CS010405: Microprocessor Systems

Credits: 4

Teaching scheme
3 hours lecture and 1 hour tutorial per week

Objectives

- *To impart the basic concepts of microprocessors and interfacing concepts.*
- *To develop an understanding about the assembly level programming.*

Module I (10 hours)

Architecture of 8085 – Registers, Instruction set of 8085 - Instruction Types – Arithmetic – Logic data transfer, Branch, Stack, I/O and Machine Control instructions - Addressing Modes - Direct and Indirect Addressing - Immediate Addressing - Implicit Addressing.

Module II (12 hours)

Subroutines - Stack Operations - Call Return sequence- Programming Examples. Timing and control unit – The fetch operation – Machine cycle and T- State instruction and data flow. Address space partitioning - Memory mapped I/O - I/O mapped I/O.

Module III (14 hours)

Interrupts of 8085 - Hardware & Software Interrupts – Enabling, Disabling and masking of interrupts – Polling – HALT & HOLD states – Programmable interrupt controller – 8259.

Module IV (12 hours)

Data transfer schemes - Programmed data transfer - synchronous and asynchronous transfer - interrupt driven data transfer – DMA data transfer. Study of Interfacing ICs – 8257,8255 programmable peripheral interface (compare it with 8155).

Module V (12 hours)

Programmable interval timer 8253, 8251 -,Interfacing Keyboard and display devices, Hardware and Software approach – USART 8251. (interfacing chips functions and internal block diagram only).

Reference Books

1. Gaonkar -*Microprocessor Architecture, Programming and Applications with the 8085* - New Age International
2. Renu Singh, B. P. Singh -*Microprocessors, Interfacing and Applications* New Age International-Third Edition
3. N K Srinath -*8085 Microprocessors programming and interfacing* - PHI
4. Adithya P. Mathur -*Introduction to Microprocessors Systems* - PHI
5. KK Tripathi, Rajesh K Gangwar -*Microprocessorand its Applications* -Acme learning
6. R.Theagarajan,S.Dhanasekaran,S.Dhanapal -*Microprocessor and ITS Applications* New Age International
7. N Senthil Kumar,M saravanan,s.jeevananthan-*Microprocessor and microcontrollers* -Oxford higher education

4/2/16

History:

- First machine (computer) ⟶ ABACUS
- Arabic number s/m ⟶ 0
- Mechanical devices → eg: Adders, Subtractors.
  - differential engine.
  - Analytical engine.
  - Differential Analyser

  } Disadvantages: huge, slow.

- Electronics
  - Semiconductor technology.
  - Electronic computers: ENIAC → UNIVAC → IBM PC → Apple
    (size ↑, speed ↓)

- Moore's law: no. of ~~tes~~ transistors that can be incorporated into an IC doubles every year.
- IC:
  - Discrete components (1 component in an IC)
  - SSI (10-100 components)
  - MSI (1000 ")
  - LSI (10000 ")
  - VLSI
  - ULSI (millions)

⇒ Generation of computers :

Gen I : 1940s
Large
Vaccum
Weeks to calculate ⇒ slow.

Gen II : transistors, FFs

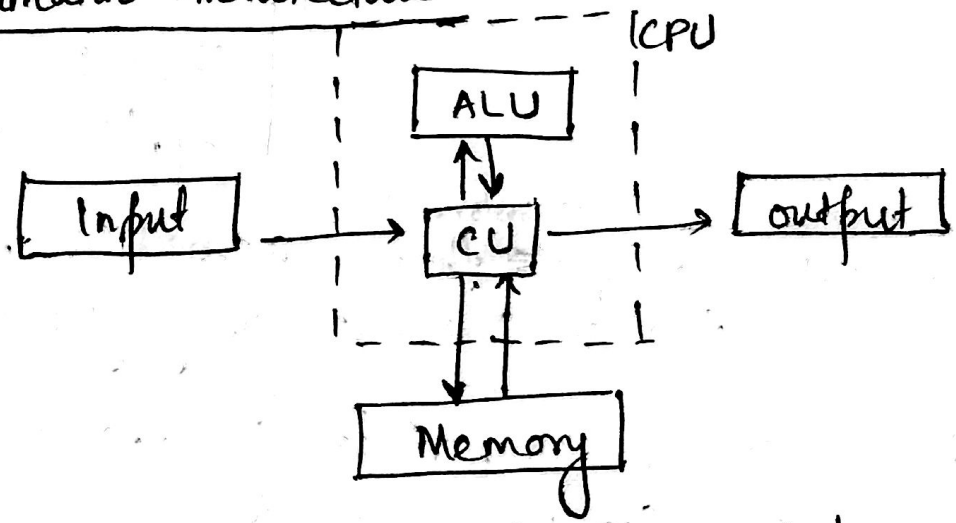
Gen III : ICs, semiconductors.

Gen IV : VLSI
production of PCs (IBM PC)

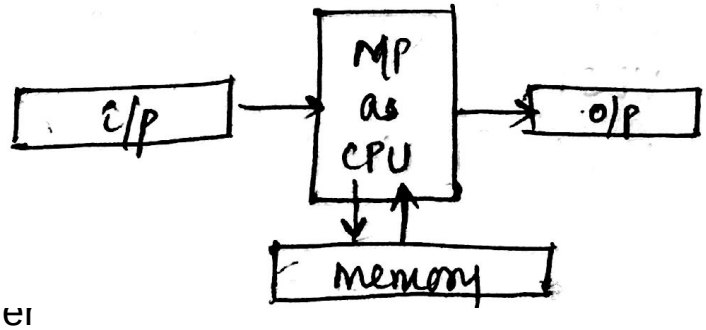
Gen V : High speed computer
Super computer (Intel, Pentium almost included)

Gen VI : Artificial Intelligence
Virtual World.

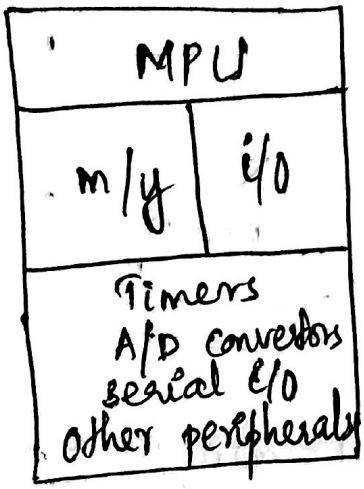
Von Neumann Architecture :



• CPU on a chip is called a microprocessor.



• Computer on a chip. ⇒ microcontrollers.



