Project 3

Overview

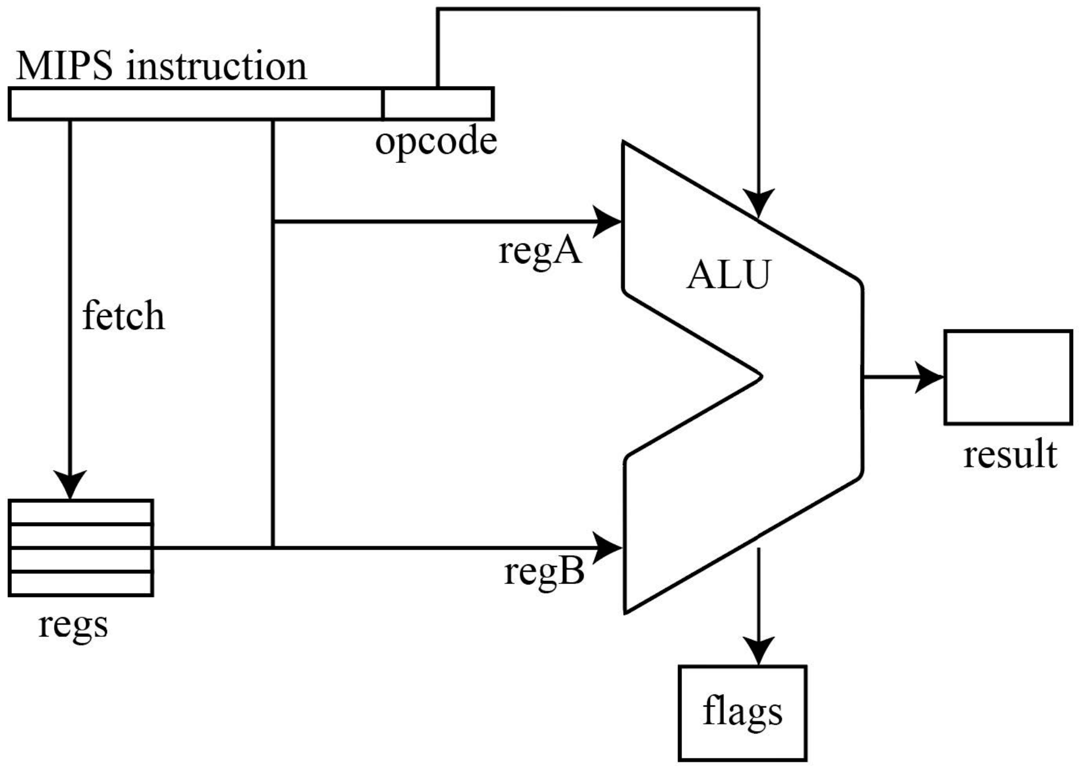
In project 3, you are required first to implement an important computation unit in CPU, the Arithmetic and Logic Unit (ALU), using Verilog language. Based on the implemented ALU, you are then required to implement a 5-stage pipelined CPU which can execute the MIPS instructions. \
Before you start working on this project, you need to:

- Learn the Verilog language.
- Review the content of ALU.
- Review the content of 5-stage Pipelined CPU.

Requirements

1. ALU

1.1 Simple ALU



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, 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)`.

1.2 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.

1.3 ALU Functions

You are required to support the following MIPS instruction in your ALU and write test bench for
each of them, accordingly.

```
- 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
```

1.4 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
```

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

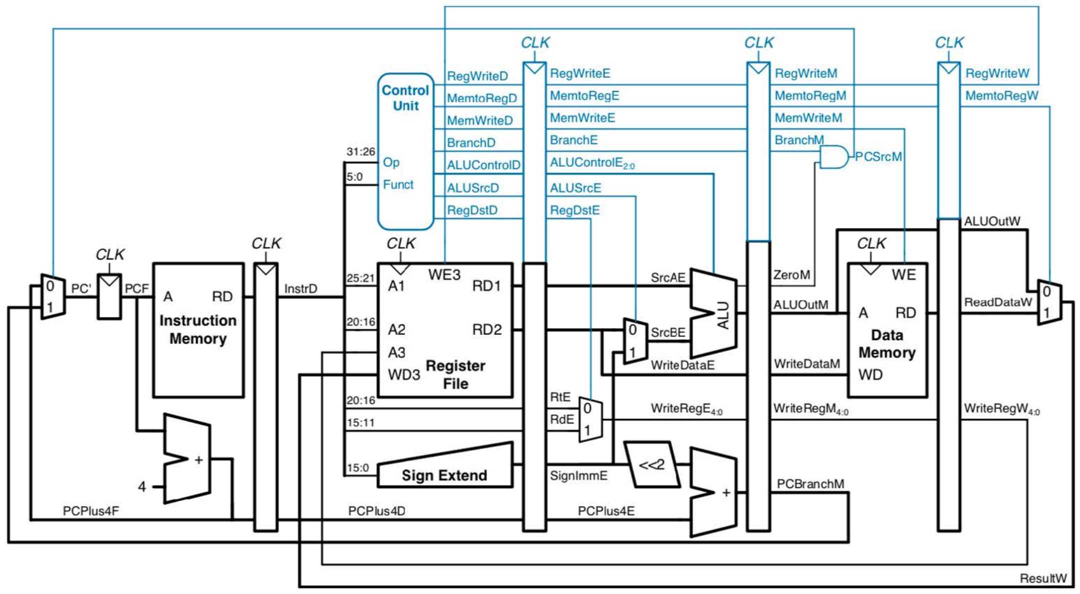
1.5 Some Reminder

For the overflow flag, you should only consider `add`, `addi`, and `sub` instruction. For the zero flag,
you should only consider `beq` and `bne` instruction. For the negative flag, you only need to deal
with `slt`, `slti`, `sltiu`, `sltu` instruction. 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.

2 Pipelined CPU

2.1 5-stage Pipelined CPU

As we learnt in the lecture, the processor acquires 5 clock cycle to execute one instruction.
In the first clock cycle, the CPU read one instruction in the instruction memory with the new given pc address. In the second, the processor will divide the instruction to different parts and decode the MIPS instruction with the registers and control unit. The operation code and function code in MIPS instruction are sent to the control unit, which recognize the type of an instruction. In the third cycle 3, ALU will handle arithmetic, logical, shifting, and conditional branch instructions. In the fourth cycle, data will be fetched from or stored to the data memory by data transfer instructions. In the fifth cycle, data will be written back to registers if needed.



2.2 Supported MIPS Instructions

You are required to implement a 5-stage pipelined CPU which supports the following MIPS instructions:

