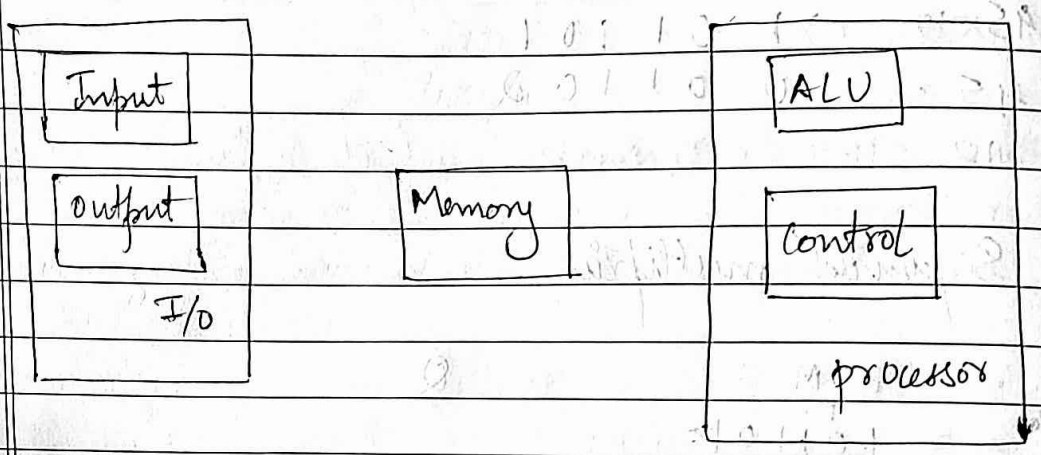
Computer Organisation & Architecture

PageWork
D D M M Y Y Y Y

Functional units :-



Computer Organisation is concerned with structural and behaviour of the computer system as seen by the user.

Computer Architecture is concerned with the way hardware components are connected together to form a computer system. It acts as an interface between the hardware & software.

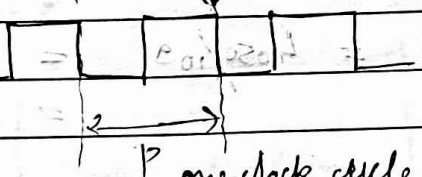
Computer Organisation is a low level design whereas Computer Architecture is a high level design.

RISC

PageWork
D D M M Y Y Y Y

Clock: defines regular time intervals called clock cycles.

P: length of one clock cycle.



Clock Rate:- The no. of clock cycles per second.
(R)

$$R = \frac{1}{P}$$

CPU speed / clock rate; is measured in Hertz.
clock

Basic performance Equation:-

$$T = \frac{N \times S}{R}$$

T: processor time required to execute a program.
N: Number of instructions.
R: clock rate
S: Number of basic instructions required to perform 1 instruction

Q) A program contain 1000 instruction out of that 25% instruction requires 4 clock cycles. 40% instruction require 5 clock cycle for execution. Find the total time required to execute the program running in a 16 Hz machine.

Soln

N = 1000

S = 25% of N = 250 instructions requires 4 clock cycles
40% of N = 400 instructions requires 5 clock cycles.
35% of N = 350 instructions require 3 clock cycles.

$R = 1\ GHz = 10^9\ Hz$

$$T = \frac{N \times S}{R} = \frac{(250 \times 4) + (400 \times 5) + (350 \times 3)}{10^9}$$

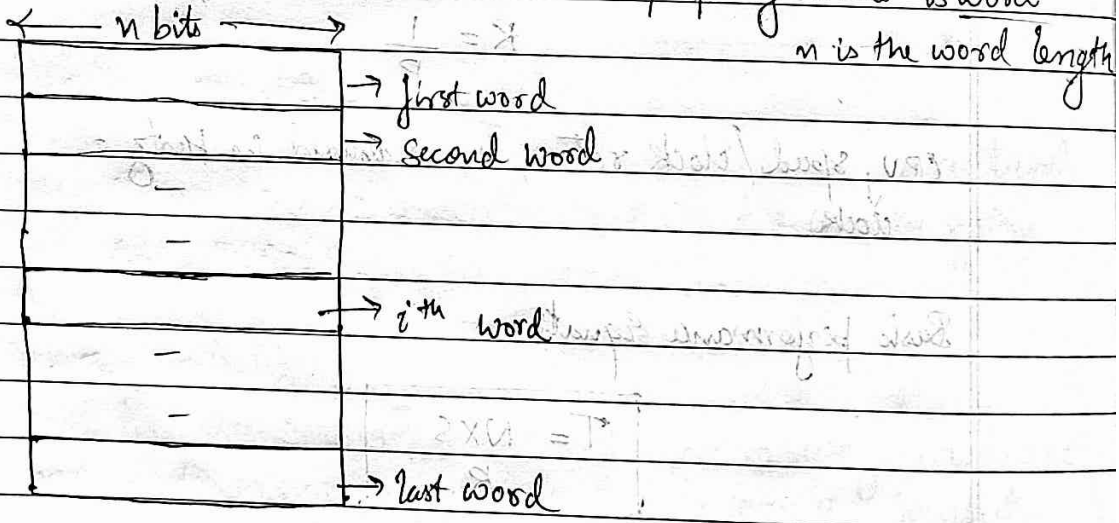
$$= 4050/10^9 = 4.05 \times 10^{-6}$$

$$= 4.05\ \mu s$$

Memory location and Addresses

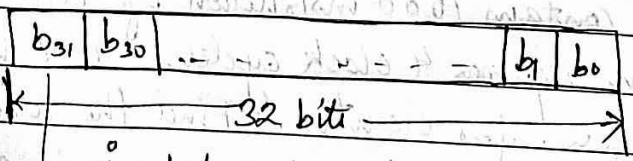
Group of n bits is word

n is the word length



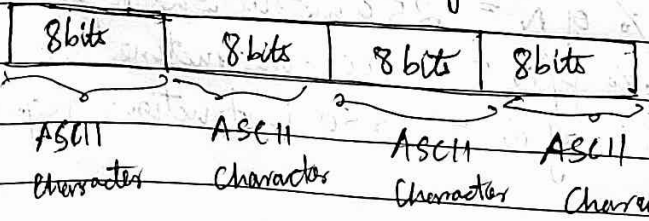
Memory words

If the word length is 32 bits, single word can store 32 bits or 4 ASCII characters.



sign bit : $b_{31} = 0$ for +ve numbers

$b_{31} = 1$ for -ve numbers.



Bus → 1) Address Bus 3) Control Bus
2) Data Bus

For storing 12345678 :- MSB LSB

⇒ Big - Endian | 12 | 34 | 56 | 78 |

⇒ little - Endian 78 56 34 12

0	0	1	2	3
4	4	5	6	7
		⋮		
2^K-4				

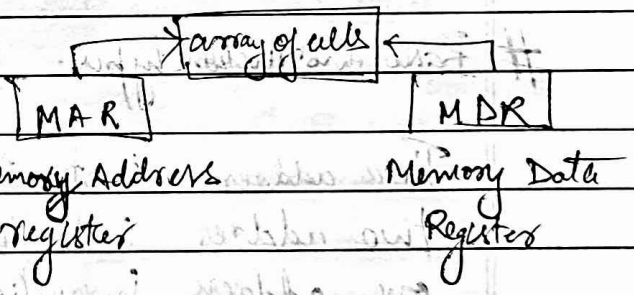
Big - Endian assignment

3	2	1	0	
7	6	5	4	
		⋮		
2^K-4				

Little - Endian assignment.

