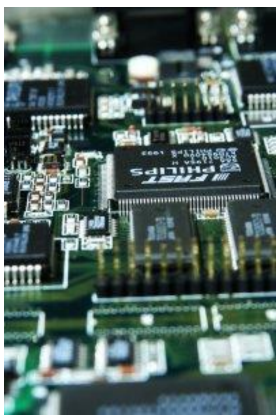
CSE 401

Computer Engineering (2)

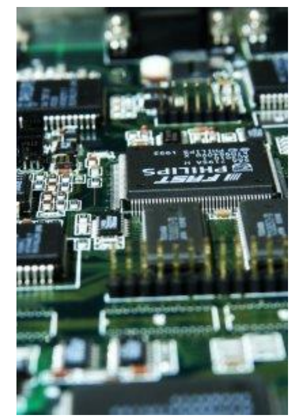
هندسة الحاسبات (2)



4th year, Comm. 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: Wednesday, April 13, 2016.
- Midterm:
  - Date: Thursday, April 21, 2014
  - Time: 10:30am – 12:00pm
  - Location: classroom #27321 (قاعة 4د)
  - Coverage: lectures #1 ➔ #7

Website: http://hshehata.github.io/courses/zu/cse401/
Office hours: Monday 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

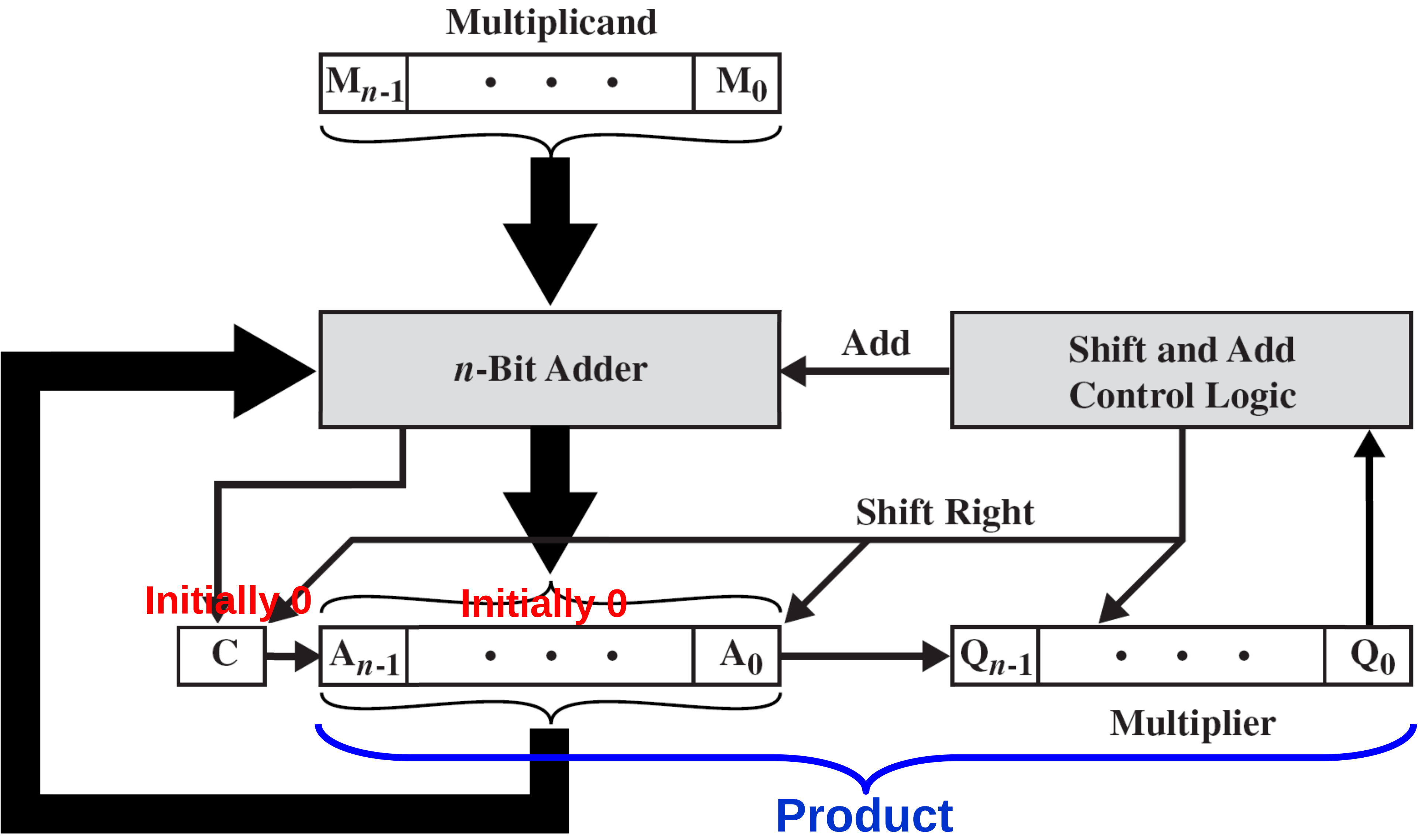
Multiplication Example

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

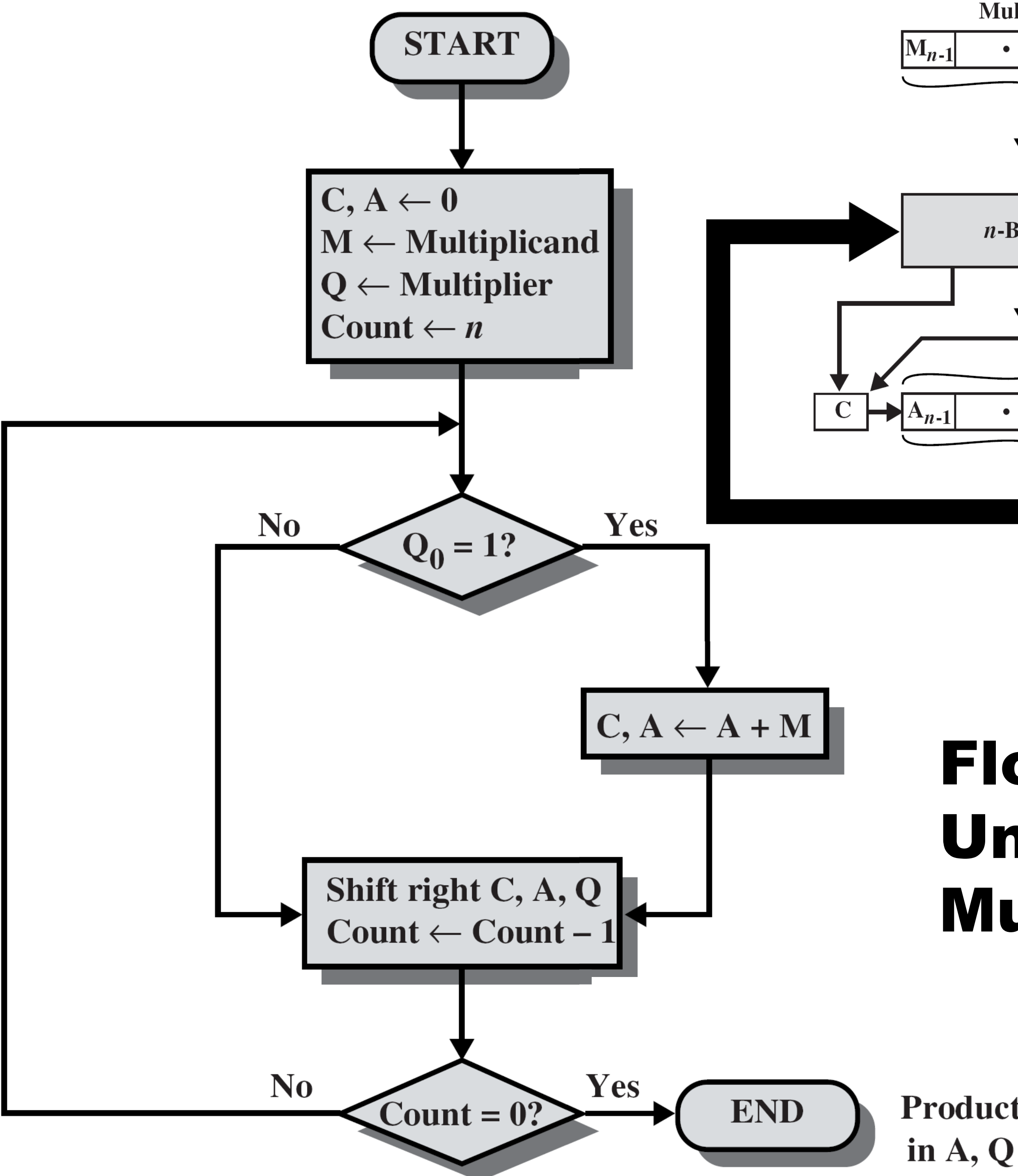
Running addition.
1 ➔ add & shift
0 ➔ shift only

