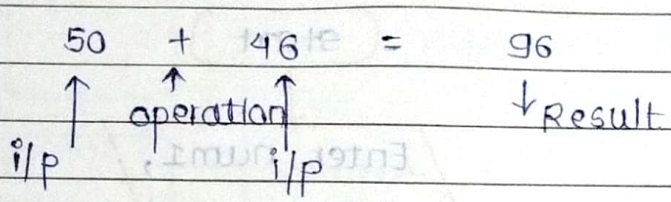
Operation : Task / action performed on inputs to get result which is required.




Algorithm :- steps performed to get the output from given inputs.

Characteristics :-

(i) finiteness : finite no. of steps.
(ii) Definiteness : Each step must perform certain action.
(iii) Input : should have input
(iv) output : Must provide output
(v) Effectiveness : Easy and effective.

flowchart :- Graphical / Pictorial / diagramatical representation.

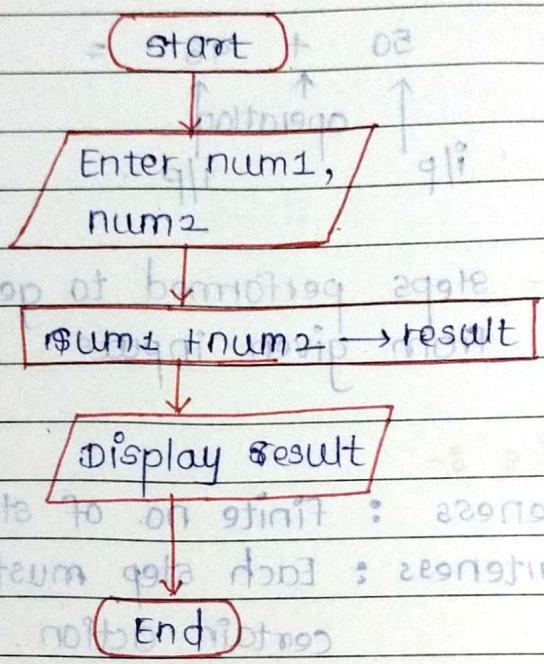
Q Draw flowchart and write algorithm for 'Addition of two numbers' ?

Algorithm :

1. start
2. Declare variables num1, num2, result
3. Read values of num1 and num2
4. result ← num1 + num2

5. Display the result
6. Stop.

Flowchart




**Editor** :- Program allows to create file containing assembly lang statements. EXAMPLE : Wordstar, notepad, PC-write save file as .ASM (that file is source file)

**ASSEMBLER** : Converts assembly lang to machine lang.

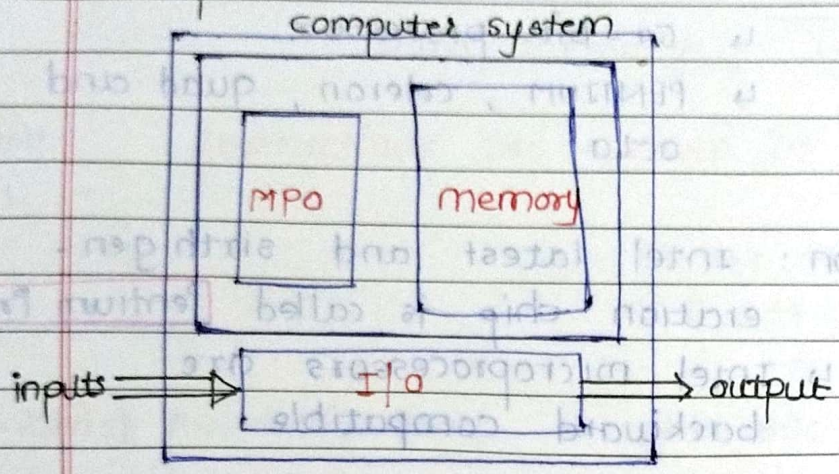
**Linker** : Links machine code with other required assembled codes. Linked for binary file that binary file called executable /exe file.

**Locator** : Program use to assign address of where the object code are to be load into memory (exe → Bin) EXE2BIN is locator program (.bin file)

**Debugger** : Program allows you to load your object code into system memory, execute /troubleshoot debug it.

Page No.:
Date: / /

⊙ Microprocessors




Evolution :

↳ First generation : 1971 - 1972

1971-72 INTEL 4004 Rockwell international pps-4

4004, 8008 INTEL 8008 etc.

↳ Second generation : 1973 - 1978

8-bit microprocessor

1973 - 78 → 8085 → 8-bit INTEL 8085 Motorola 6800 & 6801

↳ Third generation : 1979 - 1980

16-bit microprocessor

1979 - 80 → 8086/186/286 INTEL 8086/ 80186/ 80286 Motorola

→ 16-bit 68000/ 68010 etc.

(HMOS technology)

↳ Fourth generation : 1981 - 1995

1981 - 95 → 80386 - 32-bit using HMOS fabrication

→ 32-bit - INTEL 80386 and motorola 6802

backward compatible : they can run program written for a less powerful processor.

1995-to-now 64-bit – Pentium

Fifth generation : from 1995 to now
↳ 64-bit processor
↳ PENTIUM, celeron, quad and octa

Sixth generation : Intel latest and sixth generation chip is called Pentium Pro
↳ Intel microprocessors are backward compatible

## SOME BASICS

**Memory** : used to store data and programs
- messages → Whatsapp
- photos → Gallery
- songs → Song player
- documents → MS word

**I/O devices** : input - keyboard, Mouse
output - printer, desktop
(don't do any action, just gives input and output)

**Microprocessor** (μp) : contains ALU (Arithmetic logic unit)
↳ process instructions

1] **FETCH** : fetches instructions stored in memory (Instructions are always fetched from memory)

opcode : operation code : Binary pattern of instruction.

Page No.:
Date: / /

2] Decode : understand the instructions fetched from memory.

Instructions are written in following language

1] a = b+c ⟶ Higher level language (C, c++, Java, etc)

(.ASM) 2] Assembly lang ⟶ add B,C — Assembler — complier

3) 01011100 ⟶ Binary lang. object code Machine lang lower Level lang.

memory stores info in the form of

3) Execute : Executes instruction.

process of fetch, decode and Execute is called Instruction cycle.

Trillion ⟵ Tera

Billion ⟵ Giga

Million ⟵ Mega

1000 ⟵ Kilo

- Everything stored in the form of binary
- Computer understand Hexadecimal

Page No.:
Date: / /

**Number System** : → combination of alphabets & numbers

Computer uses hexadecimal number system

| 8 | 4 | 2 | 1 | Hex |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 4+2+1 = 7 |
| 0 | 0 | 1 | 0 | 2 = 2 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 8 | 4 | 2 | 1 | 8 | 4 | 2 | 1 | |
| 41H → | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 8-bit |
| 95H → | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | |
| 00H → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| FFH → | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |

| | 8-bit | 16-bit |
|---|---|---|
| start | 00H | 0000H |
| end | FFH | FFFFH |
| | Byte | word |

① HOW TO CALCULATE POWER'S OF 2

$2^0 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1K$ (1024)
$2^{20} = 1M$
$2^{30} = 1G$
$2^{40} = 1T$

$2^{16} = 2^1 \times 2^6$
$= 1K \times 64$
$= 64K$

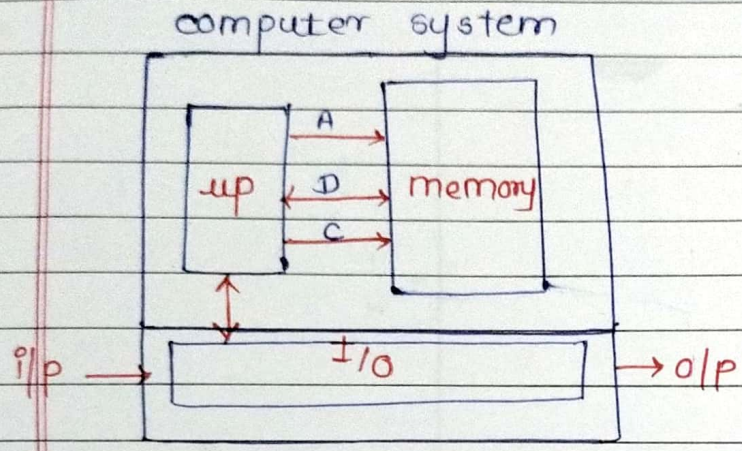
$2^{12} = 2^{10} \cdot 2^2$
$= 4K$

$2^{10} \cdot 2^6$
64K

$2^{24} = 2^{20} \cdot 2^4$
$= 16M$

$32 = 2^{30} \times 2^2$
$= 4GB$

○ **Buses** : set of wires / lines




① data
② Address
③ control
} system bus

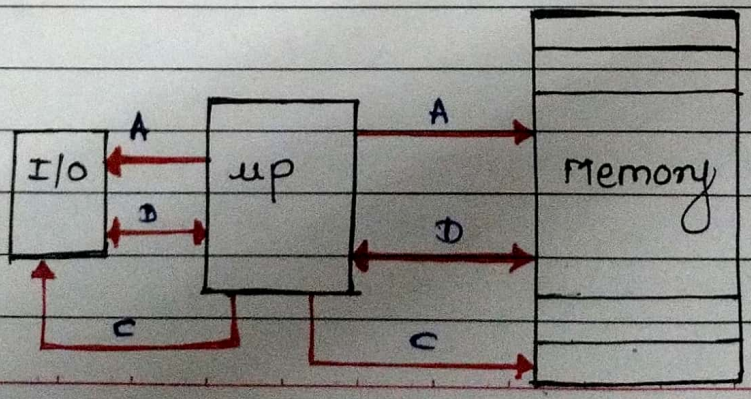
① **Address** ↳ It is unidirectional / contains parallel lines
↳ Identify Location in memory.
↳ 16-bit