Memory Operations

1) Load (or Read or Fetch)
2) Store (or Write)

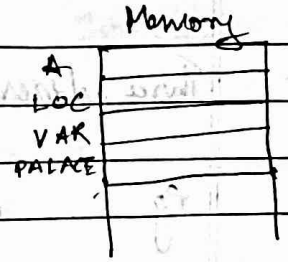


Instruction and Instruction Sequencing

Data Transfer :-

Memory location is given symbolic name.

⇒ Name for memory address location.
A, LOC, VAR, PLACE



⇒ Name for Processor register names.
R1, R2, R3, ---

⇒ Name for I/O register names.
DATAIN, DATAOUT

⇒ Register Transfer Notation (RTN)
⇒ Assembly Language Notation.

RTN :

$R_3 \leftarrow [R1] + [R2]$

$R1 \leftarrow [LOC]$

Processor
$R_1 = 10$
$R_2 = 20$
$R_3 = 30$

Assembly language Notation :-

Move LOC, R1
Add R1, R2, R3

Basic Instruction types.

Three address instruction format : operation Source 1, Source 2, Destination
two address instruction format
one address instruction format
zero address instruction format.

Label : operation Operands.

Three address :- operation Source 1, Source 2, Destination.

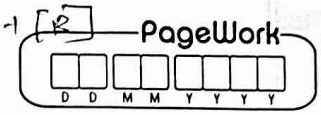
eg :- Add A, B, C

Add the contents of A and B then store the results in C.

Add A, B

① ⇒ RTN :- [B] ← [A] + [B]

② ⇒ [C] ← [B]



Two address :- Operation Source, Destination

eg :- 1) Add A, B → loses it's original value.

add the contents at A and B then store the result in B.

2) ~~Add A, B~~ ⇒ retains it's original value ← Move B, C
Move B, C Add A, C

One address :- Operation Operand.

↳ can be source or destination depending on the type of instruction

eg :- 1) Add A

↳ Second operand is taken from the accumulator and store the result in the accumulator

Acc = 20
20 + 10 = 30
processor

A	10

2)

Load (A) ⟶ Source Operand

Store (A) ⟶ destination Operand

Examples :- Add R_i, R_j or

Add R_i, R_j, R_k

↳ directly using the processor register

Move Source, Destination.

Eg:- Move R1, R2

2) Move A, R1 } Both are same instruction.
Load A, R1 }

Move R1, A } Both are same.
Store R1, A }

3) Evaluate (A+B) * (C+D)

Add A, B, R1
Add C, D, R2
Mul R1, R2, E } using 3 address instruction format.

Add A, B
Move B, R1
Add C, D
Move D, R1
Mul R1, R2
Move R2, E } using 2 address instruction format.

PageWork
D D M M Y Y Y Y

Instruction and Instruction Sequencing :-

Straight line Sequencing :- To begin executing a program, the address of the first instruction will be placed in program counter, then processor control circuits will use the information in PC to fetch and execute the instruction one at a time. in the order of increasing addresses. This is called straight line sequencing.

Q) Write a program for $C \leftarrow [A] + [B]$:

	Address	Contents
MOV A, R0	i	Move A, R0
ADD B, R0	i+4	ADD B, R0
MOV R0, C	i+8	Move R0, C

Assume.

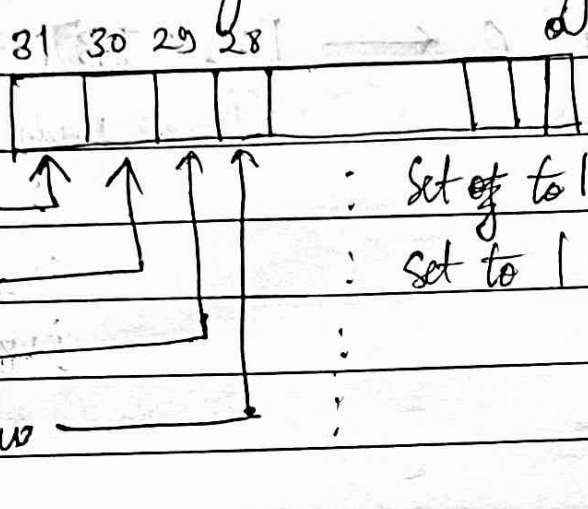
Memory word length is 32 bits
Memory is byte addressable
Three instructions are stored in successive word locations, starting at location i.
Each instruction is 4 bytes long, i+4, i+8

Instruction Execution :-

Executing a given instruction is a two phase procedure

1) Instruction Fetch
2) Instruction Execute

CPSR ⇒ Current Program Status Register.



Condition Code flags

Addressing modes:-

Different ways in which the location of an operand is specified is called addressing modes.

different addressing modes:-

1) Register mode
2) Absolute mode
} → Two Addressing modes to access variables

3) Immediate mode
4) Indirect mode
5) Index mode
6) Base with Index mode
7) Base with Index and Offset mode
} To deal with lists and array

8) Relative Mode
9) Auto increment mode
10) Autodecrement mode

PageWork
D D M M Y Y Y Y

1) ~~Two~~ Register mode:-

The operand is the content of a processor register, the name (address) of a register is given in the instruction.

eg: Move LOC, R1 ~~R0~~
Move R1, R2 → uses processor register as operand.

2) Absolute mode (Direct mode):-

The operand is in memory location, the address of memory location is given explicitly in the instruction.

eg: Move A, B..
→ Address of memory location.
Add R1, A

3) Immediate mode:- Addressing mode ~~to~~ represent constants.

Add #100, R2 : Add instruction, adds 100 to R2's content.
↳ constant value

If a value does not have # sign in front of it then it is the address of a memory location.

The # sign in front of the value indicates the immediate value ~~of~~ to be operated.

Store R2, 100H
↳ ~~it~~ it does not have # sign hence Store will consider it as address. and store the contents of R2 into the memory location 100H.

D D M M Y Y Y Y

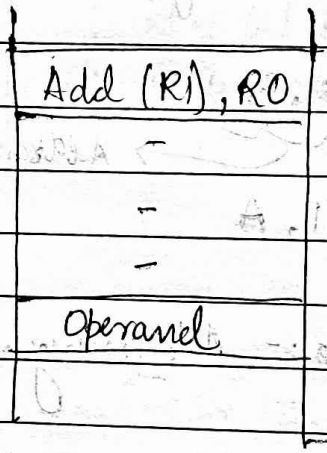
3) A = B + 6

Move B, R1 :- Register addressing mode
Add #6, R1 :- Immediate
Move R1, A :- Absolute or direct

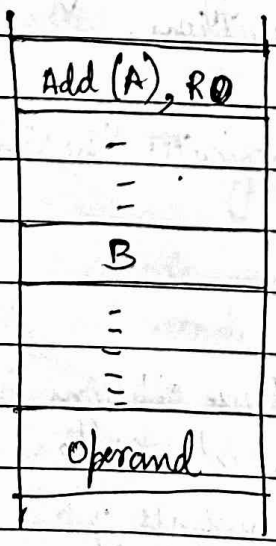
4) Indirect mode

Add (R1), R0

Main memory :-



Register R1 | B |



Add (A), R0

PageWork
D D M M Y Y Y Y

8) Adding a list of Numbers:-

Address	Contents		
	Move	N, R1	} initialization
	Move	#NUM1, R2	
	Clear	R0	
→ LOOP	Add	(R2), R0	
	Add	#4, R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0, SUM	

=> Addressing modes to deal with lists and array

5) Index mode
6) Base with Index mode
7) Base with Index and offset mode.