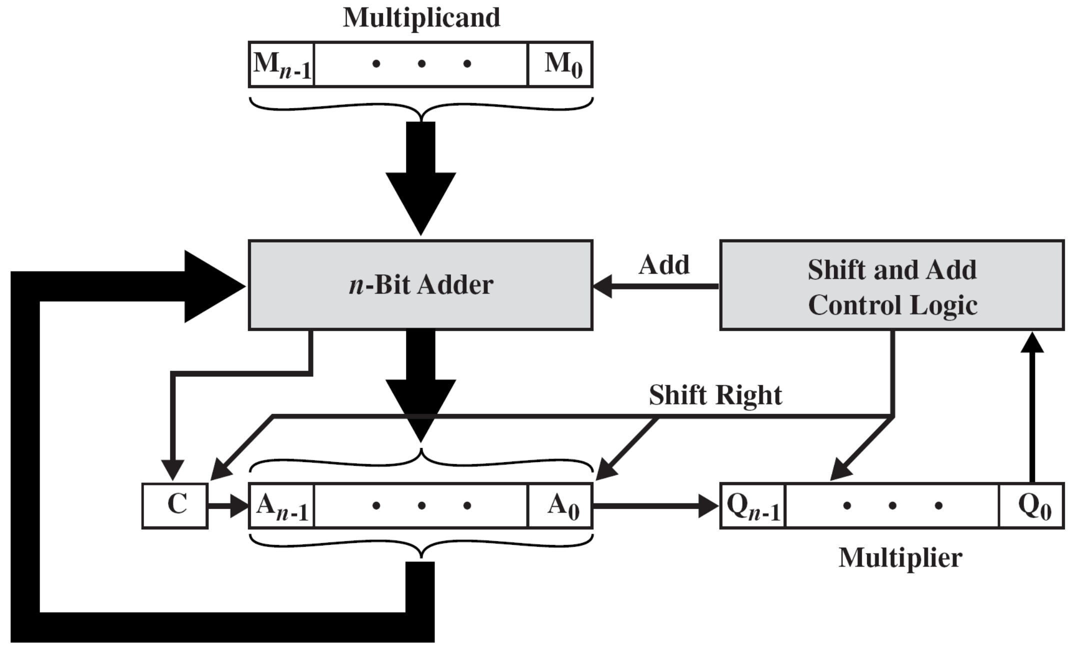
- Complex (relative to addition)!!
  - Work out a partial product for each digit.
  - Shift the partial product appropriately.
  - Add partial products.
  - Generate double-length result.

Unsigned Binary Multiplication



Flowchart for Unsigned Binary Multiplication





Execution of Example

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

Product (143)

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

Sign extension

					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)

Previous circuit can be used here with a couple of tweaks!

