Lecture 3
MUL / DIV

CPS310

Computer Organization II

WINTER 2022

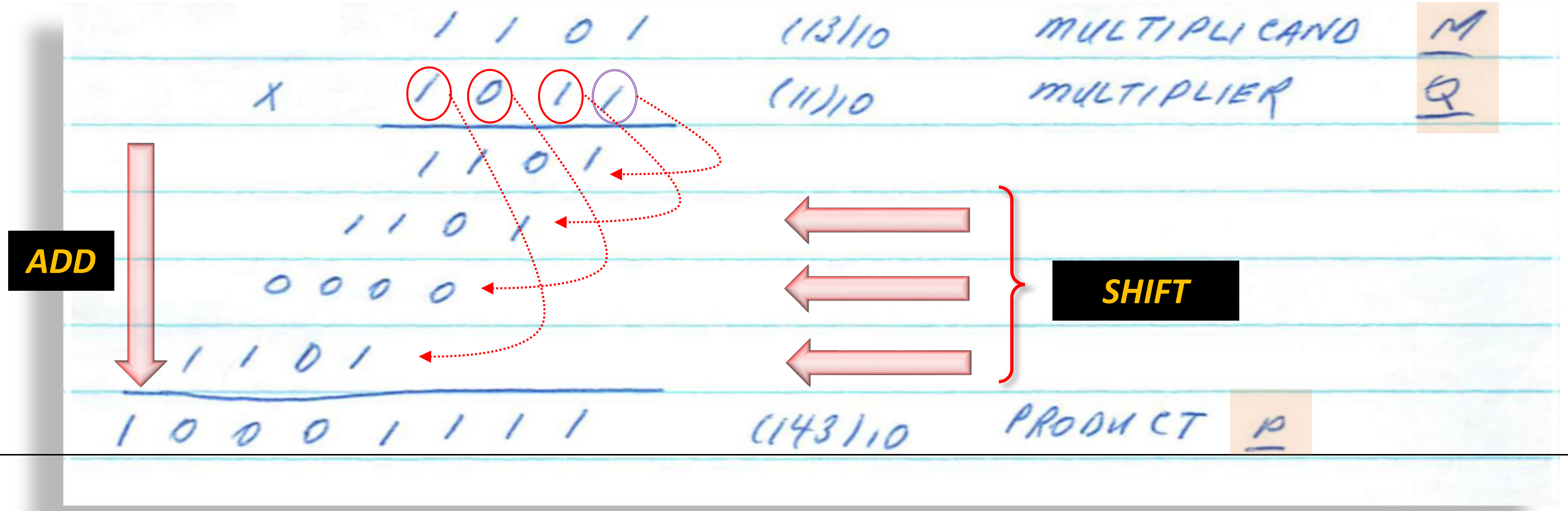
Multiplication/Division

Mul/div of fixed point can be done using:

- Addition
- Shift

Unsigned Multiplication

- Same method as used for the decimal numbers (*adding* & *shifting*)
- When we multiply **2 unsigned n-bit** numbers the result can be as large as **2n-bits**
- For **2 signed n-bit** numbers, the result can be as large as **2(n-1) +1** bits



