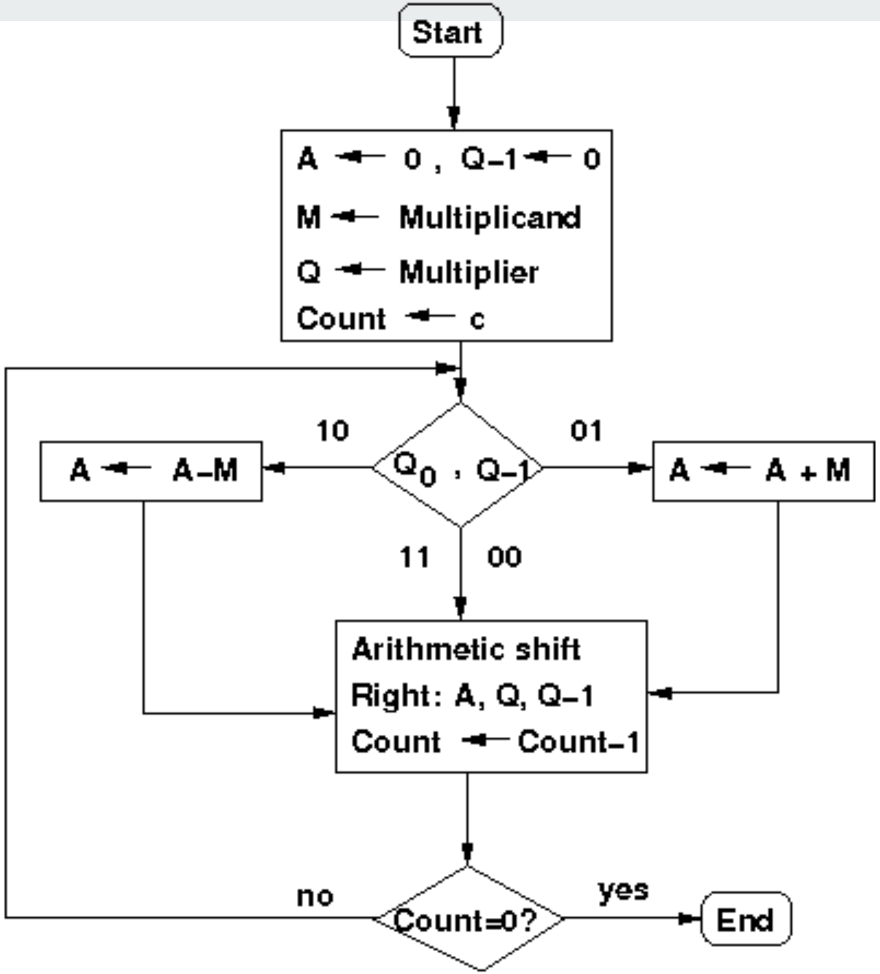
Module 4
Data Representation and Arithmetic Algorithms

Booth's Algorithm

- The booth algorithm is a multiplication algorithm that allows us to multiply the two signed binary integers in 2's complement, respectively.
- It is also used to speed up the performance of the multiplication process.
- It is very efficient too.

Booth's Algorithm

- The multiplicand and multiplier are placed in the M and Q registers respectively.
- A and Q-1 are initially set to 0.
- Control logic checks the two bits Q0 and Q-1.



Booth's Algorithm

- If the two bits are same (00 or 11) then all of the bits of A, Q, Q-1 are shifted 1 bit to the right.
- If they are not the same and if the combination is 10 then the multiplicand is subtracted from A and if the combination is 01 then the multiplicand is added with A.
- In both the cases results are stored in A, and after the addition or subtraction operation, A, Q, Q-1 are right shifted.
- The result of the multiplication will appear in the A and Q.

