6/5/22

Computer Arithmetic

Topics to be covered:

Introduction

Addition and Subtraction

Multiplication Algorithm

Division Algorithm

Floating point Arithmetic operations

Addition:

Add magnitudes | Subtract magnitudes

$(+A) + (+B) \rightarrow +(A+B)$

$(+A) + (-B) \rightarrow +(A-B)$ if $A > B$; $-(B-A)$ if $A < B$; $+(A-B)$ if $A = B$

$(-A) + (+B) \rightarrow -(A-B)$ if $A > B$; $+(B-A)$ if $A < B$; $+(A-B)$ if $A = B$

$(-A) + (-B) \rightarrow +(A+B)$

Subtraction:

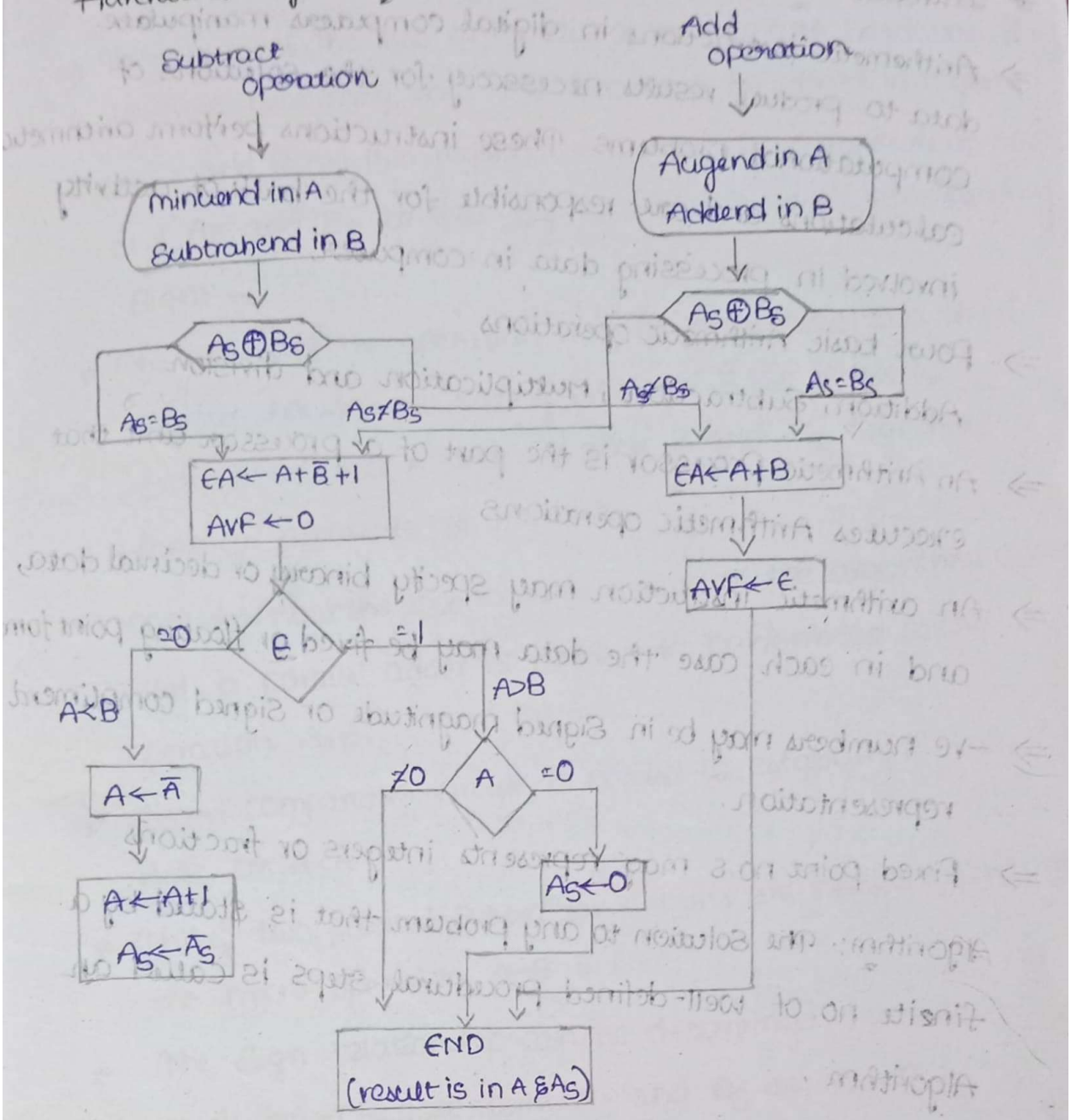
$(+A) - (+B) \rightarrow +(A-B)$ if $A > B$; $-(B-A)$ if $A < B$; $+(A-B)$ if $A = B$

$(+A) - (-B) \rightarrow +(A+B)$

$(-A) - (+B) \rightarrow -(A+B)$

$(-A) - (-B) \rightarrow -(A-B)$ if $A > B$; $+(B-A)$ if $A < B$; $+(A-B)$ if $A = B$

Hardware algorithm:



As	A	Bs	B
0	111	0	011

A = 7
B = 3

EA ← A + B

```
  111
  011      E = 1
-------
1 | 010
```

* Introduction

⇒ Arithmetic instructions in digital computers manipulate data to produce results necessary for the solutions of computational problems. These instructions perform arithmetic calculations and are responsible for the bulk of activity involved in processing data in computer

⇒ Four basic Arithmetic operations
Addition, Subtraction, Multiplication and division

⇒ An Arithmetic Processor is the part of a processor unit that executes Arithmetic operations

⇒ An arithmetic instruction may specify binary or decimal data, and in each case the data may be fixed or floating point form

⇒ -ve numbers may be in Signed magnitude or Signed compliment representation.

⇒ Fixed point no.s may represent integers or fractions

Algorithm: The solution to any problem that is stated by a finite no. of well-defined procedural steps is called an Algorithm.

* Addition & Subtraction:

It is represented in Signed magnitude and again data represented in Signed-2's compliment

It is The adopted representation for -ve numbers refers to the representation of no.s in the register before and after the execution of the arithmetic operations

Hardware implementation:

⇒ To implement the two arithmetic operations with hardware, it is first necessary that the two no.s be stored in registers

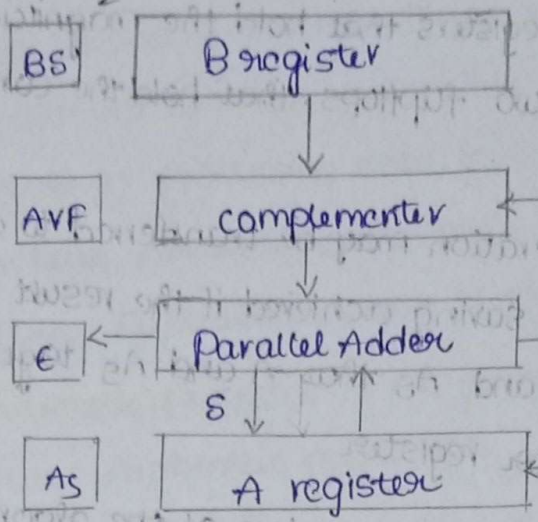
* Let A and B are two registers that hold the magnitude of no.s and, A_S and B_S be two flipflops that hold the corresponding signs.
* The results of the operation may be transferred to a third register however, a saving achieved if the result is transferred into A and A_S thus A and A_S together form an accumulator register.

⇒ Consider now the H/w implementation of the algorithms above

* First, a parallel adder is needed to perform the micro operation A+B.
* Second comparator circuit is needed to establish if A>B, A=B, or A<B.
* Third two parallel subtractor circuits are needed to perform the micro operation A-B & B-A
* The sign relationship can be determined from an exclusive-OR gate with A_S and B_S as inputs.

Signed magnitude Addition and Subtraction:

Hardware required for Signed magnitude addition and Subtraction:



As and Bs are used to store Signed bits.

$A - B = A + \bar{B} + 1$

The output carry is trasferred to flip-flop E.

The complementor consists of exclusive-OR gates and the parallel adder consists of full adder circuits.

Examples:

1. A = 7, B = 3

Addition:

As 0 0111 A

Bs 0 0011 B

$$\begin{array}{r} 0111 \\ 0011 \\ \hline 1010 \end{array}$$

$A_s \oplus B_s = 0 \oplus 0 = 0$

$EA \leftarrow A + B$

As	A
0	1010

2. A = −7, B = 6

As 1 0111 A

0110

Bs 0 ~~0011~~ B

$$\begin{array}{r} 0111 \\ 1001 \\ \hline 10001 \end{array}$$

$A_s \oplus B_s = 1 \oplus 0 = 1$

$EA \leftarrow A + \bar{B} + 1$

$AVF \leftarrow 0$

$E = 1 \Rightarrow Y$

$A \neq 0$

$10001 = -1$

3. A=-7, B=7

A_S 1 0111 A

B_S 0 0111 B

$A_S \oplus B_S = 1 \oplus 0 = 1$

$EA \leftarrow A + \bar{B} + 1$

$AVF \leftarrow 0$

```
 0111
 1000
    1
-------
1 0000
```

$E = 1 \Rightarrow Y$

$A = 0$

$A_S \leftarrow 0$

Ans: 0000

4. A=8, B=-2

A_S 0 1000 A

B_S 1 0010 B

$A_S \oplus B_S = 0 \oplus 1 = 1$

$EA \leftarrow A + \bar{B} + 1$

```
 1000
 1101
    1
-------
1 00110
```

$E = 1 \Rightarrow Y$

$A \neq 0$

End

Signed bit ↑

$001110 = 6$

5. A=9, B=4 Subtraction

A_S 0 1001

B_S 0 0100

$A_S \oplus B_S = 0 \oplus 0 = 0$

$EA \leftarrow A + \bar{B} + 1$

$AVF \leftarrow 0$

```
 1001
 1010
    1
-------
1 0101
```

$E = 1 \Rightarrow Y$

$A \neq 0$

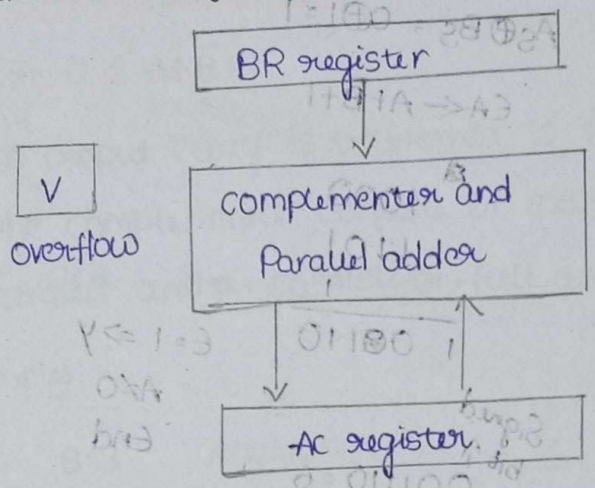
$0\ 0101 = 5$

Signed Magnitude multiplication:

```
  10111  → B(multiplicand)
 X10011  → Q (multiplier)
---------
  10111
 10111X
 00000XX
 00000XXX
 10111XXXX
---------
 1101101 01   Product
```

Addition and subtraction with Signed 2's compliment:

Hardware for Signed-2's complement addition & Subtraction:

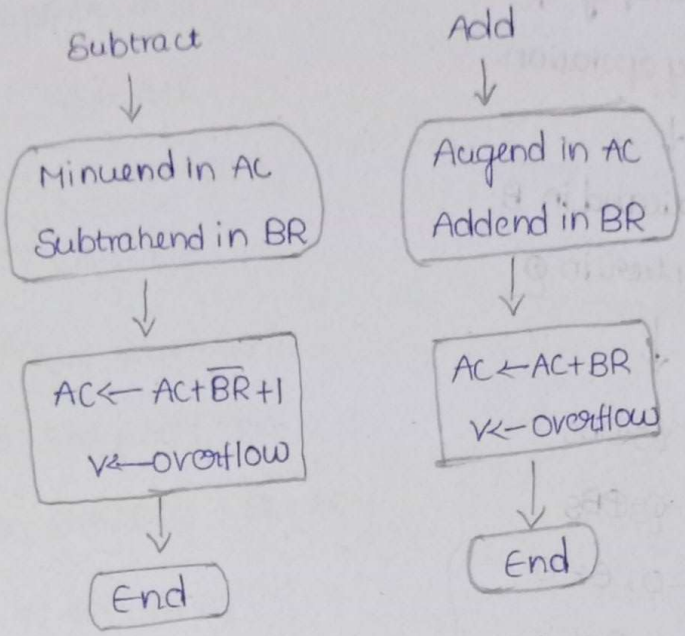


The left most bit in AC and BR represents the sign of the numbers

The overflow flipflops, V is set to 1 if there is an overflow

The output carry in this case is discarded

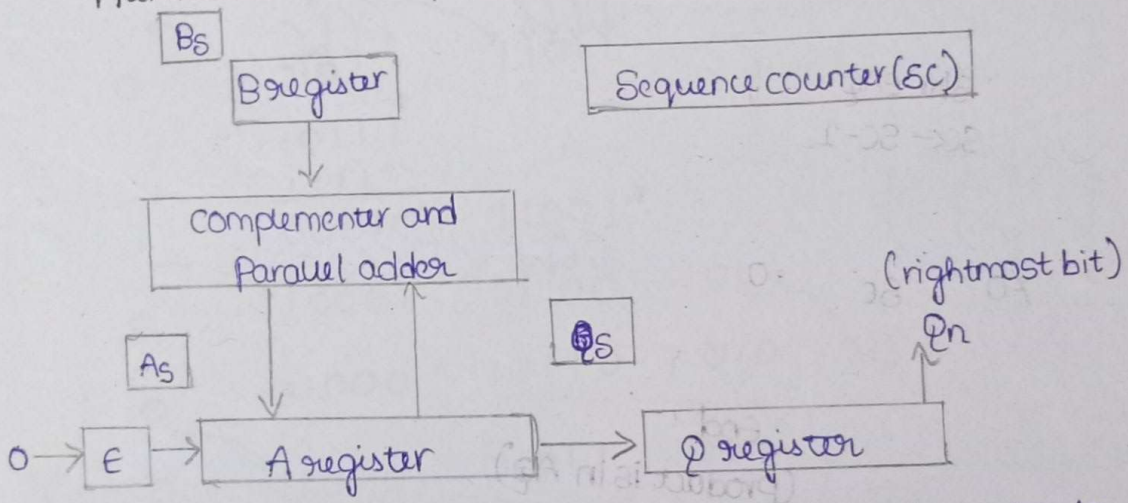
Algorithm for adding & Subtracting no.s in Signed-2's complement representation:



The Sum is obtained by adding the contents of AC and BR (including their sign bits)

The overflow bit V is set to 1 if the ex-OR of the last two carries is 1, and it is cleared to zero otherwise.

Hardware for multiply operation:

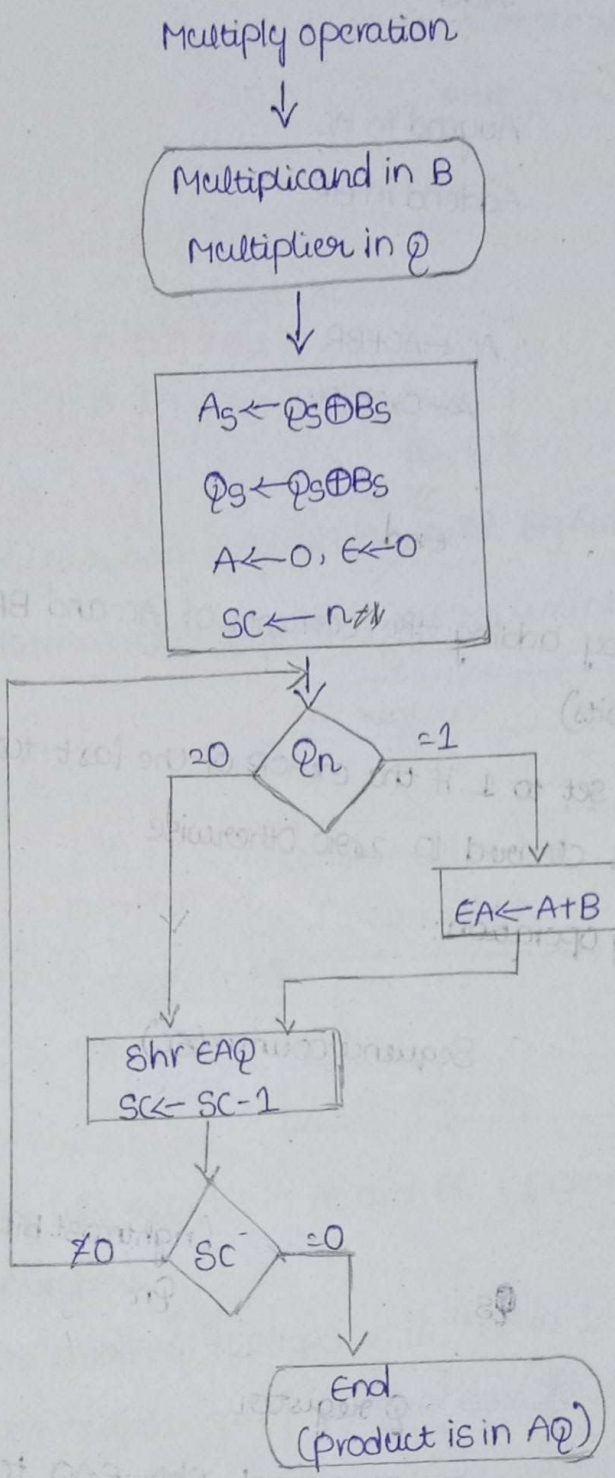


The shift will be denoted by the statement shr EAQ to designate the right shift depicted.