MUL Hardware Implementation

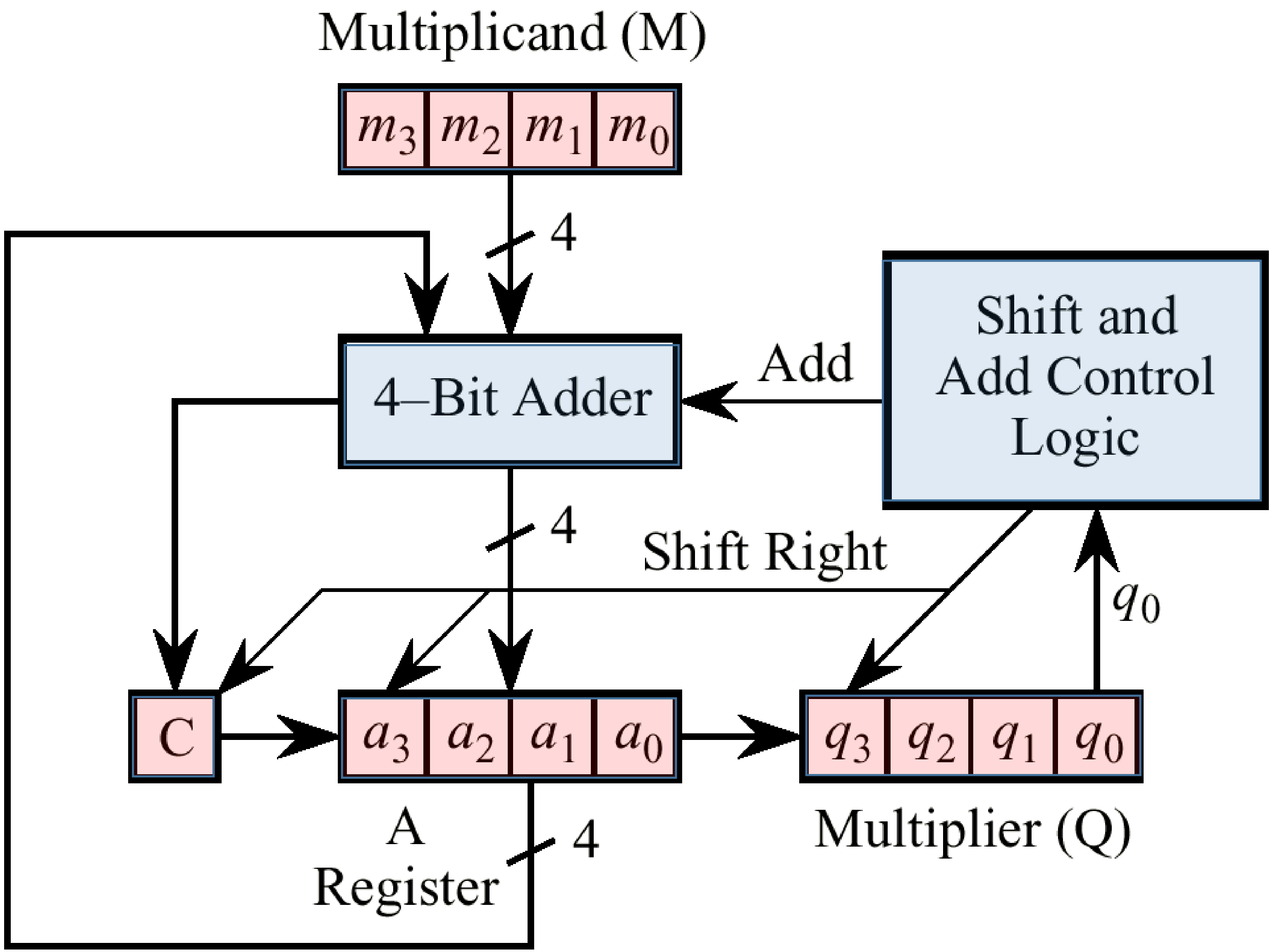
- (2) 4-bit numbers

Multiplicant

Multiplier

- (1) 4-bit adder
- (1) Control Unit
- (3) 4-bit shift register
  (**M**:$m_3m_2m_1m_0$, **Q**:$q_3q_2q_1q_0$, **A**:$a_3a_2a_1a_0$)
- (1) 1-bit register for carry (**C**)

4-bit Multiplication Hardware



M is Multiplicand (1101)
Q is Multiplier (1011)

To begin with
A is clear (0000)
C is clear (0)

Multiplicand
1 1 0 1

X 1 0 1 1
Multiplier

Multiplication Steps

0 TO BEGIN WITH CLEAR A, C
M: 1 1 0 1

1
A: 0000
M: 1101
0 1101

2
A: 0110
M: 1101
1 0011

4
A: 0100
M: 1101
1 0001

Multiplicand (M)
1 1 0 1
X 1 0 1 1
Multiplier (Q)

M is Multiplicand (1101)
Q is Multiplier (1011)

To begin with
A is clear (0000)
C is clear (0)

	C	A	Q	EXPLANATION
0	0	0 0 0 0	1 0 1 1	BEGIN
1	0	1 1 0 1	1 0 1 1	ADD M TO A
	0	0 1 1 0	1 1 0 1	SHIFT RIGHT
2	1	0 0 1 1	1 1 0 1	ADD M TO A
	0	1 0 0 1	1 1 1 0	SHIFT RIGHT
3	0	0 1 0 0	1 1 1 1	SHIFT (NO ADD)
4	1	0 0 0 1	1 1 1 1	ADD M TO A
	0	1 0 0 0	1 1 1 1	SHIFT RIGHT

