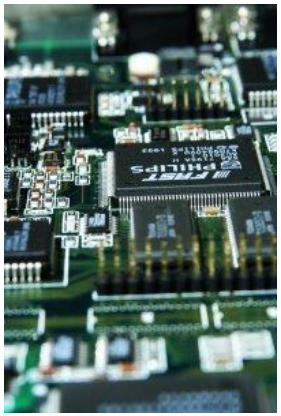
CSE 321b

Computer Organization (2)

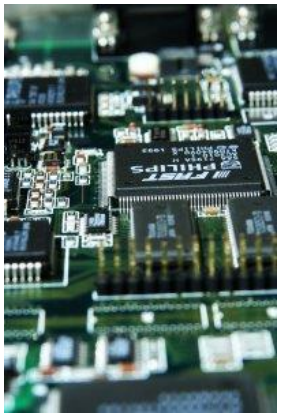
تنظيم الحاسب (2)



3rd year, Computer Engineering

Winter 2016

Lecture #9



Dr. Hazem Ibrahim Shehata

Dept. of Computer & Systems Engineering

Credits to Dr. Ahmed Abdul-Monem Ahmed for the slides

Adminstrivia

- Assignment #2:
  - —Due: **Today**
- Midterm:
  - —Date: Thursday, April 21, 2014
  - —Time: 10:30am – 12:00pm
  - —Location: classroom #27309
  - —Coverage: lectures #1 ➔ #7

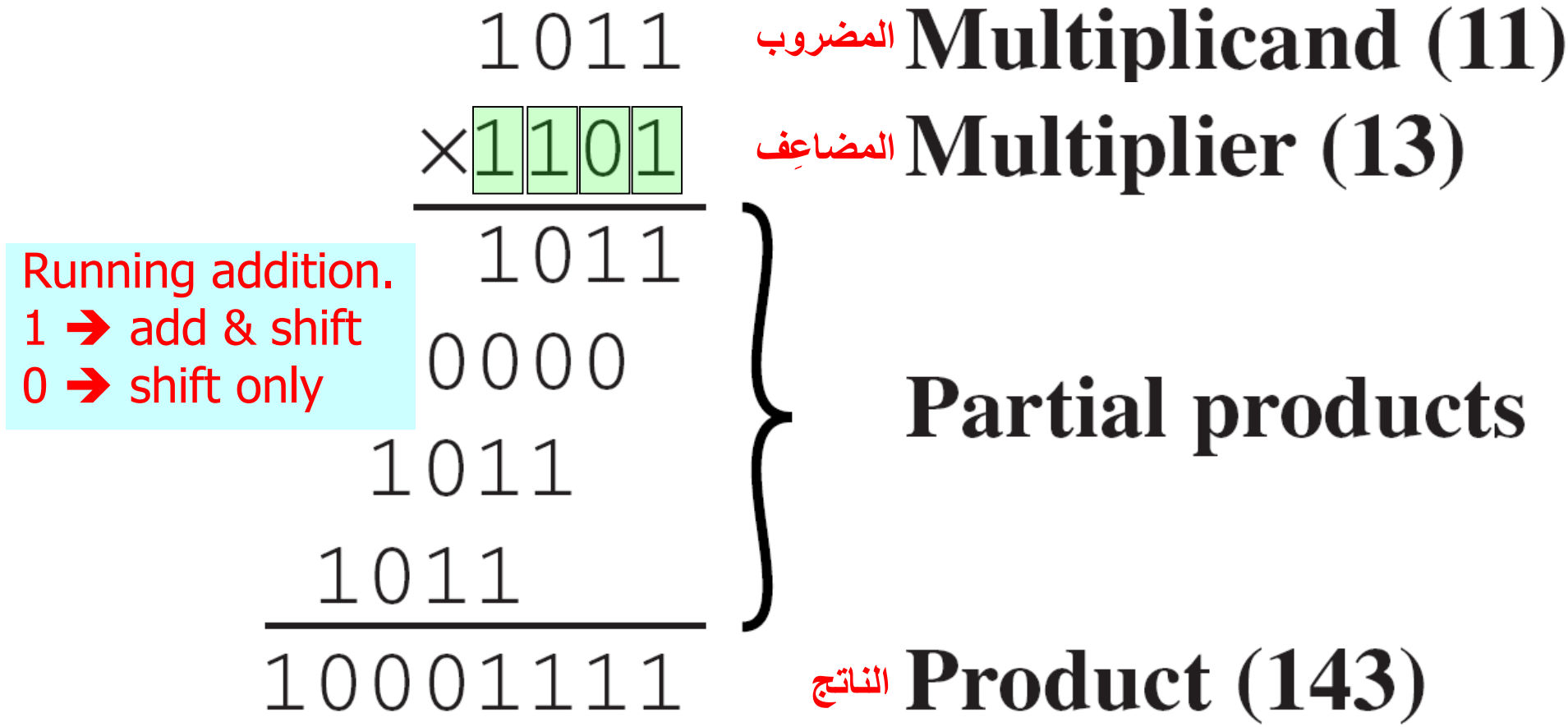
Website: http://hshehata.github.io/courses/zu/cse321b/
Office hours: Sunday 11:30am – 12:30pm

Chapter 10. Computer Arithmetic (*Cont.*)

Outline

- Integer Representation
  - —Sign-Magnitude, Two's Complement, Biased
- Integer Arithmetic
  - —Negation, Addition, Subtraction
  - —Multiplication, Division
- Floating-Point Representation
  - —IEEE 754
- Floating-Point Arithmetic
  - —Addition, Subtraction
  - —Multiplication, Division
  - —Rounding

