EE 3755

Computer Arithmetic

Handout # 2

EE [redacted] 3755 ① b

Fixed Point Multiplication

In this handout we present sequential algorithms for multiplying two fixed point numbers. Both cases of unsigned and signed-number multiplication are presented.

Ⓐ Unsigned—number multiplication

The sequential add/shift algorithm for unsigned multiplication

Consider two n-bit unsigned numbers $X = x_{n-1} x_{n-2} \cdots x_1 x_0$ and $Y = y_{n-1} y_{n-2} \cdots y_1 y_0$ with x_{n-1} and y_{n-1} being their most significant bits (MSBs) while x_0 and y_0 their least significant bits (LSBs). We are interested in computing the product $P = X \times Y$. The number X is called the multiplicand while Y is the multiplier. The product P will be a 2n-bit unsigned number.

Consider the following fields:

An <u>n-bit</u> field A (the multiplier field);
an <u>n-bit</u> field B (the field left of multiplier field);
a <u>1-bit</u> field c (the carry-out field);
an <u>n-bit</u> field M (the multiplicand field).

These fields involved in the multiplication are shown below

The sequential add/shift algorithm for unsigned multiplication is now as follows:

- <u>Initialization:</u> Initialize the field A with the multiplier (or $A \leftarrow Y$); initialize the field B with zeros (or $B \leftarrow 0, 0, \cdots, 0$); initialize field c with zero (or $c \leftarrow 0$); initialize field M with multiplicand (or $M \leftarrow X$).

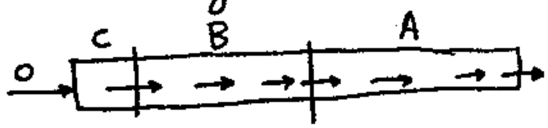
•• <u>The add/shift algorithm for unsigned multiplication:</u>

After initialization you have to perform n cycles of add/shift as follows:

— If the right most bit of field A is 1 (one) then do the following two:

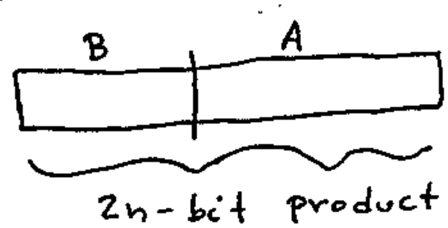
(i) $c, B \leftarrow B + M$ (carry out goes into c)

(ii) shift c, B, A one bit to the right with zero filling at left



— Else if right most bit of field A is 0 (zero) then only shift c, B, A one bit to the right with zero filling at left

After the completion of the n-th add/shift cycle the final $2n$-bit unsigned product is found in B, A or



Example 1: Using the add/shift algorithm for unsigned multiplication perform the multiplication with multiplier $= (6)_{10} = (0110)_2$, multiplicand $= (13)_{10} = (1101)_2$ and wordlength $n = 4$.

	C	B	A	
Initialization	0	0000	0110	$\rightarrow 0 \Rightarrow$ shift
result after 1^{st} shift	0	0000	0011	$\rightarrow 1 \Rightarrow$ add multiplicand and shift
+)		1101		
result of addition	0	1101	0011	
result after 2^{nd} shift	0	0110	1001	$\rightarrow 1 \Rightarrow$ add multiplicand and shift
+)		1101		
result of addition	1	0011	1001	
result after 3^{rd} shift	0	1001	1100	$\rightarrow 0 \Rightarrow$ shift
result after 4^{th} shift	0	0100	1110	

Final product

$= (01001110)_2 = (78)_{10}$

$= 13 \times 6$

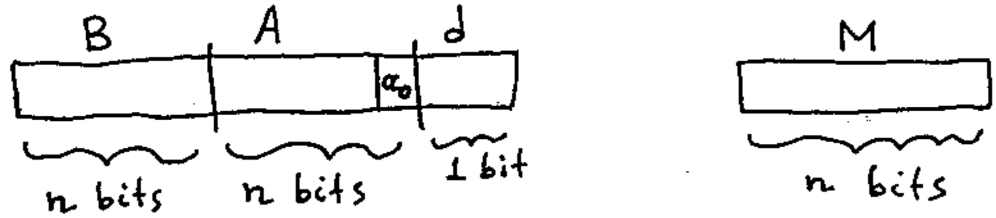
B Signed-number multiplication

Here, several efficient algorithms for multiplying signed numbers are presented. The system used for representing signed numbers is going to be the 2's complement system.

