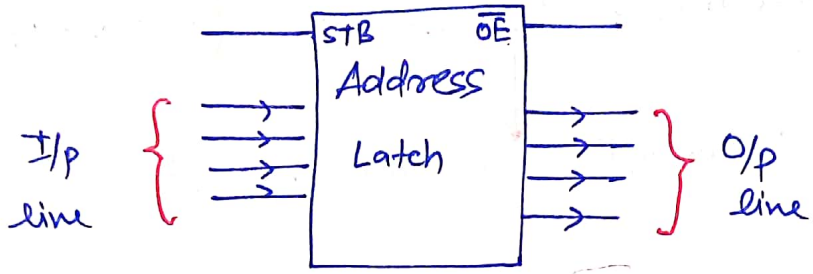
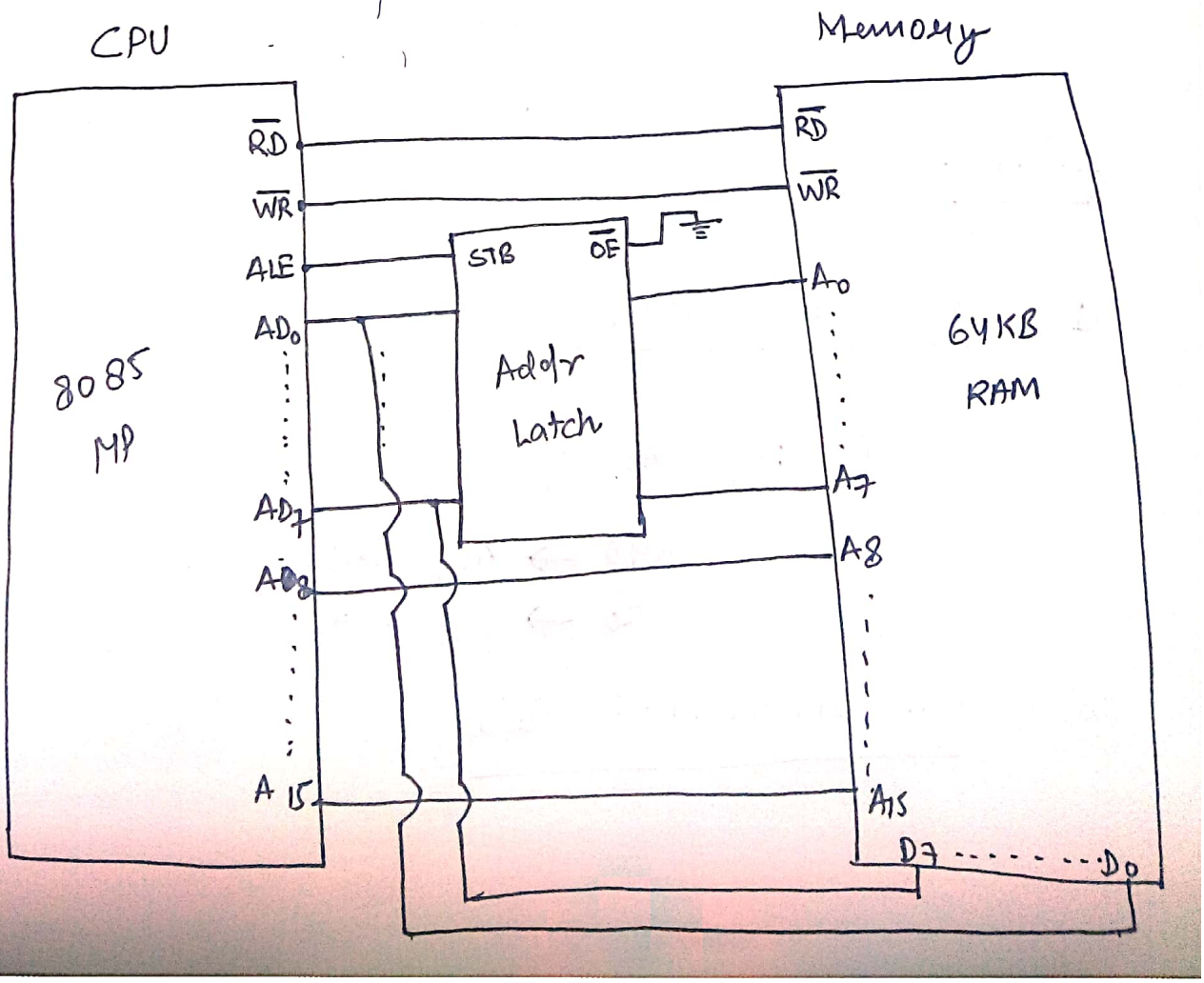


Memory Interfacing:-

- To analyze the memory interfacing, let us consider 8085 µp as a refrence CPU, it contains 16 address pins to refer 64KB space with a word length of 8-bits. So 8 data bits are time multiplexed with address pins.

- In this concept CPU pins are mapped with the memory pins respectively to integreate the CPU and memory units.



READ :

Machine Instruction : LDA, [2040] ; (Acc ← M[2040])

↓

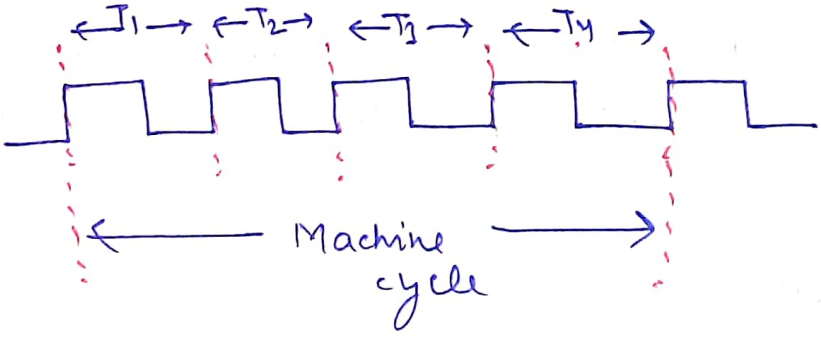
CPU generates memory request

2040	$\overline{RD}$

↓

Machine cycle { # CPU cycles req. to complete the memory cycle }

[T_1, T_2, T_3, T_4]



Actions :-

① Send the address

T_1 : ALE = 1

40 → $AD_7 - AD_5$

20 → $A_{15} - A_8$

② Send the control signal

T_2: ALE = 0

$\overline{RD}$ = 0 (low pulse) enabled

$\overline{WR}$ = 1 (high pulse) Disabled

③ Read the Data [Mem latency / Mem delay]

T_3 & T_4: ALE = 0

$D_7 - D_0 \rightarrow AD_7 - AD_0$

WRITE:

Machine instruction : STA, [8060] ; M[8060] ← Acc

↓

CPU generates the mem request

↓

8060	$\overline{WR}$

↓

Machine cycle [T_1, T_2, T_3, T_4]

Actions:-

① Send the Address

T_1: ALE = 1

60 → $AD_7 - AD_0$

80 → $A8 - A_{15}$

② Send the control signal

T_2:

ALE = 0

$\overline{RD} = 1$ (disabled)

$\overline{WR} = 0$ (enabled)

③ Write the Data

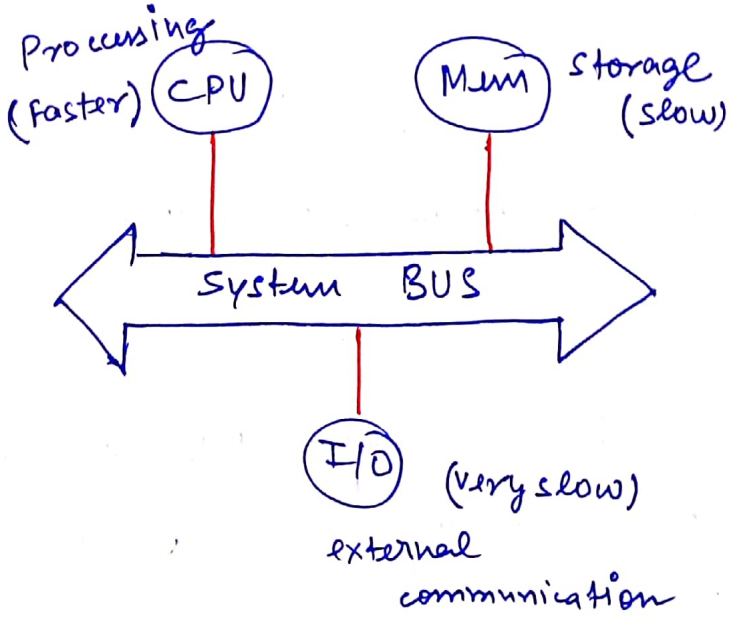
T_3 & T_4:

ALE = 0

$AD_7 - AD_0 \longrightarrow D_7 - D_0$

System BUS :-

- BUS is a communication channel.
- characteristic of a BUS is shared transmission media.
- Limitation of a BUS is only one transmission at a time.
- System BUS is a BUS structure used to provide the communication b/w the major components of a computer (CPU, memory, I/O)



CPU	Memory	I/O
Processing	Storage	External communication
Faster	Slow	very slow
Master	slave	usually slave
	Address & control signals	Port Address & control signals

- system BUS contains 3 category of a lines used to provide the communication names as:

1. Address Lines (AL's)
2. Data Lines (DL's)
3. control lines (CL's)

AL's :-

- used to carry the address, to memory & I/O.
- unidirectional
- Based on the width of address BUS, we can determine the capacity of a memory system (Main mem size)

eg :- In 8085 µp $(A_{15} - A_8\ AD_7 - AD_0)$

↓

16 AL's

↓

2^{16} cells

↓

64K cells

↓

64KB

eg :- In 8086µp $(A_{19} - A_{16}\ AD_{15} - AD_0)$

↓

20 AL's

↓

2^{20} cells

↓

1M cells

↓

1MB ← size of cell

word size = 16-bits

so memory cell interfaced

DL's :-

- used to carry the binary code b/w CPU, Memory & I/O.
- bidirectional
- Based on the width of the DataBus, we can determine the word length of the CPU.
- Based on the word length, we determine the performance of a CPU.

