Operations on Data

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

Introduction

- We have discussed the fact that data inside a computer is stored as patterns of bits
  - *Logic operations* (邏輯運算) refer to those operations that apply the same basic operation on individual bits of a pattern, or on two corresponding bits in two patterns
  - We can define logic operations at the *bit level* (位元階層) and at the *pattern level* (樣式階層)

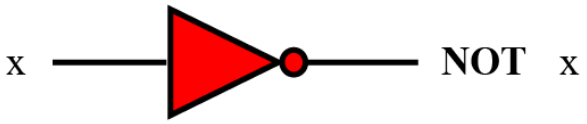
Logic Operations - Logic operations at bit level

- A bit can take one of the two values: 0 or 1
  - If we interpret 0 as the value *false* and 1 as the value *true*, we can apply the operations defined in *Boolean algebra* (布林代數) to manipulate bits
  - It belongs to a subfield of mathematics called *logic*
- We show briefly four bit-level operations that are used to manipulate bits: NOT, AND, OR, and XOR
  - A *truth table* (真值表) is used to define the values of output for each possible input. Note that the output of each operator is always one bit, but the input can be one or two bits

Logic operations at bit level

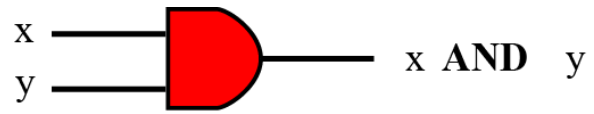
- NOT - a unary operator (一元運算子): it takes only one input
  - The output bit is the complement of the input. If the input is 0, the output is 1, if the input is 1, the output is 0
- AND - a binary operator (二元運算子): it takes two inputs
  - The output bit is 1 if both inputs are 1s and the output is 0 in the other three cases

For $x = 0$ or 1 x AND 0 → 0 0 AND x → 0



NOT

x	NOT x
0	1
1	0



AND

x	y	x AND y
0	0	0
0	1	0
1	0	0
1	1	1

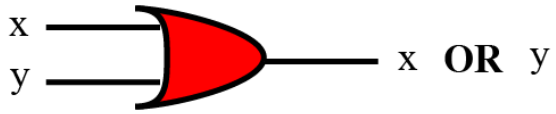
Logic operations at bit level

- OR - a binary operator
  - The output bit is 0 if both inputs are 0s and the output is 1 in the other three cases It is sometimes called the *inclusive-or operator*

For $x = 0$ or 1 $\quad x$ OR 1 $\rightarrow$ 1 $\quad$ 1 OR $x \rightarrow$ 1

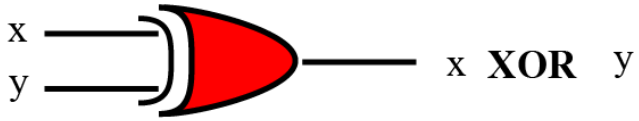
- XOR - a binary operator
  - Like the OR operator, with only one difference: the output is 0 if both inputs are 1s

For $x = 0$ or 1 $\quad$ 1 XOR x $\rightarrow$ NOT x $\quad$ 1 XOR $x \rightarrow$ NOT x



