
Computer Organization Lab 2

教授:蔡文錦

TAs:林浩君、薛乃仁、吳年茵

Objectives

In Lab 2, we are going to implement an 32-bit **ALU (Arithmetic Logic Unit)** and a **shifter** by Verilog. Through this lab, students will learn how to design the function unit of a processor to support its instruction set. Note that you should **design these circuits in gate level as combinational logic instead of sequential logic**. The ALU and shifter designed in this lab may also be used in most of the succeeding lab units.

Task Description

- Attached Files
- Arithmetic Logic Unit(ALU)
 - Overview
 - ALU Control Signals Table
 - 1-bit ALU Architecture Diagram
 - 32-bit ALU Architecture Diagram
 - Special Design Detail of The Last ALU Unit
- Shifter
 - Overview
 - Shifter Control Signals Table

Attached Files

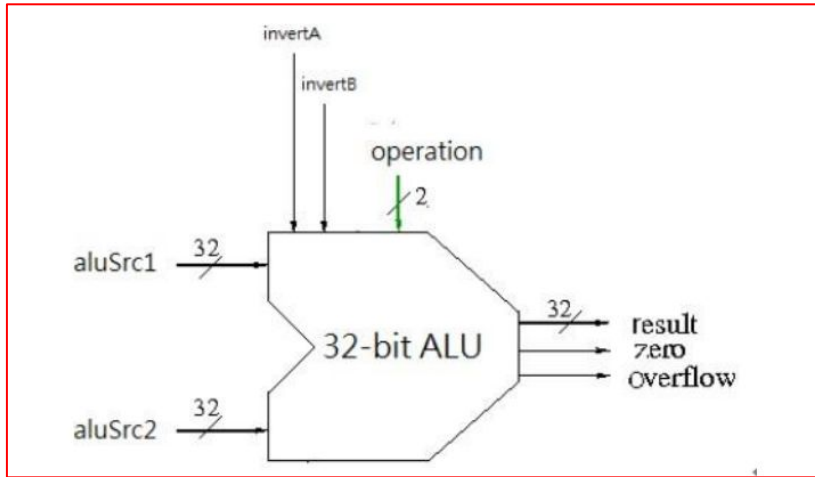
The attached files are composed of

- ALU.v
- ALU_1bit.v
- Full_adder.v
- Shifter.v
- TestBench.v

Please use these files to design your ALU and shifter architecture. You may create additional module (.v file) for your design if necessary, but we will use another testbench to test your code, so you should make sure it would work on our own testbench.

Arithmetic Logic Unit(ALU) Overview

The block diagram of the 32-bit ALU for this lab is shown in the following figure



You should generate three outputs of the ALU:

- **result:** the computation result
- **zero:** A 1-bit output control signal.
 - set to 1 when the result is 0.
 - set to 0 otherwise.
- **overflow:** A 1-bit output control signal.
 - set to 1 when the result overflows (add, sub).
 - set to 0 otherwise.

