Homework

□ Reading and homework:

- Chapter 13
- Homework due 10/15 (later than usual you're welcome!)
- \Box Homework keys posted soon.
- □ Don't forget: Prelim I on 10/12!

Copyright \odot 2002–2018 UMaine Computer Science Department – 1 / 32

COS 140: Foundations of Computer Science

Booth's Algorithm

Fall 2018

Problem	3
Number Representation	5
Sign-Magnitude Representation Overview Example Examples # bits Changing size Problems	7 8 9 10
Two's Complement Representation Overview Overview Overview Computing Examples Zero? Extending size Addition Subtraction Overflow Overflow	13 15 16 17 18 19
Basis of Booth's Algorithm Image: State of Booth's Algorithm Image: State of Booth's Algorithm Multiplication Image: State of Booth's Im	22 23 24
Booth's Algorithm Overview Registers used Operations used Overview of algorithm The algorithm.	27 28 29

mples

Problem

The problem

- $\hfill\square$ We know how to do addition in the computer...
- $\hfill\square$...but what about multiplication?
- $\hfill\square$ \quad For $n\times m\text{, could just add }n$ to itself m times
- $\Box \quad \mathsf{But can} \longrightarrow \mathsf{lot of additions!}$

E.g.: $2,999,111 \times 1,999,999,999$

 $\hfill\square$ Can we do better?

Copyright © 2002–2018 UMaine Computer Science Department – 3 / 32

The problem

- □ Booth's algorithm: algorithm for multiplication that:
 - Uses mathematics insights $\Rightarrow \downarrow \downarrow \#$ additions
 - Can be implemented in hardware
- □ First: need to understand how to represent numbers in the computer

Copyright © 2002-2018 UMaine Computer Science Department - 4 / 32

Number Representation

Numbers

 $\hfill\square$ Here: focus only on integers – floating point numbers in later class/courses

- $\hfill\square$ Many different ways have been tried
 - E.g., binary coded decimal (BCD)

 $1346 = 0001 \ 0011 \ 0100 \ 0110$

□ This class: look at most common:

- Sign-magnitude representation
- Two's complement representation

Copyright © 2002–2018 UMaine Computer Science Department – 5 / 32

5 / 32

Sign-Magnitude Representation of Numbers

- \Box Two parts to represent number *n*:
 - Sign bit:
 - ▷ leftmost (high-order) bit
 - $_{\triangleright}$ 1 = negative, 0 = positive
 - Magnitude:
 - ▶ Remaining bits
 - $\mathbf{r} = |n|$

Copyright \bigodot 2002–2018 UMaine Computer Science Department – 6 / 32

Example: Sign-Magnitude Representation

Represent 12 in 8-bit sign-magnitude representation:

- $\hfill\square$ 12 in binary is 1100
- $\hfill\square$ The sign of 12 is positive, so represented as 0.
- \Box Representation of 12: 0000 1100

Copyright © 2002–2018 UMaine Computer Science Department – 7 / 32

Example: Sign-Magnitude Representation

Represent -12 in 8-bit sign-magnitude representation:

- \Box 12 in binary is 1100 (0000 1100 in 8 bits)
- $\hfill\square$ The sign of -12 is negative, so represented as 1.
- $\hfill\square$ Representation of -12: 1000 1100

Copyright © 2002–2018 UMaine Computer Science Department - 8 / 32

Number of Bits in Representation

- □ Each computer/OS/language: several integer representations
- \Box Differ by length (# of bits)
- $\hfill\square$ \hfill Need to know how to change size of representation

Copyright © 2002–2018 UMaine Computer Science Department – 9 / 32

Changing size of representation

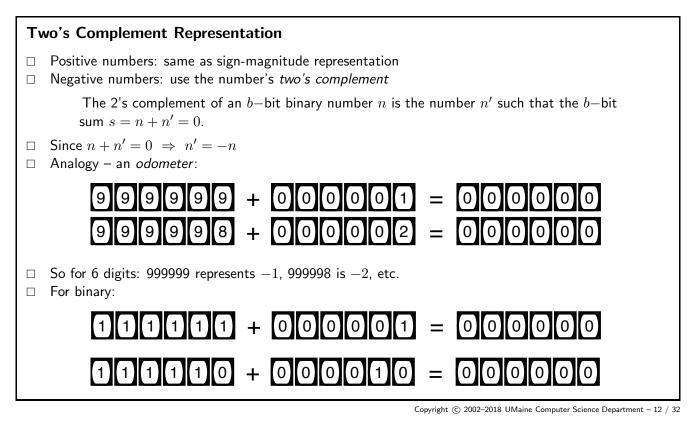
- \Box Smaller $s \rightarrow \text{larger } l$:
 - Sign bit of l = sign bit of s
 - Magnitude of l = magnitude of s
 - Will need to pad with 0s to left
 - E.g., extend $1001 \ 1010_2 \ (-26_{10})$ to 16 bits: 1000 0000 0001 1010
- \Box Larger $l \rightarrow$ smaller s:
 - Same idea
 - Will have to *truncate* bits on left
 - What if number too large?