Eg- In 8085 µp

$AD_7 - AD_0$

↓

8 DL's

↓

word length = 8-bits

↓

operations are performed on a 8-bit data code

In 8086 µp

$AD_{15} - AD_0$

↓

16 DL's

↓

word length = 16-bits

↓

operations are performed on 16-bits data code

CL's :-

- used to carry the control signals & timing signals.

- control signals are used to specify the type of operation.

- Timing signals are used to synchronize the memory & I/O operation with a CPU clock.

- Unidirectional.

NOTE: when the system contains common BUS, common control signals, and common address then there is a problem of ambiguity i.e -

I/O address space (1000)

Memory address space (0000 – FFFF)

Now, CPU generates memory request:

1000	$\overline{RD}$

1000 → A's

$\overline{RD}$ → C's

→ mem : 1000 → valid, $\overline{RD}$ → valid

→ I/O : 1000 → valid, $\overline{RD}$ → valid

so, both memory & I/O starts transmitting data and thus causing ambiguity problem.

- To handle the above problem, bus configurations are used in the computer design. These are of three kinds

(i) IOP (Input - output processor)

(ii) Isolated - I/O (I/o - mapped - I/O)

(iii) Memory mapped I/O

IOP :-

- This config uses the common control signals and address space but different BUSES for both memory and I/O.
- It is an expensive design.
- It is used in high performance computer design.

Isolated - I/O :-

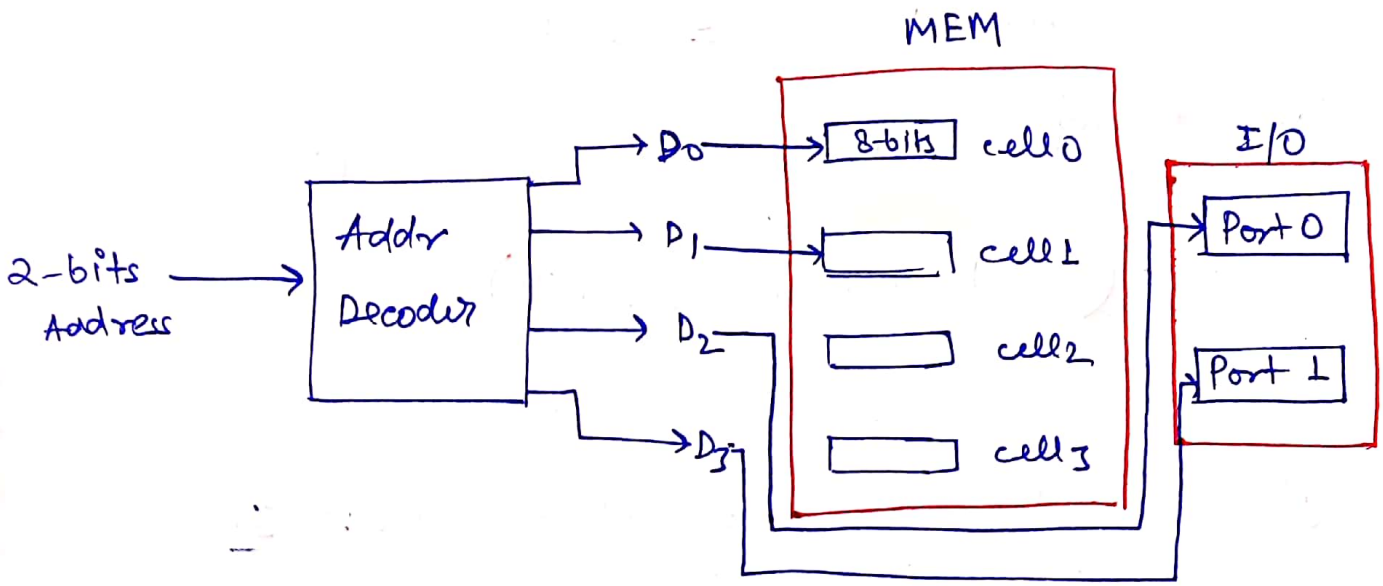
- This config uses the common BUS and common address but different control signals. for both Mem & I/O.

- $IO/\overline{M}$ pin is used to implement the isolated I/O design.
- It is used in general purpose computer design.

$IO/\overline{M}$	$\overline{RD}$	$\overline{WR}$	control signal	Instruction
0	0	1	$\overline{MEM\ RD}$	LOAD
0	1	0	$\overline{MEM\ WR}$	STORE
1	0	1	$\overline{IO\ RD}$	IN
1	1	0	$\overline{IO\ WR}$	OUT

Memory - mapped- I/O :-

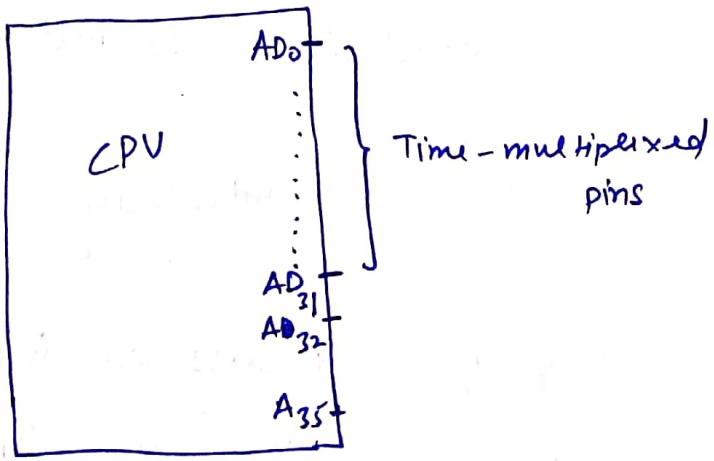
- This config uses the common BUS and control signals but unique address space for both memory and I/O.
- In this config Memory address space is shared to I/O ports so complete memory space is not is used.



MEM add : 00
01

I/O add : 10
11

Q. Consider a hypothetical CPU which supports 32-time-multiplexed pins used to carry the address & data, and 4-address pins. What is the word length of the CPU, and how much main memory is possible in computer & also report the interfacing details.



so, data BUS size = 32-bits

(a) and data word size = 32-bits

(word-length = 32-bits)

(b) Total number of address pins = 36

so, 36-pins

↓

2^{36} addresses

↓

$2^6 \times 2^{30}$ cells

↓

64 G cells

so, byte addressable memory size = 64G × 8 bytes

(= 64GB)

(c) cell size = 8-bits

word size = 32-bits

so, cell interfacing = $\frac{32}{8}$ = (4)

so, 4 cells are interfaced for word-wise accessing

Q. Consider a hypothetical CPU which supports word addressable memory. CPU contains 32-time multi-plexed pins to carry the address and data, And 4 more address pins. What is the word-length of the CPU and how much main memory of computer is required in (Bytes). Also reports interfacing details.

$$\text{word-size} = \underset{(AD_0 - AD_{31})}{\underset{|}{\text{data-lines}}} \underset{(DL's)}{} = (32\text{-bits})$$

AL's $\longrightarrow$ 36 $\longrightarrow$ 2^{36} cells

$\downarrow$

64 G cells

cell size = word size (word addressable memory)

cell size = 32-bits

so, memory size = no. of cells × cell size

$= 64\,G \times 32$

$= (256\,GB)$

{ storage sequence : word-wise
Accessing sequence : word-wise }

Q. Consider 32-bit hypothetical CPU which supports 64KB memory space. Computer design with memory-mapped I/O in which when two MSB bits of address is 1 then assign it to I/O ports and remaing address are used by membory. How many port addresses and memory addresses are possible

word size = 32-bits

memory size = 64KB
- $(2^{16}$ cells) — total addresses
- (8-bits)