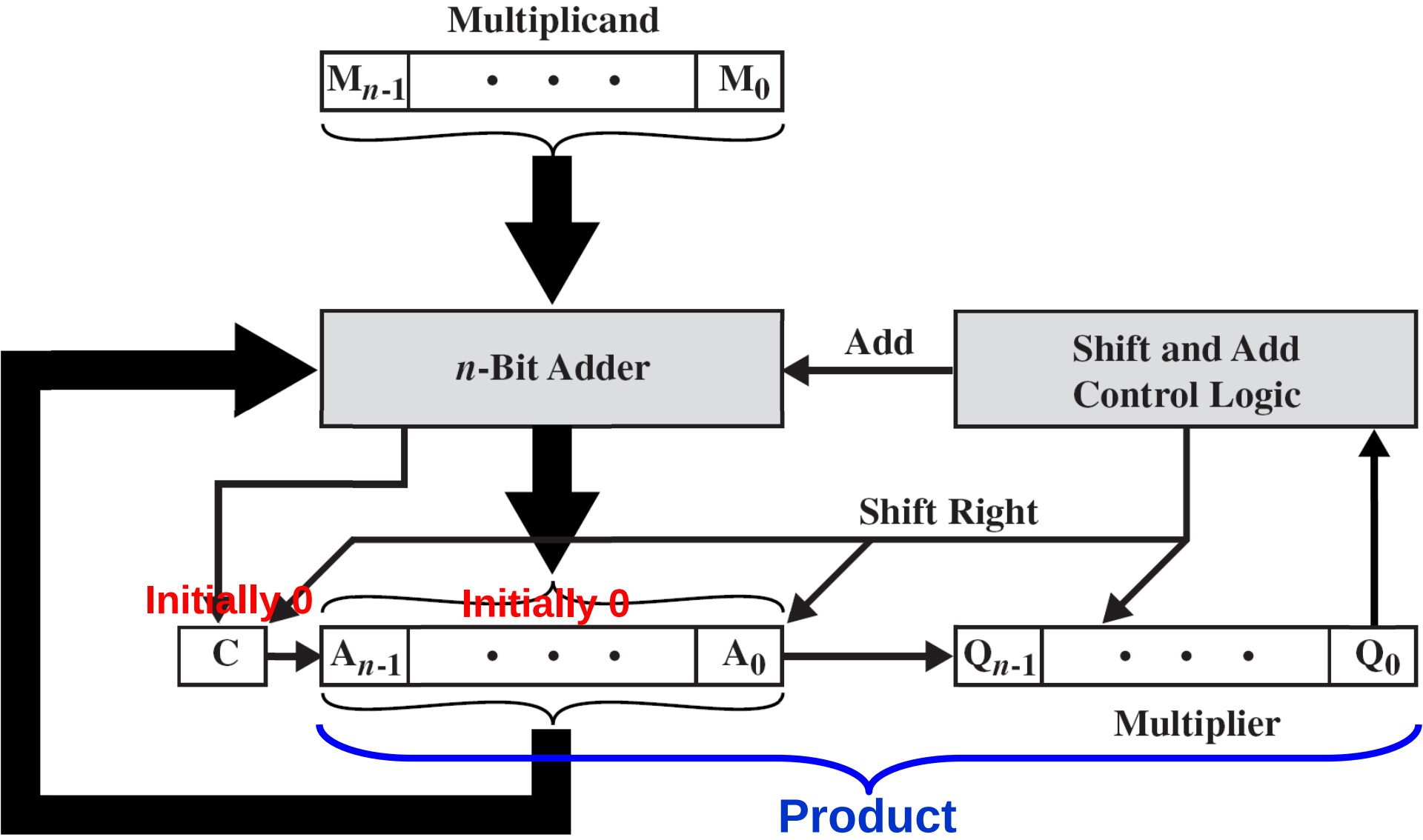
Multiplication Example

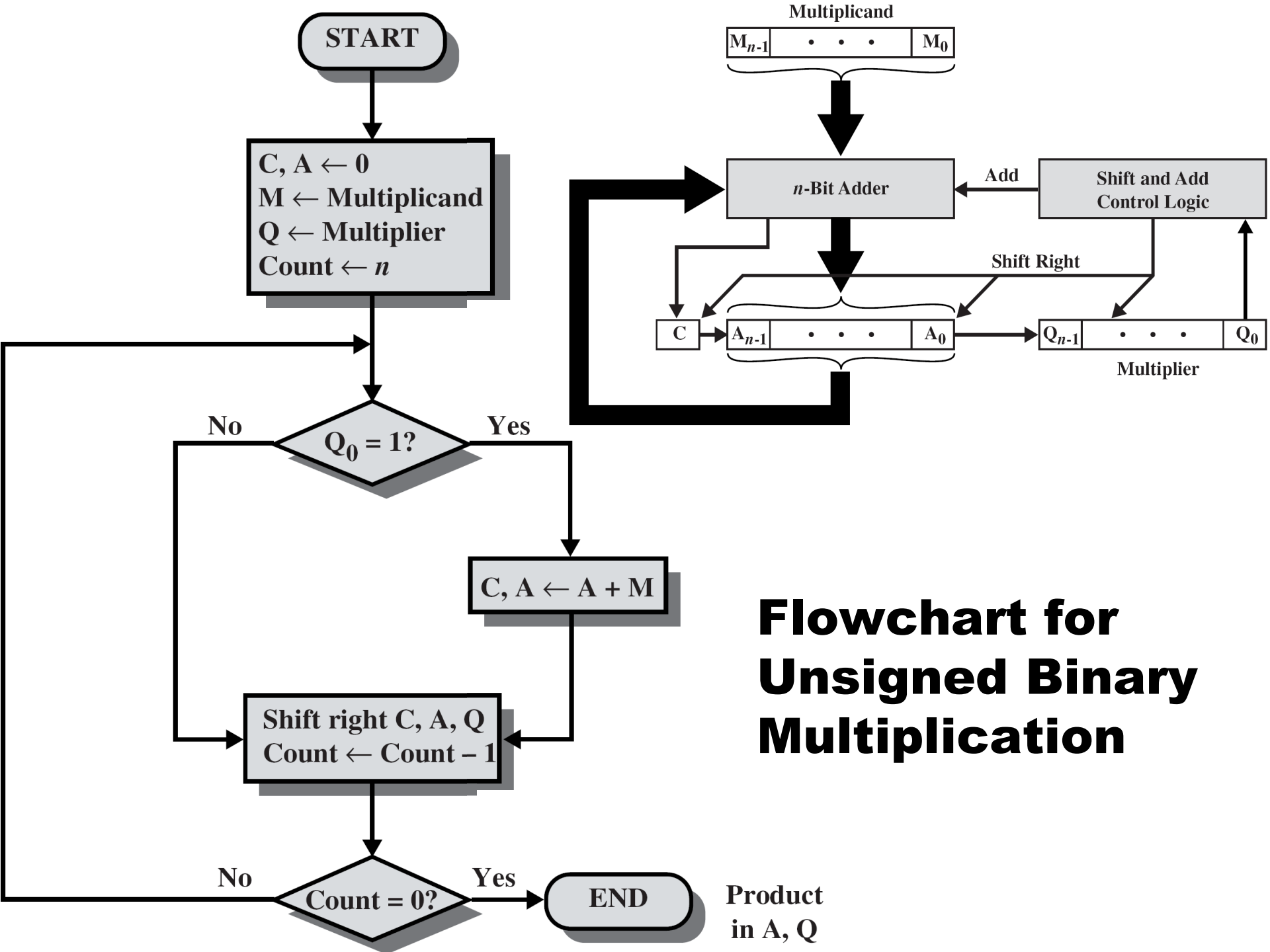


Running addition.
1 ➔ add & shift
0 ➔ shift only

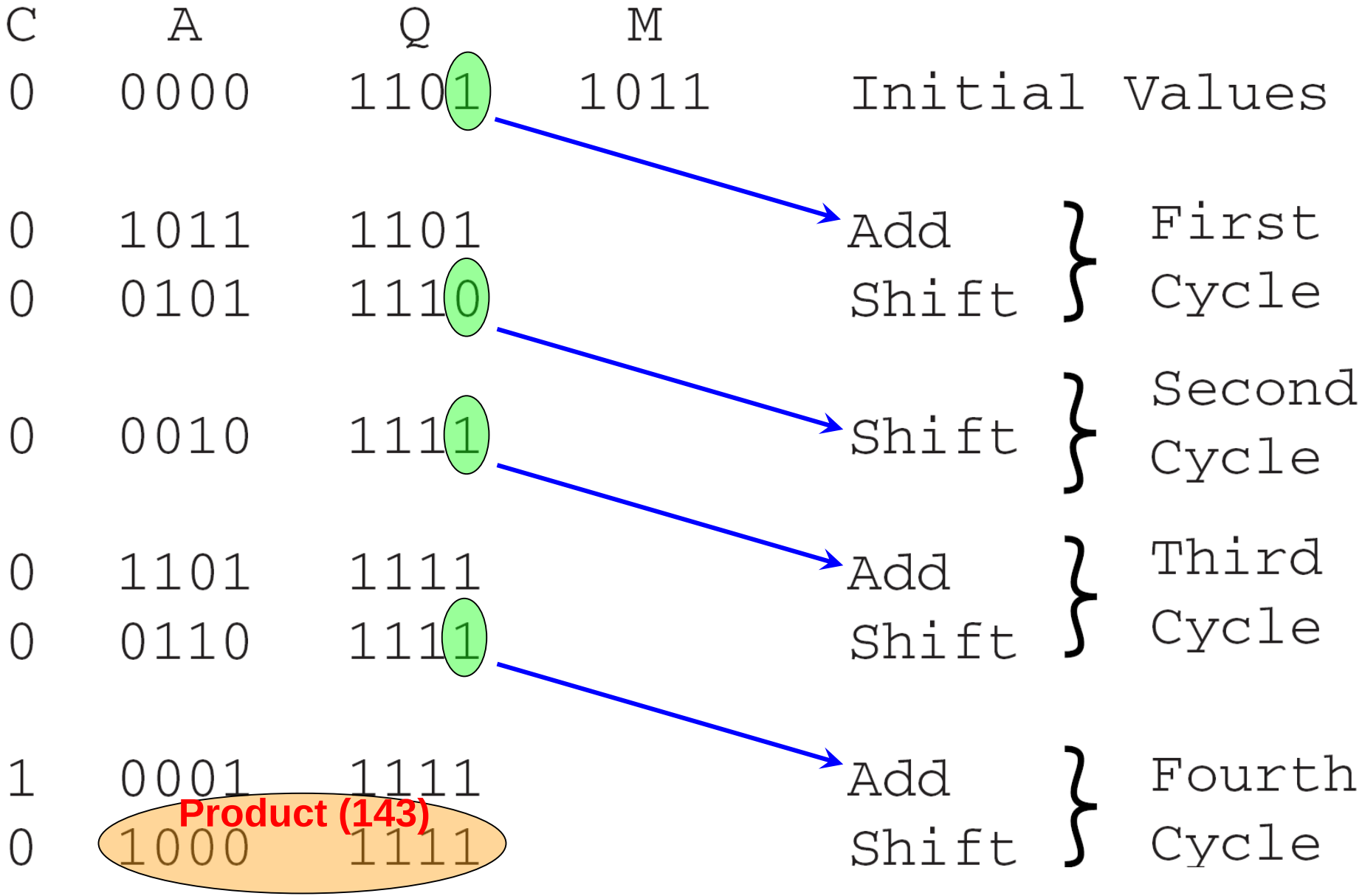
- Complex (relative to addition)!!
  - Work out a partial product for each digit.
  - Shift the partial product appropriately.
  - Add partial products.
  - Generate double-length result.

Unsigned Binary Multiplication




START
C, A ← 0
M ← Multiplicand
Q ← Multiplier
Count ← n
No
Q0 = 1?
Yes
C, A ← A + M
Shift right C, A, Q
Count ← Count – 1
No
Count = 0?
Yes
END
Product in A, Q
Multiplicand
Mn-1 • • • M0
n-Bit Adder
Add
Shift and Add Control Logic