The least significant bit of A is shifted into the most significant position of Q

Hardware algorithm:

Flowchart for multiply operation:



Multiplicand B = 10111

Multiplier in Q

$Q_n = 1$; EA ← A+B;

Shr EAQ; SC ← SC-1;

$Q_n = 1$; EA ← A+B;

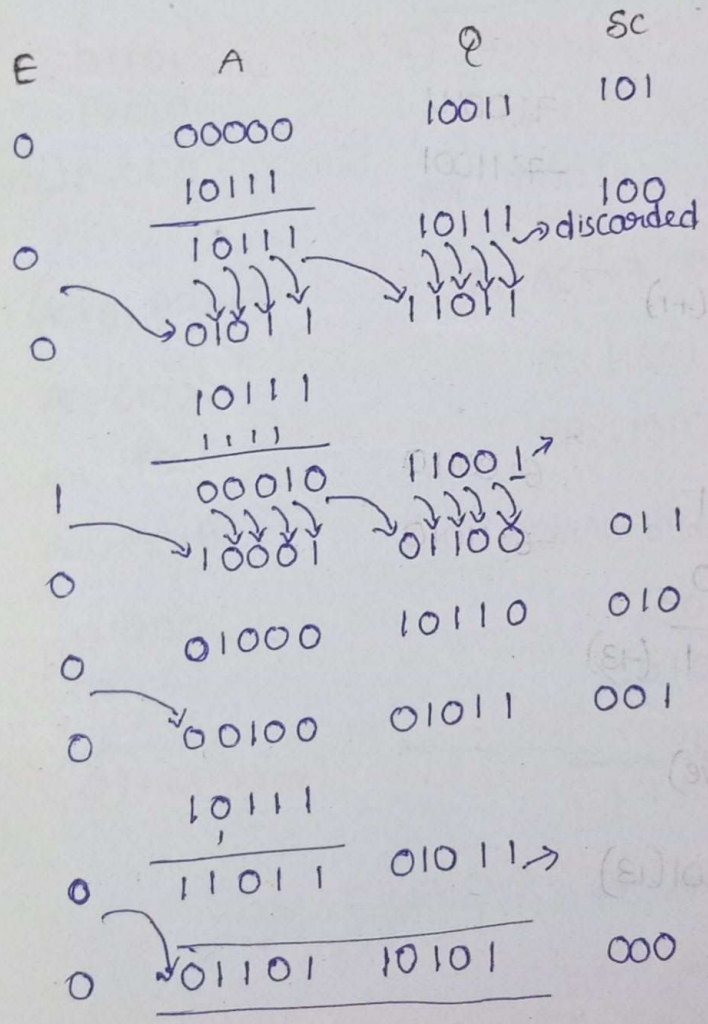
Shr EAQ; SC ← SC-1;

$Q_n = 0$; Shr EAQ; SC ← SC-1

$Q_n = 0$; Shr EAQ; SC ← SC-1

$Q_n = 1$; EA ← A+B;

Shr EAQ; SC ← SC-1



Examples:

Two's complement Addition & Subtraction:

1. A=7, B=4

AC = 7 - 00111

BR = 4 - 00100

AC+BR = 01011 (+11)

2. A=-3, B=5

AC = -3 = 1101 (3 = 0011, -3 = 1101)

BR = 5 = 0101

AC+BR = 1 0010 (2)

V=1 discarded

3. A=8, B=-7

AC = 8 - 01000 (7: 00111, -7: 11001)

BR = 7 - 11001

AC+BR = 1 00001 (+1)

4. A=-7 B=-6

AC = -7 = 11001 (6: 00110, -6: 11010)

BR = -6 = 11010

AC+BR = 1 10011 (-13)

↓

MSB(-ve)

2's of -13: 01101 (13)

5. AC = 7, BR = 4

AC : 00111

BR : 00100

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

```
  00111
  11011
  -----
1 00011  (+3)
```

6. AC = -3, BR = 5

AC : 1101

BR : 0101

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

```
  01101
  11010
  -----
1 01000
```

```
  1101
  1010
  ----
1 1000  (-8)
```

1's : 0111

2's : 1000 (8)

7. AC = 8, BR = -7

AC : 01000

BR : 11001

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

```
  01000
  00110
    1
  -----
  01111  (+15)
```

8. AC = -7, BR = -6

AC : 11001

BR : 11010

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

```
  11001
  00101
    1
  -----
  11111  (-1)
```

1's : 00000

2's : 00001 (+1)

Signed magnitude multiplication:

Hardcore:

Ex:- (+4) × (+3)

B Q

B:4 Multiplicand

Q:3 Multiplier

Bs 0 0100 B

Qs 0 0011 Q

Qn = 1

EA ← A+B

0000
(+) 0100
0100

E: Carry

Shr EAQ, SC ← SC-1

EA ← A+B 0010
0100
0110

Shr EAQ

SC ← SC-1

E	A	Q	SC	
0	0000	0011 (Qn)	4	
0	0100	0011 (Qn)	3	Qn=0 Shr EAQ SC←SC-1
	0010	0001 (Qn)		
0	0110	0001	2	
	0011	0000 (Qn)		
0	0001	1000	1	
0	0000	1100	0	

Ans: 0 0000 1100 (+12)

* (-5) × (+2)

B = -5 1 0101

B = 2 0 0010

Qn=0 Shr EAQ SC ← SC-1

Qn=1 EA ← A+B

0000
0101
0101

Shr EAQ SC ← SC-1

E	A	Q	SC
0	0000	0010	4
0	0000	0001	3
0	0101	0001	2
	0010	1000	
0	0001	0100	1
0	0000	1010	0

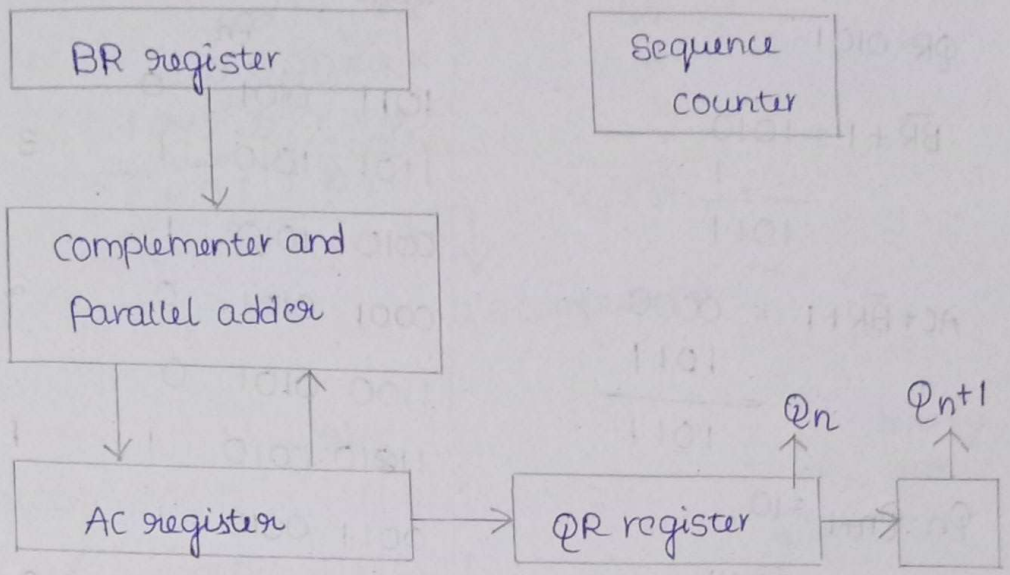
Ans :- 10000 1010 (-10)

$Q_n = 0$

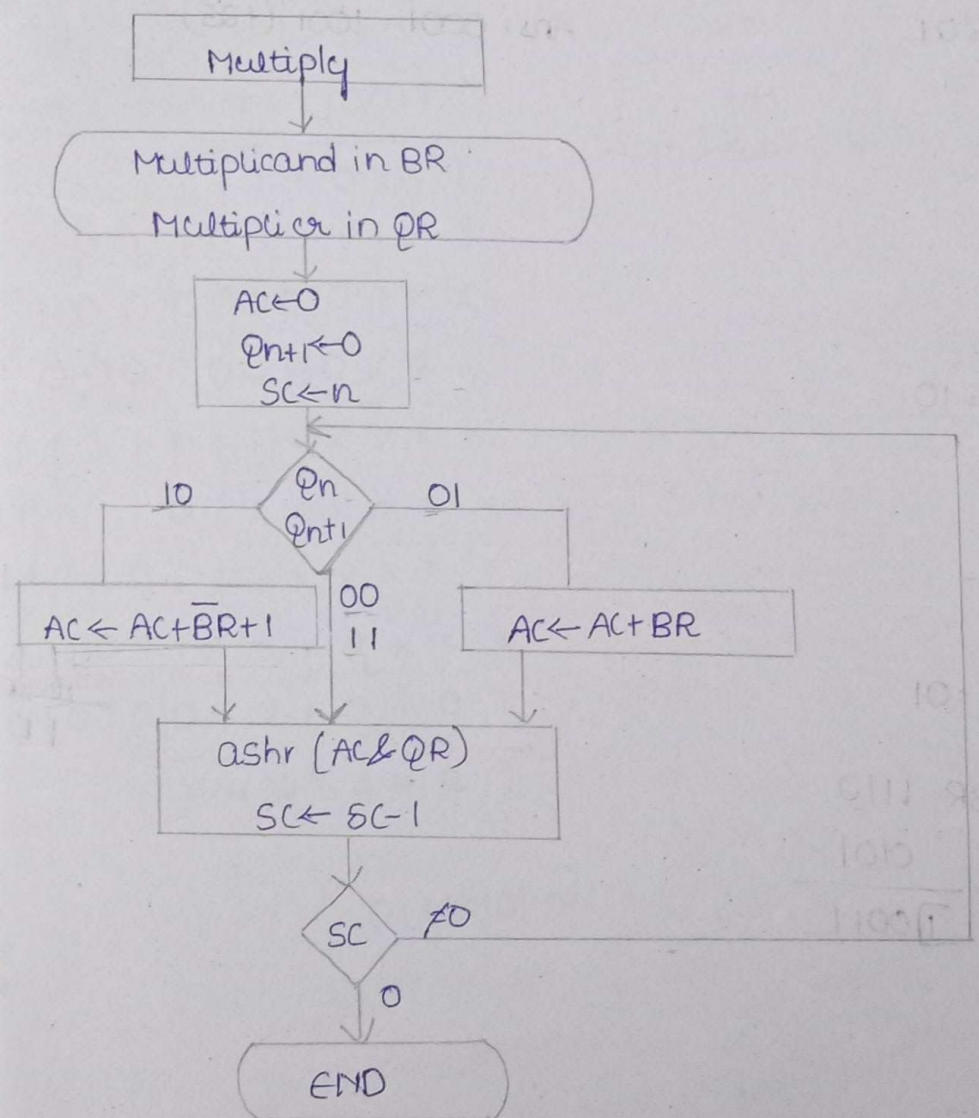
Shr EAQ

SC ← SC-1

Two's complement Multiplication (Booth's Algorithm)



Flowchart:-



Ex: (+5) x (+5)

BR = 5 Multiplicand

QR = 5 Multiplier

BR = 0101

QR = 0101

$\overline{BR}+1 = 1010 + 1 = 1011$

$AC+\overline{BR}+1 = 0000 + 1011 = 1011$

AC	QR	Q_{n+1}	SC
0000	0101 (Q_n)	0	4
1011	0101	0	
1101	1010	1	3
0010	1010	1	
0001	0101	0	2
1100	0101	0	
1110	0010	1	1
0011	0010	1	
0001	1001	0	0

Ans: 0001 1001 (+25)

1. $Q_n Q_{n+1} = 10$

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

2. $Q_n Q_{n+1} = 01$

   $AC \leftarrow AC+BR$

   1101
   0101
   ---
   1 0010

3. $Q_n Q_{n+1} = 10$

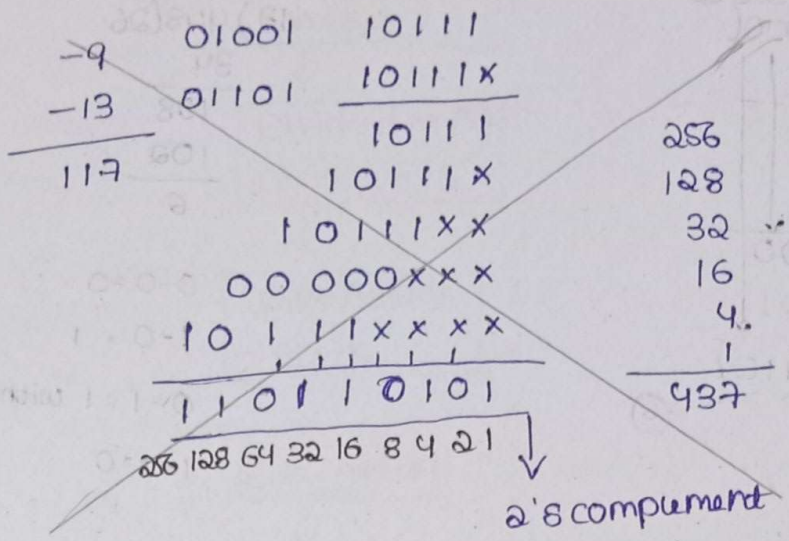
   0001
   1011

   1100

4. $Q_n Q_{n+1} = 01$

   $AC \leftarrow AC+BR$ 1110
   0101
   ---
   1 0011

Ex2: $(-9) \times (-13)$



8bit

-9	-01001	-10010	-11110111
-13	-01101	-10010	-11110011

binary

2's - 010

3's - 11

4's - 100

5's - 101

6's - 111

11110111

11110011 x

11 } faster
00 }
01 - Add
10 - Sub

11110111

11110111 x

00000000xx

000000000xxx

11110111xxxx

111101111xxxx

111101111xxxxx

110111xxxxxx

01010 01110101

128 64 32 16 8 4 2 1

01110101 ⇒ 117

Signed magnitude division:

```
            11010 (26)
1001) 01110 00000(
   -> 10001
      -----
      010110
       10001
(-)   -----
      0010100
        10001
(-)   -----
      000110  -> (6)
```

```
17) 448(26
    34
    ---
    108
    102
    ---
      6
```

0-0=0

1-0=1

0-1=1 with borrow

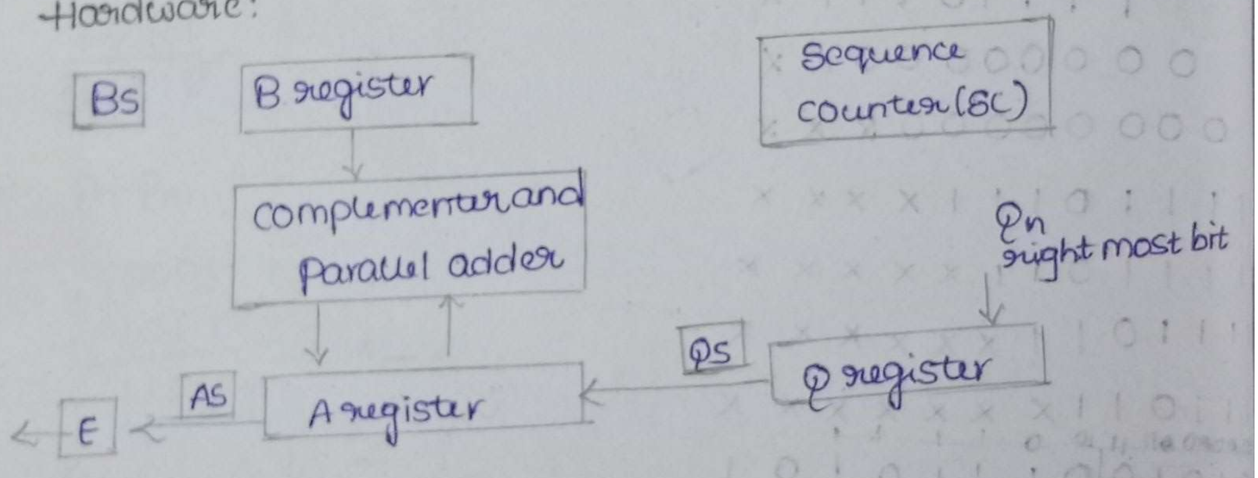
1-1=0

Divide overflow:

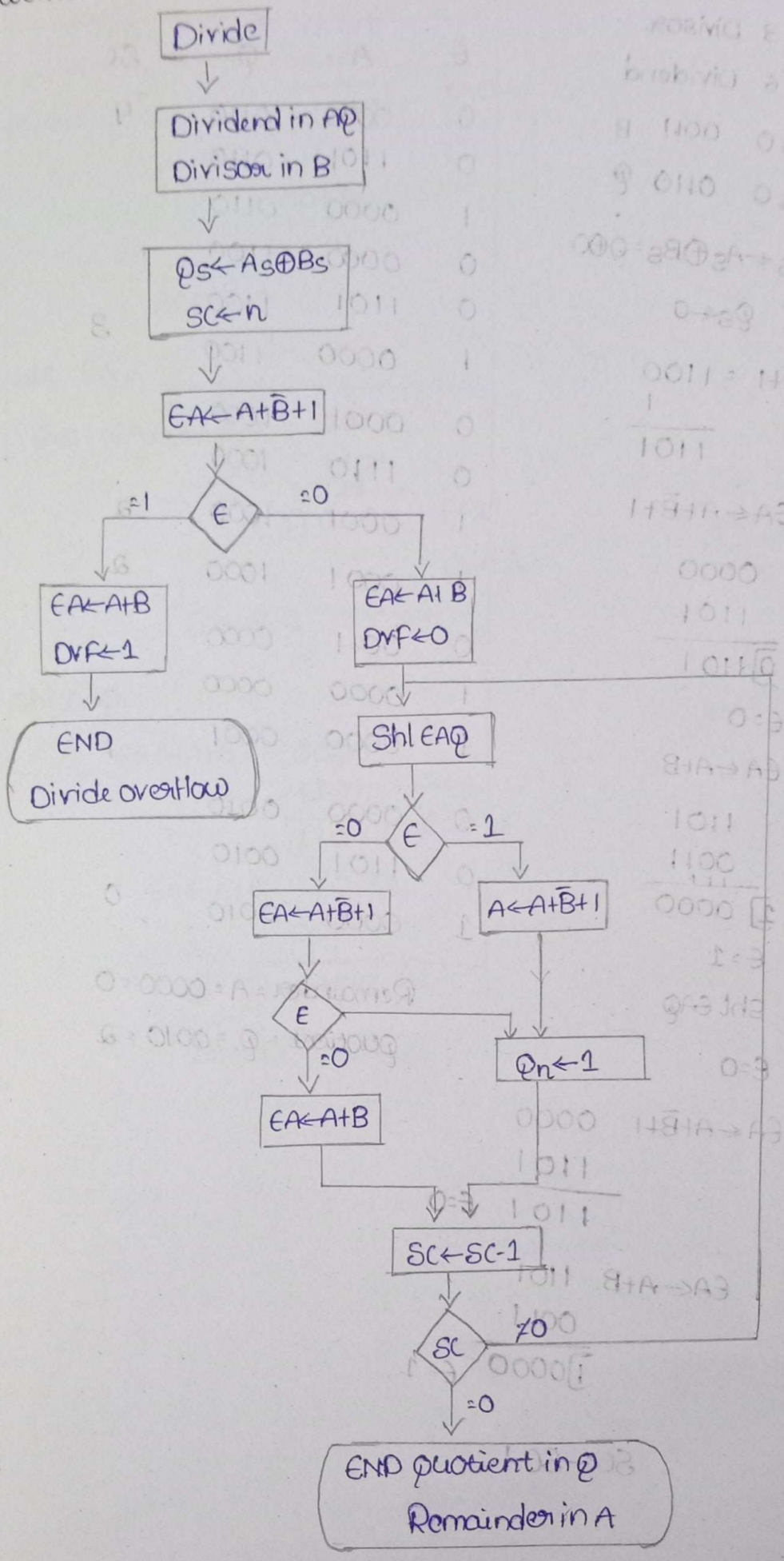
```
0110) 1001 0000(
      0110
(-)   ----
      001100
         110
      ------
         (0)
```

Signed magnitude division:

Hardware:



Flowchart:


Divide
Dividend in AQ
Divisor in B
Qs ← As ⊕ Bs
SC ← n
EA ← A + B̄ + 1
E
=1
=0
EA ← A + B
DVF ← 1
END
Divide overflow
EA ← A | B
DVF ← 0
Shl EAQ
E
=0
=1
EA ← A + B̄ + 1
A ← A + B̄ + 1
E
=0
Qn ← 1
EA ← A + B
SC ← SC - 1
SC
≠0
=0
END quotient in Q
Remainder in A

Ex:- (+6) ÷ (+3)

B = 3 Divisor

Q = 6 Dividend

Bs 0 0011 B

Qs 0 0110 Q

$Q_S \leftarrow A_S \oplus B_S = 0 \oplus 0$

$Q_S \leftarrow 0$

$\bar{B}+1 = 1100 + 1 = 1101$

1. $EA \leftarrow A + \bar{B} + 1$

```
  0000
  1101
 -----
 0 1101
```

E = 0

$EA \leftarrow A + B$

```
  1101
  0011
 -----
 1 0000
```

E = 1

Shl EAQ

E = 0

$EA \leftarrow A + \bar{B} + 1$

```
  0000
  1101
 -----
  1101   E = 0
```

$EA \leftarrow A + B$

```
  1101
  0011
 -----
 1 0000   E = 1
```

$SC \leftarrow SC - 1$

E	A	Q	SC
0	0000	0110	4
0	1101	0110	
1	0000	0110	
0	0000	1100	
0	1101	1100	3
1	0000	1100	
0	0001	1000	
0	1110	1000	
1	0001	1000	2
1	0001	1000	2
0	0011	0000	
1	0000	0000	
1	0000	0001	1
0	0000	0010	
0	1101	0010	
1	0000	0010	0

Remainder = A = 0000 = 0

quotient = Q = 0010 = 2

2. Shl EAQ

E=0 EA ← A+B̄+1 0001
1101
1110 E=0

EA ← A+B 1110
0011
1 0001

SC ← SC-1

3. Shl EAQ

E=0 EA ← A+B̄+1 0011
1101
1 0000 E=1

Q0 ← 1

SC ← SC-1

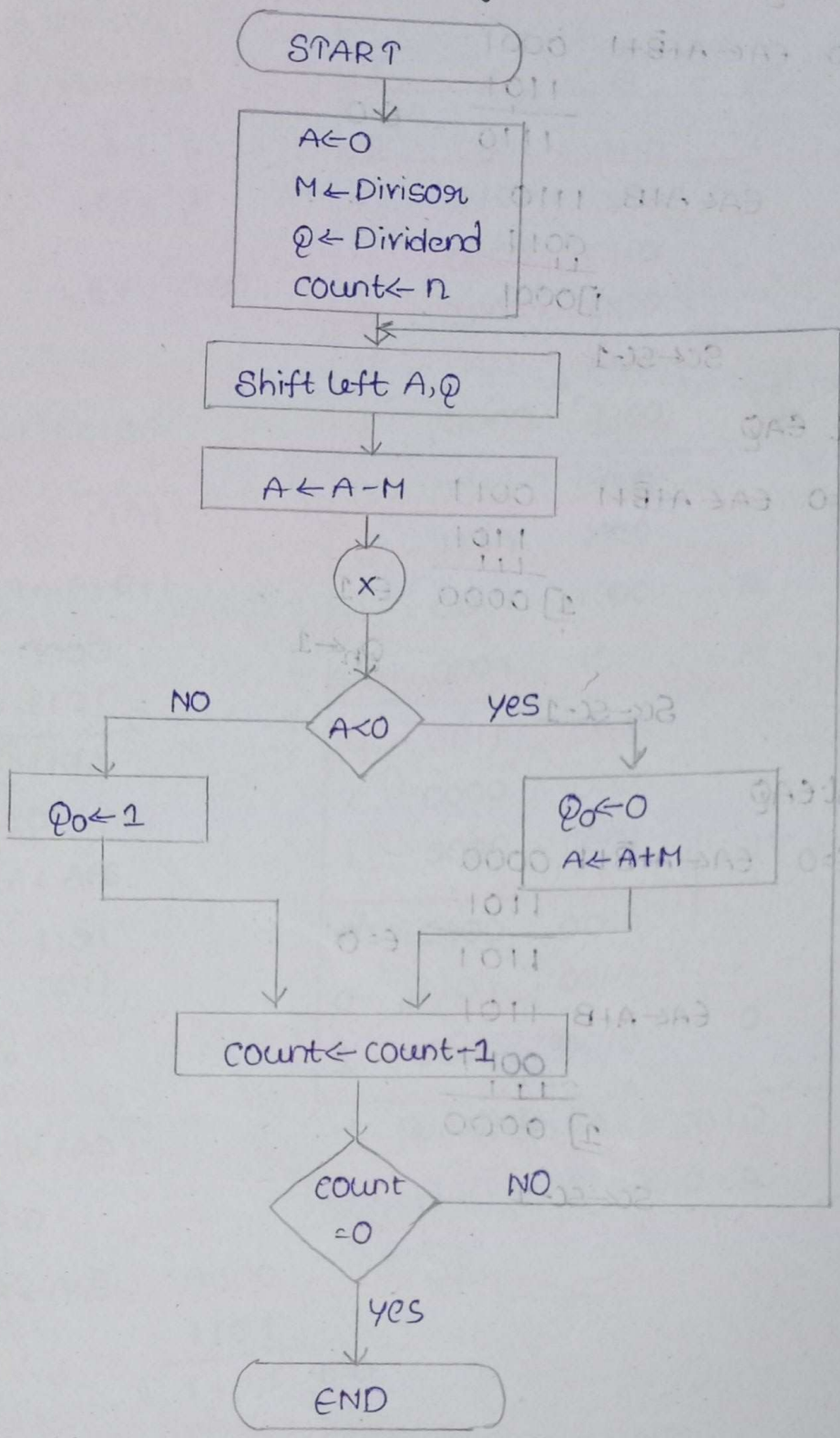
4. Shl EAQ

E=0 EA ← A+B̄+1 0000
1101
1101 E=0

EA ← A+B 1101
0011
1 0000

SC ← SC-1

Flowchart for unsigned binary division:-



If MSB=1, then A<0

Q_0 : LSB

9/6/22

Floating point multiplication:

1. Check for zeroes
2. Add the exponents
3. Multiply the Mantissas
4. Normalize the result

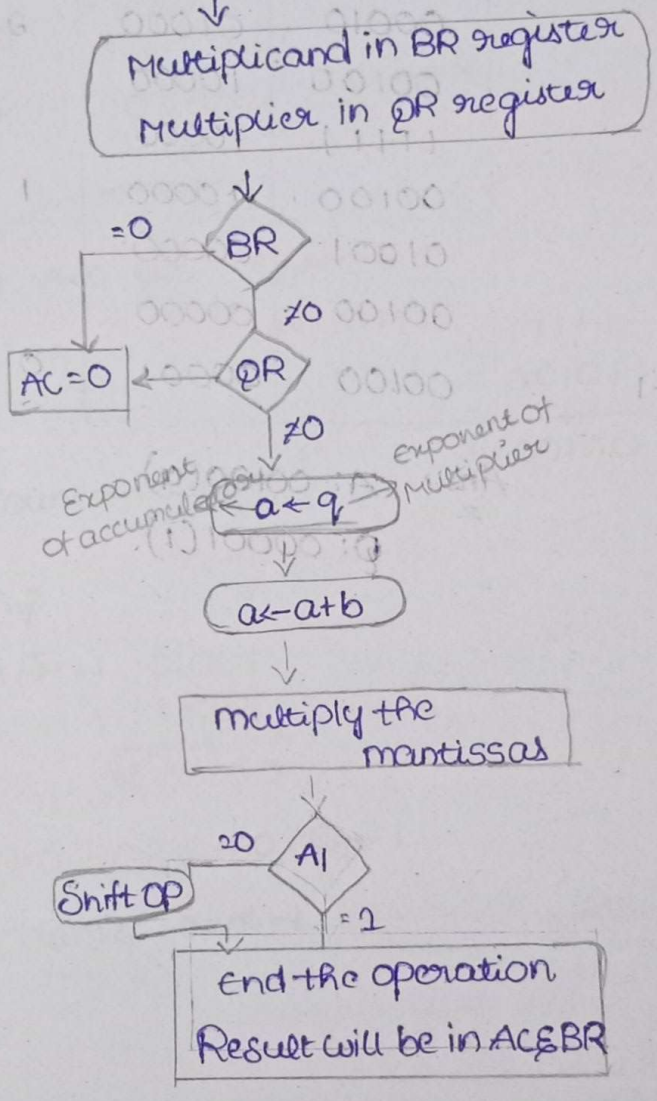
$m \times r^e$ (or) $s \times b^e$

0.53×10^5 - multiplicand (BR)

0.21×10^7 - multiplier (QR)

* Adder circuit
  ↓
  Performs addition of Exponents

Multiply operation



Example for unsigned binary division:

$(+9) \div (+5)$

Q: 01001 (+9)

M: 00101 (+5)

1. $A - M = A + \bar{M} + 1$

$\bar{M}+1$: 11011

$A \leftarrow A - M$

00000
11011

11011

$A < 0$ yes

$Q_0 \leftarrow 0$

$A \leftarrow A + M$

11011
00101

1] 00000

Count ← count - 1

2. Shl A, Q

$A \leftarrow A + \bar{M} + 1$

00001
11011

11100

$A < 0$ yes

$Q_0 \leftarrow 0$

$A \leftarrow A + M$

11100
00101

1] 00001

Count ← count - 1

A	Q	count
00000	01001	5
00000	10010	
11011	10010	
00000	10010	4
00000	00100	
11100	00100	
00001	00100	3
00010	01000	
11101	01000	
00010	01000	2
00100	10000	
11111	10000	
00100	10000	1
01001	00000	
00100	00000	
00100	00001	0

Ans: A: 00100 (4)

Q: 00001 (1)

3. Shl A,Q

$A \leftarrow A + \bar{M} + 1$

```
  00010
  11011
 -------
  11101
```

MSB=1 A<0 yes

$Q_0 \leftarrow 0$

$A \leftarrow A + M$

```
  11101
  00101
 -------
 X 00010
```

count ← count - 1

4. Shl A,Q

$A \leftarrow A + \bar{M} + 1$

```
  00100
  11011
 -------
  11111
```

MSB=1 A<0 yes $Q_0 \leftarrow 0$

$A \leftarrow A + M$

```
  11111
  00101
 -------
 100100
```

Count ← count - 1

5. Shl A,Q

$A \leftarrow A + \bar{M} + 1$

```
  01001
  11011
 -------
 X 00100
```

MSB=0 A<0 no $Q_0 \leftarrow 1$

count ← count - 1

10/6/22

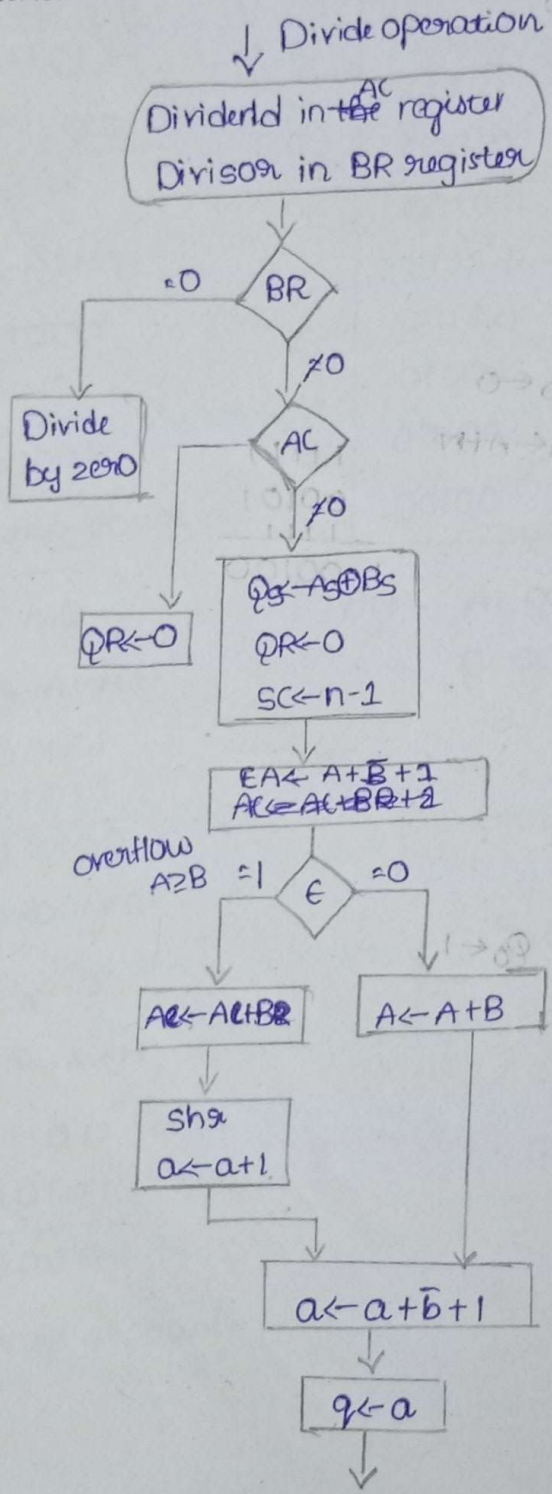
Floating Point Division Algorithm:-

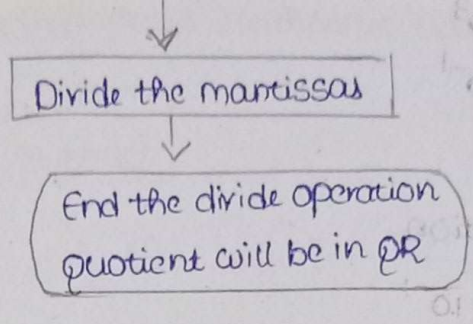
1. Check for zeroes
2. Initialize the registers
3. Align the Dividend
4. Subtract the Exponents
5. Divide the mantissas

AC → 0.236×10^5 → Dividend

BR → 0.12×10^3 → Divisor

Flowchart:-





7/6/22

Normalization:

The most significant bit mantisa shouldn't be zero

Overflow
↓
Add
↓
1.235×10^3
↓
overflow
0.1235×10^4
↓
increase the Exponent

Underflow
↓
Sub
0.035×10^2
↓
underflow
↓
0.35×10^1
decrease the exponent

Floating Point Addition or Subtraction:

Steps:

→ Check for zeros
→ Assignment of mantissas
→ Add or Subtract the mantissas
→ Normalize the result

