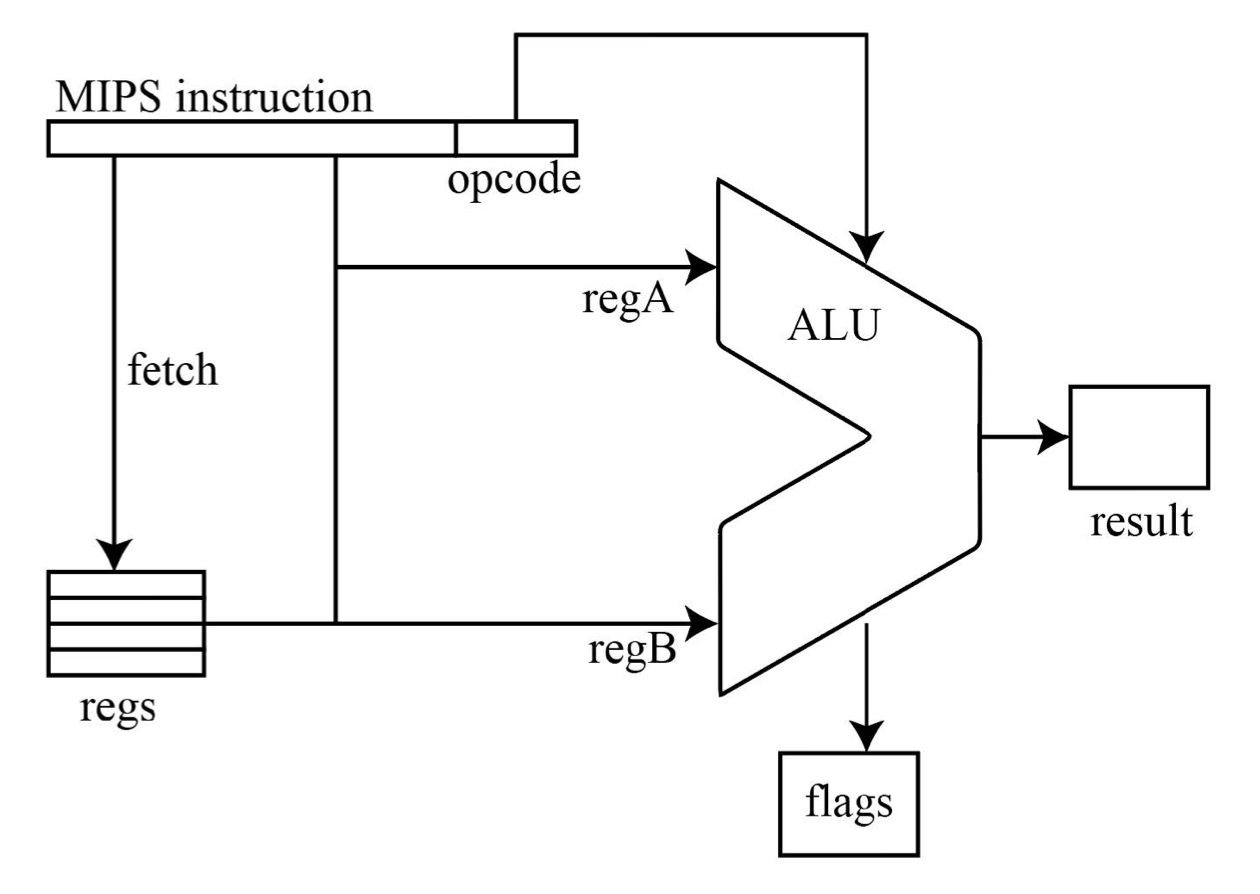
Report for CSC3050 Project 3

I implemented the hazard solutions for forwarding, stalling and branching. The CPU currently can pass the forwarding test2.

In this report, we are first required to implement the ALU unit in CPU, then we will implement a CPU based on the ALU we implemented.

1. **ALU**



The ALU receives MIPS instructions and output its computation results, which will be of future use for the CPU, and it will also output 3 flags, which are zero flag, overflow flag and negative flag.

