FCSD Notes _Module 1a

NUMBERS, ARITHMETIC OPERATIONS AND CHARACTERS NUMBER REPRESENTATION

Numbers can be represented in 3 formats: Sign and magnitude 1's complement 2's complement In all three formats, MSB=0 for +ve numbers & MSB=1 for -ve numbers. In sign-and-magnitude system, negative value is obtained by changing the MSB from 0 to 1 of the corresponding positive value. For ex, +5 is represented by 0101 & -5 is represented by 1101.

• In 1's complement system, negative values are obtained by complementing each bit of the corresponding positive number. For ex, -5 is obtained by complementing each bit in 0101 to yield 1010. (In other words, the operation of forming the 1's complement of a given number is equivalent to subtracting that number from 2n -1).

• In 2's complement system, forming the 2's complement of a number is done by subtracting that number from 2n . For ex, -5 is obtained by complementing each bit in 0101 & then adding 1 to yield 1011. (In other words, the 2's complement of a number is obtained by adding 1 to the 1's complement of that number).

• 2's complement system yields the most efficient way to carry out addition/subtraction operations.

B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

Figure 1.3 Binary, signed-integer representations.

ADDITION OF POSITIVE NUMBERS Consider adding two 1-bit numbers. The sum of 1 & 1 requires the 2-bit vector 10 to represent the value 2. We say that sum is 0 and the carry-out is 1.

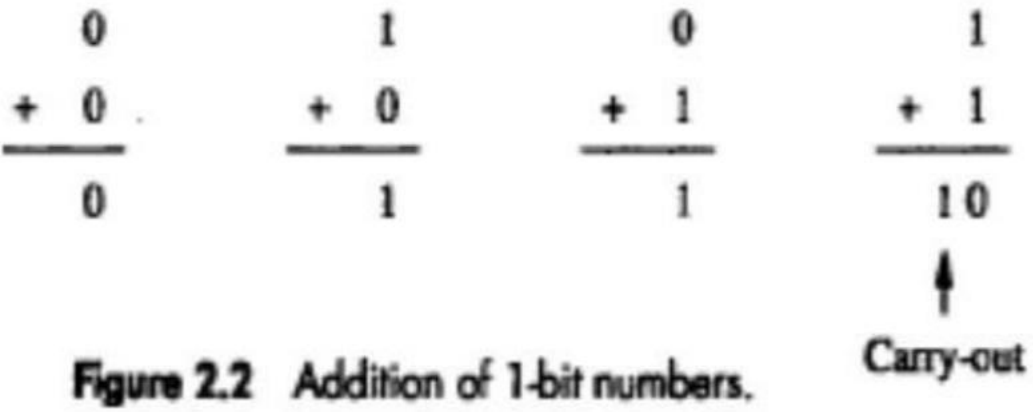


Figure 2.2 Addition of 1-bit numbers.

ADDITION & SUBTRACTION OF SIGNED NUMBERS

Following are the two rules for addition and subtraction of n-bit signed numbers using the 2's complement representation system (Figure 1.6). Rule 1: To Add two numbers, add their n-bits and ignore the carry-out signal from the MSB position. Result will be algebraically

correct, if it lies in the range -2 n-1 to +2n-1 -1. Rule 2: To Subtract two numbers X and Y (that is to perform X-Y), take the 2's complement of Y and then add it to X as in rule 1. Result will be algebraically correct, if it lies in the range (2n-1) to +(2n-1 -1). When the result of an arithmetic operation is outside the representable-range, an arithmetic overflow is said to occur. To represent a signed in 2's complement form using a larger number of bits, repeat the sign bit as many times as needed to the left. This operation is called sign extension. In 1's complement representation, the result obtained after an addition operation is not always correct. The carry-out(cn) cannot be ignored. If cn=0, the result obtained is correct. If cn=1, then a 1 must be added to the result to make it correct.

OVERFLOW IN INTEGER ARITHMETIC

When result of an arithmetic operation is outside the representable-range, an arithmetic overflow is said to occur. For example: If we add two numbers +7 and +4, then the output sum S is 1011(0111+0100), which is the code for -5, an incorrect result. An overflow occurs in following 2 cases

i) Overflow can occur only when adding two numbers that have the same sign.

ii) The carry-out signal from the sign-bit position is not a sufficient indicator of overflow when adding signed numbers.

