CS330 - Computer Organization and Assembly Language Programming

Lecture 8

-Integer Arithmetic-

Professor : Mahmut Unan – UAB CS

Agenda

- Signed Multiplication
- Booth’s Algorithm

Multiplication

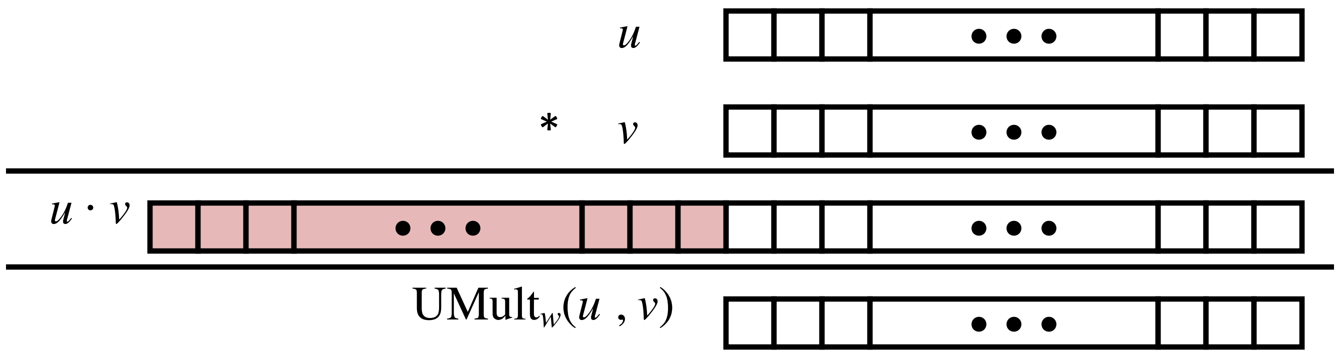
- Goal: Computing Product of *w*-bit numbers *x*, *y*
  - Either signed or unsigned
- But, exact results can be bigger than *w* bits
  - Unsigned: up to 2*w* bits
    - Result range: $0 \le x * y \le (2^{w} - 1)^{2} = 2^{2w} - 2^{w+1} + 1$
  - Two's complement min (negative): Up to 2*w*-1 bits
    - Result range: $x * y \ge (-2^{w-1})*(2^{w-1}-1) = -2^{2w-2} + 2^{w-1}$
  - Two's complement max (positive): Up to 2*w* bits, but only for $(TMin_{w})^{2}$
    - Result range: $x * y \le (-2^{w-1})^{2} = 2^{2w-2}$
- So, maintaining exact results…
  - would need to keep expanding word size with each product computed
  - is done in software, if needed
    - e.g., by "arbitrary precision" arithmetic packages

Unsigned Multiplication in C

Operands: *w* bits

True Product: 2**w* bits

Discard *w* bits: *w* bits



- Standard Multiplication Function
  - Ignores high order *w* bits
- Implements Modular Arithmetic

  $\mathrm{UMult}_w(u\,,v) \;=\; u \cdot v \bmod 2^w$

Unsigned Binary Multiplication

```
      1011      Multiplicand (11)
    ×1101       Multiplier (13)
    ------
      1011  ⎫
     0000   ⎪
    1011    ⎬   Partial products
   1011     ⎭
  --------
  10001111      Product (143)
```

Figure 10.7 Multiplication of Unsigned Binary Integers

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

```
  0111
  0100
x________
  0000
```

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

```
    0111
    0100
x_________
    0000
   0000
```

