Computer Arithmetic

ALU

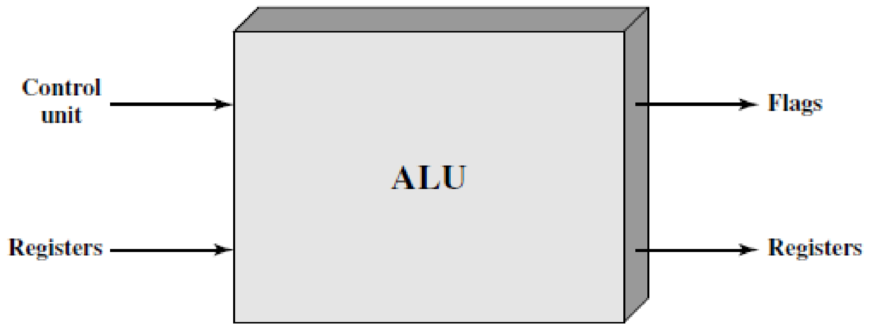
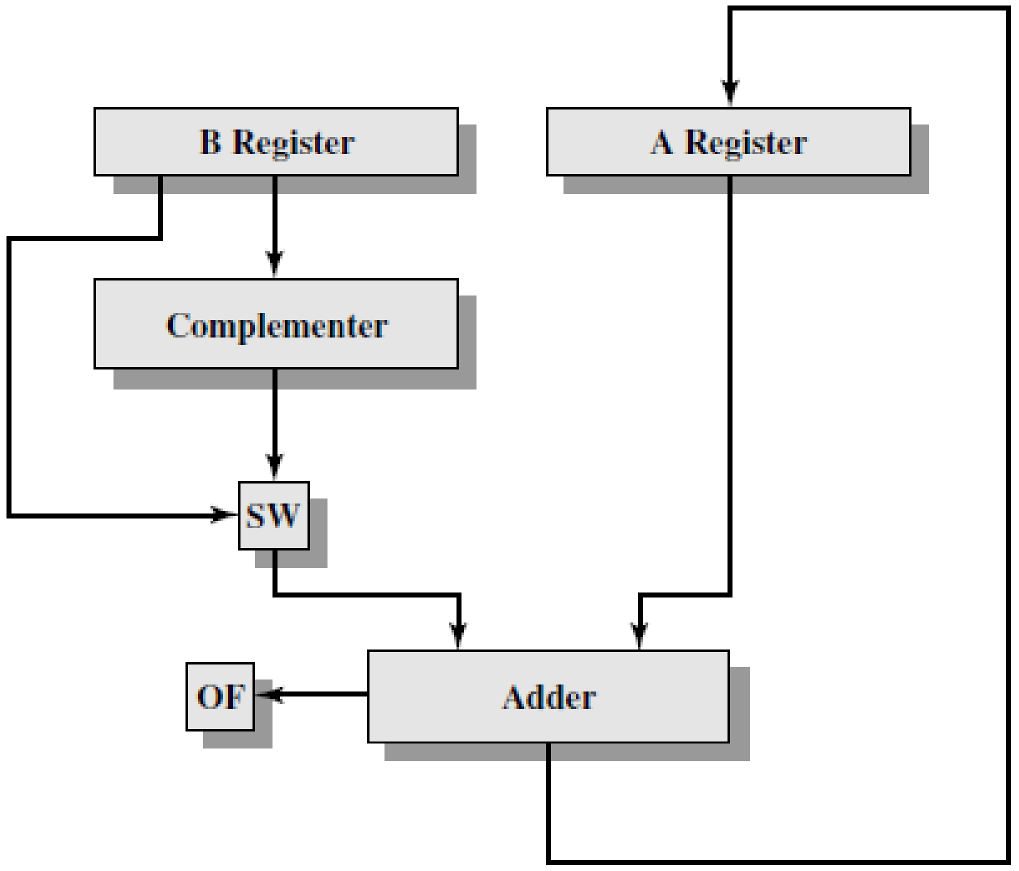


Figure 9.1 ALU Inputs and Outputs

Addition and Subtraction

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.