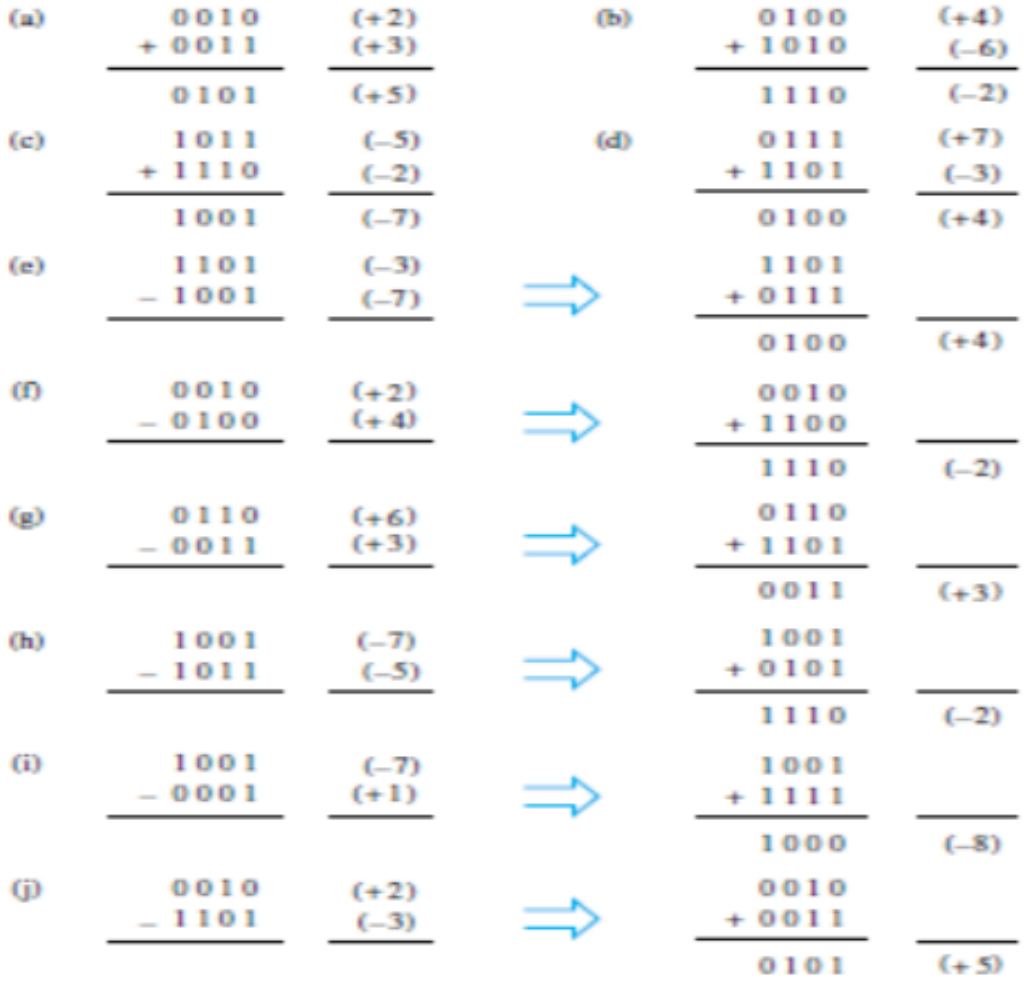


Figure: 2’s complement of Add and subtract operations

Multiplication of Positive numbers

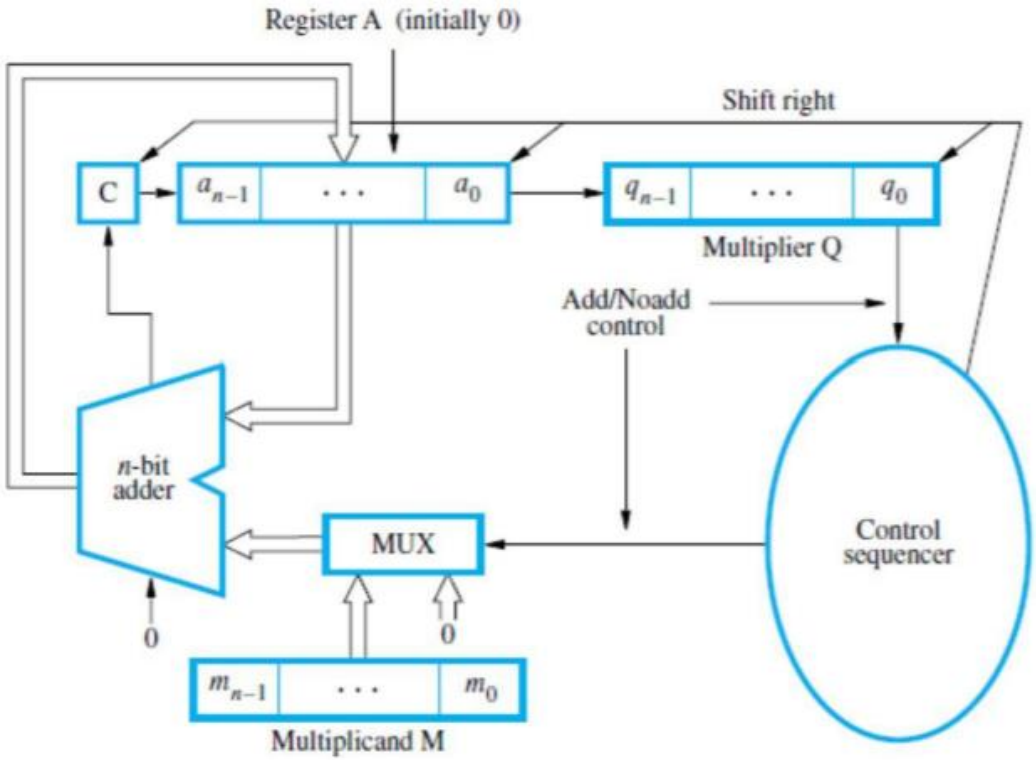
			1	1	0	1		(13) Multiplicand M
		×	1	0	1	1		(11) Multiplier Q
			1	1	0	1		
		1	1	0	1			
	0	0	0	0				
1	1	0	1					
1	0	0	0	1	1	1	1	(143) Product P

