Mid-Term Note (Spring 2025)

Course code : CSE335

Course Title : Computer Architecture & Organization

Provided by:

Name : Bishal Biswas

ID: 221-15-5394

Section: 61_G

Attention : Since this is a handwritten note, there may be minor errors like spelling mistakes or calculation mistakes or something else. For which I apologize. Those who read this note should first follow the rules & instructions of their Course teacher first and then if they want, they can follow this note additionally.

Planned effort never goes in vain.

(Best of Luck)

Lecture 1: Introduction to CAO

Architecture: Architecture is those attribute visible to the programmer like Instruction set, num. of bits used for data representation, I/O mechanisms, addressing techniques.

Ex: X86 Architecture
instruction set: X86 (CISC)
num. of Bits: 32 bit
I/O mechanism: Memory-mapped I/O

Organization: organization is how features are implemented like control signals, Interfaces, memory technology.

Ex:→ Some X86 CPU may have a dedicated hardware multiplier for fast multiplication.
→ some simple micro-controllen (Ex: early ARM cortex-M10) may used repeated addition instead of a hardware multiplier.

The Nature of computing :

→ Throughout history, people used their brains to do math.

→ Long ago, they counted using fingers, small stones or sticks.

→ Later, simpler tools helped with calculations such as :

- The abacus : A device with beads used for math. It came from Latin & was used to describe a counting board with sand. used since before 1387.

- The slide Rule : A ruler like tool that helps multiply & divide number by sliding parts of it. It was used until 1972.

  Sliding parts marked with number & the way scales are marked adding on the slide rule actually multiplies numbers.

The Element of computers :

The Brain Vs. The computer

→ When we do math using a pencil & paper, the paper helps us to store information.

The information on the paper can include :

- "Instructions" on how to solve the problem (also called a program or algorithm)
- "Numbers or data" need for the calculation.

The human brain does the actual thinking & solving, acting like a "Processor".

The brain does two main jobs :

(i) Control function : Reads & follows instructions in the right order.

(ii) Executive function : Does the actual math, like adding, subtracting, multiplying & dividing.

A computer works in a similar way to the brain. It has important parts that match what the brain does.

(i) Main memory : stores information (like Papers)

(ii) CPU (central Processing unit) : Thinks & solves Problems (like the brain)

(iii) PCU (Program control unit) : follows instructions.

(iv) ALU (Arithmetic Logic unit) : Does math (like adding & subtracting)

(v) Input/output : Lets us put information into the computer & get result (like using a keyboard or screen)

Lecture 2 : Evolution of Computers

→ Machine that could do basic math (adding, subtracting, multiplying, dividing) were made as early 16th century or even earlier.

→ A french thinker, Blaise pascal (1623-1662), created an important early calculator that could Add & Subtract.

→ In the 19th century, Charles Babbage designed the first computer that could do many steps on their own without human help.

These machines were fully mechanical.

His first machine called "The Difference Engine" was made to calculate & Print math table automatically. It could only do addition.

The Difference Engine was not enough to meet the needs. Babbage could not finish it because he thought of much better machine called "The Analytical Engine".

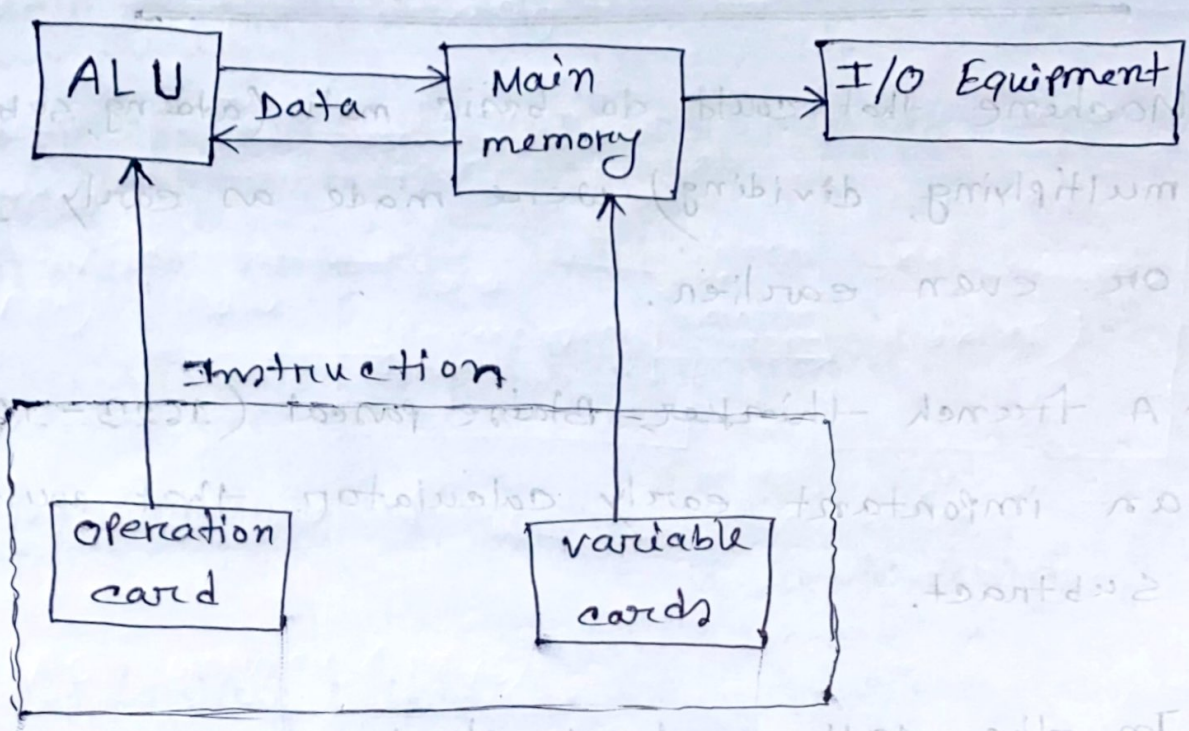


fig: Structure of Babbage's Analytical Engine

Electronic computer:

A mechanical computer has two big problems:

(i) It is slow because its parts have to move

(ii) It is not very reliable at sending digital information.

An electronic computer is much better because it uses electrons, which move almost as fast as light (300000 km/s)

Electronic parts, like vacuum tubes (invented in the early 1900s) allow computers to process & store digital information much faster than any mechanical machine.

Computer Generations :

(i) 1st Generation 1944 to 1958 → Vaccum Tubes

(ii) 2nd Generation 1959 to 1963 → Transistor

(iii) 3rd Generation 1964 to 1970 → Integrated Circuit (IC)

(iv) 4th Generation 1971 to Now

→ Large scale Integration (LSI)

→ Very Large scale Integration (VLSI)

(v) 5th Generation → Antifical Intelligence (AI)

First Generation computers :

→ The first well-known general purpose electronic computer was the ENIAC,

→ Like Babbage's Difference Engine, the ENIAC was made to help the U.S Army create math tables automatically.

→ The ENIAC was huge, weighing 30 tons & using over 18,000 vacuum tubes. It was much faster.

→ ENIAC do 10 digit multiplication within 3 milliseconds.

→ The first computer used machine language written in Binary code which computer could understand directly.

→ A big improvement came Assembly language (1950s) where instruction used symbol like : ADD X_1, X_2

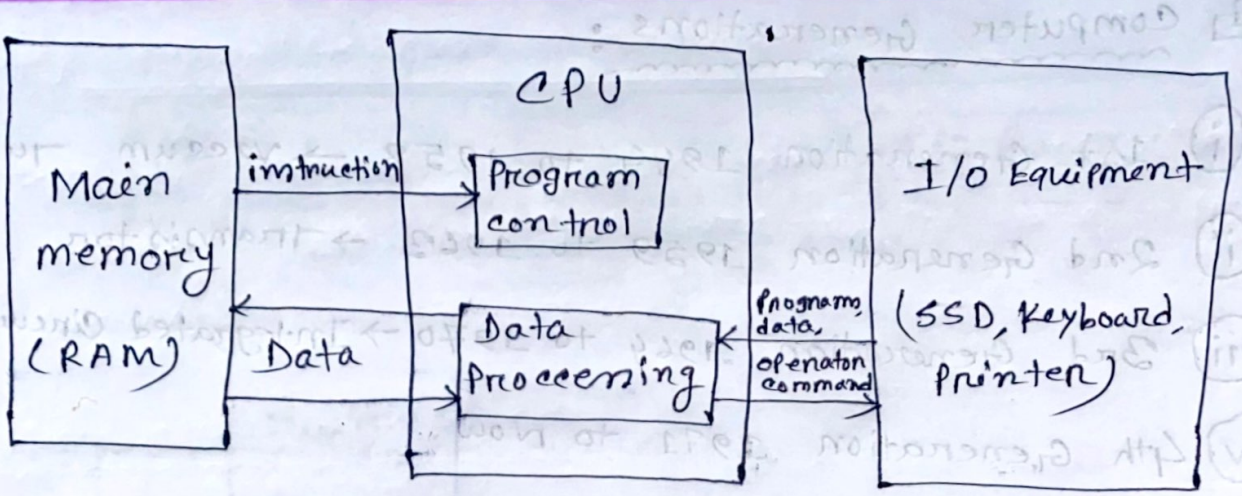


fig: structure of 1st generation computer

IAS computer (1946):

→ In 1946, John von neuman & his team designed the IAS computer at Princeton.

→ It stored programs in memory & used 40-bit words to store numbers or instructions.