Unsigned Multiplication

- 7 * 4

7 = 0111

4 = 0100

```
     0111
     0100
x_________
     0000
    0000
   0111
```

Unsigned Multiplication

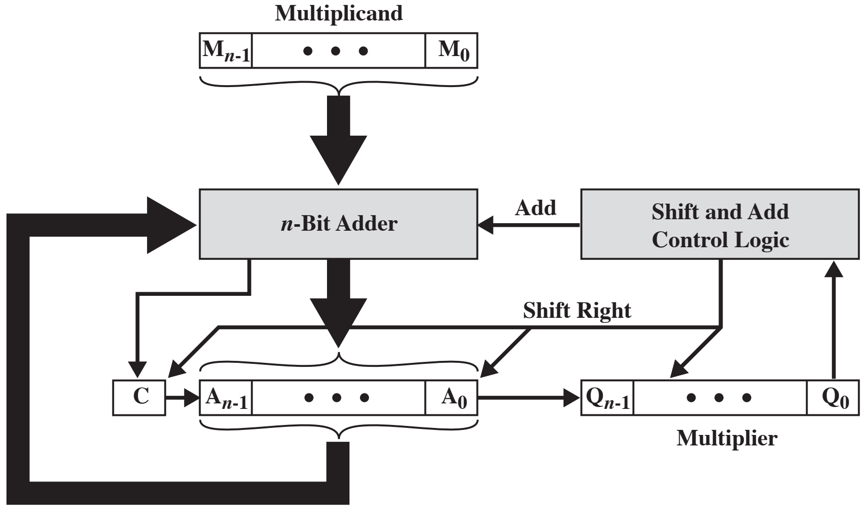
- 7 * 4

7 = 0111

4 = 0100

```
      0111
      0100
  x________
      0000
     0000
    0111
+  0000
___________
   0011100
```

= 28



(a) Block Diagram

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)

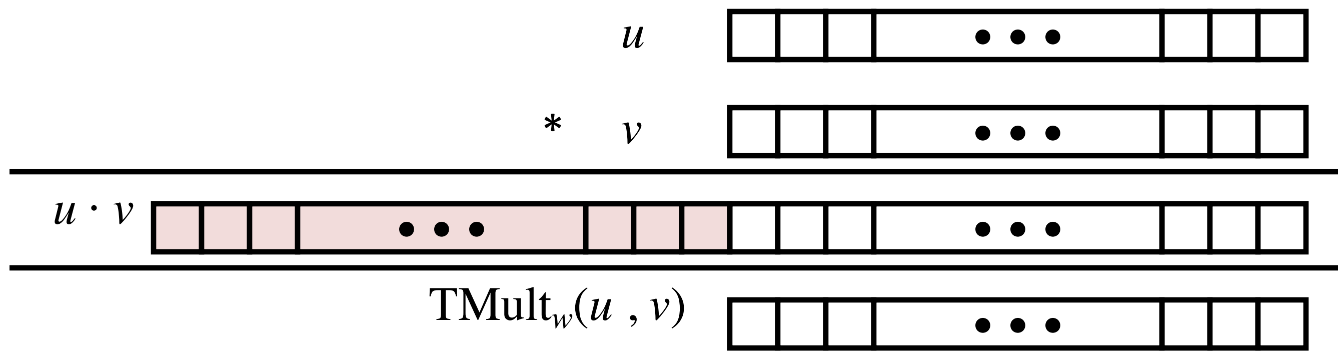
Figure 10.8 Hardware Implementation of Unsigned Binary Multiplication

Signed Multiplication in C

Operands: *w* bits

True Product: 2**w* bits

Discard *w* bits: *w* bits



- Standard Multiplication Function
  - Ignores high order *w* bits
  - Some of which are different for signed vs. unsigned multiplication
  - Lower bits are the same