⇒ Different MPs:

1 bit ⟶ smallest information computer can handle.
4 bit ⟶ nibble
8 bit ⟶ byte
16 bit ⟶ word

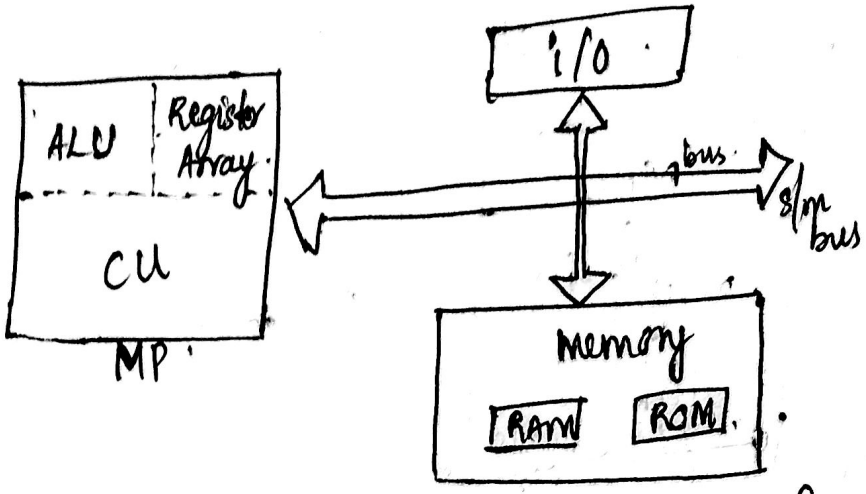
Eg: 4 bit → 4004, 4040 } handle only 4kB of memory. 4x1024

8 bit microprocessor ⟶ 1 byte ~~or~~ of data could be handled
→ 8085
→ 8080 } 64 kB

16 bit µprocessors ⟶ 8086
⟶ 80286

32 bit " ⟶ 80386
⟶ pendium

64 bit " ⟶ intel core i...
⟶ speed increased; size decreased.



- Microprocessor based s/m.

Register ⟶ temporary storage within processor.

- s/m bus
  - Address bus – to transfer addresses.
  - Data bus. – to transfer data
  - control bus – signal transfer.

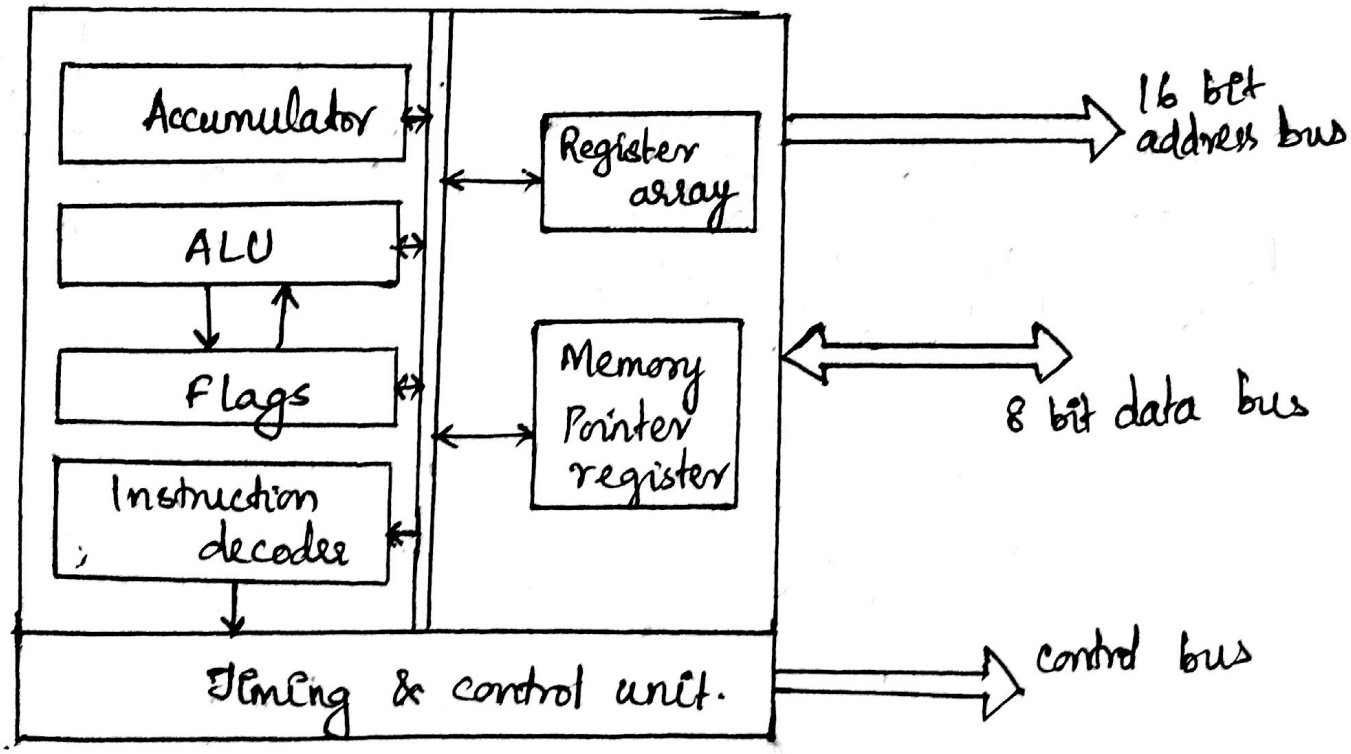
Steps involved:

Fetch ↓ Decode ↓ Execute

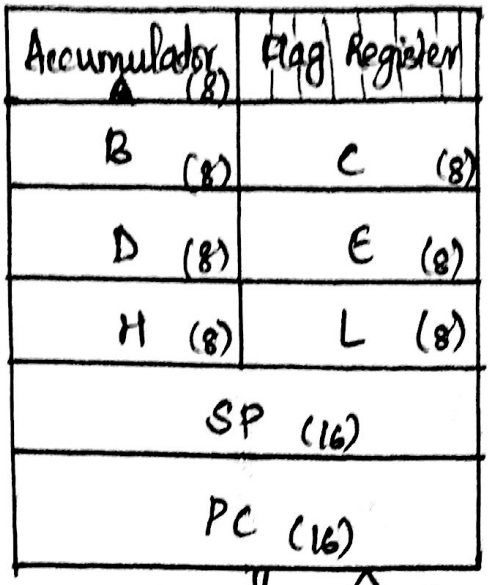
MODULE I.

8085

→ by Intel in 1976
→ NMOS technology (n-type MOS)
→ 40 pin Dual Inline package. (20-20).
→ +5V power supply.
→ 3 MHz single phase clock
→ 8 bit MP
→ 8-bit databus
→ 16 bit address bus
→ 64 kB of memory. ($2^{16} \rightarrow 65536 \simeq 64$ kB).



(Hardware model)



(Programming model)

- 8 or 8 bit GPR (B, C, D, E, H, L)
- Two 16 bit SPR (Stack Pointer, Program Counter)
- Two temporary registers (w, z)
- Flags
- Accumulators (A) (special register)
- Instruction register.

- Stack pointer (beginning addr. of stack, hold next address after jump)
- Program counter (next instruction's address)
- w, z → not used in pgms but inside μproc
  → to move temporarily
  Eg: MOV B, C (② ← ①)
- Flag → to reflect the result of an operation performed or to hold sign bit or carry after operations.
- Instruction register (holds code for an operation).
  Eg: ADD, MOV.

