CSE-323
Computer Architecture

Computer Arithmetic

Lecturer Afia Anjum
Military Institute of Science and technology

Arithmetic & Logic Unit (ALU)

- The ALU is that part of the computer that actually performs arithmetic and logical operations on data.
- Arithmetic operations include addition, subtraction, multiplication, division and more like this. On the other hand, logical operations involve Boolean logic : AND, OR, NOT and XOR
- All of the other elements of the computer system—control unit, registers, memory, I/O—are there mainly to bring data into the ALU for it to process and then to take the results back out.
- We have, in a sense, reached the core or essence of a computer when we consider the ALU.
- An ALU is based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations.

ALU Inputs & Outputs

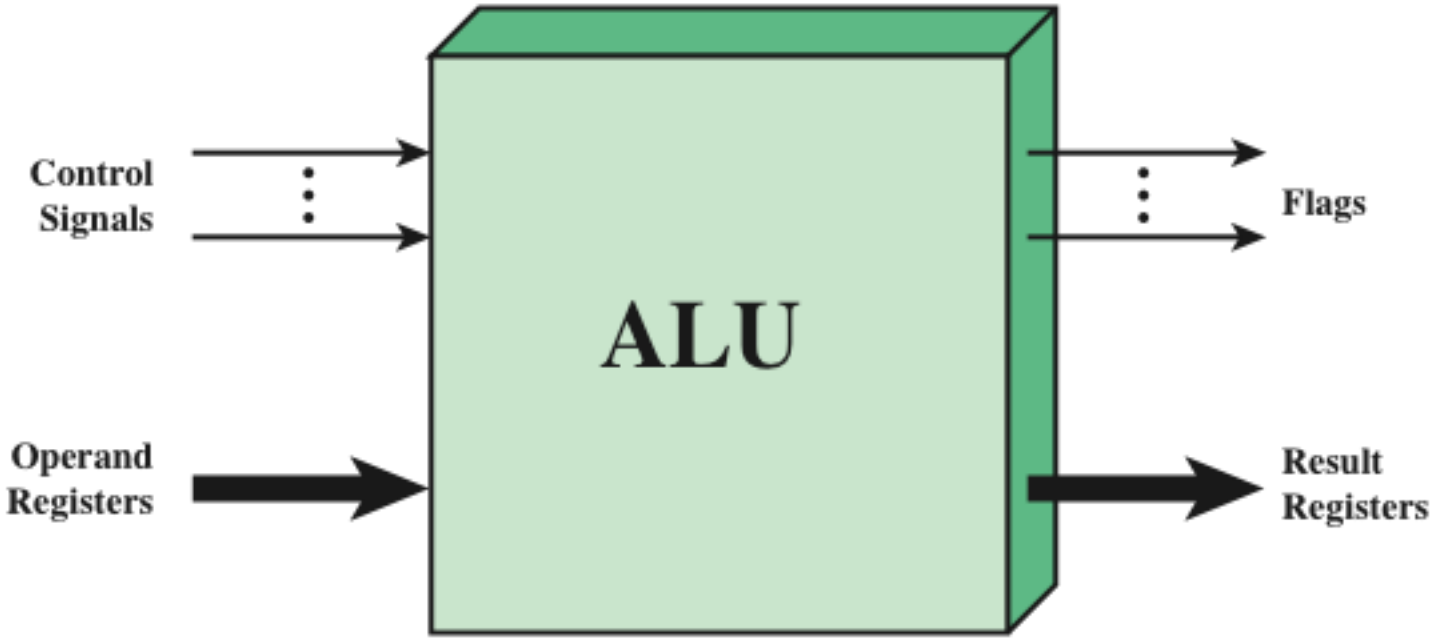


Figure 10.1 ALU Inputs and Outputs

ALU Inputs & Outputs

- **Control Signals:** The processor provides signals that control the operation of the ALU and the movement of the data into and out of the ALU.
- **Operand Registers and Result Registers:** Data (operands) are presented to the ALU in registers, and the results of an operation are stored in registers. These registers are temporary storage locations within the processor
- **Flags**: The ALU may also set flags as the result of an operation. For example, an overflow flag is set to 1 if the result of a computation exceeds the length of the register into which it is to be stored. The flag values are also stored in registers within the processor.

Representation of Signed numbers

- Signed Magnitude Representation
- Complement Representation
  - 1’s Complement
  - 2’s Complement

Signed Magnitude Representation

Rules:

- The sign bit for (+) is 0
- The sign bit for (-) is 1
- Format for a n bit representation:

1 bit	n-1 bit
Sign Bit	Actual Binary Number

Signed Magnitude Representation

Example:

- Represent +7 in Signed Magnitude Representation (in a 4 bit system)

  Decimal: +7

  Binary of 7: 111

  Signed Magnitude Representation of +7: 0111

- Represent -7 in Signed Magnitude Representation (in a 4 bit system)

  Decimal: -7

  Binary of 7: 111

  Signed Magnitude Representation of -7: 1111

Signed Magnitude Representation

Disadvantages:

- One of the bit patterns is wasted: How many possible bit patterns can be created with 4 bits? 16. In unsigned representation, we were able to represent 16 numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, and 15. But with signed magnitude, we are only able to represent 15 numbers: -7, -6, -5, -4, -3, -2, -1, 0, +1, +2, +3, +4, +5, +6, and +7. One of the 16 bit patterns is being misused. That bit pattern is 1000. When we interpret this pattern, we get '-0' which is not logical because we already have 0000 to represent 0.
- Addition doesn't work the way we want it to: Let's try subtracting 2 from 5 by adding 5 and -2.

0101 1010	5 -2
1111	**-7**

So, it doesn't work with regular binary addition algorithms.

Signed Magnitude Representation

Advantages:

- You can determine whether a number is negative or non negative simply by testing the most significant bit.

So, Signed magnitude has more disadvantages than it does advantages.

