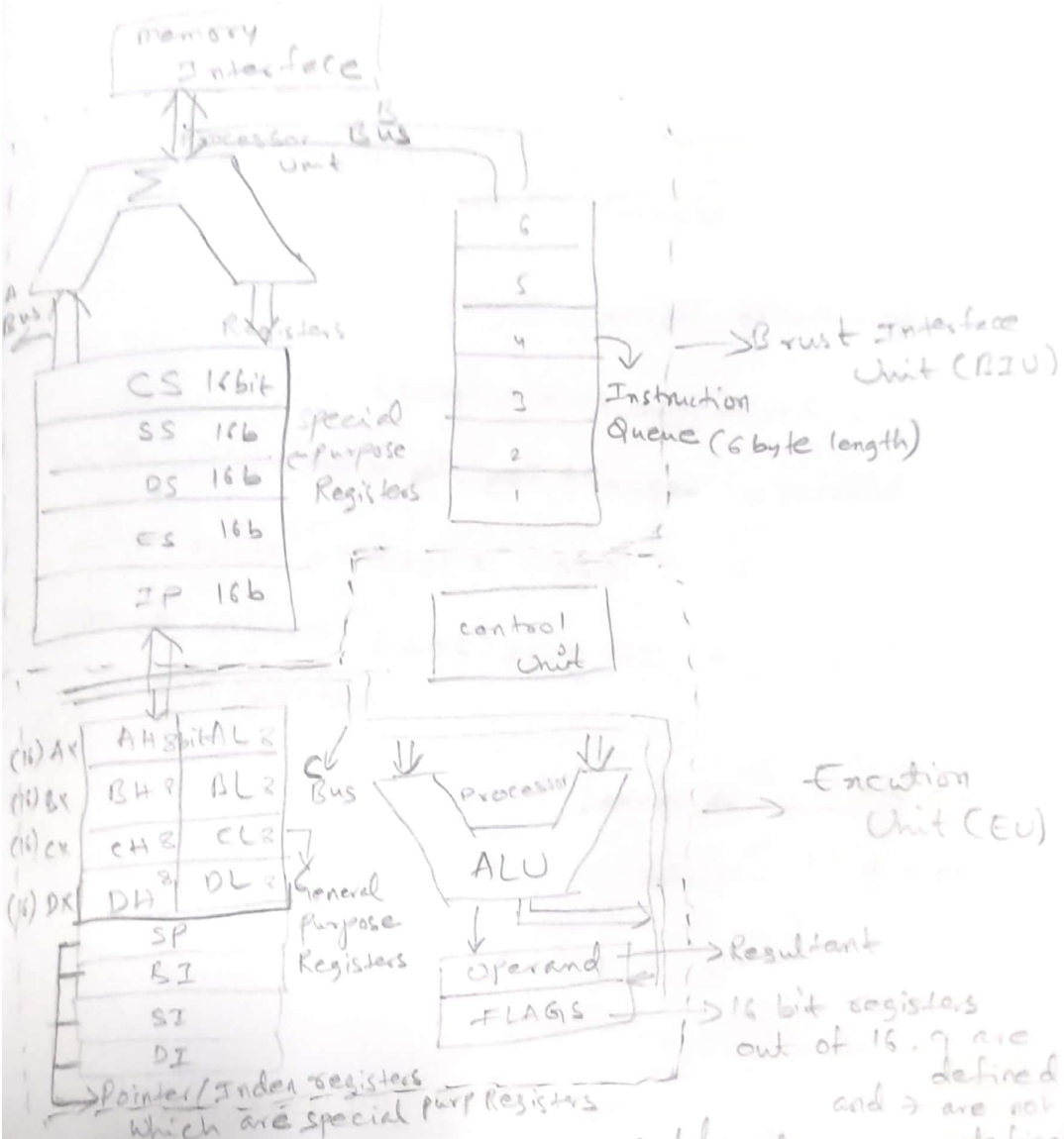
8086 Microprocessor



Processing Unit: for calculating Address

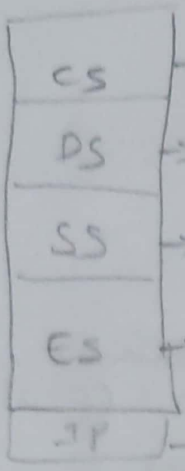
ALU: Arithmetic, Logic Unit operations.

