Project 3 report

120090643 陈启旭

1. Big picture thoughts and ideas

   An ALU is the device that performs the arithmetic operations like addition and subtraction or logical operations like AND, OR.

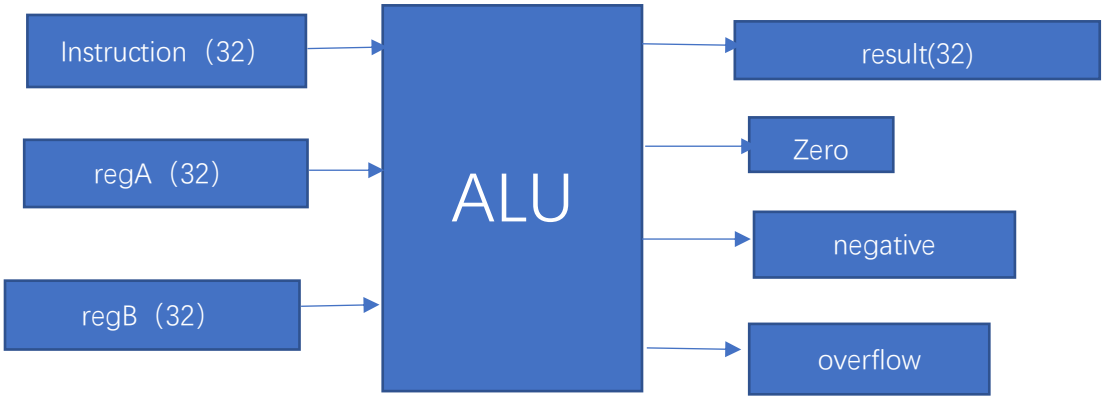
   This project is a simple CPU which supports simple instruction parsing, Register Value fetching, and ALU functions. After parsing the machine code of MIPS instruction, ALU receives three fixed inputs, opcode (4 bit), regA (32bits), and regB (32 bits), and outputs two signals, result (32 bits) and flags (3 bits).

   There are three main processes for the simple CPU:

   A. The first part of the CPU will parse the inputted instruction to find where opcodes, rs, rt, rd, shamt and function code locate.
   B. The second part of the CPU will fetch the value stored in the register according to the address of regA and regB.
   C. The third part of the CPU will execute the MIPS instructions.

