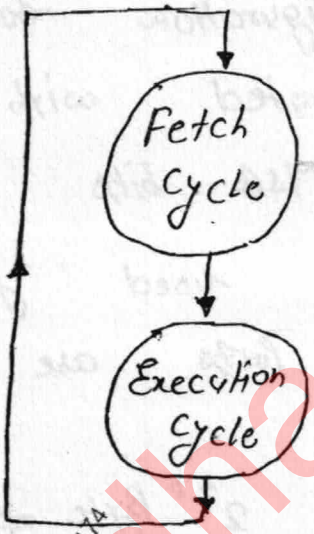
# "Instruction Cycle" :-

This concept describes the execution sequence of a Program. It contains two subcycles :

1.) fetch cycle

2.) Execution cycle




## Fetch Cycle :-

* objective of a fetch cycle is instruction fetch.
* In this Process CPU, reads the instruction from the memory, based on a PC. (Program counter)
* PC is a compulsory register in the CPU, used to hold the instruction address.
* Starting address of a Program is supplied by the user and the Next instruction address is automatically generated by the CPU, by increament the PC with a step size.
* "Step size" is a fixed constant, depends on the word length of a CPU.
* finally PC holds the address of a Next instruction during the execution of current inst".

Eg:- for 16 bit Processor

↳ 2 [8 bit] cells interfacing

Step size = 2.

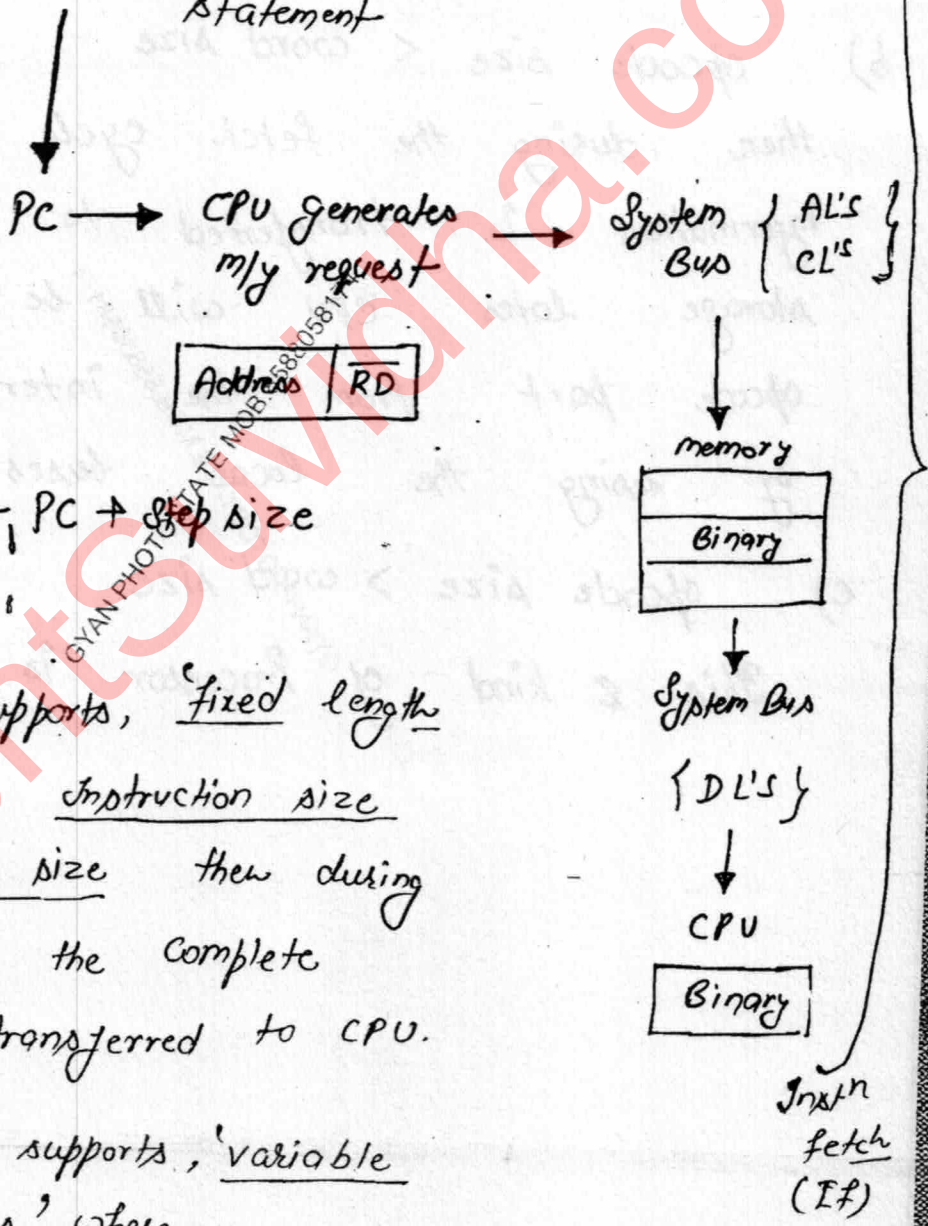
and for 8 bit Processor, step size = 1.

