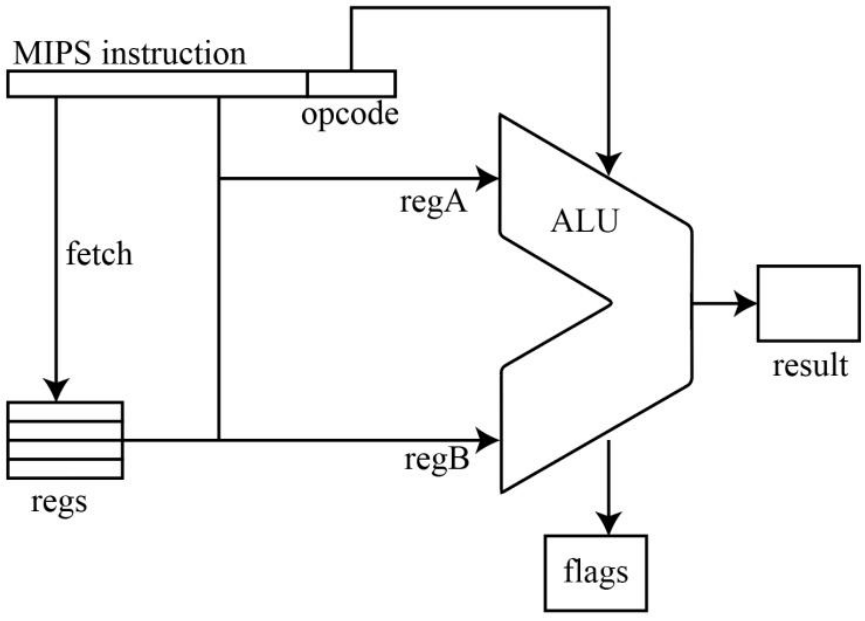
CSC3050 Project3 Report

Liang Mingrui 120090723

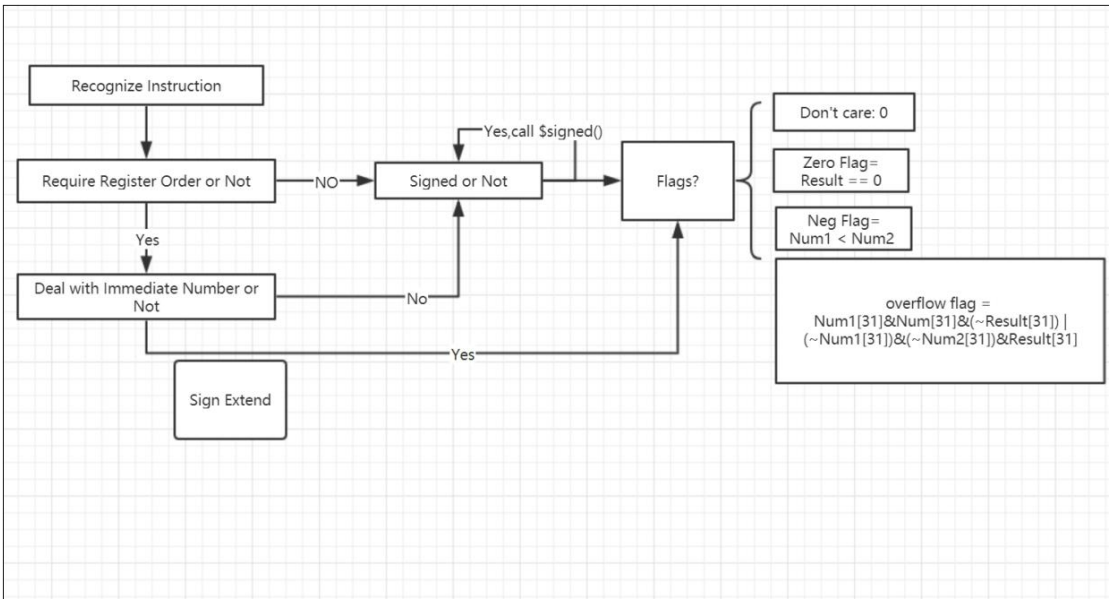
2022.3.29

Big Picture Thoughts

ALU is a unit that control lots of calculations such as add, sub, compare and so on. Usually, an alu has 32 bits, and it is consist of 32 one bit alus. All those operations I just mentioned is accessed in this way. Since it has to support many operations, there are surely some triggers to help alu to decide which operation to excute. In pratice, the operation of the alu is decided by the alu controler, a 4 bits code, which is mainly decided by aluop and functioncode. However, in this homework, the alu is not so complicate. It is directly decided by opcode and functioncode. Also, the address of register is only 0 or 1. Briefly, the mission of the whole process is to get the opcode and functioncode and pass it to the alu. And then alu will do the right operations accroding to the code. After that it will have the correct result and flags.



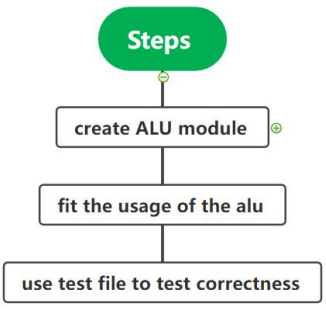
Data Flow Chart



The High Level Implementation Ideas

The whole project actually consist of many small parts, thus I divide them into small parts one by one for convenience. Firstly, the ALU.v mainly contain the excutions of the alu. And the test_ALU.v contain the test instructions to make sure the alu has correct excutions.

In order to make the whole project clearer, I draw the graph to show my big picture.



The graphic of the whole project

Following are the steps of the project:

Step 1: Create a alu module with five parameters.

Step 2: Approach all aims such add, sub, lw and so on in the alu module.

Step 3: Write a test file to test whethrt the alu have right function.

The implementation of the details

In order to satisfy the requirment, some function have to be built. There are some details of designing in the following.

ALU.v:

1. Add/Addu/Sub/Subu:

These functions can directly add or sub accroding to the value that is stored in the register. And then alu store the outcome into the result. Finally, the alu has a 3 bit flags accroding to the result and whether the overflow occures.

2. Addi/Addiu:

These function have to change the imm value to signed or unsigned accroding to the function. Second, alu has to see the instruction[25:21] to know which register to use. After that can the alu add them. And then alu store the outcome into the result. Finally, the alu has a 3 bit flags accroding to the result and whether the overflow occures.

3. And/Nor/Or/Xor:

These function can easily be accessed. I directly use the operation symbol in the iverilog for alu. I let alu store the outcome in the result. Basically, the flags of these functions are 3'b000.

4. Andi/Ori/Xori:

These function need to convert the imm value to the signed number and choose the correct register. After that alu store the outcome into the result. Also, the flags in these functions are 3'b000.

5. Beq/Bne:

These two functions directly let the result equal to 0. The flags[2] is decided by

whether regA quual to regB. Normally, flags[1:0] equal to 0.

6. Slt/Slti/Sltu/Sltiu:

These four functions are very similar. The alu only need to compare the value by the signed or unsigned number. And then let the flags[1] be 1 or 0. As for the result and the rest flags, there are all 0.

7. Lw/Sw:

These two function need to get the imm value from the instruction and then add the value in the register to it to get the address. The flags equal to 3'b000.

8. Sll/Sllv/Srl/Srlv/Sra/Srav:

These functions' flags are 3'b000. the function will shift the value in the register accroding to the signed , unsigned, register address and so on. Finally, alu store the outcome to the result.

Test_ALU.v:

This file only create some instructions to test whether my alu can operate correctly. And if my alu can operate correctly, this file will show the success.