→ Numbers were stored as signed binary fractions (e.g +.8125 or −.8125)

→ Instructions had an 8-bit opcode (what to do) & a 12-bit address (where to find data in memory)

→ The control unit read & executed instructions one by one.

Features:

→ small CPU, high-speed registers to store data.

→ Program instruction were stored in memory in the order they were executed.

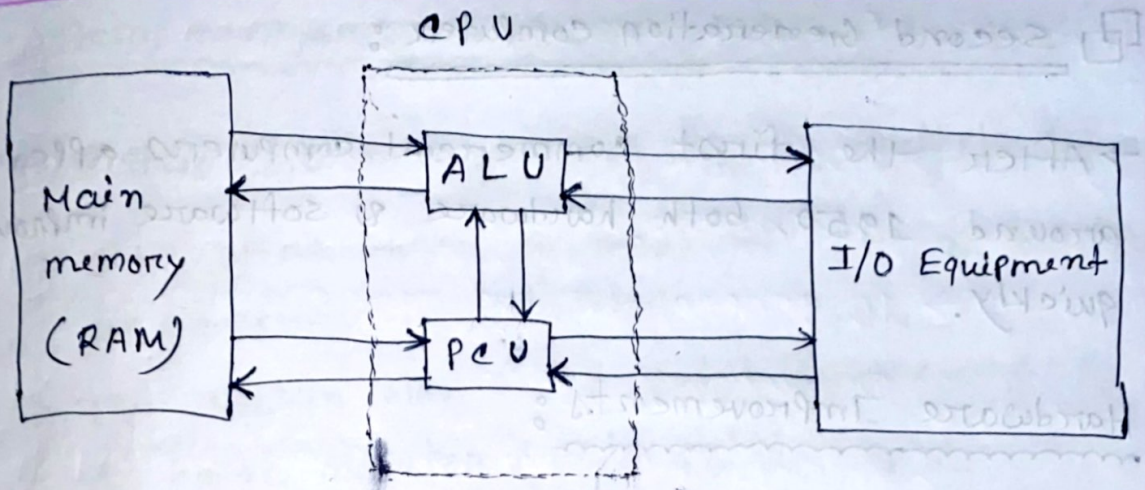


fig: structure of IAS computer

Limitation of 1st Generation computer:

→ No special registers for index control.

→ few CPU registers & No cache memory.

→ No tools to organize programs properly.

→ Better for math tasks, but text processing was slow & difficult.

→ I/O were not a priority.

Second Generation computer :

→ After the first commercial computers appeared around 1950, both hardware & software improve quickly.

Hardware Improvements :

→ Transistors replaced vaccuum tube - they were faster, smaller, cheaper & used less power.

→ Magnetic disk became the main technology for storing extra data.

→ Second-generation computer had more CPU registers like Index registers, which made handling data easier.

Programming Advances :

→ High level programming language use

→ these languages could work on different computer.

→ compiler were created to convert high-level languages into machine language

- FORTRAN → for math & science calculation
- COBOL → for business application, handling both text & numbers.
- Later BASIC, Pascal, C & Java became popular.

System management : (Batch Processing)

→ Better I/O devices made possible multiple job at once.

→ Jobs were stored on magnetic tape & processed continuously in sequence

→ This system was called batch processing & used a batch monitor. (basic OS)

→ Later computer introduced multiprogramming & time sharing operating system for better performance.

Third Generation computer :

→ IC's replaced transistors in computers, making them smaller, faster, cheaper.

→ Transistors were still used, but IC's combined many transistors on a small silicon chip.

IBM System/360 :

→ Software compatibility : All system/360 used same instruction set, so programs written for one model could run on others.

→ Upgrading was easy because didn't need to rewrite software when switching to a better model.

→ common OS/360 were used across all models.

→ Evolved into newer IBM mainframes.

Features of IBM System/360:

→ 200+ instructions (opcodes) for different operations.

→ 16 general-purpose registers for storing & handling data.

→ separate arithmetic logic units to process different data types.

→ 8-bit byte was set as the smallest unit of data for storage & transmission.

Control system in CPU:

→ CPU had two states:

(i) supervisor state → for the OS.

(ii) user state → for running applications

→ Some instructions were privileged & could only be used in supervisor state.

→ A special status register (SR) stored important information, like errors detected by the CPU.

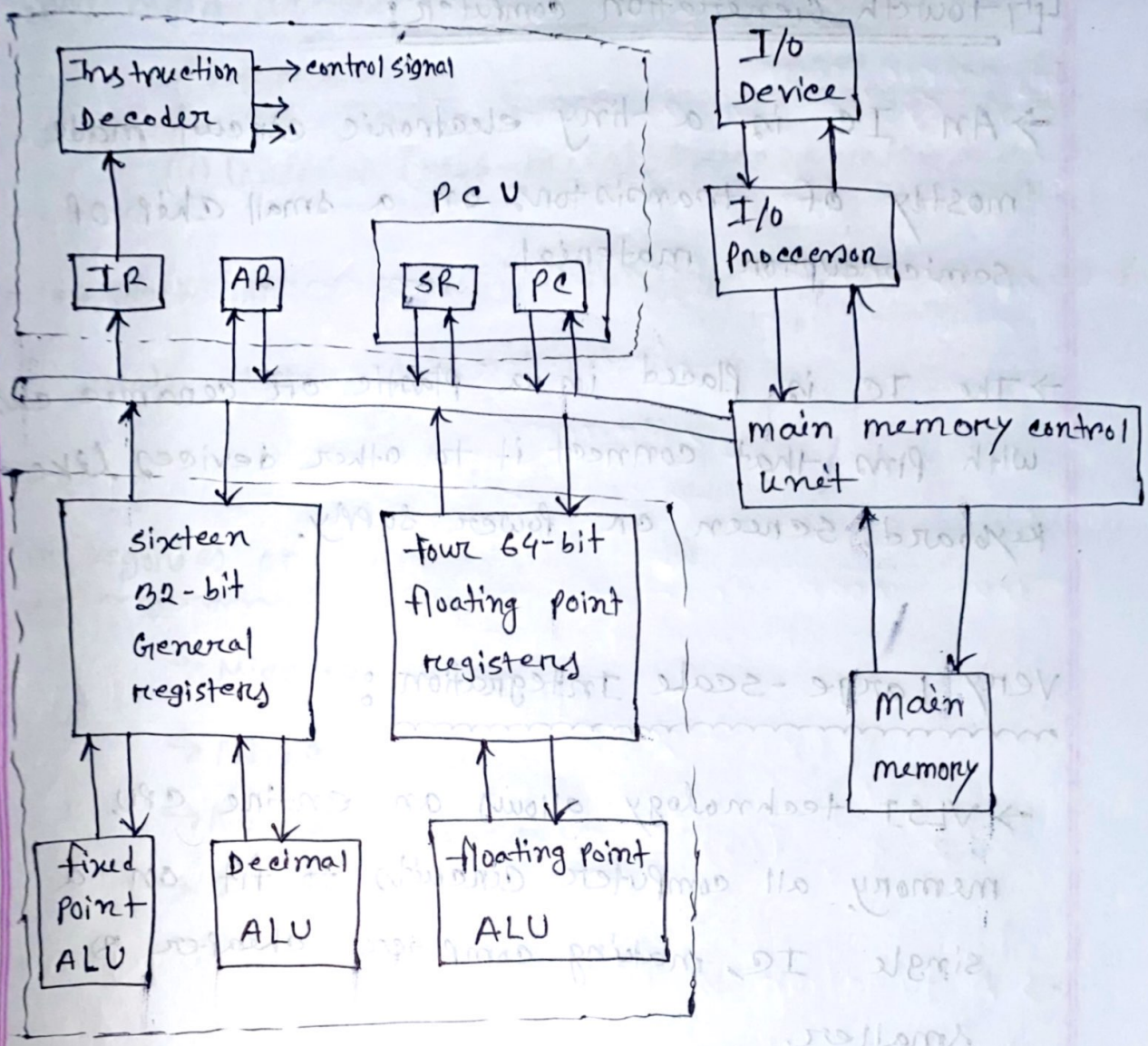


fig: structure of IBM system/360.

Fourth Generation computer:

→ An IC is a tiny electronic circuit made mostly of transistors on a small chip of semiconductor material.

→ The IC is placed in a plastic or ceramic case with pins that connect it to other devices like a keyboard, screen or power supply.

Very Large-scale Integration:

→ VLSI technology allows an entire CPU, memory, all computer circuits to fit on a single IC, making computers cheaper & smaller.

→ Multichip modules contain multiple IC chips in a single package for better performance.

→ A modern computer has IC's, I/O devices, power supply.

→ Two densest chips today are:

(i) DRAM (Dynamic Random-Access memory) - used for main memory,

(ii) Microprocessor - a CPU on a single chip.

Two main IC types:

(i) Bipolar

(ii) Unipolar (MOS - metal - oxide - semiconductor)

→ Both bipolar & MOS circuits use transistors as their main components.

Categories of computers:

→ Microcomputers.

→ Mini computers.

→ Mainframe computers.

→ Supercomputers.

