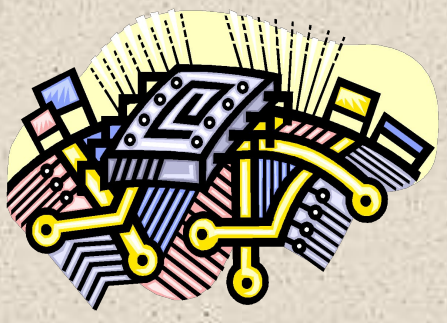
William Stallings Computer Organization and Architecture 10th Edition

Chapter 10

Computer Arithmetic

Arithmetic & Logic Unit (ALU)

- Part of the computer that actually performs arithmetic and logical operations on data
- All of the other elements of the computer system are there mainly to bring data into the ALU for it to process and then to take the results back out
- Based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations



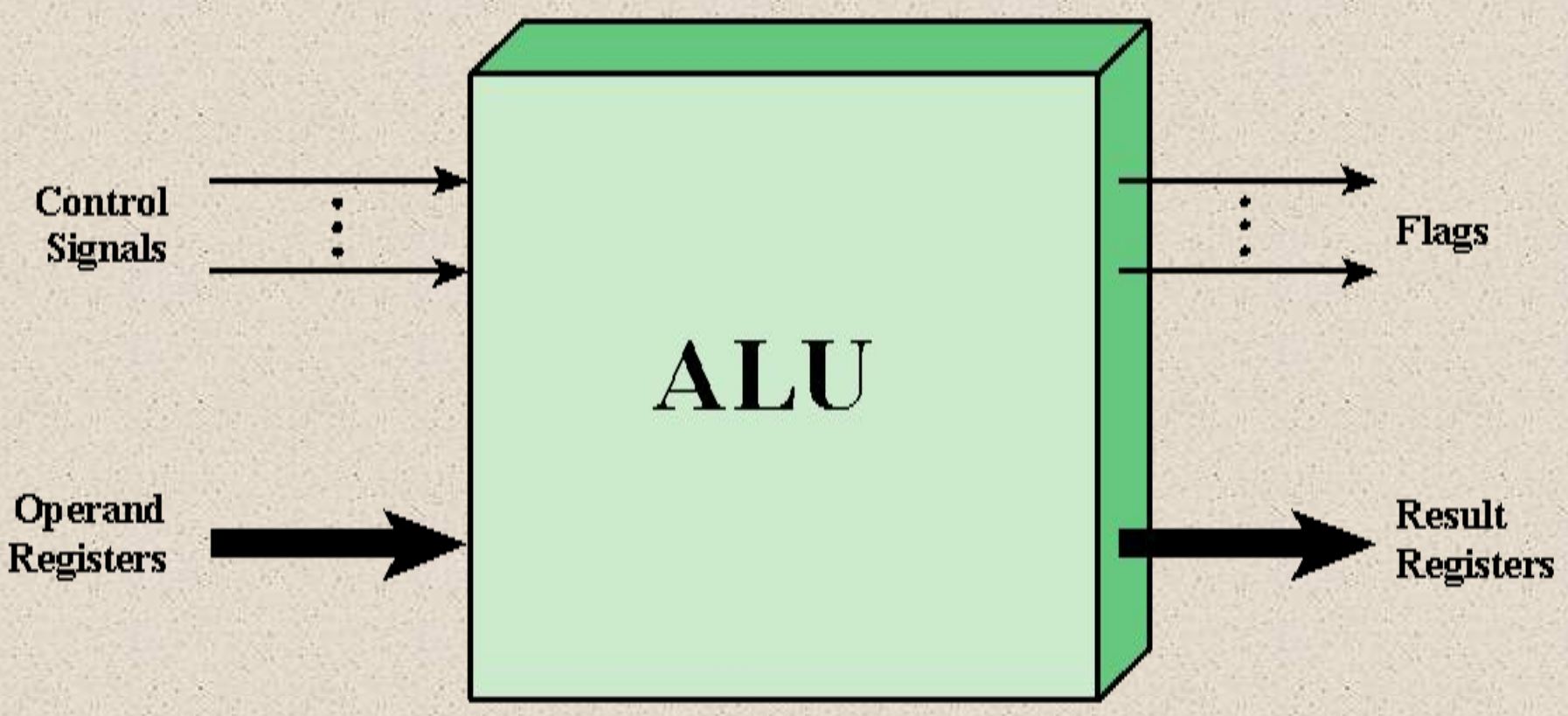


Figure 10.1 ALU Inputs and Outputs

Integer Representation

- In the binary number system arbitrary numbers can be represented with:
  - The digits zero and one
  - The minus sign (for negative numbers)
  - The period, or ***radix point*** (for numbers with a fractional component)
- For purposes of computer storage and processing we do not have the benefit of special symbols for the minus sign and radix point
- Only binary digits (0,1) may be used to represent numbers