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
x 0 0 1 1 1 1 0

→

Multiplier

0 1 0 0 0 0 0
– 0 0 0 0 0 1 0

→

Multiplier

0 +1 0 0 0 –1 0

							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	
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

2's complement of the multiplicand

0 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 0

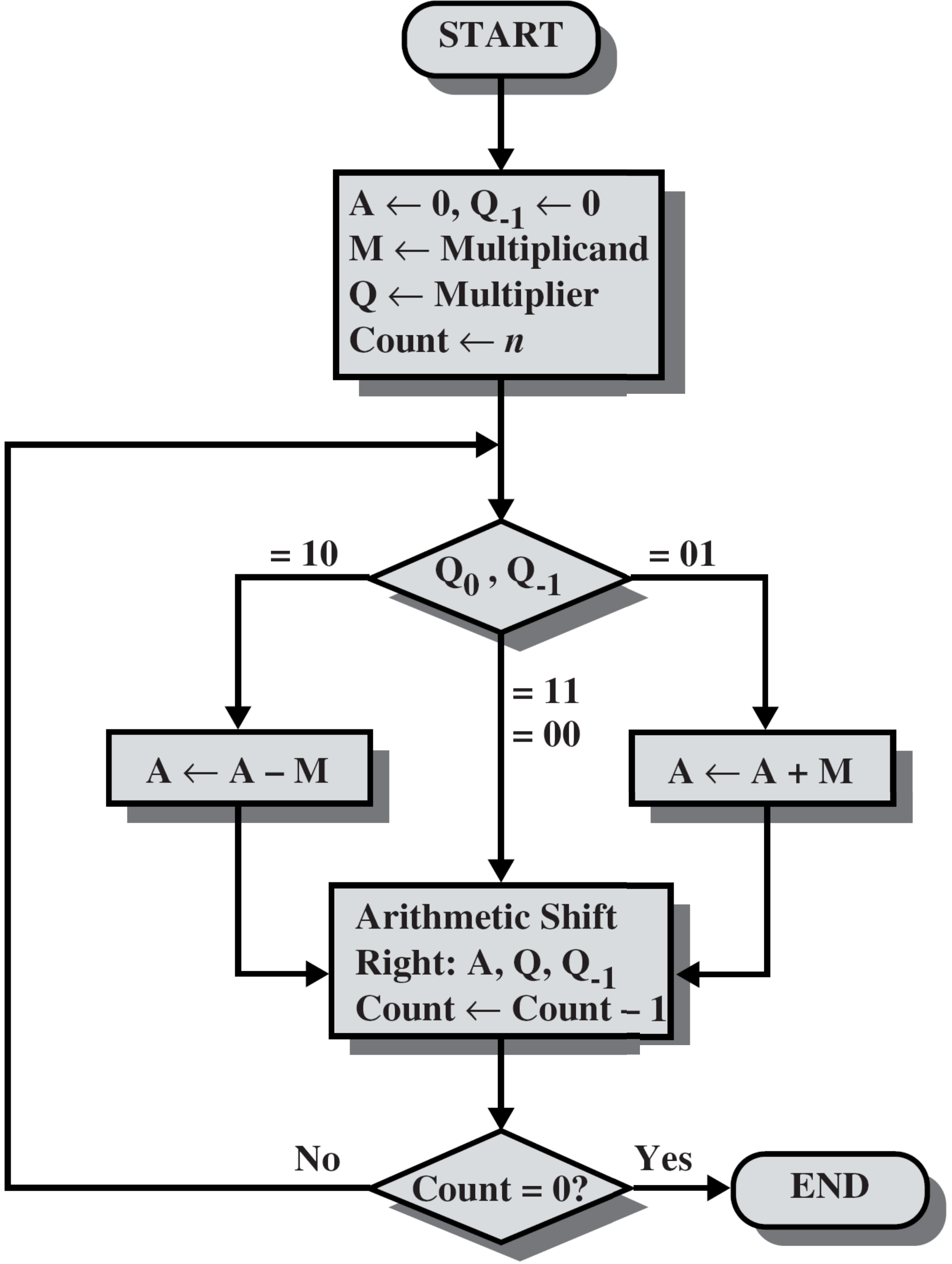
$\Downarrow$

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	$0 \times M$
0	1	$+1 \times M$
1	0	$-1 \times M$
1	1	$0 \times M$

Booth’s Algorithm Flowchart



Example on Booth’s Algorithm

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

Product (21)