② **Data** ↳ Birectional / contains parallel lines
data lines = size of data words
↳ To send data or recive data frm memor or I/O
↳ 8-bit

③ **Control** ↳ Unidirectional / contains parallel lines.
↳ Tells what to do? Read / write

**Read** :- when up gets data that operation is Read from I/o or memory

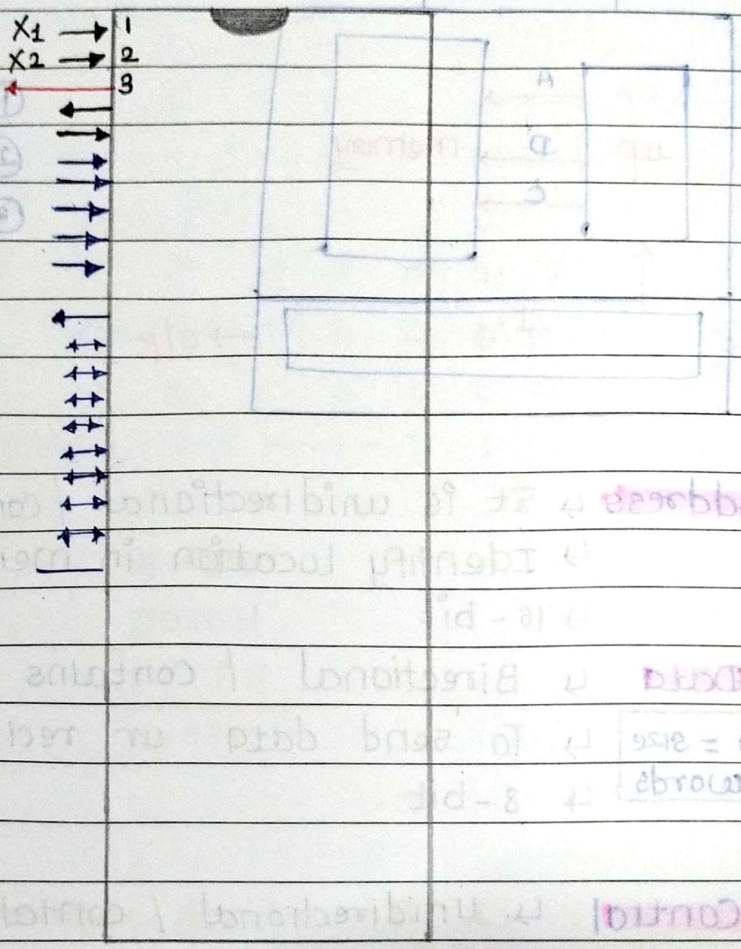
**Write** :- when up sends data to memory / IO that is said to be write.




A - Address bus
C - control bus
D - data bus

Page No.:
Date: / /

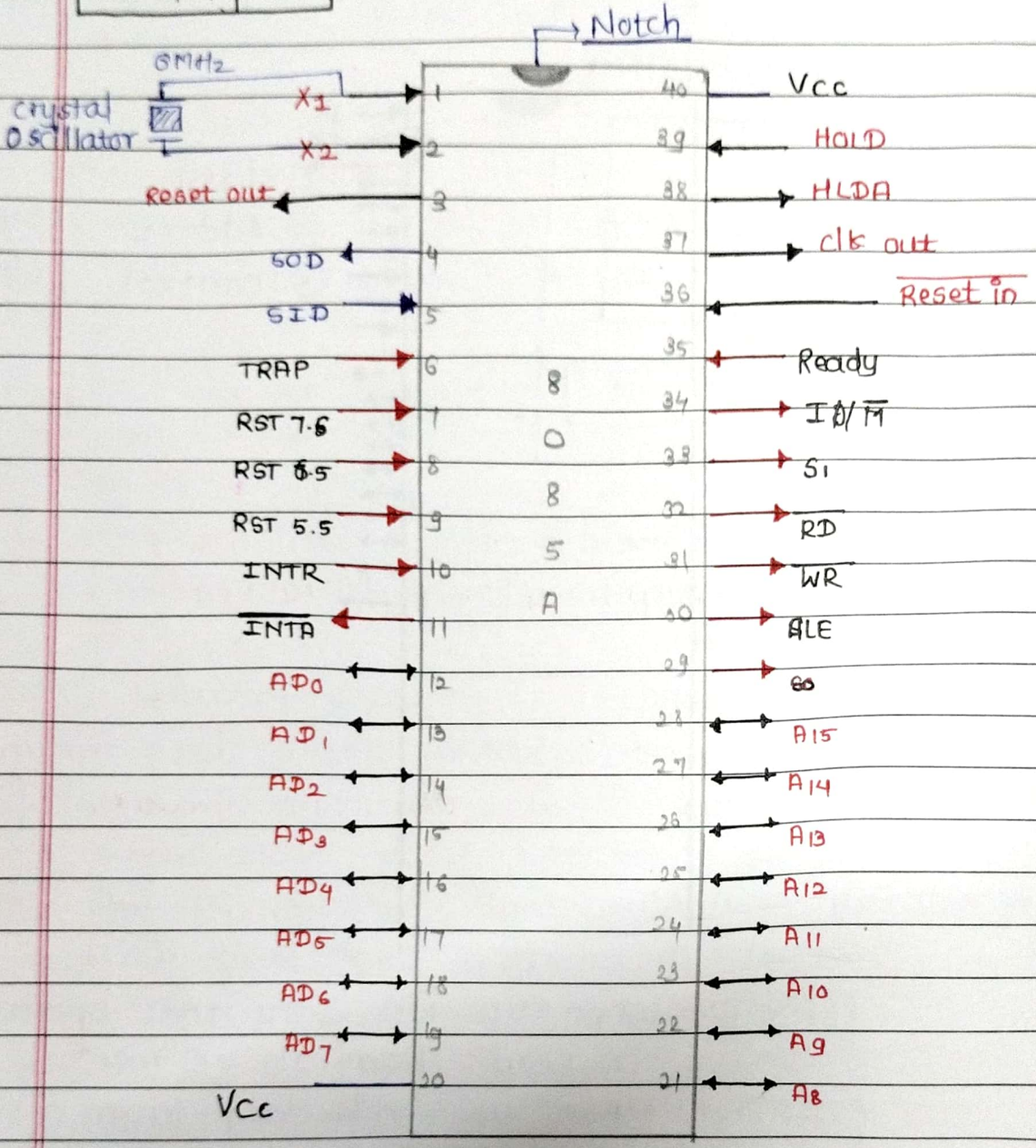
⊙ **Pin diagram** of 8085 microprocessor :



data - 8
Address - 16
IO/device - 256
Memory - 64kB

Page No.:
Date: / /

## Pin diagram of 8085




## FEATURES :

- 40 pins IC
- Rectangular in shape
- 20 pins on right side & 20 pins on left side
- At the top : Notch

→ - input pins
← - output pins
↔ - Bidirectional in/out

transistor - device use for amplification and switching signals

Page No.:
Date:

↳ N-MOS semiconductor (made up of)
↳ There are 6200 transistors (6500)
↳ 1977 - developed in
↳ 8-bit processor (there is transfer of 8-bit data at a time) ALU also 8-bit
↳ data bus ($D_7$ - $D_0$) - 8-bits
↳ Address bus ($A_{15}$ - $A_0$) - 16-bits
↓
$2^{16}$ = 65536 byte = 64 kB } memory can be connected

↳ 8-bit port address (00H - FFH)
↳ $2^8$ = 256 IO devices we can connect

Power supply :
↳ $V_{CC}$ = +5V (DC)
$V_{SS}$ = 0V — ground
↳ Speed = 3-5 Mega Hertz. (standard frequency)
↳ Multiplexed data and lower 8-bit address ($AD_0$ - $AD_7$)
↳ Performs
- Addition, 8 bit, 16 bit with and without carry, 2 digit BCD addition.
- 8 bit binary subtraction with and without borrow
- logical operation.

