MD IFTAKHAR KABIR SAKUR

COMPUTER AND COMMUNICATION ENGINEERING

25TH BATCH

SUBJECT: CCE-3501

MICROPROCESSOR AND ASSEMBLY LANGUAGE THEORY

Introduction to Microprocessor

Microprocessor সেখানেই use হয় যেখানে programming করতে হয়।

8085 → Intel এর বানানো পৃথিবীর প্রথম M.P

8086 → এটি এর পরের সংস্করণ

186

286

386 → এটি অনেক পরিবর্তন আনে

486

80586 → Pentium-1

→ Game changer processor

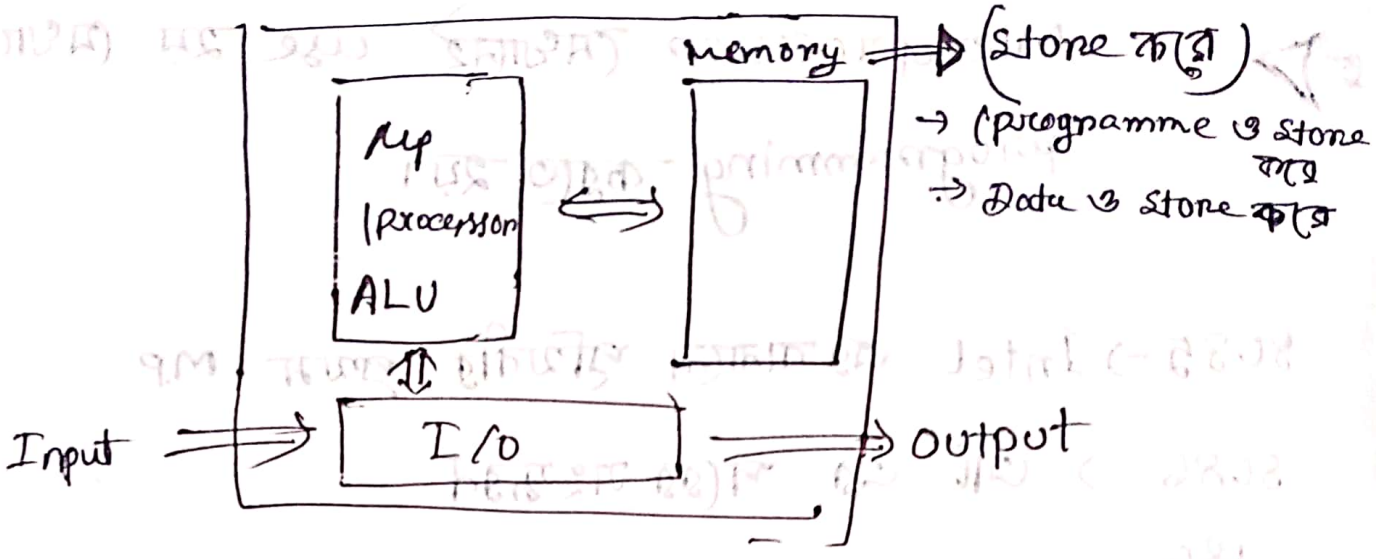
P-2, P.4 → 2000 yr

C2D

Core i9 → বর্তমান

MP → Microprocessor এভাবে লেখে

Computer System



(*) Fetch → Mp, memory থেকে Instruction কি (নেওয়া।

(*) (Instruction শুধু মাত্র memory থেকেই fetch হয়)

(*) "Decode" → Decode means understanding

Note!:

Decode (Description)

a = b + c ; Higher Level Language

ADD B, C ; Assembly Level Language

01101011 ; → (Machine Language) M/L, LLL, bin, obj

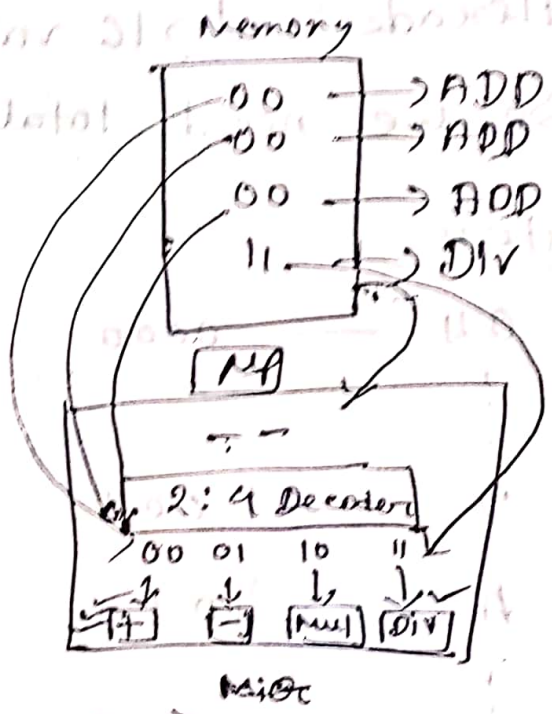
opcode → যার memory তে যাকল Instruction binary তে হয়।

(A) will create a file in .asm (which means it will be assembly language)

→ OPCode = Operation code.

Decoding :-

00 → Add
01 → Sub
10 → Mul
11 → Div



(A) এই কাজগুলো MP এর ALU Unit এ হয়ে থাকে।

Gigahaintz :-

Kilo → Thousand
Mega → Million
Giga → Billion
Tera → Trillion

3 GHz → 3 Billion operations in a second.

That means If a processor is having 3 GHz working power. Imagine how many works it is doing in a second.

(*) যত program লেখা হয় সব হেক্সাডেসিমেল এ লেখা হয়। কারণ Computer Hexadecimal এ কাজ করে।

(*) Hexadecimal → 16 values

So, we need total 4 bits

Hex.

0H — 0000

⋮

9 — 1001

A — 1010

⋮

F — 1111

(*) আমরা যেকোনো program করতে Hexadecimal এ যখন Computer এ enter করাব তা Binary হবে,

⇒ Hex to Binary

35H

3 → 0011

5 → 0101

$\rightarrow (00110101)_2$

(*) 72

0111 0010

$(01110010)_2$

(*) FF

1111 1111

11111111

(*) এখানে লক্ষ্যনীয় হলো আমরা single decimal digit এর Binary আকারে 4 bits করে বের করতে হবে।

(*) 8 bit

00 H

⋮

FFH

↓

byte

16 bit

0000 H ← Start

⋮

FFFFH ← ending

↓

word

(Difference between 8 bit & 16 bit)

(*) 25 → 8 bit

62 → 8 bit

1234 → 16 bit

4000 → 16 bit

51FC → 16 bit

5721 → 16 bit

0101 0111 0010 0001

Calculation (Important)

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

$2^6 = 64$

$2^7 = 128$

$2^8 = 256$

$2^9 = 512$

$2^{10} = 1K = (1024)$

$2^{20} = 1M$ (Mega)

$2^{30} = 1G$ (Griga)

$2^{40} = 1T$ (Tera)

$2^{11} = 2^1 \times 2^{10}$

$= 2 \times 1K$

$= 2K$

$2^{12} = 2^2 \times 2^{10}$

$= 4 \times 1K$

$= 4K$

$2^{16} = 2^6 \times 2^{10}$

$= 64 \times 1K$

$= 64K$

$2^{19} = 2^9 \times 2^{10}$

$= 512 \times 1K$

$= 512K$

$2^{20} = 2^{10} \times 2^{10}$

$= 1K \times 1K$

$= 1M \rightarrow$ Mega (million)

$2^{23} = 2^3 \times 2^{20}$

$= 8 \times 1M$

$= 8M$

Ex | pixel → picture + Element

[NOT Necessary for Academic Subject]

(*) ⇒ 8M pixel Camera means?

⇒ It has 8 Million pixels in a photo when it is captured.

Each & every single pixels are captured just in the same way that pixel is having colour in real life. So, when the file is created that file is created along with those pixels. And the colours which is in the pixel is having different binary numbers which ~~later~~ is how this entire photography system works.

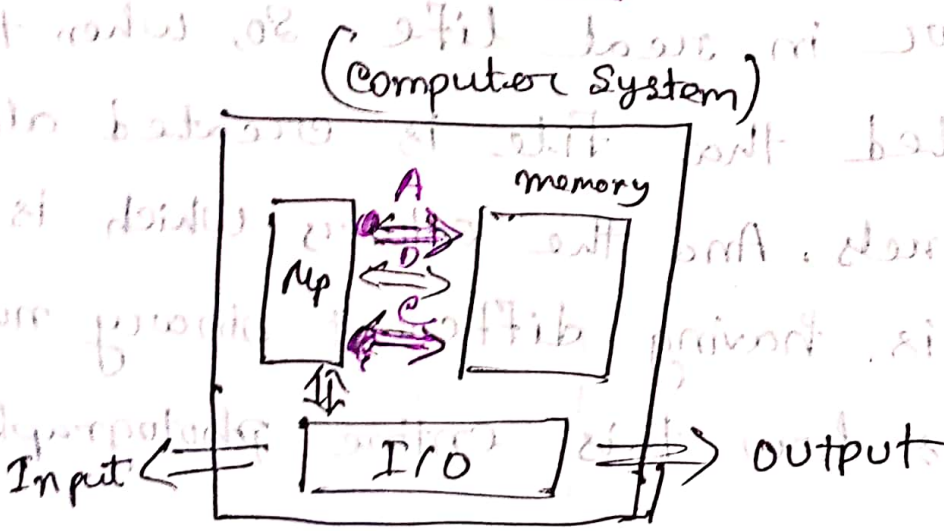
How audio & system works?

⇒ From microphone ~~del~~ makes the sound as electrical pulses. Then the electrical pulses goes to processor in A to D Converter. Then it makes as data which later creates as mp3 file.

⊛ µp এর কাজ হলো Instruction কে fetch, decode then execute করা।

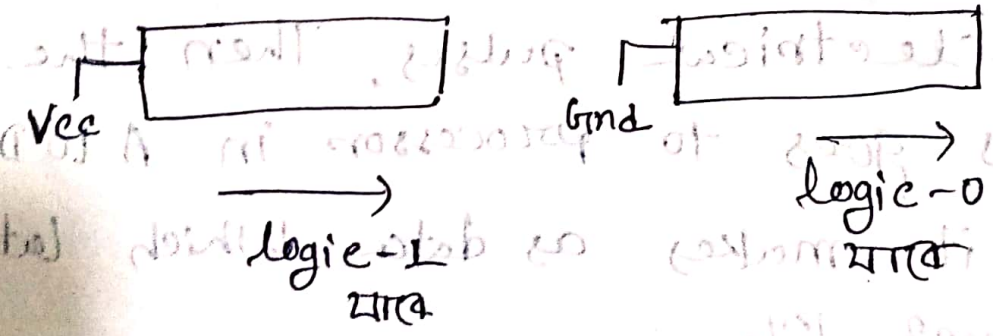
⊛ Memory → যেখানে data information store করা।

Buses



⊛ Bus:- এর মাধ্যমে Information pass করা হয়।

⊞ Bit transfer through bus:-

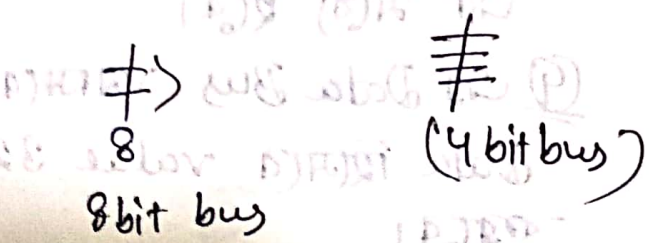


⇒ যদি bus এ Vcc Connect করা হয় তবে signal যাবে

1। আর যদি Gnd এ connect করা হয় তবে তা 0 হবে 0।
এখন, যদি continuously এই কাজটা করা হতে থাকে তবে bit transfer হতে থাকবে। যার মানে হচ্ছে এভাবেই video, Audio বা যেকোনো কিছু transfer হয়ে থাকে।

* এখন কোনো গান, ভিডিও file অনেক bit এর (Giga, mega) হয়ে থাকে। সেক্ষেত্রে এই bus switch 1s এ million times Vcc/Gnd connect করবে। যা manually impossible. Then we need Electronic switch এর প্রয়োজন হয়। যা হলো Transistor (Repeat Robotics Note)।

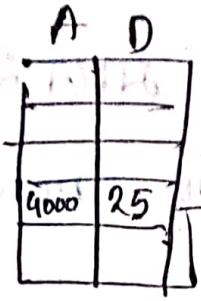
* So, bus is a set of lines.



It means যেকোনো সময় এর মধ্য দিয়ে 8 bit value transfer হবে

System Bus

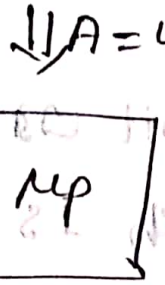
① Address Bus :- ② Data Bus Control Bus



memory তে অনেক location থাকে। আর প্রতি location এর আলাদা address থাকে।

⊛ It means location 4000 এ 25 value আছে।

এখন, এই 25 কে 32 বানাতে চাইলে :-



Note :-

① মনে রাখতে হবে, সবসময় Address Bus প্রথমে use হয়

② Address Bus একবার আর Address হিসেবে 4000 নিবে

③ এরপর এখন যেকোনো operation হলে তা এখন μp এবং 4000 এর মধ্যে হবে।

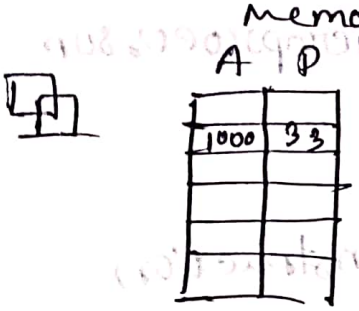
④ এর Data Bus আসবে। যা Data হিসেবে value 32 select করবে।

⑤ এবার Control Bus আসবে যার কাজ হলো কি করতে হবে তা ঠিক করা। এটি Read ও write দুইটাই করতে পারে। তো এখানে μp, C এর মাধ্যমে 4000 location এ 32 value write করবে।

Read:- Memory হতে Data নেওয়াকে বলা হয় Read. I/O হতে Data নেওয়াকে বলে Read।

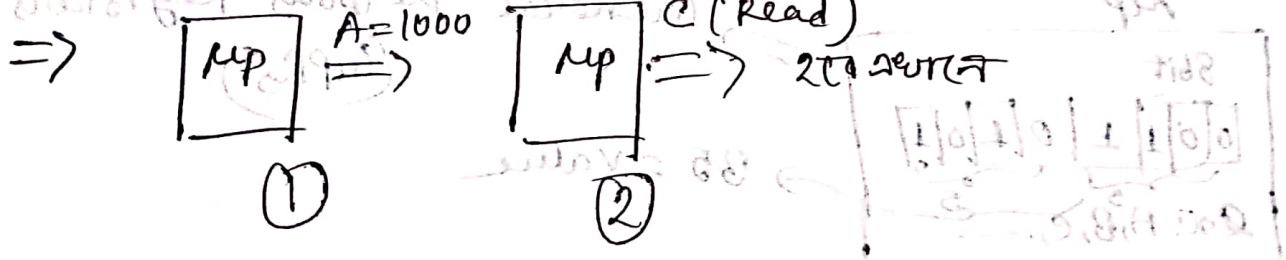
Write:- Memory তে Data পাঠানোকে write বলে। I/O কে Data পাঠানোকে বলে write।

⊛ সোজা হিসাবে, Mp যদি Data পায় তা Read Data পাঠাল তা write

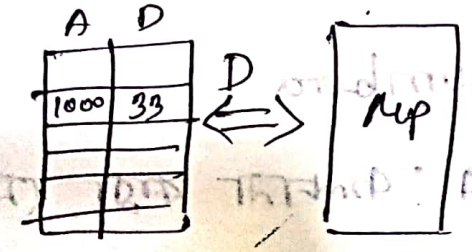


Task:-

⊛ 1000 location value কত তা কিভাবে দেখা যাবে?



① ②



③

⊛ আর, A, D & C এই তিনটাকে একত্রে System Bus বলা হয়। ~~প্রত্যেকটা কোনো System এ এই তিনটা com~~

(যেহেতু Data কে Read অথবা write করার সময় Data কে Data Bus এর মাধ্যমে আদান-প্রদান করা হয় সে কারণেই এটা Bidirectional)

যতক্ষন না কোনো System এ এই তিনটা একত্রে Connected হচ্ছে ততক্ষন পর্যন্ত System কি মাল বানাতে পারবে না।

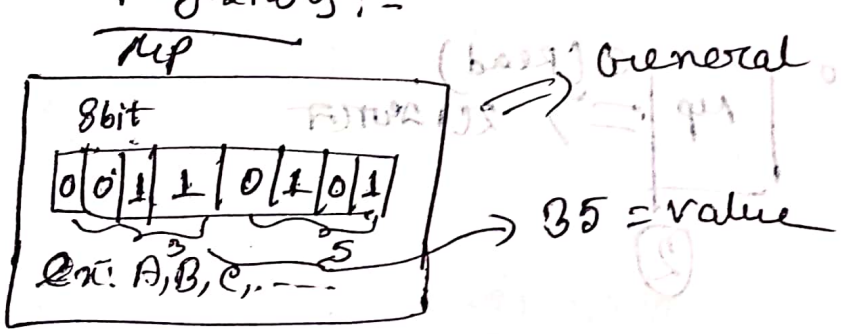
8 | 8085 processor

Microprocessor → program execute করা

* Instruction গুলোকে memory হতে microprocessor এ আনাকে বলা হয় Fetching

Decoding:- understanding the instruction

Registers :-



⇒ It is used to store numbers

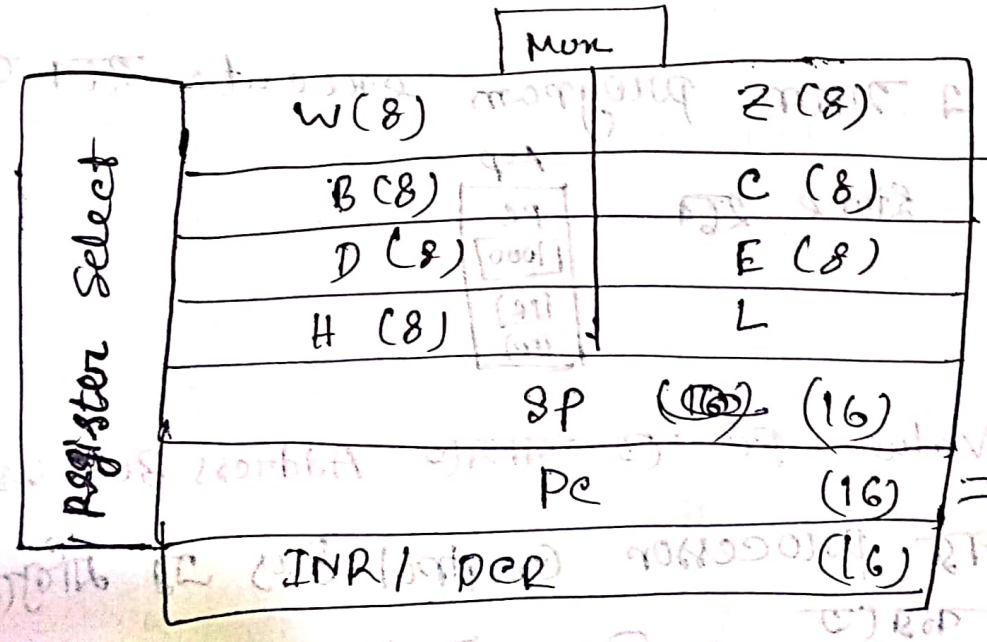
⇒ এটাকে মাকো হিসাবে কল্পনা করা যেতে পারে আর memory কে Bag হিসাবে। Bag এ যেগুলো কম গুরুত্বপূর্ন সেগুলো আর মাকেট যেগুলো গুরুত্বপূর্ন সেগুলো রাখা হয়।

