Bits / Bytes
- $1\ byte = 8\ bits$
- $1\ KB \approx 2^{10}\ bytes = 1024\ B$
- $1\ MB \approx 2^{20}\ bytes$
- $1\ GB \approx 2^{30}\ bytes$

Base + Limit (simple virtual memory)
- **Valid address condition:** $0 \leq VA < LIMIT$
- **Physical address:** If valid, $PA = BASE + VA$

Audio storage
- **Number of samples:** $N_{samples} = f_s \times T$ where f_s =sample rate (Hz), T= time (s)
- **Total bits:** $\text{bits} = N_{samples} \times bits\ per\ sample$
- **Total bytes:** $\text{bytes} = \frac{bits}{8}$

Number of values / addresses
- **Values from n bits:** $\#values = 2^n$
- **n-bit address space:** $\#addresses = 2^n$

Paging formulas
Let page size P(in bytes).
- **Split virtual address:** $\text{page\_number} = \left\lfloor \frac{VA}{P} \right\rfloor$ $\text{offset} = VA \bmod P$
- **Physical address (given page frame number PFN):** $PA = PFN \times P + offset$
- **Number of virtual pages:** $\#pages = \frac{VM\ size}{P}$
- **Number of page frames in RAM:** $\#frames = \frac{RAM\ size}{P}$

Calculation

Cache performance
- **Hit ratio:** $H = \frac{hits}{hits + misses}$
- **Effective access time (EAT):** $\text{EAT} = H \cdot t_{cache} + (1 - H) \cdot t_{mem}$

IP address counts
- IPv4: $\#addresses = 2^{32}$
- IPv6: $\#addresses = 2^{128}$

Integer ranges
- **Unsigned n-bit:** $0 \leq x \leq 2^n - 1$
- **Signed 2's complement (n bits):** $-2^{n-1} \leq x \leq 2^{n-1} - 1$

Bit shifts
Let $x$be an integer.
- **Left shift by a bits:** $x \ll a = x \times 2^a$
- **Right shift by a bits (integer):** $x \gg a \approx \left\lfloor \frac{x}{2^a} \right\rfloor$

Pipelining
Let stage times be t_A, t_B, t_C, t_D.
- **Non-pipelined instruction time:** $t_{inst} = t_A + t_B + t_C + t_D$
- **Non-pipelined throughput (instructions per second):** $\text{IPS}_{nonpipe} = \frac{1}{t_{inst}}$
- **Pipeline stage time:**
- **Pipelined throughput (ideal):** $\text{IPS}_{pipe} \approx \frac{1}{t_{stage}}$
- **Pipeline speed-up (ideal):** $\text{Speed-up} \approx \frac{t_{inst}}{t_{stage}}$

1. OPERATING SYSTEMS

→ Basic picture
User ↔ Application ↔ **Operating System (OS)** ↔ Hardware.
OS = special software that is loaded by boot code when the computer powers on.

→ What is an OS
Sequence of instructions that **controls the machine hardware** and provides an environment for applications.
Most users never see it directly – but *you* need to.

→ OS taxonomy / types
Single-user (e.g. typical desktop). | **Multi-user** (shared servers, mainframes). | **Embedded** (routers, phones, IoT devices). | **Cluster / distributed systems** (many machines cooperating). | **Real-time OS** (timing guarantees; mentioned in taxonomy).

→ What the OS does (high-level)
Resource manager – CPU, memory, devices, files. | **Virtual machine concept** – each program thinks it owns the machine. | **Implements security & protection policies** – who can access what.

→ OS structure models
Monolithic – big blob of kernel code that does (almost) everything. | **Layered** – OS split into layers that build on one another. | **Microkernel** – very small kernel (scheduling, IPC, basic memory); other services in user-space.

→ 9 Classic OS features
Support for concurrency Many tasks "at once" using **context switching**, **scheduling**, protection, synchronisation.
Sharing mechanisms Sharing code/resources like drivers, window system; one copy used by many processes.
Persistent storage Long-term data on discs/SSD/HDD; convenient, reliable access; backups.
Nondeterminacy Events arrive in unpredictable order/timing; OS must still behave correctly.
Efficiency High utilisation, low overhead, fast response.
Reliability Minimise bugs' impact; maximise Mean Time Between Failures (MTBF).
Resilience Tolerate hardware faults where possible.
Maintainability Code easy to fix/extend.
Small size Less memory, less chance of bugs, easier updates.

→ Processes & concurrency
OS = many **processes**: some for system, some for users. | **Process** = running program (code + data + state). | Multiple **processors** (CPU cores, GPUs, FPUs) can run processes in parallel. | To support concurrency, processes must: **Communicate** – share data. **Synchronise** – coordinate ordering.

