $A + A' = 1$; $A \cdot A' = 0$
    - $(A + B)' = A' \cdot B'$ (De Morgan's Law)
- **Boolean Expressions**
  - Written using variables and operators (e.g., $F = A \cdot B + C$).
- **Karnaugh Maps (K-Maps)**
  - Graphical method for minimizing Boolean functions.
  - Grouping 1s into 1, 2, 4, 8… (powers of 2).
  - Example: $F(A,B,C) = \Sigma(1,3,5,7)$ → simplified form.
- **Logic Gates**
  - Basic: AND, OR, NOT.
  - Universal: NAND, NOR.
  - Derived: XOR, XNOR.

- **Digital Components**
  - Encoders, Decoders, Multiplexers (MUX), Demultiplexers (DEMUX).
- **Combinational Circuits**
  - Output depends only on present input.
  - Examples: Half Adder, Full Adder, Subtractor, Encoder, Decoder, MUX, DEMUX.
- **Sequential Circuits**
  - Output depends on present input + previous states.
  - Uses memory elements (Flip-Flops).
  - Types: Synchronous (clock-driven), Asynchronous.
  - Examples: Counters, Shift Registers.

✅ Summary of Unit 1

- Computer fundamentals (evolution, classification, components, booting).
- Data representation (number systems, signed integers, floating-point, character codes).
- Arithmetic (fixed & floating-point operations).
- Boolean algebra, K-Maps, logic gates, combinational & sequential circuits.

UNIT II NOTES – Memory, Registers & Transfer

1. Registers & Register Transfer

- **Register**: A collection of flip-flops (storage elements) used to store binary data.
  - Example: 8-bit register stores 8 bits.
- **Register Transfer Language (RTL)**:
  Symbolic notation to describe operations in registers.
  - Example: R2 ← R1 → Contents of register R1 transferred to R2.
- **Types of Register Transfers**:
  - **Parallel Transfer**: All bits transferred simultaneously.
  - **Serial Transfer**: Bits transferred one at a time.

---

2. Bus and Memory Transfers

- **Bus**: A communication pathway for data transfer among components.
  - **Data Bus** → carries data.
  - **Address Bus** → carries address of memory/IO.
  - **Control Bus** → carries control signals (Read/Write).
- **Bus Architecture**:
  - **Single Bus** → All units share one bus.
  - **Multiple Bus** → Separate buses for memory, I/O, CPU (higher performance).
- **Bus Arbitration**:
  - Mechanism to resolve conflict when multiple devices want to use bus.
  - Methods:
    - Daisy chaining (priority chain).
    - Polling.
    - Independent request.

---

3. Arithmetic Logic & Shift Micro-operations

- **Micro-operations**: Simple operations on data in registers.
  - **Arithmetic micro-ops**: Add, Subtract, Increment, Decrement.
  - **Logic micro-ops**: AND, OR, NOT, XOR.
  - **Shift micro-ops**:
    - Logical Shift → shifts bits, fills 0.
    - Arithmetic Shift → preserves sign bit (MSB).
    - Circular (Rotate) Shift → bits wrap around.

- **Arithmetic Logic Shift Unit (ALSU)**:
  - A circuit that performs arithmetic, logic, and shift operations.
  - Implemented using a multiplexer + full adder + shifters.

---

4. Arithmetic Algorithms

- **Addition & Subtraction**:
  - Done using binary adder/subtractor circuits.
  - Subtraction often done using **2's complement addition**.
- **Multiplication**:
  - *Booth's Algorithm* (efficient method for signed numbers):
    - Encodes consecutive 1s in multiplier → reduces number of additions.
    - Example: 0011110 → replaced with +1 at left, -1 at right.
- **Division**:
  - Done using shift-and-subtract method.
  - Restoring and Non-Restoring Division algorithms.

---

5. Memory Organization & Hierarchy

- **Memory Hierarchy (Fast → Slow, Expensive → Cheap)**:
  - Registers → Cache → Main Memory (RAM) → Secondary Storage (Disk) → Tertiary (Tape).
  - Goal: Reduce average memory access time.
- **Main Memory**:
  - **RAM** (Random Access Memory):
    - Volatile.
    - Types: **SRAM** (faster, costly, cache), **DRAM** (slower, cheaper, main memory).
  - **ROM** (Read-Only Memory):
    - Non-volatile.
    - Types: PROM, EPROM, EEPROM, Flash.
- **Auxiliary Memory**:
  - Hard disks, CDs, DVDs, Magnetic tapes.
  - Large, cheap, non-volatile.
- **Cache Memory**:
  - Very fast memory between CPU & RAM.
  - **Levels**: L1, L2, L3.
  - **Mapping techniques**:
    - Direct Mapping.
    - Associative Mapping.
    - Set-Associative Mapping.
- **Virtual Memory**:
  - Technique of using disk storage to extend RAM.
  - Implemented using **Paging** (fixed-size blocks) or **Segmentation**.
  - Managed by **Memory Management Unit (MMU)**.

