Project 3

1. Overview

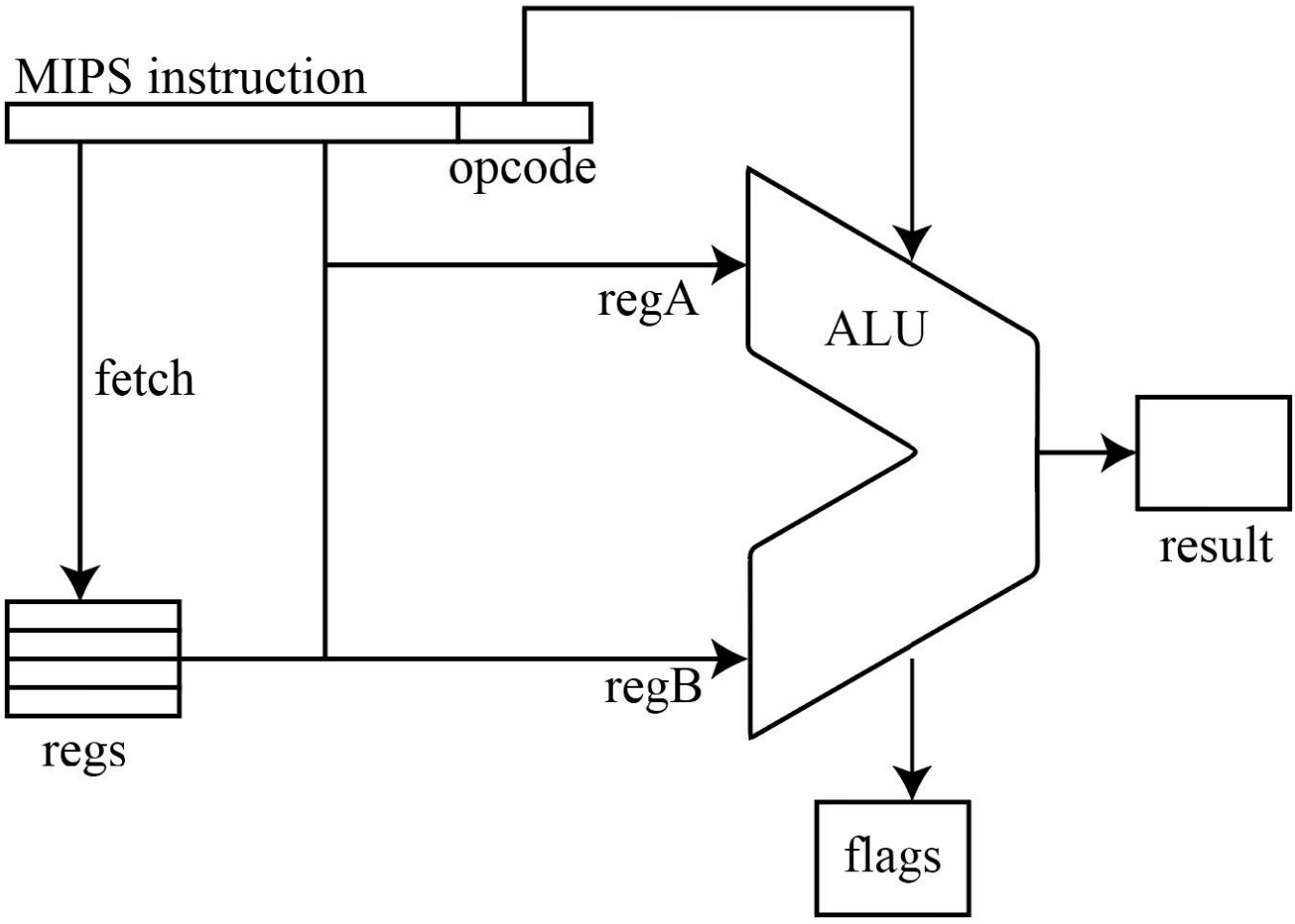
In project 3, you are required to implement an important computation unit in CPU, the Arithmetic and Logic Unit (ALU), using Verilog language. You are going to implement the functionality of ALU module by yourself. To reduce the learning curve, a poorly written template will be given.

2. Requirements

Before start, you should review the content of ALU and learn the `Verilog` language by yourself. The following gives a brief review of ALU.

