VNUHCM-UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

CSC10009 – COMPUTER SYSTEM

Assignment 01: Integer Arithmetic

Lecturer
Mr. Lê Quốc Hòa

Class
23CLC08

Students
23127226 – Nguyễn Đăng Minh

October 18th, 2024

Contents

1 Preface

1.1 About

This report clearly states the operation of the Binary-Arithmetic program, all the test cases that can happen when performing an arithmetic operation on bits. Here we will focus specifically on 8-bit signed integers, the solution is the same for different number of bits. Five operations, addition, subtraction, multiplication, division, and modulo, respectively, are stated in this project. I would like to give a special thanks to my teacher, Mr. Le Quoc Hoa, for perfectly instructing me about bits, which has helped me finishing this project easier.

1.2 Environment

The Binary-Arithmetic program is coded in language C++ 11, on Window OS. Any IDLE can be used such as Visual Studio Code, CodeBlocks, etc. To run this program, you can run the main.exe executable file or build the main.cpp file in terminal with the following command:

g++ main.cpp -o main.exe

Then run: **./main.exe**

1.3 Work Percentage

Task	**Subtask**	**Progress (%)**
Input	Check number in 8-bit range	100%
Get bit	Get bit using "»" and "&" operators	100%
Addition	Add 2 signed numbers using xor operator	100%
	Result is still 8-bit when overflow	100%
Subtraction	Get 2's complement and perform adding	100%
Multiply	Convert 2 numbers to positive and multiply	100%
	Update the sign of the result	100%
Division	Check if divide by zero	100%
	Convert 2 numbers to positive	100%
	Divide using binary-long-division method	100%
	Update the sign of the quotient	100%
Modulo	After division, return and update sign of remainder	100%

2 Get bit sequence of a number

To get the binary format of a decimal number, traverse through all bits and check if the current bit is turned on using the **shift right** (») and **and** (&) operators.

Two figures below show the get-bit functionality for signed integers (8-bit).

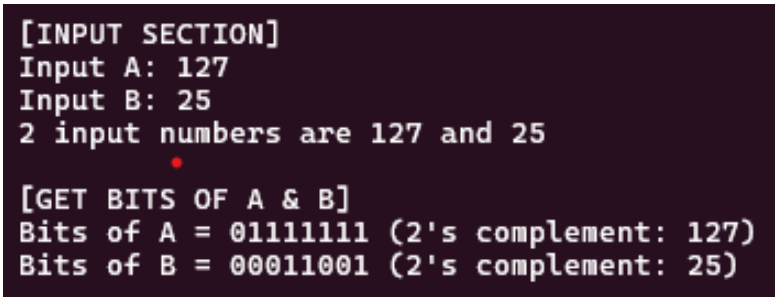

Figure 1: Present bits of positive numbers

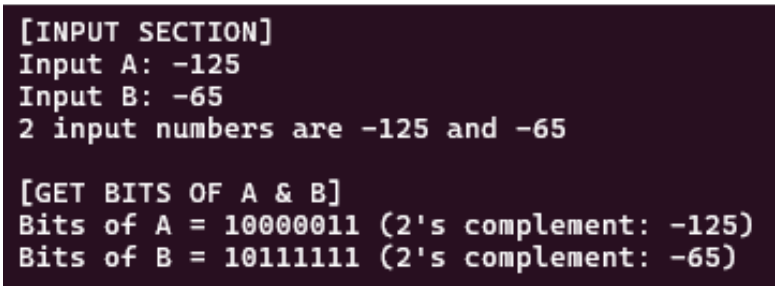

Figure 2: Present bits of negative numbers

The bit sequence is added to a char array (Ex: char arrA[8]).

The output of the sequence is actually the output of the char array after getting bits of two numbers.

3 Addition

Bits addition is simply the xor result of two bits sequence but adding the carry value. The procedure is as same as decimal presentation.