Memory - mapped - I/O :-

A_{15} A_{14} A_{13} . . . . . A_0

A_{15} A_{14} — 2 MSB's

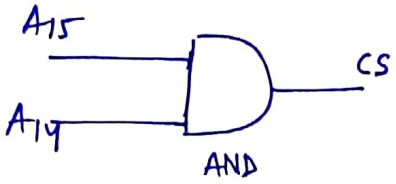
0 0
0 1
1 0 } MEM

I/O — 1 1

Port addresses = 1×2^{14}

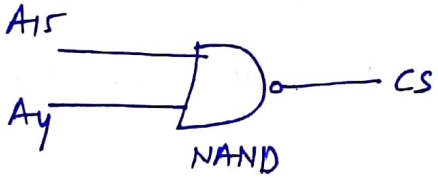
[11 000----(14)0's
  11 111----(14)1's]

Chip select Logic:



Memory addresses = 3×2^{14}

chip select logic :

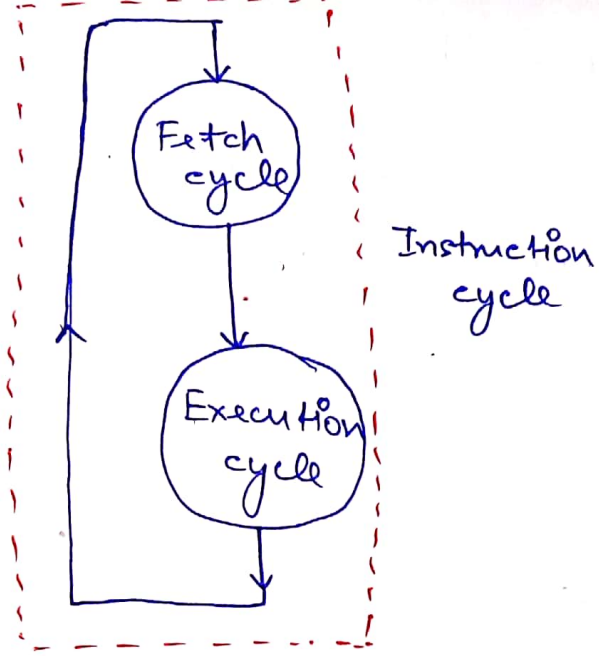


Instruction Cycle :-

- It describes the execution sequence of a program.
- It contain two sub-cycles :

Fetch cycle

Execution cycle



Fetch Cycle:-

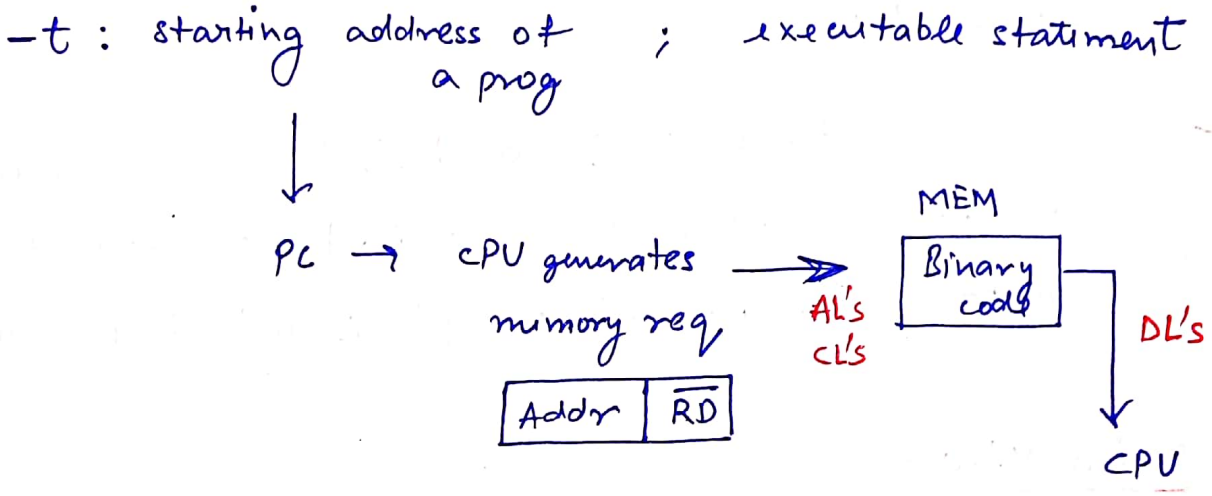
- Objective of this cycle is instruction fetch.
- In this process, CPU fetch the instruction from the memory based on PC (program counter).
- PC is mandotory register in the CPU, used to hold the instruction address.
- starting address of the program is supplied by the user and the next instruction is automatically generated by the machine by increment

$$PC \leftarrow PC + \text{step-size}.$$

- Step size depends on the word length of the CPU.

- Finally PC hold the address of a next instruction during the execution of current instruction.

Instruction Fetch –



PC ← PC + step-size

Analysis -

- when the CPU supports fixed length instructions, where (instruction size is equal to word-size), then during the fetch cycle complete instruction is fetched from the memory called as Instruction fetch.

- when the CPU supports variable length instruction where (optcode size = word size) the during the

fetch cycle, optcode is fetched known as optcode fetch, after decoding the optcode CPU fetches the remaining part of instruction from memory.

- when (optcode size < word-size), then during the fetch cycle one word information is accessed from the memory later CPU fetch the optcode from the word from its internal storage using local BUSES.

- when (optcode size > word-size), the multiple memory refrences are required to fetch the optode itself. This concept is not in practice.

Design View -

CPU : 8085 MP

word-size : 8-bits

address size : 16-bits

opcode size : 8-bits

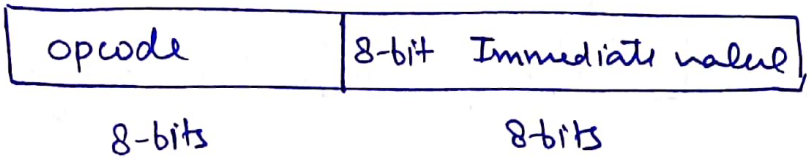
Instruction Design -

① 1 Byte (word) Instruction

opcode
8-bits

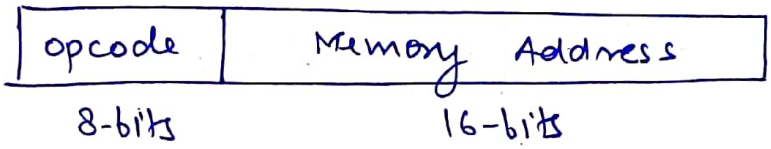
eg : SIM, RIM

② 2 - Byte (word) Instructions



eg: MVI 23H

③ 3-Byte (word) Instructions

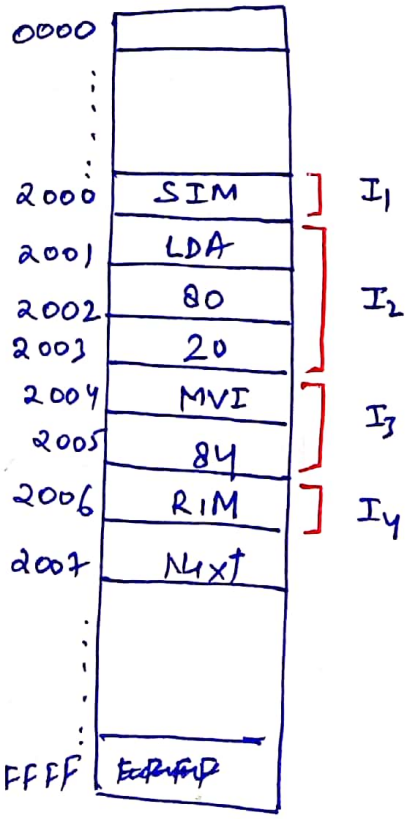


eg:- LDA [2040]

User View :-

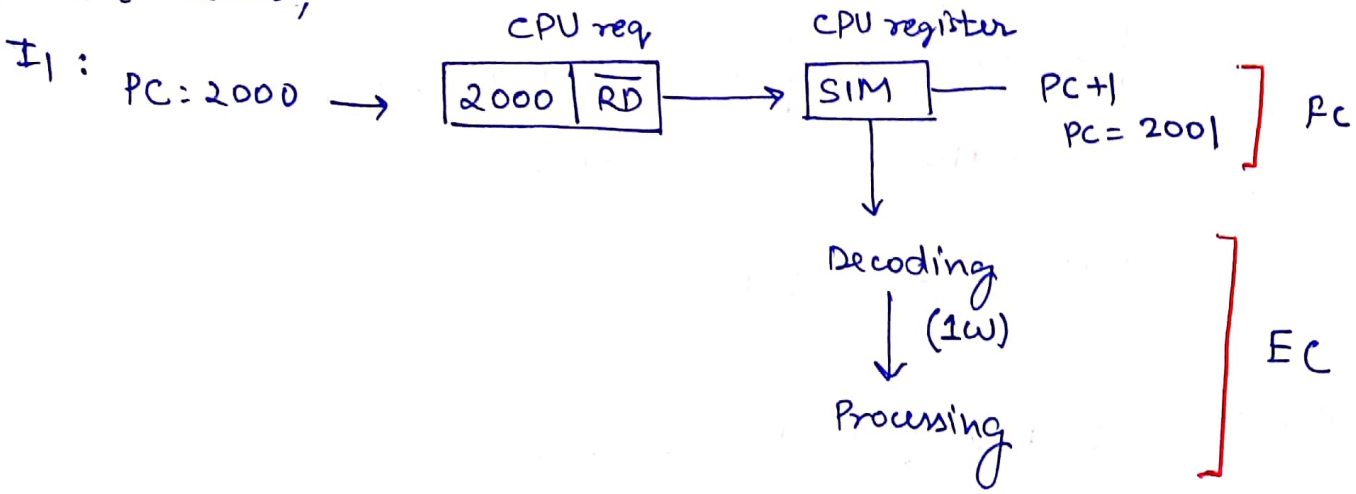
```
code -   org 2000;
         I1 : SIM ;
         I2 : LDA [2000];
         I3 : MVI 84H
         I4 : RIM
```