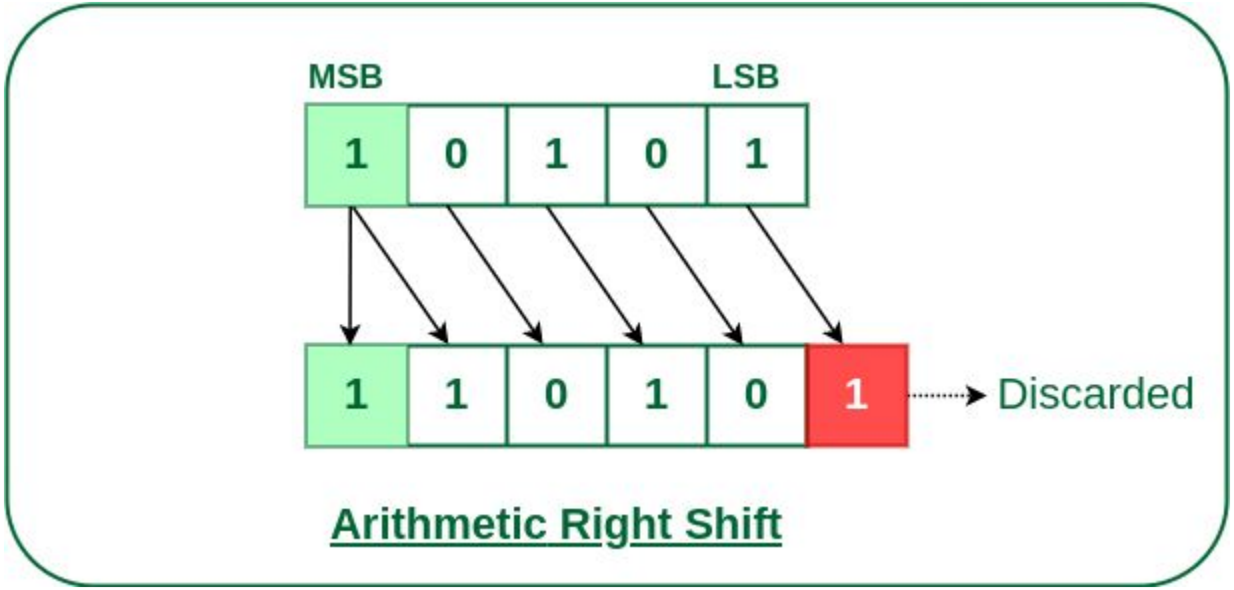
Booth's Algorithm steps

1. Start
2. Get the multiplicand (M) and Multiplier (Q) from the user
3. Initialize A= Q_{-1} =0
4. Convert M and Q into binary
5. Compare Q_0 and Q_{-1} and perform the respective operation.

$Q_0 Q_{-1}$	Operation
00/11	Arithmetic right shift
01	A+M and Arithmetic right shift
10	A-M and Arithmetic right shift

6. Repeat steps 5 till all bits are compared
7. Convert the result to decimal form and display
8. End

Booth's Algorithm



Example 1: Multiply the two numbers 7 and 5 by using the Booth's algorithm.

Given data:

First of all, we need to convert 7 and 3 into binary numbers 7 = (0111) and 5 = (0101).

M=0111

Q=0101

Count represents the number of bits, and here we have 4 bits, so set the C = 4.

Example 1: Multiply the two numbers 7 and 5 by using the Booth's algorithm.

M= 0111

- M is 2's complement of M i.e. 0111

	0 1 1 1
1's complement is:	1 0 0 0
2's complement is:	+ 1
	1 0 0 1

Thus -M = 1 0 0 1

To make calculation easy we rewrite equation AC-M as AC+ (-M)
So, we will calculate -M first so simplify operation.

Example 1: Multiply the two numbers 7 and 5 by using the Booth's algorithm.