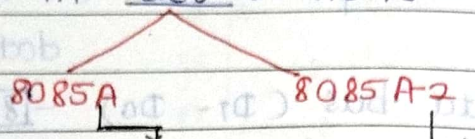
↳ X1 and X2 are connected to the crystal oscillator connected externally to 8085 up
↓
It's frequency is 6 MHz

↳ Accumulator is special purpose register used as GPR.

Page No.:
Date: / /

↳ That 6MHz frequency generated by crystal oscillator is divided by factor of 2 with help of divide by 2 counter. (ie 3MHz)

↳ 8085 is available in two versions




| | | |
|---|---|---|
| crystal oscillator: | 6.06 MHz | 10 MHz |
| internal freq | :- 3 MHz | ↳ 5 MHz |

↳ Enchanced version of 8085 is designed with HMOS transistor.

| THREE VERSIONS : | crystal | Internal freq |
|---|---|---|
| ① 8085 AH | 6.06 MHz | 3 MHz |
| ② 8085 AH-2 | 10 MHz | 5 MHz |
| ③ 8085 AH-1 | 12 MHz | 6 MHz |

↳ **Architectural features**

- **Registers** :
  - ↳ 8-bit (General Purpose Register) GPR
    A (Accumulator), B, C, D, E, H, L
  - ↳ 16-Bit (special purpose Register)
    Program counter (PC) - special purpose Register
    stack pointer (SP) -
    (IR)
  - ↳ 8-Bit (Temporary Registers)
    W and Z and temp data register

- **Interrupts** : TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR

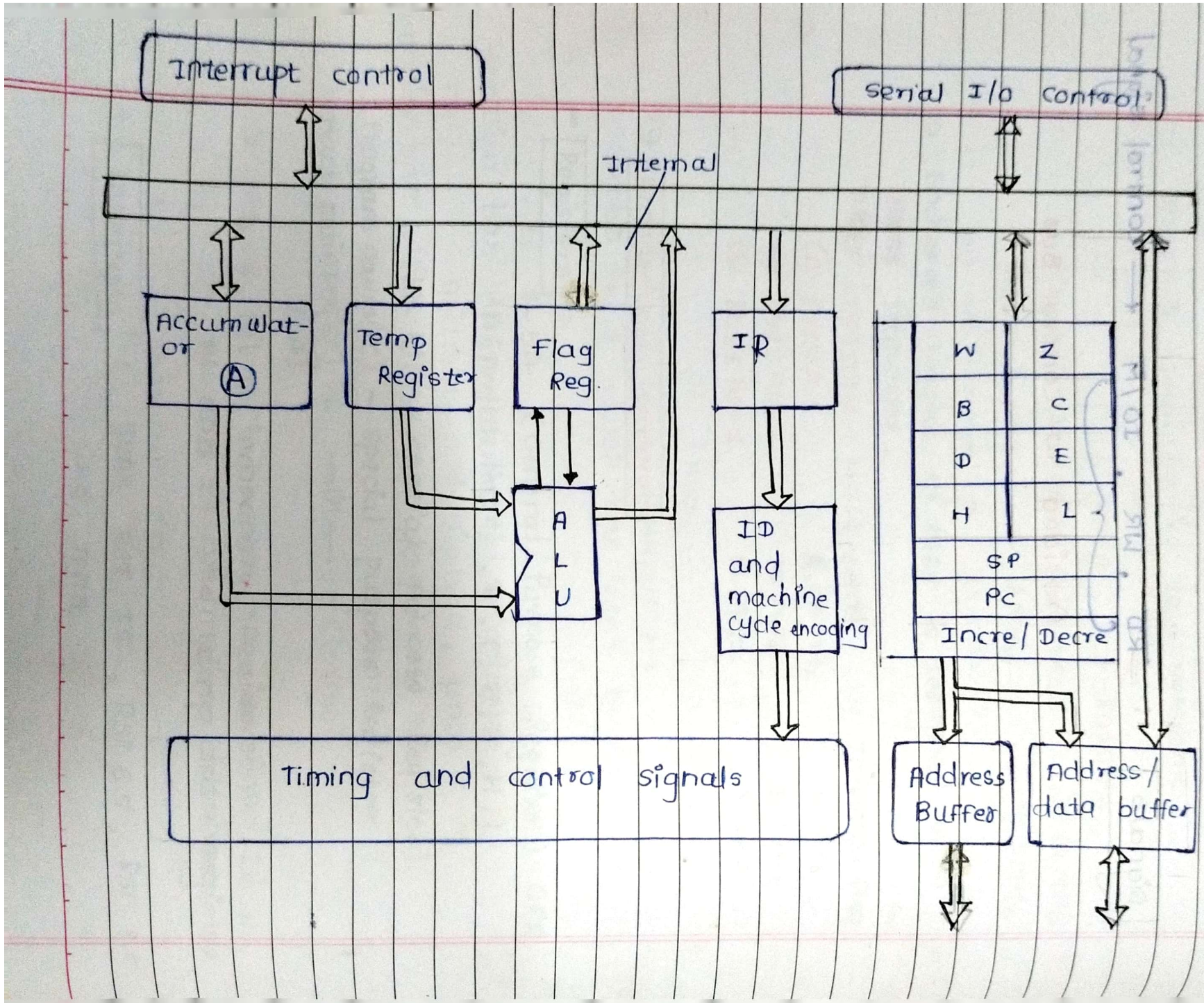
Page No.:
Date: / /

**Serial I/O** : Serial peripherals are interfaced through SID and SOD

**Signals** : $\overline{RD}$ , $\overline{WR}$ , IO/$\overline{M}$ ← control signal

Provided to control Bus

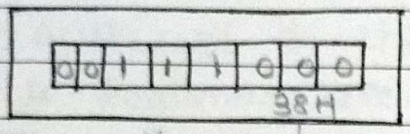
# Architecture of 8085



Variables are software
Registers are hardware

Page No.:
Date: / /

**Registers** : Present in the up, used to store numbers.
Registers are formed by set of flipflops.

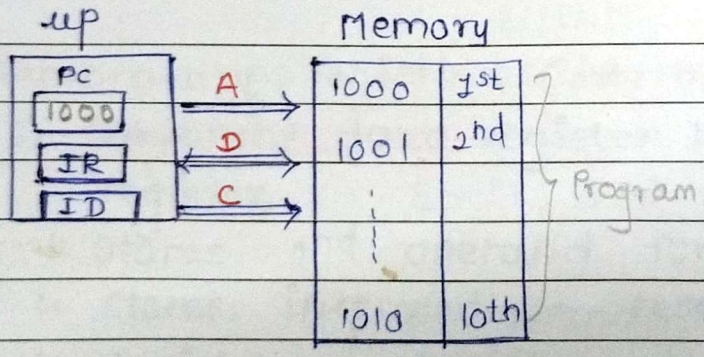



8-bit register – can store 8-bit number.
- made from 8 flipflops.

In 8085 their are (7) 8-bits **general purpose** register

- GENERAL PURPOSE REGISTER : B, C, D, E, H, L, A
- SPECIAL PURPOSE REGISTER :

(1) **PC**
- Program Counter
- Special Purpose register
- Used to hold address

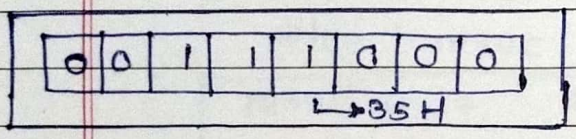


variables - software
Register - hardware

Page No.:
Date: / /

## Architecture

① Register section :
- Present in the up, used to store numbers.
- Registers are formed by flipflops
- This are PIPO registers (Parallel in parallel out)
- 8-bits Register - made up of 8 flip-flops
  - can store 8-bit number.




TYPES :
- Temprary register
- General Purpose register (GPR)
- Special Purpose register

① Temprorary registers (data manipulation)

1) Temporary data register : called as operand register.
- stores 2nd operand for operation.
- * stores intermediate result
- * register → register data transfer.

2) W, Z registers
- Not available for programmers / users.
- Used internally by up.
- XCHG → eg. ie used for swapping (which is required by up internally.)
- 8-bit
- holds data during ALU operation by control section.