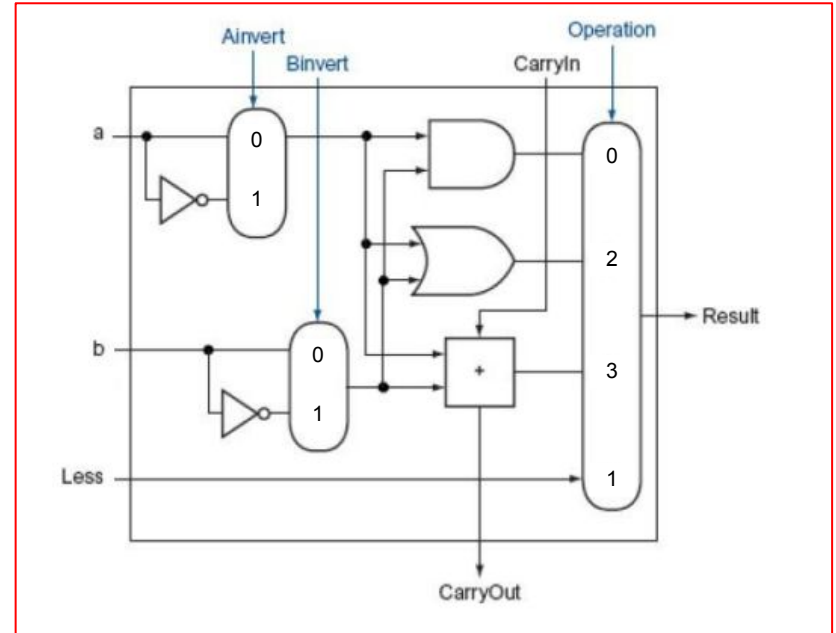
ALU Control Signals Table

The operations and corresponded control signals for the ALU are described in the following table. Note that all operations in the table are 32-bit operations. So, you should implement a typical 1-bit ALU first and then **build the 32-bit ALU by 32 1-bit ALUs**.

Operation	Control line			Value of result
	invertA	invertB	operation	
Add	1'b0	1'b0	2'b11	$\text{aluSrc1} + \text{aluSrc2}$
Sub	1'b0	1'b1	2'b11	$\text{aluSrc1} - \text{aluSrc2}$
And	1'b0	1'b0	2'b00	$\text{aluSrc1} \& \text{aluSrc2}$
Or	1'b0	1'b0	2'b10	$\text{aluSrc1} \mid \text{aluSrc2}$
Nand	1'b1	1'b1	2'b10	$\sim(\text{aluSrc1} \& \text{aluSrc2})$
Nor	1'b1	1'b1	2'b00	$\sim(\text{aluSrc1} \mid \text{aluSrc2})$
Slt	1'b0	1'b1	2'b01	$(\text{aluSrc1} < \text{aluSrc2}) ? 1 : 0$

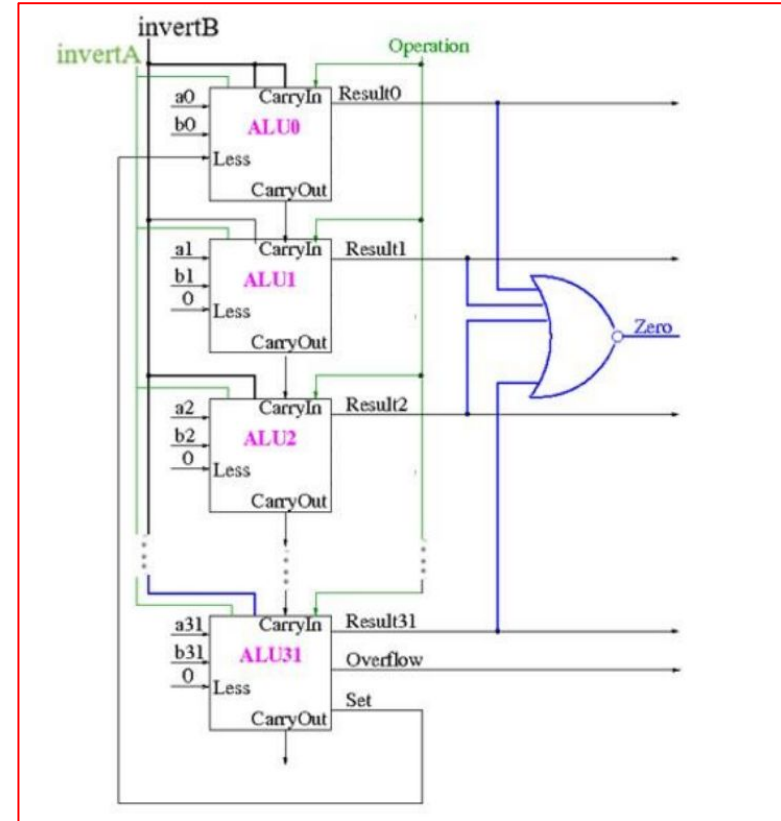
1-bit ALU Architecture Diagram

The diagram of a typical 1-bit ALU to be implemented in this lab is shown in the following figure.



