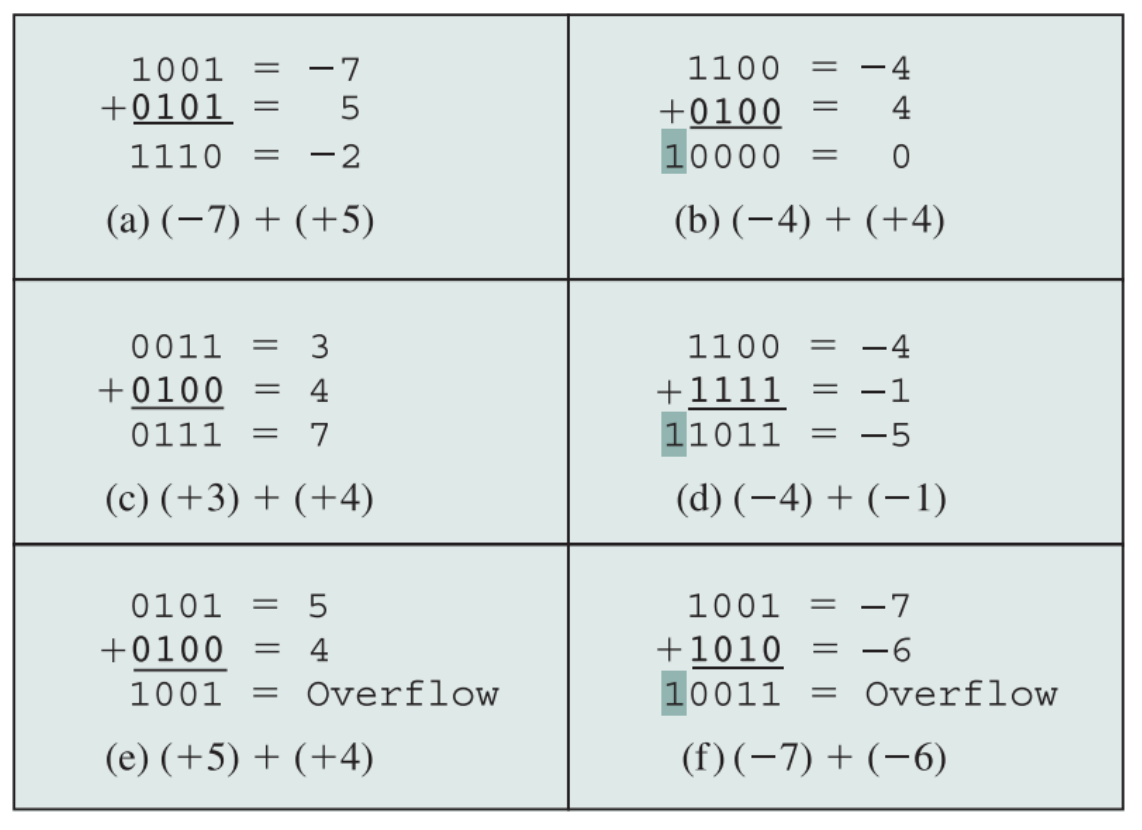
Computer Arithmetic: Part 2 (Addition Subtraction & Multiplication Algorithms)

Computer organization and architecture: Designing for performance, Author: William Stallings

Addition

- Addition proceeds as if the two numbers were unsigned integers.
- If the result of the operation is positive, we get a positive number in twos complement form, which is the same as in unsigned-integer form.
- If the result of the operation is negative, we get a negative number in twos complement form.
- Note that, in some instances, there is a carry bit beyond the end of the word (indicated by shading), which is ignored.



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

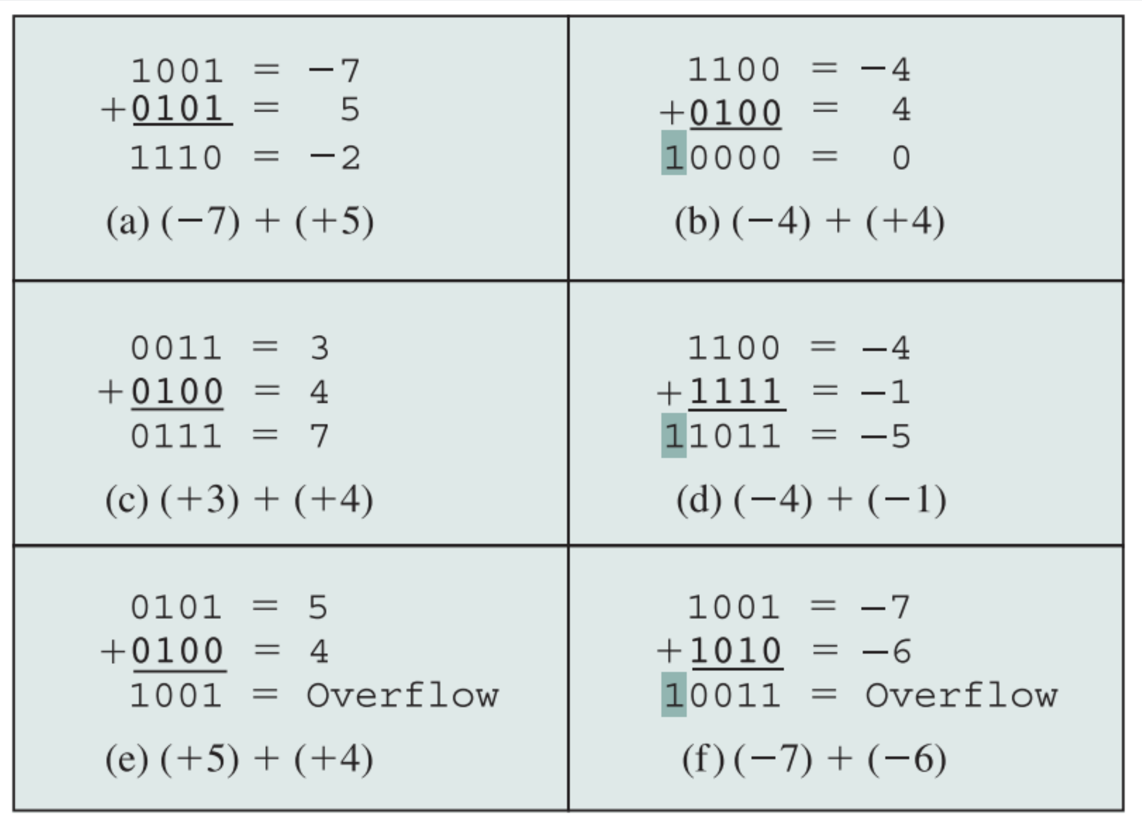
Computer organization and architecture: designing for performance, Author: William Stallings