Sign-Magnitude Representation

There are several alternative conventions used to represent negative as well as positive integers	•All of these alternatives involve treating the most significant (leftmost) bit in the word as a sign bit •If the sign bit is 0 the number is positive •If the sign bit is 1 the number is negative
Sign-magnitude representation is the simplest form that employs a sign bit	
Drawbacks:	•Addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation •There are two representations of 0
Because of these drawbacks, sign-magnitude representation is rarely used in implementing the integer portion of the ALU	

Table 10.1

Characteristics of Twos Complement Representation and Arithmetic

Range	-2_{n-1} through $2_{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract B from A, take the twos complement of B and add it to A.

Table 10.2

Alternative Representations for 4-Bit Integers

Decimal Representation	Sign-Magnitude Representation	Twos Complement Representation
+8	—	—
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
–0	1000	—
–1	1001	1111
–2	1010	1110
–3	1011	1101
–4	1100	1100
–5	1101	1011
–6	1110	1010
–7	1111	1001
–8	—	1000

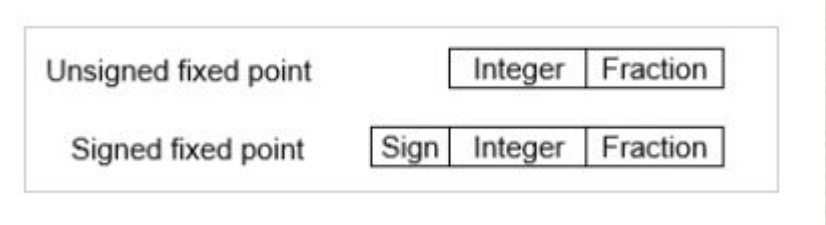
Range Extension

- Range of numbers that can be expressed is extended by increasing the bit length
- In sign-magnitude notation this is accomplished by moving the sign bit to the new leftmost position and fill in with zeros
- This procedure will not work for twos complement negative integers
  - Rule is to move the sign bit to the new leftmost position and fill in with copies of the sign bit
  - For positive numbers, fill in with zeros, and for negative numbers, fill in with ones
  - This is called *sign extension*

Fixed-Point Representation

The radix point (binary point) is fixed and assumed to be to the right of the rightmost digit

Programmer can use the same representation for binary fractions by scaling the numbers so that the binary point is implicitly positioned at some other location



Negation

- Twos complement operation
  - Take the Boolean complement of each bit of the integer (including the sign bit)
  - Treating the result as an unsigned binary integer, add 1

```
                    +18 = 00010010 (twos complement)
bitwise complement = 11101101
                     +        1
                      11101110 = -18
```

- The negative of the negative of that number is itself:

```
                    -18 =  11101110 (twos complement)
bitwise complement =  00010001
                      +        1
                        00010010 = +18
```

Negation Special Case 1

0	=	00000000	(twos complement)
Bitwise complement	=	11111111	
Add 1 to LSB		+ 1	
Result		100000000	

Overflow is ignored, so:

- 0 = 0

Negation Special Case 2

-128	=	10000000	(twos complement)
Bitwise complement	=	01111111	
Add 1 to LSB		+ 1	
Result		10000000	

So:

-(-128) = -128 X

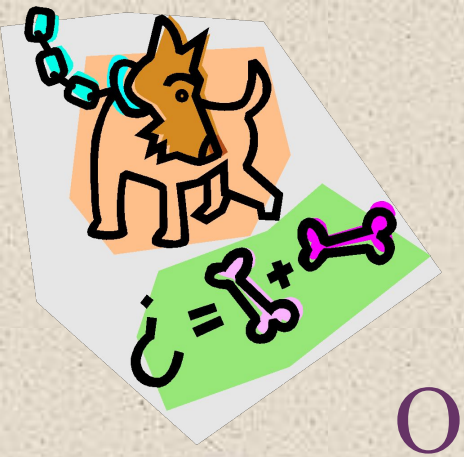
Monitor MSB (sign bit)

It should change during negation

Decimal Representation	Twos Complement Representation
+8	—
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000
–0	—
–1	1111
–2	1110
–3	1101
–4	1100
–5	1011
–6	1010
–7	1001
–8	1000

