ECE 341

Lecture # 7

Instructor: Zeshan Chishti
zeshan@pdx.edu

October 20, 2014

Portland State University

Lecture Topics

- Multiplication of Unsigned Numbers
  - Sequential Circuit Multiplier
- Multiplication of Signed Numbers
  - Booth Algorithm
- Fast Multiplication
  - Bit-pair Recording of Multipliers
- <u>Reference:</u>
  - Chapter 9: Sections 9.3.2, 9.4, 9.5.1

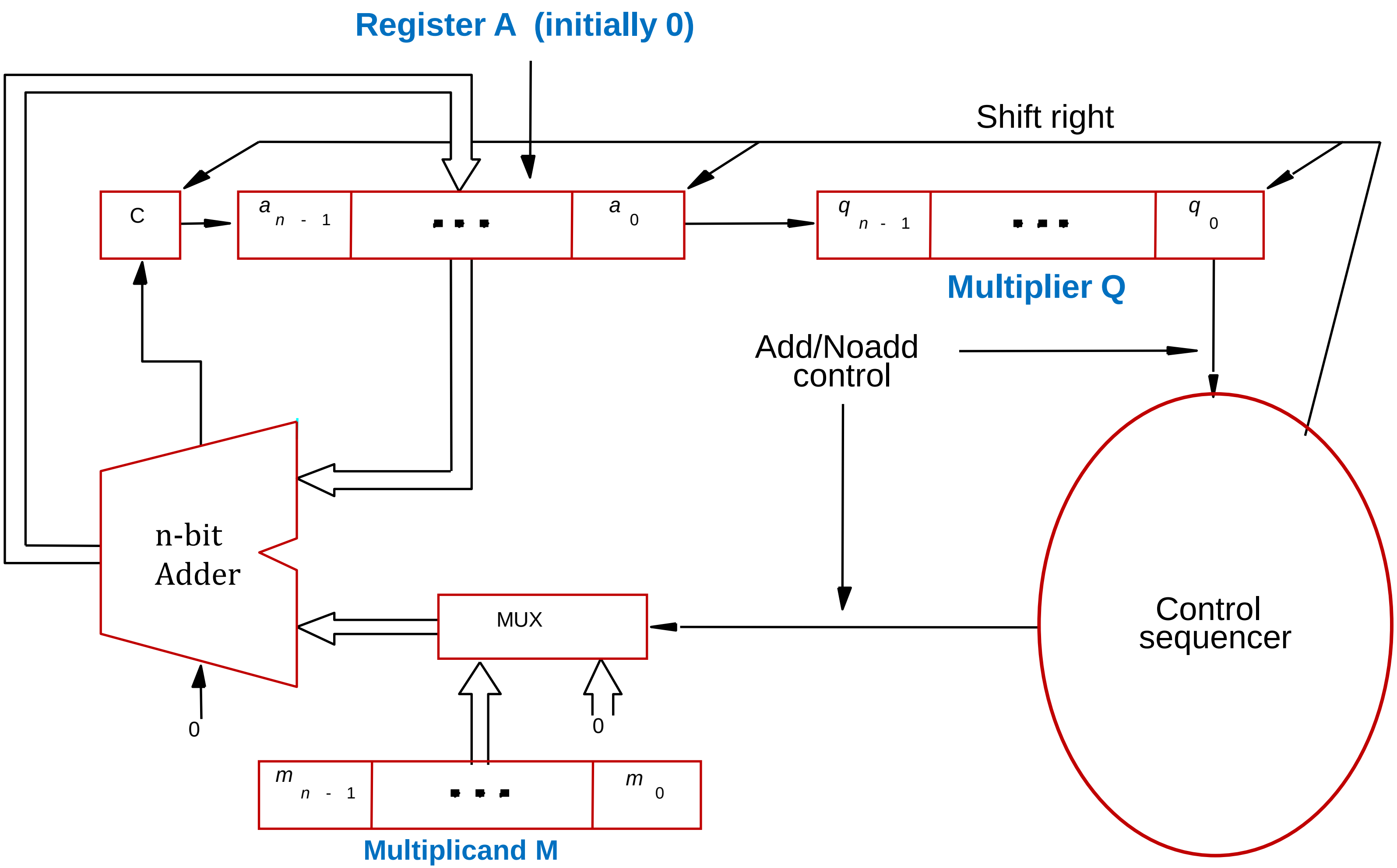
Sequential Multiplication

- Recall the rule for generating partial products:
  - If the i^{th} bit of the multiplier is 1, add the appropriately shifted multiplicand to the current partial product.
  - Multiplicand is shifted **left** when being added to the partial product

Key Observation:

