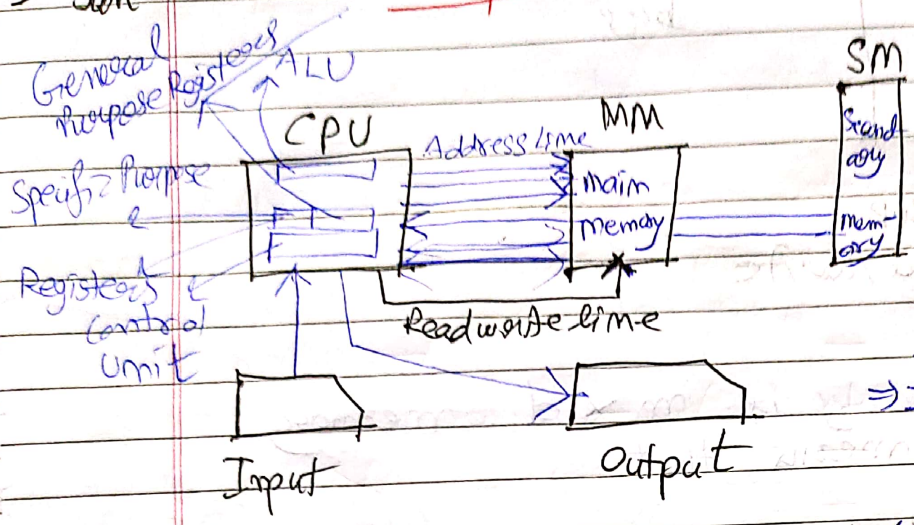
Saathi

3/1/23 Tue Jan

Date __/__/__

Computer Architecture



Block Diagram of Machine Computer

Information Processors
⇒ Input, output are both information.

CPU site for all processing / information processing
CPU interfaced with MM
CPU has address lines, will generate address.
CPU will float address.
MM should also have address line.
Address line → Output of CPU
→ Input of MM

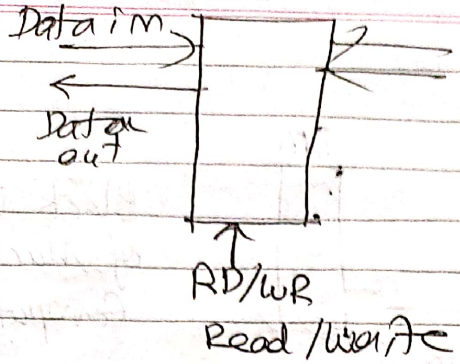
Address bus → Output of CPU
↓
Collection of address lines

Data bus → Input to CPU, output of CPU, both output of MM, Input of MM, both
↓
Generally bidirectional
↓
OR have data in and data out lines

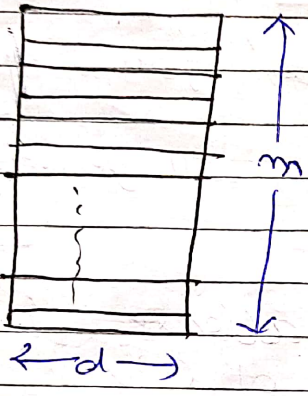
For each CPU have 1 read write signal line or 1 read & 1 write line to identify which operation
∴ Input / Instruction to MM.

Saathi

Date ___/___/___



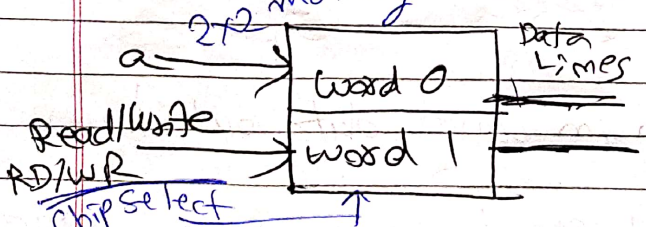
Any memory is m x d memory
It means that



d = Word size
d = 8 ; in 1 byte word size i.e., 8 bits
m = No. of words

8 GB → Byte ⇒ d = 8
8G ⇒ m = 8 × 2^{30}

m=1, d=1 1 word of size 1 bit
m=1, d=2 1 word of size 2 bit
m=2, d=1 2 words of 1 bit size
m=2, d=2 2 words of size 2 bit



a=0 → word 0; a=1 → word 1

$m \leq 2^n$ where n = No. of address lines
No. of data lines = d
RD/WR → 0 ⇒ write, 1 ⇒ Read
Chip Select → At 0 enabled, 1 disabled chip

9

Saathi

74367 → Tristate Buffer

3

Saathi

Date __/__/____

Tristate ≡ Cutoff

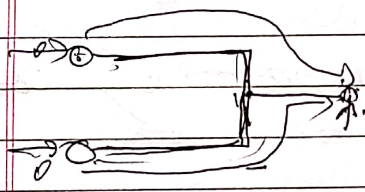
When want to receive address/data get it, otherwise not receive → At tristate

First write, then read

74367

$\overline{G_1}$	1A	1Y
0	0	0
0	1	1
1	X	~~High~~ ~~St~~ Tristate (No light flickered)

High Impedance Z



1 $\overline{G_1}$ → ~~Yellow~~ Yellow
2 1A → Blue
15 $\overline{G_2}$ → Black
14 6A → Red

$\overline{G_1}$	1A	$\overline{G_2}$	6A	Output
0	0	0	0	0
0	1	0	1	1
0	0	1	X	0
0	1	1	X	1
1	X	0	0	0
1	X	0	1	1

Page No.

Apart from CPU also there are other processors

CPU is useless without memory directly interfaced/
Memory attached to CPU is main/ primary memory.

If X access main memory, CPU X process.

Information
Instructions — Data

CPU follow instructions to process data

Main memory is comparatively costly but comparatively faster

Costly ⇒ There exist cheaper memory with similar processing.

Any memory has an access time

If faster ⇒ Access time is less.

Main memory is chosen to be comparatively less volume.

Secondary memory is less costly, more slower than main memory.
∴ Cost ↓ ⇒ ↑ Access time
↳ ↑ Volume

Date __/__/____

Transparent → ✗ know what is behind.
↓ ✗ go through it.
Everybody
using computer
✗ know how
much main memory, secondary memory
volume (apart from buyer).

Total cost of memory → Should be min.

$= C_m \times V_m + C_s \times V_s$ → Should be min.

C_m → Cost per unit vol. of main memory
V_m → Volume of main memory
C_s →
V_s →

Appears as if
~~Total~~ vol. of memory $\approx V_m$
~~Total~~ cost/unit vol. $\approx C_s$
∴ Total cost = $C_s V_m$

Computer useless without input, output devices.

CPU:

ALU → Where all arithmetic & logical operations done

Control Unit →
Major part of CPU, controls sequence of info access from, to memory, when what operation should be done
ALU receives instruction from here

Page No.

Saathi

Date __/__/____

Computation speed should meet demand

↓ Time for computation → Make CPU more complex

We are now in 5nm technology
~~i.e., effect~~ ↓
↑ fast, ↓ space

Technological development

Change computer architecture

↓

To get faster computation

Input output devices are much slower as compared to CPU.

Memory also slows down CPU
Does not mean memory is slow

i: Outside chip communication
Any outside chip communication is slow

5/1/23
Thu
Term

IAS → Immediate Access Stool → 1st comp. architecture

Saathi

Date / /

ENIAC → 1st electronic computer

→ Electronic Numerical Integrator and Computer
→ 1940s launched
→ X able to store program
→ Size: 30×50 sq. feet or 80×50 sq. feet
→ 200 kW
→ Not run more than 2 hrs.
→ Computation power: 5000 addition per second / simple operations per second
→ 18000 vacuum tubes (approx.)
→ weight 30 tonnes.

EDVAC

→ 1st comp to store program
→ Electronic Discrete Variable and computer
→

Need to realise floating points & their operations which are more complex than integer operations / arithmetic. And to also realise millions / billions per second)

→ This occurs due to change in machine comp. language, comp. architecture

5 generations of computers

1st Generation → Processors designed with vacuum tubes

~~Weight 30 tonnes~~

1 vacuum tube ≈ 1 transistor

11400 crore transistors in today's processor / CPU's processor

Page No.

Saathi

Date __/__/____

1st Generation

1st Generation → Switches were relay type

→ Memory is very limited
Mercury delay line memory

→ Primitive i/p, o/p devices

Punch cards / Machines → Used to give instructions to program

↑ Size than piano.

→ Programming Language → Machine lang., assembly lang.
↓
⇒ only binary

2nd Generation

1950s – 1965 → 2nd generation machine comp. dominated

Major change

↓
Transistors were introduced (invented in late 1940s) & then used in machine comp.'s instead of vacuum tubes.

→ Memory → Mercury delay is replaced by Drum,
~~Drum~~

→ I/p, O/p is ↑ sophisticated
keyboard introduced

→ Fortran, Cobol introduced as programming language.

