CSC3050 Computer Architecture

Project 2 report

Name: Xiao Nan

Student ID: 119010344

Date: 2021/4/8

The Chinese University of Hong Kong, Shen Zhen

1. Introduction:

In this project I implemented a simple CPU containing the function of instruction parsing, register value fetching and basic ALU functions using Verilog language. Also, I implemented a testbench to test the functionality of the simple CPU.

1. Big picture:

In general, ALU performs the required operations on the register data specified in previous steps and output the results: First, the ALU would receive the data in the registers provided from register file. Then it performs operations on the data according to the control signal given from ALU control unit. At the same time, it detects abnormal operations such as addition or subtraction overflow and when they occur, it would output flag signals such as overflow flag to indicate the abnormality. Then ALU outputs the results of the operation. The result can be a value, like an arithmetic result calculated from addition or subtraction operation, or a control signal like “zero” flag, activated after judging certain condition like whether or not the branching should occur.

1. Data flow chart:

The data flow chart is shown in figure 2 below:

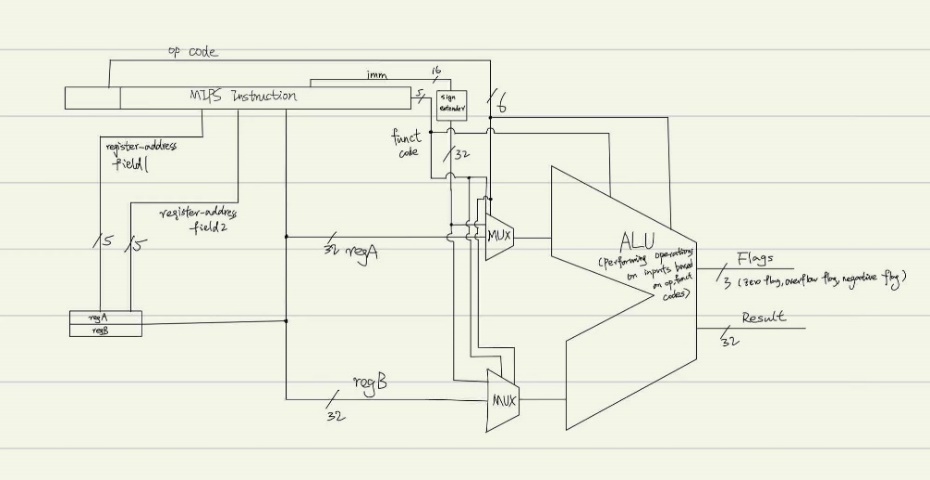


Figure 2: Data flow chart of ALU

1. Implementation ideas:

I break down the problem into two main parts: ALU unit and test unit.

The ALU unit consists of 3 parts: Instruction parsing, register fetching and arithmetic operation. In the first part, I categorize different instructions according to its op code and funct code. Then in the register fetching part, I would make the program fetch the values stored in corresponding registers according to the register address specified by the certain part of the instruction. At the same time, the program would fetch immediate number from the instruction if needed. Then in the arithmetic operation part, the program would perform arithmetic operations on the operands such as two registers or a register and an immediate according to the type of the instruction specified by its op and funct field. At the same time, the flag would be set if certain conditions (like a particular instruction being executed, overflow or negative result) are satisfied.

A test unit testing the correctness of the ALU module is also implemented. First a ALU testing module is instantiated. Then for every certain period, it would generate inputs for the series of instructions and get the corresponding result. Then it would display the result and compare with the desired result and if correct, the test is passed and in the next period the next test for the next instruction would be carried out. For special tests like “add”, “slt” which involve flags, two types of tests would be carried out: One with the flag set and one without the flag set.

1. Implementation details:

In the ALU module when parsing instruction, I use “switch case” statements to categorize different types of instructions according to their op code and funct code. Also, when fetching registers I used a two-dimensional array to store registers according to their addresses so that I can index the values stored in those registers through the register address (For example, regArray[0] meaning the value stored in register with address 00000). In arithmetic operation part I used “$signed()” in SystemVerilog for the addition of signed number and immediate number to ensure sign extension.

In the test module, I defined a task “showstate()” to printing out the current values in those registers and the instruction for the convenience of testing. Then I test each type of instruction by inputting each instruction and register values sequentially with 10ns delay and printing result with 15ns delay after each sequence of inputs using “showstate()” task. Sample results of tests with flags are shown in figure 4.2.

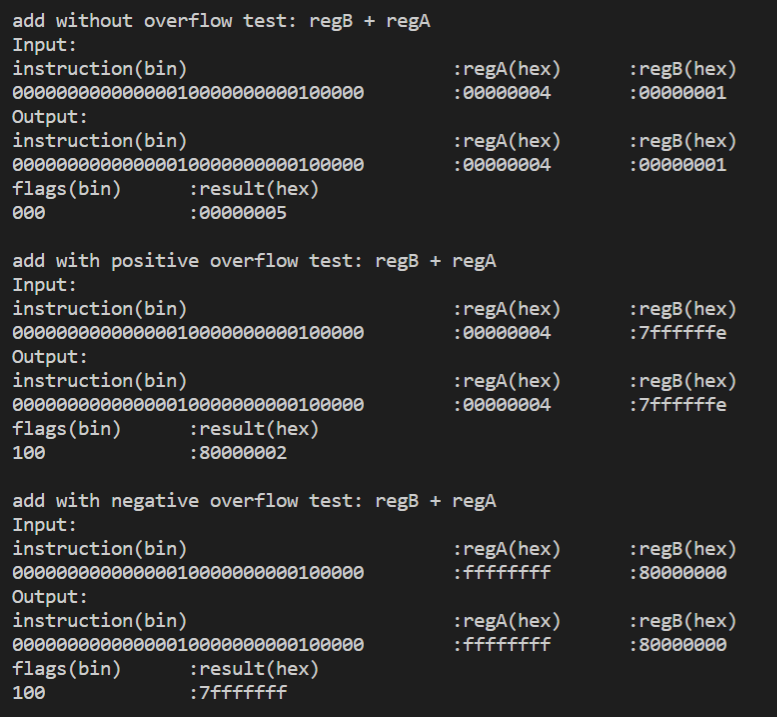


Figure 4.2: Sample tests with flag affected

1. Conclusion:

In project 2 I implemented a simple CPU module which can parse instructions and perform some ALU functions. Also, I implemented a testbench to test the correctness of the module. I divided the project into four parts: Instruction parsing, register fetching, arithmetic operations and module testing. In particular, I used a register two-dimensional array for the convenience of fetching register values using its address in ALU module, and a “showstate” task for the convenience of printing out testing result in the test module.