[CAO]

5 mark Imp

1) component

Input unit → Take input from user.

Output unit → visualise input from user

CPU → ALU → Arithematic & logical operation
(Brain of computer system) CU → Controll operations / instruction by user.
Memory unit → store.

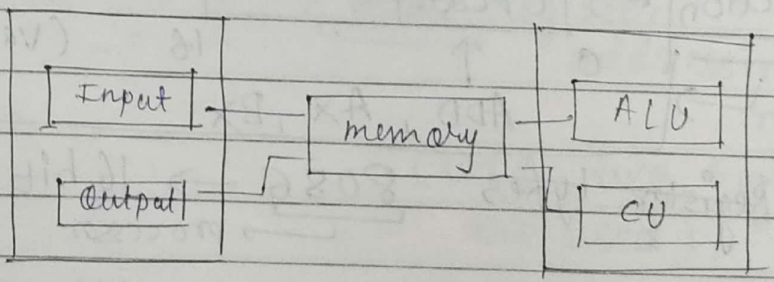
Input ~~unit~~ :- Keyboard, mouse, scanner.

Output ~~unit~~ :- Printer, Speaker, monitor.

Register is a smallest memory unit ~~of that~~.

1) Primary memory → Volatile, temporary memory,

2) Secondary memory → Non-volatile, permanent



CU :-

Architecture → tells us about flow.

ISA → (Instruction set architecture).

Addressing mode :-

Higher level language → python, C, c++

Assembly level language → (middle languages)

Machine level language → (Binary)

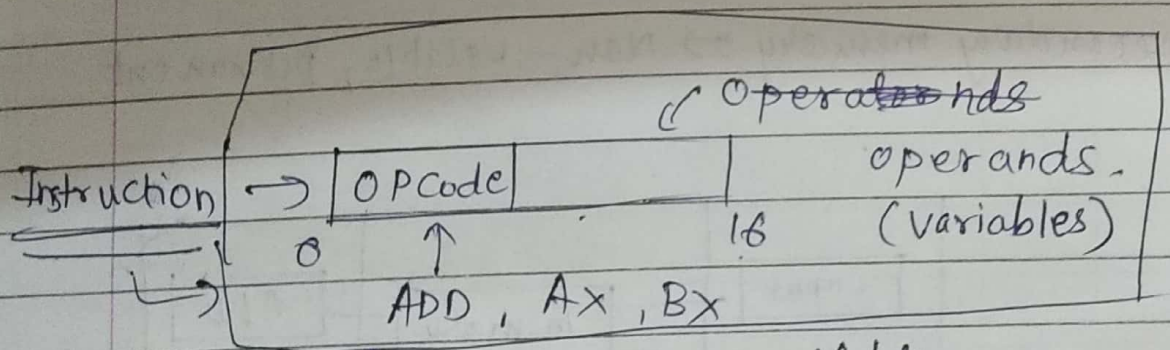
```
MOU  02, A
MOU  03, B
ADD  A, B
```

Instruction set Architecture

Instruction → command

set → Collection

Architecture → organised (structure)



~~Register types~~ 8086 → 16 bit.

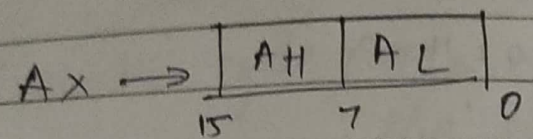
→ processor

OP code → Operation code.

Operands → Register.

Register types.

1) General purpose register
(16 bits) AX → Accumulator.
-11- BX → Base register
-11- CX → Code Register
-11- DX → Data Register.



~~Addition~~ :-

a=8	—	1000
b=4	—	100
c=a+b		1100

```
MOV   AH #08   } → Assigning values directly
MOV   AL #04   }     ↳ immediate address mode
ADD   AH, AL   → ~~Addr~~ Register addressing mode
                  (when two register are used).
```

AH

~~t imp.~~

(II) Special purpose register :-
1) Stack :- First come last out
2) Queue :- First come first out.

