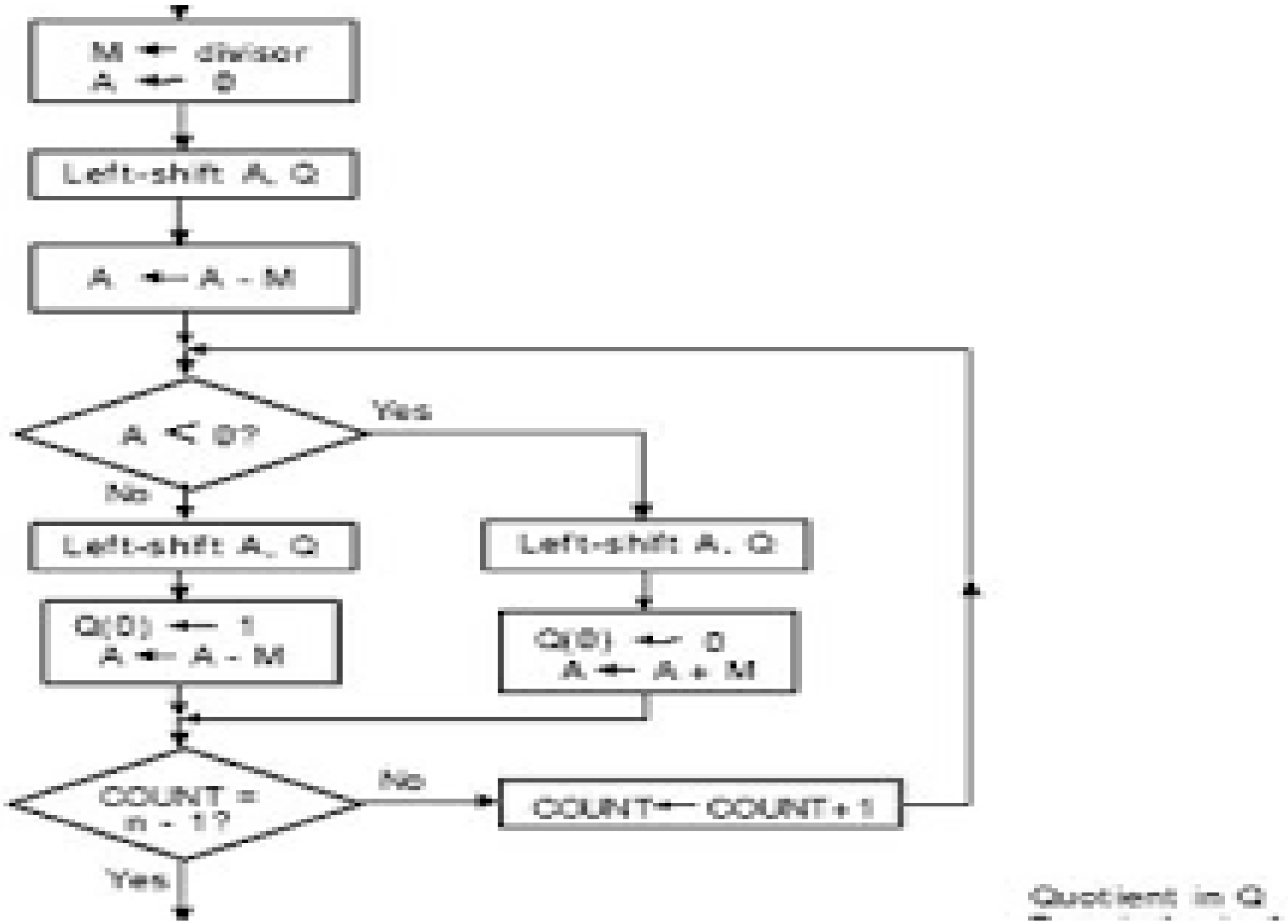
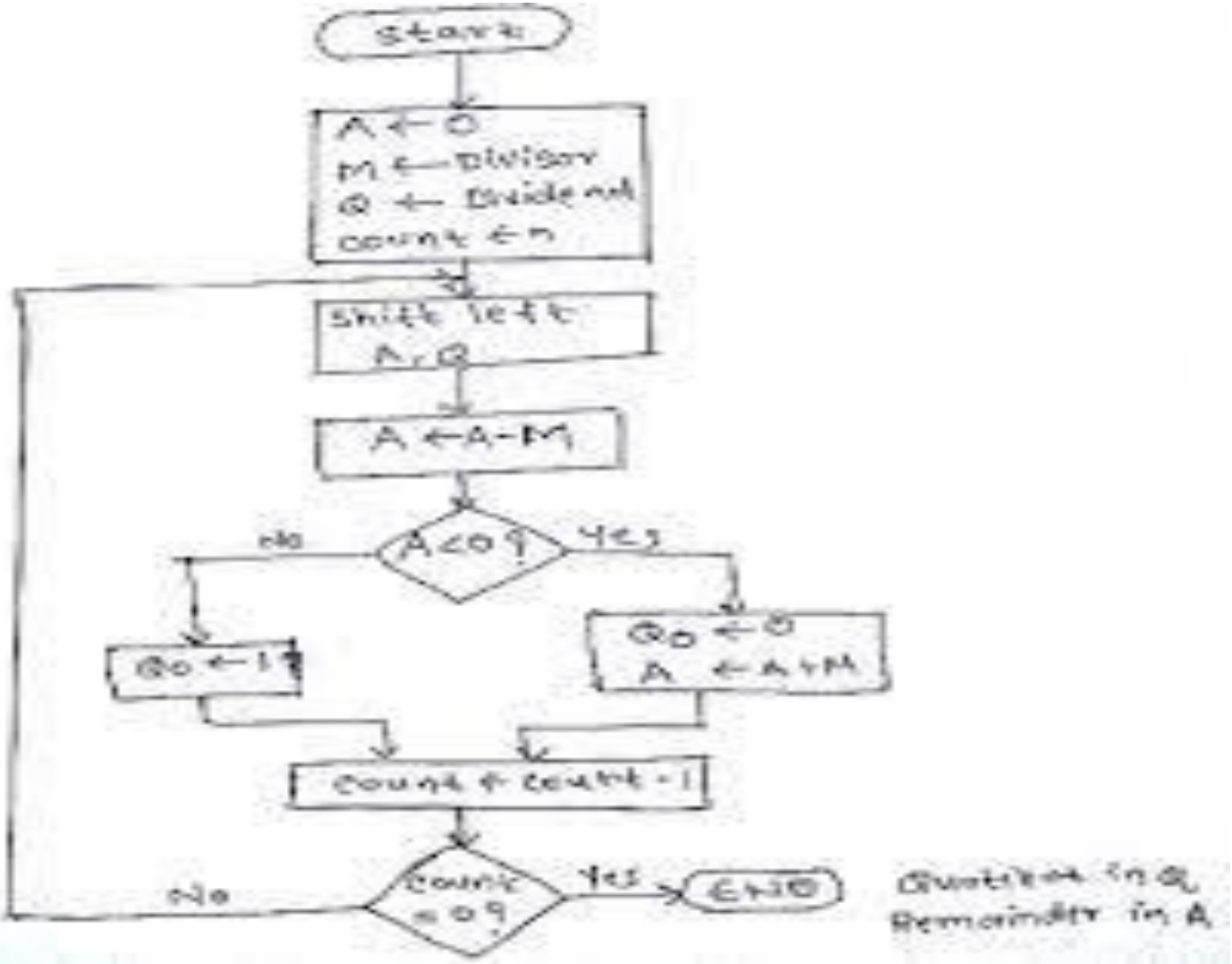
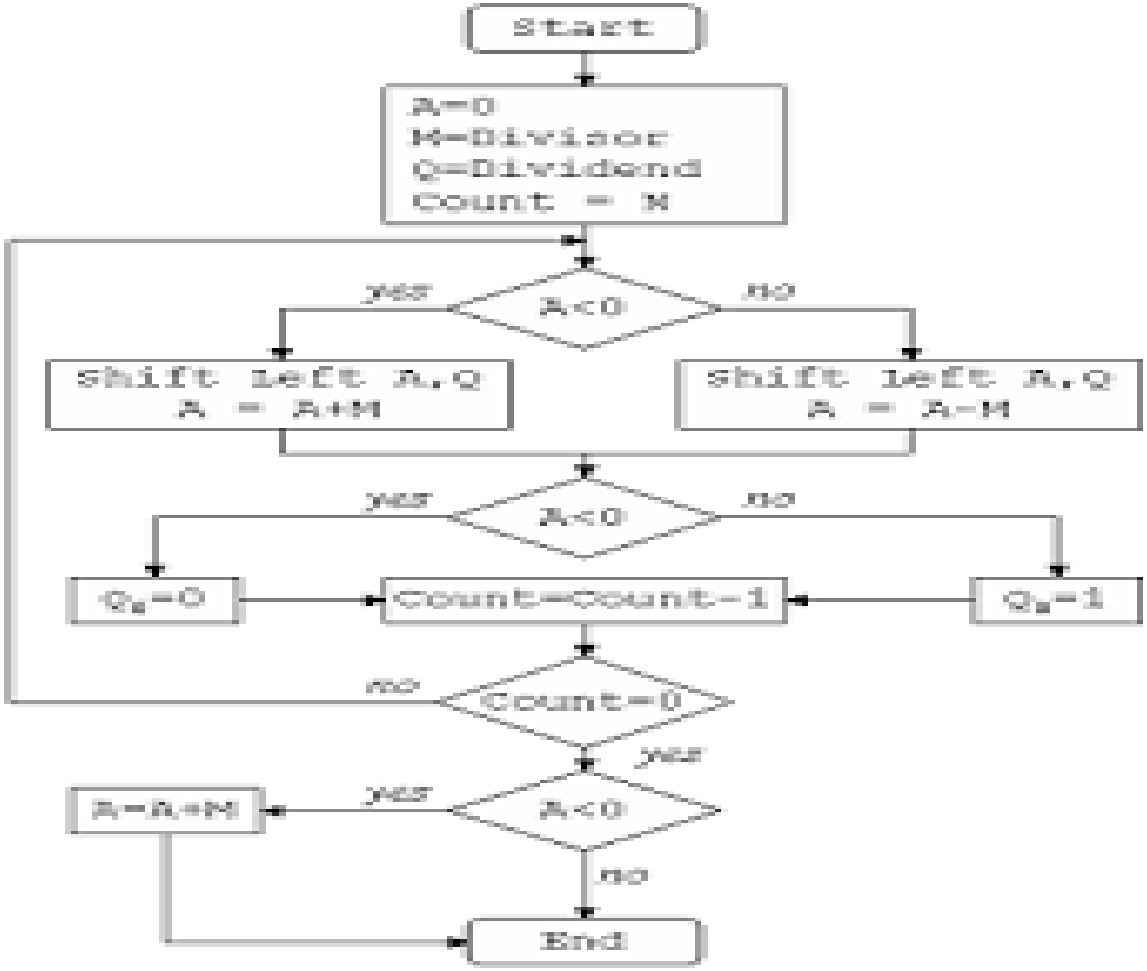
Booths division –restoring algorithm

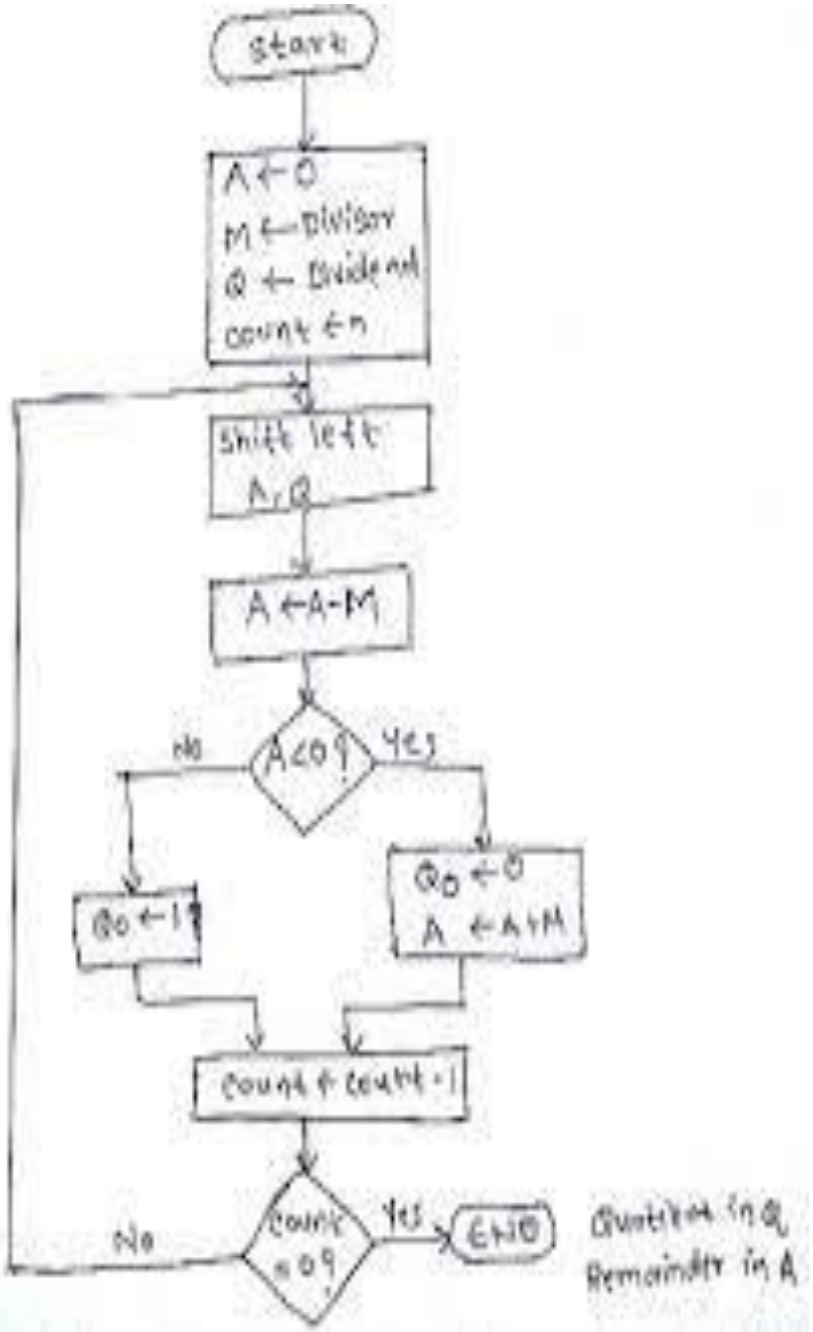


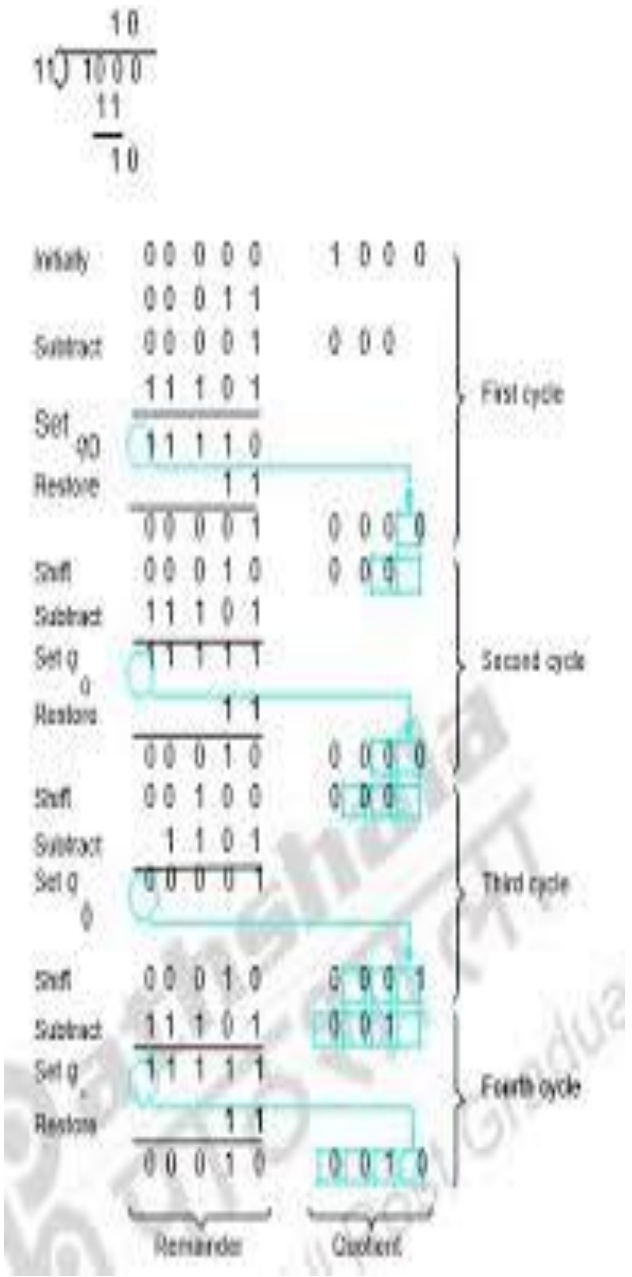
Booths division –restoring algorithm

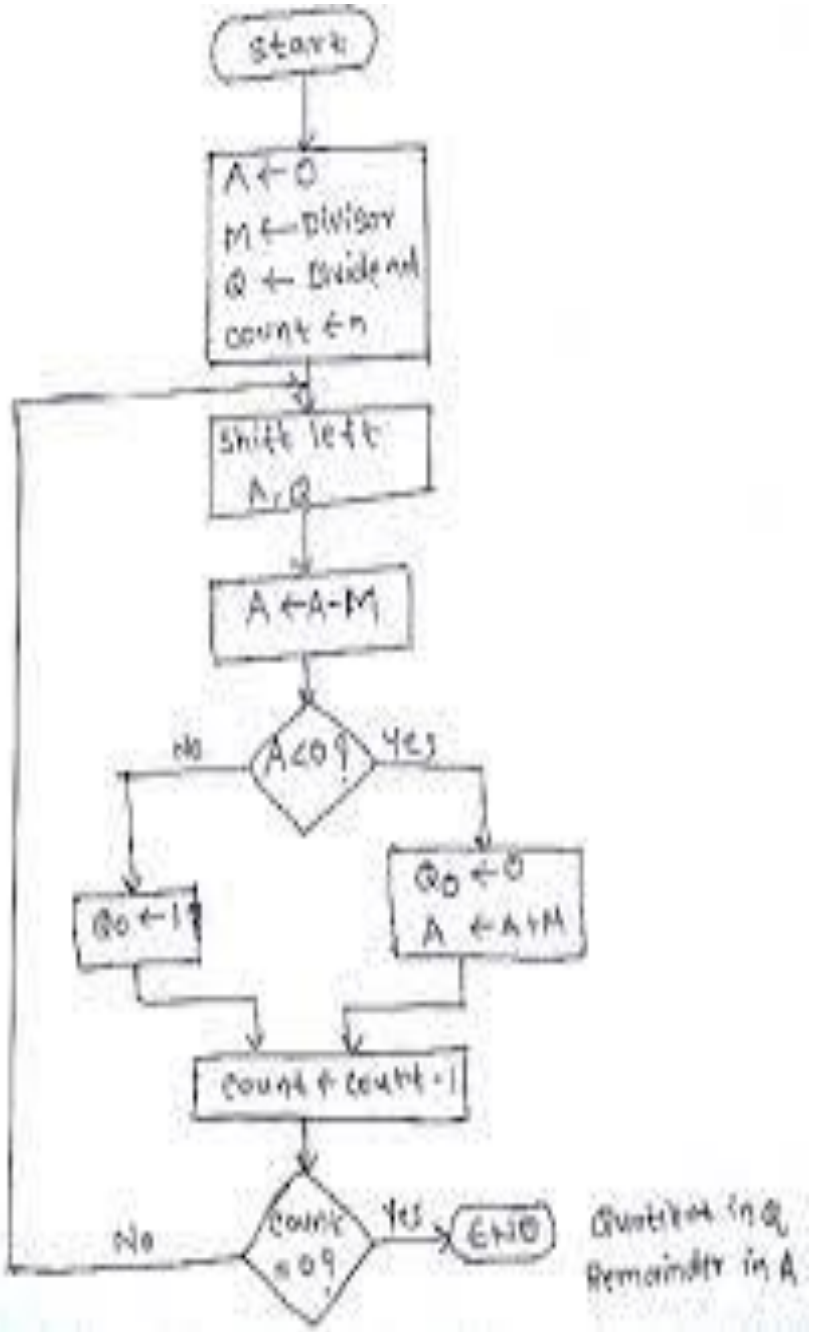


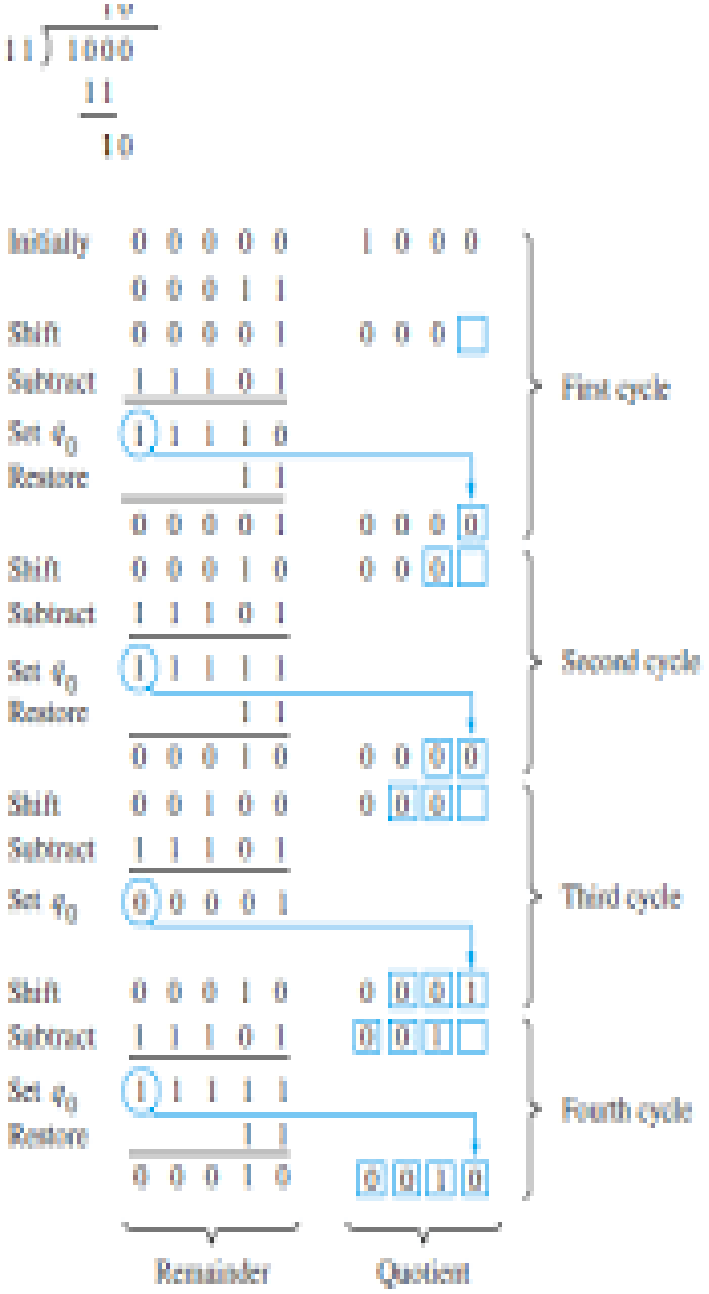
Booths division –non-restoring algorithm




