# Project 2

## 1. Overview

**In project 2, you are required to implement an important computation unit in CPU, the Arithmetic and Logic Unit (ALU), using Verilog language. You are going to describe the functionality of ALU module as well as implement a clear test bench to show the correctness of your module.**

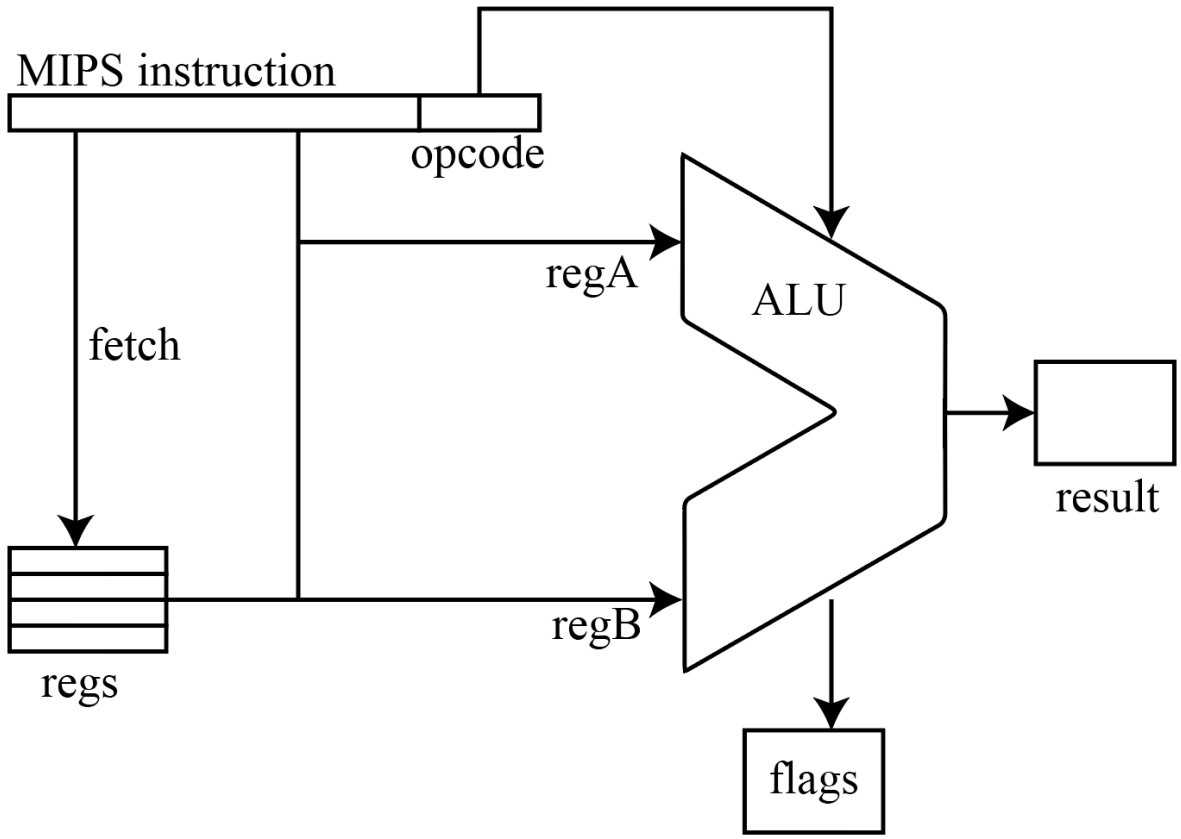
A poorly written example will be given. You can improve it or start a new one by yourself.

## 2. Requirements

Before start, you should review the content of ALU.

### 2.1 Simple CPU

Basically, you are actually going to implement a simple CPU which support simple instruction parsing, Register Value fetching, and ALU functions. As shown in the diagram below, after parsing the machine code of MIPS instruction, ALU receives three fixed inputs, `opcode(4 bit)`, `regA (32 bits)`, and `regB (32 bits)`, and outputs two signals, `result (32 bits)` and `flags (3 bits)`.




### 2.2 Simple Register Fetch

Since we are using up to 2 registers as the inputs, the size of register array is defined as 2. It means the register address in the MPIS code can only be one of `00000`, `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 and write test bench for each of them, corrodingly.

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

### 2.4 Module API

Your ALU module should be in the following format for easy grading.

```
module alu(instruction, regA, regB, result, flags)
input[31:0] instruction, regA, regB; // the address of regA is 00000, the address
of regB is 00001
output[31:0] result;
output flags[2:0]; // the first bit is zero flag, the second bit is negative
flag, the third bit is overflow flag.

// Step 1: You should parsing the instruction;
// Step 2: You may fetch values in mem;
// Step 3: You should output the correct value of result and correct status of
flags

endmodule
```

## 3. Grading

### 3.1 Requirements

1. Your project 2 should be written in Verilog only.
2. Your submission should contain at least two Verilog file:
    - ALU.v
    - test_ALU.v

   You can handle more than two Verilog files. You will need to write your own makefile.
3. You are encouraged to write your code in OO programming style.

## 4. Miscellaneous

### 4.1 Deadline

The deadline of this project is 04/11/2021 midnight.

## 4.2 Submission

You should put all of your source files in a folder and compress it in a **zip**. Name it with your **student ID**. Submit it through BB.

## 4.3 Grading

This project worth 15% of your total grade. The grading details of this project will be:

1. Correctness of ALU functionalities. - 80%
2. Completeness and readability of ALU test bench. - 10%
3. Report - 10%

**NOTICE: If your code does not support makefile, you will lose 50% automatic.**

## 4.4 Report

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

## 4.5 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.