Week-1 C

L1 How a Computer Works

- Computaⁿ → Goal : Add 2 nos.

Computer
↓
Computaⁿˢ

Assume :

1. Nos. can be represented as electrical signals
2. We have a circuit that can add such signals.

Operands: A, B → $\oplus$ → $S = A + B$ (result)

↳ Operation

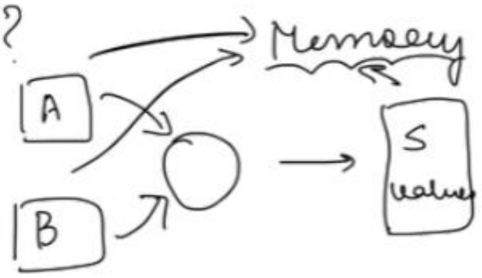
- Req. → rep. no. as electrical signals
  - digital sys. – bit (binary digit)
  - → hardware for performing opⁿ
  - digital logic
  - → comm. with outside world (devices for I/O / config.)

- Repeated computations?

$S_1 = A_1 + B_1$

$S_2 = A_2 + B_2$

⋮



- Memory → bag of words
  - put things in ⇒ write
  - see what is in it ⇒ Read
  - where in the bag it was kept ⇒ Address

- Store some bits ⇒ width
  finite space #addresses ⇒ Capacity

L2 → Generalized Memory & Computations

- $\boxed{A}$, $\boxed{B}$ → (Op) → $\boxed{S}$

  Op ↑ $\boxed{\text{Op values}}$

  ALU ↓ Arithmetic & Logical Unit

  → not only addition but

  Op → +, −, *, /,
  - Logical → compare, or, not
  - sqrt, cos, sin, etc.

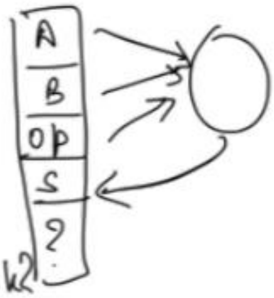
$S_1 = A_1 + B_1$

$S_2 = A_2 / B_2$

$S_3 = \mathrm{sqrt}(A_3)$

⋮

⇒



where to look?

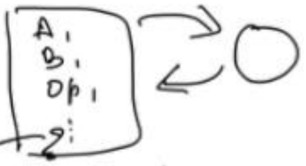
Address Map

-

A, B, S ⇒ data

Op ⇒ instructions

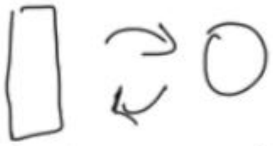
-



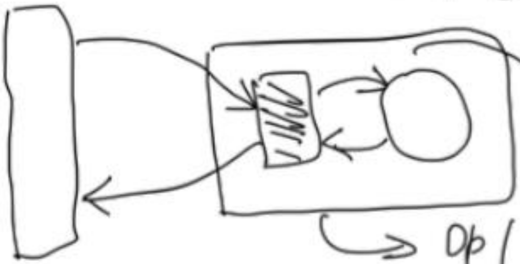
→ other programs (Peripherals) → Typical prog. / data arrangement

L3 Split Memory Architecture

- Memory Size vs Speed

- few wires	- wiring complex	- large prog. (Op eg.)
- small gates, address decoder	- chip area, delay increase	- phy. difficult to make
- compact, fast	- memory slower than Op unit	- speed



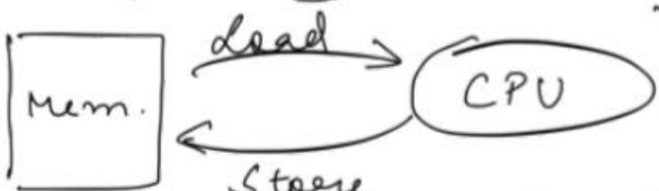
→ Registers - Temp. memory

→ Op / Compute unit

CPU ⇒ central processing unit

↳ Large memory

- Load & Store → move data from memory to registers



Store

↳ move data from CPU (Register) to memory

L4 Loops, Conditions & Branches

- CPU → capable of opⁿˢ to perform computaⁿˢ
- seq. of opⁿˢ are stored in memory, data for computaⁿ in memory, move data btw. CPU & memory. Do this very fast.
- What abt. rep. tasks?
- Prog. = seq. of opⁿˢ
  ↳ CPU tracks posⁿ of next instrucⁿ using Program Counter → 1, 2, 3 - - - -
  What to do if I want to repeat k times?

  Register or Memory ← Loop Counter ← increment & compare with k

  Program Counter ← Step 1
- Conditions or Branches
  → need not restrict Program Counter Δs to loops
  → any condⁿ can be a trigger to Δ the control flow of a prog.
- CPU → Opⁿˢ → Arith. & Logic
  → Memory → Load & Store
  → Control Flow → Branches & Loops

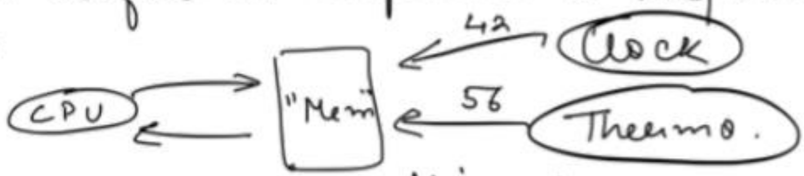
L5 Comm. Btw. Comp. Memory & Outside World

- Comm.
  - → CPU can handle computa"
  - → memory can handle storage

  How to get data into & out of memory?