Copyright © 2002–2018 UMaine Computer Science Department – 10 / 32

Problems with Sign Magnitude Representation

- \Box *Two* ways to represent 0!
- $\hfill\square$ Operations need to take sign bit into account
- $\hfill\square$ Need both addition and subtraction logic

Copyright © 2002–2018 UMaine Computer Science Department – 11 / 32



Finding 2's complement One way (e.g., 4-digit numbers): n + n' = (1)0000 $\Rightarrow n' = (1)0000 - n$ E.g., 10's complement of 0004 = 10000 - 4 = 9996 E.g., 2's complement of $0010_2 = 10000 - 10 = 1110$ Want a more efficient way For 4-digit 10's complement: What number can I add to n to get 9999? Find that, add 1, should be the 10's complement E.g., 10's complement of 0235 = 9999 - 235 + 1 = 9764 + 1 = 9765 Does this work? Yes: 0235 + 9765 = 10000 which is a 4-digit 0. Will it work for binary? And can we do it efficiently?

Copyright \odot 2002–2018 UMaine Computer Science Department – 13 / 32

Finding 2's complement

- □ Problem: How to *efficiently* find the 2's complement?
- $\hfill\square$ Example: find 8-bit 2's complement of $1000\,1100$
 - First find number I can add to give 11111111, then add 1
 - $1000 \ 1100 + 0111 \ 0011 = 1111 \ 1111$, so...
 - ...01110011 + 1 = 01110100 = 2's complement of 10001100
- \Box In example, 01110011 is the 1's complement of 10001100
- □ Easy (efficient) to find: *bitwise negation* of number
- \Box So to find 2's complement of *n*:
 - 1. Do bitwise negation of n.
 - 2. Add 1.
- $\hfill\square$ Both are easy for hardware or software
- $\hfill\square$ Note that leftmost bit still denotes sign

Copyright © 2002–2018 UMaine Computer Science Department – 14 / 32

Examples

```
\hfill\square Problem: Represent 37 in 8-bit 2's complement form
```

- Convert to binary: 00100101
- It's positive \Rightarrow done.
- $\hfill\square$ Problem: Represent -37 in 8-bit 2's complement form
 - Negative \Rightarrow find 8-bit 2's complement of 37
 - Convert 37 to binary: 00100101
 - Find 1's complement: 11011010
 - Add 1: 11011010 + 1 = 11011011
- \Box Is this correct?
 - If 1101 1011 is -n, then n = -(-n) = -(1101 1011)
 - So find 2's complement of 11011011
 - 2's complement = $00100100 + 1 = 00100101 = 37_{10}$
 - So $1101\ 1011$ represents -37_{10} .

Copyright © 2002–2018 UMaine Computer Science Department - 15 / 32

What about 0?

- □ Let's look at 8-bit 2's complement 0:
 - 1's complement: 11111111
 - 8-bit 2's complement: 11111111 + 00000001 = 00000000
- \Box ... only one representation for 0.

Copyright © 2002–2018 UMaine Computer Science Department – 16 / 32

Extending Two's Complement to More Bits						
\Box Pad highest order bits with the sign bit.						
- Extend our representation of 12 to 16 bits:						
$0000\ 1100 \Rightarrow 0000\ 0000\ 0000\ 1100$						
– Extend our representation of -12 to 16 bits:						
$1111\ 0100 \Rightarrow 1111\ 1111\ 0100$						
Check it: 0000 0000 0000 1011 - one's complement 0000 0000 0000 1100 = 12						
□ When create initial representation, make sure have enough bits to have correct sign bit.						

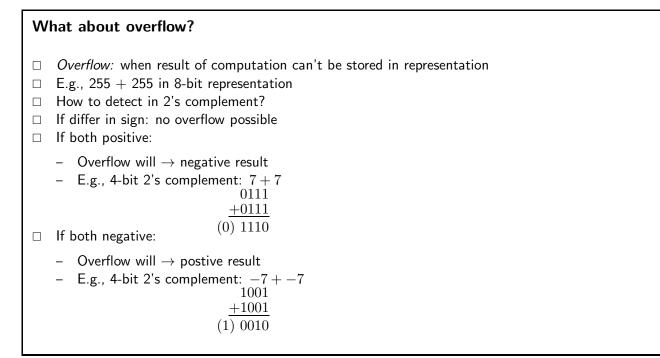
Copyright © 2002–2018 UMaine Computer Science Department – 17 / 32

