

## Lecture 21 (continued)

### Multiplying Negative Numbers

- Negative numbers in 2's complement.
- The *add-and-shift* algorithm does not work if we use negative numbers “as they are stored”.

Example:  $3_{10} * (-5)_{10} = (-15)_{10}$

In binary,  $3_{10} = 0011_2$  and  $(-5)_{10} = 1011_2$

Let the multiplicand  $M = 0011_2$

Product register:

0000 1011	initial value
0011 1011	Step 1. add
0001 1101	Step 1. shift
0100 1101	Step 2. add
0010 0110	Step 2. shift
0001 0011	Step 3. shift
0100 0011	Step 4. add
0010 0001	Step 4. shift

Here answer is  $(+33)_{10}$  and not  $(-15)_{10}$ .

- The algorithm did not work. Why?
- We started out with 4 bit numbers including the sign bit.
- The answer  $(-15)_{10}$  cannot be stored using 4 bits (sign included).
- Try  $(2)_{10} * (-3)_{10}$  at home. Here you will get  $(-6)_{10}$  as the correct answer in the lower 4 bits.
- For longer products, an easier way is to convert both numbers to positive values and then multiply.
- Treat the product as negative if one of the numbers as negative.
- There is a more elegant approach.
- Booth's algorithm will obtain the product in 2's complement for negative values (large & small) as well as for positive values.

## Booth's Algorithm

- Booth observed that consecutive 1's / consecutive 0's can be treated as a group.
- While processing a group, we only shift (no addition / subtraction).

Example:  $M * (30)_{10}$

$$M * (00011110)_2 = M * (2^4 + 2^3 + 2^2 + 2^1)$$

This also can be written as,

$$M * (30)_{10} = M * (32 - 2)_{10}$$

$$= M * (2^5 - 2^1)$$

$$= M * 2^5 - M * 2^1$$

-  $M * 2^1$  means shift M left and subtract.

+  $M * 2^5$  means shift M 5 times and add to product register.

- Examine the bit pattern of the multiplier:  $(00011110)_2$ .
- Scan from right to left.
- If there is transition 0 to 1, subtract & shift. A transition from 1 to 0 means add & shift.

Example:  $M * (122)_{10}$

Same as  $M * (01111010)_2$

$$= M * (2^6 + 2^5 + 2^4 + 2^3 + 2^1)$$

$$= M * (2^7 - 2^3 + 2^2 - 2^1)$$

Two transitions:  $0 \rightarrow 1$  (subtract) then  $1 \rightarrow 0$  (add)

01111010

Two transitions:  $0 \rightarrow 1$  (subtract) then  $1 \rightarrow 0$  (add)

- Observe that the total number of adds & subtracts are only 4 (not 5).
- $M * 2^7$  means shift the multiplicand  $M$  left 7 times and adding; this is the same as shifting the product right 7 times and then adding  $M$ .
- We have an ALU that can add or subtract as needed.
- Booth's algorithm goes right to left examining each bit in the **multiplier** and the bit immediately to the left of it. The multiplier is initially placed in the bottom half of the product register.
- One extra bit is added to the product register at the low end; it is initially set to 0.
- At every step we shift the product right by 1 bit and, depending on the 2 rightmost bits, we take the appropriate action: 00, 10, 01, 11.

**What if  $M$  is negative?**

- Then product register will have a negative value when we add  $M$  to it.
- Therefore, when we shift the product right, we use **arithmetic** shift in order to retain the sign bit.
- As with the *add-and-shift* algorithm, the algorithm requires 32 steps for 32-bit numbers.

	<u>Rightmost 2 bits</u>	<u>Action</u>
	00	→ just do arithmetic shift right
	11	→ just do arithmetic shift right
Sequence of 1s has ended	01	→ add $M$ & do arithmetic shift right
Sequence of 0s has ended	10	→ subtract $M$ & do arithmetic shift right

Example:  $3_{10} * (-5)_{10}$   
 $(0011)_2 * (1011)_2$

When we 'subtract' 3 ( $M$ ), we add 2's complement of 3; this is 1101.

Initial value	0 0 0 0 1 0 1 1 <span style="border: 1px solid black; padding: 0 2px;">0</span>	← extra bit
	$\begin{array}{r} 1\ 1\ 0\ 1 \\ \hline 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \end{array}$	10 means subtract & shift
Arithmetic shift →	1 1 1 0 1 1 0 <u>1 1</u>	after shift; 11 means just shift
	$\begin{array}{r} 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \\ \hline 0\ 0\ 1\ 1 \end{array}$	after shift; 01 means add & shift add
<b>Ignore carry out → 1</b>	0 0 1 0 0 1 1 0 1	
	$\begin{array}{r} 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0 \\ \hline 1\ 1\ 0\ 1 \\ \hline 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0 \end{array}$	after shift; 10 means subtract & shift subtract
$(-15)_{10}$ →	<span style="border: 1px solid black; padding: 0 2px;">1 1 1 1 0 0 0 1</span> 1	after shift

Answer in 2's complement.

Example:  $(-7)_{10} * (-3)_{10} = 21_{10}$

Initial value	0 0 0 0   1 1 0 1 <span style="border: 1px solid black; padding: 0 2px;">0</span>	← extra bit
	0 1 1 1	subtract -7 & shift
	<hr style="width: 100%; border: 0.5px solid black;"/>	
	0 1 1 1   1 1 0 1 0	after shift; 01 means add & shift
	0 0 1 1   1 1 1 <u>0</u> 1	add -7
	1 0 0 1	
	<hr style="width: 100%; border: 0.5px solid black;"/>	
	1 1 0 0   1 1 1 0 1	
	1 1 1 0   0 1 1 <u>1</u> 0	after shift; 10 means subtract & shift
	0 1 1 1	sub -7
	<hr style="width: 100%; border: 0.5px solid black;"/>	
<b>Ignore carry out</b> → 1	0 1 0 1   0 1 1 1 0	
	0 0 1 0   1 0 1 <u>1</u> 1	after shift; 11 means just shift
	<hr style="width: 100%; border: 0.5px solid black;"/>	
<b>Answer</b> =	<span style="border: 1px solid black; padding: 0 2px;">0 0 0 1 0 1 0 1</span> 1	after shift
	= $21_{10}$	

- Both are 4-bit numbers, including sign bit.  
 $(-7)_{10} = 1001_2$  (2's complement)  
 $(-3)_{10} = 1101_2$  (2's complement)  
 When we 'subtract'  $(-7)_{10}$ , we actually add  $7_{10}$ . 2's complement of 1001 is 0111.

### Comments

- Here we automatically get answers in the form we want.
- There is no conversion.
- Works for large products as well as small products
- At the time Booth's Algorithm was developed, shifting was much faster than addition / subtraction.
- For many bit patterns, Booth's Algorithm will perform fewer additions/subtractions than the *add-and-shift* algorithm; but for some bit patterns it will do more.
- It handles negative numbers very well, but that was not the main objective driving its development.