AC, BR ⇒ AC ← AC + BR
AC ← AC + $\overline{BR}$ + 1

1. AC = 0.000
BR = 0.000
AC ← 0

2. AC = 0.123
BR = 0.000
AC ← AC

3. AC = 0.000
BR = 0.123
AC ← BR
Add
AC ← $\overline{BR}$ + 1
Sub

Ex: $AC \leftarrow 0.538123 \times 10^3$

$BR \leftarrow 0.123000 \times 10^{-1}$

1) 10^{-1}

Shift right operation,

$AC \rightarrow 0.123000 \times 10^{-1}$

$BR \rightarrow 0.123000 \times 10^{-1}$

Add $\rightarrow$ $0.353000 \times 10^{\oplus}$ $\rightarrow$ Exponent should not be in floating point

Ex: 0.532×10^3

0.712×10^3

1.244×10^3 } Normalization process

0.1244×10^4

2) 10^3

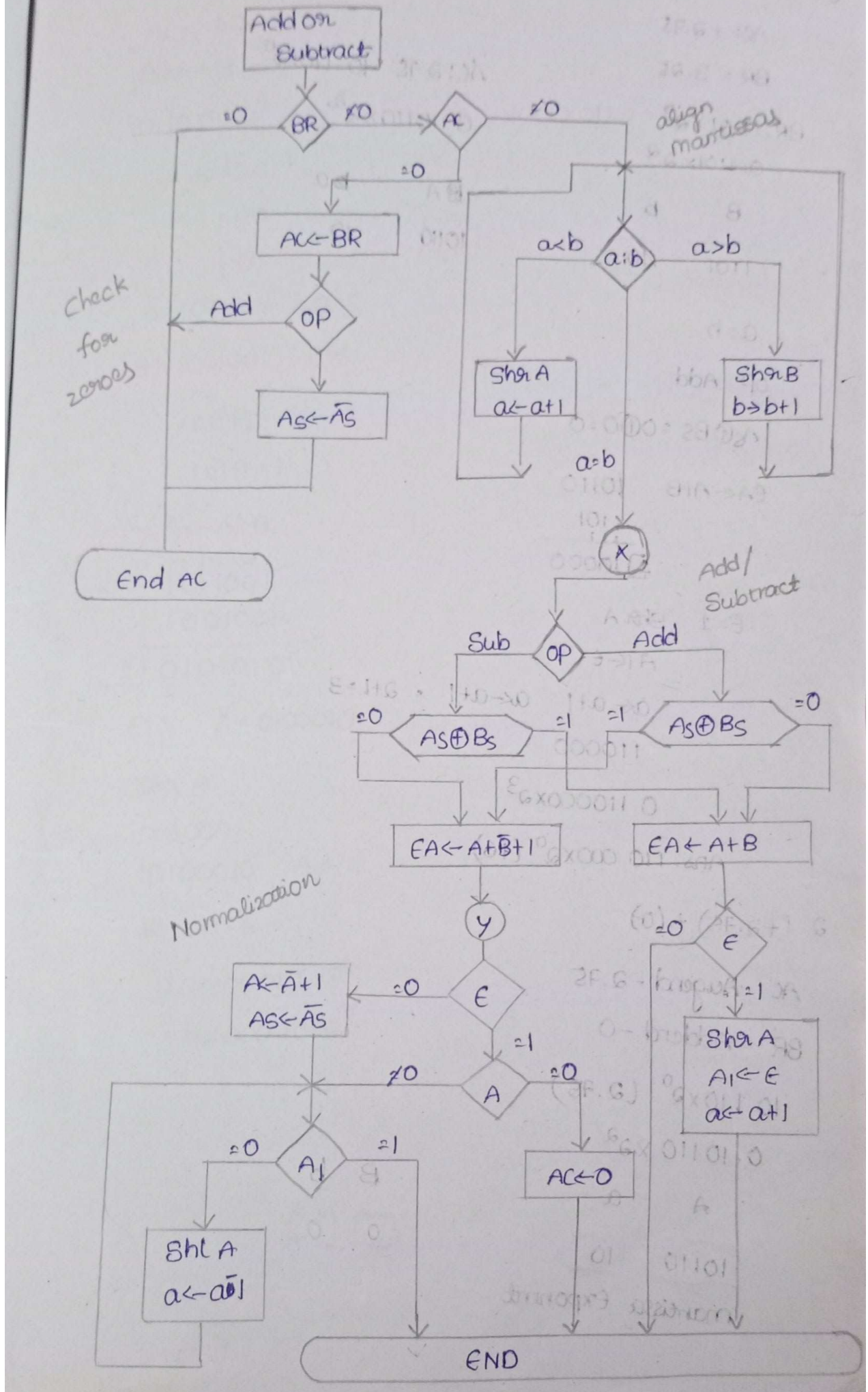
Shl

$AC \leftarrow 0.538123 \times 10^3$

$BR \leftarrow 0.000012 \times 10^3$

Add $\rightarrow$ 0.538135×10^3 $\rightarrow$ loss of bits will be less in 10^3 so we can consider 10^3

Flowchart: Floating Point Arithmetic Addition & Subtraction:-



Ex: 1. (+2.75) + (+3.25)

AC = +2.75

BR = 3.25

BR: 11.01×2^0

0.1101×2^2

B	b
1101	10

AC: 2.75 10.110×2^0

0.10110×2^2

BA	ba
10110	10

a = b

op Add

$A_S \oplus B_S = 0 \oplus 0 = 0$

EA ← A+B

```
  10110
+  1101
-------
 10000
```

E = 1 Shr A

A1 ← E

a ← a+1 a ← a+1 = 2+1 = 3

110000

0.110000×2^3

Ans: 110.000×2^0 (+6)

2. (+2.75) + (0)

AC Augend - 2.75

BR Addend - 0

10.110×2^0 (2.75)

0.10110×2^2

A	a
10110	10

mantissa exponent

B	b
0	0

3. $(-22.25)-(+18.25)$

AC = -22.25

BR = +18.25

$10110.01 \times 2^0 \Rightarrow 0.1011001 \times 2^5$ $10010.01 \times 2^0 \Rightarrow 0.1001001 \times 2^5$

.25 x 2 = 0.5

.5 x 2 = 1.0

= 0

A = 1011001, a = 5

B = 1001001, b = 5

$A_6 \oplus B_5$

$1 \oplus 0 = 1$

$EA \leftarrow A + B$

```
  1 1   1
  1011001
  1001001
-----------
1 0101010
```

E = 1 A = 0100010

Shr A

0010001

10100010 $A_1 \leftarrow E$

~~b~~ a = 6

0.10100010×2^6

101000.10×2^0

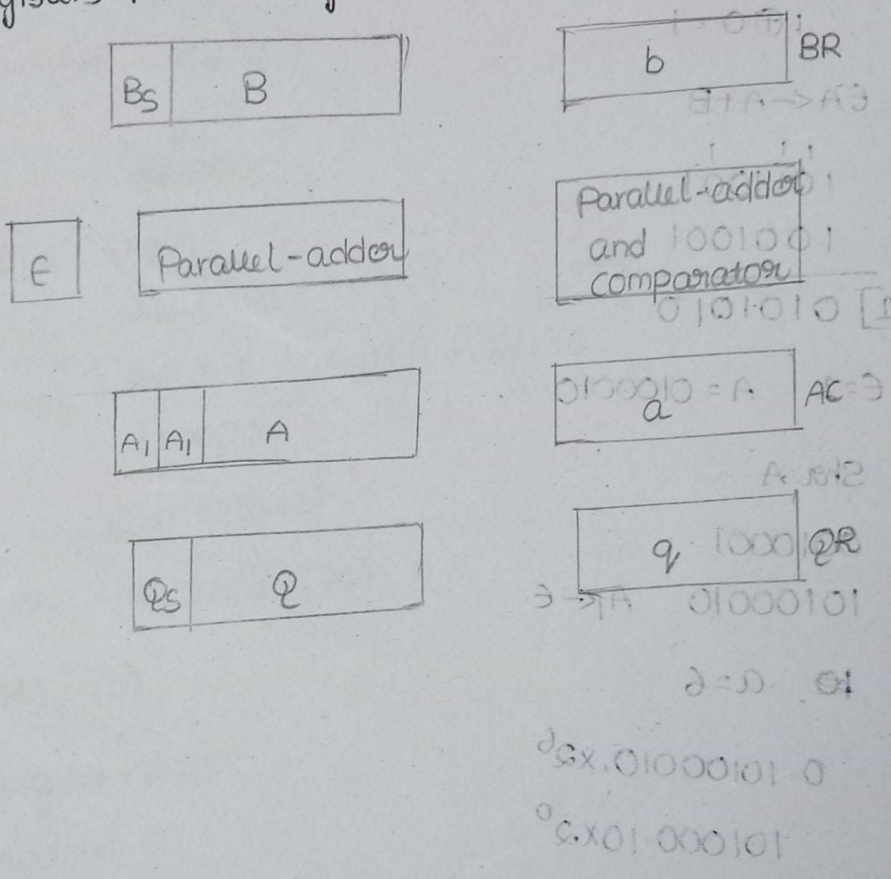
Restoring method:

The hardware method just described is called the restoring metho

Register organization:

There are 3 registers, BR, AC and QR. Each register is subdivided into two parts. The mantissa part has same upper case letters, the exponent part uses the corresponding lower case letters A parallel adder adds the two mantissas and transfers the sum into A and the carry into E. A separate parallel adder is used for the exponents. Since the exponents are biased

Registers for floating-point arithmetic operations:



Example for floating point multiplication:

$(+3.5) \times (+2.5)$

$BR = 3.5 = 11.10 \times 2^0 = 0.111 \times 2^2$

$QR = 2.5 = 10.10 \times 2^0 = 0.101 \times 2^2$

$B = 111$ $\quad a \leftarrow q$

$Q = 101$ $\quad a \leftarrow a+b = 2+2 = 4$

E	A	Q	SC
0	000	101	3
0	111	101	
0	011	110	2
0	001	111	1
1	000	111	
0	100	011	0

0.100011×2^4

1000.11×2^0

(8.75)

$Qn = 1$

$EA \leftarrow A+B$

```
  000
  111
 ----
  111
```

Shr EAQ

$SC \leftarrow SC-1$

$Qn = 0$

Shr EAQ

$SC \leftarrow SC-1$

$Qn = 1$ $EA \leftarrow A+B$

```
  001
  111
 ----
 1000
```

Shr EAQ

$SC \leftarrow SC-1$

Example for floating point division:

Ex:- (+6.5) ÷ (+3.25)

AC = 6.5 ⊙ $110.10 \times 2^0 = 0.11010 \times 2^3$

BR = 3.25 ⊙ $11.01 \times 2^0 = 0.1101 \times 2^2$

Qs = 0

$\overline{B}+1 = 1101$

```
  0010
     1
 ------
  0011
```

```
  11010
   0011
  -----
 1 00000
```

E = 1 A = 00000

```
  00000
  11010
 ------
  11010
```

A = A+B ⇒ A = 11010

Shr → 01101 | A = 01101

a = a+1 ⇒ a = 4

a = a+$\overline{b}$+1

4 - 2

a = 2

q = 2

E	A	Q	SC
0	01101	00000	5
0	10011	00000	
1	01101	00000	
0	11010	00000	
1	00000	00000	
1	00000	00001	4
0	00000	00010	
0	00110	00010	3
1	00000	00010	
0	000000	00100	
0	00110	00100	2
1	00000	00100	
0	00000	01000	
0	00110	01000	1
1	00000	01000	
0	00000	10000	
0	00110	10000	0
1	00000	10000	

Remainder = 0

Quotient = 0.10000×2^a

$\Rightarrow 10.000 \times 2^0$

$= 2$

B = 1101

B+1 = 0011

```
 01101
+0011
------
 10011
```

```
 10011
+ 1101
------
1|01101
```

```
 11010
+ 0011
------
1|00000
```

```
 00000
+ 0011
------
 00110
```

```
 00110
+ 1101
------
1|00000
```

```
 00000
+ 0011
------
 00110
```

```
 00110
+ 1101
------
1|00000
```

14/6/22

Instruction set Architecture

* An ISA is part of the abstract model of a computer that defines how the CPU controlled by the software
* It is the part of computer Architecture which is related to programming. instructions, registers, addressing modes etc
* Instruction set: It is a combination of instructions, that a processor executes
* Software co-design

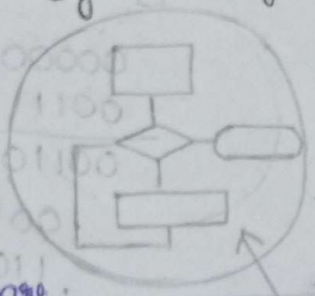
→ compiler developer use ISA to develop compilers

→ CPU designers design a CPU to implement the ISA

Some open source examples:

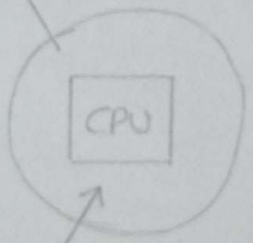
* MIPS 32, MIPS 64
* RISC-V

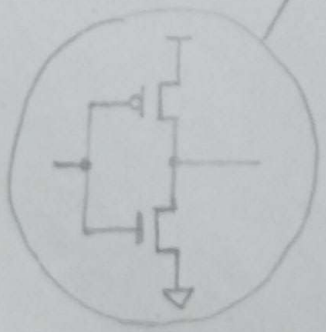
Computer System: Layers of abstraction:



Application program
Algorithms
Language

Software :
Hardware :





ISA
Microarchitecture
Circuits
Devices

Instruction types:

* Arithemetic operations

  Addition, Subtraction, multiplication, division

* Logic operations

  AND, OR, NOT etc

* Shift operations

  Logical shift left and right, and arithemetic shift right, others can be rotate shift with carry

* Memory access

  Transfer data blw the memory and registers with CPU

* Input/output access

  Some ISA can have dedicated IO transfer instructions, other might use memory-mapped IO

* Control Transfer

  Branch Instructions

* Floating-point operations

  Arithemetic operations on floating point numbers

* System control operations

  System calls

  Returns from Exceptions, instructions that read CPU State registers or write CPU control registers.

Instruction set operations:

-> All machines generally provide a full set of operations for the first three categories

-> All machines must provide instruction support for basic system functions

-> Floating point instructions are optional but are commonly provided

-> Decimal and String instructions are optional but are disappearing in recent ISAs. They can be easily emulated by sequences of simpler instructions

-> Graphic instructions are optional

Control flow instructions:

Four types are identifiable

* conditional branches
* Jumps
* Procedure calls
* procedure returns

Types of operands:

Machine instructions operate on data. The most important general categories of data are

* Addresses
* Numbers
  - Integer-fixed point
  - Floating point
  - Decimal
* Characters
* logical data

operand types:

* Main task of instruction is calculations on operands of different type

operand type	Bits	value range	C equivalent
Byte	8	-128 to +127	Signed char
unsigned byte	8	0 to 255	unsigned char
Half word	16	-32768 to +32767	Short int
unsigned half word	16	0 to 65535	unsigned short int
word	32	-2147483648 to +2147483647	int
unsigned word	32	0 to 4294967295	unsigned int
Single precision FP	32	IEEE 754: 1 sign, 8 exp, 23 mantissa	float
Double precision FP	64	IEEE 754: 1 sign, 11 exp, 52 mantissa	double

Instruction Set Architecture:

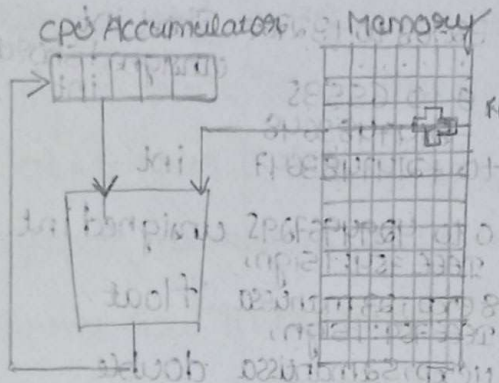
→ ISAs differ based on the internal storage in a processor. Accordingly, the ISA can be classified as follows, based on where the operands are stored and whether they are named Explicitly or Implicitly.