1’s Complement Representation

Rules for positive number(same as previous):

- The sign bit for (+) is 0
- Format for a n bit representation:

1 bit	n-1 bit
Sign Bit	Actual Binary Number

Rules for negative number:

- The sign bit for (-) is 1
- Format for a n bit representation:

1 bit	n-1 bit
Sign Bit	Bitwise complement of Actual Binary Number

1’s Complement Representation

Example:

- Represent +7 in 1’s Complement Representation (in a 4 bit system)

  Decimal: +7

  Binary of 7: 111

  1’s Complement Representation of +7: 0111

- Represent -7 in 1’s Complement Representation (in a 4 bit system)

  Decimal: -7

  Binary of 7: 111

  Bitwise Complement of Binary of 7: 000

  1’s Complement Representation of -7: 1000

2's Complement Representation

Rules for positive number(same as previous):

- The sign bit for (+) is 0
- Format for a n bit representation:

1 bit	n-1 bit
Sign Bit	Actual Binary Number

Rules for negative number:

- The sign bit for (-) is 1
- Format for a n bit representation:

1 bit	n-1 bit
Sign Bit	Bitwise complement of Actual Binary Number + 1

2’s Complement Representation

Example:

- Represent +7 in 2’s Complement Representation (in a 4 bit system)

  Decimal: +7

  Binary of 7: 111

  2’s Complement Representation of +7: 0111

- Represent -7 in 2’s Complement Representation (in a 4 bit system)

  Decimal: -7

  Binary of 7: 111

  Bitwise Complement of Binary of 7: 000

  Add 1 with 000: 000 + 1 = 001

  2’s Complement Representation of -7: 1001

Advantages of 2’s complement over 1’s complement

- The primary advantage of 2's complement over 1's complement is that 2's complement only has one value for zero. One's complement has a "positive" zero and a "negative" zero.

  For example, in a 8 bit system, in 1's complement there is a -0 (11111111) and a +0 (00000000). On the other hand, in 2's complement, there is only one value for 0 (00000000). This is because +0 → 00000000

  - 0 → 00000000 → 11111111 + 1 → 00000000

- To add numbers using 1's complement you have to first do binary addition, then add the carry value with the result. But in 2's complement you can simply ignore the carry value.

Range Extension

- It is sometimes desirable to take an n-bit integer and store it in m bits, where m > n. This expansion of bit length is referred to as **range extension**, because the range of numbers that can be expressed is extended by increasing the bit length.
- To extend the range, the rules for 2's complement representation is to move the sign bit to the new leftmost position and fill in with copies of the sign bit.
- For example:

  2's complement of +7 is 0111 in 4 bit system. If we want to extend the range and move it to 8 bit system, the value will be: 00000111

  Again, 2's complement of -7 is 1001 in 4 bit system. If we want to extend the range and move it to 8 bit system, the value will be: 11111001

Negation

In twos complement notation, the negation of an integer can be formed with the following 3 steps:

1. First, determine the 2's complement of the given integer.
2. Then, do the Bitwise complement of each bit. (including the sign bit). That is, set each 1 to 0 and each 0 to 1.
3. Now, add 1 to the result.

Example:

- Negation of +7:

  +7 = 0111 (in 2's complement)

  bitwise complement = 1000

  Add 1 : 1000

  1

  1001 = -7 in 2's complement

Negation

Example:

- Negation of -7:

  -7 = 1001 (in 2's complement)

  bitwise complement = 0110

  Add 1 : 0110

  1

  0111 = +7 in 2's complement

Negation Special Case

Example:

- Negation of +0:

  +0 = 00000000 (in 2's complement)

  bitwise complement = 11111111

  Add 1 : 11111111

  1

  100000000

- In 2's complement addition, we ignore the carry. So the final result is 00000000, which is 0.
- So, here +0 = -0 = 0

Rules of Addition

- **<u>Case 1:</u> When the positive number has a greater magnitude:**
  - In this case the carry which will be generated is discarded and the final result is the result of addition.
  - For example: we want to add +7 and -3

  +7 = 0111 (in 2's complement)

  -3 = 1101 (in 2's complement)

  Addition : 0111

  1101

  10100

  If we ignore the carry bit, the result is 0100, which is +4 in 2's complement.

Rules of Addition

- **<u>Case 2:</u> When the negative number has a greater magnitude:**
  - In this case no carry will be generated and the result of addition will be negative
  - For example: we want to add -7 and +5

-7 = 1001 (in 2's complement)

+5 = 0101 (in 2's complement)

Addition : 1001

0101

1110 = -2 in 2's complement.

Rules of Addition

- **<u>Case 3:</u> When in addition, both numbers are positive or both numbers are negative.**
  - In this case carry may or may not be generated. If generated, carry should be ignored.
  - Overflow occurs if and only if the result has the opposite sign.
  - When overflow occurs, the ALU must signal this fact so that no attempt is made to use the result.
  - Note that overflow can occur whether or not there is a carry.

Rules of Addition

- **<u>Case 3:</u> When in addition, both numbers are positive or both numbers are negative.**
  - For example: we want to add -4 and -1

-4 = 1100 (in 2's complement)

-1 = 1111 (in 2's complement)

Addition : 1100

1111

11011

If we ignore the carry bit, the result is 1011, which is -5 in 2's complement.

Notice that, the result also has the same sign as the operands. So no overflow here.

Rules of Addition

- **<u>Case 3:</u> When in addition, both numbers are positive or both numbers are negative.**
  - For example: we want to add +3 and +4

+3 = 0011 (in 2's complement)

+4 = 0100 (in 2's complement)

Addition : 0011

0100

0111 = +7 in 2's complement

Notice that, the result also has the same sign as the operands. So no overflow here.

Rules of Addition

- **<u>Case 3:</u> When in addition, both numbers are positive or both numbers are negative.**
  - For example: we want to add -7 and -6

-7 = 1001 (in 2's complement)

-6 = 1010 (in 2's complement)

Addition : 1001

1010

10011

If we ignore the carry bit, the result is 0011, which is +3 in 2's complement. WRONG ANSWER!

Also notice that, the result has the opposite sign as the operands.

So Overflow occurred here, for which reason the generated result is wrong!

Rules of Addition

- **<u>Case 3:</u> When in addition, both numbers are positive or both numbers are negative.**
  - For example: we want to add 5 and 4

5 = 0101 (in 2's complement)

4 = 0100 (in 2's complement)

Addition : 0101

0100

1001 = -7 in 2's complement WRONG ANSWER!

Also notice that, the result has the opposite sign as the operands.

So Overflow occurred here, for which reason the generated result is wrong!

How to resolve the Overflow??

There are two ways:

- Overflow flag = 1, stating that the answer is wrong
- Store the carry bit

Rules of Subtraction

- To subtract one number (subtrahend) from another (minuend),
  1. Derive 2's complement for both the numbers
  2. Do **negation** of the 2's complement of the subtrahend and add it to the minuend following the **rules of addition**.
  - **For example:** we want to subtract 5 from 7.