(a) Manual multiplication algorithm

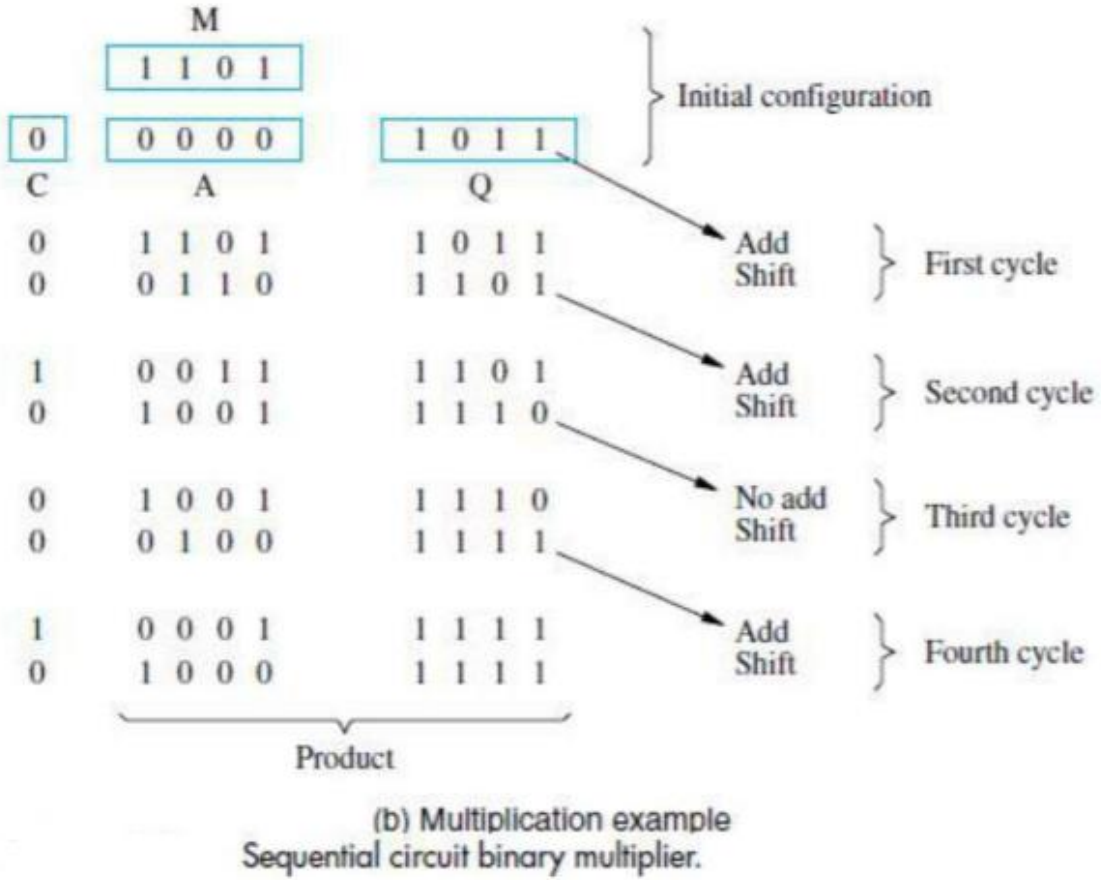
SEQUENTIAL CIRCUIT BINARY MULTIPLIER

• Registers A and Q combined hold PPi(partial product) while the multiplier bit qi generates the signal Add/Noadd.

• The carry-out from the adder is stored in flip-flop C (Figure 9.7). Procedure for multiplication: Multiplier is loaded into register Q, Multiplicand is loaded into register M and C & A are cleared to 0. If q0=1, add M to A and store sum in A. Then C, A and Q are shifted right one bit position. If q0=0, no addition performed and C, A & Q are shifted right one bit-position. After n cycles, the high-order half of the product is held in register A and the low-order half is held in register Q.



(a) Register configuration



(b) Multiplication example

Sequential circuit binary multiplier.

SIGNED OPERAND MULTIPLICATION

BOOTH ALGORITHM This algorithm generates a 2n-bit product treats both positive & negative 2's-complement n-bit operands uniformly(Figure 9.9-9.12).

Attractive feature: This algorithm achieves some efficiency in the number of addition required when the multiplier has a few large blocks of 1s. This algorithm suggests that we can reduce the number of operations required for multiplication by representing multiplier as a difference between 2 numbers. For e.g. multiplier(Q) 14(001110) can be represented as 010000 (16) -000010 (2) 001110 (14) Therefore, product P=M*Q can be computed by adding 24 times the M to the 2's complement of 21 times the M.