B pair (BC)
D pair (DE)
H pair (HL)

Bit positions of various flags in the flag register of 8085

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

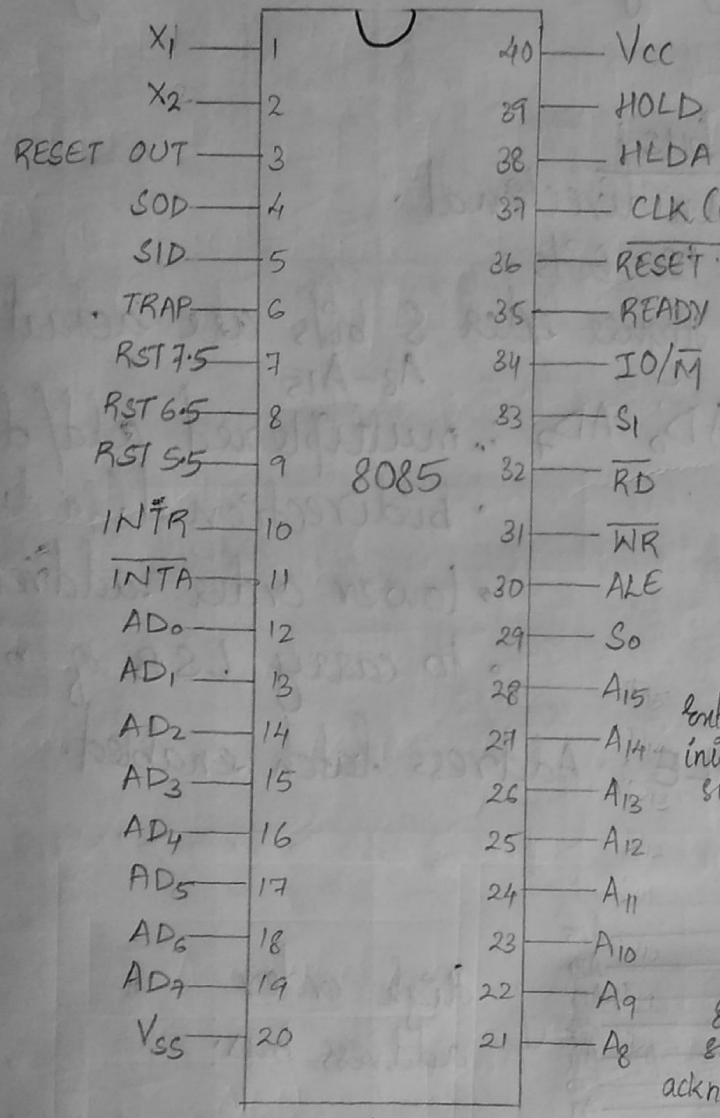
ADD B → Mnemonics

1) 0 flag.
2) carry flag.
3) auxiliary carry flag.
4) sign flag.
5) parity flag

- Accumulator is used in almost all ALU oper. and holds the result after operation.

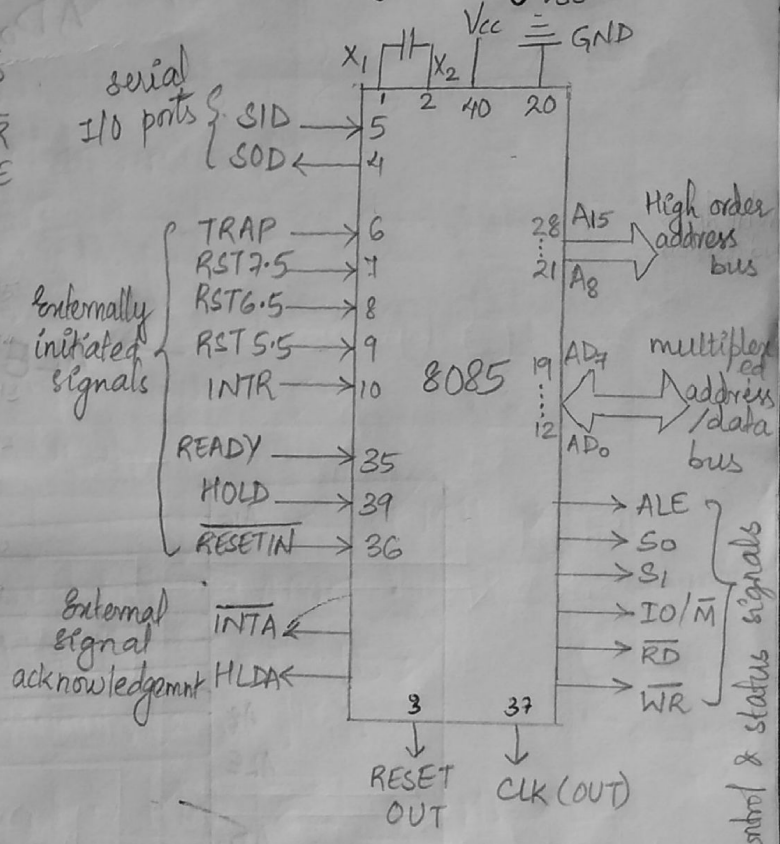
5/8/16 8 bit MP ⇒ $2^8 = 256$ diff kinds of addresses using 0,1 to the 8085 can be given.

⇒ PIN diagram of 8085:



Pin diagram

signal diagram.



Six Groups:

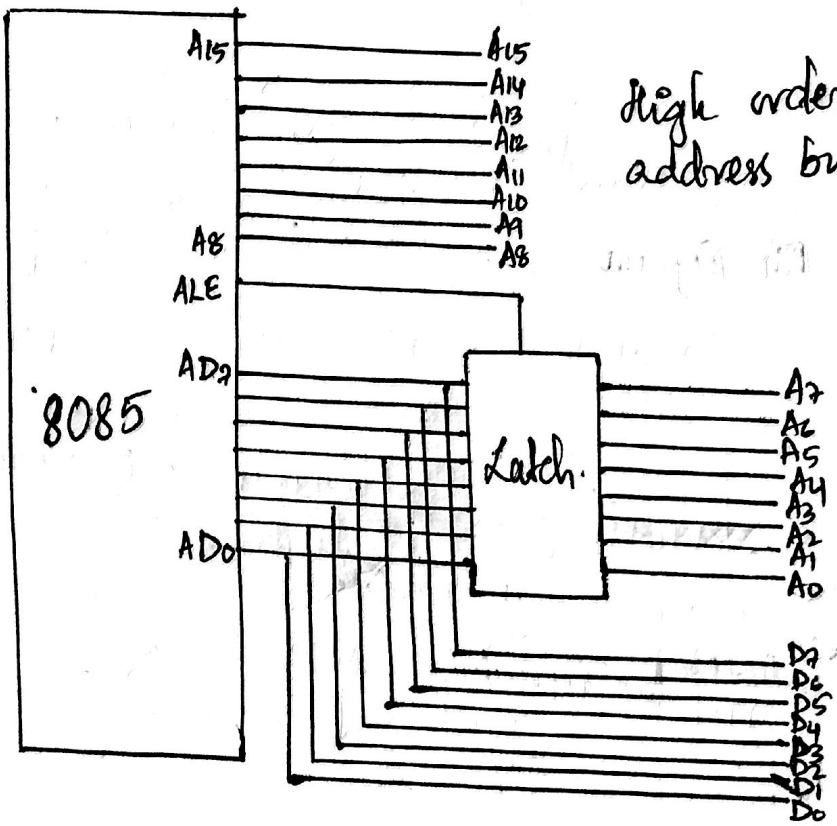
- Address bus
- Data bus
- control & status signals
- power supply or frequency signals
- externally initiated signals
- serial I/O ports

ALE → address latch enabled

Before starting the execution a +ve pulse is given to ALE. It makes 8085 to know that we are going to send the address bits through the buses. When the ALE becomes low it indicates that we are going to send data through the pins $AD_0 \rightarrow AD_7$.

