Module II

Basics of Assembly Language.

=> Instruction : opcode and Mnemonic =

* Inst- is a binary pattern representing a basic operation that can be designed inside a MP to perform a specific ().
* collection of such basic operations supported by a MP → Inst- set of MP.
* Inst- set determines wht () the MP can perform
* Each inst- in an inst- set has 2 parts : operation code (opcode) & operand.
- opcode part of inst- specifies the task to be performed by the MP.
- opcode of 8085 MP can specify basic operations like data transfer operations, arithmetic & logical operations, branch & loop operations, etc.
- operand part of inst- specifies the data to be operated on.

* various techniques to specify data for inst- in 8085 MP are —

a) 8-bit / 16-bit may be directly given in the inst- itself.
b) Address of memory location, I/O port / I/O device may be given in inst- itself.
c) Some inst- specify 2 (R).
d) In some inst-, data is implied.

* Due to different ways of specifying data for inst-, the machine codes of all inst-s are not of same length.
* In 8085 MP, inst- may of 1-byte, 2-byte & 3-byte in length.

* Assembly lang uses a mnemonic to represent each low-level machine inst- (opcode) & also each architectural (R), flag, etc.
* Mnemonic is an abbreviation for a machine lang operation.
* Each mnemonic inst- typically consists of a mnemonic opcode & zero / more symbolic operands.
* Assembly lang usually has 1 statemnt per machine inst- (1:1). Bcz assembly inst- depends on machine code inst-, every assembler has its own assembly lang
* A. lang also → symbolic machine code.
* 8085 MP has 80 basic inst-s & 246 variations in these inst-s.

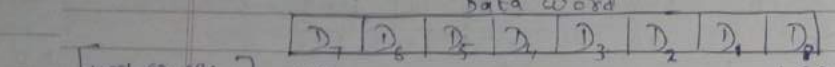
=> Instruction format of 8085 =

* Specific info fields of an inst- include —

a) operation code = Specifies the operation to be performed.
- operation is specified by binary code, hence the name operation code.
- Eg → for 8085 processor, operation code for ADD B inst- is 80 H.

b) Source / Destination operand =
- It directly specifies the source / destination operand for the inst-
- Eg → MOV A, B → B (R) contents as a source operand & A (R) contents as destination operand. bcz this inst- copies the contnts of (R) B to (R) A.

- Data in 8085 is stored in the form of 8-bit binary integrs -



[Most significant bit] (Least Sig)

- In 8085, bit 0 is refereed to as LSB & bit 7 is refereed to as MSB.
- 8085 instn may be 1, 2 / 3 byte in length.

⇒ Addressing mode of 8085 =

* There are diffent techniques by which the data / address of the data to be operated upon may be specified in an instn. This diffent technique → A. mode.
* 5 types.

(1) Direct Addressing	(2) Register Addressing
* Address of the oprand is explicitly specified within the instn itself. * All instn are 3-byte long. bt IN & OUT instn are 2-byte long. * eg → LDA 8000H. STA 8000H IN 02H OUT 01H	* Instn specifies the (R) in which the data is located. * All instn are 1-byte. * eg → ANA B MOV A, B ADD B SUB D

* LDA 8000H = Transfr the contnts in memory location 8000H to ACC.
* STA 8000H = Stores the contnts of ACC in memory location 8000H.
* IN 02H = Read 8-bit data frm port with address 02H to ACC.
* OUT 01H = output 8-bit data frm ACC to port with address 01H.

* MOV A, B = MOVE contnts of (R) B to ACC.
* ADD B = Add contnts of (R) B to ACC.
* ANA B = AND contnts of (R) B with ACC.

(3) (R) Indirect (A)	(4) Immediate (A).
* Instn specifies a (R) pair which contains the memory address where the data is located. * eg → MOV A, M ADD M SUB M DCR M.	* operand is specified in instn itself. * Data either an 8-bit quantity / a 16-bit quantity. * eg → MVI A, 62H. LXI H, 8150H. ADI 38H. SUI 38H ANI 21H.