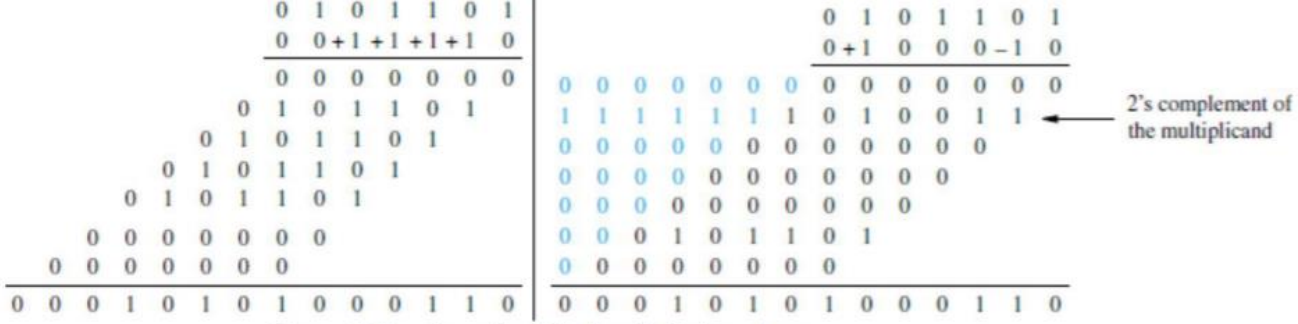


Figure 9.9 Normal and Booth multiplication schemes.

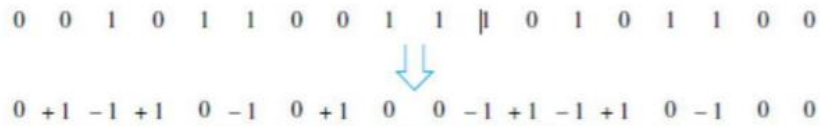


Figure 9.10 Booth recoding of a multiplier.

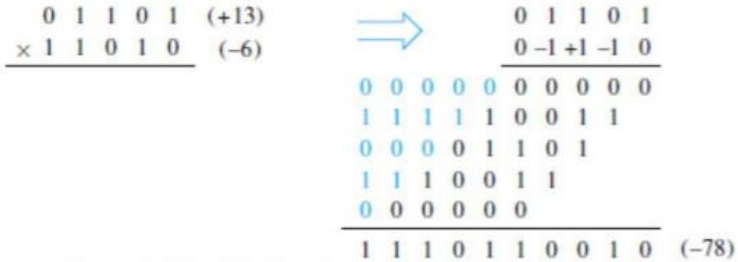
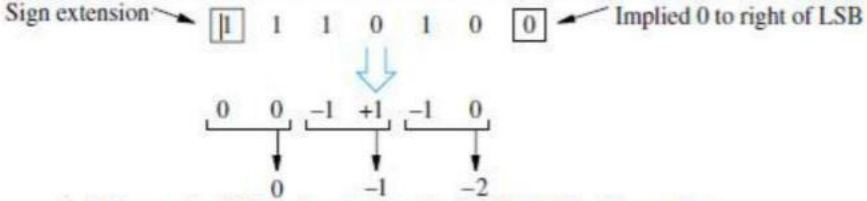


Figure 9.11 Booth multiplication with a negative multiplier.

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit $i-1$	
0	0	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

Figure 9.12 Booth multiplier recoding table.

FAST MULTIPLICATION BIT-PAIR RECODING OF MULTIPLIERS This method derived from the booth algorithm reduces the number of summands by a factor of 2 Group the Booth-recoded multiplier bits in pairs. (Figure 9.14 & 9.15). The pair (+1 -1) is equivalent to the pair (0 +1).



(a) Example of bit-pair recoding derived from Booth recoding

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

(b) Table of multiplicand selection decisions

Figure 9.14 Multiplier bit-pair recoding.

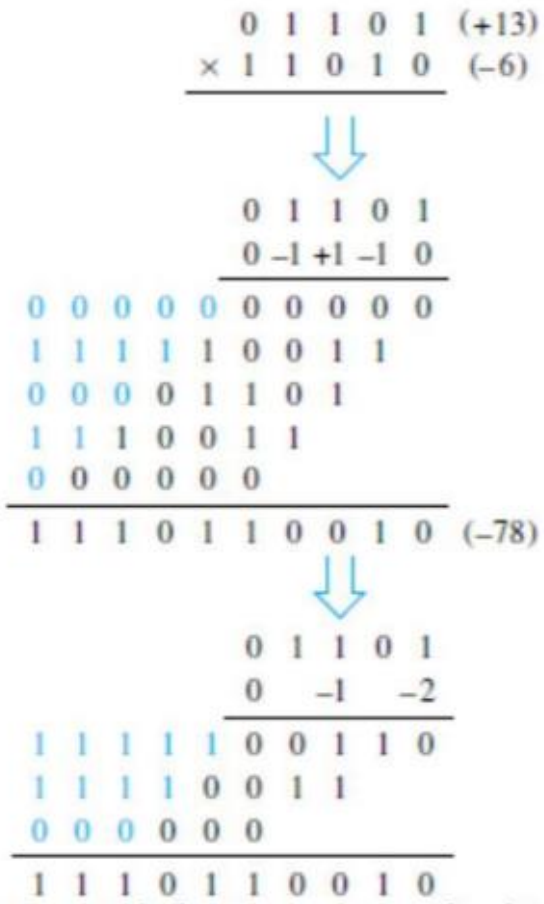


Figure 9.15 Multiplication requiring only $n/2$ summands.