Shift Right
C
An-1 • • • A0
Qn-1 • • • Q0
Multiplier
Flowchart for Unsigned Binary Multiplication

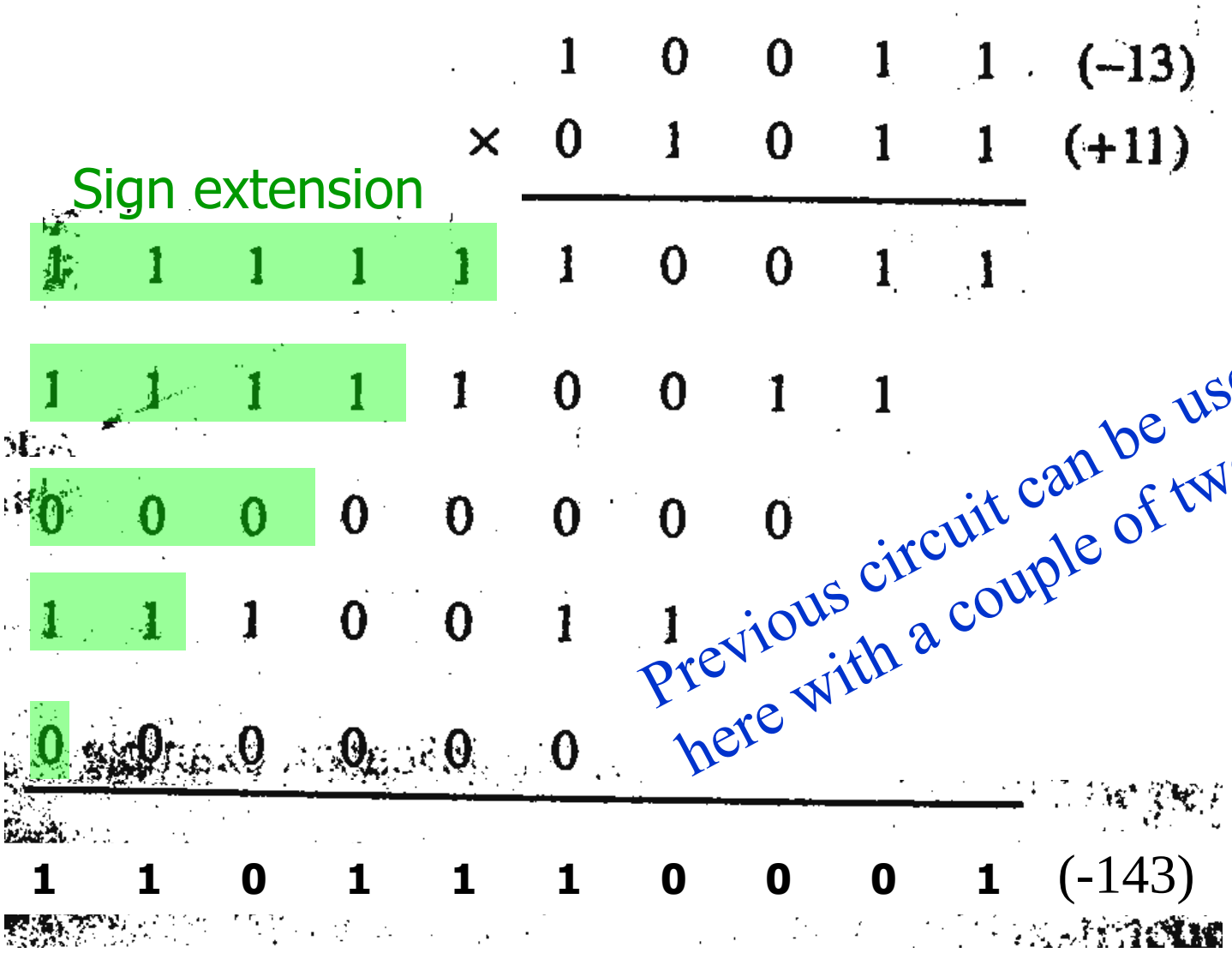
Execution of Example



Signed Binary Multiplication

- The straight forward multiplication algorithm doesn't work with signed numbers!!
- Evidence: In the previous example, suppose that M & Q are interpreted as signed numbers:
  - M = $(1011)_2$ which represents $(-5)_{10}$
  - Q = $(1101)_2$ which represents $(-3)_{10}$
  - Applying the algorithm results in a product value of $(1000\ 1111)_2$ which represents $(-113)_{10}$
  - This result is wrong! Correct value is supposed to be $(+15)_{10}$!!!!