* MOV A, M = Move contnts of memory location whose address is in H & L (R) to ACC.
* ADD M = Add the contnts of M. location whose add. is in H & L (R) to ACC.
* SUB M = Subtract .... H & L (R) frm ACC.

* DCR M = Decrement the contnts in m location whose add. is in H & L (R) by 1.

* MVI A, 62H = Move data 62H to ACC
* LXI H, 8150H = Load HL pair with data 8150H.
* ADI 38H = Add data 38H to ACC
* SUI 38H = Subtract data 38H from ACC
* ANI 21H = AND data 21H with the contnts of ACC.

(5) Implicit Addressing =

* In all a·l logic instn that require 2 operands, 1 operand is assumed to be in the Acc. & this is not explicit in the instn bt it is implied. This type of addressing where the opcode does not specify operand explicitly bt it is implied → I. A mode

* eg →
RLC = Rotate the contnts of ACC left by 1 position.
RRC = Rotate the contnts of ACC right by 1 position.
CMA = Complemnt the contnts of ACC.

=> Instruction Set =

* Instn is a binary pattern entered through an input device in memory to command the MP to perform that specific f°.
* 8 bit MP can have $2^8 = 256$ combinations of bit pattern.
* 8085 uses 246 combinations & represnt 74 instn
* There 74 diffrnt instn → Instruction set.
* Instn classification — (5 types)
  a) Data tranfer inst
  b) Arithmtic inst
  c) Logical inst
  d) Branch inst
  e) Machine control inst

* Each instn has 2 parts :
  a) opcode = operation to be performed.
  eg = ADD, SUB, ...
  b) operand = data to be operated.
  eg → ADD 2,3 , 2 & 3 are operand.

* Notations used in 8085 for instrn →
  a) M = Memory location pointed by HL (R) pair
  b) R = 8-bit (R).
  c) R_s = Source (R) (frm)
  d) R_d = Destination (R). (to).

I) Data transfer inst = (11)

Syntx. eg

* MOV Rd, Rs eg = MOV B, A.
  Source (R) always at last.
  Copy data frm source (R) into destination (R)

* MVI R, 8-bit eg = MVI B, 4FH.
  Load 8-bit data into a (R). (hexa).

MVI R → move Immediate data to R.

* LXI Rp, 16-bit — eg = LXI B, 2050H (20H – B, 50H – C)
  - L → Load
  - X → 16-bit
  - I → data.
  - [16-bit → 2 (R)]

Input output operation:

* OUT 8 bit = eg = OUT 01H
  - write data frm acc to an o/t port
  - OUT → output
  - 01 → port address.

* IN 8-bit = eg = IN 07H.
  - Read data byte frm port & store it in acc.
  - IN → input.

* LDA 16-bit = eg = LDA 2050H.
  - LD → load
  - A → ACC.
  - [16-bit-ൽ ഉള്ളത് A-യിലേക്ക് load ചെയ്യുന്നു]
  - Copy data byte into A frm memory specified by 16-bit address.

* STA 16-bit = eg = STA 2070H.
  - ST → Store
  - A → ACC.
  - [A-യിലെ ഉള്ളത് 16-bit-ലേക്ക് store ചെയ്യുന്നു]

* LDAX Rp = eg → LDAX B.
  - LD → load
  - A → ACC.
  - X → 16-bit
  - [B ഉം C ഉം ഉള്ള data എടുത്ത് ACC-ലേക്ക് load ചെയ്യുന്നു]

* STAX Rp = eg = STAX D
  - Store Acc 16-bit
  - Rp → (R) pair.