    Here, subtrahend = 5, minuend = 7

    5 = 0101 (in 2's complement)

    7 = 0111 (in 2's complement)

    Negation of 5 = -5 = 1011

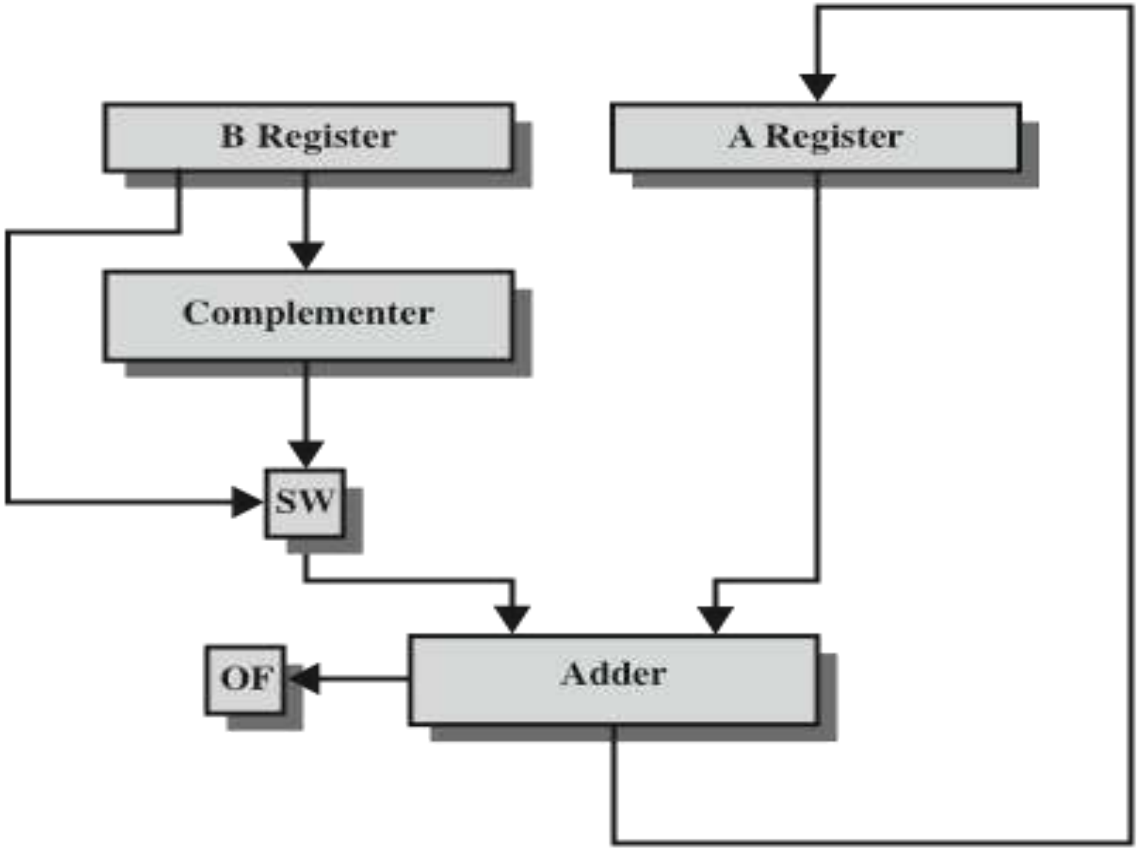
    Addition : 0111

    1011

    10010

    If we ignore the carry bit, the result is 0010, which is +2 in 2's complement.

Hardware of Addition and Subtraction



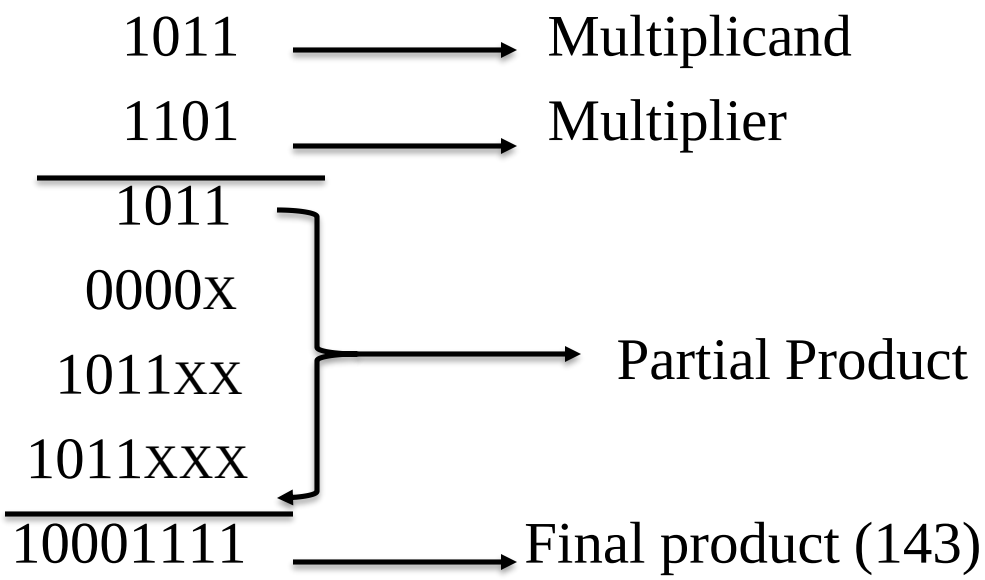
OF = overflow bit
SW = Switch (select addition or subtraction)