Overflow?

Overflow occurs when:

- Two negative numbers are added and an answer comes positive or
- Two positive numbers are added and an answer comes as negative.
- Note that overflow can occur whether or not there is a carry

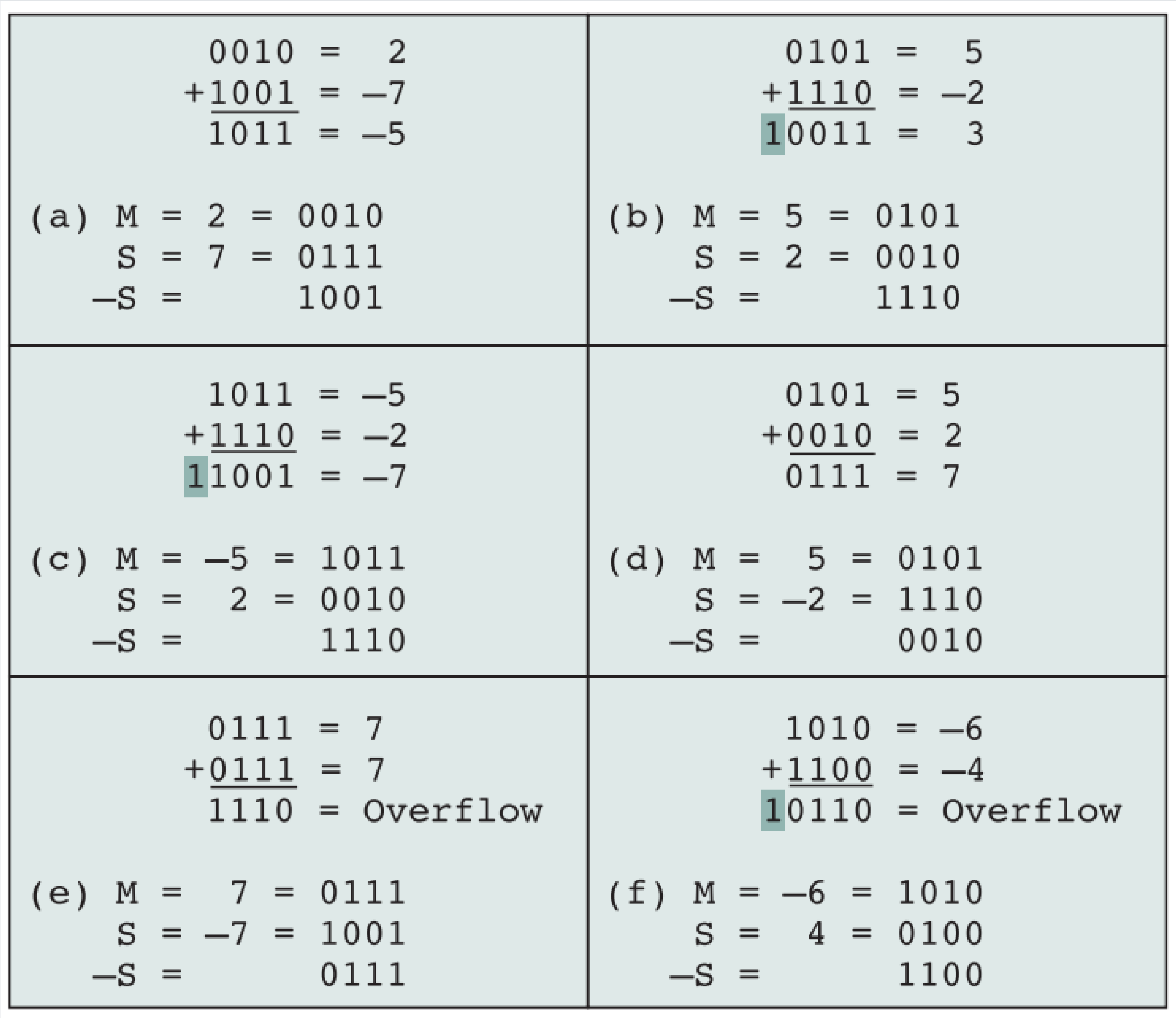
- N-bit 2’s Complement number System can represent Number from to -2^{n-1} to $2^{n-1}-1$
  4 Bit can represent numbers from **(-8 to 7)**
  5 Bit can represent numbers from **(-16 to 15)** in 2’s Complimentary System.