* MOV R, M = eg = MOV B, M.
  - [memory location-ൽ ഉള്ള data എടുത്ത് B-യിലേക്ക് move ചെയ്യുന്നു]

* MOV M, R. = eg = MOV M, C.
  - [(R) content-ൽ ഉള്ളതിനെ memory-യിലേക്ക് move ചെയ്യുന്നു]

=> Registers =
  - used to store data temporarily during the execution of prgm.

	W (R)	Z (R)	→ cannot used for users; used for Mp's internal operations
8-bit	A (R)	Flag (R)	Flag → carry.
	B (R)	C (R)	A, B, C, D, H, L — used for users
	D (R)	E (R)	
	H (R)	L (R)	
16-bit	S.P		Store address.
	PC		

1) GPR = B C, D E, H L
  - 8-bit (R)
  - user accessible
  - combined as (R) pair for 16-bit operation.
  - eg → Higher byte – B, lower byte – C.

2) Temporary (R) =
  - W, Z
  - not accessible for user.
  - 8-bit (R).
  - Store data during a.l.u logic operation

Set — 1
Reset — 0.

3) Sple rp. (R) =

a) (R) A = ACC
* Result of a. & logic operation
* Input / output operation.

b) Flag (R) = 8-bit (R).

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	X	AC	X	P	X	CY

- S → sign: 1 – -ve, 0 – +ve
- Z → zero: 1 – zero, 0 – non-zero.
- AC → Auxiliary carry
- P → parity: 1 – even parity, 0 – odd parity. eg → 10110110, 1's → 5 (odd) → odd parity
- X → not defined
- CY → carry flag [last msb aclogic carry എന്നാൽ flag set (1) ആകും. If not 0]

```
    1
 1 0 0 1 | 1 0 0 0 +
 1 0 0 1 | 1 0 0 1
(1) 0 0 1 1 | 0 0 0 1
```

carry flag. A. carry.

(lowest bit -ൽ carry ഉണ്ടെങ്കിൽ — A. carry).

c) Instn (R) = used for storing the instn currently being executed.

4) 16-bit (R) =

a) SP = 16 bit (R) used to store address of most recent stack entry

b) PC = 16 bit address of next instn to be fetched.

II Arithmetic Inst... = (19)

add

* ADD R — eg = ADD B.
  content of B + content of ACC & result stored in ACC.

* ADI 8-bit — eg = ADI 37H.
  I → immediate data
  Add 8-bit data to the content of A.

* ADD M. — eg = ADD M.
  Add content of memory to content of A.
  [memory-ിൽ ഉള്ള HL pair -ൽ ഉള്ള data + ACC data]

with carry (C)

* ADC R — eg = ADC B
  B(R) content + ACC (R) content + carry previous addition. (carry flag).

* ACI 8-bit — eg = ACI 37H.

* ADC M. — eg = ADC M.
  Add content of memory to content of A with carry.

sub

* SUB R — eg = SUB B.
* SUI 8-bit — eg = SUI 37H
* SUB M — eg = SUB M.

with borrow (B)

* SBB R — eg = SBB B
* SBI 8-bit — eg = SBI 37H
* SBB M — eg = SBB M.

R → registes
R_p → Registes pais.

Increment (+1)

* INR R eg = INR D.
  increment the contnt of (R).
* INR M eg = INR M
  HL pair contnt +1.

Decrement (-1)

* DCR R eg = DCR D
* DCR M eg = DCR M.

Increment 16bit data

* INX R_p eg = INX B
  X → 16-bit (Sec +1)
  Increment the contnts of R_p.

Decrement 16-bit data

* DCX R_p eg = DCX B.

add 16-bit data

* DAD R_p eg = DAD B.
  DAD → Doulele add.
  default {Add contnts of R_p to contnt of HL
  R_p & rslt stored in HL.

III Logical instr... = (19)

AND

* ANA R eg = ANA B
  Logically AND the contnt of (R) with contnt of A.
* ANI 8-bit eg = ANI 74H
  I → immediate data
  logically AND 8-bit data with contnt of A.