eg:-

-t : starting address of Program ← Executable statement

{trace}

↳ Line by line (Instn by Instn) execution




Analysis :-

★ when the CPU supports, 'fixed length instructions' where instruction size equal to word size then during the fetch cycle, the complete instruction is transferred to CPU.

★ when the CPU supports, 'variable length instructions' where

a) opcode = word size then during the fetch cycle, only the opcode is fetched from the m/y.

after the decoding the fetched opcode, cpu realizes the actual length of a instruction therefore CPU reads the remaining part of instruction from the memory based on the PC. when Instruction size is greater than word size.

b) opcode size < word size

then during the fetch cycle, one word information is transferred to cpu internal storage. later CPU will be accessing the opcode part from the internal storage by using the local buses.

c) opcode size > word size

This is kind of Processor is not in Practice.

Date 10/08/2017

# 8085 µp (8 bit CPU)

* word size = 8 bit
* Address size = 16 bit

{ $A_{15} - A_8$ $AD_7 - AD_0$ }

* opcode size = 8 bit

{ Opcode size = word size }

## :- Instruction Design :-

(1) 1 Byte (word) inst$^n$ :-

| opcode |
|---|

eg: SIM (Send interrupt mask)
RIM etc. ( Recieve " )

(2) 2 Byte (word) Inst$^n$ :-

| opcode | 8 bit Immediate value |
|---|---|
| 8 bit | 8 bit |

eg: MVI 28H

(3) 3 Byte (word) Inst$^n$ :-

| Opcode | memory address |
|---|---|
| 8 bit | 16 bit |

eg: LDA [2040]

:- Operational State :-

Code: org 1000

1000: $I_1$: SIM

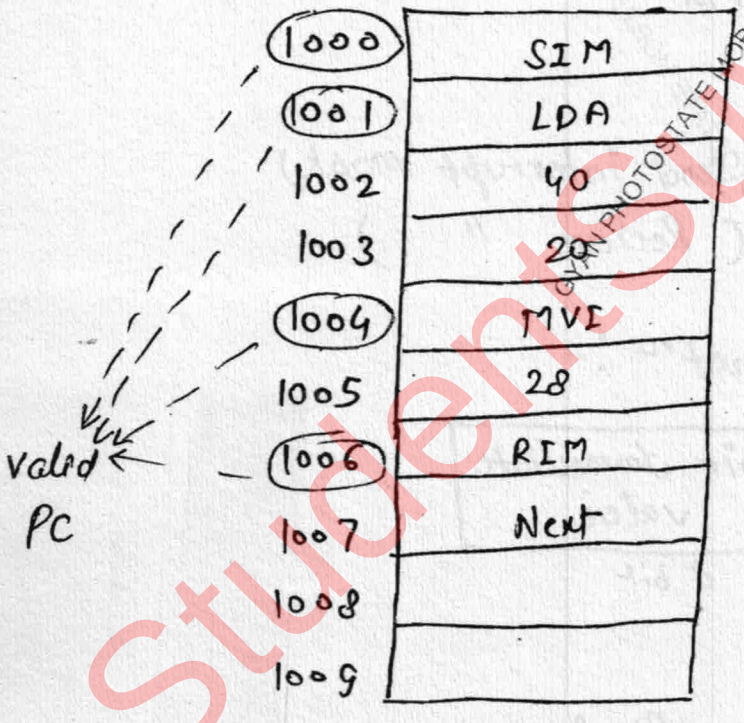
$I_2$: LDA [2040]