8|2 ⟹ Address and data bus:

* Address bus → unidirectional.
  → 16 bits.
  → Higher order 8 bits are actual addre
  → AD_0-AD_7 A_8-A_{15} :. multiplexed add/data bu
    • bidirectional (for data transf
    • lower order address bus
    • to carry LSB of ".
  → ALE : Address latch enabled.



⇒ Control and status signals:

* ALE (address latch enabled)
  → when high; latch is enabled ⇒ address is passed.
* IO/$\overline{M}$
  $\overline{m}$ → active low
  IO/$\overline{M}$ → = 0 → memory.
  IO/$\overline{M}$ → = 1 → IO.
* $\overline{RD}$ → read control signal.
  → active low. $\overline{RD}$ = 0.
* $\overline{WR}$ → write control signal.
  → active low $\overline{WR}$ = 0 performs write oprn either from IO or memory depending on IO/$\overline{M}$ signal.
* S_1, S_0 → special status signals.

$\overline{RD}$, $\overline{WR}$: control signals.
IO/$\overline{M}$, S1, S0 : status signals.
ALE : special signal.

⇒ Power supply and frequency signals:

* V_{cc} → +5V power supply.
* V_{ss} → ground reference.
* X_1, X_2 → RC or LC n/w is connected to it.
  This n/w has a crystal of frequency 6MHz. and RC or LC n/w internally divides this by 2 to produce 3MHz.

* CLK OUT → we use this pin as sys clk (timing mechanism) to other devices external to processor.

⇒ Externally initiated signals:
- externally coming into processor.

* INTR → interrupt the MP (maskable) (delayable)

* RST 7.5, RST 6.5, RST 5.5 } → Restart interrupt and move to specific memory location.

priority: 5.5 < 6.5 < 7.5.

* TRAP → non-maskable interrupt.
→ MP cannot overlook (neglect) the interrupt request coming from TRAP. It must be satisfied at that moment itself.
→ highest priority.
then RST 7.5 ≥ 6.5 > 5.5 > INTR.

* HOLD →
- DMA: Direct memory access.
- If this is enabled, DMA process can be performed without involving processor in the process.

* HLDA → Hold acknowledgement
→ to inform reception of HOLD signal.
→ outward signal.
→ initiated after HOLD signal is received.

* $\overline{INTA}$ → interrupt acknowledgement.
→ corresponding to INTR.

(acknowledgement to external signals)

* READY → used to delay the MP.
→ to inform MP to wait until the I/O or memory peripheral takes time to complete R/W opn. so that it doesn't perform anyother fn at that time

* $\overline{\text{RESET IN}}$ → Resets MP to totally fresh state.
3 thing performed when this signal is enabled.
i) PC = 0
ii) MP is reset.
iii) Buses are tristated (cleared)

* RESET OUT → acknowledgement
→ indicates MP is getting reseted at present
→ outward signal.

⇒ Serial I/O ports:

* SID → serial i/p data.
→ one bit at a time (serially) is read or written.

* SOD → serial o/p data.
→ serially data is given out (one bit at a time)

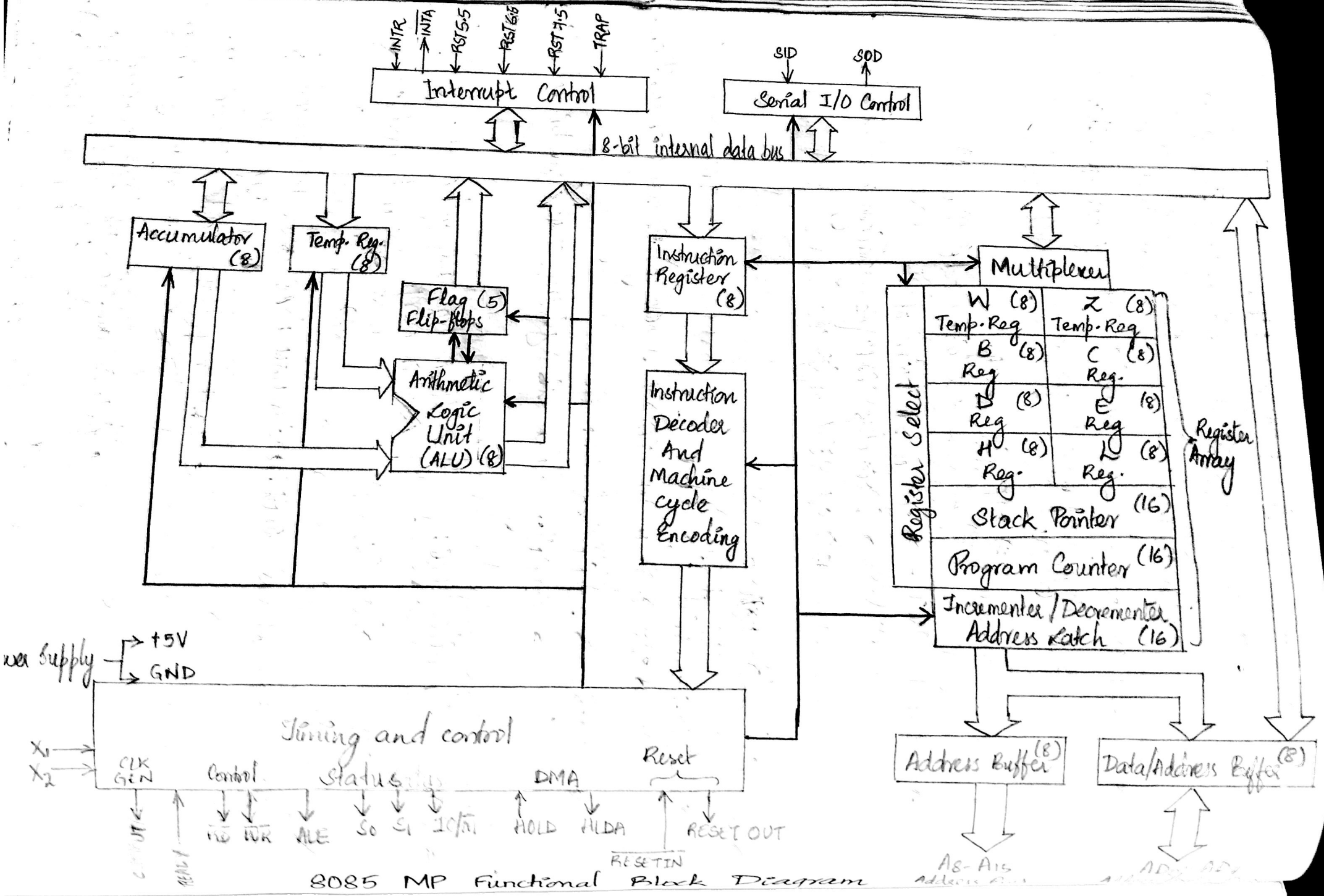
10/2/16

⇒ 8085 Architecture:

ADD B → 80H (opcode)
opcode

- All instrns are represented by opcodes in memory.
  - single operand instrn.
  - two " "

8085's main component is ALU:
- Accumulator: holds 1st operand of any ALU opn
- Temp register: holds 2nd opn " from memory or registe
- ~~ALU~~ Arithmetic and logic circuit : for performing opn and result is given to A.
- Flags: Reflect result of opn.



8085 MP Functional Block Diagram

⇒ Control and status signals :

8085 machine cycle status and control signals.