OR

x	y	x OR y
0	0	0
0	1	1
1	0	1
1	1	1



XOR

x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

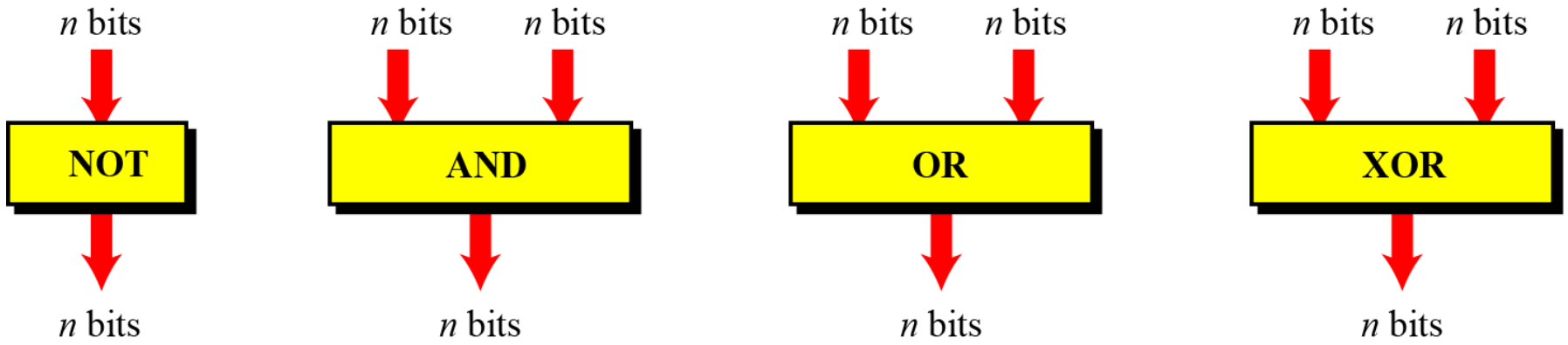
Logic operations at bit level

- The XOR operator is not actually a new operator. We can always simulate it using the other three operators. The following two expressions are equivalent:

x XOR y ↔ [x AND (NOT y)] OR [(NOT x) AND y]

Logic operations at pattern level

- The same four operators (NOT, AND, OR, and XOR) can be applied to an *n*-bit pattern
  - The effect is the same as applying each operator to each individual bit for NOT and to each corresponding pair of bits for other three operators



Logic operations at pattern level

- Use the NOT operator on the bit pattern 10011000

- Use the AND operator on the bit patterns 10011000 and 00101010

Logic operations at pattern level

- Use the OR operator on the bit patterns 10011001 and 00101110

- Use the XOR operator on the bit patterns 10011001 and 00101110

Applications

- Complementing (取補數)
  - An application of the NOT operator is to complement the whole pattern
  - Applying this operator to a pattern changes every 0 to 1 and every 1 to 0, which is sometimes referred to as a *one's complement* (一的補數) operation
- Unsetting specific bits (清除特定位元)
  - One of the applications of the AND operator is to unset (force to 0) bits in a bit pattern
  - The second input, in this case, is called a *mask* (遮罩). The 0-bits in the mask unset the corresponding bits in the first input, while 1-bits in the mask leave the corresponding bits in the first input unchanged

Applications

- Setting specific bits (設定特定位元)
  - One of the applications of the OR operator is to set (force to 1) bits in a bit pattern
  - The 1-bits in the mask set the corresponding bits in the first input, and the 0-bits in the mask leave the corresponding bits in the first input unchanged
- Flipping specific bits (反轉特定位元)
  - One of the applications of the XOR operator is to flip (complement) bits in a bit pattern
  - The 1-bits in the mask flip the corresponding bits in the first input, and the 0-bits in the mask leave the corresponding bits in the first input unchanged

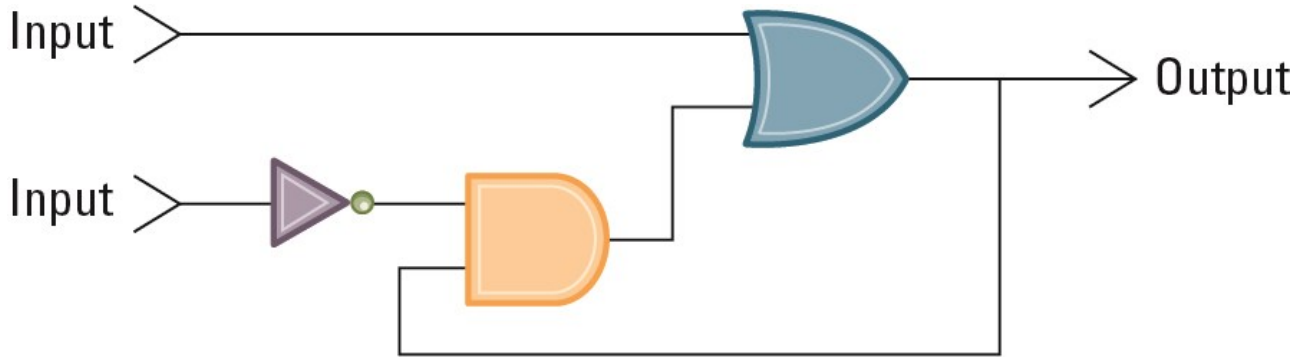
Applications

- Use a mask to unset (clear) or set the five leftmost bits of a pattern 10100110