* Single accumulator organization
* General register organization
* Stack organization

1. Single accumulator organization:

* It is one of the general purpose registers as the accumulator and uses it to necessarily store one of the operations
* This indicates that one of the operands is implied to be in the accumulator and it is enough if the other operand is specified along with the instruction

Accumulator Architecture:



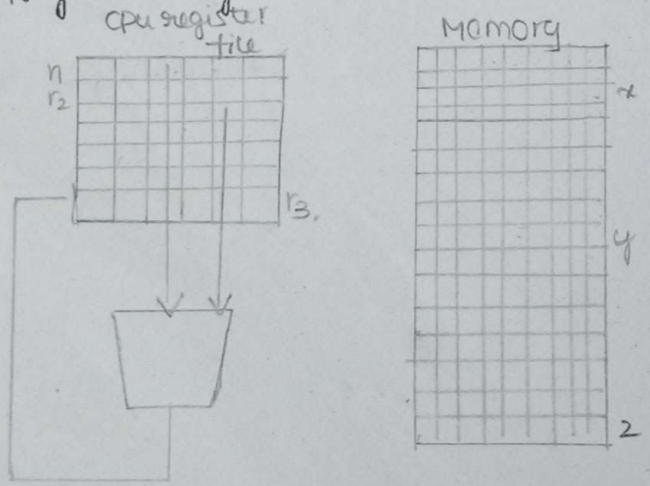
load x
add y
store 2

2. General Register organization:

* It specifies all the operands explicitly. Depending on whether the operands are available in memory or registers. It can be further classified as:

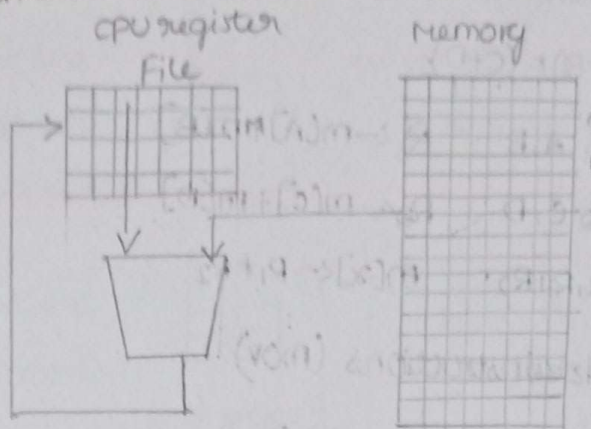
1. Register-Register :- For storing operands, such architectures are also called as used
2. Load-Store architectures
3. Register-memory :- where one operand is in register and other one in memory
4. Memory-memory :- where all the operands are specified as memory operands

Register Register architecture:



Register memory architecture:

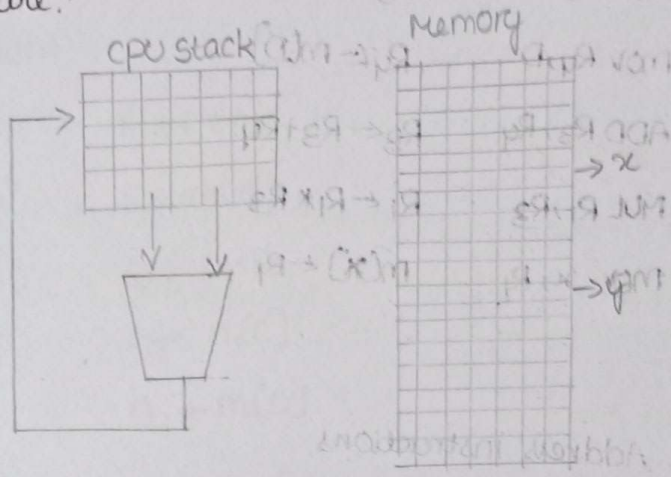
load R, x
add R, y
Store z, R



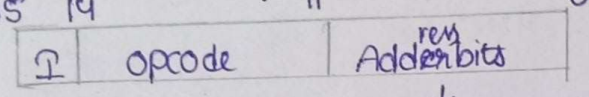
3. Stack organization:, where the operands are put into the stack and the operations are carried out on the top of the stack. The operands are implicitly specified here

Stack Architecture:

Push x
push y
add
pop x



Instruction format (16-bits)(0-15)



0 ⇒ 0-11 specify on address
Next 3 " " opcode
left most " " addressing mode
↓ operands

I=0 (Direct Addressing mode)
I=1 (Indirect " ")

1. 3 Address instructions (Direct Add 3 operands)
2. 2 " " (2)
3. 1 " " (1)
4. 0 " " (0)
5. RISC " " used load & store instructions

1. 3 Address instructions

$X = (A+B) * (C+D)$

Add R_1, A, B	$R_1 \leftarrow m[A] + m[B]$
Add R_2 C, D	$R_2 \leftarrow m[C] + m[D]$
MUL x, R_1, R_2	$m[x] \leftarrow R_1 * R_2$

2. 2 Address instructions (mov)

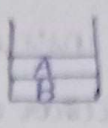
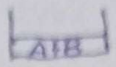
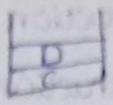
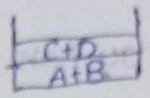
mov R_1, A	$R_1 \leftarrow m[A]$
mov R_2, B	$R_2 \leftarrow m[B]$
Add R_1, R_2	$R_1 \leftarrow R_1 + R_2$
mov R_3, C	$R_3 \leftarrow m[C]$
mov R_4, D	$R_4 \leftarrow m[D]$
ADD R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL R_1, R_3	$R_1 \leftarrow R_1 * R_3$
mov x, R_1	$m[x] \leftarrow R_1$

16/6/22

3. 1 Address instructions

LOAD A	$AC \leftarrow m[A]$
ADD B	$AC \leftarrow AC + m[B]$
STORE T	$m[T] \leftarrow AC$
LOAD C	$AC \leftarrow m[C]$
ADD D	$AC \leftarrow AC + m[D]$
MUL T	$AC \leftarrow AC * m[T]$
STORE X	$m[X] \leftarrow AC$

4. Zero Address instructions

PUSH A	TOS ← A	
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)	
POPX	M[X] ← TOS	

5. RISC - Instructions

LOAD R_1, A	$R_1 \leftarrow M[A]$
LOAD R_2, B	$R_2 \leftarrow M[B]$
ADD R_1, R_2	$R_1 \leftarrow R_1 + R_2$
LOAD R_3, C	$R_3 \leftarrow M[C]$
LOAD R_4, D	$R_4 \leftarrow M[D]$
ADD R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL R_1, R_3	$R_1 \leftarrow R_1 * R_3$
STORE X, R_1	$M[X] \leftarrow R_1$

Stack organization : would have push & pop instructions which require an address field

⇒ It is a storage structure that stores information in such a way that the last item stored is the first item retrieved

⇒ It is based on the principle ~~LOP~~ LIFO (Last-in-first-out)

Computer Instructions:

* An instruction format or instruction code is a group of bits used to perform a particular operation on the data stored in computer
* processor fetches on instruction from memory and decodes the bits to execute the instruction
* Different computers may have their own instruction set
* Instruction code is divided into two parts namely opcode and address code

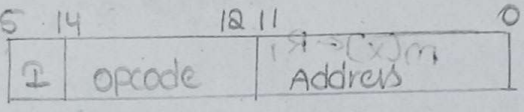
⇒ operation code consisting group of bits to define an operation such as add, subtract, multiply etc

Types of Instructions:

The basic computer has three 16-bit instruction code formats

1. Memory Reference Instructions
2. Register "
3. Input output "

Memory Reference Instructions:



0-11: specifies address

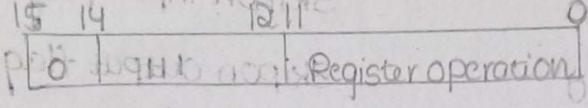
Next 3 " opcode

I=0 ← direct address

" I=1 ← Indirect address

Symbol	Hexadecimal code I=0	I=1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	ADD " to AC
LDA	2xxx	Axxx	LOAD " " "
STA	3xxx	Bxxx	Store content of in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and Save return address
ISZ	6xxx	Exxx	increment and skip if zero

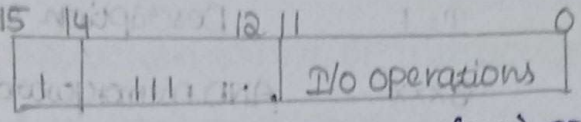
Register Reference Instructions:



first 12 bits specify the register operation
Next 3 bits equals to 111 specify opcode
The last mode bit of the instruction is 0
∴ Left most bits are always 0111 (7)

Symbol	Hexadecimal code	Description
CLA	7800	Clears AC
CLE	7400	Clears E
CMA	7200	complement AC
CME	7100	complement E
CIR	7080	Circulate right AC and E
CIL	7040	Circulate left AC and E
INC	7020	Increment AC
SPA	7010	Skip next instruction if AC is +ve
SNA	7008	" " " " " " -ve
SZA	7004	" " " " " " 0
SZE	7002	" " " " E " 0
HLT	7001	Halt Computer

I/O Reference Instructions:



First 12 bits (0-11) specifies I,O operation

The next 3 bits equals to 111 specify opcode

The last mode bit of the instruction is 1

∴ Left most 4 bits are always 1111 (which is equals to Hexadecimal F

I/O Reference Instructions:

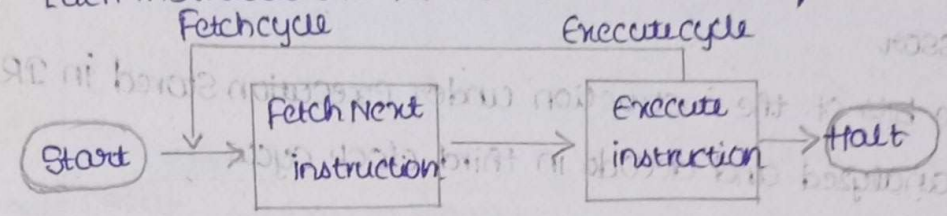
Symbol	Hexadecimal code	Description
INP	F800	Input character to AC
OUT	F400	output character from AC
SKI	F200	Skipon input flag
SKO	F100	Skipon output flag
ION	F080	Interrupt on
IOF	F040	Interrupt off

17/6/22

Instruction cycle

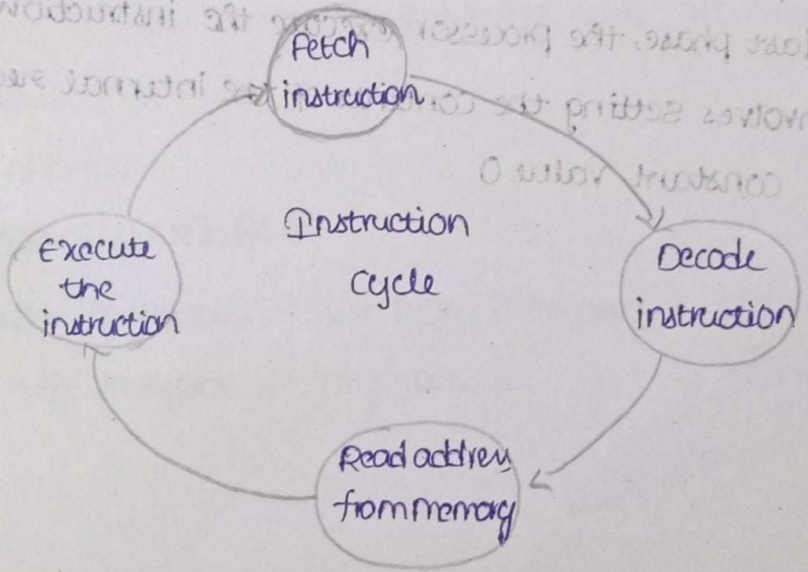
* Instruction is command which is given by user to the computer
* A program residing in the memory unit of a computer consists of a sequence of instructions
* These instructions are executed by the processor by going through a cycle for each instruction
* The time required for the fetching and execution of one simple machine instruction is the "CPU cycle". It is also known as an instruction cycle

⇒ Each instruction is further divided into sequence of phases



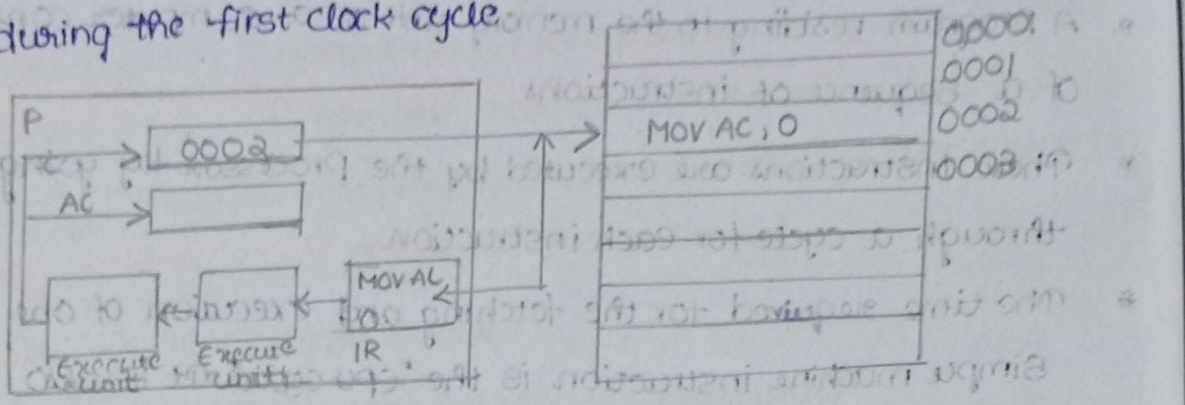
⇒ In a basic computer, each instruction cycle consists of the following phases

* Fetch instruction from memory
* Decode the instruction
* Read the effective address from memory
* Execute the instruction



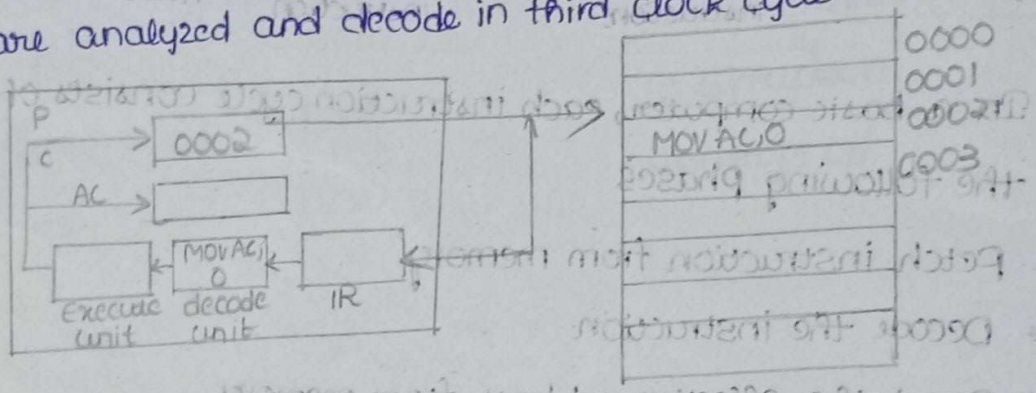
Fetch cycle:

* In this phase the SC is initialized to zero
* The address of first instruction from pc is loaded into register address during the first clock cycle



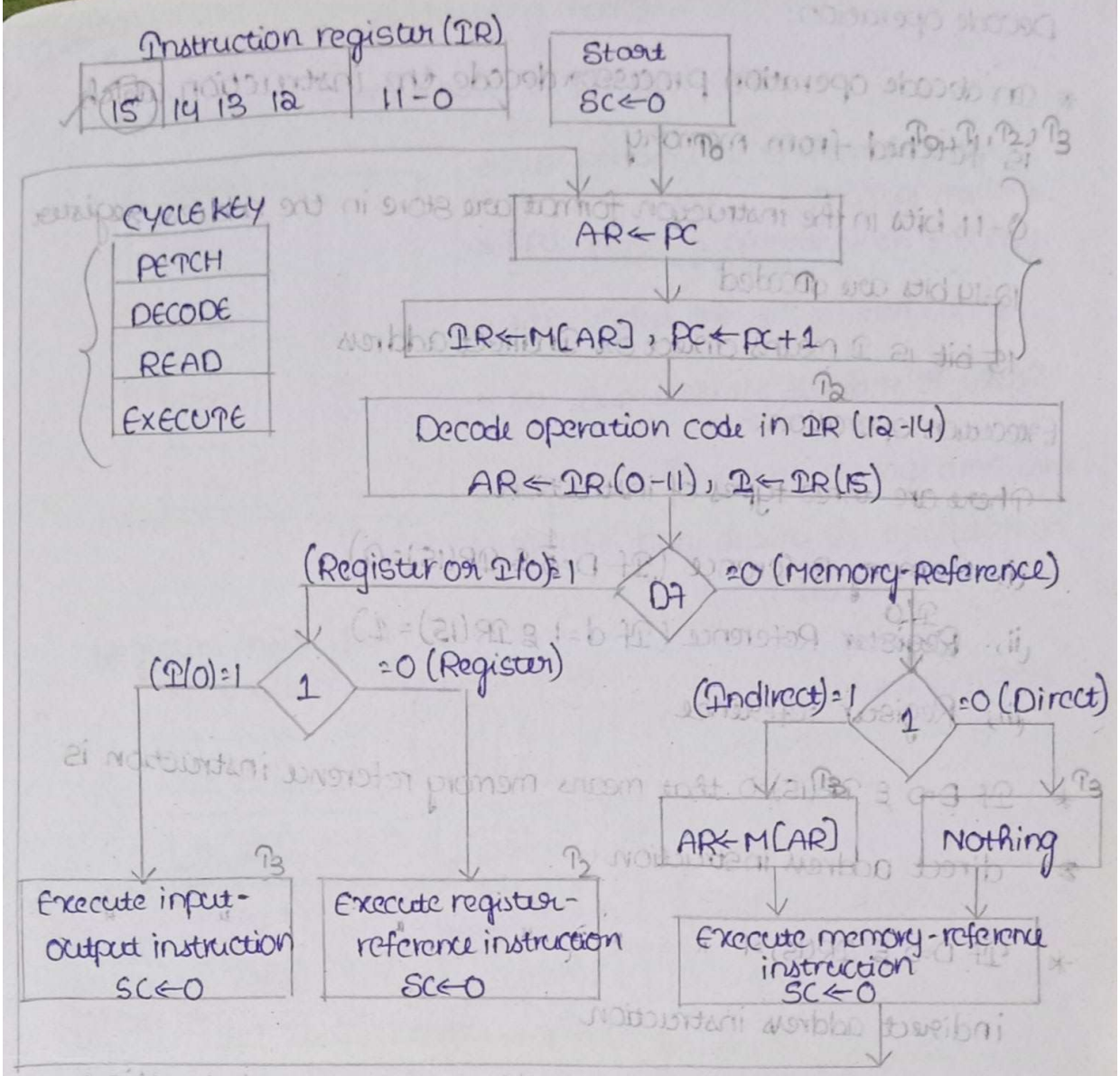
Decode cycle:

* The instruction is decoded by the instruction decoder of a processor
* All the bits of the instruction under execution stored in IR are analyzed and decode in third clock cycle



Execute cycle:

* In the last phase, the processor execute the instruction
* This involves setting the contents of the internal register AC to constant value 0



Flowchart for instruction cycle

Fetch operation:

* The circuit comprises of various registers, memory unit, I/O devices, control and timing unit ALU and a common bus
* AR←PC
* IR←M[AR], PC←
* The instruction read from memory is then placed in the instruction register.