Figure 10.6 Block Diagram of Hardware for Addition and Subtraction

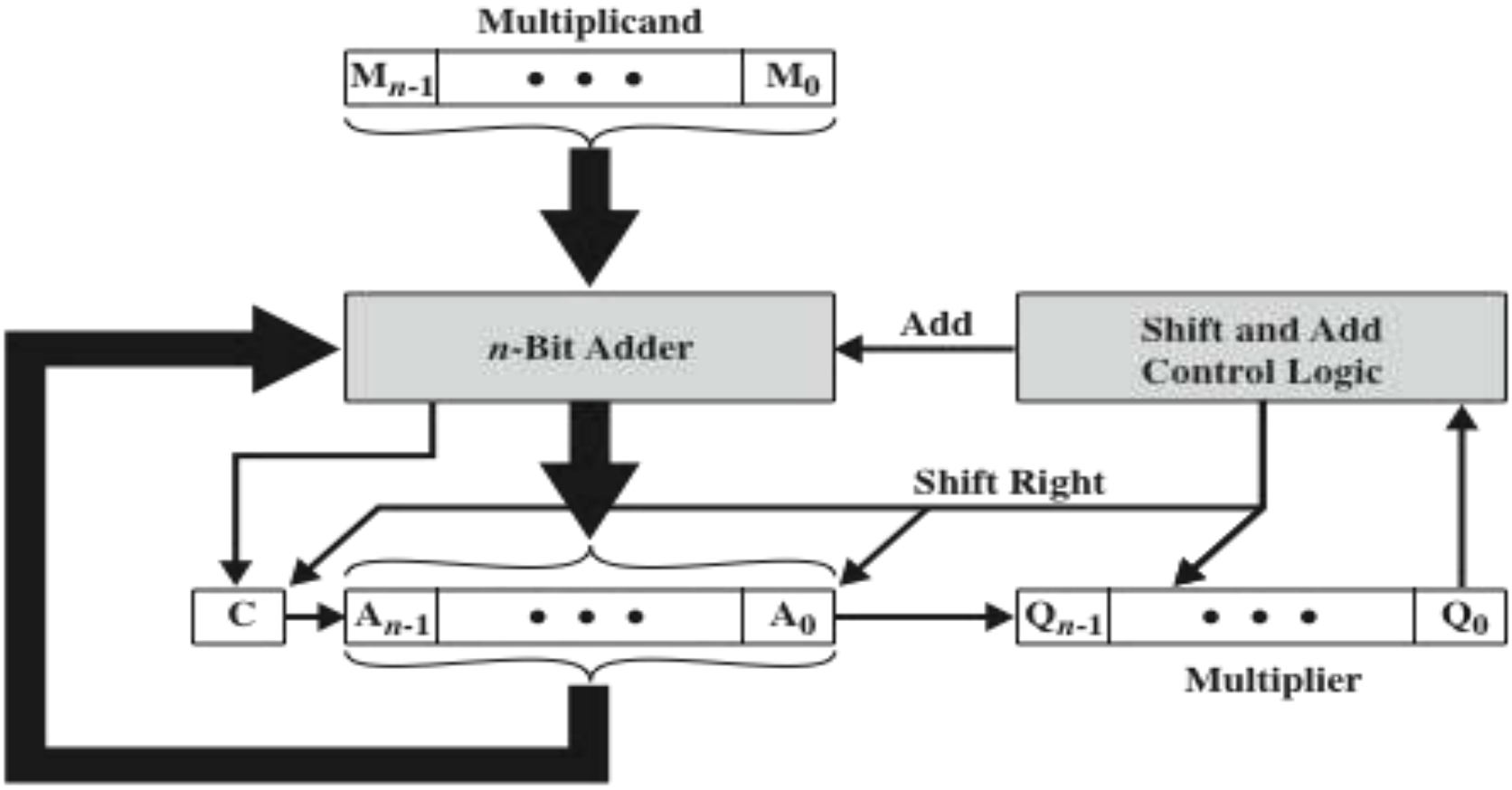
General Rules of Multiplication (unsigned)

- If we multiply a number with 0, the answer is 0
- If we multiply a number with 1, the answer is 1
- We need to proceed as per the rules of normal integer multiplication

  - **For example:** we want to multiply 11 (1011) and 13 (1101)

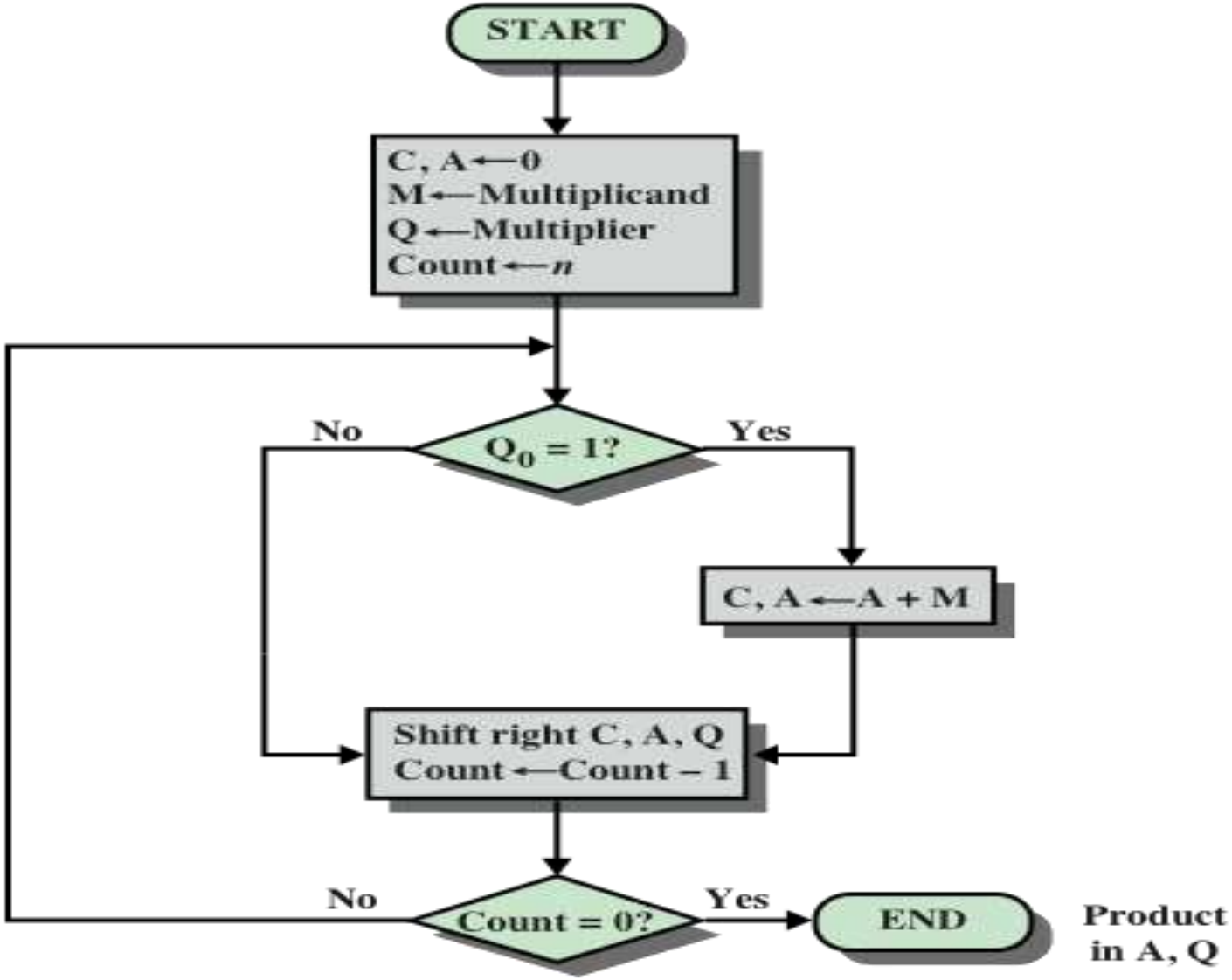


Block Diagram of Multiplication in ALU (unsigned)