2.1 Simple CPU

Basically, you are going to implement a simple CPU which supports simple instruction parsing, register value fetching, and ALU functions. As shown in the diagram below, when parsing the machine code of MIPS instruction, ALU receives three fixed inputs, `opcode (4 bit)`, `regA (32 bit)`, `regB (32 bit)`. When the instruction is excecuted, ALU outputs two signals, `result (32 bit)`, and `flags (3 bit)`.



2.2 Simple Register Fetch

Since we are using up to 2 registers as the inputs, the size of the register array is defined as 2. It means the register address in the MPIS code can only be one of `00000` and `00001`. For example, instruction[25:21] is the address of rs. However, you can manually change the value in the register during the testing.

2.3 ALU Functions

You are required to support the following MIPS instruction in your ALU:

> - add, addi, addu, addiu
> - sub, subu
> - and, andi, nor, or, ori, xor, xori
> - beq, bne, slt, slti, sltiu, sltu
> - lw, sw
> - sll, sllv, srl, srlv, sra, srav

2.4 Some Specifications

For the overflow flag, you should only consider `add`, `addi`, and `sub` instructions. For the zero flag, you should only consider `beq` and `bne` instructions. For the negative flag, you only need to deal with `slt`, `slti`, `sltiu`, and `sltu` instructions. For example, at any time, when you execute `addu` instruction, the overflow flag will remain zero. And for `subu` instruction, even the result is less than 0, the negative flag will remain zero.