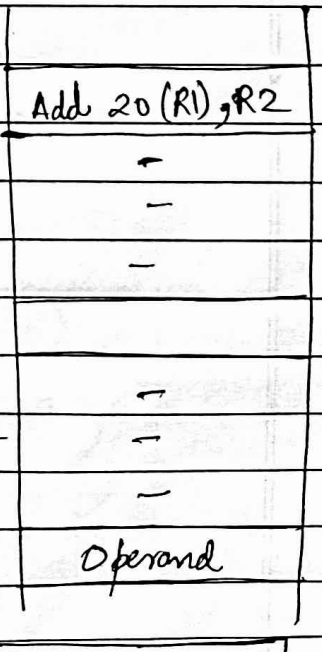
5) Index Mode.

Add 20(R1), R2

EA = X + [Ri]

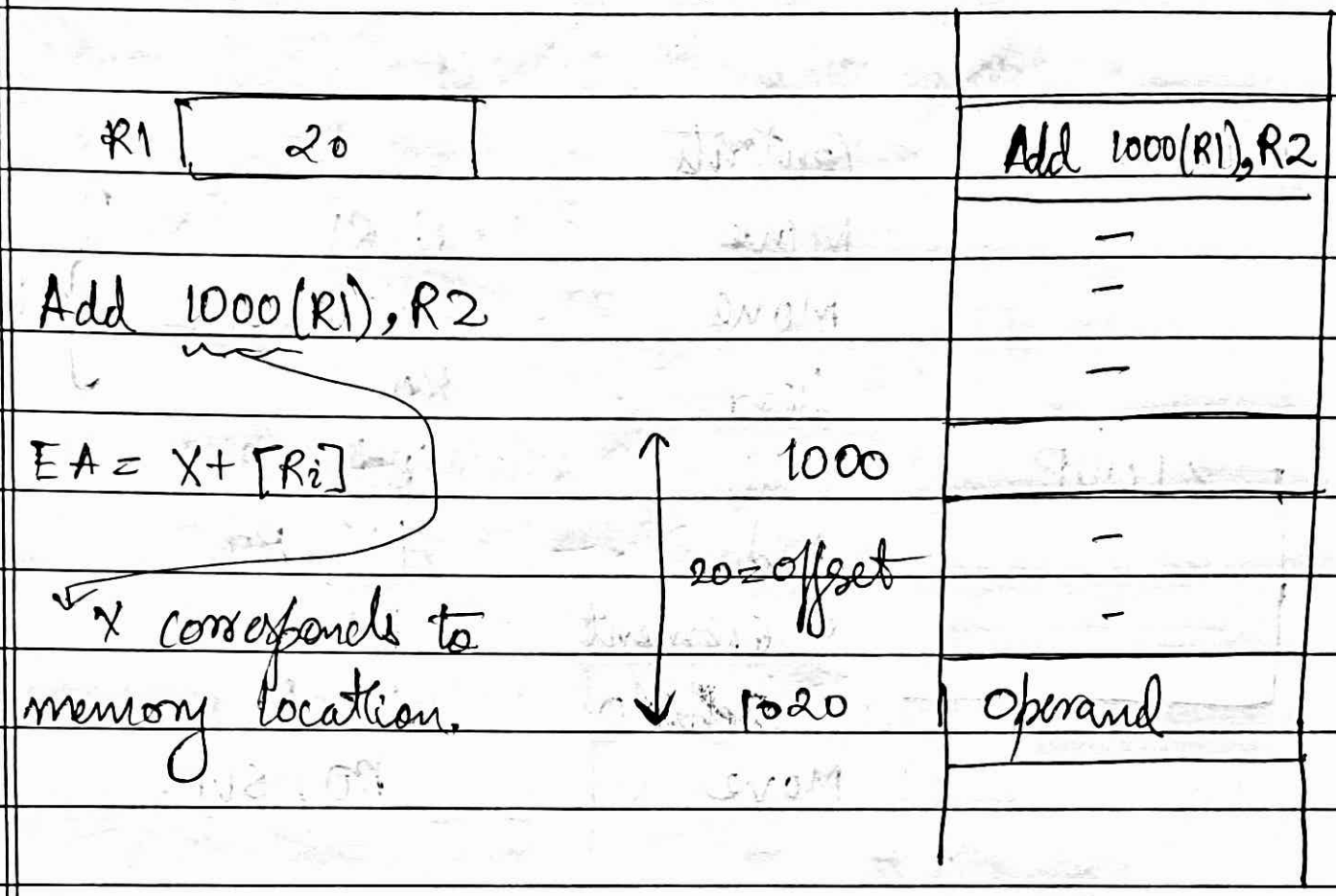
X defines the offset (displacement)

Operand at 1020 will be added to R2.

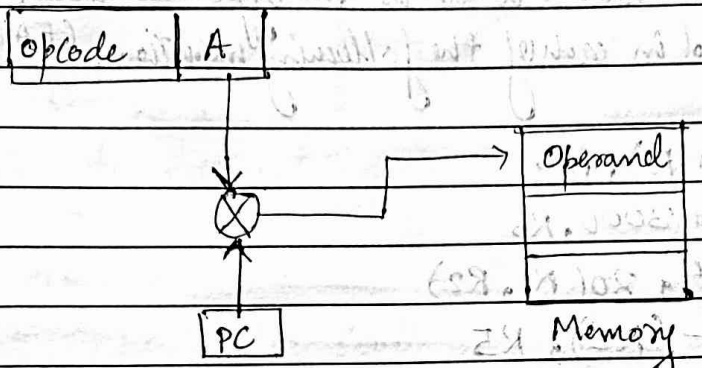


D D M M Y Y Y Y

Offset is in index ~~reg~~ register:-



8) Relative mode.



Common use to specify target address in Branch instruction.

Addressing modes for accessing data items in successive locations in memory.