Special purpose Registers: Hold Base Addresses/next instruction Address

Pointer / Index register

SP	→ stack pointer
BI	→ Base Index
SI	→ source Index
DI	→ Destination Index

Special Registers



CS → code segment memory

DS → Data Segment memory

SS → stack segment memory (LIFO)

ES → Extra Segment memory

IP → Instruction pointer (point)

Physical Address = Segment Register * 10H + Offset (I_P)

= 1020H * 10200H + offset value

= It gives 20 bit Address to the Address Bus

20 bit capacity ____ Address

16 bit capacity ____ Data

____ Control

Example of Addition

(Example: These are 16 bit registers)

(3) ADD AX BX

(1) move AX [2000H] (∵ [ ] → Indicates the memory)

(AX → destination, [2000H] → source)

(2) move BX [2004H]

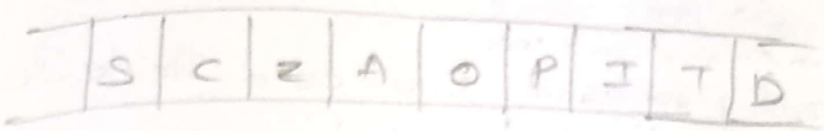
For product operation

=> 16 bit binary multiplications stores in AX and DX regi

if 8 bit binary multiplication stores in only AX regist

FLAGS Registers

⇒ 9 flag bits are



Microprocessor:

⇒ It is an Electronic component that is used by a computer to do its work.

⇒ It is a Central processing Unit on a single IC containing millions of very small components including transistor, resistor and diode that work together.

Ex: 8086 microprocessor enhanced version of 8085 microprocessor, designed by Intel in 1976.

⇒ 8086 Microprocessor:

⇒ It is 16 bit microprocessor having 20 address lines and 16 data lines that provide upto 1MB storage.

⇒ It supports two mode operations i.e Maximum and Minimum mode.

Maximum mode is suitable for system having multiple processors.

Minimum mode is suitable for system having a single processor.

Features of 8086

The most prominent features of a 8086 microprocessor are as.

* It has an Instruction Queue, which is capable of storing six instruction bytes from memory result in faster processing.

* It was 16-bit processor having 16-bit ALU, 16-bit registers, internal and External-16 bit data bus result in faster Processing.

⇒ It is available in 3 versions based on frequency of operation-

- 8086 → 5MHz
- 8086-2 → 8MHz
- 8086-1 → 10MHz

⇒ It uses two stages of pipelining i.e fetch stage and Execute stage which improves performances.

⇒ Fetch can be prefetch upto 6 byte of instructions and store them in Queue.

⇒ Execute stage Executes the Instructions.

⇒ It has 256 Vectored Interrupts

⇒ It consists of 29,000 transistors.

Architecture of 8086 Microprocessor

⇒ 8086 microprocessor divide into two functional Units i.e

i) BIU (Bus Interface Unit)

ii) EU (Execution Unit).

Bus Interface Unit (BIU):-

⇒ It take care of all data and addresses transfers on the bus for the EU.

⇒ EU has no direct connection with system buses so, this is possible with the BIU.

⇒ BIU & EU are connected with Internal bus.

It has the following functional parts.

i) Segment / special purpose Registers.

ii) Instruction Queue.

iii) processor. & iv) Instruction pointer.

Instruction Queue:

- BIU contain the Instruction Queue.
- BIU gets upto 6 bytes of next instructions and store them in instruction Queue.
- When EU Executes instruction and is ready for it's next instruction, then simply read the instruction from instruction Queue. in increased Execution speed.

=> Fetching the next instruction while the current instruction Executes is called Pipeliming.

Segment registers:

- BIU has 4 segment buses ie CS, DS, SS & ES. It hold the address of Instruction and data in memory.
- It also contain one pointer register IP, which hold the address of the next instruction to Execute by the EU.

Four segments are.

CS – Code Segment.

->Where the Executable programme is stored
->Used for addressing memory location in the code segment of the memory.

DS – Data Segment.

-> It consist of Data used by the program and is accesed in the data segment by an offset address.

SS – Stack Segment.

->handles memory to store data and addresses during Execution.

ES – Extra Segment

-> Es is a Additional Data segment.
->which is Used by the string to hold Extra Destination Data.

Instruction pointer: It is a 16-bit register Used to hold the address of the next instruction to be Executed.

Execution Unit (EU)

=> EU gives instruction to BIU stating from where to fetch the data and then decode and Execute those Instruction.

It has the following functional parts.

i> General purpose Registers
ii> Pointer / Index Registers
iii> ALU
iv> Operand & FLAGS. FILO

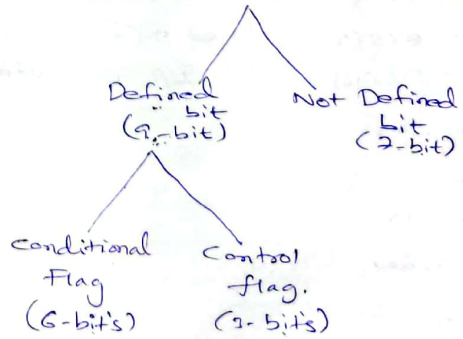
ALU:

It handles all the Arithmetic & Logical operations like +, -, *, /, OR, NOT, AND operations.