Signed Multiplication Example: +ve Multiplier, –ve Multiplicand



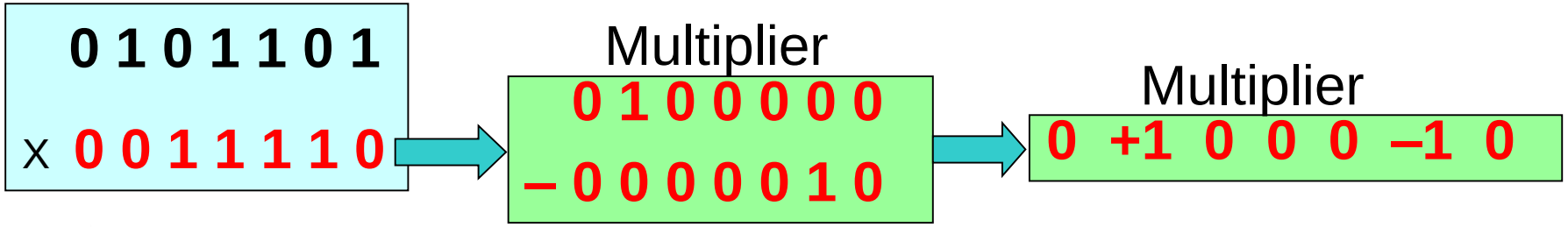
Signed Multiplication Algorithm #1

1. If **multiplier → +ve** & **multiplicand → +ve**:
   - Follow unsigned multiplication algorithm
2. Else if **multiplier → +ve** & **multiplicand → -ve**:
   - Follow unsigned mult. algorithm with 2 changes:
     a. Set the carry register (C) to 1 after first addition.
     b. Use "arithmetic shift" instead of "logical shift", while considering C to be the sign bit!
3. Else if **multiplier → -ve** & **multiplicand → +ve**:
   - Negate (find 2's compl. of) multiplier & multiplicand.
   - Proceed as case 2.
4. Else **multiplier → -ve** & **multiplicand → -ve**:
   - Negate (find 2's compl. of) multiplier & multiplicand.
   - Proceed as case 1.

Signed Multiplication Algorithm #2

1. Convert multiplicand (M) & multiplier (Q) to their absolute (positive) values |M| & |Q|.
2. Run the unsigned multiplication algorithm on |M| & |Q| to obtain the final product (P).
3. Adjust the sign of P (by 2's complementation where needed) according to the following rule:
   - sign(P) = sign(M) X sign(Q)

Signed Multiplication Algorithm #3 (Booth's Algorithm)




```
                                0  1  0  1  1  0  1
                                0  0 +1 +1 +1 +1  0
                             ------------------------
                                0  0  0  0  0  0  0
                             0  1  0  1  1  0  1
                          0  1  0  1  1  0  1
                       0  1  0  1  1  0  1
                    0  1  0  1  1  0  1
                 0  0  0  0  0  0  0
              0  0  0  0  0  0  0
           ------------------------------------------
           0  0  0  1  0  1  0  1  0  0  0  1  1  0
```

Booth's Algorithm – Example

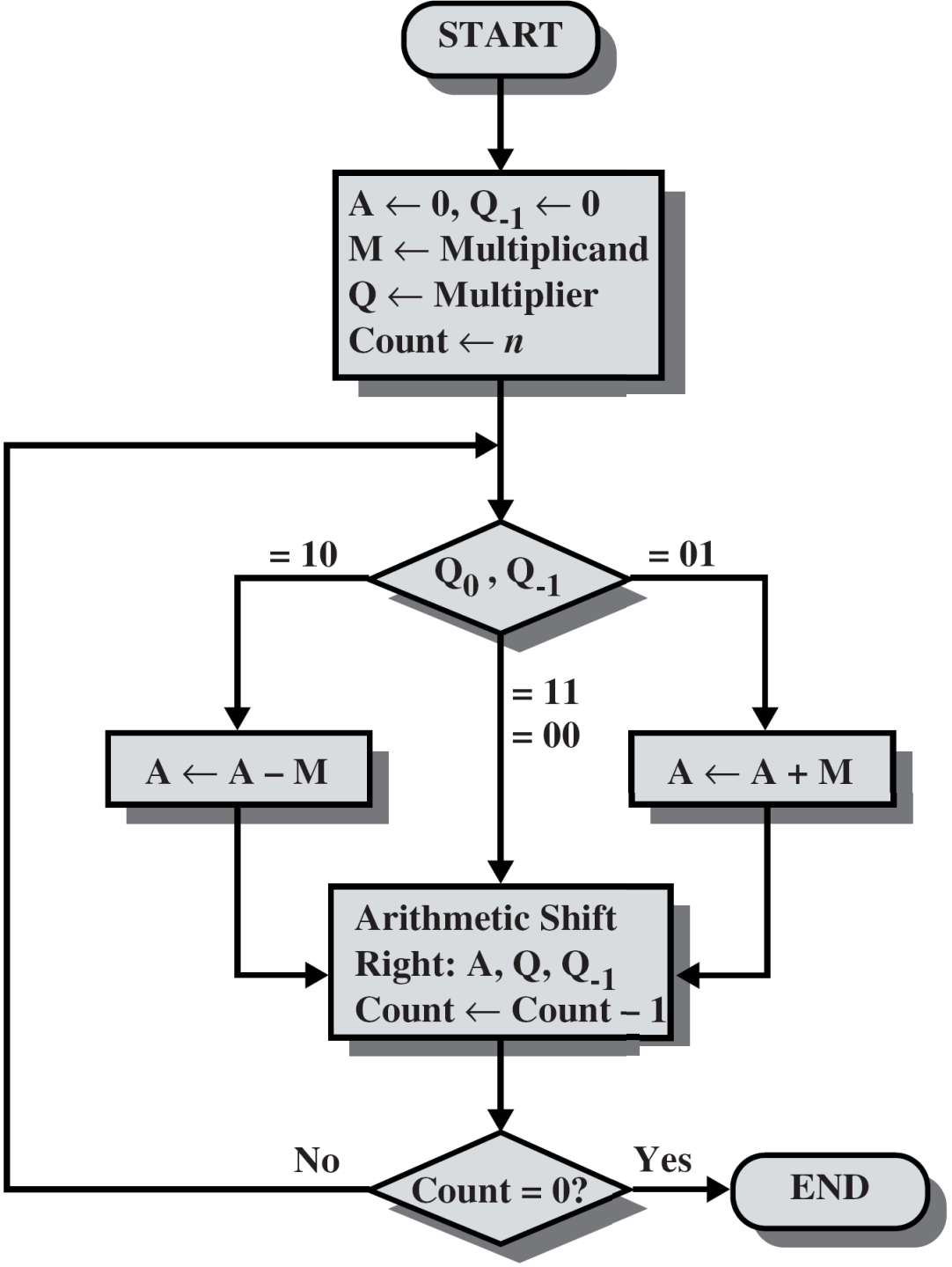
							0	1	0	1	1	0	1	
							0	+1	0	0	0	−1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	0	1	0	0	1	1		← 2's complement of the multiplicand
0	0	0	0	0	0	0	0	0	0	0	0			
0	0	0	0	0	0	0	0	0	0	0				
0	0	0	0	0	0	0	0	0	0					
0	0	0	1	0	1	1	0	1						
0	0	0	0	0	0	0	0							
0	0	0	1	0	1	0	1	0	0	0	1	1	0	