9. Auto increment mode ⇒ Move R0, (R2)+

10. Auto decrement mode

Processor	Memory address	Memory location
R0 = 5	→ LOC (1000)	5
R2 = LOC (1000)	LOC (1001)	
LOC (1001)		

Auto decrement mode :-

Move -(R0), R1

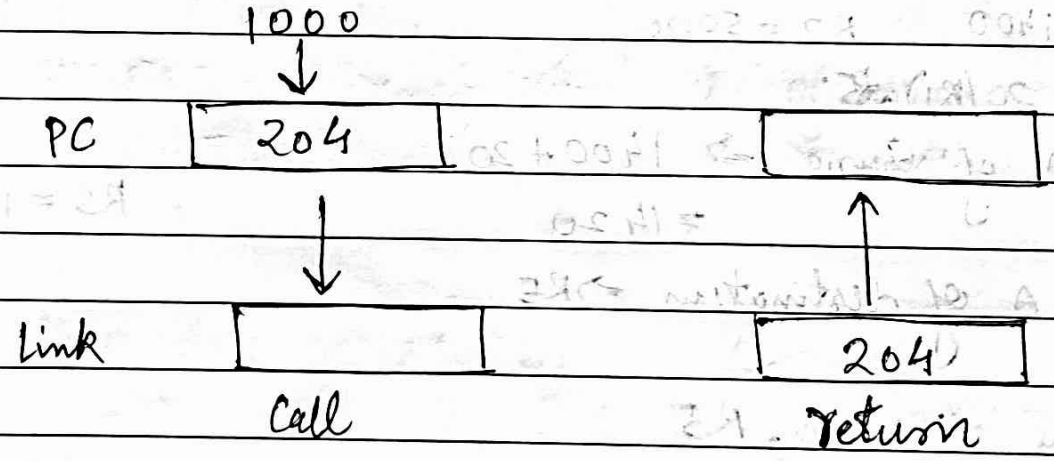
R0 = Loc (1000)	LOC (1000)	35
↙		
Loc (4000)	LOC (1001)	
R1 = 35 ←		

Subroutines

Memory location	Calling Program	Memory location	Subroutine SUB
	:		
	:		
200	call SUB ——→	1000	first instruction
204	next instruction ←——		:
			:
			return

The way in which a computer makes it possible to call and return from subroutines is referred to as its subroutine linkage method.

Link register : saves the return address in a specific location



Nested Subroutine uses Stack.

PageWork

D D M M Y Y Y Y

Stack Frame :-

SP → (Stack Pointer) points to top of the Stack	Saved[R0]	Stack frame for called Subroutine.
	Saved[R1]	
	local var 3	
	local var2	
FP → (Frame Pointer) points to the current active frame	Saved[FP]	
	return addrs	
	local var1	
	local var2	
		← old TOS

Encoding of Machine instruction

Encoding instruction into 32 bit word :-

← 8 bits →	7 bits	7 bits	7 bits	3 bits
Opcode	Source	Destination	Addressing mode	other info

Operation code ↑ operands.

a) One word instruction :- ⟶ RISC

8	7	7	10
opcode	source	dest	other info

b) Two word instruction.-

opcode	source	dest	other info
Memory address / Immediate operand			

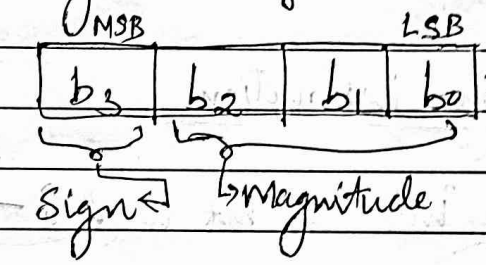
Three operand instruction:-

Opcode	Ri	Rj	Rk	Other info

Number representation

→ Sign and magnitude
→ One's complement
→ Two's complement

Sign and magnitude system



if MSB = 0 then the number is positive
" = 1 " " " " " negative

eg:-

	b_3	b_2	b_1	b_0	
+0	0	0	0	0	2 representation of zero
−0	1	0	0	0	
+6	0	1	1	0	
−6	1	1	1	0	
+7	0	1	1	1	
−7	1	1	1	1	

1's Complement

	b_3	b_2	b_1	b_0
+7	0	1	1	1
−7	1	0	0	0
+0	0	0	0	0
−0	1	1	1	1

Here also we have 2 different representation for zero

+1	0	0	0	1
−1	1	1	1	0

2's Complement system:

2's Complement = 1's complement + 1

+2 → 0 0 1 0

1's 1 1 0 1

\+ 1

2's 1 1 1 0 → −2

+5 → 0 1 0 1

1's 1 0 1 0

\+ 1

2's 1 0 1 1 → −5

+7 → 0 1 1 1

1's 1 0 0 0

\+ 1

2's 1 0 0 1 → −7

```
  1
  1
-----
 [1] 0
  ↳ Carry out
```

+0 → 0 0 0 0

1's 1 1 1 1

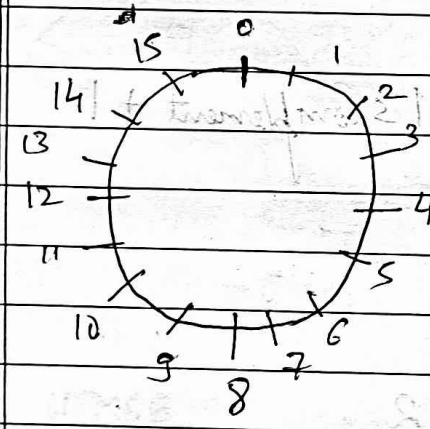
1

2's 0 0 0 0 ⟶ -0

∴ So, here zero has only one unique representation.

2's complement arithmetic

N = 16



(a+b) mod N

(7+4) mod 16

= 11

∴ Locate 7 then move 4 units in the clock wise direction to arrive at the answer 11.

a)

+2	0 0 1 0	
+3	0 0 1 1	
+5	0 1 0 1	

+4	0 1 0 0	→ written in 2's
-6	1 0 1 0	→ " "
-2	1 1 1 0	→ " "

Subtraction Rule:- Two numbers X and Y (that is to perform X-Y); take the 2's complement of Y and add it to X.

eg: X -3 1101
Y -7 1001

2's complement of -7 ⇒ 1001
0110
1
0111

1 1 1 1
-3 ⇒ 1101
2's complement of -7 ⇒ 0111
X-Y ⇒ 0100 ⟶ +4

eg X +6 0110
Y +3 0011

2's complement of +3 ⇒ 0011
1100
1
1101

(X-Y) ⇒ 0110
1101
0011 ⟶ +3

Overflow in integer arithmetic

(+7) + (+4)

X +7 0111
Y +4 0100
S 1011 ⟶ -5 overflow has occured

Illy if sum of two -ve numbers yield a +ve number then

PageWork

D D M M Y Y Y Y

Sum has overflowed.

$(-5) + (-6)$

X	-5	1	0	1	1
Y	-6	1	0	1	0
S		0	1	0	1

$\longrightarrow +5$ overflow has occured.

$(+7) + (-4)$

		1	1		
X	$+7$	0	1	1	1
Y	-4	1	1	0	0
S		0	0	1	1

$\longrightarrow +3$ No overflow.

UNIT-2

Arithmetic Unit & Processor Unit

Multification of unsigned numbers:-

Q) Find the product multiplicand (M) 13 and multiplier (Q) 11

M X Q = ?