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

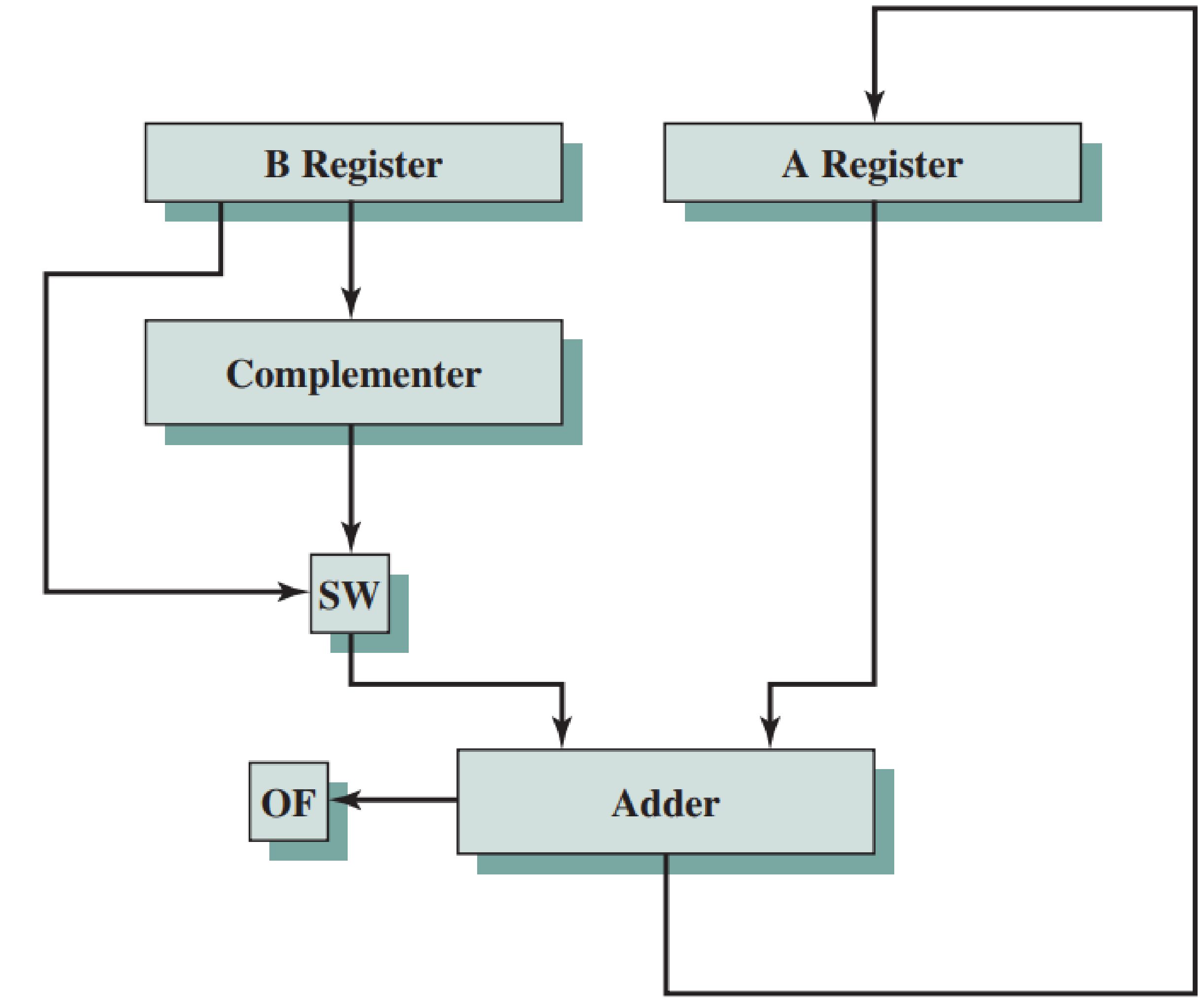
Subtraction

- To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.
- Subtraction is achieved using addition.