FLAG Register:

- It is a 16-bit registers that behaves like a flip-flop's. ie Changes its status according to the result stored in Accumulator.
- Finally It shows the Status of the operand.

FLAGS Are divided into (16-bit)



Conditional Flags:

=> It represent the result of the last Arithmetic or logical Instruction executed. 1/0

* carry flag → General carry generated.
* Auxiliary flag → Nibble - Nibble carry generated
* parity flag → Even no. of 1's - parity set else flag is reset
* zero flag → when zero then flag is set (1)
* Sign flag → negative sign is set else to 0
* Overflow flag → when system capacity exceeds

Control flags:

=> control flags control the operation of the EU.

* Trap flag → one ALU is Active - set In multiset mode - RESET
* Interrupt flag
* Direction flag → Data Access mode

General purpose Registers:

=> There are 8 General purp. Registers.

=> These Register can Individually store 8-bit data and can be used in pairs to store 16 bit data.

AH, AL → AX (16) Accumulator register.
BH, BL → BX (16) Base register.
CH, CL → CX (16) refered to as counter.
DH, DL → DX (16) hold I/O port address.

Pointer / Index Registers

- SP, stack pointer
- BP, Base pointer
- SI, source Index
- DI, Destination Index.

ASCII Adjacent After addition (AAA)

	ASCI	Hexadecimal.
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39

ASCII Hexadecimal representation.

Algorith for AAA

if lower nibble of AL > 9 (or) AF = 1

AL = AL + 6
AH = AH + 1
CF = 1
AF = 1

else

AF = 0
CF = 0

Ex:

```
mov AL 37H
mov BL, 35H.
ADD AL, BL
AAA
HLT
```

0011 0111
0011 0101

0110 1100 > 9
+1 0110

0000 0010
+1
0001 0010
1

```
mov AL 31H      AF=0
                0011 0001
mov BL 34H      0011 0100
                0110 0101
ADD AL, BL      0000
AAA
HLT
```

AL = AL + BL

AX = 00105 (AF = 0, CF = 0)

$= (5)_{10}$

```
mov AL 38H      AF=1
                0011 1000
mov BL 38H      0011 1000
                0111 0000
ADD AL BL          0110
AAA             0000 0110
HLT
```

AH = AH + 1

= 01

AX = 0106 $= (16)_{10}$

ASCI Adjust After Subtraction (AAS)

if lower nibble of AL > 9 or AF = 1

AL = AL − 6

AH = AH − 1

CF = 1

AF = 1

else

AF = 0

CF = 0

Clear higher nibble of AL

Ex:

```
mov AL, 38H     0011 1000
mov BL, 32H     0011 0010
                0000 0110
SUB AL, BL
AAS
HLT
```

AH = 00H, AL = 06

AX = 00 06

$= (6)_{10}$

```
mov AL, 32H     0011 0010
mov BL, 37H     0011 0111
                1111 1011 > 9
SUB AL, BL         0110
AAS             0000 0101
HLT
```

AH = 00

AAM (ASCI Adjust After multiplication)

Algorith

AH = AL / 10 = 15 / 10 = 1

AL = Remainder = 5

Ex:

```
mov AL, 5
mov BL, 3
MUL BL
AAM
HLT
```

AL = AL × BL (5 × 3)

= 15

AL = 0FH

AH = 01, AL = 05

AAD (ASCII Adjust Before Division)

Algorithm

AL = (AH x 10) + AL

AH = 00

Ex: mov AX 0205 H

AH	AL
02	05

AAD — AL = 02 x 10 + 05

HLT — AL = 25

AH = 00

AX = 0025

⇒ Four stages are called one Instruction Cycle.

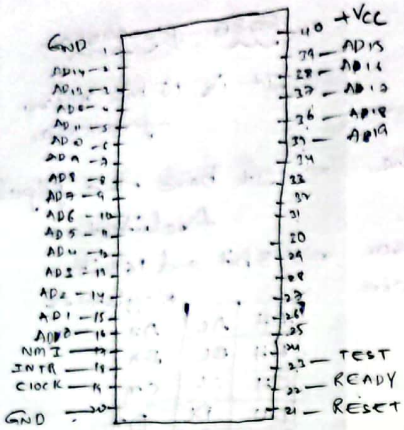
The four stages are

fetch ↓ Decode ↓ Execute ↓ WB

Comparision Between two microprocessor. ie 8085 & 8086 processors.