2nd Generation

→ Batch Processing → Execution of programs in a sequence
→ Jobs placed in batches

3rd Generation

3rd Generation
→ 1960 – 1970
→ IC technology was introduced (Major Change)
→ SSI, MSI were in place
→ Semiconductor memory introduced
→ Pipelining & computation introduced to improve speed of machine
→ Cache also introduced
→ High level language
→ C also introduced

→ Multiprogramming, Time sharing, Parallel Processing introduced

Multiprogramming → Cover in OS

Many processes occur when run 1 program → Some waste processes, other user programs, system programs/processes

↓

↑ Than 1 program simultaneously stay in system

↓

Multiprogramming

Contr
CKT

Date ___/___/___

Saathi

Time Sharing

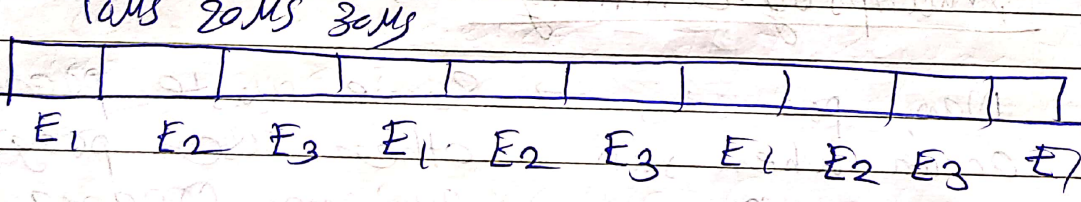
Consider 3 programs simultaneously in sys - E_1, E_2, E_3 (say)

All multiprogramming X time sharing but all time sharing ✓ multiprogramming

CPU so fast that X know if CPU give service to other in certain time

Some time E_1 given time for 10μs.
Next 10μs E_2
Next 10μs E_3
Next 10μs E_1
(the 10μs bordeans)

This is time quanta & Round Robin Schedule



We X identify for 20 μs give service to another

Parallel Processing

Introduced in 3rd generation

Saathi

Date ___/___/_____

Fourth Generation:

→ 1970 - 1980
→ LSI VLSI, Ultra Large Scale Integration
→ Parallel Processing known
→ Massively Parallel Processing introduced
↓
How much parallel processing possible
→ Very sophisticated i/p, o/p ~~so that can handle~~

→ ~~Sophisticated~~ execution to handle scalar, vector data introduced.

→ Realisation of high speed computation.

Fifth Generation:

→ 1990 - Present
Major Changes / Objectives
→ Expert Sys replace human brain
→ Hardware will process knowledge, voice, video, image
→ Ultra VLSI, Ultra LSI, Wafer Scale Integration came
→ Complete departure from Conventional machine / Von Neumann Machine

Prior to 2000
peripheral industry ~~x interested~~ to accept dataflow machine
Peripheral dominated

2 grps → Processor prefer
→ Peripheral prefer

Page No.

Saathi

Date __/__/__

~~1975 Initially Supercomputer dominated~~

~~later Minicomp~~ ~~with~~ ~~micro~~ ~~procs~~

Microcomputer was not dominated by supercomputer till 2005.

Supercomputer → Large size

↓

Till ~~X~~ demand

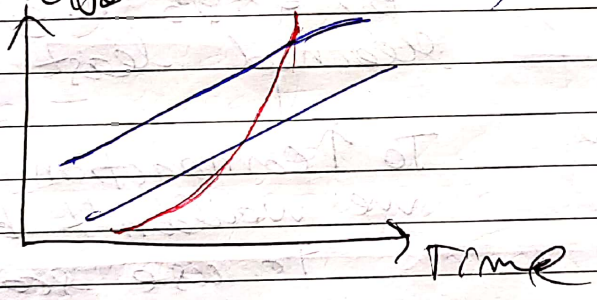
~~X~~ dominate

~~Cost~~

Requirements

↓Size, ↓Power, ↑ Computation Power, ↓Cost, ↑ Speed

Performance (Wronskit)



↑ Transistors in single chip to ↓ size

Compaction density

No. of gates per sq. cm

This should be ~~less~~ more

Moore's Law

In 1965 Moore proposed

The introduction of func. in chip will show exponential growth

No. of transistors in chip double in 24 months, but ~~Moore~~ everyone thought it would do so in 3 years

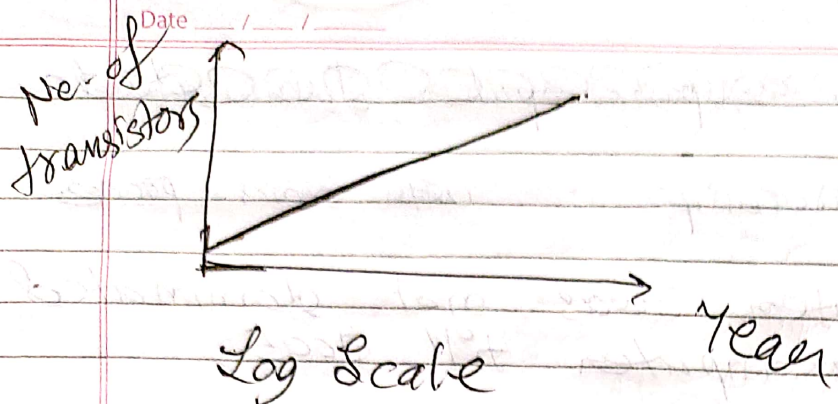
↑ Transistors ⇒ ↑ Computation function included

Every transistor generate heat

↑ Compaction density ⇒ ↑ Heat in small area.

Some area in chip hottest → damage of chip. ∴ X Easy to increase Compaction density.

Date / /



~~Moore's~~ If need ↑ compaction density we need to change scale of ~~integration~~ & technology used.

25nm tech.
18nm tech.
7nm tech.
5nm tech.
4nm tech.

→ Scaling of tech. to ↑ compaction density

↓

By 2025 may have reached least size

↓

To ↑ compaction density we wouldn't be able to use scaling.

Power Consumption : Power has ↓

1971 → Intel 4004 → 4 bit processor

↓

No. of transistors = 23 000

Now XION 5 ~~[illegible] transistors~~

↳ 800 crore transistors

Date __/__/____ Saathi

Cost:

Coodge Law

Performance of machine comp. $\propto$ (Price)^2

Performance

↓

User's Point of View (CP, O, V.)

↳ How much time to execute program

↓

Depends on

(i) Program size

Say NI instructions

Find no. of prog cycles required per instruction

Called CPI.

Find no. of progs for instruction

Find time taken per prog.

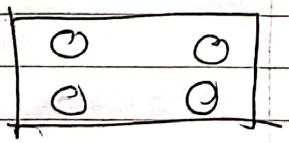
NI × CPI × T = Total time

Total time

Quantum Clod Automaton (QCA)

Info flow through coulombic interaction

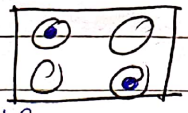
Key device is a cell



4 holes where e⁻ can settle

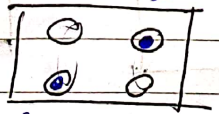
By some means 2 e⁻'s put in same hole.

Now 2 stable states



Logic 0

or



Logic 1

→ Max. dist. b/w e⁻s for 1 stable

Date __/__/____

Saathi

Based on this design diff. logic gates

Changes

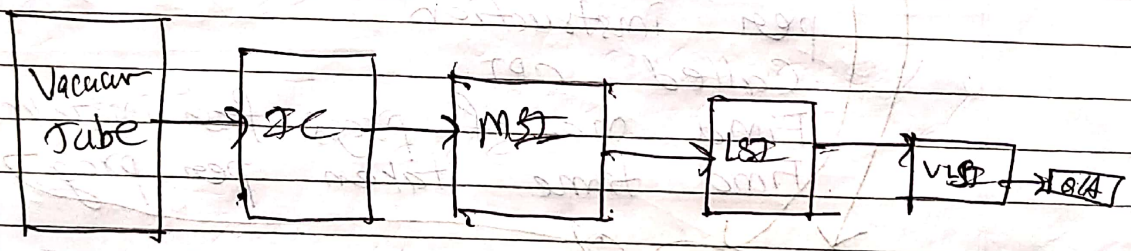
Vacuum Tube
Big Size, Weight
↑ Cost, Power
Consumption

→ IC
Comparatively
lesser cost, size,
power consumption

MST

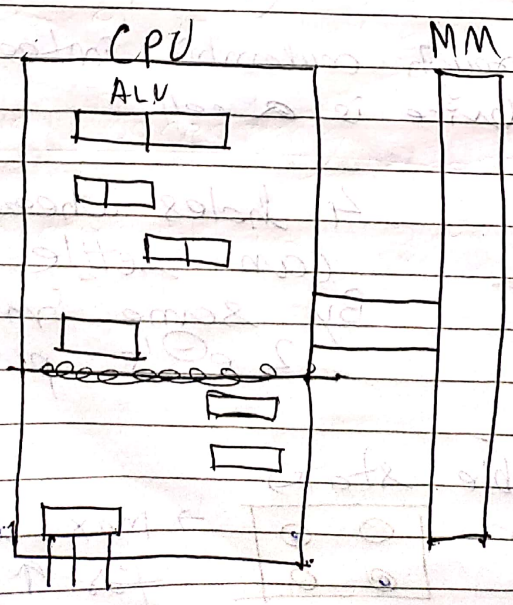
LST

VLST → ULA



11/1/23 Wed Jan

Von Neumann Architecture



ALU
Arithmetic Logical ckts,
Accumulator

Page No.

Date __/__/__

Accumulator → Temporarily stores the operand
↓
Temporarily stores output of ALU ckts

1 operand → DR
other → Accumulator

DR → Data Register
MDR Memory Data Register

Data transfer blw memory & CPU will be through DR

AR → Address Register
MAR Memory Address Register

Any memory access
say CPU access an address
This address first stored in AR.
Address of memory location first stored in AR

DR, AR → Control transfer blw memory, CPU

PC → Program Counter
Stores address of the memory location from where next access will be done
i.e, Stores next executable instruction address

IR → Instruction Register
After instruction available at CPU, it should be decoded → Func. of IR, Decoder
Depend on instruction control ckt. generates sequence of signals to do operation

Control ckt.