Final Product

At each step look at Q_0

IF $Q_0 = 1$
ADD A to M & SHIFT

IF $Q_0 = 0$
SHIFT C A Q

Unsigned Division

- Division very similar to Multiplication

- In **Mul**, we shift the product to **right**
- In **Div**, we shift the quotient to **left**

- In **Mul**, we add
- In **Div**, we subtract

optional

Division

- (Dividend) ÷ (Divisor) = (Quotient)

- If 2 unsigned n-bit numbers are divided, the result can be max of n-bit

optional

DIVISION Hardware Implementation

- (1) 5-bit adder
- (2) control unit
- (1) 4-bit register for dividend (Q)
- (2) 5-bit register for divisor (M) and remainder (A)

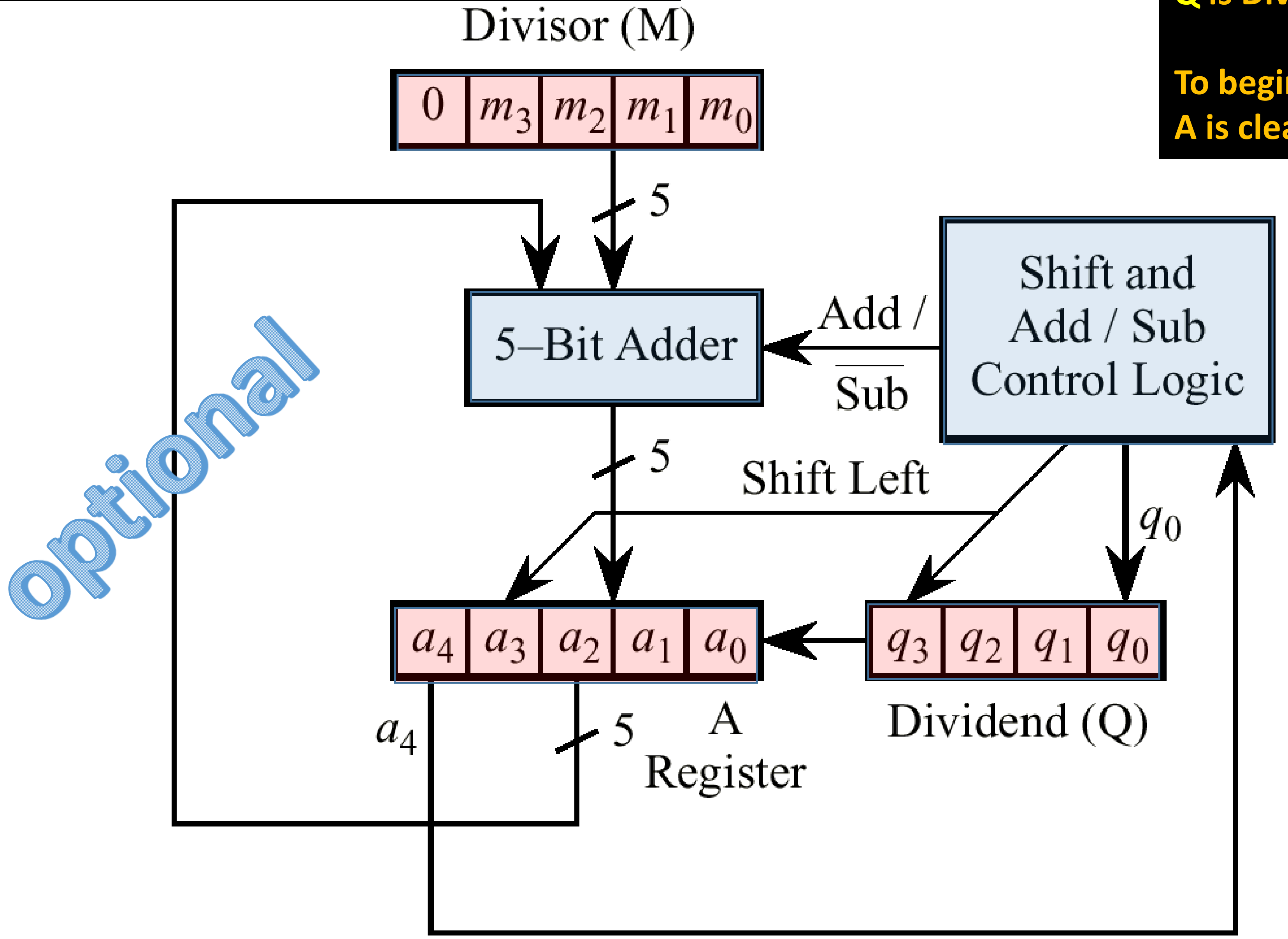
optional

- 5-bit registers are needed to look after the sign of the intermediate results
- Dividend is unsigned but subtraction may result in negative numbers

Hardware Implementation

M is Divisor (0011)
Q is Dividend (0111)

To begin with
A is clear (0000)



Division Steps

Optional

A	Q	Description
0 0 0 1 1	DIVISOR (M)	
0 0 0 0 0	0 1 1 1	Description
0 0 0 0 0	1 1 1 0	Shift A Q Left
1 1 1 0 1	1 1 1 0	Subtract M from A
0 0 0 0 0	1 1 1 0	Restore A
0 0 0 0 0	1 1 1 0	Clear Q_0 (based on A_4)
0 0 0 0 1	1 1 0 0	Shift A Q Left
1 1 1 1 0	1 1 0 0	Subtract M from A
0 0 0 0 1	1 1 0 0	Restore A
0 0 0 0 1	1 1 0 0	Clear Q_0 (based on A_4)
0 0 0 1 1	1 0 0 0	Shift A Q Left
0 0 0 0 0	1 0 0 0	Subtract M from A
0 0 0 0 0	1 0 0 1	Set Q_0 (based on A_4)
0 0 0 0 1	0 0 1 0	Shift A Q Left
1 1 1 1 0	0 0 1 0	Subtract M from A
0 0 0 0 1	0 0 1 0	Restore A
0 0 0 0 1	0 0 1 0	Clear Q_0 (based on A_4)