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

Block Diagram of Hardware for Addition and Subtraction



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

Multiplication

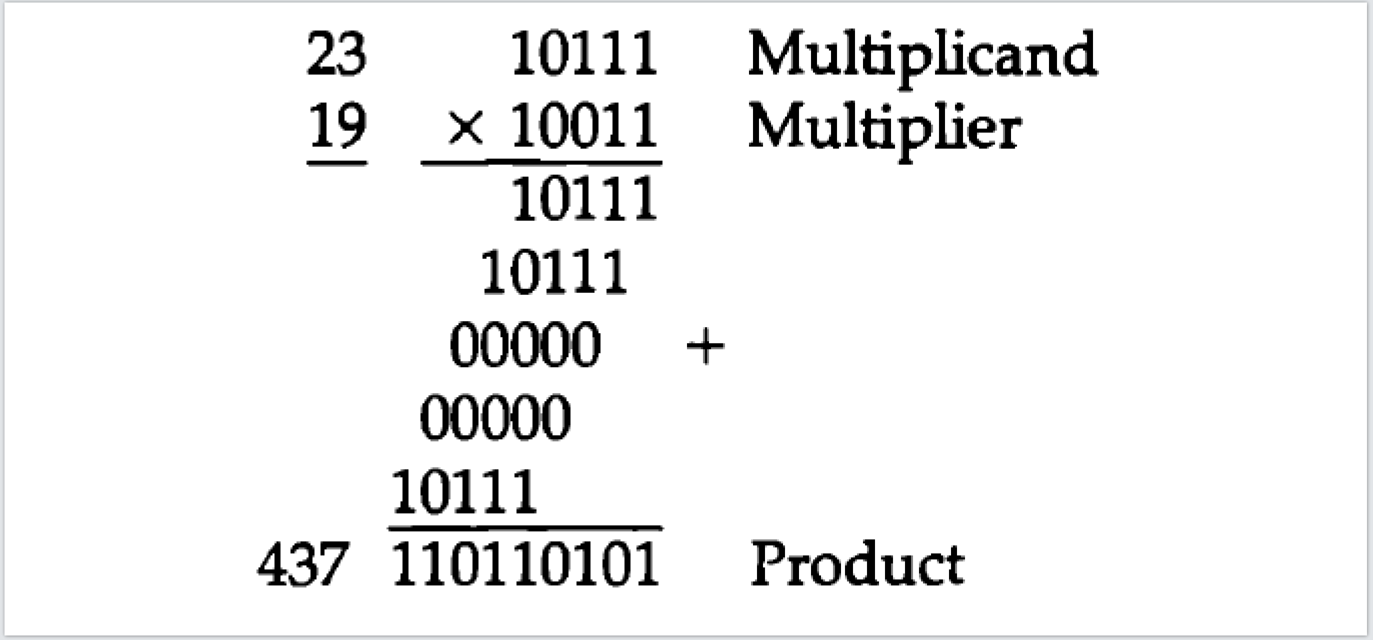
TERMS to know

```
Terms   :  Multiplicand  :          4
           Multiplier    :    X     3
--------------------------------------------------------
           Product       :         12
```

Binary Multiplication

A	B	A X B
0	0	0
0	1	0
1	0	0
1	1	1

Multiplication of unsigned integers, Pen and Paper Method



```
 23       10111    Multiplicand
 19     × 10011    Multiplier
         10111
        10111
       00000    +
      00000
     10111
437  110110101     Product
```

1. Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.

2. The partial products are easily defined. When the multiplier bit is 0, the partial product is 0. When the multiplier is 1, the partial product is the multiplicand.

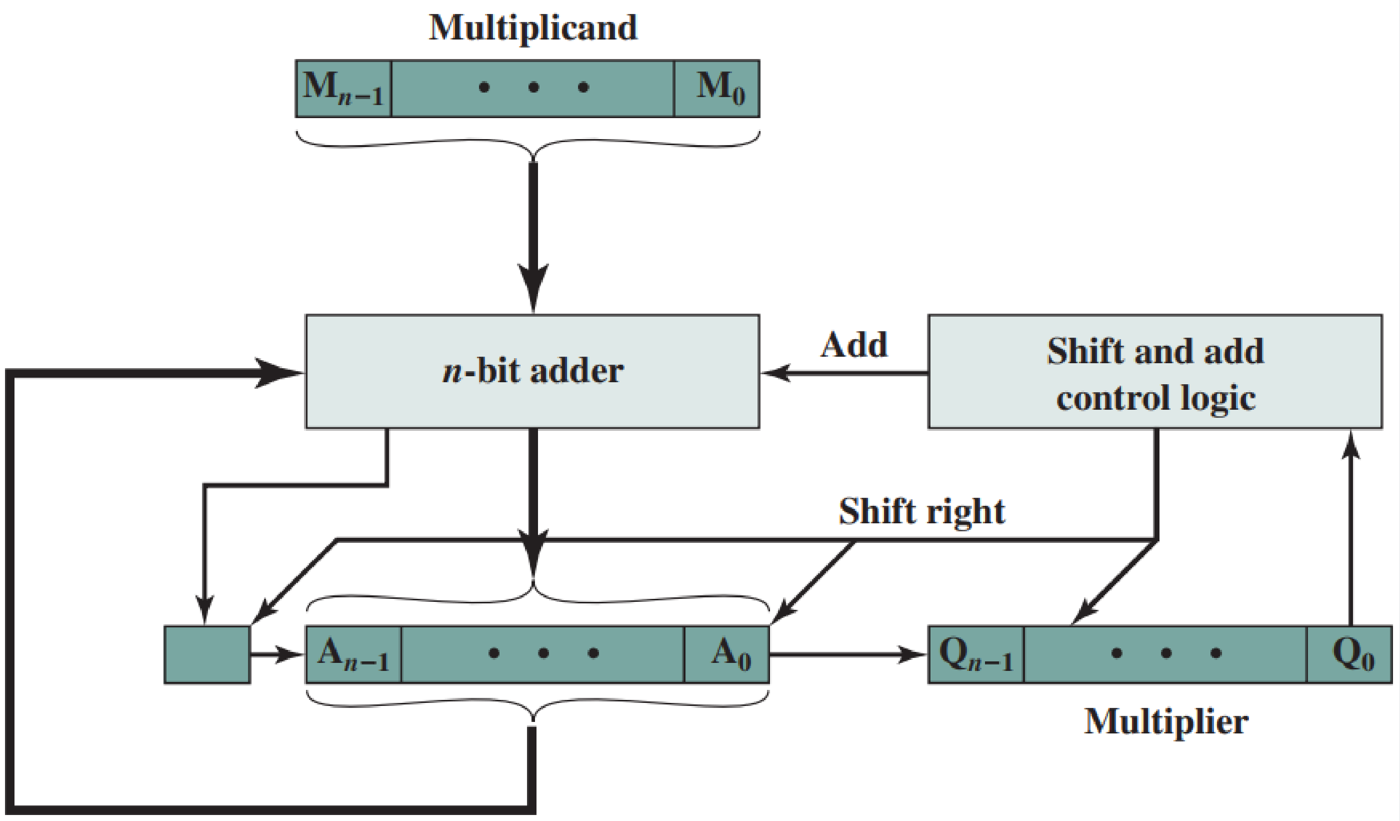
3. The total product is produced by summing the partial products. For this operation, each successive partial product is shifted one position to the left relative to the preceding partial product