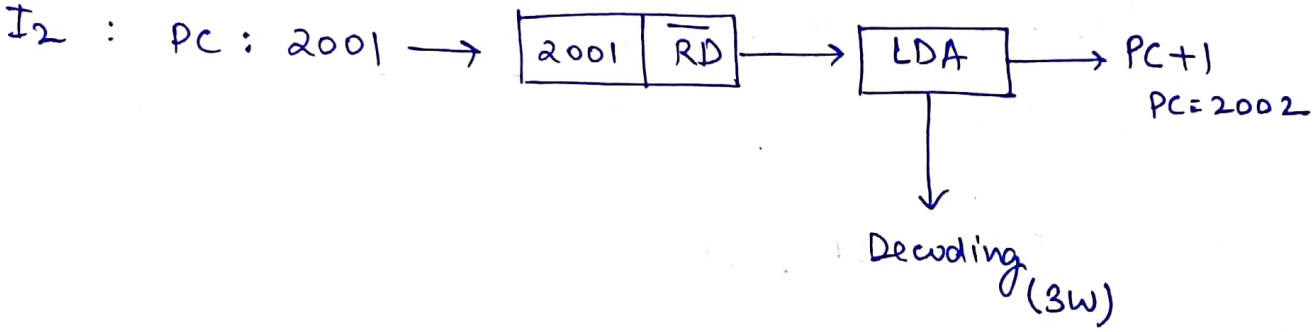
Storage :-

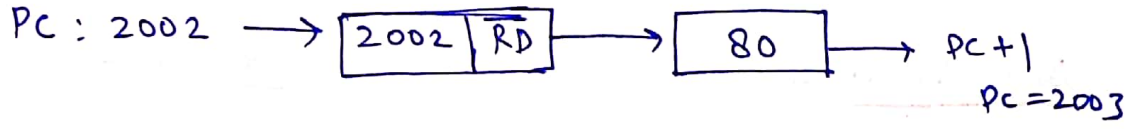


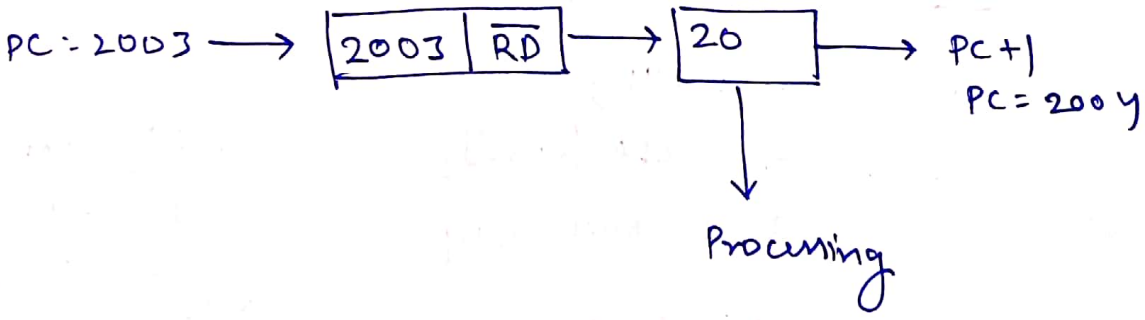
Execution :-

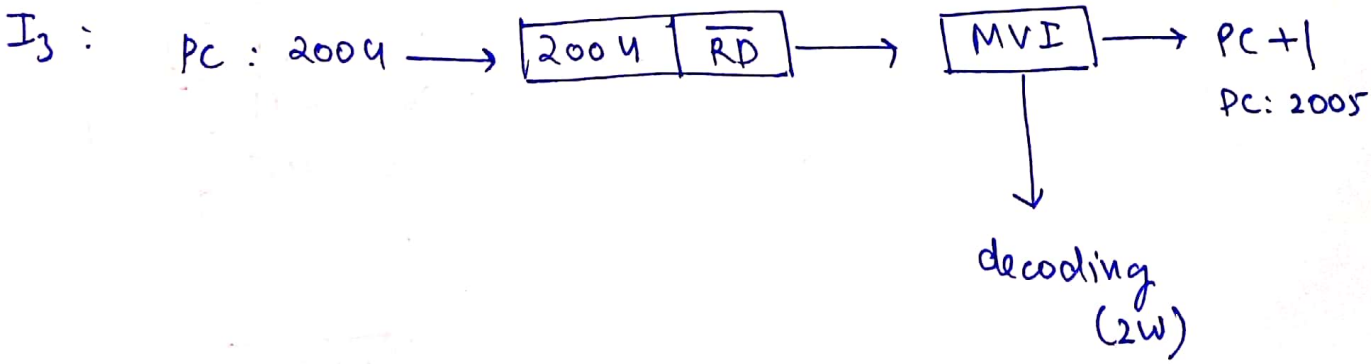
-t 2000;

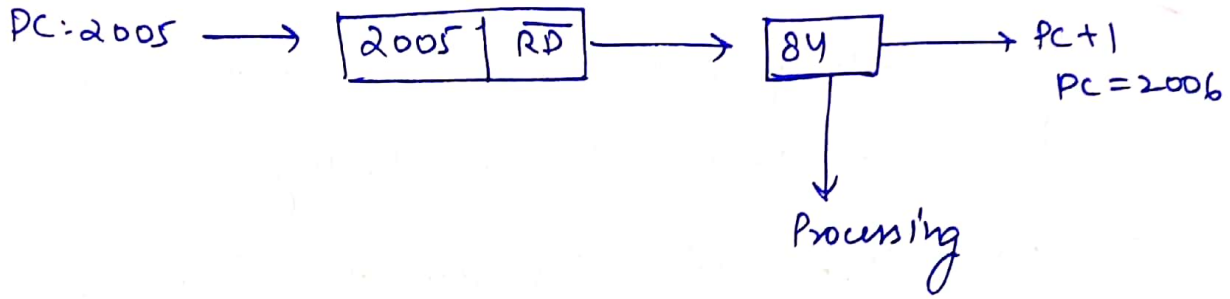


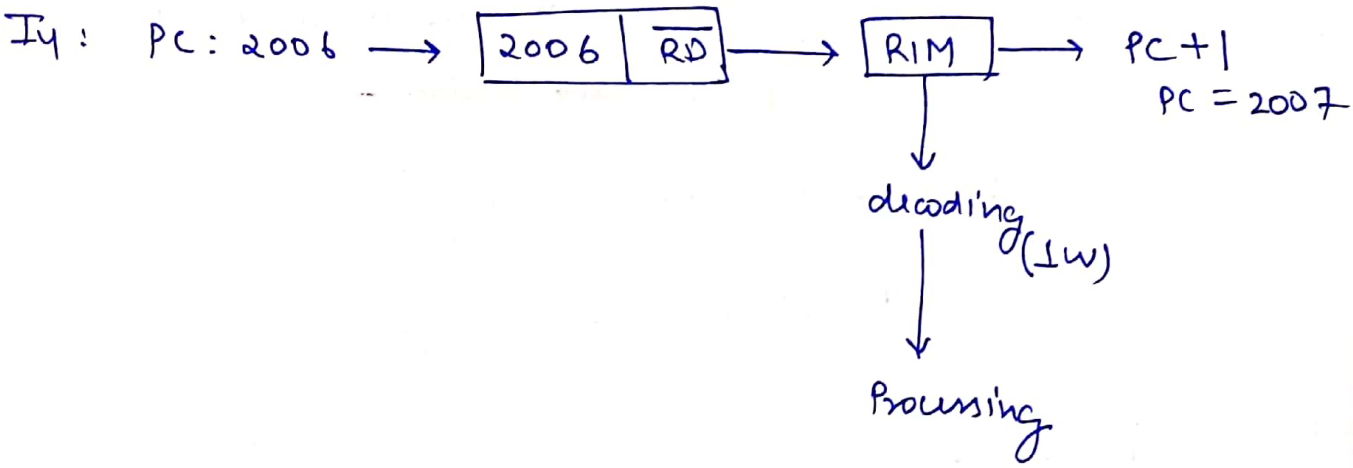












so, Instruction cycle -

	FC	EC	
	IF	Decoding	Execution
I_1	1MR	—	?
I_2	1MR	2MR	?
I_3	1MR	1MR	?
I_4	1MR	—	?