- Adding a **left-shifted** multiplicand to an **unshifted** partial product is equivalent to adding an **unshifted** multiplicand to a **right-shifted** partial product

Sequential Circuit Multiplier



Sequential Multiplication Algorithm

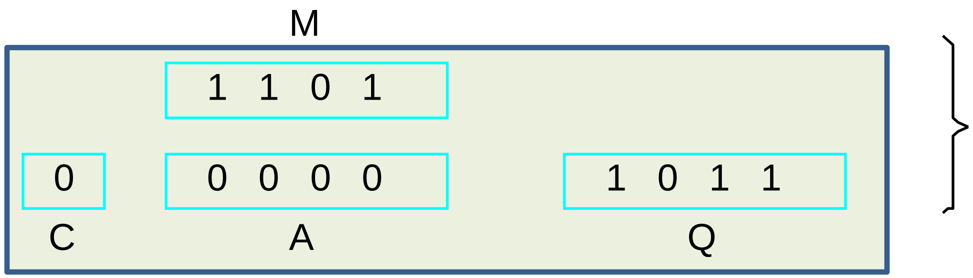
- Initialization:
  - Load multiplicand in "M" register, multiplier in "Q" register
  - Initialize "C" and "A" registers to all zeroes

- Repeat the following steps "n" times, where "n" is the number of bits in the multiplier
  - If (LSB of Q register == 1)

    A = A + M (carry-out goes to "C" register)
  - Treat the C, A and Q registers as one contiguous register and shift that register's contents right by one bit position

- After the completion of "n" steps
  - Register "A" contains high-order half of product
  - Register "Q" contains low-order half of product

Sequential Multiplication Example



Initial configuration

Sequential Multiplication Example

	M		
	1 1 0 1		
0	0 0 0 0	1 0 1 **1**	Initial configuration
C	A	Q	
0	1 1 0 1	1 0 1 1	**A += M**
0	0 1 1 0	1 1 0 1	**Shift**

First cycle

Sequential Multiplication Example

C	A	Q		
	M: 1 1 0 1			
0	0 0 0 0	1 0 1 1		Initial configuration
0	1 1 0 1	1 0 1 1	A += M	First cycle
0	0 1 1 0	1 1 0 **1**	Shift	
1	0 0 1 1	1 1 0 1	**A += M**	**Second cycle**
0	1 0 0 1	1 1 1 0	**Shift**	**Second cycle**

Sequential Multiplication Example

M

1 1 0 1

C	A	Q		
0	0 0 0 0	1 0 1 1		Initial configuration
0	1 1 0 1	1 0 1 1	A += M	First cycle
0	0 1 1 0	1 1 0 1	Shift	
1	0 0 1 1	1 1 0 1	A += M	Second cycle
0	1 0 0 1	1 1 1 **0**	Shift	
0	1 0 0 1	1 1 1 0	**No add**	**Third cycle**
0	0 1 0 0	1 1 1 1	**Shift**	

Sequential Multiplication Example

C	A	Q		
	M: 1 1 0 1			Initial configuration
0	0 0 0 0	1 0 1 1		
0	1 1 0 1	1 0 1 1	A += M	First cycle
0	0 1 1 0	1 1 0 1	Shift	
1	0 0 1 1	1 1 0 1	A += M	Second cycle
0	1 0 0 1	1 1 1 0	Shift	
0	1 0 0 1	1 1 1 0	No add	Third cycle
0	0 1 0 0	1 1 1 **1**	Shift	
1	0 0 0 1	1 1 1 1	**A += M**	**Fourth cycle**
0	1 0 0 0	1 1 1 1	**Shift**	

Product

Signed Multiplication

Signed Multiplication

- Considering 2's-complement signed operands, what will happen to $(-13)\times(+11)$ if following the same method of unsigned multiplication?

Sign extension is shown in blue

					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)

We must extend sign-bit value of multiplicand to left as far as product will extend

Signed Multiplication (cont.)

- If the multiplier is +ve:
  - The unsigned multiplication hardware works fine as long as it is augmented to provide for sign extension of partial products
- If the multiplier is –ve:
  - Form the 2's-complement of both the multiplier and the multiplicand and proceed as in the case of a +ve multiplier
  - This is possible because complementation of both operands does not change the value or the sign of the product

- A technique that works equally well for both negative and positive multipliers – **Booth algorithm**

Booth Algorithm

- Booth algorithm treats both positive and negative 2's complement operands uniformly
- To understand Booth algorithm:
  - Consider a multiplication scenario, where the multiplier has a single block of 1s, for example, 0011110. How many appropriately shifted versions of the multiplicand are typically added to derive the product?

							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

We need as many additions as the number of 1s

Booth Algorithm (cont.)