- Use a mask to flip the five leftmost bits of a pattern 10100110

Digital circuits

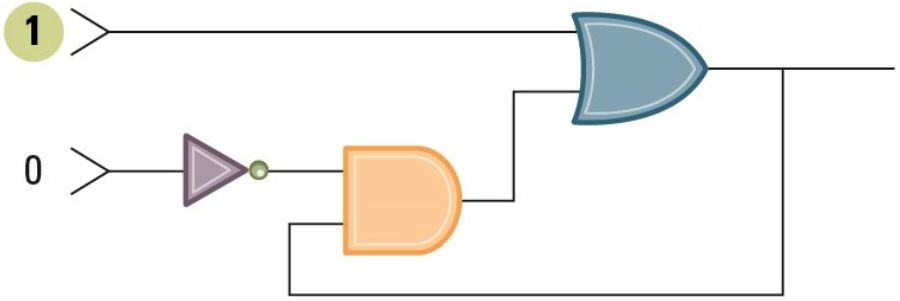
- A computer is normally built out of standard components that we collectively refer to as *digital circuits*
- *Logic gate* (邏輯閘) is a device that computes a Boolean operation
  - Often implemented as small electronic circuits called transistors
  - Can be constructed from a variety of other technologies
  - Provide the building blocks from which computers are constructed
- One important circuit is known as a *flip-flop* (正反器)



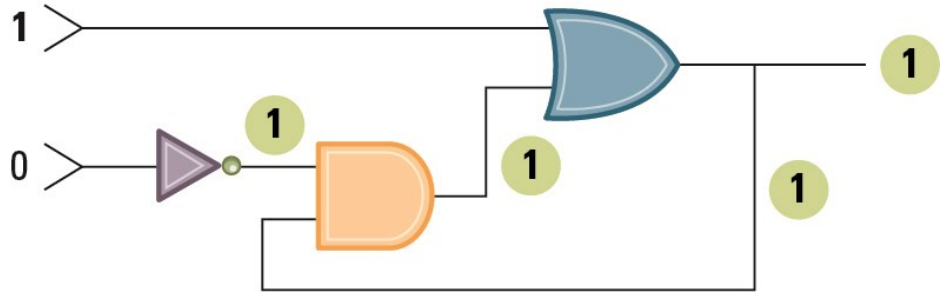
Digital circuits

- This act as a fundamental unit of computer memory
  - One input line is used to set its stored value to 1
  - One input line is used to set its stored value to 0
  - While both input lines are 0, the most recently stored value is preserved

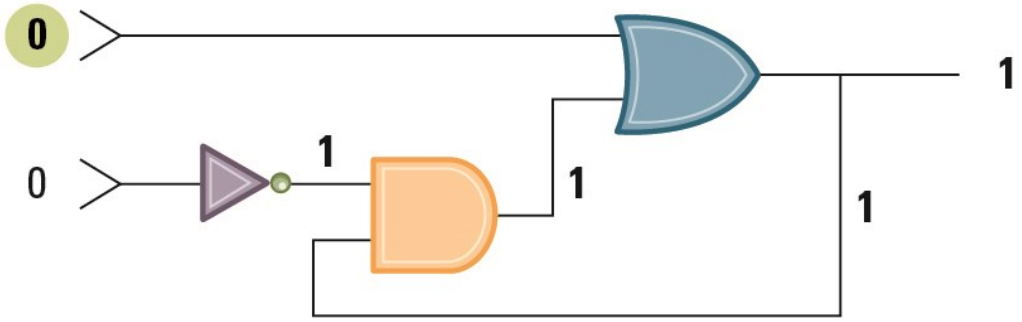
a. First, a 1 is placed on the upper input.



b. This causes the output of the OR gate to be 1 and, in turn, the output of the AND gate to be 1.



c. Finally, the 1 from the AND gate keeps the OR gate from changing after the upper input returns to 0.



