CSC3050 Project 3

April 10, 2022

Build an ALU

Student ID: 120090266

Student Name: Feng Yutong

This assignment represents my own work in accordance with University regulations.

Signature: Feng Yutong

1 Overview

The task of this assignment is to implement the Arithmetic and Logic Unit(ALU), a computation unit in CPU. Basically, it supports simple instruction parsing, register value fetching and ALU functions. The program consists of two parts: AUL_test.v and ALU.v. The big picture thoughts and ideas are as followed:

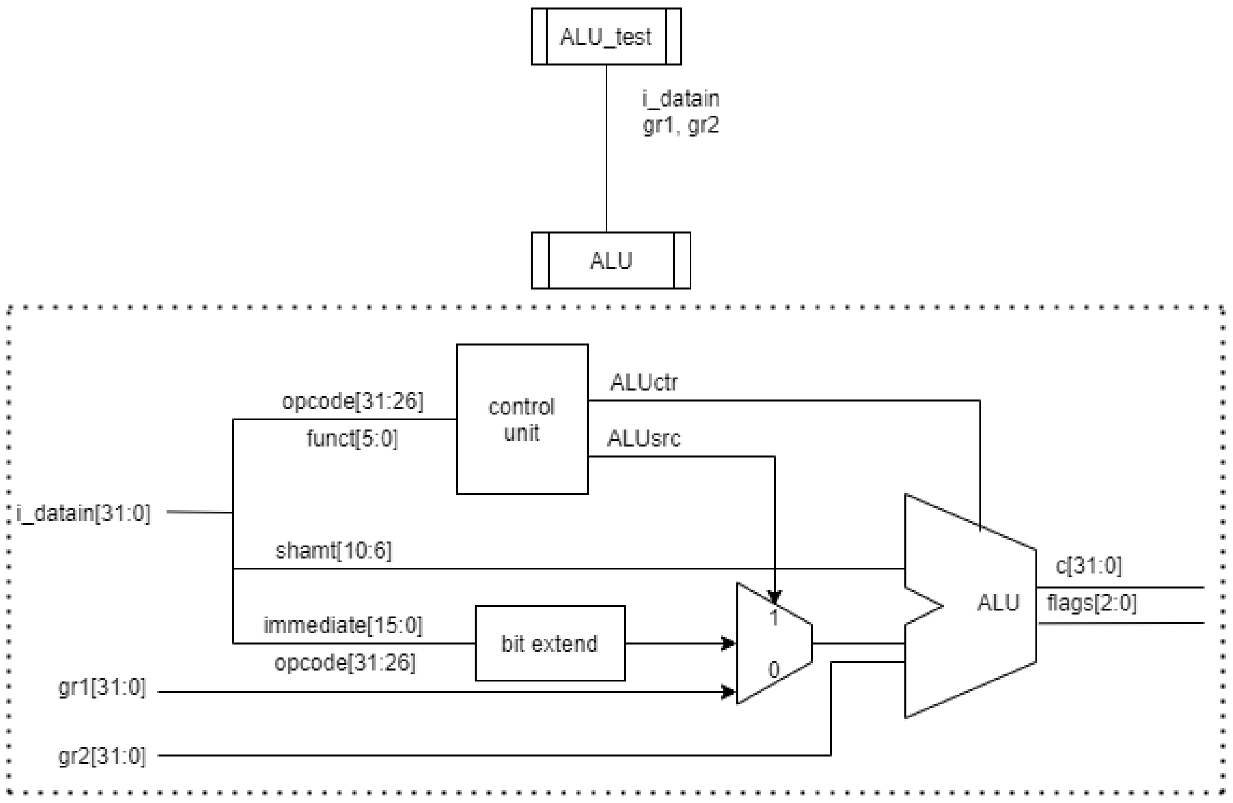

Figure 1: Diagram for ALU

2 High Level Implementation Ideas

test_ALU

This part provides testbench and calls ALU according to a simulated time clock. It outputs result for checking. Note that reg_A and reg_B stores the actual value taken by ALU for calulation, **which means they may not be equal to the value stored in gr1 and gr2.**

control unit

This part will generate ALUctr and ALUsrc according to opcode and funct. It applies "merge like terms" ideas. Since it is a choosing process, using switch can make the code neat. ALUctr and ALUsrc table are listed below.