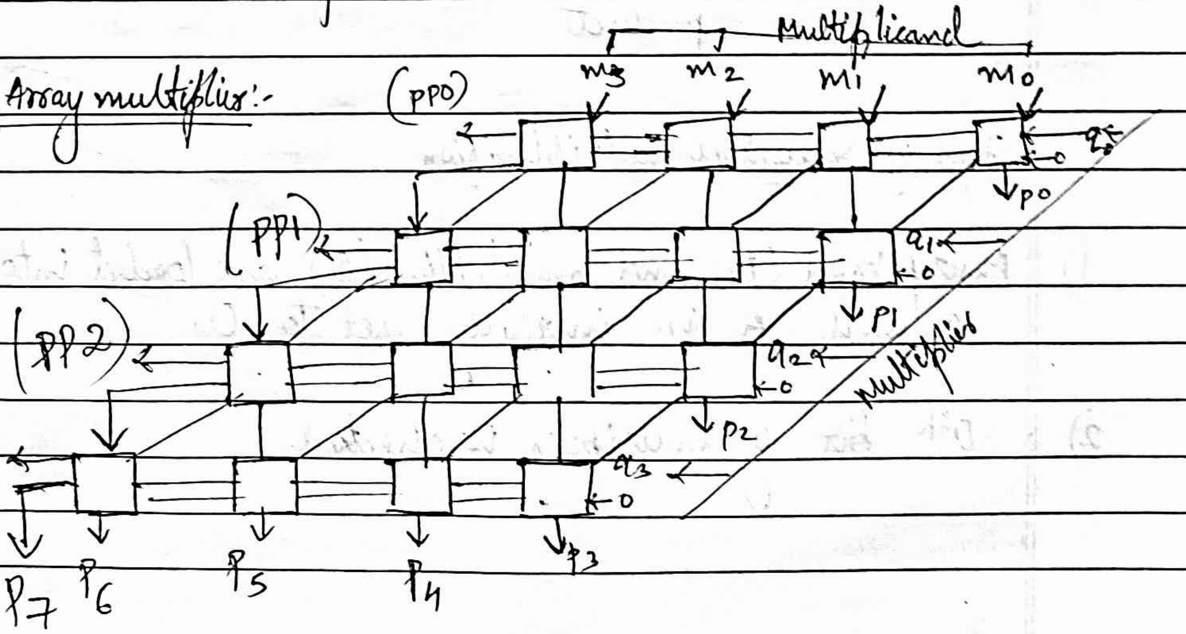
```
M     1 1 0 1      (13) Multiplicand
Q     1 0 1 1      (11) Multiplier
      -------
      1 1 0 1
    1 1 0 1
  0 0 0 0
1 1 0 1
---------------
1 0 0 0 1 1 1 1    (143)
```

```
2 | 13
2 |  6   1
2 |  3   0
     1   1
    1101
```

Implementation of Multiplication

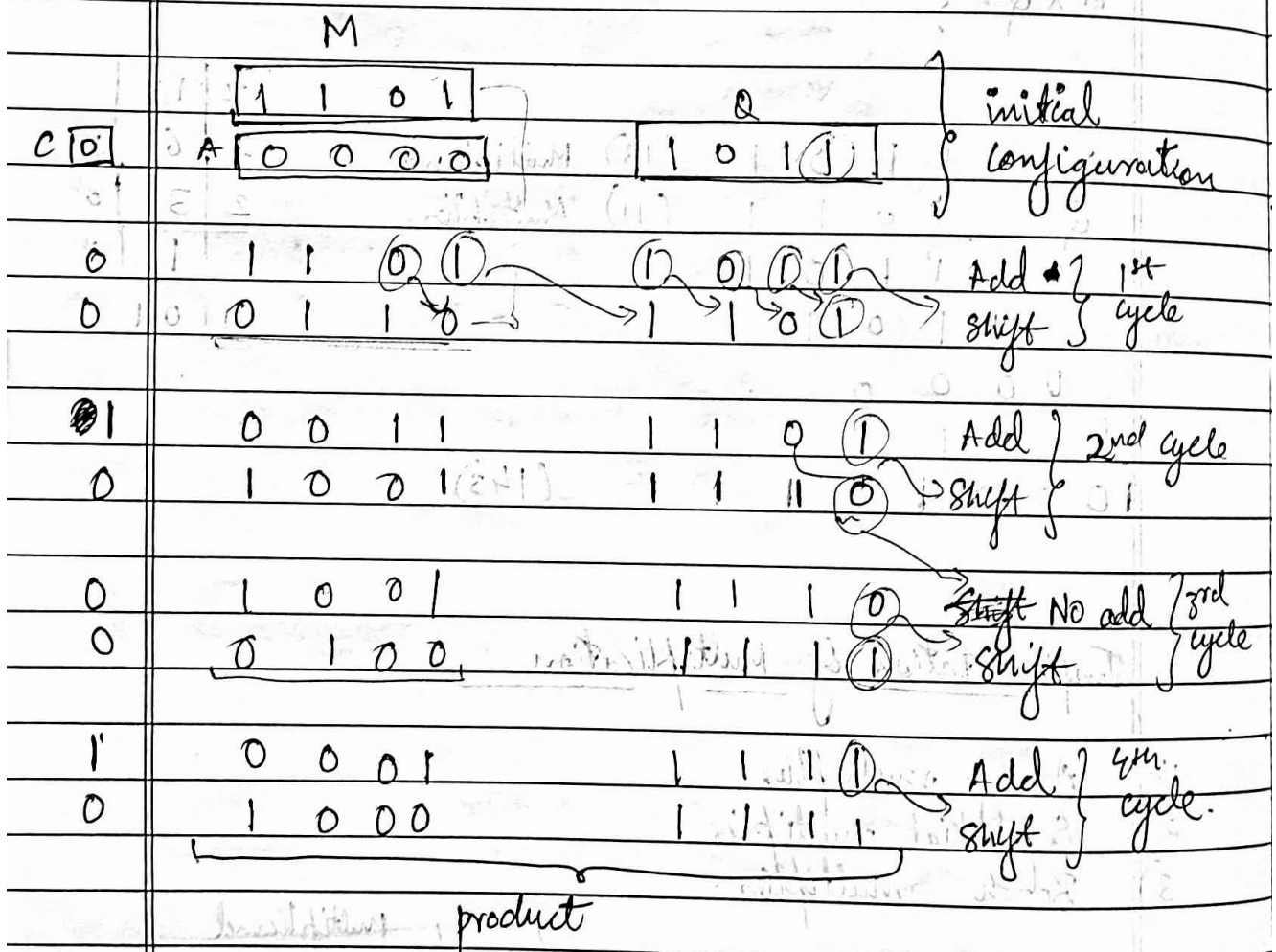
1) Array multiplier
2) Sequential multiplier
3) Booth multiplier.

Array multiplier:-



Q) Find the product of Multiplicand 13 and multiplier 11 using sequential multiplication.

1 1 0 1 M (13)
1 0 1 1 Q (11)
1 0 0 0 1 1 1 1 P (143)



Steps of sequential multiplication

1) Multiplicand (M) and multiplier (Q) are loaded into registers C and A are initially set to 0.

2) 0th Bit of multiplier is checked

3) If bit 0 = 1
Multiplicand and partial product is added and all the bits of Q, A_n and C registers are shifted to right one bit.

4) If bit 0 = 0
No addition is performed, only shift operation is carried out.

Step 2, 3, 4 are repeated till the answer is reached.