Booth's Algorithm, –ve Multiplier

$$
\begin{array}{rrrrrrl}
 & 0 & 1 & 1 & 0 & 1 & (+13) \\
\times & 1 & 1 & 0 & 1 & 0 & (-6) \\
\hline
\end{array}
\quad \Longrightarrow \quad
\begin{array}{rrrrrrrrrrl}
 & & & & & 0 & 1 & 1 & 0 & 1 & \\
 & & & & & 0 & -1 & +1 & -1 & 0 & \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & & \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & & & \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & & & & & \\
\hline
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & (-78)
\end{array}
$$

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

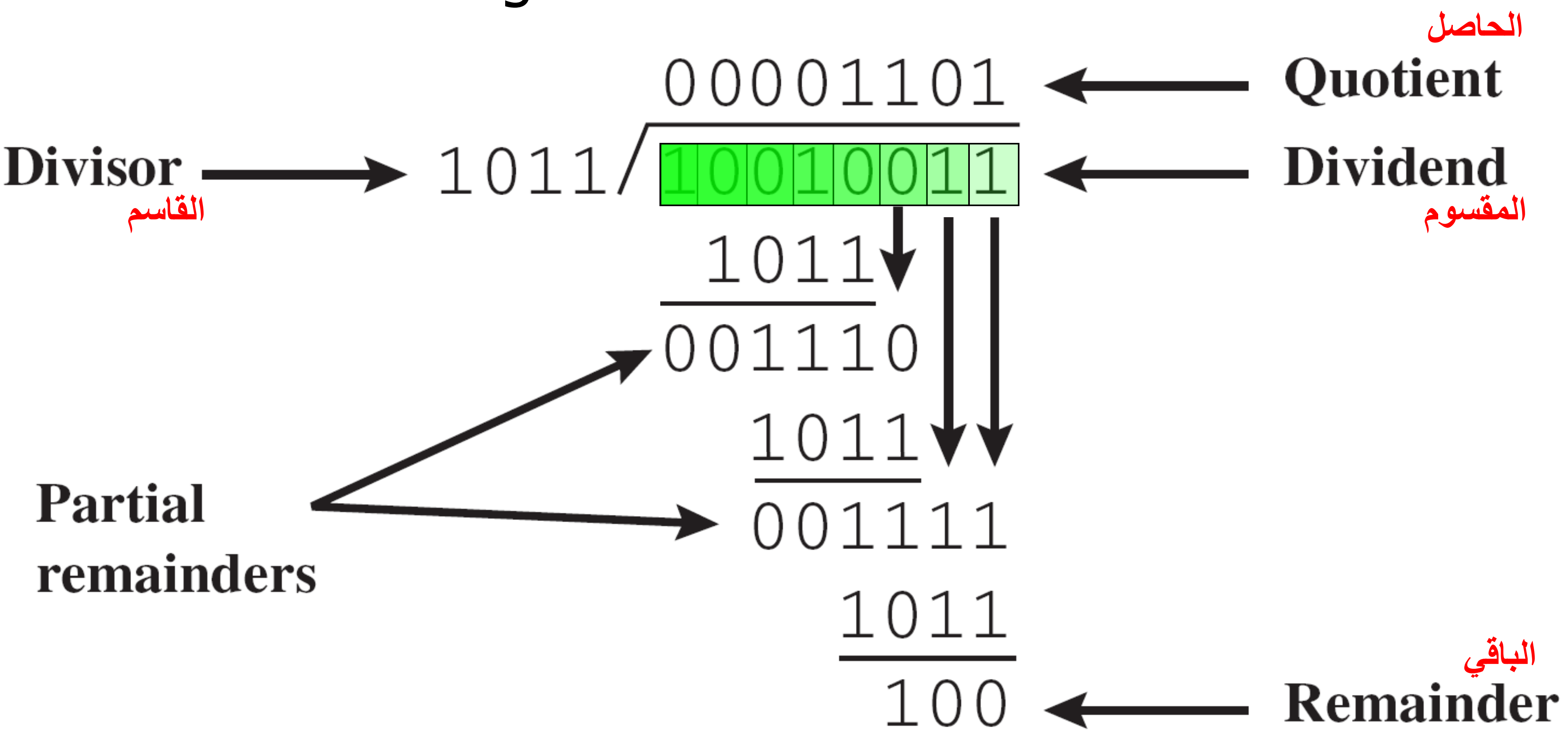
Ordinary Multiplier	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0
	0	-1	0	0	+1	-1	+1	0	-1	+1	0	0	0	-1	0	0