- **Memory Management Hardware**:
  - **Page Table**: Maps virtual addresses to physical addresses.
  - **TLB (Translation Lookaside Buffer)**: Cache for page table entries.

✅ Summary of Unit II

- Registers and RTL describe internal CPU data movement.
- Bus = communication backbone, arbitration resolves conflicts.
- Arithmetic Logic Shift Unit = performs micro-ops.
- Arithmetic algorithms: Add/Sub (2's complement), Booth multiplication, Division.
- Memory hierarchy: Registers → Cache → RAM → Disk.
- Cache speeds up memory access; Virtual Memory gives illusion of large memory.
- MMU + TLB handle address translation.

UNIT III NOTES – Control Design

1. Control Unit (CU)

- **Definition**: The Control Unit directs the flow of data between the CPU, memory, and I/O devices.
- It interprets instructions from programs and generates necessary control signals.
- **Two Types of Control Design**:
  - **Hardwired Control Unit**
  - **Microprogrammed Control Unit**

---

2. Fundamental Concepts of Control

- **Register Transfers**: Moving data from one register to another.
  Example: R2 ← R1
- **Performing Arithmetic/Logical Operations**:
  Example: R3 ← R1 + R2
- **Fetching a Word from Memory**:
  Example: MAR ← Address, MBR ← Memory[MAR]
- **Storing a Word in Memory**:
  Example: Memory[MAR] ← MBR

---

3. Execution of a Complete Instruction

Instruction execution involves **Instruction Cycle**:

1. **Fetch**: Instruction fetched from memory into Instruction Register (IR).
2. **Decode**: CU decodes opcode and generates control signals.
3. **Execute**: ALU performs operation / data transferred / branch handled.
4. **Store**: Result stored back to register or memory.

Cycle continues for next instruction.

---

4. Multiple-Bus Organization

- **Single Bus** → all CPU, memory, and I/O share one bus (simple, but slow).
- **Multiple Bus** → separate buses for CPU-memory and CPU-I/O (parallelism, faster).
- Example: **3-bus structure** → Data Bus, Address Bus, Control Bus.

5. Hardwired Control Unit

- **Concept**: Control signals generated by fixed hardware circuits (combinational logic).
- Uses *decoders, encoders, and logic gates*.
- **Advantages**:
  - Very fast execution.
- **Disadvantages**:
  - Difficult to modify (inflexible).
  - Complex design for large instruction sets.
- **Use case**: RISC processors (simple instructions, speed critical).

---

6. Microprogrammed Control Unit

- **Concept**: Control signals generated by executing a sequence of **microinstructions** stored in a **Control Memory (CM)**.
- Each instruction = sequence of microinstructions → forms a **microprogram**.
- **Microinstruction** = binary word specifying control signals.
- **Microprogram Sequencing**:
  - Control memory stores microinstructions.
  - Control Address Register (CAR) → points to current microinstruction.
  - Control Data Register (CDR) → holds the microinstruction.
- **Branching in Microprograms**:
  - **Wide-Branch Addressing** → Microinstruction contains full address of next instruction.
  - **Microinstruction with Next-Address Field** → Has explicit field for next microinstruction.
  - **Prefetching Microinstruction** → Next microinstruction fetched in advance (pipelining CU).
- **Advantages**:
  - Easier to modify (just change microprogram).
  - Flexible for complex instruction sets.
- **Disadvantages**:
  - Slower than hardwired.
- **Use case**: CISC processors (complex instructions, microprogramming preferred).

7. Comparison: Hardwired vs Microprogrammed CU

Feature	Hardwired CU	Microprogrammed CU
Speed	Very Fast	Slower
Flexibility	Difficult to modify	Easy to modify
Complexity	High (large ISAs)	Lower (uses microcode)
Best for	RISC CPUs	CISC CPUs

✅ Summary of Unit III

- Control Unit = brain of CPU, generates control signals.
- Fundamental operations: register transfers, arithmetic/logic, memory fetch/store.
- Instruction cycle = Fetch → Decode → Execute → Store.
- Multiple-bus organization improves speed over single-bus.
- Hardwired CU → fast but inflexible (RISC).
- Microprogrammed CU → slower but flexible (CISC).