8085 processor	8086 Processor.
⇒ It is 8 bit microprocessor	⇒ It is 16 bit micro -processor.
⇒ It doesn't have the Pipeline	⇒ It have the pipeline Architecture
⇒ Inside the processor the All registers are 8 bit capacity	⇒ 8 bit and 16 bit Registers AH AL AX BH BL BX CH CL CX DH DL DX 8 bit 16 bit
⇒ Data Bus capacity is 8 bit Data bus 16 bit Address Bus	⇒ capacity is. 16 bit Data bus 20 bit Address bus
⇒ only one processing Unit ie one ALU	⇒ It has two processing Units ie one ALU & another one is BIU proce -ssor one – Address calculation another – Arithmetic operation

8086 processor pin Diagram



AD0 — AD15 → Combining Address and Data purpose it is used

→ This pins are for both data & Address

→ 16 pins it have.

A_0 — A_{19} → Total 20 pins

→ Two pins for Ground. (1 & 20)

NMI → The processor has to Accept the Interept

→ Non markable Interpet

INTR → Interact

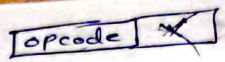
Addressing Mode:-

⇒ There are so many Addrssing mode in 8086 processor

⇒ How processor will Accept the data is Addressin mode (way of operating the operand).

1. Implicit Addressing mode.

In this only opcode part no operand field.



Ex: CMA (Complement Accumulator Register).

$AC \leftarrow \overline{AC}+1$

INCA (Increment Accumulator Register)

$AC \leftarrow AC+1$

Here there is no operand mode only opcode mode is present

CMA

INCA

2. Immediate Addressing Mode:-

⇒ Assigining value to the memory.

- Address point has Value

Ex: ADD 10 — $AC \leftarrow AC+10$ (AC has value 10)

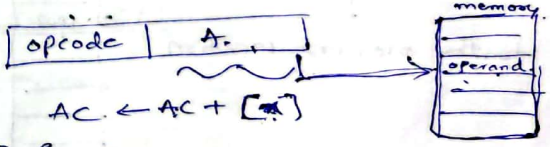
MOV R_1, 15 — $R_1 \leftarrow 15$ (R_1 has value 15)

⇒ operand Address field has Value/Data are called Immediate Addressing mode.

Ex: MUL 15 — $AC \leftarrow AC*15$

3. Direct Addressing Mode.

⇒ From memory location operand Access the Data.

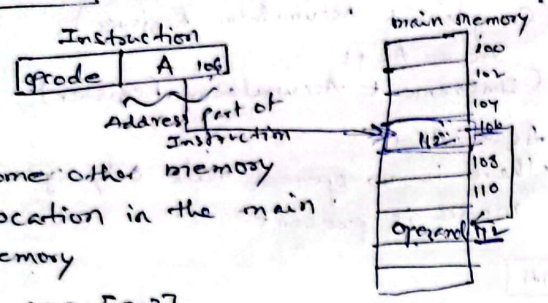


$AC \leftarrow AC + [\ ]$

Ex: ADD R_1, [2004H]

$R_1 \leftarrow R_1 + 2004H$

Indirect Addressing Mode



Ex: ADD [[A]]

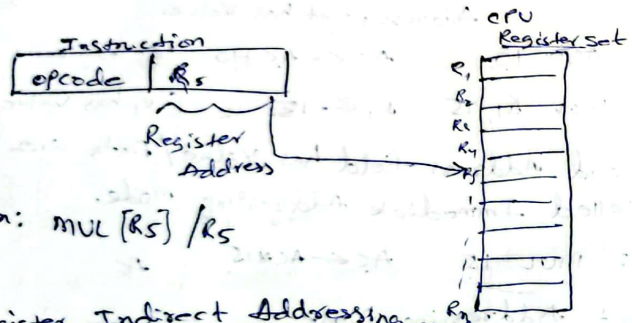
Ac ← Ac + [[A]].

Ac ← Ac + D.

⇒ pointer to the memory location.

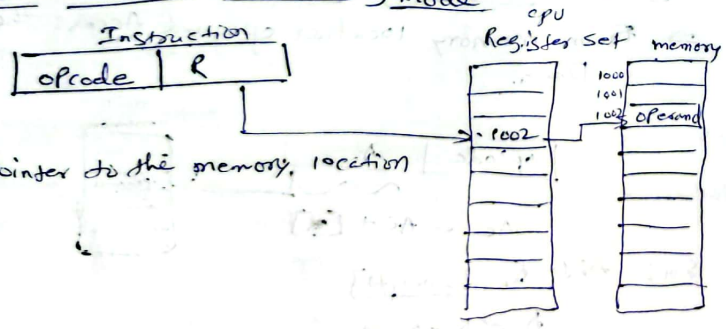
⇒

Register Direct Addressing mode / Register Addressing mode



Ex: MUL [R5] / R5

Register Indirect Addressing mode



⇒ pointer to the memory location

Element of Machine Instruction :

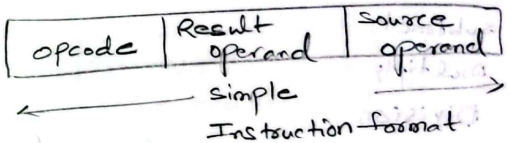
The Elements of Machines are.