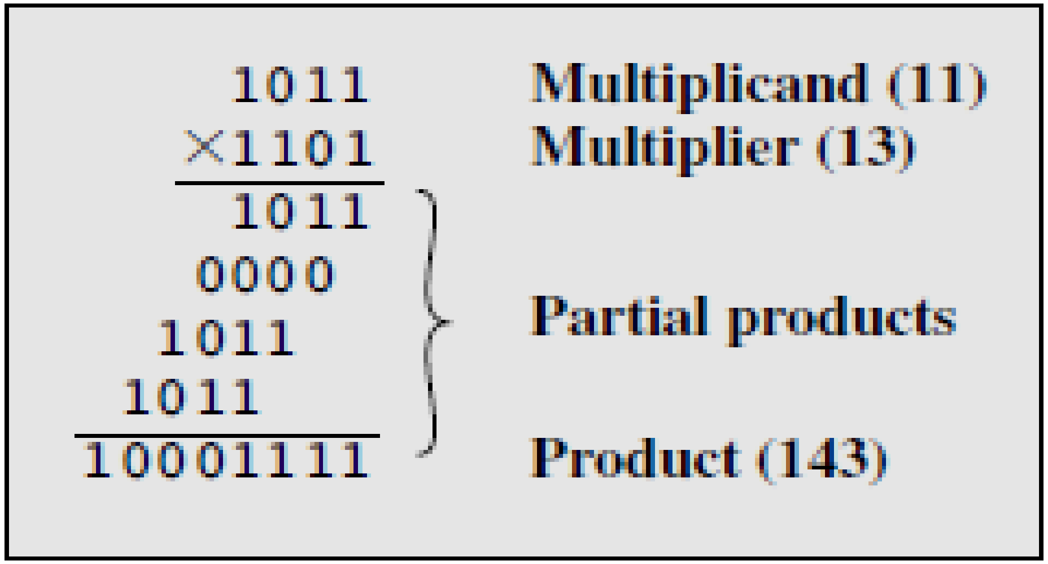
SUBTRACTION RULE: To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.



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

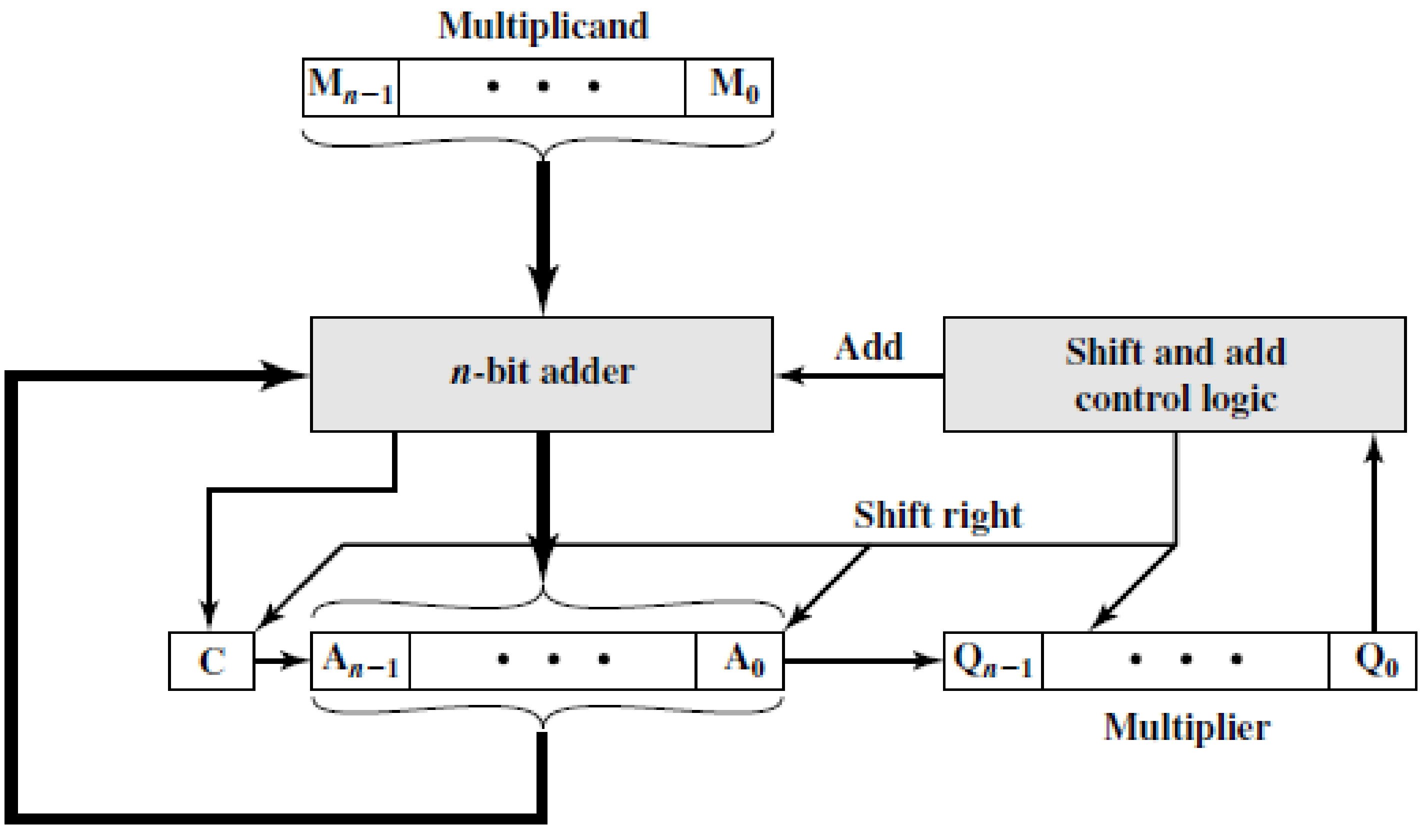
Figure 9.6 Block Diagram of Hardware for Addition and Subtraction