HL pair function as data & Memory pointer

Page No.:
Date: / /

2) General Purpose Register :
- 6 general purpose registers. (8-bits)
- B, C, D, E, H and L
- Valid pairs BC, DE and HL can hold 16-bit of data.
- Used by programmers
- holds data, Result of ALU and address.

3) Special Purpose Register :

Special use

① Accumulator :
- Used as GPR
- 'A' register / holds first operand for operation
- Stores result of ALU operation.

② Flag register :
- Stores status
- ALU can write the flag or read the flag.
- 8-bit

In 8085 5 flags

① sign flag

Indicates sign of result.

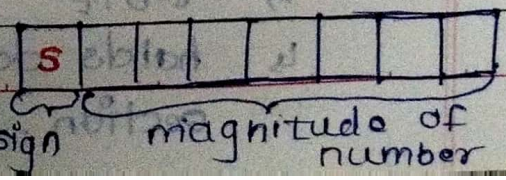
If ALU result negative ← [0 = +ve, 1 = -1]

If result of ALU is positive

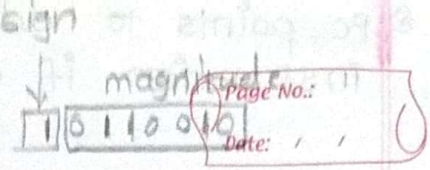
[signed number] ; MSB is reserved for sign for 8-bit

If 1 ← MSB → If 0

Negative number | Positive number



Set means 1
Reset means 0

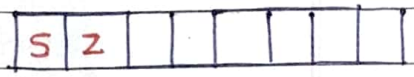



↳ for logical and unsigned arithmetic operation no significance of sign bit (all bits are used to represent magnitude of number)

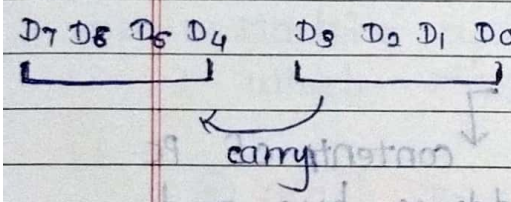
* similar for 16 and 32 bit, MSB is reserve for sign of result of signed Arithmetic operation

② Z - zero flag
↳ set if result of ALU is 0
else reset if ALU result is non zero

| result 0 = 1 |
| --- |
| result non zero = 0 |



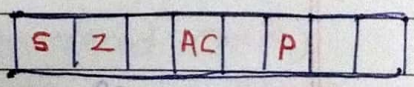

③ AC - Auxiliary carry
↳ Used only for BCD operation





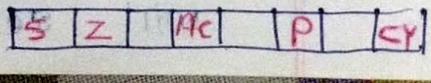
④ P - Parity flag
↳ After ALU result if result has even no of 1s then P-flag is set else reset

| even parity = 1 |
| --- |
| odd parity = 0 |




⑤ CY - Carry flag
carry is set when result generates carry out or borrow in in MSB.

| 1 - carry |
| --- |
| 0 - no carry |



⑤ PC points to address specified by JUMP and CALL instruction if condition satisfied

Page No.:
Date: / /

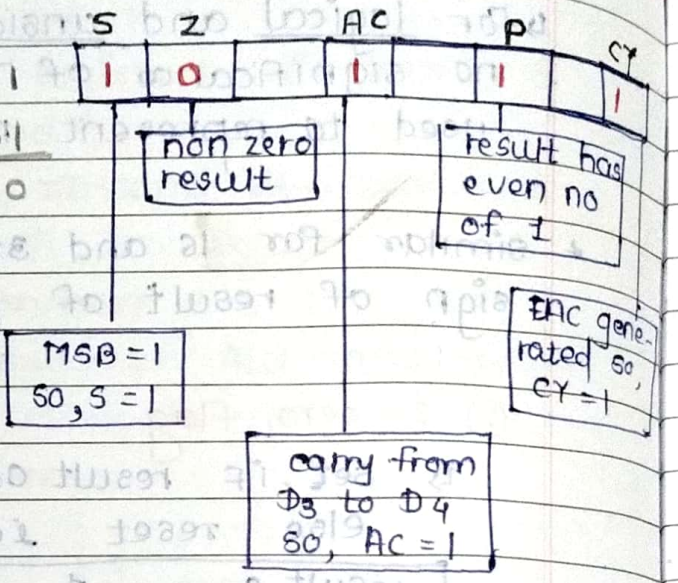
**Effect on Flag**

Q = CB + B9

CB = 1100 1011

B9 = 1011 1001

1000 0100




③ **Program Counter** :

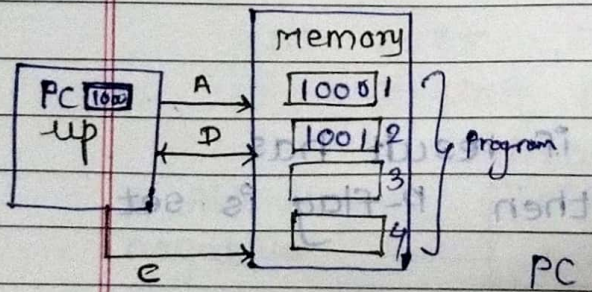
↳ used to hold address

↳ Always points to next instruction to be fetch.

↳ During instruction fetch :

up place contents of PC on to address bus and fetches first byte of instruction.

up increment (by 1) to PC to point next instruction (side by side executes instruction which is fetched already).




↳ Size depends on address bits in 8085 up

size of PC = 16 bits

↳ When reset activated in 8085 up then PC will set to 0000H ie first instruction

④ Instruction Register (IR)

↳ Holds opcode (Binary Pattern of ins.) that is decoded & executed.

↳ further sent to Instruction decoder.

↳ Takes opcode value only.

⑤ Instruction Decoder (ID) and machine cycle encoder

↳ ~~AND called machine cycle encoder~~.

Accepts opcode from IR and decode it and decoded info send to control Logic.

1) what operation is performed
2) who will perform
3) No of operands

↳ 8085 executes 7 types of machine cycle.

↳ Machine cycle endcoder give info about which machine cyele is currently executed.

⑥ Stack pointer (SP):

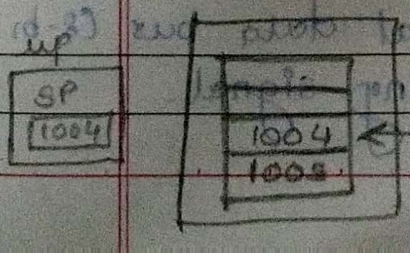
↳ 16-bit

↳ Reserved portion of memory.

↳ Holds address or points to where top of stack.

↳ Register pair info is stored or taken back.

↳ SP increment after each read (POP)
SP decrement after each write (Push)



tristate mode: logic 0, 1, and high impedance.
in it pin neither connect to ground nor to supply.

② Buffer and latch:

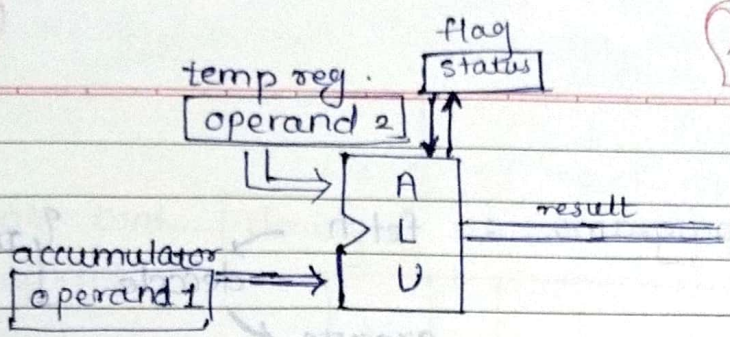
① Address Buffer. (A7 - A15)
- ↳ unidirectional 8-bit buffer
- ↳ drives higher order address bus (A7 - A15)
- ↳ Certain conditions like reset, hold, halt the address buffer is not in used so, the buffer puts
  - ↳ address bus in tristate mode.

② Address / Data buffer (AD0 - AD7)
- ↳ bidirectional 8-bit buffer
- ↳ drives lower order address bus ($A_0 - A_7$) and data bus ($D_0 - D_7$) these are multiplexed ↳ $AD_0 - AD_7$
- ↳ when not in used buffer puts
  - ↳ address bus in tristate mode.

③ Increment / decrement latch:
- ↳ 16-bit
- ↳ used to increment PC
- ↳ used to increment / decrement SP

③ ALU:
- ↳ Arithmetic and logic unit.
- ↳ Performs Arithmetic and logical operations.
  - ↓ Add / sub
  - ↓ AND / OR
- ↳ width depends upon = internal data bus (8-bit)
- ↳ Controlled by control and timing signal.

sequential logic Circuit : o/p depends only on present i/p's.