Lecture 3 : Design Methodology

system Design:

→ A system is a group of connected parts that work together for a purpose. A computer is an example of a system because it has many parts, like

- → Processor
- → Memory
- → Storage

that work together to perform task.

$$f(x) = x + 2$$

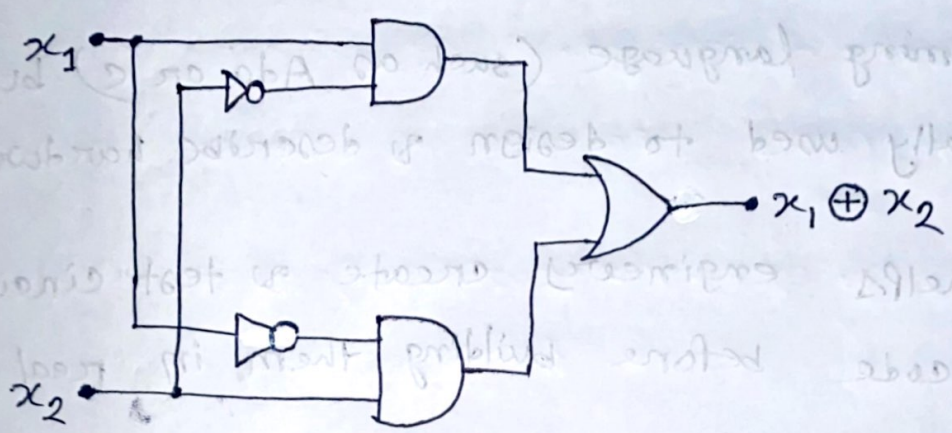
$$\therefore x = 3 ; \quad f(3) = 3 + 2 = 5$$

→ A function is like a rule that changes one thing into another.

System Design :

→ Structure : refers to how a system is built (its parts & connections)

→ Behavior : refers to how the system works (its outputs for given inputs)



Input		output
x_1	x_2	$f(x_1 \oplus x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

fig : A block diagram representing a XOR logic circuit.

XOR → Exclusive - OR

→ If both inputs are the same (0,0 or 1,1) the output is 0.

→ If the inputs are different (0,1 or 1,0) the output is 1.

This rule can also be written as mathematical equation :

$$f(x_1, x_2) = x_1 \oplus x_2$$

Hardware description language : HDL is like a Programming language (such as Ada or C) but specifically used to design & describe hardware.

→ HDL helps engineers create & test circuits using code before building them in real life.

Design Process :

→ Analysis means figuring out what a system does based on how it is built.

→ Design (or Synthesis) is the opposite – it means creating a system that performs a specific function.

Since computer are very complex, designing them is broken into smaller tasks.

Each part is designed separately & the whole system is improved step by step through repeated testing & changes (an iterative process).

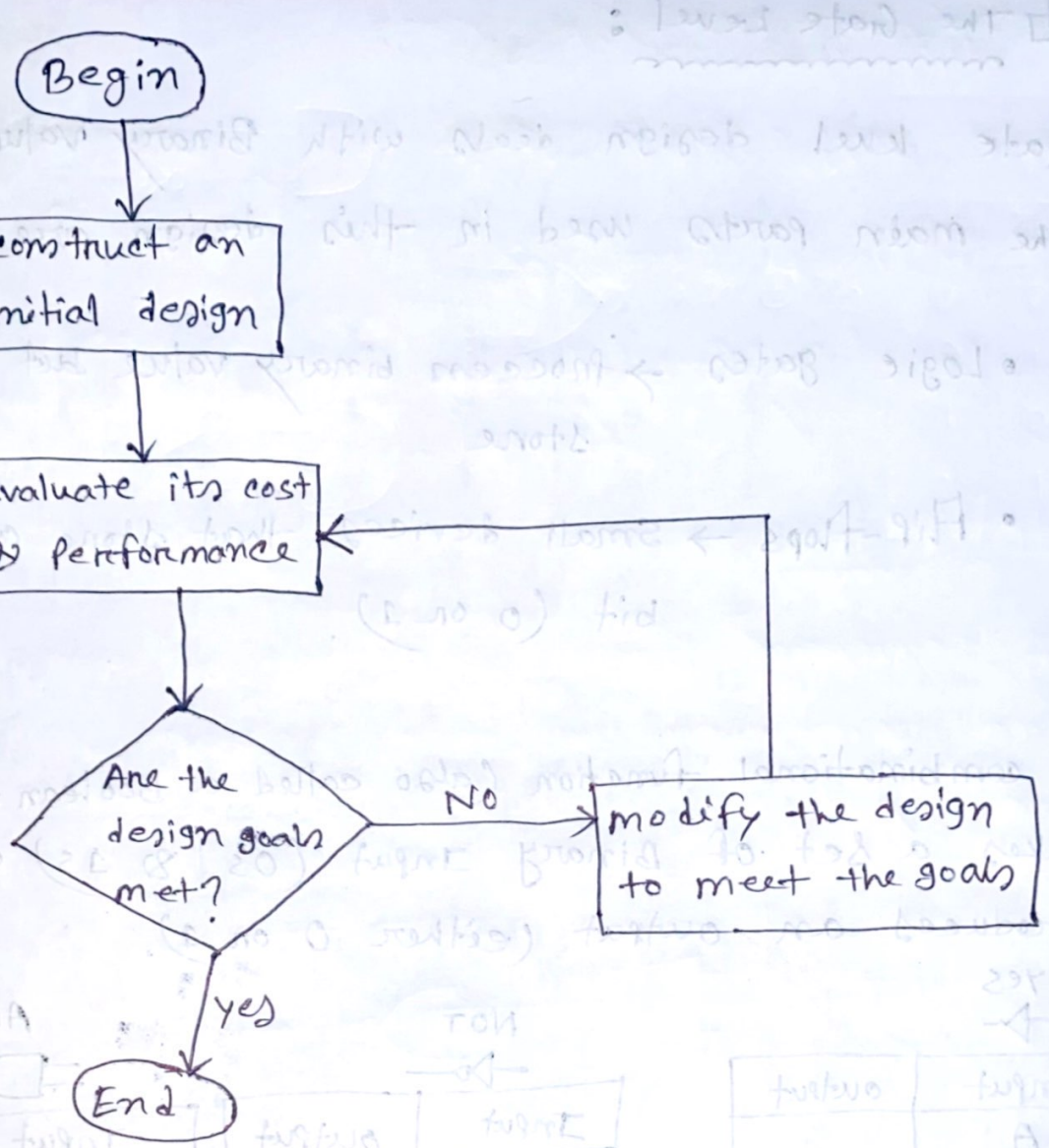


fig: flowchart of an iterative design process.

Design levels:

(i) The processor level, also called the architecture, behavior or system level.

(ii) The register level, also called the register-transfer level (RTL).

(iii) The gate level, also called the logical level.

The Gate Level :

Gate level design deals with Binary values (0 & 1)

The main parts used in this design are.

- Logic gates → Proccess binary value but doesn't store
- Flip-flops → Small devices that store a single bit (0 or 1).

A combinational function (also called a Boolean function) takes a set of Binary Input (0s & 1s) and Produces an output (either 0 or 1).

YES

Input	output
A	
0	0
1	1

NOT

Input	output
A	
0	1
1	0

AND

Input		output
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

OR

Input		output
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

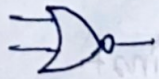
XOR

Input		output
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

NAND

Input		output
A	B	
0	0	1
1	0	1
0	1	1
1	1	0

NOR

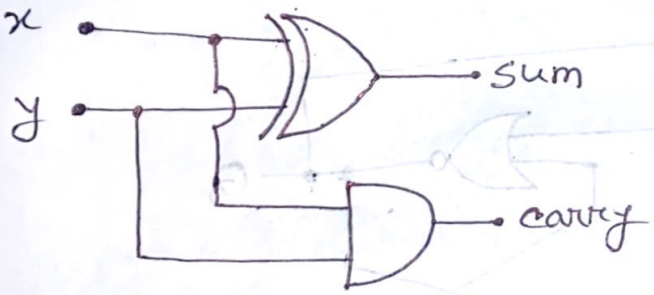


Input A	Input B	output
0	0	1
1	0	0
0	1	0
1	1	0

XNOR

Input A	Input B	output
0	0	1
1	0	0
0	1	0
1	1	1

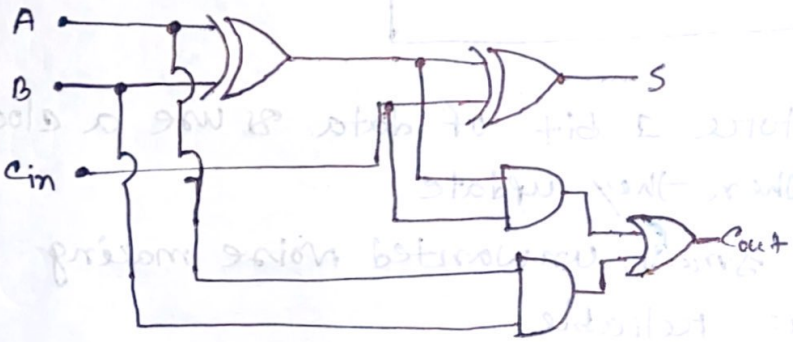
The Gate Level : Half adder



Truth Table

Input x	Input y	output sum	output carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

full-adder :



Add 3 bit

Truth Table

Input A	Input B	Input Cin	output S	output Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum of Product (SOP):