1001 = –7 +0101 = 5 1110 = –2	1100 = –4 +0100 = 4 10000 = 0
(a) (–7) + (+5)	(b) (–4) + (+4)
0011 = 3 +0100 = 4 0111 = 7	1100 = –4 +1111 = –1 11011 = –5
(c) (+3) + (+4)	(d) (–4) + (–1)
0101 = 5 +0100 = 4 1001 = Overflow	1001 = –7 +1010 = –6 10011 = Overflow
(e) (+5) + (+4)	(f) (–7) + (–6)

Figure 10.3 Addition of Numbers in Twos Complement Representation

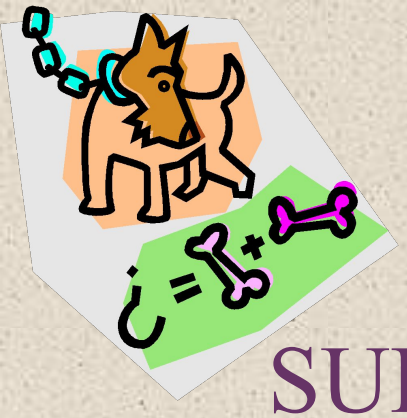


Overflow

Rule

OVERFLOW RULE:

If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the opposite sign.



Subtraction

Rule

SUBTRACTION RULE:

To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.

0010 = 2 +1001 = −7 1011 = −5	0101 = 5 +1110 = −2 10011 = 3
(a) M = 2 = 0010 S = 7 = 0111 −S = 1001	(b) M = 5 = 0101 S = 2 = 0010 −S = 1110
1011 = −5 +1110 = −2 11001 = −7	0101 = 5 +0010 = 2 0111 = 7
(c) M =−5 = 1011 S = 2 = 0010 −S = 1110	(d) M = 5 = 0101 S =−2 = 1110 −S = 0010
0111 = 7 +0111 = 7 1110 = Overflow	1010 = −6 +1100 = −4 10110 = Overflow
(e) M = 7 = 0111 S = −7 = 1001 −S = 0111	(f) M = −6 = 1010 S = 4 = 0100 −S = 1100

Figure 10.4 Subtraction of Numbers in Twos Complement Representation (M – S)