Date ___/___/___

Saathi

At a time 1 instruction faced, decoded, executed

Adv

Program & Data & instruction will be stored in same place / memory

1946-48 Von Neumann proposed & developed

Icache
Main memory to CPU data transfer operation cause to Icache

Dcache
Operand to Dcache

Let machine have few no, of instructions like x → memory location

LOAD ~~Load~~ x — Content of accumulator will be x

STORE ~~Store~~ x — Accumulator content go to

ADD ~~Add~~ x — Accumulator content go to (x + DR)s content

AND x — Accumulator content go to x AND DR's content

JUMP x — Program counter counter is x

JUMPZ x — If accumulator content 0 jump to d

RSHIFT — Accumulator content shifted right by 1 and

Comp — Accumulator content complement of accumulator

Macro instructions

Date ___/___/___

Content of PC transferred to AR
Instruction read from memory

When instruction fetch DR any info has 2 codes
- operation code
- operation code

PC ready for next instruction

If 1 word → PC increment by 1
k words → PC increment by k

Now decode if instruction is load
If load, then perform operations

Load → Fetch from memory location x

AR contains x
Content of memory x go to DR
Ultimate destination accumulator
∴ From DR to accumulator

Micro instructions / operations → set of instructions to perform 1 macro instructions

Load → 3 micro operations

For each macro instruction there is 1 micro program

Saathi

Date

Transfer data from accumulator to memory location x

Accumulator → DR → But then address lost

So transfer
DR operand part to AR first
→ Address x in AR

Control unit generates control signal CE which says operation is to be executed.
Control unit generates sequences of signals which execute micro instructions

old edition
Computer Architecture & Organisation
JP Hayes

Disadv: If operand, program in same memory may execute unwanted instructions

This is resolved by TAG architecture
For every memory word, there is additional bit that says character of word / whether to be executed or not.

But TAG is costly due to additional bit memory
↑ Time to decode also ∴ extra memory
For each
16 bit word → 17 bits

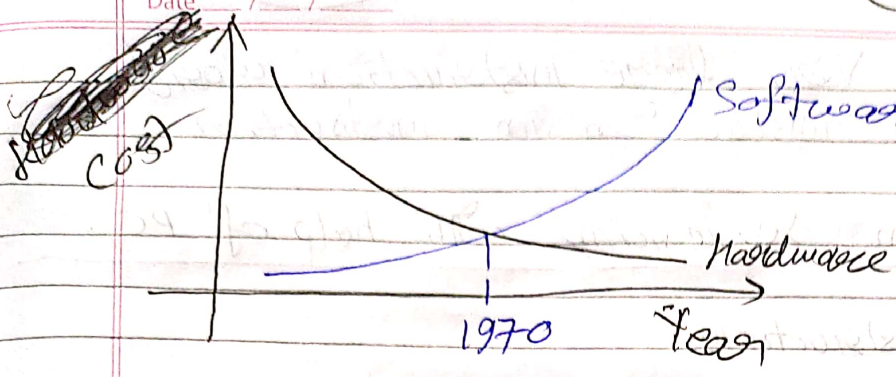
∴ Initially TAG ✗ so popular

12/1/23 Thu Jan

CPU Bus

Page No.

Date / /



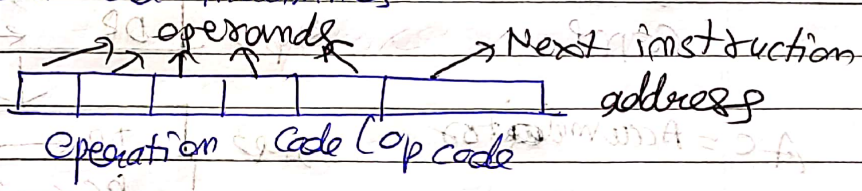
∴ Now TAG architecture dominates
∵ hardware cost ↓

12/1/23 Thu Jan

CPU should maintain sequence in which instructions should be executed,

I_1
I_2
I_3
⋮

In old machines:



Each instruction has an operation code,
Next instruction inside instruction itself

But for each instruction, length of instruction becomes large, memory required ↑, memory was costly, width

CPU Bus, also should be very large through which CPU collect info. from memory ⇒ ↑ Pins in bus.

⇒ ↑ Size of CPU

∴ Discard putting next instruction in instruction
Now PC stores next instruction address
∴ Reduce length of instruction

Page No.

Date ___/___/___

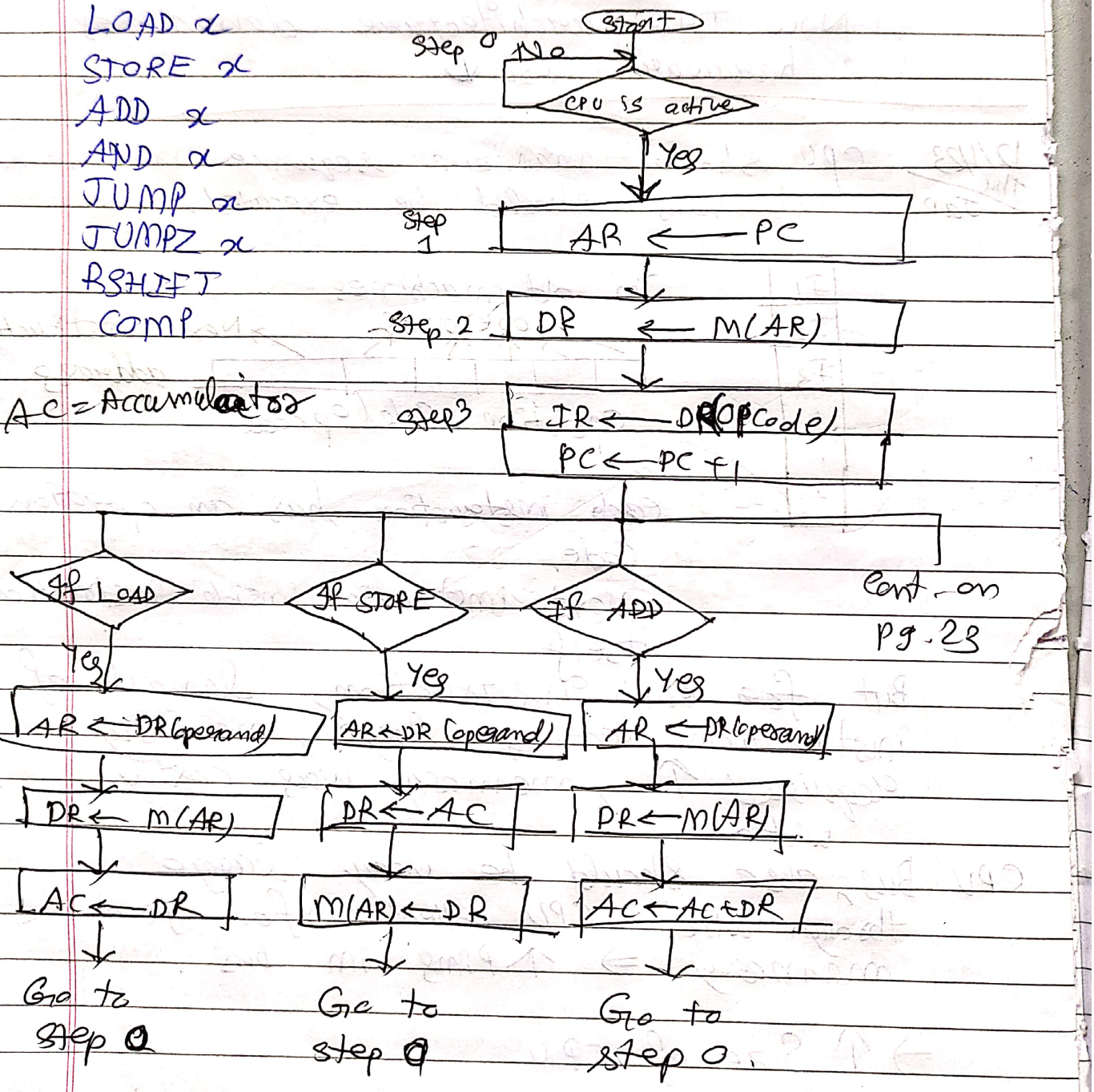
VLIW → Very large instruction word
↳ Next instruction in instruction.

Instruction Sequencing with help of PC:

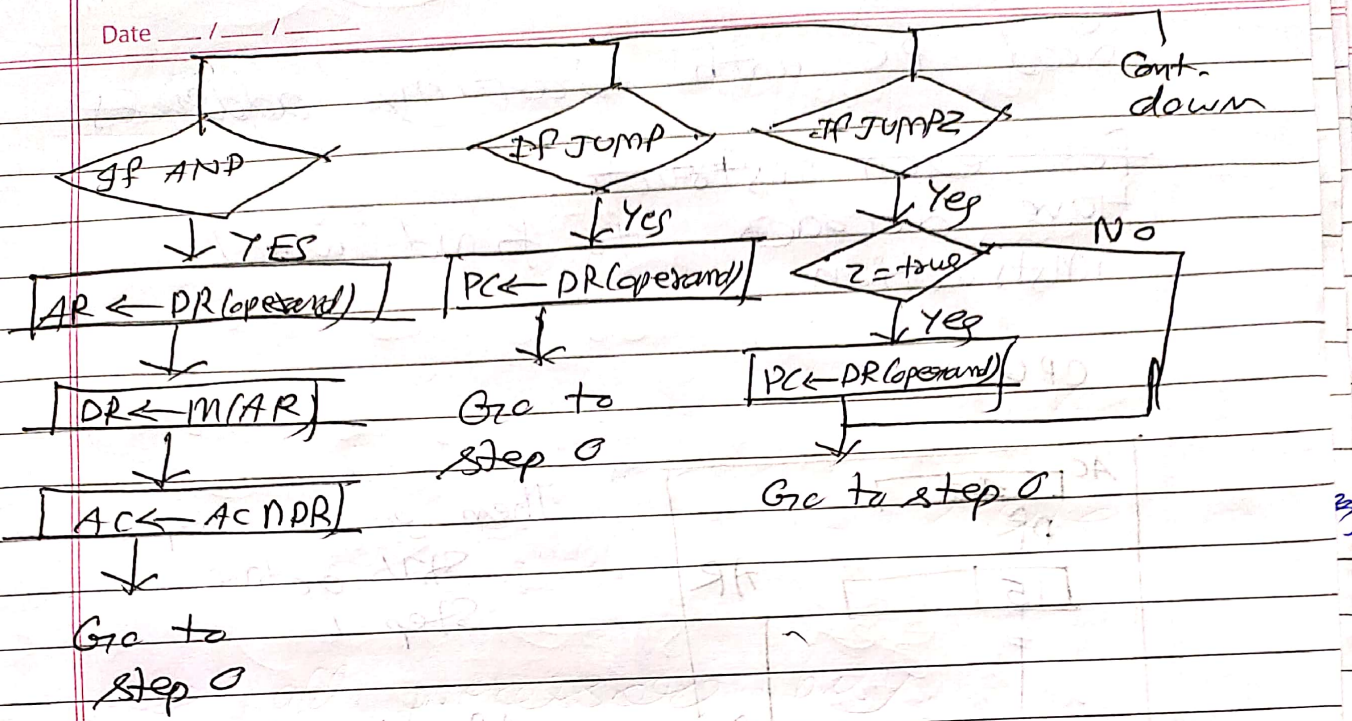
Macro Instructions

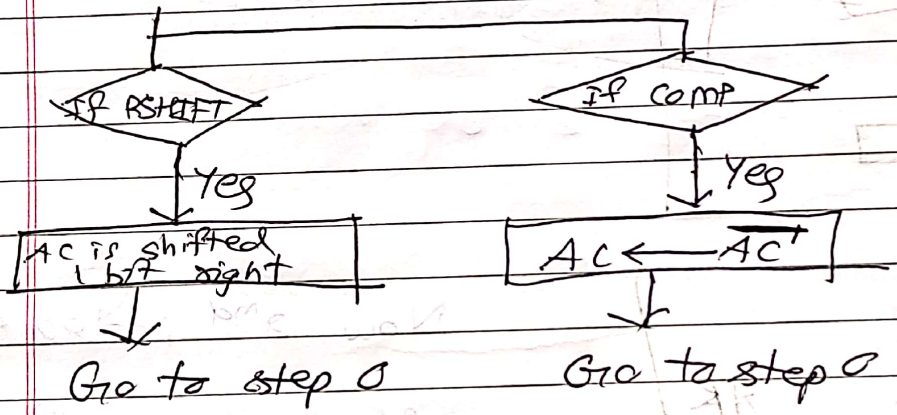
LOAD x
STORE x
ADD x
AND x
JUMP x
JUMPZ x
RSHIFT
COMP

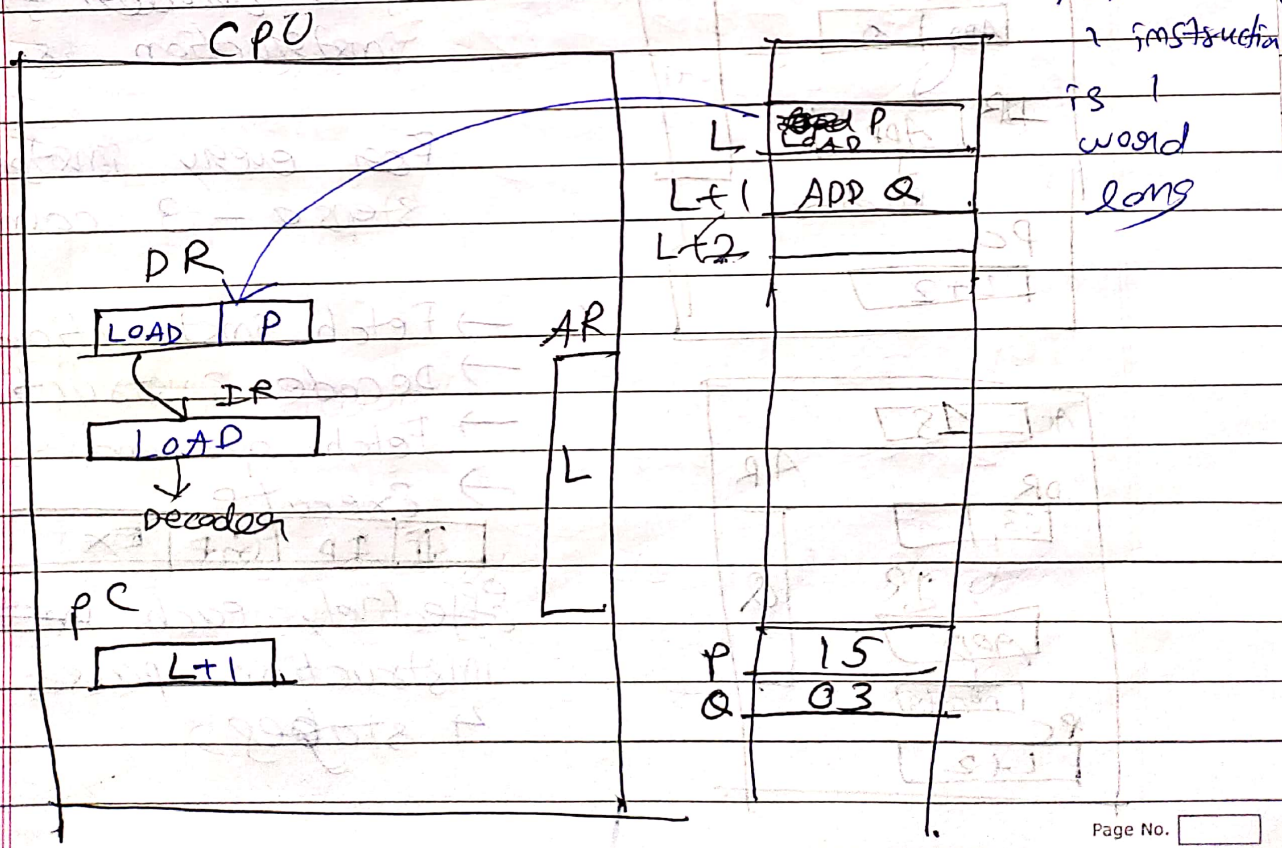
AC = Accumulator



Check if CPU Switched on or not







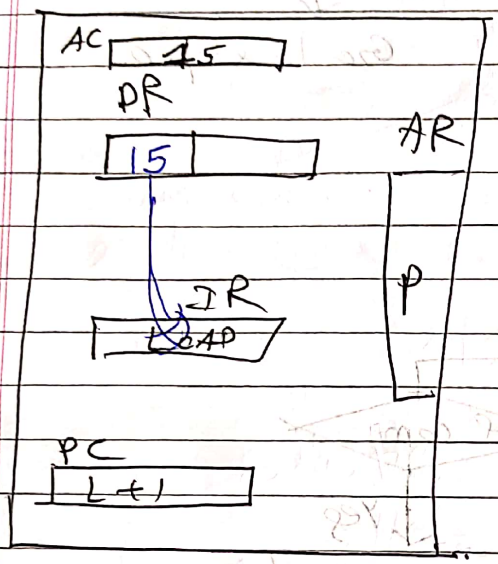
Saathi

Date ___/___/_____

Load PC with ~~starting~~ address.

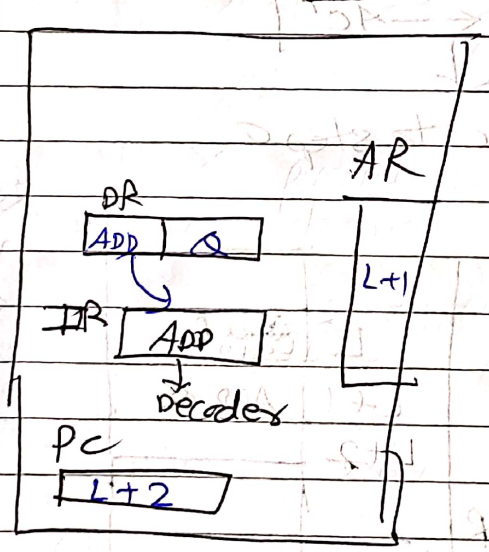
~~For each instructi~~,
Have decoder to find which instruction

CPU



Then go to Step 0
∴ SHL active
∴ Step 1

AR : L+1



Now 2nd instruction

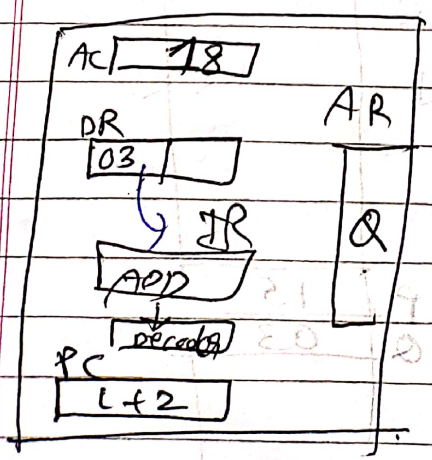
After decoder find instruction is. ADD

For every instruction steps 0 – 3 common

∴ → Fetch instruction
→ Decode instruction,
→ Fetch operand
→ Execute

IF	ID	GF	EX

Effectively each ~~instructure~~ instruction passes above 4 ~~stages~~



Date ___/___/___

Upgradations to Von Neumann machine

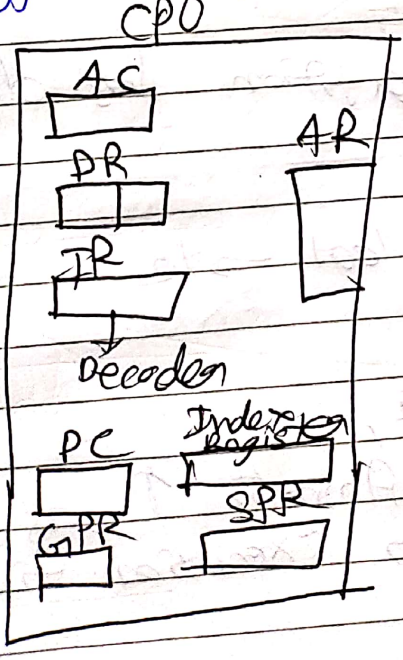
1. Less no. of registers in Von Neumann
AR, PC, DR, IR, Accumulator

In today's machine there are ↑ no. of registers like General Purpose Registers, Special purpose registers that may not be directly accessible by programmer.