↳ stores result in accumulator.
↳ Provides status of operation in flag register.

④ Control Section, Serial I/O and Interrupt :

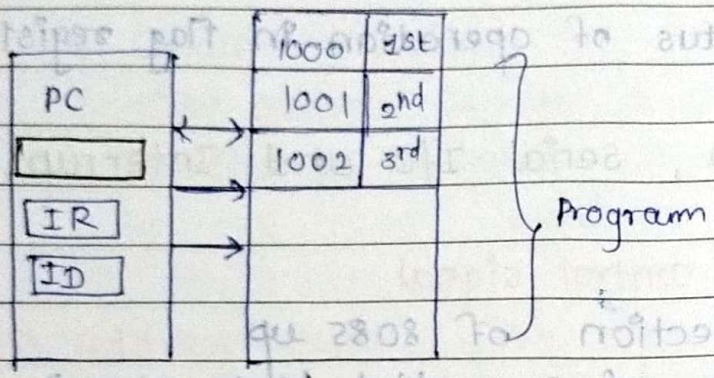
1) Timing and control signal
↳ Control section of 8085 up
↳ made up of sequential logic circuit
↳ provide control over all internal and external circuit
↳ Recieves info from decoder
↳ And tells other devices what to do

2) Serial I/O
↳ SID and SOD are used when in some cases we required to transfer data serially.
↳ SID accepts input
↳ SOD accepts output

3) Interrupts
↳ TRAP
RST 7.5
RST 6.5
RST 5.5
INTR

Priority

up executes programs : fetch → decode → execute } Instruction cycle




PC : Always contains address of next instruction (when instruction is decode & executes it will increment side by side)

Address from PC (16-bit)

16-bit address of instruction

8-bit $A_{15} - A_8$

8-bit $AD_0 - AD_7$

using INR

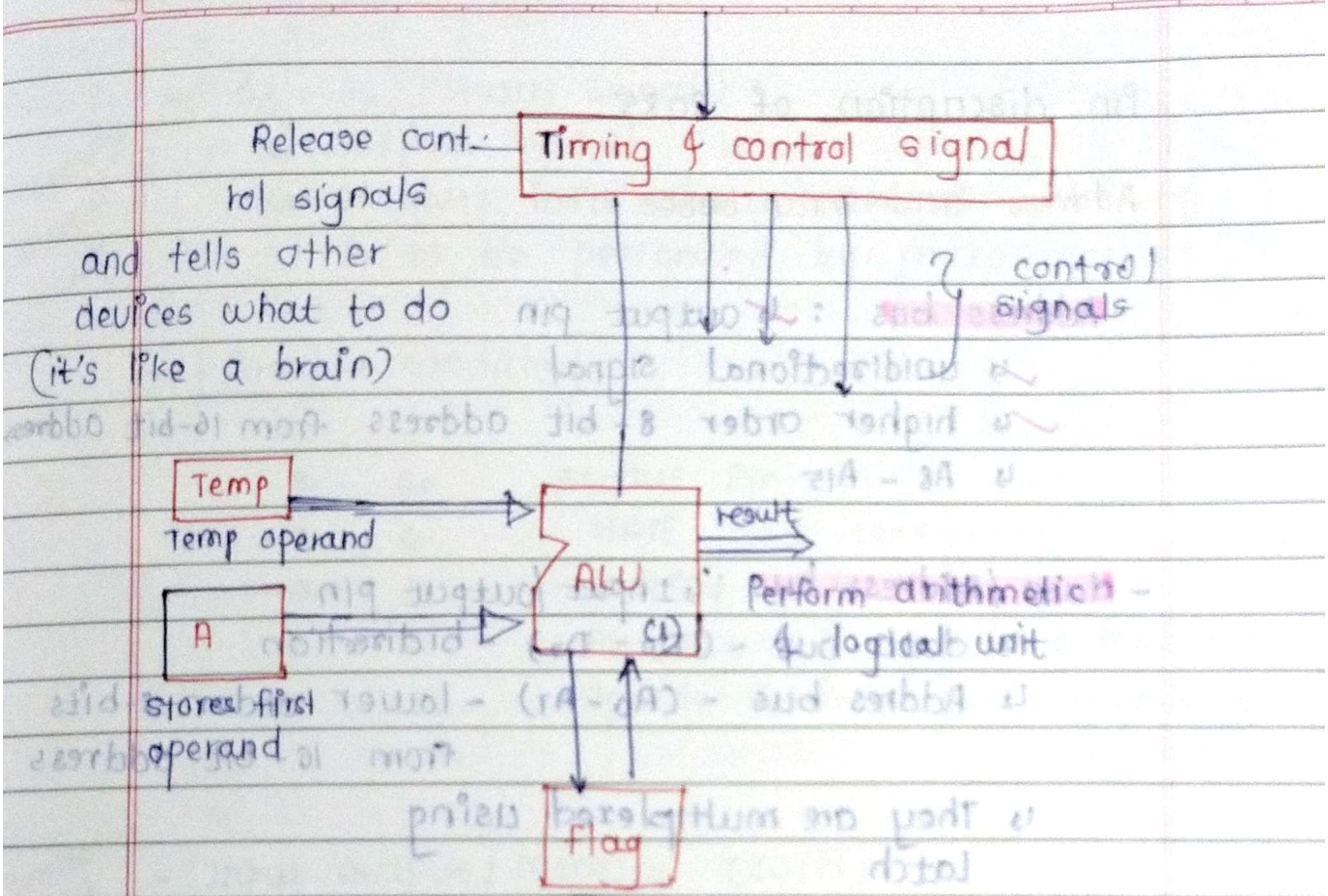
By giving $\overline{RD}$ signal it fetches instruction from memory

Through data bus ($AD_0 - AD_7$)

memory

IR

ID decodes Opcode




ISR - Interrupt service routine

Hardware (a) Software (b)

Q Why 8085 is 8-bit processor

↳ As we can transfer 8-bit data at a time as well as ALU size is also 8-bit

Page No.:
Date: / /

## Pin discription of 8085

### 1) Address and Data buses :

- Address bus : ↳ output pin
  - ↳ unidirectional signal
  - ↳ higher order 8-bit address from 16-bit address
    - ↳ $A_8 - A_{15}$

- data / Address bus : ↳ input / output pin
  - ↳ data bus - ($D_0 - D_7$) - bidirection
  - ↳ Addres bus - ($A_0 - A_7$) - lower order 8-bits from 16-bit address
  - ↳ They are multiplexed using latch
  - ie $AD_0 - AD_7$

During operation cycle first part is lower order address and in later part data bus

They are demultiplexed in order to reduce no. of pins

### 2) status and control signals

- ALE (Address Enable Latch)
  - ↳ output status signal
  - ↳ Gives discription of $AD_0 - AD_7$
    - ① when ALE is ACTIVE (Logic = 1) : $AD_0 - AD_7$ will contents address
    - ② when ALE is low (Logic = 0) : $AD_0 - AD_7$ will contents data.
  - ↳ ALE pin is connected to the latch to seprate data and address

Normally for reading/writing data from or to memory it requires 3 clk cycle.

Page No.:
Date: / /

- $S_0$ - $S_1$ (status signal):
  - ↳ output signal
  - ↳ It gives info about operation which is to be performed by microprocessor.