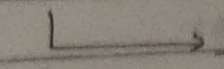
Instruction likhane ka tarika

| OP code | Mode | operand |

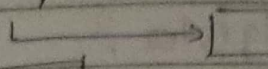
Addressing Modes:- (way to write instruction)

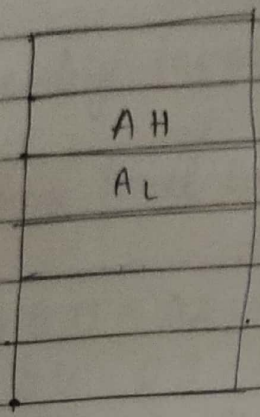
1) Immediate addressing mode :- For Directly assigning values

Eg: MOV AH, #20H



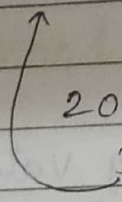
MOV AL, #10H





2) Register addressing mode :- when two registers are used. (Register to register data transfer)

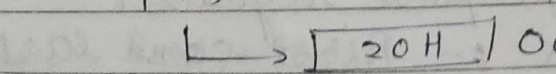
Eg:- ADD AH, AL



Value is stored in AH

3) Register indirect mode :- ~~were~~ we get the address (address of instruction) of the register and then value gets stored.

MOV AX, (R)



4) Direct addressing mode :- Location pe jakar fetch the value.

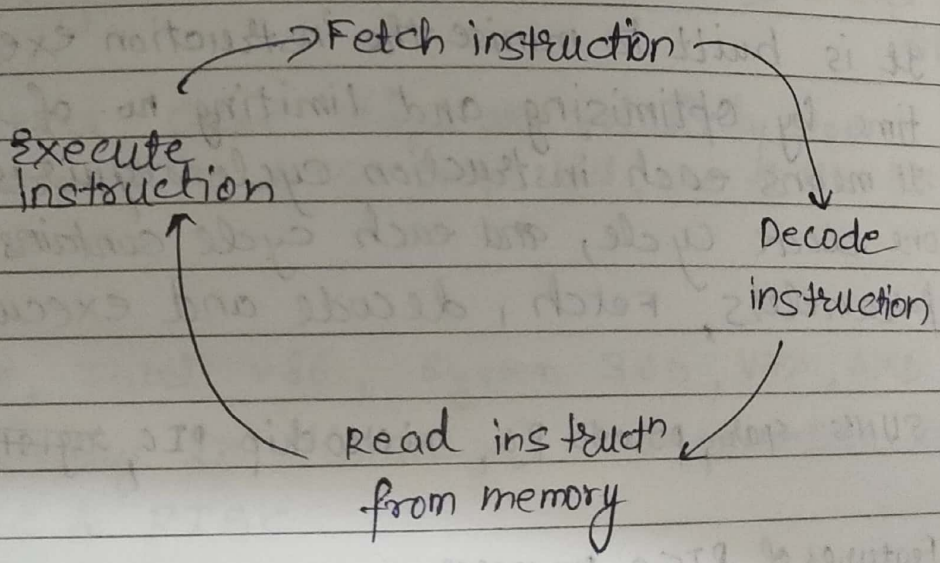
MOV AX, 2000H
Address

5) Indirect addressing mode :- Location ka location

MOV Ax, (2000H)

* Instruction cycle (This cycle continues till the user keeps giving input)

~~Fetch instruction~~



Eg :- ADD AX, BX
ADD 02, 03
010 011
101

1) CISC & RISC are processor
2) They are architecture

* Cisc and Risc Architecture.

1) Risc → Reduce instruction set Computer
&
Cisc → complex instruction set computer

Risc :-
1) ~~fuit~~ Reduce instruction set computer.
& Reduce instruction set computer, a microprocessor architecture with a simple collection of and highly customise set of instruction.

It is built to minise the instruction execution time by optimising and limiting no. of instruction. It means each instruction cycle requires only one clock cycle, and each cycle contains three parameters, Fetch, decode and execute.

eg:- SUN's spar, powe PC, microchip PIC, ~~rgtst~~, RISC-V

Features of RISC processor.

1) One cycle execution time
2) Pipelining technoiqu.
3) A large no. of registers.

(Register to Register) → Data Transfer

CISC :- 1) Complex instruction set computer, developed by the intel.

2) It has a large no. of complex instruction that range from simple to very complex and specialise in the assembly level language, which takes a long time to execute the instruction. So,
(approaches)
So, CISC reducing the of instruction on each program and ignoring the no. of cycle per instruction

It emphasises to build complex instruction directly in the hardware. (hardware is always faster than software).