1). The sequential Booth algorithm for signed (2's complem.) multiplication (examining two bits at a time).

Consider two n-bit signed numbers (the 2's complement system is used for representing signed numbers). Let the two signed numbers be $X = x_{n-1} x_{n-2} \cdots x_1 x_0$ and $Y = y_{n-1} y_{n-2} \cdots y_1 y_0$ with x_{n-1} and y_{n-1} being their sign bits. We are interested in computing the product $P = X \times Y$. The number X is the multiplicand while Y is the multiplier. The product P will be a 2n-bit signed number.

Consider the following fields:
An n-bit field A (the multiplier field); an n-bit field B (the field left of the multiplier field); a 1-bit field d (the dummy field) which is right of the multiplier field; an n-bit field M (the multiplicand field). These fields involved in the signed multiplication are shown below



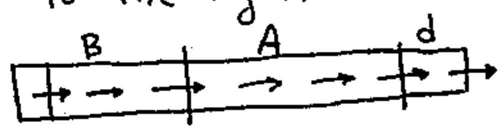
The sequential Booth algorithm for signed multiplication (examining two bits at a time) is now as follows:

- <u>Initialization</u>: Initialize the field A with the multiplier (or $A \leftarrow Y$); initialize the field B with zeros (or $B \leftarrow 0, 0, \ldots, 0$); initialize field d with zero (or $d \leftarrow 0$); initialize field M with multiplicand (or $M \leftarrow X$).

•• <u>The Booth algorithm (examining two bits at a time)</u>

After initialization you have to perform n cycles as follows. Call a_0 to be the right most bit of the field A. Then

— If $a_0, d = 0, 0$ or if $a_0, d = 1, 1$ then shift B, A, d one bit to the right with sign extension at the left

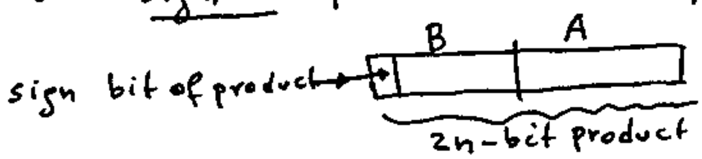


— If $a_0, d = 0, 1$ then do the following two:
(i) $B \leftarrow B + M$ (ignore carry out of addition)
(ii) shift B, A, d one bit to the right with sign extension at the left.

— Else if $a_0, d = 1, 0$ do the following two:
(i) $B \leftarrow B - M$ ($B - M$ means $B + (2\text{'s compl. of } M)$; ignore carry out of addition).

(ii) shift B, A, d one bit to the right with sign extension at the left.

After the completion of n such cycles the final 2n-bit <u>signed</u> product P is found in B, A or



Example 2: Using the Booth algorithm that relies on examining two bits at a time, perform the signed multiplication with multiplier $= (5)_{10} = (00101)_2$, multiplicand $= (-12)_{10} = (10100)_2$ and word length $n = 5$

	B	A	d	
Initialization	00000	00101	0	$\to$ 1,0 $\Rightarrow$ subtract multiplicand and shift
+)	01100			
result after subtraction	01100	00101	0	
result after 1^{st} shift	00110	00010	1	$\to$ 0,1 $\Rightarrow$ add multiplicand and then shift.
+)	10100			
result after addition	11010	00010	1	
result after 2^{nd} shift	11101	00001	0	$\to$ 1,0 $\Rightarrow$ subtr. multiplicand and then shift.
+)	01100			
result after subtr.	01001	00001	0	
result after 3^{rd} shift	00100	10000	1	$\to$ 0,1 $\Rightarrow$ add multiplicand and then shift
+)	10100			
result after addition	11000	10000	1	
result after 4^{th} shift	11100	01000	0	$\to$ 0,0 $\Rightarrow$ just shift
result after 5^{th} shift	11110	00100	0	

Final product $= (1111000100)_2$

$= (-60)_{10} = (-12) \times 5.$

2) The modified Booth algorithm.