4. The multiplication of two n-bit binary integers results in a product of up to 2n bits in length (e.g., 11 * 11 = 1001).

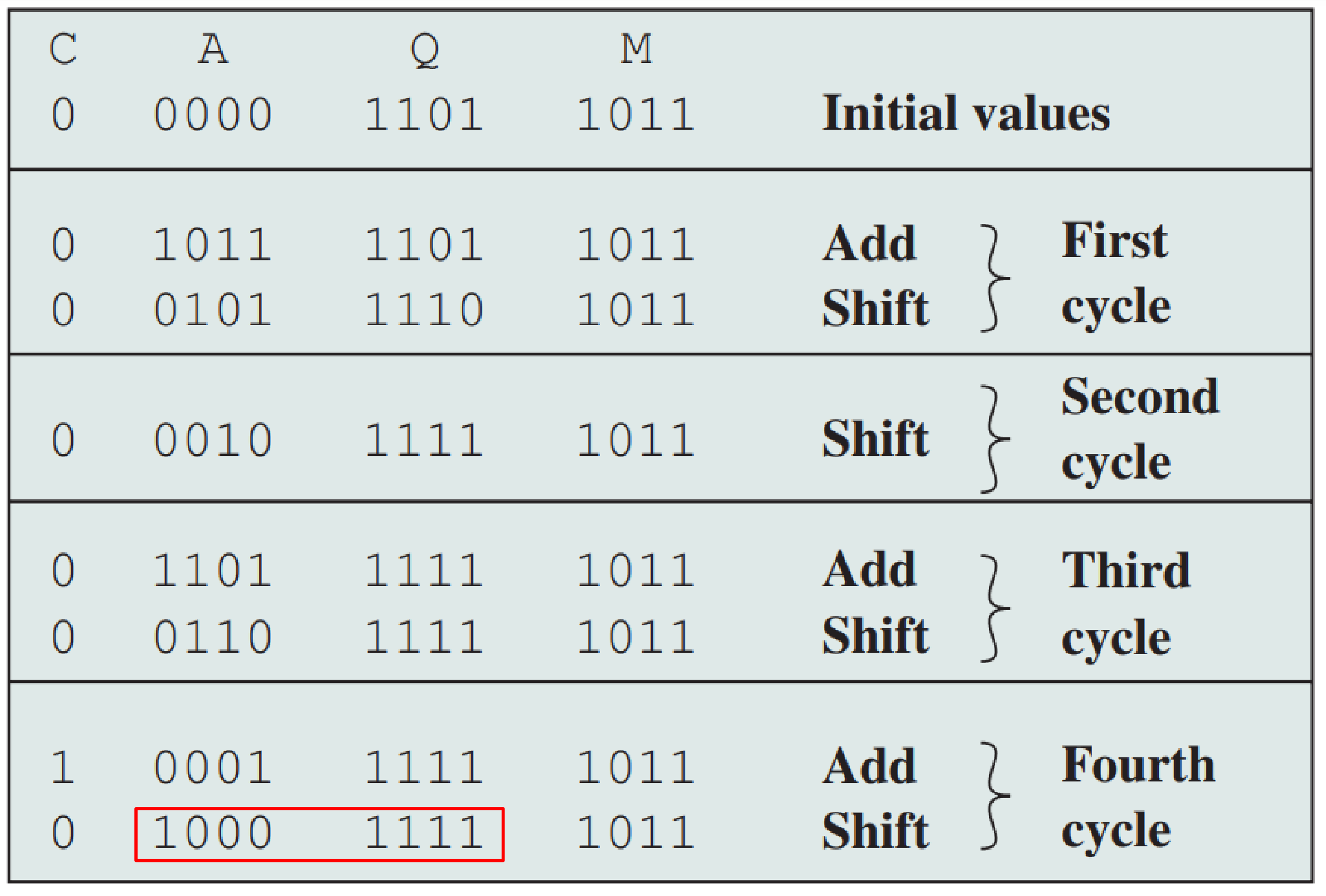
Unsigned Integer Multiplication

Improving the pen and paper method.

