Introduction To Processor Design

5/1/26

- Moore's law allowed us to pack more transistors onto the same chip, increasing computation speed.
  However, non-ideal effects start to become more prominent in the lower dimensions, leading to the death of Moore's law.

- Now, developments in processor architecture are focusing more on domain-specific accelerators like AI accelerators, photonic computing, IMC, etc.

Textbook: Computer Architecture: A Quantitative Approach, Patterson, Hennessey and Kazgrakis (7th | 6th Edn.)

- RISC - Reduced Instruction Set Computer (Open Source)
  ARM - Advanced RISC Machines (Proprietary)

★ Grading:

Quiz - 10%

Endsem - 30%

Project - 60% → Assignments included (each 5%) ← Probably 2

- Modern processors have heterogeneous tiling, ie, subprocessors that are optimized for certain applications like GPUs, NPUs, TPUs.

  TPU ⟶ Systolic Architecture (Google's TPU)

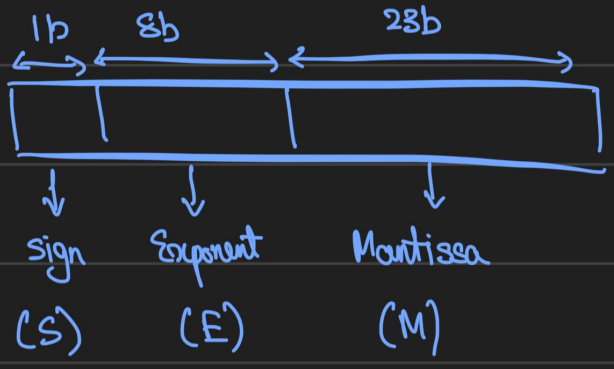
8|1|26

✱ Computer Abstractions and Technology :-

- Edge Computing – local Computation. Best for speed and privacy.
  Cloud Computing – Centralized Computation. Best for accessibility and performance.

→ Floating Point Number System :-

- IEEE 754 is a standard used to represent 32 bit floating point numbers.



32 bit – Single prec.
64 bit – Double prec.

- The number is of the form $s(1.M \times 2^{E})$
  - ↳ E – biased, ie,
    - > 127, +ve exponent
    - < 127, –ve exponent
    - = 127, 0 exponent

- To do floating point multiplication,

$$s_1(1.M_1 \times 2^{E_1}) \times s_2(1.M_2 \times 2^{E_2}) = (s_1 s_2)(1.M_1 M_2 \times 2^{E_1+E_2})$$

$s_1 s_2 \longrightarrow$ sign $\Rightarrow$

s_1	s_2	$s_1 s_2$
+	+	+
+	–	–
–	+	–
–	–	+

} XOR (if + = 0, – = 1)

$M_1 M_2 \longrightarrow$ III to regular multiplication

$E_1 + E_2 \longrightarrow$ Done using normal addn. instruction

- This type of computation is used since it gives us greater accuracy using lesser amount of bits.

Mini - Assignment : Design a floating-point ALU.

→ Understanding Performance :-

- Algorithms – Determines the number of operations to be performed

- Language, Compiler, Architecture – Determines no. of machine instructions per operation. (Compiler Architecture determines Compiler efficiency)

- Processor & Memory – How fast instructions are executed

- I/O System – Determines how fast I/O operations are executed. (Interface, Bandwidth, Protocol)

High Level Language

↓ Compiler

Assembly Language → Instruction Set

↓ Assembler

Hardware Representation

→ Inside The Processor :-

- Datapath – A sequence of operations on data
  Control – Sequencing of datapaths
  Cache – Collection of SRAMs for fast memory access

- Response Time - Time taken to finish an operation
  Throughput - No. of tasks performed by the processor in unit time.

- Performance = 1/Execution Time

- The operations of the CPU is clocked by a digital clock of a certain frequency. Higher frequency → More operations.

CPU Time = CPU Clock Cycles × Clock Time Period

Clock Cycles = Instruction Count × Cycles per Instruction
↳ Depends on Compiler, ISA, Program

⇒ CPU Time = Ins. Cnt × CPI × T

→ Power Metrics :-

$$\text{Power} = f C V^2$$

f - Clock frequency
C - Capacitive load
V - Operating Voltage.