0010 QUOTIENT

DIVISOR 11 | 0111 DIVIDEND

$$0111 \div 11 = 0010$$

A: 00000
M: - 00011
00000
+ 11101
11101

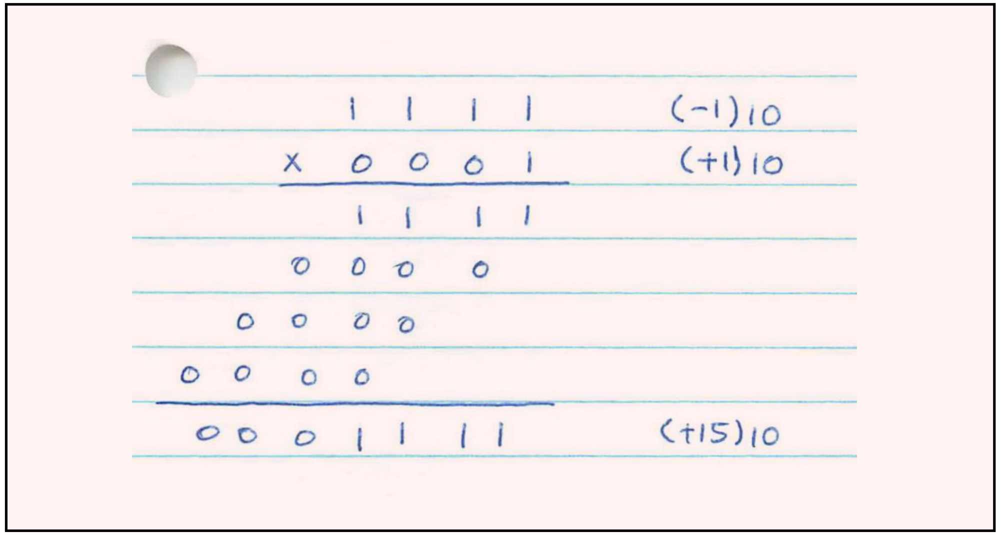
A: 00001
M: - 00011
00001
+ 11101
11110

A: 00011
M: - 00011
00011
+ 11101
00000

A: 00001
M: - 00011
00001
+ 11101
11110

Signed Multiplication

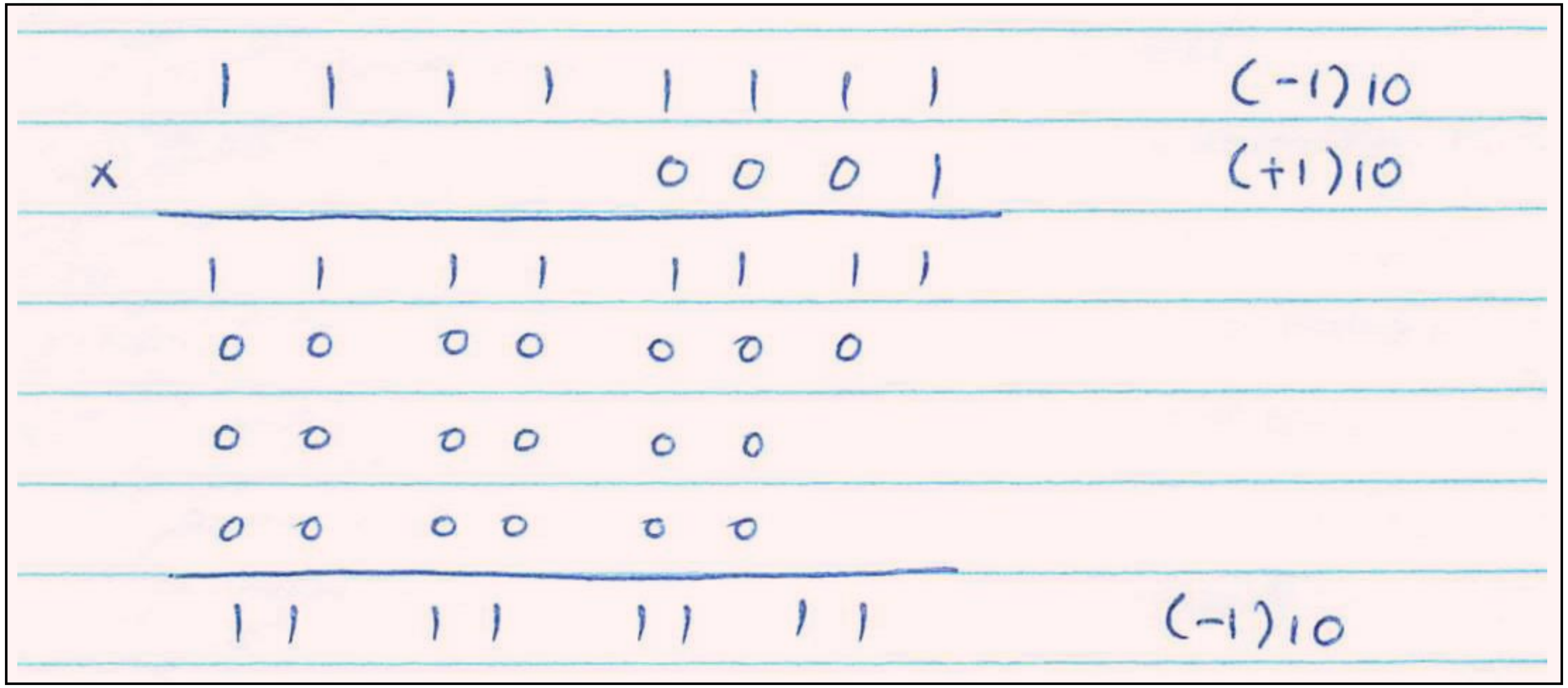
- If we use the same approach as the unsigned multiplication,
- OK with positive numbers
- But the result could be incorrect for a negative number



- Why wrong? Could not preserve the negative sign

Signed Multiplication

- Error only in case of a negative number
- The sign bit was not extended to the left of the result



- No discussion re signed division

Floating Point Math

ADD / SUB

- Make the exponents of 2 numbers equal
- Then add/sub mantissa

$$
\begin{array}{ll}
 & 0.101 \times 2^3 \\
+ & 0.111 \times 2^4 \\
\\
 & 0.0101 \times 2^4 \\
+ & 0.1110 \times 2^4 \\
\hline
 & 1.0011 \times 2^4
\end{array}
$$