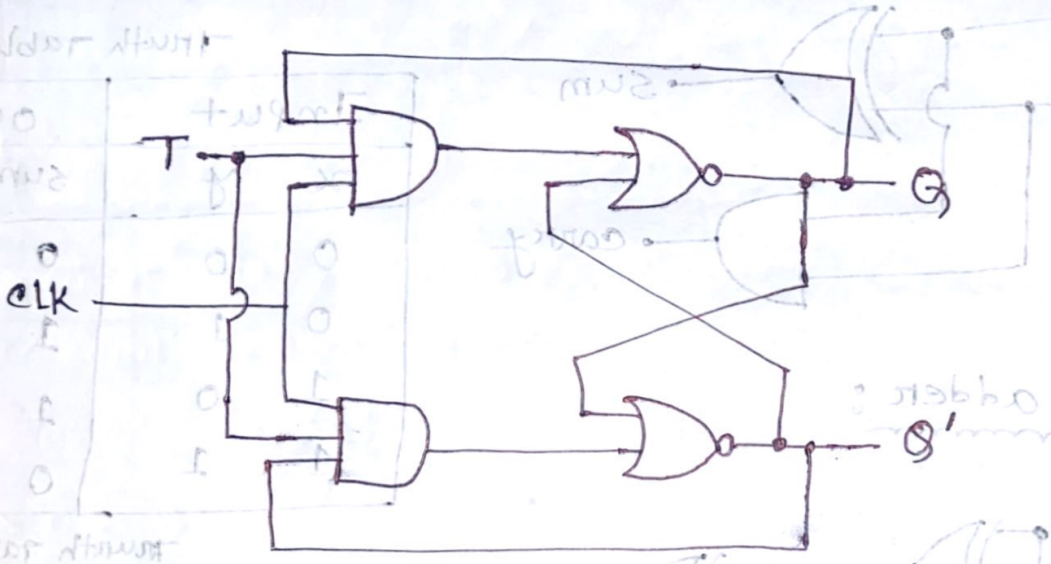
$S_0 = x_0 y_0 c_{-1} + x_0 y_0 c_{-1} + \dots$; Add first then OR

Product of sum (POS)

$c_0 = (x_0 + c_{-1})(x_0 + y_0)(y_0 + c_{-1})$; OR first then AND

Flip-flops:

A sequencial circuit is a type of circuit that can remember past input by using small memory units called flip-flops.



→ Flip-flops store 1 bit of data & use a clock signal to control when they update

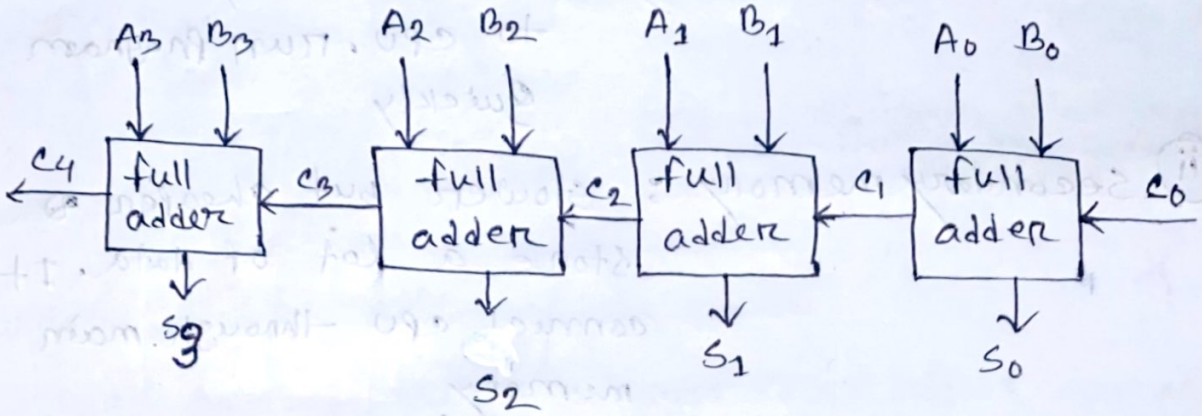
→ They ignore small, unwanted noise making circuit more reliable.

→ The circuit has two main parts:

(i) combinational logic – do calculation

(ii) flip-flops – store past info.

→ Sequencial circuit depends on both current & past input.



The Processor Level:

→ There are four main groups of Processor level components

- → Processor
- → Memories
- → I/O devices
- → Interconnection network

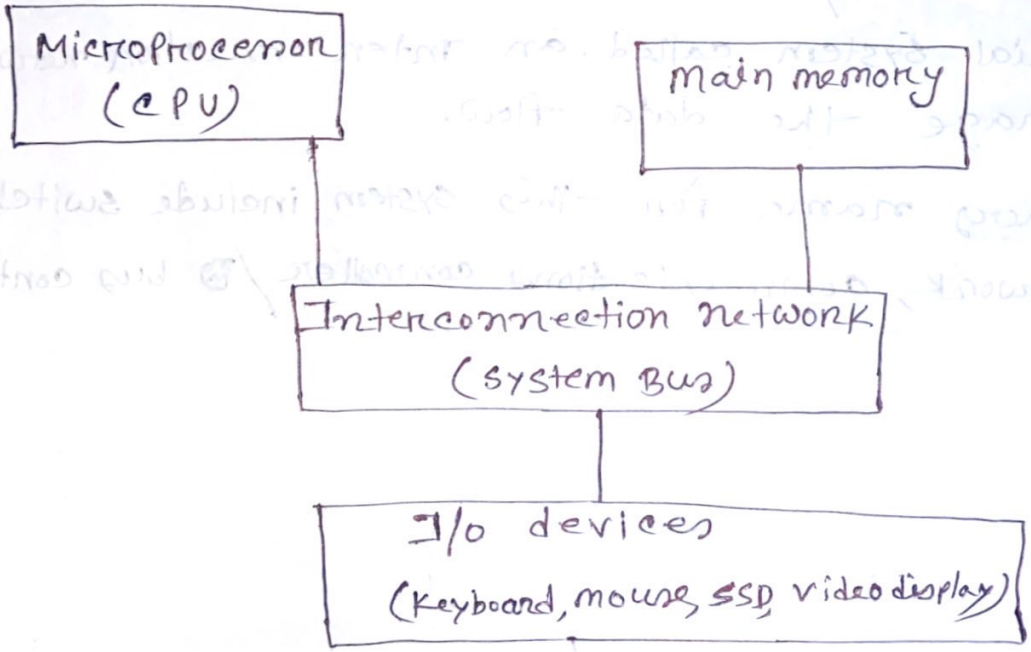


fig: mayor component of a computer system.

→ Memory divided into several major subsystem

(i) main memory (RAM) : fast, directly connected to cpu. run program quickly

(ii) Secondary memory : slower but cheaper & store a lot of data. It connect cpu through main memory.

(iii) cache memory : very fast, placed between cpu & main memory. some cache may even be inside the cpu to speed up data access.

[+] Interconnection Network :

→ When many components need to communicate, a special system called an Interconnection network manage the data flow.

→ Others name for this system include switching Network, communications controller / bus controller.

System Bus:

A bus is a set of wires that connects different parts of a computer. The number of wires is called the Bus width (wider buses can transfer more data at once)

Types of Bus lines:

(i) control Bus – manages commands & operations
(ii) Data Bus – transfer actual data
(iii) Address Bus – sends memory address to locate data.