Т١	Two's complement addition					
	Simple: just add the two number, whether they're positive <i>or</i> negative! E.g., two positive numbers, 4-bit representation: $4 + 3$ 0100 +0011					
	(0) 0111					
	E.g., positive and negative, 4-bit representation: 4 + -3 $\begin{array}{c} 0100\\ \underline{+1101}\\ (1)\ 0001\end{array}$					
	E.g., Two negative numbers, 4-bit representation: -4 + -3 1100 $+1101$ (1) 1001					

Copyright \bigcirc 2002–2018 UMaine Computer Science Department – 18 / 32

Two's complement subtraction \Box Simple: just negate the subtrahend and add to the minuend \Box E.g., what is 4 - 3 in 4-bit 2's complement arithmetic?0100+1101(1) 0001

Copyright © 2002–2018 UMaine Computer Science Department – 19 / 32



Copyright © 2002–2018 UMaine Computer Science Department - 20 / 32

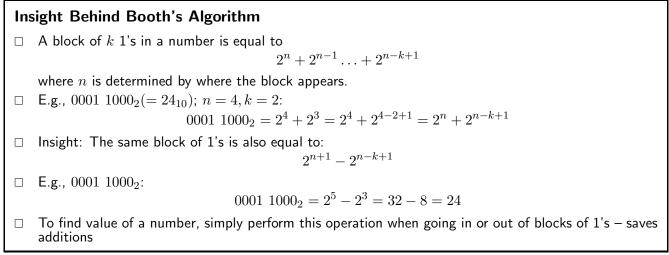
21 / 32

Basis of Booth's Algorithm

Long-hand Multiplication

- □ From elementary school...
- $\hfill\square$ For each digit in the multiplier:
 - Start creating partial product in the proper column.
 - Multiply each digit in the multiplicand to form a partial product.
 - Add all the partial products together (with each being in its proper columns).
- □ Intuition for multiplication of unsigned numbers. Sped up by fact that can only use 1's (add the multiplicand and shift to next column) and 0's (shift to next column).
- $\hfill\square$ We would like to not do so many additions!

Copyright © 2002–2018 UMaine Computer Science Department - 21 / 32



Copyright © 2002–2018 UMaine Computer Science Department - 22 / 32

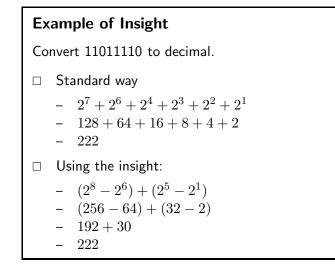
Why Does This Work?

 \Box If you think about it, adding 2^{n-k+1} to the number is the same as adding a number with only a 1 in that position, which is guaranteed to give a new number with a 1 in 2^{n+1} and 0's where the 1's were:

$$\begin{array}{r}
 011100 \\
 +000100 \\
 100000
 \end{array}$$

- \Box Let the first number be x, with k = 3 1s and n = 4
- \Box The second number is 2^{n-k+1}
- \Box The sum is 2^{n+1}
- \Box So: $x + 2^{n-k+1} = 2^{n+1}$
- \Box So: $x = 2^{n+1} 2^{n-k+1}$

Copyright © 2002–2018 UMaine Computer Science Department – 23 / 32



Copyright © 2002–2018 UMaine Computer Science Department – 24 / 32

Multiplication using the insight Suppose we have a number such as 0110₂ we wish to multiply by another number, say 0010₂ We know that: 0110₂ = 2³ - 2¹ So, multiplying both sides by 0010₂ gives: 0110₂ × 0010₂ = (2³ - 2¹) × 0010₂ Which can be rewritten as: 2³ × 0010₂ - 2¹ × 0010₂ This can save additions (subtractions): Old way of multiplication: addition for each 1 in multiplier This way: need 1 subtraction for each group of 1s, addition for the partial sums (differences) Also need way to multiply by powers of two: just shifting

Copyright \bigodot 2002–2018 UMaine Computer Science Department – 25 / 32

Booth's Algorithm

- □ Booth's algorithm is just the implementation of this insight, with some clever optimizations
- \Box Requires:
 - way to keep track of which bit we're on
 - way to keep track of beginning, end of sequence of 1's
 - way to form 2's complement
 - way to *shift* over multiplicand for adding
 - way to add
 - holder for product

Copyright © 2002–2018 UMaine Computer Science Department – 26 / 32

Registers Used by Booth's Algorithm

Assuming n-bit numbers:

Register	<u>Size</u>	Description						
Q	n	Initially holds multiplier, ultimately						
		holds low-order n bits of product						
Q-1	1	Holds previous low-order bit of Q – lets						
		us tell if block of 1's has started/stopped						
М	n	Multiplicand						
А	n	Holds high-order portion of result						
Count	_	Holds the number of bits in the multi-						
		plicand/multiplier						

Copyright © 2002–2018 UMaine Computer Science Department - 27 / 32

Operations Used by Booth's Algorithm

Arithmetic Shift: a fast machine operation which moves all bits over one position and repeats the sign bit (most significant bit) in the newly open position.