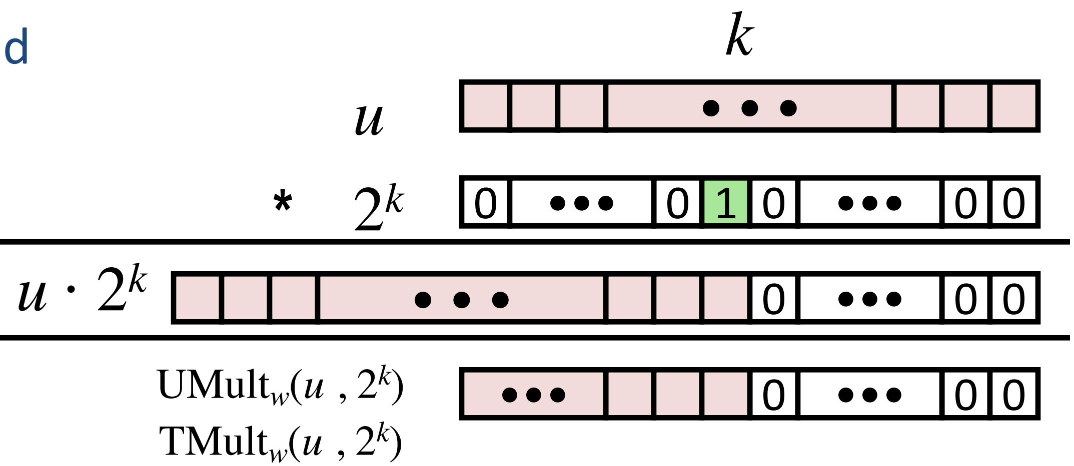
Power-of-2 Multiply with Shift

- **Operation**
  - **`u << k`** gives **`u * 2`**^{k}
  - Both signed and unsigned

Operands: *w* bits

True Product: *w*+*k* bits

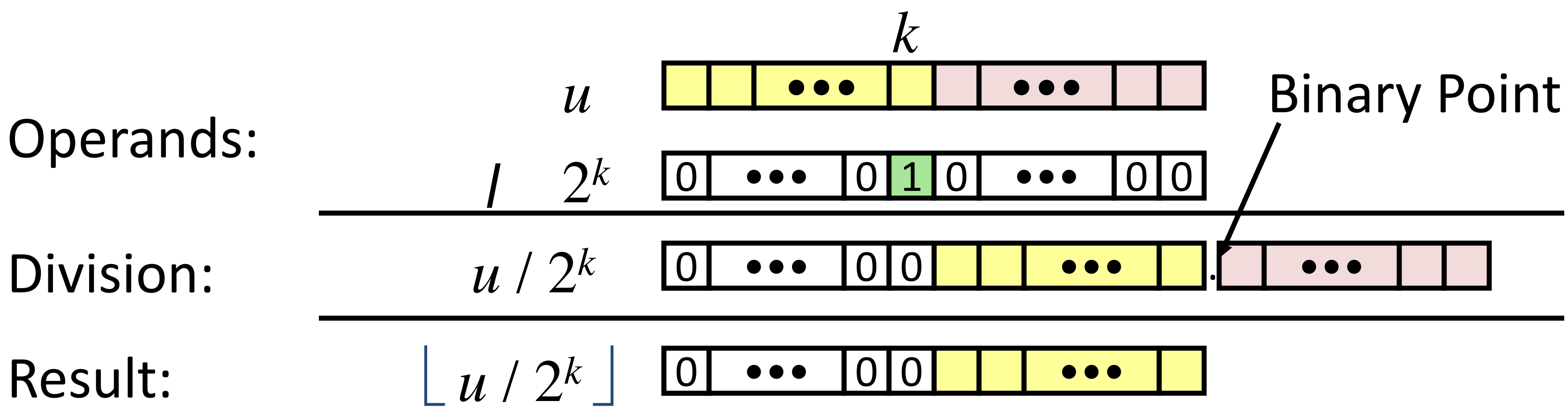
Discard *k* bits: *w* bits



- **Examples**
  - **`u << 3 == u * 8`**
  - **`(u << 5) - (u << 3)== u * 24`**
  - Most machines shift and add faster than multiply
    - Compiler generates this code automatically

Unsigned Power-of-2 Divide with Shift

- **Quotient of Unsigned by Power of 2**
  - `u >> k` gives $\lfloor$ `u / 2`^{k} $\rfloor$
  - Uses logical shift



	Division	Computed	Hex	Binary
`x`	15213	15213	`3B 6D`	`00111011 01101101`
`x >> 1`	7606.5	7606	`1D B6`	`00011101 10110110`
`x >> 4`	950.8125	950	`03 B6`	`00000011 10110110`
`x >> 8`	59.4257813	59	`00 3B`	`00000000 00111011`

Booth's Algorithm

- A Q Q_{-1} Action

----

00 → Shift (Arithmetic Right Shift)