Multiplication



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

Figure 9.7 Multiplication of Unsigned Binary Integers



(a) Block diagram

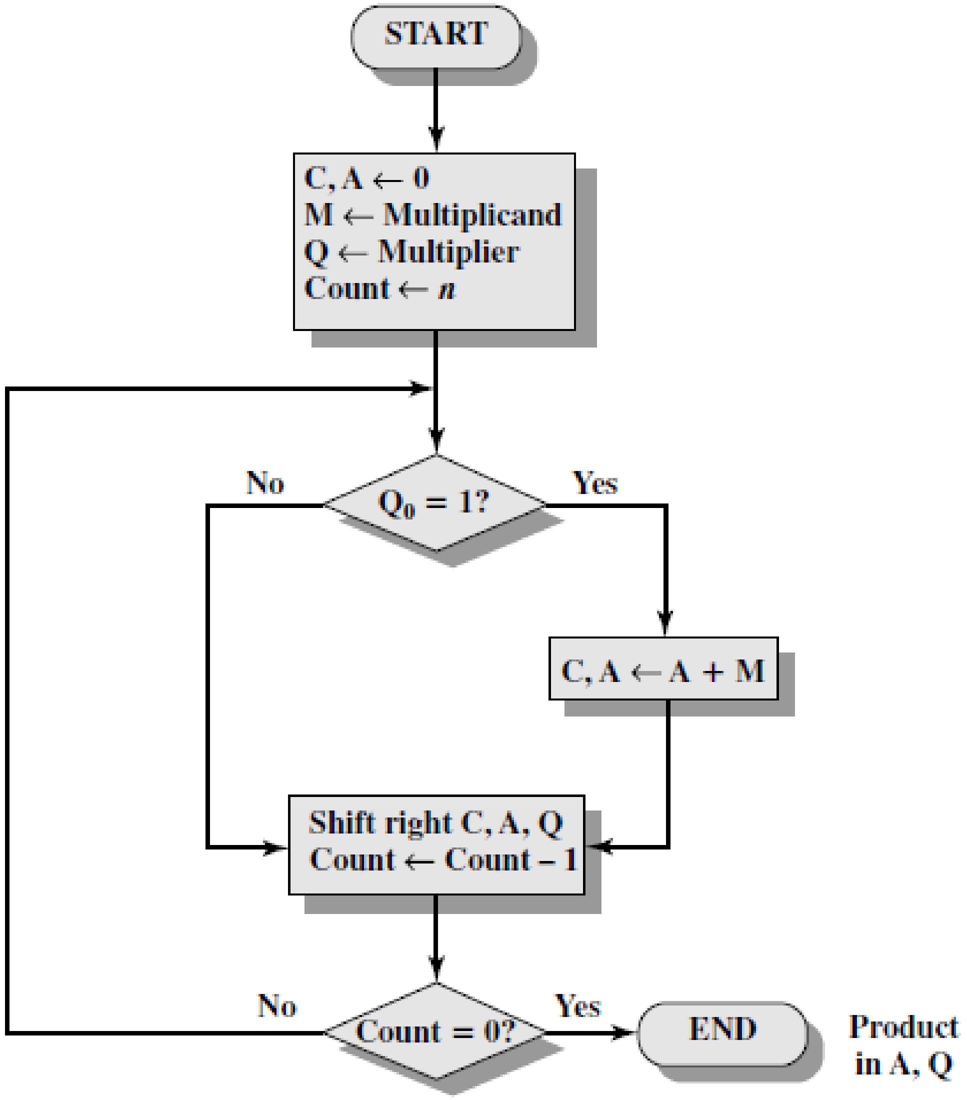


Figure 9.9 Flowchart for Unsigned Binary Multiplication

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	

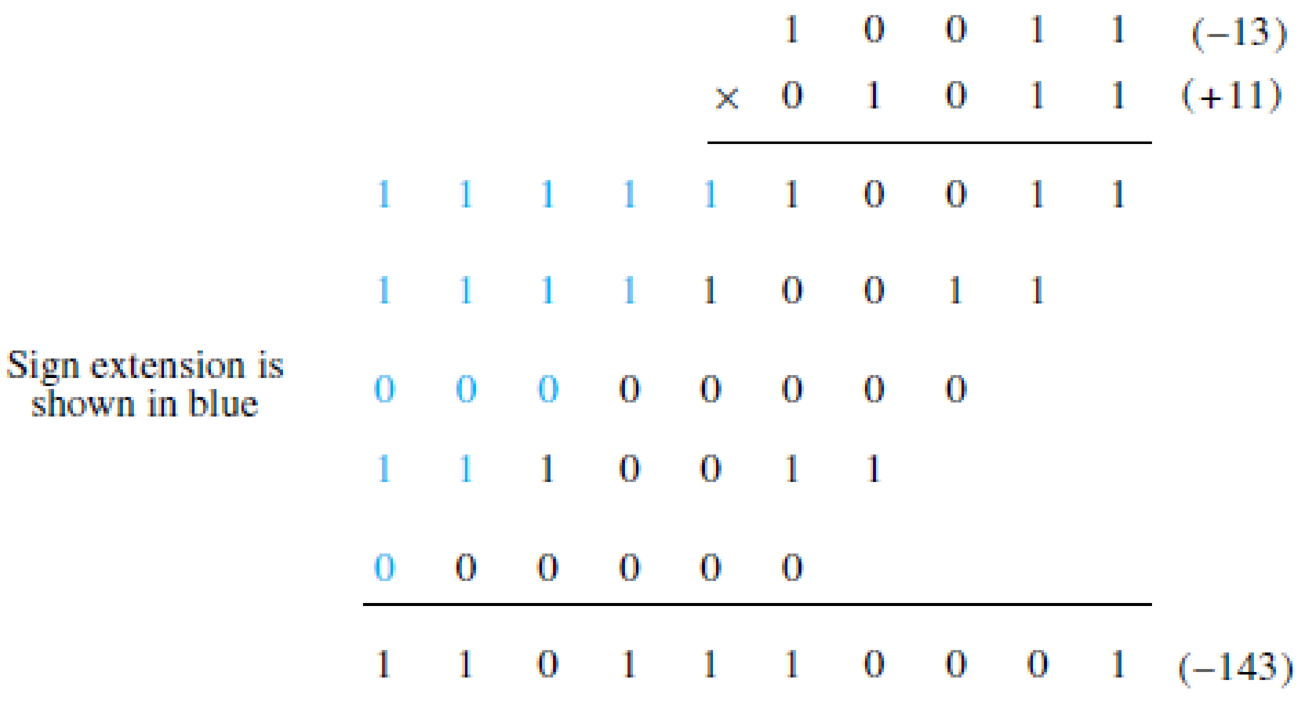
(b) Example from Figure 9.7 (product in A, Q)

Figure 9.8 Hardware Implementation of Unsigned Binary Multiplication

$$
\begin{array}{r l}
1011 & \\
\times\ 1101 & \\
\hline
00001011 & 1011 \times 1 \times 2^0 \\
00000000 & 1011 \times 0 \times 2^1 \\
00101100 & 1011 \times 1 \times 2^2 \\
01011000 & 1011 \times 1 \times 2^3 \\
\hline
10001111 &
\end{array}
$$

Figure 9.10 Multiplication of Two Unsigned 4-Bit Integers Yielding an 8-Bit Result

Negative Multiplicand



```
                    1  0  0  1  1   (−13)
                 ×  0  1  0  1  1   (+11)
                 ----------------
     1  1  1  1  1  1  0  0  1  1
     1  1  1  1  1  0  0  1  1
     0  0  0  0  0  0  0  0
     1  1  1  0  0  1  1
     0  0  0  0  0  0
     ----------------------------
     1  1  0  1  1  1  0  0  0  1   (−143)
```