Consider the Booth algoritm that has just been presented (the one that relies on examining two bits at a time). If for such a Booth algorithm we initialize the field d with one ($d \leftarrow 1$) and leave everything else unchanged, then the algorithm will be computing multiplicand × multiplier + multiplicand.

<u>Example 3</u>: Using the modified Booth compute multiplicand × multiplier + multiplicand where multiplicand $= (-12)_{10} = (10100)_2$, multiplier $= (5)_{10} = (00101)_2$ and $n = 5$.

	B	A	d	
Initialization	00000	00101	1	1,1 ⇒ just shift
result after 1st shift	00000	00010	1	0,1 ⇒ add multiplicand and then shift
+)	10100			
result after addition	10100	00010	1	
result after 2nd shift	11010	00001	0	1,0 ⇒ subtract multiplicand and then shift
+)	01100			
result after subtraction	00110	00001	0	
result after 3rd shift	00011	00000	1	0,1 ⇒ add multiplicand and then shift
+)	10100			
result after addition	10111	00000	1	
result after 4th shift	11011	10000	0	0,0 ⇒ just shift
result after 5th shift	11101	11000	0	

Final result $= (1110111000)_2 = (-72)_{10}$

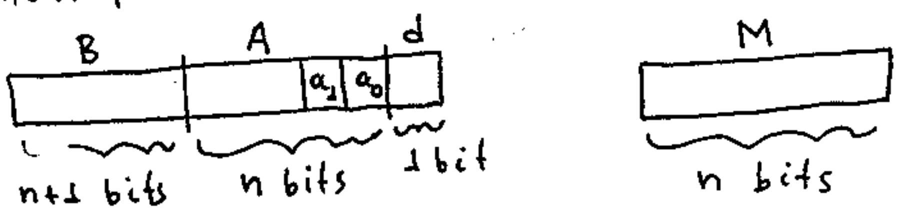
$= (-12) \times 5 + (-12)$

<u>Question</u>: Assuming that the original Booth algorithm works can you prove that the modified version also works?

3) The sequential Booth algorithm for signed (2's complem) multiplication (examining three bits at a time).

Consider again two n-bit signed numbers (2's complement system is used for representing signed numbers) the multiplicand $X = x_{n-1} x_{n-2} \cdots x_1 x_0$ and the multiplier $Y = y_{n-1} y_{n-2} \cdots y_1 y_0$ (x_{n-1} and y_{n-1} are the sign bits of X and Y). We are interested in computing the product $P = X \times Y$.

Consider the following fields:
An n-bit field A (the multiplier field); an (n+1)-bit field B (the field left of the multiplier field); a 1-bit field d (the dummy field) which is at the right of the multiplier field; an n-bit field M (the multiplicand field). These fields are shown below



The Booth algorithm for signed multiplication (examining three bits at a time) is now as follows:

- Initialization : Initialize the field A with the multiplier (or $A \leftarrow Y$); initialize the field B with zeros (or $B \leftarrow 0, 0, \cdots, 0$); initialize field d with zero (or $d \leftarrow 0$); initialize field M with multiplicand (or $M \leftarrow X$).

•• The Booth algorithm (examining three bits at a time)

Call a_1, a_0 to be the two right most bits of the multiplier field A.

After initialization you have to perform $\frac{n}{2}$ cycles as follows:

You will be examining the three bits a_1, a_0, d at the same time and do the following:

(i) $B \leftarrow B + k \times M$ (ignore carry out of addition; k is a function of a_1, a_0, d and is given by the table below.)

(ii) Following the above addition shift B, A, d two bits to the right with sign extension at the left.

After the completion of $\frac{n}{2}$ such cycles the product P is found in B, A.

The table below gives the value of k (k is the number of copies of the multiplicand to be added to B)

a_1	a_0	d	k
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	−2
1	0	1	−1
1	1	0	−1
1	1	1	0

Question: The just now presented algorithm relies on $\frac{n}{2}$ cycles which means that n must be even. What happens if n is odd?

Question: Can you prove that if the just now presented algorithm gets initialized with $d \leftarrow 1$ (everything else unchanged) it then computes multiplicand × multiplier + multiplicand?

Example 4: Using the Booth algorithm that relies on examining three bits at a time perform the multiplication with multiplicand $= (-3)_{10} = (111101)_2$, multiplier $= (29)_{10} = (011101)_2$ and $n = 6$.