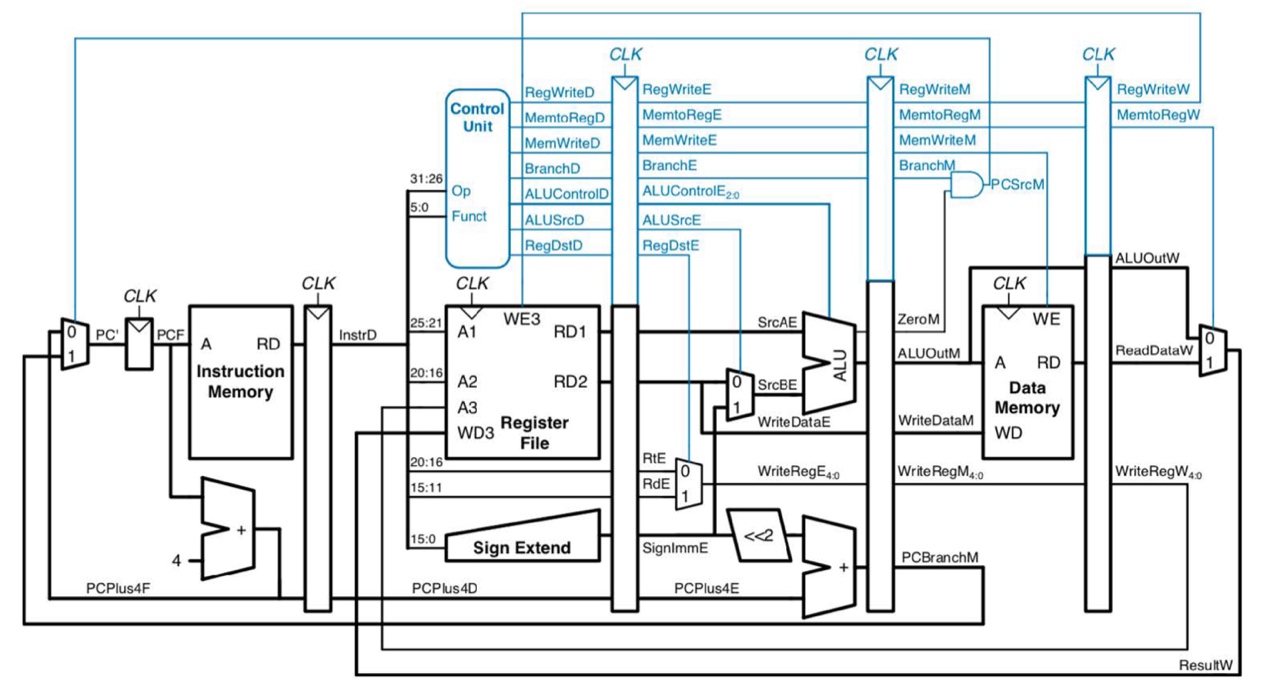
The big picture of the ALU is as above, the inputs are the MIPS instructions and the data contained in regA and regB. Its implementation idea is similar to that of project 2, where we first parse the MIPS instructions into different fields, and then based on the function code and op\_code we are able to know what instructions it is and perform the corresponding operations.

In this project, we only have 2 registers, and their addresses are designated to be 00000 for regA, and 00001 for regB. Therefore, the rs, rt field of the instructions can only be of these 2 values. One thing we need to consider about is that we need to figure out which register corresponds to which field, for example, whether we data in regA (address 00000) is fed into the rs field, or is fed into the rt field. This problem can be resolved by parsing the MIPS instruction and obtaining the address for register rs and register rt, if register rs is 00001, then we should feed the data in regB into register rs.

Another concern is about overflow. In add operations a+b=c, overflow occurs when the MSB of a and b are both 1, while MSB of c is 0, or when the MSB of a and b are both 0, while MSB of c is 1. In minus operation a-b=c, overflows occurs when the MSB of a and b are 0 and 1, while MSB of c is 1; or when the MSB of a and b are 1 and 0, while MSB of c is 0.

1. 5-staged Pipeline CPU

Next, we will implement the 5-staged pipeline CPU, and the ALU can be combined into it in the EXE stage. The demo one without handling hazard is shown below.



In the first clock cycle IF, the CPU read one instruction in the instruction memory with the new given pc address, then pc moves to the next instruction. But there will be 4 possibilities of the instructions, first is that it normally moves to its adjacent instruction (pc+4). Second, is the branch case, where it moves to PC+4+(immediate<<2). Third is the jump case, where it moves to address combined by {(PC+4)[31:28], target\_address\_field, 00}. In the fourth case for jr, it will moves to the address return by register rt.

The IF/ID register inside can be used to temporarily pass the instruction fetch from the instruction memory provided and the PC\_plus\_4.

In the second ID, the CPU will divide the instructions into different parts and decode the MIPS instruction. The operation code and function code in MIPS instruction are sent to the control unit, which recognize the type of an instruction and sent out certain signals. Part of these signals are shown in the figure above, such as the RegWriteD signal, which determines whether there is write bach to registers file in the future stage, the MemToRegD signal, which indicates whether we need to access the data memory, the ALUControlD signal, which will guide the ALU to do the corresponding operations in the next stage… The signals sent out for each instruction are determined case by case, for more detail, please refer to my *control unit.v*. Moreover, the register file is also implemented and maintained in this stage, where the data from WB stage will be written into it in some instructions such as add, sub.

The ID/EX register will pass those signals, and the address for rs, rt, rd to the next stage.

