COMP2611: Computer Organization

Booth Algorithm and Division

Booth Algorithm and Division

Introduction of Booth algorithm

- Examples

Division

- Examples

Exercises

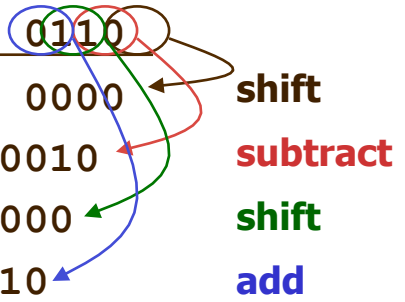
Booth's Algorithm for Signed Multiplication

- If the multiplicand or multiplier is negative, we first negate it to get a positive number
- Use any one of the existing methods to compute the product of two positive numbers
- The product should be negated if the original signs of the operands disagree
- **Booth's algorithm**: a more efficient and elegant algorithm for the multiplication of signed numbers

Motivation behind Booth Algorithm

- Let's consider multiplying 0010_2 and 0110_2

	Convention		Booth	
Multiplicand		0010		0010
Multiplier	x	0110		0110
	+	0000	+	0000
	+	0010	-	0010
	+	0010	+	0000
	+	0000	+	0010
Product	=	0001100	=	0001100



Idea of Booth Algorithm

- Looks at two bits of multiplier at a time from right to left
- Assume that shifts are much faster than adds
- Basic idea to speed up the calculation: avoid unnecessary additions

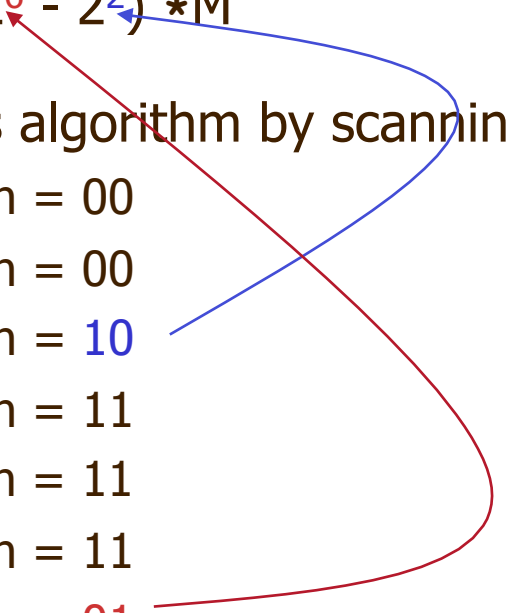
Example to Explain the Math

- Multiplier = 00111100
  - i.e. $i_1 = 2$, $i_2 = 5$
- M x 00111100 = $2^2 * M + 2^3 * M + 2^4 * M + 2^5 * M$

  $= 2^2 * (2^0 + 2^1 + 2^2 + 2^3) * M$

  $= 2^2 * (2^4 - 1) * M$

  $= (2^6 - 2^2) * M$



- Running the Booth's algorithm by scanning multiplier from right to left
  - Iteration 0, pattern = 00
  - Iteration 1, pattern = 00
  - Iteration 2, pattern = 10
  - Iteration 3, pattern = 11
  - Iteration 4, pattern = 11
  - Iteration 5, pattern = 11
  - Iteration 6, pattern = 01

To find out why, do the math:

- Consider a series of ones in the multiplier (from bit i_1 to bit i_2)
- M: multiplicand; multiplying M with this series of ones results in

$$\text{Prod} = (2^{i1})*M + (2^{i1+1})*M + \ldots + (2^{i2})*M$$
$$= (2^{i2+1}-2^{i1})*M$$

- Thus, $(i_2\text{-}i_1)$ adds in revised algorithm $\Rightarrow$ one add and one subtract

Detailed algorithm:

- We look at 2 bits at a time (current bit and previous one):
  - 00: middle of a string of 0's; no arithmetic operation
  - 01: end of a string of 1's; add M to the left half of product
  - 10: start of a string of 1's; subtract M from the left half of product
  - 11: middle of a string of 1's- no arithmetic operations
- Previous bit is set to 0 for the first iteration to form a two-bit pattern

Booth's Algorithm: Example

- Multiply 14 with -5 using 5-bit numbers (10-bit result)

Booth's Algorithm for Binary Multiplication Example

Multiply 14 times -5 using 5-bit numbers (10-bit result).