Digital circuits

- This demonstrates how devices can be constructed from logic gates, a process known as *digital circuit design*, which is an important topic in engineering
  - Check Appendix E for more details
- The concept of a flip-flop provides an example of abstraction and the use of abstract tools
  - A computer engineer does not need to know the internals of flip-flop. Instead, only an understanding of the flip-flop's external properties is needed to use it as an abstract tool
  - The design of computer circuitry takes on a hierarchical structure, each level of which uses the lower level components as abstract tools
- A technology known as *very large-scale integration (VLSI)* (超大型積體電路), is used to create miniature devices containing millions of flip-flops, logic gates along with their controlling circuitry

Shift Operations (移位運算)

- Shift operations move the bits in a pattern, changing the positions of the bits
  - We can divide shift operations into two categories: *logical shift* (邏輯移位) operations and *arithmetic shift* (算數移位) operations
- A logical shift operation is applied to a pattern that does not represent a signed number
  - The reason is that these shift operations may change the sign of the number that is defined by the leftmost bit in the pattern
- Arithmetic shift operations assume that the bit pattern is a signed integer in two's complement format

Logical shift operations – Simple shift

- A simple right (left) shift operation shifts each bit one position to the right
  - The rightmost (leftmost) bit is lost and a 0 fills the leftmost (rightmost) bit

a. Logical right shift

b. Logical left shift

- Use a logical left shift operation on the bit pattern 10011000

Logical shift operations – Circular shift (循環移位)

- A circular shift operation (or rotate operation) shifts bits, but no bit is lost or added
  - A circular right shift (left shift) shifts each bit one position to the right (left). The rightmost (leftmost) bit is circulated and becomes the leftmost (rightmost) bit

a. Circular right shift

b. Circular left shift

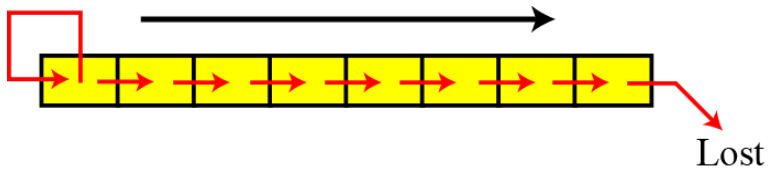
- Use a circular left shift operation on the bit pattern 10011000

Applications

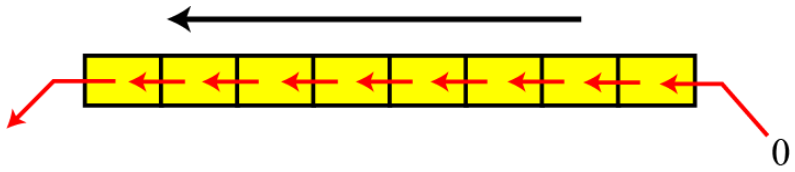
- Assume that we have a pattern and we need to use the third bit (from the right) of this pattern in a decision-making process. We want to know if this particular bit is 0 or 1

Arithmetic shift operations

- Arithmetic right shift is used to divide an integer by two, while arithmetic left shift is used to multiply an integer by two
  - These operations should not change the sign (leftmost) bit. An arithmetic right shift retains the sign bit, but also copies it into the next right bit, so that the sign is preserved
  - An arithmetic left shift discards the sign bit and accepts the bit to the left of the sign bit as the sign
    - If the new sign bit is the same as the previous one, the operation is successful, otherwise, an overflow or underflow has occurred and the result is not valid



a. Arithmetic right shift



b. Arithmetic left shift

Arithmetic shift operations

- Use an arithmetic right shift operation on the bit pattern 10011001

- Use an arithmetic left shift operation on the bit pattern 11011001

Arithmetic shift operations

- Use an arithmetic left shift operation on the bit pattern 01111111. Is the result valid or not?