→ Two core concurrency concepts
Mutual exclusion (mutex) More than one process wants a shared but unshareable resource (printer, table row, mailbox). Only one can use it at a time.
Synchronisation One process must wait for another to reach a certain point before continuing.

→ Threads
Lightweight "mini-processes" inside a process, sharing code & data but having their own stack & registers. | Used heavily in GUI/window systems and servers.

→ OS Kernel
Kernel = most critical part of OS. **|** Is either: Core layer of layered system; or Main body of a monolithic OS. | Often split into **upper** and **lower** halves; lower is the Hardware Abstraction Layer (HAL) that hides hardware details.

→ Kernel responsibilities
Process switching & scheduling. | **Inter-process communication (IPC) & synchronisation**. | **Hardware interfacing** (drivers).

→ Key kernel components
First-level interrupt handler – reacts instantly to interrupts | **Communication primitives** – pipes, message queues, semaphores, etc | **Low-level scheduler** – chooses which process/thread runs next.

→ Process Control Block (PCB)
1. Data structure used by OS to describe a process.
2. Typically holds: process ID | state (running/ready/blocked) | registers | program counter | stack pointer | memory info (base/limit or page table), scheduling info | open files, etc.

2. MEMORY MANAGEMENT, VIRTUAL MEMORY AND PAGES

→ Basic idea of memory
1. Stores **binary data**: numbers, text, images, programs. | Programmers think in **variables & types**; hardware sees only addresses & bits. | Same data comes back out if you read from the same address.

→ Real vs virtual addresses
Real (physical) address – location in actual RAM. Unique. | **Virtual address** – what your program sees. | OS + hardware (MMU) translate virtual → physical. | Benefits: isolation, protection, easier programming.

→ What OS does for addresses
Address translation – via MMU & page tables / base+limit. | **Address checking** – ensure within legal range. | **Memory protection** – read/write/execute permissions.

2.1 Base & Limit (simple virtual memory)
→ Idea
Early CPUs had simple hardware: each process gets a contiguous region of physical memory.
Two registers per process: **BASE**: starting physical address of process. **LIMIT**: size of its address space.

→ Address translation
Program uses **VirtualAddress** (0..LIMIT−1). | Hardware checks VirtualAddress < LIMIT. | PhysicalAddress = **BASE + VirtualAddress** if valid, else memory fault.

→ Pros / cons
- Simple, fast hardware.
- Leads to fragmentation, poor utilisation, hard to grow address space.

2.2 Paging systems (modern virtual memory)
→ Concepts
Virtual memory and **physical memory** are both split into fixed-size blocks: **Pages** (virtual). **Page frames** (physical). MMU translates (page_number, offset) → (frame_number, offset).

→ Address breakdown
Given page size **P** bytes: | page_number = floor(VA / P) | offset = VA mod P | RealAddress = frame_number × P + offset

→ Page tables
OS stores, per process, a **page table** with one entry per virtual page. Each Page Table Entry (PTE) typically has: | **V (valid)** bit – is this page currently in a frame? | **PROT / protection** – read/write/exec permissions/users. | **M (modified/dirty)** bit – page changed since loaded. | **OWN** / software bits – used by OS. | **PFN** – page frame number.

→ Page faults – when they occur
Page number invalid / out of range. | Address beyond allocated virtual memory. | Legal but never referenced before (needs to be created). | Page is currently swapped out to disk (page file). | Access violates permissions.

→ Page-in operation (handling a "not in RAM yet" fault)
Find a free page frame (or pick a victim to evict). | Read required page from disk into that frame. | Update PTE (set V bit, PFN, clear M, etc.). | Restart / resume the faulting instruction.

→ Fetch strategies
Demand fetch – only load page when first referenced. | **Anticipatory / prefetch** – guess which pages will be needed soon and load early.

→ Retirement / replacement strategies
Random – pick any page frame. Simple but poor. | **Least recently used (LRU)** – evict page that hasn't been used for the longest time. Better locality but more overhead.

→ Overheads
Address translation requires at least one extra memory access (to read PTE) → effectively "double read" unless you use a cache like TLB.

3. FILING SYSTEMS

→ Purpose of file systems
Long-term information storage – reliable, efficient, long-lasting. | Store data on media like HDD/SSD, optical discs, flash drives, etc. | Provide **security and access control**. | Survive reboots and crashes (ideally).

→ Common file systems
FAT, NTFS (Windows), HFS/APFS (Apple), ext2/3/4, btrfs, xfs, zfs, ufs, vxfs, ISO9660 (CD-ROM), NFS, ReiserFS, etc.

→ What are files?
Containers for: data | program source | binaries | configuration, etc.
File system must support: 1. Creation, deletion, modification. | Read, write, append. | Naming and directories. | Protection, privacy, security. | Efficient access.