A	Q	Q-1	M		
0000	0101	0	0111	Initial value	
1001	0101	0	0111	A ← A-M	First cycle
1100	1010	1	0111	shift	
0011	1010	1	0111	A ← A+M	Second cycle
0001	1101	0	0111	shift	
1010	1101	0	0111	A ← A-M	Third cycle
1101	0110	1	0111	shift	
0100	0110	1	0111	A ← A+M	Fourth cycle
0010	0011	0	0111	shift	

00100011 -> 35

Thus

0111 * 0101 = 00100011

Example 2: Multiply the two numbers -6 and 2 by using the Booth's algorithm.

Given data:

M= $(-6)_{10}$ = 1010

-M is 2's complement of M

-M= $(6)_{10}$ = 0110

	0 1 1 0
1's complement:	1 0 0 1
2's complement:	1 0 1 0

Q= 2 = 0010

Example 3: Multiply the two numbers -6 and 2 by using the Booth's algorithm.

A	Q	Q_{-1}	operation	cycle
0000	0010	0	Initial	
0000	0001	0	A.S.R.	1st cycle
0110	0001	0	(i) A = A − M ⇒ 0000 + 0110 = 0110	2nd cycle
0011	0000	1	(ii) A.S.R.	
1101	0000	1	(i) A = A + M 0011 + 1010 = 1101	3rd cycle
1110	1000	0	(ii) A.S.R.	
1111	0100	0	A.S.R.	4th cycle

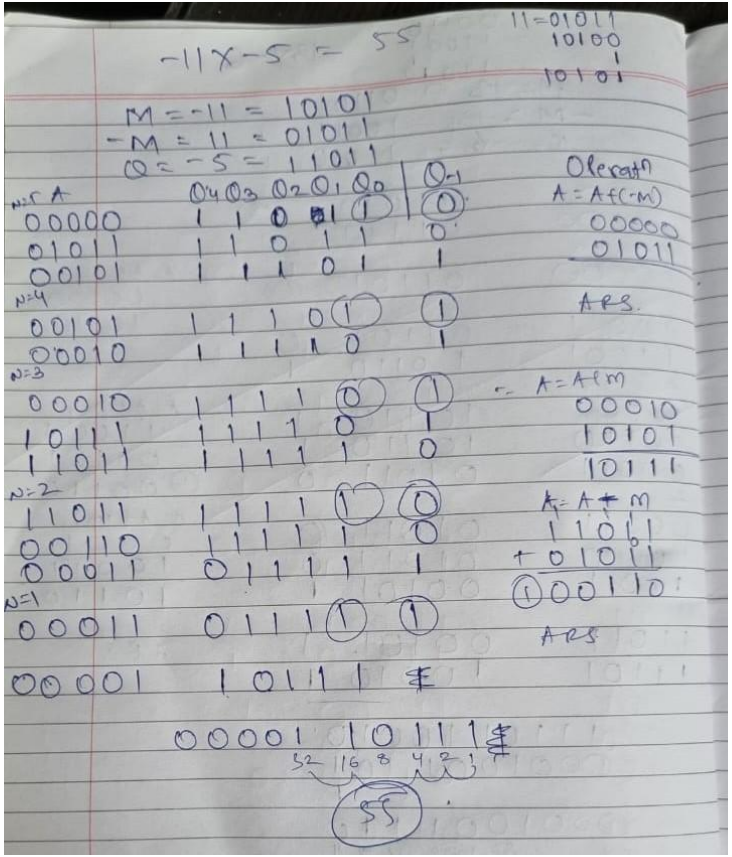
(−12)

