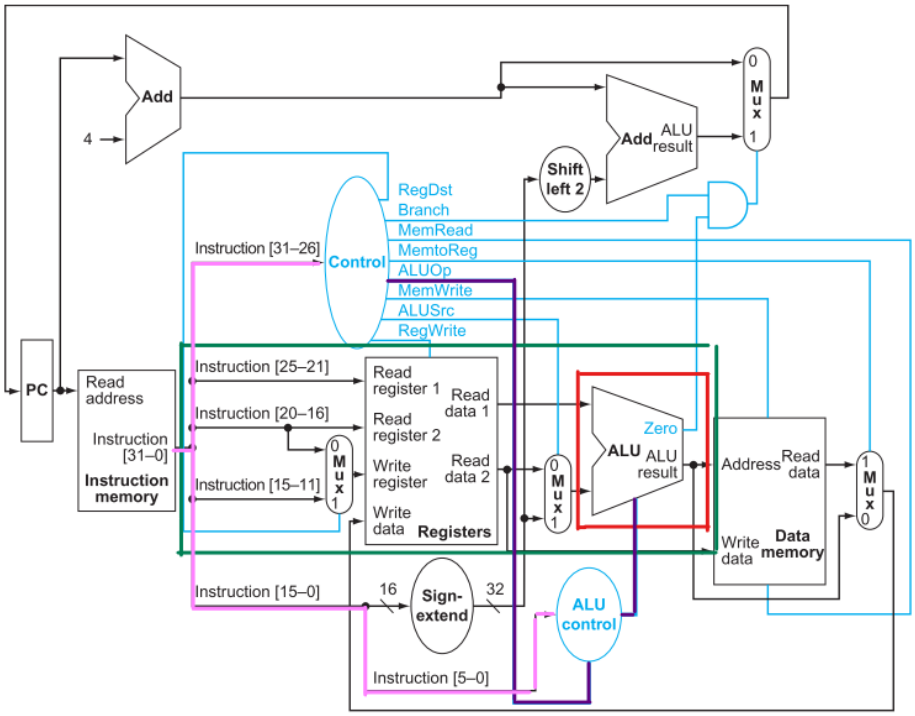
HM3 report

120090694 Tong Zhen

ALU design idea

In this assignment, we were asked to handle instruction and design ALU(red). ALU can be used for algebraic and logical operations. What functions should be determined by ALUOp and ALU Control（purple).These corresponding `ALUOp` and `ALU Control` correspond to op and imm in Instruction [31:26] and [15:0](pink). So what we need to do is all marked with color.