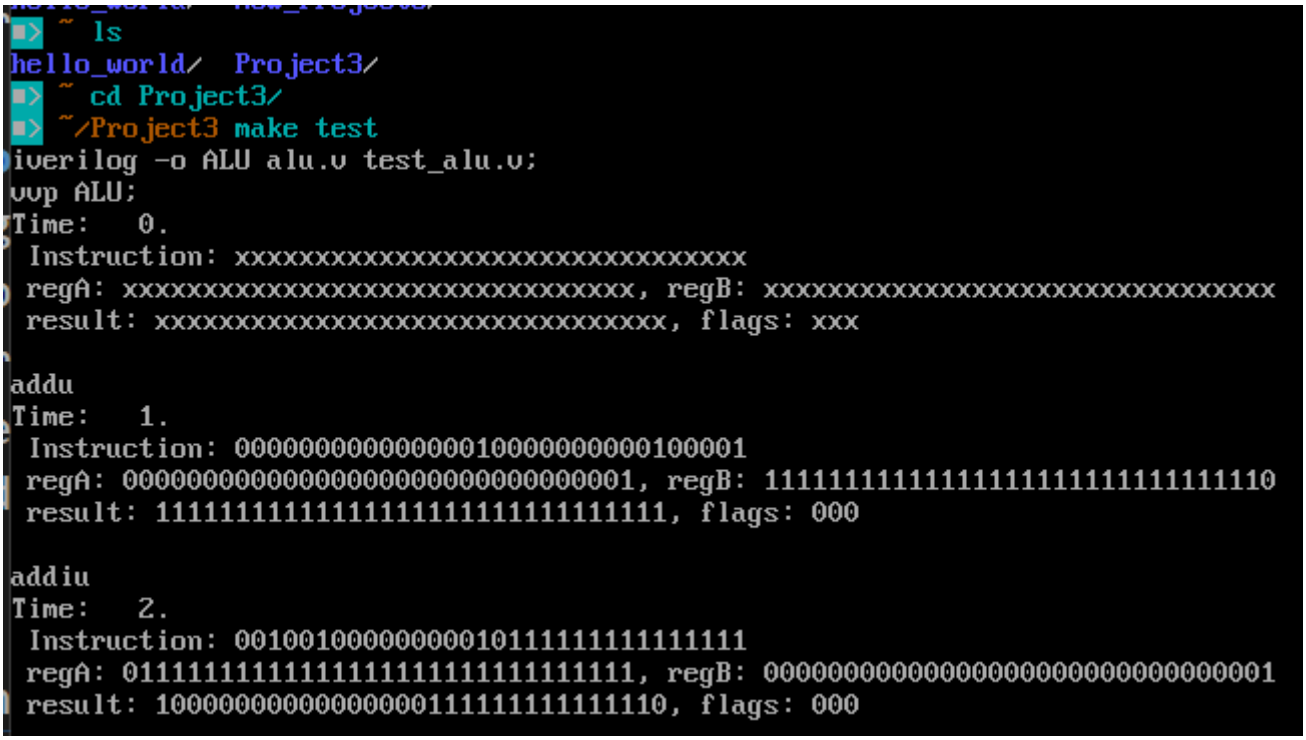
3. How to Use the Given Template

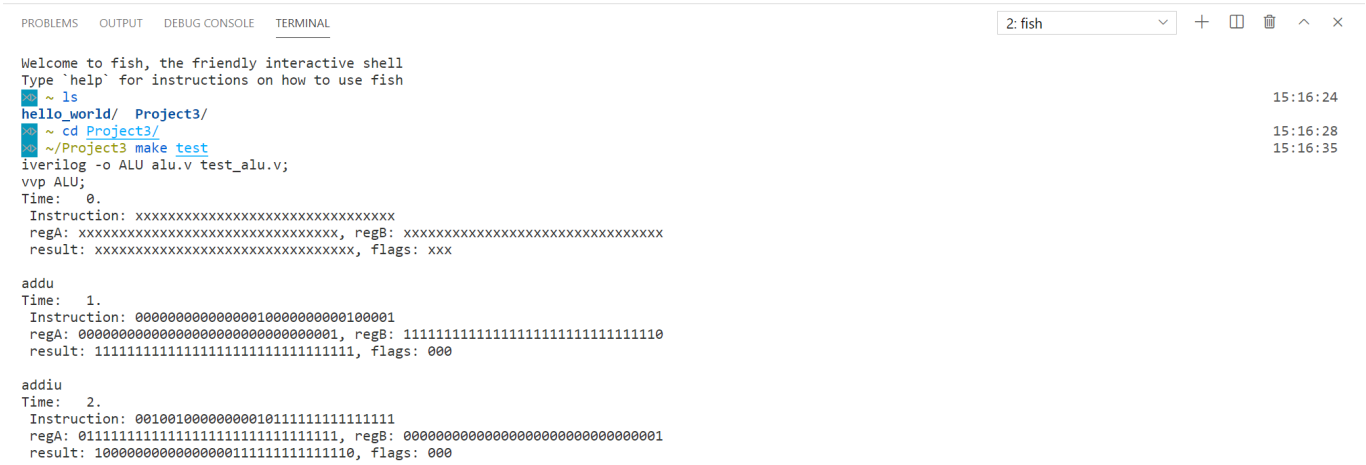
A poorly written ALU template is given in the `alu.v`, which already has the instruction parsing and register fetching parts. Furthermore, two instructions, `addu` and `addiu` have already been implemented as examples. A testbench `test_alu.v` is provided for testing your code, which already contains the test cases for addu and addiu.

To run the given template, you should:

- Transfer the folder `Project3` to the virtual machine.
- Enter the folder `Project3`.
- Compile and run the code by instruction `make test` in the terminal.

The process of compiling and running the code can be done both in the terminal of the virtual machine and VSCode, as shown in the following figures.





What you need to do based on the given template:

- Improve the poorly written `alu.v` by adding more implementations of instruction.
- Written the test for the new instructions in the testbench `test_alu.v`.

Note: I strongly recommend that after completing the template-based work, you can implement it from scratch by yourself.

4. Grading

4.1 Submission Requirements

- Your project 3 should be written in Verilog only.
- Your submission should contain at least the following file:
  - alu.v
  - test_alu.v
  - makefile
  - Report.pdf

- If necessary, you can submit more Verilog files beyond those mentioned above and you can also rewrite the makefile.
- You should place all of your source files in a folder and compress it into a .zip file. Name it with your student ID and submit it through BB.
- **Submission Deadline: 23:59, November 26, 2023**
- **If you submit after the deadline, you will lose 10 points for each day of delay, and if you submit more than 3 days late, you will receive a score of 0.**

4.2 Report

- The report of this project should be `no longer than 5 pages`. Keep your words concise and clear.
- In your report, you should include:
  - Your big picture thoughts and ideas, showing us you really understand ALU functionality.
  - A data flow chart explaining what you have extend.
  - The high level implementation ideas. i.e., how you break down the problem into small problems, and the modules you implemented, etc.
  - The implementation details. i.e., explain some special tricks used.
- In your report, you should not:
  - Include too many screenshots of your code.
  - Copy and paste others' reports.

4.3 Grading Details

- This project worth 15% of your total grade. The grading details of this project will be:
  - Correctness of ALU functionalities `80%`
  - Completeness and readability of ALU test bench `10%`
  - Report `10%`

4.4 Honesty

We take your honesty seriously. **If you are caught copying others' code, you will get an automatic 0 in this project.** Please write your own code.