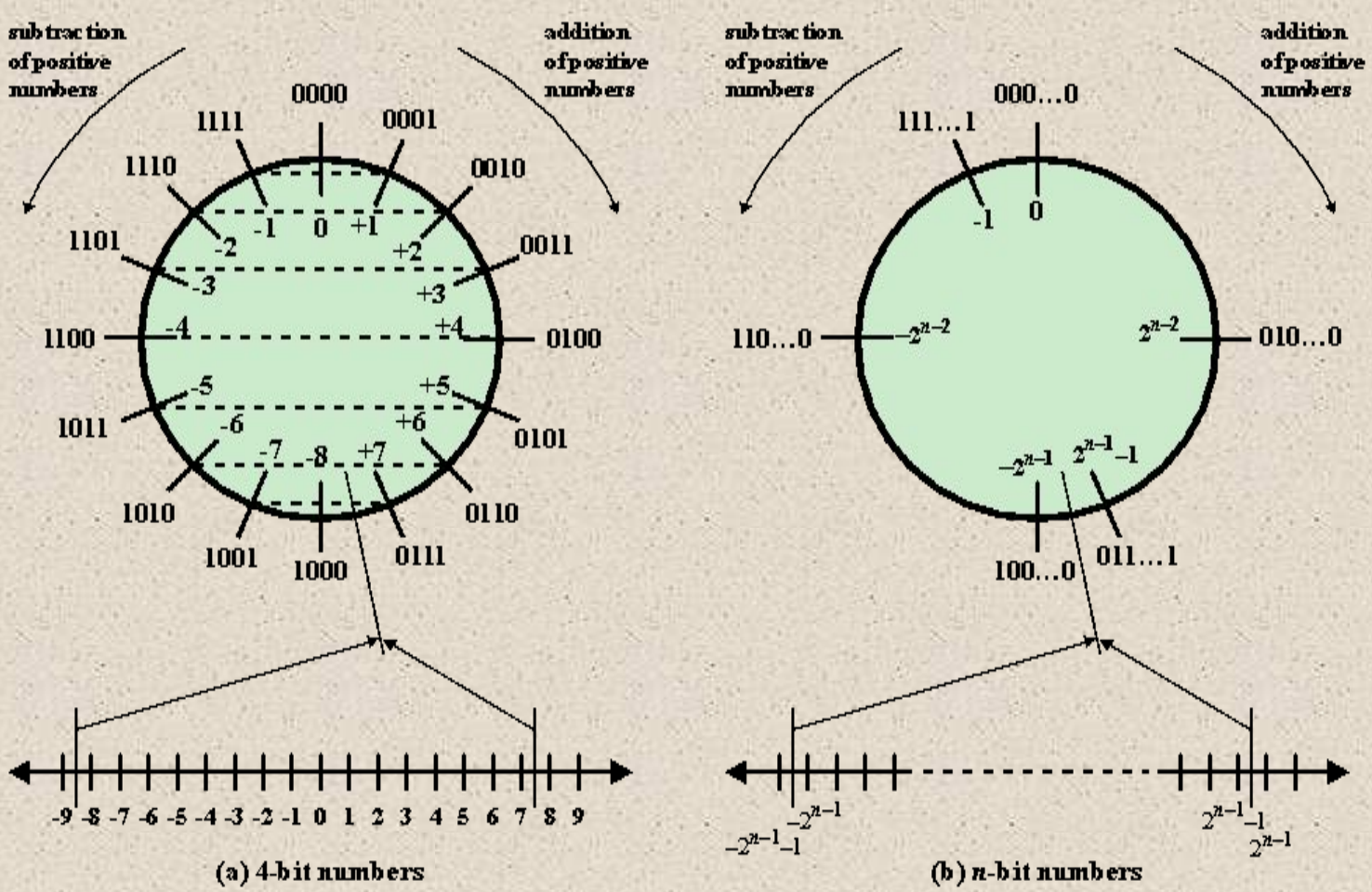
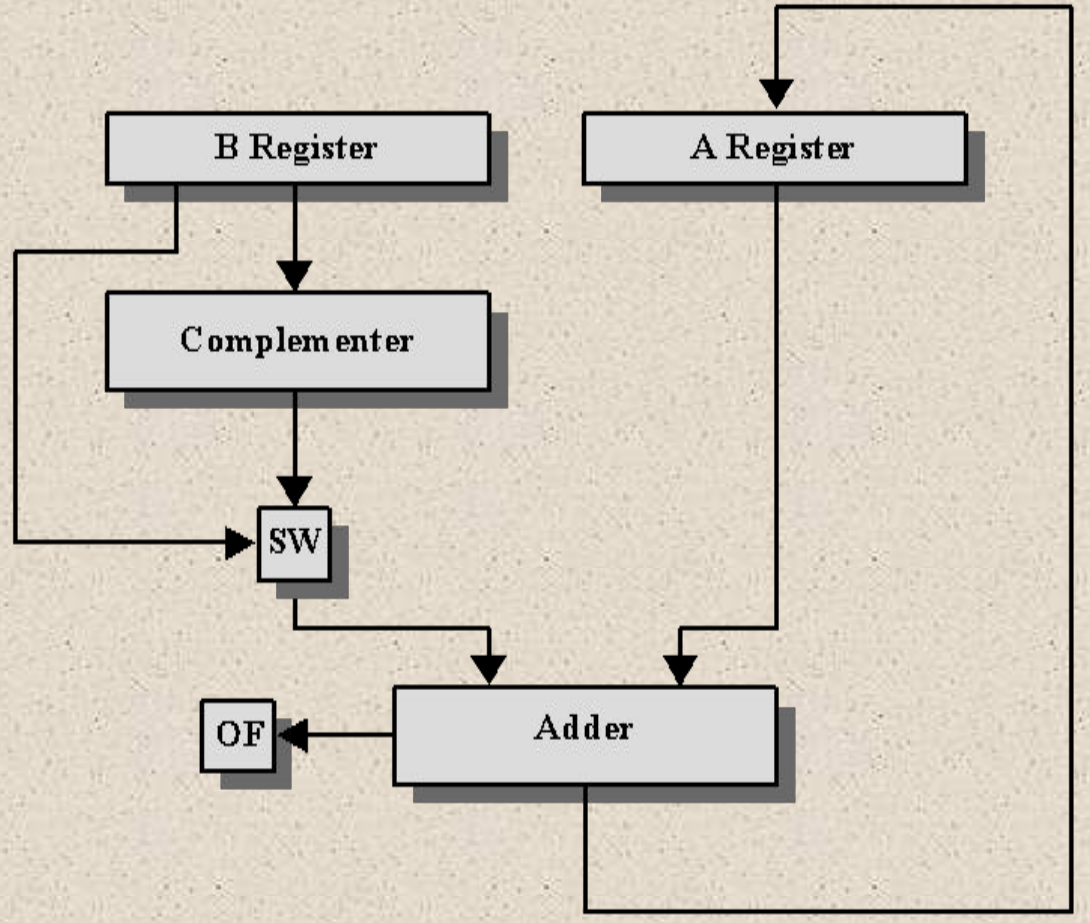