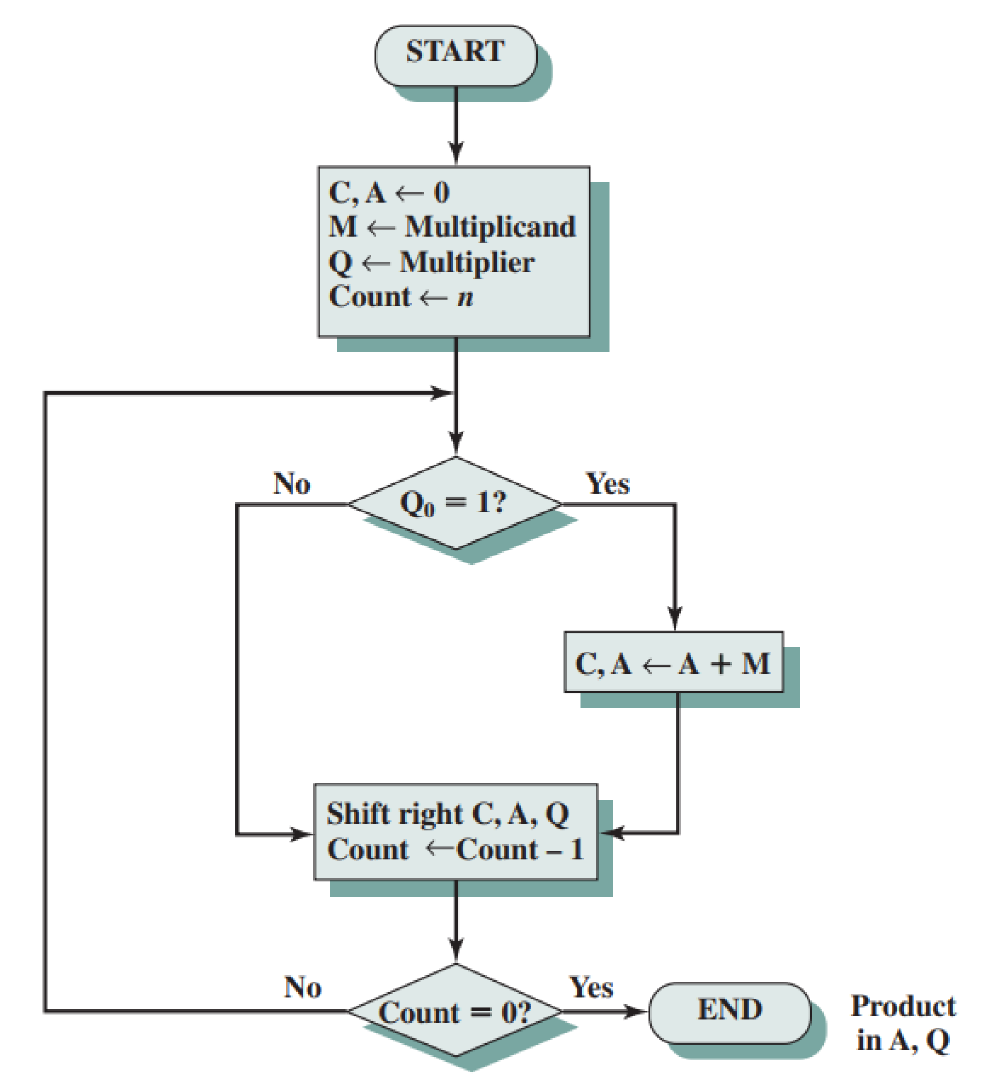
- First, we can perform a running addition on the partial products rather than waiting until the end.
- This eliminates the need for storage of all the partial products; fewer registers are needed.
- Second, we can save some time on the generation of partial products.
- For each 1 on the multiplier, an add and a shift operation are required; but for each 0, only a shift is required.



Flowchart and Example 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	



11*13 = 143; i.e., 1000 1111

Booth's Multiplication Algorithm

- **Multiplying binary integers** in signed 2’s complement representation
- Lesser number of additions/subtractions required.
- For 0's in the multiplier, no addition just shifting
- And a strings of 1 in the multiplier from bit weight 2^k to 2^m can be treated as 2^{k+1} to 2^m.
- Example:

  15 = (1111), then 4 X 15 = 4 X 2^4 – 4 X 2^0= 64 – 4 = 60

- The multiplier and multiplicand are placed in the Q and M registers. There is also a 1-bit register placed logically to the right of the least significant bit (Q_0) of the Q register and designated Q_{-1}

- The results of the multiplication will appear in the A and Q registers.