→ Core file operations
Open – get a handle / descriptor | **Read** – from current offset | **Write** – to current offset | **Seek/rewind** – move read/write pointer | **Close** – release descriptor.

3.1 File organisation on disk
Contiguous organisation

→ Representation
File stored in a sequence of consecutive blocks on disk. | Directory stores start block and length.

→ Advantages
Very simple implementation. | Excellent sequential access performance. | Good for static (fixed-size) executables.

→ Disadvantages
External fragmentation – free space split into many small gaps. | File growth is hard – might have to move entire file. | May require periodic **compaction** or **defragmentation**. | File Map organisation (FAT / block map)

→ Idea
Separate structure (FAT / block group) maps each block to "next block". | File is a linked list of blocks; directory just stores start block.

→ Advantages
Avoids external fragmentation (any free block can be used). | Easy to grow/shrink files.

→ Disadvantages
Map itself can be large and must be read → overhead. | Random access is slow (must follow pointers). | Extra disk accesses to traverse the map.

3.2 Directories & metadata
→ Directory entry typically contains
File name and type. | Permissions and access control bits. | Owner and group. | Timestamps – creation, last access, last modification, possibly last backup. | File size and/or block count. | Pointer(s) to data blocks / inodes.

3.3 Maintenance & backup
→ File system maintenance
Compaction – rearrange files to reduce fragmentation. | **Defragmentation** – move blocks so each file is more contiguous. | **Consistency checks** (fsck, chkdsk) – detect/correct metadata errors. | **Formatting** – create new, empty file system structures. | **Detect bad blocks** – mark them unusable and map around them.

→ Backup strategies
Goal: be able to **restore part or all** of the file system after failure.
Types: **Full backup** – everything. | **Differential** – changes since last full backup. | **Incremental** – changes since last *any* backup (full or incremental).
Requirements: reliable | complete | correct | minimal redundancy | physically secure.
Media: cloud | second internal disk | external HDD/SSD | tape | USB | optical discs.

4. NETWORKING PART 1 – Connectivity, IP Model & Ethernet

→ What is a computer network?
System that allows **data interchange between computers**. | A simple peripheral (e.g. printer directly plugged into a PC) is not usually called a "network".

→ Physical connectivity
Links can be **wired** (Ethernet, fibre, USB, CAN, etc.) or **wireless** (Wi-Fi, 5G, Bluetooth, etc.). | Network can connect two machines (point-to-point) or many machines (LAN, WAN, Internet).

→ Internet
A **connection protocol** between computers; can be one hop or many hops over heterogeneous physical links.

4.1 OSI model (reference)
→ 7-layer OSI model (classic)
Physical | Data link | Network | Transport | Session | Presentation | Application
(Used mainly as a conceptual model; real Internet stack is simpler.)

4.2 Internet / TCP-IP 5-layer model
→ Layers (bottom → top)
Physical – sends bits across a medium. | **Network Interface** – groups bits into frames/packets for specific media; hides differences between local networks. | **Internet layer – IP** (Internet Protocol) provides best-effort, end-to-end delivery of datagrams (no guarantee of arrival). | **Transport layer** – splits / reassembles data into segments/datagrams; uses protocols such as TCP and UDP. | **Application layer** – protocols such as HTTP, SMTP, POP3, FTP, NFS, SMB/NETBIOS, etc.

→ Encapsulation idea
Each layer wraps the data from above with its own **header** (and sometimes footer). | On receive, headers are stripped layer by layer.

4.3 Transport layer basics (TCP vs UDP)
→ **UDP (User Datagram Protocol)**
Best effort, connectionless, no guarantee of delivery/order. | Very low overhead, good for time-critical traffic (DNS queries, streaming, VoIP). | Endpoint identified by (IP address, port).
→ **TCP (Transmission Control Protocol)**
Connection-oriented: uses handshake to establish a reliable session. | Guarantees in-order, error-checked delivery of a byte stream. | Bi-directional connection; resends lost segments. | Endpoint also identified by (IP address, port).

4.4 Protocol originators & standards
Internet protocols (TCP/IP, etc.) originated with **DARPA**.
Maintained by **IETF** via public **RFCs (Requests For Comments)**.
Standards can be: **De jure** – by law / official standards bodies. | **De facto** – widely used in practice, become standards by usage.

4.5 Hardware / Ethernet & MAC addresses
→ Hardware (link) layer
Implemented by **Network Interface Cards (NICs)** and drivers. | Uses **MAC (Media Access Control) addresses**, unique per interface (48-bit).

→ Ethernet frame format (simplified)
Preamble (8 bytes) – sync. | **Destination MAC (6)** | **Source MAC (6)** | **Type (2)** – indicates payload protocol (e.g. IPv4, ARP). | **Payload** – data (e.g. IP datagram, ARP message). | **CRC (4)** – error-detecting code.