Machine Cycle	Status $IO/\overline{M}$	S_1	S_0	Control Signals.
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	$\overline{RD}, \overline{WR} = Z$ and $\overline{INTA} = 1$
Hold	Z	X	X	
Reset	Z	X	X	

Z → Tristate (neither ON/OFF)
X → Unspecified

- To execute an instruction, generate opcode (hex code) stored in memory.

2005 → MOV C, A : move contents of accumulator to C reg.
(destn, source)

2006 → ADD B

Convert the mnemonic to hexcode (4F → opcode) entered in the ~~8085~~ main memory (outside processor). PC will initially ~~store~~

be loaded with 1st instn's address → Reg. select unit select PC and activate it → address from PC is taken and placed in address latch → Inc/Dec will ~~d~~ increment value in PC to next address → from the address latch, address is taken to address and data bus → ~~given to internal~~ bus.

16 lines of address comes to main memory and goes to (2005) which contains 4F. In control unit, $\overline{RD}$ signal sent to main memory. Reach loc^n (2005) and takes the inf^n 4F will be taken through the data bus to internal data bus to the instn register → taken to instn decoder → decode 4F → understands the opr^n and tell to control logic. CL activates accumulator. Content taken from A → Reg. ~~sele~~ select ~~signal~~ activates and tell multiplexer to place data in C register from A.

Add → accumulator → ALU → perform add^n → result ~~stor~~ ~~rest~~ stored back in accumulator.

12/2

$2^8 = 256$ instruction from 0 → 256 are possible in 8085. They are written first in mnemonics and then converted to hex code to be stored in MP. ~~Later~~ MP converts it to binary. However, only 246 are used

MOV → ~~80H~~ 4F → 0100 0000.

ADD → 80H → ....

opcode	operand

instruction.

opⁿ code

→ ADD B

→ MVI B, 32H. (move immediate)

→ CMA : implicit operand. (operand in address itself)

: ~~mov~~ complement A reg.

- 8 bit data.
- 16 bit data
- internal regs.
- memory loc^n
- Address
- implicit

} diff kind of operands.

⇒ Instruction Size:

* 1 byte → requires only 1 byte of main mly.
* 2 byte → " " 2 "
* 3 byte → " " 3 " "

- Add B : requires only 1 byte of memory to be stored in mly

↳ 80H

CMA, ~~INR~~ INRA → one byte each

80H	} 1 byte.
3E	} 2 bytes
32	
3A	} 3 bytes.
50	
20	

mly

- MVI A, 32H : 2 bytes are used
  - one for opcode.
  - one for operand

↓ 3E

- LDA 2050H : 3 byte instⁿ.

↓ 3A ; 16 byte operand. requires 2 bytes. ; 2050 : 20 50

20 → higher bytes, 50 → lower bytes.

⇒ Instruction Type:

5 diff categories of instructions in 8085.

⇒ **Data transfer instructions:**

- also diff Data movement or Data copy.

- copies content of source to destn.
- none of the flags are ~~alter~~ affected

a) B/w registers:

MOV Rd, Rs

→ 1 byte

Eg: MOV D, B : move B content to D reg.

- we have A, B, C, D, E, H, L → 7 possible registers

∴ Total 49 instructions are possible here.

7 { MOV A,A MOV B,A ; MOV A,B ; ... MOV A,L MOV B,L ... (7)

⇒ 49 instrns. But MOV A,A, MOV B,B, MOV C,C, ..., MOV L,L are useless.

MOV D, B :

B | 72 | xx | C

D | .. | xx | E

⇒ new content of D after execution of this instrn:

B | 72 | xx | C

D | 72 | xx | E

15/2

b) B/w memory and registers.

MOV Rd, M
MOV M, Rs

} requires 1 byte to store in main memory

whatever content in HL register is taken as content of memory (i.e) address to be accessed

H | 20 | 05 | L

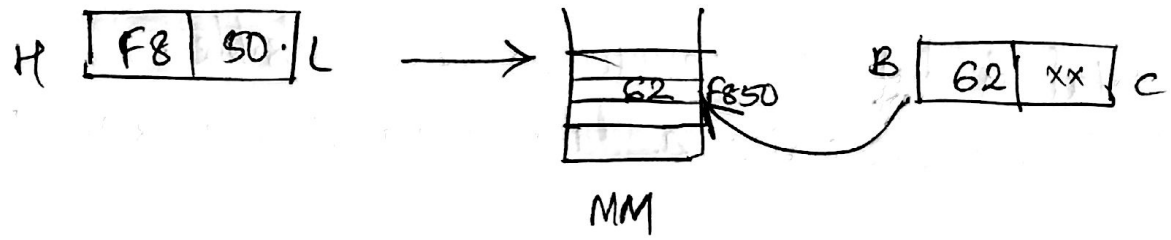
M

∴ MOV B, M ⇒ content in ...

Instruction set classification.

* Data transfer instructions.
* Arithmetic instructions
* Logical instructions
* input/output operation
* stack operations
* Machine control ~~cont~~ operations
* Branch instrn.

So, the instrn MOV M, B is performed as:

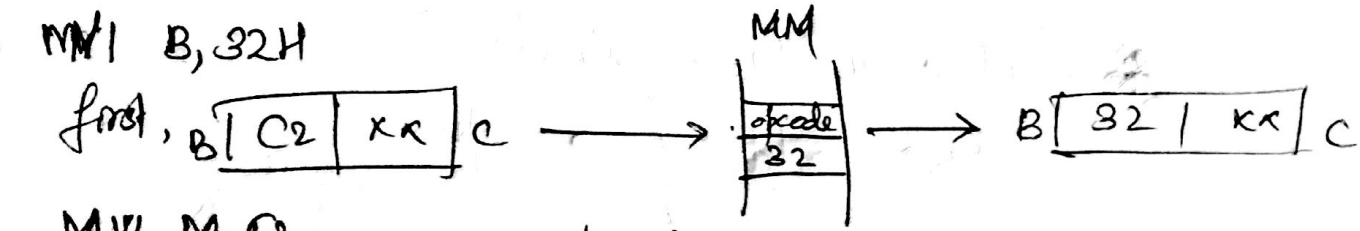


MOV Rd, M ⟶ 7 instructions of this type
MOV M, Rs ⟶ 7 " " " } 14 instructions possible b/w Memory and reg.

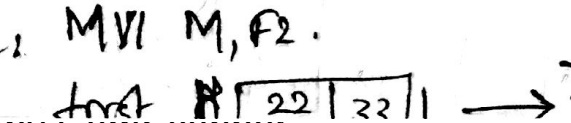
c) Specific data byte to reg or m/y:

- MVI ⟶ move immediate.
- MVI Rd, data. ⟶ 2 bytes reqd.
  MVI M, data. ⟶ " } 8 instructions of this type.

Eg: MVI B, 32H



Eg: MVI M, F2.



d) Load 16 bit number in a reg. pair:

- LXI → Load register pair immediate.
  → Load extended reg. "

- **LXI rp, data**

rp : B, D, H, SP (SP → stack pointer)

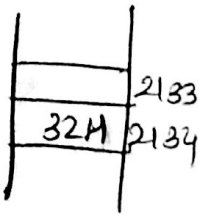
B ⇒ BC
D ⇒ DE
H ⇒ HL

Eg: LXI B, 2050.