32-bit ALU Architecture Diagram

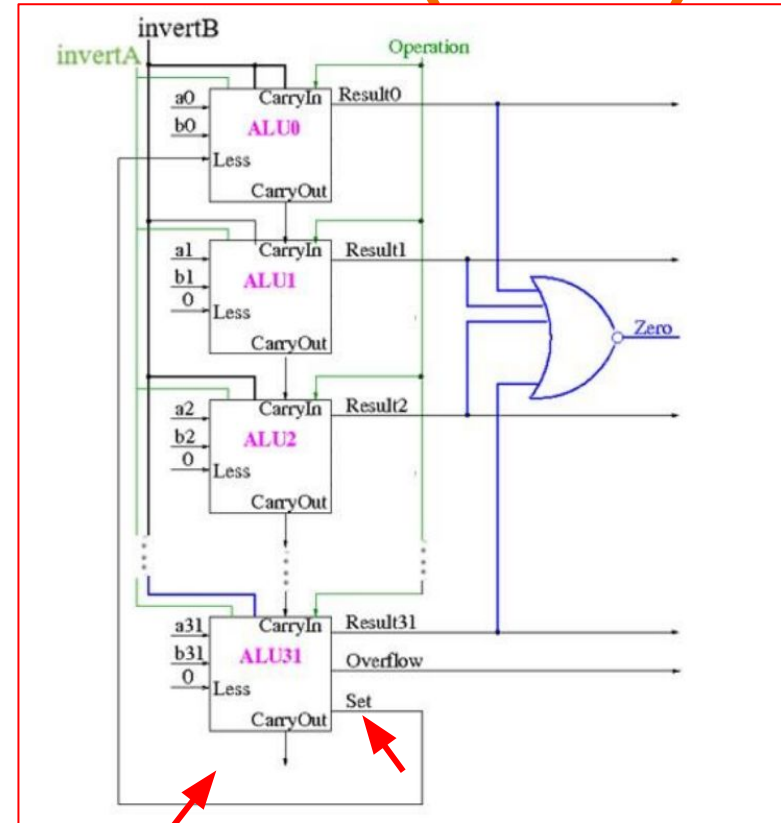
Each input signal of the whole 32-bit ALU will be connected to the corresponding input of 1-bit ALU. Moreover, carryOut of ALU_i will be connected to carryIn of ALU_{i+1} .



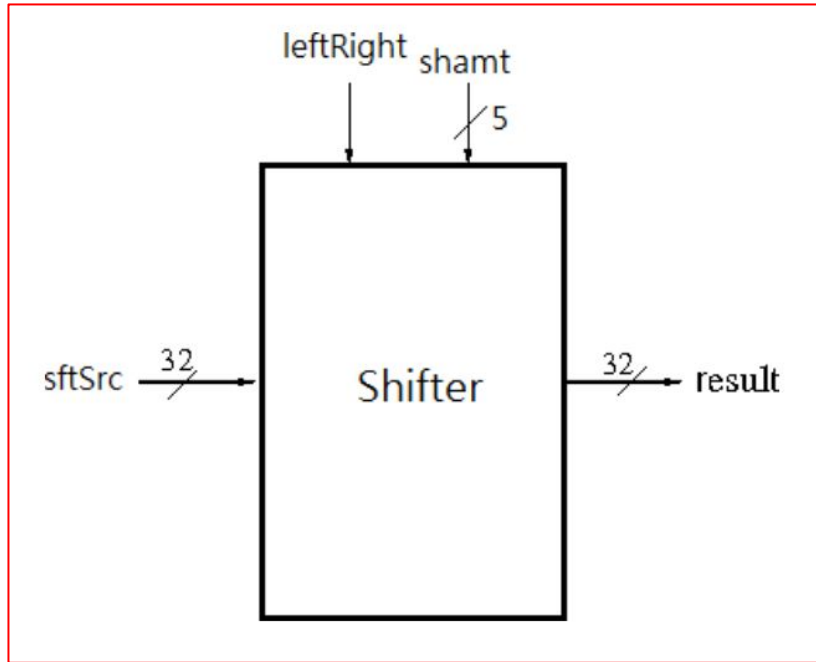
Special Design Detail of The Last ALU Unit(ALU31)

The design of ALU31 is different from that of other typical 1-bit ALUs. There are two additional signals generated by ALU31 and are described as follows:

- **set:** A 1-bit control line, it is generated by ALU31 and send to ALU0 as the “**Less**” signal.
- **overflow:** A 1-bit output control signal, it is set to 1 when overflow occurs (add, sub).