INTEGER DIVISION

• An n-bit positive-divisor is loaded into register M. An n-bit positive-dividend is loaded into register Q at the start of the operation. Register A is set to 0 (Figure 9.21). • After division operation, the n-bit quotient is in register Q, and the remainder is in register A

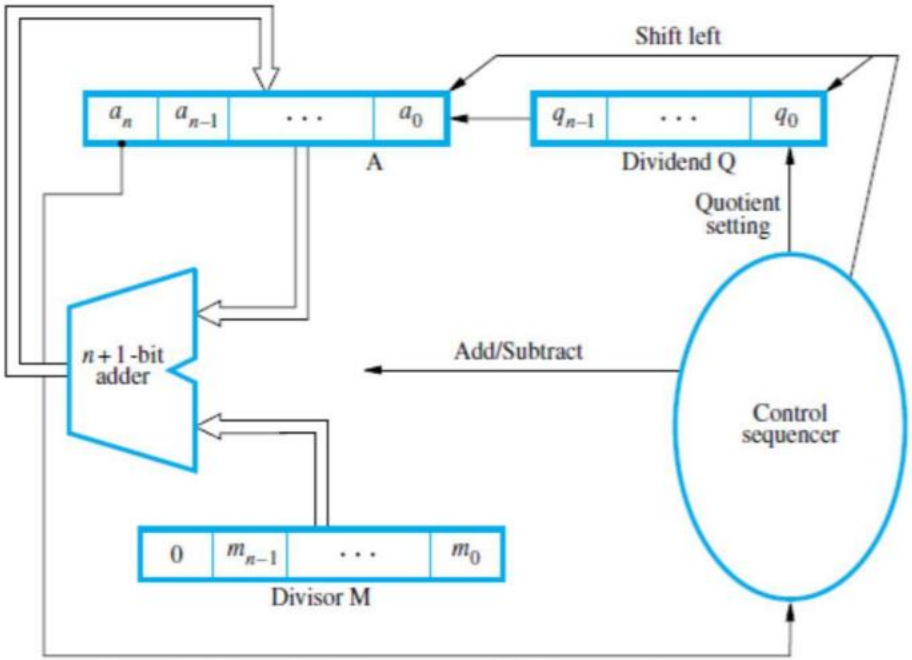


Figure 9.23 Circuit arrangement for binary division.

```
      21                  10101
13 ) 274          1101 ) 100010010
     26                    1101
     --                    ----
     14                   10000
     13                    1101
     --                    ----
      1                    1110
                           1101
                           ----
                              1
```

Figure 9.22 Longhand division examples.

NON-RESTORING DIVISION • Procedure: Step 1: Do the following n times i) If the sign of A is 0, shift A and Q left one bit position and subtract M from A; otherwise, shift A and Q left and add M to A (Figure 9.23). ii) Now, if the sign of A is 0, set q0 to 1; otherwise set q0 to 0. Step 2: If the sign of A is 1, add M to A (restore).

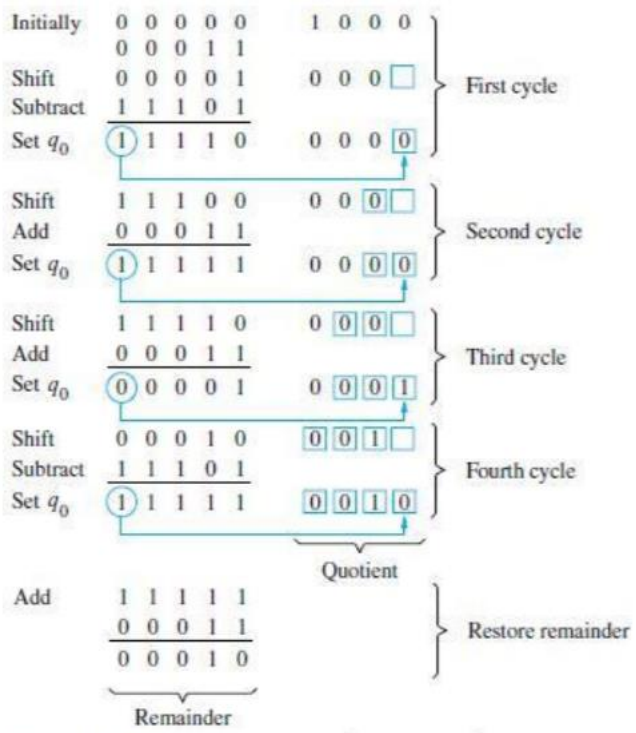


Figure 9.25 A non-restoring division example.

RESTORING DIVISION Procedure: Do the following n times Shift A and Q left one binary position (Figure 9.22). Subtract M from A, and place the answer back in A If the sign of A is 1, set q0 to 0 and add M back to A(restore A). If the sign of A is 0, set q0 to 1 and no restoring done.

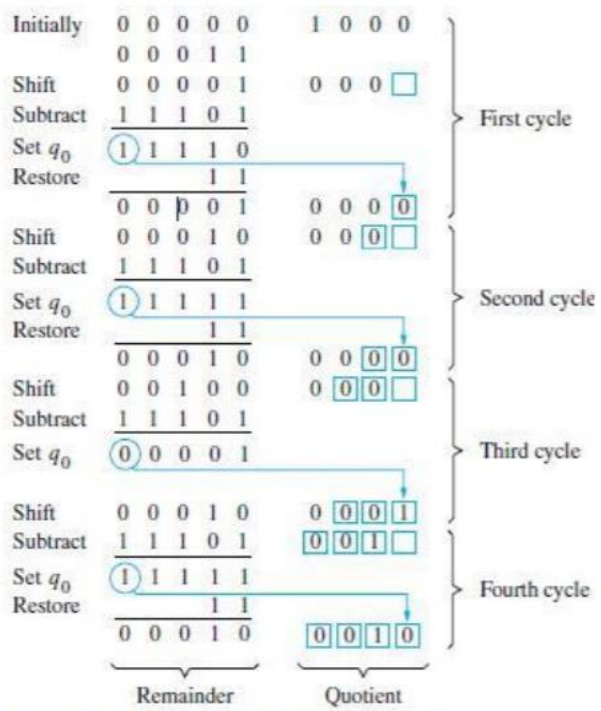


Figure 9.24 A restoring division example.