Start
A ← 0
M ← Divisor
Q ← Dividend
count ← n
Shift left A, Q
A ← A-M
A<0?
no
yes
Q0 ← 1
Q0 ← 0
A ← A+M
count ← count - 1
count = 0?
No
Yes
END
Quotient in Q
Remainder in A

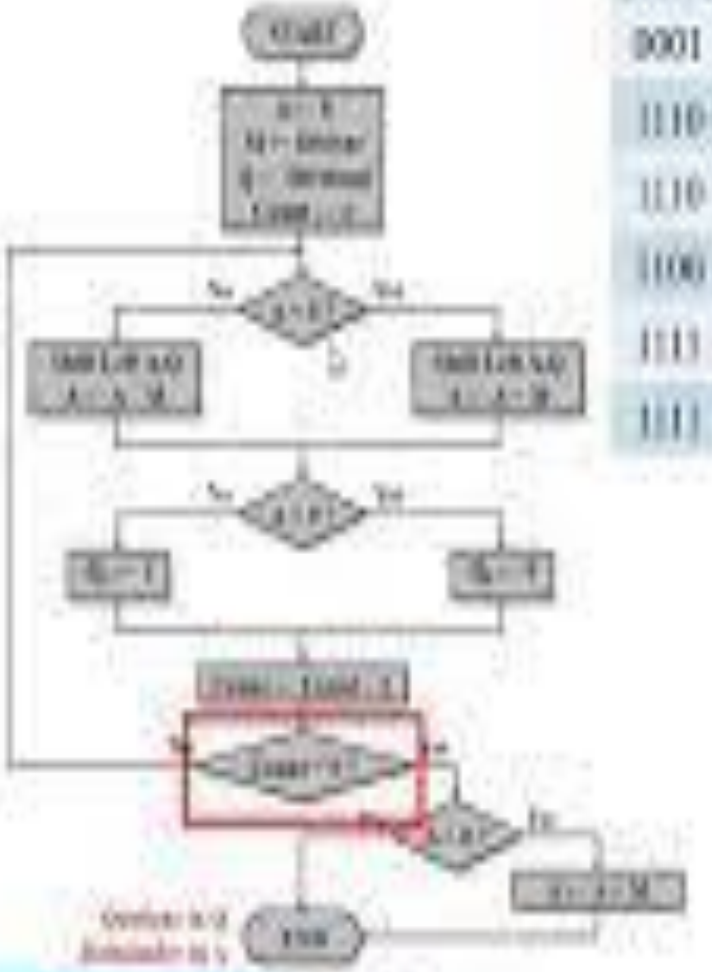

10
11) 1000
11
10
Initially
Subtract
Set q0
Restore
Shift
Subtract
Set q1
Restore
Shift
Subtract
Set q2
Restore
Shift
Subtract
Set q3
Restore
First cycle
Second cycle
Third cycle
Fourth cycle
Remainder
Quotient


Start
A ← 0
M ← Divisor
Q ← Dividend
count ← n
Shift left A,Q
A ← A-M
A<0 ?
No
Yes
Q0 ← 1
Q0 ← 0
A ← A+M
count ← count - 1
count = 0 ?
No
Yes
END
Quotient in Q
Remainder in A


1011 1000
11
10
Initially 0 0 0 0 0 1 0 0 0
0 0 0 1 1
Shift 0 0 0 0 1 0 0 0
Subtract 1 1 1 0 1
Set q_0 1 1 1 1 0
Restore 1 1
0 0 0 0 1 0 0 0 0
First cycle
Shift 0 0 0 1 0 0 0 0
Subtract 1 1 1 0 1
Set q_0 1 1 1 1 1
Restore 1 1
0 0 0 1 0 0 0 0 0
Second cycle
Shift 0 0 1 0 0 0 0 0
Subtract 1 1 1 0 1
Set q_0 0 0 0 0 1
0 0 0 1 0 0 0 0 1
Third cycle
Shift 0 0 0 1 0 0 0 0 1
Subtract 1 1 1 0 1 0 0 1
Set q_0 1 1 1 1 1
Restore 1 1
0 0 0 1 0 0 0 1 0
Fourth cycle
Remainder
Quotient

Booth's Non-Restoring Division Algorithm

Deo's



A	Q	M	Count	Operation
0000	1000	0011	4	Initialization
0001	0000	0011		Shift Left
1110	0000	0011		A=A-M
1110	0001	0011	3	Q_0=0
1100	0010	0011		Shift Left
1111	0010	0011		A=A+M
1111	0010	0011	2	Q_0=0

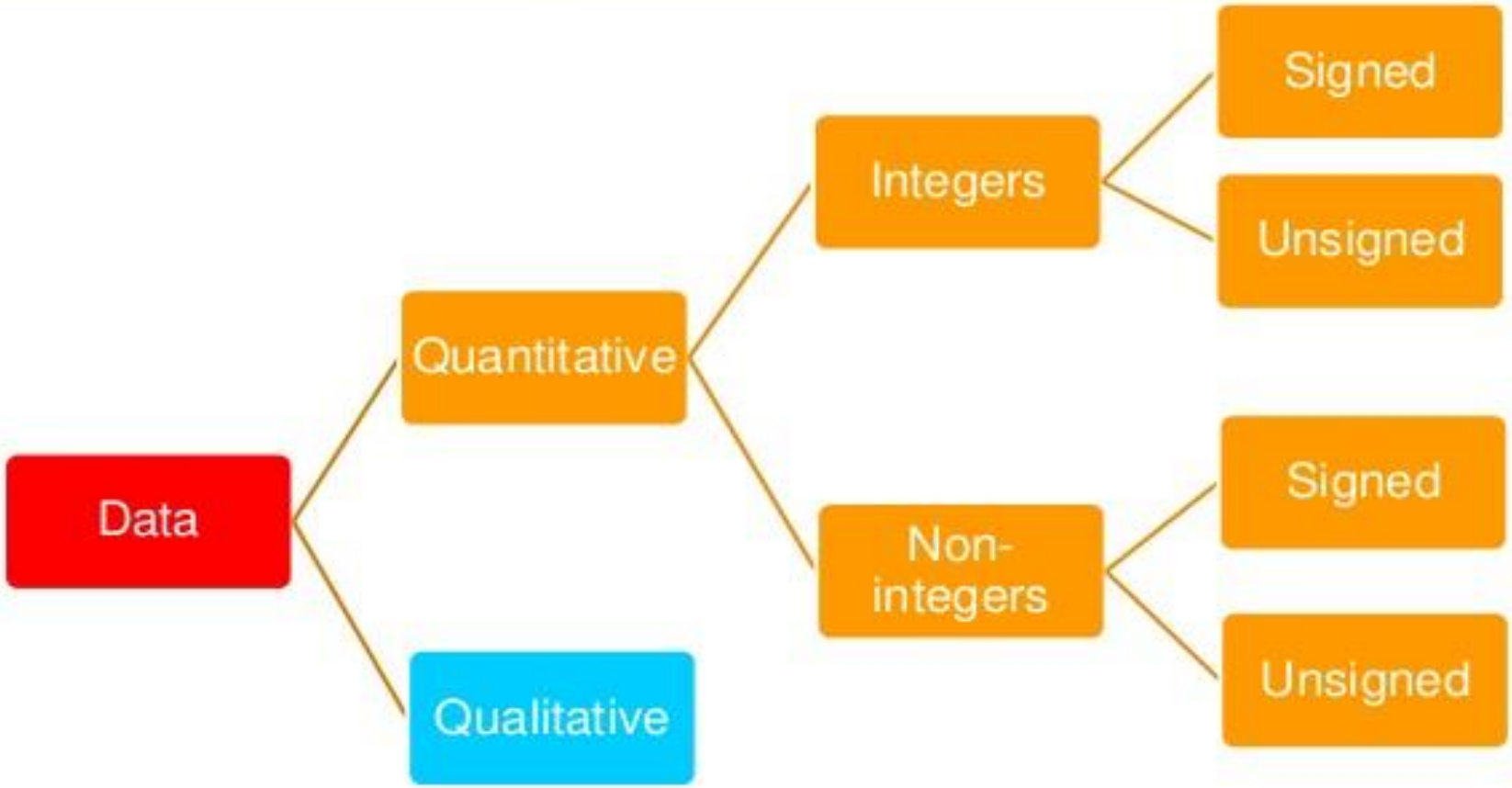
Division Example Using Serial Divider

Data Representation

Outline

- Representing numbers
  - Unsigned
  - Signed
  - Floating point
- Representing characters & symbols
  - ASCII
  - Unicode

Data Representation (Cont.)



Number Systems

- Decimal number system
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Binary number system
  - 0, 1
- Octal number system
  - 0, 1, 2, 3, 4, 5, 6, 7
- Hexadecimal number system
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Quantitative Numbers

- Integers
  - Unsigned 20
  - Signed +20, -20
- Non-integers
  - Floating point numbers - 10.25, 3.33333…, 1/8 = 0.125

Signed Integers

- We need a way to represent negative values
- 3 representations
  - Sign & Magnitude representation (S&M)
  - Complement method
  - Bias notation or Excess notation

1. Sign & Magnitude Representation

- *n*-bit unsigned magnitude & sign bit (S)
- If S
  - 0 – Integer is positive or zero
  - 1 – Integer is negative or zero
- Range $-(2^n-1)$ to $+(2^n-1)$

2. Complement Method

- Base = Radix
  - Radix *r* system means *r* number of symbols
  - e.g., binary numbers have symbols 0, 1
- 2 types
  - *r*'s complement
  - $(r-1)$'s complement
  - Where *r* is radix (base) of number system
- Examples
  - Decimal 9's & 10's complement
  - Binary 1's & 2's complement

Binary addition:-

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Binary subtraction:-

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

1's complement

- Calculated by
  - $(2^n - 1) - m$
- If $m = 0101$
  - 1's complement of m on a 4-bit system

$$\begin{array}{r} 1\ 1\ 1\ 1 \\ \underline{0\ 1\ 0\ 1\ -} \\ 1\ 0\ 1\ 0 \end{array}$$

  - This represents -5 in 1's complement

Finding 1's Complement – Short Cut

- Invert each bit of *m*
- Example
  - *m* = 0 0 1 0 1 0 1 1
  - 1's complement of *m* = 1 1 0 1 0 1 0 0

  - m = 0 0 0 0 0 0 0 0 = 0
  - 1's complement of *m* = 1 1 1 1 1 1 1 1 = -0

- Values range from -127 to +127

Addition with 1's Complement

- If results has a carry add it to LSB (Least Significant Bit)
- Example
  - Add 6 and -3 on a 3-bit system
  - 6 = 1 1 0
  - -3 = 1 0 0 +

    = 1 0 1 0

    → 1 +

    0 1 1

2's Complement

- Doesn't require end-around carry operation as in 1's complement
- 2's complement is formed by
  - Finding 1's complement
  - Add 1 to LSB
- New range is from -128 to +127
  - -128 because of +1 to negative value

Example – 2's Complement

- Find 2's complement of 0101011
  - m = 0 1 0 1 0 1 1
  - = 1 0 1 0 1 0 0
  - 1 +
  - 2's = 1 0 1 0 1 0 1
- Short-cut
  1. Search for the 1st bit with value 1 starting from LSB
  2. Inver all bits after 1st one

Example – 2's Complement (Cont.)

- Add 6 and -5 on a 4-bit system

5 = 0101

-5 = 1011

6 = 0 1 1 0

-5 = 1 0 1 1+

1 0 0 0 1 → 0 0 0 1

Discard

1's vs. 2's Complement

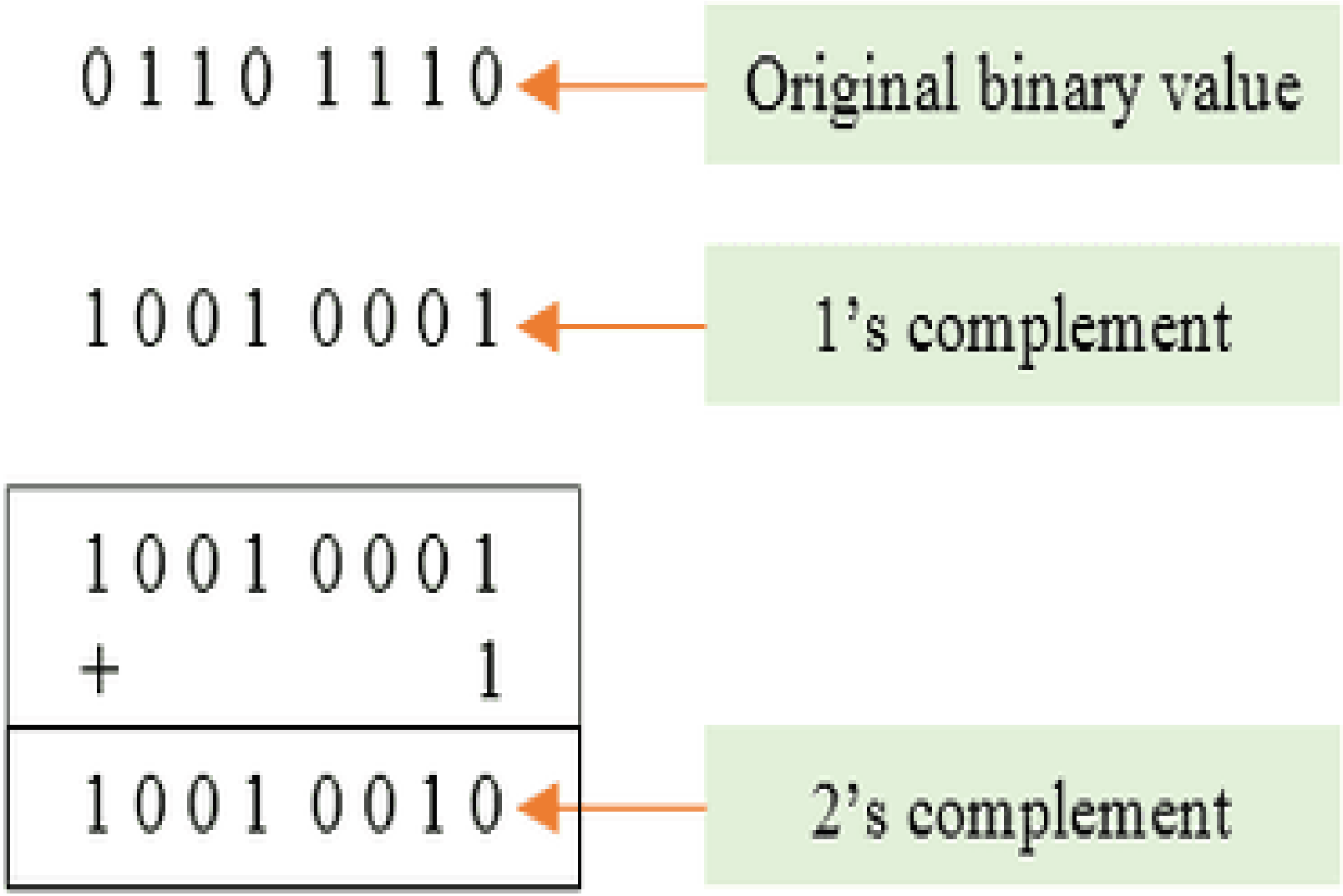
- 1's complement has 2 zeros (+0, -0)
- Value range is less than 2's complement

- 2's complement only has a single zero
- Value range is unequal

- No need of a separate subtract circuit
  - Doing a NOT operation is much more cost effective interms of circuit design
- However, multiplication & division is slow

Booths Multiplication Algorithm

- 2’S complement of number
- Addition or subtraction using 2’s complement
- Arithmetic right shift operation
- Booths multiplication algorithm flowchart
- Booths multiplication algorithm Examples


0 1 1 0 1 1 1 0
Original binary value
1 0 0 1 0 0 0 1
1's complement
1 0 0 1 0 0 0 1
+ 1
1 0 0 1 0 0 1 0
2's complement

Binary representation of 5 is: 0 1 0 1

1's Complement of 5 is: 1 0 1 0

2's Complement of 5 is: (1's Complement + 1) i.e.

1 0 1 0 (1's Compliment)

+ 1

1 0 1 1 (2's Complement i.e. -5)

Binary representation of 3 is: 0 0 1 1

1's Complement of 3 is: 1 1 0 0

2's Complement of 3 is: (1's Complement + 1) i.e.