There are also index registers
During operand addressing → Direct
↳ Immediate
↓
Another addressing will use index registers to find address of operand.

ADD Ix Y → Using index register Ix to find operand address

Effective operand address = Y + Ix



2. Von Neumann → Simple operation

Today → ↑ Complicated operations like multiplication, division
↓
Costly but fast

Von Neumann → 1 ALU

3. Today → k partition ALU
↓
ALU divided into parts
↓
All func.s of ALU may not be required simultaneously
↓
Integer arithmetic operations | Shifter | Floating point operations, etc.

This satisfies 2 requirements

Comp. Arch. → Mostly designed by Survey & much maths.

If × partition, all operations require same time say t.

If partition, each partition has diff. computation time

Integer → t_1 ; Float → t_2

Obviously $t_2 > t_1$
$t_2 > t$ but $t > t_1$
∴ If use integer arithmetic ↑ as compared to float arithmetic → execution time ↓

Date ___/___/___

Saathi

Multiple instructions can be handled in CPU all using ALU

If each instruction use diff. partition of ALU

can execute all instructions simultaneously

∴ Execute faster,

4. k no. of ALU's

Now execute k no. of instructions simultaneously,

5. If von Neumann has instructions $I_1, I_2, I_3, \ldots$

I_2 x touched till I_1 completed

Any communication between CPU, memory costly takes lot of time

Communication within CPU / within memory faster.

If in CPU can have queue of registers,

say I_1 in execution then can get ready for execution of say I_4,

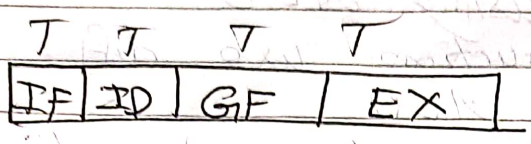
Instruction will be prefetched

This is instruction prefetching

Date ___/___/___

Saathi

6. Pipelining:



Say all parts take T time

When IF of I_1 complete & I_1 goes to ID, can start IF of I_2.

If first finish I_1 & then start I_2 & so on then for n instructions 4Tn time

$4T + (n-1)T$ time of execution here

$4nT >> 4T + (n-1)T$

When $I_1 \rightarrow GF$

$I_2 \rightarrow ID$

then $I_3 \rightarrow IF$

7. Parallel Processing:

Simultaneously more than 1 instruction executed → Say one executing, other fetching operand

Von Neumann → One by one

8. Speculation / Speculative Execution:

Speculation is inbuilt in architecture of machine.

Date __/__/____

OS:

```
if  D = 0  then
      r = 1 + 5
else
      r = 1 + D
```

If os finds 2 processing units available will execute if in 1 & else in other processor.

Keep all processors busy.

Speculation is similar to this.

Control Speculation

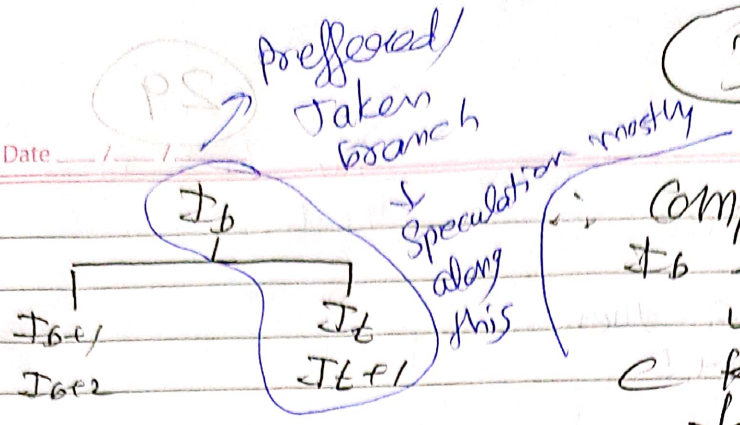
Data Speculation

Say program with

```
:
:
Ib      → Branch Instruction → Say if c = 9 go to It
Ib+1
Ib+2
:
:
:
It      → Target
It+1
```

When CPU execute Ib, will decode & deduce branch instruction.
But X know how to proceed till e known, can't predict next instruction

Date



∴ Computation of I_b should wait till c known

But ✗ afford this delay

∴ Speculate:

Say cond^n satisfied, Then go to I_t

Have to make sure speculation is rarely false,

If say speculation is false, then should flush out data of ~~data~~ execution when take speculation true,

But this is not so simple,
So store it temporarily, so that can delete result if speculation wrong,

Data Specalation

Say I_1, I_2, I_3

I_1: a, y
I_2: x, y
I_3: x, y

I_2 ✗ proceed till I_1 done
But have parallel processing

May be that all of I_1, I_2, I_3 ready to execute (processors available)

But have to wait for dependency
If know x, y can execute instruction

Date ___/___/___

So speculate data

Again if speculation incorrect, data flush out.

Today misspeculation is close to 0%.

9. Green Computing

If speculation incorrect will be losing some time

If use eager execution i.e., follow both paths simultaneously - One correct, one incorrect. Discarded incorrect

↑ Delay

↓ Faster Execution

↓

All instructions executed ⇒ Power consumed

I_b

I_{b+1} I_{b+2} I_f I_{f+1}

Rule of green computing ⇒ If followed then machines with ↓ power

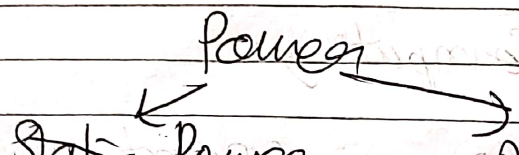
Now

↓

If portion in ALU xutilised → Switched off to save power

Date __/__/__

So replace 1 powerful core processor with ↑ power, ↑ fast computation
with multiple ↓ power processors (multicore).
Each core will have lesser ~~execution~~ execution speed than single core



$P_D / P_d = CV^2f$ * check

Speed of execution depends on clock freq.

↑ Clock freq, → ↑ Running time

↑ Cores, with ↓ clock freq, → ↓ Power

X reduce V beyond certain threshold as otherwise noise will destroy system.

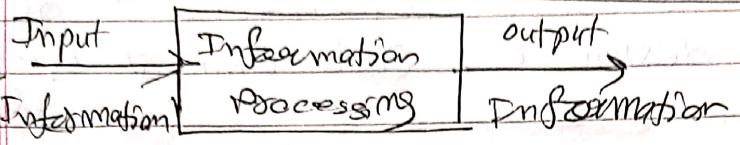
∴ Can only control f.

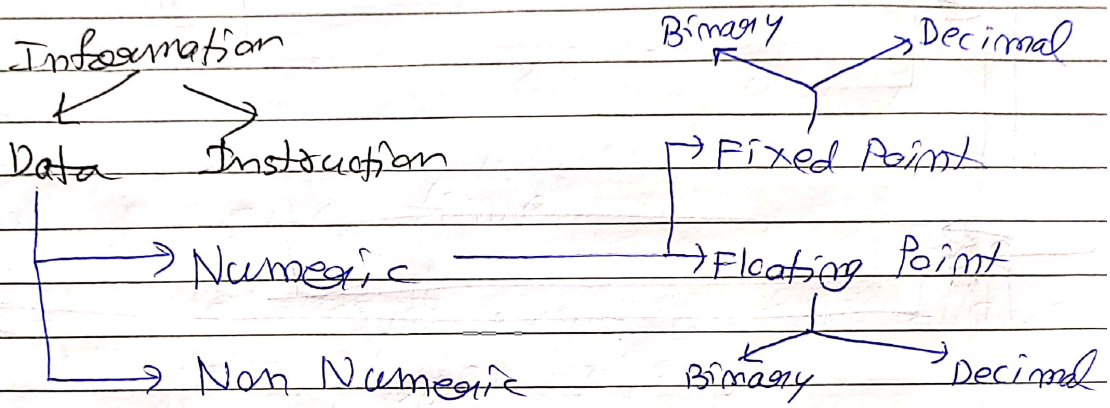
CMP (Chip Micro Processors)
Within single chip, CPU includes billions of processors / processor core.

Date __/__/____

Bottleneck von Neumann
↳ Program, Data stored in same place.