01→ Add Numbers (A=A+M), Shift

10→ Subtract Numbers (A=A-M), Shift

11→ Shift (Arithmetic Right Shift)

Exercises

- 6*(-2)
- 5*(-4)
- 14*(-5)

Booth's Algorithm Steps

- Prepare the following values
  - Multiplicand →M
  - Multiplier → Q
- Calculate the two's complement
  - -M & -Q
- Decide how many bits required
  - if you need w bits to represent M or Q
  - you need w iteration
  - the result will be 2*w bits
  - A value will be A=0000…0 (number of zeros=w)
  - Q-1 value is always 0

$6 * (-2) =$

- A Q Q_{-1} Action

----

6 → 0110 → Multiplicand →M

6 * (-2) =

- A Q Q_{-1} Action

----

6 → 0110 → Multiplicand →M

1001

1

+__________

-M = 1010

6 * (-2) =

- A Q Q_{-1} Action

---

6 → 0110 → Multiplicand → M

1001

1

+__________

-M = 1010

2→ 0010 → Multiplier → Q

1101

1

+__________

1110

$6 * (-2) =$

- A Q Q_{-1} Action

----

6 → 0110 → Multiplicand → M

1001

1

+___________

-M = 1010

2→ 0010 → Multiplier → Q

1101

1

+___________

-2→1110

A =0000

$Q_{-1} = 0$

$M=0110 \quad -M = 1010 \quad Q=1110 \quad A=0000 \quad Q_{-1} =0$

Step	A	Q	Q_{-1}	Action
1	0000	1110	0	
2				
3				
4				

4 Bits→ 4Iterations
Result = 8 bits

00	Arithmetic Right Shift
11	Arithmetic Right Shift
01	ADD (A=A+M)+ Shift
10	SUBTRACT (A=A-M)+Shift

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000	1110	0	00→ Shift(AR)
2				
3				
4				

$M=0110 \quad -M = 1010 \quad Q=1110 \quad A=0000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2				
3				
4				

Arithmetic Right Shift → Duplicate the left most bit

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3				
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

$M=0110 \quad -M = 1010 \quad Q=1110 \quad A=0000 \quad Q_{-1} = 0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3	1101 1110	0011 1001	1 1	11→ Shift(AR)
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3	1101 1110	0011 1001	1 1	11→ Shift(AR)
4	1110 1111	1001 0100	1 1	11→ Shift(AR)

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0110 -M = 1010 Q=1110 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1110 0111	0 0	00→ Shift(AR)
2	0000 1010 1101	0111 0111 0011	0 0 1	10 → Subtract (A=A-M) A = 0000 -M =1010 A-M=1010 Shift(AR)
3	1101 1110	0011 1001	1 1	11→ Shift(AR)
4	1110 1111	1001 0100	1 1	11→ Shift(AR)

1111 0100 = -12

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

Example 2 → 5 * -4

- A Q Q_{-1} Action

---

5 → 0101 → Multiplicand → M

1010

1

+__________

-M = 1011

4→ 0100 → Multiplier → Q

1011

1

+__________

-4→1100

M=0101 -M = 1011 Q=1100 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000	1110	0	00→ Shift(AR)
2				
3				
4				

4 Bits→ 4Iterations
Result = 8 bits

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0101 -M = 1011 Q=1100 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2				
3				
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0101 -M = 1011 Q=1100 A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3				
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=0101   -M = 1011 Q=1100   A=0000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3	0000 1011 1101	0011 0011 1001	0 0 1	10→ Subtract A=0000 -M= 1011 A=A-M → 1011 Shift(AR)
4				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

$M=0101 \quad -M = 1011 \quad Q=1100 \quad A=0000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3	0000 1011 1101	0011 0011 1001	0 0 1	10→ Subtract A=0000 -M= 1011 A=A-M → 1011 Shift(AR)
4	1101 1110	1001 1100	1 1	11→ Shift(AR)