FLOATING-POINT NUMBERS & OPERATIONS IEEE STANDARD FOR FLOATING POINT NUMBERS • Single precision representation occupies a single 32-bit word. The scale factor has a range of 2-126 to 2+127 (which is approximately equal to 10+38). The 32 bit word is divided into 3 fields: sign(1 bit), exponent(8 bits) and mantissa(23 bits). Signed exponent=E. Unsigned exponent E'=E+127. Thus, E' is in the range 0<E'

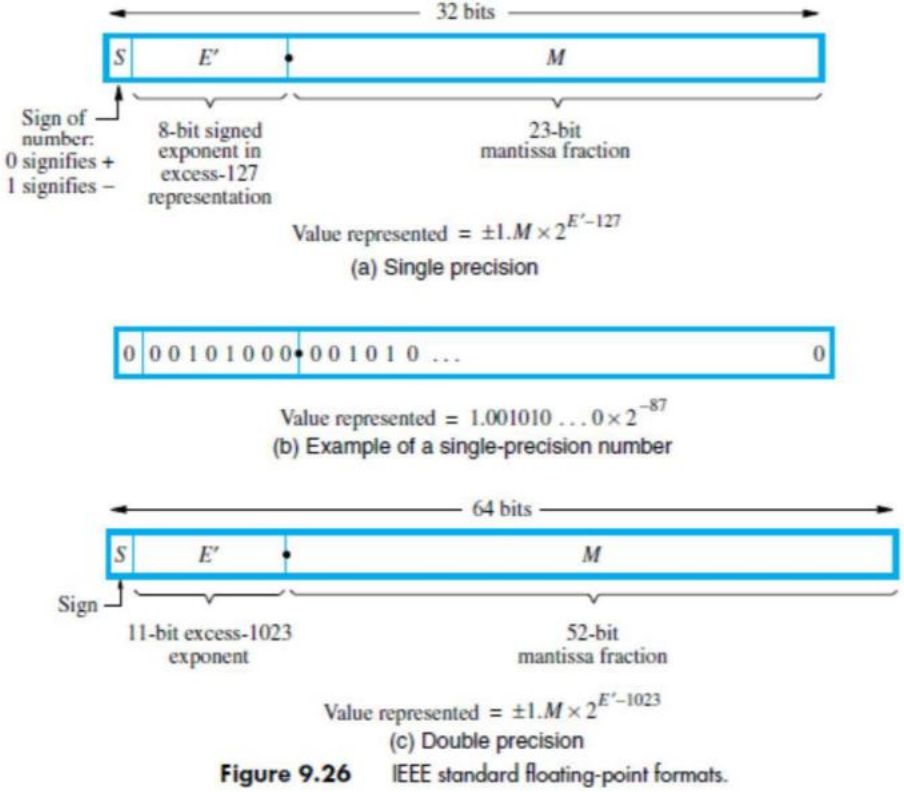


Figure 9.26 IEEE standard floating-point formats.

NORMALIZATION When the decimal point is placed to the right of the first(non zero) significant digit, the number is said to be normalized. If a number is not normalized, it can always be put in normalized form by shifting the fraction and adjusting the exponent. As computations proceed, a number that does not fall in the representable range of normal numbers might be generated. In single precision, it requires an exponent less than -126 (underflow) or greater than +127 (overflow). Both are exceptions that need to be considered.

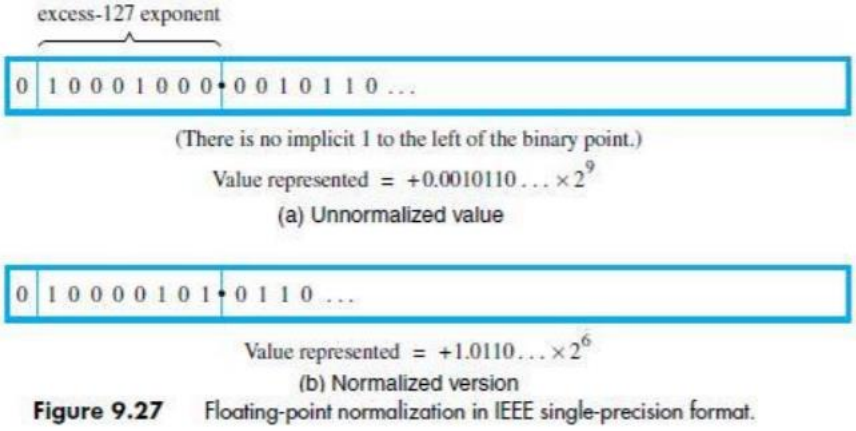


Figure 9.27 Floating-point normalization in IEEE single-precision format.

