Principle of Computer Organization

Lab01: Implementation of Multiplier and Divider

1. Purpose

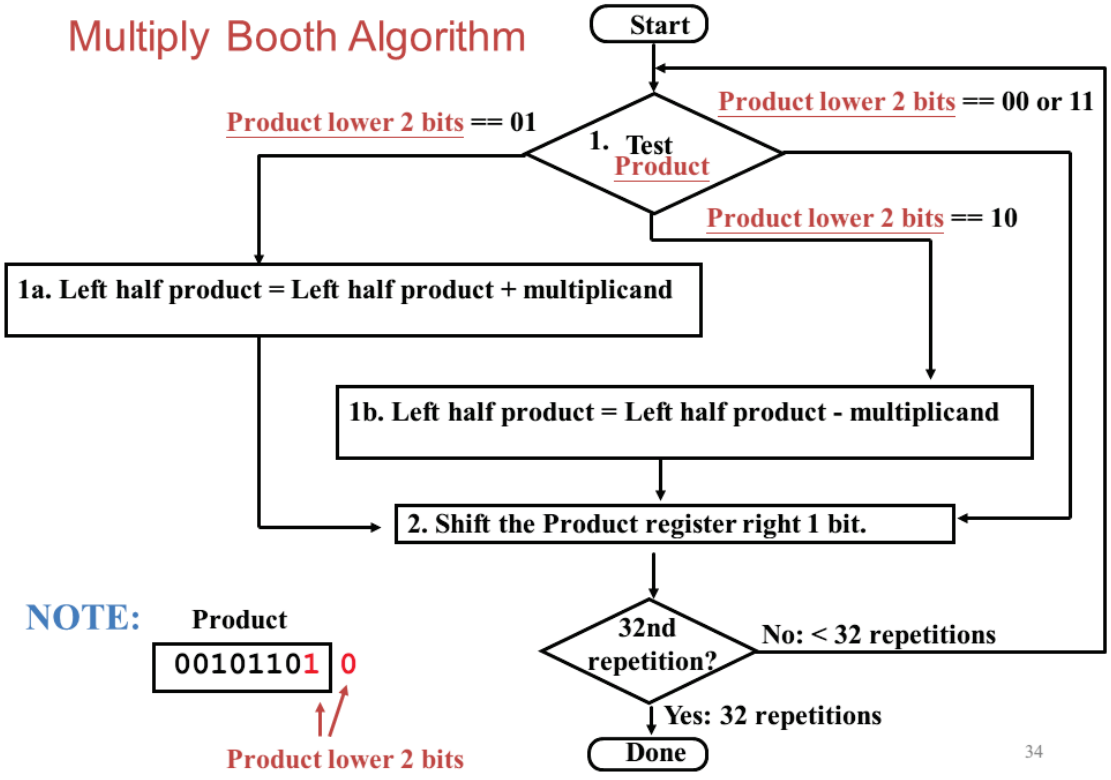
Understand the basic principle of multiplier and divider.
Know how to implement a multiplier and a divider.