UNIT IV NOTES – Processor & Input-Output Organization

1. Processor Design

Processor Organization

- **General Register Organization**
  - Uses a large set of registers to reduce memory references.
  - Instructions specify register addresses.
  - **Advantages**: Faster execution, flexible programming.
- **Stack Organization**
  - Operands stored in a stack (LIFO structure).
  - Implicit addressing (Top of Stack).
  - Instructions: PUSH, POP.
  - **Advantages**: Simple instruction format.
  - **Disadvantages**: Slower (stack access vs. registers).

Addressing Modes

Ways to specify operands in instructions.

1. **Immediate** → Operand is part of instruction. (Ex: `ADD R1, #5`)
2. **Direct** → Address of operand given directly. (Ex: `LOAD R1, 5000`)
3. **Indirect** → Address field points to another memory location holding operand.
4. **Register** → Operand stored in CPU register.
5. **Register Indirect** → Register contains memory address of operand.
6. **Relative** → Effective address = Program Counter + displacement.
7. **Indexed** → Effective address = Base register + index.

Instruction Format

- **Fields in Instruction**:
  - **Opcode** → Operation to perform.
  - **Address field(s)** → Location of operands.
  - **Mode field** → Addressing mode.

Types:

- **0-address** → Stack machines (e.g., `ADD` operates on TOS).
- **1-address** → Accumulator machines (`ADD X`).
- **2-address** → Register-memory machines (`MOV A, B`).
- **3-address** → General register machines (`ADD R1, R2, R3`).

Data Transfer & Manipulations

- **Data Transfer**: Move data between CPU, memory, I/O. (LOAD, STORE).
- **Data Manipulation**: Arithmetic (ADD, SUB), Logical (AND, OR), Shift (SHL, SHR).

Program Control

- Instructions that alter flow of execution.
    - **Conditional Branch** → Jump if condition true.
    - **Unconditional Branch** → Always jump.
    - **Subroutine Call & Return** → Transfer control to subroutine, then return.
    - **Program Interrupt** → Control transferred to interrupt service routine (ISR).

Reduced Instruction Set Computer (RISC)

- **Characteristics**:
    - Few, simple instructions.
    - Fixed-length instruction format.
    - Load/Store architecture (memory access only via LOAD/STORE).
    - Large number of registers.
- **Advantages**: Faster execution, easier pipelining.
- **Examples**: MIPS, ARM.

---

2. Input-Output Organization

I/O Interface

- Connects I/O devices with CPU & memory.
- Provides buffering, error detection, device addressing.

Modes of Transfer

1. **Programmed I/O**:
    - CPU controls all transfers by executing I/O instructions.
    - Simple but CPU is busy-waiting.
2. **Interrupt-Driven I/O**:
    - Device sends an interrupt when ready.
    - CPU executes ISR, efficient use of CPU.

3. **Direct Memory Access (DMA)**:
    - Device transfers data directly to memory without CPU intervention.
    - DMA controller manages transfer.
    - CPU only initializes and gets interrupt at completion.
    - Used for high-speed devices (disk, network).

Interrupts & Handling

- **Interrupt**: Request to CPU to stop current execution & handle urgent task.
- **Types**:
    - Hardware (I/O devices).
    - Software (system calls).
- **Interrupt Handling Process**:
    - CPU saves current state.
    - Jumps to ISR (Interrupt Service Routine).
    - After completion, restores state and resumes.

Input-Output Processor (IOP)

- A dedicated processor for I/O operations.
- Offloads CPU from handling I/O tasks.
- Communicates with CPU via memory and interrupts.

Serial Communication

- **Parallel Communication** → Multiple bits transmitted simultaneously (fast, costly).
- **Serial Communication** → One bit at a time (cheap, long-distance).
    - **Synchronous**: Data sent with a clock signal.
    - **Asynchronous**: Uses start/stop bits, no shared clock.