* ANA M eg = ANA M.
  logically AND contnt of M with contnt of A.
  (HL - R_p)

OR

* ORA R eg = ORA B
* ORI 8-bit eg = ORI 74H
* ORA M eg = ORA M.

XOR

* XRA R (exclusive-OR) eg = XRA B
* XRI 8-bit eg = XRI 74H
* XRA M. eg = XRA M.

combare

* CMP R eg = CMP B.
  combare the contnt of (R) with contnt of A.
  (Subtraction used)
* CPI 8-bit eg = CPI 34H.
* CMP M. eg = CMP M.
* STC (Set carry flag)
* CMC (Complemnt carry flag)
* CMA (Complemnt ACC)

bit rotate

* RAL (Rotation left) → data 1-bit left shift
* RAR (Rotation right) → " " right "
* RLC (Rotate left with carry)
* RRC (" right with carry.)

IV Branch instn = (7)

* JMP 16-bit address eg = JMP 2050H.
  (Jump to 2050 adres)
  change prgm sequence to specified memory.

* JZ 16-bit address eg = JZ 2050 H.
  (JZ → Jump zero to 16bit address)
  if the zero flag is set.

* JNZ 16-bit address eg = JNZ 2050H.

JNZ → jump non-zero

If the zero flag is reset.

* JC 16-bit eg = JZ 2050H.

JC → Jump with carry.

If the carry flag is set.

* JNC 16-bit eg = JNC 2050H.

JNC → Jump non zero with carry.

If the carry flag is reset.

* CALL 16-bit eg = CALL 2040H

[ഒരു particular sub prgm -നെ call ചെയ്യാൻ വേണ്ടി Mp -യ്ക്ക് ഉപയോഗം]

change the prgm sequence to the location of subroutine.

(ആ location-ന്റെ address ആണ് ഇവിടെ കൊടുക്കുന്നത്)

* RET

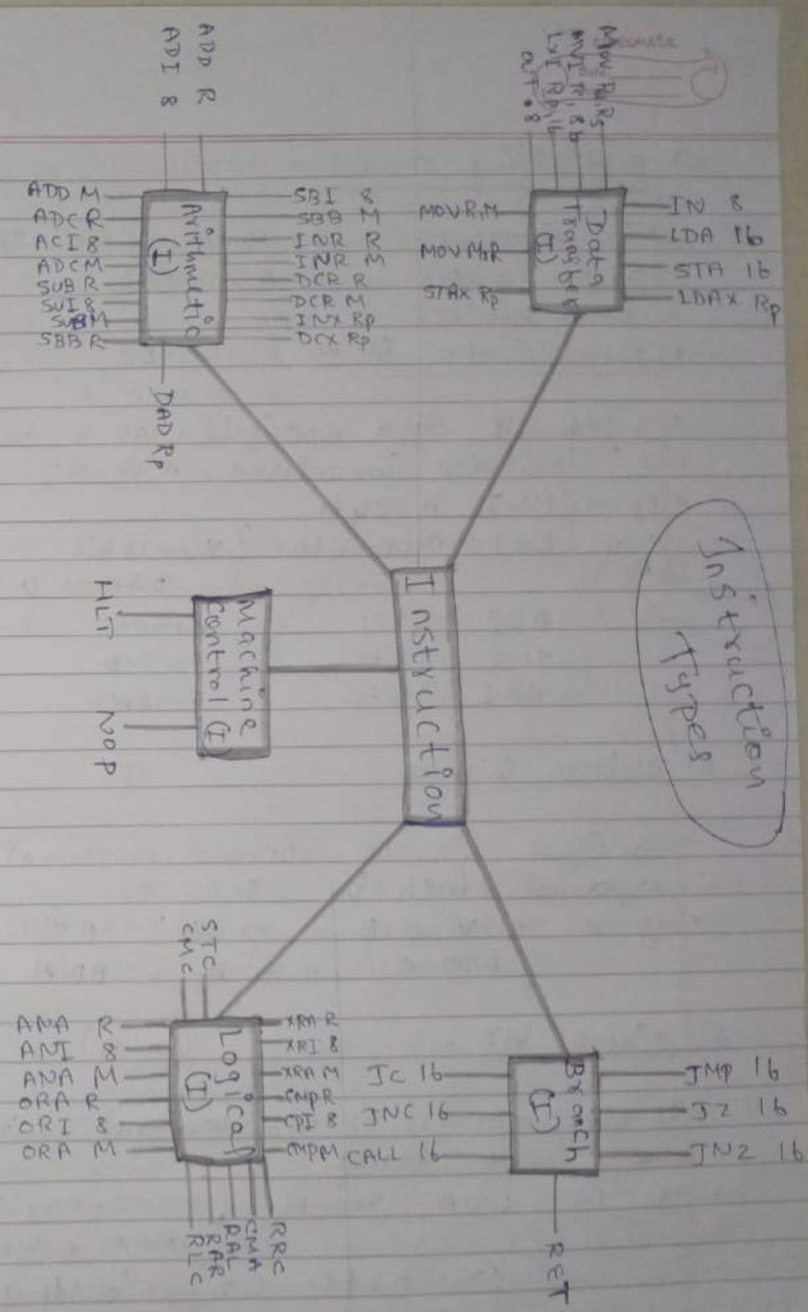
RET → return to the main prgm after completing subroutine prgm.