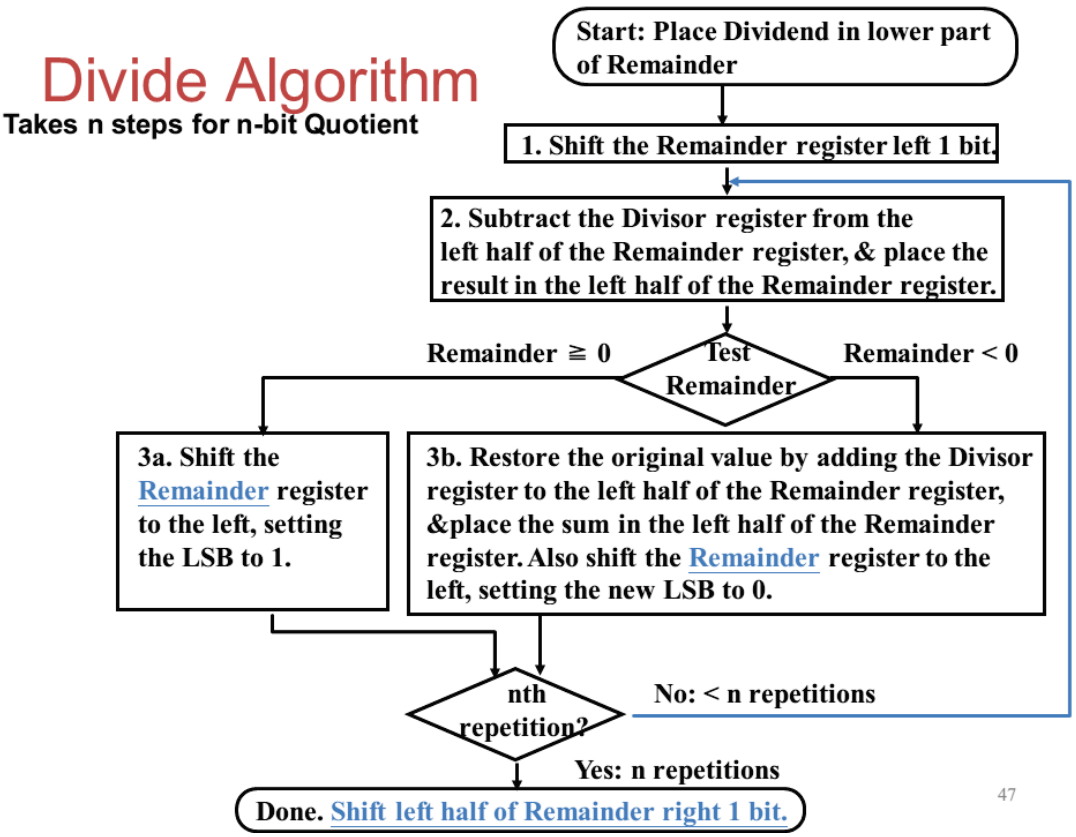
2. Introduction

In this lab experiment, you are going to implement a 32-bit *multiplier* simulator as well as a 32-bit *divider* simulator using C language. Your simulators will perform multiply and divide operations for 32-bit binary numbers. Some C files implementing each component of the *multiplier* and the *divider* will be provided to you, you are required to modify and fill in the body of the functions in these files.

3. Specification of the simulators

The multiplier and divider simulators are respectively implemented by Booth algorithm and the algorithm illustrated in the flowchart as below:





Specially, you need to employ a *full-adder* to perform addition and subtraction operations. The *full-adder* can be implemented by the following formulas. You may check your *full-adder* according to the truth table.

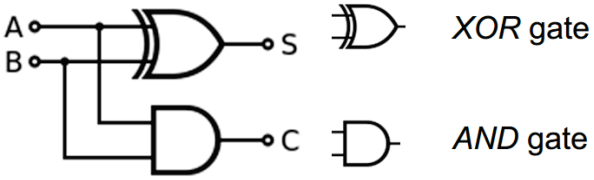
S = A xor B xor carry_in
carry_out = A&B | A&carry_in | B&carry_in

A	B	carry_in	carry_out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

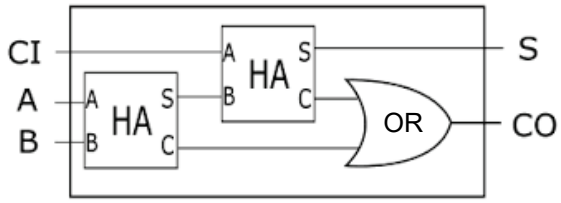
◆ fulladder(…)
Performs addition with 1-bit signed numbers.