B | 20 | 50 | C

Eg: LXI H, 2134 ; MVI M, 32H

⇒ H | 21 | 34 | L → MVI M, 32H →



* total 4 diff LXI instructions are possible.
• allows us to write a data to a specific Memory lo

e) Load and store:

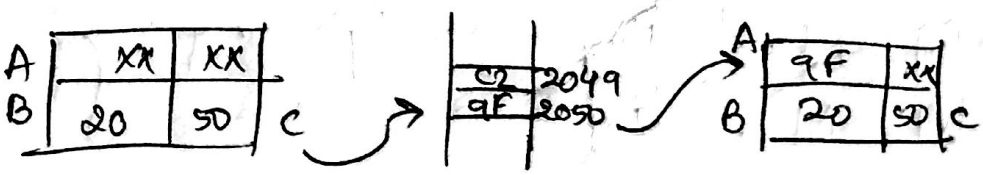
* Load accumulator indirect.

LDAX H ⇔ MOV

1 byte, 2 instr. ← **LDAX Rp.**

Rp → B, D.

Eg: LDAX B : whatever value in BC pair's (16 bit) addressed loc will be loaded to accumulator.



* Load accumulator direct:

LDA addr → 3 bytes

on instr"

Eg; LDA 2233H: A | 63 | ... → C2 (2232), F9 (2233) → A | F9 |

- we directly specify address to be accessed.

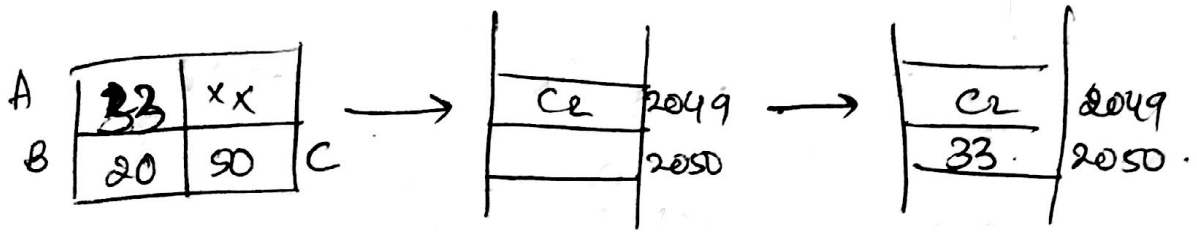
* Store Accumulator Indirect.

STAX Rp

$R_p \rightarrow B, D$.

↳ 1 byte.

Eg: STAX B: whatever content in A is stored to address in BC pair.

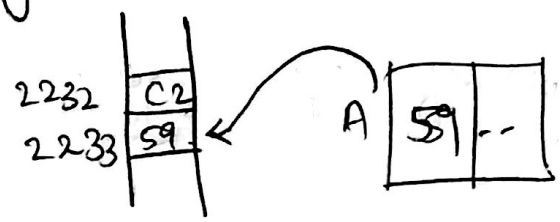


* store accumulator direct

STA addr

→ 1 byte.

Eg: STA 2233H



- STAX H ⟺ MOV M,A

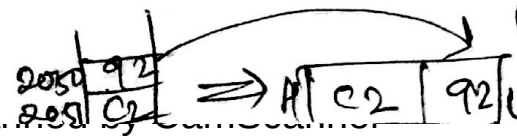
- To add 32+33:

i) MVI A, 32.
ii) MVI B, 33
iii) Add B
iv) STA 2233

MVI M,32.
LDA 2233

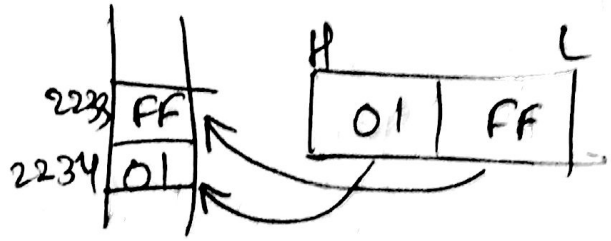
* LHLD addr.

- Load to HL pair.
- LHLD 2050: whatever ~~content~~ address is given, its value is given to L reg and content of next addr (incremented) is given to H.



* SHLD addr → 1 instr.

Eg: SHLD 2233



* XCHG

- exchange contents of HL and DE pair.

H | 20 | 50 | L. → XCHG → D | 22 | 33 | E.

XCHG ⇒ H | 22 | 33 | L. D | 20 | 50 | E

- 49 + 14 + 8 + 4 + 4 + 2 + 2 + 1 → total 84 instructions

⇒ **Arithmetic Instructions:** value of flag is set.

* **Addition:**

ADD R/M

→ 7+1 instructions
→ one byte each.

Eg: ADD B : adds A (first op) content with B (second op)
ADD M.

ADD B : A ('47H) : 0100 0111
B (51H) : 0101 0001
1001 1000 → 98.

98 is stored in A.

value of flag is { S=1 Z=0
AC=0 P=0 CY=0

ADD M (A) → 76H
HL → 2005

0111 0110
1010 0010
1 1000

2005 [A2]

(A) → 98H → 0111 0110
(B) → 91H → 1010 0010
1 0001 1000

S=0
Z=0
AC=0
P=1
CY=1.

2byte [ADI data]

Add immediate

ADI 59H.
(A) ← (A)+59H
A → 4A.

0100 1010
0101 1001
1010 0011

S=1 AC=1
Z=0 P=1 CY=0.

↓
A3.

one bit. [ADC R/M] → add carry (to add 8+ bit nos)
• multibyte.

(A) ← (A) + (R/M) + CY

Eg: BC → 2498
DE → 5491
89

{ ADD.
ADC.

Eg: (A) → 12H
(D) → 4AH

[00001]
S Z AC P CY

ADC D ⇒ 12
4A
1

0001 0010
0100 1010
1
0101 1101 ....
5 D.

[00000]

Eg: ADC M.

(A) → 23

HL → 2005

2005 → 20.

| 0 0 0 0 1 |

soln:

23	0010	0011
20	0010	0000
+ 1		1
44	0100	0100
	4	4.

2byte. **ACI data** : (A) ← (A) + (data) + CY.

Eg: ACI 57H

(A) → 26.

| 0 0 0 0 1 |

CY

soln:

57	0101	0111
26	0010	0110
+ 1		+ 1
7E	0111	1000
	7	E

* Subtraction:

SUB R/M

- 7+1 instructions
- 1 byte each
- flag modified after opn.

SUB R → (A) ← (A) − (R)

SUB M

After subtn, value in carry flag is always 2's complement

Eg: SUB C

(A) → 37H (C) → 40H.

0100 0000
↓
1011 1111
+ 1
11000000

	0011 0111
+	1100 0000
0	1111 0111

0 flag complemented after subtn → | 0 0 0 0 1 |

1/1111 0111

SUI data

→ subtract immediate from accumulator

Eg: SUI 37H

(A) → 40H

40H − 37H ⇒ 0100 0000

+

37 : 0011 0111 2.

* 1100 1000

+ 1

1100 1001

SBB R/M

(A) ← (A) − (R/M) − Borrow (CY) → current CY flag value.

(A) − ((R/M) + Borrow)

SBB B ⇒ (A) → 37H

(B) → 3FH

CY → 1

→ 2 carry setting : during execution

• after operation CY is complemented

SBI data

Eg: SBI 25 H

(A) → 37H

CY = 1

0010 0101

+ 1 (CY)

0010 0110 (CY=0)

1101 1001

+ 1

1101 1010

0011 0110
1101 1010

1] 0001 0000

CY=0.