12/1/26

Ex: Represent -5.5 in the floating point number system.

$$-5.5 = -101.1_2$$

$$= (-1.011 \times 10^{2})_2$$

1	129
0	64
0	32
0	16
0	8
0	4
0	2
	1

Exponent $= 2 + 127 = \underline{129} = 10000001$

Mantissa $= 011000.....$

$\Rightarrow -5.5 = 11000000101100000000000000000000$

sign | exponent | mantissa

Exp: Express $01000001101010100....$ in Decimal

$\Rightarrow (1.0110101 \times 10^{10000011})_2$

$1.0110101 = 1 + 0.25 + 0.125 + 0.03125 + 0.007805$

$\approx \underline{1.414}$

$10000011 = 2^7 + 2^1 + 1 = 128 + 2 + 1 = \underline{131}$

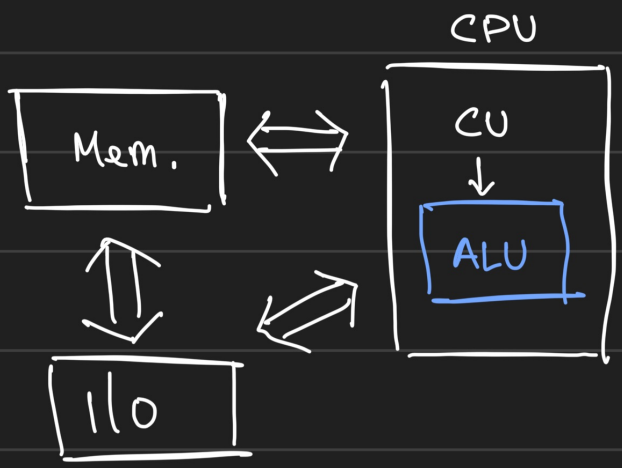
76543210

$\Rightarrow$ Exp : $131 - 127 = \underline{4}$

$\Rightarrow N = 1.414 \times 2^4 = 1.414 \times 16$

$= \underline{22.624}$

→ Von Neumann Architecture :-



- Traditional computational model.

- Registers: Memory devices placed closed to the computational units, to speed up operations. Made using flipflops so expensive to fabricate.

19/1/26

→ Instructions :-

- The most basic operations / tasks that can be done by the processor. Expressed in Assembly Language.

- Compilers convert high-level code into Assembly language in the form of instructions. Assembler converts assembly into binary.

- **Instruction Set:**

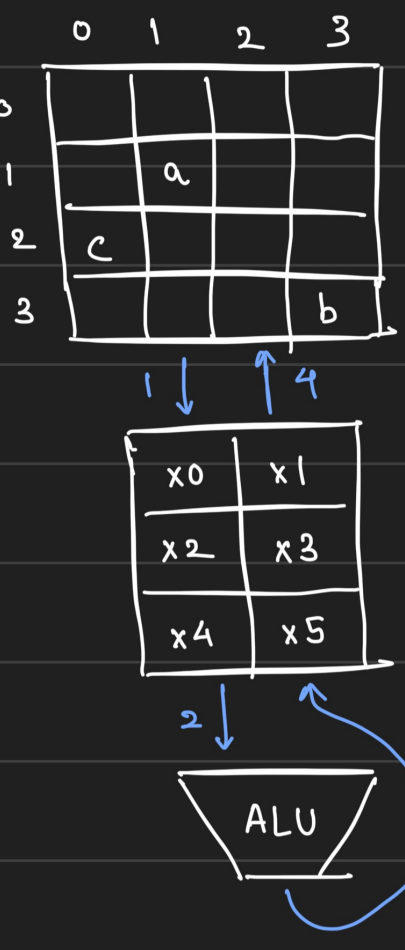
The set / group of instructions that are possible for a processor. ex: ARM, x86, RISC V.

- Different processors can have different instruction sets of varying complexity.

- **RISC V Instruction Set :-**

- Developed in UC Berkeley as open ISA, and is now maintained by the RISC-V foundation.

a: M(1,1)
b: M(3,3)
c: M(2,0)



Capacitive (DRAM)
↑
Main Memory (RAM, Cache, etc)
↓
SRAM

Registers ⟶ Flip flops.

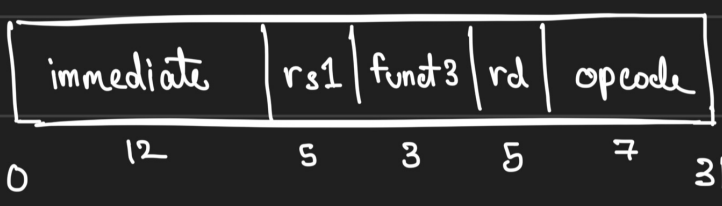
Operation: $c = a \times b$

CISC	RISC
MULT M(2,0) M(1,1) M(2,3)	LOAD x0 M(1,1)
↑ destination, M(1,1) M(2,3) source	LOAD x1 M(3,3)
	MULT x2 x0 x1
	STOR M(2,0) x2

Note: LOAD : Memory → Register, STOR : Register → Memory.