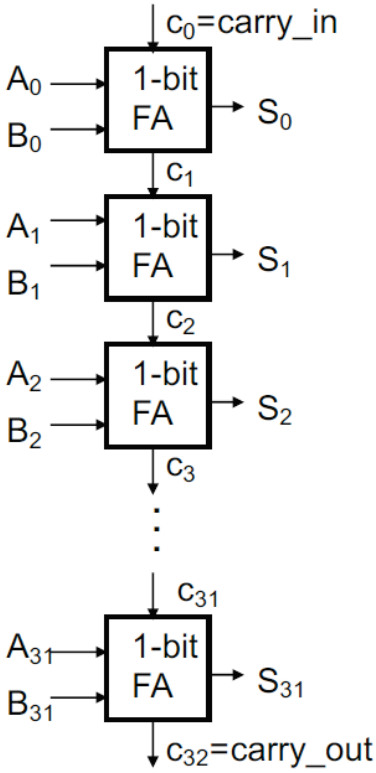
```
void full_adder(IN BOOL a, IN BOOL b, IN BOOL carry_in,
                OUT BOOL * carry_out, OUT BOOL * s)
{
  // for example:
  // *s = a ^ b ^ carry_in;    // s = a xor b xor carry_in
}
```

You are encouraged to implement the *full-adder* using basic logic gates. A 1-bit *half-adder* is implemented at first, which contains two logic gates: a *XOR* gate and an *AND* gate. And then, a 1-bit *full-adder* is built using two *half-adders* and an *OR* gate.



Inputs		Outputs	
A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0






```
void halfadder(IN BOOL a, IN BOOL b, OUT BOOL * c, OUT BOOL * s)
{
      *s = xor_gate(a, b);
      *c = and_gate(a, b);
}

void full_adder(IN BOOL a, IN BOOL b, IN BOOL carry_in,
                        OUT BOOL * carry_out, OUT BOOL * s)
{
      // for example:
      // halfadder();
      // halfadder();
      // *s =
      // *carry_out =
}
```

Finally, a 32-bit *full-adder* is implemented by just connecting the carry-out of the least significant bit of 1-bit *full-adder* to the carry-in of the next least significant bit of another 1-bit *full-adder*, and...

Here, we set WIDTH to 32.

◆ fulladder32(…)
Performs addition with 32-bit signed numbers.

```
void fulladder32(IN int a, IN int b, OUT int * c)
```

```
{
    int i;
    BOOL A_bin[WIDTH], B_bin[WIDTH], S_bin[WIDTH];
    for (i = 0; i < WIDTH; i++){
        A_bin[i] = get_bit(a,i);   // representing A in binary.
        B_bin[i] = get_bit(b,i);   // representing B in binary.
    }

    BOOL C[32+1];     // Declaration of carry in and carry out;

    C[0] = 0;

    // Addition of A0 and B0, and the carry out is stored in C[1].
    fulladder(A_bin[0], B_bin[0], C[0], &C[1], &S_bin[0]);

    for (i = 0; i < WIDTH; i++){
        change_bit(c, i, S_bin[i]);
    }

}
```

The Booth algorithm can deal with multiplication with signed numbers, however, the divide algorithm mentioned above just deal with **unsigned** numbers. Hence, we calculate division with sign number as 他 following procedure:

1) Store the sign of dividend and divisor;
2) Make dividend and divisor positive, represented by unsigned binary numbers;
3) Perform the divide algorithm;
4) Complement quotient and remainder.

The signs of quotient and remainder are determined by signs of dividend and divisor according to the table below:

Dividend	Divisor	Quotient	Remainder
+	+	+	+
+	-	-	+
-	+	-	-
-	-	+	-

The C files to simulate the *multiplier* and *divider* are provided; you just need to modify and fill in the body of the functions of *fulladder.c* (**Task 1**), *fulladder32.c* (**Task 2**), *multiplier.c* (**Task 3**) and *divider.c* (**Task 4**).

4. Notes

This experiment will be compiled and marked using Dev C++. You can download it from the web (http://www.bloodshed.net/dev/devcpp.html) and install it on your computer.

You should not do any “print” or ”printf()” operation in these three files: main.c, functions.c and test.c; otherwise, the operation will disturb the marking process and your marks will be deducted.

To run the compiled executable, the testing result will be printed, “AC” means the results of your program are accepted, otherwise, “WA” means your program can not pass the testing.

To submit your lab report, at the beginning your lab1.c, type in your **English** name, e.g.

```
/*
* Designer:     name, student id
*/
```

And upload all your C files and the report to yyliang.fit.must@gmail.com.
The subject of your .zip/.rar file must be: **CO101 Project student_name**.