1 1 0 0 (1's Compliment)

\+ 1

1 1 0 1 (2's Complement i.e. -3)

Now 2 + (-3) = 0 0 1 0 (2 in binary)

\+ 1 1 0 1 (-3)

1 1 1 1 (-1)

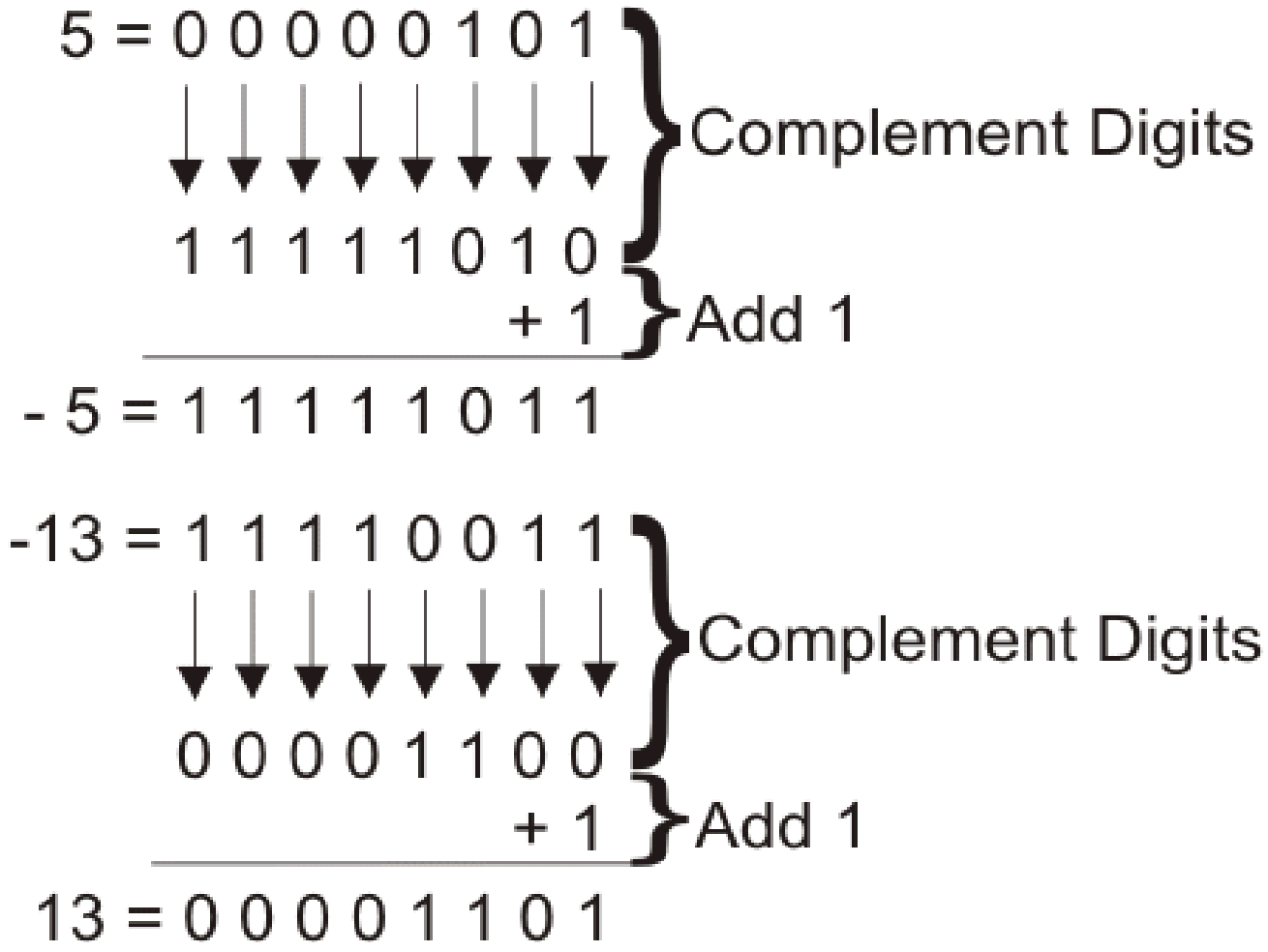
Now, to check whether it is -1 or not simply. takes 2's Complement of -1 and kept -ve sign as it is.

-1 = 1 1 1 1

2's Complement = -(0 0 0 0)

\+ 1

-(0 0 0 1) i.e. -1


5 = 0 0 0 0 0 1 0 1
Complement Digits
1 1 1 1 1 0 1 0
+ 1
Add 1
- 5 = 1 1 1 1 1 0 1 1
-13 = 1 1 1 1 0 0 1 1
Complement Digits
0 0 0 0 1 1 0 0
+ 1
Add 1
13 = 0 0 0 0 1 1 0 1

2s Complement Addition/Subtraction

- **Examples: 4-bit binary system**

```
  +3      0011
+ +4    + 0100
----    -------
  +7      0111
----    -------
```

```
  -2      1110
+ -6    + 1010
----    -------
  -8     11000
----    -------
```

```
  +6      0110
+ -3    + 1101
----    -------
  +3     10011
----    -------
```

```
  +4      0100
+ -7    + 1001
----    -------
  -3      1101
----    -------
```

- Which of the above is/are overflow(s)?

2s complement addition and subtraction

- Simple addition and subtraction
  - simple scheme makes 2s complement the virtually unanimous choice for integer number systems in computers

4	0100	−4	1100
+3	0011	+(−3)	1101
7	0111	−7	11001

4	0100	−4	1100
−3	1101	+3	0011
1	10001	−1	1111

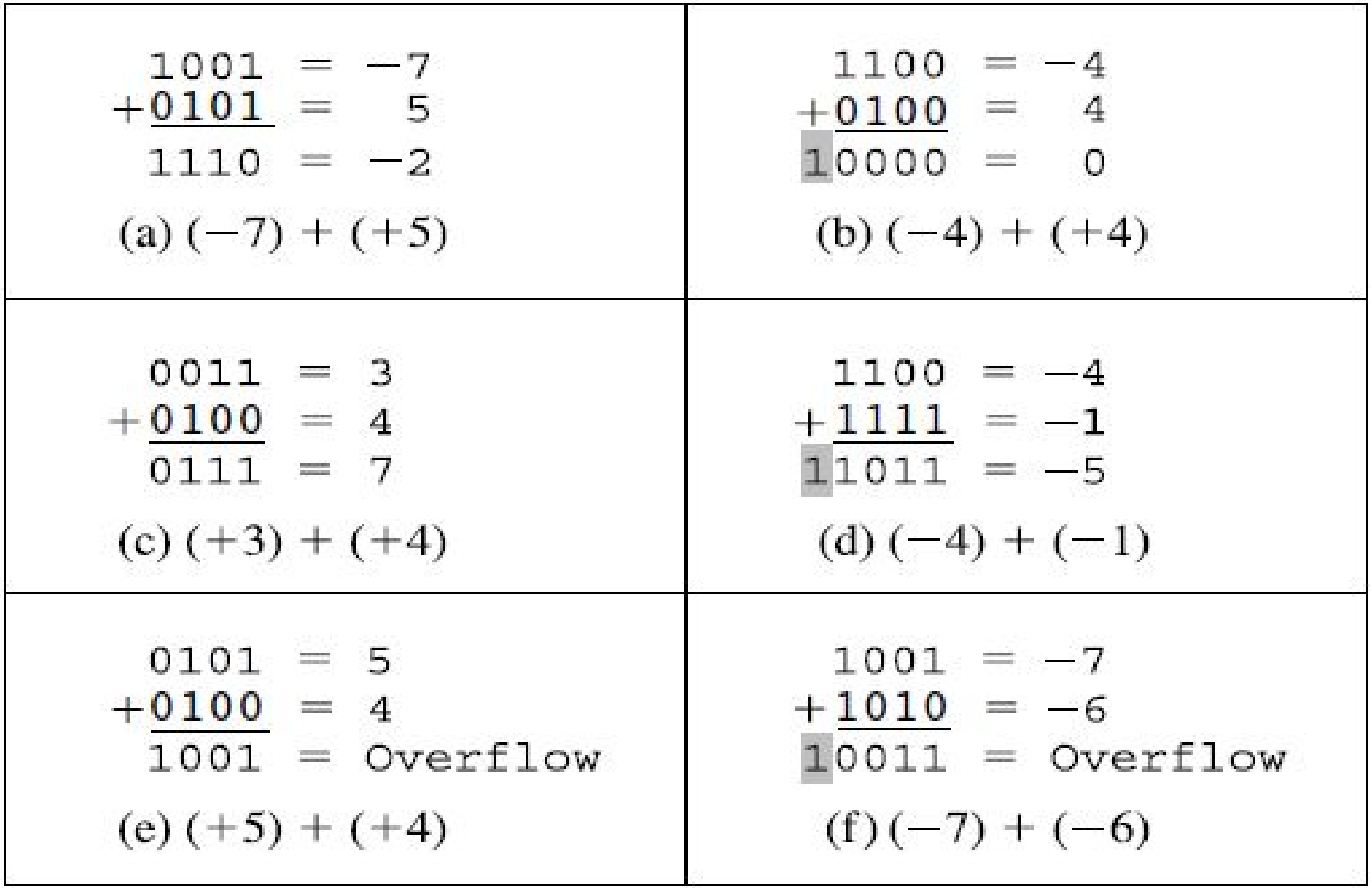
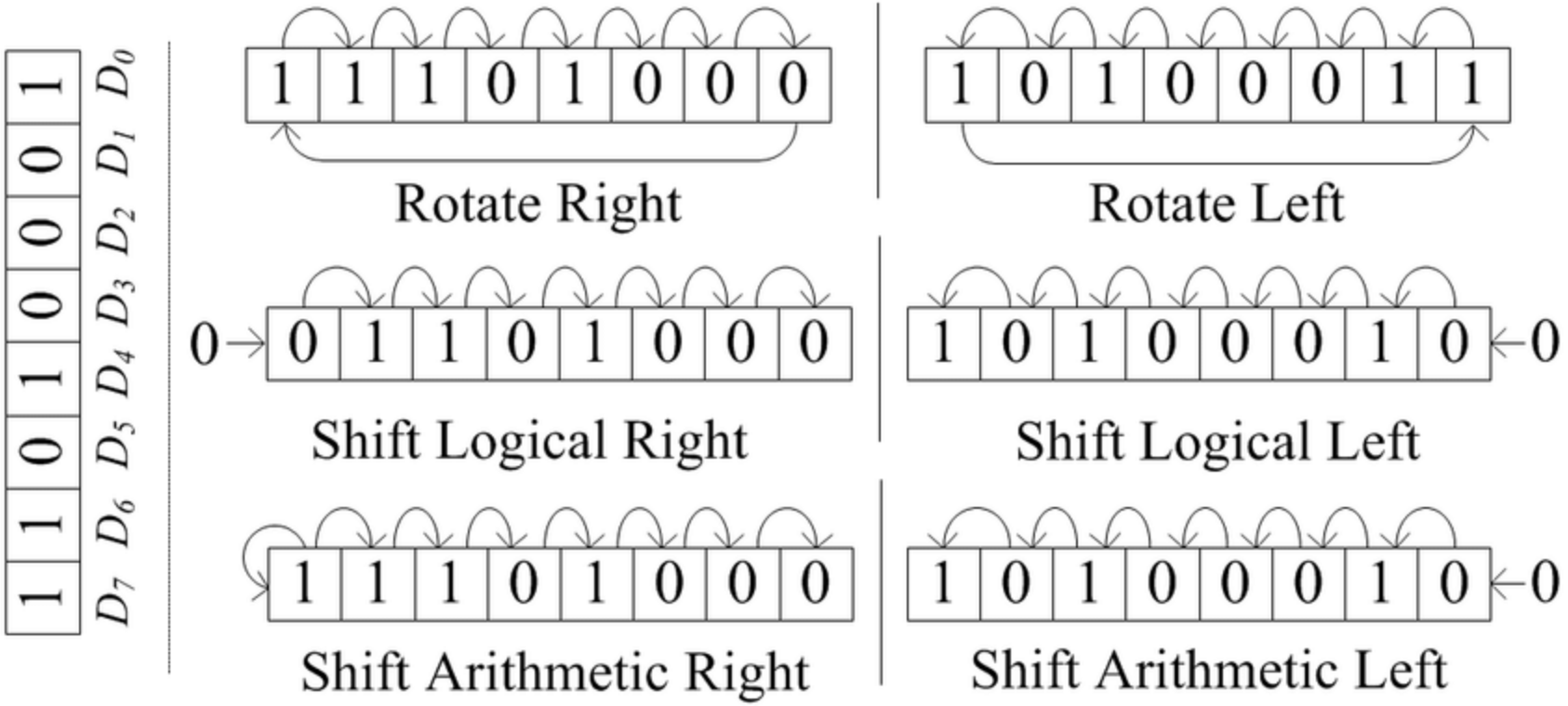
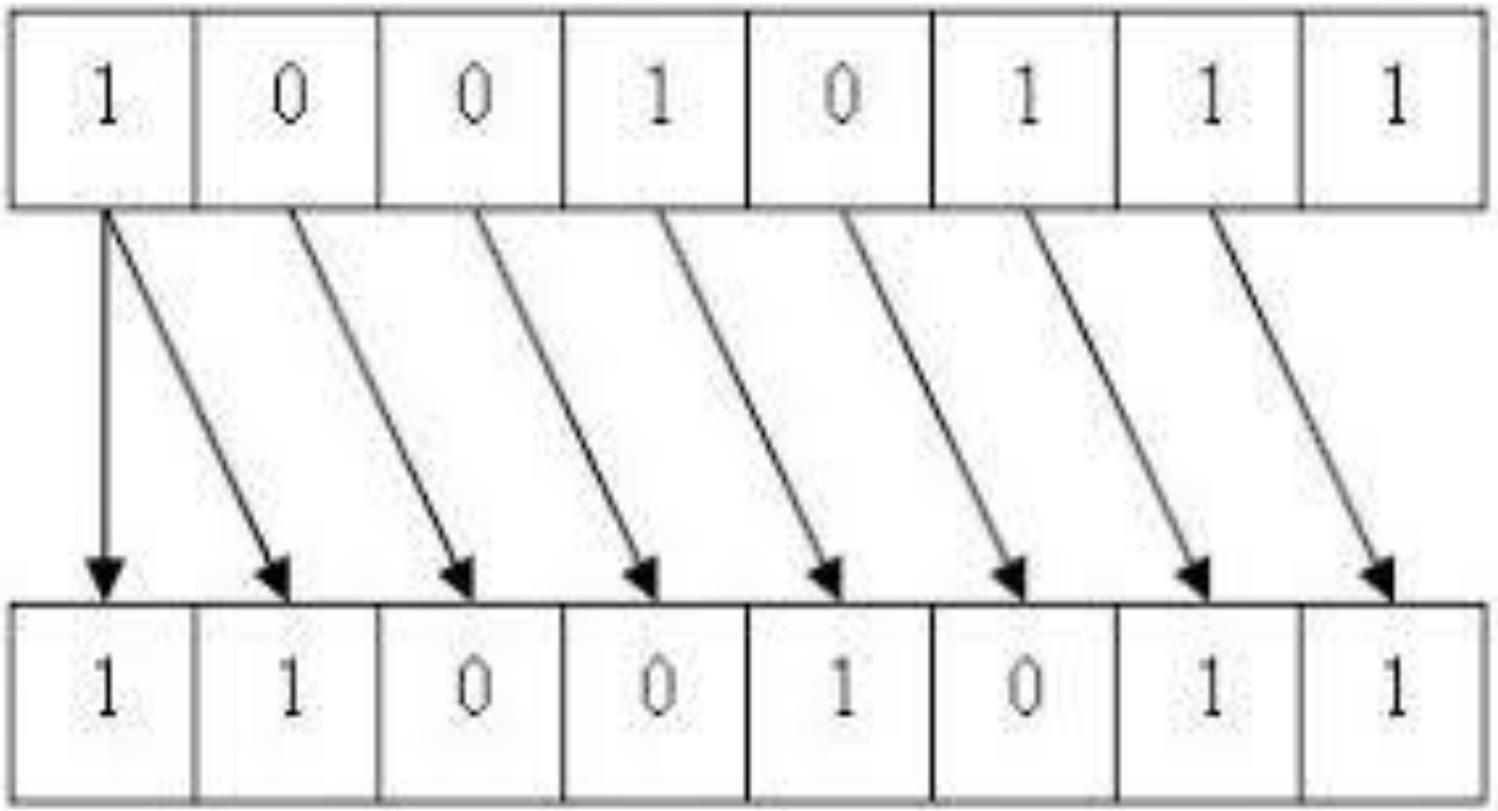
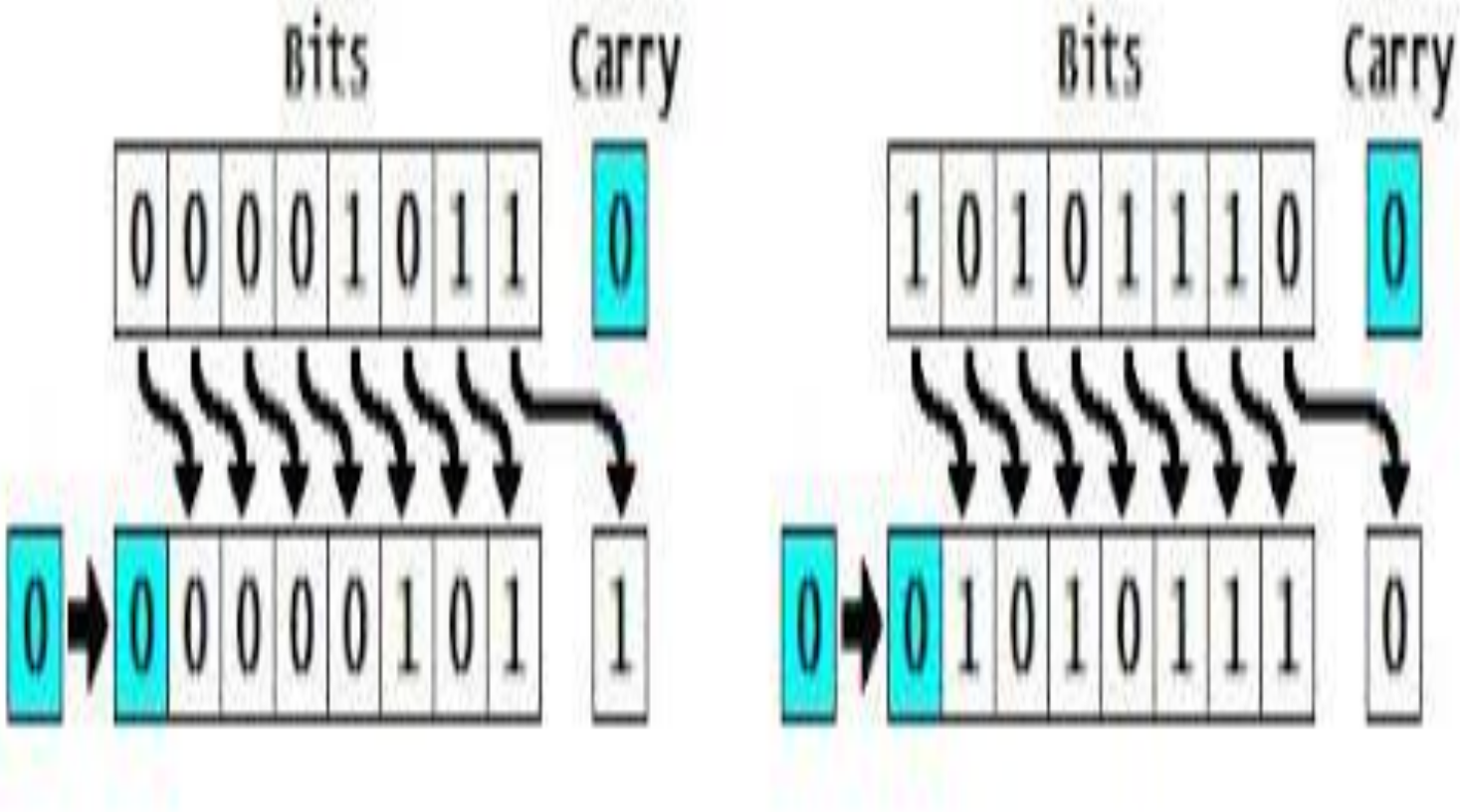


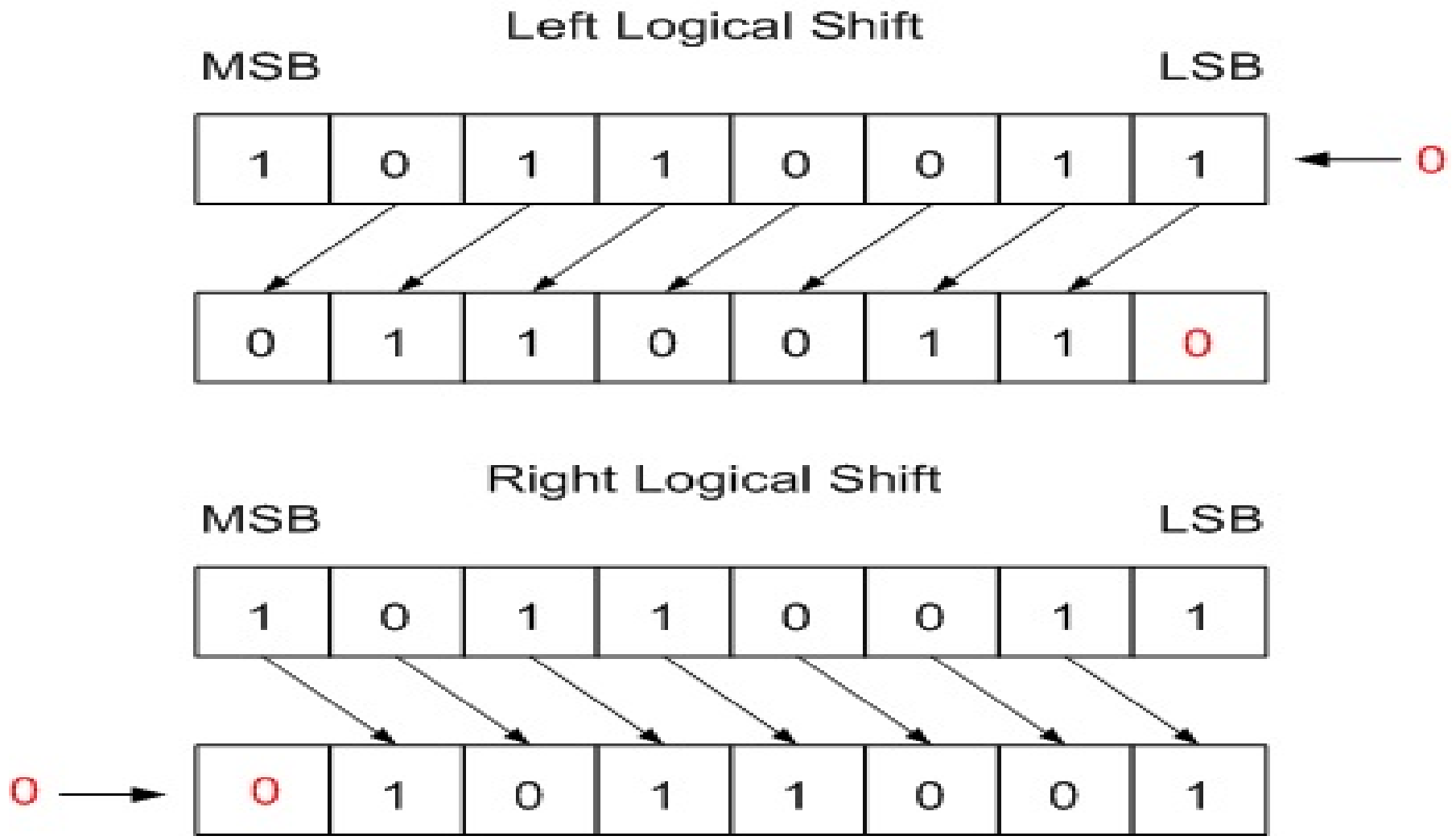
Figure 9.3 Addition of Numbers in Twos Complement Representation