V Machine Control instn = (2)

* HLT → stop processing ie wait

* Nop → No operation ⇒ Nop.

Do not perform any operation.



⇒ Addressing modes =

* The different ways that a MP can access data → A. modes
* 5 A. modes.

1) Immediate (A) = (I)

* 8 | 16-bit data specified as a part of inst
* 'I' indicates immediate A. modes.
* eg → MVI A, 34H
  LXI D, 10FFH } X → 16 bit
  D & E ⇒ D = 10H, E = FFH
  ADI 45H → add
  SUI 37H → Sub.
  ANI 74H. → AND

2) Register (A) = (R)

* Specifies source operand, destination operand | both in 8085 (R).
* eg → MOV A, B ⇒ (8bit) B → A (8bit)
  ADD C, ADC B, ANA B.

3) Direct (A) = (nn H)

* 16-bit address of the operand within the inst~ itself
* eg → LDA 3000 H → (3000th rom ad-222 data A acc load)
  IN 07H. } input/output
  OUT 01H. }
  STA 2070H → (Store acc data A to memory)

(I) → inst~

4) Register Indirect (A) = R_p

* Memory address where the operand located is specified by the content of R_p
* eg → LDAX B → (X-means 16-bit so R_p)
  MOV M, A.
  STAX B

5) Implicit (A) =

* opcode specifies the address of operand.
* only opcode, no operand.
* eg → CMA, RAL, RLC, STC } opcode.

⇒ Prgm Control Instr~ =

* machine code that are used by machine/ in assembly lang by user to command the processor act accordingly.
* 6 types —

a) Combare (I) = CMP R_1, R_2

b) Unconditional Branch (I) = JMP L2, MOV R_3

c) conditional Branch (I) = BE R_1, R_2, L1

BE → Branch if Equal.
checks if the cntnts of 2 (R) (R_1 & R_2) are equal, if they are equal then the prgm will jump to the M. address (L_1), if they are not equal the prgm will continue executing the next inst~

d) Subroutines =
prgm fragmnt that lives in user space, performs a well-defined task.
eg = CALL and RET.

e) Halting inst = Nop & HALT.

f) Interrupt inst = RESET, TRAP, INTR.