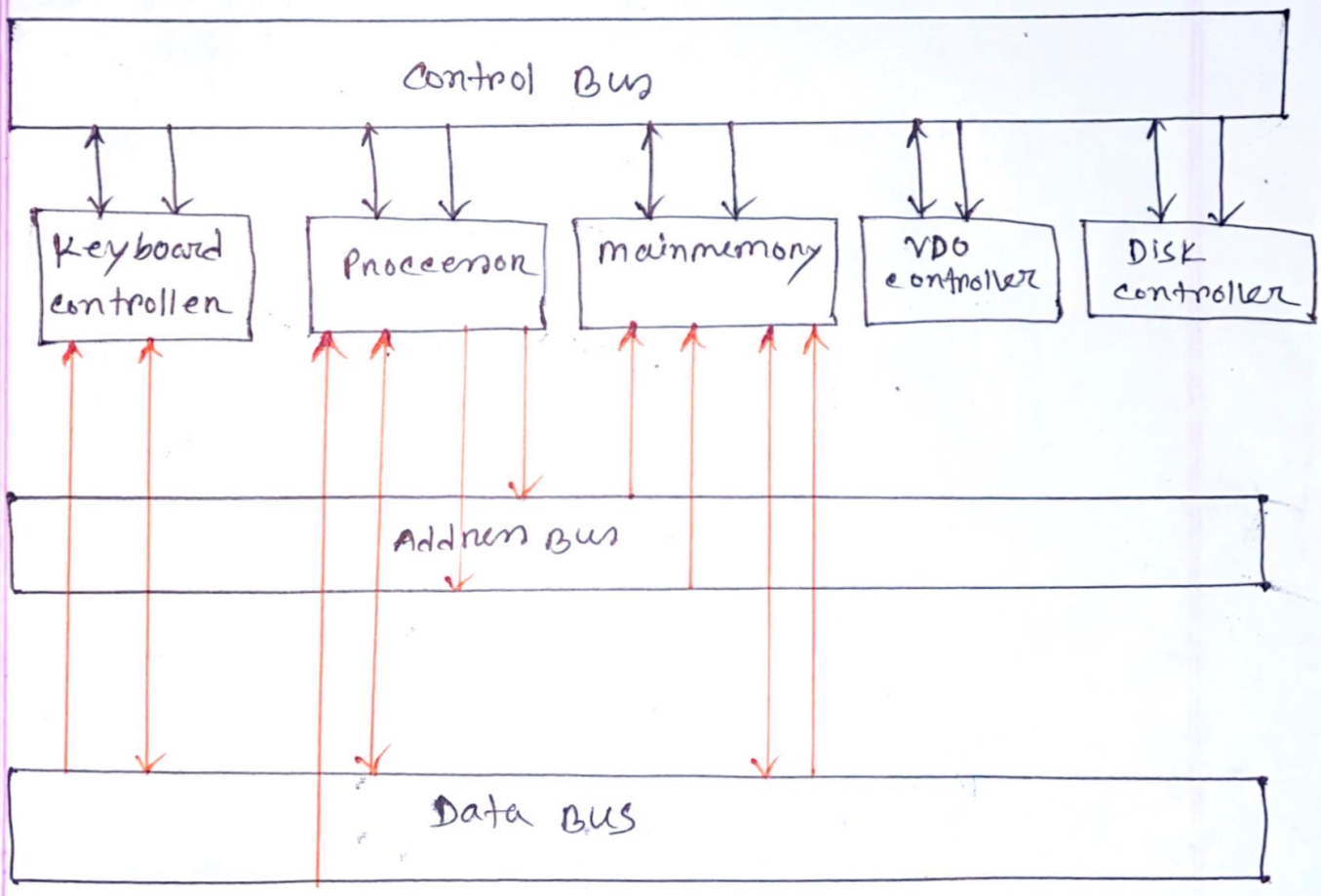
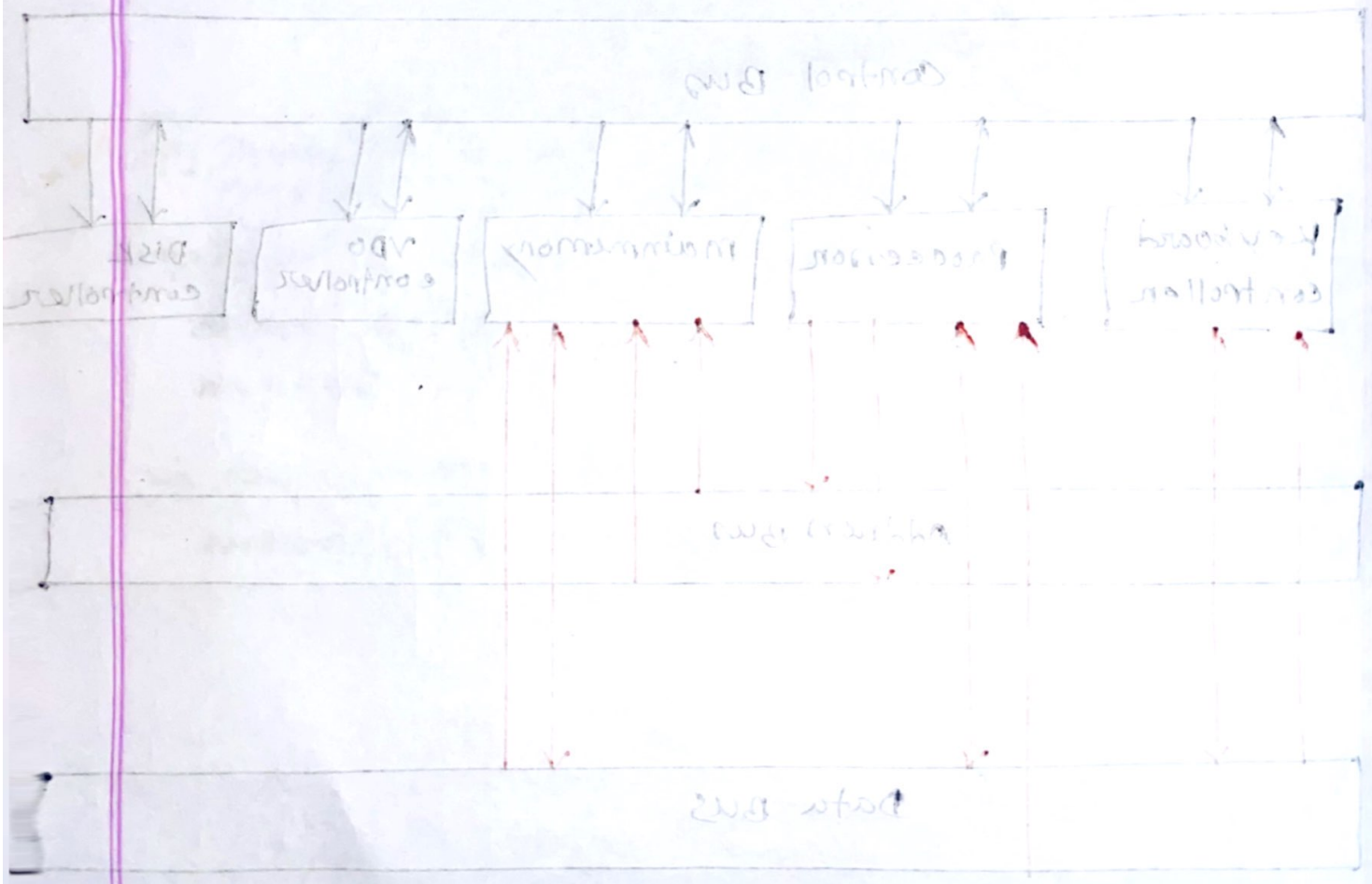


Fig: System Bus.

Interrupts: An Interrupt is a signal from some device/source seeking the attention of the Processor.

Types of Interrupts:

- → Program
- → Timer
- → Input/output
- → Hardware failure



Lecture 4: Data Representation

Binary number system

→ A positional number system

→ Has only 2 symbol or digits (0 and 1). Hence its base is 2.

→ The maximum value of a single digit is 1.

→ Each position of a digit represents a specific power of the base (2).

⇒ this number system is used in computers.

Example: $(10101)_2$

$$\Rightarrow (1\times2^4)+(0\times2^3)+(1\times2^2)+(0\times2^1)\times(1\times2^0)$$

$$= 16+0+4+0+1$$

$$= (21)_{10}$$

Bit

→ Bit stands for binary digit

→ A bit in computer terminology means either a '0' or a '1'.

→ A binary number consisting of 'n' bits is called an 'n'-bit number.

Octal number system

⇒ A positional number system

→ Has total 8 symbol or digit (0 to 7). Hence it's base = 8

→ The maximum value of a single digit is 7.

→ Since there are only 8 digits, 3 bits ($2^3 = 8$) are sufficient to represent any octal number in binary.

Example: $(2057)_8$

$= (2 \times 8)^3 + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$

$= 1024 + 0 + 40 + 7$

$= (1071)_{10}$

Hexadecimal Number system

→ A positional number system

→ Has total 16 symbols or digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (10 = A), (11 = B), (12 = C), (13 = D), (14 = E), (15 = F)}

→ Hence its base 16.

→ The maximum value of a single digit is 15.

→ It has 4 bit ($2^4 = 16$) are sufficient to represent any hexadecimal number in binary.

Example: $1AF_{16} \Rightarrow (1 \times 16^2) + (A \times 16^1) + (F \times 16^0)$

$= (1 \times 16^2) + (10 \times 16^1) + (15 \times 1)$

$= (431)_{10}$ Ans.

Decimal Number system

→ A Positional number system

→ Has 10 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
Hence, it's base = 10.

→ The maximum value of a single digit is 9. (one less than the value of the base)

→ we use this number system in our day to day life

Example: $(2586)_{10}$

$\Rightarrow (2 \times 10^3) + (5 \times 10^2) + (8 \times 10^1) + (6 \times 10^0)$

$\Rightarrow 2000 + 500 + 80 + 6$

$\Rightarrow (2586)_{10}$

Convert Another base to a Decimal Number

octal to Decimal

$(4706)_8$

$\Rightarrow 4\times8^3 + 7\times8^2 + 0\times8^1 + 6\times8^0$

$= 2048 + 448 + 0 + 6$

$= (2502)_{10}$

Binary to Decimal

$(110101.101)_2$

$\Rightarrow 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$

$\Rightarrow 32 + 16 + 0 + 8 + 0 + 1 + 1/2 + 0 + 1/8$

$\Rightarrow (53.625)_{10}$

Hexadecimal to Decimal

$(2AF)_{16}$

$\Rightarrow 2\times16^2 + A\times16^1 + F\times16^0$

$\Rightarrow 512 + (10\times16) + (15\times1)$

$\Rightarrow (687)_{10}$.

In Hexadecimal number,
A = 10
F = 15

Any base to Decimal

$(21.02)_4$

$\Rightarrow 2\times4^{1} + 1\times4^{0} + 0\times4^{-1} + 2\times4^{-2}$

$\Rightarrow 8 + 1 + 0 + \frac{1}{8}$

$\Rightarrow (9.125)_{10}$

Decimal to Another number

Decimal to Binary

$(45.65)_{10} = (?)_2$

Here,

2 | 45	— 1	MSB
2 | 22	— 0	
2 | 11	— 1	
2 | 5	— 1	
2 | 2	— 0	
2 | 1	— 1	LSB
0		

0.65	0.30	0.60	0.20
× 2	× 2	× 2	× 2
1.30	0.60	1.20	0.4

The Ans. is $(101101.1010....)$

Decimal to octal

$(952.65)_{10} = (?)_8$

8 | 952		
8 | 119	— 0	↑
8 | 14	— 7	↑
8 | 1	— 6	↑
0	— 1	↑

0.65	0.20	0.60	0.80
× 8	× 8	× 8	× 8
5.20	1.60	4.80	6.40
→	→	→	→

The Answer is $(1670.5146...)_8$

Decimal to Hexadecimal

$(95.87)_{10} = (?)_{16}$

16 | 95		
16 | 5	— 15 (F)	↑
0	— 5	↑

0.87	0.92	0.72	0.52
× 16	× 16	× 16	× 16
13.92	14.72	11.52	8.32
D	E	B	
→	→	→	→

The Answer is $(5F.DEB8....)_{16}$

Decimal to Anybase

$(45.25)_{10} = (?)_6$

6 | 45		
6 | 7	— 3	↑
6 | 1	— 1	↑
0	— 1	↑

0.25	0.50
× 6	× 6
1.50	3.00
→	→

The Answer is (113.13).