⇒ operation code / opcode: specifies the operation to be performed.

⇒ Source operand reference: It involves one or more source operands.
i/e that are input for the operation.

⇒ Result / Destination operand refe : It produce a result.

⇒ Next Instruction refence: It tells the Processor, where to fetch the next instruction, After completion of Instruction.



Source and Result operands can be in one of four Areas.

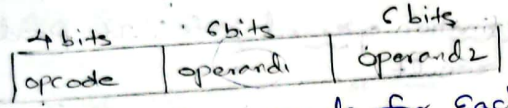
i> main / virtual memory
ii> processor register.
iii> Immediate I/p.
iv> I/o Devices.

Instruction Representation

⇒ It is Assume that it is a 16-bit cpu.

4 bits are used to provide the operation code. So, we may have 16 ($2^4 = 16$) different set of Instruction.

There are two operands. To specify each operand 6 bits are used.



64 different operands for each operand.

Opcodes are represented by abbrevations called mnemonics. indicate the operations.

ADD → Add
SUB → subtract
MUL → multiply
DIV → Division
LOAD → memory to cpu loads the data
STORE → cpu to memory to store data

Instruction Types

The Instruction set of cpu can be categorized

i> Data processing → ALU Instructions.
ii> Data Storage → moving data b/w memory & cpu registers.
iii> Data movement → I/O movements.
iv> control.

Number of Address

⇒ It defines the no, of operand parts in an Instruction.

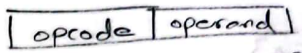
⇒ It is either Unary (one operand) or binary (two operand)

Four common Instruction / Address formats are

i> Zero-Address Instruction

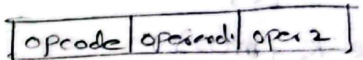
Here no, operand part present so it is zero Address Instruction

ii> one-Address Instruction



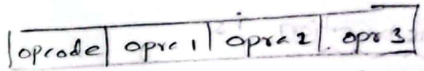
Here only one operand part is present so, it is one-Address Instruction

iii> Two-Address Instruction.



Here two operand parts are present

iv> Three-Address Instruction



Here three operand parts are present.

ADD, SUB, MUL, Div } These are for Arithmetic operations.

MOV — transfer type operation.

LOAD, STORE } Transfer to and from memory & AC registers.

$X = (A+B) * (C+D)$ Represent By Using Different Address Instruction.

Three Address Instruction is.

```
ADD R1, A, B.   ; R1 ← M[A] + M[B]
ADD R2, C, D    ; R2 ← M[C] + M[D]
MUL X, R2       ; M[X] ← R1 * R2 ;
```

But we can't use Three-Address Instruction in 8086 micro-processor.

Two Address Instruction is

```
MOV R1, A     R1 ← M[A]
ADD R1, B.    R1 ← R1 + M[B]
MOV R2, C     R2 ← M[C]
ADD R2, D.    R2 ← R2 + M[D]
MUL R1, R2    R1 ← R1 * R2
MOV X, R1     M[X] ← R1
```

One-Address Instruction is

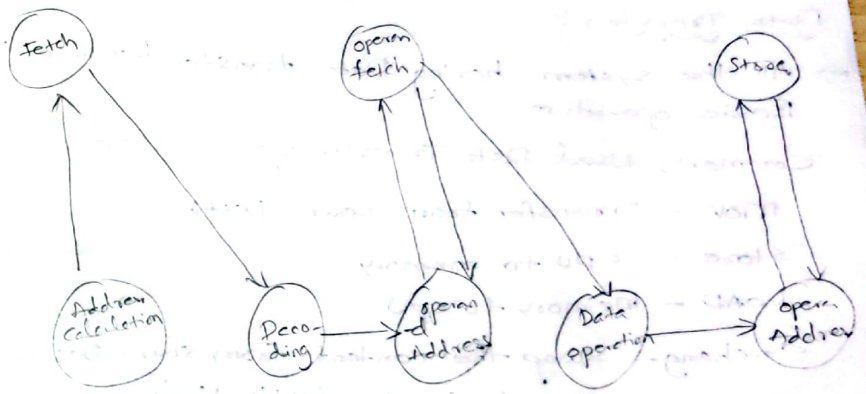
```
LOAD A      Ac ← M[A]
ADD  B      Ac ← Ac[C] + M[B]
STORE T     M[T] ← Ac
LOAD C      Ac ← M[C]
ADD  D      Ac ← Ac + M[D]
MUL  T      Ac ← Ac * M[T]
STORE X     M[X] ← Ac
```

All operations are done between Ac register & memory operands T is the Address

Zero-Address Instruction is

```
PUSH A      TOS ← A      TOS → top of stack.
PUSH B      TOS ← B
ADD         TOS ← (A+B)
PUSH C      TOS ← C
PUSH D      TOS ← D
ADD         TOS ← (C+D)
MUL         TOS ← (C+D) * (A+B)
POP X       M[X] ← TOS.
```

Instruction cycle / Element of machine Instruction



Instruction cycle State Diagram

Types of operations / Instruction types

⇒ The number of different opcodes and their types widely from machine to machine.