```
Data transfer instructions:
- lw, sw

Arithmetic instructions:
- add, addu, addi, addiu, sub, subu
Logical instructions

- and, andi, nor, or, ori, xor, xori
Shifting instructions
- sll, sllv, srl, srlv, sra, srav

Branch/Jump instructions:
- beq, bne, slt
- j, jr, jal
```

2.3 Usage of the provided modules

We have provided the instruction memory and data memory to you. For the simplicity, the memory space of the instructions and data are separate. The usage is listed below. You can modify the module. However, you should preserve the **module name** ,**RAM**, and **DATA_RAM**variables. \
Note: The address unit of both memory is in word. Hence, you should manually transfer address from byte space to word space.

```
module InstructionRAM // read-only
( // Inputs
input CLOCK // clock
, input RESET // reset
, input ENABLE
, input [31:0] FETCH_ADDRESS
// Outputs
, output reg [31:0] DATA
);
//...
endmodule
module MainMemory //data memory
( // Inputs
input CLOCK // clock
, input RESET // reset
, input ENABLE
, input [31:0] FETCH_ADDRESS
, input [64:0] EDIT_SERIAL // {1'b[is_edit], 32'b[write_add],
32'b[write_data]}
// Outputs
, output reg [31:0] DATA
);
//...
endmodule
```

2.4 Testing Samples

You should evaluate the performance of your code and write your report based on these testing samples. The folder includes 8 testing cases and you can find the details in the "About test.pdf" file.
Regarding the code you submit. your CPU should display the whole Main Memory in the screen.
The strategy to end your program is at the point where your CPU execute `32'hffffffff` instruction.
we will use your testbench module and an instruction file named "instructions.bin" to test your
project. Please set the input file name as above in your project.

3. 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:
  1. Your big picture thoughts and ideas, showing us you really understand pipelined 5-
     stage CPU.
  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.
- In your report, you should not:
  1. Include too many screenshots your code.
  2. Copy and paste others' report.

Submission and Grading

1. Requirements

- Your project 3 should be written in Verilog language only.
- You should implement ALU under path `/src/alu`
- You should implement CPU under path `/src/cpu` :
- You should write your own tests for the ALU and CPU implementation.
- You can handle more than two Verilog files.

- You are encouraged to write your code in OO programming style.
- You are encouraged to use our provided tests under `/testcase`. These cases will also be used for grading.

- In grading, we would replace `test_alu.v test_cpu.v data.bin instruction.bin` to test your implementation. **! Make sure**:
    - `make test` is available for compilation and testing under `/src/alu` and `/src/cpu`
    - `alu.v test_alu.v cpu.v test_cpu.v` exist and get compiled using `make test`
    - Your implementation of CPU loads instructions from `src/cpu/CPU_instruction.bin`
    - Your implementation of CPU updates the data memory after execution to `src/cpu/data.bin`
    - For report, your submission should be `/report/report.pdf`
    - State whether you solve the hazards at the beginning of your report

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.
- The deadline for this report is 2023/04/23. If you fail to submit your assignment by the deadline, 10 points will be deducted from your original mark for each day you are late, up to a maximum of 30 points. Assignments submitted more than 3 days after the deadline will not be considered valid and will be marked as 0 points.

3. Grading Details

- Implement the functionalities of ALU - 15%
- Implement the 5-stage CPU functionalities without any hazards - 30%
- Implement the 5-stage CPU functionalities encountering hazards - 30%
    - The more hazards solved, the higher grades.
- Provide your own test codes - 5%
- Reports - 20%

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**.