31/7/23.

* Questions to be noted.

level-(1-4) ✓ ① Control, ② Machine ③ System software
④ Assembly lauw

High level Code → × Machine Code → (Binary 0/1)

* NHC, Patterson

1) $(100)_{base\,x} + (304)_{base\,x} = (404)_{base\,x}$

~~1x10~~ ~~+x0~~ ~~1x1~~

$1\times b^2 + 0\times b^1 + 0\times b^0 + 3\times b^2 + 0\times b^1 + 4\times b^0$

$= 4\times b^2 + 0\times b + 4\times b$

$\Rightarrow b^2 + 3b^2 + 4 = 4b^2 + 4$

2) $(78)_{10} = (\quad)_7$

$7\times 7^1 + 8\times 7^0 = (\quad)_7$

$49 + 8 = (\quad)_7$

7 | 78	
7 | 11	-1
7 | 1	-4
	0 -1

1.

Decimal → Binary radix

3) $11001 \rightarrow 1\times 2^4 + 2^3 + 1\times 2^0$

$= 16 + 8 + 1$

$= 25$

→ 00110

+1

00111 → $2^2 + 2 + 1$

-7

2's complem → decimal

① check +/-

② invert all no's

③ Add 1

A B C D E

$(\emptyset\times 2 \in D)_{16}$

2 | 3	
2 | 1	-1
	0 -1

2 | 1120	
2 | 560	-0
2 | 280	-0
2 | 140	-0
2 | 70	-0
2 | 35	-1
2 | 17	-1
2 | 8	-1
2 | 4	-0
2 | 2	-0
	1 -0

0000

0111 0000

$10^3 + 10^2 + 10^1 + 1$

2 | 120	
2 | 60	-0
2 | 30	-0
2 | 15	-0
2 | 7	-1
2 | 3	-1
2 | 1	-1
	0 -1

decimal - binary

2 | 66	
2 | 33	-0
2 | 16	-1
2 | 8	-0
2 | 4	-0
2 | 2	-0
	1 -0

01

(0x2ED) 16 → hexa → Binary → Grp of 4

Grp of 4

69 decimal

2 | 69	1
2 | 34	1
2 | 17	0
2 | 8	1
2 | 4	0
2 | 2	0
2 | 1	0
0	1

1000101

2 | 68	0
2 | 34	0
2 | 17	0
2 | 8	1
2 | 4	0
2 | 2	0
2 | 1	0
0	1

1000100

2 | 120	0
2 | 60	0
2 | 30	0
2 | 15	0
2 | 7	1
2 | 3	1
2 | 1	1
2 | 0	1
0	

2 | 2	0
2 | 1	0
0	1

(111 1000) (0 1000101) (1000100)

111100000000010 1000101 1000100

0 1 2 ... 9, A, B, C, D, E, F

10 11 12 ... 15

2 | 14	0
2 | 7	1
2 | 3	1
2 | 1	1
2 | 0	1
0	

1110

2 | 13	1
2 | 6	0
2 | 3	1
2 | 1	1
2 | 0	1
0	

1101

00 10 1110 1101

(5)