Floating Point Math

MUL/DIV

- Calculate Sign, Exponent, Fraction separately

Sign: sign the same → +

sign different → -

Exponent: Add for mul

Sub for div

Fraction: mul / div

Floating Point Math – Multiplication Example

$(+\ 0.101 \times 2^{2}) \quad \times \quad (-\ 0.110 \times 2^{-3})$

$= -\ (\ 0.101 \times -\ 0.110\ ) \quad \times \quad (2^{2} \times 2^{-3})$

$= -\ (\ 0.01111) \quad \times \quad (2^{2\ -3})$

$= -\ 0.01111 \quad \times \quad 2^{-1}$

Floating Point Math – Division Example

$(+\ 0.110 \times 2^5)\ \ /\ \ (+\ 0.100 \times 2^4)$

$= +\ (\ 0.110\ /\ 0.100)\ \ \times\ \ (+\ 2^5\ /2^4)$

$= +\ (1.10)\ \ \times\ \ (2^{5-4})$

$= +\ 1.10\ \ \times\ \ 2^1$

Booth Algorithm

- To simplify the task of multiplication

- Works and treats positive and negative numbers uniformly

Booth Algorithm

Mechanism:

- Works on the principal that strings of **ones** or **zeros** in multiplier

  • do not require addition
  • just shift is needed

- Addition/subtraction only happens at the **boundaries** of the strings

Binary numbers – side note

$101 \times 2^0 =$ **1 0 1**

$101 \times 2^1 =$ **1 0 1 0** SHIFT LEFT BY 1 BIT

$101 \times 2^2 =$ **1 0 1 0 0** SHIFT LEFT BY 2 BITS

$101 \times 2^3 =$ **1 0 1 0 0 0** SHIFT LEFT BY 3 BITS

Booth Algorithm

	0	1	0	1	0	1	$(+21)_{10}$	←	Multiplicand
X	0	0	1	1	1	0	$(+14)_{10}$	←	Multiplier
		+ 1			- 1			←	Booth Record

0 to 1 transition is recorded as: -1

1 to 0 Transition is recorded as: +1

NOTE:

if the LSB of Multiplier is 1, assume a 0 to 1 transition, hence, -1 in the booth record

Booth Algorithm

Use the booth record & the Multiplicand to perform the multiplication

	5	4	3	2	1	0		
	0	1	0	1	0	1	$(+21)_{10}$	← Multiplicand
X	0	0	1	1	1	0	$(+14)_{10}$	← Multiplier
		- 1			- 1			← Booth Record

1 0 1 0 1 1 $(-21)_{10}$

$(-1 \times 21 \times 2^{1})_{10}$	1	1	1	1	1	1	0	1	0	1	1	0
$(+1 \times 21 \times 2^{4})_{10}$	0	0	0	1	0	1	0	1	0	0	0	0
$(294)_{10}$ (1)	0	0	0	1	0	0	1	0	0	1	1	0

Booth Algorithm

So the number of steps have been reduced

But is this always the case?

0	0	1	1	1	0	$\rightarrow$	$(+14)_{10}$	Multiplicand
0	1	0	1	0	1	$\rightarrow$	$(+21)_{10}$	Multiplier
+1	-1	+1	-1	+1	-1	$\leftarrow$	if the LSB of Multiplier is 1, assume a 0 to 1 transition, hence, -1 in the booth record	

Booth Algorithm

3 Addition, 3 Subtraction are required

$(-1 \times 14 \times 2^0)$	1	1	1	1	1	1	1	1	0	0	1	0
$(+1 \times 14 \times 2^1)$	0	0	0	0	0	0	0	1	1	1	0	0
$(-1 \times 14 \times 2^2)$	1	1	1	1	1	1	0	0	1	0	0	0
$(+1 \times 14 \times 2^3)$	0	0	0	0	0	1	1	1	0	0	0	0
$(-1 \times 14 \times 2^4)$	1	1	1	1	0	0	1	0	0	0	0	0
$(+1 \times 14 \times 2^5)$	0	0	0	1	1	1	0	0	0	0	0	0

$(+294)_{10}$	0	0	0	1	0	0	1	0	0	1	1	0

1 1 0 0 1 0 → $(-14)_{10}$