0 →	0 0 0 →	Add of 0
1 →	0 0 1 →	Add of (0 and 1)
2 →	0 1 0 →	Add of two 1
3 →	0 1 1 →	Add of three 1
4 →	1 0 0 →	" " four 1
5 →	1 0 1 →	" " five 1
6 →	1 1 0 →	" " six 1
7 →	1 1 1 →	" " seven 1
8 →	1 0 0 0 →	" " Eight 1
9 →	1 0 0 1 →	" " nine 1
10 = A →	1 0 1 0 →	" " ten 1
11 = B →	1 0 1 1 →	" " eleven 1
12 = C →	1 1 0 0 →	" " twelve 1
13 = D →	1 1 0 1 →	" " thirteen 1
14 = E →	1 1 1 0 →	" " fourteen 1
15 = F →	1 1 1 1 →	" " fifthteen 1

Binary to octal (short: 421)

$(1\,1\,1\,1\,0\,1\,0\,1 \cdot 1\,0\,1\,1)_2 = (?)_8$

$= \underset{421}{011}\ \underset{421}{110}\ \underset{421}{101} \cdot \underset{421}{101}\ \underset{421}{100}$

$= (3\ 6\ 5 \cdot 5\ 4)_8$

(Ans.)

0 — 0
1 — 101
2 — 010
3 — 011
4 — 100
5 — 101
6 — 110
7 — 111

Binary to Hexadecimal (short : 8421)

$(110,110\ 1,0\ 110 \cdot 11)_2 = (?)_{16}$

$\Rightarrow \underset{8421}{0110}\ \underset{8421}{1101}\ \underset{8421}{0110} \cdot \underset{8421}{1100}$

$\Rightarrow (6 \quad \underline{13} \quad 6 \cdot \underline{12})$

$\Rightarrow (6D6.C)_{16}$

Any base to Any base

$(545)_6 = (?)_4$

Step 01: convert from base 6 to base 10.

$(545)_6$

$= (5 \times 6^2) + (4 \times 6^1) \times (5 \times 6^0)$

$= 180 + 24 + 5$

$= (209)_{10}$

Step 2: convert base 10 to base 4 /
Decimal to Anybase.

4 | 209		
4 | 52	— 1	↑
4 | 13	— 0	↑
4 | 3	— 1	↑
0	— 3	↑

Here,
$(209)_{10} = (3101)_4$
So,
$(545)_6 = (209)_{10} = (3101)_4$
thus $545 = (3101)_4$ (Ans)

Octal to Binary

$(6\ 3\ 2\ 5\ .\ 4)_8 = (?)_2$

110 011 010 101 100

$\Rightarrow (110011010101.100)_2$ (Ans.)

Octal to Hexadecimal

$(6\ 3\ 2\ 5\ .\ 4\ 1)_8 = (?)_{16}$

110 011 010 101 100 001 **Step 01** → octal to binary

$\Rightarrow \underset{8421}{1100}\ \underset{8421}{1101}\ \underset{8421}{0101} \cdot \underset{8421}{1000}\ \underset{8421}{0100}$

Step 02 → Binary to Hexadecimal

$\Rightarrow (12\ 13\ 5\ .\ 8\ 4)$

$\Rightarrow (C\ D\ 5\ .\ 8\ 4)_{16}$ (Ans.)

Hexadecimal to Binary

$(F\ 6\ D\ 4\ 2\ .\ E\ 3\ C)_{16} = (?)_2$

1111 0110 1101 0100 0010 1110 0011 1100

$\Rightarrow (1111011011010100001011100011100)_2$ (Ans.)

Hexadecimal to Octal

$(F\ 6\ D \cdot E)_{16} = (?)_8$

1111 0110 1101 1110

$= 1111\ 0110\ 1101 \cdot 1110$ [Step 01: Hexadecimal to Binary]

$= \underset{421}{111}\ \underset{421}{101}\ \underset{421}{101}\ \underset{421}{101} \cdot \underset{421}{111}\ \underset{421}{000}$ [Step 02: Binary to Octal]

$= (7555.7)_8$ (Ans.)

Computer Arithmetic

Binary state

0 → off

1 → on

Binary number system

→ System Digits : 0 and 1

→ Bit (short for binary digit) : A single binary digit

→ LSB (Least significant bit) : The rightmost bit.

→ MSB (Most significant bit) : The leftmost bit.

→ Upper Byte (or nybble) : The right-hand byte (or nybble) of a pair.

→ Lower Byte (or nybble) : The left-hand byte (or nybble) of a pair.

→ The term nibble used for 4 bits being a subset of byte.

Binary Equivalents

→ 1 Nybble (or nibble) = 4 bits

→ 1 Byte = 2 nybbles = 8 bits

→ 1 Kilobyte (Kb) = 1024 bytes

→ 1 megabyte (MB) = 1024 Kilobytes = 1048576 bytes = $(1024)^2$

→ 1 Gigabyte (GB) = 1024 megabytes = 1073741824 bytes
= $(1024)^3$

Binary Addition

Rule:

① 0 + 0 = 0

② 0 + 1 = 1

③ 1 + 0 = 1

④ 1 + 1 = 0 plus a carry of 1 to next higher column.

Example:

```
      1 1
  0 0 0 1 1 0 1 0
(+) 0 0 0 0 1 1 0 0
  ---------------
  0 0 1 0 0 1 1 0
```

(Ans.)

Example:

```
    1 1 1 1 1
  0 0 0 1 0 0 1 1
(+) 0 0 1 1 1 1 1 0
  ---------------
  0 1 0 1 0 0 0 1
```

(Ans.)

Complementary method

Date

Rule: $C = B^n - 1 - N$

- C → complement of the number
- B → base of the number
- n → number of digit
- N → the given number

Example: find 37_{10} complement value.

$\Rightarrow C = B^n - 1 - N$

$\Rightarrow 37_{10} = (10^2 - 1 - 37)$

$= 62$ (Ans.)

⊛ complementary of Binary number / 1's complement :

1 0 1 1 0 0 1 0

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

0 1 0 0 1 1 0 1 → 1's complement

Example: Subtract 56_{10} from 92_{10} using complementary method

Step: 1

complement of 56_{10}

$\Rightarrow 10^2 - 1 - 56$

$\Rightarrow 43_{10}$

Step-2:

$92 + 43$

$= 135$

↳ carry bit

Step 3:

$35 + 1$

$= 36$

(Ans.)

Binary subtract with complementary method

Ex: 0111000_2 from 1011100_2

Soln:

$0\ 1\ 1\ 1\ 0\ 0\ 0_2$

$\downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow$

$1\ 0\ 0\ 0\ 1\ 1\ 1 \rightarrow$ 1's complement

Here,

1011100

$(+)\ 1000111$

10100011

1 → carry bit

Now,

0100011

$(+)\quad 1$

0100100 (Ans.)

Ex: subtract 100011_2 from 010010_2 using complementary method.

100011

$\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow$

$011100 \rightarrow$ 1's complement

Here, 010010

$(+)011100$

101110

There is no carry bit, we have to complement the sum and attach a negative sign to it.

Result: -010001_2 (complement of 101110)

Binary addition with complementary method

Example: $-22+13 = (?)$

Soln:

	128	64	32	16	8	4	2	1	
22 =	0	0	0	1	0	1	1	0	(Binary value)
13 =	0	0	0	0	1	1	0	1	(Binary value)

(22) 1's complement is

```
0 0 0 1 0 1 1 0
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
1 1 1 0 1 0 0 1
              + 1
-------------------
(-22) = 1 1 1 0 1 0 1 0   → 2's complement of (22)
```

[Work alway take with 4/8 bit]

```
-22 = 1 1 1 0 1 0 1 0
 13 = 0 0 0 0 1 1 0 1
(+)
-------------------
 -9 = 1 1 1 1 0 1 1 1
      ↳ sign bit carry 0.
```

sign
1 = (-ve)
0 = (+ve)

∴ Answer is $(-1\,1\,1\,1\,0\,1\,1\,1)$.