1 1 0 0 0 1
D7 D6 D5 D4 D3 D2 D1 D0
1 1 1 0 1 0 0 0
Rotate Right
1 0 1 0 0 0 1 1
Rotate Left
0
0 1 1 0 1 0 0 0
Shift Logical Right
1 0 1 0 0 0 1 0 0
Shift Logical Left
1 1 1 0 1 0 0 0
Shift Arithmetic Right
1 0 1 0 0 0 1 0 0
Shift Arithmetic Left

Arithmetic's right shift

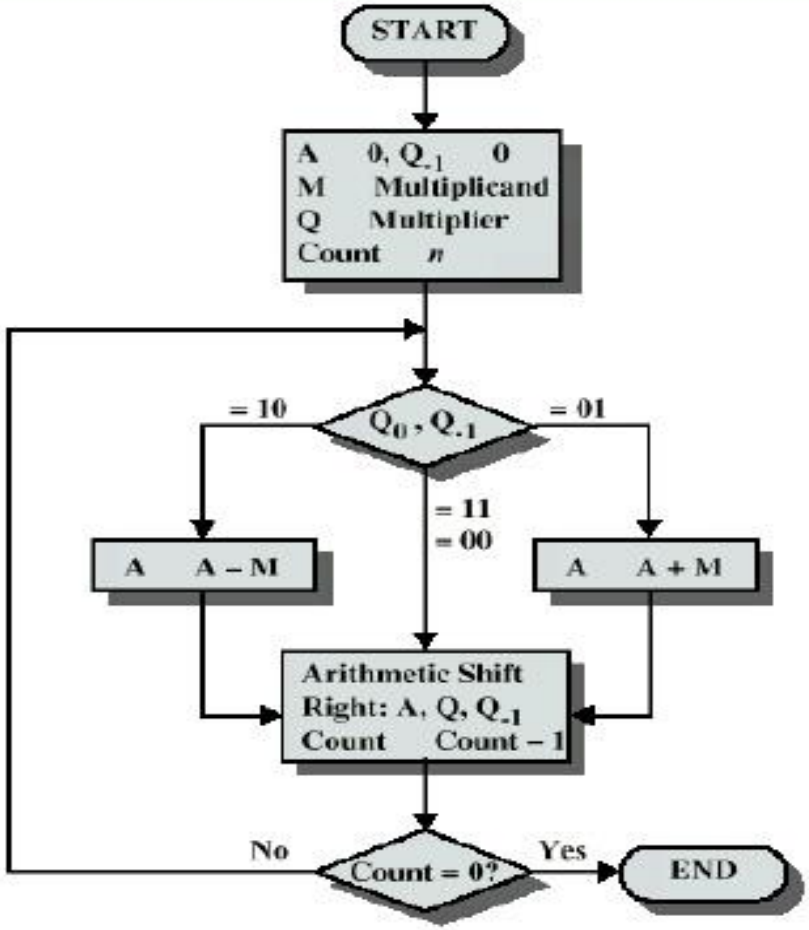



Bits
Carry
0 0 0 0 1 0 1 1
0
0
0 0 0 0 0 1 0 1
1
Bits
Carry
1 0 1 0 1 1 1 0
0
0
0 1 0 1 0 1 1 1
0

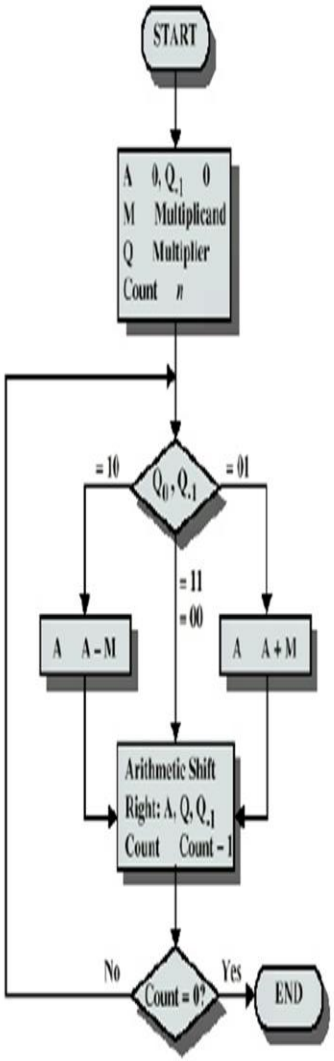

Left Logical Shift
MSB
LSB
1 0 1 1 0 0 1 1
0
0 1 1 0 0 1 1 0
Right Logical Shift
MSB
LSB
1 0 1 1 0 0 1 1
0
0 1 0 1 1 0 0 1

Booths Algorithm flowchart

Booth's Algorithm



Booth's Algorithm



5 X 4

multiplicand (5) 0101 ; multiplier (4) 0100

counter	A	Q	Q_{-1}	operation
	0000	0100	0	initial
4	0000	0010	0	ASR
3	0000	0001	0	ASR
2	1011	0001	0	A = A - B
	1101	1000	1	ASR
1	0010	1000	1	A = A + B
	0001	0100	0	ASR

0101 → 1010 + 1 = 1011

1101 + 0101 = 0010 (carry 1)

$2^4 + 2^2 = 20$

2s Complement Addition/Subtraction

- **Examples: 4-bit binary system**

```
  +3      0011
+ +4    + 0100
----    -------
  +7      0111
----    -------
```

```
  -2      1110
+ -6    + 1010
----    -------
  -8     11000
----    -------
```

```
  +6      0110
+ -3    + 1101
----    -------
  +3     10011
----    -------
```

```
  +4      0100
+ -7    + 1001
----    -------
  -3      1101
----    -------
```

- Which of the above is/are overflow(s)?

2s complement addition and subtraction

- Simple addition and subtraction
  - simple scheme makes 2s complement the virtually unanimous choice for integer number systems in computers

4	0100	−4	1100
+3	0011	+(−3)	1101
7	0111	−7	11001

4	0100	−4	1100
−3	1101	+3	0011
1	10001	−1	1111

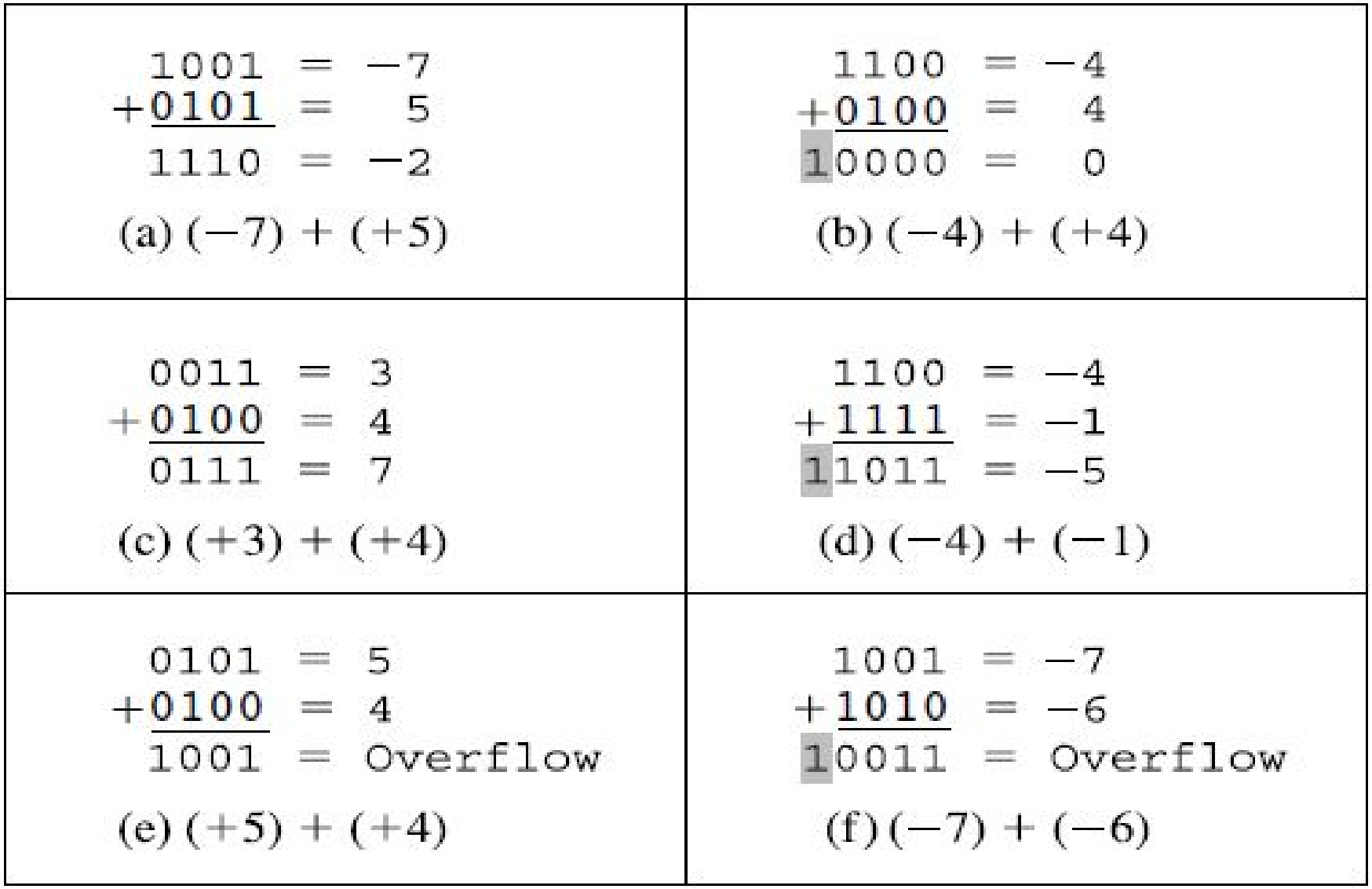


Figure 9.3 Addition of Numbers in Twos Complement Representation

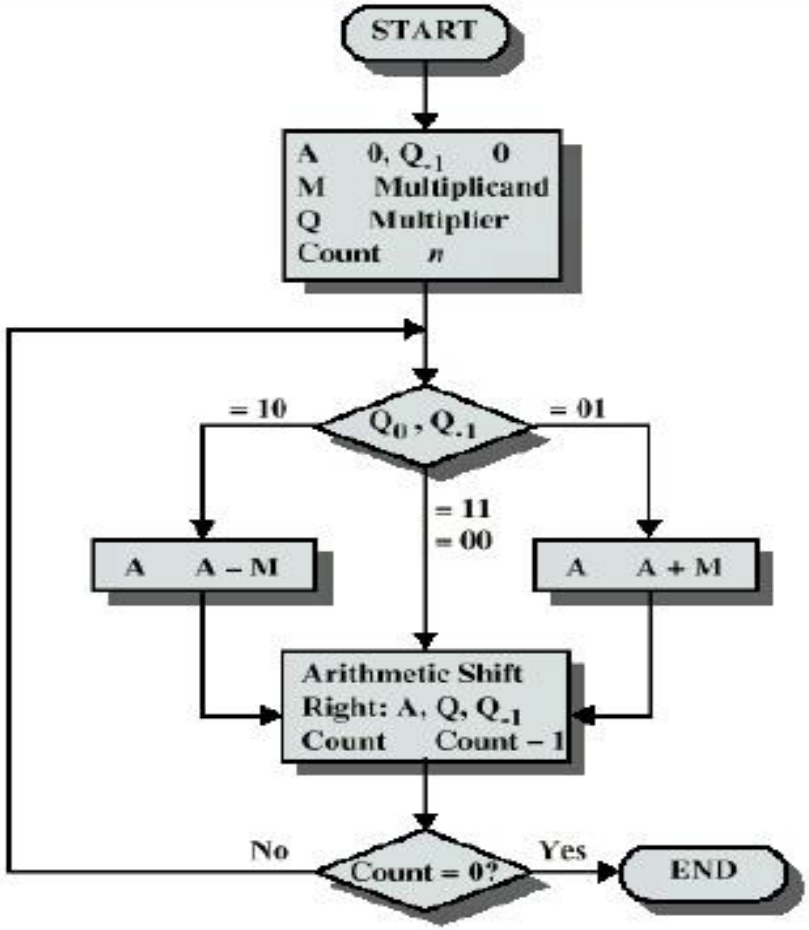
Booths multiplication

Points to remember

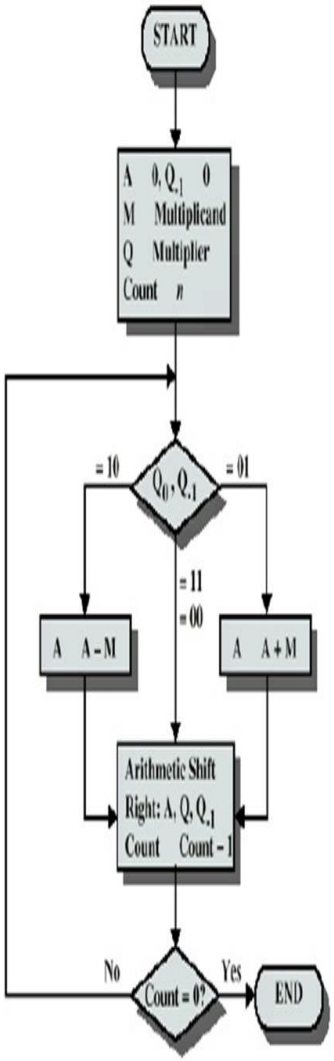
- When using Booth's Algorithm:
  - You will need twice as many bits in your **product** as you have in your original two **operands**.
  - The **leftmost bit** of your operands (both your multiplicand and multiplier) is a SIGN bit, and cannot be used as part of the value.

Booths Algorithm flowchart

Booth's Algorithm



Booth's Algorithm



5 X 4

multiplicand (5) 0101, multiplier (4) 0100

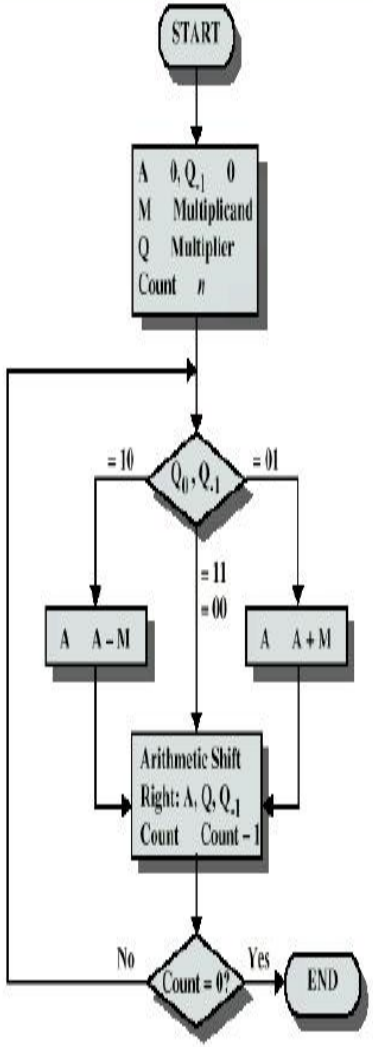
counter	A	Q	Q_{-1}	operation
	0000	0100	0	initial
4	0000	0010	0	ASR
3	0000	0001	0	ASR
2	1011	0001	0	A = A - B
	1101	1000	1	ASR
1	0010	1000	1	A = A + B
	0001	0100	0	ASR

0101 → 1010 + 1 = 1011

1101 + 0101 = 0010

$2^4 + 2^2 = 20$

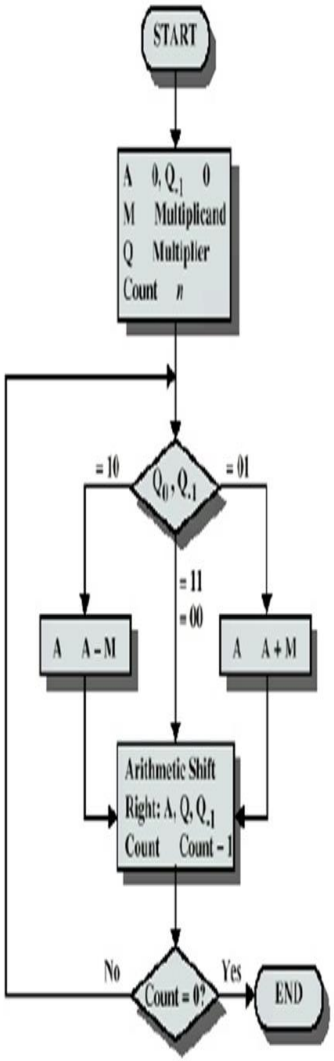
Booth's Algorithm

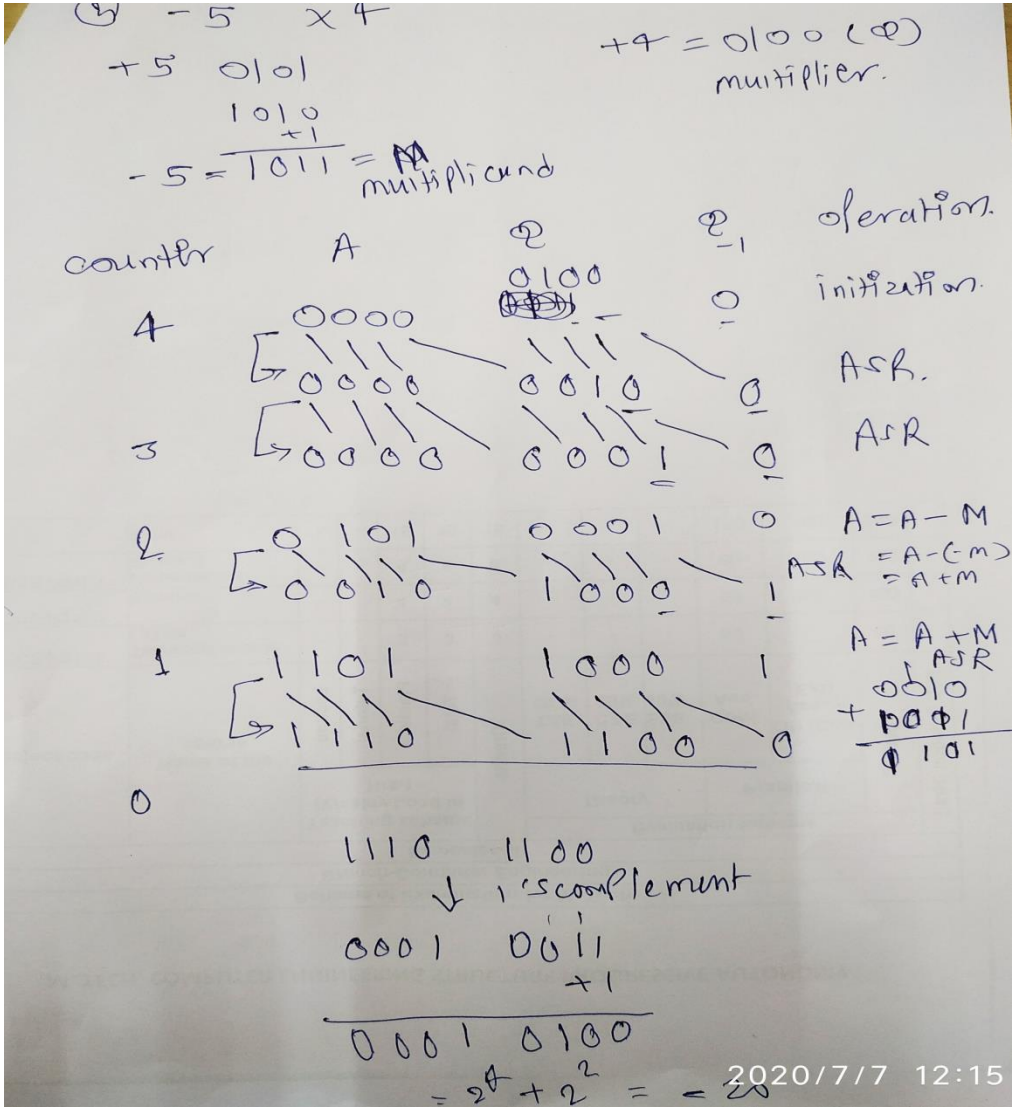


Example of Booth's Algorithm

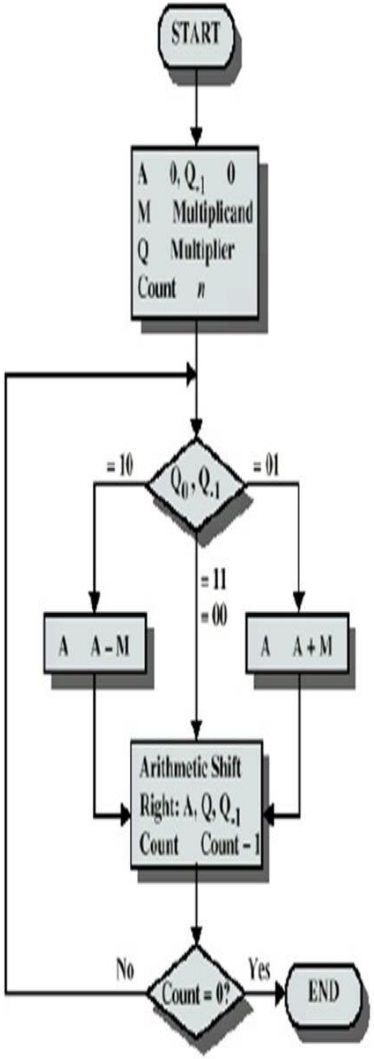
A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