	B	A	d
Initialization	0000000	011101	0

+) $\underbrace{1111101}_{1\times\text{multiplicand}}$ → 010 ⇒ $k = 1$ ⇒ add $1\times$multiplicand and then do 2-bit right shift

	B	A	d
result after addition	1111101	011101	0
result after 1^{st} shift	1111111	010111	0

+) $\underbrace{0000011}_{-1\times\text{multiplicand}}$ → 110 ⇒ $k = -1$ ⇒ add $-1\times$multiplicand and then do 2-bit right shift.

	B	A	d
result after addition	0000010	010111	0
result after 2^{nd} shift	0000000	100101	1

+) $\underbrace{1111010}_{2\times\text{multiplicand}}$ → 011 ⇒ $k = 2$ ⇒ add $2\times$multiplicand and then do 2-bit right shift

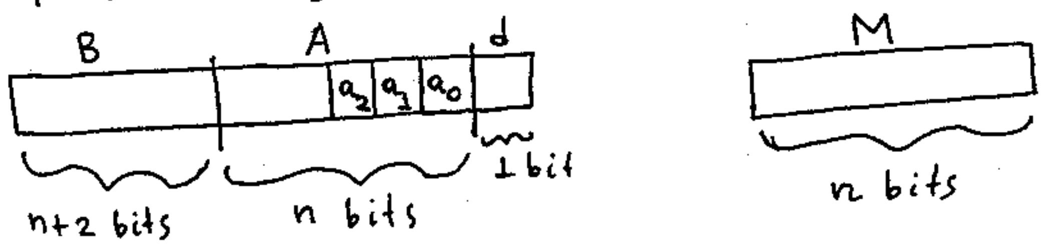
	B	A	d
result after addition	1111010	100101	1
result after 3^{rd} shift	1111110	101001	0

Final product $= (111111010100 1)_2$
$= (-87)_{10} = (-3) \times 29$.

4) The sequential Booth algorithm for signed (2's complement) multiplication (examining four bits at a time).

Again consider two n-bit signed numbers (2's complement system is used for representing signed numbers):
The multiplicand $X = x_{n-1} x_{n-2} \cdots x_1 x_0$ and the multiplier $Y = y_{n-1} y_{n-2} \cdots y_1 y_0$ (x_{n-1} and y_{n-1} are the sign bits of X and Y). We are again interested in computing the product $P = X \times Y$.

Consider the following fields:
An n-bit field A (the multiplier field); an (n+2)-bit field B (the field left of the multiplier field);
a 1-bit field d (the dummy field) which is at the right of the multiplier field; an n-bit field M (the multiplicand field). These fields are shown below



The Booth algorithm that relies on examining four bits at a time is now as follows:

- Initialization: Initialize the field A with the multiplier (or $A \leftarrow Y$); initialize the field B with zeros (or $B \leftarrow 0, 0, \ldots, 0$); initialize field d with zero (or $d \leftarrow 0$); initialize field M with multiplicand (or $M \leftarrow X$).

•• The Booth algorithm (examining four bits at a time)

Call a_2, a_1, a_0 to be the three right most bits of the multiplier field A.

After initialization you have to perform $\frac{n}{3}$ cycles as follows:

You will be examining the four bits a_2, a_1, a_0, d and do the following:

(i) $B \leftarrow B + k \times M$ (ignore carry out of addition; k is a function of a_2, a_1, a_0, d and is given by the table below.)

(ii) Following the above addition shift B, A, d three bits to the right with sign extension at the left.

After the completion of $\frac{n}{3}$ such cycles the product P is found in B, A.

The table below gives the value of k (k is the number of copies of the multiplicand to be added to B).

a_2	a_1	a_0	d	k
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	2
0	1	0	0	2
0	1	0	1	3
0	1	1	0	3
0	1	1	1	4
1	0	0	0	−4
1	0	0	1	−3
1	0	1	0	−3
1	0	1	1	−2
1	1	0	0	−2
1	1	0	1	−1
1	1	1	0	−1
1	1	1	1	0

- For this last algorithm n must be a multiple of 3
- Also, if this last algorithm gets initialized with $d \leftarrow 1$ (everything else unchanged) it then computes multiplicand × multiplier + multiplicand.