=> Increment

* Increment:

→ carry not modified.

INR R/M

→ increment R or M.

Eg: INR D.

(D) → FF H.

Ans) 1111 1111
+ 1

neglect 1] 0000 0000
S=0 Z=1 P=1 AC=1 CY=0

INX Rp

→ increment register pair.

Rp → B, D, H, SP

Q→ HL → 9FFF.

INX H ⇒ Ans) A000.

* No flags are affected (none of the flags)

* Decrement: – only carry remain unmodified, all other flags are modified.

DCR R/M	DCX Rp

eg: DCR B
(B) → 00H
add two compl*.

DCX H
HL → F123.

⟹ Logical Operations:

– AND, OR, XOR, NOT

ANA R/M	ANI data

→ logical AND operation
→ S, Z, P modified based on result
→ carry is always reset.
→ AC is always set.

eg: ANI 57H.

eg: ANA D.

(A) → 54
(D) → 82

0101 0100
1000 0010
0000 0000

S=0 CY=0
Z=1 AC=1
P=1

ORA R/M

ORI data

→ S, Z, P.

→ CY, AC & reset.

XRA R/M

XRI data

→ S, Z, P

→ C, AC reset always

CMA

→ complement Accumulator available

→ operand is implicitly ~~specified~~ inside the instruction.

→ 1 byte

→ no flags affected.

CMC → complement carry flag.

→ only CY is complement

→ no other flags are affected.

Eg: 0 0 0 0 1 1 ⇒ 0 0 0 0 1 0

S Z AC P CY

STC → set CY.

→ CY always equal to 1.

⇒ Compare Instructions:

→ for 2 operands : 1st operand is Accumulator.
2nd " " R/M.

CMP R/M

* first the 2 operands are subtracted and CY flag is set based on the result.

(A) < R/M ⟶ CY is set (1) · Z is reset (0)

(A) = R/M ⟶ CY is reset (0) Z is set (1).

(A) > R/M ⟶ CY and Z is reset (0 0).

- Actual subtraction is not happening but operation is subtraction (internally)
- Content of source or operand is not modified.

Same operations for | CPI data |

(A) < data ⟶ CY = 1 Z = 0

(A) = data ⟶ CY = 0 Z = 1

(A) > data ⟶ CY = 0 Z = 0

Eg: CMP B.

(A) → 57H 0101 0111

(B) → 62H 1001 1110

1111 0101

S = 1
P = 1
AC = 1.

→ 0110 0010
1001 1101 (2's)

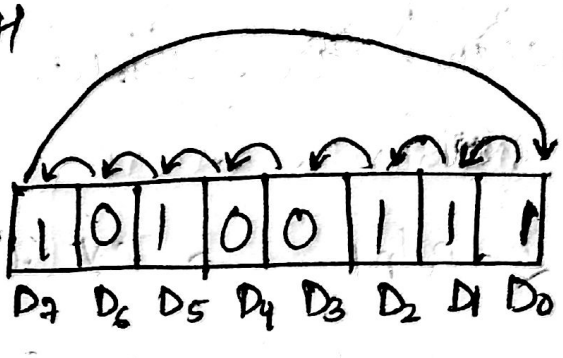
CY = 1
Z = 0.

⇒ Rotate Instructions:

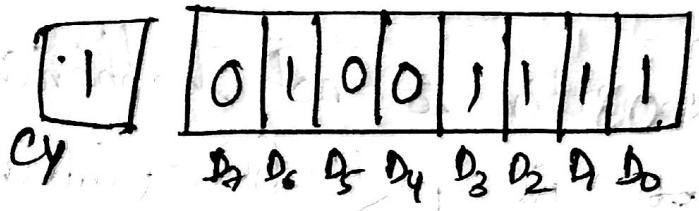
RLC → Rotate (A) left.
→ S, Z, P, AC are not modified.
→ CY is modified based on D_7.

Eg: RLC

(A) → AFH
CY → 0



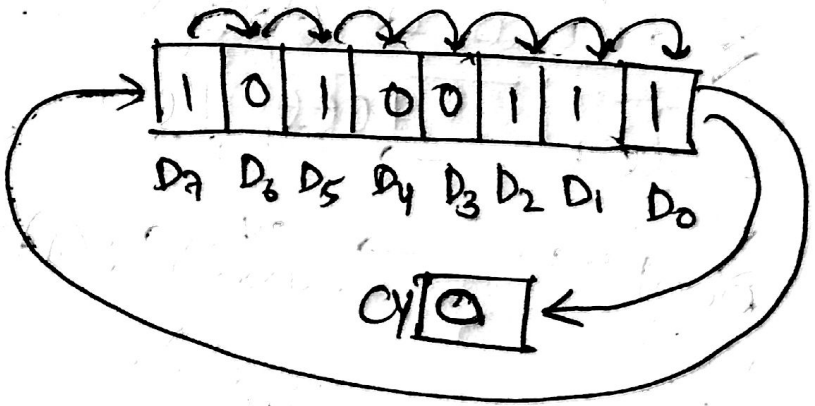
- each bit moved
- $D_7 \rightarrow D_0$
- $D_7 \rightarrow$ CY.



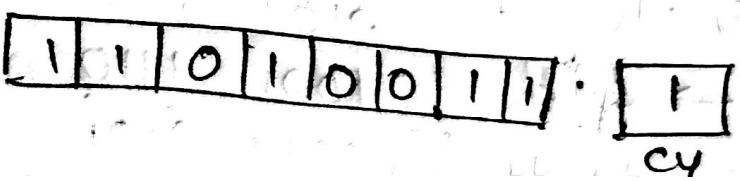
Result:
4F
CY = 1.

RRC → Rotate (A) Right.

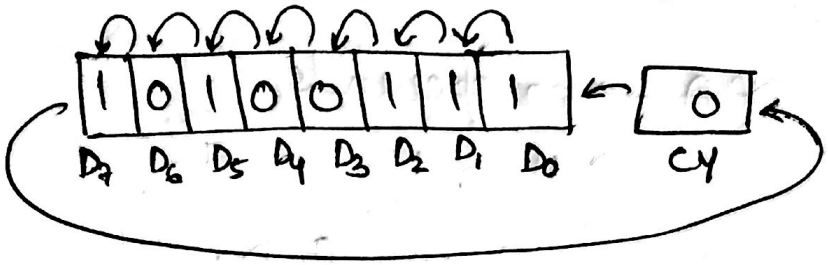
Eg: (A): AFH CY → 0.

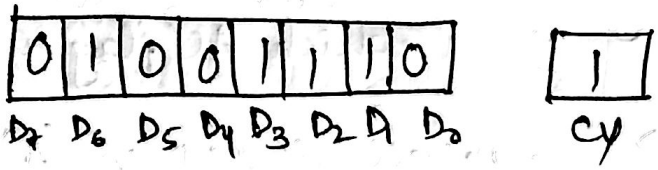


Ans)

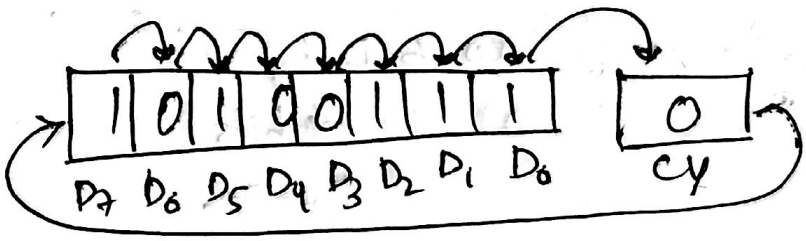


RAL ⟶ Rotate (A) left through carry.