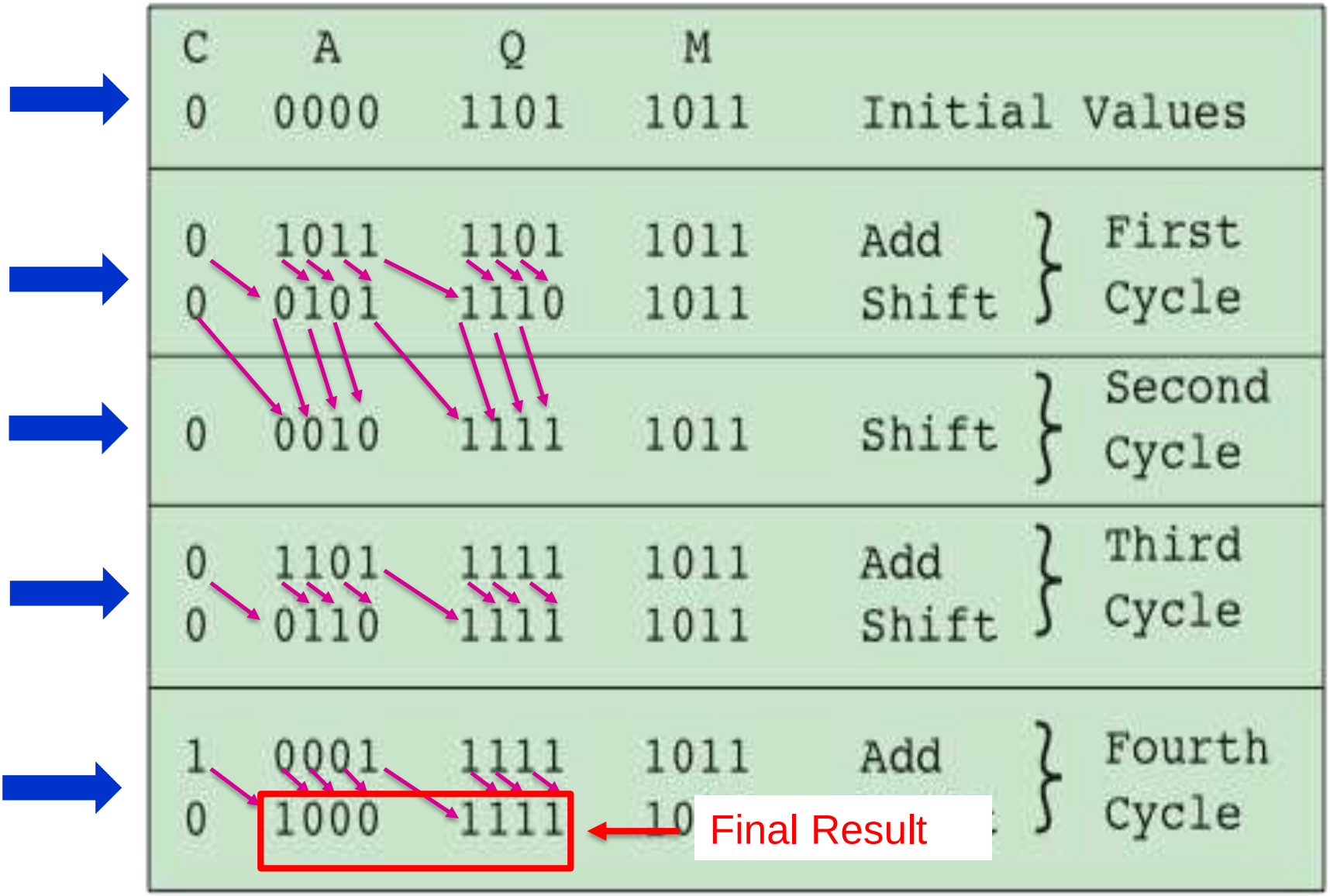


(a) Block Diagram

Flow Chart of Multiplication in ALU (unsigned)



Simulation of Multiplication in ALU (unsigned)



Block Diagram for Multiplication in ALU (Signed)

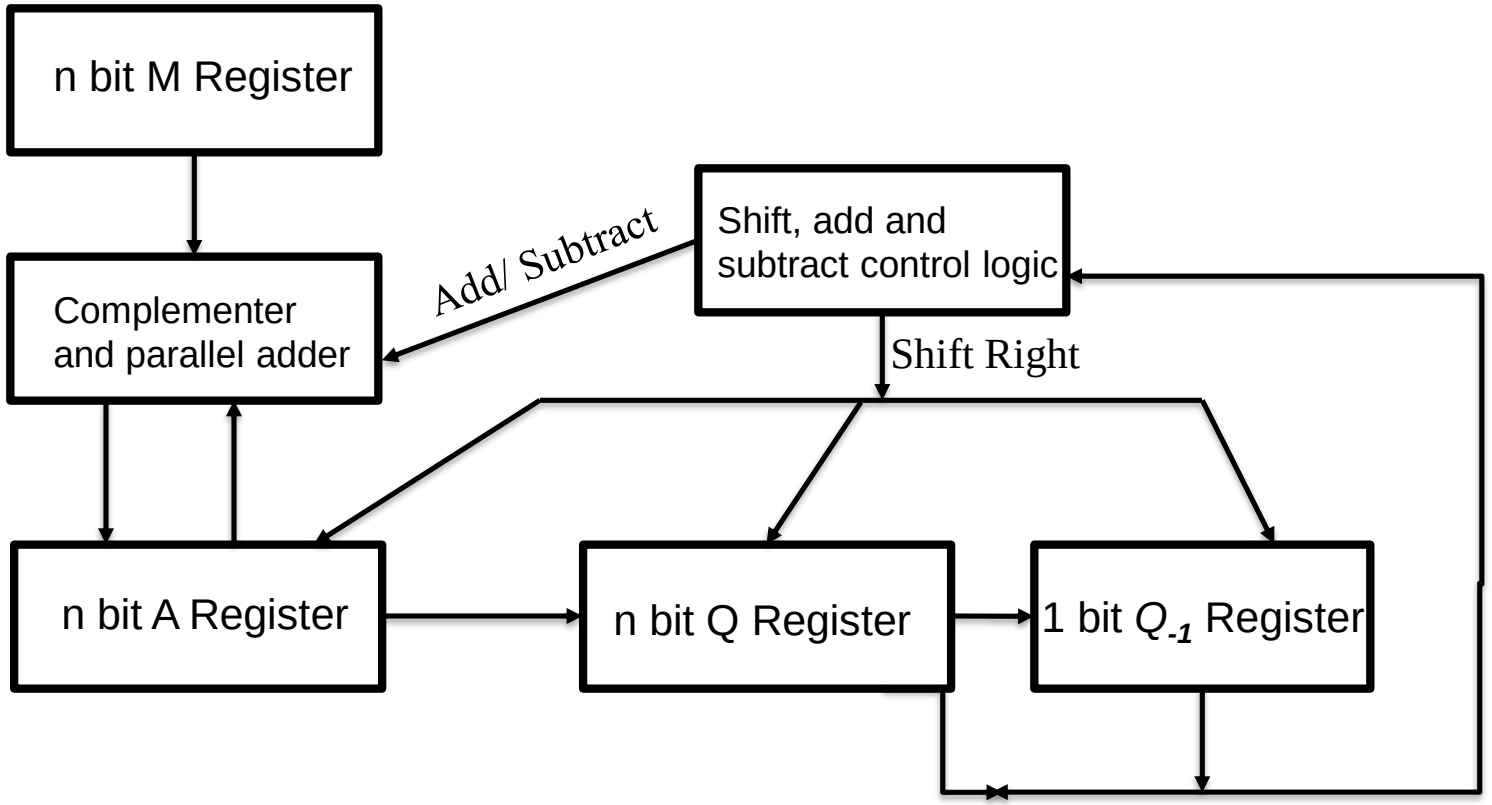


Fig: Block Diagram of Booth's Algorithm for 2's Complement Multiplication

Flow Chart of Multiplication in ALU (signed)