Figure 10.5 Geometric Depiction of Twos Complement Integers

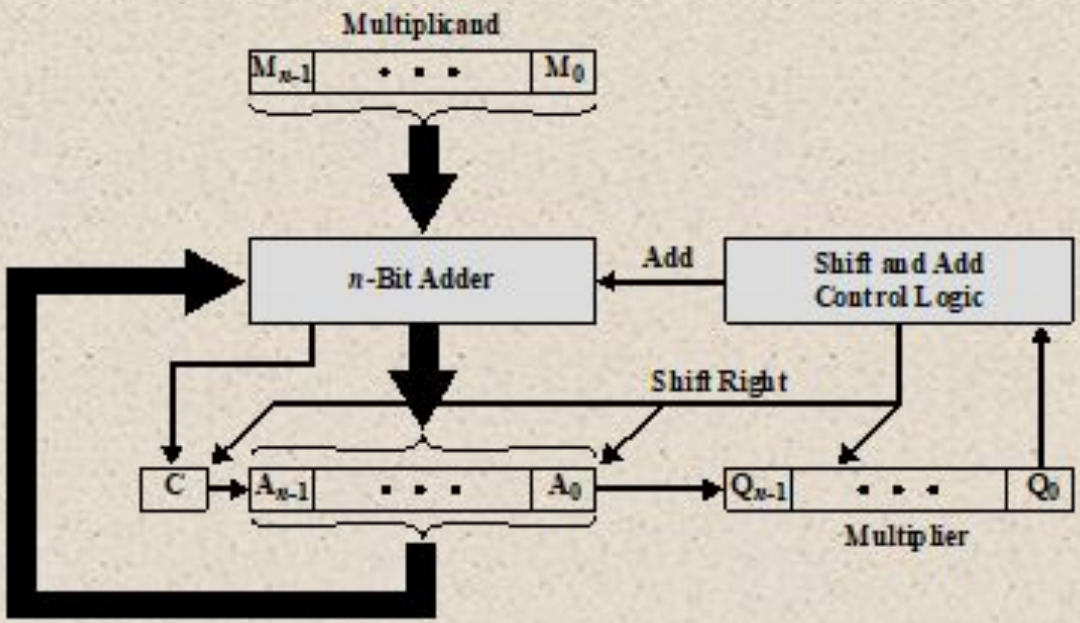


OF = overflow bit
SW = Switch (select addition or subtraction)

Figure 10.6 Block Diagram of Hardware for Addition and Subtraction

```
   1011      Multiplicand (11)
  ×1101      Multiplier (13)
  ------
   1011  }
  0000   }   Partial products
 1011    }
1011     }
--------
10001111     Product (143)
```

Figure 10.7 Multiplication of Unsigned Binary Integers



(a) Block Diagram

C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

(b) Example from Figure 9.7 (product in A, Q)