0	0	1	0	1	1	0	0	1	1	1	0	1	0	1	1	0	0
								⇓									
0	+1	−1	+1	0	−1	0	+1	0	0	−1	+1	−1	+1	0	−1	0	0

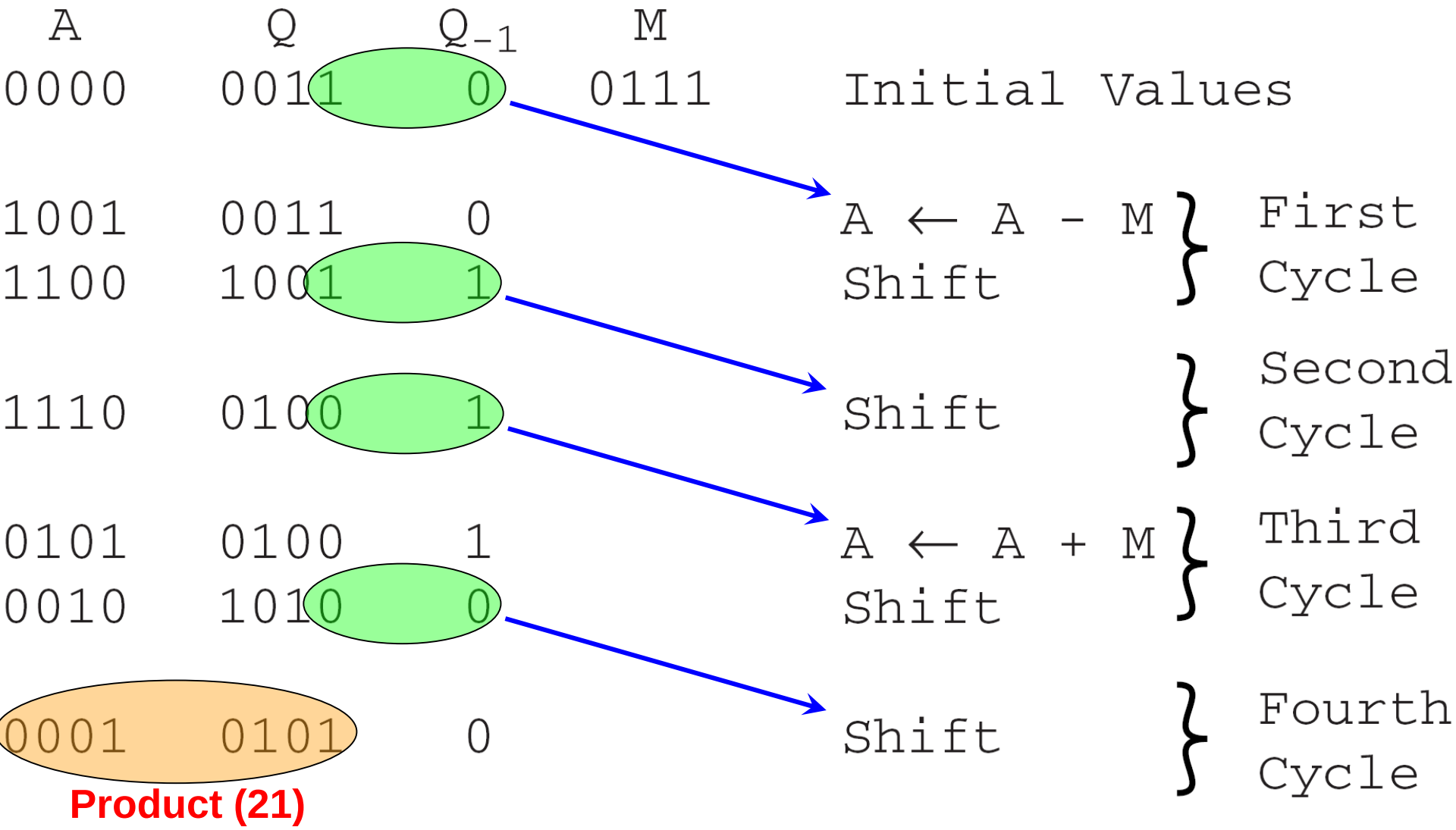
Booth's Algorithm – Rule

Multiplier		Version of multiplicand selected by bit i
Bit i	Bit i–1	
0	0	0 × M
0	1	+1 × M
1	0	–1 × M
1	1	0 × M