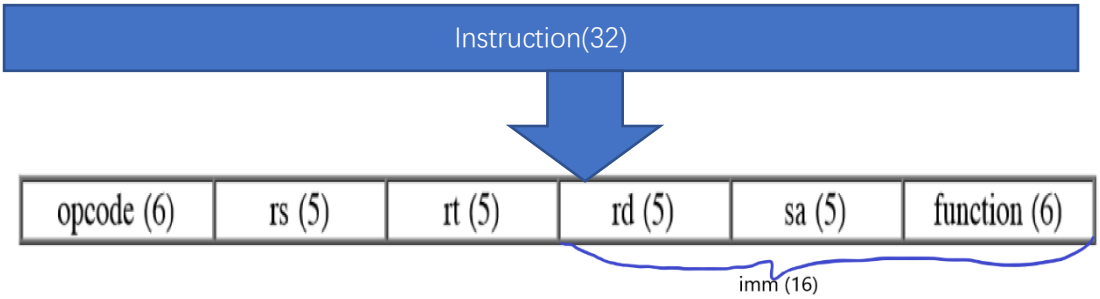
2. Data flow chart

a) Overview of the data flow chart of ALU

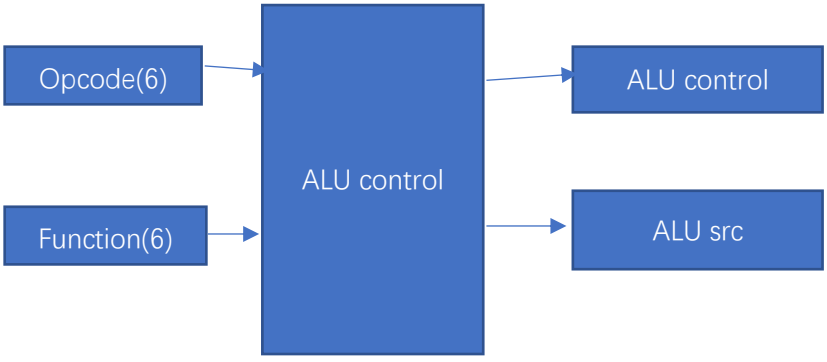


b) Inside the ALU module

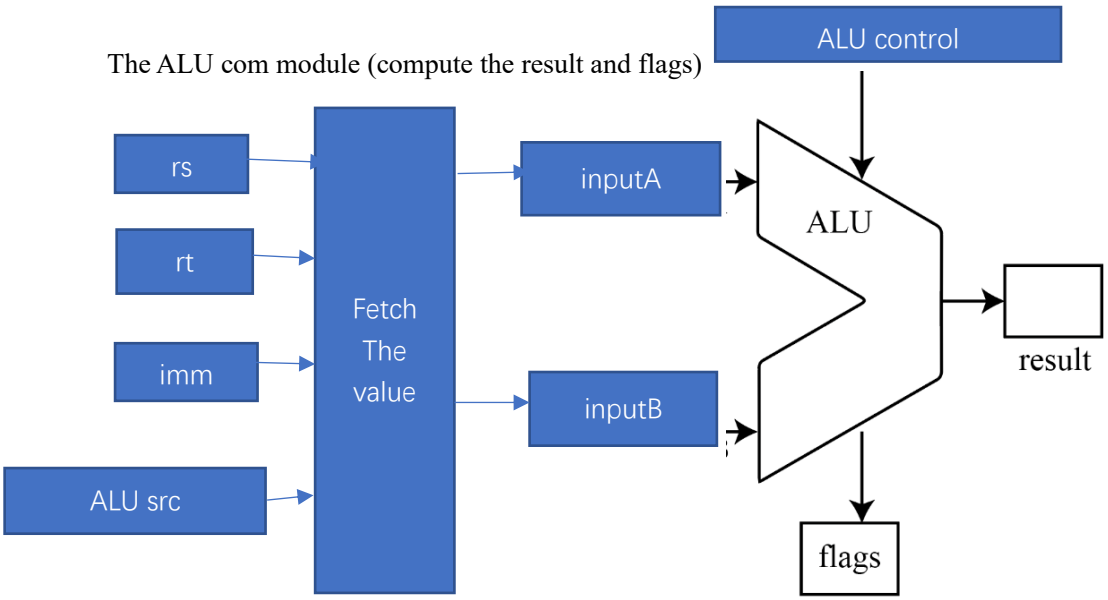
Parsing the instruction



The ALU control module (produce ALU control and ALU src)



The ALU com module (compute the result and flags)



Flag has three bits: the first bit is zero flag, the second bit is negative flag, the third bit is overflow flag.

3. High level implementation ideas
   a) Overview
      The simple CPU include two modules: ALU control, ALU component.
   b) Parsing the instruction
      The first 6 bits will be the opcode;
      Next 5 bits will be address of rs (00000 or 00001)
      Next 5 bits will be address of rt (00000 or 00001)
      [10:6] will be the shamt, the program will extend it to 32 bits with 0
      [5:0] will be the function code

      [15:0] will be the immediate number. The code will extend it with its MSB, which is called imm1; If extended with 0, it is called imm2.
   c) ALU control module
      This module will produce the ALU control and ALU src code for the instruction
      This module will read in the opcode of the instruction to determine which type the instruction is. If the instruction is R-type, then ALU src will be set to 0; If the instruction is I-type, then ALU src will be set to 1
      After finding the value of ALU src, this module will find the ALU control for the instruction, using the phase: case; For example:

```
case(funct)
                6'b100000://add
                begin
                    ALUctr=4'b0010;
                end
```

If the instruction is andi, ori, xori and nori, the program will generate a one bit code called signextend whose value is 1. For other operation, signextend is set to 0.

d) Fetching the value stored in the register

For this part , the program will check the rs, rt, shamt, imm code got from instruction parsing.

For this part, the program will find determine which will be inputted to the core of the ALU. The program will first check the ALU src got from the ALU control.

If the ALU src equals to 0, the program will check the value of regA and regA. The program will also check the address code of rs and rt. As the address of regA is 00000, the address of regB is 00001, if the value of rs is 00000 and the value of rt is 00001, the value of regA will be given to inputA (equivalent to rs in R type instruction) and the value of regB will be given to inputB(equivalent to rt in R type instruction).

If the ALU src equals to 1, the program will set the value of inputB to be the immediate operand. If signextend code is 1, then inputB equals to imm1; Else, the inputB is set to imm2.

e) The ALU com module

This module will perform the core function of the ALU.

This module will read the inputA and the inputB from the previous value fetching process.

The ALU will read the ALU control code to determine which function will the ALU manipulate. The program realizes this process using case.

For example, if the ALU control code is 0110, the ALU com will calculate the value the sum of inputA and inputB. Then it will assign this sum to the wire "result".

```
case(aluctr)
    4'b0010://add
    begin
        result = A+B;
```

Additionally, after calculating the result part, this module will calculate the negative, zero and overflow flag.

If the instruction is slt , slti , sltiu , sltu and the result is negative, the negative flag will be set to 1;

If the instruction is beq, bne and the inputA and inputB is equal, the zero flag will be set to 0;

If overflow occurs in the calculation, the overflow flag will be set to 1

f) test bench

The test bench is included in test_ALU.v file. It will input the instruction, value of the regA and the value of the regB.

It will then order the program to output the result and the value of flags.

4. The implementation details

Nothing ver

a) The overflow flag

For instruction add and sub, if overflow occurs in these calculation , then the overflow flag will be set to 1.

The way to judge whether overflow occurs according to the textbook is to compare whether the carryin and the carryout for the last bit is the same. However, this method will require many if and else. The simple way to do this is to use a combinational logic:

Take add as example:

If one input is positive and the other negative, then overflow cannot occur obviously.

If both input is positive, that is, if and only if the MSB of the result is 0, then overflow occurs.

If both input is negative, that is, if and only if the MSB of the result is 1, then overflow occurs.

The circuit looks like this for add operation:

```
overflow = (A[31]^B[31]) ? 0: (result[31]^A[31])
```

The circuit looks like this for sub operation:

```
overflow = (A[31]^B[31]) ? (result[31]^A[31]):0;
```

b) Some special instructions:

Although the program has to support 16 kinds of instructions, some instructions share the same ALU function.

For example, the instruction of beq will do the same performance as sub.

For the instruction of slt, slti, sltu,sltiu , the result will output 32'b1 if value of rs is smaller than rt, the negative flag's value will also be 1.

I don.t think other aspects are too tricky.

C) For the testbench of the program, it will print out the instruction(in hex), opcode(in hex), function code(in hex), the inputted regA or regB(in hex), ALUctr, ALUsrc, result(hex), two operands for the calculation(inputA and inputB，in hex), zero, negative, overflow flags and the three bit flags(in binary)