14 in binary: `01110`

-14 in binary: `10010` (so we can add when we need to subtract the multiplicand)

-5 in binary: `11011`

Expected result: -70 in binary: `11101 11010`

Booth's Algorithm: Example

Multiply 14 with -5 using 5-bit numbers (10-bit result)

Step	Multiplicand	Action	Multiplier upper 5-bits 0, lower 5-bits multiplier, 1 "Booth bit" initially 0
0	01110	Initialization	00000 11011 0
1	01110	10: Subtract Multiplicand	00000+10010=10010 10010 11011 0
		Shift Right Arithmetic	11001 01101 1
2	01110	11: No-op	11001 01101 1
		Shift Right Arithmetic	11100 10110 1
3	01110	01: Add Multiplicand	11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.) 01010 10110 1
		Shift Right Arithmetic	00101 01011 0
4	01110	10: Subtract Multiplicand	00101+10010=10111 10111 01011 0
		Shift Right Arithmetic	11011 10101 1
5	01110	11: No-op	11011 10101 1
		Shift Right Arithmetic	11101 11010 1

Booth Algorithm and Division

Review of Booth algorithm

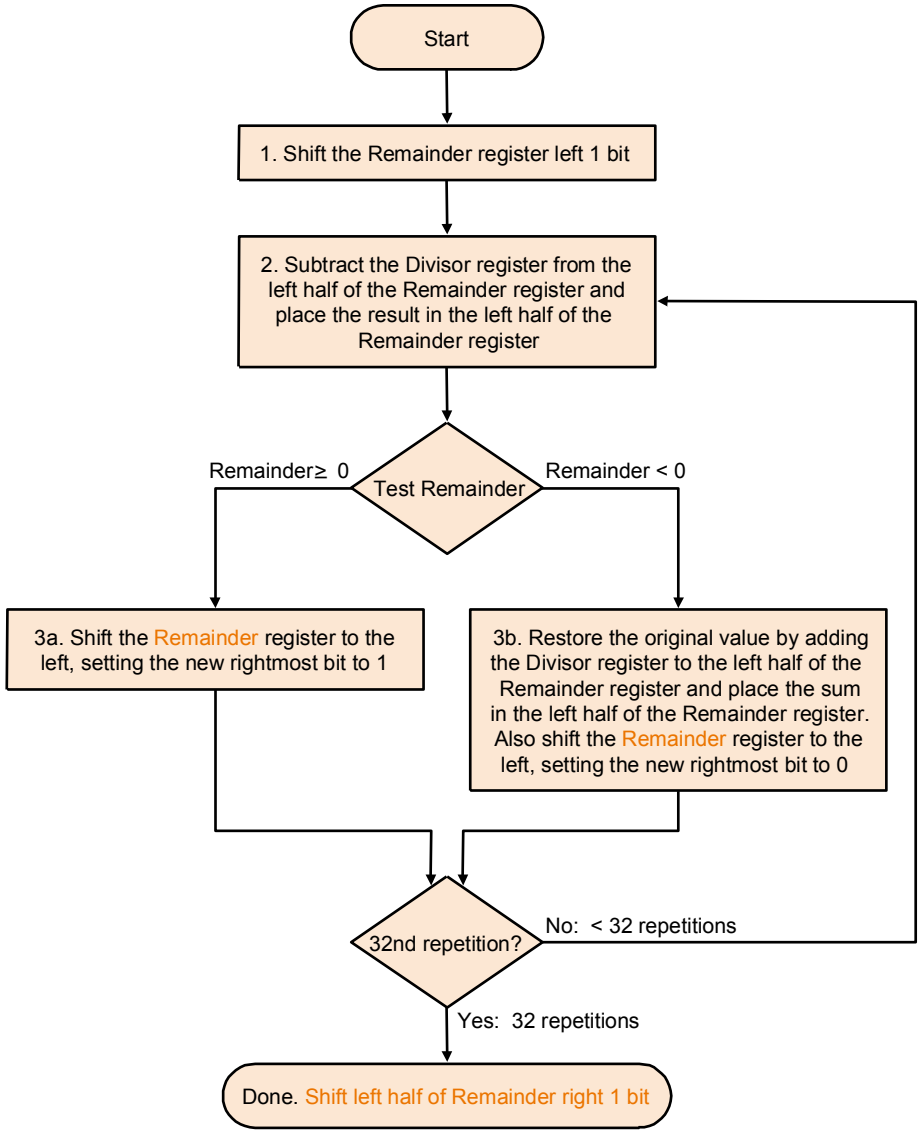
- Examples

Division

- Examples

Exercises

Division Algorithm – Improved Version



Division: Examples

Answer the following in binary form (numbers are in base 10 , convert to 4-bit binary numbers)

- Divide 10 by 3 => 1010 by 0011
- Divide 5 by 7 => 0101 by 0111

Booth Algorithm and Division

Introduction of Booth algorithm

- Examples

Division

- Examples

Exercises

Exercises

Answer the following in binary form (numbers are in base 10 , convert to 4-bit binary numbers)

- Multiply -2 by -7 (result in 8-bit binary numbers)
- Multiply -8 by 4 (result in 8-bit binary numbers)
- Divide -7 by 2
- Divide -8 by -2