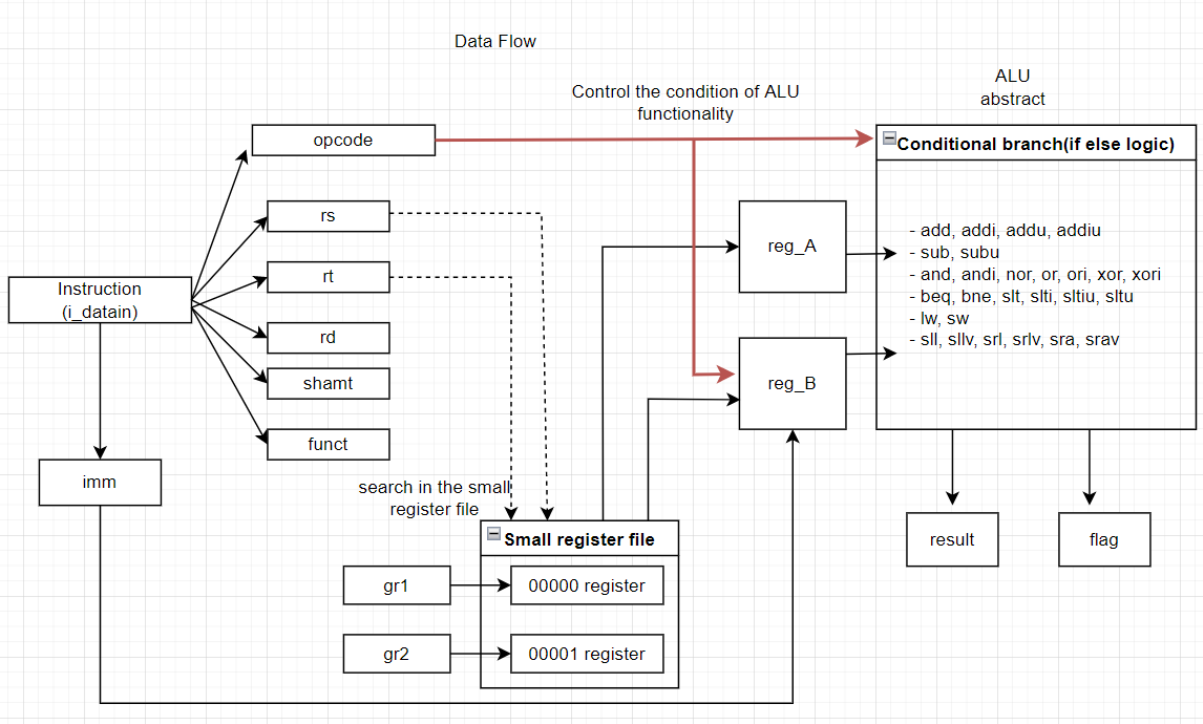


ALU has two inputs reg_A and reg_B and two outputs Result and flags. Actually, it has another input called ALU control. But in my verilog program, I've replaced this part with the if else logic, which has the advantage of not having to worry about the tedious ALU op. Another benefit is the code readability increased.

Implementation Ideas & Data Flow

Reg_B is the second input ot the ALU. Its value depends on the ALU control, which in this project is abstracted to the corresponding value of Opcode（red line). The

main part of the ALU implementation is a set of if else condition. After determining reg_A, reg_B, ALU condition, we get the calculated reg_C and judge the flags.



Detailed Implementation

Extra bit for signed overflow check

I used double bit judgment to check overflow. Originally there was only one sign bit, but now we add another 1-bit `reg` called `extra` for convenience.

```
{extra, reg_C} = reg_A + reg_B;
overflow = extra ^ reg_C[MSB];
```

Now we only care about `extra` and `reg_C[MSB]`

When the `{extra, reg_C}` = 01, the result is positive, so there's a positive overflow.(Positive numbers are too big. When the `{extra, reg_C}` = 10, the result is negative, negative overflow occurs (negative numbers are too small). When the `{extra, reg_C}` = 00 or 11 just didn't overflow.

(we are not asked to check the unsigned overflow addu, addiu,ect.)

Immidiate extension

First, we need to clarify a very confusing matter. According to the ISA of MIPS. `Addiu` and `sltiu`, two unsigned I-type instruction, are not meant to be unsigned when extending the IMM. It is executed according to the unsign rule in the extended operation. In MIPS `imm` extension rules, the high 16-bit filling is completed according to the MSB of `imm`

```
imm_32 = {{16{imm[15]}}, imm};
```

However, in the logical I-type instructions (such as `addi`, `ori`, `xori`), the imm extension bit is specified as 16'b0 **(logic related instructions are the special case of IMM extension)**

Logic instruction

& || ^ ~ are used in the logic operation.

Signed compare: `slt` set less than

The if the reg_A is less reg_B then the result should be assert 1. And the negative flage is assert 1.

My program determined less by subtraction.

In `slt` , there are 4 cases.

- reg_A[MSB] is negative and reg_B[MSB] is positive, then negative flag = 1
- reg_A[MSB] is positive and reg_B[MSB] is negative , then negative flag = 0
- reg_A[MSB] is positive and reg_B[MSB] is positive , compare
- reg_A[MSB] is negative and reg_B[MSB] is negative , compare the twos complement

Shift rt rather than rs

According to the textbook,

> "*Shift register rt left (right) by the distance indicated by immediate shamt or the register rs and put the result in register rd.* ***Note that argument rs is ignored for sll, sra, and srl***."

So, when we are doing shifting, we are shifting the rt, which is **reg_B.**

Shift variable upper limit

For example, in `srav`, `srlv` , `sllv` directive takes a maximum of 5 bits of variable instead of all of them. This operation affects the last bit of the move because $11111_2 = 31_{10}$ .For a 32-bit data, the last bit can never be moved out at one shift operation. Therefore no matter how big the variable is, we only care about the last 5 bit of it in order to do the shift.

Save word & load word

these two instruction use ALU as a adder to fetch a target memory address

Input and Output Description

input

In the `test_ALU.v` there are 3 input `i_datain` , `gr1` , `gr2` . The `gr1` is the data of register whose address is 00000 and `gr2` is the data of register whose address is 00001.

`reg_A` : The mapped data input for rs

`reg_B` : The mapped data input for rt

output

`result(c)` : hex. A 32-bit number as the result of the ALU operation

`flags` : bin. A 3-bit number for zero, negative, overflow mark.

Auxiliary

`imm` : hex. A 16bit number

`shamt` : shift amount

Supplimentary

https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf

Computer Organization and Design T H E H A R D W A R E / S O F T W A R E I N T E R F A C E