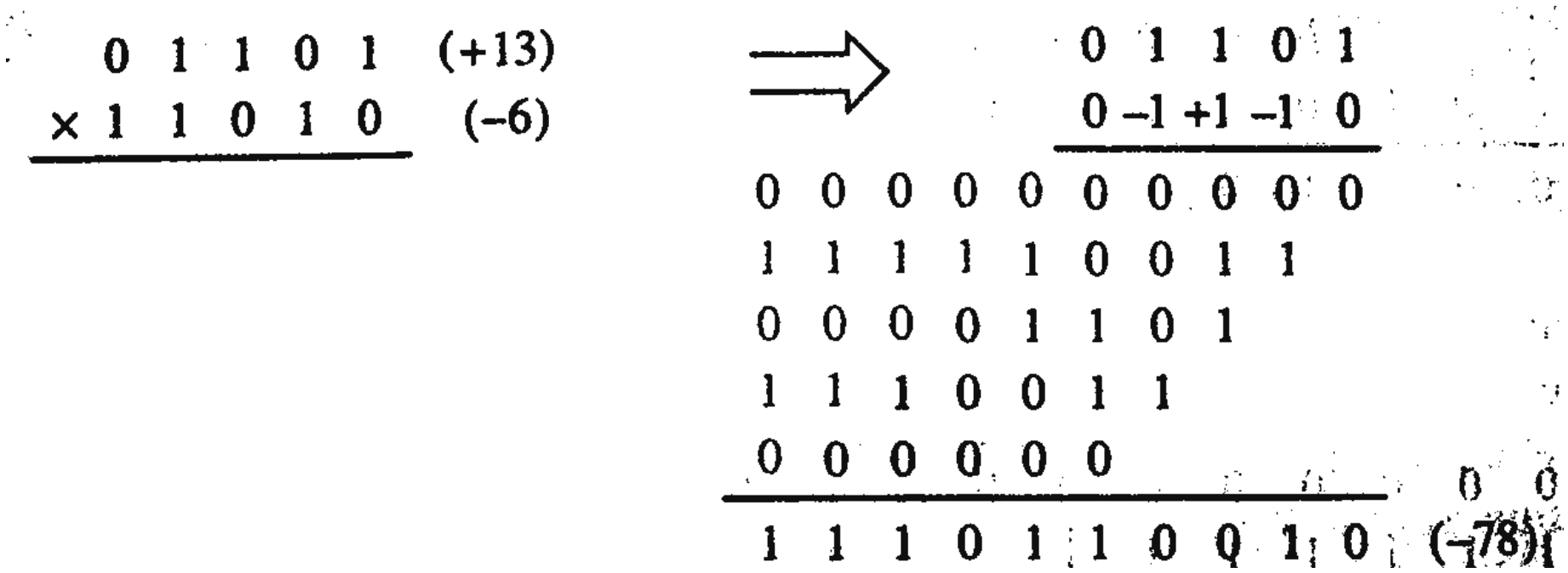
Booth's Algorithm Flowchart



Example on Booth's Algorithm



Booth's Algorithm, –ve Multiplier



Booth's Algorithm - Cases

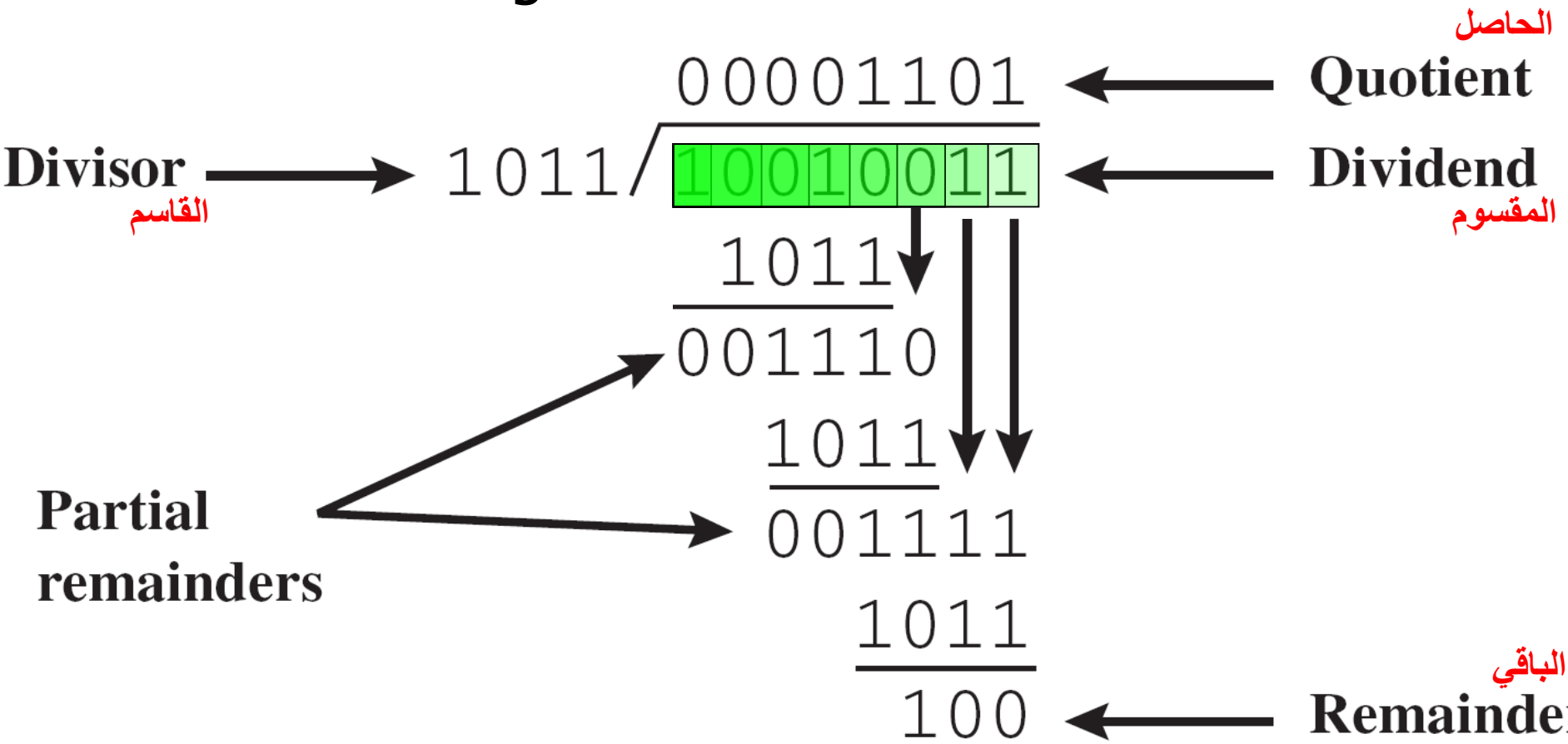
Worst-case Multiplier	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
Ordinary Multiplier	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0
	0	-1	0	0	+1	-1	+1	0	-1	+1	0	0	0	-1	0	0
Good Multiplier	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1
	0	0	+1	0	0	0	0	-1	0	0	0	0	+1	0	0	-1

Booth's Algorithm – Pros:

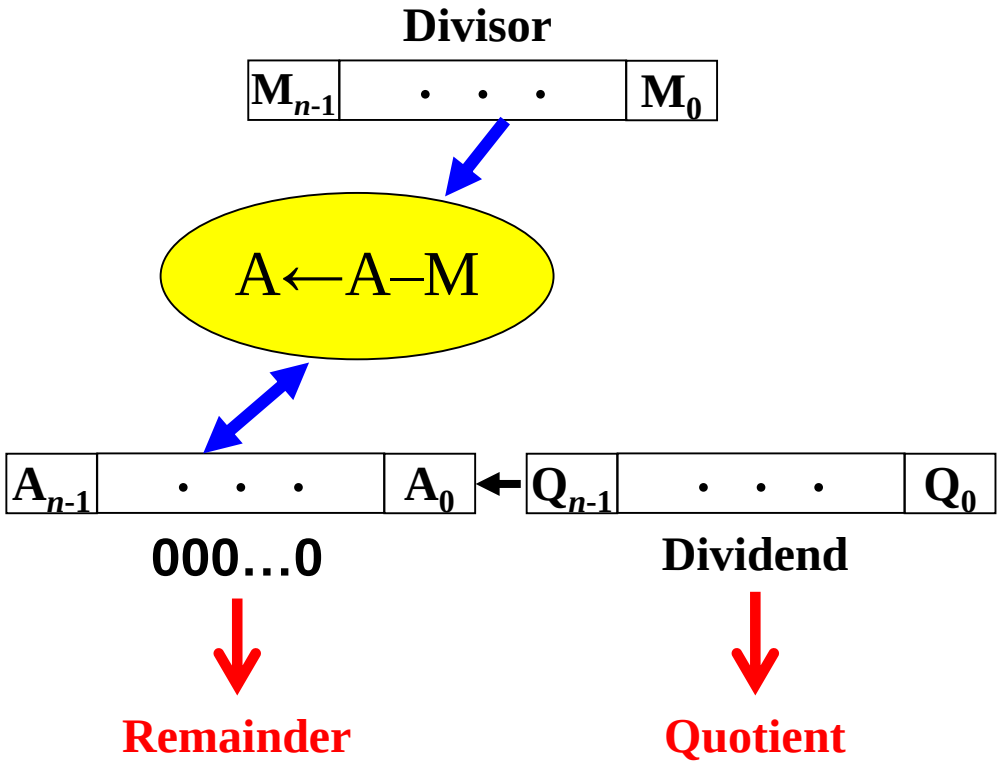
- Treats +ve and –ve multipliers uniformly.
- Use fewer additions if the multiplier has large blocks of 1's.
- On average, has the same efficiency as the normal algorithm.