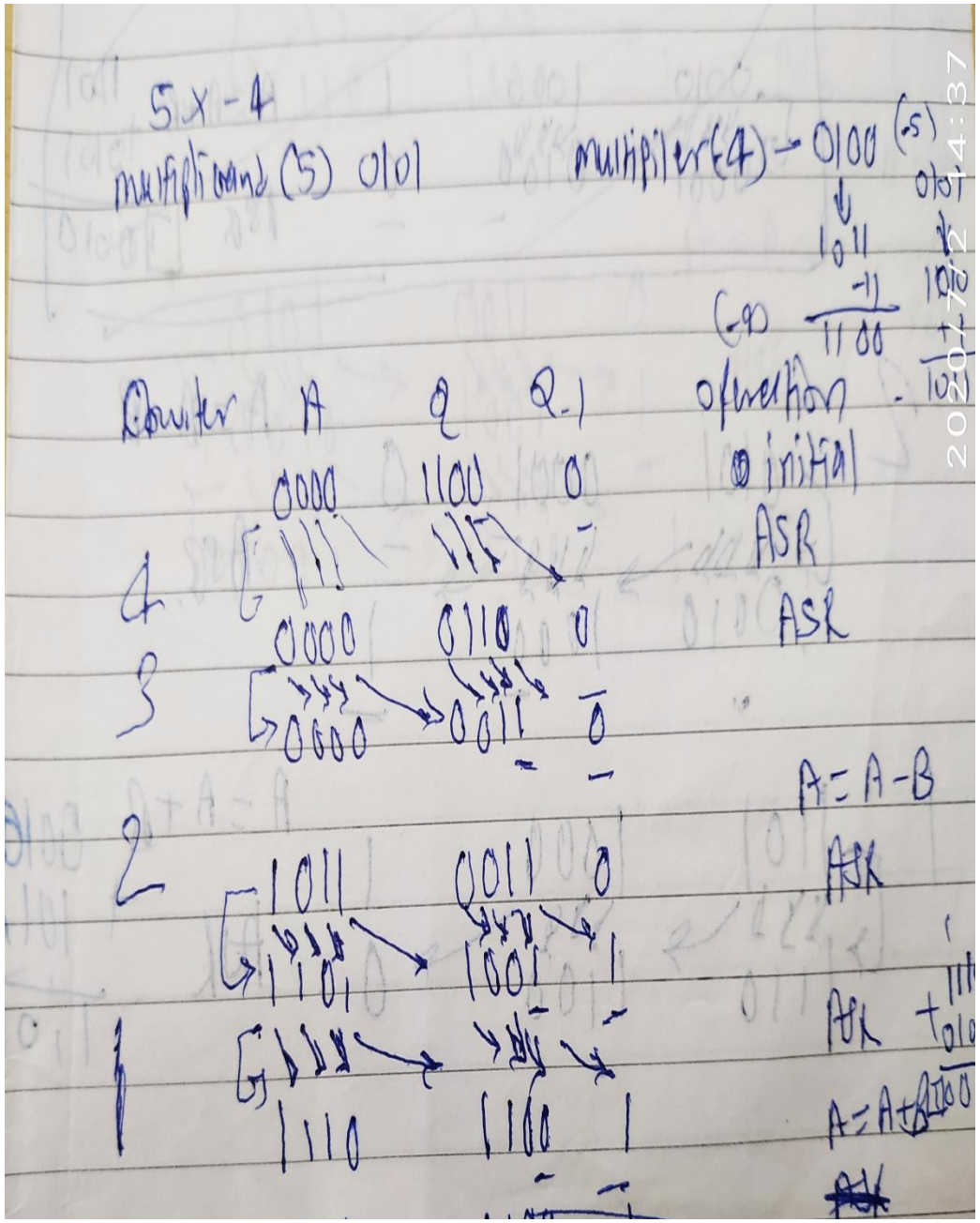
Booth's Algorithm



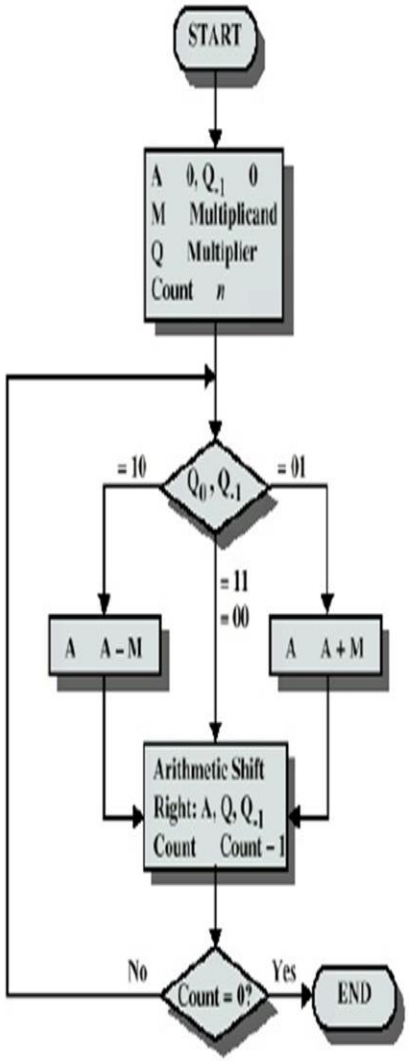


Booth's Algorithm





Booth's Algorithm



1) -5 x -4

multiplicand (-5) → 0101 → 1010 +1 → 1011

multiplier (-4) → 0100 → 1011 +1 → 1100

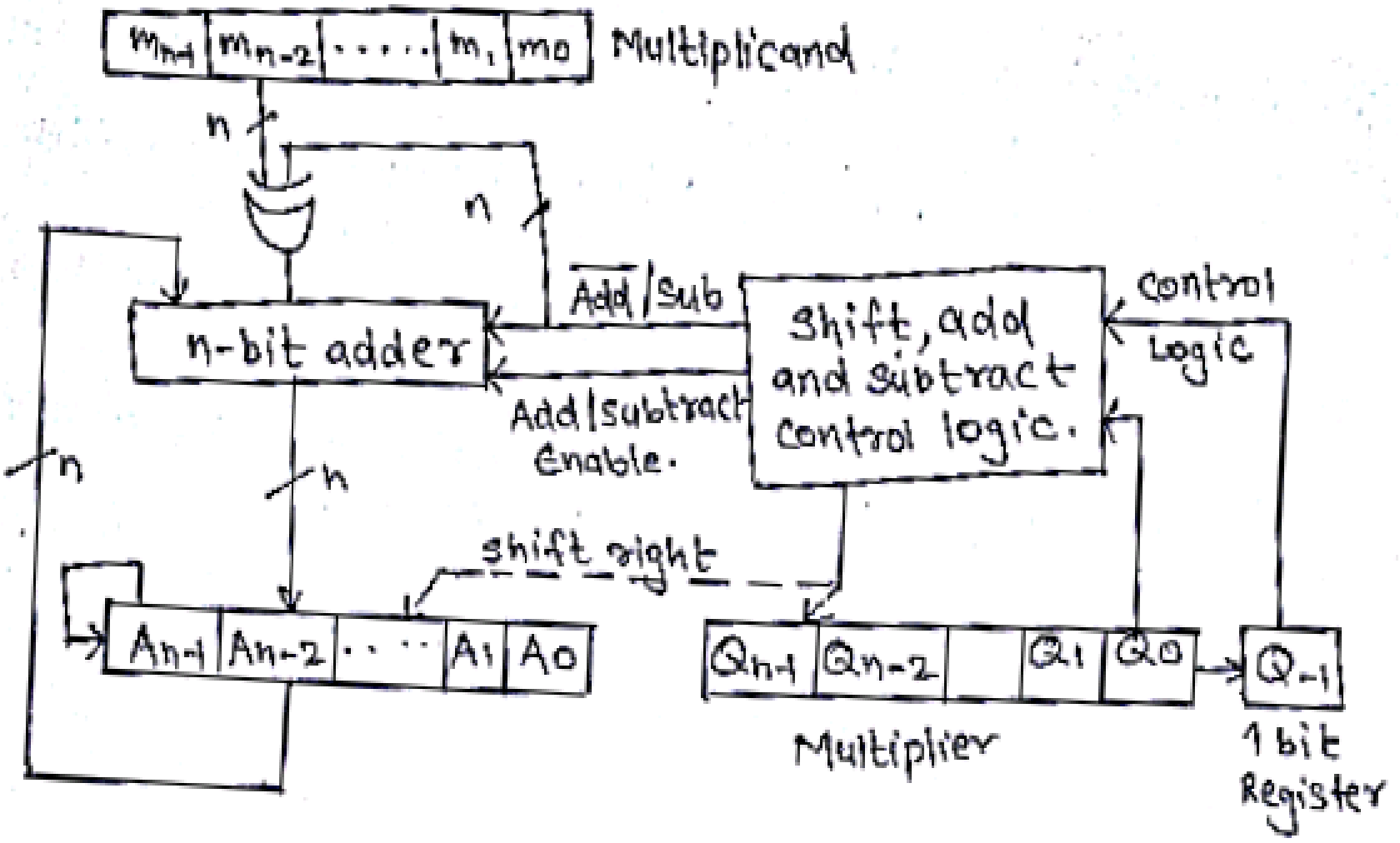
Counter	A	Q	Q-1	operation
	0000	1100	0	initial
	0000	0110	0	ASR
	0000	0011	0	ASR
	1011	0011	0	A = A - B
	1101	1001	1	ASR
	1110	1100	1	ASR

B - 0101

A = A - B = 0 - (-5) + 6

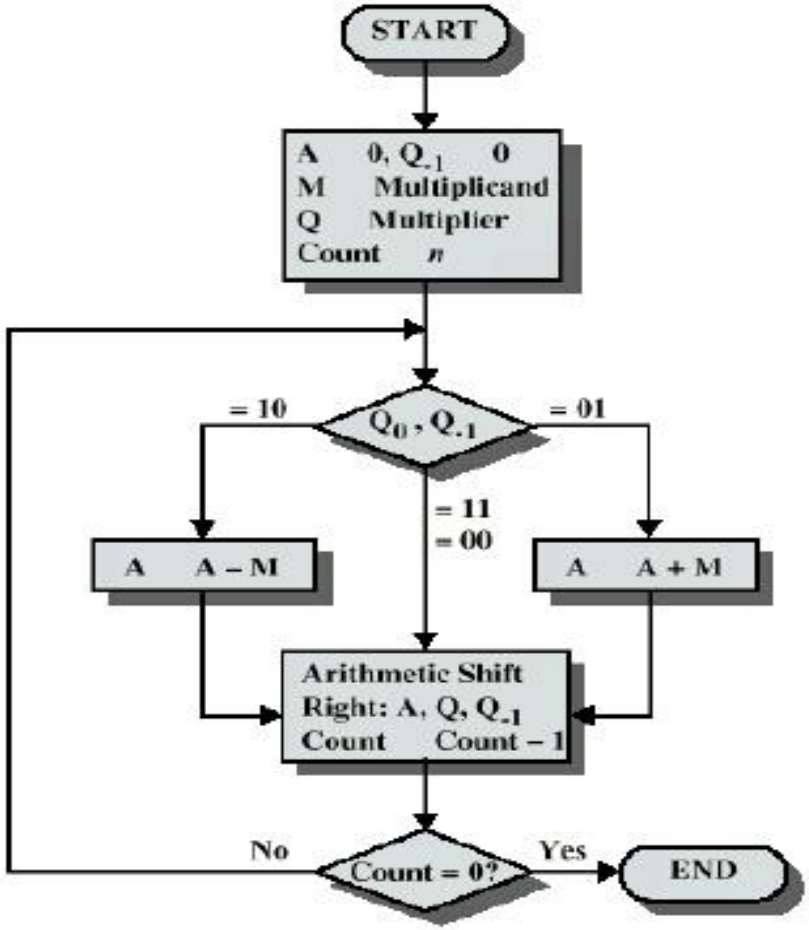
0101	0011	0	
0010	1001	1	ASR
0001	0100	1	ASR

Booths algorithm hardware implementation

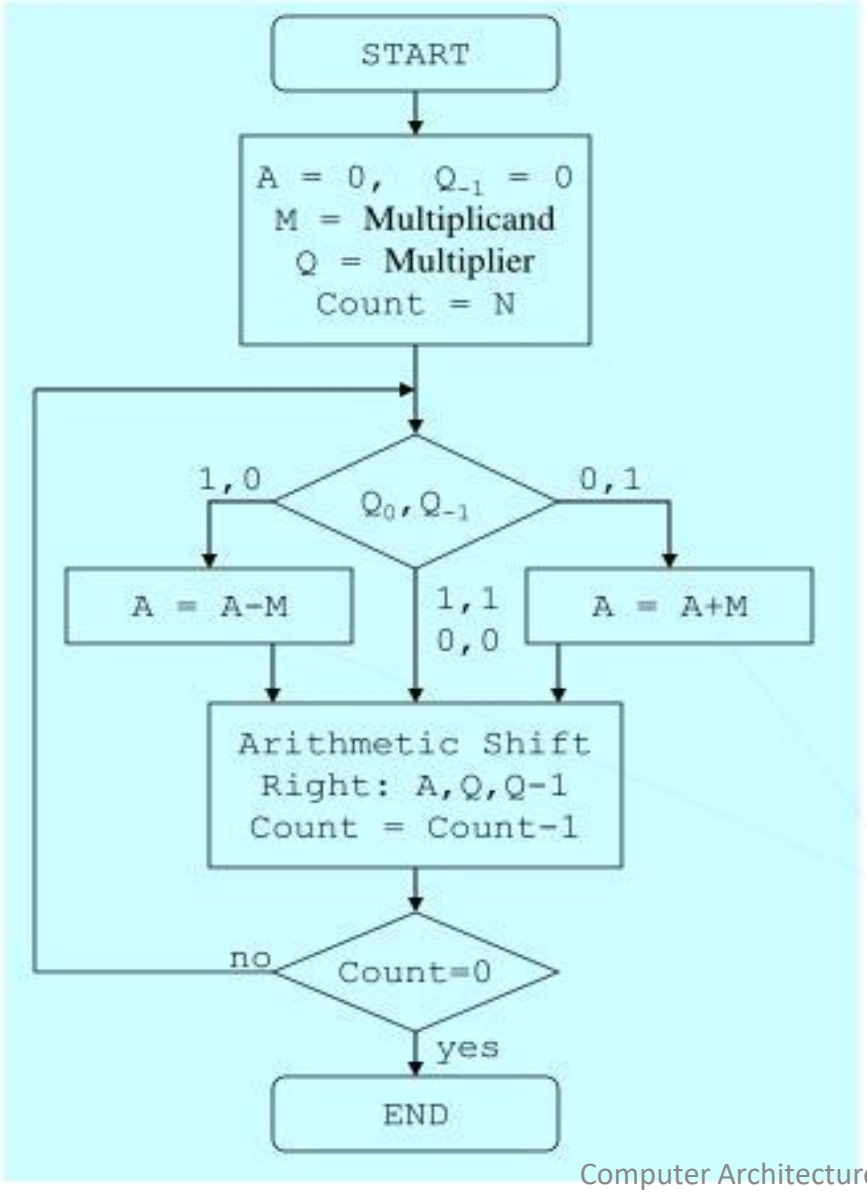


Booths Algorithm flowchart

Booth's Algorithm



Booth's Algorithm for 2's Complement Multiplication



M = 0101 -M = 1011
Q = 0110
N = 4

A	Q	Q_{-1}	N
0000	0110	0	4
0000	0011	0	3
1011	0011	0	
1101	1001	1	2
1110	1100	1	1
0011	1100	1	
0001	1110	0	0

Answer is in A,Q

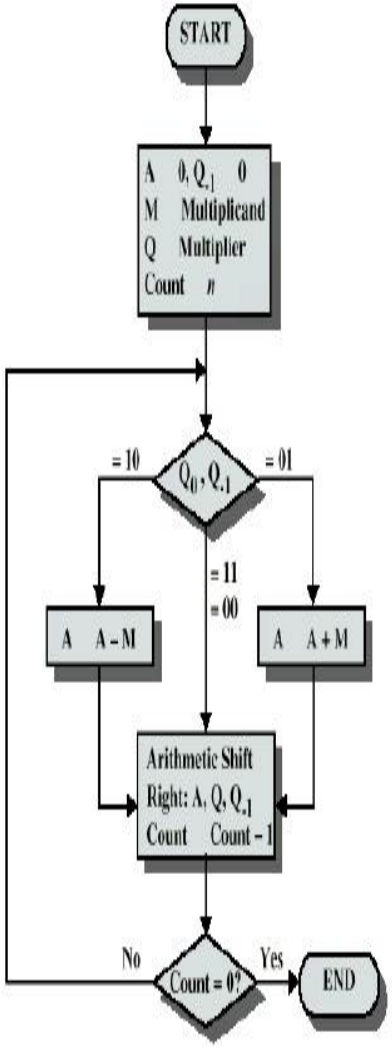
$00011110_2 = 30_{10}$

Two's complement multiplication can be performed using Booth's Algorithm.

To save time, both M and -M can be computed ahead and provided as one of the inputs to the adders for A-M and A+M.

Instead of a loop, the math processor can provide separate hardware for each stage of the multiplication algorithm.

Booth's Algorithm



	A	Q	Q_{-1} M=0010
6 * 2 = 12	0000	0010	0
6 = 0110 2 = 0010	0000	0001	0
Q_0 Q_{-j} 0 0 1 1 1 0 0 1	1010 1101	0001 0000	0 1
	0011 0001	0000 1000	1 0
	0000	1100	0

To begin

- Decide which operand will be the **multiplier** and which will be the **multiplicand**
- Convert both operands to **two's complement** representation using X bits
  - X must be at least one more bit than is required for the binary representation of the numerically larger operand
- Begin with a product that consists of the multiplier with an additional X leading zero bits

Example

- In the week by week, there is an example of multiplying **2 x (-5)**
- For our example, let's reverse the operation, and multiply **(-5) x 2**
  - The numerically larger operand (5) would require 3 bits to represent in binary (101). So we must use AT LEAST 4 bits to represent the operands, to allow for the sign bit.
- Let's use 5-bit 2's complement:
  - -5 is 11011 (multiplier)
  - 2 is 00010 (multiplicand)

Beginning Product

- The multiplier is:

  11011

- Add 5 leading zeros to the **multiplier** to get the **beginning product**:

  00000 11011

Step 1 for each pass

- Use the **LSB** (least significant bit) and the **previous LSB** to determine the arithmetic action.
  - If it is the FIRST pass, use **0** as the previous LSB.