⇒ Some General types of operations are

1. Data Transfer
2. Arithmetic
2. Logical
3. conversion
5. Input output (I/O)
6. system control
7. Transfer control

Data Transfer :-

⇒ All the system having Data transfer, because it is Basic operation.

Commonly Used Data Transfer operations are

- MOV. – Transfer from sour – Destin
- Store – CPU to memory
- LOAD – Memory to CPU
- Exchang – Swap the content from sour – Destin
- Clear – reset (0 to destination)
- Set – 1 to Destination
- Push – Source to top of stack
- POP – top of stack to Destination.

Arithmetic

- Addition
- Subtraction
- Multiplication
- Division
- Negate – change sign of operand
- Increment
- Decrement

Logical

- AND
- OR
- NOT
- Exclusive OR
- Test
- Compare
- Shift
- Rotate.

monday

CONVERSION

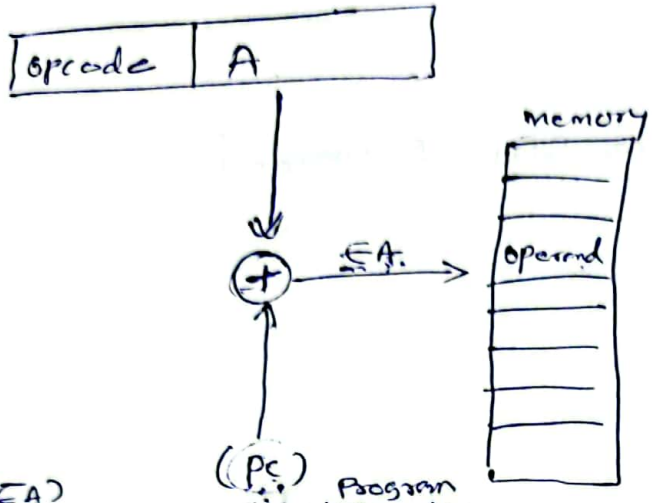
Ex: Convert number from Decimal to Binary.

I/O

⇒ Transfer the Data to the specified Instruction

Addressing Mode's continuation

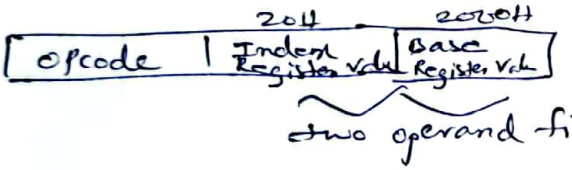
⇒ Relative Addressing Mode



(EA) Effective Address = Address part of Instruction + (PC)

Base Index Addressing mode

MOV AX 10
MOV EA EA



Ex: MOV R_1, (R_1 + R_2) / mov R_1, R_2

R_1 → Base Register value
R_2 → Index Register value

R_1 → Base
R_2 → Index

EA = 2000 + 20

SI → DS
DI → ES

Source Index (SI) store the Data segment
Destination In (DI) Store the Extra Segment.

Auto Increment & Auto Decrement Addressing mode

Ex: ADD R_1, [R_2]++
$R_2 \leftarrow R_2 + 1$ then
$R_1 \leftarrow R_1 + R_2$
Increment

ADD [R_1]--, R_2
$R_1 \leftarrow R_1 - 1$;
Decrement

Unit-II

03/04/2023
Monday.

Data Representations

Sign magnitude
1's complement
2's complement
} Representation of signed numbers in 3 ways.

Sign Magnitude

0 000 + 0
Sign | Magni-tude

0 001 + 1
.
.
.
0 111 +7

1 000 -0
1 000 -1
.
.
.
1 111 -7

1's complement

0000 +0
0001 +1
.
.
.
0111 +7

1111 -0
1110 -1
1000 -7

2's complement

0000 +0
0001 +1
0010 +2
.
.
.
00111 +7

1111 -1
1110 -2
1101 -3
1100 -4
1011 -5
1010 -6
1001 -7
1000 -8

Range: $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$ 1's complement Range.

Range: $-(2^{n-1})$ to $+(2^{n-1}-1)$ 2's com

for Ex $n=4$ bits
-8 to $+7$ is the range

Fractional numbered representation

i> Fixed point representation
ii> Floating point representation

Ex: 532.125 Decimal point :

Before fraction — After fraction.

for 16 bit

S	BF	AF
1	7	8

Converting to Binary format.