2 | 124	0
2 | 62	0
2 | 31	1
2 | 15	1
2 | 7	1
2 | 3	1
2 | 1	1
0	

1111100

11011 · 1101 · 0101 · 10010

$(abc)_x = (124)_{10}$

$124 = ax x^2 + bx x^1 + cx$

$= ax^2 + bx + c$

$4x16 + 100 + 4 \cdot 4x4^2 + 5x4$

$64 + 8$

$5x4 + 4x4^2$

10) 64 → 63

2|32
2|16 - 0
2|8 - 0
2|4 - 0
2|2 - 0
2|1 - 0
0

0 1 1 1 1 1 1
0 1 0 0 0 0 0
1 0 0 0 0 0 0

0 0 0 0 0 0 0
0 1 1 1 1 1 1
1 0 0 0 0 0 0

1 0 0 0 0 0 0 → -32

0 1 0 0 0 0 0
1 0 1 1 1 1 1
1 1 0 0 0 0 0 0

-32 → +32

-3 -2 -1 0 0 1 1 1 1 1 1

-32 10

1 0 0 0 0 0 0
(32) 0 1 1 1 1 1 1
+1
0 0 0 0 0 0

2|31
2|15 - 1
2|7 - 1
2|3 - 1
2|1 - 1
0 - 1

0 0 1 1 1 1 1 1
1 1 0 0 0 0 0 0

-32 to 31

-25 to 25 (31)

* Popular Analog Computer - ?
  ↓ ~~Dantar~~ Mantar

Analog comp > > Digital Comp (some aspects)

*
* finite no. of states in Digital Comp
  You can't predict no. of states in analog com
* finite no. of configurations Ex: playing chess.
  Ex: football
* As soon as counting, what is no. of states
  (Reliable)
  ↓ (On computable numbers)
  If u give input, o/p will come or not
  & what is it in certain time.

Digital Computer →

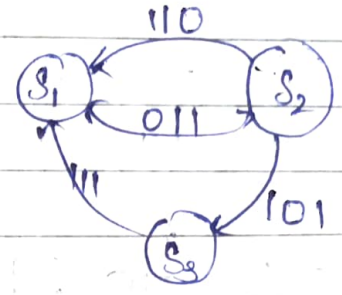
↓

Set of States → Finite State Machine.

↓

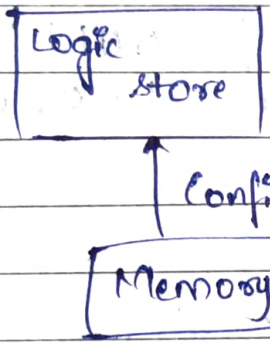
Configurations (bits)

✓ it can do only certain no. of things



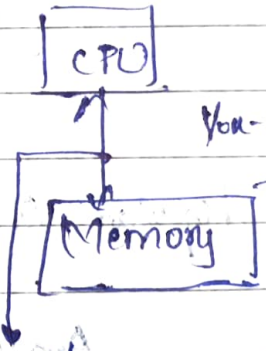
* (Dark Silicon)

⇒



Logic | Computations

→



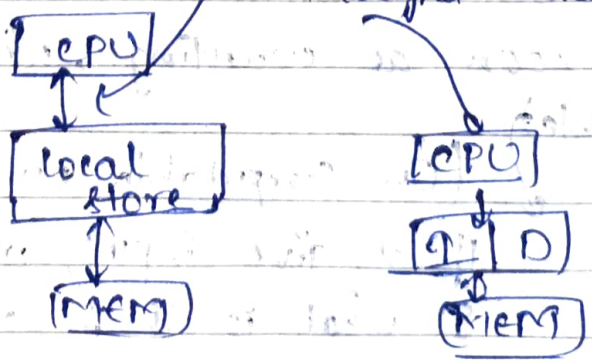
Von-neumann Architec

Von-neumann BottleNeck.

* Solving Von-neumann BottleNeck:

→ Harvard Architecture

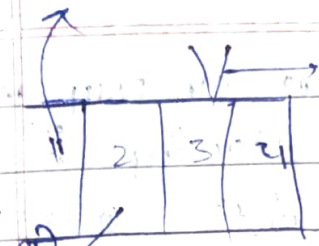
→ Modified Harvard Architecture.



(Instructions) (separate place)

Von-neumann

for finding prime no's → can solve many other prob



Tape (Memory)

head can go any where (Processer)
↳ within finite no. of states we can compute the prob.

→ can find out prime numbers
Shoer's theorem

→ Turning didn't assumed time.

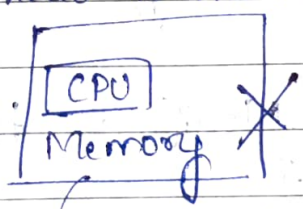
* Von-neumann asking to it can transfer should be there.
↳ everything depends upon definition of states (1, 2, 3, 4).
* All states are pratically there.