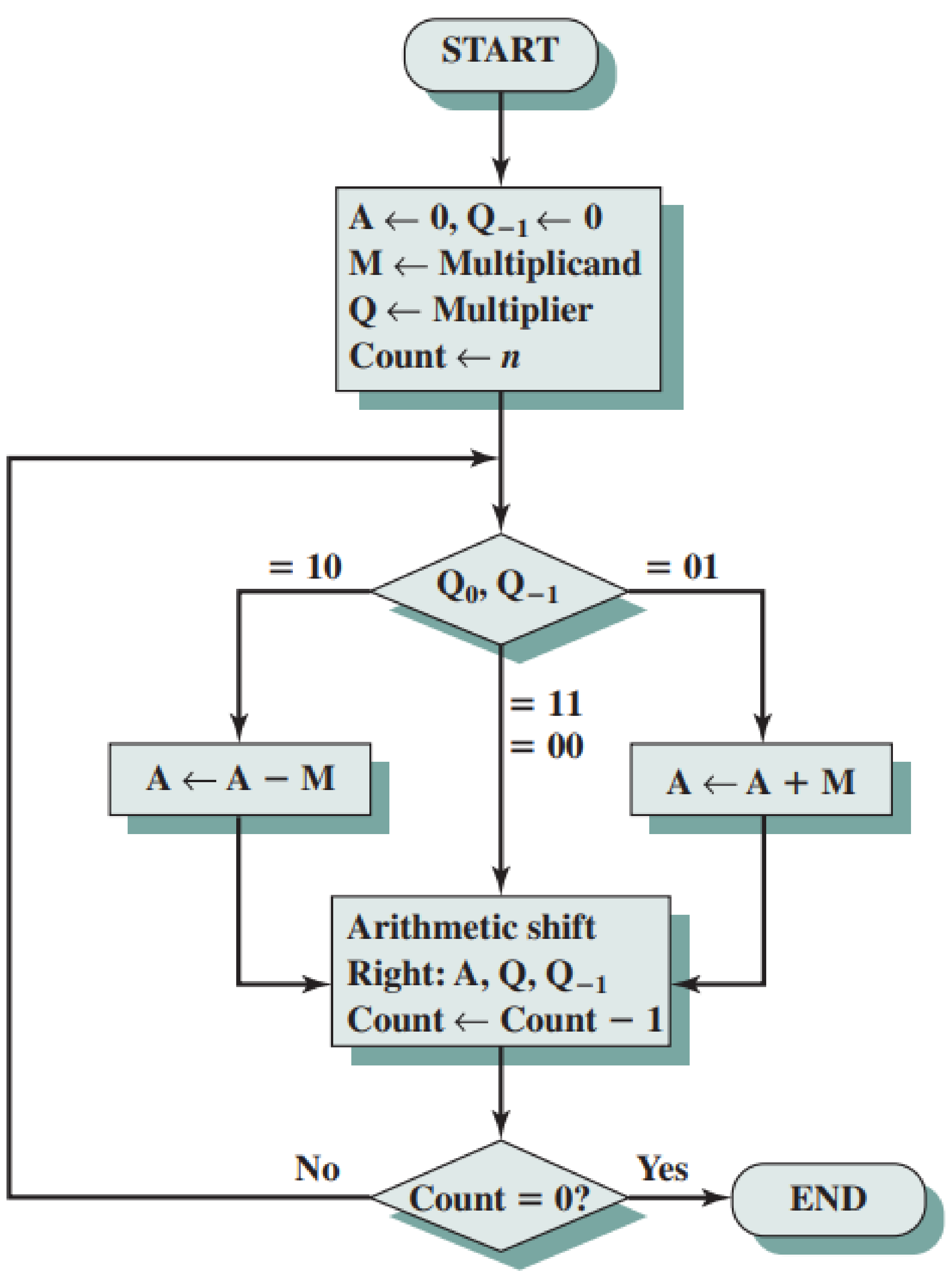
- A and Q_{-1} are initialized to 0. As before, control logic scans the bits of the multiplier one at a time. Now, as each bit is examined, the bit to its right is also examined.

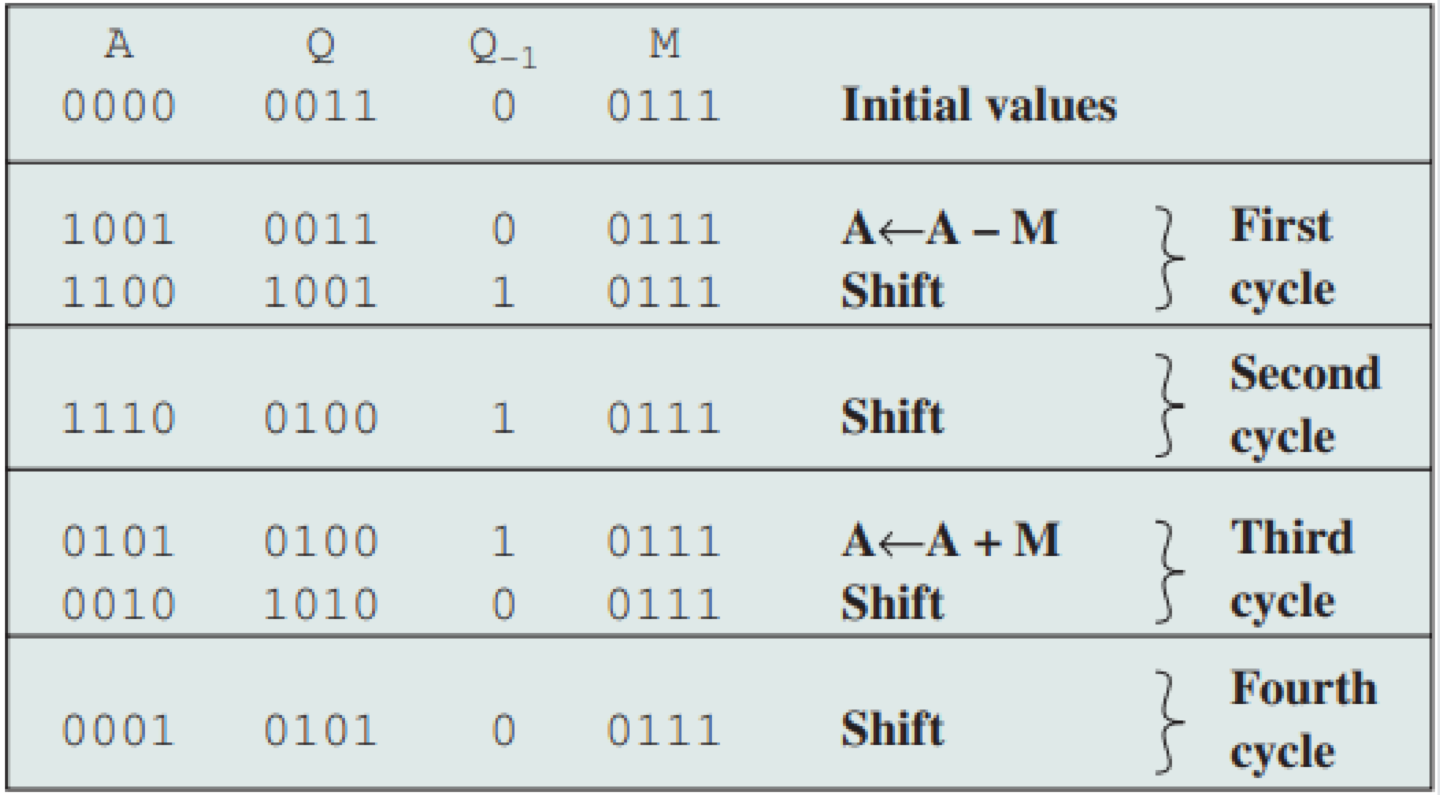
- If the two bits are the same (1–1 or 0–0), then all of the bits of the A, Q, and Q_{-1} registers are shifted to the right 1 bit.