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

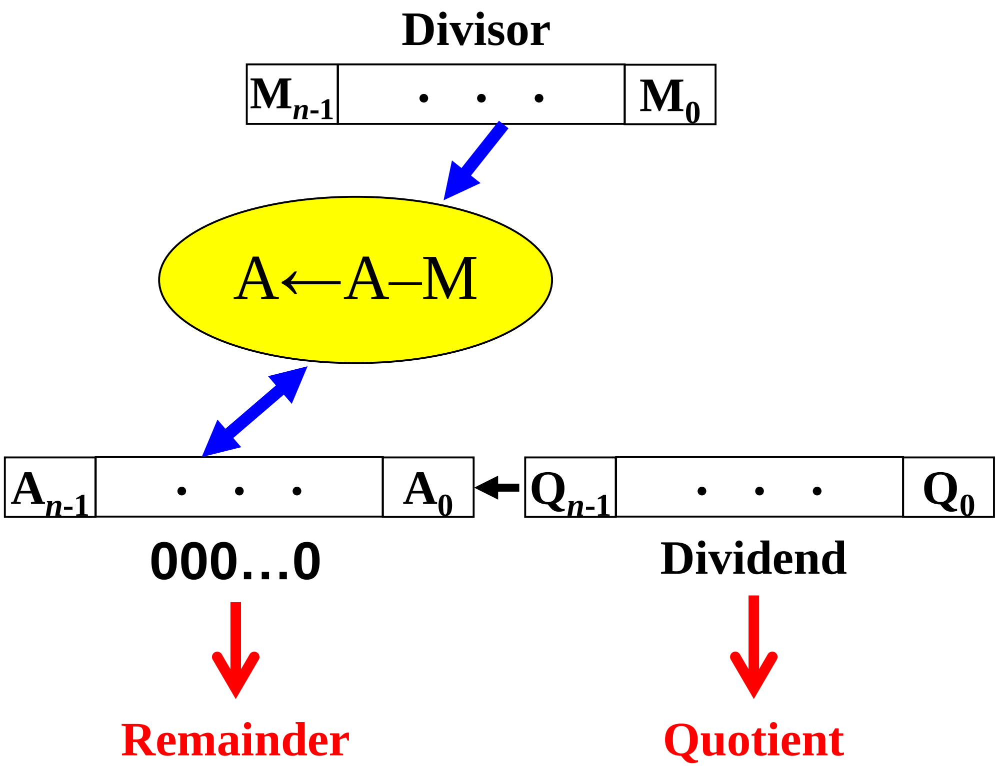
Booth's Algorithm – Pros:
- Treats +ve and –ve multipliers uniformly.
- Use fewer additions if the multiplier has large blocks of 1's.
- On average, has the same efficiency as the normal algorithm.