18/1/23 Wed Jan





Early machine computer had only assembly machine language i.e, hardware supports only simple structured information

In high level language have array but In earlier computer → & declare array

This translation requires additional cost
↳ compiler
↳ System software support
↳ Delay

have to use other way to do an array, routines through which when translated base hardware can support

This gap is called semantic gap

Gap b/w language & hardware support

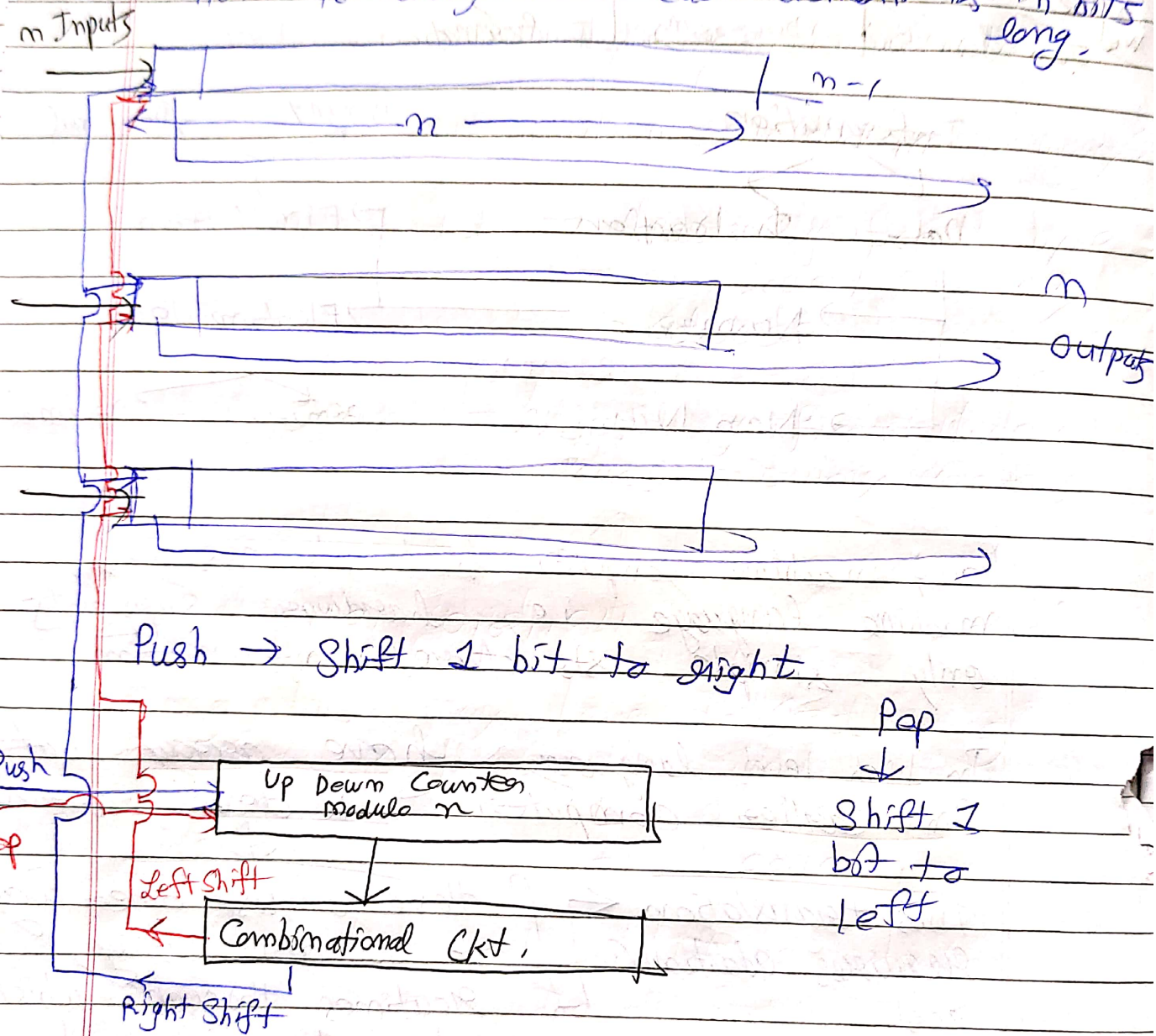
Page No.

Date

Construction of Stack

n elements, each element size l

Use shift register (bidirectional) in no. to design it, Each element is n bits long.



Push → Up down counter increment 1 & right shift

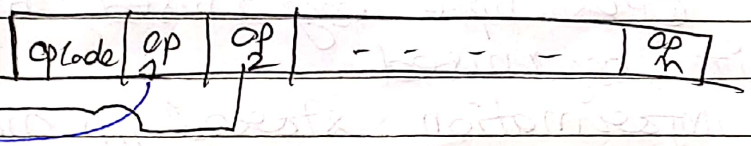
Pop → Up down counter decrement 1 & left shift

Date ___ / ___ / ____

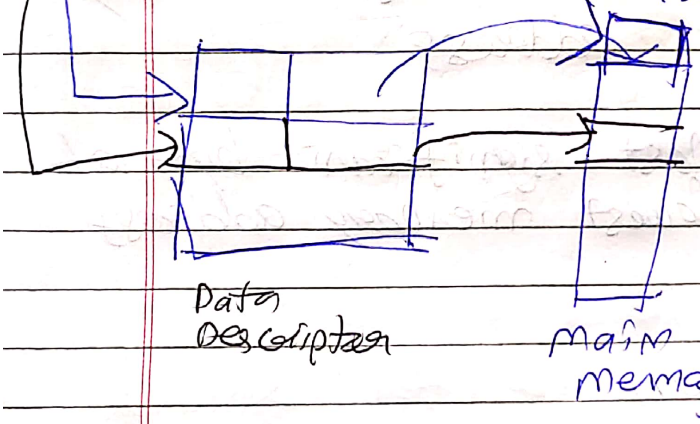
Saathi

One way to ↓ semantic gap is:
Data Descriptors.

Say declare variable with type, public/private/protected or not, ...
How to describe this to machine?



Each op (operand) is a variable
In data descriptor for each operand we can have some information



Tag also helps to ↓ semantic gap.

Tag can be 1bit, 2bits, 3bits, ---- long depending on use.

Tag may be used to tell type of variable like long, short, etc.

n bit processor ⇒ Can process n bits at a time.

8086 → 8088 Both have 16 bit processor
8088 → ↓ bus width than 8086 → ↓ size of chip.

Saathi

Date ___/___/___

$m \times d$ memory

$d = 1$ → Bit organised

$d = 8$ → Byte organised

$d = 16, d > 8$ → Ward organised

Memory of CPU byte organised, but processor bit organised

How is information stored in memory?

- Little Endian → Least significant byte at lowest memory address
- Big Endian → Most significant byte at lowest memory address

Ex: 11001101 01011011

Big

L	11001101
L+1	01011011

Little

L	01011011
L+1	11001101

Most machines have Bi Endian i.e, both big & little endian, with a switch can access which one.

Date __/__/____

Saathi

3 digit decimal → 10^3 patterns
3 digit binary → 2^3 patterns

$10^3 > 2^3 \Rightarrow$ Decimal store ↑ info than binary
↓
↑ Efficient

But to store will require ↑ memory in decimal as here have to distinguish 10 values whereas in binary 2 values

Consider n digit no. in base b.
We have to find most efficient b value

$b^n = N$

Cost $\propto n$; Cost $\propto b$

$\Rightarrow$ Cost $= k n b = k \frac{\log_e N}{\log_e b} b$

N fixed

want to find least cost

$\Rightarrow$ Derivative of cost = 0

$\frac{dc}{db} = 0$

$\frac{d}{db}\left(k' \frac{b}{\log_e b}\right) = 0$ $\quad k' = k\frac{\log_e N}{\log_e b}$

$\log_e b - 1 = 0$

$\log_e b = 1$

If base is e then,

$b = e \rightarrow$ optimum

$b \approx 2.78 \rightarrow$ b can't be fraction

$b = 3$ is unreliable

$b = 2 \rightarrow$ Some cost is sacrificed, but it reduces complications, ↑ reliable

Page No.

Saathi

19/1/23 Thu Jan Date __/__/____

~~Information~~

Data covered in 11th, 12th, digital logic

Instruction Set:

Instruction Set should be:

(1) **Complete:**

Instructions included should be such that programmer can solve most common problems with them

(2) **Efficient:**

Frequently used instruction should execute fast
For this may have to ↑ delay of other X frequently used instructions.

(3) **Regular:**

Instruction set should be regular,
Say instruction I do task T

Then another instruction I_1 should be able to do T' i.e, opposite of T.

(4) **Compatible:**

Atleast part of design should be available in market, otherwise design cost ↑ as design everything then.

Date ___/___/_____

Saathi

16 bit adder in market
Can design 32 bit adder from this

But for 24 bit adder need new design with new floorplan, routing, etc.

∴ Should be compatible,

Some standard instructions should be included like:

→ Data movement (Move from 1 register to another, Loading instructions)

→ Arithmetic instructions (addition, subtraction)

→ Logical instructions (and, or, compare)

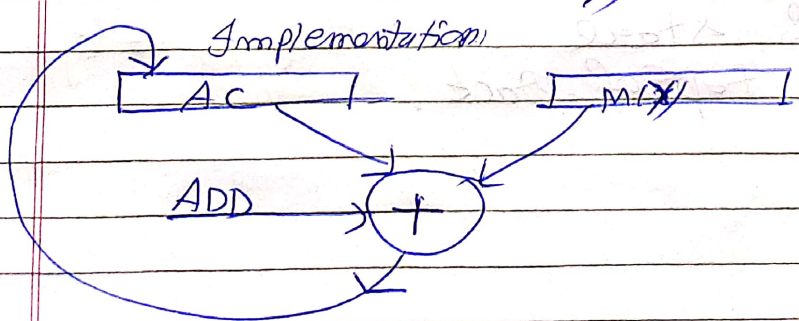
→ Branch control instructions (jump, jumpz, jump positive, jump negative)

→ Debugging instructions (break)

→ Input output instructions

→ Misseellaneous instructions

In von Neumann Architecture
For ADD X: → 1 addressed instruction
AC ← AC + M(X) → AC, AC, M(X) = 3 operands