Compare: a fast machine instruction that checks to see if two bytes or words are the same **Add:** a fast machine instruction that adds two numbers together

Complement: a fast machine instruction that gives the complement of all bits (some machines may have a machine instruction for two's complement)

Copyright © 2002–2018 UMaine Computer Science Department – 28 / 32

Booth's Algorithm Overview

- $\hfill\square$ Will start with low-order bits of product in A (0s), multiplier in Q
- $\hfill\square$ For each digit seen, regardless of what it is or what was seen before, shift once it has been handled.
 - This is equivalant to moving partial products over in long multiplication.
 - Instead of shifting multiplicand left before adding, we'll shift product (and multiplier) right product shifts into Q over time, multiplier shifts out.
- $\hfill\square$ When enter a group of 1's from the right, subtract the multiplicand from the accumulating product.
- $\hfill\square$ When leave a group of 1's from the right, add the multiplicand to the accumulating product.
- $\hfill\square$ The last two steps apply the basic insight, multiplied by the multiplicand.

Copyright © 2002–2018 UMaine Computer Science Department – 29 / 32

Booth's Algorithm 1. Initialize registers with proper information. Count is the number of bits, A is 0, and Q-1 is 0. Q is the multiplier.

- 2. Compare the least significant bit of Q and Q-1 to see if entering or leaving a group of 1's:
 - (a) If the least significant bit of Q is 1 and Q-1 is 0, subtract M from A.
 - (b) If the least significant bit of Q is 0 and Q-1 is 1, add M to A.
 - (c) Otherwise, do nothing.
- 3. Prepare for next bit.
 - (a) Arithmetic shift right A, Q, Q-1. (Shift along these registers as though they were one continuous register.)
 - (b) Reduce the count by 1.
 - (c) If count is 0 end. Result is in AQ. Otherwise, go to step 2.

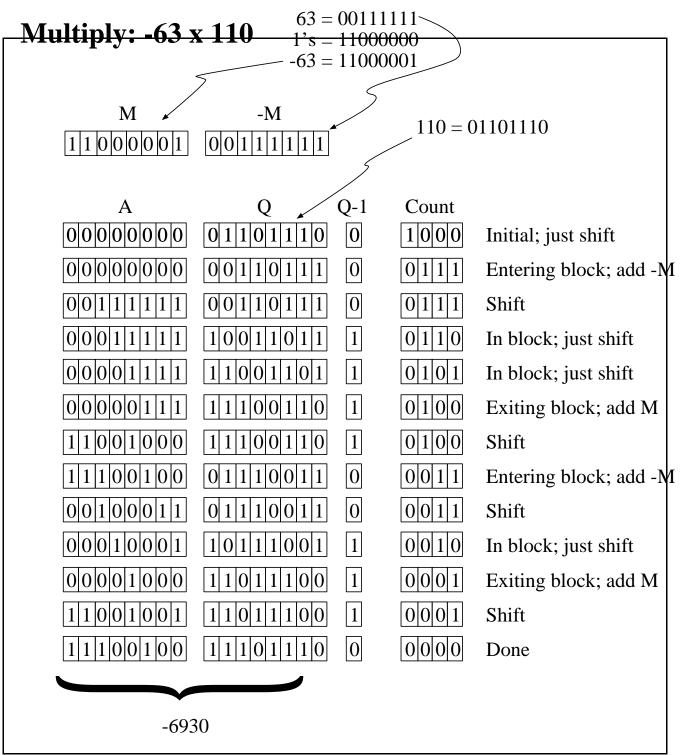
Copyright \odot 2002–2018 UMaine Computer Science Department – 30 / 32

Example

2 times 7, using 4 bit numbers. Multiplier (Q) is 0111. Multiplicand (M) is 0010. Two's complement of multiplicand: 1110.

A	Q	Q-1	С	
0000	0111	0	4	Initialize; 1-0
+1110				Subtract M from A (add -2)
1110	0111	0	4	Now shift
1111	0011	1	3	Now shift
1111	1001	1	2	Now shift
1111	1100	1	1	0-1
+0010				Add M to A (add 2)
0001	1100	1	1	Now shift
0000	1110	0	0	Done: answer $= 14$

Copyright © 2002-2018 UMaine Computer Science Department - 31 / 32



Copyright © 2002–2018 UMaine Computer Science Department - 32 / 32