**Project3-Report**

**120010058**

**ALU FUNCTIONALITY**

ALU executes to do mathematical and logical operation in CPU. When getting the result, some flags like negative flag, zero flag, and overflow flag will be recognize in the whole system which gives system signals what to do next. For example, overflow flag tells cpu there is overflow and the result is not the correct result.

When ALU executes logical operation, it will compare two registers or one register and the immediate bit by bit. For example, in AND logic, when two bits are both 1, the corresponding bit of result will be 1.

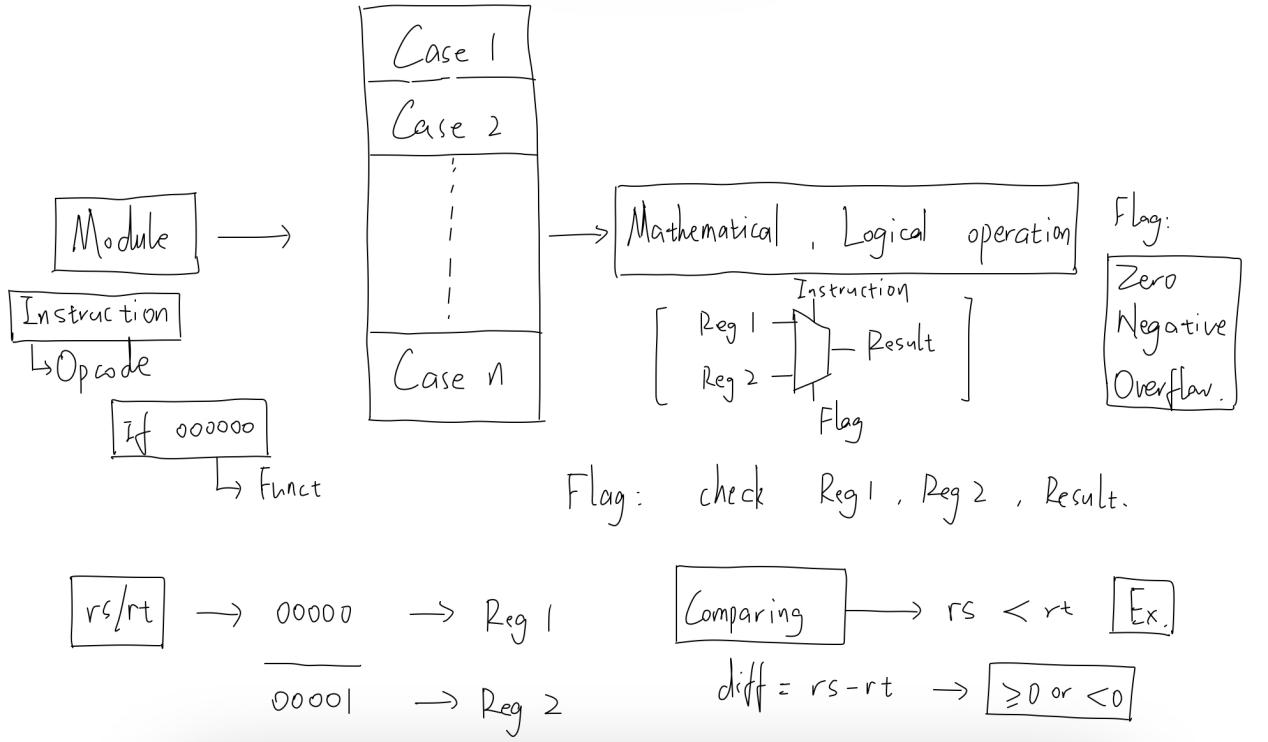
In some left movement or right movement order, ALU will do this execution and allow CPU to do the following things.

In ALU, there should be a aluctr which controls what execution ALU should do. The input will be two registers (and in p3 instructions) in 32-bit mode. Sometimes clk allows continuing output. The output of ALU is the result of execution between registers and immediate, as well as the flags. Negative flag is 1 when result is negative. Zero flag is 1 when result equals 0. Overflow flag is 1 if the sign of result is not correct. For example, if reg A and reg B are all positive but the result is negative, there is an overflow.

In project 3, three inputs: instructions, reg 1 and reg 2 are put in the alu module. By recognizing different sections in instruction, opcode and function of I-type MIPS code will be got. Then by recognizing correct MIPS code, the module will execute corresponding operation. The different opcode and function will be divided case by case. And for two result as inputs, two units of address will be used and therefore 00000 and 00001 will occupy the position of rs or rt part in instruction, decided by which register the MIPS code will use. By recognizing corresponding binary code of registers, which reg will be used as rs or rt is decided. If binary code of rs is 00000, then reg1 will be rs, and if that is 00001, reg2 will be rs. For some special instructions, negative flag or overflow flag or zero flag will be recognized.

In some special cases, like the instructions which requires comparing between two registers, like beq, alu will do a subtraction between two registers. Whether the result is negative or non-negative will take the place of reg1 < reg2 as the check point of the following operation.

**DATA FLOWCHART**



**HIGH IMPLEMENTATION**

To recognize different instructions, the input instructions should be divided into different part like opcode and function. From 31 to 26 bit, opcode is chosen and used to recognize R type and different I type instruction. When opcode is 000000, position from 5 to 0 bit is recognized as function. Different function represents different instructions. Case operation recognizing function in case operation recognizing opcode is utilized.

In each instructions (cases), if the operation is not add or logic operation between registers, which register is being used as rs or rt is important. By analyzing the binary code of rs or rt in 32-bit instructions, corresponding register is used as the main operator. (00000 for register 1 and 00001 for register 2). For some special cases, like add, overflow flag is set to be one if the final result has binary form in 33 bits which is more than 32 bits.

The variable result is set in each case and out of the always part, it is transferred to **wire** final\_result as the output of ALU part. In flag setting part, variables are set and connected to form the **wire** flag.

**IMPLEMENTATION DETAILS**

In order to get different flags, special tricks are used.

Overflow: Analyzing sign-bit of two input registers and result. If sign-bits of two registers are both 1 or both 0, while that of result is opposite, there is overflow (meaning positive + positive = negative or negative + negative = positive).

Negative: when there is comparing between two registers, difference between two registers is calculated. Ignoring overflow, if the sign-bit is 1, meaning negative, negative overflow is 1. If it is 0, meaning larger than 0, negative flag is 0.

Zero: when there is comparing between two registers, difference is got and if the result equals to 0, zero flag is set to be 1.

When analyzing I-type instructions, immediate sometimes is significant. When doing extension, I use a variable sign to store the sign-bit of 16-bit immediate (instruction[15:0]). Then according to instructions, choosing sign-extension or zero-extension, I connect sign-bit and immediate together.