Figure 10.8 Hardware Implementation of Unsigned Binary Multiplication

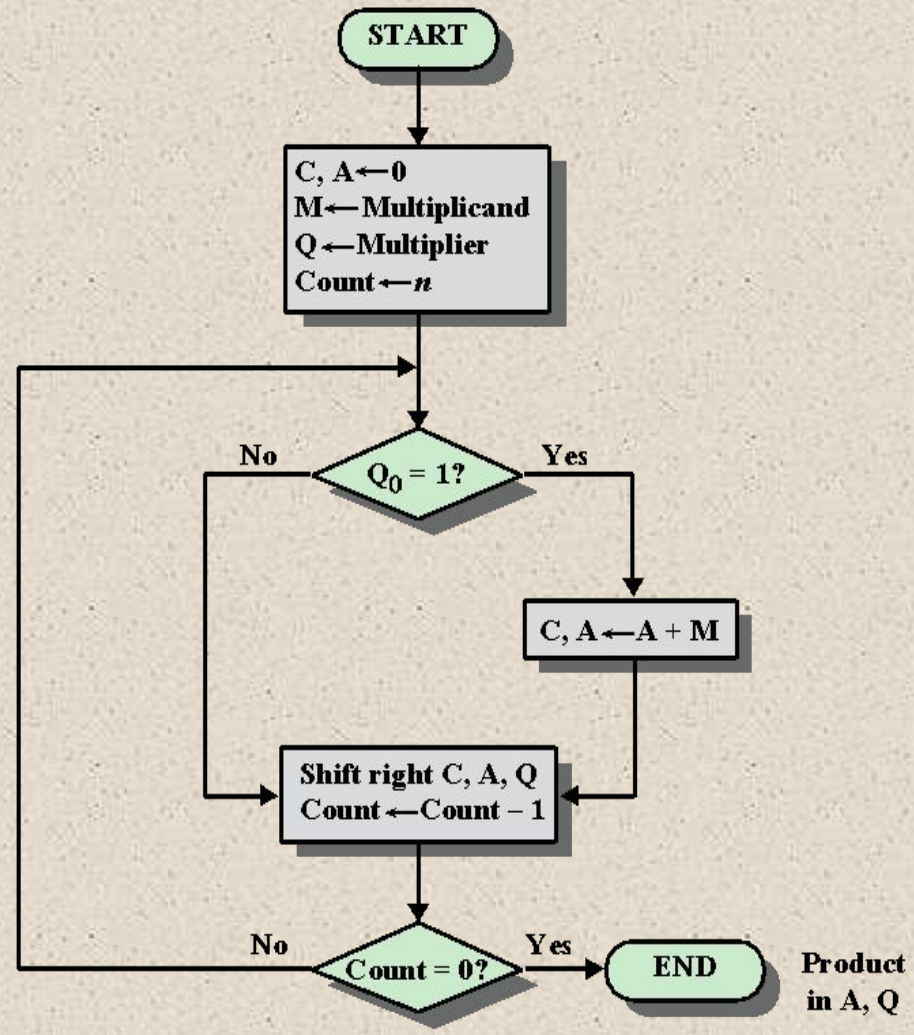


Figure 10.9 Flowchart for Unsigned Binary Multiplication

```
    1011
   ×1101
00001011   1011 × 1 × 2⁰
00000000   1011 × 0 × 2¹
00101100   1011 × 1 × 2²
01011000   1011 × 1 × 2³
10001111
```

Figure 10.10 Multiplication of Two Unsigned 4-Bit Integers Yielding an 8-Bit Result

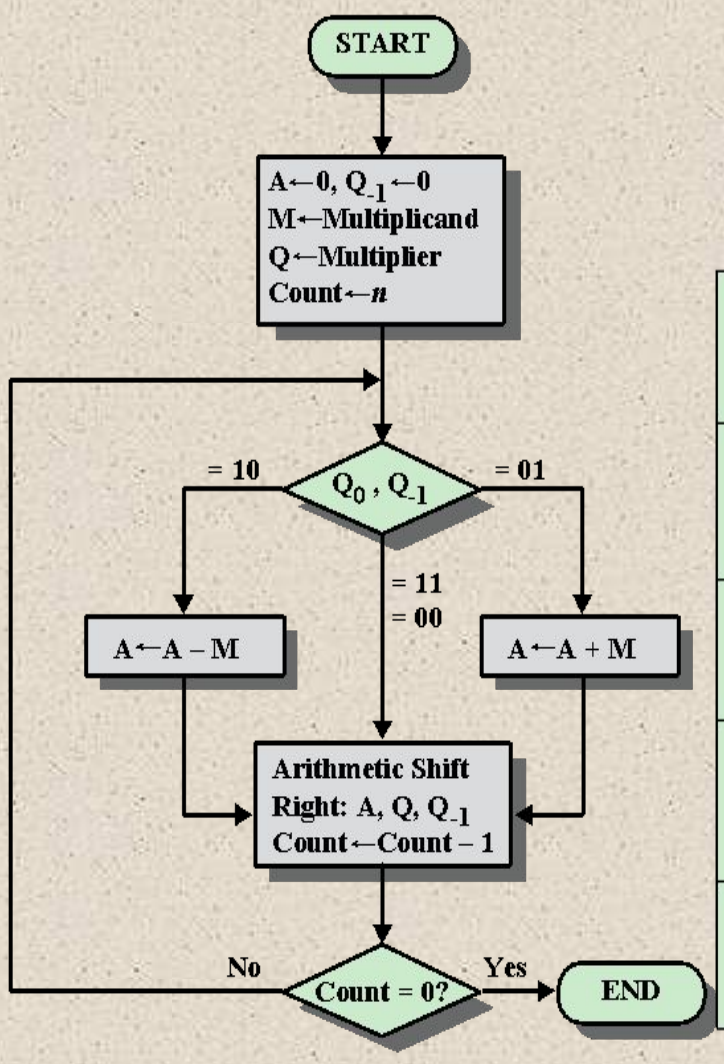
```
    1001  (9)
   ×0011  (3)
00001001  1001 × 2^0
00010010  1001 × 2^1
--------
00011011  (27)
```