- As shown above, a program in RISC is much bigger than in a CISC type instruction set, since, the data loading operations (LOAD, STOR) are included in the instruction.

- In RISC, any instruction is 32 bits long in its machine language (binary) form.



Whereas, in CISC, the decoding of the instruction is more complicated (Assembly → Machine) than in RISC. There is no specific bit-length for the instruction in a CISC machine.

Therefore, the hardware design is more simpler for RISC machines, due to the ease in decoding the instruction.

- RISC can also exploit pipelining, ie, instructions in parallel, and other instruction - specific optimizations.

- RISC is more popular in specific - purpose machines, like accelerators, Edge devices, low power devices, etc.

- We know that,

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Ins.}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Ins.}} \times \frac{\text{Time}}{\text{Cycle}}$$

(CPI) (Clock freq.)

In RISC ISA, (as compared to CISC)

Instruction / Program is higher.

Cycles / Instruction is lower.

So, both of these metrics make it so that RISC and CISC machines are comparable in performance, on similar hardware.

22/1/26

- RISC V follows the little endian method of storage, ie, LSBs' are stored in the lower memory indices and MSBs' in the higher indices.

	MSB							LSB
number	1	0	1	1	0	1	0	0
index	111	110	101	100	011	010	001	000

- $f = (g+h) - (i+j)$,

```
add t0, g, h   // temp var t0 = g+h
add t1, i, j   // temp var t1 = i+j
sub f, t0, t1  // var f = t0 - t1
```

Operand - Source Registers, Result - Destination Register.

We can design the CPU in such a way that all the 4 operands are passed in the same instruction. However, due to the upper limit of an instruction in RISC V (32 bits), they would have to occupy a smaller space within the instruction.

◦ RISC V Register :-

x0 – Const 0 → Can be used to copy data using ADD instruction

x1 – Return Address

x2 – Stack Pointer → Order in which func. are seen

x3 – Global Pointer → Global Vars.

x4 – Thread Pointer

x5 – x7, x28 – x31 – Temporaries

x8 – Frame Pointer

x9, x18 – x27 – Saved Registers

x10 – x11 – Function Arguments / Results

x12 – x17 – Function Arguments

} 32 × 64-bit Register file

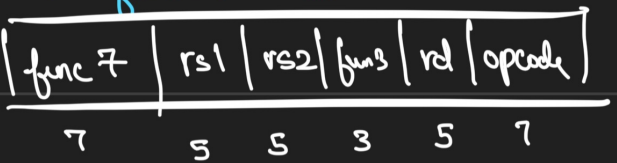
Design Principle:

Smaller is faster

◦ Instruction Format :-

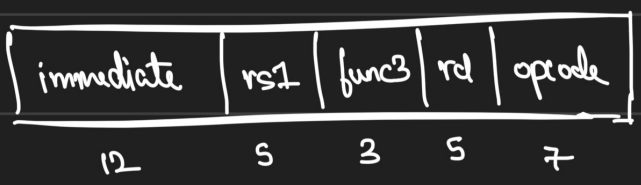
- opcode (7b) – Operation Code
- rd (5b) – Destination Register
- func3 (3b) – 3-bit function code (additional opcode)
- rs2, rs1 (5b) – Source Register
- func7 (7b) – 7-bit function code.

} R-format (registers)



- opcode (7b)
- immediate (12b) : Constant operand (offset
- rs1 (5b)
- rd (5b)
- func 3 (3b)

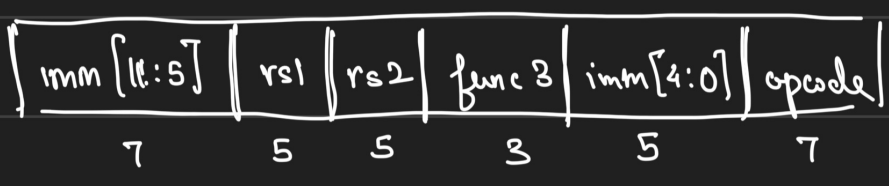
} I - format (immediate)



Reign Principle : Good Principle ⇒ Good Compromise (Different formats must be as similar as possible)

- opcode (7b)
- immediate ([11:5], [4:0] - 12b) : Offset added to base address (rs1)
  ↳ Destination Register
- rs1, rs2 (5b)
- func 3 (3b)

} S format



- Branch Addressing :-

- Branch instructions have opcode, two registers and target address.
- PC relative address (PC - Program Counter) encodes the target address as some offset of the PC.

◦ Jump Addressing :-

- Jump and Link instructions (jal) uses 20 bit immediate for long jumps.

- If the address is much farther away, the target is loaded from the memory into a register.