In the third cycle EXE, ALU will handle arithmetic, logical, shifting, and conditional branch instructions. But before putting data in the ALU, we need to determine the data for the 2 input ports of the ALU, i.e., srcAE and srcBE. We will add a multiplexer before each of them, the multiplexer for srcAE will determine whether the data from register rs, or the *shamp* field of the instruction will pass, and the multiplexer for srcAE will determine whether the data from register rt, or the sign immediate number will pass. These will be determined by the ALUSrcE and SigShamp signal based on the different instructions. Moreover, in this stage we will also determine the address for the register that we will write data into in instructions such as add, sub.

The EXE/MEM register will pass signal needed for future memory access and data write back process, the output of the ALU, and the address of the register where we will write data in it.

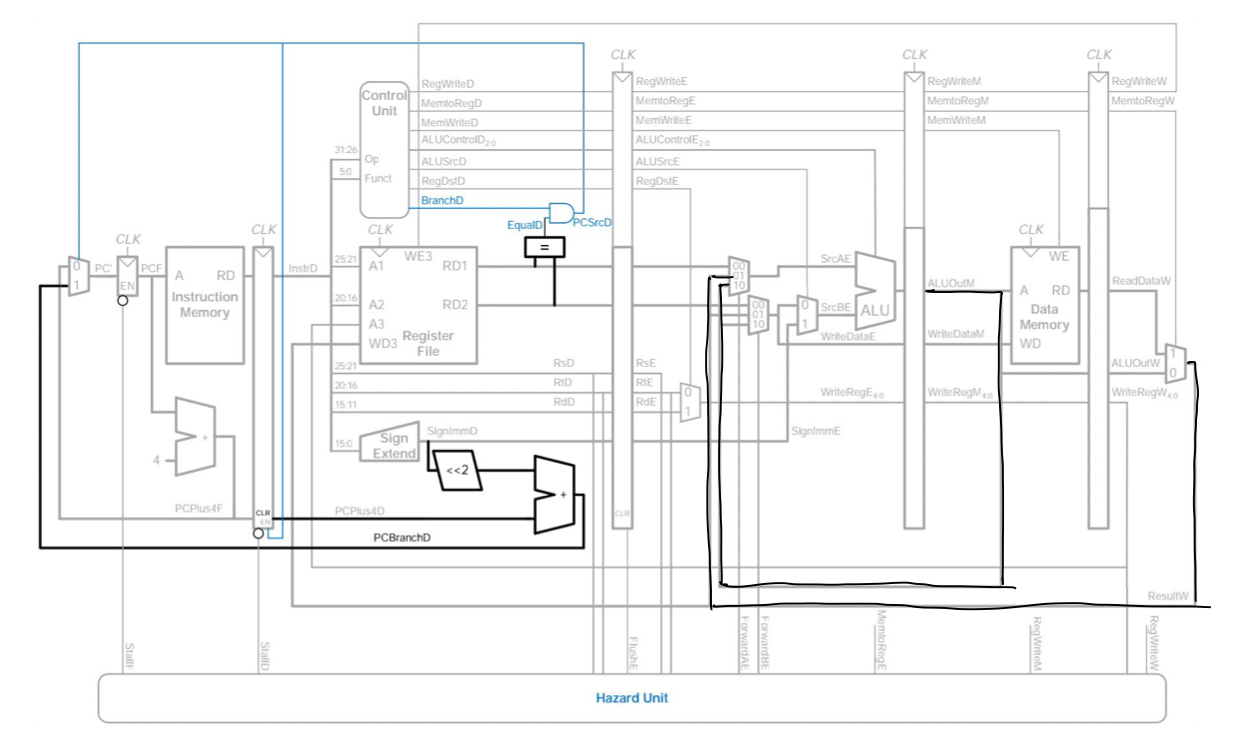
In the fourth cycle, we will do memory access. The data memory is provided, where we can access data from it, such as lw instruction, or write data into it, such as sw.

The MEM/WB register will pass signal needed for future data write back process, the output of the ALU, and the address of the register where we will write data in it.

In the fifth cycle, data will be written back to register file if needed. The WriteRegW will transfer the address of register to be written back in the ID stage stage.