$I_3$: MVI A 28H

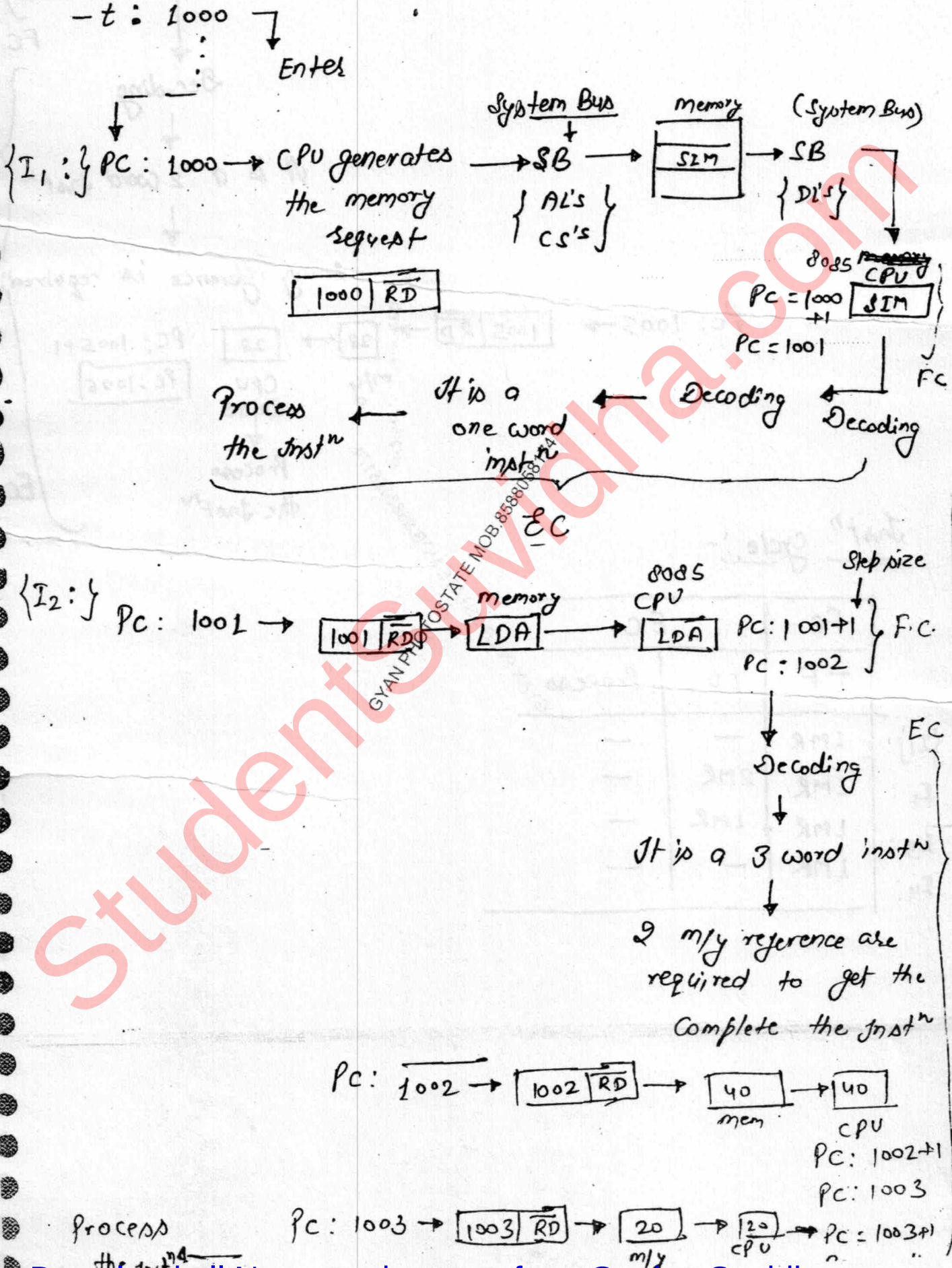
$I_4$: RIM

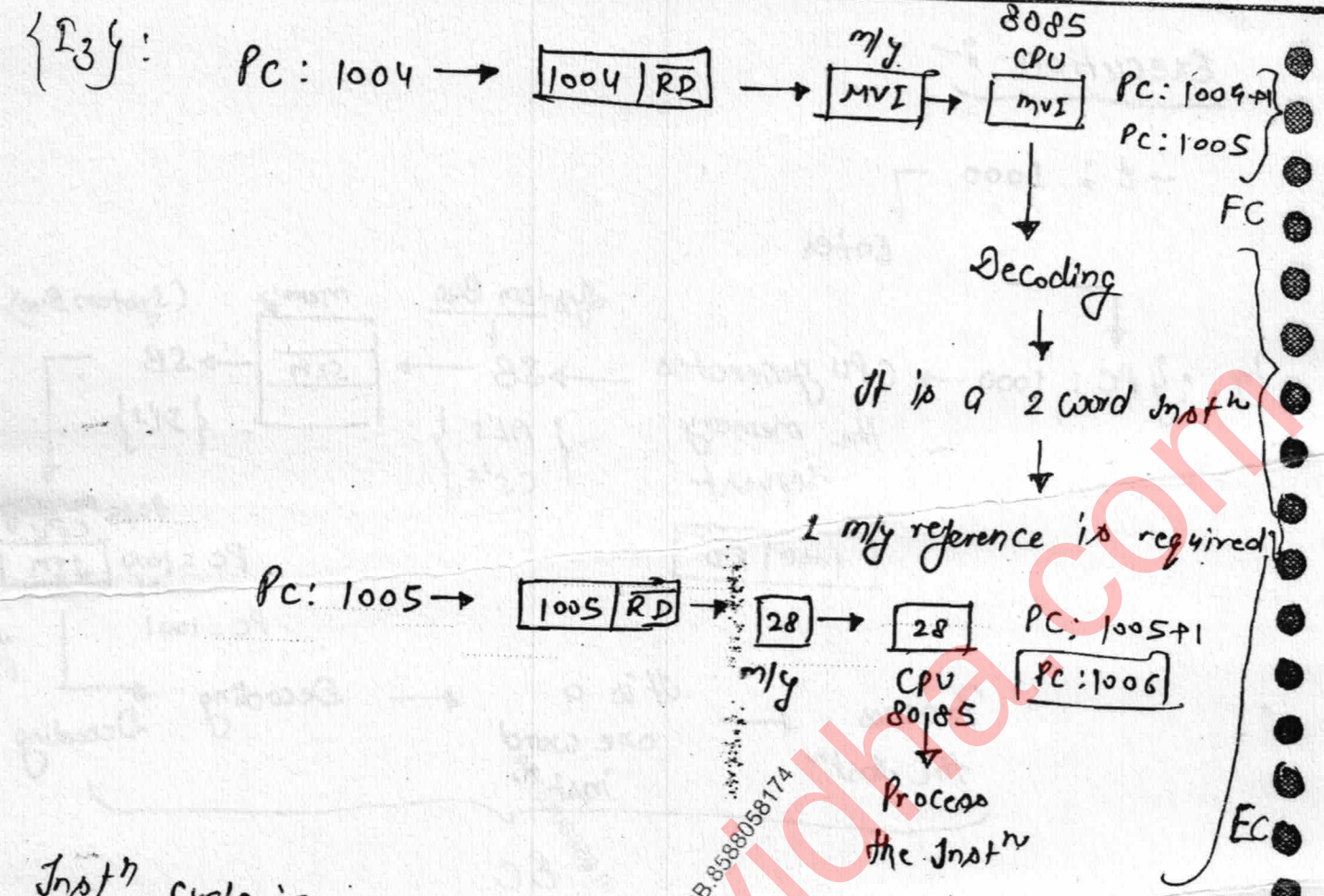
⋮ ⋮

Memory Storage :-



## Execution :-






Instn cycle :-

| | FC | EC | |
|---|---|---|---|
| | IF | ID | Process |
| {I1}: | 1MR | — | — |
| I2 : | 1MR | 2MR | — |
| I3 : | 1MR | 1MR | — |
| I4 : | 1MR | — | — |

# 8086 µP (16 bit CPU)

★ word size = 16 bit

★ Opcode size = 8 bit

{ Opcode size < word size }

## Execution :—

-t : 1000 ↓ Enter

{I/f: PC: 1000 → CPU generates m/y request → SB {AL's CS's} → 1000 | RD → m/y: SIM, LDA → SB {DI's} → Prefetch Queue (Internal storage) → 8086 CPU: LDA SIM, PC: ~~1000~~ 1000 → 2, PC: 1002 } PC

↓ Read opcode from the Queue

↓ SIM

↓ Decoding

↓ It is 1B Instⁿ

↓ Process the instⁿ

"Parallel" (Pipeline) ⟷

PC: 1002 ← m/y: 40, 20 ← CPU Queue: 2040 LDA ← PC: 1002+2, PC: 1004

↓ Read opcode from Queue

↓ LDA

↓ Decoding

↓ It is 2B Instⁿ → Access 2B from Queue → Process Instⁿ

Que:- Consider a 32 bit hypothetical processor used to execute the following code:

| Instⁿ | Size (in words) |
|---|---|
| $I_1$ | 2 |
| $I_2$ | 3 |
| $I_3$ | 1 |
| $I_4$ | 2 |
| $I_5$ | 1 |
| $I_6$ | 2 |
| $I_7$ | 1 |

Assume that Program stored in the m/y with the starting address 1000 decimal onwards. what could be the value present in the PC during the execution of a $I_6$ instruction.