8 bit register এ 8টি Flip-Flop থাকে যার bit = 0 & 1

⊛ 02+03 = 05

C program (Higher Level Language)

```
Int a,b,c;
variable { a = 2;
           b = 3;
           C = a+b;
```

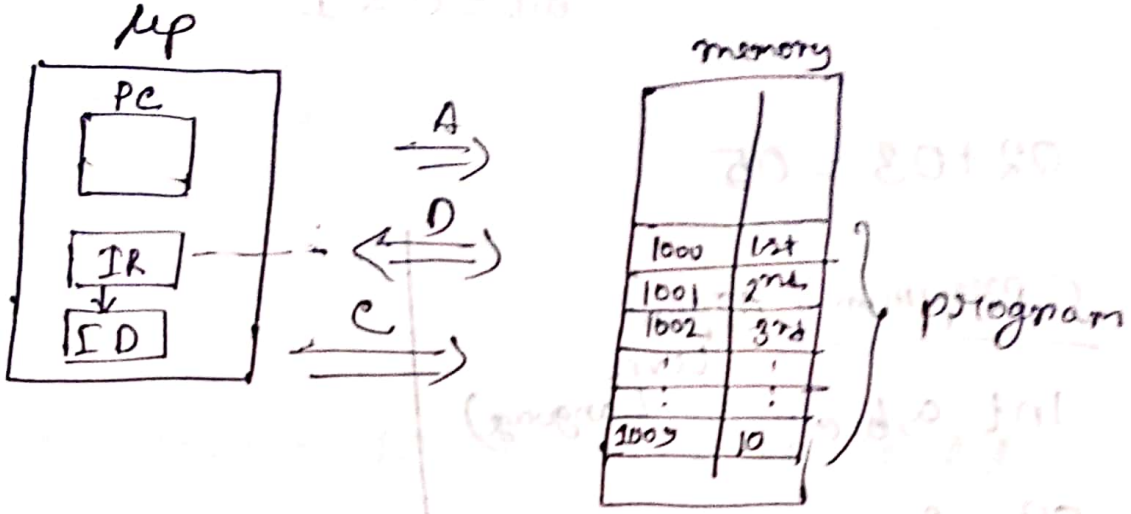


⇒ PC Register এ Program এর Address থাকে

A, B, D, H, C, E, L ⇒ General perpous Registers

PC ⇒ সকল Instruction এর start এর diagram হলো PC।

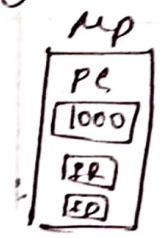
Program Execution



⇒ যে program execute করতে হবে তার Address টা store হয় PC তে।

Suppose,

1000 এ থাকা program execute হবে তাহলে ঐ value store হবে



এই Address value PC তে আসতে Address Bus use হবে। এরপর processor control Bus এর মাধ্যমে Read ~~Instruction~~ করতে পাঠাবে ঐ 1000 Address এ থাকা ~~value~~ Instruction টা। এরপর এই ~~value~~ Instruction processor এর Data Bus এর মাধ্যমে ~~আসে~~ processor এ আসবে।

এরপর এই Instruction টি Special Register IR এ Store হবে। (IR = Instruction Register)।

এভাবেই Fetching Step কাজ হয়।

Decoding Step:-

এই ID = Instruction Decoding এ হবে।

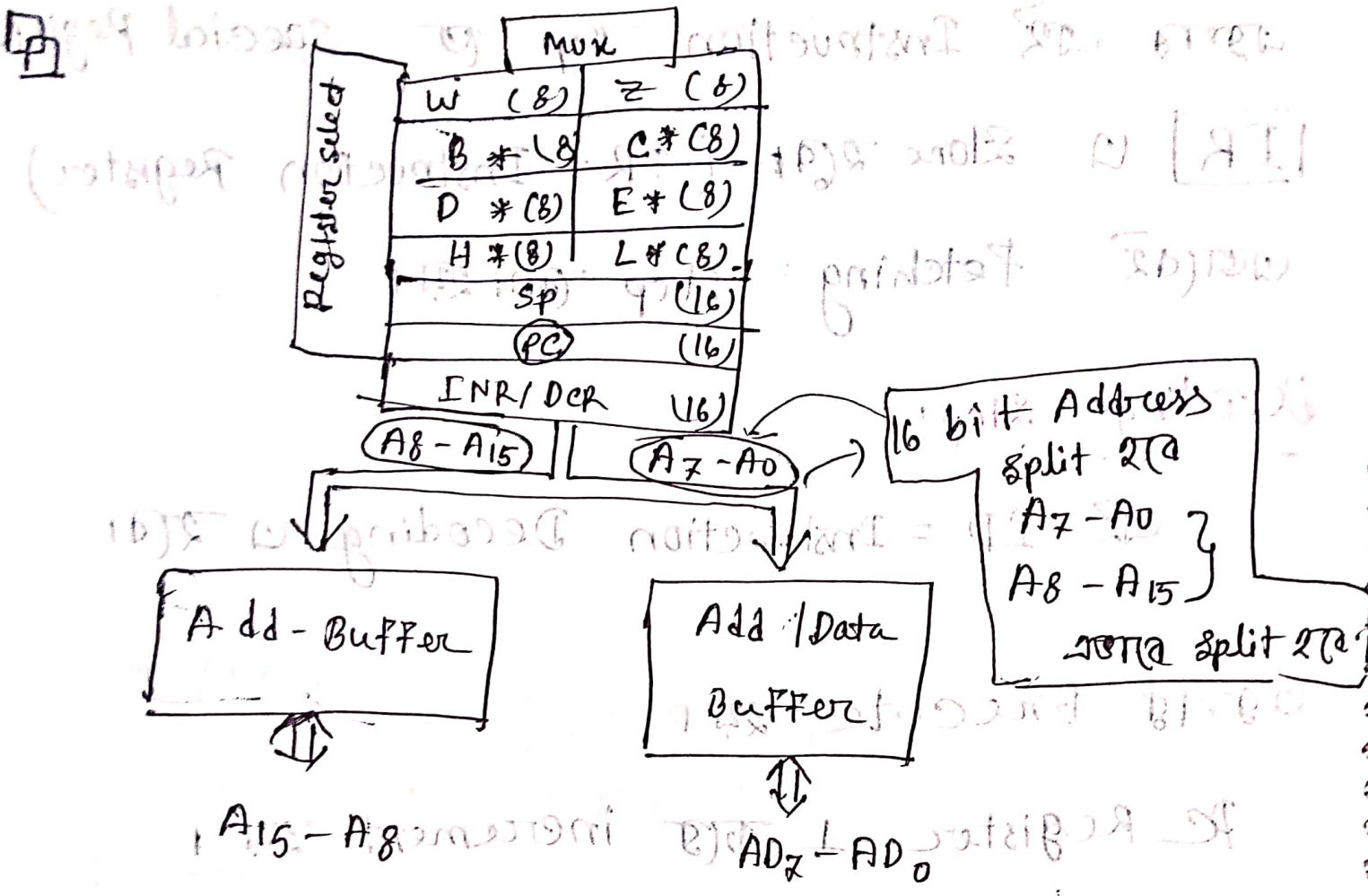
এরপর Execute হয়।

PC Register 1 করে increment হয়।

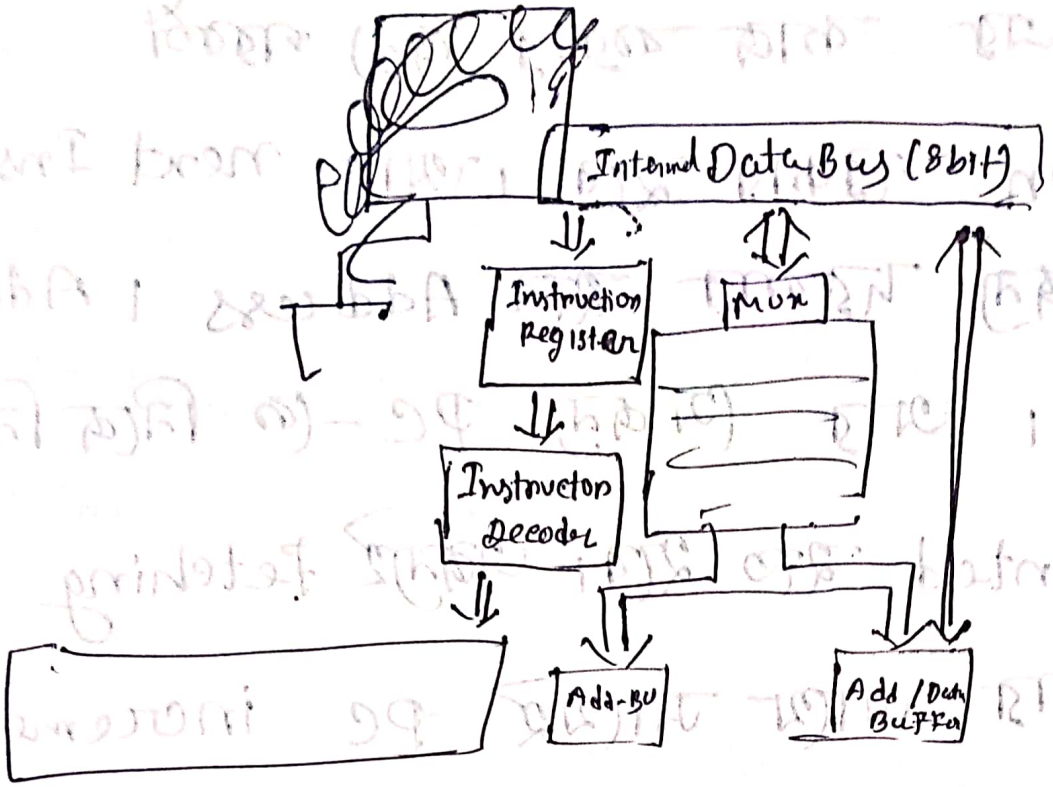
* যখনই কোনো Instruction fetch হয় এবং এরপর তা Decode করা হয়, এর সাথে সাথেই PC 1 করে increment হয়ে যায়।

এর মানে: PC always Contains of the next instruction। এর মানে হলো যদি দ্বিতীয় Instruction নিয়া কাজ করা হয় তবে PC তে ~~প্রথম~~ তৃতীয়টার ~~Instru Address~~ Instruction এর Address থাকে।

PC = 16 Bit এর হয়।



⇒ PC হতে আসা 16 bit এর Address দুইভাগে ভাগ হবে। $A_0 - A_7$ ও $A_8 - A_{15}$ (Address Bus) এর মাধ্যমে যাবে। এভাবে এই দুভাগে 16 bit এর Address যাবে memory তে। যা processor C (Bus) এর মাধ্যমে Read Signal memory তে দিবে। পরে memory হতে Instruction আসবে Data Bus এর মাধ্যমে up তে। (Note:- Data Bus 8 bit। সুতরাং একটি 8 bit এর address send করবে)। এখানে $D_0 - D_7$ হয়।



= এখানে এই ~~16bit~~ (Data Bus এর মাধ্যমে ~~Instructio~~ Instruction, (Internal 8bit Data Bus) এর মাধ্যমে Instruction টি Instruction Register এ store হবে। এবং এর মাধ্যমেই Fetching step সমাপ্ত হয়।

এখান থেকে শুরু হয় Decoding। এ মা Mp Decode করে বুঝতে পারবে কি করতে হবে।

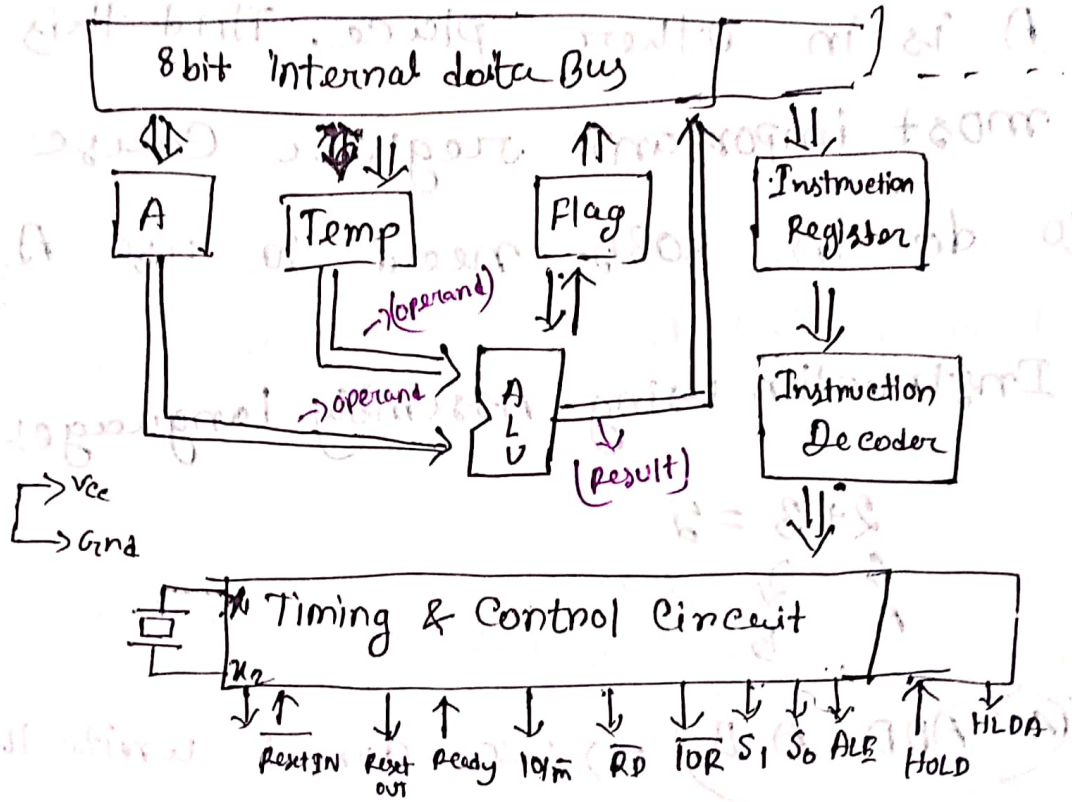
এবার এই Decoding Information টি Timing & Control Circuit কে দিবে। যা ~~Ex~~ Execution করবে। এভাবেই Instruction চক্র শেষ হয়।

একজন MP এর কাজ করার জন্য পরবর্তী Instruction দরকার হবে। আর next Instruction এর জন্য দরকার হবে Address। Address দিবে PC। আর সেজন্য PC-কে নিজে নিজেই Incremented হতে হবে। একজন্যই Fetching শেষ হওয়ার সাথে সাথেই PC increment হয়ে যাবে। আর সেটা করার জন্য তার একটা Dedicated (INR/DCR) (Incrementer/Decrementer) প্রয়োজন হবে।

Timing & Control Circuit :- (It's like a brain)

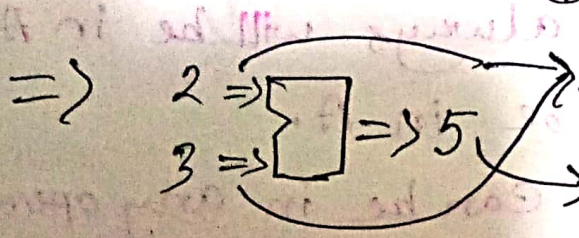
यहाँ Control signal release करता है।

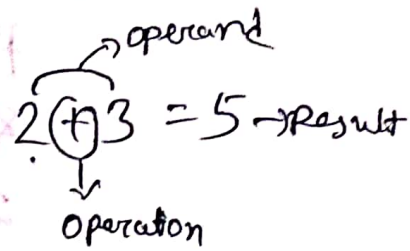
PC (Address) → Instruction Fetch → Decode → Timing & Control Circuit → Execution

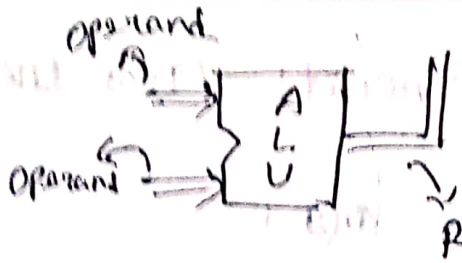


ALU :-

All other components are rectangle in shape but why ALU is different?







B, C, D, E, H, L, A => All are general purpose registers.

From this 6 are in different place, only A is in other place. And this is the most important register cause all things to do in 8085 need to use A.

Instruction using Assembly Language:-

2 + 3 = 5

A B

~~A ← ADD A, B~~ => we don't write this

we write:- ADD B

In 8085 => In Arithmatic or logic operation the first operand always will be in A & result will be in A.

2nd operand can be in any operand.

(*) ADD B (A

ADD C

ADD D

2) ~~(*) when B is~~ first A & B registers will be added, then C will be added with that result (A+B+C). Then ADD D will add D with all (A+B+C) & will make (A+B+C+D) & all of them are being accumulated in A.

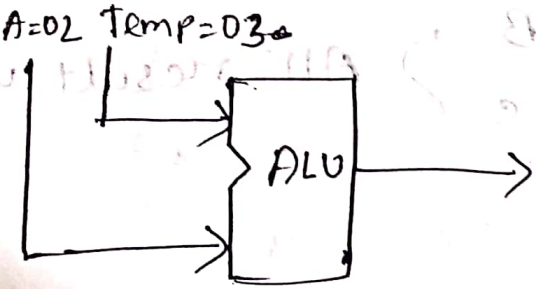
In some up this accumulator can be R0, W Register.

(*) Sub B → A-B
Sub C → A-C
> All result will be in A

⊛ How ALU works:- (See the Figure of ALU)

A B
↑ ↑
02 03

Value(02)
So, First will move to ALU from A(02).
The other value(03) can come from any other register using (8 bit internal Bus) & will go to temp register then it will move to ALU. Then the value (Result coming from ALU) will go to 8 bit internal data Bus & will store in A register.



Entire MP works (Microprocessor's work)

Instruction (stored in memory) ⇒ Fetch ⇒ PC (to get address) ⇒ Address ⇒ Address Bus ⇒ Memory

Memory ⇒ MP (Read) instruc. ⇒ Instruction using Data Bus ⇒ 8 bit Internal Data Bus ⇒ Instruction Register ⇒ (Fetching ends here)

Instruction Register ⇒ Instruction Decoder ⇒ Understanding the instruction

Understanding the instruction ⇒ Timing & control Circuit (It is like a Brain. Release signal so that others can work like the way they should).

Timing & control Circuit ⇒ For the giving instruction T & C will release signal to chose registers

For the giving instruction ⇒ One will be A

For the giving instruction ⇒ Other will be selected from Register Select

One will be A ⇒ The instruction from the register will use 8 bit Internal Bus

Other will be selected from Register Select ⇒ The instruction from the register will use 8 bit Internal Bus

The instruction from the register will use 8 bit Internal Bus ⇒ Temp. Register ⇒ Both register will receive signal From T & C to work just like the way instruction given

Both register will receive signal From T & C to work just like the way instruction given ⇒ Both register will work like that

Both register will work like that ⇒ T & C will give signal to ALU to work like instruction

T & C will give signal to ALU to work like instruction ⇒ ALU will give a result which will use 8 bit internal data Bus

ALU will give a result which will use 8 bit internal data Bus ⇒ Then the result will store to A register again

Then the result will store to A register again ⇒ And when all these work was happening PC was automatically incremented by 1

And when all these work was happening PC was automatically incremented by 1 ⇒ [And this process will be repeated like this]

Flag register :-

There are many flags in the flag register

All flag register is 1 bit. There are 5 flags in 8085.

⇒ Carry Flag :- operation এ কোনো carry ছিল কি না তা detect করে।

⇒ Sign Flag :- will the result be positive or negative? Sign flag tells us this.