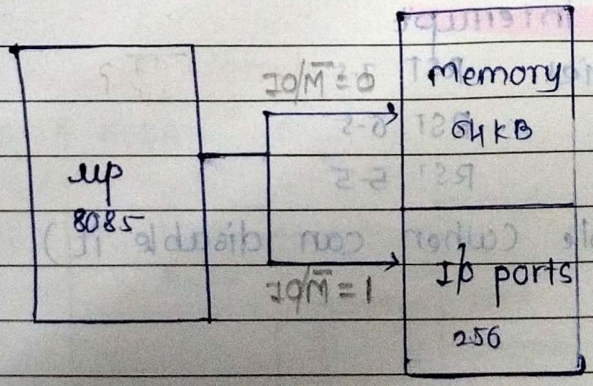
specify status of µp

**Possible condition**

| $S_1$ | $S_0$ | status Pin |
|---|---|---|
| 0 | 0 | HALT (No task) |
| 0 | 1 | writing (3 clk) |
| 1 | 0 | Reading (3 clk) i.e Reading of data |
| 1 | 1 | Opcode fetch (4 clk) i.e Reading instruction |

control signal ($\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$)

- Input output / memory (IO/$\overline{M}$)
  - ↳ output signal
  - ↳ Gives info of operations to be performed with memory or I/O device




IO/$\overline{M}$ = 0 then operation with memory

IO/$\overline{M}$ = 1 then operation with I/O

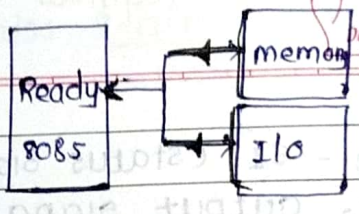
0 → activate
1 → deactivate

low level signals

There are two operations Read or write from memory or IO device
- ↳ $\overline{RD}$ and $\overline{WR}$ are used for this

| for memory | for I/O device |
|---|---|
| $\overline{RD}$ = 0, $\overline{WR}$ = 0 } Read/write from memory | $\overline{RD}$ = 0, $\overline{WR}$ = 0 } Read/write from I/O |

When Memory/IO device speed is not matched with up speed Ready pin is used to communicate up with memory/IO.




control signal

- READY :
  - ↳ Input signal
  - ↳ It detects whether I/O peripherals are ready for data transfer or not
  - ↳ synchronize up with slower peripherals.

up checks Ready pin status for 1 clk cycle

3) Interrupt Signals : stops current activity

- TRAP (RST 4.5)
  - ↳ Non maskable (we can't disable it)
  - ↳ Highest priority
  - ↳ when TRAP is active up performs internal restart.

saves return address and 0 is transfer to INTA

↳ whenever up interrupt it performs this two tasks

- Restart interrupt

Priorities: RST 7.5
RST 6.5
RST 5.5

↳ maskable (when can disable it)

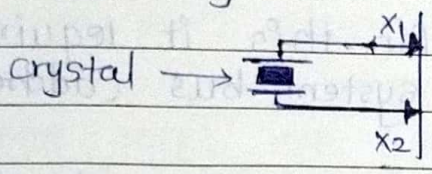
maskable

- INTR
  - ↳ lowest priority
  - ↳ maskable

- INTA (↳ output)
  - ↳ Interrupt Acknowledgment
  - ↳ low level signal
  - ↳ Active = 0 → tells that up is interrupted.

generate constant amplitude and frequency

4) Clock signals
- X1, X2
  - Input signal connected to crystal oscillator




  - X1, X2 drives internal clock generator.

5) Serial I/O signals :
  - SID (Serial Input data)
    - Serial input pin
    - Instruction pin is RIM (Recieve interrupt mask)
  - SOD (serial output data)
    - serial output pin
    - SIM (serial interrupt mask) - Instruction

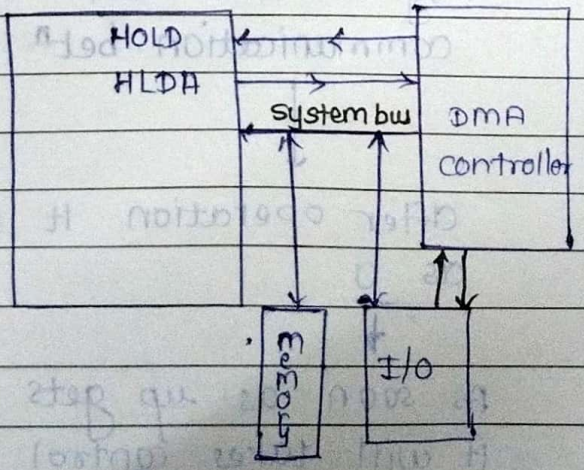
0/1 0/1
(serial comm)

1 — 1
0 — 0
0 — 1
(parallel comm)

data bus.

6) DMA request signals
- HOLD & HLDA

DMA (direct memory access controller)



Page No.:
Date: / /

↳ DMA can access memory directly, It helps to communicate memory and I/O devices in case of large data transfer (to save time)

↳ But for this it requires control over system bus (address, data, control)

↳ so it will send HOLD request as 1 to up

↳ As soon as up get HOLD req as 1 it will stops current operation and send HLDA (Hold acknowledgment) as 1

↓

when DMA recieves HLDA as 1, it means up stoped its internal and external operations and thus

↓

DMA takes control over the system bus and starts direct communication bet$^n$ I/O & memory.

↓

After operation it will send HOLD as 0

↓

As soon as up gets HOLD as 0 it will takes control over its system bus and starts its current operation

3 ways to wake up up

↳ HRQ = 0 from DMA
↳ Reset pin = 0
↳ interrupt

Booting Program : set GPR, flag register, etc to 0 and restart up.

7) Reset signals :

- RESET IN
  - ↳ input signal
  - ↳ low level signal (0-active)
  - ↳ reset in = 0 clear PC (Restart ie runs booting program)
  - ↳ After reset up starts execution from 0000H

- RESET OUT ↳ High level
  - ↳ Output signal
  - ↳ Indicates up is reset
  - ↳ It is connected to other devices and so that devices also restarts as reset = 1

8) Clock out
- output pin.
- → जेवढी frequency input मिळे मेले तेवढी same up output मिळे generate होते.
  and provide that generated frequency to devices which works on 3MHz.

10) Vss -0
Vcc -+5V

# UNIT 3: INSTRUCTION SET 8086

Page No.:
Date: / /

## Instruction format :

- It varies from 1 to 6 byte.
- First two bytes content is opcode and addressing mode. and remaining bytes contains may or may not 8 or 16-bit displacement and operands.

1] opcode - one byte instruction

2] opcode | reg - one byte instruction reg mode

3] opcode | 11 | Reg | R/M - Reg to Reg.

4] opcode | mod | Reg | R/M - Reg to/from memory with no disp.

5] opcode | mod | Reg | R/M | disp

opcode | mod | Reg | R/M | low disp | High disp } with 8/16 bit disp.

6] opcode | 11 | Reg | R/M | operand - Immediate operand to Reg (8-bit)

7) opcode | 11 | Reg | R/M | L. operand | H-operand (16-bit)

8) opcode | 11 | Reg | R/M | L.disp | H.disp | L.opera | H.operand.

First byte - opcode

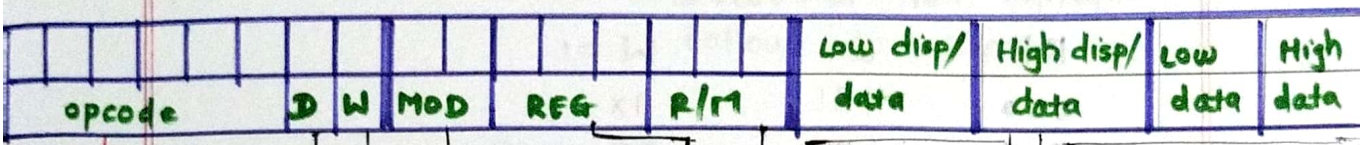
Second byte = Addressing mode or sometimes part of opcode

remaning bytes - cases dependent

(I) No additional byte
(II) immediate operand (8-16 bit)
(III) displacement (8-16 bit)
(IV) operand with displacement

If disp. is 16 bit then lower order 8-bits always appears first.

Byte 1 | Byte 2 | 3 | 4 | 5 | 6




opcode : operation code or instruction code. Identifies basic instruction

identifies size
W = 0 (byte)
W = 1 (word)

Register

Not always present depends on instruction

Used to EA calculation

direction is to/from reg
D=0 [destination specified in reg]
D=1 [source specified]

Reg mode/memory mode with disp length

**REG** If opcode is multibyte then REG field is replace by S, V, Z opcode bits. thus total bits are 5 (S, V, Z, W, D).

S = 0 → No sign extension
= 1 → (W=1) then extend 8-bits with MSB bit.

W = 0 — byte instruction
= 1 — word (2 byte) instruction

D = 0 — source specified in reg
= 1 — destination —11— ——

Z = 0 —— repeat loop
= 1 —— repeat loop

V = 0 —— shift / rotate count is one
= 1 —— shift / rotate count is in CL reg.

**Mod** : 2-bit field tells that from where the operands are ie one of operand is in memory or both are from reg.

00 – memory mode no disp.
01 – memory mode with 8-bit disp
10 – ——— 11 ——— 16-bit disp
11 – Reg mode no disp.