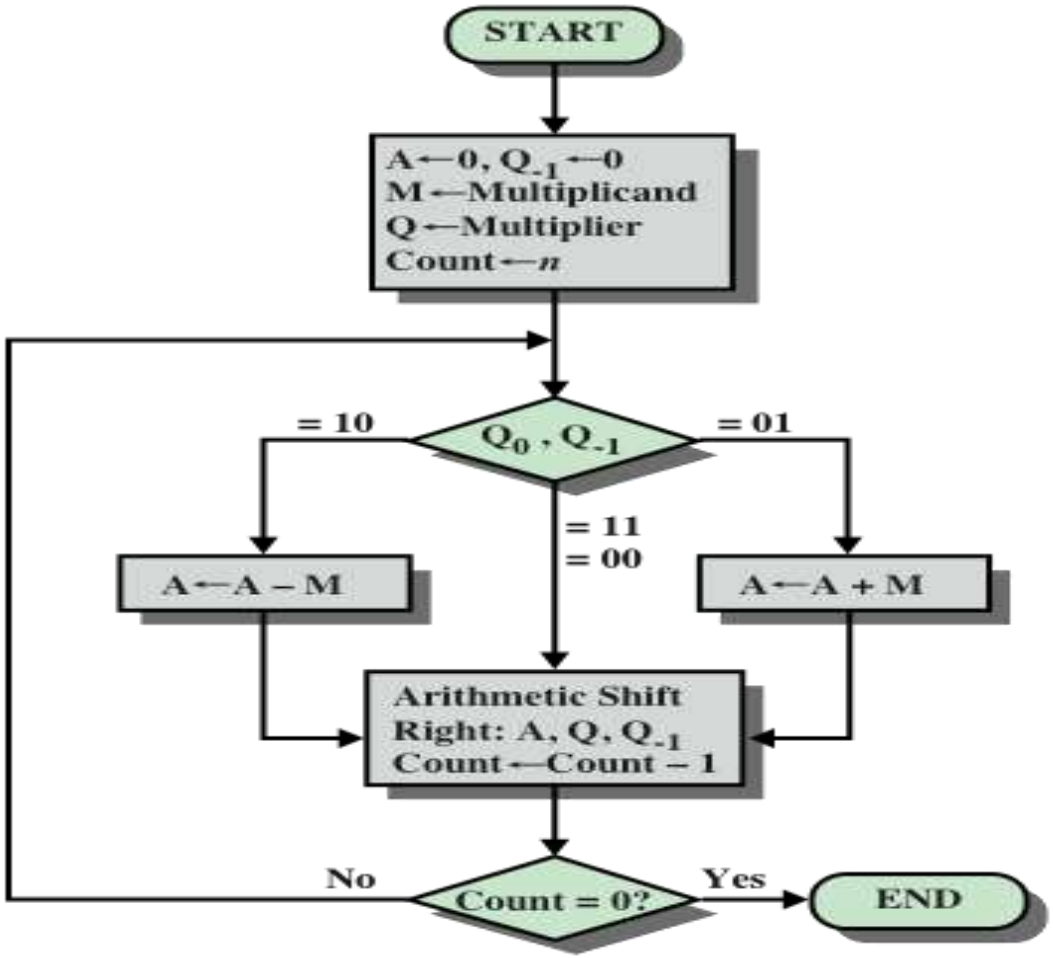
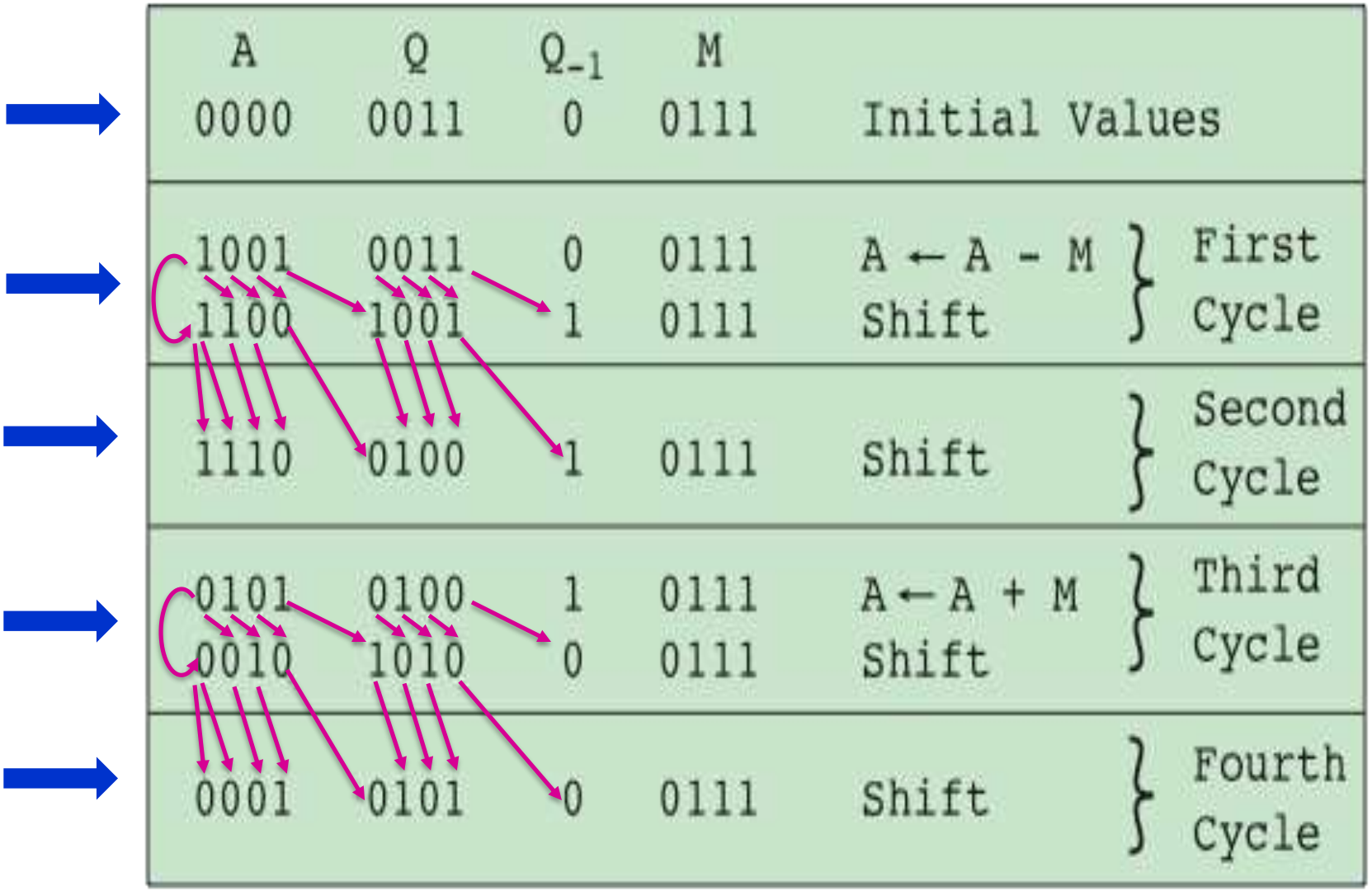