→ Ethernet behaviour
Originally used **CSMA/CD** on shared media; nowadays mostly switched (collision-free). | Frame is best effort; errors may cause drops, not retries (reliability from upper layers).

5. NETWORKING PART 2 – TCP/IP, Routing & Naming

→ Encapsulation recap
Application data → wrapped in **TCP/UDP segment** → wrapped in **IP datagram** → wrapped in **Ethernet frame** for transmission.

5.1 IP datagram basics
Contains an **IP header + payload** (usually TCP or UDP). | Header fields include: Source & destination **IP addresses**. | Protocol | (TCP/UDP/ICMP). | Total length. | Header checksum. | TTL (Time To Live). | Flags & fragmentation info.

5.2 Datagrams & fragmentation
Each layer has a maximum payload size (MTU). | If incoming block too big, IP splits into multiple **fragments** (each its own datagram). | Receiver reassembles fragments before passing up.

5.3 Transport layer & ports
→ Ports
16-bit number used to identify **application** on host. | Many applications can use the network simultaneously; ports demultiplex traffic.

→ Using ports
Application "listens" on a port.
For TCP: server listens; client connects to (server_IP, server_port). Server may move the connection to a new ephemeral port.
For UDP: app receives any datagram sent to its listening port.

5.4 Hardware address resolution – ARP
Software uses **IP addresses**, but Ethernet sends to **MAC addresses**. | **ARP (Address Resolution Protocol)** maps IP → MAC on local network. | Host A sends ARP request (broadcast at Ethernet level) asking "who has IP X?". | Host B with IP X replies with its MAC. | Mapping cached for a while, then forgotten.

5.5 IP addressing and IPv6
IPv4: 32-bit addresses, dotted decimal (e.g. 192.168.1.10). Limited space (~4.3B addresses).
IPv6: 128-bit addresses, hex groups (e.g. 210F:6154:D017:9FBB::). Huge space.
In address like 192.168.1.234, part identifies **network** (e.g. 192.168.1), rest is **host interface** (e.g. 234).
One computer with multiple network interfaces can have **multiple IPs**; on same network, those IPs must be different.

5.6 Routing
Routing ensures IP datagrams reach the correct network. | If destination network == current network → deliver directly to host. | Else → consult **routing table**, send to gateway router / ISP. | **Router** has interfaces on two or more networks and forwards datagrams between them.

5.7 Naming & DNS
→ Why names?
IP addresses are hard to remember; we use hostnames like www.singaporetech.edu.sg.

→ Domain structure
Names are hierarchical: e.g. library.singaporetech.edu.sg | sg – country (delegated to local authority). | edu / ac – sector (education). | singaporetech – organisation. | library – specific host.

→ DNS (Domain Name System)
Distributed database mapping **names ↔ IP addresses**.
Implemented by hierarchy of **DNS servers** (often duplicated for reliability).
Common record types: **A** – name → IPv4 address. | **PTR** – address → name (reverse lookup). | **CNAME** – alias/another name. | **TXT** – arbitrary text (e.g. SPF). | **SOA** – start of authority. | **MX** – mail exchanger for domain.

→ DNS queries
Client uses a local **resolver**; it can check: Local host file. | Cached entries (with TTL). | Configured DNS servers (starting at root).
Reverse lookup: IP A.B.C.D encoded as D.C.B.A.in-addr.arpa.

→ Root DNS servers
Special servers serving top-level domains (.com, .org, .sg, .eu, etc.).
Anycast: each root name (e.g. A.root-servers.net) maps to multiple actual machines around the world.

6. NETWORKING PART 3 – Application Layer & Email

6.1 Application layer basics
→ Role
Application layer "talks" to **transport layer (TCP/UDP)**. | Programmer uses **sockets** (same idea in C, Java, Python, etc.).

→ Client–server model
Server application listens on known port. | Client applications connect to server (same or different machine). | Client ↔ server communication done over TCP or UDP.

→ Data units
UDP = messages / datagrams (collections of bytes).
TCP = byte stream (continuous sequence of bytes).

6.2 Common application protocols & ports
DNS – Domain Name Service (UDP or TCP port 53). | **SMTP** – Simple Mail Transfer Protocol (TCP port 25). | **POP3** – Post Office Protocol v3 (TCP port 110). | **HTTP** – Hypertext Transfer Protocol (TCP port 80). | **FTP** – File Transfer Protocol (TCP ports 20 & 21). | **IRC** – Internet Relay Chat (port 194). | **NNTP** – Network News Transfer Protocol (port 119). | **NFS** – Network File System (ports 111 & 2049). | **SMB/CIFS / NETBIOS** – Windows file sharing (ports 137–139, 445).