If ADD = 1 then activated & add

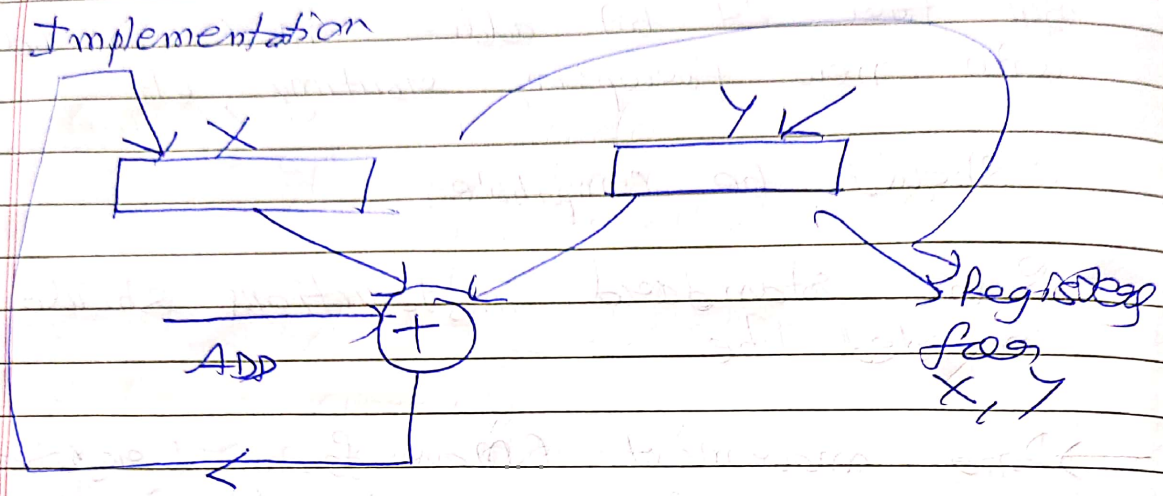
Saathi

Date

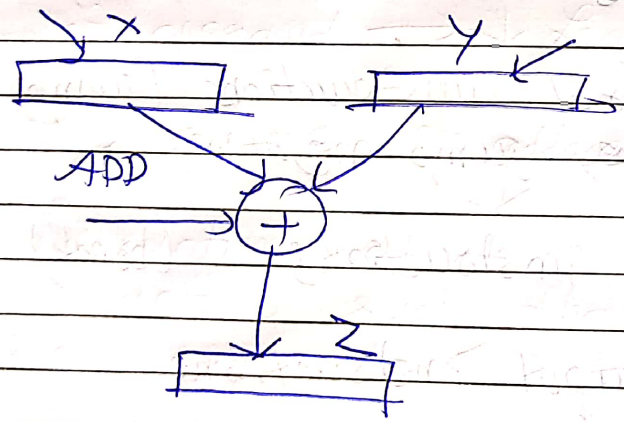
2 addressed instruction

Say have ADD X, Y and want
$X \leftarrow X + Y$ → 2X + 1Y = 3 operands

Implementation



ADD X, Y, Z → 3 addressed instruction
$Z \leftarrow X + Y$ → X, Y, Z are operands



Say have ADD → zero address instruction
∴ No address mention

Add topmost element of stack & store result at top of stack.

Date __/__/____

Saathi

Instruction format

operation Code (op code)	op1	op2	...	opn

L = Instruction Length

To optimise program → Program should have
↓ memory
Should need ↓ memory to store instruction
↓
∴ Instruction length should be optimum
↓
Also no. of instructions should be optimised

Say have $p = a+b+c$
For ADD X, Y → 2 operations
For ADD X → 3 operations
↳ First load X, then 3 additions

Explicit operand → operand specified
Implicit operand → operand X specified

In ADD X
AC ← AC + M(X) → 2 implicit + 1 explicit

ADD X, Y, Z
Z ← X + Y → All 3 explicit operands

Saathi

19/Thu

Date __/__/__

First try to ↓ length of individual instruction

Components
- operation Code
- Operand fields
  - No. of operands
  - Size of operand

to ↓ length of instruction may
(i) ↓ No. of operands
(ii) ↓ Size of each operand
(iii) ↓ Opcode

No. of operands depends on how many operands implicitly specifying

↑ Implicit operand ⟶ ↓ Length of instruction

For this define m addressed machine

↓

An instruction set of machine computer all instructions such that no. of explicit operands ≤ m,

m addressed ⇒ Atmost m explicit operands

In 0 addressed machine there are 2 single addressed instructions are there
(i) Push (ii) Pop.
In 0 addressed instruction length = Opcode
Drawback:
To solve a problem may need to give ↑ info.

Date ___/___/___

Saathi

Q. $z = w - x + y$

Find how many instructions to solve for 3 addressed, 2 addressed, 1 addressed, 0 addressed machine.

3 addressed:

2

```
SUB z, w, x        z = w - x
ADD z, z, y        z = z + y
```

2 addressed:

3

```
ADD w, y           w = w + y
SUB w, x           w = w - x
ADD z, w           z = z + w
```

1 addressed:

4

0 addressed:

6

```
PUSH w
PUSH x
SUB
PUSH y
ADD
POP z
```

Date ___ / ___ / ___

Operand Addressing Mode → + Length of op code

Date __/__/____ Saathi

Operand Addressing Mode → ↓ Length of op code

25/1/23 Wed Jan

(ii) AND x → 1 operand
AND x, y, z → 3 operands
x ← y AND z
Te ↓ size of each operand

(a) Immediate Operand Addressing

ADD 07 → Supplying operand within instruction itself
07 is operand ~~address~~

ADD x → x is operand address

AC ← AC + 07

Have | opcode | 07 | → ↓ size for 07 need 1 byte

Called → Immediate operand Addressing

ADD immediate → 2 byte instruction (1 byte for o

But in ADD x

| opcode | x |

x → Depends on of address lines of CPU

Here give x, & value of x M(x)

2 byte if each 1 byte

Total = 1+2 = 3 byte

Called absolute/direct operand address

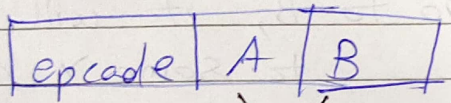
memory addressing
operand address is memory

An direct operand addressing
→ Size depends on no. of CPU address line
→ Access memory ⇒ Slower

But in immediate operand addressing
it is faster as immediate access.

(b) Register direct addressing
✗ mention memory address
✓ mention register address
Say which register has operand.

Mov A, B
A ← B



→ Has less storage than direct operand addressing
A, B have size < 1 byte
total storage

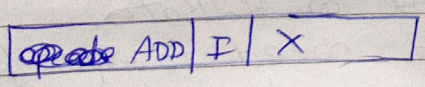
Addresses → Contain operands

8085 → 8 general purpose registers.
Registers are fastest memory in system.
∵ inside CPU

Memory of latest tech is faster than CPU
Access time of CPU from memory is slower than from register

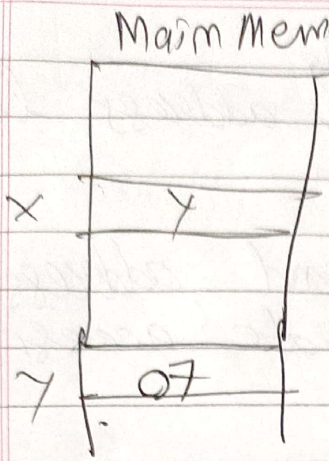
(c).1 Indirect Addressing → Memory Indirect

ADD I X



If X memory address, then say require 2 bytes.

Date __/__/____

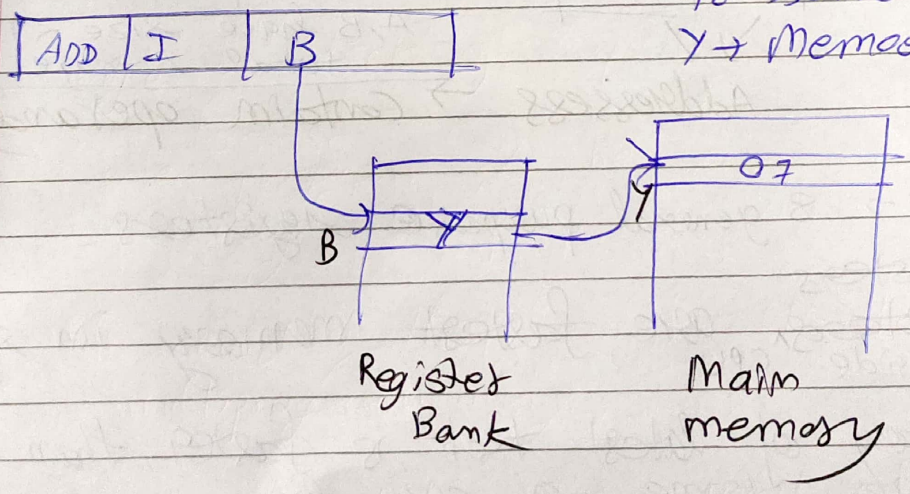


Go to X
Get ~~Read~~ Y
Go to Y
Get ~~Read~~ operand

By changing Y, can change operand

1.2 Register Indirect ~~Addressing~~

ADDI B
AC ← AC + B
Size ↓ : register
Also when go to B, it will point to some Y (given)
Y → Memory address



(d) Relative Addressing

Say have 1234 5678 in Kolkata
↓
Relative in India, Absolute in Kolkata
But if X in kolkata need to specify ~~003~~ 033 to make in kolkata
↳ Absdute wrt India

Date / /

PC relative

ADD R x → Go to PC, move by x ~~some address down~~, get operand

Absolute → Specify x → 2 byte

If PC → 2 byte
x → X be more than 8 bit

ADD R x → operand at PC + x

ADD	R	x

↓

Atmost 8 bit
X 2 byte
X specify absolute address of operand

(e) Indexed Addressing

Specify index register no.