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

$M=0101 \quad -M = 1011 \quad Q=1100 \quad A=0000 \quad Q_{-1} = 0$

Step	A	Q	Q_{-1}	Action
1	0000 0000	1100 0110	0 0	00→ Shift(AR)
2	0000 0000	0110 0011	0 0	00→ Shift(AR)
3	0000 1011 1101	0011 0011 1001	0 0 1	10→ Subtract A=0000 -M= 1011 A=A-M → 1011 Shift(AR)
4	1101 1110	1001 1100	1 1	11→ Shift(AR)

1110 1100 =-20

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

Example 3 → 14 * -5

- A Q Q_{-1} Action

---

14 → 01110 → Multiplicand → M

5 Bits→ 5 Iterations
Result = 10 bits

Example 3 → 14 * -5

- A Q Q_{-1} Action

----

14 → 01110 → Multiplicand → M
10001
1
+__________
-M = 10010
5→ 00101 → Multiplier → Q
11010
1
+__________
-5→11011

$M=01110 \quad -M = 10010 \quad Q=11011 \quad A=00000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	
2				
3				
4				
5				

5 Bits→ 5 Iterations
Result = 10 bits

00	Arithmetic Right Shift
11	Arithmetic Right Shift
01	ADD (A=A+M)+ Shift
10	SUBTRACT (A=A-M)+Shift

$M=01110 \quad -M = 10010 \quad Q=11011 \quad A=00000 \quad Q_{-1} = 0$

Step	A	Q	Q_{-1}	Action
1	00000 10010 11001	11011 11011 01101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010 A-M= 10010 Shift (AR)
2				
3				
4				
5				

00	**Arithmetic Right Shift**
11	**Arithmetic Right Shift**
01	**ADD (A=A+M)+ Shift**
10	**SUBTRACT (A=A-M)+Shift**

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010
	10010	11011	0	A-M= 10010
	11001	01101	1	Shift (AR)
2	11001	01101	1	Shift (AR)
	11100	10110	1	
3				
4				
5				

00	Arithmetic Right Shift
11	Arithmetic Right Shift
01	ADD (A=A+M)+ Shift
10	SUBTRACT (A=A-M)+Shift

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	10→ SUBTRACT (A=A-M)+Shift
				A = 00000
				-M = 10010
	10010	11011	0	A-M= 10010
	11001	01101	1	Shift (AR)
2	11001	01101	1	Shift (AR)
	11100	10110	1	
3	11100	10110	1	01→ ADD (A=A+M)+ Shift
				A = 11100
				M = 01110 Carry - ignored
	01010	10110	1	A+M= 01010 → Actually 101010
	00101	01011	0	Shift (AR)
4				
5				

$M=01110$ $-M = 10010$ $Q=11011$ $A=00000$ $Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	00000 10010 11001	11011 11011 01101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010 A-M= 10010 Shift (AR)
2	11001 11100	01101 10110	1 1	Shift (AR)
3	11100 01010 00101	10110 10110 01011	1 1 0	01→ ADD (A=A+M)+ Shift A = 11100 M = 01110 Carry - ignored A+M= 01010 → Actually 101010 Shift (AR)
4	00101 10111 11011	01011 01011 10101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00101 -M = 10010 A-M= 10111 Shift (AR)
5				

M=01110 -M = 10010 Q=11011 A=00000 Q_{-1} =0

Step	A	Q	Q_{-1}	Action
1	00000 10010 11001	11011 11011 01101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010 A-M= 10010 Shift (AR)
2	11001 11100	01101 10110	1 1	Shift (AR)
3	11100 01010 00101	10110 10110 01011	1 1 0	01→ ADD (A=A+M)+ Shift A = 11100 Carry - ignored M = 01110 A+M= 01010 → Actually 101010 Shift (AR)
4	00101 10111 11011	01011 01011 10101	0 0 1	10→ SUBTRACT (A=A-M)+Shift A = 00101 -M = 10010 A-M= 10111 Shift (AR)
5	11011 11101	10101 11010	1 1	Shift (AR)