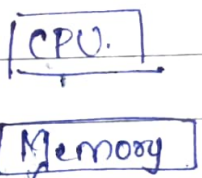
Elan Turing
finite states,

Von-Neumann → Tape is constraint

* Memory (Tape) → finite in size
Processer (head) → No Rules
↳ head to move to other state.

*





fundamental computing, whether it is finite state or not
→ I/p ⟹ O/p

** CPU and memory are not co-located bcz it is essential for entire computer to transition through finite states. If the CPU & Memory

are distinct then processer can only get finite no: of information allowing it to transition to a deterministic state.

* What if we constraint head.

Alternate of Von-Neuman architecture is Dataflow Computer → Big Data

→ Data flows to the Computer.

* Memory is Infinite.

Von-Neuman Architecture.

① Compute of data are kept separate

② Thin link connects the 2 which is sufficient for state transitions of the machine

③ Thin link is a bottleneck which can be handled by,

a) Harvard / Modified Harvard Architecture
Introduced caches

b) Multi-theading

c) Near memory processing

d) In-memory processing

* Searching - indeterministic ??

[1 1 1] → only integers, that's so possible.

* Suduko - indeterministic [can write program but chances of not giving o/p]

Instruction Set Architecture (ISA)

Turing Completeness:

— Any Machine that can simulate a generic set of problems solvable by a Turing Machine.

Defn of TM (theory)

can solve any prob

The Machine is turning if after some time the machine stops.

— HALTING

What is Machine?

* A mathematical language to define a machine.

VHM → Engg soln but theoritical processing ✓ memory ✗

* A unification of all languages is required.

* 10110101 → Instruction
  ↓
  Executes on an CPU.

* Executable file → which is not program
  ↓
  compile → executable file
  text file with 1100 → executable file
  ↳ Not executed.

Execution? Interpretation of CPU. [Understand

Greek → Interpret ✗
Processing ✓ (Movement of mouth)

1011 01010001

↓

add

↳ mnemonic (symbol) → 1st high-level language

based language

① Designed the mathematical lang

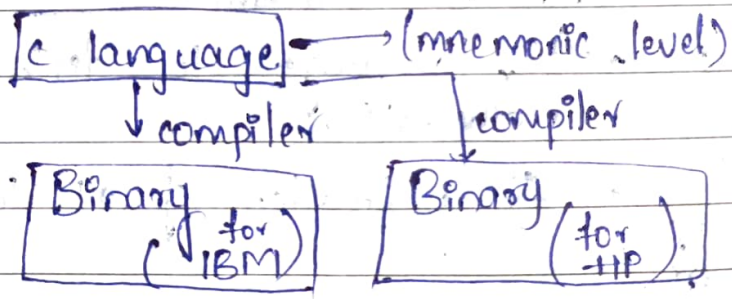
② Enginee → Machine

Mathematical lang → 1101..