Reading from memory

- Memory is a black box. Arbitrary logic / elec. / return sth. useful as response to request.
- Peripherals

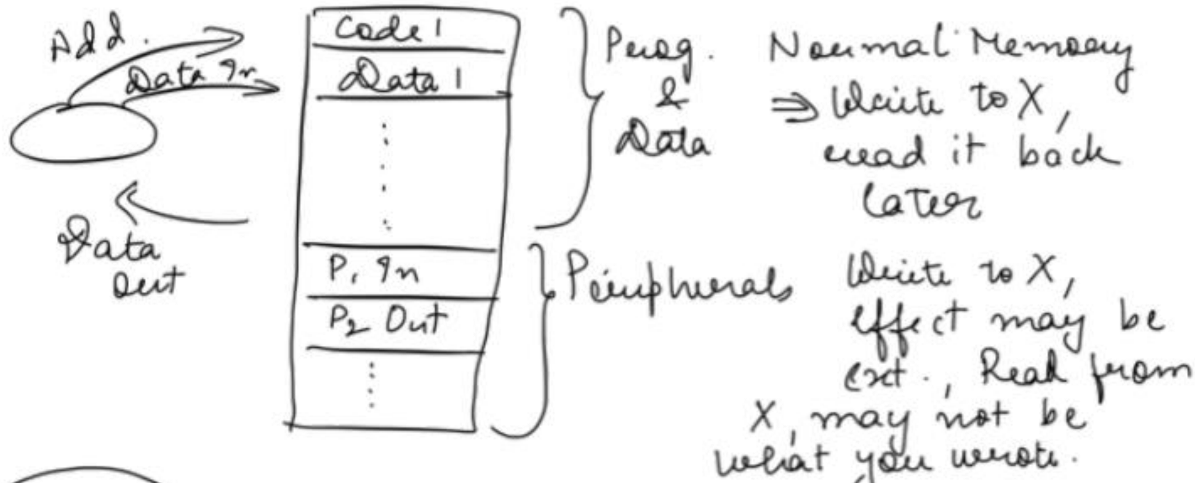


  make up your own mapping!

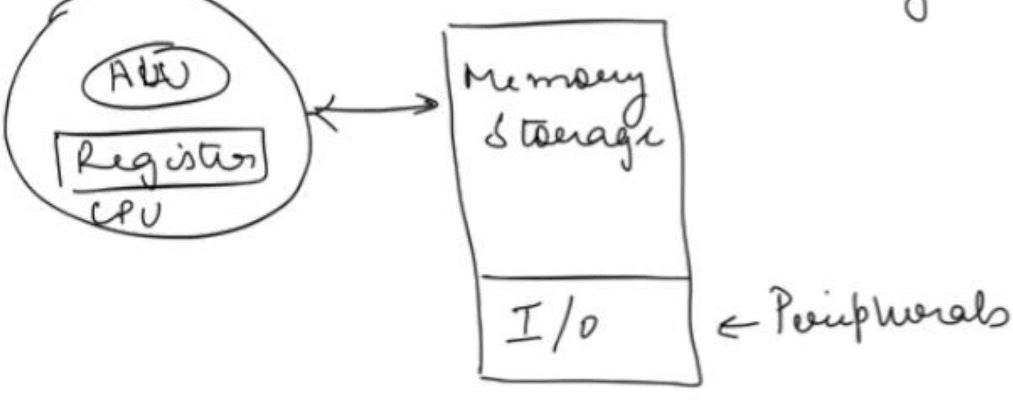
- writing to memory

  write to an add., can reuse same address as read.

-



-



L6 What is Programming

$$ax^2 + bx + c = 0$$

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- define inputs (a, b, c) & outputs (x_1, x_2)
- maps inputs to a memory model so we can load / store as needed.
- break the computaⁿ into steps that our comp. can perform

$$t_1 = b \times b$$
$$t_2 = a \times c$$

↳ temp. storage = variables

- some obs.
  1. every step is broken in terms of comp. primitives.
  2. load / store is skipped - small enough to fit in registers
  3. seemingly obv. steps split out in detail (4ac)
  4. temp. storage use can be further reduced.
- put the variables in app. parts of the memory map.
- put down the seq. of operaⁿˢ needed to do the computaⁿ
- Algos + Data Structures = Programs

L7 Intro to C lang.

- Motivaⁿ
  - → need for OS for new machine
  - → normally prog. in assembly lang.
  - → portability of prog. (use features of new processors)

- compilaⁿ
  - → convert high lvl. descripⁿ to machine code
  - → new lang. — existing ones specialized for other uses.
  - → Bootstrapped compilers
    1. Assembly converts C to assembly
    2. C converts C to assembly.

- Popularity of C
  - → designed very close to hardware & OS
  - → minimal reg. — easily portable
  - → good target for any new processor/sys.
  - → most common first lang.

- Properties
  - → Imperative (ALGOLs) → specify commands
  - → static type & weakly typed (easy conversion)
  - → structured prog. (funcⁿ & blocks)
  - → prog. as free form text → easy to store & share.