:- memory storage:-

32 bit:- 4 cell addr. (Interfacing)

| Instⁿ | Size |
|---|---|
| $I_1$ | 2 |
| $I_2$ | 3 |
| $I_3$ | 1 |
| $I_4$ | 2 |
| $I_5$ | 1 |
| $I_6$ | 2 |
| $I_7$ | 1 |

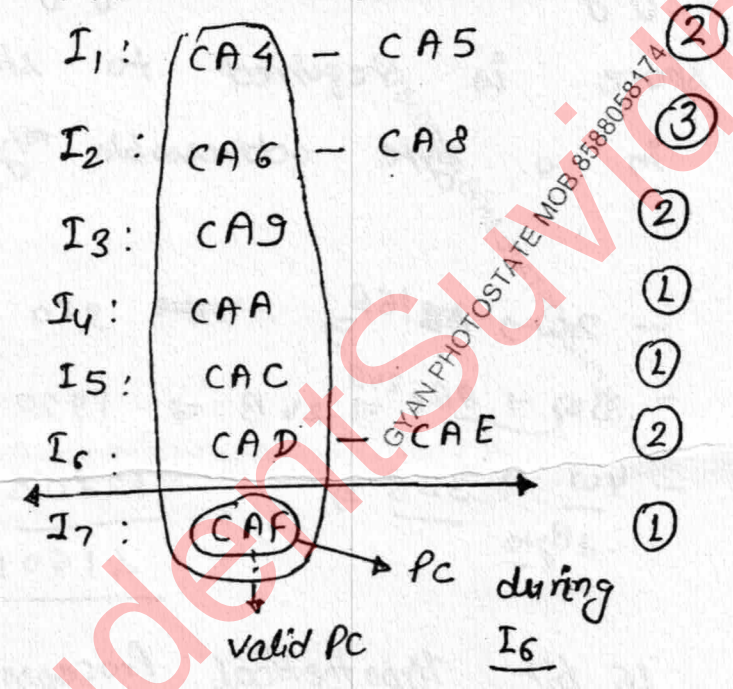
Byte Addressible storage (By default)

$I_1$ :- 1000 – 1007
$I_2$ :- 1008 – 1019
$I_3$ :- 1020 – 1023
$I_4$ :- 1024 – 1031
$I_5$ :- 1032 – 1035
$I_6$ :- 1036 – 1043
$I_7$ :- 1044 – 1047

b) Assume that Program is stored in a word addressible m/y with a starting address $(CA4)_H$ what could be the value present in the PC during the execution of $I_6$ instruction.

$(CA4)_H$ → Starting address

Word Addressable Storage






★ Que:- Consider a Hypothetical CPU which supports 24 bit instructions. Program is stored in the m/y with the starting address of 300 decimal onwards. which one of the following is a valid PC.

24 bit ⇒ 3B (By default m/y is Byte Addressable)

PC: 600 % 3 ⇒ 600

a) 400 b) 500 c) 600 d) 700

Note "valid PC is a multiplicative factor of 3"

~~Byte~~ ~~Add~~. So

600 mod 3 = 0

So 600 valid PC.

Que:- Consider a 64 bit Hypothetical CPU which supports three category of instructions with a respective size of 2w, 3w, and 4w long. Program contain 60 category 1 instn, 80 category 2 instn, and 40 category 3 instn. How much space is required to store the Program in a Byte addressable m/y. (in bytes).

C1 : 60 — 2w → ~~8B~~ 16B ⇒ ~~960~~ 960 B

C2 : 80 — 3w → 8B → 24B ⇒ 24 B ⇒ 1920 B

C3 : 40 — 4w → 32B ⇒ 1280 B

8 Bytes

4160 B

Que:- Consider a 16 bit Hypothetical Processor which supports 8 Byte long instructions. Program contain 100 instructions. Processor uses the word addressible m/y how many word space is required in the m/y to store the program.