Q. Consider 32-bit hypothetical CPU used to execute the following program code. Program is stored in the memory with a starting address of $(1000)_{10}$ onwards. During the execution of a I_5 instruction what could be the value present in the PC when the system supports ~~Byte codes~~ –

(a) Byte addressable memory

(b) Word addressable memory

Program –

Instruction	Size in words
I_1	2
I_2	2
I_3	1
I_4	2
I_5	1
I_6	2
I_7	1

(a) Byte addressable memory {cell size = 8-bits}

storage

I_1 : 1000 — 1007

I_2 : 1008 — 1015

I_3 : 1016 — 1019

I_4 : 1020 — 1027

I_5 : 1028 — 1031

I_6 : 1032 — 1039

I_7 : 1040 — 1043

1044

Since during execution of a particular Instⁿ PC hold address of next instruction so,

during Execn I_5, PC hold value (addr of I_6) i.e 1032

(b) Word addressable memory {cell size = 32-bits (1 word)}

I_1 : 1000 — 1001

I_2 : 1002 — 1003

I_3 : 1004

I_4 : 1005 — 1006

I_5 : 1007

I_6 : 1008 — 1009

I_7 : 1010

1011

during Execn I_5, PC holds address of I_6 i.e 1008

Q. Consider 24-bit hypothetical CPU which supports 1 word long instructions. Program is stored in the memory with a starting address of $(300)_{10}$. Which one of the following is a valid PC.

(a) 400 (b) 500

(c) 600 (d) 700

word-size = 24-bits

Instruction-size = 1 word = 24-bits

memory is (byte-addressable)

so, cell size = 8-bits

Instruction takes $24/8$ = 3 cells for storage

so,

PC

I_1 : 300 — 302

I_2 : 303 — 305

I_3 : 306 — 308

I_4 : 309 — 311

⋮

I_n : 600

so, PC showed by multiple of 3 onwards 300.

Q. Consider 16-bit hypothetical CPU which supports 3 categories of instruction with there respective size of 16-bytes, 8 bytes, 4 bytes. Program contain 40, 20 and 50 instruction respectively. Program is stored in the word addressable memory. How many cells of storage space required.

cell size = 16-bits (Word addressable)

So, Instruction in word format

I_1 → 16-bytes → 8 words

I_2 → 8-bytes → 4 words

I_3 → 4-bytes → 2 words

So,

Total no. of cells required $= (I_1 \times 40) + (I_2 \times 20) + (I_3 \times 50)$

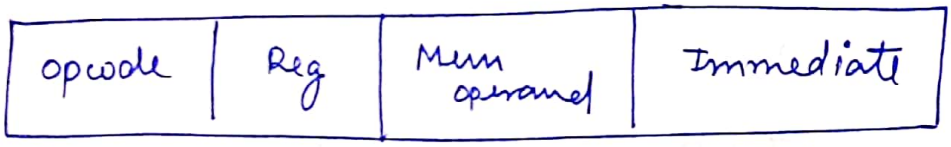
$= 500$ cells

Q. Consider a hypothetical CPU which supports instruction with opcode ~~width~~, register operand, memory operand and 17-bit immediate operand fields.

System supports 200 instructions, 24 registers and 64 KB memory space.

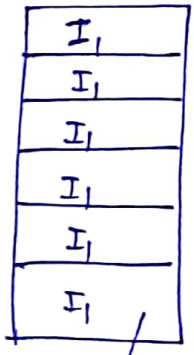
Program contains 100 instructions, How much storage space is required in bytes to store the program?

Instruction Design:



$\log_2 200 \cong 8$-bits $\log_2 24 \cong 5$-bits $\log_2 64K = 16$-bits 17-bits

So, Instruction size = 46-bits. $\cong$ 6 bytes



2-bits are left but I_2 will be stored in next cell

So, Prog size = 100 Instruction

= 100 × 6 cells

= 600 cells

= 600 Bytes

Execution Cycle:-

- Objective of this cycle is processing of a currently fetched instruction from the memory.
- Instruction format is required to process the instruction

- Instruction format describes the layout design (internal structure) of an instruction.

- Instruction format is classified into 5 types - based on the CPU organization -

- CPU organization is classified into 3 types based on the availability of ALU operands (data paths)

  (i) stack CPU
  (ii) Accumulator CPU
  (iii) General purpose register CPU

  } classification of ISA machines (Instruction - set architecture)

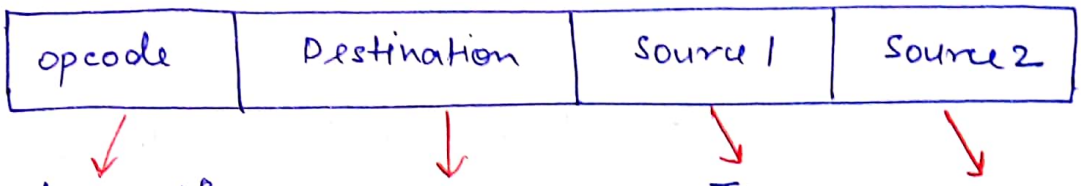
Stack CPU :-

- In the CPU organization ALU operations are performed only on the stack data so both of the operands are always required in the stack.

- After the processing, result is also kept into the stack.

- In the stack TOS (top-of-stack) become default location so its address is not required in the format.

- compatible instruction format is :

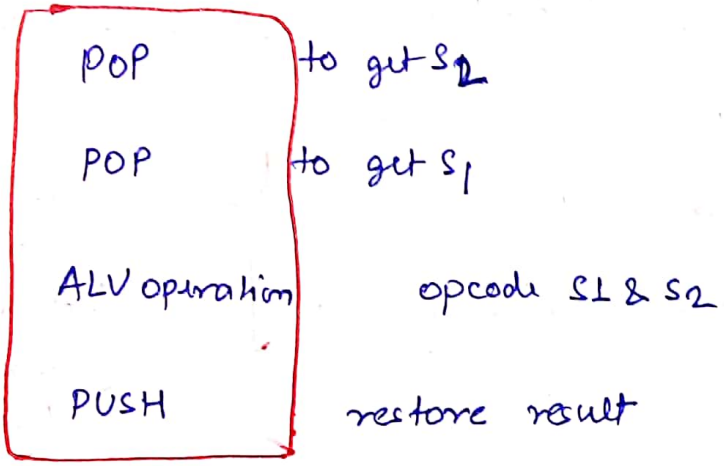


Type of operation (ALU operation) data manipulation

TOS

TOS

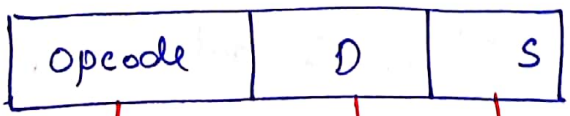
TOS



(TOS ← TOS ⊕ TOS)

ALV operation

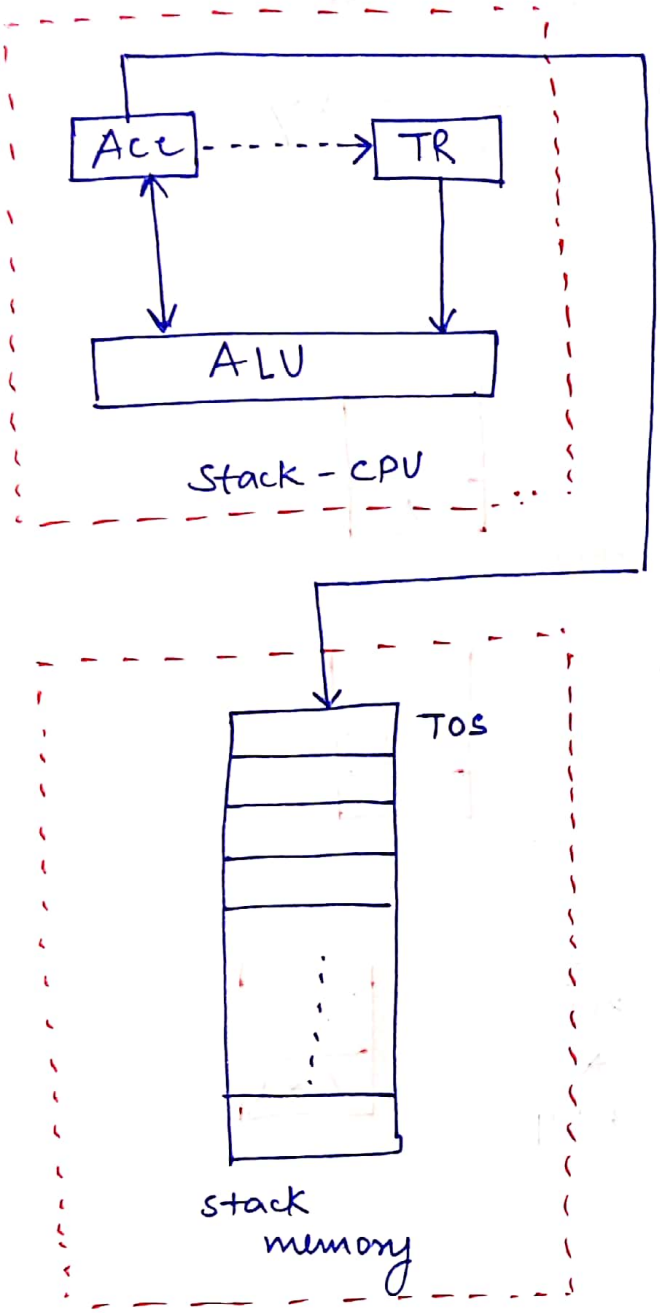
- In this CPU organization data-transfer ~~organi-~~ operation (PUSH & POP) are designed as one address instruction.