- Possible arithmetic actions:
  - **00** → no arithmetic operation
  - **01** → add multiplicand to left half of product
  - **10** → subtract multiplicand from left half of product
  - **11** → no arithmetic operation

Step 2 for each pass

- Perform an **arithmetic right shift** (ASR) on the entire product.

- NOTE: For X-bit operands, Booth's algorithm requires X passes.

Example

- Let's continue with our example of multiplying **(-5) x 2**
- Remember:
  - -5 is 11011 (multiplier)
  - 2 is 00010 (multiplicand)

- And we added 5 leading zeros to the **multiplier** to get the **beginning product**:

```
00000 11011
```

Example continued

- Initial Product and previous LSB

  00000 11011 0

(Note: Since this is the first pass, we use 0 for the previous LSB)

- Pass 1, Step 1: Examine the last 2 bits

  00000 11011 0

The last two bits are 10, so we need to:

subtract the **multiplicand** from left half of product

Example: Pass 1 continued

- Pass 1, Step 1: Arithmetic action

```
(1)  00000     (left half of product)
    -00010     (mulitplicand)
     11110     (uses a phantom borrow)
```

- Place result into **left half** of product

```
11110 11011  0
```

Example: Pass 1 continued

- Pass 1, Step 2: ASR (arithmetic shift right)
  - Before ASR

    11110 11011 0

  - After ASR

    11111 01101 1

  (left-most bit was 1, so a 1 was shifted in on the left)

- Pass 1 is complete.

Example: Pass 2

- Current Product and previous LSB

  11111 01101 1

- Pass 2, Step 1: Examine the last 2 bits

  11111 01101 1

  The last two bits are 11, so we do NOT need to perform an arithmetic action --

  just proceed to step 2.

Example: Pass 2 continued

- Pass 2, Step 2: ASR (arithmetic shift right)
  - Before ASR

    `11111 01101 1`

  - After ASR

    `11111 10110 1`

  (left-most bit was 1, so a 1 was shifted in on the left)

- Pass 2 is complete.

Example: Pass 3

- Current Product and previous LSB

  11111 10110 1

- Pass 3, Step 1: Examine the last 2 bits

  11111 10110 1

  The last two bits are 01, so we need to:

  add the **multiplicand** to the left half of the product

Example: Pass 3 continued

- Pass 3, Step 1: Arithmetic action

```
(1)  11111    (left half of product)
    +00010    (mulitplicand)
     00001    (drop the leftmost carry)
```

- Place result into **left half** of product

```
00001 10110 1
```

Example: Pass 3 continued

- Pass 3, Step 2: ASR (arithmetic shift right)
  - Before ASR

    `00001 10110 1`
  - After ASR

    `00000 11011 0`

  (left-most bit was 0, so a 0 was shifted in on the left)

- Pass 3 is complete.

Example: Pass 4

- Current Product and previous LSB

  00000 11011 0

- Pass 4, Step 1: Examine the last 2 bits

  00000 11011 0

The last two bits are 10, so we need to:

subtract the **multiplicand** from the left half of the product

Example: Pass 4 continued

- Pass 4, Step 1: Arithmetic action

```
(1)  00000     (left half of product)
    -00010     (mulitplicand)
     11110     (uses a phantom borrow)
```

- Place result into **left half** of product

```
11110 11011  0
```

Example: Pass 4 continued

- Pass 4, Step 2: ASR (arithmetic shift right)
  - Before ASR

    11110 11011 0
  - After ASR

    11111 01101 1

  (left-most bit was 1, so a 1 was shifted in on the left)

- Pass 4 is complete.

Example: Pass 5

- Current Product and previous LSB

  11111 01101 1

- Pass 5, Step 1: Examine the last 2 bits

  11111 01101 1

  The last two bits are 11, so we do NOT need to perform an arithmetic action --

  just proceed to step 2.

Example: Pass 5 continued

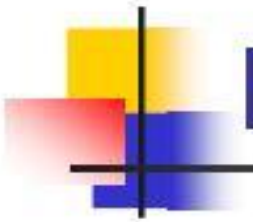
- Pass 5, Step 2: ASR (arithmetic shift right)
  - Before ASR

    `11111 01101 1`
  - After ASR

    `11111 10110 1`

  (left-most bit was 1, so a 1 was shifted in on the left)

- Pass 5 is complete.



Final Product

- We have completed 5 passes on the 5-bit operands, so we are done.

- Dropping the previous LSB, the resulting **final product** is:

  11111 10110

Verification

- To confirm we have the correct answer, convert the 2's complement **final product** back to decimal.
- Final product: `11111 10110`
- Decimal value: `-10`

which is the CORRECT product of:

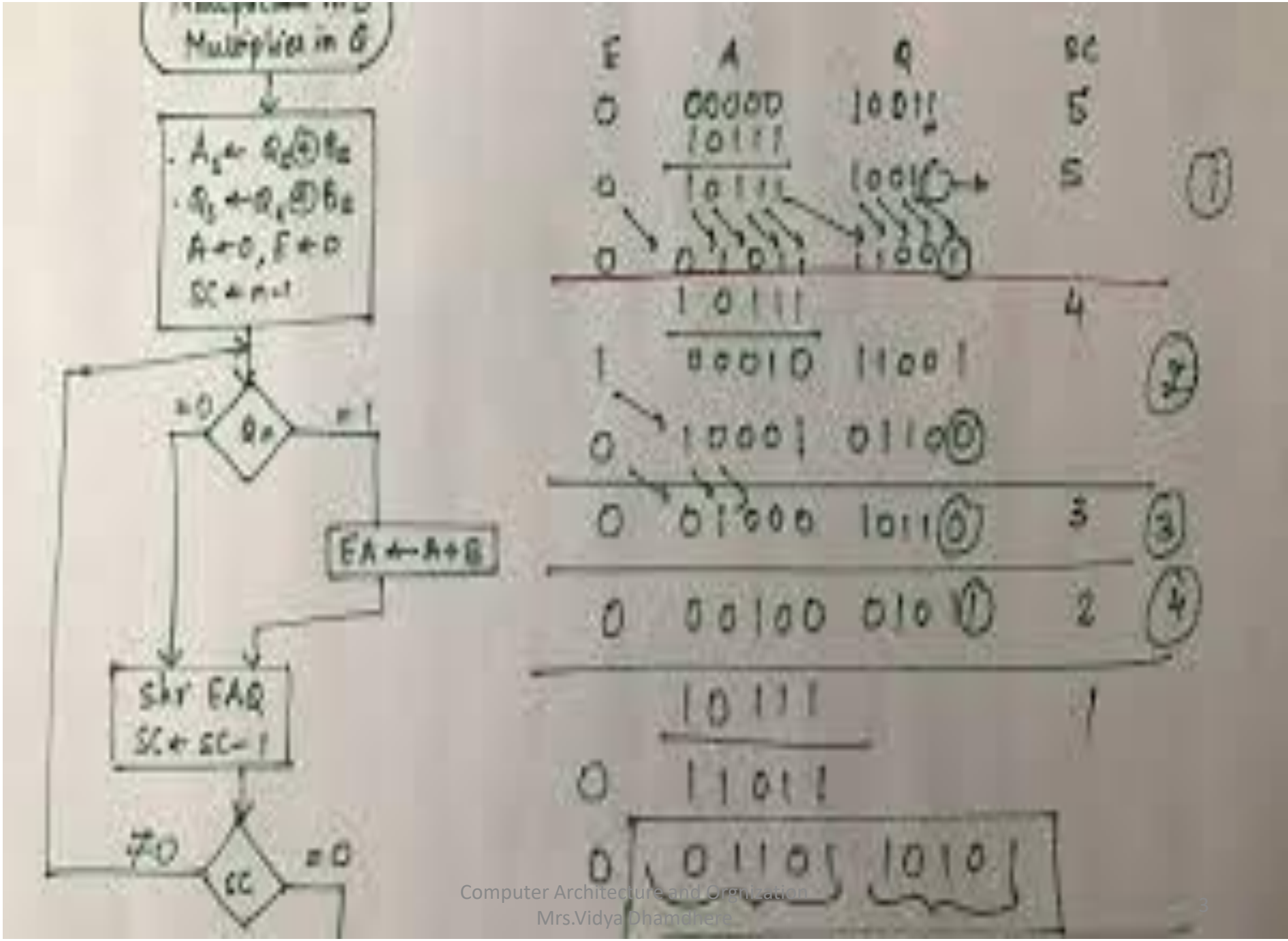
`(-5) x 2`

UNIT II Data path unit

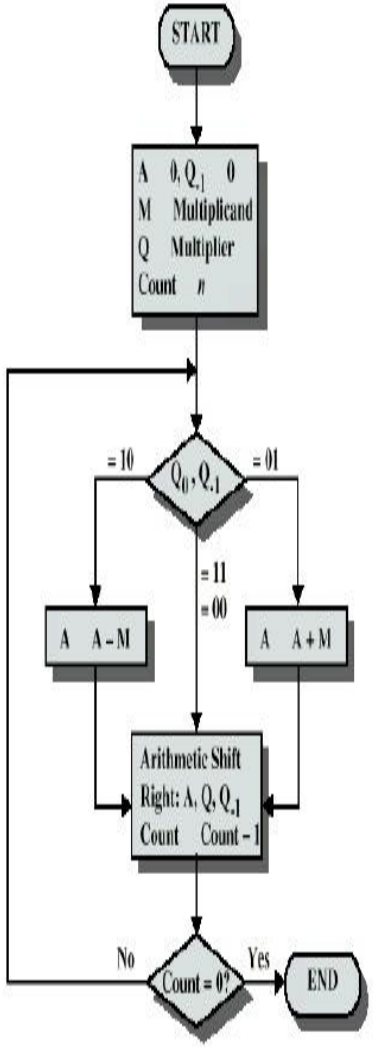
Data Representation, Fixed and Floating point numbers, Signed numbers, Fixed-Point Arithmetic, Booths Algorithm-multiplication , Arithmetic Logic unit, Floating point representations, IEEE standards, Floating point arithmetic.

Booths Multiplication Algorithm

- 2’S complement of number
- Addition or subtraction using 2’s complement
- Arithmetic right shift operation
- Booths multiplication algorithm flowchart
- Booths multiplication algorithm Examples


Multiplier in B
A ← A ⊕ B
Q ← Q ⊕ B
A ← 0, E ← 0
SC ← n
Q=
=0
=1
EA ← A + B
shr EAQ
SC ← SC − 1
SC
≠0
=0
E A Q SC
0 00000 10111 5
10111
0 10111 1001 5
0 10111 1001 1
10111 4
0 01010 11001 2
0 10000 01100
0 01000 00110 3
0 00100 00011 1 2
10111
0 11011 1
0 0110 1010

Booth's Algorithm



Example of Booth's Algorithm

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

multiplication of -5×4 $(-5 = 1011) = B$

steps	A	Q	Q-1	operation
	0000	0100	0	Initial
1	0000	0010	0	Right shift
2.	0000	0001	0	Right shift
3.	0101	0001	0	$A \leftarrow A - B$
	0010	1000	1	Right shift
4.	1101	1000	1	$A \leftarrow A + B$
	1110	1100	0	Right shift

(-20)

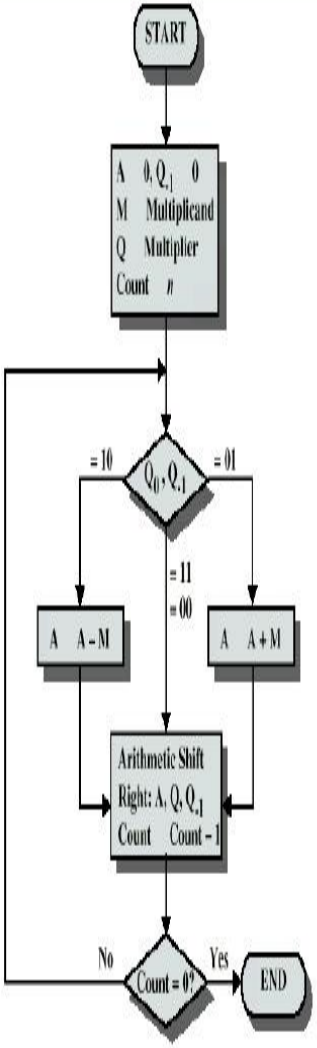
verify:

$-5 \times 4 = -20$

$5 = 0101$

0001 00

Booth's Algorithm



Booth's Algorithm Example

Multiply 4 X -9

00100
X 10111

	LMProd RMProd	
	00000101110	
Step 0	+11100	(sub 4 = add -4)
Step 1	11100101110	
	11110010111	(shift after add)
Step 2	11111001011	(shift without add)
Step 3	11111100101	(shift without add)
	+00100	(add +4)
	00011100101	
Step 4	00001110010	(shift after add)
	+11100	(sub 4 = add -4)
	11101110010	
Step 5	11110111001	(shift after add)

Remember
4 = 00100
-4 = 11100

1111011100 = -(0000100011 + 1)
= -(0000100100)
= -36 = 4 X -9!

CASE 2 – Negative Multiplier: $(5)_{10} X (-4)_{10}$

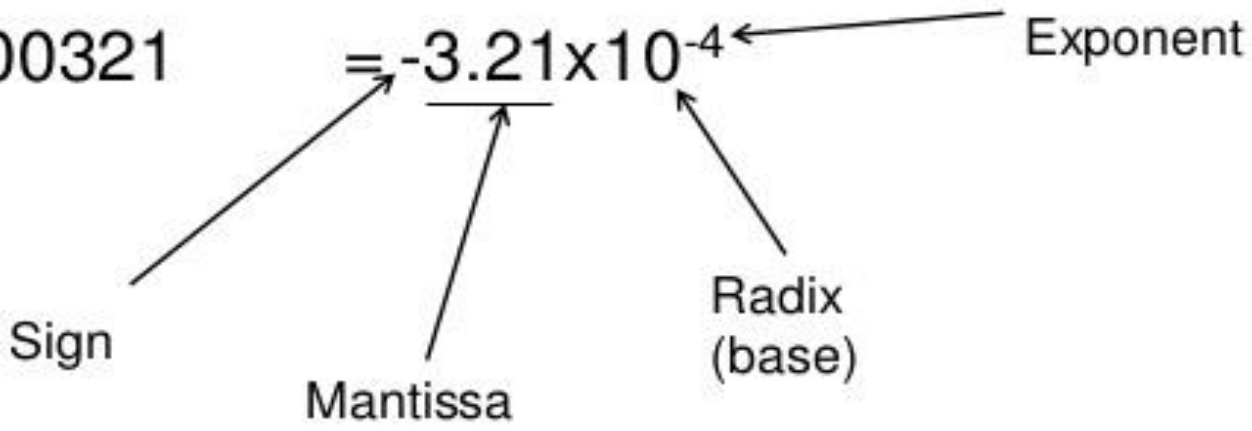
Multiplicand (B) = 0101(5) , Multiplier (Q) =1100(- 4)

Steps	A	Q	Q-1	Operation
	0000	1100	0	Initial
Step 1 :	0000	0110	0	Right Shift
Step 2 :	0000	0011	0	Right Shift
Step 3 :	1011	0011	0	A ← A - B
	1101	1001	1	Right Shift
Step 4:	1110	1100	1	Right Shift
	1110	**1100**		

Result : 11101100 = -20 (2's complement of 20)

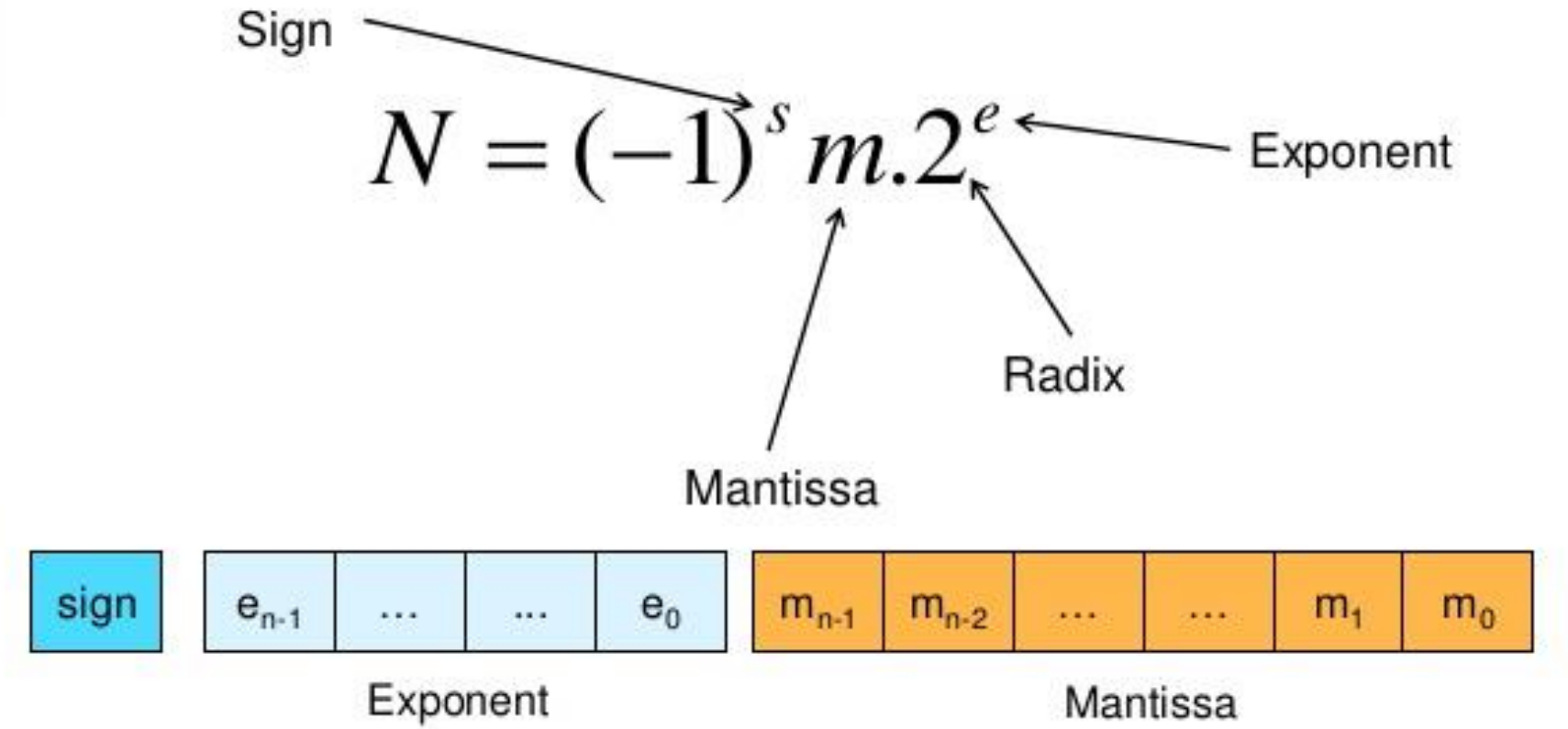
Floating Point Numbers

- We needed to represent fractional values & values beyond $2^n - 1$
- $+3207.23 = 3.20723 \times 10^3$
- $-0.000321 = -3.21 \times 10^{-4}$



28

Floating Point Numbers (Cont.)