6.3 Review TCP vs UDP for applications
UDP for **simple, fast, one-off** exchanges (e.g. DNS queries) or **real-time streaming** (VoIP, Skype). | TCP for **reliable transfers** (emails, web pages, large file downloads). | Programmer chooses based on trade-offs: reliability vs latency/overhead.

6.4 SMTP & POP3 (email transport)
→ SMTP basics
Defined in RFC 5321. | SMTP server listens on TCP port 25. | Used to **send** and sometimes receive mail.

→ Models
Smarthost – client sends all outgoing mail to ISP's SMTP server; ISP then forwards onward. | **Direct** – client sends directly to destination's SMTP server (if any).

→ POP3 / IMAP
Used by client to **retrieve** mail from a server (maildrop).| POP3: simple "download and delete". | IMAP: server-side folders, more advanced.

→ SMTP conversation example
Handshake, MAIL FROM, RCPT TO, DATA, message contents, final "." line, server replies 250 OK, QUIT. | Receiver addresses in SMTP envelope don't have to match To:/From: in header → spoofing possible.

6.5 Intermediate SMTP hosts & webmail
Mail often passes through several **intermediate SMTP servers**; each can delay, lose, or modify mail. | Each hop usually logs message ID, sender/recipient, etc. | Webmail (Gmail, Outlook.com): you use **HTTP** in browser; webmail provider runs SMTP/POP/IMAP internally for you.

6.6 Email format & security issues
→ Format
Defined by RFC 5322.
SMTP DATA command carries message as plain **ASCII text**: **Header** lines. | Blank line. | Optional **body**.
Lines separated by CRLF.

→ Headers
FieldName: field body (e.g. Subject: Hello). | Standard fields: To:, From:, Date:, Subject:, multiple Received: lines. | Custom fields usually start with X- (e.g. X-Bogosity).

→ Received headers
Each SMTP server inserts a **Received:** header at the top. | Combined list can be used to trace message path (if servers are honest).

→ Security
To:, From:, Date:, Subject: can be **forged** easily. | Email is typically unencrypted; can be read/tampered with in transit. | Take-home: **do not trust plain email for sensitive info**; use end-to-end encryption (PGP, etc.) where needed.

8. Computer Architecture: History, Structure & Organisation

→ What is inside a computer

Blocks of electronic circuits: **CPU** (Central Processing Unit). | **Memory** (RAM, ROM, cache). | I/O controllers, buses, etc.
Connected via **buses** (address, data, control).
These move and manipulate **binary data** according to a **program**.

→ Computer Architecture vs Organisation

Architecture – what the computer *does*; functional behaviour and visible features. | **Organisation** – how those functions are implemented; structural relationships and hardware details. | Same architecture (e.g. ARM) can be implemented with different organisations (e.g. different pipelines, caches).

→ Layered view of computer system

From bottom up: Electronic gates → logic circuits. | Micro-architecture & control unit. | Instruction set architecture (machine code). | Assembly language. | High-level languages (C, Java, Python). | Applications.
BIOS/firmware runs at boot to initialise hardware and load OS.

→ Von Neumann vs Harvard architectures

Von Neumann: | Single memory holds both **program instructions** and **data**. | CPU uses same bus to fetch instructions and data. | Simpler, but instruction and data fetch compete for bandwidth (bottleneck).
Harvard: Separate memories (and often buses) for **program** and **data**. | CPU can fetch instruction and data in parallel → potential speedup. | Used in many microcontrollers and DSPs.

→ CPU interaction with memory

To execute instruction: Fetch instruction from memory (often via instruction cache). | Decode instruction. | Fetch operands (from registers or memory). | Execute in ALU / other functional units. | Write results back to registers or memory.
Program Counter (PC) holds address of next instruction.
Registers store temporary data and addresses.
Instruction cache (i-cache) stores recently used instructions near the CPU.

9. Information Representation & Data Formats

9.1 Number systems

→ Decimal (base 10) – digits 0–9.

→ Binary (base 2) – digits 0,1.

→ Hexadecimal (base 16) – digits 0–9, A–F.

→ Conversion notes

Binary ↔ decimal – positional weights 2^0, 2^1, 2^2, ...
Binary ↔ hex – group bits in 4s.

9.2 Fixed-point integer representation

→ Unsigned n-bit integer

Range: 0 ... $2^n - 1$

→ Signed (two's complement)

Range: -2^{n-1} ... $2^{n-1} - 1$
To get negative: invert bits + 1.

9.3 Fixed-point fraction

Some bits represent integer, some represent fractional part.
Fraction bits correspond to 1/2, 1/4, 1/8, ...