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

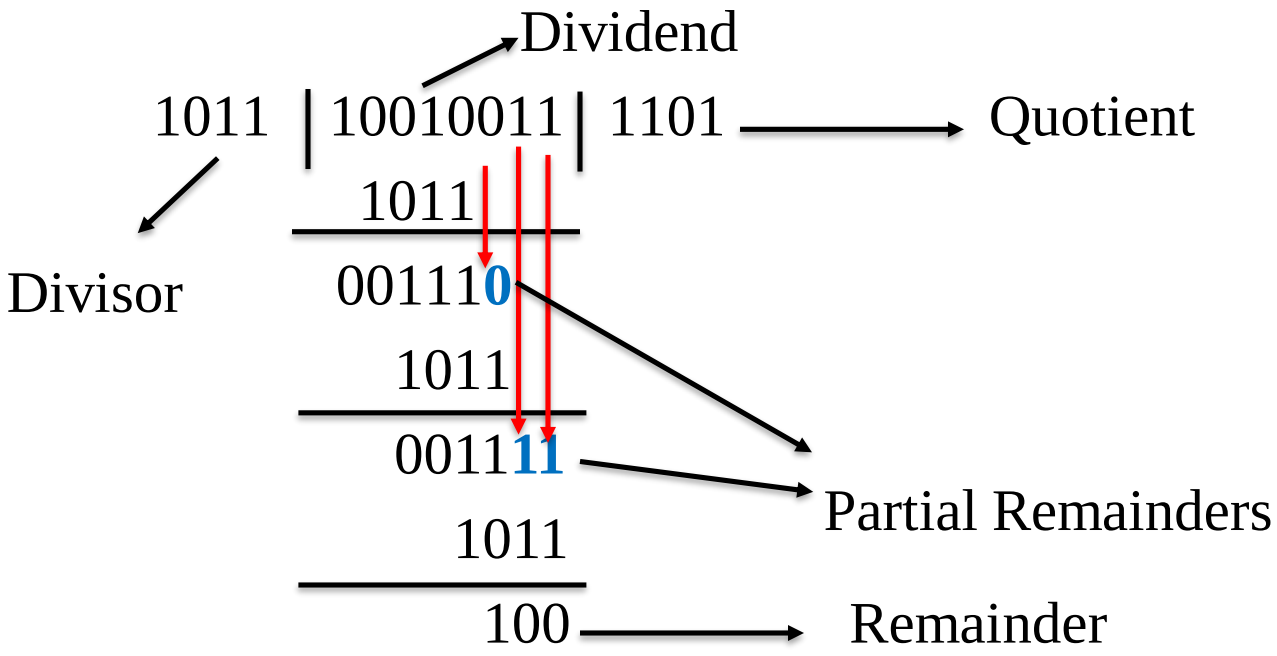
Simulation of Multiplication (7 x 3) in ALU (signed)



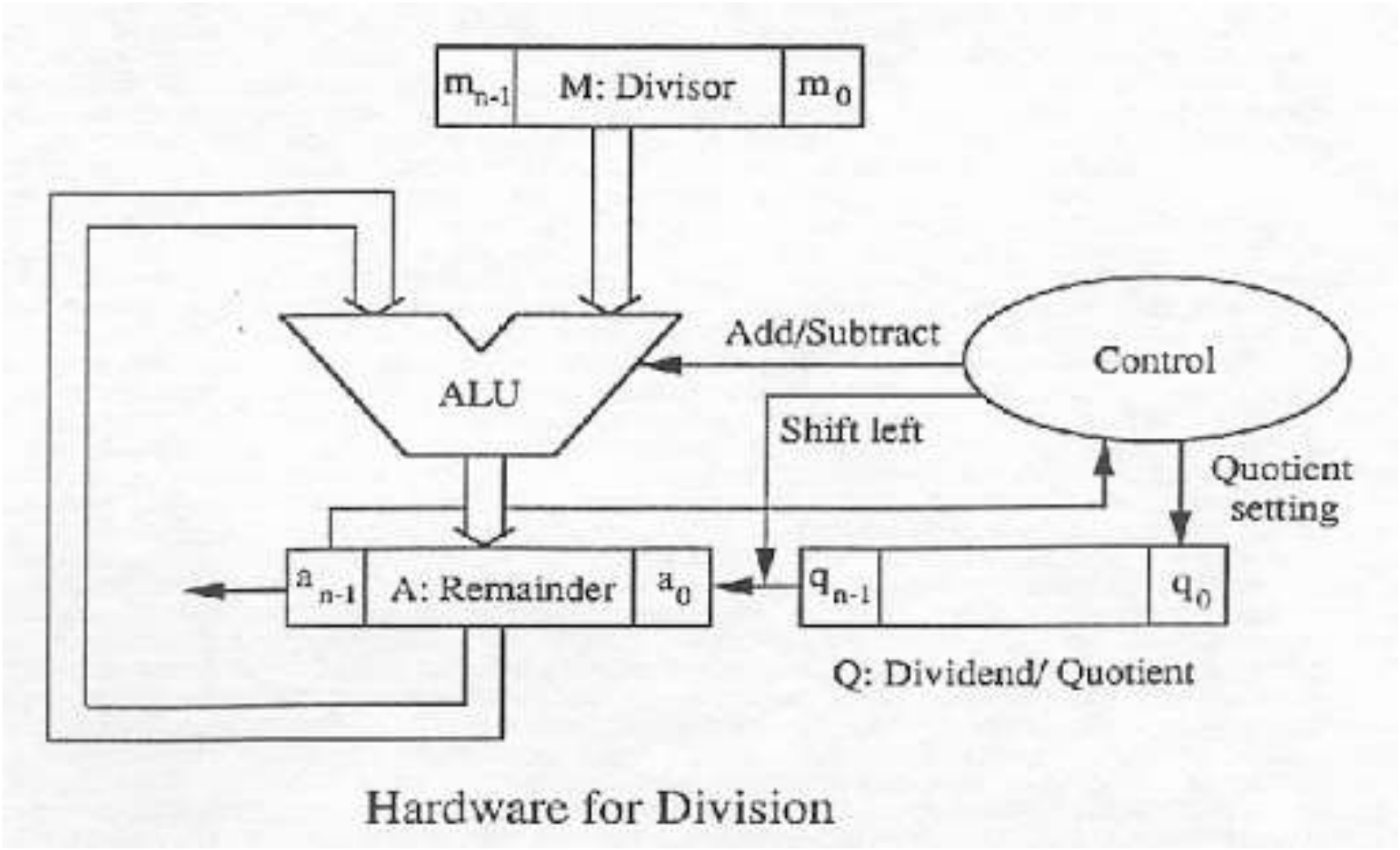
A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	A ← A - M	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	A ← A + M	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