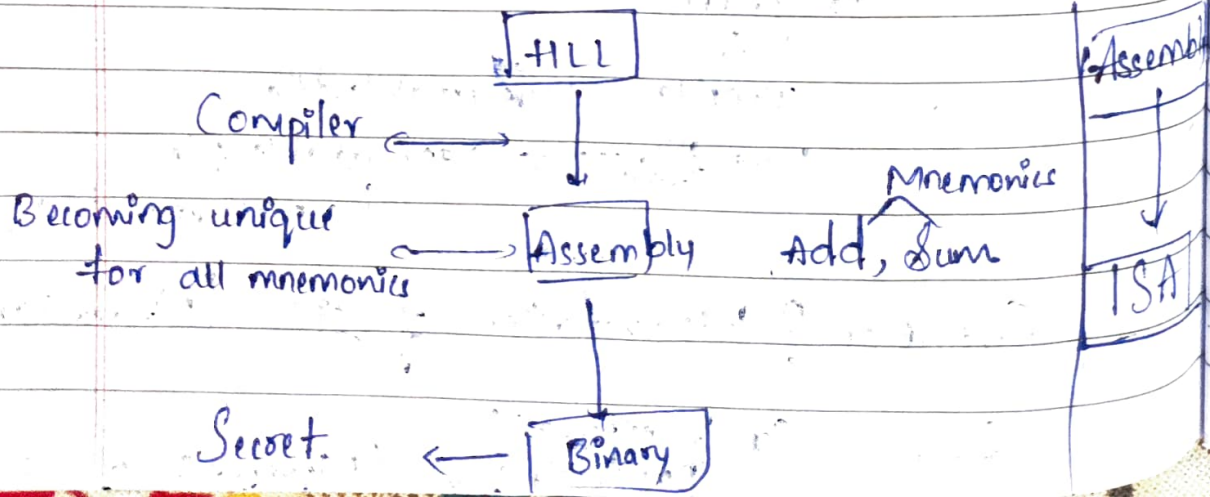
* Compiler - Translation of english to low-level lang (110...)

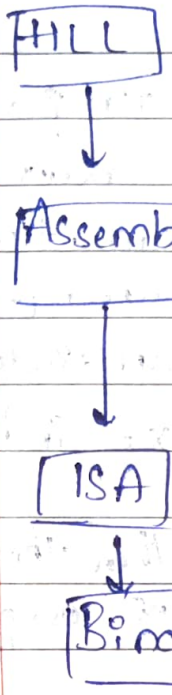
* Instruction - Executes in an CPU.

Interpretation of CPU.



Mnemonic based language → Assembly language





ISA_s → Is a translation of lang

RISC CISC

If i have
(Assembly lang)
sbn

```
sbn  a, b, line no.
     a = a - b
     if (a < 0)
        goto line no.
```

Q) Do (a+b) using only sbn?

initialize, temp = 0

```
1. sbn temp, a, 2   // temp = temp - a
2  sbn b, temp, exit  // b = b - (-a) = b + a
exit:
```

3. Instructions

Q. Try to sum 1+2+...10 using sbn

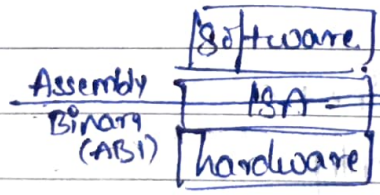
3 registers: temp = 0, one = 1, Pindex = 10, Sum = 0 } Initial values.

Programming lang

1. sbn temp, temp, 2
2. sbn temp, index, 3 // temp = temp - index
   = -index
3. sbn sum, temp, 4 // sum = sum - temp
   sum = sum - (-index)
   = sum + index
4. sbn index, one, edit // index = index - one.
5. sbn temp, temp, 6 ⟶ When there are
6. sbn, temp, one, ! no computers.
   edit.

→ ISA implemented
Contract b/w software & hardware.
Software engg dont need to learn hardware

→ Instruction set Architecture. (ISA)



A good interface b/w software & hardware.

→ Legacy codewares

Software / Binary ✓ Binary / Hardware

~~Assem~~
Application Binary (ABI)

→ Hardware people don't want to reveal Binary

a = a+b . // HLL

↳ load a

load b

add a,b

store a

load → mnemonic → symbol for a binary pattern

symbol / mnemonic

* HLL Instruction ≠ Assembly instn.