Figure 9.8 Sign extension of negative multiplicand.

```
    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 9.11 Comparison of Multiplication of Unsigned and Twos Complement Integers

Booth's Algorithm

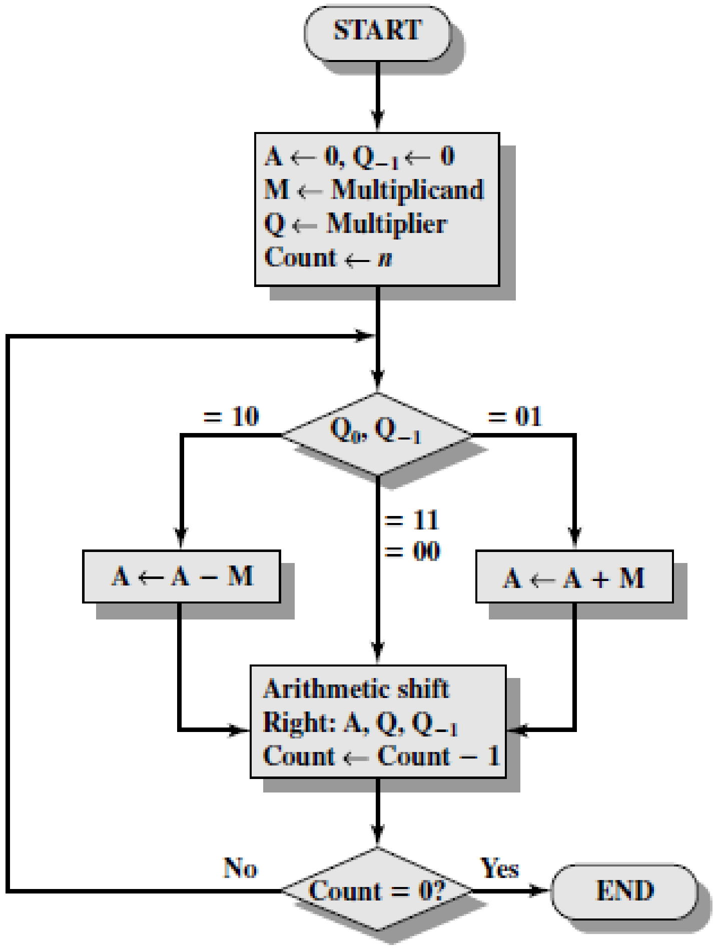


Figure 9.12 Booth's Algorithm for Twos Complement Multiplication

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 9.13 Example of Booth's Algorithm (7×3)

How Booth's algorithm works

Consider first the case of a positive multiplier

- consisting of one block of 1s surrounded by 0s

$$M * (00011110) = M * (2^4 + 2^3 + 2^2 + 2^1)$$
$$= M * (16 + 8 + 4 + 2)$$
$$= M * 30$$

$$M * (00011110) = M * (2^5 - 2^1)$$
$$= M * (32 - 2)$$
$$= M * 30$$

the product can be generated by one addition and one subtraction of the multiplicand.

Booth's algorithm conforms to this scheme by performing a subtraction when the first 1 of the block is encountered (1–0) and an addition when the end of the block is encountered (0–1).

Let *X* be a negative number in twos complement notation

Representation of $X = \{1x_{n-2}x_{n-3} \ldots\ldots x_1x_0\}$

$$X = -2^{n-1} + (x_{n-2} \times 2^{n-2}) + (x_{n-3} \times 2^{n-3}) + \cdots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

$$\text{Representation of } X = \{111 \ldots\ 10x_{k-1}x_{k-2} \ldots\ x_1x_0\}$$

$$X = -2^{n-1} + 2^{n-2} + \cdots + 2^{k+1} + (x_{k-1} \times 2^{k-1}) + \cdots + (x_0 \times 2^0)$$

$$2^{n-2} + 2^{n-3} + \cdots + 2^{k+1} = 2^{n-1} - 2^{k+1}$$

$$-2^{n-1} + 2^{n-2} + 2^{n-3} + \cdots + 2^{k+1} = -2^{k+1}$$

$$X = -2^{k+1} + (x_{k-1} \times 2^{k-1}) + \cdots + (x_0 \times 2^0)$$

Consider the multiplication of some multiplicand by (-6). In twos complement representation, using an 8-bit word, (-6) is represented as

11111010.

$-6 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1$

$M * (11111010) = M * (--2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^1)$

$M * (11111010) = M * (-2^3 + 2^1)$

$M * (11111010) = M * (-2^3 + 2^2 - 2^1)$

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.

```
      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 9.14 Examples Using Booth's Algorithm

Exercise

- Given x and y in twos complement notation i.e., x=0101 and y=1010,compute the product p=x*y with Booth’s algorithm
- Use the Booth algorithm to multiply 23 (multiplicand) by 29 (multiplier), where each number is represented using 6 bits