$(11110100)_2 = (-12)$

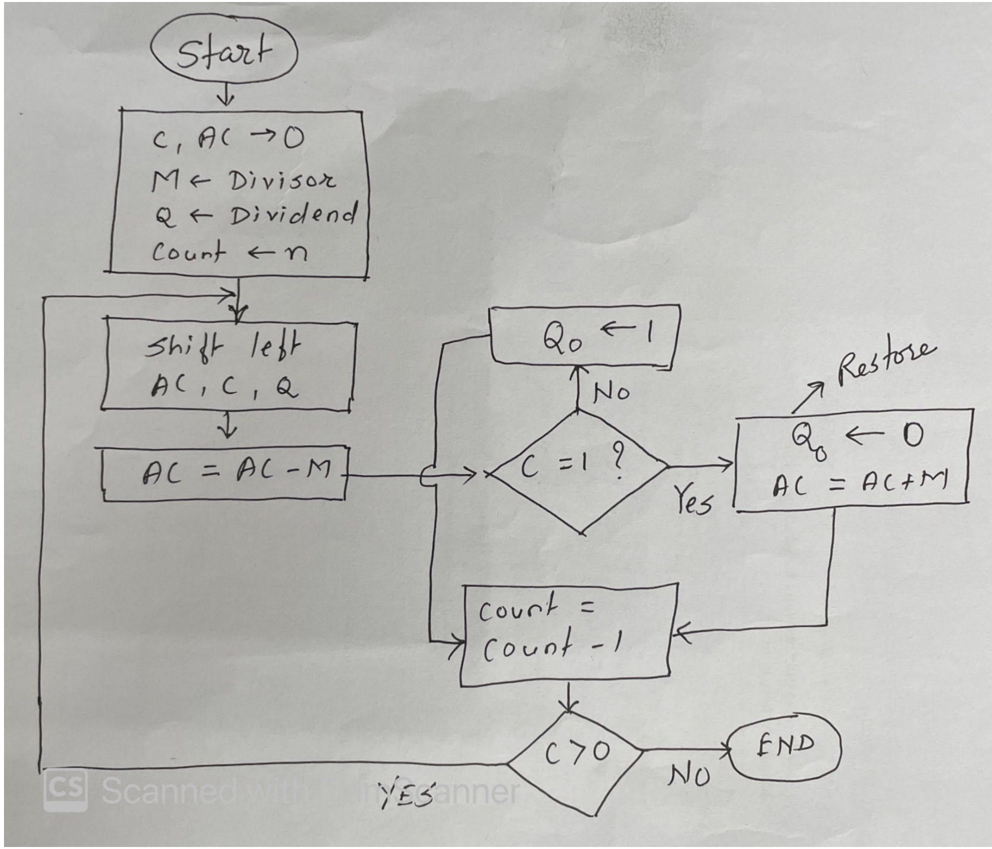
To verify, take 2's complement of $(11110100)_2$

$$\begin{array}{r} 00001011 \\ +\quad\quad\quad 1 \\ \hline 00001100 \end{array} = (12)$$

Example 4: Multiply the two numbers -11 and -5 by using the Booth's algorithm.



Restoring Division algorithm



Restoring Division method-Example

$17 \div 3$

Dividend = 17 , Divisor = 3

$Q = 17 = (10001)_2$ $M = 3 = (00011)_2$

$n = 5$ (5 cycles)

$(n+1)$ bits are used for handling the borrow

thus,

$M = (000011)_2$ $-M = 111100 + 1$, $-M = 111101$

	C	AC	Q	
	0	00000	10001	← initial
1st cycle	0	00001	0001□	left shift
	1	11110	0001$\boxed{0}$	AC = AC − M: 000001 + 111101 = 111110
	0	00001	00010	AC = AC + M: 111110 + 000011 = ~~0~~000001

Note: when C = 1 then there is need to restore

	C	AC	Q	
2nd cycle	0	00010	0010□	left shift
	1	11111	0010$\boxed{0}$	AC = AC − M: 000010 + 111101 = 111111
	0	00010	00100	Restore AC = AC + M: 111111 + 000011 = ~~0~~000010
3rd cycle	0	00100	0100□	left shift
	0	00001	0100$\boxed{1}$	AC = AC − M: 000100 + 111101 = ~~0~~000001
4th cycle	0	00010	1001□	left shift
	1	11111	1001$\boxed{0}$	AC = AC − M: 000010 + 111101 = 111111
	0	00010	10010	Restore AC = AC + M: 111111 + 000011 = ~~0~~000010
5th cycle	0	00101	0010□	left shift
	0	00010	0010$\boxed{1}$	AC = AC − M: 000101 + 111101 = ~~0~~000010
	0	00010	00101	