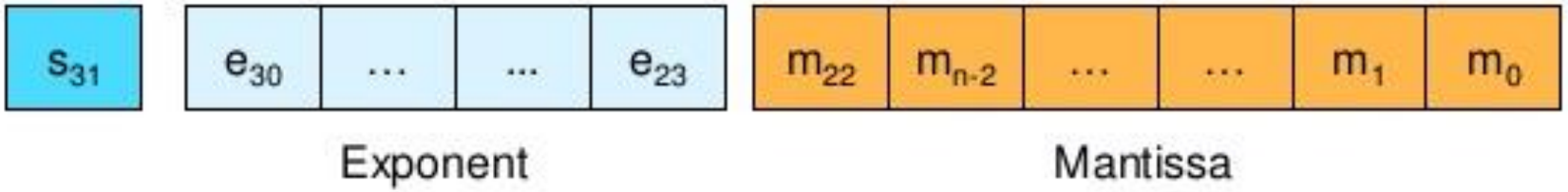
IEEE Floating Point Standard (FPS)

- 2 standards

1. Single precision
   - 32-bits
     - 23-bit mantissa
     - 8-bit exponent
     - 1-bit sign

2. Double precision
   - 64-bits
     - 52-bit mantissa
     - 11-bit exponent
     - 1-bit sign

IEEE Floating Point Standard (Cont.)



$$N = (-1)^s [1.m] \times 2^{E-127}$$

- E – 127 = *e*
- E = *e* + 127
- What is stored is E

Single Precision

- 23-bit mantissa
  - *m* ranges from 1 to $(2 - 2^{-23})$
- 8-bit exponent
  - E ranges from 1 to 254
  - 0 & 255 reserved to represent $-\infty$, $+\infty$, NaN
  - *e* ranges from -126 to +127
- Maximum representable number
  - $2 \times 2^{127} = 2^{128}$

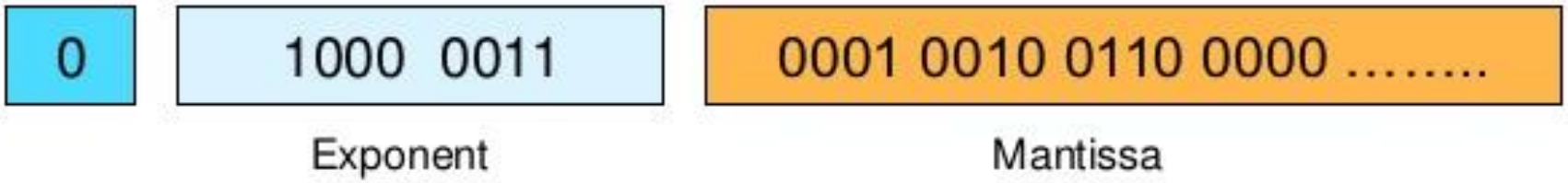
Example – Single Precision

- IEEE Single Precision Representation of 17.15_{10}
- Find 17.15_{10} in binary

$$17.15_{10} = 10001.0010011_2$$
$$= 1.00010010011 \times 2^4$$

$E = e + 127$

$E = 4 + 127 = 131$



Floating-Point Numbers

- Floating-point number systems circumvent the limitation of having a constant number of integer and fractional bits
  - They allow the representation of very large and very small numbers

- The binary point floats to the right of the most significant 1
  - Similar to decimal scientific notation
  - For example, write 273_{10} in scientific notation:
    - Move the decimal point to the right of the most significant digit and increase the exponent:

$$273 = 2.73 \times 10^2$$

- In general, a number is written in scientific notation as:

$$\pm M \times B^E$$

Where,

- M = mantissa
- B = base
- E = exponent
- In the example, M = 2.73, B = 10, and E = 2 (that is, $+2.73 \times 10^2$)

Floating-Point Numbers

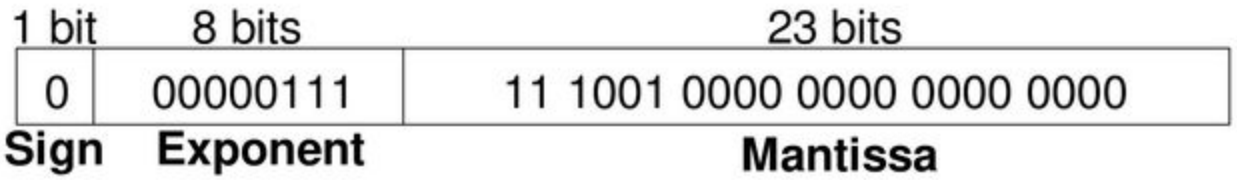
- Floating-point number representation using 32 bits
  - 1 sign bit
  - 8 exponent bits
  - 23 bits for the mantissa.

- The following slides show three versions of floating-point representation with 228_{10} using a 32-bit
  - The final version is called the IEEE 754 floating-point standard

Floating-Point Representation #1

- First, convert the decimal number to binary
  - $228_{10} = 11100100_2 = 1.11001 \times 2^7$

- Next, fill in each field in the 32-bit:
  - The sign bit (1 bit) is positive, so 0
  - The exponent (8 bits) is 7 (111)
  - The mantissa (23 bits) is 1.11001

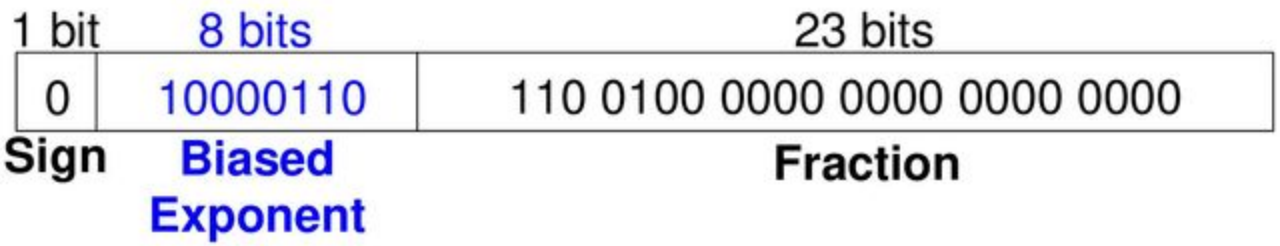


Floating-Point Representation #2

- You may have noticed that the first bit of the mantissa is always 1, since the binary point floats to the right of the most significant 1
  - Example: $228_{10} = 11100100_2 = \mathbf{1}.11001 \times 2^7$
- Thus, storing the most significant 1 (also called the implicit leading 1) is redundant information
- We can store just the fraction parts in the 23-bit field
  - Now, the leading 1 is implied

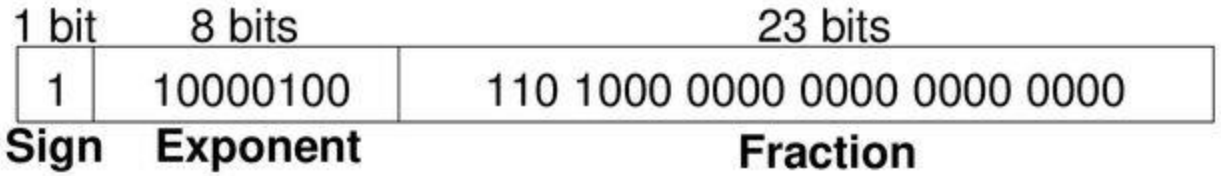
Floating-Point Representation #3

- The exponent needs to represent both positive and negative
- The final change is to use a biased exponent
  - The IEEE 754 standard uses a bias of 127
  - Biased exponent = bias + exponent
    - For example, an exponent of 7 is stored as $127 + 7 = 134 = 10000110_2$
- Thus , 228_{10} using the IEEE 754 32-bit floating-point standard is



Example

- Represent -58_{10} using the IEEE 754 floating-point standard
  - First, convert the decimal number to binary
    - $58_{10} = 111010_2 = 1.1101 \times 2^5$
  - Next, fill in each field in the 32-bit number
    - The sign bit is negative (1)
    - The 8 exponent bits are $(127 + 5) = 132 = 10000100_{(2)}$
    - The remaining 23 bits are the fraction bits: $11010000..000_{(2)}$



  - It is 0xC2680000 in the hexadecimal form
    - Check this out with the result of the sample program in the slide# 3

Floating-Point Number Precision

- The IEEE 754 standard also defines 64-bit double-precision that provides greater precision and greater range
  - **Single-Precision** (use the `float` declaration in C language)
    - 32-bit notation
    - 1 sign bit, 8 exponent bits, 23 fraction bits
    - bias = 127
    - It spans a range from $\pm 1.175494 \times 10^{-38}$ to $\pm 3.402824 \times 10^{38}$
  - **Double-Precision** (use the `double` declaration in C language)
    - 64-bit notation
    - 1 sign bit, 11 exponent bits, 52 fraction bits
    - bias = 1023
    - It spans a range from $\pm 2.22507385850720 \times 10^{-308}$ to $\pm 1.79769313486232 \times 10^{308}$
- Most general purpose processors (including Intel and AMD processors) provide hardware support for double-precision floating-point numbers and operations

Double Precision Example

- Represent -58_{10} using the IEEE 754 double precision
  - First, convert the decimal number to binary
    - $58_{10} = 111010_2 = 1.1101 \times 2^5$
  - Next, fill in each field in the 64-bit number
    - The sign bit is negative (1)
    - The 11 exponent bits are $(1023 + 5) = 1028 = 10000000100_{(2)}$
    - The remaining 52 bits are the fraction bits: $11010000..000_{(2)}$
  - It is 0xC04D0000_00000000 in the hexadecimal form
    - Check this out with the result of the sample program in the slide# 4

IEEE Floating Point Standard (FPS)

- 2 standards

1. Single precision
   - 32-bits
     - 23-bit mantissa
     - 8-bit exponent
     - 1-bit sign

2. Double precision
   - 64-bits
     - 52-bit mantissa
     - 11-bit exponent
     - 1-bit sign

Single Precision

- 23-bit mantissa
  - *m* ranges from 1 to $(2 - 2^{-23})$
- 8-bit exponent
  - E ranges from 1 to 254
  - 0 & 255 reserved to represent $-\infty$, $+\infty$, NaN
  - *e* ranges from -126 to +127
- Maximum representable number
  - $2 \times 2^{127} = 2^{128}$

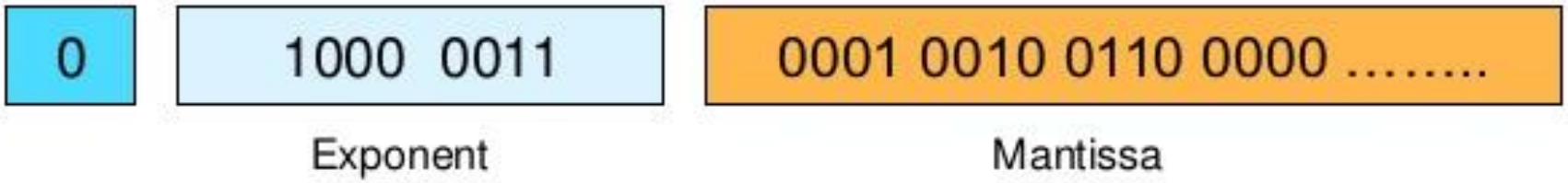
Example – Single Precision

- IEEE Single Precision Representation of 17.15_{10}
- Find 17.15_{10} in binary

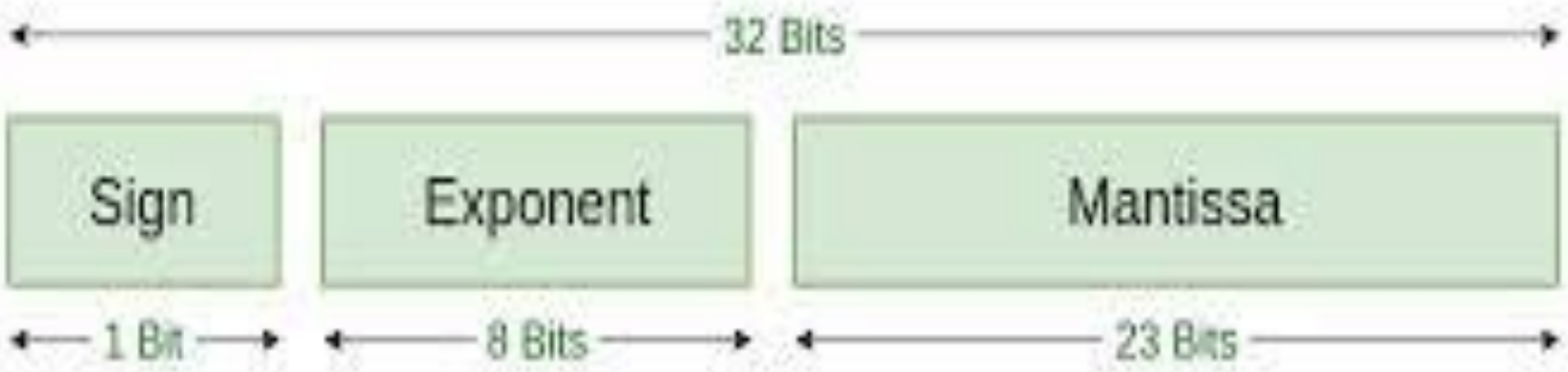
$$17.15_{10} = 10001.0010011_2$$
$$= 1.00010010011 \times 2^4$$

$E = e + 127$

$E = 4 + 127 = 131$



IEEE standard Single precision



Single Precision
IEEE 754 Floating-Point Standard

Single precision

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E’=E+127
5. Represent E’ into binary number.
6. Put S,E and M values in single precision format.

Single precision

85.125

85 = 1010101

0.125 = 001

85.125 = 1010101.001

=1.010101001 x 2^6

sign = 0

1. Single precision:

biased exponent 127+6=133

133 = 10000101

Normalised mantisa = 010101001

we will add 0's to complete the 23 bits

The IEEE 754 Single precision is:

= 0 10000101 01010100100000000000000

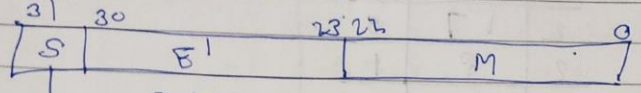
Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

IEEE Standard For Floating point Numbers

IEEE - Institute of Electrical and Electronics Engineers.

IEEE 754 standards



sign no.
0 = +ve
1 = -ve

8 bit signed exponent in excess-127 representation

23 bit mantissa

Sigle precision format

10.125 Represent in single precision formats.

convert given number into Binary

2	10	
2	5	0
2	2	1
	1	0

Integer part

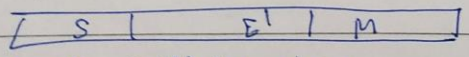
0.125 x 2 =	0.25	0
0.25 x 2 =	0.5	0
0.5 x 2 =	1.0	1
	0	

fraction part.

$(10.125)_{10} = 1010.001$

Normalize the number.

$1010.001 = 1.010001 \times 2^3$



S = 0 +ve number

E = 3 M = 010001

Bias for single precision format is = 127

$E' = E + 127$

$E' = 3 + 127$

$E' = 130_{10}$

2020/7/8 21

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

③ Represent E' into Binary E' = 137

2	137	
2	68	1
2	34	0
2	17	0
2	8	1
2	4	0
	2	0
	1	0

$E' = 10001001_2$

S	E'	M
0	1000 1001	01000 ---

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

PAGE NO:
DATE:

(2) Represent 0.0625_{10} in single precision format?

Ans. (1) convert given decimal number in binary.

integer part = 0

fraction part

$0.0625 \times 2 = 0.125$	0
$0.125 \times 2 = 0.250$	0
$0.250 \times 2 = 0.5$	0
$0.5 \times 2 = 1.0$	1
0	

$0.0625_{10} = 0.0001_2$

(2) Normalize the number

$0.0001 = 1.0 \times 2^{-4}$

(3) Single precision representation

$S = 0 \quad E = -4 \quad m = 0001$

$E' = E + 127$

$= -4 + 127$

$= 123_{10}$

(4) $E' = 123_{10}$ convert into binary

2	123	
2	61	1
2	30	1
2	15	0
2	7	1
2	3	1
	1	1

$E' = 1111011_2$

∴ number in single precision format

S	E'	M
0	01111011	00010 - - - - -

2020/7/

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E=127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

PAGE NO :
DATE :

(2) ~~15.2~~ Represent 15.2_{10} in single precision format.

Ans. ① convert given number in single precision format

integer part

2	15	
2	7	1
2	3	1
	1	1

Fraction part

$0.2 \times 2 = 0.4$	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

$15.2_{10} = 1111.00110_2$

② Normalize the number.

$1111.00110 = 1.111001_{10} \times 2^3$

③ Represent in single precision format:

$S = 0 \quad E = 3 \quad m = 111001$

$E' = E + 127$

$= 3 + 127 = 130_{10}$

④ convert E' into (130_{10}) into binary

2	130	
2	65	0
2	32	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	0

$130_{10} = (10000010)_2$

S	E'	M
0	10000010	11100110

2020/7/8

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

PAGE NO :
DATE :

④ Represent 28.5_{10} into single precision format.

Ans. ① convert given number into binary.

Integer part

2	28	
2	14	0
2	7	0
2	3	1
	1	1

Fraction part

$0.5 \times 2 = 1.0$

$0 \times 2 = 0$

$28.5 = 11100.10$

② Normalization $\rightarrow 1.110010 \times 2^4$

② Single precision format

S = 1 because number is -ve E = 4

M = 110010

$E' = E + 127$

$= 4 + 127$

$E' = 131_{10}$

④ E' convert into binary

2	131	
2	65	1
2	32	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	0

$E' = 10000011_2$

⑤

S	E'	M
1	10000011	110010----

2020/7

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

(a) 100.125

Convert it into binary

$(100)_{10} = (1100100)_2$

Fraction part

$0.125 \times 2 = 0.250$

$0.25 \times 2 = 0.50$

$0.50 \times 2 = 1.00$

$\therefore \quad (100.125)_{10} = (1100100.001)_2$

To convert above no in single precession format we will conve

$N \times 2^{E-127}$

$1100100.001 = 1.100\,100001 \times 2^6$

$= 1.100100001 \times 2^{133-127}$

0 sign | 1000101 E | 1001 100000 100000 significant (B)

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

(b) 42.625

$(42)_{10} = (101010)_2$

$0.625 \times 2 = 1.350$

$0.350 \times 2 = 0.70$

$0.70 \times 2 = 1.40$

$0.40 \times 2 = 0.80$

$0.80 \times 2 = 1.60$

$0.60 \times 2 = 1.20$

$0.20 \times 2 = 0.40$

$\therefore \quad (0.625)_{10} = 101011001100110$

$42.625 = 101010.\ 101\ 0110\ 0110\ 0110$

Convert it into $1.N \times 2^{E-127}$

$1.01010101100110 \ldots \times 2^5$

$= 1.01010101\ 0110\ 0110 \times 2^{132-127}$

0 10000100 010 10 101 011 0110 0110 011

sign bit Exponent mantissa

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

(c) 17.25

$(17)_{10} = (10001)_2$

$0.125 \times 2 = 0.50$

$0.50 \times 2 = 1.00$

$\therefore (17.125)_{10} (10001.01)_2$

Convert it into the form $(1.N) \times 2^{E-127}$

$10001.01 = 1.000101 \times 2^4$

$= 1.000101 \times 2^{131-127}$

0 10000011 0001 0100 0000 0000 0000 0000

ign Exponent significand (M)

Steps for solving single precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias single precision format E'=E+127
5. Represent E' into binary number.
6. Put S,E and M values in single precision format.

Represent $(178.1875)_{10}$ in single and double precision floating point format.

Step I : Convert decimal number in binary format integer part first

$$
\begin{array}{r|l}
16 & 178 \quad 11 \rightarrow B \\
\hline
 & 16 \\
\hline
 & 18 \\
 & 16 \\
\hline
 & 2
\end{array}
$$

$(178)_{10} = (B2)_{16} = (\underbrace{1\ 0\ 1\ 1}_{B}\ \ \underbrace{0\ 0\ 1\ 0}_{2})_2$

Fractional Part :

$0.1875 \times 2 = 0.3750 \quad 0$

$0.3750 \times 2 = 0.750 \quad 0$

$0.750 \times 2 = 1.5 \quad 1$

$0.5 \times 2 = 1.0 \quad 1$

$0 \quad 0$

$= 0.0011$

$\therefore$ Binary number $= 1011\ 0010 + 0.0011 = 1011\ 0010.\ 0011$

is - 1022<=E<=1023.

Example 1 : Represent 1259. 125_{10} in single precision and double precision formats.

Solution : Step 1 : Convert decimal number in binary format.

```
   78                4
16)1259           16)78
  112               64
  ----              --
  0139              14 = E
  128
  ---
  011 = M
```

$$= 4\,EBH = \frac{100}{4} = \frac{1110}{E}\ \frac{1011}{B}$$

$0.125 \times 2 = 0.25 \quad 0$

$0.25 \times 2 = 0.5 \quad 0$

$0.5 \times 2 = 1.0 \quad 1$

0

$= 0.001$

$\therefore$ Binary number $= 10011101011 + 0.001$

$= 10011101011.001$

Step 2 : Normalize the number

$10011101011.001 = 1.0011101011001 \times 2^{10}$

Now we will see the representation of the numbers in single precision and double precision formats.

Step 3 : Single precision representation for a given number S = 0. E = 10 and M = 00111 01011 001

Bias for single precision format is = 127

$E' = E + 127 = 10 + 127 = 137_{10}$

$= 10001001_2$

$\therefore$ Number in double precision format is given as

0	1000 1001	00111010110001 0
Sign	Exponent	Mantissa (23 bits)

Step 4 : Double precision representation for a given number

S = 0, E = 10, and M = 0011101011001

Bias for double precision format is = 1023

$E' = E + 1023 = 10 + 1023 = 1033_{10}$

$= 10000001001_2$

$\therefore$ Number in double precision format is given as

0	1000 1001	0011101011001 0
Sign	Exponent	Mantissa (52 bits)

Example 2 : Represent–307.1875$_{10}$ in single precision and double precision formats.