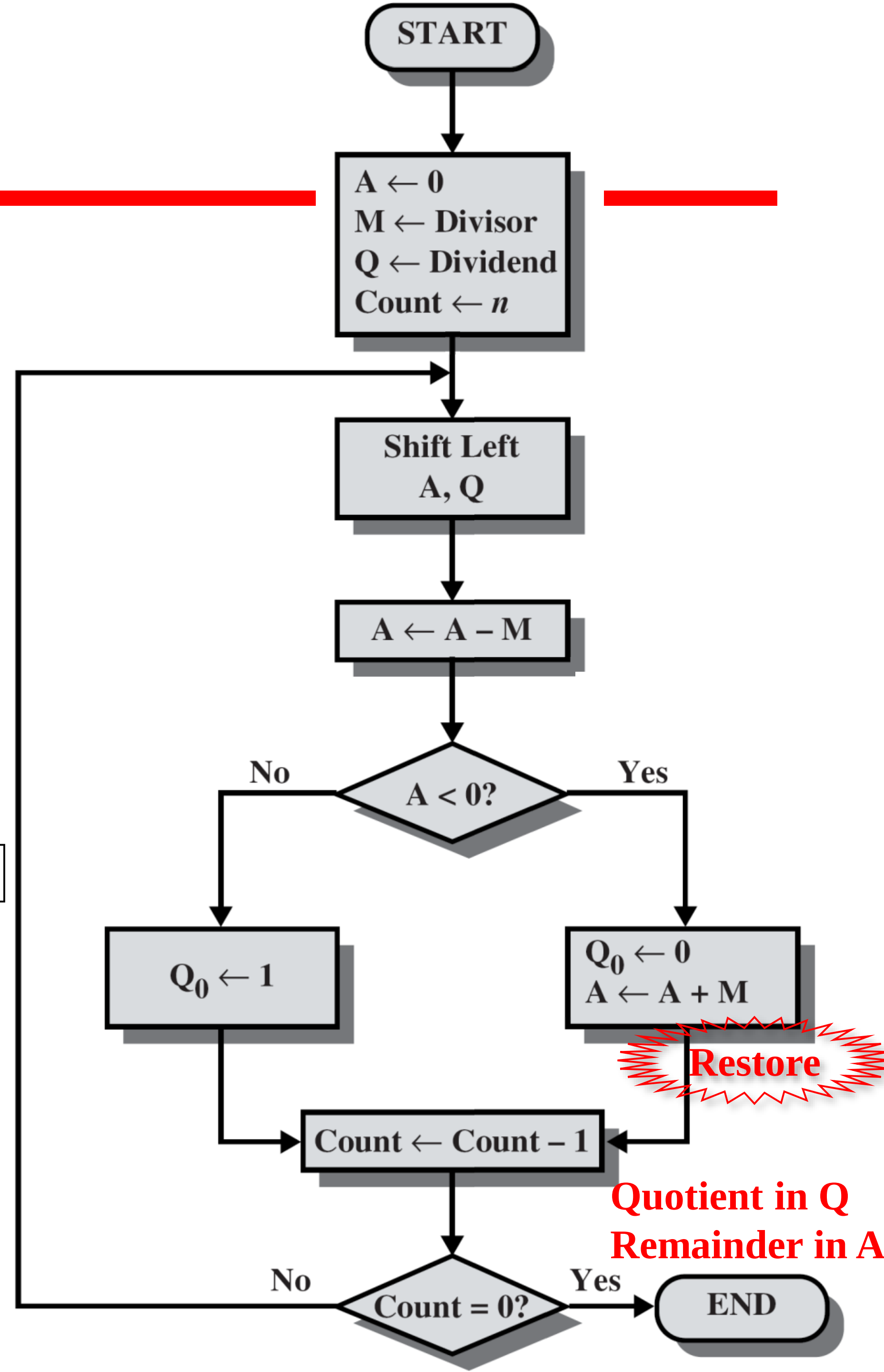
Division

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



Restoring Division (Positive Integers)