⇒ Zero Flag :- It is used if the result is zero.

⇒ Parity Flag :- It will tell us if there is an even parity or an odd parity.

⇒ Auxiliary Flag :- This flag is used in BCD (Binary Coded Decimal) operations.

The status of this flag is updated for every arithmetic or logical operation performed by ALU.

* When ALU perform an operation it creates two things

(i) Result. which goes directly to the A register through Bus.

(ii) Status of the operation. which goes to Flag register.

That means,

=) Flag register contains the status of the current result.

* ALU can Read the Flag & can write the Flag that's why it is bidirectional.

Register pair:-

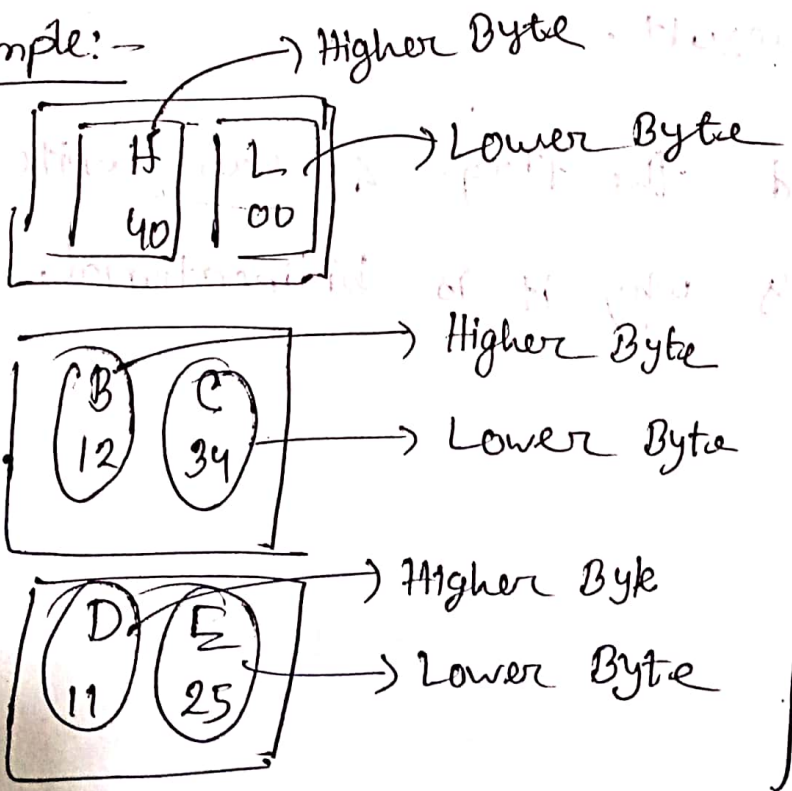
During programming we need 16 bit sometimes. So, we take two registers & make them pairs. But we can't make pairs randomly. There are some rules of doing it.

B → C

D → E

H → L

Example:-



Note:-
এগুলোকে আলাদা আলাদাও ব্যবহার করা যায়, আবার একসাথেও ১ pair হিসেবে use করা যায়।

W → Z (Temporary Register pair)

=) This pair is not available to the programmer. This can only be used by μp. We can never read its datas. Can never see too.

XCHG (Exchange) =) Instruction

HL & DE pairs value are ~~begin~~ exchanged by it.

Suppose,

HL	DE
1234	5678

Here we need a temporary register too exchange. Just like in programming we use temp variable.

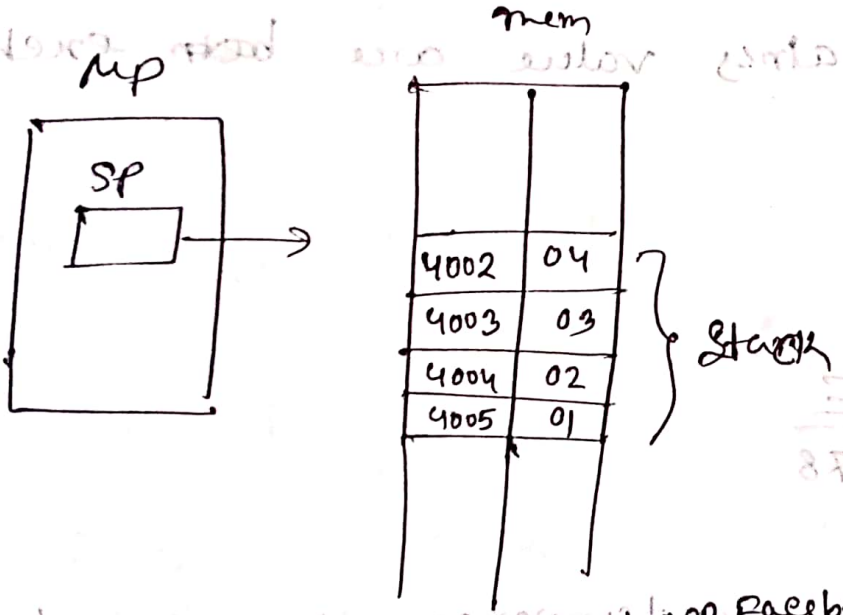
So, μp will take the value of HL & into some temporary place like WZ. And the value would by WZ → 1234. The μp will take DE's value & will store it in HL. Later, it will take

value from WZ, and will give it to DE (1234)

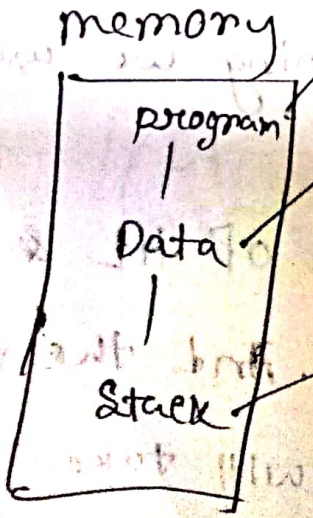
80,

HL	DE
5678	1234

SP (Stack pointer):- (STACK → LIFO)



(messenger, WhatsApp, Facebook.....)



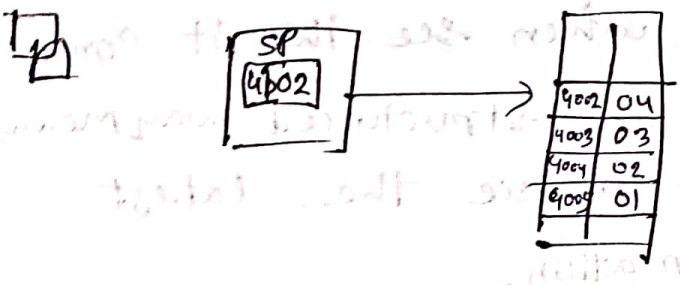
Random Data
It is sorted in random way

It is stored in structured way.

=) Queue operation এ আমাদের দুইটা Address মনে রাখতে হয়। Last এ কে আসলো আর First হতে কে গেলো।

But, in stack we don't need to do this. We can ~~withdraw people~~ remember just one address & can work two. That's why stack is used more than queue.

However, the address of stack is stored in SP.



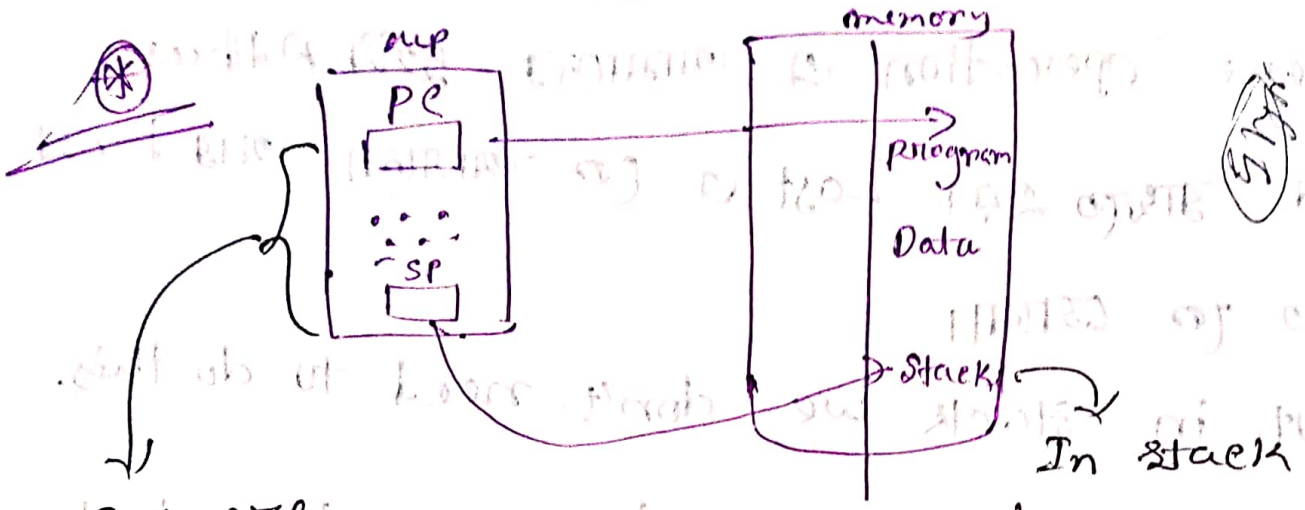
TOP of Stack

value of TOS : 04

Address of TOS : 4002

∴ SP = 4002

So, SP means :- It is the address of TOP of Stack.



Both are registers. But PC is used more than SP.

In stack we do only two things
(i) push
(ii) pop

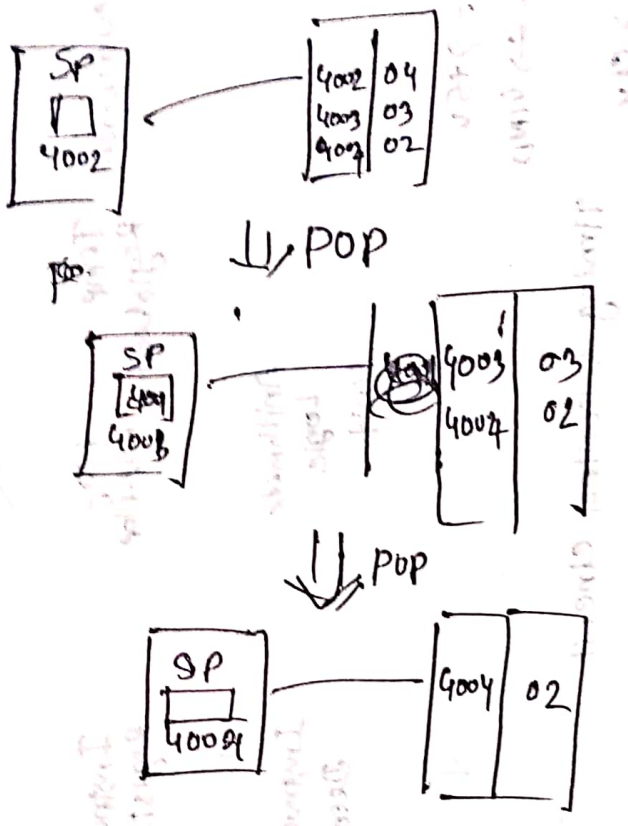
⇒ This Stack Form is used in:

→ messaging
→ E-mail
→ Photo saving
etc.

} Cause when see this it come to us as structured way means we get to see the latest information.

(*) Each time the push happens the SP gets decremented. Cause the value gets added at the top of the stack & SP give focus on the top level data's address. That means it gets decremented.

POP: Every POP SP is incremented



INR / DCR :-

→ PC gets incremented everytime

→ SP either gets incremented or decremented

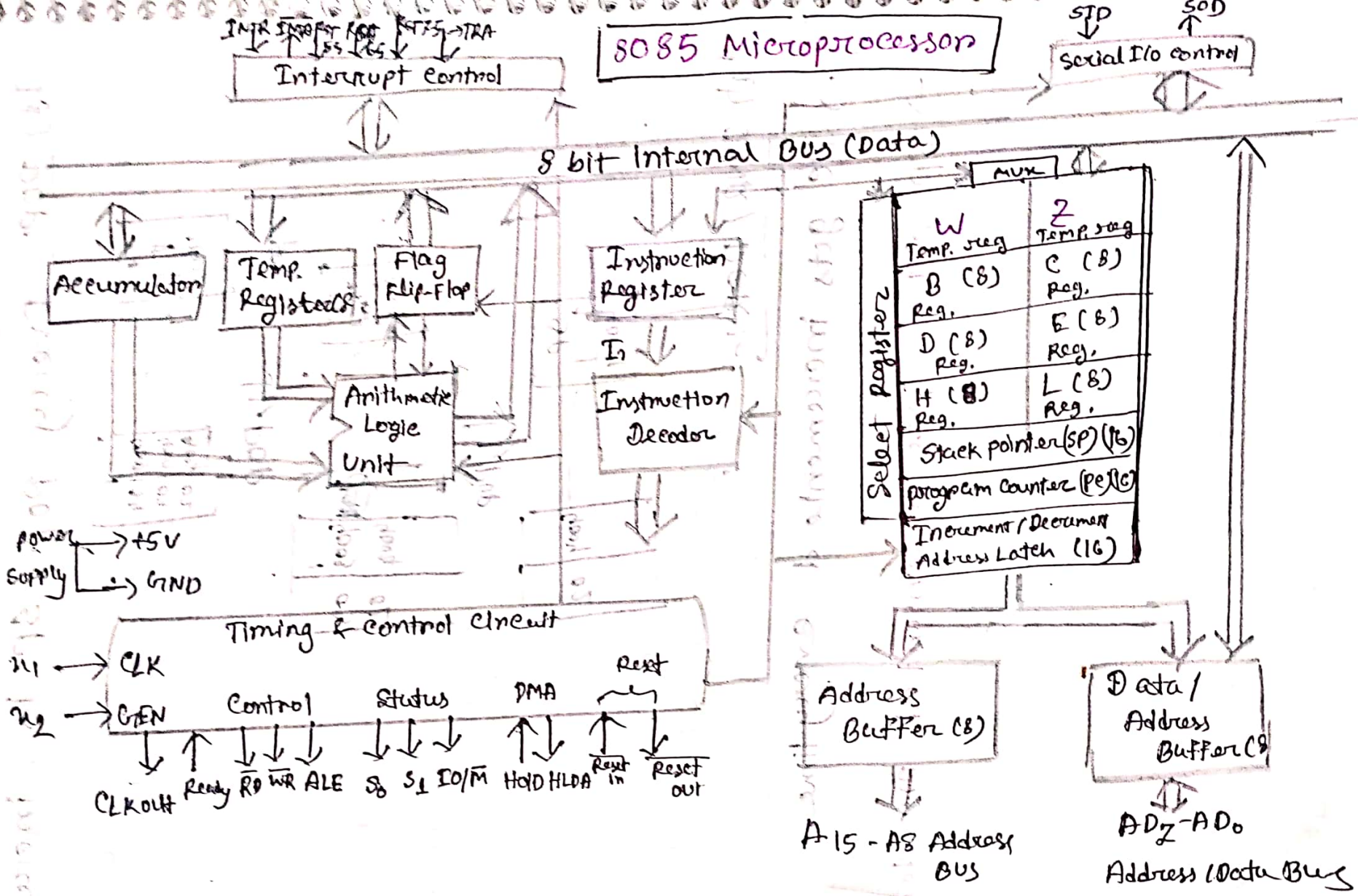


Figure:- 8085 Microprocessor Architecture

Instruction for programming of 8085

MD Iftakhar Kabir Sakun
E201037

① MVI instruction:- (Move Immediately)

This instruction supports immediate addressing mode for specifying the data in the instruction.

[Immediate value or data]

Example:- MVI B, 20H

↳ put the value 20 in B register

20H → Hexadecimal

② LXI :- There are 4 type of LXI instruction

(i) LXI B

(ii) LXI D

(iii) LXI H

(iv) LXI SP

These all are used to load the 16-bit address into register.

Example:-

(i) LXI B, 200DH → register pair

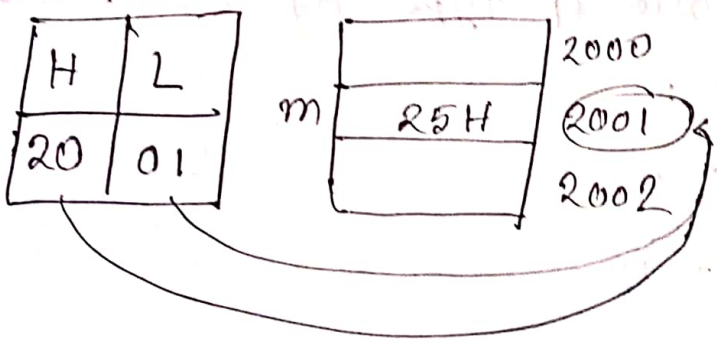
↳ put the data in B register. B ← 20, C ← 00

③ m (memory pointer):-

It holds the address of a particular memory location. They can store 16-bit address as they work in pair. Program Counter. H & L register pair is used as a memory pointer. And it holds 16-bit address of memory location.

Example:-

MVI m, 25H



⇒ HL pair এর মধ্যে যে value থাকে সেই value থেকে যে memory address টা পাওয়া যায়, সেই Address এই data store হয়। একেই memory pointer বলে।

(1) **mov**:- This instruction copies the data item referred to by its second operand (register contents, memory contents, or a constant value) into the location referred to by its first operand (a register or memory).

Example:-

(i) mov B,C → Copy the value of C to B

B ← C

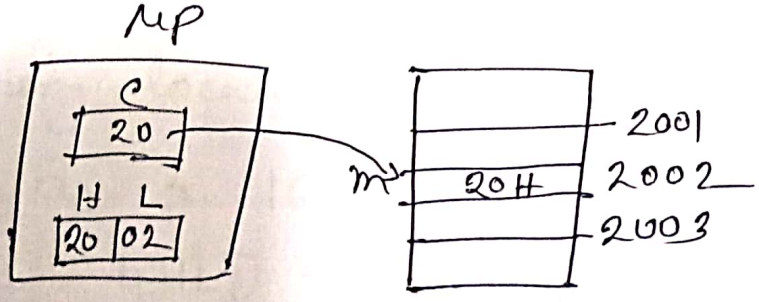
(ii) mov B,m → Copy the value of m to B

B ← m

(iii) mov m,C

↳ source
↳ Destination

=> copy the valye of C to m



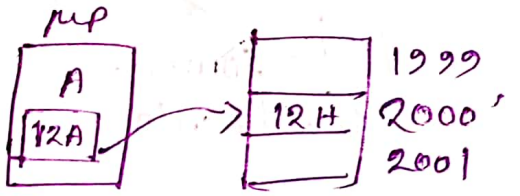
⑤ LDA :-

LDA is a mnemonic (स्मृतिसंकेत) that stands for (Load Accumulator with the contents from memory).

In this instruction Accumulator will get initialized with 8 bit content from 16 bit memory address as indicated in the instruction as a 16. This instruction uses absolute addressing for specifying the data.

Example :-

LDA 2000H → Load directly the value in 2000H (Address) to Accumulator



⑥ STA :- STA is a mnemonic that stands for Store Accumulator Contents in memory. In this instruction, Accumulator 8-bit content will be stored to a memory location whose 16-bit address is indicated in the instruction as a 16. This instruction uses absolute addressing for specifying the destination.

Example1 -

STA 3000H;

=) Store the value of Accumulator in 3000H

↓
Address

A [3000H]

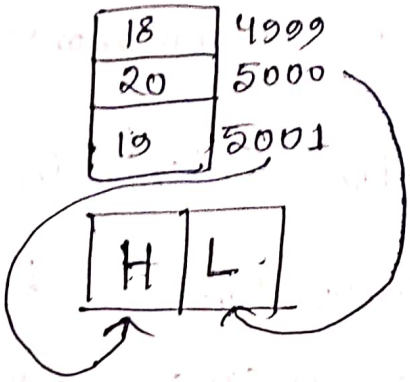
⑦ LHLD:- It is a mnemonic that stands for Load HL pair using Direct Addressing memory from memory location whose 16-bit address is denoted as a 16.