ALU

ALUctr	instructions	ALUctr	instructions
00000	add,addi	01001	nor
00001	addu,addiu	01010	or,ori
00010	sub	01011	xor,xori
00011	subu	01100	slt,slti
00100	srl	01101	sltu,sltiu
00101	srlv	01110	lw,sw
00110	sra	01111	beq,bne
00111	srav	10000	sll
01000	and,andi	10001	sllv

Table 1: ALUctr and instructions

ALUsrc	instructions
0	add,addu,sub,subu,and,nor,or,xor,slt,sltu,sll,sllv,sra,srav,beq,bne
1	addi,addiu,lw,sw,andi,ori,xori,slti,sltiu

Table 2: ALUsrc and instructions

This part will excute different ALU functions. Since it is a choosing process, using switch can make the code neat. The address of gr1 and gr2 is 00000 and 00001. Reg_A and Reg_B will take value of gr1 and gr2 according to the instruction. Note that if ALUsrc equals one, Reg_B will take value of the bit extend of immediate. Result is store is c[31:0], flags[2:0] stores zero flag, negative flag and overflow flag from left to right.

For the overflow flag, only add , addi , and sub instructions are considered. For the zero flag, only beq and bne instructions are considered. For the negative flag, only slt , slti , sltiu , sltu instructions are considered. Other instructions set to the default value.

3 Implement Details

Bit extend

16-bit immediate should be extended to 32 bits. The most significant bit decides its sign. Sign extend requires to fill the former 16 bits with sign bit:

```
im = {{16{i_datain[15]}}, i_datain[15:0]};
```

Zero extend requires to fill with zero bit:

```
im = {{16{1'b0}}, i_datain[15:0]};
```

ALU function

add, addu, sub, subu For instructions that do not consider sign bit, use unsigned register reg_Au and reg_Bu to calculate. Otherwise, use signed registers. To detect overflow, check whether carry in of MSB of reg_C equals the carry out of MSB of reg_C. For simplicity, concat

MSB with its orignial input, the first bit and second bit can represent carry out and carry in respectively since both adding the same valude do not affect final comparison.

```
        // add
{fg, reg_C} = {reg_A[MSB], reg_A} + {reg_B[MSB], reg_B};
overflow = fg ^ reg_C[MSB];
```

lw,sw Since this program only does ALU operation and does not simulate memory, ideal ALU funciton is add operation. However, the essence of them is different so they are not merged to add.

beq, bne Since this program only does ALU operation and does not simulate branch, ideal ALU function is subtraction operation. However, the essence of them is different so they are not merged to sub.

sra, srav, srl, srlv In verilog, >> is a binary logical right shift, while >>> is a binary arithmetic right shift. They can be used to differentiate.

slt, slti, sltu, sltiu This program set reg_C = reg_A < reg_B since results of the four instructions do not matter. This way can also skip overflow problem may encounter.

logical operation For logical operation, just directly use logical operator. They are as followed.

```
reg_C = reg_A & reg_B; //and, andi
reg_C = reg_A | reg_B; //or, ori
reg_C = ~(reg_A | reg_B); //nor
reg_C = reg_A ^ reg_B; //xor,xori
reg_C = reg_B << reg_A; //sll,sllv
```

record changes In test_ALU.v, timescale(1ns/1ps) is set and use **monitor** to output the result synchronous. In ALU.v, **always @(i_datain,gr1, gr2)** is used to perform ALU function when input is changed.

4 Compile and run

4.1 Complie and run

To run the program, move to the folder of ALU.v and enter **make** in the terminal to run the program. Results are displayed in the terminal.