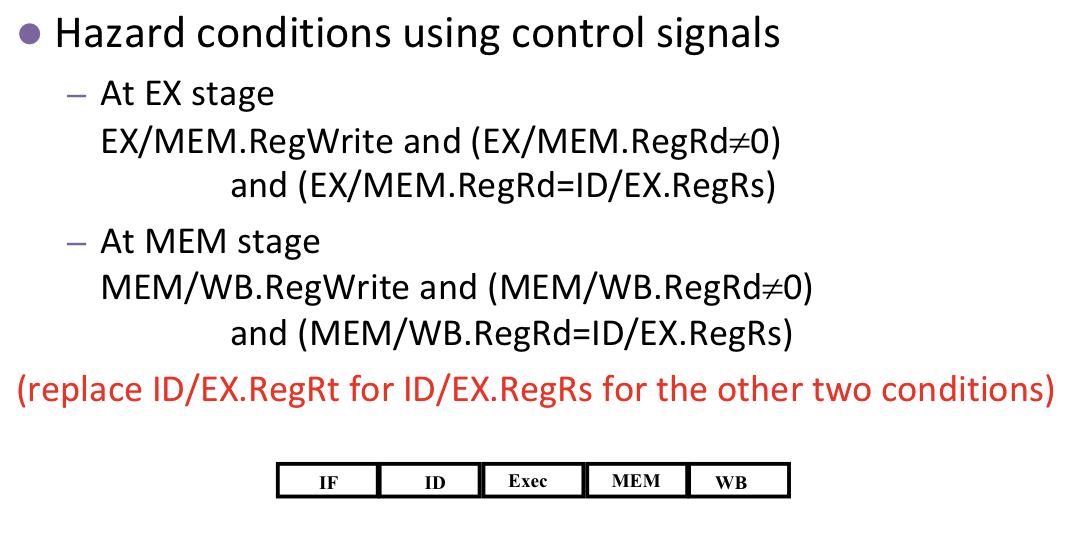
1. CPU with Hazard

The entire CPU module which can deal with hazard is shown below.



I have implemented the hazard processing functions for 3 types of hazards, even though some of them may malfunction.

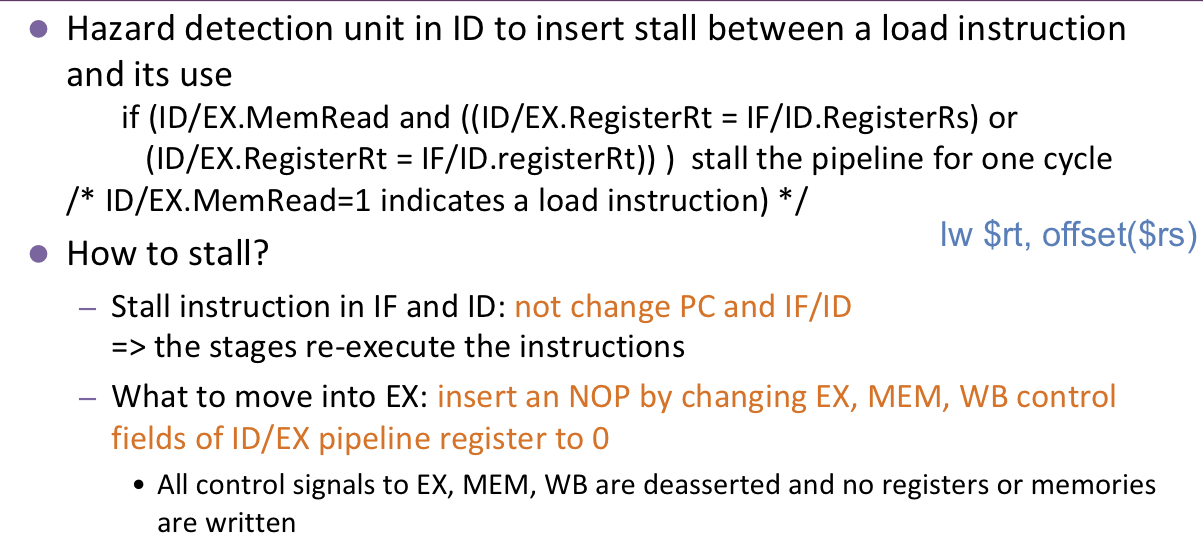
* 1. Forwarding



Forwarding can be resolved by connecting the ALUOutM output of EXE/MEM or ALUOutW output of MEM/WB to the srcAE or srcBE input port of the ALU. Therefore, we need an extra multiplexer before the input of ALU, pre\_mux\_AE, pre\_mux\_BE, to judge whether the regular data or ALUOutM, ALUOutW pass. This is determined by signal forward\_m\_a, forward\_m\_b, forward\_w\_a, forward\_w\_b in my code. It is also known that if both forwarding occurs in one port, we will pass the data of most recent one, that is the ALUOutM.

* 1. Stalling

We maintain a stall\_signal for the stalling, which first freeze the move on of pc in the next cycle and flush the ID/EX registers, which in fact insert an empty stage.



* 1. Branch Hazard

We will put the detection of branching in advance to the second stage, where there is a unit which determine whether data for rs and rt decoded are equal. We calculated the branch address simultaneously and fed it to the pc for the next instruction. Therefore, we need to flush the IF/ID register.

Moreover,

Currently, my CPU is able to pass testcase 1, 2 and 4.