ADD Ir p

Search for Ir
Content will be ~~arranged~~ accessed

(f) Base Addressing

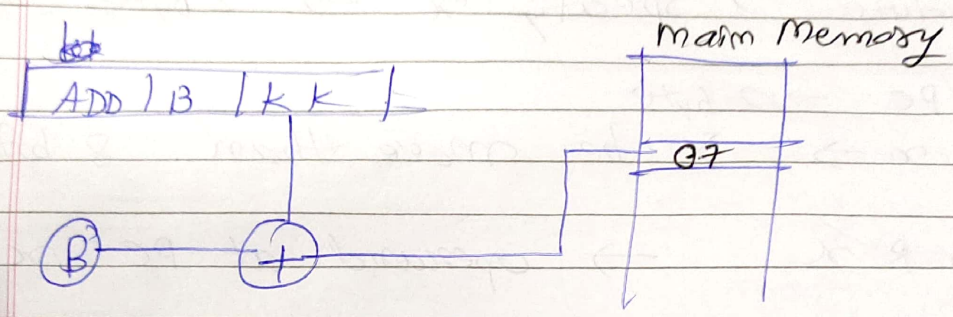
Sort of relative addressing

In CPU have base register

| ADD | B | KK |

Actual operand → Base address displaced by KK

When program loaded in memory we x gne address of variables



(g) Page Addressing

Page → Block of equal size

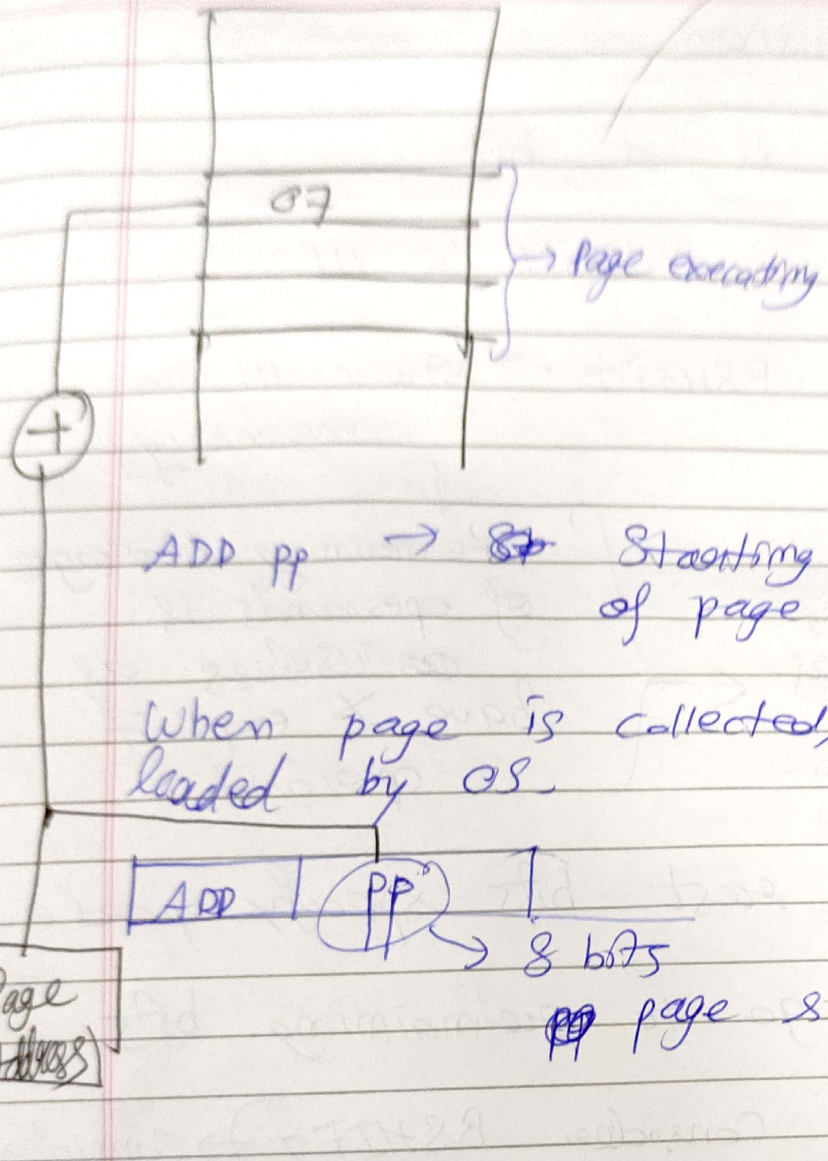
Whole program x be accomodated in memory at particular time

If could, then difficult to do multiprogramming

Resident part of OS → Imp. part of OS that is loaded in main memory

Program divided in no. of blocks
All blocks x available simultaneously
Only required blocks in main memory at particular time.

If block sizes are diff., then called segments.



ADD pp → Starting from address of page, go pp distance

When page is collected, page register loaded by OS.

pp page size can't be very large

(iii) How define Opcode for instruction

(a) Fixed format

Say have 3 bit opcode

⇒ No. of instructions $= 2^3 = 8$

↓ size of opcode → Limit no. of instructions

↓

But can use 1 level of decoding to find instruction

Date __/__/____

Saathi

(b) **Opcode Extension**

Say opcode is 3 bit

Let opcode of RSHIFT be 111

When use RSHIFT → store 111 in memory

↓

Remaining storage of operands is useless as have x explicit operand

Consider remaining storage as opcode extension

↓

If 111, next bits specify opcode

i.e,

If 111, go to remaining bits

If 00000 Consider RSHIFT
If 00001 consider CLR
↑ no. of instructions

→ Consider each instruction 8 bit

Drawback

With ↓ no. of bits, can ↑ no. of instructions

But now 1 level of decoder ✗ sufficient as if 111 need another decoder

Decoding slower

Date

Data operand Dependent opcode

Say have 3 bits which specify ADD

Say ADD 2 instruction (i) ADD integer (ii) ADD float

need 2 opcodes

ADD x, y, z → Depending on type of operand will go to integer/floating point unit
x ← y+z

Here operand should have tag that is to be decoded

Frequency Dependent opcode

If opcode of all instructions fixed maybe some space of instruction wasted

ADD (B) Register Address → X same length of instructions

ADD (X) operand Address

∴ We have instructions with diff. addressing modes & sizes.

LOAD (X) 1 byte 2 bytes
↓
1+2 = 3 byte instruction

MOV A B 2 bits Registers +3 bit each for 8085
↓
1 byte instruction

Some instructions used frequently & some not so frequently

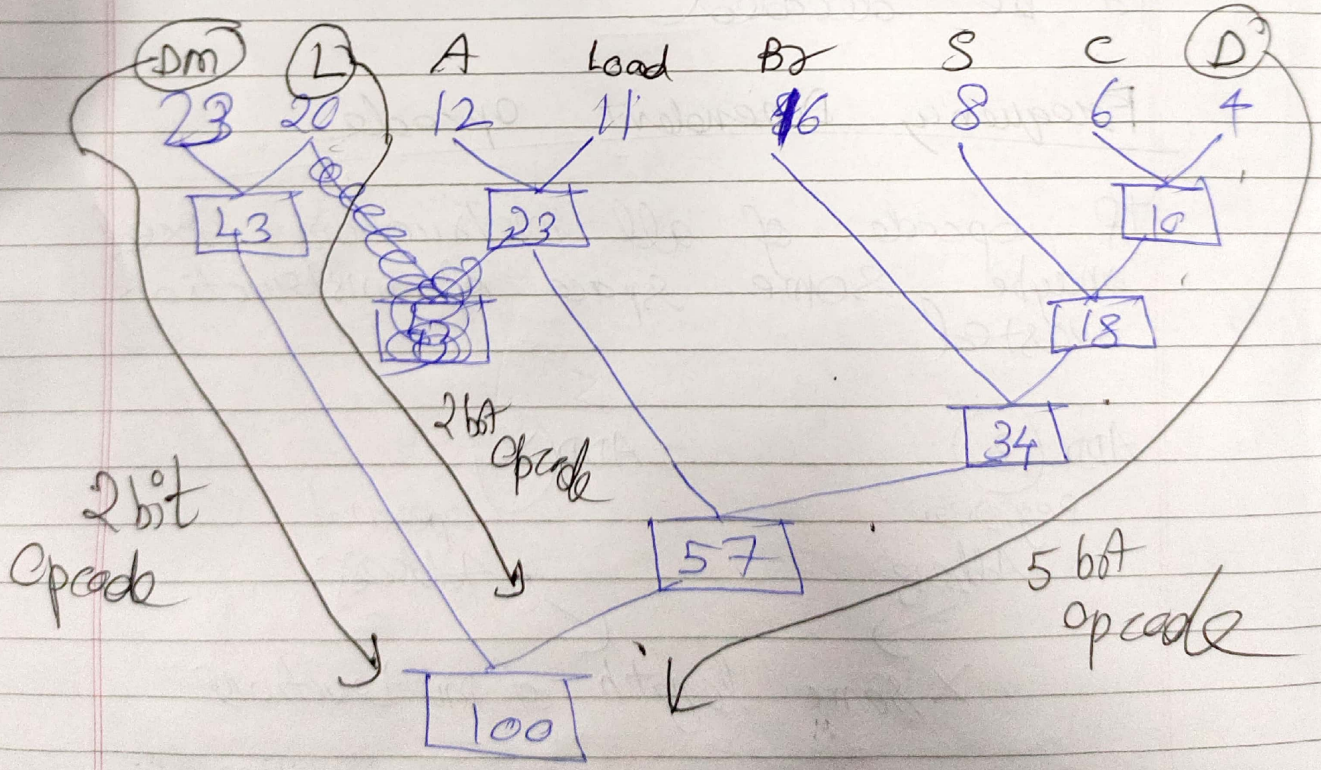
Depending on freq. → one with ↑ freq, try to reduce opcode of instruction

Consider following freq. of instructions

Data Movement	Logical	Arithmetic	Load	Branching
23 DM	20 L	12 A	11 Load	16 Br

Store	Compile	Debug
8 S	6 C	4 D

Above are the branches of node we will merge ones with least weight



apsara

Date: __ . __ . ____

laccess

We want to fetch ↑ no. of operations at a time