Example:- LHLD 5000H

=> Load the value of 5000H & 5001H (address) in HL pair Directly.

L ← 5000 H

H ← 5001 H



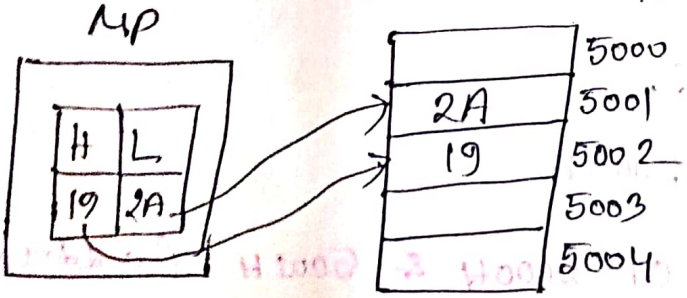
(8) SHLD :- SHLD is a mnemonic. which stands for Store HL pair using Direct addressing in memory location whose 16-bit address is denoted as a16.

As HL pair has to be stored. So it has to be stored in two consecutive locations starting at the address a16. We know that H & L are 8-bit registers.

Example:-

SHLD ~~5000~~ 5001 H -

↳ Store the value of HL pair Directly in 5001 H (address)



L → 5001 H

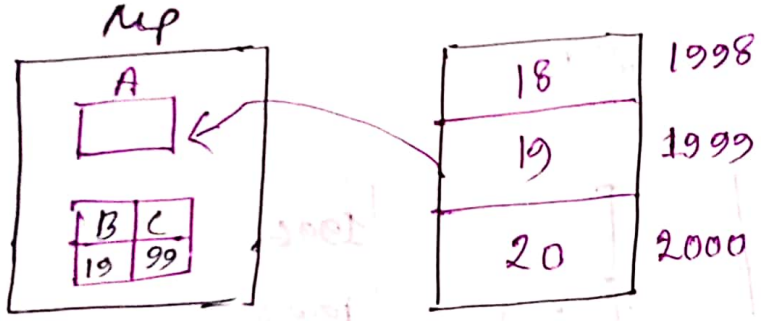
H → 5002 H

(9) LDAX :- It is a mnemonic that stands for Load Accumulator from memory pointed by extended register pair denoted as "pp" in the instruction.

This instruction uses register indirect addressing for specifying the data. It occupies only 1-Byte in the memory.

Example :-

LDAX B; A → [BC]



⇒ Memory Address - 16 Bit

→ বাকি সব - 8 bit

→ দুইটা Register = 16 bit

→ BC pair এর মধ্যে যে pair আছে তা কোনো memory Address।

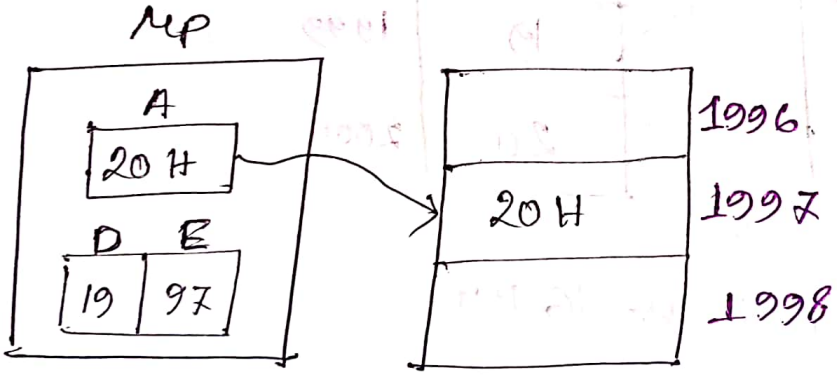
(10) STAX :-

It is a mnemonic that stands for Store Accumulator contents in memory pointed by extended register denoted as "rp". This instruction uses register indirect addressing for specifying the data. It occupies only 1-Byte in the memory.

Example:-

STAX D;

A → [DE] (or) Store ~~2C~~



(11) PCHL :- (Load the program counter with HL contents)

It is an unconditional branch where control of program is transferred to memory address equivalent to (HL) register pair content.

It falls in Register Addressing Mode and is a Single Byte instruction. This instruction will affect the flow of program execution if due core not taken.

Example :- PCHL

PC ← HL

(12) SPHL :- An instruction with help of which Stack pointer will get initialized with the contents of register pair HL. It is an indirect way of initializing the Stack pointer.

SPHL

SP ← HL

DAX BC HL

SHA X DE

Arithmetic Group

(1) ADD R :-

Adds the value in Rn & operand2. That means, it adds together in two operands, storing the result in its first operand.

Example:- (i) ADD B

$A \leftarrow A+B$

(ii) ADD M

$A \leftarrow A+M$

(2) ADI :- (Add immediate to Accumulator)

Here "d8" stands for any 8 bit or 1 Byte of data. This instruction is used to add 8 bit immediate data to the Accumulator. The result of Addition will be stored in the accumulator.

Example:-

ADI 25H

$A \leftarrow A+25H$

③ SUB :-

It is used to subtract of binary data in byte, word and doubleword size. For subtracting 8 bit, 16 bit or 32-bit operands, respectively.

Example :-

(i) SUB B

$A \leftarrow A - B$

(2) SUB M

$A \leftarrow A - M$

④ SUI :- (Subtract Immediate From Accumulator)

And here d8 stands for any 8 bit or 1-Byte data. This instruction is used to subtract 8 Bit immediate data from the Accumulator.

Example :-

SUI 20H

$A \leftarrow A - 20H$

⑤ ADC :- (ADD with Carry)

It is executed as a part of a multibyte or multiword addition in which an ADD instruction is followed by an ADC instruction.

This instruction can be used with a LOCK pre-fix to allow the instruction to be executed atomically.

Exam :-

(i) ADC B

A ← A + B + CF

(ii) ADC M

A ← A + M + CF → Carry Flag.

⑥ ACI :- (ADD with Carry immediate to Accumulator)

This instruction is actually meant for adding one 8-bit immediate data or operand to the Accumulator along with the Carry value.

Example:-

ACI 25H

A ← A + 25H + CF

(7) SBB :- (Subtract with Borrow)

R stands for any of the following 7 registers, and also memory location M as pointed by HL register pair. This instruction is used to compute subtraction between contents of R register from the accumulator's content, along with the carry value (Borrow).

Example:- (i) SBB B

A ← A - B - CF

(ii) SBB M

A ← A - M - CF

(8) SBI :- (Subtract with Borrow Immediate From Accumulator)

d8 = Any 8 bit data as operand. This instruction is used to subtract 8-bit immediate data

from the Accumulator along with the Carry (Borrow) value.

Example:- SBI 25H

A ← A - 25 - CF

(9) INR:- (Increment)

This instruction is used to add 1 with the Contents of R. So, the previous value in R will get increased by amount 1 only.

Example:-

INR B

B ← B+1

(10) INX rp (Increment Extended Register)

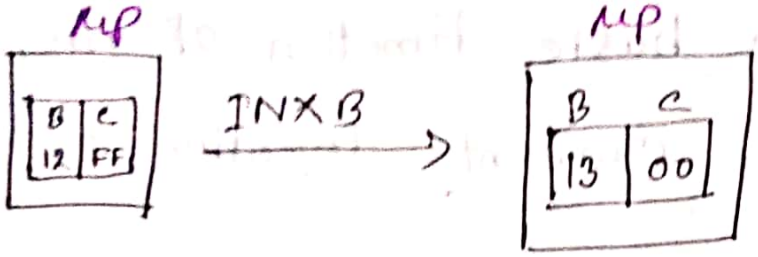
Here, rp (register pair) = BC / DE / HL

This instruction will be used to add 1 to the present content of the rp.

Example!-

INX B

BC ← BC+1



(11) DCR: (Decrement)

R stands for any of the following registers, or memory location M pointed by HL pair. This instruction is used to decrease the content of register R.

Example!-(i) DCR B

B ← B-1

(ii) DCR M

M ← M-1

(12) Dex!- (Decrement the SP content by 1)

=> DCX B

BC ← BC-1

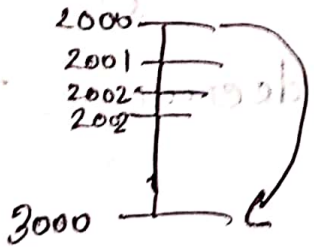
Branch Control Group

(i) JMP :- (Jump)

It performs the basic function of transferring control from the current location to a new location.

(a) Unconditional Jump

JMP 3000H



(b) Conditional Jump :-

(i) - JC → Jump if Carry (Carry હોય તો Jump નહીં તો NO Jump)

⇒ JC 3000H | CF = 0 ✗ CF = 1 ✓

(ii) JNC → Jump if NO Carry

Example :- JNC 3000H | CF = 0 ✓ CF = 1 ✗

Carry ન હોય તો Jump આપો & NO Jump

8085 pin configuration

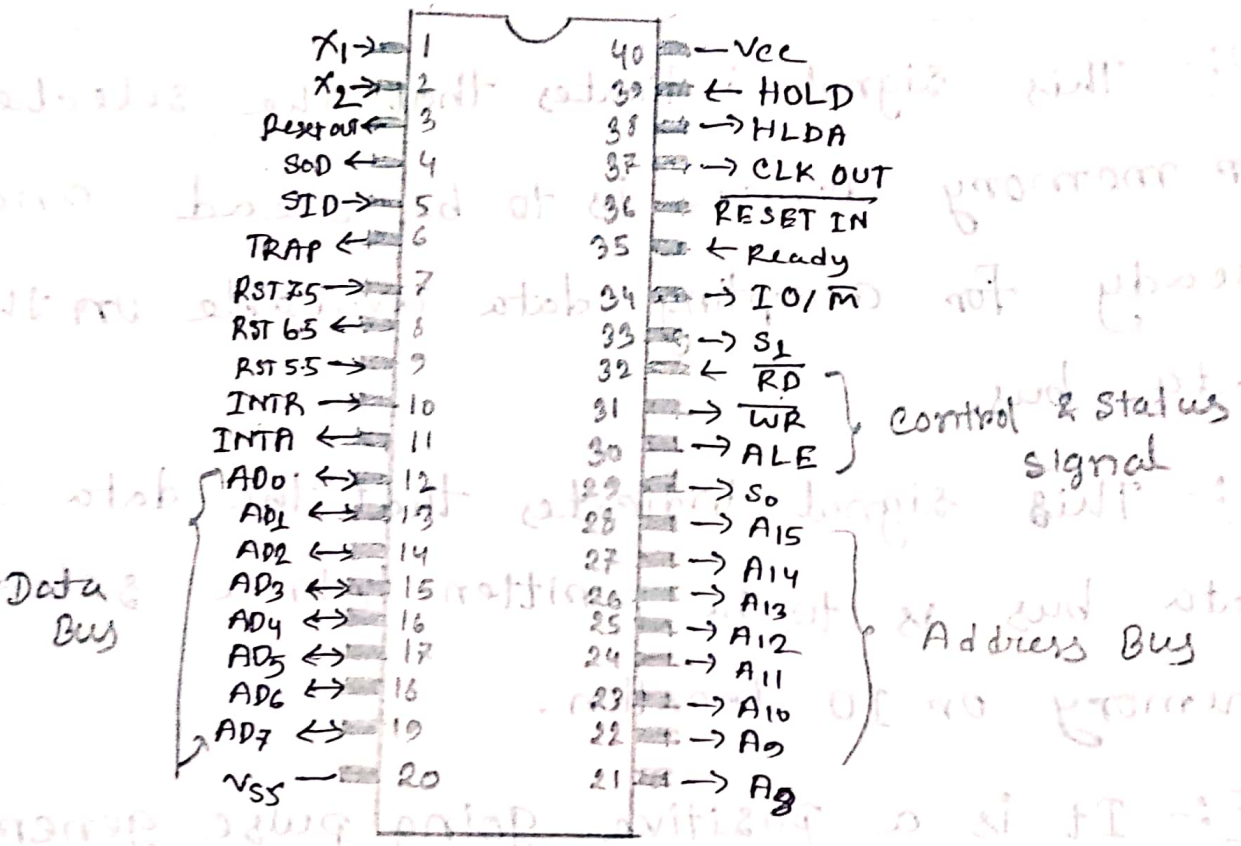


Figure:- 8085 Pin Diagram

This can be classified into seven groups :-

Address bus :-

A15 - A8 , It carries most significant 8 bits of memory I/O address.

Data Bus :-

AD7 - AD0 it carries the least significant 8 bi address & data bus.

3] Control and Status Signals:-

There are 3 control signal & 3 status signals

RD:- This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.

WR:- This signal indicates that the data on the data bus is to be written into a selected memory or IO location.

ALE:- It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals:

IO/$\overline{M}$:-

This signal is used to differentiate between IO and memory operation.

When it is high that indicates IO operation & when it is low that indicates Memory operation.

S1 & S0 :-

These signals are used to identify the type of current operation.

Vcc & Vss :-

There are 2 power supply signals - Vcc & Vss.

Vcc indicates +5v power supply & Vss indicates ground signal.

Clock Signals

X1, X2 :- A crystal (RC, LC, N/W) is connected at these 2 pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

CLK OUT:-

This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated Signals

INTA :- It is an interrupt acknowledgement signal.

RESET IN:-

This signal is used to reset the microprocessor by setting the program counter to zero.

RESET OUT:-

This signal is used to reset all the connected devices when the microprocessor is reset.

READY:-

This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.

HOLD:- This signal indicates that another master is requesting the use of the address and data buses.

HLDA (HOLD Acknowledge):-

It indicates that the CPU has received the hold request & it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Serial I/O signals

These signals are used for serial communication

① SOD (Serial output Data line):-

The output SOD is set/reset as specified by the SIM instruction.

② SID (Serial Input data Line):-

The data on this line is loaded into accumulator whenever a RIM instruction is executed.

Instruction Set

(1) Data Transfer group :

(i) MVI B, 20H → put the value 20 in B register

MVI → I থাকলে Data বুঝায়
move immediately

B ← 20

(ii) LXI B, 2000H → Load extended pair Immediately

X থাকা মানে Register pair use করবে

B ← 20
C ← 00

W	Z
B	C
D	E
H	L

(iii) MVI M, 25H

M → memory pointer
[যেখান একটা memory location যেটা HL pair এর যে value থাকে সেটাকে Address দেয়]

M ← 25H
→ means

HL pair

H	L
20	00

	1999
25H	2000
	2001
	2002

(iv) mov B, C → Means ⊕ Copy the value of C to B

B ← C

(v) Mov B, M

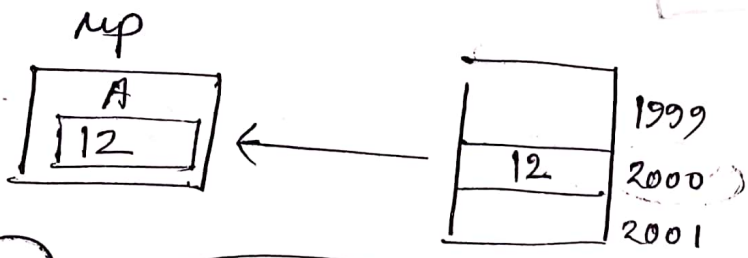
B ← M

(vi) Mov M, C

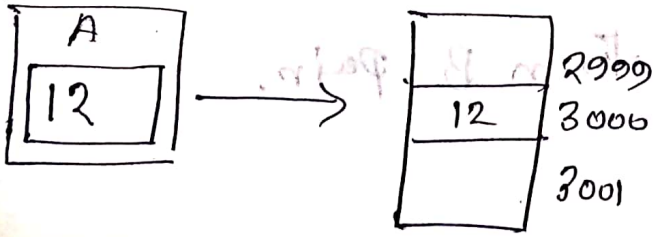
M ← C

(vii) LDA 2000H → Load Directly the value in 2000H to Accumulator

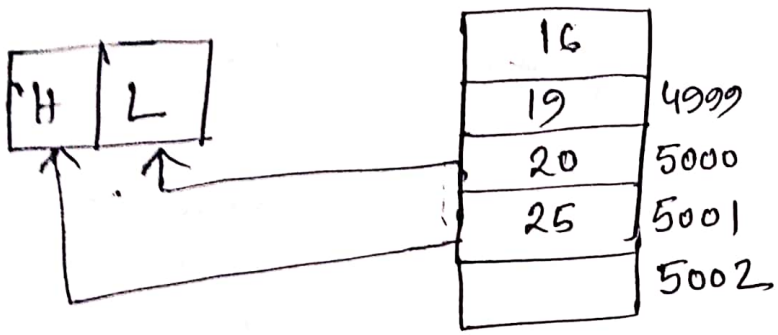
Address



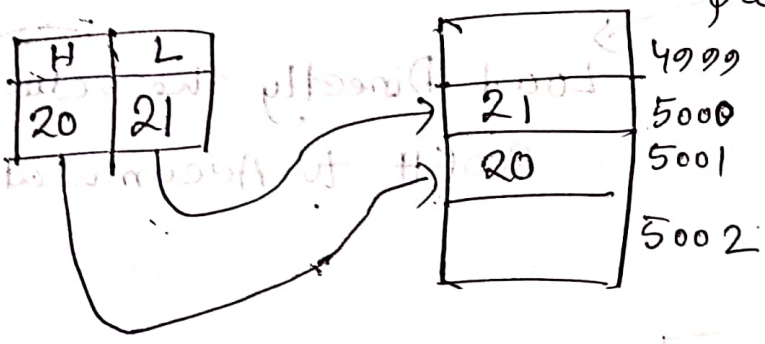
(viii) STA 3000H; → Store the value of Accumulator in 3000H (Address)



(ix) LHLD 5000H → Load the values of 5000H in HL pair directly



(x) SHLD 5000H → Store the values OF HL pair in 5000H (Address)

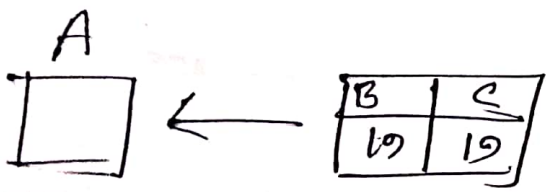


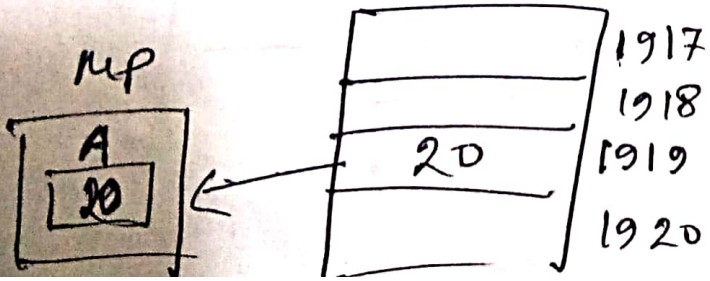
L (Lowest) → 5000

~~High~~

H (Highest) → 5001

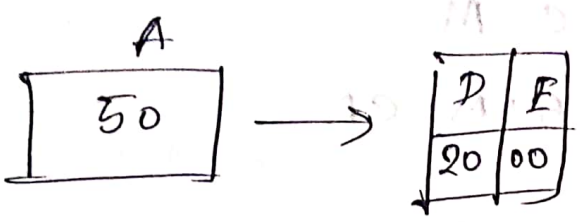
(xi) LDAX B; → Load the value to Accumulator from B pair.