9.4 Floating point (concept)

Number ≈ (−1)^sign × 1.mantissa × 2^(exponent−bias) | Allows representing a very wide range of values (very small and very large).

9.5 Binary arithmetic (ALU)

ALU performs add, subtract, multiply, divide, logic ops (AND, OR, XOR, NOT, etc.).
Addition – same rules as decimal but in base 2, with carries. | **Subtraction** – using borrow or by adding two's complement. | **Multiplication** – repeated shift and add. | **Division** – repeated subtract or shift and subtract. | Shifting left / right by a bits ≈ ×2^a / ÷2^a.

9.6 Memory addressing, endianness & pointers

→ Memory

Array of bytes, each at an address. | **Pointer** = variable that holds an address.

→ Endianness

Big-endian – most significant byte at lowest address. | **Little-endian** – least significant byte at lowest address (e.g. x86). | **Bi-endian** – can be configured either way (e.g. ARM). | Programmer usually doesn't need to worry; compiler/OS handle it.

9.7 Bits, bytes, words, wordlength & effects

Bit – 0 or 1. | **Nibble** – 4 bits. | **Byte** – 8 bits. | **Word** – group of bits stored/addressed together; size depends on architecture (8, 16, 32, 64 bits). | **Wordlength** – number of bits per word. | Affects range & precision for numbers. | Larger wordlength → higher quality (e.g. audio), but more memory & bandwidth. | Trade-off: **Quality vs memory size / performance**.

→ Storage calculation example (audio CD)

44.1 kHz × 16 bits × 3600 s × 2 channels ≈ 2.54 × 10^9 bits ≈ 318 MB per channel → ~636 MB for stereo.

10. CPU Basics & Instruction Set Architecture (ISA)

10.1 RISC vs CISC

→ RISC (Reduced Instruction Set Computers)

Simple, fixed-format instructions (load, operate, store). | Each instruction does a small job that can usually complete in 1 clock cycle. | Pros: simple hardware, easy pipelining, fewer cycles per instruction. | Cons: more instructions per program.

→ CISC (Complex Instruction Set Computers)

Rich instructions that may perform multiple steps (load+operate+store). | Pros: fewer instructions per program; compact code. | Cons: variable length, more cycles per instruction, more complex hardware.

10.2 CPU basics

→ CPU main components

Registers – small, ultra-fast storage for operands, addresses, flags. | **ALU** – performs arithmetic/logic. | **Control unit** – orchestrates fetch–decode–execute. | **Internal buses** – move data between units (operand buses, result bus). | **Program Counter (PC)** – address of next instruction. | **Instruction cache / pipeline** – store/fetch upcoming instructions.

→ Internal CPU bus & MAC

CPU often has multiple execution units; data moved via internal buses. | **MAC (Multiply–Accumulate)** unit performs multiply + add in a single step (useful in DSP).

10.3 Instruction cycle

Stages (conceptually): **Fetch** – get instruction from memory/i-cache using PC. | **Decode** – figure out what to do and which operands. | **Load operands** – from registers or memory. | **Execute** – ALU / MAC / other units process. | **Write back** – store result, update flags, PC.
Instruction time = time to complete all stages (may be multiple clock cycles).

10.4 Instruction Set Architecture (ISA)

ISA = set of machine-level instructions & how they behave. | Includes opcodes, formats, data types, addressing modes, registers, etc. | High-level language programs are compiled to ISA instructions.

11. Memory Types, Operations & Management. Peripherals & I/O

11.1 Types of memory

→ Volatile vs non-volatile

Volatile – loses content when power is off (RAM, cache). | **Non-volatile** – keeps content (ROM, Flash, HDD, SSD).

→ RAM vs ROM

RAM – fast, read/write, smaller capacity, directly accessed by CPU. | **ROM** – slower, read-mostly, larger capacity, stores firmware/boot code.

→ DRAM vs SRAM

DRAM – stored with capacitors; needs refresh; slower but denser → main memory. | **SRAM** – stored with flip-flops; very fast but expensive → caches.

→ Memory hierarchy

Registers → cache → main memory → secondary storage.
As you go down: capacity ↑, cost/bit ↓, speed ↓.

→ Cache & hit ratio

Cache stores copies of frequently used memory locations. | **Cache hit** – data found in cache → fast. | **Cache miss** – data not there → fetched from main memory, cache updated. | Performance measured by **hit ratio = hits / total accesses**.

11.2 Memory errors & parity

Errors can be **hard** (permanent) or **soft** (transient, e.g. noise, cosmic rays). | **Parity bit**: add 1 extra bit so that number of 1s is even (even parity) or odd (odd parity). | Can detect single-bit errors, but not all multi-bit errors.

