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	shift
	+	0010	-	0010	subtract
	+	0010	+	0000	shift
	+	0000	+	0010	add
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

- 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

Booth's Algorithm

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, $(i_2 - i_1)$ adds in revised algorithm ⇒ 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: Examples

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

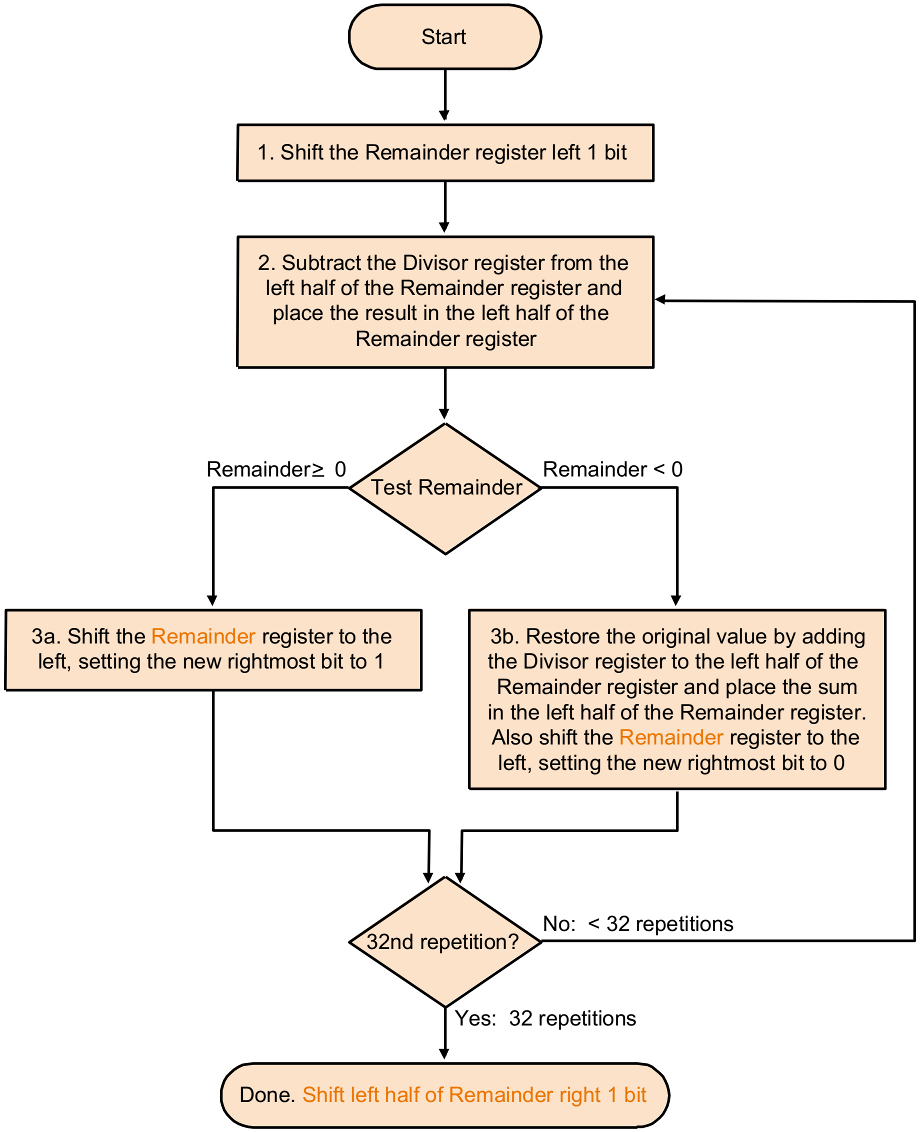
Booth Algorithm and Division

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
- ❑Divide 5 by 7

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 10-bit binary numbers as 4-bit multiplicand not enough)

- Divide -7 by 2

- Divide -8 by -2

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

Solution: **0000 1110**

Step	Multiplicand	Action	Multiplier
0	1110	Initialization	0000 1001 0
1	1110	10: subtract multiplicand Shift right	0000 + 0010 = 0010 0010 1001 0 0001 0100 1
2	1110	01: add multiplicand Shift right	0001 + 1110 = 1111 1111 0100 1 1111 1010 0
3	1110	00: No operation Shift right	1111 1101 0
4	1110	10: subtract multiplicand Shift right	1111 + 0010 = 0001 0001 1101 0 **0000 1110** 1

Exercises

❑Multiply -8 by 4 (result in 10-bit binary numbers as 4-bit multiplicand not enough)

Solution: 11111 00000

Step	Multiplicand	Action	Multiplier
0	11000	Initialization	00000 00100 0
1	11000	00: no operation Shift right	00000 00100 0 00000 00010 0
2	11000	00: no operation Shift right	00000 00010 0 00000 00001 0
3	11000	10: subtract multiplicand Shift right	00000 + 01000 = 01000 01000 00001 0 00100 00000 1
4	11000	01: add multiplicand Shift right	00100 + 11000 = 11100 11100 00000 1 11110 00000 0
5	11000	00: no operation Shift right	11110 00000 0 **11111 00000** 0

Exercises

- Divide -7 by 2 => Q = (-) 0011 = 1101, R= 0001

Step	Divisor (D)	Remainder (R)	Remark
0	0010	0000 0111 0000 1110	Initial state R = R << 1
1	0010	1110 1110 0000 1110 0001 1100	Left(R) = Left(R) – D Undo R = R << 1, R0 = 0
2	0010	1111 1100 0001 1100 0011 1000	Left(R) = Left(R) – D Undo R = R << 1, R0 = 0
3	0010	0001 1000 0011 0001	Left(R) = Left(R) – D R = R << 1, R0 = 1
4	0010	0001 0001 0010 0011	Left(R) = Left(R) – D R = R << 1, R0 = 1
Extra		**0001 0011**	Left(R) = Left(R) >> 1

Exercises

- Divide -8 by -2 => Q = 0100, R= 0000

Step	Divisor (D)	Remainder (R)	Remark
0	0010	0000 1000 0001 0000	Initial state R = R << 1
1	0010	1111 0000 0001 0000 0010 0000	Left(R) = Left(R) – D Undo R = R << 1, R0 = 0
2	0010	0000 0000 0000 0001	Left(R) = Left(R) – D R = R << 1, R0 = 1
3	0010	1110 0001 0000 0001 0000 0010	Left(R) = Left(R) – D undo R = R << 1, R0 = 0
4	0010	1110 0010 0000 0010 0000 0100	Left(R) = Left(R) – D undo R = R << 1, R0 = 0
Extra		**0000 0100**	Left(R) = Left(R) >> 1