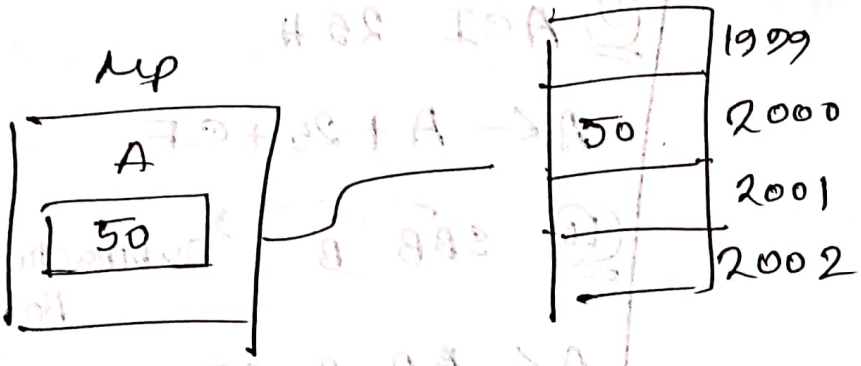




(xii) STAX D;

→ Store the value to pair from Accumulator





(xiii) PCHL →

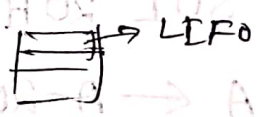
PC = program Counter

PC ← HL

(xiv) SPHL →

SP = Stack pointer

SP ← HL



Arithmatic Group

① ADD B

Accumulator A ← A + B

② ADD M

A ← A + M

③ ADI 25H

A ← A + 25H

④ SUB B

A ← A - B

⑤ SUB M

A ← A - M

⑥ SUI 20H

A ← A - 20H

⑦ ADC B → Add with carry

A ← A + B + CF

↳ Carry Flag

⑧ ADC M

A ← A + M + CF

⑨ ACI 25H

A ← A + 25 + CF.

⑩ SBB B → Subtraction Borrow

A ← BA - B - CF

⑪ SBB M

A ← A - M - CF

⑫ SBI 25

A ← A - 25 - CF

⑬ INR B

B ← B + 1

⑭ INX B

B ← BC + 1

Note :-

ADC use করি যখন দুইটি ~~registration~~ register এর value যোগ করি আর Carry পরবর্তী দুইটি register এর সাথে যোগ করা হয়।

(15) INR M

M ⟹ M ← M+1

(16) DCR B

B ← B-1

(17) DCX B

BC ← BC-1

(18) DCR M

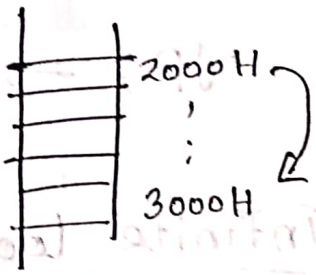
M ← M-1

Branch Control Group

① JMP (Jump)

(i) Uncod. Unconditional Jump

JMP 3000H



(ii) Conditional Jump

(Flag দিয়ে condition দিতে পারি)

(a) JC 3000H

→ Jump if carry CF=0

CF=1

(b) JNC (Based on Carry Flag)

Jump if no Carry

(iii) Zero Flag :-

@ JZ : Jump if result is zero

(b) JNZ: Jump if result is not zero

(iv) Parity Flag:-

(a) JPE ; → Jump if parity flag is even

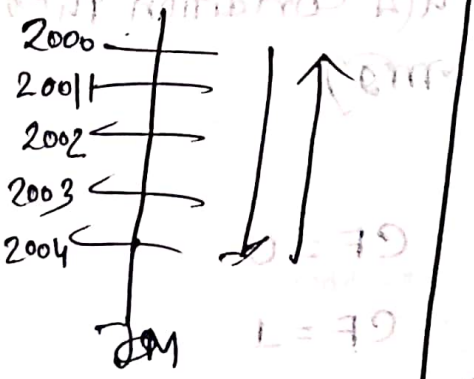
(b) JPO ; → Jump if parity flag is Odd

(v) Sign flag:-

JM; → Jump if result is Negative/Minus

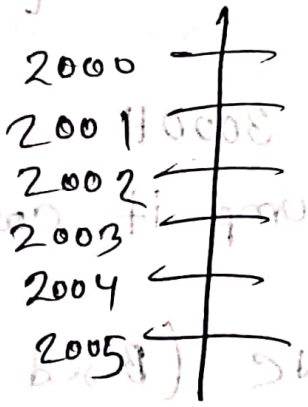
JP; → Jump if result is positive

Infinite loop:-



JMP 2004H

Finite loop:-



MVI A 05H

DCR A

JNZ 2001

Q write a program that has Finite loop with 10 iteration.

→ MVI C, 0AH

Back: DCR C

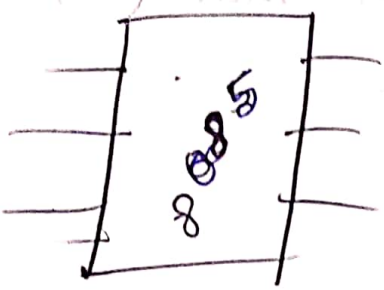
JNZ Back

HLT

HLT → To Finish a program

Address	Memory	Hexadecimal Code
8000H	MVI A	3E
8001H	04	04
8002 H	MVI B	06
8003 H	03	03
8004 H	ADD B	80
8005 H	STA	32
8006 H	00	00
8007 H	20	20
8008 H	HLT	76

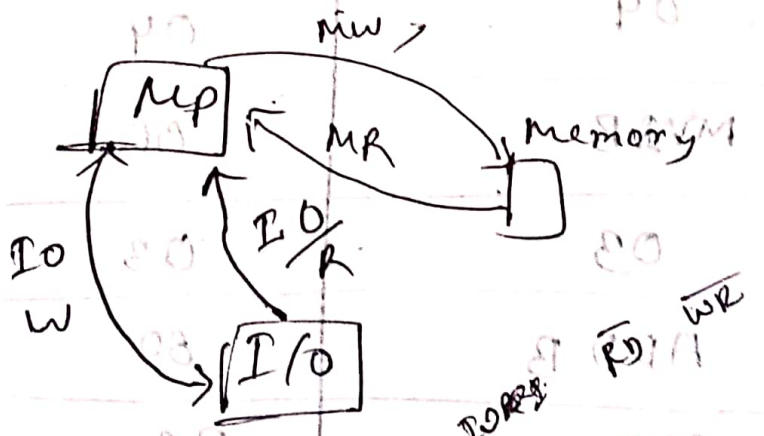
P1



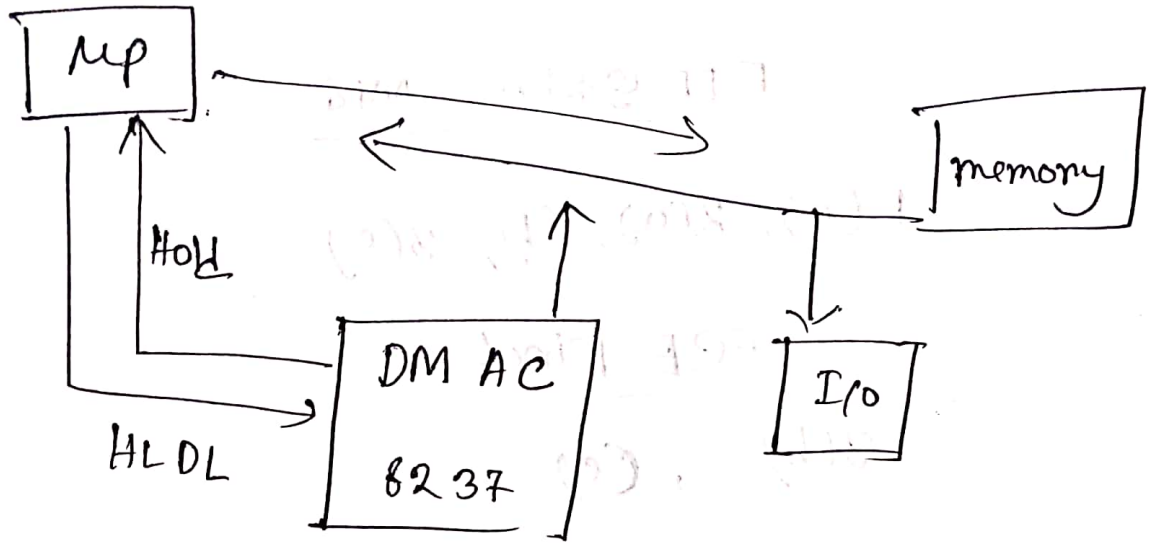
4 2 1 0 4 2 1 0
1 1 1 1 1 1 1
$(77)_{10}$

Input / output ও নিয়ন্ত্রক Pin গুলোর অবস্থা:-

	IO	$\overline{RD}$	$\overline{WR}$	S_0	S_1
MR	0	0	1	0	1
MW	0	1	0	1	0
IOR	1	0	1	0	1
IOW	1	1	0	1	0



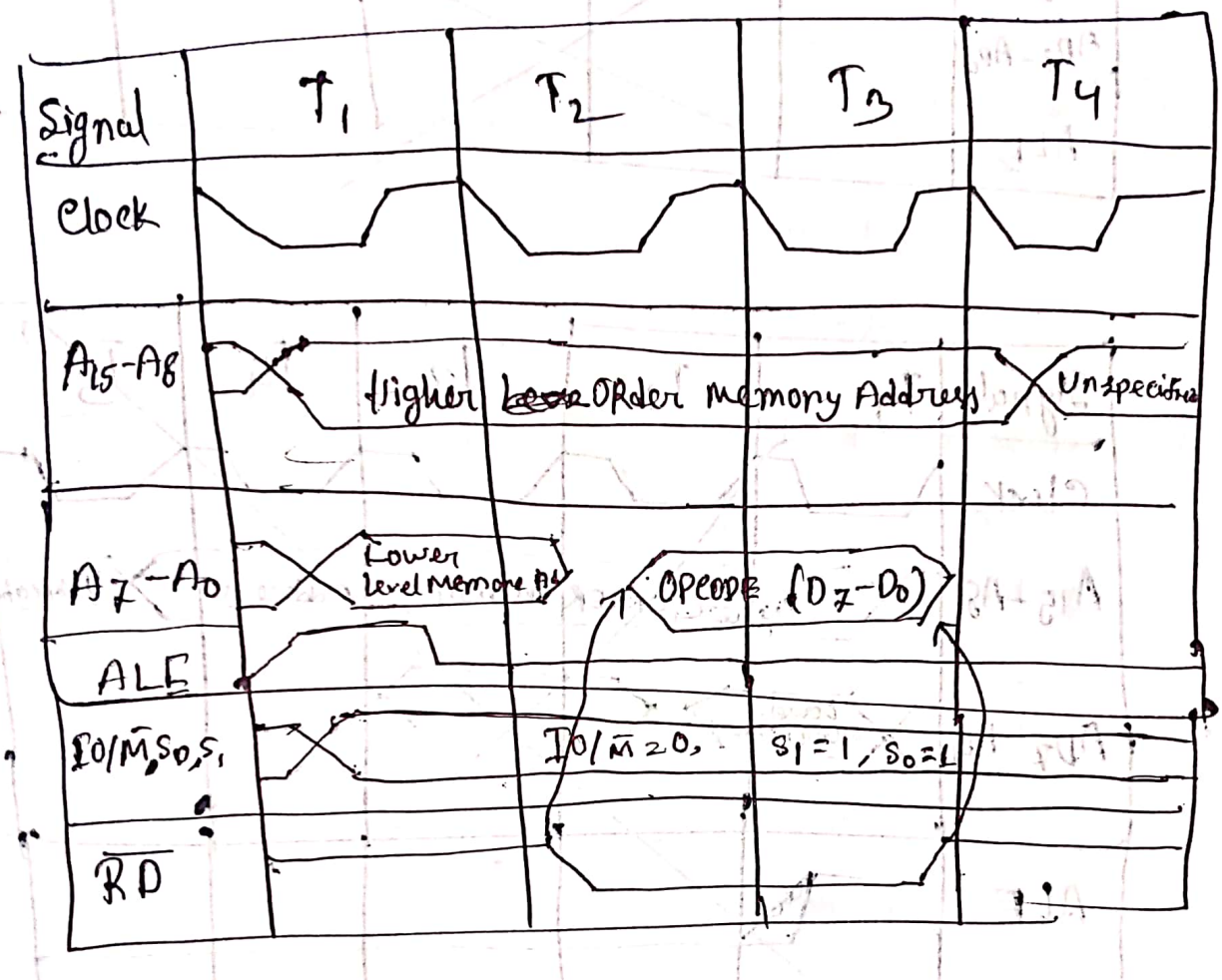
Concept OF DMA :-

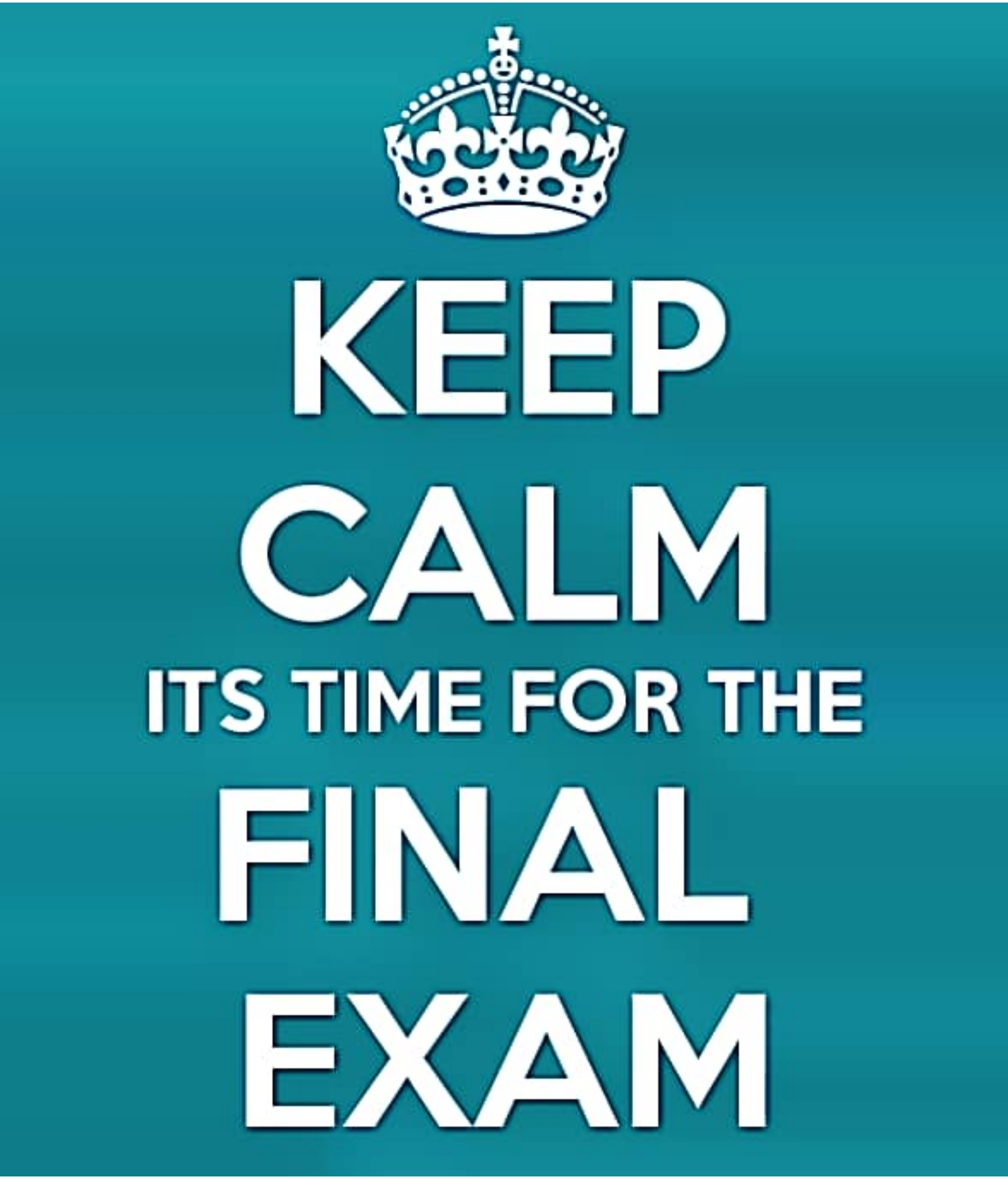


program :-

Save the value using memory pointer :-

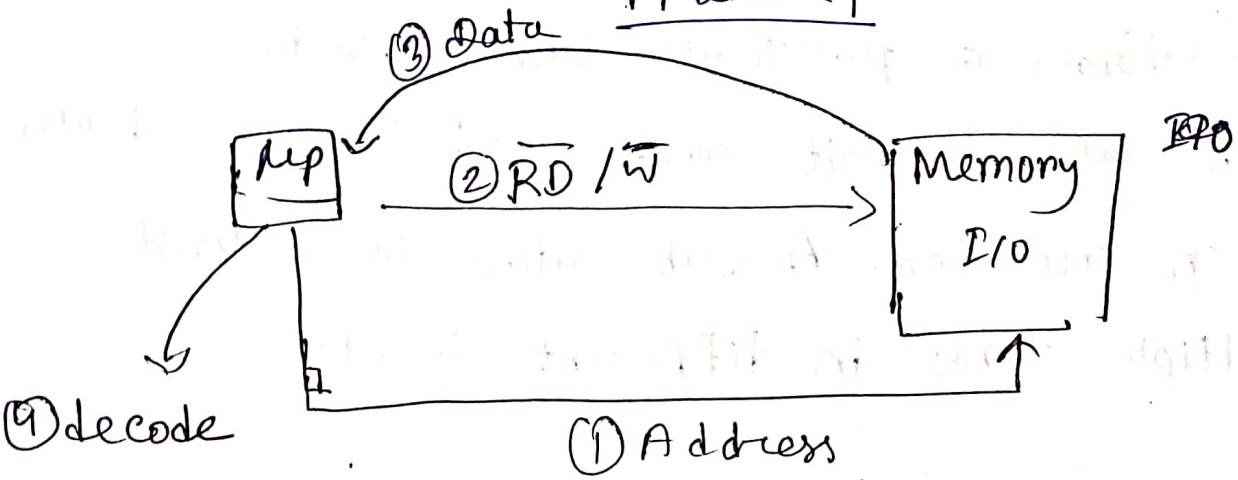
```
MVI  C, 00H ;
LXI  H, 2000H
MOV  A, M ;
INX  H ;
ADD  M ;
JNC  SKIP ;
INR  C ;
SKIP:
    INX  H ;
    MOV  M, A ;
    INX  M ;
    MOV  M, C ;
    HLT
```


Signal
T1
T2
T3
T4
Clock
A15-A8
Higher Order memory Address
Unspecified
A7-A0
Lower Level Memory Address
Opcode (D7-D0)
ALE
IO/M, S0, S1
IO/M = 0,
S1 = 1, S0 = 1
RD


KEEP
CALM
ITS TIME FOR THE
FINAL
EXAM

Final Exam

T-State



MR → 3T (clock pulse) (1, 2, 3)
(Memory Read)

MW
(Memory write) → 3T (clock pulse) (1, 2, 3)

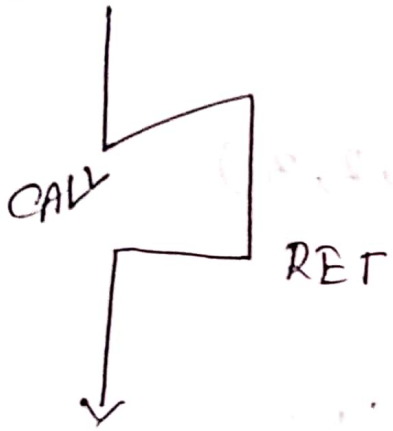
IOR
(Input output Read) → 3T (clock pulse) (1, 2, 3)