BF

2 | 532
2 | 266 — 0
2 | 133 — 0
2 | 66 — 1
2 | 33 — 0
2 | 16 — 1
2 | 8 — 0
2 | 4 — 0
2 | 2 — 0
1 — 0

AF

2 | 125
2 | 62 — 1
2 | 31 — 0
2 | 15 — 1
2 | 7 — 1
2 | 3 — 1
1 — 1

0.125 × 2 = 0.250 — 0
0.250 × 2 = 0.500 — 0
0.5 × 2 = 1.0 = 1

0 0 1
2^{-1} 2^{-2} 2^{-3}
$\frac{1}{2}$ + $\frac{1}{4}$ + $\frac{1}{8}$

$(532.125)_{10} = (1000010100.1011111)_2$

0.125 × 2 = 0.250 — 0
0.250 × 2 = 0.500 — 0
0.500 × 2 = 1.000 — 1

$(532.125)_{10} = (1000010100.001)_2$

$(101)_{10} \rightarrow (01100101)_2 \rightarrow (145)_8 \rightarrow (65)_{16}$.

Floating Point representation

IEEE 754 format.
Institute of Electrical and Electronic Engineering
two way representation
i> singl precision Representation.
ii> Double precision "

⇒ In a 32-bit format for single precision representation Divide into 3 types.

S	Exponent	Mantissa
1 bits	8 bits	23 bits

Sign bit → 0 → positive.
1 → Negative

Exponent → Bias Exponent (only positive Values)
Real mean (both +ve & -ve)
→ It is only Bias Exponent.

Excess = $2^{k-1} - 1$ (∵ k = capacity of Exponent) :
$= 2^{8-1} - 1$
$= 2^7 - 1 = 127$

Now Biased Exponent = Actual Exponent / Real Exponent + Excess 127

Actual Exponent = Bias Exponent − Excess 127.

Here bias Exponent is $2^8 - 256$
Bias Exponent = 0 to 255.
After Excluding LSB & MSB i.e 0 and 255
i.e Range is 1 to 254

Real Exponent / Actual Exponent Range =

Bias Exponent − Excess 127

= 1 − 127 to 254 − 127

= −126 to +127 Range of Real Exponent in single precision Representation

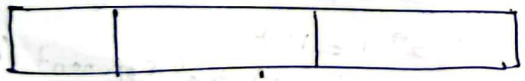
Actual Expo = −126 to +127

Procedure to represent single precision

1. Convert fractional into Binary
2. Normalize the fractional number.

   i.e $(-1)^S * (1.M) * \text{Base}^{\text{Exponent}}$

   Here S = sign
   M = mantisa
   B = Base.
   it is Real exponent
3. Bias exponent = Real exponent + Excess 127
4. In single precision (32 bit format) write number



Ex: 21.125

Binary format = 10101.001

Real exponent = 1.0101001×2^{0} (4 times)

$= 1.0101001 \times 2^{0+4}$

$= 1.0101001 \times 2^{4}$ 1-M

Biased Exponent = 4 + 127 = 131

$(131)_{10} = (10000011)_2$

0	10000011	01010010000000000000000

IEEE 754 Double precision representation

Procedure

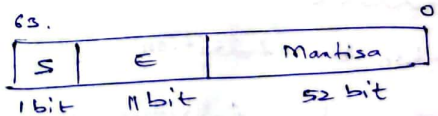
i> Fractional number convert into Binary format

ii> Normalize the fractational number.

$(-1)^S * (1.M) * B^{E}$

iii> Bias Exponent = Real Exponent + Excess 127.

iv> In double precision (64 bit format) write number.



Bias Exponent Range = 0 to $2^{11} - 1$

$= 0$ to 2047

$= 0$ to 2046

$= 1$ to 2046

Excess = $2^{K-1} - 1$

$= 2^{11} - 1$

$= 1023$.

Real Exponent = Bias Exponent - Excess 127

= 1 to 2046 $-_1$ (1023)

= 1 - 1023 to 2046 - 1023

= -1022 to 1023.