However, CISC chips are relatively slower as compared to RISC chips but, use little instruction that s risc

Eg:- Intel x86, System 360, VAX, AMD

(Memory to Memory) → Data Trans-fe

* CISC & RISC

1) Data Transfer
2) Pipelining
3) Addressing modes.

(Memory to Memory)

Eg → ~~CISC~~ RISC

ADD, AX, BX	LOAD AX
	LOAD BX
	MOV AX, #02
	MOV BX, #03
	ADD AX, BX

★ 2's Complement.

1) -3 →

3 → 0011

1st Complement → 1100

2nd Complement → 1100
+1
1101

-3 → 1101

2) -7

7 = 0111

1's Complement → 1000

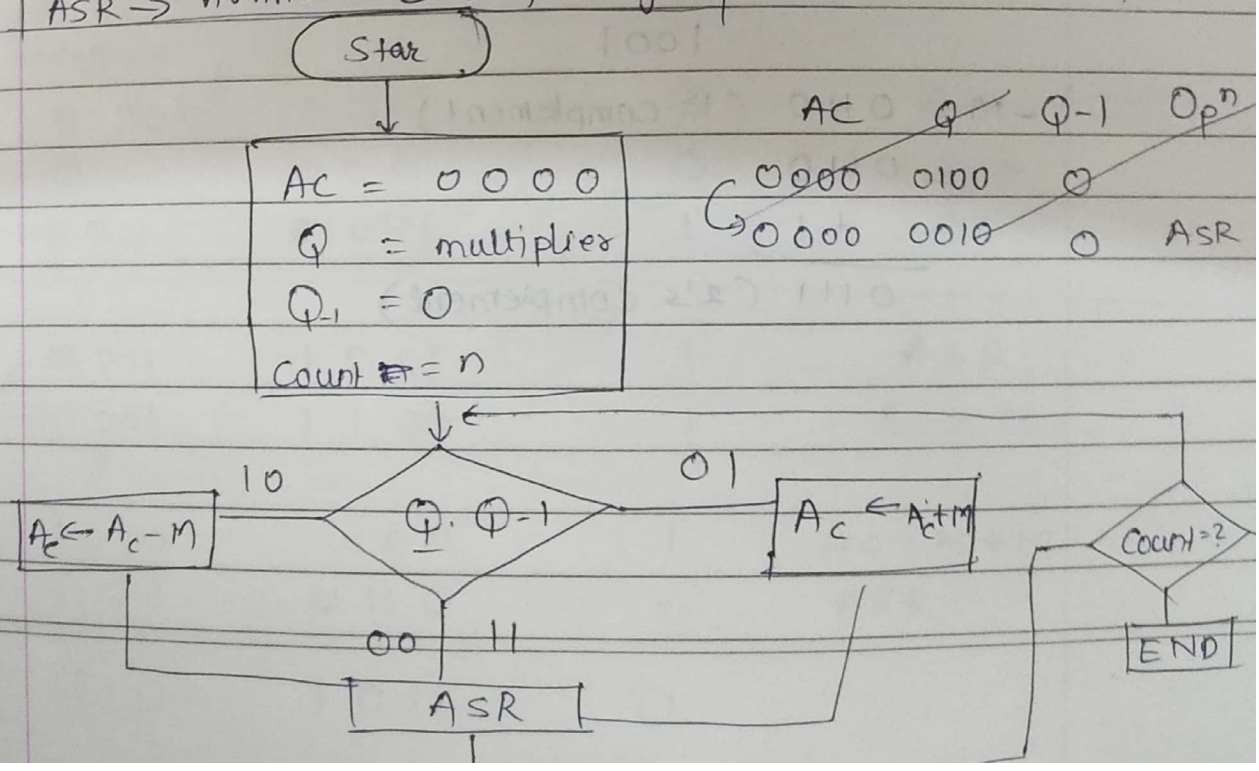
2's Complement → 1000
+1
1001

-7 → 1001

★ Booth's Algo

ASR → Arithematic shift right

−5 × 4
M Q



	AC	Q	Q-1	Opⁿ
1)	0000	0100	0	ASR
	0000	0010	0	ASR
2)	0000	0010	0	ASR
	0000	0001	0	
3)	0000	0001	0	
	0101	0001	0	AC = AC − M

Q. (−7) × 3 =

Soln → M = −7

7 = 0111

1's compliment = 1000

2's complement = 1000
+ 1
1001

−M = 0110 (1's complement)
= 0110
+ 1
0111 (2's complement)