PUSH TOS Mem

POP Mem TOS

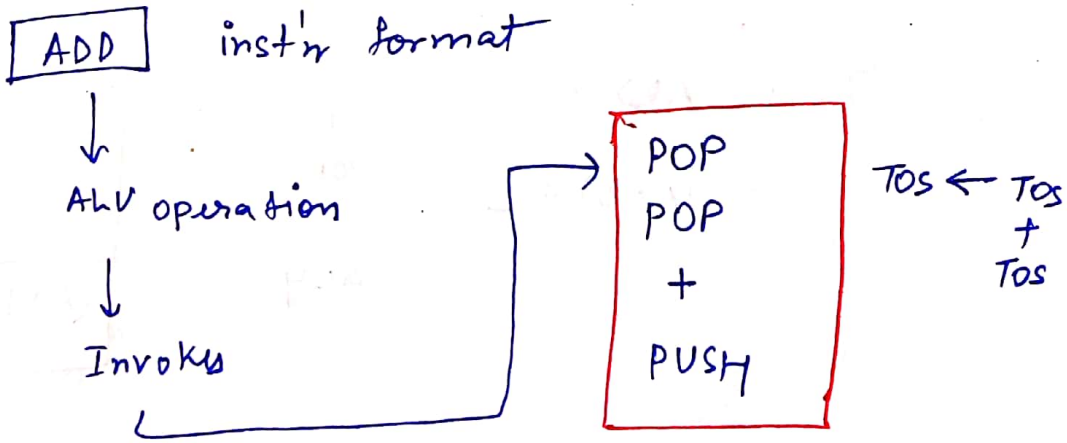
* TOS field is not required to mentioned



* ACC is the accumulator as it provide input & store output of CPU.

* TR is the temporary register

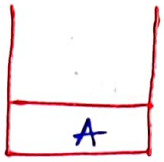
- ADD ; operands are in stack memory -



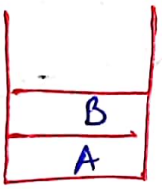
Eg:- $(A * B) + C$ and variables are in memory not in stack memory then,

Codes

I_1 : PUSH A ;

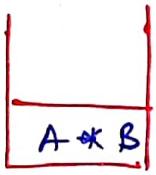


I_2 : PUSH B ;

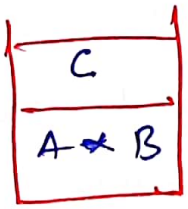


I_3 : MUL ; (A*B)

POP
POP
*
PUSH



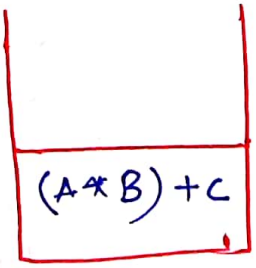
I_4 : PUSH C ;



I_5 : ADD ;

$(A * B) + C$

POP
POP
+
PUSH



Q: Consider the following statment running on a stack-CPU. CPU supports 8-bit opcode & 4GB mem space.

$$X = (A+B) * (C-D)$$

variables are in the main memory.

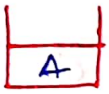
(a) How many M/c instructions are required?

(b) How much space is required in bytes to store the program.

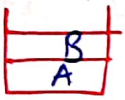
$$X = (A+B) * (C-D)$$

Code

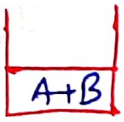
I_1 : PUSH A;



I_2 : PUSH B;



I_3 : ADD;



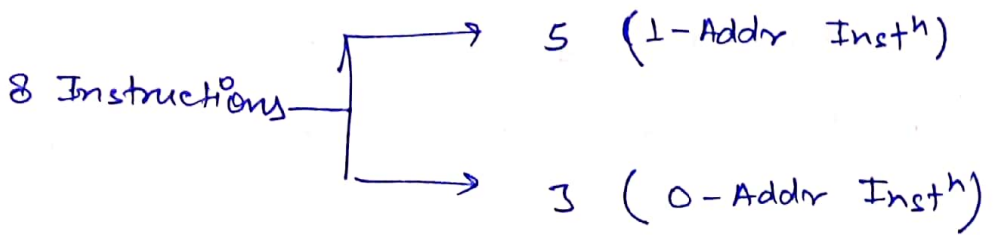
I_4 : PUSH C;

I_5 : PUSH D;

I_6 : SUB;

I_7 : MUL;

I_8 : POP X;



0 - Addr Instⁿ : opcode

8-bits (1B)

1 - Addr Instⁿ : opcode | Addr

8-bits $\log_2 49$ = 32-bits

40-bits (5B)

Prog size = $(5 \times 5B) + (3 \times 1B)$

= (28 Bytes)

Q. Consider the following code; executed on a stack CPU (Assume stack is empty)

I_1 : PUSH b

I_2 : PUSH X

I_3 : ADD

I_4 : POP C

I_5 : PUSH C

I_6 : PUSH y

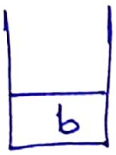
I_7 : ADD

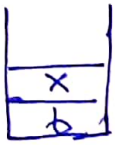
I_8 : PUSH C

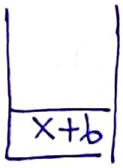
I_9 : SUB

I_{10} : POP Z

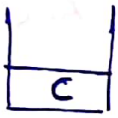
What is the o/p of code ?

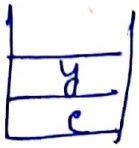
I_1 : 

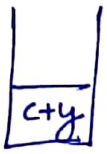
I_2 : 

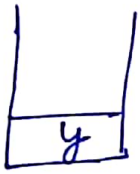
I_3 : 

I_4 : $c = b + x$ 

I_5 : 

I_6 : 

I_7 : $c+y$ 

I_9 : $c+y-c$ 

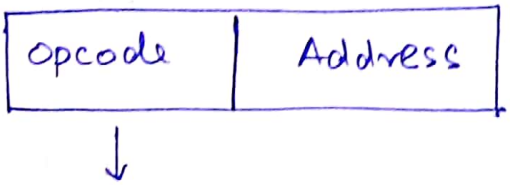
I_{10} = $z = y$ 

(output : z contains y
stack empty)

Accumulator CPU :-

- In this organization one of the ALU operand is always required in register (accumulator). And the other operand is present either in the register or in memory.
- After the data processing result is placed into a accumulator.

- In the CPU design, Acc become a default location • Compatible format is :