$M=01110 \quad -M = 10010 \quad Q=11011 \quad A=00000 \quad Q_{-1}=0$

Step	A	Q	Q_{-1}	Action
1	00000	11011	0	10→ SUBTRACT (A=A-M)+Shift A = 00000 -M = 10010
	10010	11011	0	A-M= 10010
	11001	01101	1	Shift (AR)
2	11001	01101	1	Shift (AR)
	11100	10110	1	
3	11100	10110	1	01→ ADD (A=A+M)+ Shift A = 11100 M = 01110
	01010	10110	1	A+M= 01010 → Actually 101010
	00101	01011	0	Shift (AR)
4	00101	01011	0	10→ SUBTRACT (A=A-M)+Shift A = 00101 -M = 10010
	10111	01011	0	A-M= 10111
	11011	10101	1	Shift (AR)
5	11011	10101	1	Shift (AR)
	11101	11010	1	

Carry - ignored

11101 11010 =-70

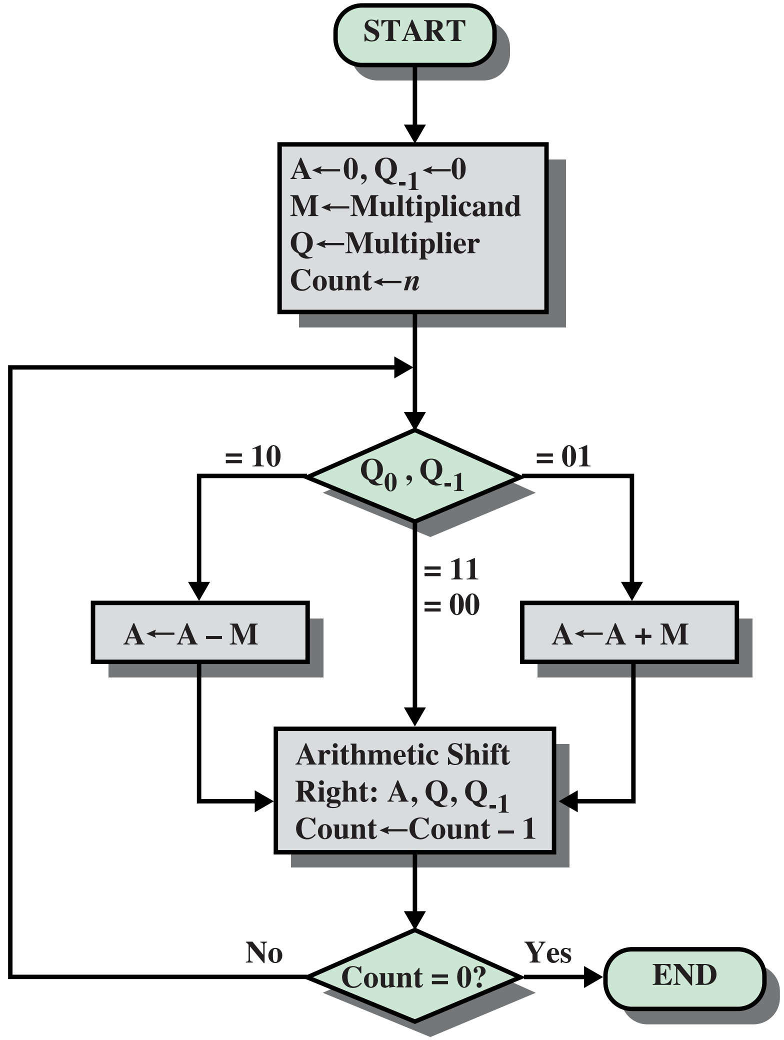


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

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

Figure 10.13 Example of Booth's Algorithm (7× 3)

```