11.3 Memory management unit (MMU) & virtual memory (recap)

MMU handles translation from virtual to physical addresses and implements virtual memory using paging and possibly segmentation. | Virtual memory allows program size > physical RAM by swapping pages to disk.

11.4 Peripherals & I/O

→ Peripherals

Devices that let computer interact with outside world: keyboard, mouse, monitor, printer, network, sensors/actuators, etc. | Connected via external serial/parallel ports (USB, HDMI, etc.) to **I/O interfaces**.

→ I/O bus and system bus

I/O bus (address, data, control) shared by multiple peripherals. | System bus = address + data + control buses connecting CPU, memory, I/O.

→ Addressing peripherals

Each device has an address (or range).
CPU sends: Address → which device. | Control signals (read/write, etc.). | Data to/from device.

11.5 System bus allocation schemes (I/O addressing)

Separate I/O: Separate address, data, control buses for memory vs I/O. | Different instruction set for memory vs I/O. | Pros: efficient, no contention between memory and I/O. | Cons: more hardware, more complex.
Isolated I/O: Shared **address & data** buses; separate **control lines** for memory vs I/O. | Memory and I/O have different address spaces. | I/O addresses often called **ports**.
Memory-mapped I/O: Same address, data, control buses used for both memory and I/O. | I/O devices appear as special memory addresses. | Pros: same instructions for memory and I/O; simpler programming, simpler logic. | Cons: reduces address space available for normal memory.

12. Pipelining & Polling

12.1 Pipelining

→ Analogy (laundry)

Tasks: wash, dry, fold, put away. | Non-pipelined: finish all stages for load 1 before starting load 2. Machines idle a lot. | Pipelined: as soon as washer is free, start next load while dryer works on previous load.

→ Definition

Pipelining = overlapping execution of multiple instructions by splitting instruction execution into stages handled by different hardware units.

→ Typical CPU stages (example)

FETCH instruction. | DECODE instruction. | LOAD operands. | EXECUTE. | RESULT & SAVE.

→ Performance effect

Latency – time to complete *one* instruction (often slightly higher with pipeline). | **Throughput** – instructions completed per second (much higher once pipeline is full). | Ideal: throughput improves roughly by number of pipeline stages (if all stages same time and no hazards).

→ Hazards (only conceptually)

Structural – hardware resource conflicts. | Data – instruction depends on result of previous one. | Control – branches change instruction flow.
(You mainly need to know they *reduce* ideal pipeline performance.)

12.2 Polling

→ What is polling?

CPU repeatedly **checks device status** in a loop to see if device needs attention.

→ Pros

Very simple to program and understand. | CPU has precise control over check order.

→ Cons / trade-off

Wastes CPU time checking when nothing has changed. | Latency between checks can be large (especially with multiple devices).

→ Multiple-device polling

Main loop checks each device in turn (GPIOs, timers, etc.). | Larger number of devices → bigger latency before some device is serviced.

13. Interrupts & Direct Memory Access (DMA)

13.1 Interrupts

→ Idea

Interrupt = hardware or software signal that causes CPU to temporarily stop current code and run an **Interrupt Service Routine (ISR)**.

→ Why use interrupts?

Device can notify CPU only when it needs attention. | CPU doesn't waste time polling. | Allows CPU to continue other work until event occurs.

→ Types

Hardware interrupts – from external devices (keyboard, mouse, timers, network, etc.). | **Software interrupts / exceptions** – triggered by CPU (e.g. divide-by-zero, system call instruction).

→ Interrupt vector table

Fixed memory table mapping **interrupt numbers → ISR addresses**. | Each interrupt has an entry.

→ What happens on an interrupt?

CPU detects interrupt, suspends current execution. | CPU **saves context** (PC, registers, flags). | CPU looks up ISR address from **vector table**. | CPU jumps to ISR and executes it (service device, clear flags). | ISR returns; CPU restores saved context. | Execution resumes where it left off.

→ Pros / cons

Efficient CPU usage; good responsiveness.
– Code more complex; must handle concurrency and re-entrancy.

13.2 DMA – Direct Memory Access

→ Motivation

Many I/O operations involve moving large blocks of data between device and memory. | If CPU does this byte-by-byte, it wastes cycles and slows everything.

→ DMA concept

DMA controller can transfer data between memory and device **without involving CPU for each byte/word**. | CPU sets up DMA: source address, destination address, length, direction. | DMA then performs transfer over system bus. | When done, DMA triggers an interrupt to notify CPU.

→ Benefits

Frees CPU to do other work while data moves. | Higher throughput for bulk transfers (disk, network, etc.).