IOW
(Input output write) → 4T
- RD / W
- Address
- Data
- Decode

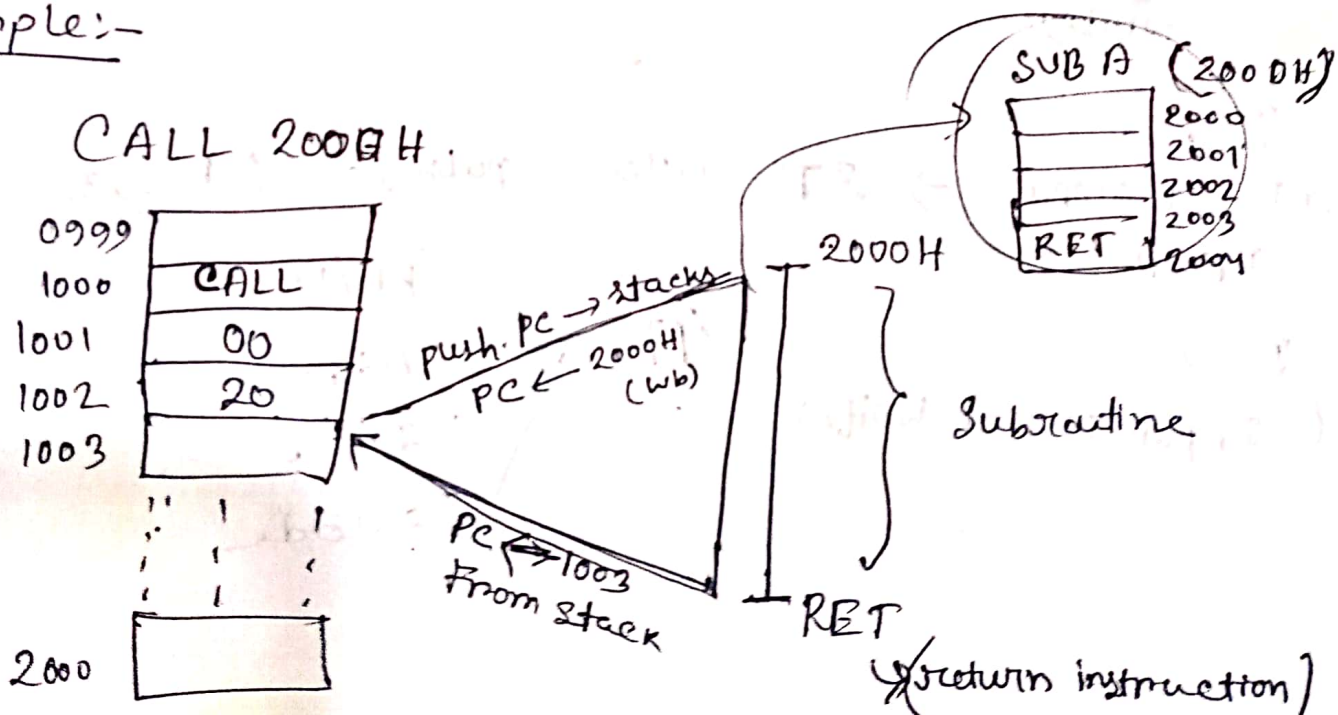
SubRoutine:-

It is a sequence of program instructions code that performs a particular task. This is packaged as a unit and used in a particular task when needed. A subroutine is a unit in multiple times in different location.

Here



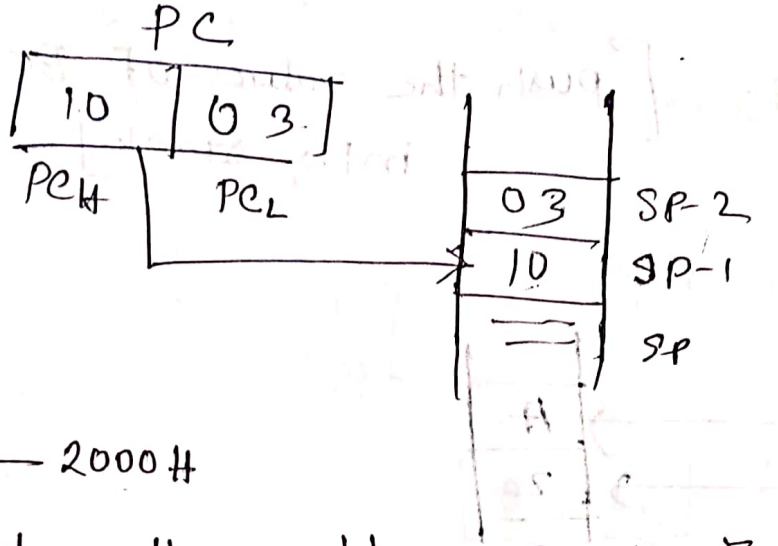
Example:-



Here, when PC comes to 2000 gets subroutine. But PC is already incremented by 1 to 2001. So this address needs to be saved in stack as 1002 (02 20). Then PC will be freed to go to the address, 2000H.

During CALL:-

① Push PC → Stack

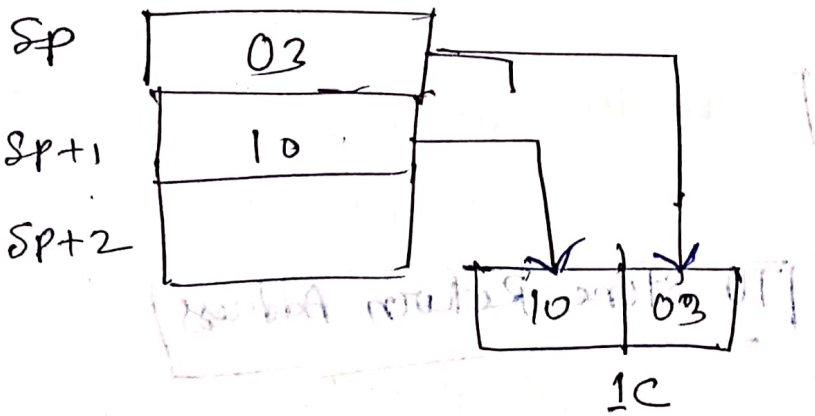


② PC ← 2000H

PC gets the address 2000H from WZ register.

During RET:-

POP PC ← Stack

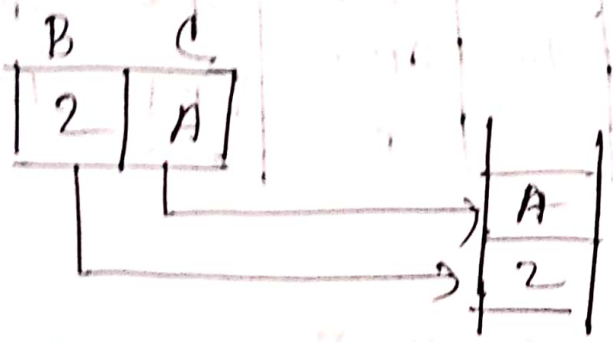


Use of Stack

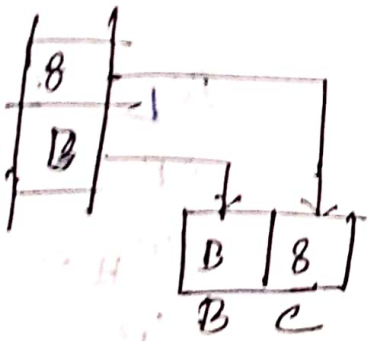
(1) To Store Data:

~~Push B~~

(a) PUSH B; [push the values of BC pair into stack]



(b) POP B; [push the top 2 values of stack into BC pair]



TO Store Return Address

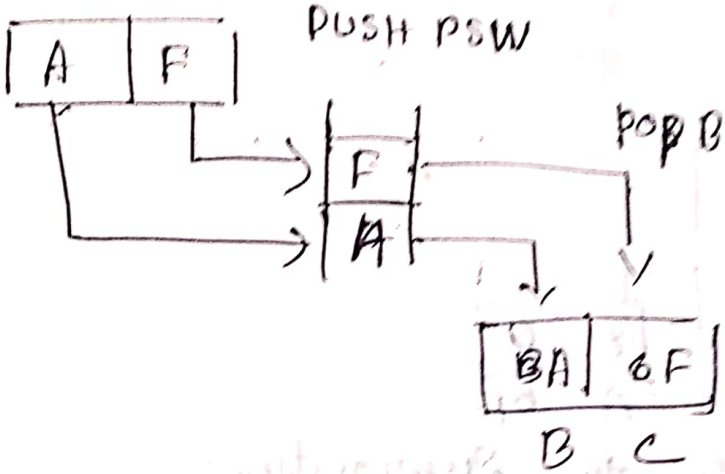
(a) During any the execution or CALL instruction

TO read / write Flag:-

Read Flag:-

PUSH PSW → PUSH the value of A & F into stack

POP B → Pop B.



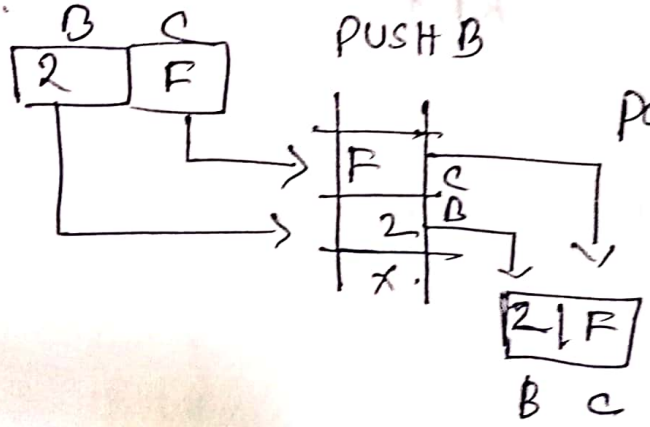
Write Flag:-

PUSH B

POP PSW

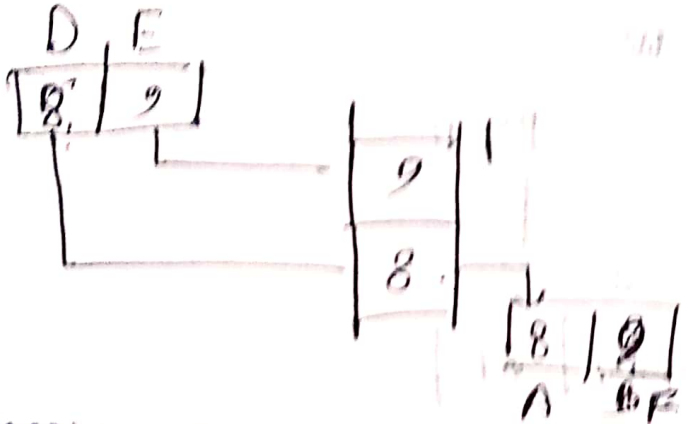
POP the top 2 values of stack

PUSH into the A & C



(The contents are then popped into the PSW, and the byte that was originally loaded into C register is now written into the Flag register)

```
MVI E, 91H
MVI D, 85H
PUSH D
POP PSW
```



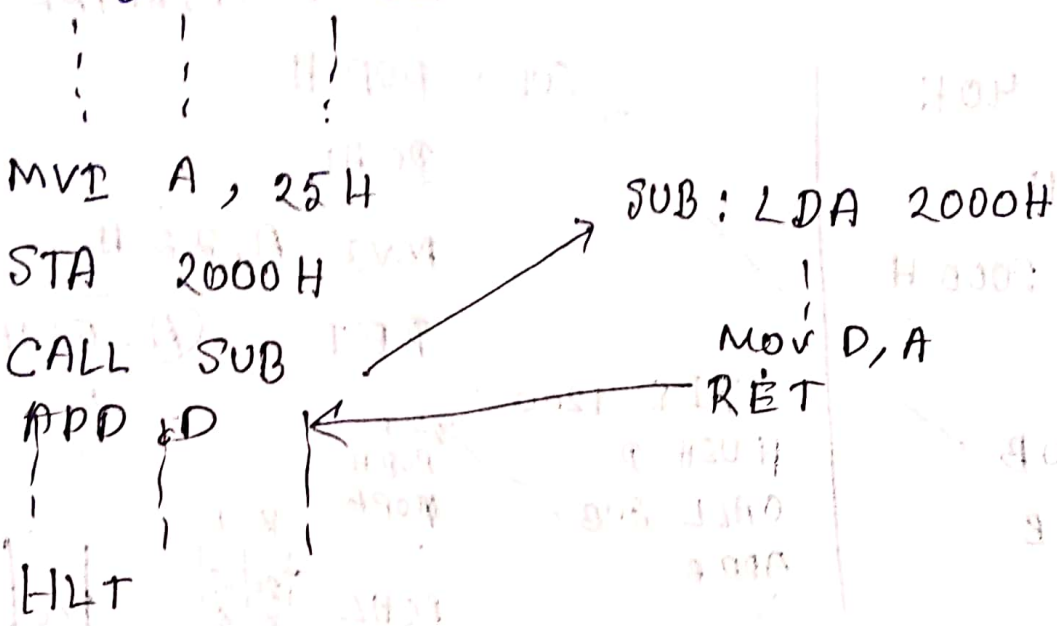
Passing parameter to Subroutine:

(1) Using Register:

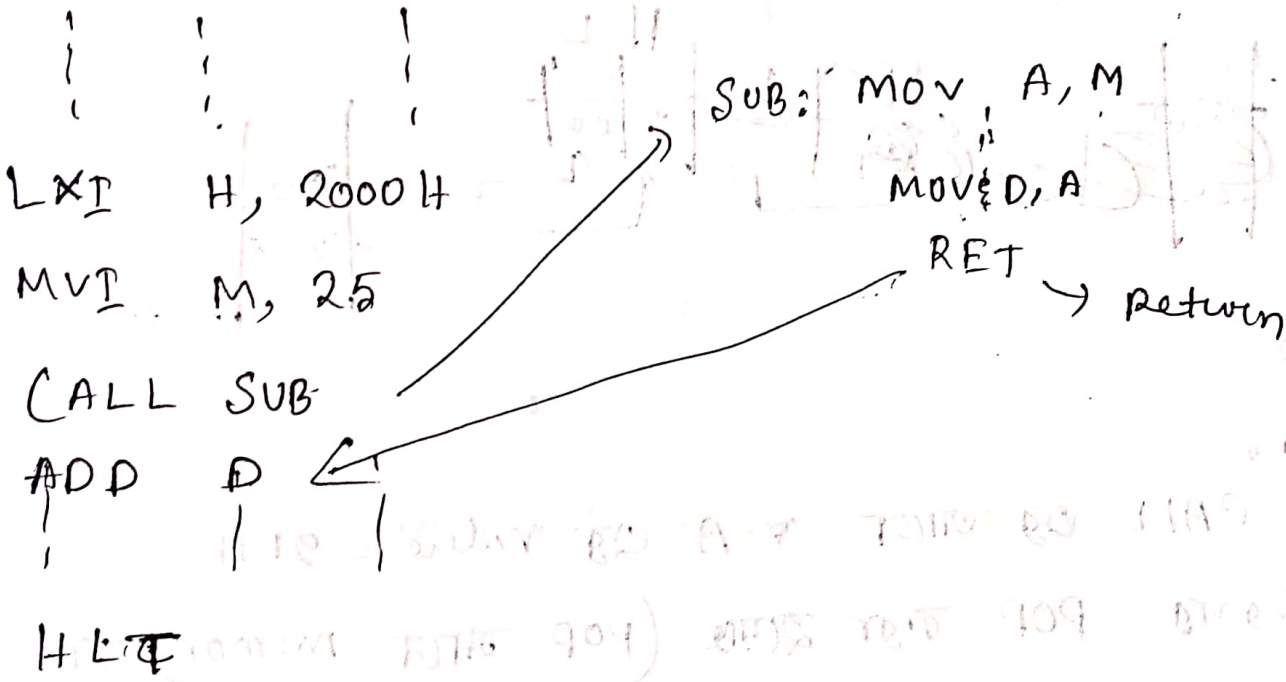
```
        |   |   |
        |   |   |
        |   |   |
MVI C, 05H
CALL SUB
        |   |   |
        |   |   |
        |   |   |
HLT
```

```
SUB : MOV A, C
        |
        |
      RET
```

② Using Memory:-



③ Using Memory pointer:-



⇒ Value to be passed is moved into the memory pointed "M". There might be any random value. Then the subroutine accesses the value from M.
Care has to be taken that after the value is put into M the value of HL pair not to be changed until subroutine accesses the value of M.

(4) Using Stack:-

we can POP out 2 bytes from the top of the stack. Register uses :- BC, DE, HL, AF

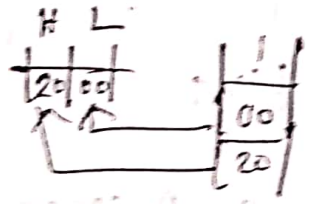
```
MVI A, 40H
ADI 50H
LXI H, 2000H
INR A
CALL SUB
ADD B
```

```
SUB: POP H
     PCHL
     MVI A, 55H
     RET
```

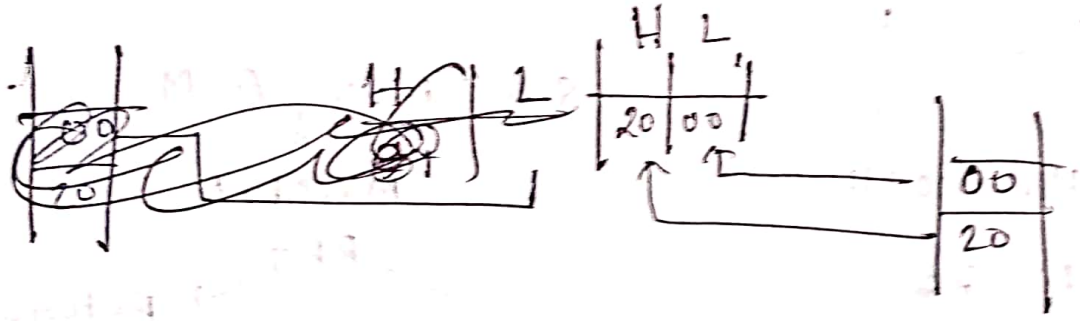
A = 55H

```
LXI D 1200
PUSH D
CALL SUB
ADD B
:
HLT
```

```
SUB:
POP H
POP B
:
PCHL
```



50H + 40H = 90H + 1 = 91H



এখানে,

SUB CALL এর আগে ∴ A এর value = 91H

এরপর POP করা হয়েছে (POP মানে memory হতে value কে pair এ নিয়ে আসা), তারপর Accumulator এ 55H value আনা হয়েছে।

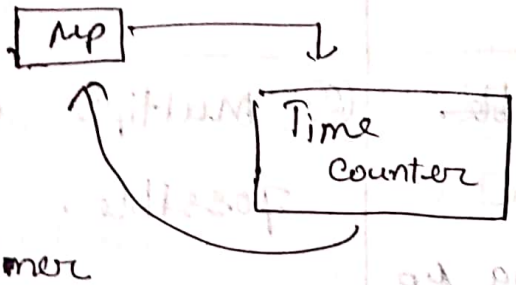
সুতরাং A এর value 91H হতে 55H হয়েছে।

Delay & Delay Routine:-

Video :- Bharat Acharya

(1) Software Delay:- It is a dummy loop written by a programmer. It is cheaper. Simple circuit, Flexible.

(2) Hardware Delay:-



* Bigger timer we will give, the bigger delay we will get.

Hardware one is "a physical chip producing a delay." It is expensive. Complex circuit, less Flexible.

The biggest ~~even~~ advantage of hardware delay is:-

Can save clocks during multi processing situations.

