C – Week - 2

L1 No. representa^n in a computer

- Humans use variety of symbols. Computers are built using only 0 & 1.
- Inputs of quad. eq^n → a, b, c. Are these arbitrary no.? How are they stored? How are they recognized as nos.?
- $5 \rightarrow 5 \times 10^0$, $27 \rightarrow 2 \times 10^1 + 7 \times 10^0$
- Binary nos.

$$5 \text{ (decimal)} = \underset{2^2}{4} + \underset{2^1}{0} + \underset{2^0}{1}$$

$$(5)_{10} \rightarrow (101)_2 \quad = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 101 \text{ (binary)}$$

$$27 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

- How many bits do we need in gen.?

  8 bit nos. → $0 \longrightarrow 255$ $\boxed{(2^n - 1)}$

  $2^8 - 1 = 256 - 1 = 255$

- Physical memory → circuits, uniform size preferable. CPU registers. Engg. tradeoffs.
- C → basic int. data type ⟶ 32 bits
  ↳ primitive data type, portability.

L2 Negative No. & Hexadecimal Representation

MSB ← 00001111 → LSB
↳ Most Significant Bit ↳ Least

- For negative integers, MSB → 1 (-ve), → 0 (+ve)

Signed / 0011 = +3
Magnitude 1011 = -3
Convention

1 bit for sign → 0 to $2^{N-1}-1$ redundant
→ $-(2^{N-1}-1)$ to 0

- Complement Convention

$x \geq 0 \Rightarrow$ store x as reg. binary no.
$x < 0 \Rightarrow$ store compliment of x

Compliment $\Rightarrow 2^{N-1} - x$ for N-bit no.

Eg. 8 bits 17 ⇒ 00010001
-17 ⇒ $2^8 - 17 = 239$
= 1110 1111

Largest +value ⇒ +127 = 0111 1111
" - " ⇒ -128 = 1 0000000

Adv. → covers full range, easy to compute in hardware, arithmetic easy

- Binary is too difficult for humans to read ⇒ hexadecimal.

$16 = 2^4 \Rightarrow$ 1 digit in base 16 = 4 bits

$(0101)_2 = (5)_{10} = (5)_{16}$
$(1001)_2 = (9)_{10} = (9)_{16}$
$(1010)_2 = (10)_{10} = (A)_{16}$
$(1101)_2 = (13)_{10} = (D)_{16}$

0 . . . . 9, A, B, C, D, E, F

- 8 bit values -

$(\underbrace{0001}_{1}\ \underbrace{0001}_{1})_2 = (17)_{10} = (11)_{16}$

$(\underbrace{0101}_{5}\ \underbrace{1011}_{B})_2 = (91)_{10} = (5B)_{16}$

8 bits ⇒ 0x00 to 0xFF

- Integers -
1. fixed width - possibly CPU or lang. dep.
2. -ve nos. → 2's complement is preferred.
3. hexadecimal - useful intermediate.

L3 Floating Pt. Representaⁿ

$\pi = 3.14$, $c = 3 \times 10^8$ m/s → scientific notaⁿ

- Fixed pt. notaⁿ - integers with scaling factor

16 bit → $(-2^{15}, +2^{15}-1) \times \underbrace{2^{-8}}_{\text{scale factor}}$

- Floating pt. notaⁿ

32 bits → 1 sign, 8 exponent, 23 bits value

- single precision (32 bits) → 10^{-38} → 10^{38}
- double " (64 bits) → 10^{-308} to 10^{308} (scientific computing)
- Half " (16 bits) → used in ML.

L4 Character Representaⁿ & Encoding

- comp. process text, encode text into binary, max. 200 charac. → 2^8 bits are enough.
- ASCII → maps charac. to no./codes
- unicode → consistent encoding, representaⁿ specifies code pts., diff. encodings possible.
- ASCII → 8 bits / charac., but ltd.
  Unicode + UTF-8 → 8 → 32 bits / charac.
  ↳ complex, needs careful handling
- use of std. libraries recommended.
- comp. store & process bits. Memory (array of bits). Registers (grp. of bits)

L5 Instructions Encoding

- Basic op^{ns} defined for CPU.
  ⟹ ALU, Load/Store (Memory/storage), Branching (control flow)
- not all instructions are available on all CPUs. But, can replicate intended behaviour thru other means.
- Eg. of RISC-V -32bit-ISA architecture.
  ↳ other and even smaller encodings are possible.

Eg. 32-bit memory
1M loca^{n} ⇒ 4 MB
Code: 200K " ⇒ 800 kB
Data: 300K " ⇒ 1200 kB

no physical distinc^{n} btw. code & data.

L6 Program Compila^{n}

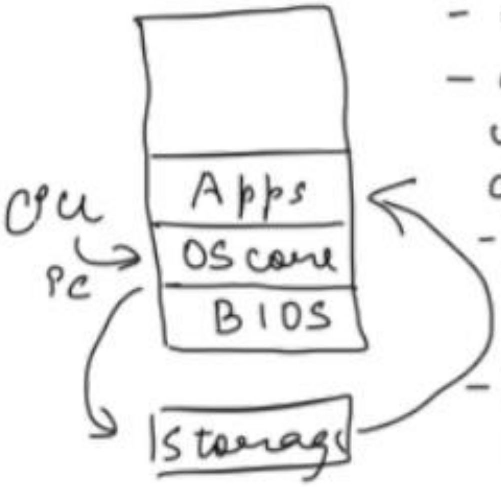
- programs as text → human readable, colours, background, etc. (not part of prog. but IDE), indenta^{n} & formatting may be relevant, files help to organize text into computer readable form.

Human readable → $\boxed{\text{Transforma}^{n}\text{ Prog.}}$ → Machine interpretable.

- Compiler is previously created prog. Input prog. is data for compiler. Output of a prog. is a new executable prog.
- many steps are involved in converting human-readable prog. to machine interpretable prog. → syntax check, map statements to equivalent codes, generate actual machine code.
- Optimiza^{ns} → speed, prog. size, etc.
- can be CPU dependent.

L7 Role of OS in a Computer

- write the prog. to enter into comp. I/O using peripherals, save data as encoded text using file sys.
- compile the prog., it is multiprocessing (can execute several prog.)
- Key things → compiler app, browser/ web based, memory mgmt.
- execute the prog. → load into memory after flow of control.
- In all these activities, OS acts as the coordinator.
- OS → is a prog., 1st prog. executed on startup, coordinates loading & executing other prog.
- Kernel - OS core - almost negligible interac^n with humans.
- Booting / Bootstrapping



- CPU power on
- custom memory - preloaded with code to access disk, activate video, keyboard, etc.
- BIOS accesses storage, loads OS kernel to memory
- OS takes over, starts loading apps.

"Fly by pulling yourself up with your bootstraps".