⇒ 100 instn ⇒ 8 bytes (1 instn) ⇒ 800 Bytes

4 words

⇒ 100 instn × 4 ⇒ 400 words

# Execution cycle :-

* Objective of a Execution cycle is processing of a currently fetched instruction.
* Instruction format is required in the Execution cycle to identify the opcode information.
* Layout design of a Instruction is called as instruction format.
* Instruction format is divided into five types based on the CPU organization.
* CPU organization is divided into three types' based on the availability of a ALU operands :-
  1) Stack - CPU
  2) Accumulator - CPU
  3) General Register - CPU

## Stack CPU :-

* In this Organization, ALU operations are always performed on a stack data that means Both of the operands are always required in the stack. After the Processing Result is also place into a stack.

* Stack is a part of memory, initialized with a Stack-Pointer (SP) register

* In the stack insert and delete operations are takes place at the same end, name as Top of stack (TOS). So TOS become default.

:- Instruction design :-

OPCode : '0' Address Instⁿ format

↓

Type of operation

↓

ALU operation : $S_1$ TOS, $S_2$ TOS, D TOS

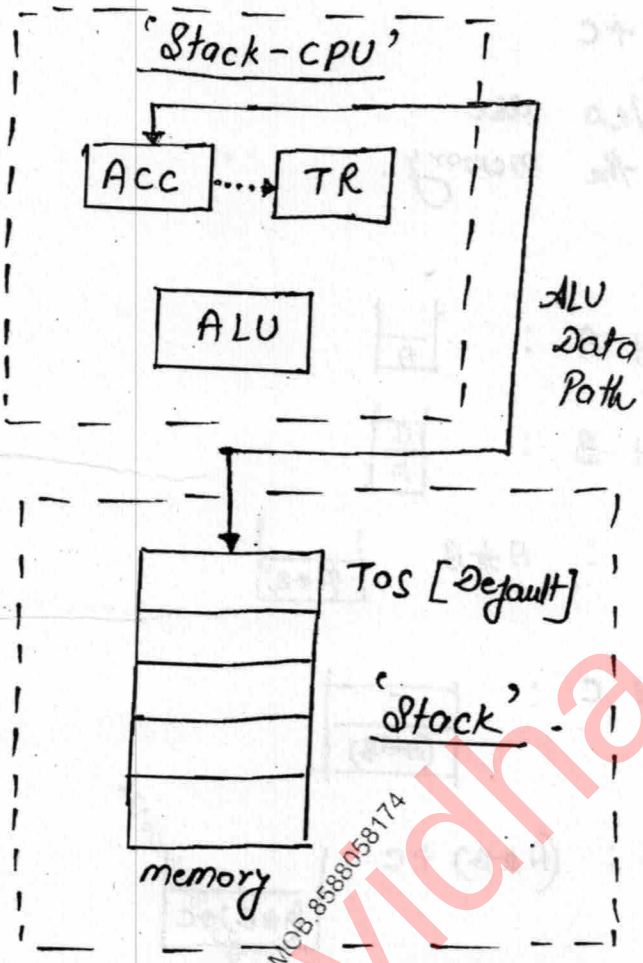
⇓ involves

| | |
|---|---|
| POP | To get the '$S_2$' |
| POP | To get the '$S_1$' |
| ALU opⁿ | operation on $S_1$ & $S_2$ |
| PUSH | Store the result |

↓

TOS ← TOS ( ALU opⁿ) TOS

* In this organization PUSH & POP instⁿ are used as Data Transfer Instructions, designed as a 'one address instructions'.






Program :-

Instruction : ADD : M/y

ADD : Stack CPU PC: Sequential Address

ADD

Inst$^{n}$ format

Type of operation

ALU operation

+

Pop Pop + Push

FC

EC

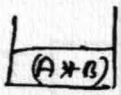
eg: (A*B)+C

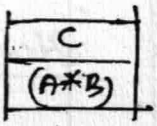
variables are in the memory.

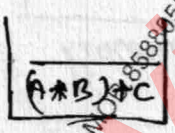
Code:

$I_1$: PUSH A : 


$I_2$: PUSH B : 


$I_3$: mul : A*B 

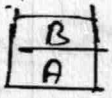

$I_4$: PUSH C : 

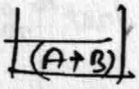

$I_5$: ADD : (A*B)+C 

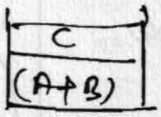

eg 2: X = (A+B) * (C+D)

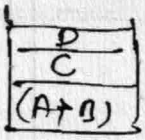
L.H.S variables are in R.H.S the memory.