~~do~~

ISA_1 ↓ n_1 no. of instructions

ISA_2 ↓ n_2 no. of instru

$n_1 < n_2$

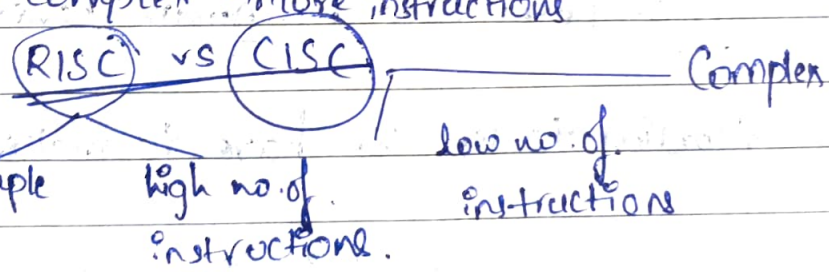
14/8/ H/w with support for $x >> n_2 >> n$

Founder of intel, moore

Transistors through logic circuits.

More complex → More Power, More transistors.

Less complex → More instructions



Examples: CISC → x86 (IA64) Linux
x32 (IA32)

ISA

RISC → ARM
MIPS
RISC-V → 1st opensource ISA

↓ Not Complex instructions

CISC → no. of. inst → lower / lower no. of circuit
frequency don't know also
Better. apart from Processer.

used for Complex, simple

* CISC:

If x86 → ISA ISA composed of
→ processer.

* Processer → which ISA, we are going to implement in

* Usage Processer
lower power constraint Not Complex, as simple as,

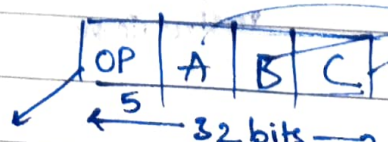
X86 never used in Smart Phones

According to ISA, we have to make OS.

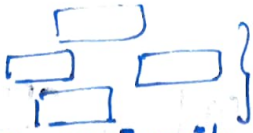
Parameters for designing an ISA:

* An instruction is called word when it is in memory.

- Design Philosophies



OP code number



} in Processer

$2^2 = 4$ circuits. → 2 bits in OP code

OP) OP code is a numeric identifier for the circuit which it invokes.

* Registers are temporary peice of memory which helps instruction life cycle.

CPU

regis-ters.

Factors: 1) Memory Operations
2) No. of Instructions

* Registers are kept inside the processors to reduce main memory access

Main memory is typically made of D-RAM & capacitor based technology.

* We minimize accesses to main-memory bcz,
  1. Von-Neumann BottleNeck
  2. Main-memory latency
* Addressing Modes are implemented in order to minimize the time taken for searching element.

RISC → fixed, CISC → Variable length (16 byte, 8 byte, ..)

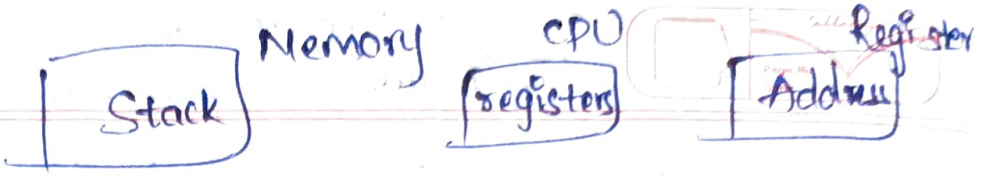
* Addressing ~~mode~~ of registers is [DIRECT] only.

Every Register has name.

If the address is in itself

* Register Addressing → return form function.
* Address is also in Register.

Register | Address



9bits → 29 bits → 512 bytes

* Memory → byte Addressble.
  Memory is always byte Addressble or Word Addressable
  ↓ Every byte uniquely. ↓ Every word uniquely
* Word → It's just Instruction.
* If instruction in Memory, it is called Word.
  → 32bit instruction

2^{10} = 1024 = 1KB.
2^{20} = 1MB.
2^{30} = 1GB.
2^{32} = 4GB.

* higher Instruction lengths mainly to access larger memories.
* lower size instruction → (Simpler) Advantage
  8bit

** 8bit → 128 bit processors
  processors ↓ Used based on Usecase.

Instruction Processer Registers.