1- Addr Inst^{h} format

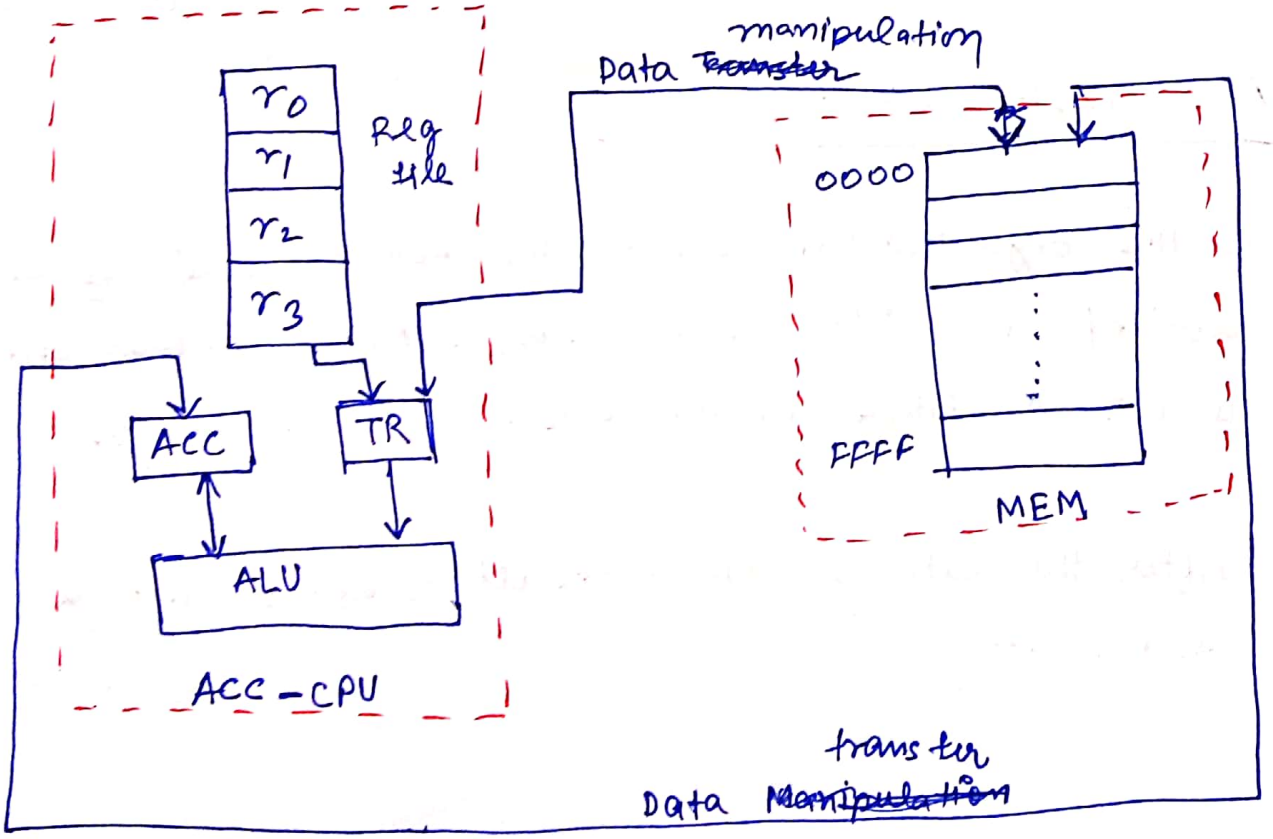
↓

Type of operation

↓

	S	D	
Data transfer	Mem	Acc	$\overline{MEMRD}$ (LOAD)
	Acc	Mem	$\overline{MEMWR}$ (STORE)

	S_1	S_2	D
Data manipulation	Acc	reg/ Mem	Acc



Inst^{n} : ADD [2000] ; Mem

↓

ADD [2000] ; ACC-CPU

PC: sequential Address

} FC

| ADD | [2000] | Inst^{n} format

↓

ALU op^{n}

↓

+

↓

(ACC ← ACC + M[2000])

} EC

Eg : (A*B) + C , variables are in the main memory.

Code :-

I_1 : LOAD A ; ACC ← M[A]

I_2 : MUL B ; ACC ← ACC * M[B]

I_3 : ADD C ; ACC ← ACC + M[C]

Q. Consider the following statements running on a Acc - CPU without Internal Data forwarding operation.

$$X = (A+B) * (C+D)$$

variables are in the memory.

(a) How many M/c instructions are required

(b) How many memory spills required.

* Spill means Intermediate result is stored into a memory location

(a)

I_1 : LOAD A ; Acc ← M[A]

I_2 : ADD B ; Acc ← Acc + M[B]

{Spill} I_3 : STORE T ; M[T] ← Acc

I_4 : LOAD C ; Acc ← M[C]

I_5 : ADD D ; Acc ← Acc + M[D]

I_6 : MUL T ; Acc ← Acc * M[T]

I_7 : STORE X ; M[X] ← Acc

(b) 1 spill is required.

Q. X = (A+B) / (C+D) variables are in the main memory.

To optimize the code, follow the order of evaluation i.e evaluate (C+D) first.

I_1 : LOAD C

I_2 : ADD D

I_3 : STORE T {Spill}

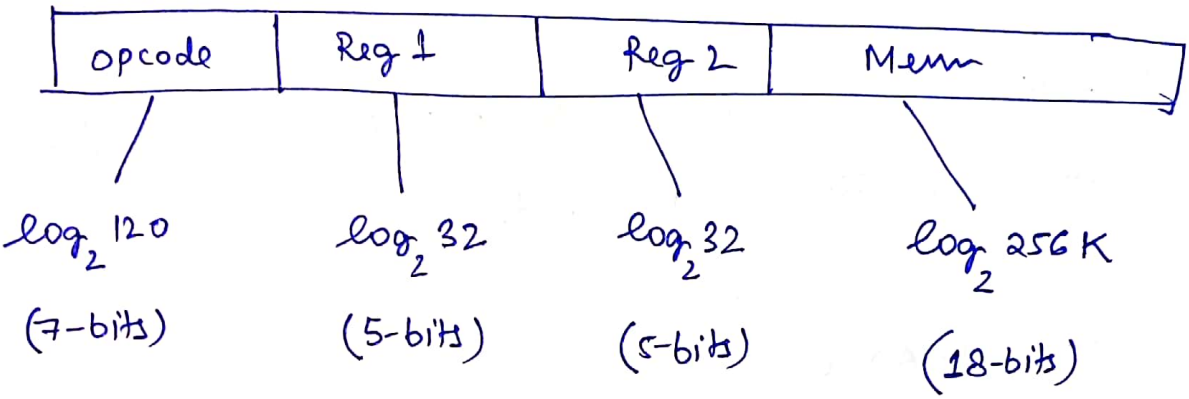
I_4 : LOAD A

I_5 : ADD B

I_6 : DIV T

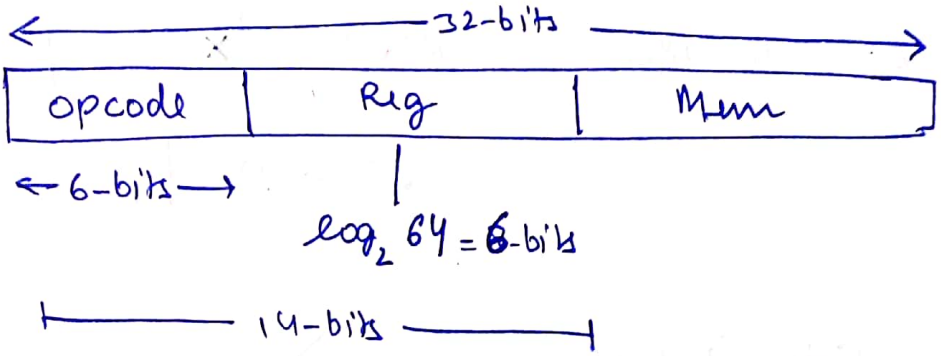
I_7 : STORE X

Q. Consider a hypothetical CPU which supports instructions with 2 reg operands and 1 mem operands. CPU supports 120 inst^n, 32 register and 256 KB memory space. How many bits are required to encode the instruction (instruction-size).



Instruction size = 35-bits

Q. Consider a 32-bit hypothetical CPU which support instructions with register operand and memory operand field. 1 word long instruction is placed in the word addressable memory. Register file size = 64. CPU supports 6-bits opcode. How much main memory possible in bytes.



So, Mem address size = 32 − 14 = 20-bits

So, number of cells = 2^{20}

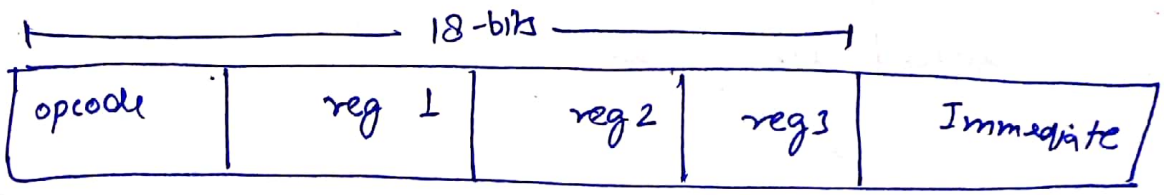
Cell size = 1 word = 32-bits

So, memory size = $2^{20} \times 32$ bits

$= 2^{20} \times 4$ Bytes

$= 2^{22}$ (4 MB)

9. Consider a 32-bit hypothetical CPU which supports instruction with 3 reg operands and immediate operand field. One word Instruction is placed in 256 MW memory. CPU supports 64 Instructions and 12 registers. What is the largest unsigned constant possible in the instructions.



64 instruction $= 256 \times 2^{20} \times$ word

$= 2^{28} \times$ word

Immediate field size $= 32 - 18$

$= 14$-bits

so, range of unsigned number $= 0$ to $2^{n}-1$

$= 0$ to $2^{14}-1$

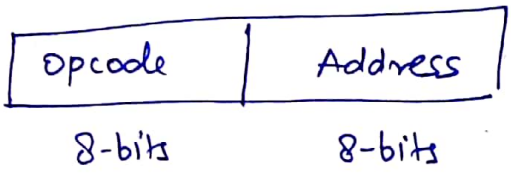
(Largest unsigned number $= 2^{14}-1$)

"Expand Opcode technique" is required in the fixed length instruction CPU design, to implement the different instruction formats. In this technique address field will be appended to free opcodes to generate the derived opcode therefore derived opcode length is increases. (Requirement is atleast one opcode must be free after allocation of a primitive inst^{n} to report derived inst^{n}).