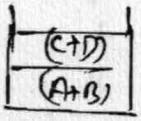
$I_1$: PUSH A : 

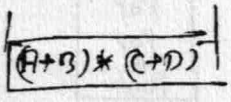

$I_2$: PUSH B 


$I_3$: ADD : A+B 


$I_4$: PUSH C : 


$I_5$: PUSH D : 


$I_6$: ADD : C+D 


$I_7$: mul : (A+B)*(C+D) 


T... assembly stack

**Que:** Consider the following code, running on stack CPU (Stack is empty).

CPU supports 8 bit opcode & 4GB memory.

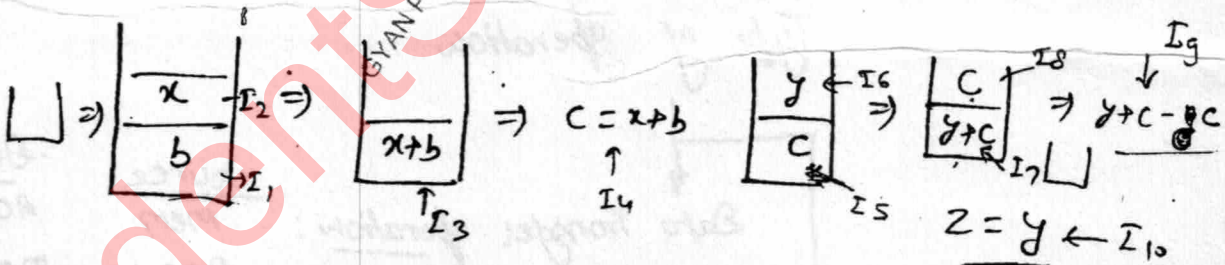
$I_1$: PUSH b – 5B
$I_2$: PUSH x – 5B
$I_3$: ADD – 1B
$I_4$: POP c – 5B
$I_5$: PUSH c – 5B
$I_6$: PUSH y – 5B
$I_7$: ADD – 1B
$I_8$: PUSH c – 5B
$I_9$: SUB – 1B
$I_{10}$: POP z – 5B

(a) what is the o/p of a Program code

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

(b) ⇒ 38B

a)




b)

8 bit opcode = 1B

Stack - CPU

10 instⁿ × 1B = 10B

4GB ⇒ # memory Address = $\log_2^{4G}$ ⇒ 32 bit Address = 4B

ALU Instⁿ

| opcode |
|---|
| 8 bit |

= 1B

Push & Pop → 1 address

| opcode | Addr |
|---|---|

= 5B

# Accumulator – CPU :-

* In this organization, one of ALU operand is always required in accumulator and the another operand is present either in a Register or in a memory. After the Data Processing result is placed into a Accumulator.

* In the CPU design, only one Accumulator is present so, It become the default location