Signed Multiplication

Case 1) Multiplicand is (-ve) and Multiplier is (+ve).

e.g
1 0 0 1 1 (-13)
0 1 0 1 1 (+11)

1 1 1 1 1 1 0 0 1 1
1 1 1 0 1 1 0 1 1 1
0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 1 1
0 0 0 0 0 0 0
1 1 0 1 1 1 0 0 0 1 (+143)

Sign extension is written in black

+13 = 1 1 0 1
0 0 1 0
1
-13 1 0 0 1 1

PageWork
D D M M Y Y Y Y

Booth Algorithm

Booth multiplier recoding table :-

Multiplies		Version of multiplicand
Bit i	Bit i-1	selected by bit i
0	0	0 × M
0	1	+1 × M
1	0	-1 × M
1	1	0 × M

Eg:- 1) Multiplier is scanned from right to left

1 1 0 1 0 [0

0 -1 +1 -1 0

2) recode 1 0 1 1 0 0 [0

-1 +1 0 -1 0 0

3) Booth multiplication with negative numbers.

(+13) × (-6)

2's complement for (-6)

1101 (+13) ⟶ 1101

0110 (-6) ⟶ 1010

0110 Add sign bit ⇒ 01101 (+13)

01001 11010 (-6)

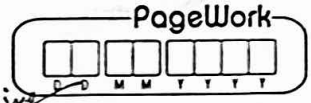
10010

recode the multiplier. 11010 [0

. 0 -1 +1 -1 0

1+1 = 10
1+1+1 = 11
1+1+1+1 = 100
1+1+1+1+1 = 101
1+1+1+1+1+1 = 110

Whenever we are multiplying with a negative then take 2's complement of multiplicand.



=>
```
    0 1 1 0 1
  x 0-1 1-1 0
  ----------
  0 0 0 0 0 0 0 0 0 0
  1 1 1 1 0 1 0 1 1
  0 0 0 1 1 0 1
  1 1 1 0 0 1 1
  0 0 0 0 0
  ----------
1 1 1 0 1 1 0 0 1 0   (-78)
```
ignore

2's complement of multiplicand.

```
0 1 1 0 1
1 0 0 1 0
        1
---------
1 0 0 1 1
```

6) M = -13
Q = -20

2's complement

01101 (-13) → 10011 (-13)
10100 (-20) → 01100 (-20)

```
01101    10100
10010    01011
    1        1
---------  -----
10011    01100
```

Add sign bit to both 0 1 1 0 1 (-13)
0 1 0 1 0 0 (-20)

recode the multiplier (Q): 1 0 1 0 0 [0
-1 +1 0 -1 0 0

=>
```
    1 1 0 0 1 1
  x -1 1 0 -1 0 0
  ---------------
  0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 1 1 0 1
  0 0 0 0 0 0 0 0 0 0
  1 1 1 0 0 1 1
  0 0 0 1 1 0 1
  ---------------
0 0 1 0 0 0 0 0 1 0 0   (260)
```

2's complement of multiplicand

```
1 1 0 0 1 1
0 0 1 1 0 0
        1
---------
0 0 1 1 0 1
```

45x - 24

PageWork

D D M M Y Y Y Y

c) M = -13
Q = +9

2's complement

1101 (-13) ⟶ 0011 (-13)
1001 (9) ⟶ 1001 (9)

1101 Adding sign bit 10011 (-13)
0010 01001 (9)
1
0011

Recoding the multiplier 01001 0
+1 -1 0 +1 -1

10011
1 -1 0 1 -1

2's complement of multiplicand

10011
01100
1
01101

0 0 0 0 0 0 1 1 0 1
1 1 1 1 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 1 1 0 1
1 1 0 0 1 1
1 1 1 0 0 0 1 0 1 1 (-117)

d) M = +14 2's complement
Q = -8 1110 (14) ⟶ 1110 (14)
1000 (-8) 1000 (-8)

1000 : Adding sign-bit 01110
0111 11000
1
1000

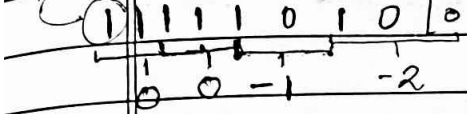
Recoding the multiplier.
1 1 0 0 0 0
0 -1 0 0 0

2's complement of multiplicand
01110
10001
1
10010

01110
0 -1 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
1 1 1 0 0 1 0
0 0 0 0 0 0 1 1
1 1 1 0 0 1 0 0 0 0 (-112)
1

Bit pair Recoding of Multipliers

sign extending



1 1 1 0 1 0 0
0 -1 -2

Table of multiplicand selection decisions

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
i+1	i	i-1	
0	0	0	0 × M
0	0	1	+1 × M
0	1	0	+1 × M
0	1	1	+2 × M
1	0	0	-2 × M
1	0	1	-1 × M
1	1	0	-1 × M
1	1	1	0 × M

a) M = +13
Q = -6

Note:-
-2 : take 2's complement of multiplicand then multiply with 10
+2 : multiply with 10

2's complement → Adding sign bit

1101 (+13) → 1101 → 01101
0110 (-6) → 1010 → 11010

recoding the multiplier:

Bit pair recoding 1 1 1 0 1 0 0

0 1 1 0 1
0 -1 -2

0 0 1 1 0
1 0 0 1 0
0 0 0 0

01101
10010
1
10011

-2: 01101
10010
1
10011
× 10
00000
10011
100110

PageWork D D M M Y Y Y Y

01101
0 -1 -2

0 0 0 0 0 0 0 1 1 0
1 1 1 1 0 0 1 1
0000 0 0

1 1 1 0 0 1 0 0 1 0 (-78)

Q) M = -11 → Binary 1011 — 2's complement for -11: 0101
Q = 27 → 11011 — ~~00101~~

1011
0100
   1
0101

Adding the sign bit: 110101, 000101

110101
011011

11011
00100
    1
00101

recoding 011011 0
+2 -1 -1

~~110101~~
~~+2 -1 -1~~
~~0000000011011~~
~~001011~~

2's complement of multiplicand.
110101
001010
     1
001011

+2: multiply with 10
110101
10
000000
110101
1101010

+2 -1 -1

0 0 0 0 0 0 0 0 1 0 1 1

0 0 0 0 0 0 1 0 1 1

0 1 1 0 1 0 1 0

1 1 0 1 1 0 1 0 1 1 1 1 (-297)

Q) M = +23 | Binary format 10111 | 2's complement 10111 (23)

Q = -20 | 10100 | 01100 (-20)

10100 / 01011 / 1 / 01100

10111 / 01000 / 01001

Add the sign bit 010111

~~101001~~ 101100

Recoding the multiplier

101100 0

-1 -1 0

~~1010 0110~~

~~-1 -2 +1~~

=> 010111

-1 -2 1

000000010111

1111 1000 0010

11 101001

1 1 1 1 0 1 1 1 1 0 1 1 1 1

2's complement of multiplicant

010111

101000

101001

× 10

0000000

101001

1010010

010111

-1 -1 0

000000 000000

111110 1001

110100 1

111000 110100. (-460)

PageWork
D D M M Y Y Y Y

(CSA)