Analysis :-

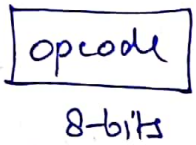
① variable length Instⁿ supported CPU design :-

- 1-addr Instⁿ format



Is = 16-bits

- 0-addr Instⁿ format

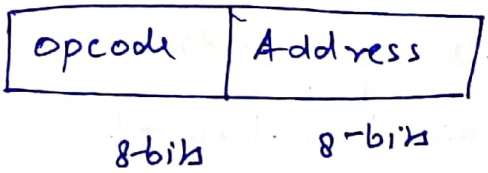


Is = 8-bits

{no, need of Expand-opcode techniqne}

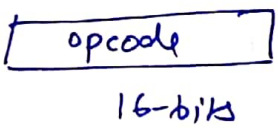
② ~~variable~~ fixed length Instⁿ supported CPV :-

- 1-addr Instⁿ format



Is = 16-bits

- 0-addr Instⁿ format



Is: 16-bits

Expand - opcode techniqne :-

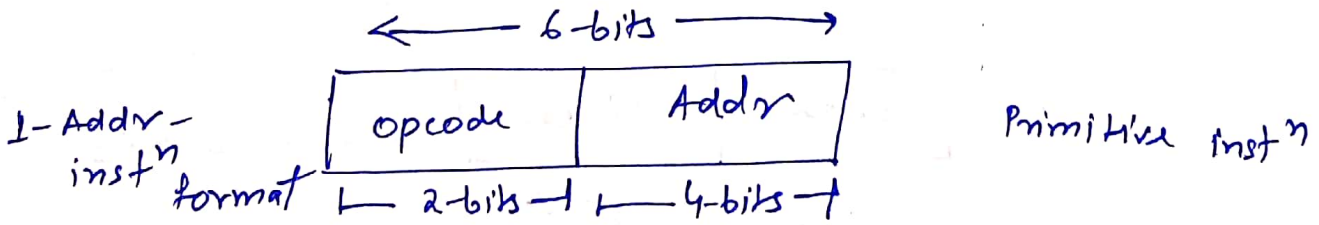
Step-1 :- Identify the primitive Inst^n format in the CPU (smallest opcode Inst^n is primitive Inst^n)

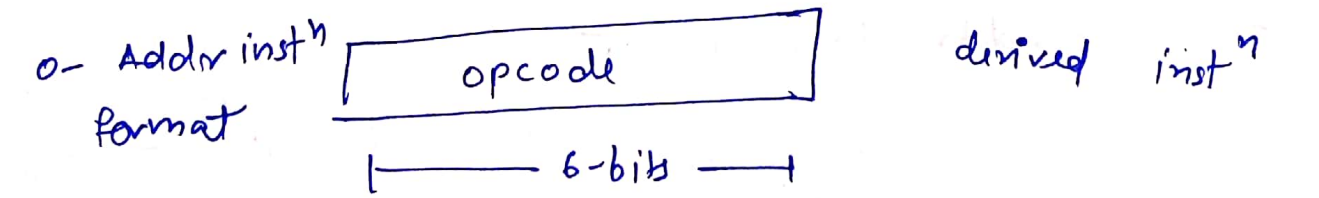
Step-2 :- Identify the no. of Primitive inst^n possible

Step-3 :- Identify the free opcodes after allocating the primivitive inst^n.

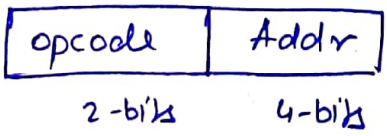
Step-4 :- Calculate the derived opcodes by multiply the free opcodes with a decoded value of the Addr field.

Eg:- CPU supports 6-bit Inst^n & 4-bit Address. If '3' 1-Addr Inst^n are existed then how many 0-addr Inst^n are formulated ?





Step-1



Step-2

operations $= 2^n = 2^2 = 4$

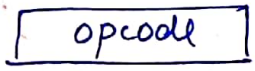
free — 00 Addr

allocated { 01 Addr
10 Addr
11 Addr

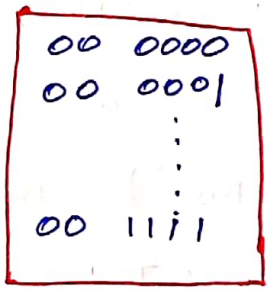
Step-3

Free opcodes $= 4 - 3 = 1$

Step-4



0-Addr Inst^n format

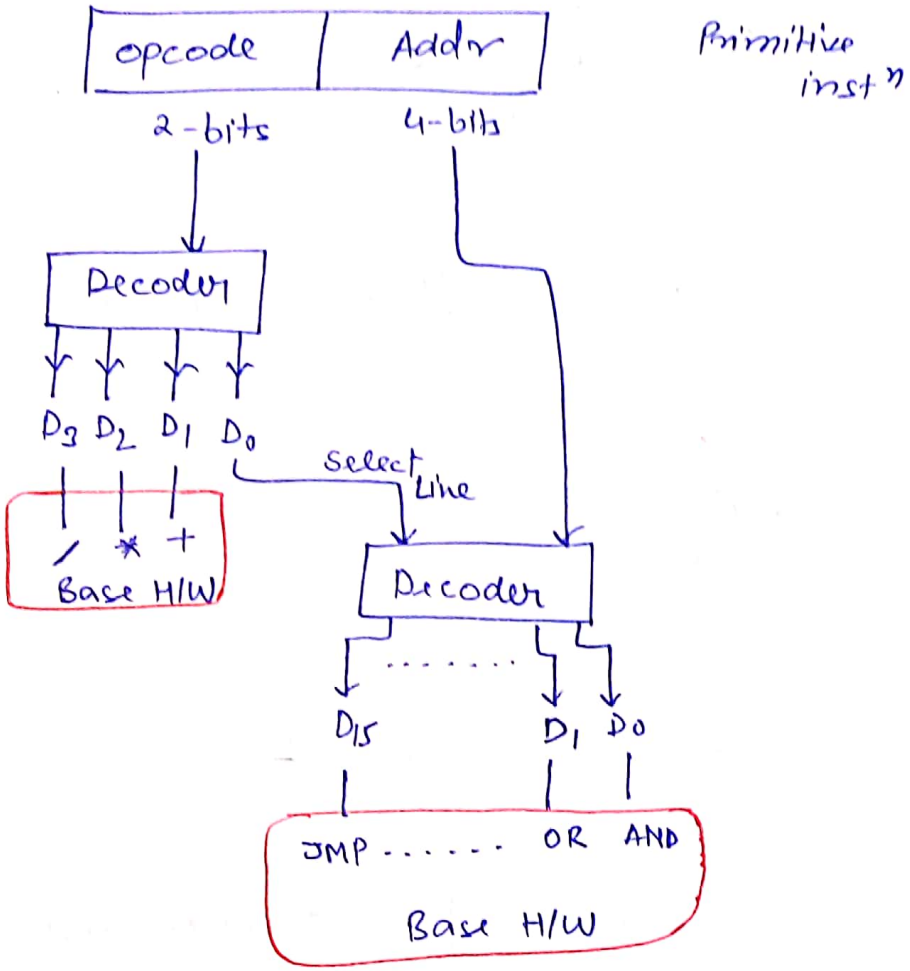


16 opcodes

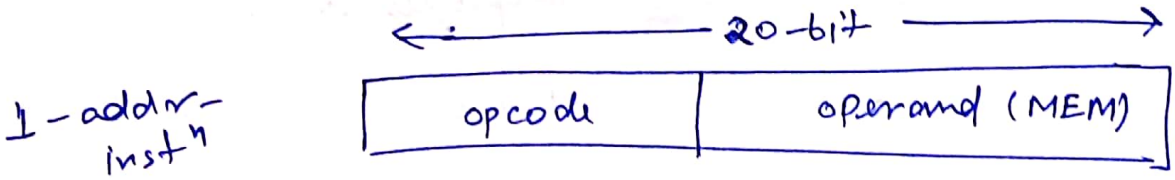
$= \text{free opcodes} \times 2^{\text{Addr size}}$

$= 1 \times 2^4$

$= 16$ derived opcodes



Q. Consider [20-bit hypothetical CPU which support 1 word long instruction placed in 8KW memory. It there exist 126 [address inst^n], then how many 0-addr instn are formulated.



address size = 8K W

= 8K × W

no. of cells = $2^{\log_2 8K} = 2^{13}$

So, opcode size = 20 − 13 = 7 bits

So, Total no. of opⁿ = 2^7 = 128

and allocated opⁿ = 126

Free opcodes = 128 − 126 = 2 opcodes

So, 0 − Addr − Instⁿ = 2×2^{13}

= (2^{14})