Adding two positive numbers (the 2's complement is a function returns 8-bit signed integer):

```
[INPUT SECTION]
Input A: 25
Input B: 6
2 input numbers are 25 and 6

[GET BITS OF A & B]
Bits of A = 00011001 (2's complement: 25)
Bits of B = 00000110 (2's complement: 6)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 00011111 (2's complement: 31)
```

Figure 3: Adding 25 and 6

Adding a positive and negative number:

```
[INPUT SECTION]
Input A: 25
Input B: -6
2 input numbers are 25 and -6

[GET BITS OF A & B]
Bits of A = 00011001 (2's complement: 25)
Bits of B = 11111010 (2's complement: -6)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 00010011 (2's complement: 19)
```

Figure 4: Adding 25 and -6

Adding two negative numbers:

```
[INPUT SECTION]
Input A: -25
Input B: -6
2 input numbers are -25 and -6

[GET BITS OF A & B]
Bits of A = 11100111 (2's complement: -25)
Bits of B = 11111010 (2's complement: -6)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 11100001 (2's complement: -31)
```

Figure 5: Adding -25 and -6

When adding is overflow (outside 8-bit signed integers range):

```
[INPUT SECTION]
Input A: 127
Input B: 25
2 input numbers are 127 and 25

[GET BITS OF A & B]
Bits of A = 01111111 (2's complement: 127)
Bits of B = 00011001 (2's complement: 25)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 10011000 (2's complement: -104)
```

Figure 6: Adding 127 and 25

To classify the -104 value, we can run this code in C++ and the output would also be -104.

```
char a = 127;
a += 25;
cout << (int)a; // -104
```

Figure 7: Confirming the result

4 Subtraction

Subtraction is adding two numbers, but the second number would be the 2's complement.

Subtracting two positive numbers:

```
[INPUT SECTION]
Input A: 100
Input B: 25
2 input numbers are 100 and 25

[GET BITS OF A & B]
Bits of A = 01100100 (2's complement: 100)
Bits of B = 00011001 (2's complement: 25)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 01111101 (2's complement: 125)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 01001011 (2's complement: 75)
```

Figure 8: Subtracting 100 and 25

Subtracting a positive and a negative number:

```
[INPUT SECTION]
Input A: 100
Input B: -25
2 input numbers are 100 and -25

[GET BITS OF A & B]
Bits of A = 01100100 (2's complement: 100)
Bits of B = 11100111 (2's complement: -25)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 01001011 (2's complement: 75)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 01111101 (2's complement: 125)
```

Figure 9: Subtracting 100 and -25

Subtracting two negative numbers:

```
[INPUT SECTION]
Input A: -100
Input B: -25
2 input numbers are -100 and -25

[GET BITS OF A & B]
Bits of A = 10011100 (2's complement: -100)
Bits of B = 11100111 (2's complement: -25)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 10000011 (2's complement: -125)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 10110101 (2's complement: -75)
```

Figure 10: Subtracting -100 and -25

When subtracting is overflowed:

```
[INPUT SECTION]
Input A: 120
Input B: -128
2 input numbers are 120 and -128

[GET BITS OF A & B]
Bits of A = 01111000 (2's complement: 120)
Bits of B = 10000000 (2's complement: -128)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 11111000 (2's complement: -8)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 11111000 (2's complement: -8)
```

Figure 11: Subtracting 120 and -128

Classify the result:

```
char a = 120;
char b = -128;
a = a - b;
cout << (int)a; // -8
```

Figure 12: Classifying the result

5 Multiplication

The multiplication procedure is as same as in decimal multiplication. In this program, we will display the result as 16-bit signed integers.

Multiply two positive numbers:

```
[INPUT SECTION]
Input A: 120
Input B: 30
2 input numbers are 120 and 30

[GET BITS OF A & B]
Bits of A = 01111000 (2's complement: 120)
Bits of B = 00011110 (2's complement: 30)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 10010110 (2's complement: -106)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 01011010 (2's complement: 90)

[MULTIPLICATION OF A & B IN BITS]
Convert A to positive: 01111000
Convert B to positive: 00011110
bitA * bit(B) = 0000111000010000 (2's complement: 3600)
```