Difference between Software & Hardware delay:-

Software Delay	Hardware Delay
(1) MP free থাকে না delay এর সময়ে।	(1) এখানে MP free থাকে। কারণ delay টা তখন করে Hardware দ্বারা করা হয়।
(2) Multiple Delay ~~possible~~ impossible. Cause the delay is happening using MP.	(2) Multiple delay possible.
(3) It is not expensive	(3) It is expensive
(4) It is easy	(4) It is complex

Software Delay

It is three types:-

1. Using NOP instruction (No operation) [Nested Loop]
2. Using 8-bit Register
3. Using 16-bit Register

(1) Using single 8-bit Register:-

```
        MVI B, 0AH      | 7T
REPEAT: DCR B           | 4T
        JNZ REPEAT      | 10T / 7T
```

Total time delay inside the loop = (4T + 10T) × 10T

= 14T × 10T

= 140T states

= 140T − 3T

= 137T states

In this program in DCR, B it's 4T

JNZ Repeat = 10T

Total 14 T will run for 0A H time which is 10 times.

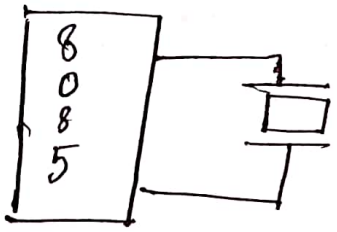
But in the last step where JNZ will be False we will have ~~7T~~ 10T - 7T ; So, 10T - 7T = 3T

So, 140T - 3T = 137T states

Total delay generated = delay outside the loop + delay inside the loop

= (137T + 7T) states

= 144T states



$F_{in} = \frac{2}{2} MHz = 1 MHz$

$T = 1/F_{in} = 1\ \mu sec$

144T states = 144 μsecs

Example-02:-

255 → FF

```
MVI C, FF
BACK: DCR C → 4T
      JNZ BACK
      4T
      RET
      10T
```

6+4 = 10T ←

3 bit এর জন্য Delay.

→ Memory Tstate = 3T

→ 4T = (4+3)T = 7T.

→ Address (Address 16 bit
8 bit এর read
করতে 2 বার
3×2 = 6T)

Total Delay = 255 × 14T × 0.33 μs

= 1181.73 μs =

10T = 10 × 0.33 μs =

4T = 4 × 0.33 μs =

Tstate
Memory Read → 3T
Memory write → 3T
Input/output Read → 3T
" " write ⇒ 3T
Opcode Fetch → 4T
RET → Return এড্রেস আসে Stack থেকে PC তে value আসে 2 বার।

16-bit Register:- (Delay)

```
LXI B, Count; → 10T
Back: DCX B ; → 6T
      MOV A,B ; → 4T
      ORA, C ; → 4T
      JNZ BACK; → 10T
      RET → 10T
```

24T

Here,
Count = FFFF = 65,535

B C
FF FF = 3T + 3T + 4T = 10T

DCX B = FFFF - 1

FF FE ← 3+3 = 6T

1, → এবার কাউন্টার হবে FFFE

FF (A) OR, FE (C) = 1

↓

আর এটা ততক্ষণ চলবে যতক্ষণ না 00 হয়।

Total T state,

→ $10 \times 0.33\mu s = 3.3\mu s$

→ $65535 \times 24T$

$= 65535 \times 24 \times 0.33$

$= 519043.8\mu s = 0.5190438s = 0.52s$

→ $10 \times 0.33 = 3.3\mu s$

অর্থাৎ যদি এটা 2 বার চালায় তবে এটা, অর্থাৎ, 120s চালালে 1 minute delay হবে।

RET 10T কেন?

=> মূলত PC কে Memory এ Address 2 বার আনতে হয়েছে 16 bit register।

So Memory Read ও দুইবার হচ্ছে।

MR × 3 2 = 3T × 2 = 6T

আবার RET একটা opcode fetch। তাই এটা = 4T

∴ 6T + 4T = 10T

Interrupt

Special Condition that makes up execute an ISR.

Hardware → ২ types

→ RST

* Software interrupt কেবল দিলে Hardware interrupt disable করা যায়।

* Vector → যে ISR এর address fixed

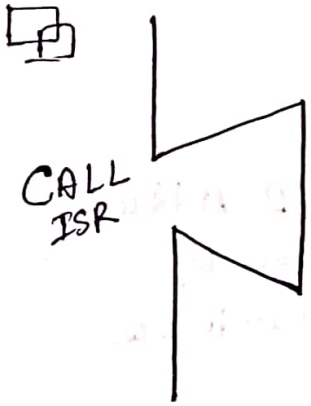
Trigger → H trigger

→ Label trigger

E → H → 0 to 1
L → Lable वाला अक्षर

Interrupt	priority	Triggering	Maskable by SIM	Disable by DI	vector	vector Address
TRAP	1	E/L	NO	NO	y	0024 H
RST 7.5	2	E	y	y	y	003CH
RST 6.5	3	L	y	y	y	0034 H
RST 5.5	4	L	y	y	y	002CH
INTR	5	L	NO	y	NO	GET ISR Address From External Hardware

~~Sir ~~ pdf~~



ISR → Sub Routine
→ Interrupt वाला Subroutine Call वाला

SIM instruction - 8 bit (Set Interrupt Mask)

SOD	SDE	xxx	R7.5	MSE	M 7.5	M 6.5	M 5.5
					1	0	1

Interrupt Structure

(5 marks का पूछा जाता है)

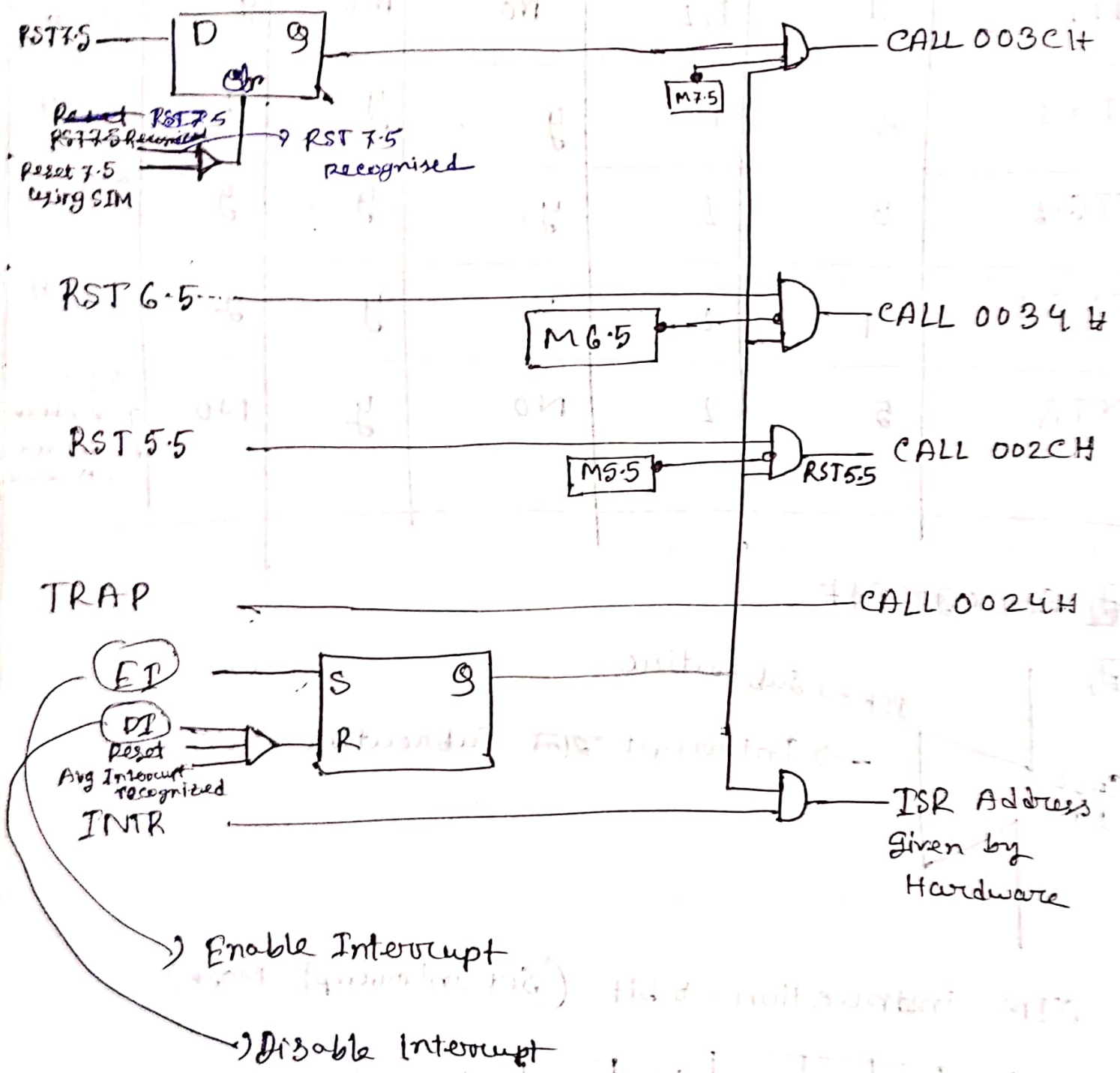


Figure:- Interrupt Structure

SIM :- (Set Interrupt Mask)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SOD	SDE	X	R 7.5	MSE	M 7.5	M 6.5	M 5.5

- **SOD** — Serial Output Data
  - 1 → Send 1 on SOD
  - 0 → Send 0 on SOD
- **SDE** — Serial Data Enable
  - 1 → SOD active
  - 0 → Noting
- **Reset 7.5** if bit is,
  - 1 → Reset
  - 0 → Nothing (Not Reset)
- **Masking set Enable**
  - 1 → Enable Mask
  - 0 → Nothing
- **M7.5 = 0** হলে Signal হলে Allow
  - 1 হলে Masked or Blocked
- **M6.5 = 1** হলে Masked
  - 0 হলে Allow

Question

Write an opcode to Mask 7.5, 5.5 using SIM instruction

SOD :- এর দ্বারা microprocessor কে serially output দিতে পারে

SDE :- যেহেতু 8085 serial Communication নয় তাই এখানে serial Data Enable করতে হয়। যা SDE দ্বারা করা হয়।

RST 7.5 :- এটাই একমাত্র যেখানে Reset অপশন আছে। এটা Hardware Interrupt কে reset করতে পারি। কারণ এখানে D-Flip Flop আছে।

MSE :- যদি MSE-1 হয় তবে M7.5, M6.5, M5.5 (যেগুলো Masking) কাজ করবে। 0 হলে কোনোটিই কাজ করবে না।

M-7.5, M-6.5, M-5.5 :-

যদি এদের signal 1 হয় তবে ~~কাজ হবে~~ Masked (Block) হবে 1 হলে হবে না।

Interrupt (understanding)

=) কোনো program কে কিছু নির্দিষ্ট সময়ের জন্য block করতে Interrupt ব্যবহৃত হয়। দুইভাবে হয়। Hardware & Software।

Hardware : keyboard এ কোনো key press করা।

~~Software~~ :- New System hit হলে fan on হওয়া।
Mouse এ কাজ করা।

8085 এ :-

TRAP, RST 7.5
RST 6.5
RST 5.5
INTR

} এগুলো pin গুলো একসাথে লাগানো যখন এই pin এর উপর কোনো কিছু আসবে তখন Interrupt হবে। তো এটা যতক্ষন কাজ করবে ততক্ষনই Interrupt হতে থাকবে।

Software Interrupt :-

Software or program এর দ্বারা সৃষ্ট Interrupt

RST 0 হতে RST 7 পর্যন্ত Software interrupt হয়।

চাইলে program এ

vectored Interrupts :-

vector → একটা Fixed Address এ কিছু program রাখা থাকে। তো যখন Interrupt হবে তখন ঐ Address এ যেই program রাখা আছে তা execute করবে।

Non-vector → এখানে কোনো Address Fixed থাকবে না। এখানে, কোনো particular Address এর program গুলো execute হবে।

⇒ RST 7.5
RST 6.5
RST 5.5
TNR
} vector Interrupt

TRAP → Non-vector Interrupt

Non-Maskable Interrupt :-

স্বাভাবিকভাবে, Interrupt গুলো µp বর্তমান কাজকে বাদ দিয়ে অন্য কাজ করবে। কিন্তু Non-Maskable Interrupt গুলো সেটাকে µp কখনও Ignore করতে পারে না।

Maskable Interrupt :-

আর এক্ষেত্রে µp চাইলে Ignore করতে পারে। অর্থাৎ কোনো কাজ করতে থাকলে µp তা করে তারপর Interrupt

বাড়তে পারে।

TRAP → ~~Maskable Interrupt~~ Non-maskable
(System এ Hit হলে, তখন Interrupt করে Fan চালানো। এক্ষেত্রে তাৎক্ষণাৎ Fan টি চালাতে হবে।)

Maskable হলো mask লাগানো → Ignore করা যায়

Non-Maskable হলো Mask না লাগানো → Ignore করা যায় না।

Maskable → Low priority একারণে এটাকে ignore করা সম্ভব।

Question

Explain the execution of subroutine:

Question

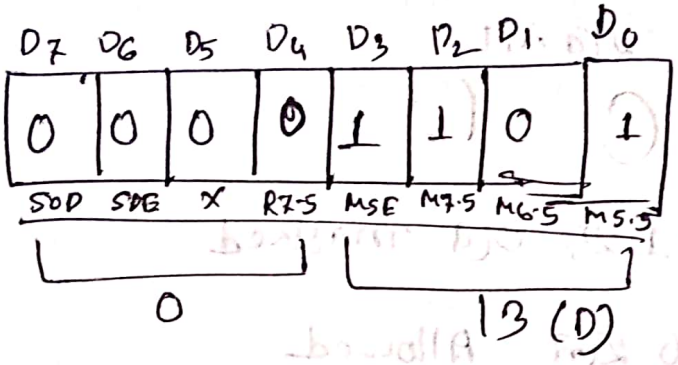
Difference between CALL & JUMP.

SIM understanding

~~Set Interrupt Mask~~

write a program that will allow 6.5 & block the other interrupt.

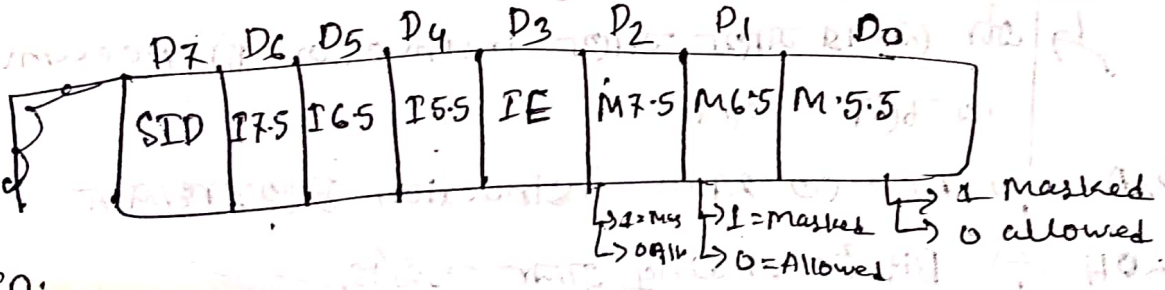
⇒




```
MVI A, 0D
SIM
```

RIM (Read Interrupt Mask)

Status রিড read করে



SID:

SID :- Serial Input Data. RIM এর মধ্যে MP কে Input নিতে চাইলে SID Pin use হয়। অর্থাৎ এখানে Read হয় যার এর মাধ্যমে Input signal Input read করে।

I7.5 :- Interrupt 7.5 ⇒ indicate if 7.5 interrupt pending কি না।

I6.5 & I5.5 ও same pending interrupt check করে।

IE :- Interrupt Enable. যদি bit 1 হয় তবে enable আর 0 হলে enable হবে না।

M7.5, M6.5, M5.5 :-

যদি এদের signal 1 হয় তবে masked

আর 0 হলে Allowed

Example: Check whether 6.5 is pending. If it is pending enable 6.5 without effecting any other interrupt.

=>

	D7	D6	D5	D4	D3	D2	D1	D0
RIM	SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
	0	0	1	0	0	0	0	0
			2		0			

Code:-

RIM → এটার সাথে যাবে Instruction টা Accumulator এ চলে যাবে

MOV B, A → Acc তে থাকা instruction B তে রাখবে

ANI 20H → Bit টা pending আছে কিনা,

JNZ Next → I6.5 যদি 0 হয় তবে ফলাফল 0

EI → Interrupt enable

RET → I6.5 যদি 1 হয় তবে ফলাফল 20

Next:- MOV A, B → B তে যা রেখেছিলাম তা ফিরিয়ে আনলাম

ANI 0D

OR করবে ORI 08H

SIM

SIM

	SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5
	0	0	0	0	1	1	0	1
		0				D		

এখানে string value MSE তে 0 না করলে বা 1 থাকে বাকি সব 0 হবে, ফলে Bit value ভুল

Courtesy: Youtube

Create a Delay about 1 sec Using 8085 µP

MD Iftakhar Kabir Sakur
E201037

```
MVI  C, x H   → (7T)
LOOP1: LXI H, y. H  → Loading in HL pair → (10T)
LOOP: NOP     → No operation → 4T
      DCX H   → Register pair decreased → 6T
      MOV A, H → 4T
      ORA L    → 4T
      JNZ LOOP → 10/7T
      DCR C    → 4T
      JNZ LOOP1 → 10/7T
      HLT      → 4T
```

এখানে x or y এর value ৪ bit 2 digit হিসেবে বসাতে হবে।

We know,

1T state = 0.33×10^{-6} sec

$\therefore 1 \text{ sec} = \frac{1}{0.33 \times 10^{-6}}$ = $3.03030.. \times 10^6$