⇒ Instruction cycle =

* Time required to execute & fetch an entire instru → I·cycle.
* It consist :

a) Fetch cycle =
next inst. is fetched by address stored in pc & then stored in the (I) (R).

b) Decode (I) =
Decorder interprets the encoded (I) frm (I) (R).

c) Reading effective address. =
The address gvn in (I) is read frm main memory & required data is fetched.

d) execution cycle =
* Consist of memory read (MR), memory write (MW), Input output read (IOR) & input output write (IOW)

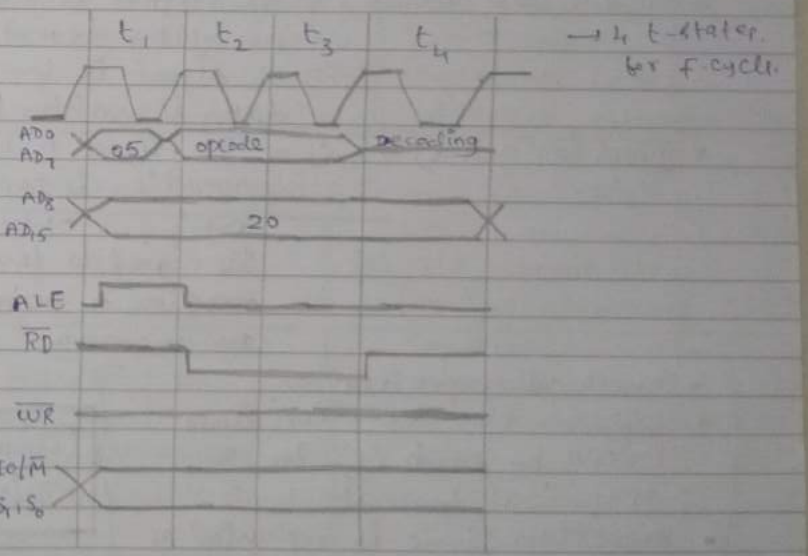
e) Machine cycle =

[Fetch cycle → 4 t-states
execution cycle → 3 t-states]

Time required by the Mp to complete an operation of accessing memory / input output devices → M·cycle
* t-state = 1 time period of freq of Mp.
• t-state is measured frm the falling edge of 1 clock pulse to the falling edge of nxt clock pulse.
* Fetch cycle takes 4 t-states & execution cycle takes 3 t-states.

⇒ Timing Diagram =

T.D for fetch cycle / opcode fetch =



* 05 = lower bit of address where opcode is stored.
Multiplexed add & data bus $AD_0 - AD_7$ used

* A0 = higher bit of ad where opcode is stored.
M. address & data bus $AD_8 - AD_{15}$ used.

* ALE = provide signal for multiplexed ad & data bus
If signal is high |1, m. ad & data bus will be used as ad bus
If signal is low |0, m. ad bus will be used as data bus

* RD (low active) = If signal is high |1, no data is read by Mp
If signal is low |0, data is written by Mp.

* WR (low active) = If signal is high |1, no data is written by Mp
If signal is low |0, data is written by Mp.

* IO/M (low active) & S_1, S_0 = If signal is high |1, operation is performing on input output.
If signal is low |0, operation is performing on memory.

- Graphical representation
- represents the execution time taken by each (I) in a graphical format
- Execution time is represented as T-States

Timing Diagram

=> Memory Read & write machine cycle =

I Memory Read M. cycle	II Memory write M. cycle
* 8085 executes the memory read cycle to read the contnts of R/w memory/ROM.	8085 executes the memory write machine cycle to store the data into data memory.
* length of this M. cycle is 3 T-States ($T_1 - T_3$).	length of this M. cycle is 3 T-States ($T_1 - T_3$)
* Here, processor places the address on the address lines from the stack pointer, HPR pair /pc & through the read process, reads the data from the addressed memory location.	Here, processor places the address on address lines from stack pointer / HPR pair & through the write process, stores the data into addressed memory location

=> Addressing I/O Devices = (2 types)