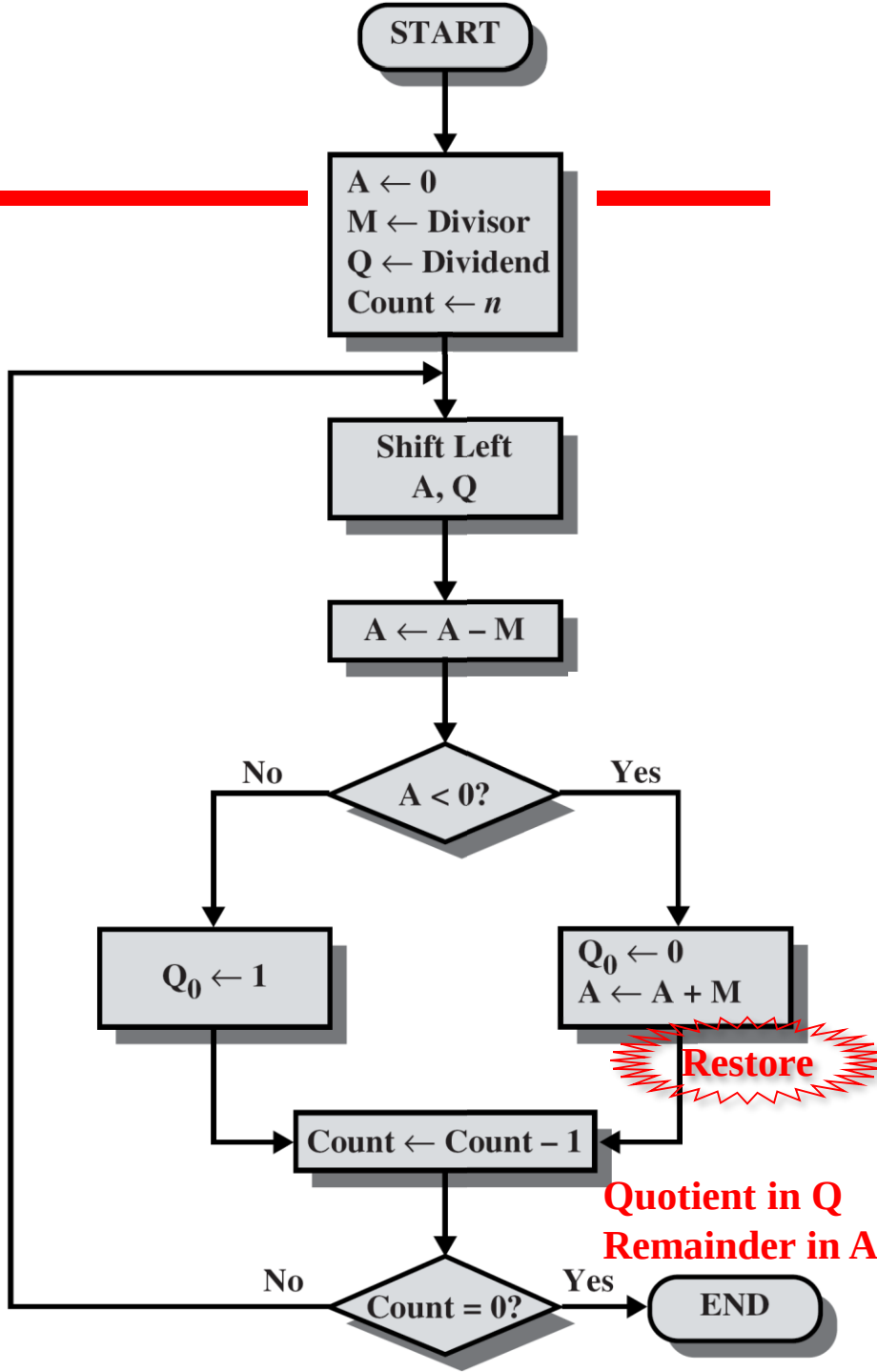
Division

- More complex than multiplication.
- Negative numbers are really bad!
- Based on long division.



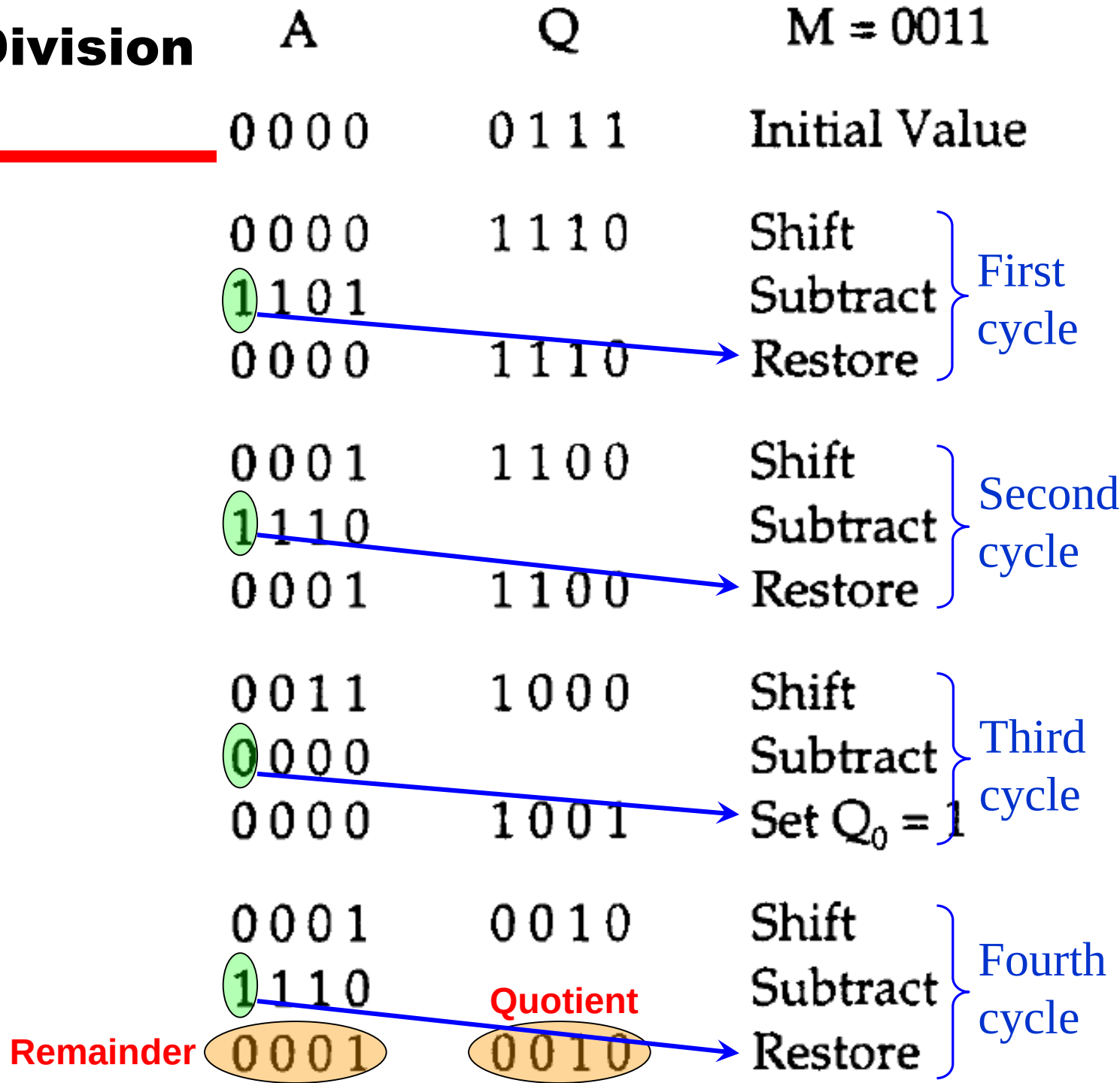
Restoring Division (Positive Integers)