Division

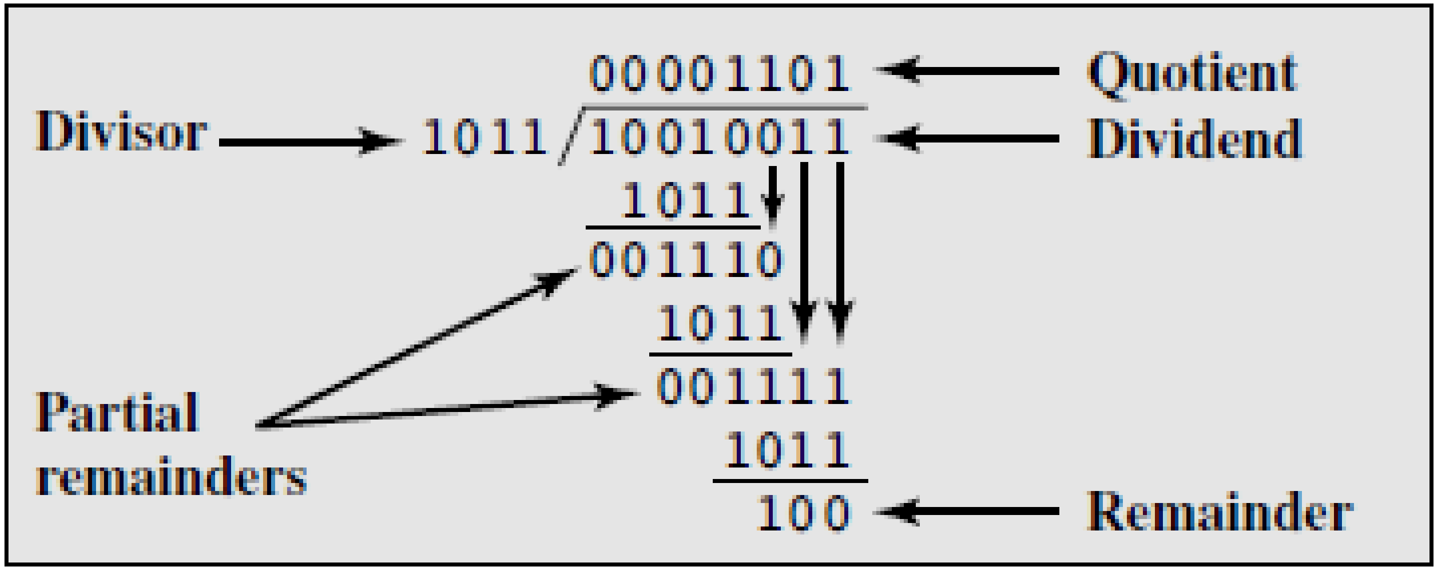


Figure 9.15 Example of Division of Unsigned Binary Integers

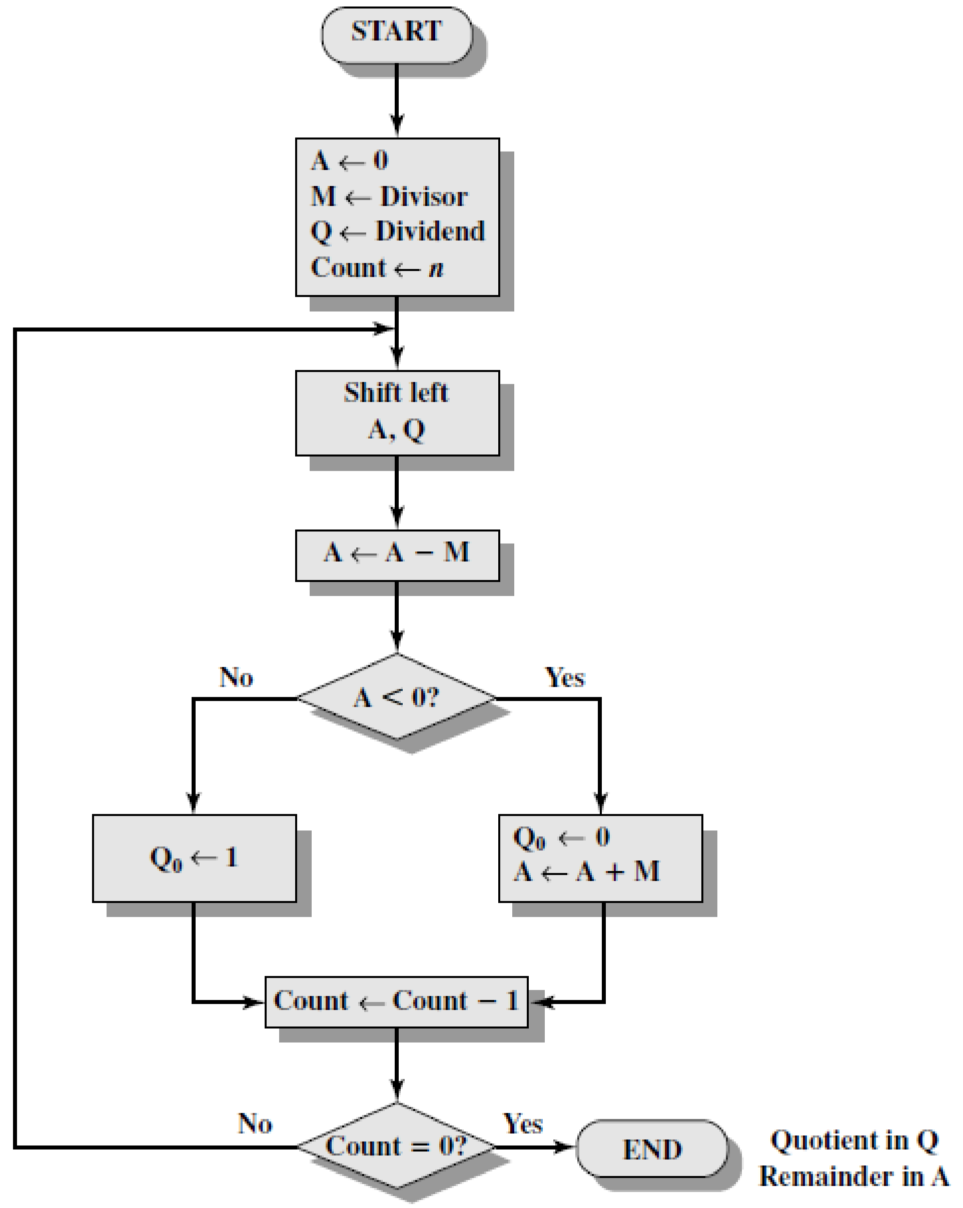


Figure 9.16 Flowchart for Unsigned Binary Division

The algorithm assumes that the divisor V and the dividend D are positive and that |V|= |D |V|< |D|. If |V|= |D|, then the quotient =1and the remainder=0. If |V|> |D|, then Q=0 and R=D.The algorithm can be summarized as follows:

1. Load the twos complement of the divisor into the M register; that is, the M register contains the negative of the divisor. Load the dividend into the A, Q registers.