Decode operation:

* In decode operation processor decode the instruction which is fetched from memory

  0-11 bits in the instruction format are store in the address register

  12-14 bits are decoded

  15 bit is I means direct or Indirect address

Execution operation:

There are three types of instruction:

(i) Memory Reference (If D=1 & IR(15)=0)

(ii) I/O Register Reference (If d=1 & IR(15)=1)

(iii) Register reference

* If D=0 & IR(15)=0 that means memory reference instruction is direct address instruction

* If D=0 & IR(15)=1 " " indirect address instruction

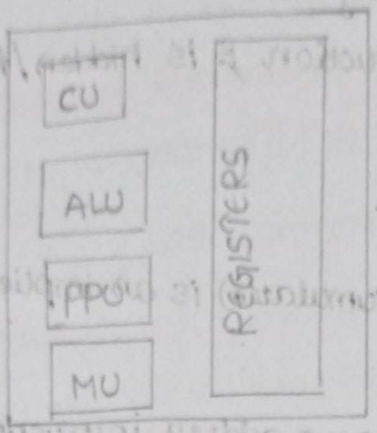
* AR ← M[AR], AR holds the address part of the instruction

* After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase

20/6/22

Addressing modes

Prerequisite - Introduction to parts of CPU



→ ALU: performs all arithematic & logical micro operations

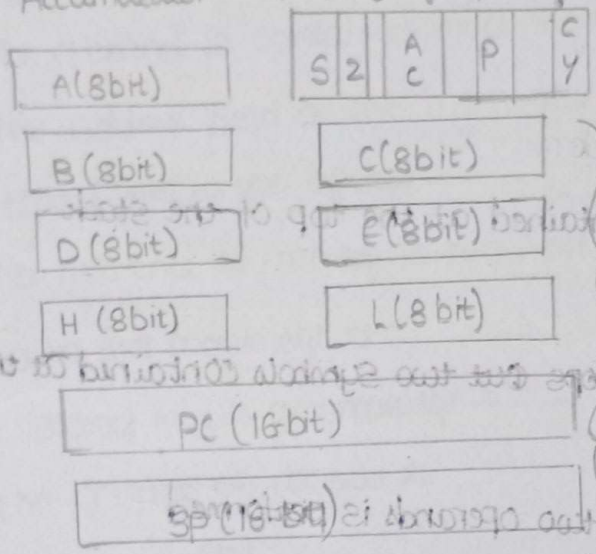
→ FPU: performs operations on FP nos

→ MU: Stores the set of instructions

→ CU: Supervises the sequence of micro ops

→ Registers: Temporary storage area, which holds the data during the execution of an instruction

Registers inside CPU:



* Addressing modes: The way in which the operand is taken from register or memory

MOV B, A → operands

↓

operation

Types of Addressing modes:

1. Implied Addressing mode:

* It is also called Inherent Addressing mode
* The operand is implied by the instruction & is hidden/fixed inside the instruction

Examples:

⇒ The instruction CMA (complement Accumulator) is an implied mode instruction

⇒ In a stack organized computer, zero address instructions are implied mode instructions

(∵ operands are always implied to be present on the top of the stack)

2. Stack Addressing mode:

The operand is contained at the top of the stack

Ex: ADD

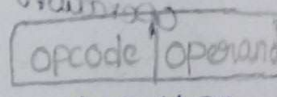
⇒ This instruction simply pops out two symbols contained at the top of the stack

⇒ The addition of those two operands is performed

⇒ The result so obtained after addition is pushed again at the top of the stack

3. Immediate Addressing mode:

* The operand is specified within the instruction



* operand itself is provided in the instruction rather than its address

Ex: Add 10 will increment the value stored in the accumulator by 10

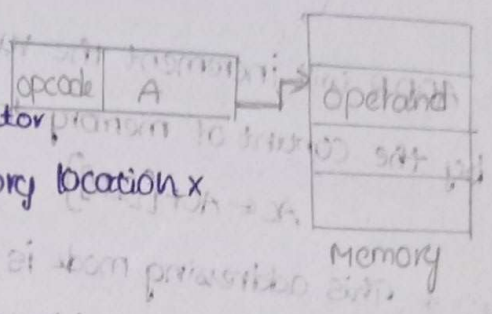
MOV R #20 initializes register R to a constant value 20

4. Direct Addressing mode:

* The address field of the instruction contains the effective address of the operand
* Only one reference to memory is required to fetch the operand
* It is also called as absolute addressing mode

Ex: ADD x will increment the value stored in the accumulator by the value stored at memory location x

$AC \leftarrow AC + [x]$

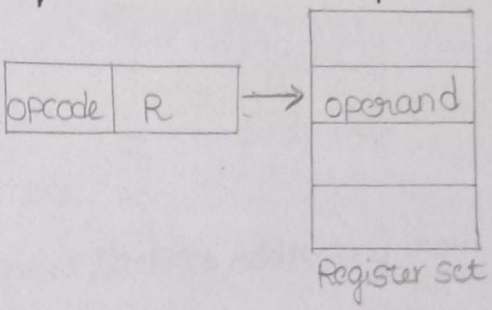


5. Register direct addressing mode:

* The operand is contained in a register set
* The address field of the instruction refers to a cpu register that contains the operand
* No reference to memory is required to fetch the operand

Ex: ADD R will increment the value stored in the accumulator by the content of register R

$AC \leftarrow AC + [R]$

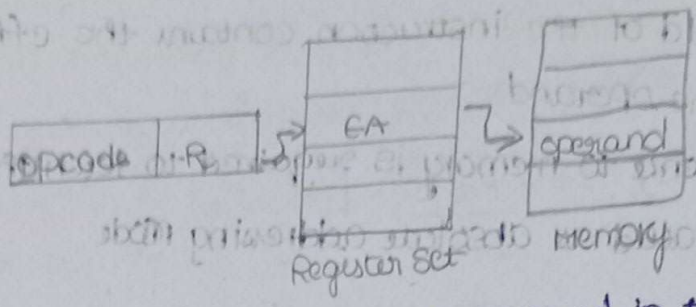


Note: 1. This addressing mode is similar to direct addressing mode

2. The only difference is address field of the instruction refers to a cpu register instead of main memory

6. Register Indirect addressing mode:

* The address field of the instruction refers to a cpu register that contains the effective address of the operand
* only one reference to memory



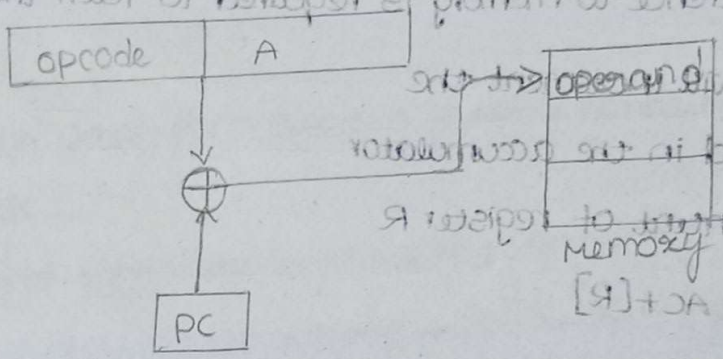
Ex:- ADDR will increment the value stored in the accumulator by the content of memory location specified in register R

$AC \leftarrow AC + [[IR]]$

Note:- * This addressing mode is similar to indirect addressing mode
* The only difference is address field of the instruction refers to a CPU register

8. Relative Addressing mode:

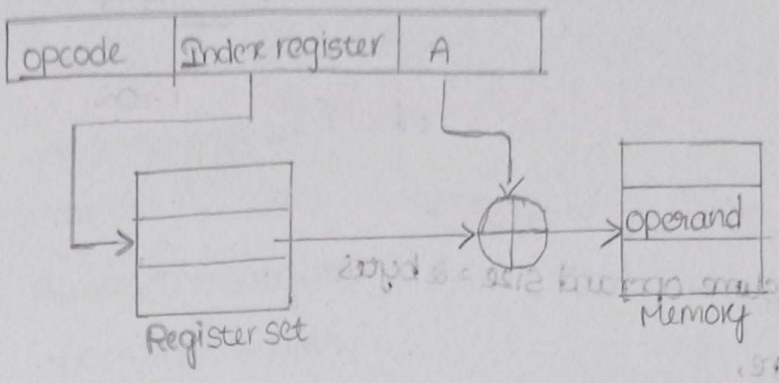
* Effective Address = content of program counter + Address part of the instruction



Note:- * PC always contains the address of the next instruction to be executed
* After fetching the address of the instruction, the value of PC immediately increases
* The value increases irrespective of whether the fetched instruction has completely executed or not

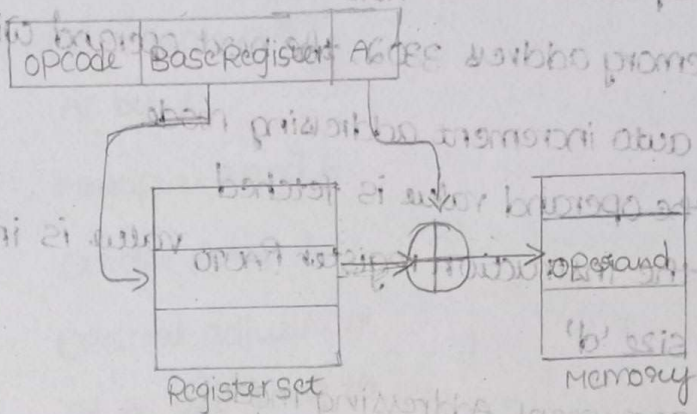
8. Indexed addressing mode:

* Effective Address = Content of Index Register + Address part of the instruction



9. Base Register Addressing Mode:

* Effective Address = Content of Base Register + Address part of the instruction



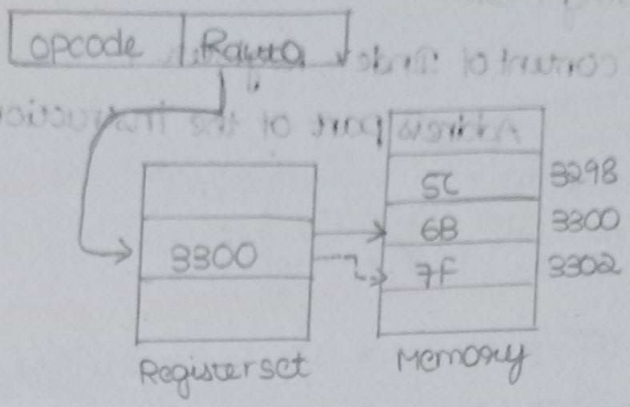
10. Auto Increment Addressing mode:

* It is a special case of Register Indirect Addressing mode where Effective Address of the operand = content of register

⇒ In this addressing mode

* After accessing the operand, the content of the register is automatically incremented by step size 'd'.
* Step size 'd' depends on the size of operand accessed
* Only one reference to memory is required to fetch the operand

Ex:



Assume operand size = 2 bytes

Here,

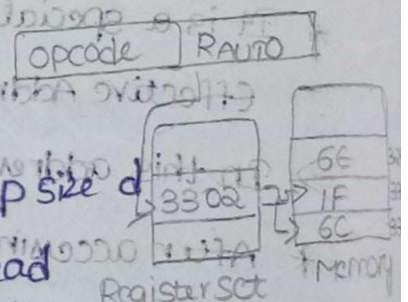
* After fetching the operand 6B, the instruction register R_{Auto} will be automatically incremented by 2
* Then, updated value of R_{Auto} will be 3300+2 = 3302
* At memory address 3302, the next operand will be found

Note: In auto increment addressing mode

* First, the operand value is fetched
* Then, the instruction register R_{Auto} value is incremented by step size 'd'

11. Auto-Decrement Addressing mode:

⇒ Effective address of the operand
= content of register - step size



⇒ In this addressing mode
content of register decremented by step size d
After decrementing, the operand is read

⇒ Assume operand = 2 bytes

Here, First the instruction Register R_{Auto} will be decremented by 2

Then, updated value of R_{Auto} will be 3302-2 = 3300

At memory address 3300, the operand will be found

21/06/22

Types of instructions:

1. Arithmetic instructions

Mnemonic	Description	Byte	cycle
Add A,Rn	Adds the register to the AC	1	1
Sub A,Rn	Subtracts the register from the AC with a borrow	1	1
INC A	Increments the Ac by 1	1	1
DEC A	Decrements the AC by 1	1	1
MUL AB	Multiplies A and B	1	5
DIV AB	Divides A by B	1	5
DA A	Decimal adjustment of AC according to BCD code	1	1

Etc.,

* The first part of the instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation)

* The other part of the instruction called OPERAND is separated by MNEMONIC

2. Branch instructions:

1. Unconditional Jump instructions

upon their execution a jump to a new location from where the program continues execution is executed

2. conditional Jump instructions

a jump to a new program location is executed only if specified condition is met, otherwise program normally proceeds with the next instruction

3. Data Transfer instructions:

* Data transfer instructions move the content of one register to another
* The register the content of which is moved remains unchanged
* If they have the suffix "x" (MOVx), the data is exchanged with external memory

4. Logic instructions:

* Logic instructions perform logic operations upon corresponding bits of two registers
* After execution, the result is stored in first operand

5. Bit- oriented instructions:

* Similar to logic instructions, bit oriented instructions performs logic operations
* The difference is that these are performed using single bits

23/6/22

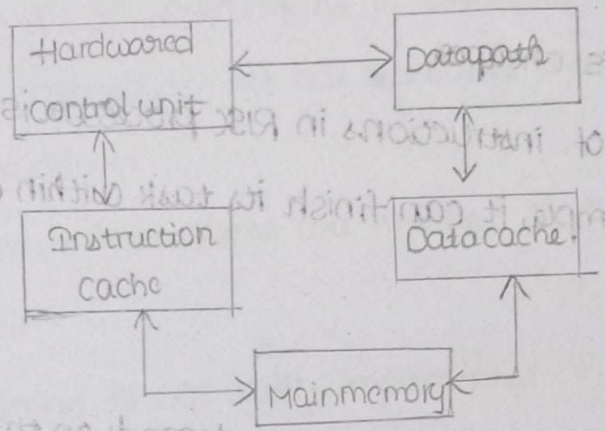
RISC and CISC Architecture

1. What is RISC?

* A reduced instruction set computer is a computer that only uses simple commands and executes within a single clock cycle as its name proposes, "Reduced instruction set".
* Its architecture includes a set of instructions that are highly customized
* used to execute several difficult commands by merging them into simpler ones
* RISC processor needs a number of transistors to design and it reduces the instruction time for execution

Ex:- alpha, AVR, ARM, PIC, PA-RISC - - - -

RISC Architecture:



* It is a cpu design plan based on simple orders & acts fast
* The instruction set of processor is simplified to reduce the execution time
* Each instruction of similar length

  These are wound together to get compound task done in single operation
* Most commands are completed in one machine cycle

Characteristics of RISC processor:

* Efficient pipeline architecture
* The no. of instructions are restricted
* Addressing modes are less
* It consists of a relatively few instructions (<50)
* " " " mostly Register-to-Register operations
* " contains Hardwared control unit
* Complexity is in the compiler
* very few instructions refer memory
* It consists of Multiple Register set
* An avg. CPI is less than 1.5 in RISC processor

Advantages:

* Performance of this processor is good because of the easy & limited no. of instruction set
* Design making is cheaper
* The execution of instructions in RISC processor is high
* It is very simple, it can finish its task within a single clock cycle

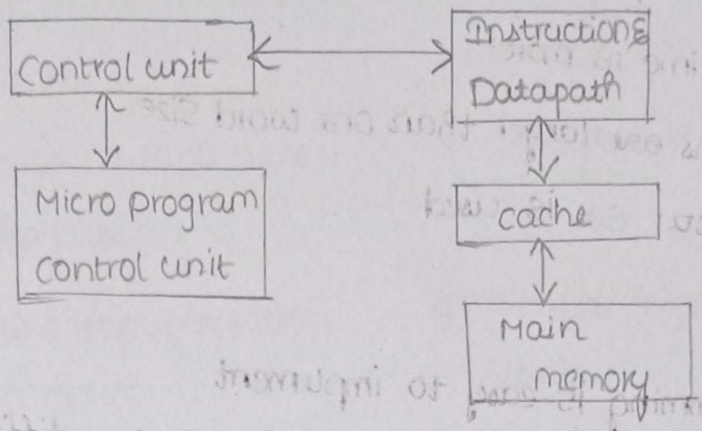
Disadvantages:

* The performance of a RISC processor depends on the code that is being executed
* The processor spends much time waiting for first instruction result

What is CISC?