Page No.:
Date: / /

**REG** : When reg is one of operand of instruction.
depends on w value

| REG | W = 0 | W = 1 |
|---|---|---|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

**R/M** : Register / Memory (3-bits)
depends on mode (mod)

**data transfer**

① Reg to Reg :

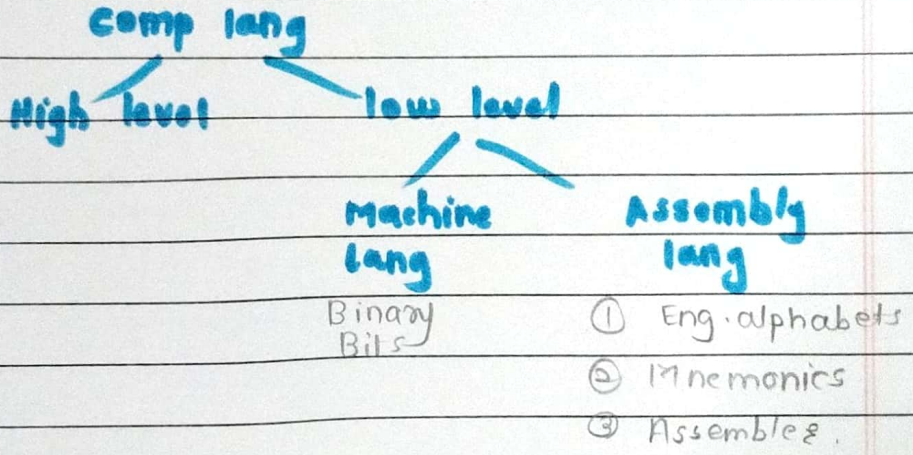
Mod = 11

| R/M | W = 0 | W = 1 |
|---|---|---|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

② Reg to from mem (with 8/16/ no disp)

Page No.:
Date: / /

**Program :** Set of instruction
↳ direction which cpu follows to execute task.

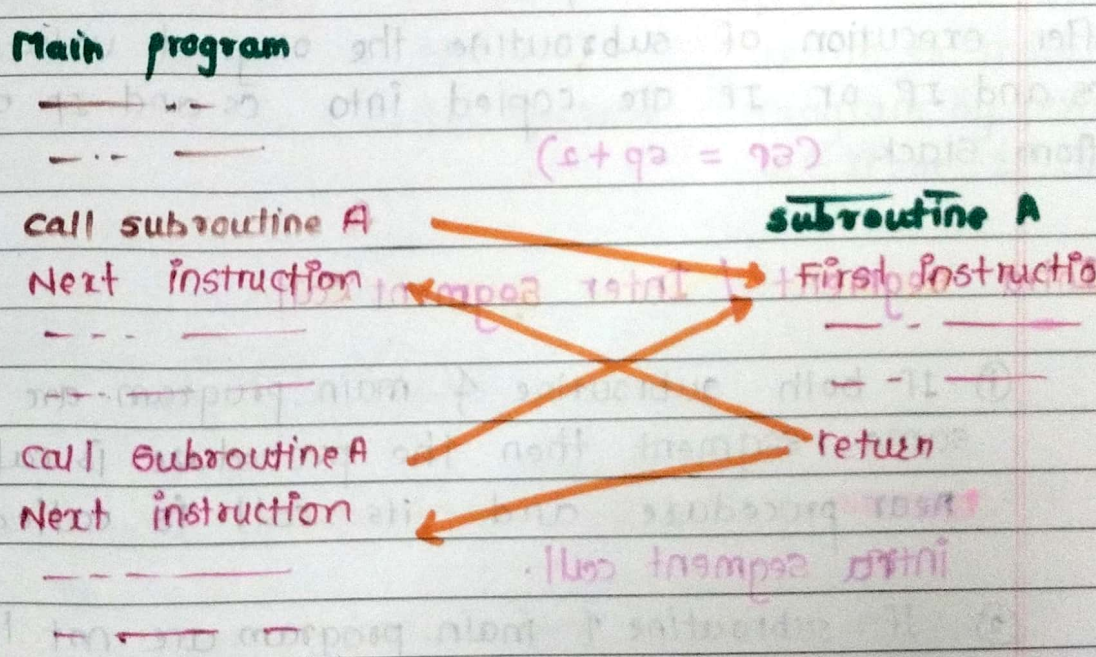


# Branch Instruction

Page No.:
Date: / /

**Branch instruction:**
Useful for programmers for performing operations selectively and repetitively.

- **Branch instructions:**
  - conditional jump
  - unconditional jump
  - iteration instruction
  - call instruction
  - Return instruction




**Subroutine** (subroutine also called procedure):

- Special program or special segment of program called for execution from any point in program.
- Whenever we need subroutine only one single instruction is inserted in program.

- To branch subroutine
  - i) change value of IP (if subroutine & program lies in same segment)

ii) change CS and IP (if subroutine and program are not in same segment)

↳ After execution of subroutine control transffered to the original CS and IP or IP values (which are preserved in stack)

① whenever subroutine is called that time address of next instruction is firstly stored in stack (SP = SP-2) ie content of IP or CS & IP. and CS & IP or IP loaded with new values.

② After execution of subroutine the original values of CS and IP or IP are copied into CS and IP or IP from stack (SP = SP+2).

## Intra segment / Inter segment call

① If both subroutine & main program are in same segment then the procedure is called near procedure and its call is called intra segment call.

② If subroutine & main program are not in same segment then procedure is called far procedure and its call is called inter segment call.

INTER segment - ① content of CS & IP loaded into stack
② New content is loaded into CS & IP according to instruction
③ After execution original CS & IP content is place into CS & IP

**CALL**

(No flags are affected)

| Mnemonic | Meaning | Syntax | operation |
|---|---|---|---|
| CALL | call to subroutine | CALL operand | Execution continues from address specified by operand upto return. |

**Return** After subroutine execution value of IP or CS & IP are return to their original addresses content with new values.

(No flag affected)

| Mnemonics | Meaning | Syntax | Operation |
|---|---|---|---|
| RET | return | RET<br>RET operand | return to main program by restoring CS & IP or IP content |

**LOOPS**

→ used of repeatitive task.

LOOP short-label operation : CX = CX - 1

→ when CX ≠ 0 — jump to location defined by short label

CX = 0 — then execute next sequential instruction.

CX ← loops works with content of CX.

↳ represents no. of times loop is to be repeat.

+ when loop execute decrement CX & then check content of CX.

**General format:**

Loop r8
↳ signed 8-bit value.

[do not affect any flag]

[USED FOR BACKWARD JUMP ONLY] — max distance is of 128 bytes.

```
Loop AGAIN :  DEC  CX
              JNZ  AGAIN
↳ Repeat task
```

**Loop instructions:**

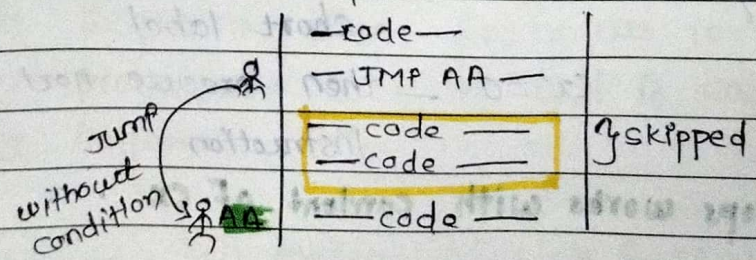
① LOOP short-label :- jump to location given by short label (cx ≠ 0)

② LOOPE short-label : } jump if cx ≠ 0 and ZF = 1 (set)
LOOPZ —11— :

③ LOOPNE —11— } jump if cx ≠ 0 ZF ≠ 0 (reset)
LOOPNZ —11—

## ⊙ Control Flow & Jump instruction:

① Unconditional Jump:




Syntax : JMP operand

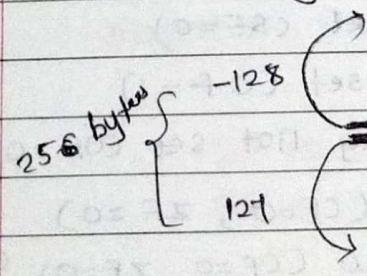
JMP types :

(i) Near
(ii) Short
(iii) Far

1] Far : (Inter segment jump)
Jump to an instruction present in diff^nt segment than current.
We have to modify cs & ip for this

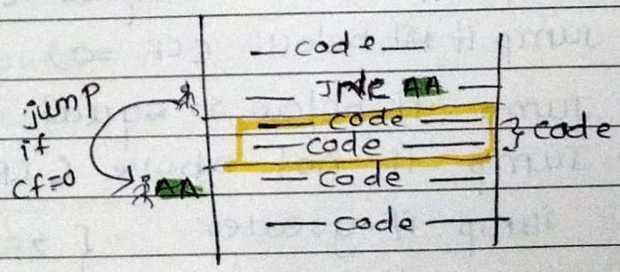
2] Near : (Intra segment jump)
Jump to an instruction present in current code segment.
We have to modify ip content for this.

3] Short :
It has range limit : (-128 + 127) bytes.




(2) Conditional Jump :

JNC means Carry flag not set (CF = 0)




Jump only when condition will statisfied, & affect done on flags.