Carry Save addition of Summands

8) M : 1 0 1 1 0 1

Q : x 1 1 1 1 1 1

1 0 1 1 0 1	A	FED CBA level 1 CSA
1 0 1 1 0 1	B	C2 S2 C1 S1 level 2 CSA
1 0 1 1 0 1	C	C2 C3 S3 level 3 CSA
1 0 1 1 0 1	D	C4 S4 = final add^n
1 0 1 1 0 1	E	product
1 0 1 1 0 1	F	

1 0 1 1 0 1	A
1 0 1 1 0 1	B
1 0 1 1 0 1	C
1 1 0 0 0 0 1	S1
0 0 1 1 1 1 0	C1
1 0 1 1 0 1	D
1 0 1 1 0 1	E
1 0 1 1 0 1	F
1 1 0 0 0 0 1	S2
0 0 1 1 1 1 0	C2
1 1 0 0 0 0 1	S1
0 0 1 1 1 1 0	C1
1 1 0 0 0 0 1	S2
1 1 0 1 0 1 0	S3
0 0 0 0 1 0 1	C3
0 0 1 1 1 1 0	C2
0 1 0 1 1 1 0	S4
0 0 1 0 1 0 0	C4
1 0 1 1 0 0 0	product
0 0 1 1	

PageWork
D D M M Y Y Y Y

Integer Division

Algorithm for restoring division:-

1) A ← 0 (n+1 bits), M ← Divisor, Q ← Divident (n bit), Count ← 0

2) Shift A & Q left one binary position.

3) A ← A − M

4) if (A) is 1 → sign bit value of A
   - → Yes — Assign $Q_0 \leftarrow 0$, A ← A+M (Restore)
   - → NO — $Q_0 \leftarrow 1$

5) Count ← Count − 1

6) Repeat 2, 3, 4, 5 n times.

7) Dividend : Q → 1010 → 4 bit long ⇒ n = 4
   Divisor : M → 0011

   ⇒ Register A should be assigned with (n+1) zeroes

   Count = 4.

	A	Q
	0 0 0 0 0	1 0 1 0
2)	0 0 0 0 1	0 1 0 [ ]

3) $A \leftarrow A - M$

M ⇒ 00011

11100

1

11101

00001

11101

A ← ① 1110

4) → $Q_0 \leftarrow 0$

Q

010 [0]

$A \leftarrow A + M$

11110

00011

① 00001

carry ignore

5) Count = 4 − 1

Count = 3

2nd cycle

Repeat step 2, 3, 4, 5

A Q

00001 0100

00010 1100 [0]

11101

① 11111

1 00 [0]

00011

carry ① 00010

Count = 2

PageWork

D D M M Y Y Y Y

3rd cycle

	A	Q
	00010	1000
Shift:	00101	000
	11101	
ignore 1	00010	
		0001

Count = 1

4th cycle

	A	Q
	00010	0001
Shift	00100	0010
	11101	
	00001	0011

A = 00001 (Remainder)

Q = 0011 (Quotient)

~~Count~~ Count = 0

Q) Dividend Q → 1100 $n = 4$

Divisor M → 0011

	A	Q	2's complement of M
1st cycle	00000	1100	M 00011
	00001	100 □	11100
	11101		11101
	(1) 1110	Q0 ↓	
	00011	100 0	
	(1) 00001		
2nd cycle	00011	000 □	
	11101		
	(0) 00000		
		000 1	
3rd cycle	000001	001 □	
	11101		
	(1) 11101		
	00011	001 0	
	(0) 00000		
4th cycle	00000	010 □	
	11101		
	(1) 11101	↓	
	00011	010 0	
	(0) 00000		

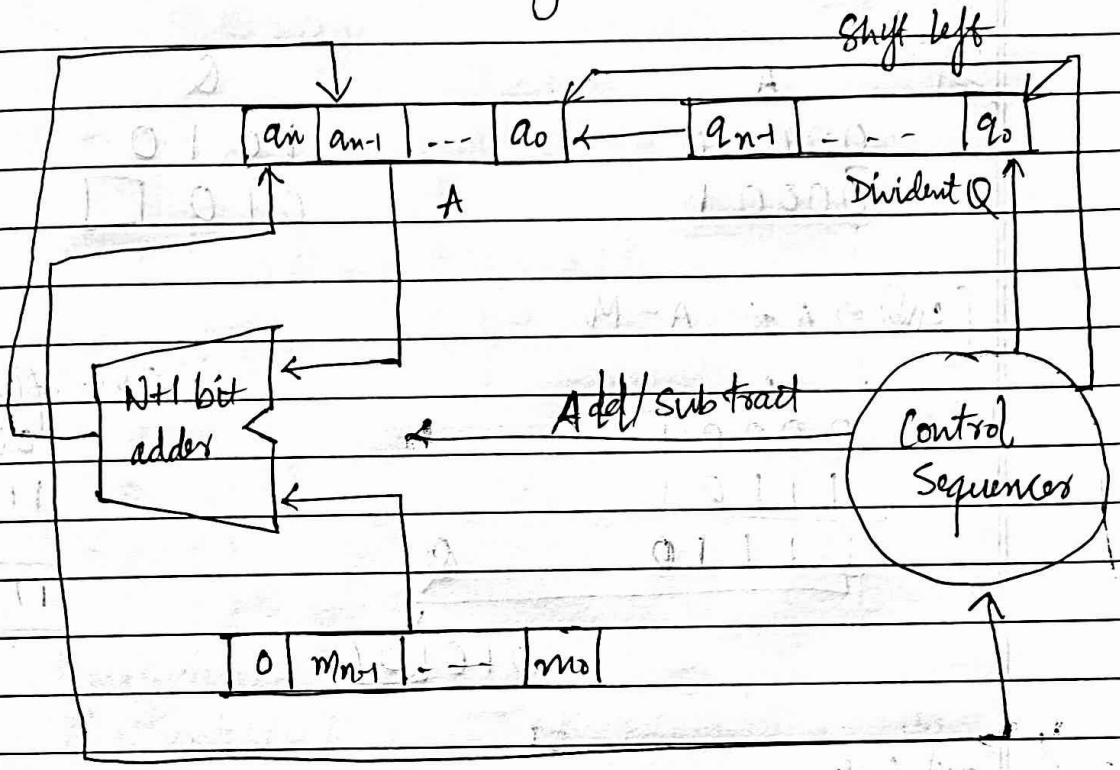
A	Q
00000	0100

⇒ Remainder = 0 (0)

Quotient = 0100 (4)

Circuit Arrangement

Restoring division



Non-Restoring Division- Steps

1) A ← 0 , M ← Divisor , Q ← Divident , count ← n
2) Shift left A & Q
3) If Sign of A is 0
   - → Yes , A ← A - M (Sub) → { A is 1 , $Q_0 = 0$; A is 0 , $Q_0 = 1$
   - → No , A ← A + M (Add) → { A is 1 , $Q_0 = 0$; A is 0 , $Q_0 = 1$
4) Repeat 2 & 3 for n times.
5) If the sign of A is 1, add divisor to A

Q) Divisor & M : 0011
Dividend Q : 1010

cycle 1

A	Q
00000	1010
00001	010 [ ]