* It was developed by the intel corporation and it is complex instruction Set computer
* This processor includes a huge collection of simple to complex instructions

CISC Architecture



* It is a cpu design plan based on single commands, which are skilled in Executing multi-step operations
* Single set of instructions is protected in several steps
* Each instruction set has addition than 250 separate instructions
* Maximum instructions are finished in two to ten machine cycle
* pipelining is not easily implemented

Ex: IBM 370/108
intel 80x86
VAX computer

Characteristics of CISC:

* CISC Supports high-level languages for simple compilation
* variable length instruction format used by CISC Architecture

* large no. of instructions, typically from 100 to 250 instructions
* A large variety of instructions, typically from 5 to 20 different addressing modes
* Not pipelined or less pipelined
* Complexity is in the Microprogram
* An average CPI is between 2 & 15
* Execution time is more
* Instructions are larger than one word size
* Single register set is used

Advantages:

* Microprogramming is easy to implement
* A single instruction can be executed by using different low-level tasks

Disadvantages:

* The pipeline execution within the CISC will make it difficult to use
* As compared to RISC, CISC processors are very slow while executing every instruction cycle on every program

CISC	RISC
1. CISC architecture gives more important to architecture	1. RISC architecture gives more important to software
2. complex instructions	2. Reduced instructions
3. It access memory directly	3. It requires registers
4. coding in CISC processor is simple	4. coding in RISC processor requires more no. of lines
5. It takes multiple times to execute	5. Single cycle.
6. pipeline implementation is complex	6. pipeline implementation is Simple
7. complexity lies in micro program	7. complexity lies in computer

24/6/22

Register Transfer Language and Micro operation

Register Transfer language:

* A specific notation used to specify the digital system
* For any function of the computer, the register transfer language can be used to describe the (sequence of) micro operations
* Register transfer language
  - A symbolic language
  - A convenient tool for describing the internal organization of digital computers
  - can also be used to facilitate the design process of digital systems

Micro operations: It is defined as the operations on the data in registers

* The functions built into registers are examples of micro-operations

  Shift, load, clear, increment ... etc

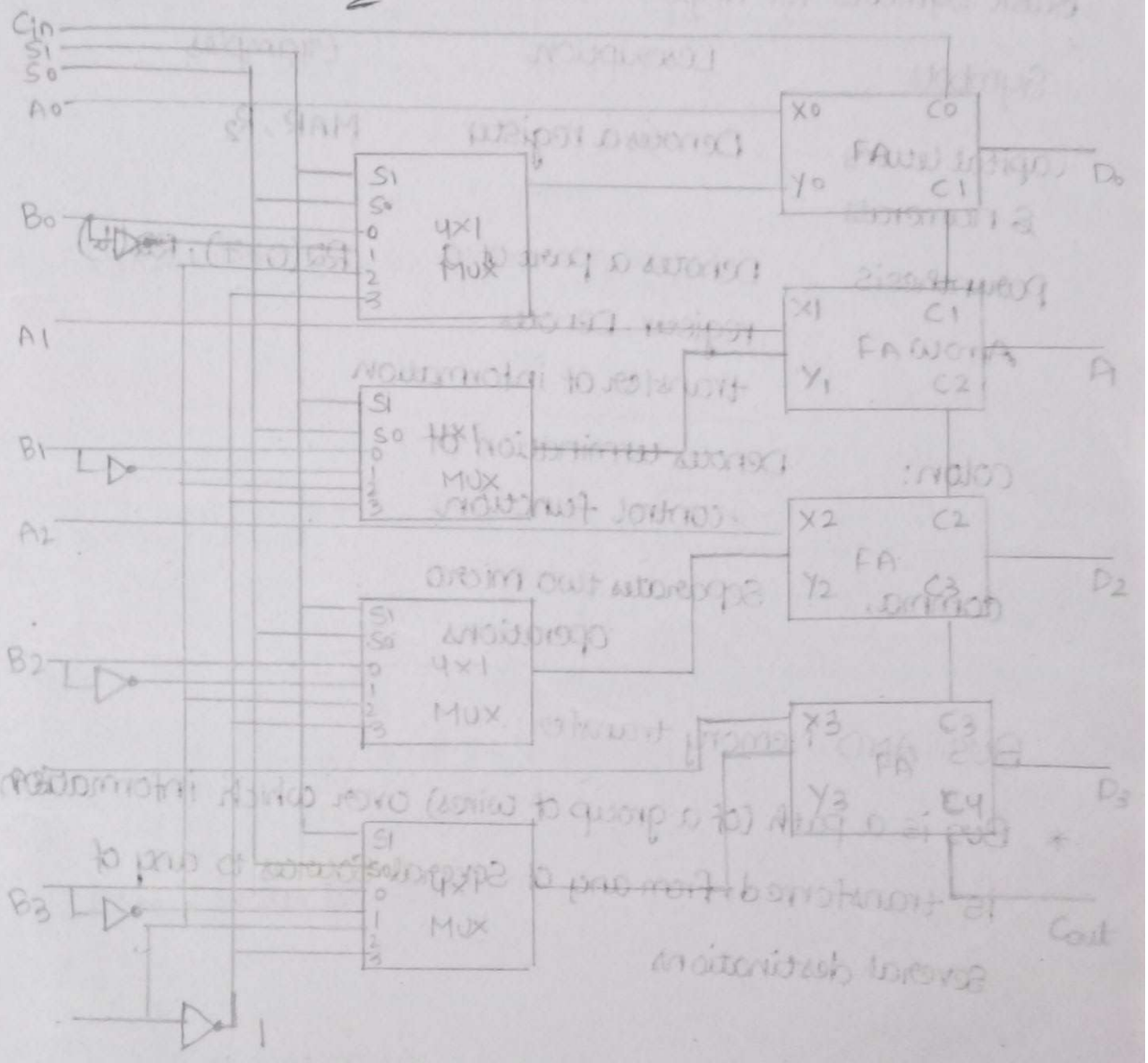
Micro operations

- Register transfer micro operations
  * $R_2 \leftarrow R_1$
  * one clock pulse
  * $R_2 \leftarrow R_1$; the contents of R_1 are not altered by copying them to R_2
- Arithmetic micro operations
  - Add with carry
  - Subtract with borrow
  - Transfer/load etc
- Logic micro-operations
  - Bit wise ops
  - Most used
    * AND ($\wedge$)
    * OR ($\vee$)
    * XOR ($\oplus$)
    * NOT
  * 16 different logic ops with 2 binary vars
  * n binary vars - 2^n functio
- Shift micro-ops
  - Logical shift
  - Circular "
  - Arithmetic "

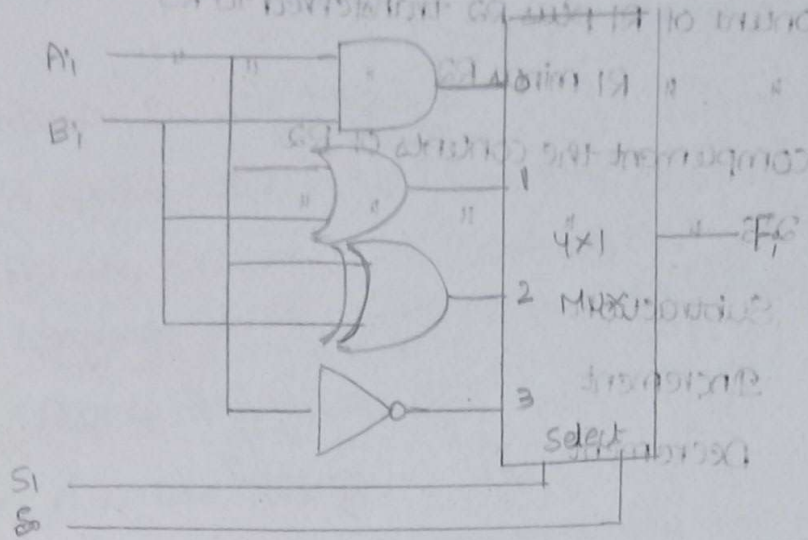
Summary of Typical Arithmetic Micro-operations:

- $R_3 \leftarrow R_1 + R_2$ — content of R1 plus R2 transferred to R3
- $R_3 \leftarrow R_1 - R_2$ — " " R1 minus R2 " " "
- $R_2 \leftarrow R_2'$ — complement the contents of R2
- $R_2 \leftarrow R_2' - 1$ — 2's " " " " "
- $R_3 \leftarrow R_1 + R_2' + 1$ — Subtraction
- $R_1 \leftarrow R_1 + 1$ — Increment
- $R_1 \leftarrow R_1 - 1$ — Decrement

Arithmetic Circuit:



Hardware implementation of logic micro operation:

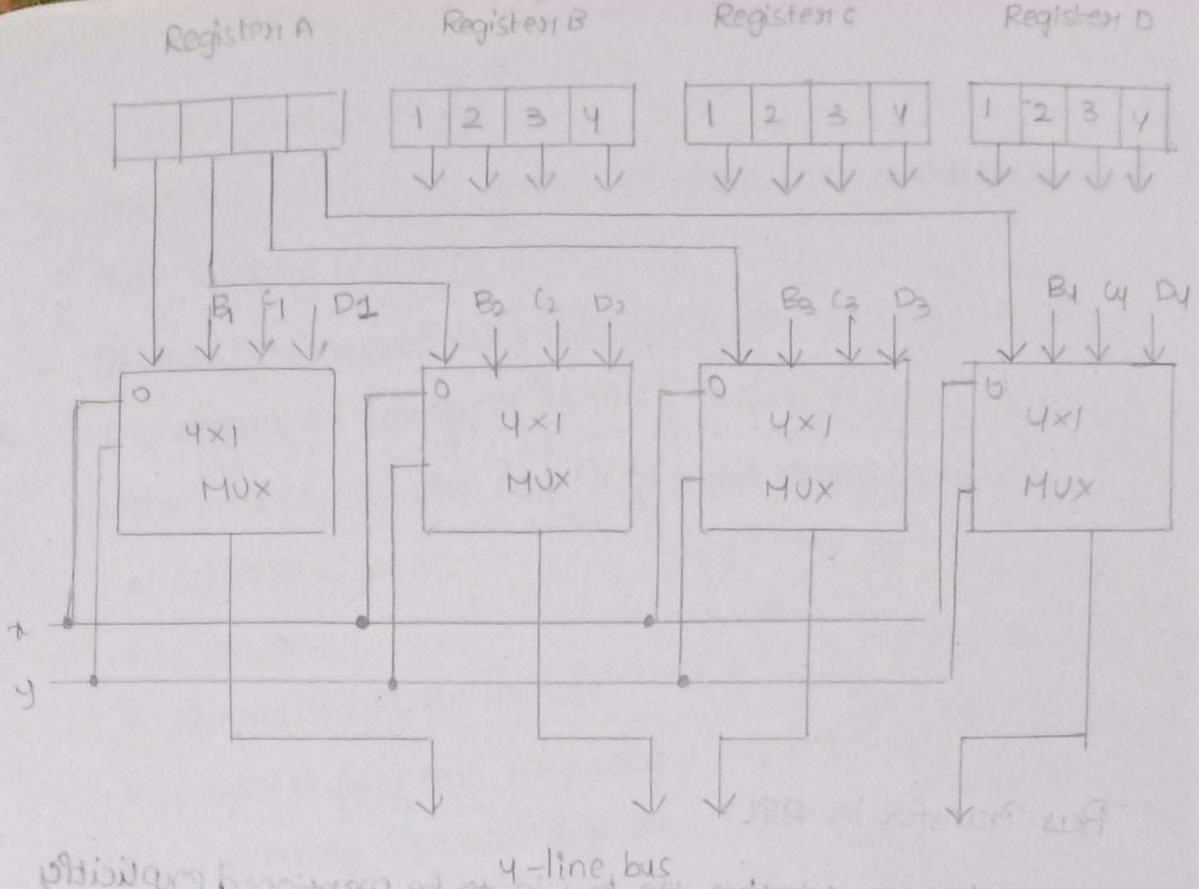


Basic Symbols for Register Transfers:

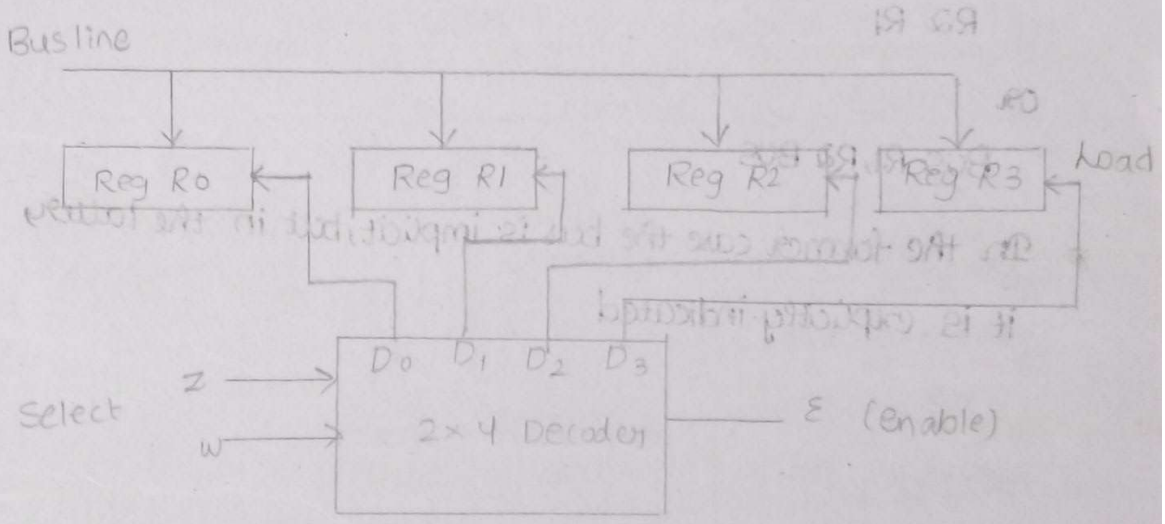
Symbols	Description	Examples
Capital letters & numerals	Denotes a register	MAR, R_2
Parenthesis	Denotes a part of a register	R2(0-7), R2(4)
Arrow	Denotes transfer of information	
Colon:	Denotes termination of control function	
comma,	Separates two micro operations	

BUS AND Memory transfer:

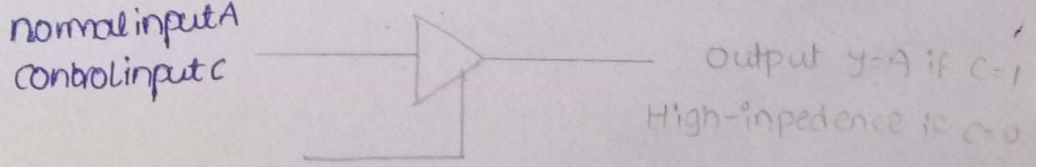
* Bus is a path (of a group of wires) over which information is transferred, from any of several sources to any of several destinations

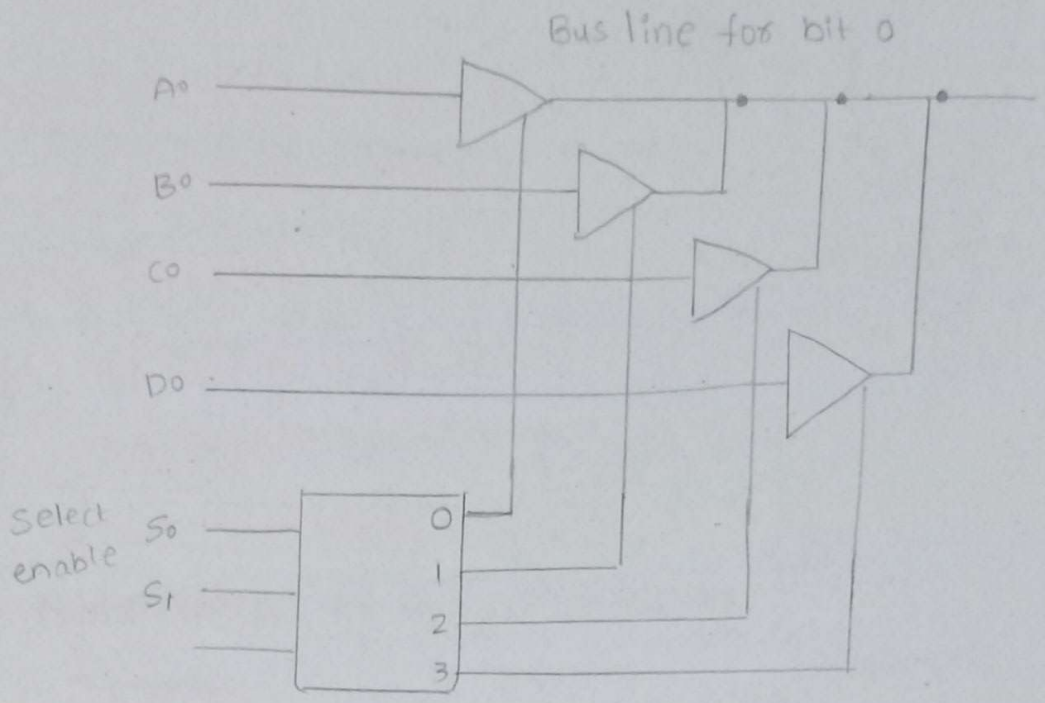


Transfer from Bus to destination register:



Three-States Bus Buffers:





Bus Transfer in RTL:

* Depending on whether the bus is to be mentioned explicitly or not, register transfer can be indicated as either

R2 R1

or

BUS R1, R2 BUS

* In the former case the bus is implicit, but in the latter it is explicitly indicated

27/06/22

The 8085 Microprocessor Architecture

The 8085 and its buses:

* The 8085 is an 8-bit general purpose microprocessor that can address 64k byte of memory
* It has 40 pins and uses +5V for power. It can run at maximum frequency of 3MHz
  - The pins on the chip can be grouped into 6 groups
    * Address Bus
    * Data bus
    * Control and Status Signals
    * Power Supply and frequency
    * Externally Initiated Signals
    * Serial I/O ports

8085 Microprocessor - Functional units:

Accumulator:

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU

Arithmetic and logic unit:

As the name suggests, it performs arithmetic and logical operation like Addition, Subtraction, AND, OR, etc on 8-bit data

General purpose register:

* There are 6 general purpose registers in 8085 processor, i.e., B, C, D, E, H & L. Each register can hold 8-bit data
* These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L

Program counter:

It is a 16-bit register used to store the memory address location of the next instruction to be executed

Stack pointer:

It is also a 16-bit register works like stack, which is always incremented / decremented by 2 during push & pop operations

Temporary register:

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations

Flag register:

It is an 8-bit register having five 1-bit flip flops, which holds either 0 or 1 depending upon the result stored in the accumulator

⇒ These are the set of 5 flip-flops

* Sign(S)
* zero(Z)
* Auxiliary carry(AC)
* parity(P)
* carry(C)

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

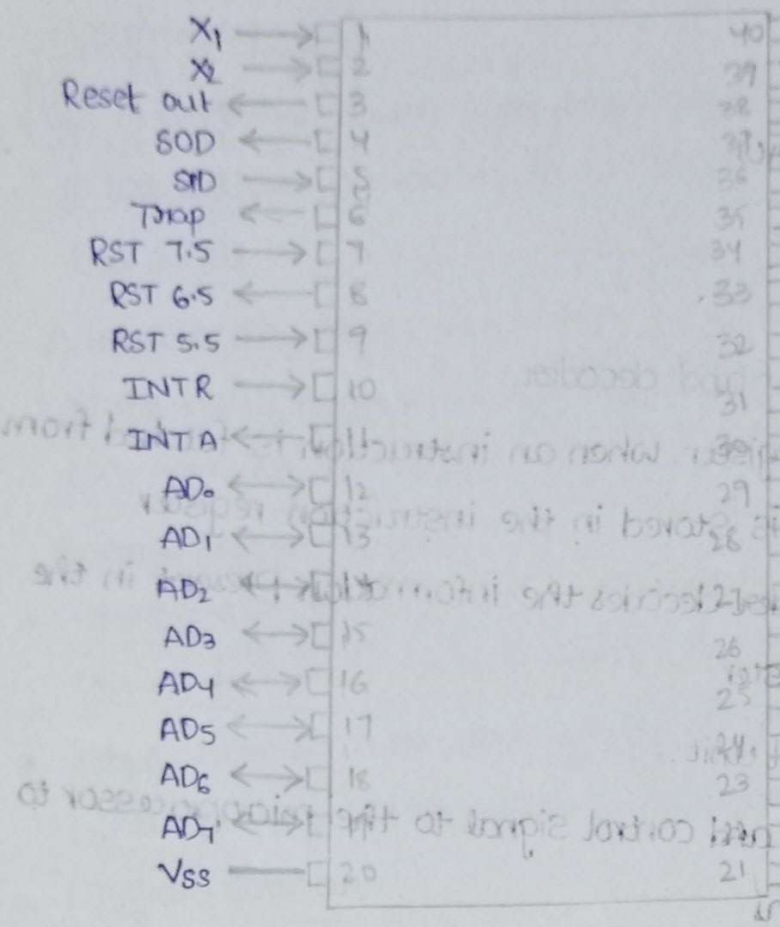
Instruction register and decoder:

* It is an 8-bit register. when an instruction is fetched from memory then it is stored in the instruction register
* Instruction decoder decodes the information present in the Instruction register

Timing and control unit:

* It provides timing and control signal to the microprocessor to perform operations
* Control signals: READY, RD', WR', ALE
* Status signals: S0, S1, IO/M'
* DMA signals: HOLD, HLDA
* RESET signals: RESET IN, RESET OUT
  - RESET IN ⇓ when RESET IN = 0, PC = 0 and microprocessor moves into reset state
  - RESET OUT ⇓ It is used to reset the condition of MPU and even used to reset other devices

Microprocessor pin Diagram:



Address/Data bus: The address/data bus pins are from A0-A15. These signals are unidirectional where it sends signals only from the microprocessor to the peripheral devices

Status & control Signals: Based on the pin combination of IO/M and S0, S1, the data bus status is decided

ALE: It is Address latch Enable Signal. The data bus moves into an active state when ALE = 1. The lower 8 bits are enabled during the initial T state of the machine cycle

S0, S1: These are called Status Signals where the 4 combinations of these pins determine the operations of writing, reading, halt and fetching

* IO/M S0 S1 Status of Data bus
* L H H opcode fetch
* L H L Memory read
* L L H Memory write

* H H L I/O read
* H L H I/O write
* H H H interrupt acknowledge
* L L L Halt

Clock frequency and power supply: The pins are described as

* VCC - Need a +5V powers supply
* VSS - pin for ground reference
* CLK(OUT) - It is used as a clock signal to refer to other devices
* X_1, X_2 - The frequency is divided b/w these two pins. To execute the system at 3MHz, it should have a total frequency of 6MHz

Serial I/O ports:

8085 performs serial transmission through SOD and SID signals. SID is the input data line while SOD is the output data line

Interrupts and peripheral signals

To interrupt a program in 8085, it uses five signals and those are:

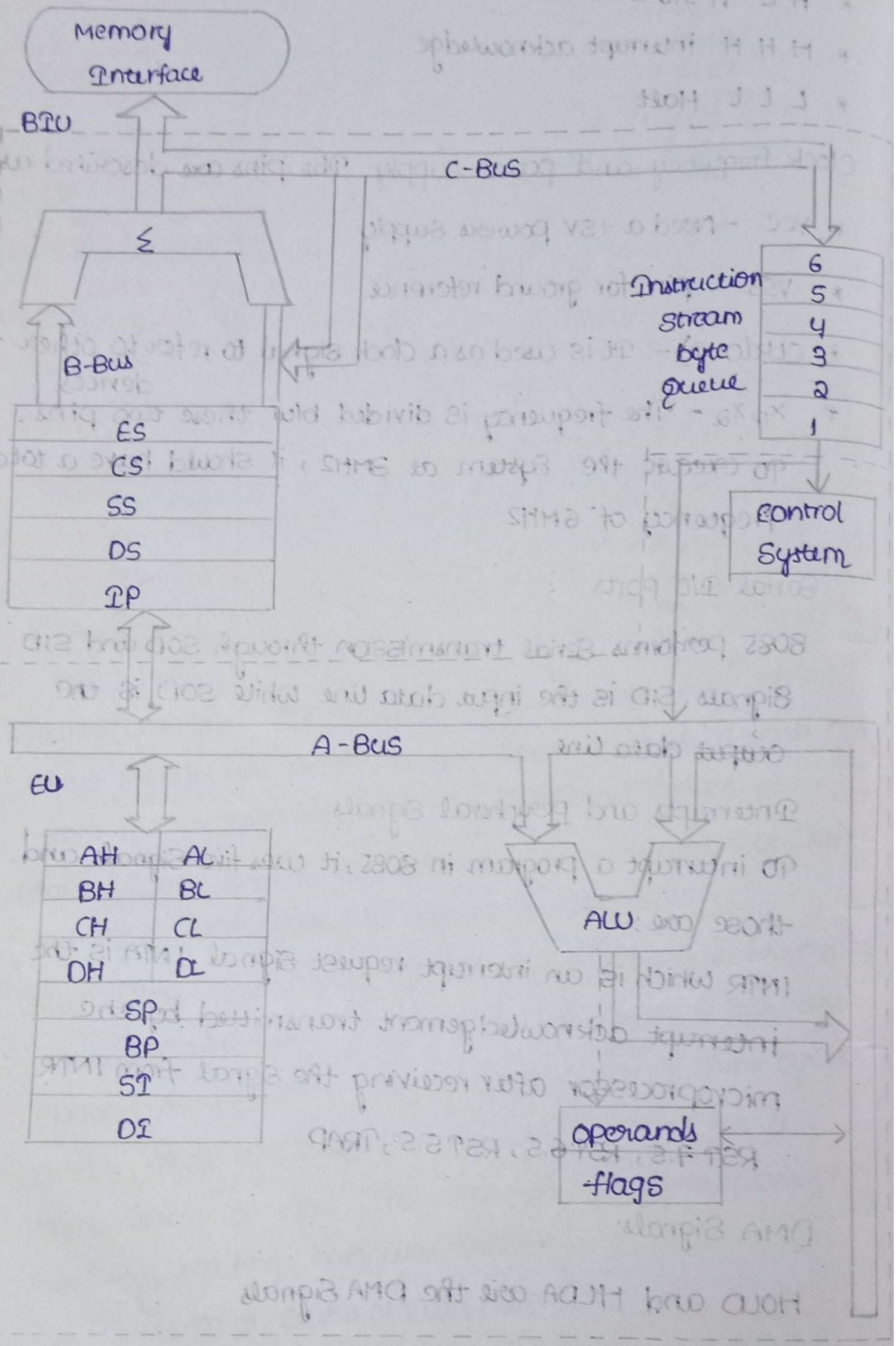
INTR which is an interrupt request signal. INTA is the interrupt acknowledgement transmitted by the microprocessor after receiving the signal from INTR

RST 7.5, RST 6.5, RST 5.5, TRAP

DMA Signals:

HOLD and HLDA are the DMA signals

8086 Architecture:



8086 16 bit flag register:

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Carry flag: (CF)

Set by carry out of MSB

parity flag: (PF)

Set if result has even parity

Auxiliary carry flag: (AF)

Set by carry out of MSB for BCD

zero flag: (ZF)

Set if result is equal to zero

Sign flag: (SF)

MSB of result

Sing step trap flag: (TF)

If program is executed step by step it is set to 'one'

Interrupt enable flag: (IF)

If it sets to one, interrupts are enable

If it sets to zero, interrupts are disable

String Direction flag: (DF)

It is based on direction of the string. set if dir of str is

ltr Forward-one, backward-zero

Overflow flag: (OF)

If any overflow is there then we set 'one' otherwise it is 'zero'

1kg = 1024 bytes = 2^{10}

1024k = 1MB = 2^{20}

1024M = 1GB = 2^{30}

1024G = 1TB = 2^{40}

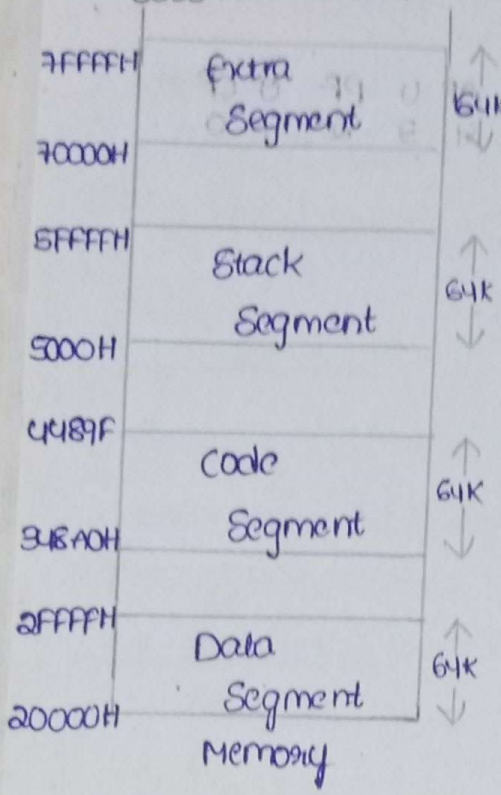
64k = 64 × 1024 = 65536

= FFFF (Hexa)

Size of address bus = 20 bits

Size of data bus = 16 bits.

8086 Architecture:



(ES) Extra segment register: It holds base address of the extra Segment [7000]

(CS) Code segment register: It holds base address of code Segment [348A]

(DS) Data Segment register: It holds base address of data Segment [2000]

(SS) Stack Segment register: It holds base difference address of Stack Segment (5000)

(IP) Instruction pointer: Stores offsets of code segment [0-FFFF]

CS: IP

348A : 12FB → (In Range [0, FFFF])

Offset

348A0 → hardwired zero

12FB

35B9B → Effective address for CS

SP → Stack pointer → holds offsets related to stack segment

SS: SP

5000: F321

50000

F321

5F321

BP → Base pointer → holds offsets related to data segment or extra segment

DS: BP

ES: BP

Ex: 2000: FF32F → Not valid offset

DS: BP

SI → Source Index } For arrays

DI → Destination Index }

In 8086 architecture, memory is divided into four segments

* Data segment and extra segment are used to store user defined data
* Code segment is used to store programs
* Stack segment is used to store parameters, return address
- es, context related to programs etc

8086 Architecture is divided into two units

1. Bus interface unit (BIU)
2. Addressing Modes

Bus Interface Unit (BIU):

It interfaces with external bus to get data from memory or to write data into memory. It contains different segment registers like extra segment register, code segment register, data segment register, stack segment register and Instruction pointer [ES, CS, DS, SS, IP]. The instructions fetched from memory are placed in instruction stream byte queue from where it is given to control system. control system decodes the instructions and gives necessary signals to A-bus in execution unit to perform operations, which are necessary flags in execution unit are set/reset based on the result of the recently executed instruction. user-defined variables are stored in operands

8086- Pin configuration:-

GND: Larger current to the ground

AD14-AD0, AD15: These lines are multiplexed lines

INTR: It's not user controlled, we can enable/disable

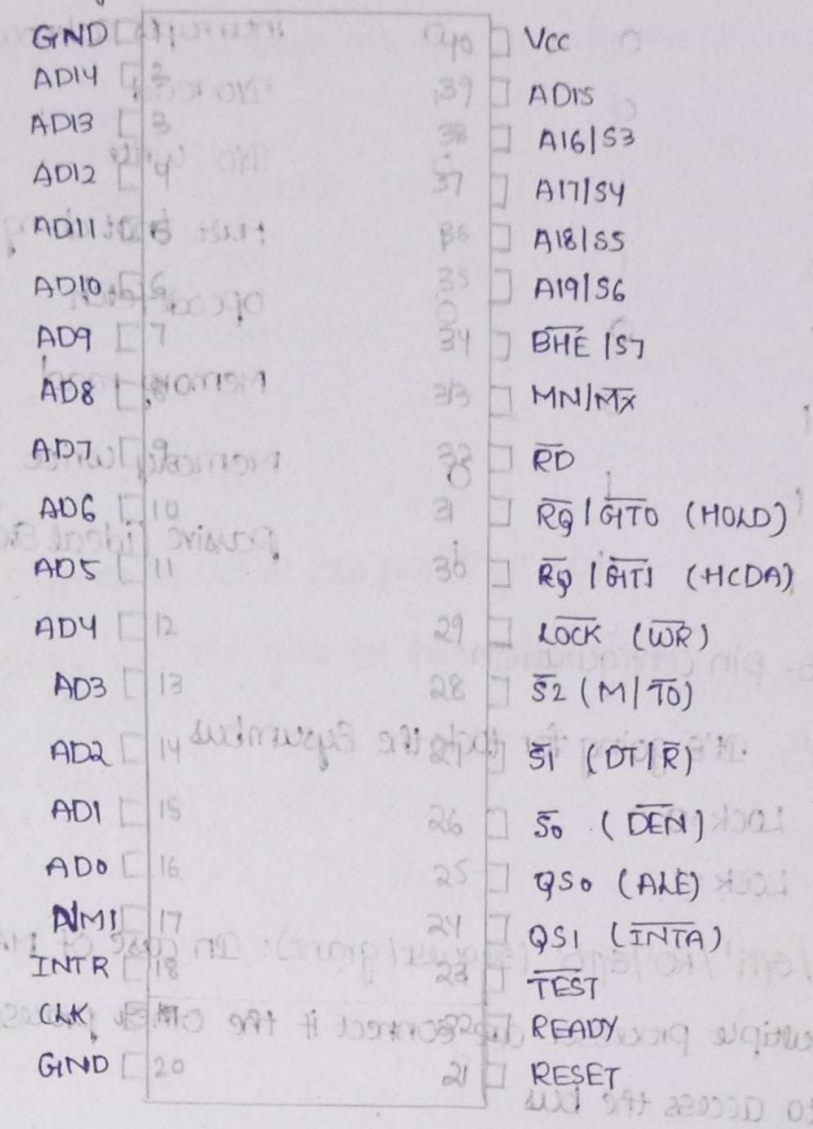
NMI: It's not user controlled, it's reach processor when interrupts generate once

CLK: It's signal used for synchronization b/w processor and all components

READY: active high pin (logic 1), used by the external peripheral

- READY=0 (ready for accept data)
- READY=1 (not ready for accept data)

8086 pin diagram:



TEST: This pin is used along with wait instruction

QS1 & QS2: It gives the status of instructions queue

QS1	QS0	Meaning
0	0	No operation
0	1	First byte of opcode from queue
1	0	queue is cleared
1	1	next byte of opcode from queue

* S0', S1', S2': This indicates this type of operations carried out by the processor.

$S0'$	$S1'$	$S2'$	Function
0	0	0	interrupts acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt (not doing anything)
1	0	0	opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	passive (ideal state)

8086- pin configuration:

* Lock': It's going for lock the Systumbus
  - Lock=0
  - Lock=1
* $RT'/GT1'/RQ'/GT0'$ (request/grant): In case of MAX mode if multiple processor are connect if the other processor wants to access the bus
* MN/MX': Selecting min mode /max mode
  - pin connected to Vcc/programmed to 1 → MIN MODE
  - pin connected to GROUND → MAX MODE
* BHE'/S7 : bus high enable
  - It's representing what type of data processor accessing
  - BHE A0 INDICATION
  - 0 0 whole word (64k)
  - 0 1 upper data / from odd. bank
  - 1 0 lower data/ from even bank
  - 1 1 none
* A19/S6 - A16/S3: address/Status bus multiplexed
  - S6 : always remains logic 0

- S5: indicate condition of FLAG bits

S4, S3: represent which segment is currently the processor accessing

S4	S3	Segment
0	0	ES
0	1	SS
1	0	CS
1	1	DS

RD': read signals is active low pin (logic = 0)

VCC: MAX voltage we can give at these is +5.0V