1) memory mapped I/O	(2) I/O mapped I/O. (peripheral mapped I/O)
* Input/output devices are treated as memory locations & the data is transferred b/2 Mp & input/output devices.	Input/output devices are mapped to specific input/output ports.
✓ * Addressing syntax is memory address space	Addressing syntx is I/O address space
✓ * Access through memory access	I/O access.
✓ * multiple bus cycles	single bus cycle.
✓ * eg → 8086, ARM	eg → 8085, 6502

→ Assembly lang =

* prgm is a set of inst. arranged in the specific sequence to do the specific task
* Instead of using machine opcodes directly, I may also write a prgm using mnemonic operation codes (called A. lang) corresponding to the machine opcodes.
Such prgms written in A. lang of a Mp → assembly lang prgm.
* Before executing an A. lang prgm that contains mnemonic opcodes must be translated into its equivalent native machine opcodes using a translator prgm → Assembler.

→ programs →

1) Store the data byte 52H into memory locations 2000H.

A) 2000H →(store) 52H

5 2
8 → 0101 0010
16 → 2000

(2 ways)

52 → A now — MVI A, 52H.
A → M.loc — STA 2000H
Stop. — HLT.

(STA → store Acc to 2000H)

① MVI M, 52H. (mov I data 52 to M. location)
(or)
② MVI A, 52H.

(or)

① .
LXI H, 2000H — (load 16-bit I. data to H.)
MOVI M, 52H
HLT

2000 H
H = 20H L = 00H.

2) Addition of 2 8-bit no. (20H & 40H)

A) for addition → ADD B (or) / ADI 8-bit (or) / ADD M.

store 2 no. { MVI A, 20H.
MVI B, 40H.
ADD B → perform operation
STA 2200H. → 2200H memory loc - store answer.
HLT.

3) Add the content of m.loc 2000H & 2001H & place the rslt in m.loc 2002H.

A)
LXI H, 2000H
MOV A,M
INX H. — INX → increment 16-bit.
ADD M — & store rslt in Acc.
INX H — 2001. & ADD M result → A.
MOV M,A — A → M.
HLT

4) Sorting of an array of no.

05 → 3041H (ascending order)
08 → 3042H
07 → 3043H.

A) 03 (total 3 num) store 3040H. (03 → 3040H)

05	3041H	1st → compare 5 & 8
08	3042H	2nd → compare 8 & 7
07	3043H.	Sorted ✓

store 3041H in HL pair (R) → LXI H, 3041H
MOV A,M.   A=05
INX H.   A=08 (3042)
CMP M   (A−M)   A=07 (3043)
05−08=−03 (carry ind)
no change

5) Add 2 8-bit nos (99H & 39H)

A)
2501 H → 99H.
2502 H → 39H.
rslt 2503 H → 99H + 39H = D2H.

```
  + 1001 1001  (99H)
    0011 1001  (39H)
    1101 0010  (D2H)
```

1101 — 13 → D
0010 — 2 → 2

```
(2501H) LXI H, 2501H   ; Get ad of 1st no in HL pair
        MOV A,M        ; Get 1st operand in ACC
(2502H) INX H          ; increment cntnt of HL pair
        ADD M          ; Add 1st & 2nd oprand.
        INX H          ; HL points 2503H.
        MOV M,A        ; Store rslt at 2503H.
        HLT
```

6) Subtract 2 bit nos (49H & 32H)

A)
2501H → 49H.
2502 H → 32H.
rslt 2503H → 49H − 32H = 17H.

```
(2501) LXI H, 2501H.
       MOV A,M
(2502) INX H.
       SUB M.          ; Sub 1st to 2nd oprand
(2503) INX H
       MOV M,A
       HLT
```

7) Add 2 16-bit nos (1C15H & 5AB7H)

A)
1C15 H.   5AB7 H
15 → 2501 H   B7 → 2503 H
1C → 2502 H   5A → 2504 H.
rslt → 1C15 + 5AB7 = 76CCH
CC → 2505 H
76 → 2506 H.