$\therefore 1 \text{ hr} = 3030303.03$

So program অনুযায়ী, $7 + x(10 + y(4+6+4+4+10) - 3 + 4 + 10) - 3 + 4$

Last ↑ unsuccessful

Last ↑ unsuccessful

$= 3030303$

$\Rightarrow 7 + x + (10 + 28y + 11) + 1 = 3030303$

$\Rightarrow 8+x(21+28y) = 3030303$

$\Rightarrow x(21+28y) = 3030295$

⇒ যেহেতু,

$x = FF$ or 255

$\therefore 21+28y = (3030295)/255$

$= 11883.50$

$\Rightarrow 28y = 11862.50$

$\Rightarrow y \cong 423$

y Decimal Number কে Hexa তে convert করতে হবে

$(423)_{16}$

16 | 423
16 | 26 - 7A
1 - A

$= (01A7)_{16}$

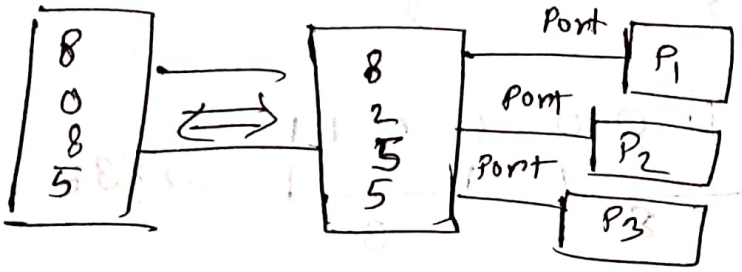
$\therefore$ $x = 255\ H = 0FFH$

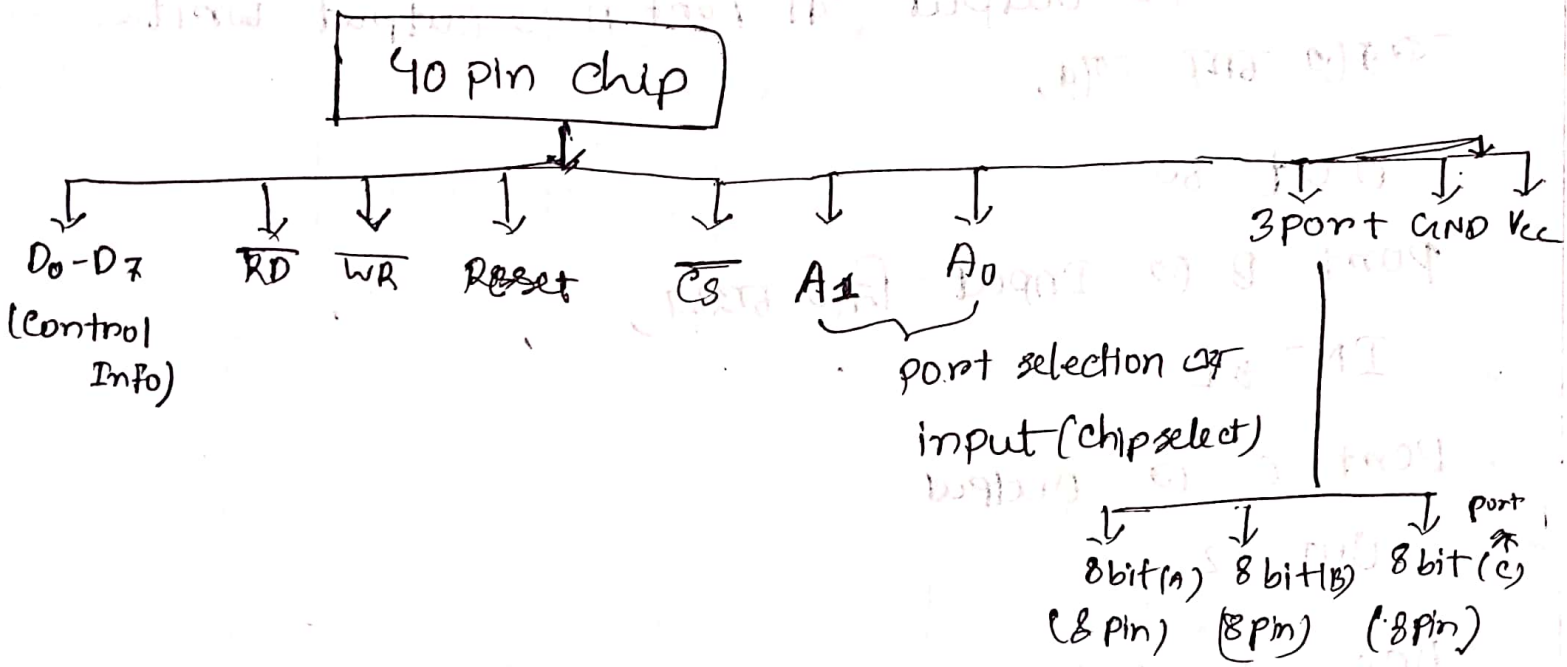
$y = 01A7\ H$

[এখানে x কে নিজের মধ্যে value ধরে হয়েছে larger value ধরতে হবে]

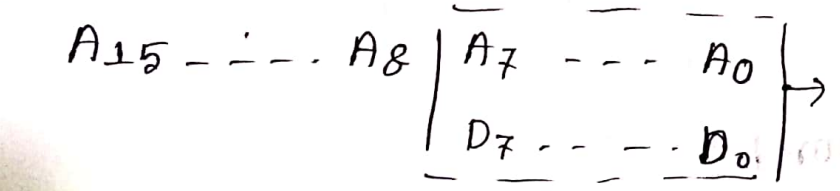
8255 programmable peripheral Interface

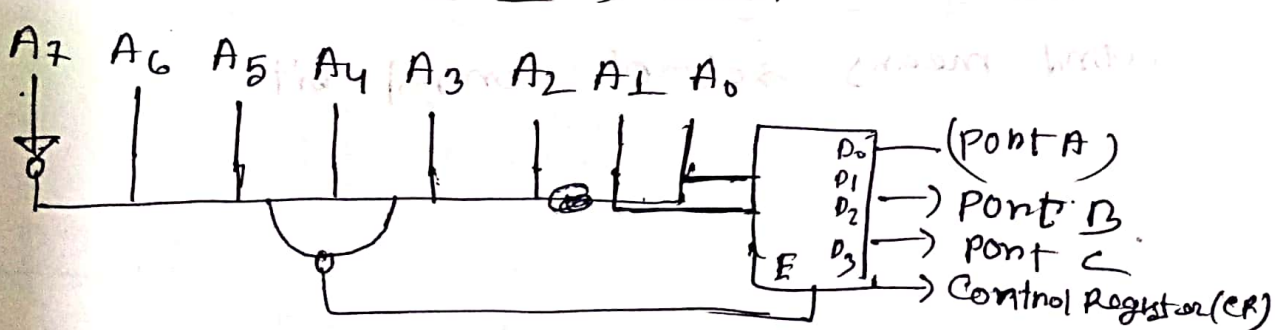
8085 Communicate করে 8255 এর সাথে, 8255 এর সাথে আছে বিভিন্ন PPI আছে।





In 8085 :-





Port A enable করতে চাইলে, 1 0 0 0 0 0 0 0 → 80H (Port A এর Address

Port B এর কন্ট্রো!- 1 0 0 0 0 0 0 1 ⇒ 81 H
8 1

Port C " " = 1 0 0 0 0 0 1 0 ⇒ 82H
8 2

Control Register = 1 0 0 0 0 0 1 1 ⇒ 83H
8 3

8255,
Port A কে output বা Port A কে output write করতে চাচ্ছ তবে,

Out 80

Port B কে Input নিতে চাইলে,

IN 81

Port C কে output

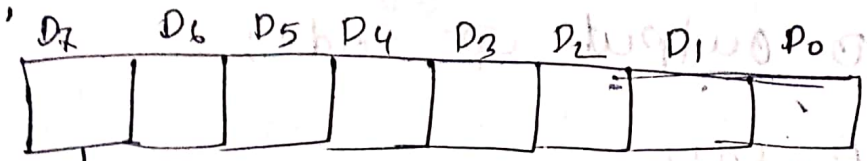
OUT 82

Acc কে output

OUT 83

Control word :-

word means set of control bit



1 (I/O) mode

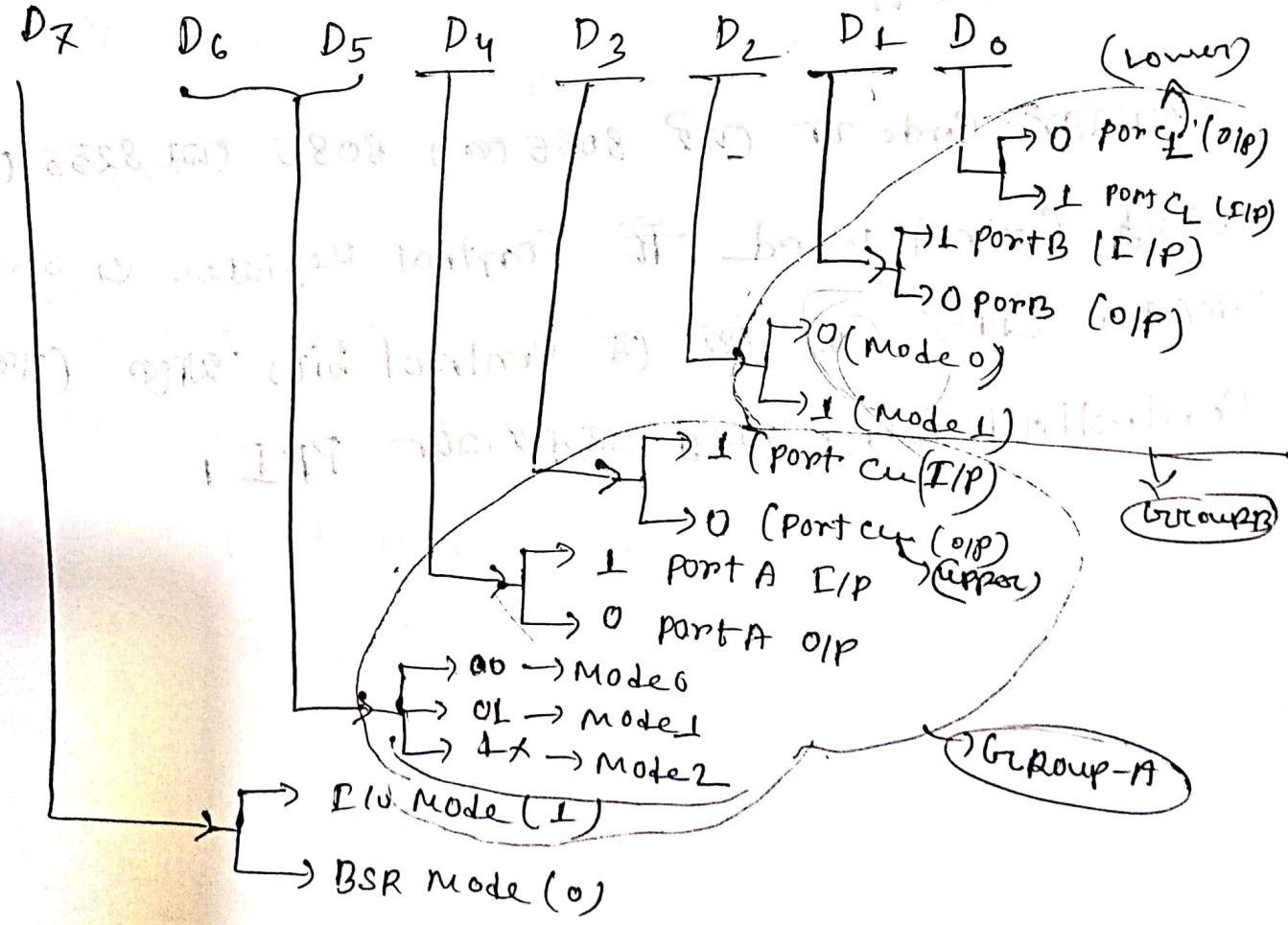
0 (BSR Mode)
Bit set Reset Mode

Mode 0 | Mode 1 | Mode 2

Simple I/O mode
A, B, C

Handshake I/O
A, B
Port C is used for handshaking

(Bidirectional I/O)
A



Port A কে output এর code:-

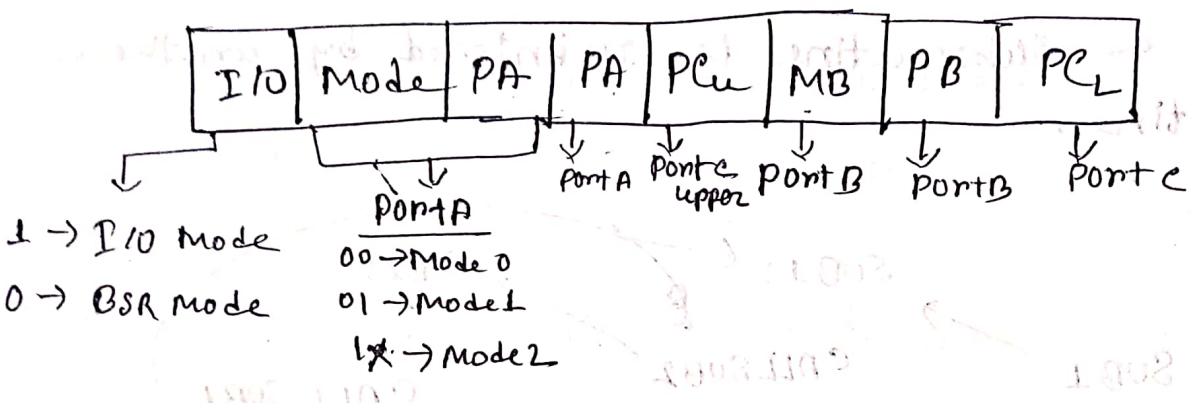
```
MVI A, 80H
OUT 83H  → Control Register
MVI A, FF
OUT 80H
```

Port B থেকে Read

```
MVI A, 82H
OUT 83H
IN 81H
```

আমরা Code টা দেই 8085 কে, 8085 দেয় 8255 কে, 8255 Control word টি Control Register এ save করে। এবং CW এ যে Control bits থাকে সেগুলো Controlled হয়। আর একটাই এটা PPI।

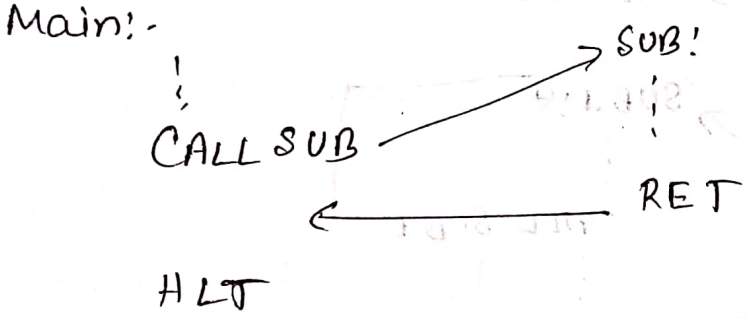
8255



Type of Sub-routine:-

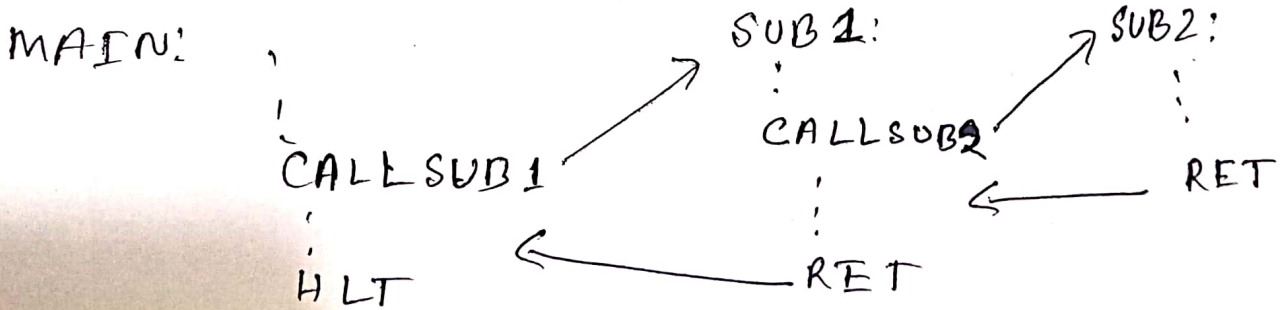
① Simple Subroutine:-

This one does not call another subroutine.



② Nested Subroutine:-

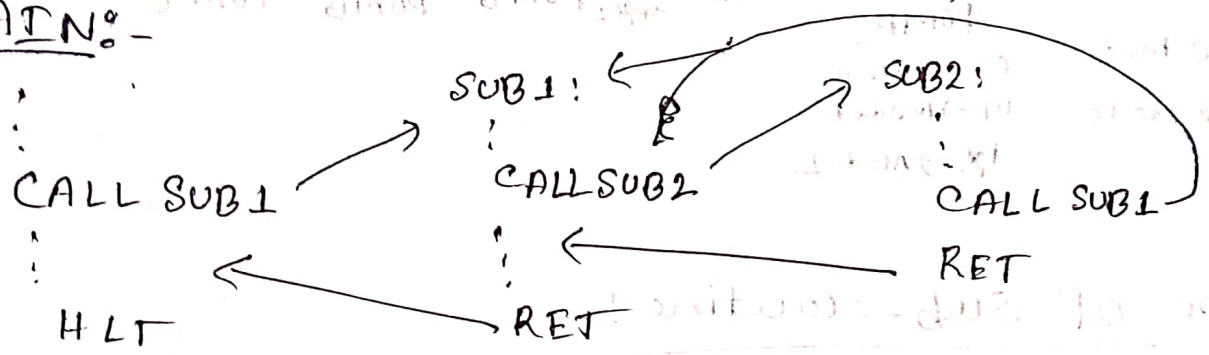
A subroutine calls another subroutine



(3) Re-entrant Subroutines :-

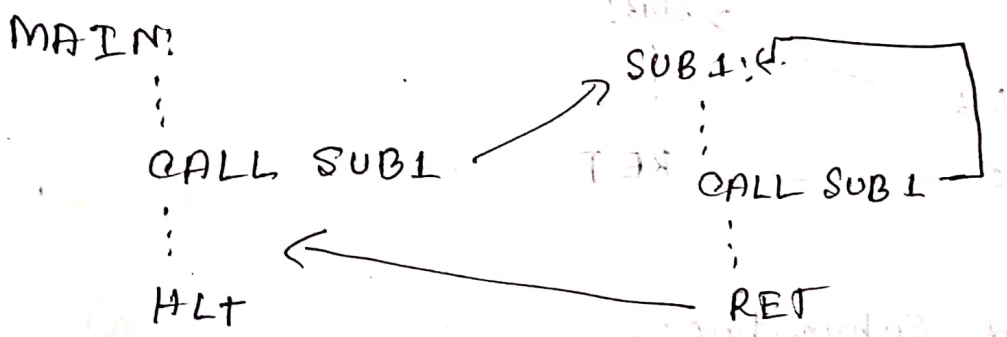
When a subroutine is reentered by another subroutine.

MAIN :-



(4) Recursive Subroutines :-

No. + Subroutine calls itself.

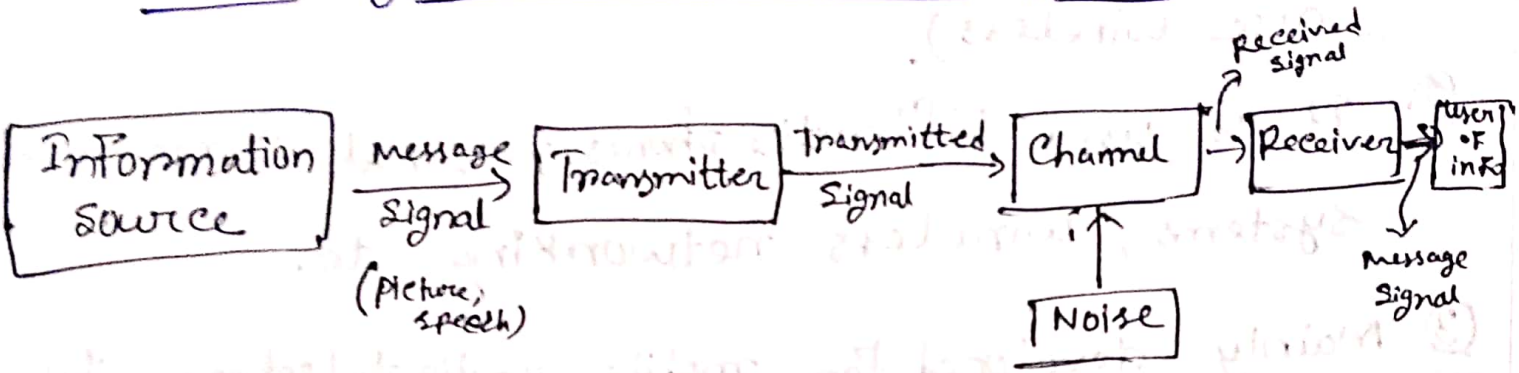


Hardware vs Software (Interrupt)

Hardware	Software
(1) Hardware is Generated From an external device	(1) Generated by an internal System.
(2) Don't increment PC	(2) Increment PC.
(3) Such as request to start an I/O or occurrence of a hardware failure.	(3) Can be invoked with the help of INT instruction.
(4) Lowest priority than S/W interrupt	(4) Has highest priority.
(5) An asynchronus event	(5) Synchronus event
(6) Two types (i) Maskable Interrupt (ii) Nonmaskable interrupt	(6) Two types (1) Normal interrupts (2) Exception

CCE-3607.
Cellular Mobile Communication

Block diagram of Communication System:-



Elements of basic Communication System :-

- Information or input signal
- Input transducer
- Transmitter
- Communication channel or medium
- Noise
- Receiver
- Output transducer