- If the two bits differ, then the multiplicand is added to or subtracted from the A register, depending on whether the two bits are 0–1 or 1–0.

- Following the addition or subtraction, the right shift occurs. In either case, the right shift is such that the leftmost bit of A, namely A_{n-1}, not only is shifted into A_{n-2}, but also remains in A_{n-1}.

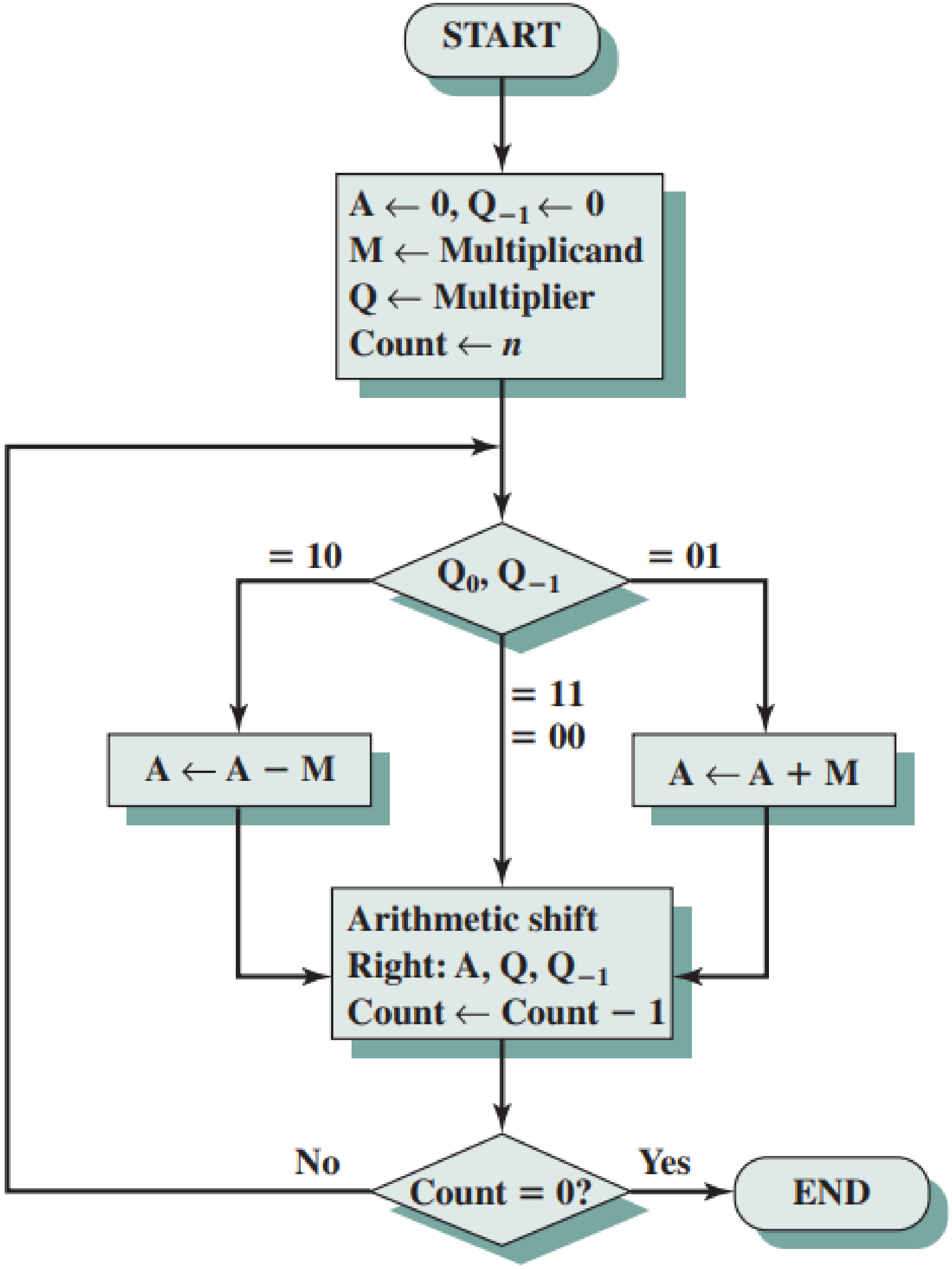
- This is required to preserve the sign of the number in A and Q. It is known as an arithmetic shift.





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**

Example of Booth's Algorithm (7 × 3)



Thank You