SPECIAL VALUES The end values 0 and 255 of the excess-127 exponent E' are used to represent special values. When E'=0 and the mantissa fraction m is zero, the value exact 0 is represented. When E'=255 and M=0, the value ∞ is represented, where ∞ is the result of dividing a normal number by zero. when E'=0 and M!=-, denormal numbers are represented. Their value is X2-126 When E'=255 and M!=0, the value represented is called not a number(NaN). A NaN is the result of performing an invalied operation such as 0/0 or . ARITHMETIC OPERATIONS ON FLOATING-POINT NUMBERS Multiply Rule Add the exponents & subtract 127. Multiply the mantissas & determine sign of the result. Normalize the resulting value if necessary. Divide Rule Subtract the exponents & add 127. Divide the mantissas & determine sign of the result. Normalize the resulting value if necessary. Add/Subtract Rule Choose the number with the smaller exponent & shift its mantissa right a number of steps equal to the difference in exponents(n). Set exponent of the result equal to larger exponent. Perform addition/subtraction on the mantissas & determine sign of the result. Normalize the resulting value if necessary.

Problem 1: Represent the decimal values 5, -2, 14, -10, 26, -19, 51 and -43 as signed 7-bit numbers in the following binary formats: sign-and-magnitude 1's-complement 2's-complement Solution: The three binary representations are given as:

Decimal values	Sign-and-magnitude representation	1's-complement representation	2's-complement representation
5	0000101	0000101	0000101
−2	1000010	1111101	1111110
14	0001110	0001110	0001110
−10	1001010	1110101	1110110
26	0011010	0011010	0011010
−19	1010011	1101100	1101101
51	0110011	0110011	0110011
−43	1101011	1010100	1010101

Problem 2: (a) Convert the following pairs of decimal numbers to 5-bit 2's-complement numbers, then add them. State whether or not overflow occurs in each case. a) 5 and 10 b) 7 and 13 c) –14 and 11 d) –5 and 7 –3 and –8 Repeat Problem 1.7 for the subtract operation, where the second number of each pair is to be subtracted from the first number. State whether or not overflow occurs in each case.
Solution: (a)

(a)	(b)	(c)
00101 + 01010	00111 + 01101	10010 + 01011
01111	10100	11101
no overflow	overflow	no overflow

(d)	(e)	(f)
11011 + 00111	11101 + 11000	10110 + 10011
00010	10101	01001
no overflow	no overflow	overflow

(b) To subtract the second number, form its 2's-complement and add it to the first number.

(a)	(b)	(c)
00101 + 10110	00111 + 10011	10010 + 10101
11011	11010	00111
no overflow	no overflow	overflow

(d)	(e)	(f)
11011 + 11001	11101 + 01000	10110 + 01101
10100	00101	00011
no overflow	no overflow	no overflow

Problem 3:

Perform following operations on the 6-bit signed numbers using 2's complement representation system. Also indicate whether overflow has occurred.

010110 +001001	101011 +100101	111111 +000111
011001 +010000	110111 +111001	010101 +101011
010110 −011111	111110 −100101	100001 −011101
111111 −000111	000111 −111000	011010 −100010

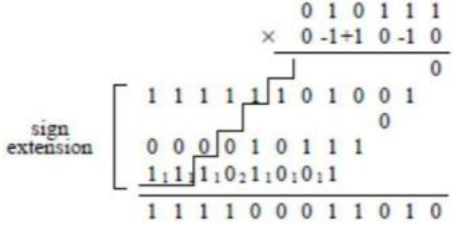
Solution:

010110 + 001001 011111	(+22) + (+9) (+31)	101011 + 100101 010000 overflow	(−21) + (−27) (−48)	111111 + 000111 000110	(−1) + (+7) (+6)
011001 + 010000 101001 overflow	(+25) + (+16) (+41)	110111 + 111001 110000	(−9) + (−7) (−16)	010101 + 101011 000000	(+21) + (−21) (0)

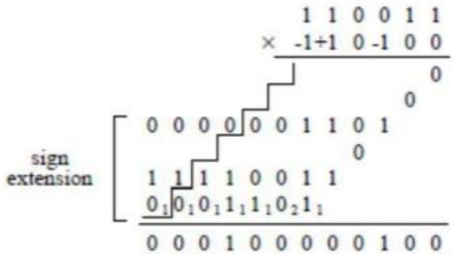
010110	(+22)	010110
− 011111	− (+31)	+ 100001
	(−9)	110111
111110	(−2)	111110
− 100101	− (−27)	+ 011011
	(+25)	011001
100001	(−31)	100001
− 011101	− (+29)	+ 100011
	(−60)	000100
		overflow
111111	(−1)	111111
− 000111	− (+7)	+ 111001
	(−8)	111000
000111	(+7)	000111
− 111000	− (−8)	+ 001000
	(+15)	001111
011010	(+26)	011010
− 100010	− (−30)	+ 011110
	(+56)	111000
		overflow

Problem 4: Perform signed multiplication of following 2's complement numbers using Booth's algorithm. (a) A=010111 and B=110110 (b) A=110011 and B=101100 (c) A=110101 and B=011011 (d) A=001111 and B=001111 (e) A=10100 and B=10101 (f) A=01110 and B=11000 Solution:

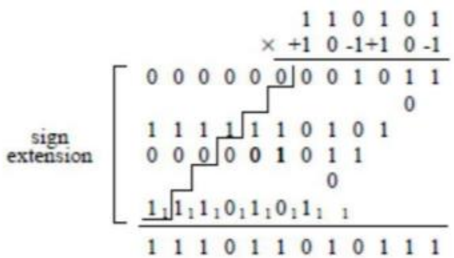
```
  010111        +23
× 110110      × -10
              -230
```

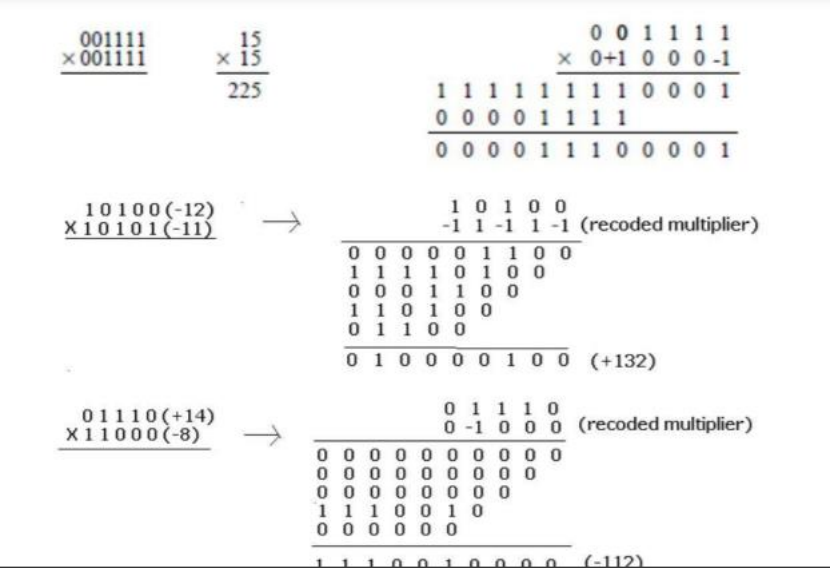



```
  110011        -13
× 101100      × -20
               260
```



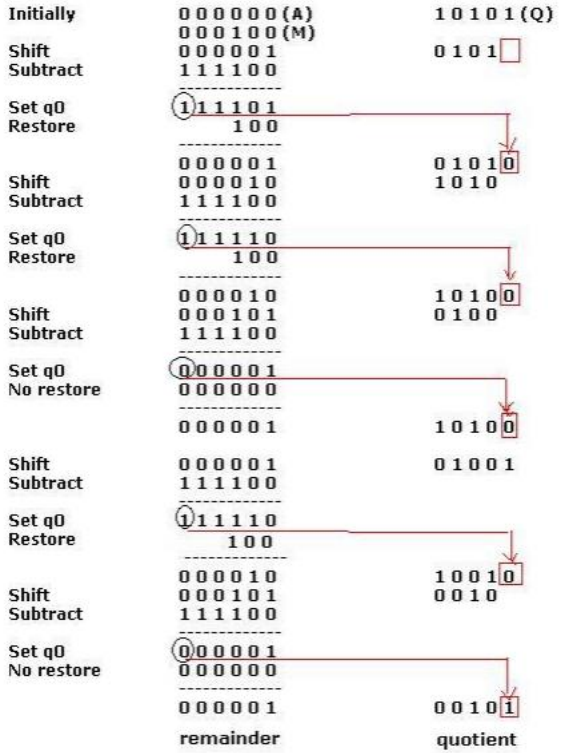

```
  110101        -11
× 011011      ×  27
              -297
```



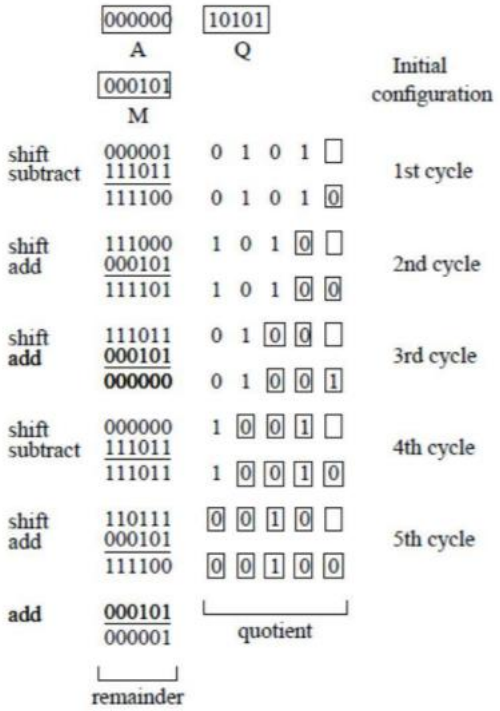


Problem 6: Given A=10101 and B=00100, perform A/B using restoring division algorithm.

Solution:



Problem 7: Given A=10101 and B=00101, perform A/B using non-restoring division algorithm. Solution:


000000
A
10101
Q
000101
M
Initial configuration
shift
subtract
000001
111011
111100
0 1 0 1
1st cycle
shift
add
111000
000101
111101
1 0 1 0
2nd cycle
shift
add
111011
000101
000000
0 1 0 0
3rd cycle
shift
subtract
000000
111011
111011
1 0 0 1
4th cycle
shift
add
110111
000101
111100
0 0 1 0
5th cycle
add
000101
000001
quotient
remainder