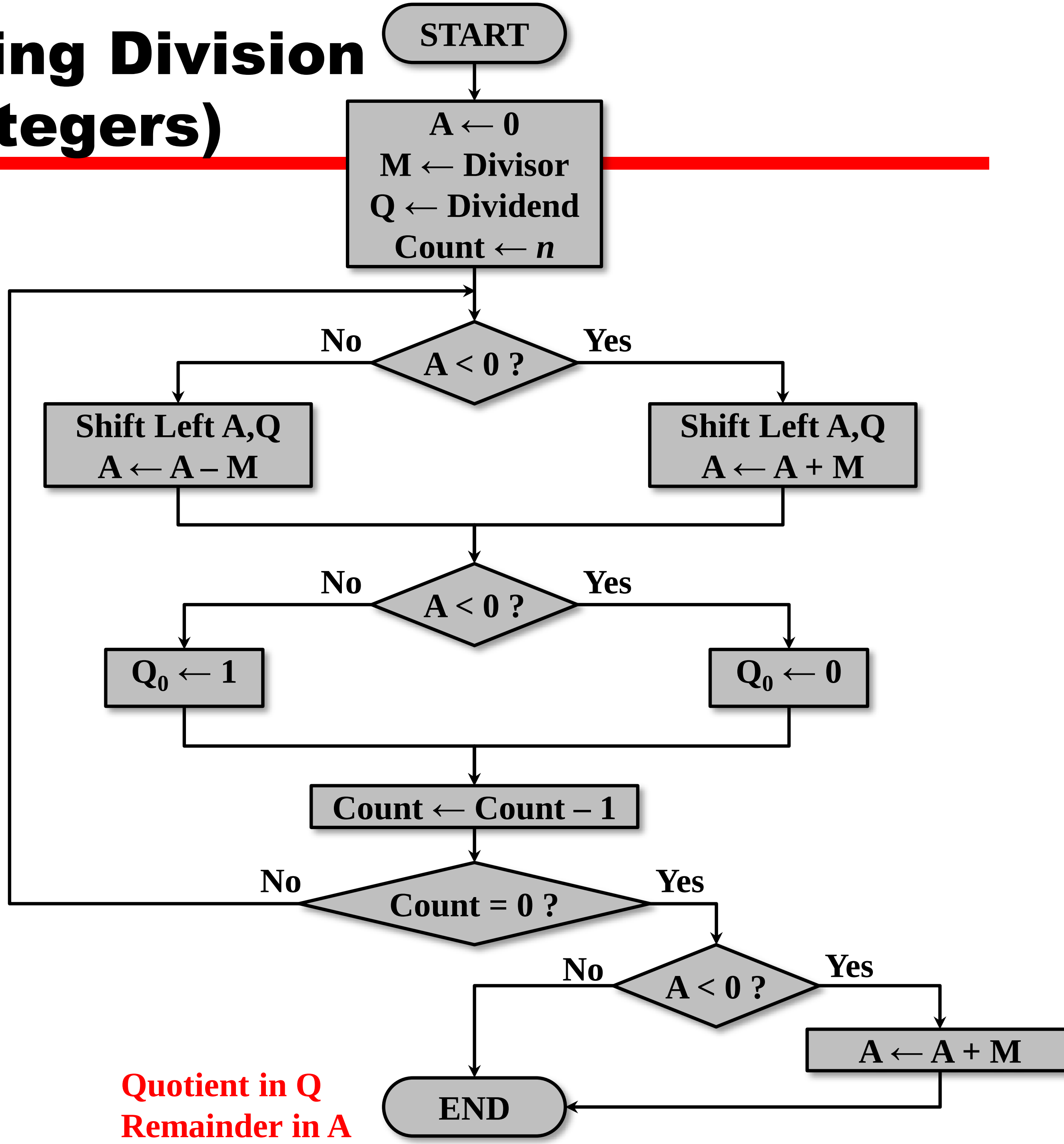


Restoring Division Example

7/3

A	Q	M = 0011	
0 0 0 0	0 1 1 1	Initial Value	
0 0 0 0	1 1 1 0	Shift	First cycle
1 1 0 1		Subtract	
0 0 0 0	1 1 1 0	Restore	
0 0 0 1	1 1 0 0	Shift	Second cycle
1 1 1 0		Subtract	
0 0 0 1	1 1 0 0	Restore	
0 0 1 1	1 0 0 0	Shift	Third cycle
0 0 0 0		Subtract	
0 0 0 0	1 0 0 1	Set $Q_0 = 1$	
0 0 0 1	0 0 1 0	Shift	Fourth cycle
1 1 1 0		Subtract	
Remainder 0 0 0 1	Quotient 0 0 1 0	Restore	

Non-Restoring Division (Positive Integers)



Non-Restoring Division Example

7/3

A	Q	M = 0011	
0000	0111	Initial Values	
0000	111?	Shift	First cycle
1101	111?	Subtract	
1101	111**0**	$Q_0 \leftarrow 0$	
1011	11**0**?	Shift	Second cycle
1110	11**0**?	Add	
1110	11**00**	$Q_0 \leftarrow 0$	
1101	1**00**?	Shift	Third cycle
0000	1**00**?	Add	
0000	1**001**	$Q_0 \leftarrow 1$	
0001	**001**?	Shift	Fourth cycle
1110	**001**?	Subtract	
1110	**0010**	$Q_0 \leftarrow 0$	
0001	**0010**	Add	
Remainder	Quotient		

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