(sub) → A ← A − M

A	
00001	
11101	
11110	Q

2's complement M
00011
11100
1
11101

	Q
0101	0

2nd cycle

A	Q
11110	0100
11100	100 [ ]

(add) → A ← A+M

A	
11100	
00011	
11111	Q

	Q
100	0

3rd cycle

A	Q
11111	000 [ ]

(Add) → A ← A+M

A	
11111	
00011	
00010	Q

	Q
000	1

4th cycle

A	Q
00010	0001
00100	0010

(Sub) | A ← A − M

```
 00100
 11101
(0)0001
```

A	Q
00001	0011

Remainder = 1
Quotient = 0011

a) Q : 1011
M : 0101

cycle 1

A	Q
00000	1011
00001	011

(Sub) | A ← A − M

2's M = 11011

```
 00001
 11011
(1)1100
```

A	Q
11100	110

2nd cycle

A	Q
(1)1000	110

(Add)

```
 11000
 00101
(1)1101
```

A	Q
11010	110

3rd cycle

	A	Q
	① 1 0 1 1	1 0 0 1
(Add)	A ← A+M	
	1 1 0 1 1	
	0 0 1 0 1	
1)	⓪ 0 0 0 0	1 0 0 1

4th cycle

	A	Q
	⓪ 0 0 0 1	0 0 1 1
(Sub)	A ← A − M	
	0 0 0 0 1	
	1 1 0 1 1	
	① 1 1 0 0	0 0 1 1 0

A : ① 1 1 0 0

↳ Add divisor to A to get remainder.

```
  1 1 0 0
  0 0 1 0 1
  ---------
  0 0 0 0 1
```

Quotient : 0 0 1 0

IEEE Standard for Floating point numbers

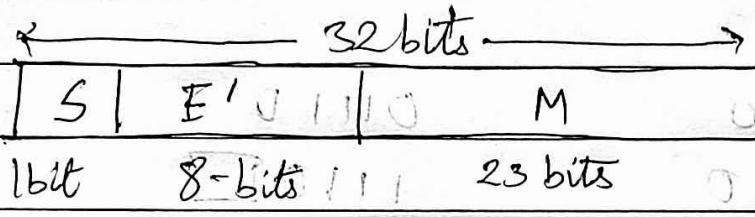
Floating Point Numbers has 3 parts

M :- Mantissa $\pm M \times B^E$

B : Base

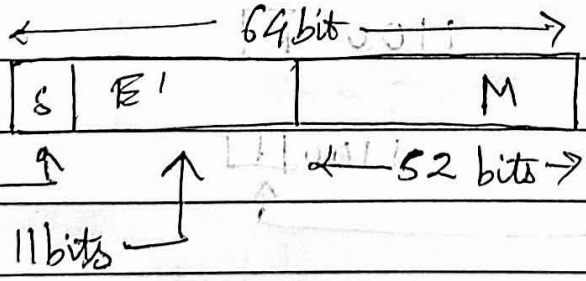
E : Exponent

Single Precision Format



Normalize :- $\pm 1.M \times 2^{E'-127}$

Double Precision Format



Normalize :-

$\pm 1.M \times 2^{E'-1023}$

a) $(1259.125)_{10}$ in ~~Sig~~ Single and double precision IEEE floating point format.

Step 1) Convert decimal to Binary form.

$(1259)_{10} = 10011101011$

$(0.125)_{10} \Rightarrow$

$0.125 \times 2 = 0.25$	0	001
$0.25 \times 2 = 0.50$	0	
$0.5 \times 2 = 1.0$	1	

$(1259.125)_{10} = (10011101011.001)_2$

Step 2) Normalize the number:

10011101011.001

$\rightarrow 1.0011101011001 \times 2^{10}$

Step 3) Represent in Single precision form

$\pm 1.M \times 2^{E'-127} \rightarrow 1.00111010110 01 \times 2^{10}$

$\Rightarrow 2^{E'-127} = 2^{10}$

$E' - 127 = 10$

$E' = 137 \rightarrow$ Binary form 10001001

0	10001001	00111010110010000000000
S-1 bit	E' - 8 bits	M - 23 bits

Step 4) Represent in Double precision form.

$1.00111010111001 \times 2^{10}$

$\pm 1.M \times 2^{E'-1023}$

$\Rightarrow 2^{E'-1023} = 2^{10}$

$E' - 1023 = 10$

$E' = 1033. \rightarrow$ Binary form 10000001001

0	10000001001	00111010111001000...00
S-1bit	E'-11bits	M-52bits

02) Compute the single precision IEEE 754 binary representation of the number $(-118.625)_{10}$

Ans :-

Step 1:- $(118)_{10} = 1110110$

$(0.625)_{10}$ = $0.625 \times 2 = 1.25$ | 1 | .101

$0.25 \times 2 = 0.50$ | 0

$0.50 \times 2 = 1$ | 1

$(118.625)_{10} = (1110110.101)_2$

Step 2: Normalize the number

1110110.101

$\Rightarrow 1.110110101 \times 2^6$

3 : $\pm 1.M \times 2^{E'-127} \rightarrow 1.110110101 \times 2^6$

$2^{E'-127} \rightarrow 2^6$

$E' = 127 + 6 = 133 \rightarrow 10000101$

1	10000101	11011010000000000000000
S-1bit	E-8bits	23 bits

Q3) $(85.125)_{10}$

step 1) $(85)_{10} = 1010101$

$(0.125)_{10}$ =

$0.125 \times 2 = 0.25$	0	1001
$0.25 \times 2 = 0.50$	.0	
$0.50 \times 2 = 1$	1	

$(85.125)_{10} = (1010101.001)_{10}$

2) 1.0101001×2^6

3) $\pm 1.M \times 2^{E'-127} = 1.0101001 \times 2^6$

$2^{E'-127} = 2^6$

$E' = 133 \longrightarrow 10000101$

0	10000101	01010010000000000000000
S-1bit	E'-8bit	M-23bit

4) $\pm 1.M \times 2^{E'-1023} = 1.0101001 \times 2^6$

$2^{E'-1023} = 2^6$

$E' = 1029 \longrightarrow 10000000101$

0	10000000101	0101001000...000
S-1bit	E'-11bits	M-52-bits

Pagework

D D M M Y Y Y Y

a) $(263.3)_{10}$

$(263)_{10}$ 100000111

$(0.3)_{10}$

$0.3 \times 2 = 0.6$	0
$0.6 \times 2 = 1.2$	1
$0.2 \times 2 = 0.4$	0
$0.4 \times 2 = 0.8$	0
$0.8 \times 2 = 1.6$	1
$0.6 \times 2 = 1.2$	1
$0.2 \times 2 = 0.4$	0

$(263.3)_{10} = 100000111.0100110011...$

Single Precision format :-

$\pm 1.M \times 2^{E'-127} \rightarrow 1.00000111010011... \times 2^8$

$2^{E'-127} \rightarrow 2^8$

$E'-127 = 8$

$E' = 8+127 = 135$

$M \Rightarrow 10000111$

0	10000111	00000111010011...
S-1 bit	E'-8 bit	M-23 bits