- Since 0011110 = 0100000 – 0000010, if we use the expression to the right, what will happen?

							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

We need only 2 additions

Booth Algorithm (cont.)

- In general, in the Booth scheme, -1 times the shifted multiplicand is selected when moving from 0 to 1, and +1 times the shifted multiplicand is selected when moving from 1 to 0, as the multiplier is scanned from right to left

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 recoding of a multiplier

Booth Algorithm Example for Negative Multiplier

	0	1	1	0	1	(+13)
X	1	1	0	1	0	(- 6)

					0	1	1	0	1	
					0	-1	+1	-1	0	
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					
1	1	1	0	1	1	0	0	1	0	(- 78)

Works fine as long as we do appropriate sign extension for the negative summands (versions of the multiplicand)

Booth Multiplier Recording Table

Multiplier		Version of multiplicand selected by bit
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 Algorithm Efficiency

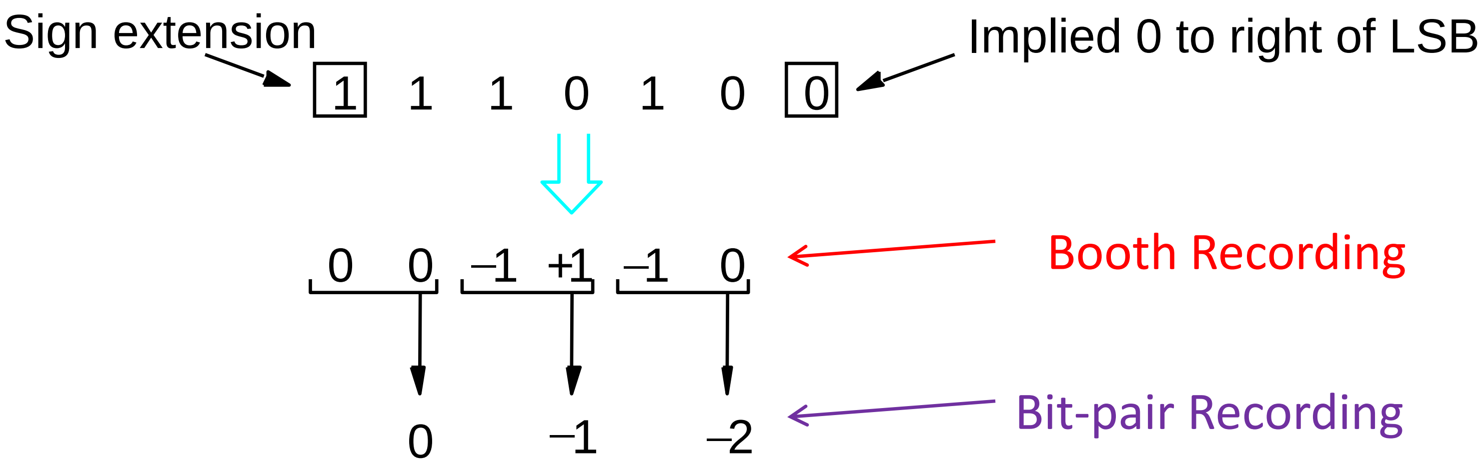
- Worst case: 0's and 1's are alternating => *n* summands
- Best case: a few long strings of 1's (skipping over 1s) => small no. of summands

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	0	1	1	1	1	1	0	0	0	0	1	1	1
	0	0	0	+1	0	0	0	0	-1	0	0	0	+1	0	0	-1

Fast Multiplication

Bit-Pair Recoding of Multipliers

- For each pair of bits in the multiplier, we require at most one summand to be added to the partial product
- For *n*-bit operands, it is *guaranteed* that the max. number of summands to be added is *n*/2



Example of bit-pair recoding derived from Booth recoding

Bit-Pair Recoding of Multipliers (cont.)

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	0 X M
0	0	1	+1 X M
0	1	0	+1 X M
0	1	1	+2 X M
1	0	0	−2 X M
1	0	1	−1 X M
1	1	0	−1 X M
1	1	1	0 X M

Table of multiplicand selection decisions

Example

```
  0 1 1 0 1  (+13)
× 1 1 0 1 0  (-6)
-------------
```

```
                    0  1  1  0  1
                    0 -1 +1 -1  0
                 ----------------
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
-----------------------------
1  1  1  0  1  1  0  0  1  0  (-78)
```

Booth Recording results in 3 summands

```
                    0  1  1  0  1
                    0    -1    -2
                 ----------------
1  1  1  1  1  0  0  1  1  0
1  1  1  1  0  0  1  1
0  0  0  0  0  0
-----------------------------
1  1  1  0  1  1  0  0  1  0
```

Bit-Pair Recording results in 2 summands