Example: $-22 + (-13) = (?)$

Soln:

	128	64	32	16	8	4	2	1	
22 =	0	0	0	1	0	1	1	0	(Binary value)
13 =	0	0	0	0	1	1	0	1	(Binary value)

Here,

$22 = 00010110$

1's comp. $= 11101001$

$+ 1$

$-22 = 11101010$ (2's complement)

Again,

$13 = 00001101$

1's comp. $= 11110010$

$+ 1$

$-13 = 11110011$ (2's complement)

$-22 = 11101010$

$-13 = 11110011$

(+)

$-35 = 111011101$

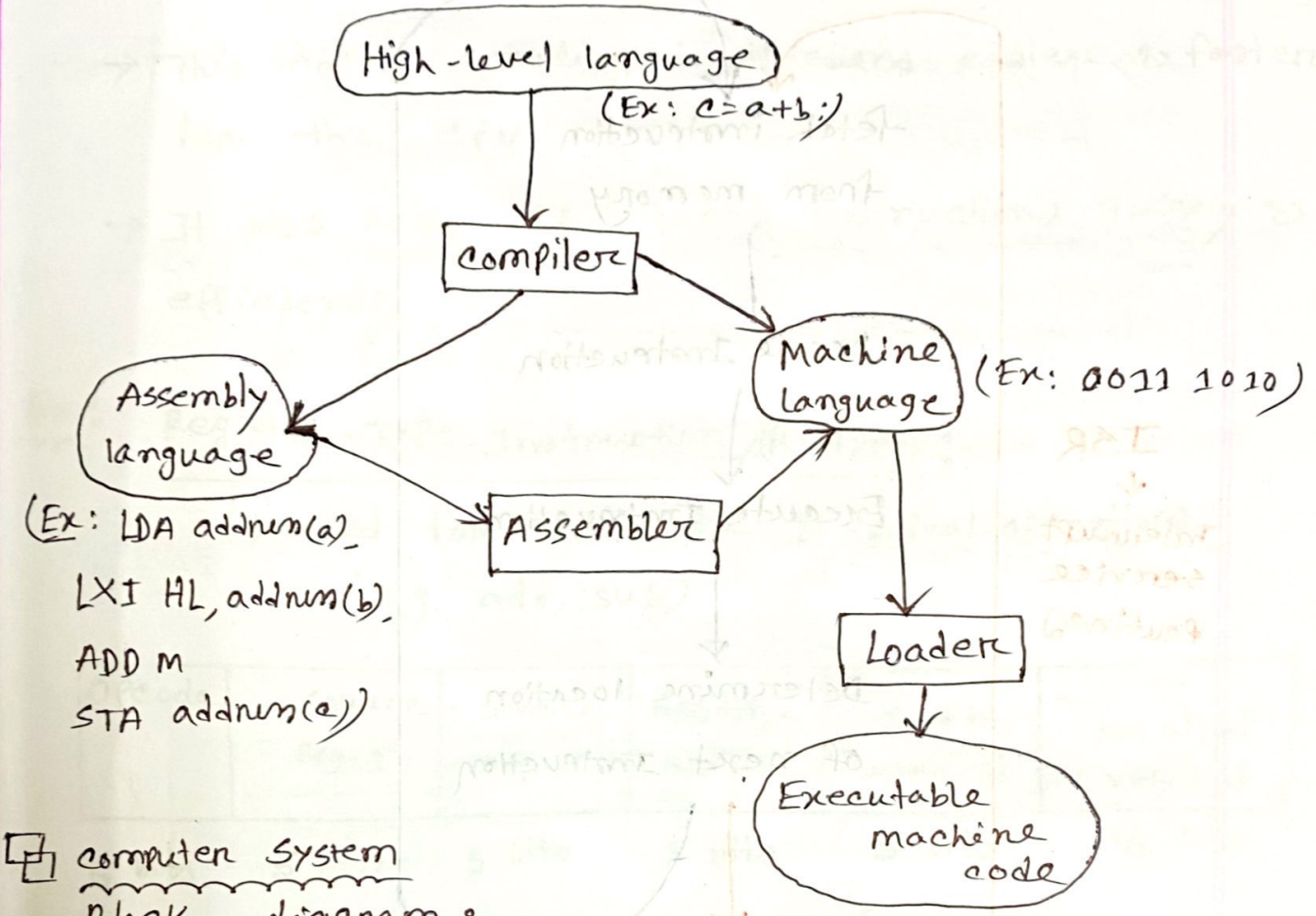
1 — carry bit

1 — sign bit

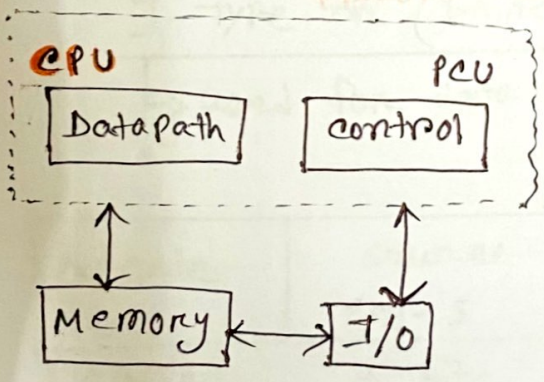
Answer is $-35 = (-11011101)$.

Lecture - 5 : Processor Basics
(Pipelining, Instruction set design)

Computer Program Execution :



Computer System Block diagram :



CPU : central processing unit

- **Datapath** : arithmatic operation ALU component, work as data processing unit.
- **control** : means program control unit. manage the flow of data & instruction, (fetch, decode, Execute)

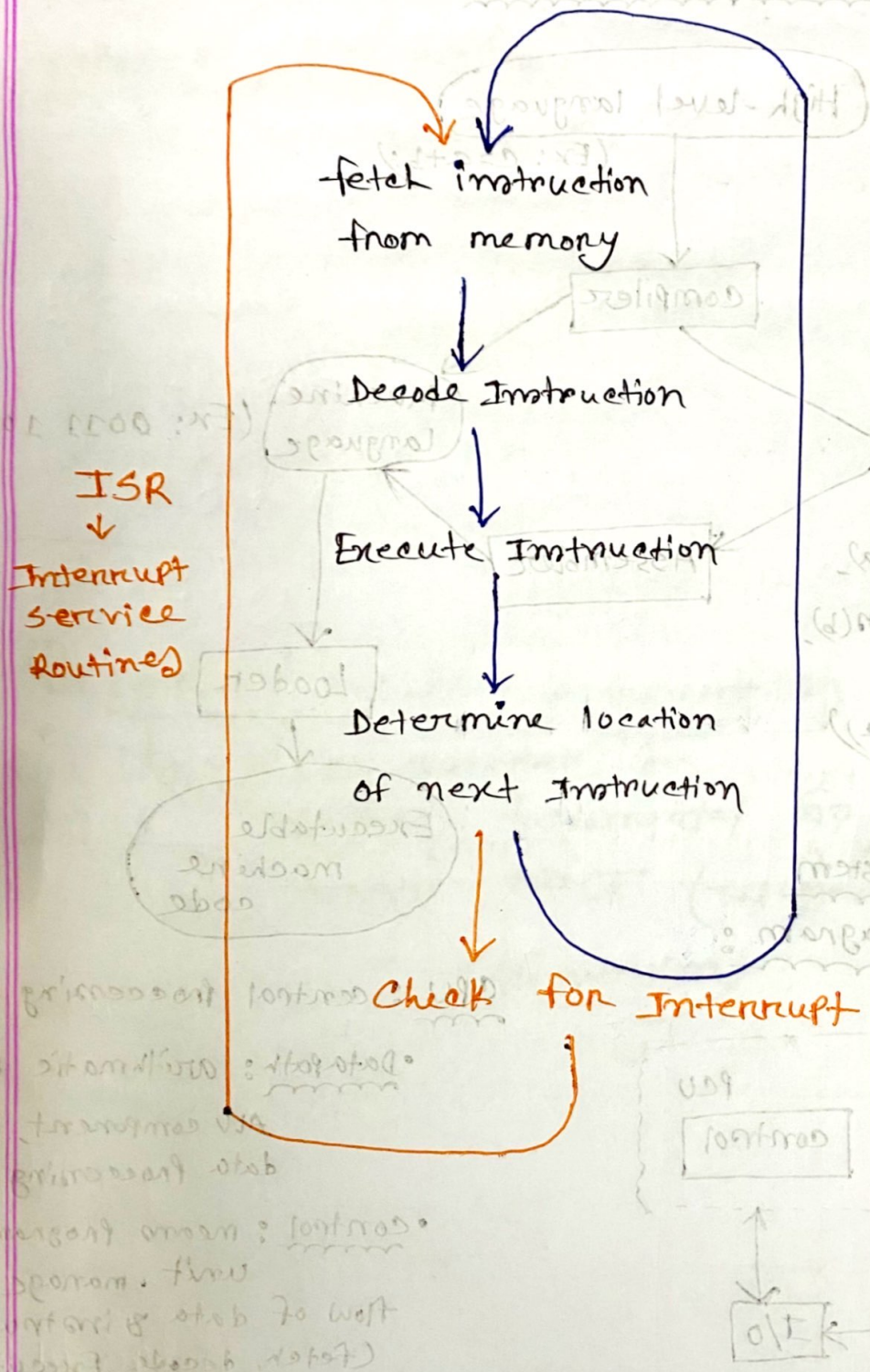
memory : store data & instruction from CPU.

I/O : communicate with external devices.

Basic operation of a computer system:

→ Also called Instruction cycle

→ describes how cpu processes instructions.



Instruction set design :

(i) fixed-length Instruction format :

→ All instructions use the same number of bits. (32 bit) MIPS

→ This makes decoding instructions easier & faster for the CPU

→ It also helps CPU fetch instructions quickly & efficiently.

Ex: Register-Type Instruction (R-Type) :

↳ used for arithmetic & logical operations (e.g. add, sub)

Opcode	Source Reg-1	Source Reg-2	destin. register	Shift ammount	function variant
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Type or (Immediate-Type) Instructions :

↳ used for data transfer & immediate value operation

Opcode	source Reg-1	source Reg-2	address
6 bits	5 bits	5 bits	16 bits

This two example for MIPS (million of instruction per second) microprocessor. (32 bit)

(ii) Variable-Length Instruction format :

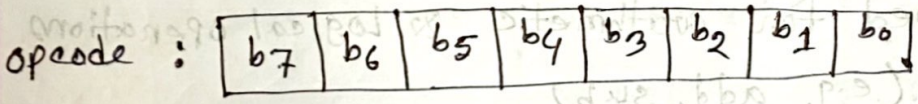
→ Different instructions use different num. of bits.

→ Short instructions save memory, while long instructions can handle more complex tasks.