The dividend must be expressed as a 2n-bit positive number. Thus, for example, the 4-bit 0111 becomes 00000111.

2. Shift A, Q left 1 bit position.

3. Perform This operation subtracts the divisor from the contents of A.

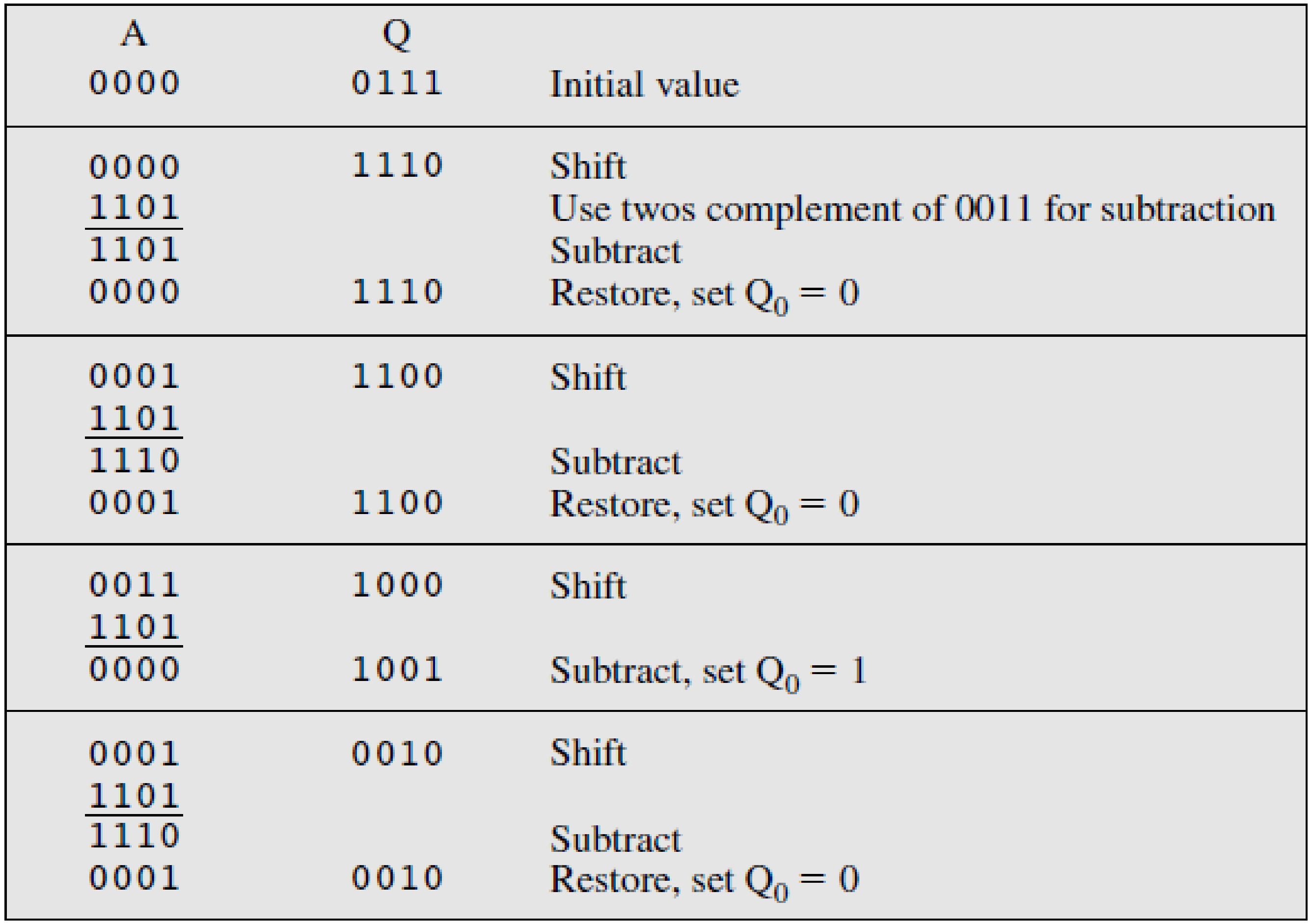
4. a. If the result is nonnegative (most significant bit of A=0), then set Q0=1

b. If the result is negative (most significant bit of A=1), then set Q0=0, and restore the previous value of A.