Shifter Overview



The Shifter Contains the following inputs and outputs:

- **sftSrc:** A 32-bit input data, which is the source data of the shifter.
- **leftRight:** A 1-bit input control signal.
 - Set to 0 —> logical right shift
 - Set to 1 —> logical left shift
- **shamt:** A 5-bit input data, which represents the number of bit positions to be shifted.
- **result:** A 32-bit output data, which represents the shifting result of the shifter.

Shifter Control Signals Table

Note that all the operations of the shifter are 32-bit, which may logical left/right shift by X bits position each time.

Operation	Control line		Value of result
	leftRight	shamt	
Shift right logical by X bits	1'b0	5'dX	sftSrc >> X
Shift left logical by X bits	1'b1	5'dX	sftSrc << X

Testing Data Format

- The format of the test data for ALU is shown below.

(invertA)(invertB)(operation)(aluSrc1)(aluSrc2)

For example:

00110000000000000000000000000000100000000000000000000000000011

means

add 2 3

- The format of the test data for shifter is shown below.

(leftRight)(shamt)(sftSrc)

For example:

100001000000000000000000000000001001

means

shift 9 left by 1

Environment

Install iverilog by

- Ubuntu: apt install iverilog
- MacOS: brew install icarus-iverilog

You can use TestBench.v and the testing data to test your code. Put all .v files and .txt(test and ans file) in the same path and use iverilog to compile TestBench.v.

Testing Method

File Structure:

```
> tree .
.
├── ALU_1bit.v
├── ALU.v
├── Full_adder.v
├── Shifter.v
├── Testbench.v
└── testcases
    ├── test_ALU.txt
    └── test_Shifter.txt

2 directories, 7 files
```

Command:

```
> iverilog Testbench.v
> ./a.out
Score: 9/9

Testbench.v:142: $finish called at 100000 (1ps)
```

- We will execute the above command to verify your results, covering 70 ALU test cases and 30 Shifter test cases, **1 point for each test case.**
- **Ensure the program compiles successfully by executing the command `iverilog Testbench.v`**

Grading Policy

- ALU
 - Each operation have 10 hidden cases, and you will get **1 points** for each correct testcase, totally 10 points for each operation.
- Shifter
 - Each operation have 15 hidden cases, and you will get **1 points** for each correct testcase, totally 15 points for each operation.
- We will use the same method to judge your result, make sure you can pass the testing script on your local environment.
- **Any assignment work by fraud will get a zero point.**
- **No late submission!**
- **Deadline: 7/26 23:55**

Submission

- The files you should hand in include:
 - **ALU.v**
 - **ALU_1bit.v**
 - **Shifter.v**
 - **Any other files you need in your design**
 - **No additional directory!**
- Compress the above file into one zip file, and name your zip file as **HW2_{studentID}.zip (e.g. HW2_0811510.zip)**
 - Make sure not to add an extra folder layer. You can use the following command to zip your files to ensure the format is correct.
- **Wrong format will have 10% penalty.**

```
> unzip -l HW2_0811510.zip
Archive: HW2_0811510.zip
  Length      Date    Time    Name
-----
      885  2024-07-05  14:30  ALU_1bit.v
     1682  2024-07-05  14:30  ALU.v
      427  2024-07-05  14:30  Full_adder.v
      403  2024-07-05  14:30  Shifter.v
-----
     3397
      4 files
```


Reference

- [ALU筆記](#)
- [Overflow Design](#)