* Do ASR after [illegible]

2nd method

	AC	Q	Q-1	Operation
1)	0000	0011	0	AC = AC - M
	1111			= 0000 + 0111
	1000	0001	1	= 0111
2)	1100	0000	1	
	1100	0000	1	
3)	1100	0000	1	AC ← AC + M
	0101	0000	1	1100
	0010	1000	0	+1001
				0101

1st method :-

	AC	Q	Q-1	Operation
~~0000~~	~~0000~~	~~0011~~	~~0~~	~~AC = AC - M~~
1)	0000	0011	0	AC = AC - M
	0111	0011	0	ASR
	0011	1001	1	~~ASR~~
2)	0011	1001	1	ASR
	0001	1100	1	~~AC = M +~~
3)	1010	1100	1	AC = AC + M
	1101	0110	0	ASR
4)	1110	1011	0	

Q (3) × (-2)
M Q

Q = -2
2 = 0010

-2 = 1101 (1st complement)
+1
1110 (2's complement)

M = 0011
-M = 1100
+1
1101

	AC	Q	Q-1	Operation
1)	0000	1110	0	ASR
	0000	0111	0	
2)	0000	0111	0	AC = AC - M
				0000 + 1101
				1101
3)	1101	0111	0	
	1110	1011	1	~~ASR~~
4)	1110	1011	1	ASR
	1111	0101	1	
4)	1111	0101	1	ASR
	1111	1010	1	

7 × 3
M Q

M = 0111	-M = 1000
	+1
	1001
AC = 0000	
Q = 0011	
Q = 0	

	AC	Q	Q-1	Operation
1)	0000	0011	0	AC = AC-M
	1001	0011	0	ASR
	1100	1001	1	
2)	1100	1001	1	ASR
	1110	0100	1	
3)	1110	0100	1	AC = AC+M
	0101	0100	1	ASR
	0010	1010	0	
4)	0010	1010	0	ASR
	0001	0101	0	

Q $6 \times (-2)$:
M Q

M = 0110

$-M = 1001$
$+1$
1010

Q = 0010
$-Q = 1101$
$+1$
1110

	AC	Q	Q_{-1}	Op^n
1)	0000	1110	0	ASR
	0000	0111	0	
2)	0000	0111	0	AC ← AC − M
	1010	0111	0	ASR
	1101	0011	1	
3)	1101	0011	1	ASR
	1110	1001	1	
4)	1111	0100	1	

* Floating point Addition

1) 0.75 + (-0.625)

sol⇒ 0000.75 (4-bit representation)

0.75 x 2 = 1.5 ⇒ 1
0.5 x 2 = 1 ⇒ 1
0 x 2 = 0 ⇒ 0
0 x 2 = 0 ⇒ 0

0000.1100

0.625 :
0.625 x 2 = 1.25 ⇒ 1
0.25 x 2 = 0.5 ⇒ 0
0.5 x 2 = 1 ⇒ 1
0 x 2 = 0 ⇒ 0

0000.625 ⇒ ~~0.1010~~ 0

-0.625 ⇒ 0000.1010 (1's compliment)

~~0.0101~~ 1111.0101 (2's compliment)

-0.625 ⇒ ~~0.1010~~ +1

1111.0110

0000.1100
+ ~~0000.1110~~
~~0.1010~~

0000.1100
+ ~~0000.0110~~
~~0000.0010~~

0000.0010

2) 0·82 + (−0·525)

Sol^{n} ⇒ 0·82 × 2 = 1·64 = 1
0·64 × 2 = 1·28 = 1
0·28 × 2 = 0·56 = 0
0·56 × 2 = 1·12 = 1

0000·1101

0·525 × 2 = 1·05 → 1
0·05 × 2 = 0·1 ⇒ 0
0·1 × 2 = 0·2 = 0
0·2 × 2 = 0·4 = 0

0000·1000

1111·0111 (1's)
+1
1111·1000 (2's)

0000·1101
+ 1111·1000
0000·0101

0000·0101

IEEE standard single & double precision format :-

IEEE standard for floating point arithematic (IEEE 7504) is a technical standard for floating point computation which was established in 1985 by institute of electrical and electronic engineers. The standard address many problem found in the diverse floating point implementations that made them difficult to use reliably and reduce the probability.

IEEE std. 7504 is the most common floating point representation for real number on computers including intel based PCs, macs and unique platforms.