1st way

```
LHLD 2501H   ; Get 1st 16-bit no in HL pair
               LHLD → Load H-L pair direct
XCHG         ; Save 1st 16 bit no in DE
               XCHG → Exchange.
               (Cntnts of HL pair & DE pair)
LHLD 2503H   ; Get 2nd 16-bit no in HL pair
MOV A,E      ; Get lower byte of 1st no.
               E → A (L byte of 1st no)
               'E' (R) contains lower byte of 1st no
ADD L        ; Add lower byte of 2nd no.   (A+L)
               'L' (R) contains lower byte of 2nd no
MOV L,A      ; rslt stored in L (R).
MOV A,D      ; Get higher byte of 1st no.
               'D' → higher byte of 1st no.
ADC H.       ; Add higher byte of 2nd no with carry.
               'H' → higher byte of 2nd no.
MOV H,A      ; rslt stored in H (R).
SHLD 2505 H  ; Store 16-bit rslt of addition
               SHLD → Store HL direct
HLT
```

using DAD (I)

(or) 2nd way

```
LHLD 2501H        ; Get 1st 16-bit no.
XCHG              ; Save 1st 16-bit no. in DE
LHLD 2503H        ; Get 2nd 16-bit no. in HL
DAD D             ; Add DE & HL
SHLD 2505H.       ; Store 16-bit rslt in
HLT                        [2505H & 2506H
```

8) Add contents of 2 memory loc.

A) 2500H → 7FH

2501H → 89H.

rslt 7FH + 89H = 108H

```
  1 1 1 1   1 1 1
  0 1 1 1  1 1 1 1  +        108 H
  1 0 0 0  1 0 0 1           ↓
1 0 0 0 0  1 0 0 0       0 1 08 H.
↓   0         8.
1
                         2502 H → 08H
                         2503 H → 01H.
```

7F →
8 4 2 1
0 1 1 1 → 7
1 1 1 1 → 15(F)
1 0 0 0 → 8
1 0 0 1 → 9

```
LXI H,2500H   ; HL points to 2500H.
MOV A,M       ; Get 1st operand
INX H         ; HL points to 2501H
ADD M         ; Add 2nd operand
INX H         ; HL points to 2502H.
MOV M,A       ; Store lower byte of rslt at 250
MVI A,00H     ; initialise higher byte of
              ; (A) with the value 00H.
ADC A         ; Add carry in high byte rslt
INX H         ; HL point to 2503H
MOV M,A       ; Store higher byte of rslt
HLT                  ] at 2503H.
```

(move immediate) — MVI

9) find 1's complement of 96H.

A) 2501H → 96H (1001 0110)

1's complement = 69H (0110 1001).

rslt → 2502 H → 69H

```
LDA 2501H     ; Get the no. IN ACC.
CMA           ; take its complement.
STA 2502H     ; Store rslt in 2052H
HLT.
```

10) find 2's complement of 96.

A) 96 → 1001 0110.

1's → 0110 1001   69 · +
                    1
2's → 0110 1010   6 10 → 6A
       ↓   ↓
       6   A

```
LDA 2501 H
CMA
ADI ,01 H.    ; Add 1 in the no.
STA 2502 H
HLT
```

11) find larger of 2 no.

```
A) LXI H, 2501H   ; Get ad of 1st no. in HL pair
MOV A,M           ; 1st no. in ACC.
INX H             ; ad. of 2nd no. in HL pair
CMP M             ; Compare 2nd no. with 1st no.
JNC AHEAD         ; Jump if no carry Ahead.
```

(In case of smaller of 2 no. (a) we use JC AHEAD in this line)

(The processor checks the carry flag to determine whether a carry has occurred in the previous arithmetic inst.. If no carry has occurred, the processor jumps to location labelled 'AHEAD' in the prgm.)

```
MOV A, M
STA 2503H
HLT.
```