Arithmetic operations

- Arithmetic operations involve adding, subtracting, multiplying, and dividing. We can apply these operations to integers and floating-point numbers
- All arithmetic operations can be applied to integers. We only discuss addition and subtraction of integers here

Arithmetic operations - two's complement integers

- We first discuss addition and subtraction for integers in two's complement representation
  - One of the advantages of two's complement representation is that there is no difference between addition and subtraction
    - When the subtraction operation is encountered, the computer simply changes it to an addition operation, but makes two's complement of the second number

$$A - B \leftrightarrow A + (\bar{B} + 1)$$

Where $(\bar{B} + 1)$ means the two's complement of B

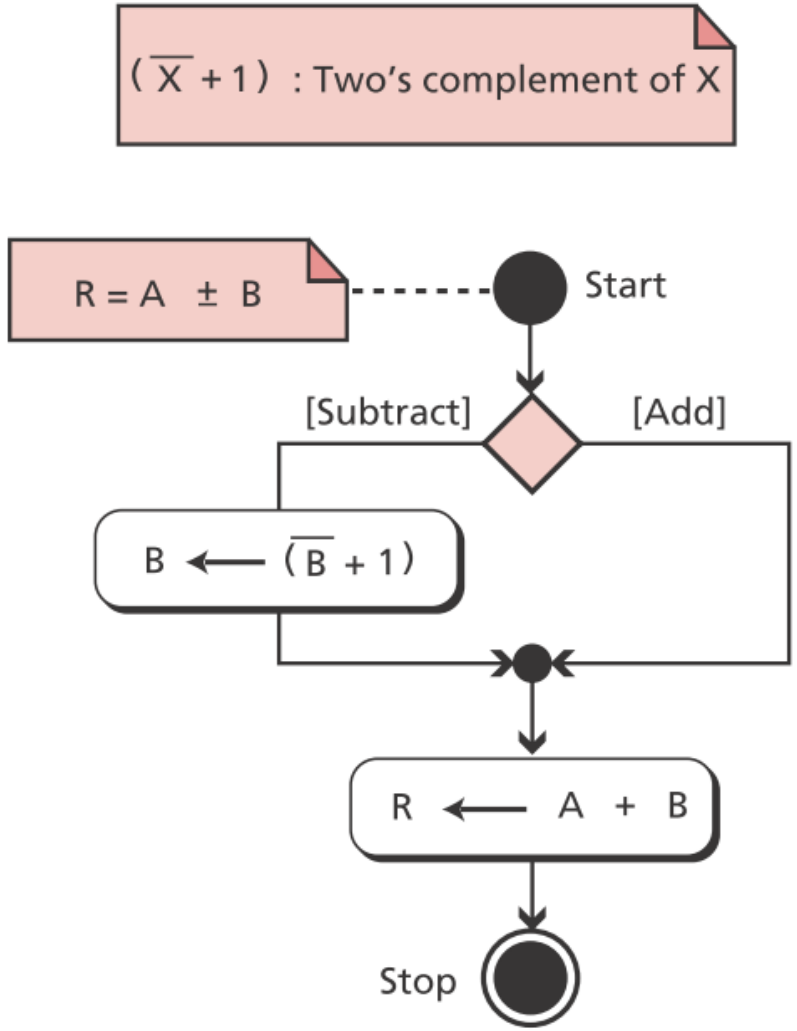
Arithmetic operations - two's complement integers

- We should remember that we add integers column by column
  - In each column, we have either two bits to add if there is no carry from the previous column, or three bits to add if there is a carry from the previous column

Column	*Carry*	*Sum*	*Column*	*Carry*	*Sum*
Zero 1s	0	0	Two 1s	1	0
One 1	0	1	Three 1s	1	1

Arithmetic operations - two's complement integers

1. If the operation is subtraction, we take the two's complement of the second integer. Otherwise, we move to the next step
2. We add the two integers



Arithmetic operations - two's complement integers

- Two integers $A = (00011001)_2$ and $B = (11101111)_2$ are stored in two's complement format. Show how B is added to A and show how B is subtracted from A

Arithmetic operations - two's complement integers

- Two integers $A = (00010001)_2$ and $B = (00010110)_2$ are stored in two's complement format. Show how B is subtracted from A

- Two integers $A = (11011101)_2$ and $B = (00010100)_2$ are stored in two's complement format. Show how B is subtracted from A

Arithmetic operations - two's complement integers

- Two integers $A = (01111111)_2$ and $B = (00000011)_2$ are stored in two's complement format. Show how B is added to A. Is the result valid or not?