(a) Unsigned integers

```
    1001  (-7)
   ×0011  (3)
11111001  (-7) × 2^0 = (-7)
11110010  (-7) × 2^1 = (-14)
--------
11101011  (-21)
```

(b) Twos complement integers

Figure 10.11 Comparison of Multiplication of Unsigned and Twos Complement Integers



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

Figure 10.13 Example of Booth's Algorithm (7× 3)

Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

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

Figure 10.13 Example of Booth's Algorithm (7× 3)

```
    0111
   ×0011      (0)
11111001      1-0
0000000       1-1
000111        0-1
00010101      (21)
```

(a) (7) × (3) = (21)

```
    0111
   ×1101      (0)
11111001      1-0
0000111       0-1
111001        1-0
11101011      (-21)
```

(b) (7) × (–3) = (–21)

```
    1001
   ×0011      (0)
00000111      1-0
0000000       1-1
111001        0-1
11101011      (-21)
```

(c) (–7) × (3) = (–21)

```
    1001
   ×1101      (0)
00000111      1-0
1111001       0-1
000111        1-0
00010101      (21)
```

(d) (–7) × (–3) = (21)

Figure 10.14 Examples Using Booth's Algorithm

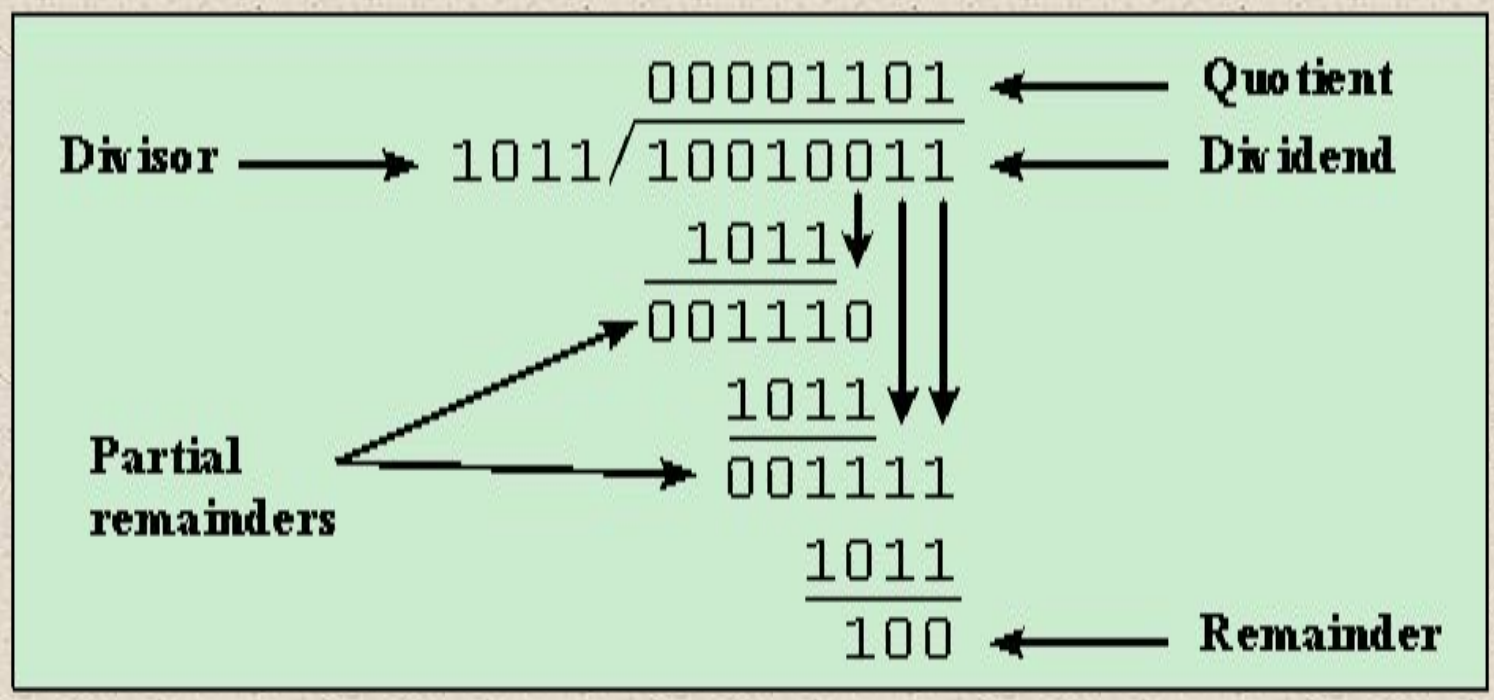


Figure 10.15 Example of Division of Unsigned Binary Integers

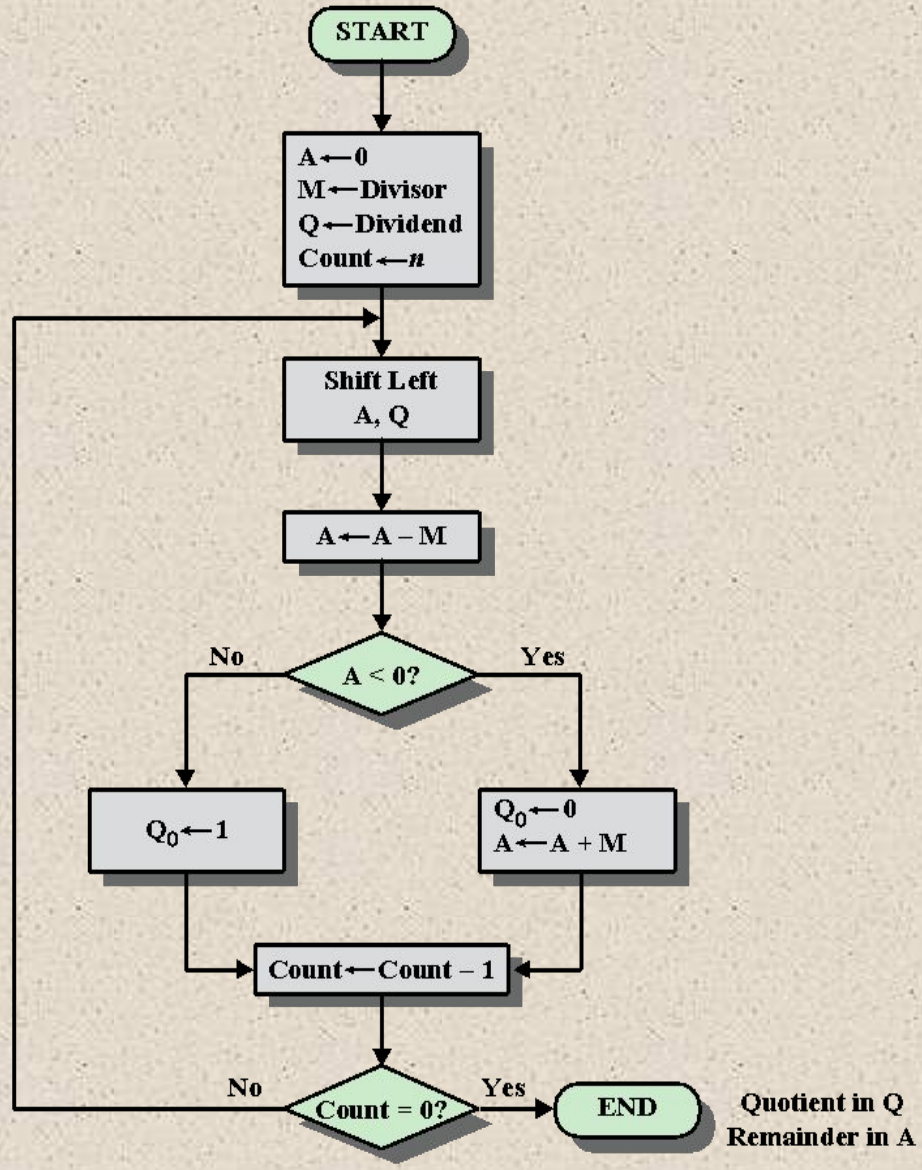


Figure 10.16 Flowchart for Unsigned Binary Division

A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set Q_0 = 0
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set Q_0 = 0
0011	1000	Shift
1101		
0000	1001	Subtract, set Q_0 = 1
0001	0010	Shift
1101		
1110		Subtract
0001	0010	Restore, set Q_0 = 0

Figure 10.17 Example of Restoring Twos Complement Division (7/3)