Ex: -21.125

Convert into Binary format

$\Rightarrow 10101.001 \times 2^0$

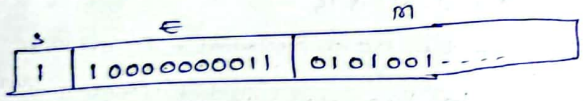
$\Rightarrow 1.0101001 \times 2^4$ converted into Normalization

i.e 1.M

∴ Bias exponent = 4+1023 = 1027

S | 10000000011 | 0101001 0 0 0 0

C 0 3 5 2

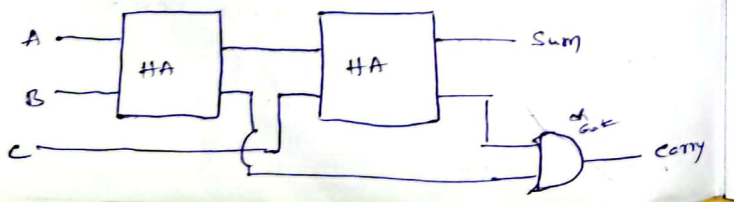


Full-Adders & Half-Adders

Half-Adder — two bit Addition

Full-Adder — Three bit Addition

Two Half-Adders is Equal to Full Adder.



Ripple Carry Adder & Carry look Ahead

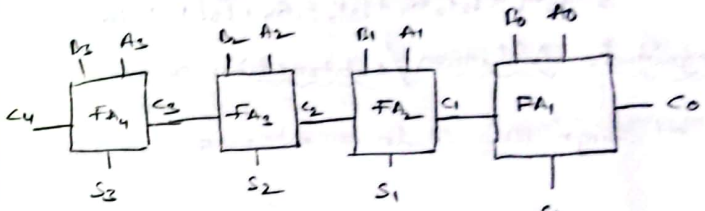
Ripple carry Adder

C_4 C_3 C_2 C_1 C_0 C_0 is initial carry i/p.

A_3 A_2 A_1 A_0

B_3 B_2 B_1 B_0

S_3 S_2 S_1 S_0



Drawback: Each Full Adder depends on the previous number i.e carry

Carry look Ahead Adder

$G_i = a_i . b_i$ ⇒ Generator i.e carry for H.A

$P_i = a_i \oplus b_i$ ⇒ parity i.e Sum for HA.

Full Adder carry is

$$C_{i+1} = G_i + P_i C_i$$

i=0 then $C_1 = G_0 + P_0 C_0$

i=1 then $C_2 = G_1 + P_1 C_1$

$= G_1 + P_1 (G_0 + P_0 C_0)$

$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$

$i=2 \Rightarrow$ then

$$C_2 = G_2 + P_2 C_2$$
$$= G_2 + P_2 C_2$$
$$= G_2 + P_2 (G_1 + P_1 G_0 + P_0 P_1 C_0)$$
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_0 P_1 P_2 C_0$$

$i=3 \Rightarrow$ then

$$C_4 = G_2 + P_3 C_2$$
$$= G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_0 P_1 P_2 C_0)$$
$$C_4 = G_3 + P_3 G_2 + P_2 P_2 G_1 + P_3 P_2 P_1 G_0 + P_0 P_1 P_2 P_3 C_0$$

Sign Magnitude Addition/subtraction operation.

Example of sign magnitude number is

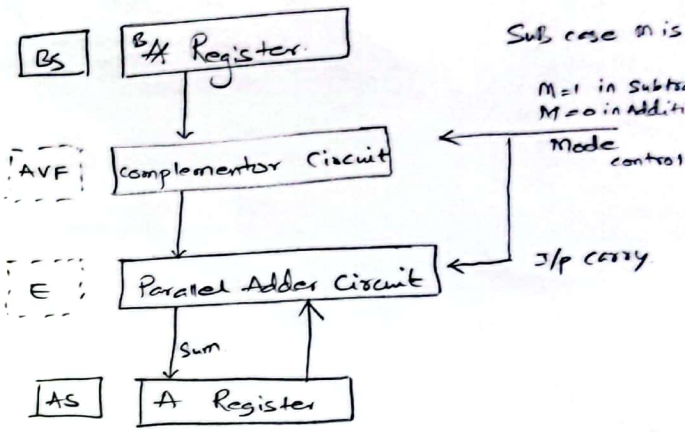
4 bit

1 0 0 1

sign magnitude

Magnitude Addition operation	Magnitude Subtraction operation.		
	A>B	A<B	A=B
(+A) + (+B) → +(A+B)			
(+A) + (−B)	+(A−B)	−(B−A)	+(A−B)
(−A) + (B)	−(A−B)	+(B−A)	+(A−B)
(−A) + (−B) → −(A+B)			
(+A) − (−B)	+(A−B)	−(B−A)	+(A−B)

Hardware Implementation.



if M=0 circuit is not Activated

Sub case M is set to 1

M=1 in Subtraction.
M=0 in Addition

Mode control

I/p carry

As & Bs Store the sign bits
Magnitudes are stored in A and B Register.

subtraction operation ⇒

$$A-B = A + \bar{B} + 1$$
$$= A + 2\text{'s complement of } B$$

AVF flip-flop set to 1 only for Addition operation if the result is more than capacity of the Register.

In Subtraction no overflow occur so AVF set to 0.

E → carry out