5. Repeat steps 2 through 4 as many times as there are bit positions in Q.

6. The remainder is in A and the quotient is in Q.

Example of Restoring Twos Complement Division (7/3)



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 $Q_0 = 0$

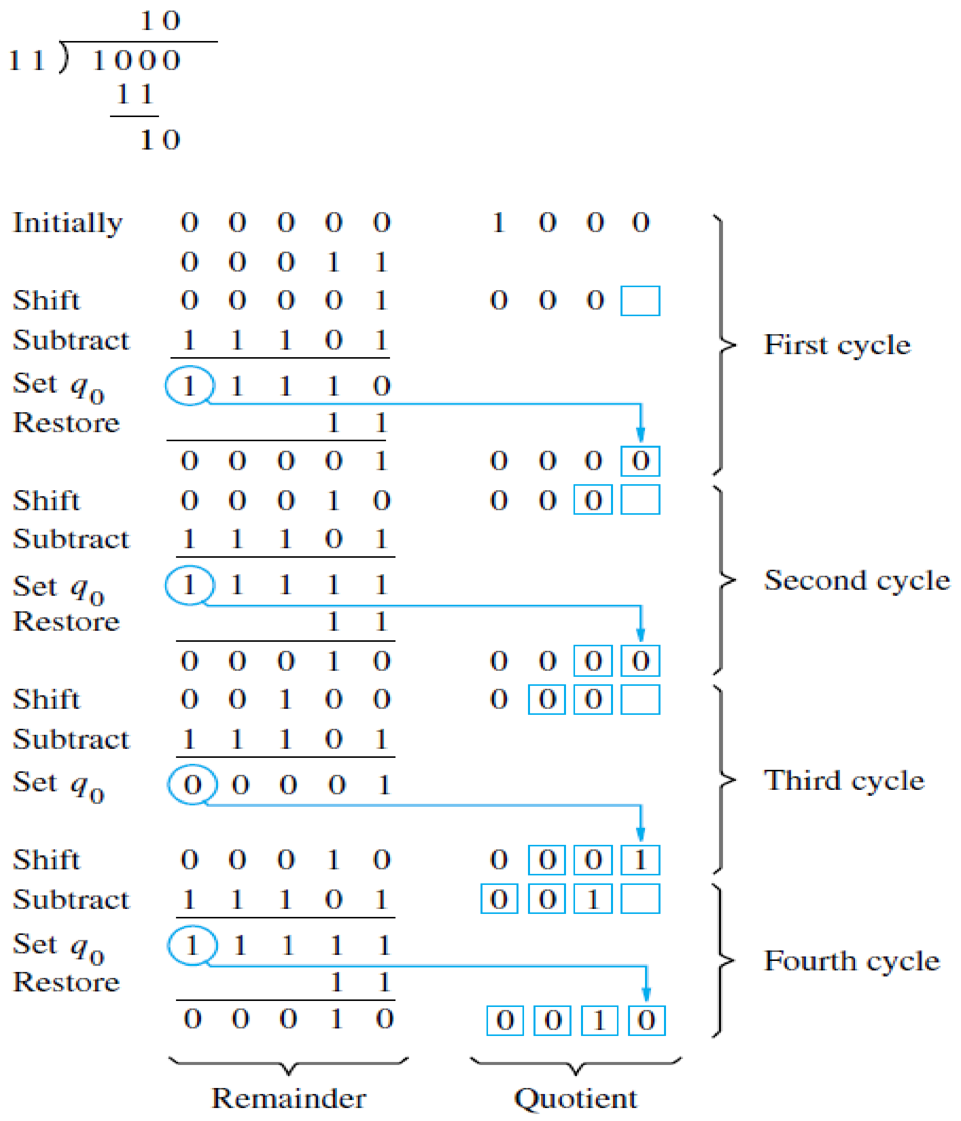


Figure 9.24 A restoring division example.

Consider the following examples of integer division with all possible combinations of signs of D and V:

$$\begin{array}{llcll}
D = 7 & V = 3 & \Rightarrow & Q = 2 & R = 1 \\
D = 7 & V = -3 & \Rightarrow & Q = -2 & R = 1 \\
D = -7 & V = 3 & \Rightarrow & Q = -2 & R = -1 \\
D = -7 & V = -3 & \Rightarrow & Q = 2 & R = -1
\end{array}$$

The reader will note from Figure 9.17 that $(-7)/(3)$ and $(7)/(-3)$ produce different remainders. We see that the magnitudes of Q and R are unaffected by the input signs and that the signs of Q and R are easily derivable form the signs of D and V. Specifically, $\text{sign}(R) = \text{sign}(D)$ and $\text{sign}(Q) = \text{sign}(D) \times \text{sign}(V)$. Hence, one way to do twos complement division is to convert the operands into unsigned values and, at the end, to account for the signs by complementation where needed. This is the method of choice for the restoring division algorithm [PARH00].

Exercise

- Divide 145 by 13 in binary twos complement notation, using 12-bit words. Use the restoring division algorithm

Reference

William Stallings, “Computer Organization and Architecture – Designing for Performance”, 9th edition, PHI, 2015.