Arithmetic operations - sign-and-magnitude integers

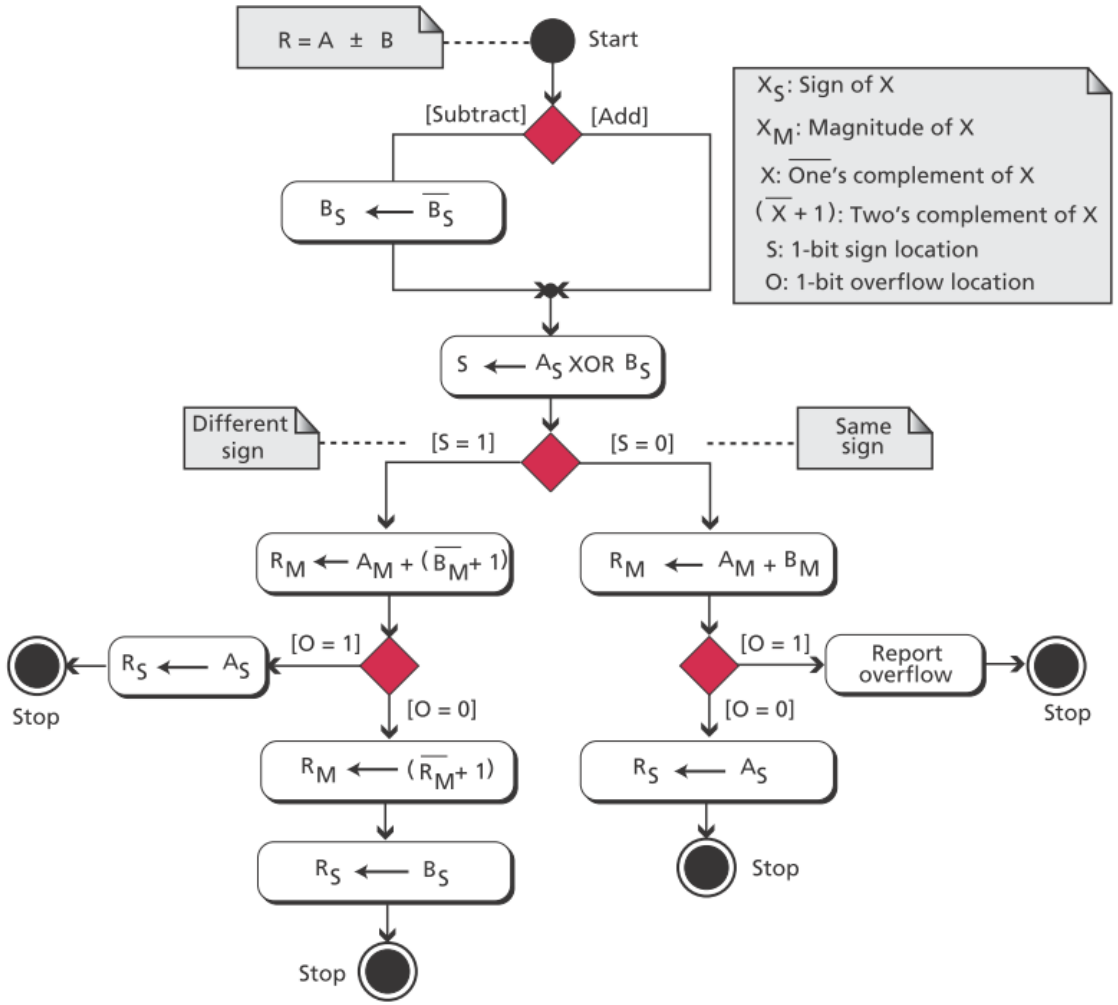
- Addition and subtraction for integers in sign-and-magnitude representation look very complex
  - We have four different combinations of signs (two signs, each of two values) for addition, and four different conditions for subtraction. This means that we need to consider eight different situations
  - Check Appendix I for more details

Arithmetic operations - reals

- All arithmetic operations such as addition, subtraction, multiplication, and division can be applied to reals stored in floating-point format
  - Multiplication of two reals involves multiplication of two integers in sign-and-magnitude representation. Division of two reals involves the division of two integers in sign-and-magnitude representations
  - Check Appendix J for more details

Appendix

Arithmetic operations - sign-and-magnitude integers



Arithmetic operations - reals