Solution : **Step 1 :** Convert decimal number in binary format integer part.

Integer part :

$$16 \overline{)307} \quad 19 \quad 16 \overline{)19} \quad 1$$

$$\begin{array}{r} 16 \\ \hline 147 \\ 144 \\ \hline 3 \end{array} \qquad \begin{array}{r} 16 \\ \hline 3 \end{array}$$

$3 = 3$

$= 133\ H = \frac{1}{1} = \frac{0011}{3} \quad \frac{0011}{3}$

Fractional part :

$0.1875 \times 2 = 0.3750 \quad 0$

$0.3750 \times 2 = 0.750 \quad 0$

$0.750 \times 2 = 1.5 \quad 1$

$0.5 \times 2 = 1.0 \quad 1$

0

$= 0.0011$

Binary number $= -100110011 + 0.0011$

$= -100110011 . 0011$

Step 2 : Normalize the number -100110011.0011

$= -1.001100110011 \times 2^8$

Now we will see the representation of the numbers in single precision and double precision formats.

Step 3 : Single precision representation for a given number

$S = 1$, $E = 8$, and $M = 0011\ 0011\ 0011$

Bias for single precision format is $= 127$

$E' = E + 127 = 8 + 127 = 135_{10}$

$= 10000111_2$

Number in single precision format is given as

$\underbrace{1}$ $\underbrace{1000\ 0111}$ $\underbrace{00110011001100 \ldots\ldots 0}$

Sign Exponent Mantissa (23 bits)

Step 2 : Normalize the number

$$1011\ 0010.0011 = 1.01100100011 \times 2^7$$

Step 3 : Representation in signal precision for given number

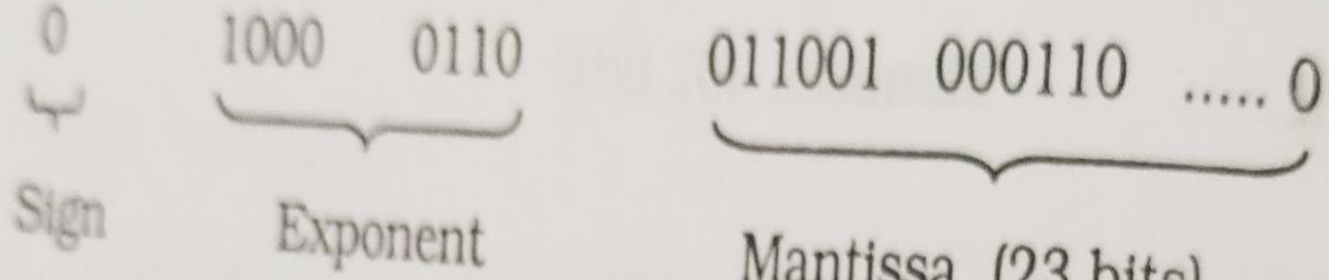
$S = 0,\ E = 7$ and $M = 011\ 001\ 000\ 11$

Bias for single precision format is $= 127$

$$E' = E + 127 = 7 + 127 = 134_{10}$$

$$= 10000110_2$$

∴ Number in single precision format is given as



Step 4 : Representation in double

IEEE Floating Point Standard (FPS)

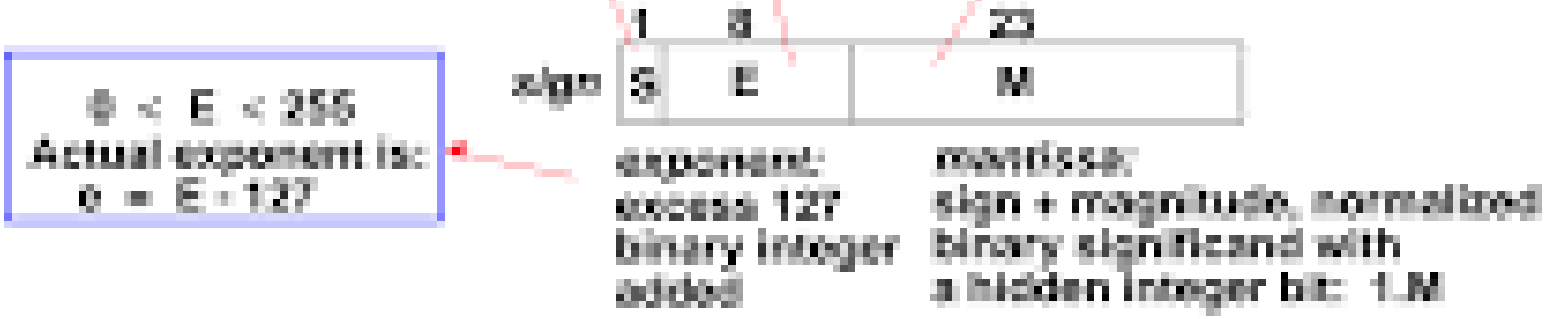
- 2 standards

1. Single precision
   - 32-bits
     - 23-bit mantissa
     - 8-bit exponent
     - 1-bit sign

2. Double precision
   - 64-bits
     - 52-bit mantissa
     - 11-bit exponent
     - 1-bit sign

Representation of Floating Point Numbers in Single Precision *IEEE 754 Standard*

$$\text{Value} = N = (-1)^S \times 2^{E-127} \times (1.M)$$



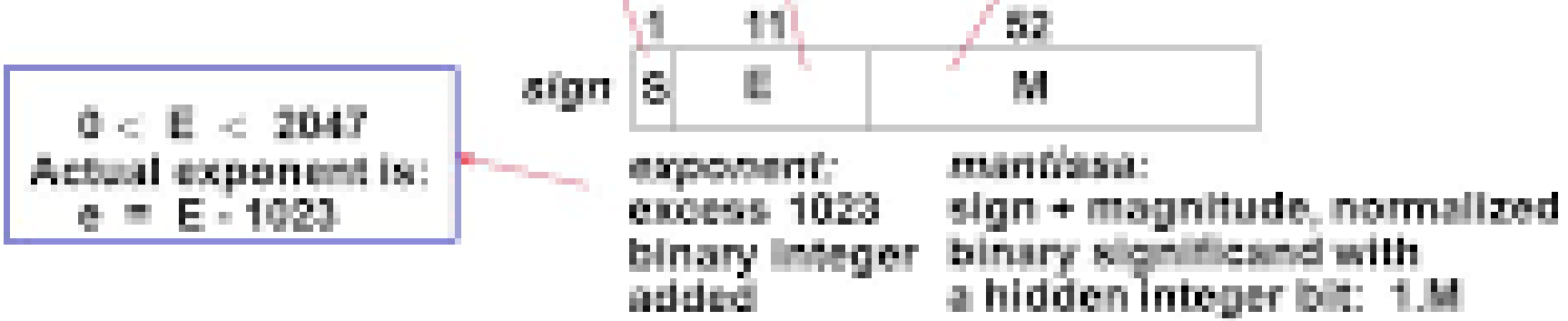
Example: 0 = 0 00000000 0 . . . 0 -1.5 = 1 01111111 10 . . . 0

Magnitude of numbers that can be represented is in the range: $2^{-126}(1.0)$ to $2^{127}(2 - 2^{-23})$

Which is approximately: 1.8×10^{-38} to 3.40×10^{38}

Representation of Floating Point Numbers in Double Precision *IEEE 754 Standard*

Value = N = $(-1)^S$ X 2^{E-1023} X (1.M)

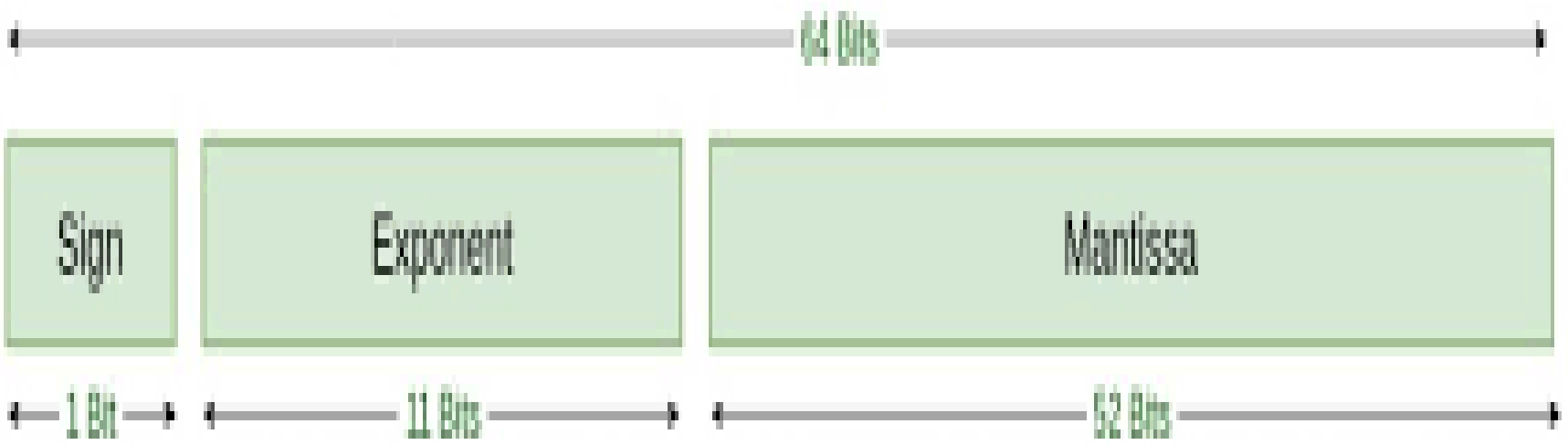


Example: 0 = 0 00000000000 0 . . . 0 -1.5 = 1 01111111111 10 . . . 0

Magnitude of numbers that can be represented is in the range: 2^{-1022} (1.0) to 2^{1023} ($2 - 2^{-52}$)

Which is approximately: 2.23×10^{-308} to 1.8×10^{308}

Double Precision-IEEE754 format



Double Precision

Steps for solving double precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias double precision format E’=E+1023
5. Represent E’ into binary number.
6. Put S,E and M values in double precision format.

① Represent the following number in double precision format.

100.125_{10}

Step1 convert given number into binary

2	100	
2	50	0
2	25	0
2	12	1
2	6	0
2	3	0
	1	1

integer Part.

$0.125 \times 2 = 0.250$ 0

$0.250 \times 2 = 0.500$ 0

$0.500 \times 2 = 1.00$ 1

fraction Part.

Step2 Normalized the number.

$100.25_{10} = 1100100.001$

$= 1.100100001 \times 2^6$

Step3 S = 0 E = 6 M = 10010001

Step4. $E' = E + 1023$

$= 6 + 1023 = 1029_{10}$

Step5. convert E' into binary

2	1029	
2	514	1
2	257	0
2	128	1
2	64	0
2	32	0
2	16	0
2	8	0
2	4	0

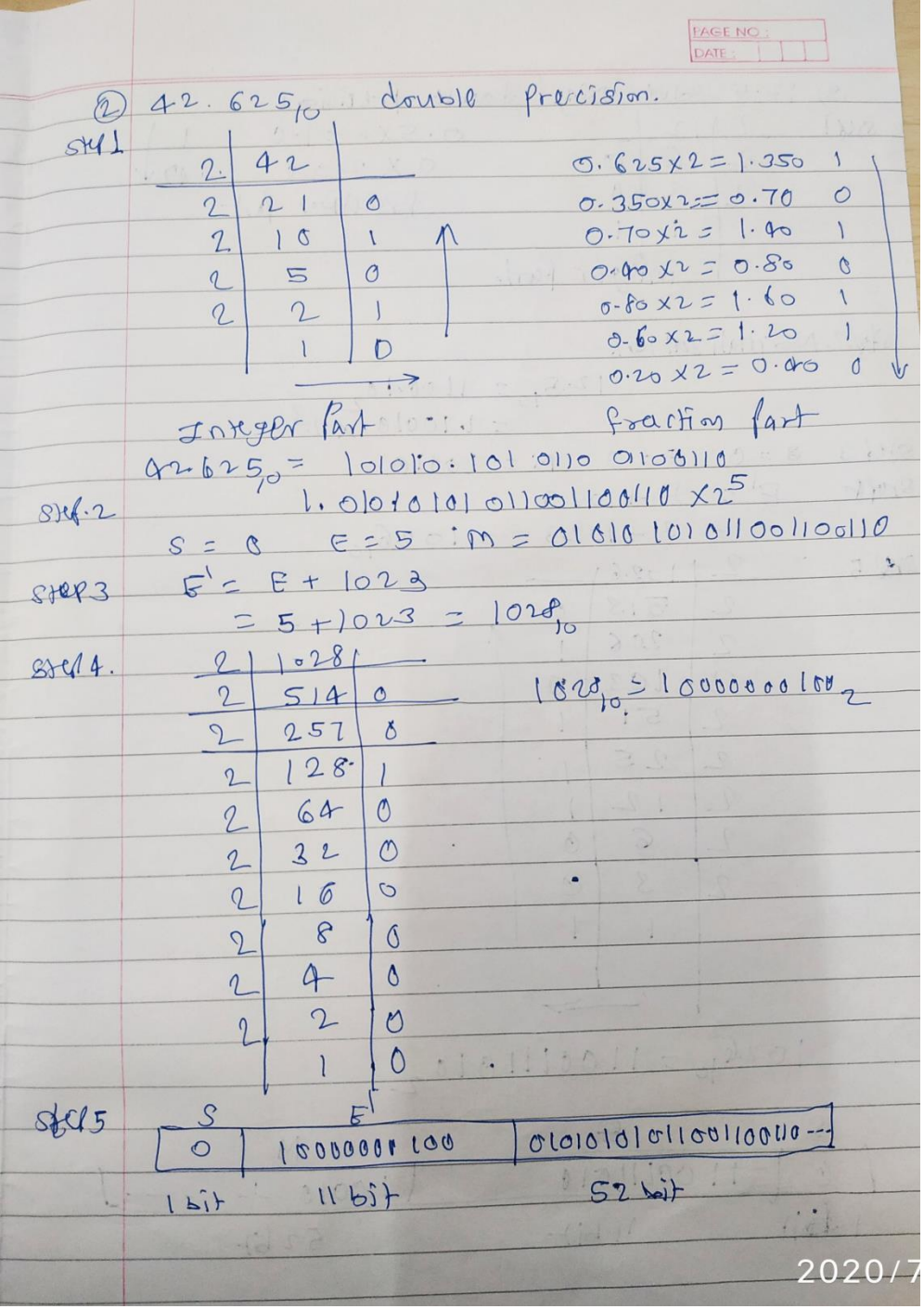
$1029_{10} = 10000000101_2$

S	E	M
0	10000000101	10010001 ----
1 bit	11 bits	52 bits

2020/7/9 15:19

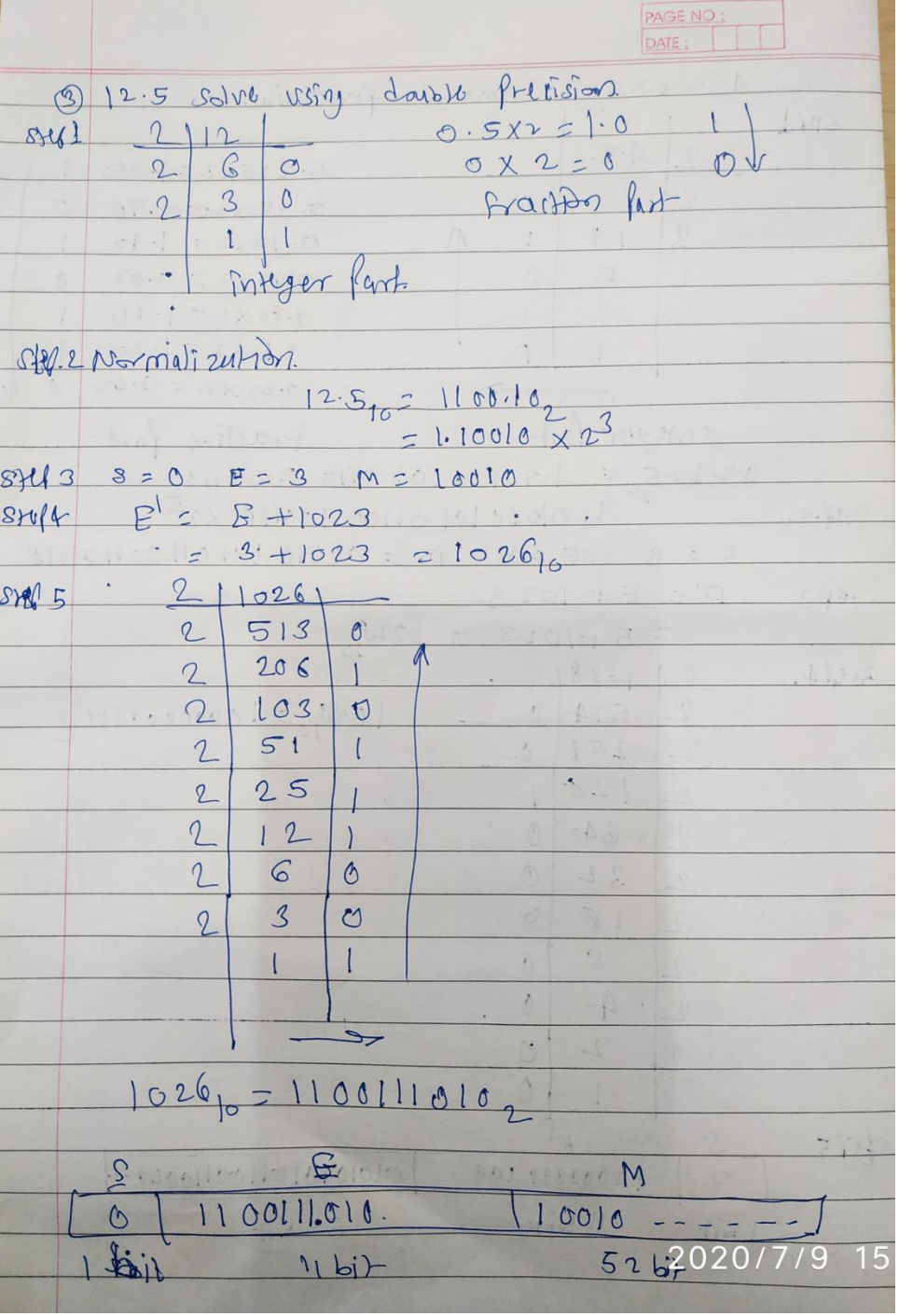
Steps for solving double precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias double precision format E'=E+1023
5. Represent E' into binary number.
6. Put S,E and M values in double precision format.



Steps for solving double precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias double precision format E'=E+1023
5. Represent E' into binary number.
6. Put S,E and M values in double precision format.



Steps for solving double precision problems

1. Covert given decimal number into binary(integer and fraction part).
2. Normalize the binary number and find values of Sign, Exponent, Mantissa.
3. If given number is +ve then S=0 if not S=1 i.e.-Ve
4. Bias double precision format E'=E+1023
5. Represent E' into binary number.
6. Put S,E and M values in double precision format.

is - 1022<=E<=1023.

Example 1 : Represent 1259. 125_{10} in single precision and double precision formats.

Solution : Step 1 : Convert decimal number in binary format.

```
   78                4
16)1259           16)78
  112               64
  ----              --
  0139              14 = E
  128
  ---
  011 = M
```

$$= 4\,EBH = \frac{100}{4} = \frac{1110}{E}\ \frac{1011}{B}$$

$0.125 \times 2 = 0.25 \quad 0$

$0.25 \times 2 = 0.5 \quad 0$

$0.5 \times 2 = 1.0 \quad 1$

0

$= 0.001$

$\therefore$ Binary number $= 10011101011 + 0.001$

$= 10011101011.001$

Step 2 : Normalize the number

$10011101011.001 = 1.0011101011001 \times 2^{10}$

Now we will see the representation of the numbers in single precision and double precision formats.

Step 3 : Single precision representation for a given number S = 0. E = 10 and M = 00111 01011 001

Bias for single precision format is = 127

$E' = E + 127 = 10 + 127 = 137_{10}$

$= 10001001_2$

$\therefore$ Number in double precision format is given as

0	1000 1001	00111010110001 0
Sign	Exponent	Mantissa (23 bits)

Step 4 : Double precision representation for a given number

S = 0, E = 10, and M = 0011101011001

Bias for double precision format is = 1023

$E' = E + 1023 = 10 + 1023 = 1033_{10}$

$= 10000001001_2$

$\therefore$ Number in double precision format is given as

0	1000 1001	0011101011001 0
Sign	Exponent	Mantissa (52 bits)