General Rules of Division (unsigned)

- We need to know some rules of subtraction
  - 0-0 = 0; 0-1 = 1, Carry 1; 1-0 = 1; 1-1 = 0
  - **For example:** we want to divide 147 (10010011) by 11 (1011)

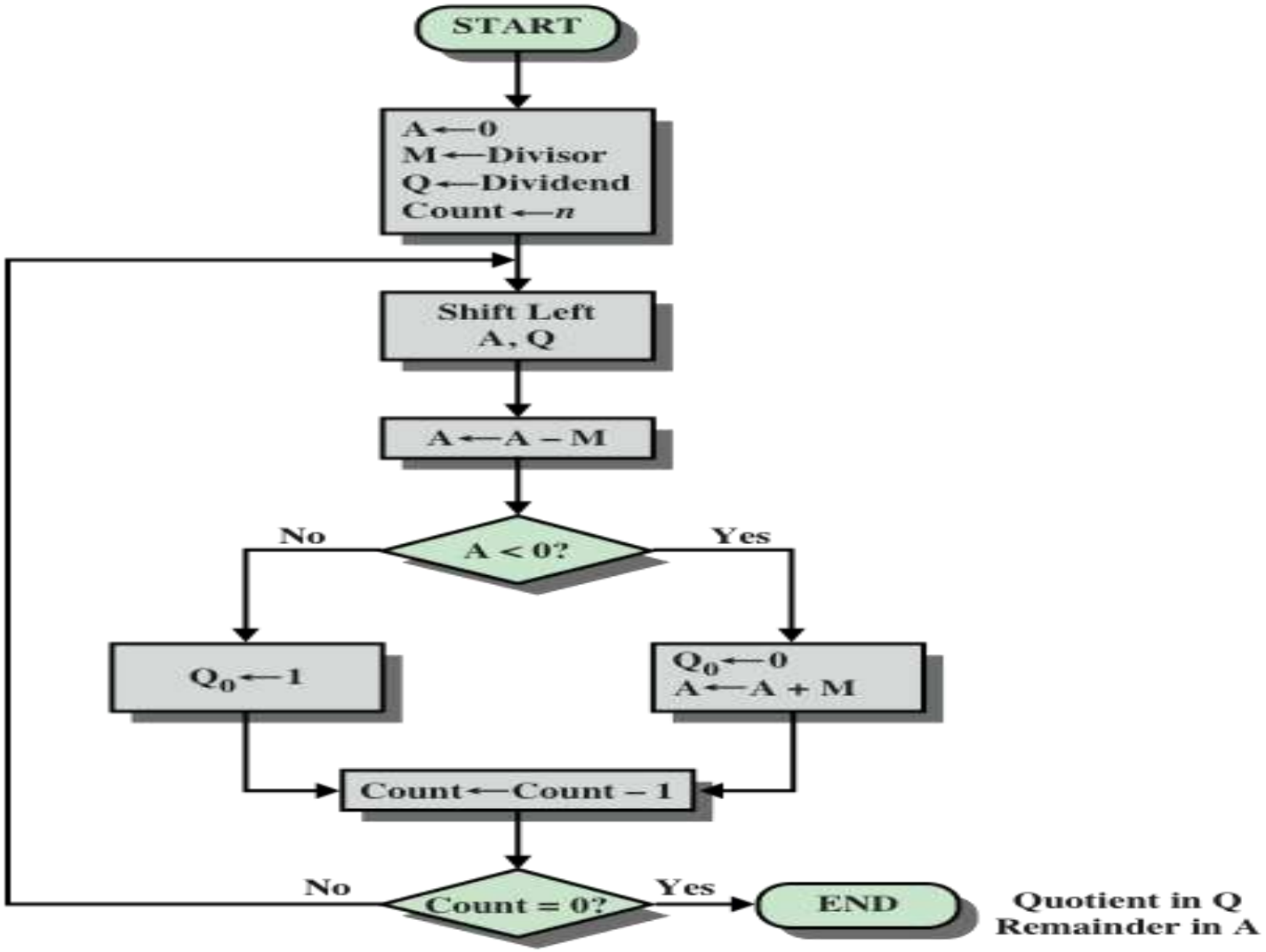


Block Diagram for Division in ALU (Signed)



Hardware for Division

Flow Chart of Division in ALU (signed)



Simulation of Division (6/2) in ALU (signed)

- 2’s complement of 6 = 0110
- 2’s complement of 2 = 0010
- Here, 6 is the Dividend So we’ll keep it in Q
- Here, 2 is the Divisor So we’ll keep it in M
- The Final Remainder is in A and Quotient in Q

Simulation of Division (6/2) in ALU (signed)

A	Q	M		
0000	0110	0010	Initial Values	
0000	110__	0010	Left Shift A,Q	1st Cycle
1110	110__	0010	A = A – M	
0000	1100	0010	A<0 so, Qo = 0, A= A+M	
0001	100__	0010	Left Shift A,Q	2nd Cycle
1111	100__	0010	A = A – M	
0001	1000	0010	A<0 so, Qo = 0, A= A+M	
0011	000__	0010	Left Shift A,Q	3rd Cycle
0001	000__	0010	A = A – M	
0001	0001	0010	A>0 so, Qo = 1	
0010	001__	0010	Left Shift A,Q	4th Cycle
0000	001__	0010	A = A – M	
0000	0011	0010	A>0 so, Qo = 1	


THANK YOU