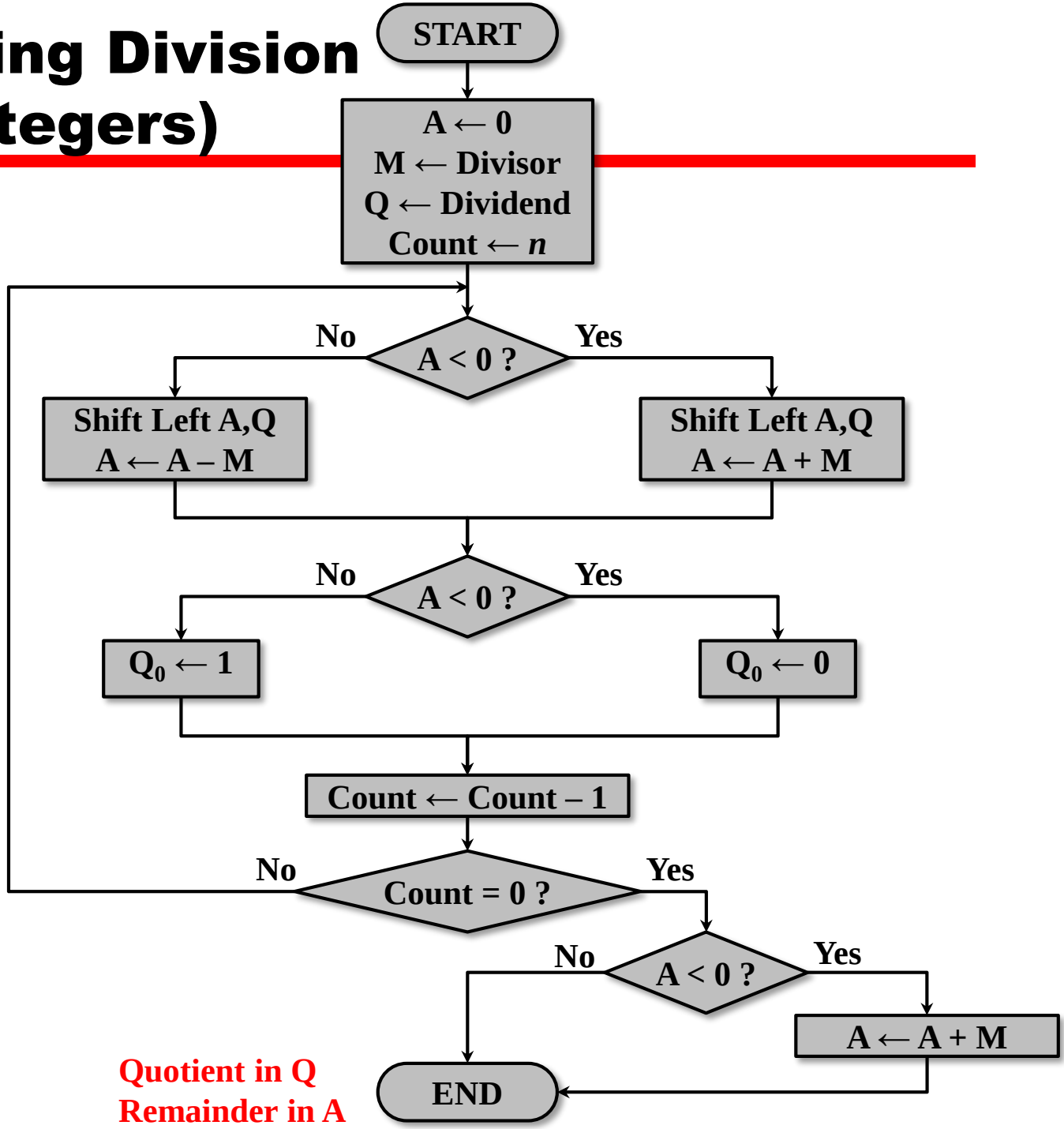


Restoring Division Example

7/3



Non-Restoring Division (Positive Integers)



Quotient in Q
Remainder in A

Non-Restoring Division Example

A	Q	M = 0011	
0000	0111	Initial Values	
0000	111?	Shift	First cycle
1101	111?	Subtract	
1101	1110	$Q_0 \leftarrow 0$	
1011	110?	Shift	Second cycle
1110	110?	Add	
1110	1100	$Q_0 \leftarrow 0$	
1101	100?	Shift	Third cycle
0000	100?	Add	
0000	1001	$Q_0 \leftarrow 1$	
0001	001?	Shift	Fourth cycle
1110	001?	Subtract	
1110	0010	$Q_0 \leftarrow 0$	
0001	**0010**	Add	
Remainder	**Quotient**		

7/3

Dealing with Signed Integers

- Given a dividend (D) and divisor (V) where both are signed integers in the 2's complement representation.
- Division can be carried out as follows:
  1. Convert D & V to their absolute (+ve) values |D| & |V|.
  2. Run either restoring or non-restoring division on |D| & |V| to obtain the quotient (Q) and the remainder (R).
  3. Adjust the sign of Q and R (by 2's complementation where needed) according to the following rules:
     - sign(Q) = sign(D) X sign(V)
     - sign(R) = sign(D)

Reading Material

- Stallings, Chapter 10:
  - Pages 331 – 341