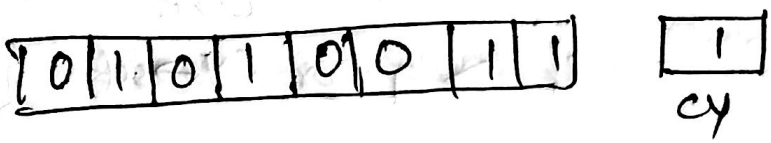




RAR ⟶ Rotate (A) right through carry.



$D_0 \rightarrow CY \rightarrow D_7$.



⟹ **16 bit addition:**

DAD Rp $\quad Rp \rightarrow B, D, H, SP$.

$(HL) \leftarrow (Rp) + (HL)$.

Eg: DAD H $\quad$ H | 02 | 42 | L.

↓

H | 04 | 84 | L

Ans)

$$\begin{array}{r} 0242 \\ 0242 \\ \hline 0484 \end{array}$$

DAA → Decimal Adjust Accumulator.

DAA → no operands
→ all flags are altered.

↳ get an answer in BCD.
→ only instn of 8085 MP of this kind.
→ " " " on which AC flag is used internally.

Eg:
(A) → 39_{BCD}
(B) → 12_{BCD}
} answer stored in (A)

To convert to BCD: ⌐ 8 bits ¬

4 bits	4 bits
$D_4 - D_7$.	$D_0 - D_3$.
+ 60H.	+ 06H
if > 9 or CY → set	if > 9 or AC → set

```
 0011  1001
 0001  0010
-------------
 0100  1011
+ 000   0110   (06)
-------------
 0101  0001
```

Eg: (A): 85_{BCD}.
+ 68_{BCD}.
153.

```
 1000  0101
 0110  1000
-------------
 1110  1101
+ 0110  0110
-------------
1| 0101  0011
```

CY = 1
S = 0
P = 0
= 0
= 1

⇒ Input / Output Instructions:

→ I/O devices.
→ take data to processor. (IN)
→ put n from processor. (OUT)
→ value taken is placed in accumulator.

- to access memory : 16 bit address → A_0 — A_{15}
- to access IO devices : 8 bit n. → A_8 — A_{15} or A_0 - A_7

→ IN port_address
(A) ← i/p device.

Eg: IN 57H.

→ OUT port_addr ⇒ (A) → o/p device.

~~(A)~~

Eg: OUT 60H.

⇒ Branch Instructions:

* Jump instructions — unconditional / conditional
* call and return instructions.
* Restart instructions.

* JMP 16 bit → unconditional: whatever be the status of flags, JMP is executed.

Eg: 2000 JMP 2050

2050 → ADD B
INRA.

- Based on status of AC flag - no conditional jump instruction is given.
  But based on other 4 flags, we can have conditional jump instruction.
  S, Z, P, CY.

JC	16 bit	(CY=1)	→ Jump on carry.
JNC	16 bit	(CY=0)	→ Jump on no carry.
JP	16 bit	(S=0)	→ Jump on +ve (plus)
JM	16 bit	(S=1)	→ Jump on minus.
JPE	16 bit	(P=1)	→ Jump on even parity
JPO	16 bit	(P=0)	→ " " odd "
JZ	16 bit	(Z=1)	→ " " zero
JNZ	16 bit	(Z=0)	→ " " non-zero

→ call and return instructions are used to manage subroutines.

CALL 2050
↓
moves to destination.
↓ fn executed.
return on encountering RET

→ Restart instructions are to restart after interrupt.

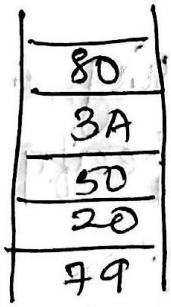
⇒ Machine control Instructions:

* **HLT** → Halt.
  - stop execution (as final step in pgm).
  - buses in high impedance.
  - Registers unaltered.

* **NOP** → No operation.
  - To introduce the delays (increase processing time)
  - Troubleshooting.

* Related to interrupts.
  * EI ; enable interrupt.
  * DI ; disable interrupt.
  * SIM ; set interrupt mask.
  * RIM : Read interrupt mask.

Eg: ADDB.
LDA 2050.
ADD A



ADDB
NOP
NOP
NOP.
ADD A.

⇒ Addressing Modes:

* Immediate
* Register.
* Direct / Absolute
* Register Indirect.
* implicit / implied

operands

- 8/16 bit data
- Register
- memory location.
- 8/16 bit address.
- Implicit.

MVI A, 87H ; LDA 2500

ADD B

MOV A, M

[illegible] 50H (from i/p).

CMA

Immediate : operand is immediately specified in instruction itself. MVI A, 87H

Register : operands are specified using register.

ADD B. ; MOV B, C.

Direct / Absolute : Directly specify our operand by its ~~ad~~ location.

LDA 2500H ; IN 87H.

↓ port address

Register Indirect : using register pair, we indirectly specify operand.

LDAX B.

Implicit / Implied : operand implicitly specified in instruction

CMA.

Q⇒ To m

Ans

Q⇒

Ans

Q⇒

Ans

Q⇒

Ans

LDA
LXI
SUB
STA

Q⇒ To load a number 87H into the accumulator and move it to register B.

Ans
```
MVI A, 87H.
MOV B, A.
HLT.
```

Q⇒ Load the number 37H into register B and display it on the output port whose address is 03H.

Ans
```
MVI B, 37H
MOV A, B.
OUT 03H
HLT.
```

Q⇒ Load the number 93H in accumulator and B7H in C register and add these number and add 35H to the result.

Ans
```
MVI A, 93H.
MVI C, B7H.
ADD C.
ADI 35H.
HLT
```

Q⇒ Subtract contents of memory location 2051 from the location 2050 and store result in 2052.

2052 = 2050 - 2051.

Ans
```
LDA 2050
LXI H, 2051
SUB M.
STA 2052.
```

```
LDA 2051
MOV B, A.
LDA 2050
SUB B.
STA 2052.
```

```
LDA 2051
MOV B, A.
LHLD 2050
SUB L.
STA 2052.
```

Q⇒ Add first 5 natural numbers.

Ans

```
MVI  A, 01H
ADI  02H
ADI  03H
ADI  04H
ADI  05H.
```

```
MVI B, 02H
MVI A, 01H.
ADD B
INR B.
ADD B
INR B.
ADD B.
```

```
2050  MVI A, 00H
2052  MVI B, 01H.
2054  MVI C, 05H
2056  ADD B.
2057  INR B.
2058  DCR C.
2059  JNZ 2056
```

} loop.

Q⇒ Add two number 9B and A7 and display result in o/p port. 03H. If carry occurs, display carry in output port 03H instead of result.

Ans

```
MVI A, 9BH
MOV B, A.
MVI A, A7
ADD B.
~~OUT 03 JNC~~.
~~05~~ JC label.
   OUT 03H
label MVI A, 01H.
   OUT 03H
```