* No. of Wires ↑ → Problem is More complex

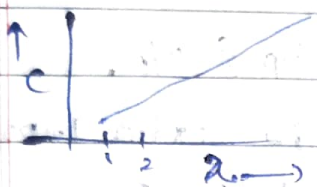
opcode	n_1	n_2	n_3	..	n_x

x no of partitions

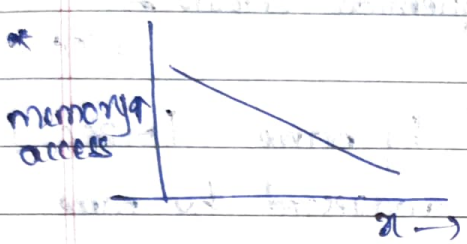
x address instruction.

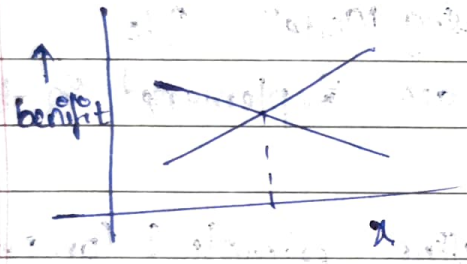
$x \uparrow \longrightarrow$ complexity increases.

$x \uparrow \longrightarrow$ memory access $\downarrow$



$c = a + b$.





$c = (a+b) \times \frac{d}{e}$

↓

there are

$c = a + b$;

$c = c \times d$;

$c = c / e$;

* 0 address instruction is necessary when there is no general purpose registers.

↓

it can store any address.

Program Counter, it can store only specific data.

↓

special purpose registers.

↓

stack or some special purpose register

↓

It is in main-memory.

* opcode | | | .

$x = 10$

↓

Registers.

Why ? am calling address Instruction

Clock → Sequential Circuit

* If it is a Sequential Circuit → May be Memory
  ↑
  Apart from i/p & o/p,
  there will be extra Input.
* If it is not a Sequential Circuit → Not Memory

Register can be addressed by name. [DIRECT]
Main Memory can't be addressed by name

* Can we change addressing Mode? No.

⇒ All addressing modes are implemented in Hardware

How do we define Simple & Complex.
Simple Instruction?
Uniform in length (all instruction)
All computations are simple.

* RISC-V
  32 bit Variant → 32 registers. - each 32 bits length
  Length of Instruction

CISC: 512 regis / \ } 64 bit Os.
Each Register : Instruction size
= 128 bit length

RISC: Regular

opname rd, rs, rt → Operand Registers

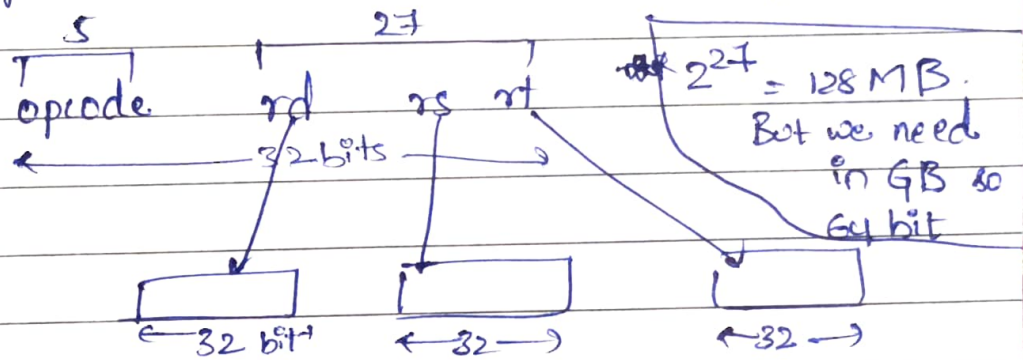
destination Source Target

3 address Register

*** How can rd, rs, rt registers can Interchange, Give an example!.

*** Why do we have 32bit Instruction set & 64 bit Instruction set ?

Longer Instructions → Access more Memory



Addressing at word & Byte level