$\underbrace{0}_{C}$ $\underbrace{00010}_{\text{AC Remainder}}$ $\underbrace{00101}_{\text{Q Quotient}}$

$(10)_2 = (2)_{10}$ $(101)_2 = (5)_{10}$

R = 2 , Q = 5

Q. 5

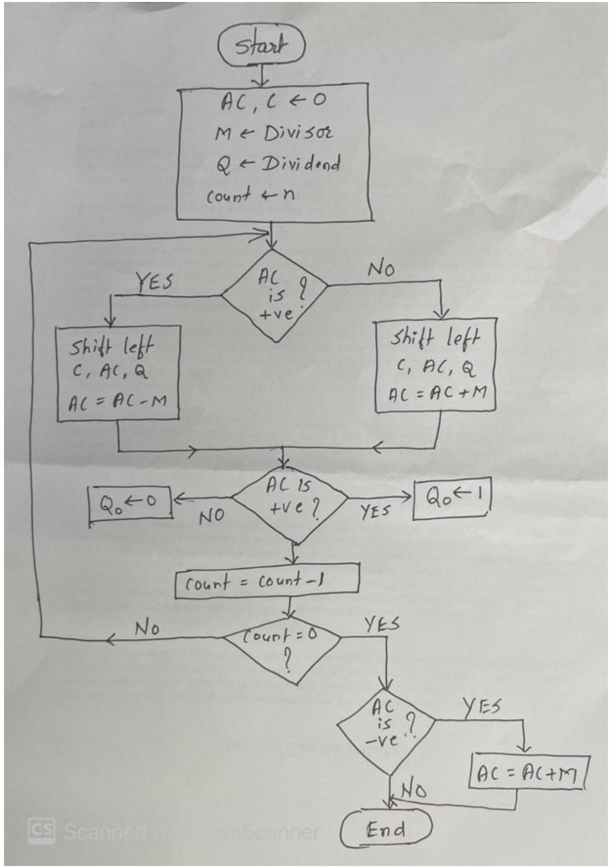
3 | 17

15

R 2

CS Scanned with CamScanner

Non-Restoring Division method



Non-Restoring-Example

Q) Divide $(1011)_2$ with $(0011)_2$ using non-restoring division method.

⇒ $Q = (1011)_2$

$M = (00011)_2$ $\qquad -M = (11101)_2$

	C	AC	Q	Operation
	0	0000	1011	Initial value
1st cycle	0	0001	011□	Shift left
	1	1110	011 $\boxed{0}$	AC=AC − M 00001 + 11101 11110
2nd cycle	1	1100	110□	shift left
	1	1111	110 $\boxed{0}$	AC= AC+M 11100 + 00011 11111
3rd cycle	1	1111	100□	shift left
	0	0010	100 $\boxed{1}$	AC= AC+M 11111 + 00011 1 00010
	0	0101	001□	shift left
	0	0010	001 $\boxed{1}$	AC = AC − m 00101 + 11101 1 00010

+ve

If count is zero, we check A is negative or not. If its negative, we do AC = AC + M otherwise end it.
Here A is +ve because C is '0' in forth cycle so we directly end it.

$R = (00010)_2 = (2)_{10}$

$Q = (0011)_2 = (3)_{10}$

$(1011)_2 \div (0011)_2$ i.e. $(11)_{10} \div (3)_{10}$

Hence $Q = (3)_{10}$ & $R = (2)_{10}$

Datatype representation

1. Fixed point number representation
2. Floating point number representation

1. Fixed point number representation

$(+7)_{10} = (0111)_2$

$(-7)_{10} = (1001)_2$ by taking the 2's complement of +7

2. Floating point number representation

It has three parts:

1. Mantissa
2. Base
3. Exponent

2. Floating point number representation

For example,

Number	Mantissa	Base	Exponent
3 X 10 ^ 6	3	10	6
110 X 2 ^ 8	110	2	8
6132.784	6132784	10	-3

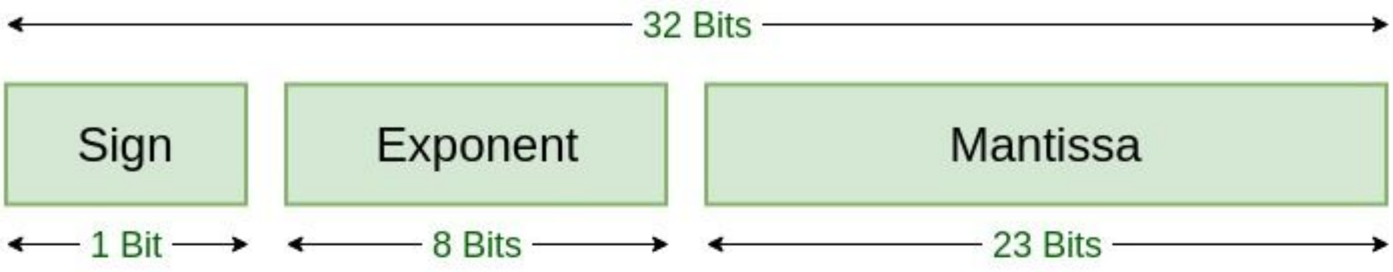
2. Floating point number representation

But processor can not understand these things, so IEEE made a special format for floating point numbers

IEEE 754 Floating point number representation

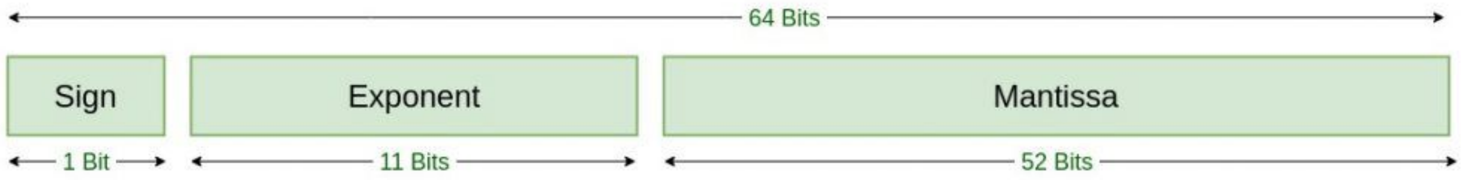
1. Single precision format
2. Double precision format

1. Single precision format



It have total 32 bits (0-31)
Bias =127
Exponent E = e +127

2. Double precision format



It have total 64 bits (0-63)

Bias =1023

Exponent E = e +1023

Examples

Represent $(1259.125)_{10}$ in a single and double precision format

Solution:

Step 1: convert decimal number to binary

$$(1259)_{10} = (10011101011)_2$$

$$(0.125)_{10} = (001)_2$$

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

Examples

Step 2: Normalize the number : To normalize the number, shift the radix i.e. dot before the first 1 in a number as follows:

$(10011101011.001)_2$ = (1.0011101011001 X 2 ^ 10) is the normalized number.

Here exponent e = 10

Examples

Step 3: single precision format

To find SPF, find out E. (E=e+127)

$$E = e + 127$$
$$= 10 + 127$$
$$= 137$$

Now convert 137 into binary:

$(137)_{10} = (10001001)_2$

31	30 — 23	22 — 0
0	10001001	0011101011001......00.......
1-bit	8-bit	23-bit

Examples

Step 3: double precision format

To find DPF, find out E. (E=e+1023)

$$E = e + 1023$$
$$= 10 + 1023$$
$$= 1033$$

Now convert (1033) into binary:

$(1033)_{10} = (1000000100)_2$

64	62 52	51 0
0	1000000100	001110101100.....00....
1-bit	11-bit	52-bit