* Instruction design is :-

Instn design : [Opcode | Address] ; I - Address

Instn format

Type of operation

Data Transfer operation : Source → Dest^n

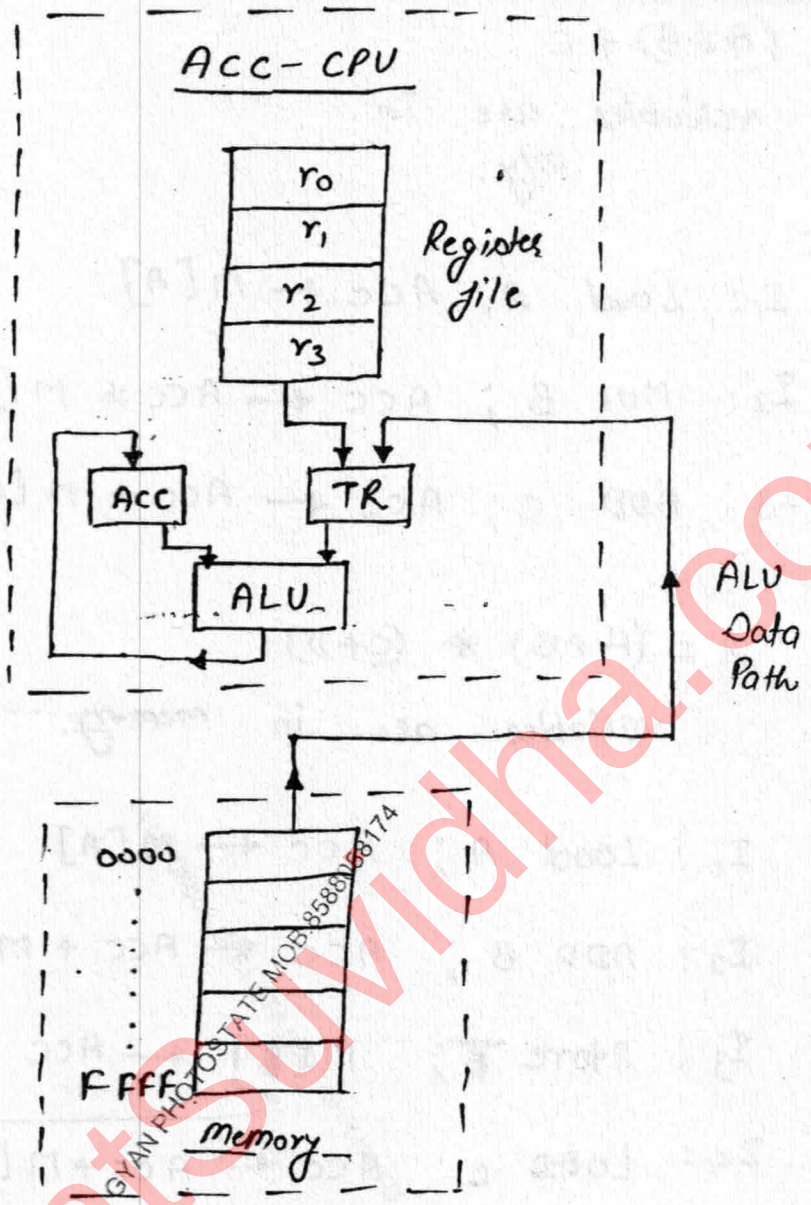
mem → ACC ($\overline{memRD}$) : Load

ACC → mem ($\overline{memWR}$) : Store

ALU operation : $S_1$ (ACC) $S_2$ (Reg/mry) D (ACC)

ACC ($\overline{memRD}$) : Load

mem ($\overline{memWR}$) : Store




Program :-

Instⁿ : Add [4000] ; memory

↓

Add [4000] ; Acc - CPU

PC : sequential address } Fc

↓

Add [4000] ; Instⁿ format

↓

ALU opⁿ

↓

+

⇓

ACC ← Acc + M [4000] } EC.

eg: $(A*B)+C$

variables are in m/y.

Code:

$I_1$: Load A; Acc ← M[A]

$I_2$: MUL B; Acc ← Acc * M[B]

$I_3$: ADD C; Acc ← Acc + M[C]

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

variables are in memory

Code:-

$I_1$: Load A; Acc ← M[A]

$I_2$: ADD B; Acc ← Acc + M[B]

$I_3$: Store T; M[T] ← Acc {Spilling}

T; Temporary variable

$I_4$: LOAD C; Acc ← 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

Que: $X = (A+B) / (C+D)$; variables are in m/y.

Code:-

$I_1$: Load C; Acc ← M[C]

$I_2$: ADD D; Acc ← Acc + M[D]

$I_3$: Store T; M[T] ← Acc

$I_4$: Load A; Acc ← M[A]

$I_5$: ADD B; Acc ← Acc + M[B]

To minimize the instⁿ follow the order.

$I_6$ : DIV T ; Acc ← Acc / M[T]
{(A+B)/(C+D)}

$I_7$ : Store X ; M[X] ← Acc

** Analysis :-

{ How to Design Different Inst$^n$ with Different format in the CPU }

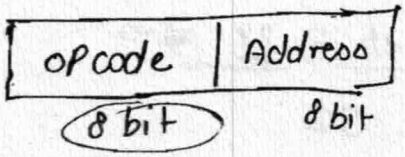
~~(a) variable length supported~~

* "Expand Opcode Technique" is used in the "fixed length instructions" supported CPU design, to implement the different instructions with various format. In this Process Low-order instruction is derived by merging the address field to a free opcodes. therefore derived opcode length is increases.

(a) variable Length Inst$^n$ supported CPU design :-

* Opcode size : 8 bit
* Address size : 8 bit

(1) 1-Address Inst$^n$ design :-




(2) 0-Address Inst$^n$ design :- { No Need of a Expand opcode Technique }