LAB 1 — Linux + Shell + C (VERY EXAM-POPULAR)

Essential Commands
Directory ops: mkdir, cd, pwd, ls, ls -la, file * | Viewing files: cat, head, tail | Counting: wc → lines, words, bytes | Redirection: > overwrite, >> append | Pipes: cmd1 | cmd2 | Search/transform: grep, grep -E, sed 's/a/b/g' | History / help: history, man cmd, cmd --help
Regex Quick Patterns
Contains "zoo": grep -E zoo | "zo+" = "zoo", "zooo..." | "zo*" = "z", "zo", "zoo..." | Starts with: ^zoo | Word boundary: \<zoo | Seven-letter words c.....n: grep -E "^c.....n$"
Loops, If, Variables
For loop: for ((i=1;i<=10;i++)); do echo $i; done
Arithmetic: z=$((x-y))
If (strings): [$S1 = $S2]
If (numbers): -eq -ne -lt -le -gt -ge
Wildcards
* = match all filesExample: /etc/*.conf
Shell script basics
#!/bin/bash echo "one"
Make executable: chmod guo+rwx file.sh
Run: ./file.sh
C compile
Compile: gcc -o myprog myprog.c | Redirect output: ./myprog > dump
Find username / passwd line
Username: whoami
Locate user in passwd: grep "$(whoami)" /etc/passwd
Web log analysis (very examinable!)
Unique hosts: awk '{print $1}' access_log | sort -u | wc –l
Count IPs starting with 148. awk '{print $1}' access_log | grep "^148\." | wc -l
Most accessed file: awk '{print $7}' access_log | sort | uniq -c | sort -h

LAB 2 — Networking + Sockets

Network Info Commands
Interfaces:
ifconfig or nmcli d show <iface>
MAC address: ip link | IP (v4/v6): ip addr | ARP table: cat /proc/net/arp | Device stats: cat /proc/net/dev
/proc system info
Memory: cat /proc/meminfo | CPU: cat /proc/cpuinfo | Disk: cat /proc/diskstats | Per-process info: /proc/<PID>/status (VmSize, VmPeak, etc.)
Connectivity
Check interface up: ifconfig / nmcli g | Ping IP: ping 8.8.8.8 | Ping domain: ping bbc.co.uk | DNS queries: nslookup, dig | Trace route: tracepath host or traceroute host
Client / Server architecture (C TCP sockets)
Client uses:
gethostbyname() | socket(AF_INET, SOCK_STREAM, 0) | connect() | write(), read()
Server uses:
socket() | bind() | listen() | accept() | read() / write()
Compile:
gcc client.c -o client | gcc server.c -o server
Run:
Server: ./server <port> | Test via: telnet localhost <port>
Manual HTTP (EXAM TRICK)
telnet <host> 80 | GET / HTTP/1.1 | HOST: <host>

LAB 3 — Raspberry Pi + GPIO + Ultrasonic Sensor + Python

Playing Sound (pygame)

```
import pygame
pygame.init()
snd = pygame.mixer.Sound("file.wav")
snd.play()
```

gpiozero Buttons + LEDs

```
from gpiozero import Button, LED
btn = Button(17)
led = LED(4)
btn.when_pressed = led.on
btn.when_released = led.off
```

Blink:

```
while True:
    led.on(); time.sleep(1)
    led.off(); time.sleep(1)
```

Button → Sound:

```
btn.when_pressed = snd.play
```

Ultrasonic Sensor (HC-SR04)
(MOST EXAMMABLE SECTION)
Distance formula

$$d = \frac{34300 \times t}{2} = 17150 \times t$$

Where:
speed of sound = 343 m/s = 34300 cm/s
t = pulse duration (seconds)
Key Measurements
Trigger pin HIGH for **10 µs**
Echo pin HIGH for travel-time
Min range 2 cm, max ~4 m
Essential Code Pattern

```
GPIO.setmode(GPIO.BCM)
TRIG=23; ECHO=24

GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

GPIO.output(TRIG, False)
time.sleep(2)

# Trigger
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)

# Measure pulse
while GPIO.input(ECHO)==0:
    start = time.time()
while GPIO.input(ECHO)==1:
    end = time.time()

distance = (end-start)*17150
distance = round(distance,1)
```

Beep Frequency Logic
(Assignment 1)

```
def beep_freq():
    d = distance()
    if d > 50: return -1      # no beep
    elif d >= 30: return 1    # 1 beep/sec
    elif d >= 20: return 0.5  # 2 beeps/sec
    elif d >= 10: return 0.25 # 4 beeps/sec
    else: return 0            # continuous
```

Main loop:

```
freq = beep_freq()
if freq == -1: off
elif freq == 0: on
else: on 0.2s, off freq seconds
```

Distance → Different Audio Files
(Assignment 2)
Distance (cm)Action>50No sound50–30Play audio130–20Play audio220–10Play audio3<10Play audio4