→ Types :

Lower 8-bit

| SF | ZF | | AC | | PF | | CY |
|---|---|---|---|---|---|---|---|

Page No.:
Date: / /

1. JC – Jump when carry flag set (CF = 1)
2. JNC – Jump when carry flag not set (CF = 0)
3. JZ – Jump when zero flag set (ZF = 1)
4. JE – if ZF = 1 then both operands are equal so it will jump.
5. JNZ – Jump when zero flag reset (ZF = 0)
6. JNE – ZF = 0 mean operands are not equal so it will jump
7. JPO – If parity is odd then jump (if PF = 0) ie parity reset
8. JPN – If no parity (PF = 0) i.e parity reset.
9. JPE – Parity even i.e set (PF = 1) then jump
10. JP – If parity set (PF = 1) then jump
11. JS – If sign flag set (SF = 1)
12. JNS – If sign flag reset (SF = 0)
13. JO – If overflow flag set (OF = 1)
14. JNO – If overflow flag not set (OF = 0)
15. JA – jump if above (CF = 0, ZF = 0)
    JNBE – jump if not below (CF = 0, ZF = 0) ↳ equal
16. JB – jump if below (CF = 1)
    JNAE – Jump if not above or equal (CF = 1)
17. JAE – jump if above or equal (CF = 0)
    JNB – jump if not below (CF = 0)
18. JBE – Jump if below or equal (CF = ZF = 1)
    JNA – Jump if not above (CF = 1)
19. JG – jump if greater { ZF = 0, SF = OF
    JNLE – jump if not less or equal
20. JGE – jump if greater or equal { SF = OF
    JNL – jump if less
21. JL/JNGE – jump if less / jump if not Greater or equal (SF ≠ OF, SF = 1, SF = 0)
22. JLE/JNG – jump if less or equal / if not greater (ZF = 1 or SF ≠ 0)
23. JCXZ – if CX = 0 then jump

signed comparison

signed operation

# String Instructions :

Key points :

① movs, compare, scan, load, store are basic string operations.

② for string operation we need starting and end address and length of string.

③ Length stored in – cx register

④ Physical address generation string.

i] DS : SI — address of source string

ii] Es : SI — address of destination string

segment offset

⑤ During string instruction operation SI & DI ~~flag~~ is increment or decrement on basis of Direction flag.

clear direction flag

DF = 0 — increment → CLD ↑

DF = 1 — decrement → STD ↓

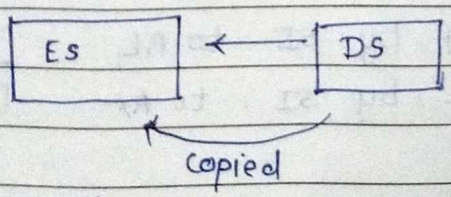
set direction flag

## 1) Movs :

movs destination string, Source string.

i) MovsB – byte operation

ii) Movsw – word operation.



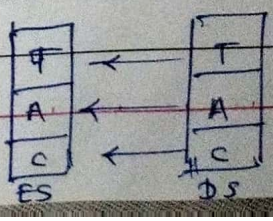

ES : [DI] = Ds : [SI]

Byte operations

DF = 0 → SI ⇒ SI + 1, DI ⇒ DI + 1

DF = 1 → SI ⇒ SI - 1, DI ⇒ DI - 1

for word +2 or -2



Page No.:
Date: / /

## 2) CMPS :

Compare string byte or word.

CMPSB – byte  CMPSW → word.

comparision is done byte by byte using substraction & sets flags according to result.

Byte → for word +2, -2
- Df = 0 = increment by 1 SI/DI  DS:[SI] – ES:[DI]
- Df = 1 = decrement by 1 SI/DI

## 3) Scas :

Scas destination string:

for byte scasB: This instruction compares byte in AL with byte pointed by DI in ES & sets flags according to results.

for word scasw: This compares bytes in AX with byte pointed by DI in ES & set flags.

byte for word +2, -2
- Df = 0 , DI + 1  ES:[DI] – AL – byte
- Df = 1 , DI – 1  ES:[DI] – AX – word.

## 4) LODS :

Byte → copies string pointed by SI to AL  [LODSB]

word → ——— " ——— by SI to AX  [LODSW]

for byte → for word +2, -2
- Df = 0 – increment SI  AL = DS:[SI]
- Df = 1 – Decrement DI  AX = DS:[SI]

## 5] STOS :

[STOSW] word → copies word from AX to ES addressed by DI

[STOSB] byte → copies ~~byte~~ word from AL to ES addressed by DI

increment by 1 { Byte ⇒ Df = 0 then increment DI ES:[DI] = AL

Df = 1 then decrement DI ES:[DI] = AX

word decrement | increment by +2 -2

## 6) REP : Repeat until condition exists.

REP — if CX = 0

REPE / REPZ ⟶ CX = 0 , ZF = 0

Repeat while 0 & equal

REPNE / REPNZ ⟶ CX = 0 or ZF = 1

Repeat while not equal & zero

## ⊙ Assembler directives :

↳ Gives direction to the assembler.
↳ No machine codes are generated for assembler directives.
↳ Called as pseudo instruction.
↳ This are the instructions but not executable here.

**It do not generate machine executable code.**

USE
↳ specify start & end of program.
↳ attach value for variable
↳ allocate storage location
↳ define start & end of segment / Macros / procedures.

## # Assembler directives :

(I) data definition & storage allocation -
(II) Program organization
(III) procedure directives
(IV) Macro —11—
(V) data control —11—

(I) DB - Define byte : 8-bits.

Range $00_H - FF_H$ signed
$00_H - 7F_H$ Positive
$80_H - FF_H$ negative.

Syntax : var_name db value

```
no1    db  20H
no2    db  ?
```

↳ byte is reserved but don't know value

DW : define word (16-bits)
↳ 2 memory locations for each variable.

Range : 0000H - FFFFH unsigned
0000H - 7FFFH positive
8000H - FFFFH Negative.

Syntax : Var_name DW value

DD - define double word
DQ - define quad. word
DT - define Ten-byte

② EQU : Equate
Use to assign constant value to symbol.
PI EQU 3.14

③ ORG :
Origin
Used to assign starting address for a program/ data segment.
ORG 1000H
Effective address 1000H here.

④ Even :
Inform assembler to store program in an even address

⑤ Assume :
Informs assembler the name of program/data seg that should be for a specific segment.

Assume segreg : seg_name

⑥ END : use to terminate program ; statements after end will simply ignored .

⑦ ENDS : end of segment .

```
segname  segment
   —
   —
segname  ends
```

⑧ PROC : Begining of procedure .
ENDP : End of procedure
Near : Intrasegment (another seg)
Far : Intersegment (same seg)

```
Proc_name  PROC  [Near/Far]
   —
   —
   RET
Proc_name  Endp
```

⑨ Macro : begining of macro
ENDM :- end of macro

```
Macro_name  MACRO  [Arg1, Arg2...]
   —
   —
   —
Macro_name  ENDM
```

(10) Page :

Incidate max lines on a page and max characters on a line.

No. of lines per page = 10 to 255

No. of character per line = 60 to 132.

syntax : Page [length],[width]

↓ lines per page

↳ characters per lines.

(11) TITLE :

Used to provide TITLE for program

max char in title = 60

TITLE Here is title.

(12) ALIGN :

Align the next instruction on an address which corresponding to specified values.

ALIGN number

↳ ranges 2, 4 --- power of 2

Align 2 → start data seg on word boundry.

Align 4 → start data seg on double boundry

(13) GLOBAL :

(14) Extern :

This indicates that names or label followed with Extern are defined in some other assembly module publically.

```
extern DISP ; FAR
```

Here disp is label of far type procedure present in another module.

so assembler will put info in obj code to link this two modules with the help of linker.

(15) Global :

instead of using public & extern we can use global.

```
Global factor
```

Here factor is global so we can used it all modules that are linked

(17) Include :

Use to insert block of source code from name file into current source module

```
include path : file. name.
```

(18) Name :

assign name to each module when program consisting several modules.

(19) Public :

It informs assembler and linker that identified variable is available for all linked modules.

## + Operators :

① SHORT :
operator tells that only 1 byte displacement is needed to code a jump instruction

② TYPE :
operators find type of variable (actually finds no. of bytes in variable)

for byte type → it gives 1 value
for word type → it gives 2 value

③ Length :
find no. of elements in named data item like a string or an array.
length is in Hex

CX contains length

④ DUP :
for allocation of space.
It is used after storage allocation directive DB, DW

FORMAT : var_name datatype number dup (val)

⑤ Offset :
determines offset.