     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 10.14 Examples Using Booth's Algorithm

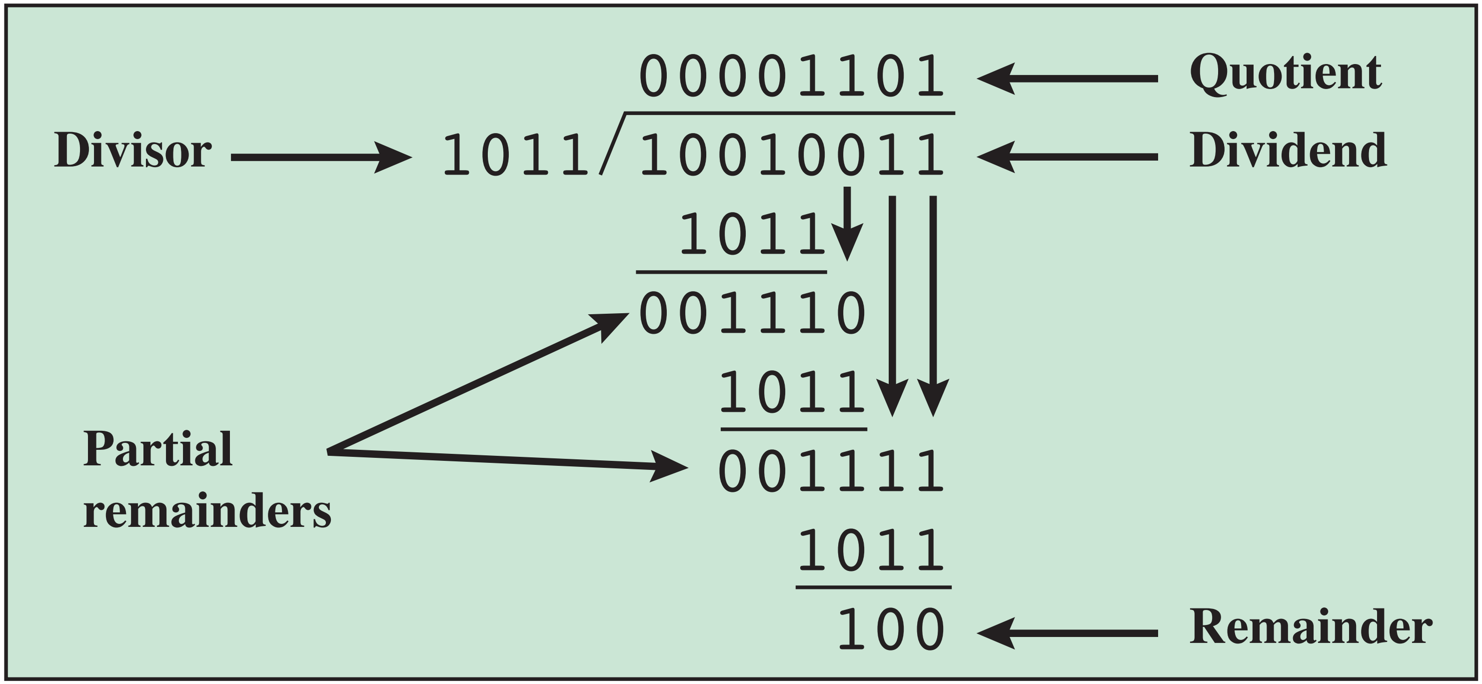


Figure 10.15 Example of Division of Unsigned Binary Integers

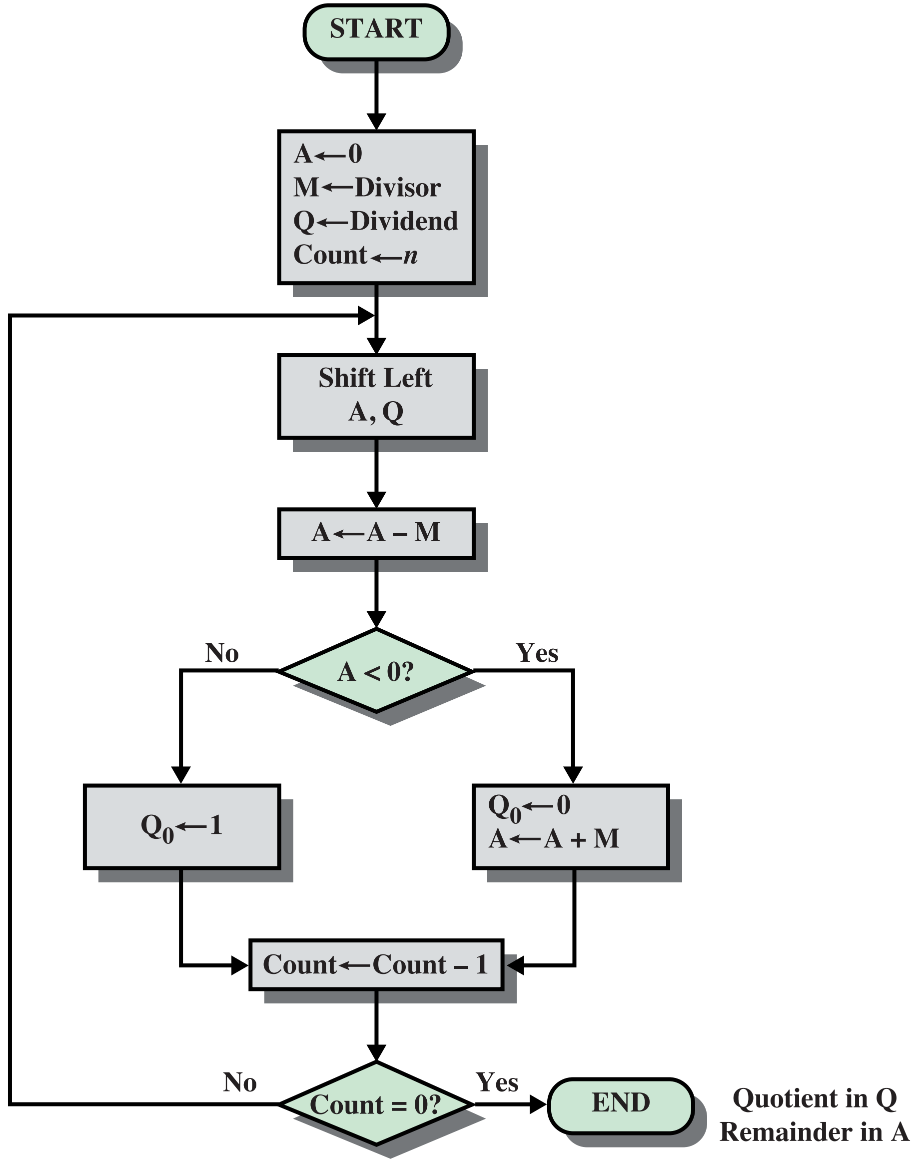


Figure 10.16 Flowchart for Unsigned Binary Division

A	Q	
0000	0111	Initial value
0000 1101 1101 0000	1110 1110	Shift Use twos complement of 0011 for subtraction Subtract Restore, set $Q_0 = 0$
0001 1101 1110 0001	1100 1100	Shift Subtract Restore, set $Q_0 = 0$
0011 1101 0000	1000 1001	Shift Subtract, set $Q_0 = 1$
0001 1101 1110 0001	0010 0010	Shift Subtract Restore, set $Q_0 = 0$

Figure 10.17 Example of Restoring Twos Complement Division (7/3)

Arithmetic: Basic Rules

- **Addition:**
  - Unsigned/signed: Normal addition followed by truncate, same operation on bit level
  - Unsigned: addition mod 2^w
    - Mathematical addition + possible subtraction of 2^w
  - Signed: modified addition mod 2^w (result in proper range)
    - Mathematical addition + possible addition or subtraction of 2^w

- **Multiplication:**
  - Unsigned/signed: Normal multiplication followed by truncate, same operation on bit level
  - Unsigned: multiplication mod 2^w
  - Signed: modified multiplication mod 2^w (result in proper range)

Why Should I Use Unsigned?

- ***Don't*** **use without understanding implications**
  - Easy to make mistakes

```
unsigned i;
for (i = cnt-2; i >= 0; i--)
  a[i] += a[i+1];
```

Why Should I Use Unsigned? (cont.)

- ***Do* Use When Performing Modular Arithmetic**
  - Multiprecision arithmetic
- ***Do* Use When Using Bits to Represent Sets**
  - Logical right shift, no sign extension