Figure 13: Multiplying 120 and 30

Multiply a negative and positive number:

```
[INPUT SECTION]
Input A: -89
Input B: 111
2 input numbers are -89 and 111

[GET BITS OF A & B]
Bits of A = 10100111 (2's complement: -89)
Bits of B = 01101111 (2's complement: 111)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 00010110 (2's complement: 22)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 00111000 (2's complement: 56)

[MULTIPLICATION OF A & B IN BITS]
Convert A to positive: 01011001
Convert B to positive: 01101111
bitA * bit(B) = 1101100101101001 (2's complement: -9879)
```

Figure 14: Multiplying -89 and 111

Multiply two negative numbers:

```
[INPUT SECTION]
Input A: -128
Input B: -127
2 input numbers are -128 and -127

[GET BITS OF A & B]
Bits of A = 10000000 (2's complement: -128)
Bits of B = 10000001 (2's complement: -127)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 00000001 (2's complement: 1)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 11111111 (2's complement: -1)

[MULTIPLICATION OF A & B IN BITS]
Convert A to positive: 10000000
Convert B to positive: 01111111
bitA * bit(B) = 0011111110000000 (2's complement: 16256)
```

Figure 15: Multiplying -128 and -127

Multiply with zero:

```
[INPUT SECTION]
Input A: 25
Input B: 0
2 input numbers are 25 and 0

[GET BITS OF A & B]
Bits of A = 00011001 (2's complement: 25)
Bits of B = 00000000 (2's complement: 0)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 00011001 (2's complement: 25)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 00011001 (2's complement: 25)

[MULTIPLICATION OF A & B IN BITS]
Convert A to positive: 00011001
Convert B to positive: 00000000
bitA * bit(B) = 0000000000000000 (2's complement: 0)
```

```
[INPUT SECTION]
Input A: -128
Input B: 0
2 input numbers are -128 and 0

[GET BITS OF A & B]
Bits of A = 10000000 (2's complement: -128)
Bits of B = 00000000 (2's complement: 0)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 10000000 (2's complement: -128)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 10000000 (2's complement: -128)

[MULTIPLICATION OF A & B IN BITS]
Convert A to positive: 10000000
Convert B to positive: 00000000
bitA * bit(B) = 0000000000000000 (2's complement: 0)
```

6 Division and modulo

I use binary-int-division technique [1] to solve the binary division between two bits sequences.
The modulo (A % B) is the remainder of the division.
If dividend is 0:

```
[INPUT SECTION]
Input A: 0
Input B: 25
2 input numbers are 0 and 25

[GET BITS OF A & B]
Bits of A = 00000000 (2's complement: 0)
Bits of B = 00011001 (2's complement: 25)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 00011001 (2's complement: 25)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 11100111 (2's complement: -25)

[MULTIPLICATION OF A & B IN BITS]
Convert A to positive: 00000000
Convert B to positive: 00011001
bitA * bit(B) = 0000000000000000 (2's complement: 0)

[DIVISION OF A & B IN BITS]
Quotient of A / B = 00000000 (2's complement: 0)
Remainder of A / B (A % B) = 00000000 (2's complement: 0)
```

Figure 16: Dividend is 0

If divisor is 0:

```
[INPUT SECTION]
Input A: 126
Input B: 0
2 input numbers are 126 and 0

[GET BITS OF A & B]
Bits of A = 01111110 (2's complement: 126)
Bits of B = 00000000 (2's complement: 0)

[ADDITION OF A & B IN BITS]
bit(A) + bit(B) = 01111110 (2's complement: 126)

[SUBTRACTION OF A & B IN BITS]
bit(A) - bit(B) = 01111110 (2's complement: 126)

[MULTIPLICATION OF A & B IN BITS]
Convert A to positive: 01111110
Convert B to positive: 00000000
bitA * bit(B) = 0000000000000000 (2's complement: 0)

[DIVISION OF A & B IN BITS]
Cannot perform divide by zero!
```

Figure 17: Divide by 0

If dividend is smaller than divisor:

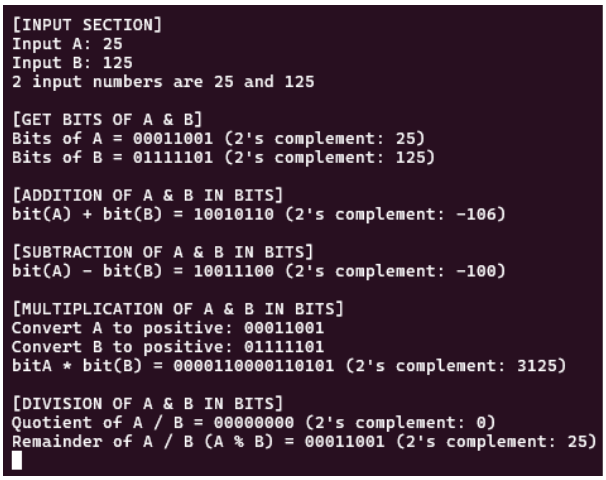

Figure 18: Dividend $<$ Divisor

Divide two positive numbers:

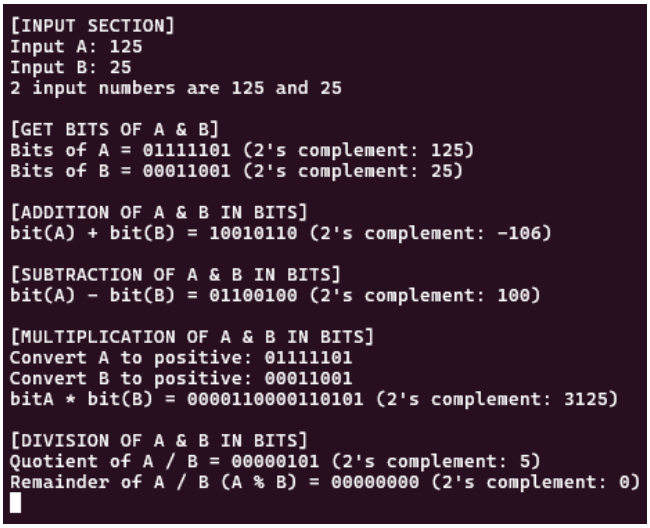

Figure 19: Divide 125 by 25

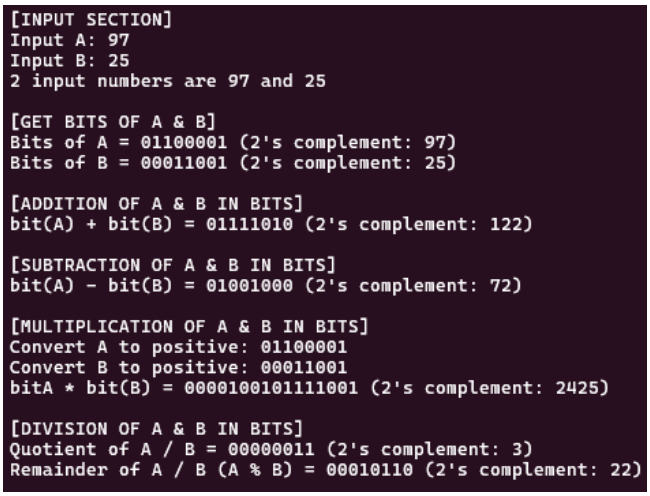

Figure 20: Remainder > 0

Divide a positive by a negative number:

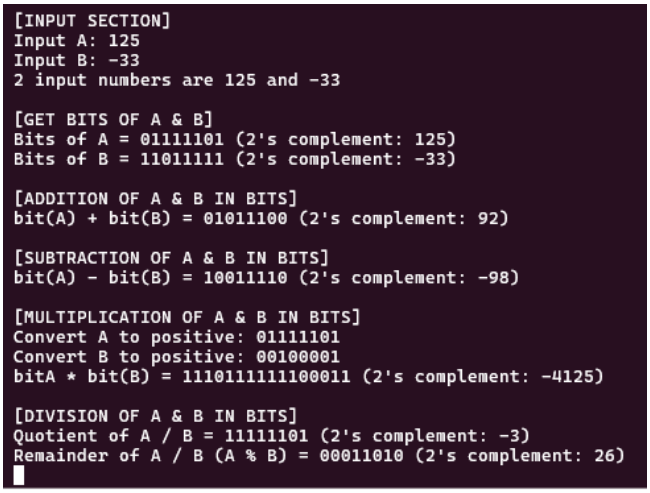

Figure 21: Divide 125 by -33

Divide a negative by a positive number:

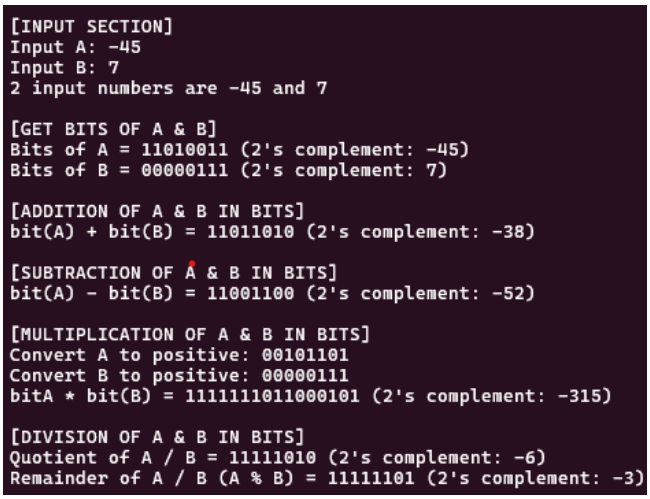

Figure 22: Divide -45 by 7

Divide two negative numbers:

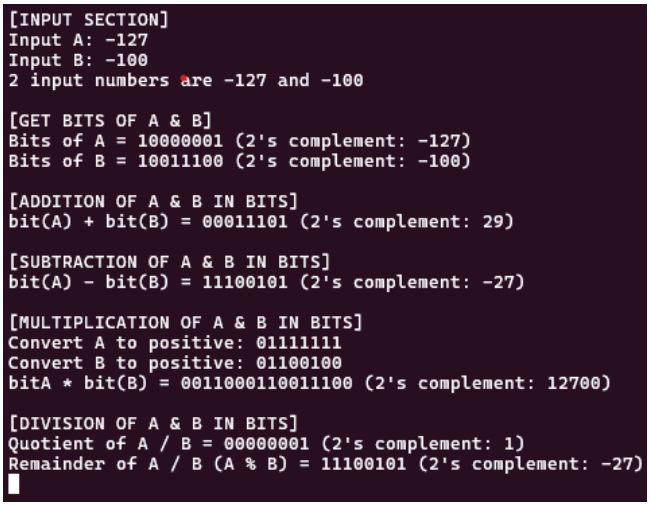

Figure 23: Divide -127 by -100

Overflow in division: occurs only in one special case (-128 / -1).

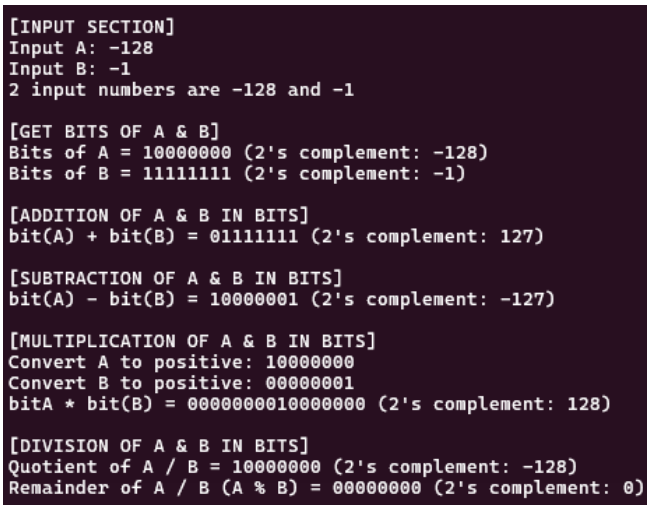

Figure 24: Divide -128 by -1

The result may seems wrong but actually, the 2's complement presentation of 128 is equal to -128 (10000000).

References

[1] Luis Llamas. *How to divide integers in binary*. URL: https://www.luisllamas.es/en/multiply-or-divide-binary/#binary-integer-division.