✅ Summary of Unit IV

- Processor organization → Register vs. Stack.
- Addressing modes → Immediate, Direct, Indirect, Register, Relative, Indexed.
- Instruction formats → 0-, 1-, 2-, 3-address.
- Program control → Branch, subroutine, interrupts.
- RISC → Simpler, faster, load/store, easy pipelining.
- I/O organization → Programmed, Interrupt-driven, DMA.
- Interrupt handling → Save state → ISR → Restore state.
- IOP → dedicated for I/O, reduces CPU load.
- Serial communication → cheaper but slower, supports async/sync modes.

UNIT V – DEVICE SUBSYSTEMS & ADVANCED ARCHITECTURES

Device Subsystems

- **External Storage Systems**
  - **Magnetic Disk Drives**
    - Platter-based, rotating disks with read/write heads.
    - Organization: Track → Sector → Block.
    - Disk access time = **Seek time + Latency + Transfer time**.
  - **Optical Memory** (CD, DVD, Blu-ray)
    - Uses **laser beams** for reading/writing.
    - Data stored as pits (0) and lands (1).
    - Advantage: High durability & portability.
- **Basic I/O Controllers**
  - **Keyboard Controller**: Converts keystrokes → scan codes → CPU.
  - **Mouse Controller**: Captures movement/click → signals CPU.
  - **Video Controller (GPU)**: Generates video signals for display; manages frame buffer.
- **RAID (Redundant Array of Independent Disks)**
  - Technique to improve **performance + reliability** by combining disks.
  - **RAID Levels:**
    - RAID 0 → Striping (performance, no redundancy).
    - RAID 1 → Mirroring (reliability).
    - RAID 5 → Striping + Parity (balanced).
- **I/O Performance Factors**
  - Bandwidth (rate of data transfer).
  - Latency (delay in response).
  - Throughput (data processed per time).
- **SMART Technology (Self-Monitoring, Analysis, and Reporting Technology)**
  - Detects **disk health issues** (bad sectors, high temperature, errors).
  - Helps in **fault detection + preventive maintenance**.
- **Processor to Network Interfaces**
  - Network Interface Card (NIC) connects CPU ↔ network.
  - Supports Ethernet, Wi-Fi, etc.
  - Handles error detection, buffering, packet framing.

Advanced Processor Concepts

RISC vs CISC Architecture

- **RISC (Reduced Instruction Set Computer):**
  - Small, simple instructions.
  - One instruction per cycle (load/store architecture).
  - Examples: MIPS, ARM, SPARC.
- **CISC (Complex Instruction Set Computer):**
  - Large set of complex instructions.
  - Fewer lines of code, but execution is slower.
  - Example: Intel x86.

Basic MIPS Implementation

- MIPS (Microprocessor without Interlocked Pipeline Stages) is a **RISC-based** architecture.
- Uses **32-bit fixed-length instructions**.
- Stages: Instruction Fetch → Decode → Execute → Memory → Write-back.

Pipelining

- Technique to **overlap execution of instructions**.
- Improves throughput.
- Hazards:
  - Structural (hardware conflict).
  - Data (dependency on result).
  - Control (branching issues).

Instruction-Level Parallelism (ILP)

- Executing **multiple instructions simultaneously**.
- Achieved by:
  - Pipelining
  - Superscalar execution (multiple pipelines)
  - Out-of-order execution.

Parallel Processing Challenges

- Synchronization overhead.
- Memory consistency problems.
- Load balancing across processors.

Flynn's Classification

- Based on **Instruction stream (I) & Data stream (D):**
  - SISD → Single Instruction, Single Data (old sequential).
  - SIMD → Single Instruction, Multiple Data (vector processors).
  - MISD → Multiple Instruction, Single Data (rare).
  - MIMD → Multiple Instruction, Multiple Data (multicore).

Hardware Multithreading

- **Multiple threads** share CPU resources.
- Types:
  - Coarse-grained (switch on long stalls).
  - Fine-grained (switch every cycle).
  - Simultaneous Multithreading (SMT, e.g., Intel Hyper-Threading).

Multicore Processing

- Multiple processing cores on a single chip.
- Provides **true parallel execution**.
- Challenges:
  - Efficient **cache coherence**.
  - Inter-core communication.
  - Software optimization for parallelism.

✅ Quick Summary for Exam

- Device subsystems → Disk, Optical, Controllers, RAID, SMART, Network.
- RISC vs CISC → Simplicity vs Complexity.
- MIPS → RISC example with pipelined design.
- Pipelining → Increases speed but has hazards.
- Flynn's classification → SISD, SIMD, MISD, MIMD.
- Advanced processing → ILP, Multithreading, Multicore.