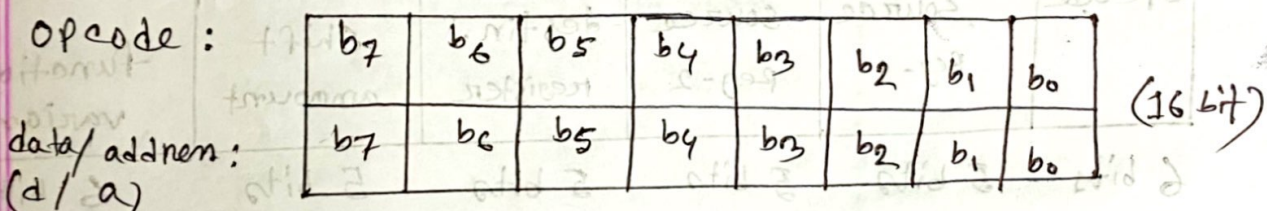
→ It's harder for the CPU to fetch & decode these instructions quickly.

Ex: variable length instruction formats used in the Intel 8080 architecture.

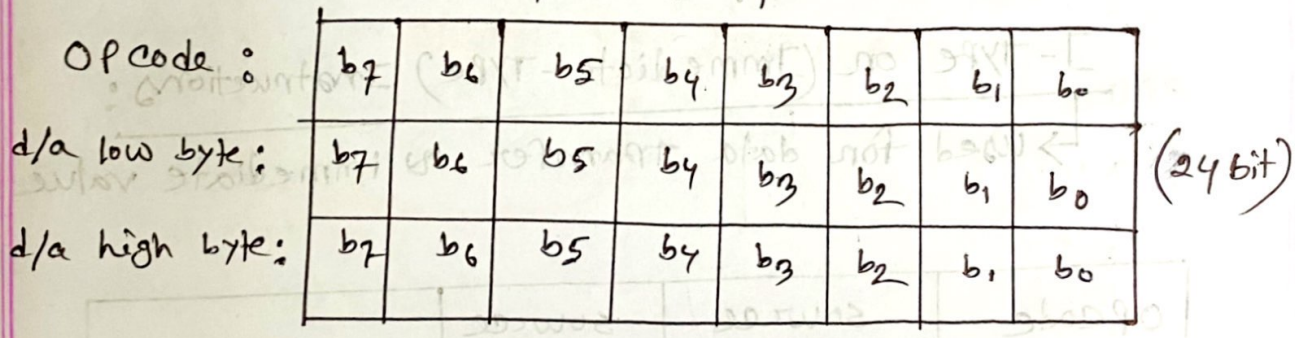
One-Byte format



Two-Byte format



Three byte format



Opcode: part of machine instruction that tells the CPU what operation to perform.

Number of Data Operands :

Zero-Operand Instructions :

→ No data is included in the instruction

→ data is accessed from the stack (LIFO : Last-In-First-Out)

One-Operand Instruction :

→ One piece of data is Provided

→ The CPU uses a special register called the Accumulator (Acc) as the second data input & where the result is stored

Two-Operand Instruction :

→ Two pieces of data are Provided

→ The result is stored in one of the data locations.

Three-Operand Instruction :

→ Three pieces of data are Provided

→ The result is stored in a separate location.

Ex: $Z \leftarrow x + y$

0-Operand:

```
Push Rx
Push Ry
add
Pop Rz
```

1-Operand:

```
load Acc, Rx
add Ry
move Rz, Acc
```

2-Operand:

```
add Rx, Ry
move Rz, Rx
```

3-Operand:

```
add Rz, Rx, Ry
```

Ex: $x \leftarrow 5 + (3/2 * 7)$

→ (follows standard of operator precedence)

0-Operand:

```
Push 5
Push 3
Push 2
div (top two value of stack)
Push 7
mul
add
Pop Rx
```

1-operand

```
load Acc, 3
div Acc, 2
mul Acc, 7
Add Acc, 5
move Rx, Acc
```

2 - operand :

```
move  Rx, 3
div   Rx, 2
mul   Rx, 7
add   Rx, 5
move  Ry, Rx
```

3 - operand :

```
div   Rz, 3, 2
mul   Rz, Rz, 7
add   Rx, Rz, 5
```

Endian Mode :

Big Endian :

→ Bytes are order from left to Right

→ MSB is stored at lowest address

→ Use Network Protocol TCP/IP

0 byte	0	1	2	3	word 0
4 byte	4	5	6	7	word 1
8 byte	8	9	10	11	word 2
12 byte	12	13	14	15	word 3

Ex: Instruction :

Store - 12345678H, ABCDE0H

Address	Data
000000H	0H
⋮	⋮
ABCDE0H	12H
ABCDE1H	34H
ABCDE2H	56H
ABCDE3H	78H
⋮	⋮
FFFFFFFH	

Note : The address of every word is the address of starting byte.

Little Endian :

→ Bytes are stored from lest most significant (Reverse order)

→ used by most mondern Proceessors (Intel x86)

Ex :

বিপরীত অর্ডার হলে সমান 0 add করে নিবো।

Store 12345678H, ABCDEOH

Adress	Data
0H	78H
1H	56H
2H	34H
3H	12H

Mordern CPUs :

Mordern CPU speed up techniques :

(i) cache memories : small, fast memory to store frequently used data.

(ii) Special purpose instructions : designed for specific task

(iii) Redundant Hardware : using extra hardware for faster execution

(iv) Support for Parallel Processing : Allow multiple processes or instructions to run at the same time.

(v) Pipeline : Breaks down tasks into smaller steps so the CPU can Process different steps at the same time.

Two types of CPU Architecture :

(i) RISC (Reduced Instruction set computer) :
→ uses simple instructions that execute quickly

(ii) CISC (complex Instruction set computer) :
→ uses complex instructions to do more work with fewer instructions.

Ways of modern CPU improve performance:

(i) Multipurpose register

(ii) Additional data & Instruction types

(iii) Status Registers

(iv) Program control stack

PipeLine:

→ A Pipeline in a CPU is a process where different steps of multiple instructions are handled at the same time.

two stage:

(i) Fetch stage: Program control unit (PCU) gets instruction from memory.

(ii) Execute stage: Data Processing unit (DPU) performs the instructions.

→ Pipeline is important for fast complete CPU tasks.

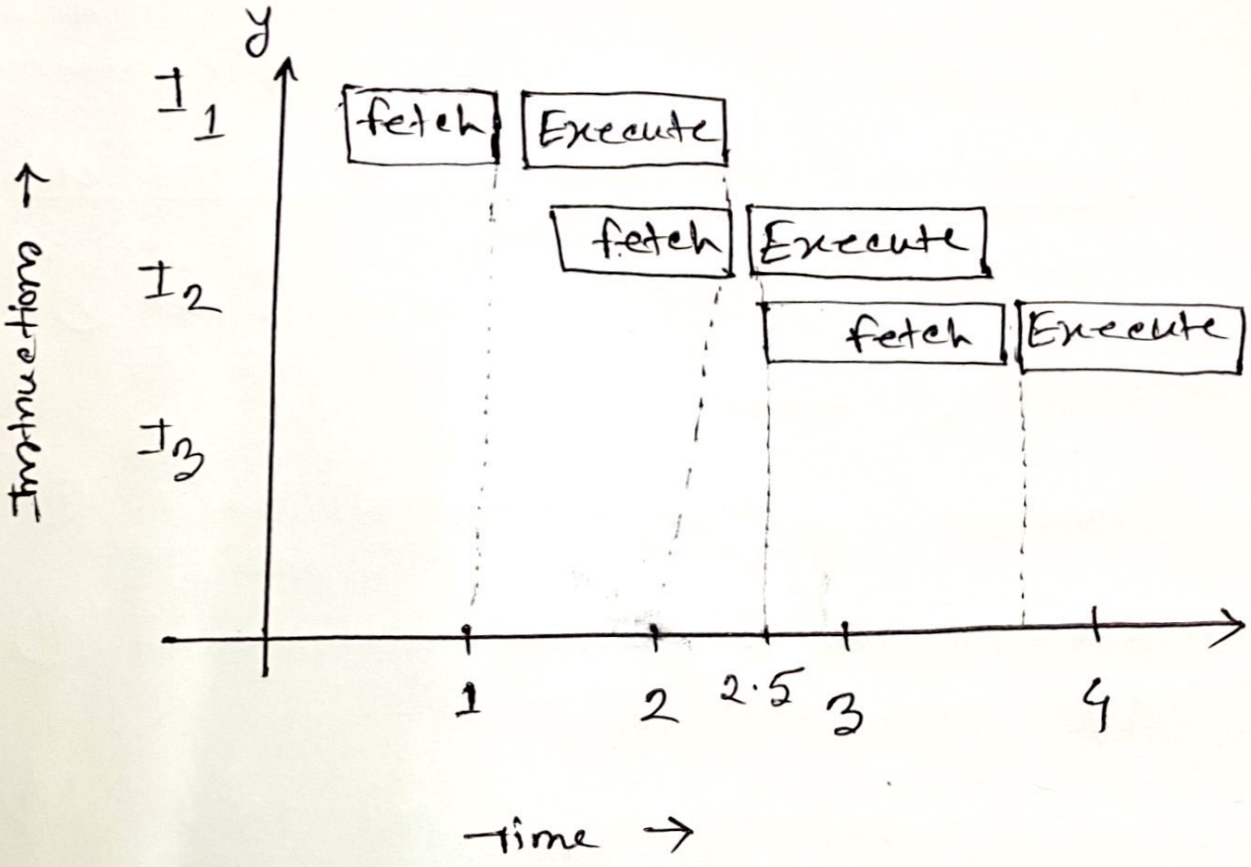


fig: Instruction pipeline.