Project 3 Report of CSC3050

Bodong Yan 119010369

A. Overall Design

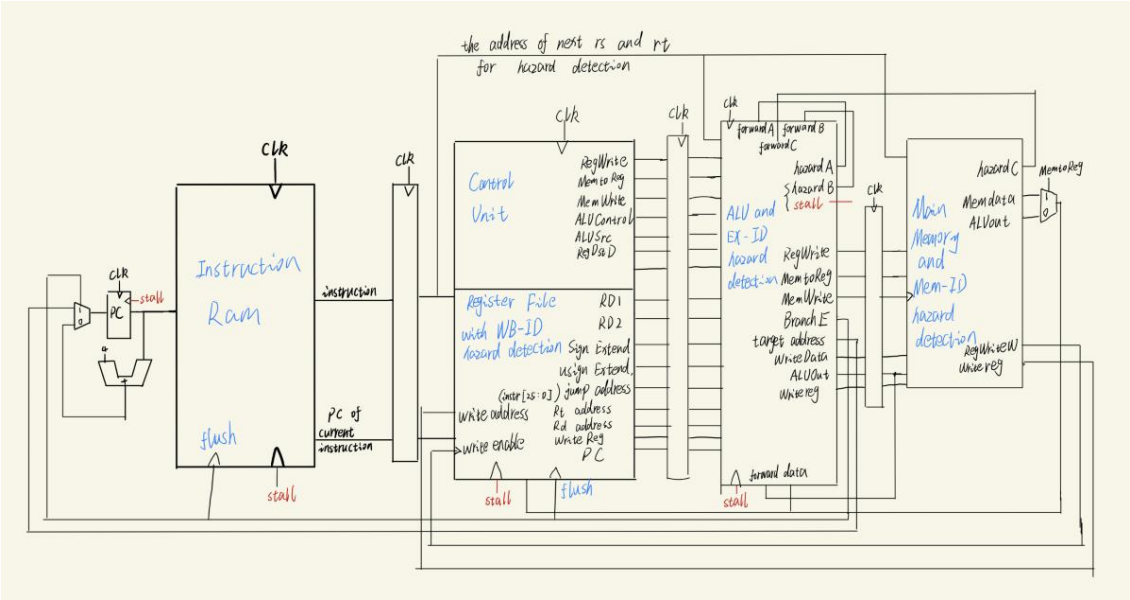
This project is to design a 5-stage CPU with pipelines, so each stage is written in one single file with one module. The Programme Counter stage is written together with the test bench, and the clock is made by a reg type variable using “forever” statement. As is announced, the PC will start at 0 address, and when the 32'hffffffff instruction is detected, the CPU process will be terminated after 7 time unit.

The pipeline implementation is the most significant part in this project. But I found that the latches in verilog is hard to use, so I choose another way to realize the pipeline. For one pipeline input and output, I use one wire type variable in test bench, and this wire variable is used as both the output of the previous module and the input of next module. When a clock comes, even though the value of the wire will be updated by the previous module immediately, the next module will do operation on the present value of the wire rather than the updated value. So I use this characteristic to implement the pipeline.

The function of each unit is implemented as is required in the instruction, and the Control Unit is written together with the Register File. The PC will provide the instruction address, the Instruction Ram will output the corresponding instruction, the Register File and Control Unit will decode the given 32-bit instruction and send control signal, the ALU will do the required operation on the two input operand, and the Main Memory module will do the memory operation.

Also, the hazards covered by the instruction tutorial can be handled by this CPU. But I didn’t write a separate hazard handling unit, the hazard detection, data forwarding, stalling are all handled in ALU module and Main Memory module.

B. Data Flow Chart with Explanation



The data flow is almost the same as is required. But there are still some small differences. For the PC, I didn't pass the "current PC + 4" on, I just pass the current PC value, and the target address is calculated in ALU if it needs to branch or jump. And there's no branch signal from the ID stage, since the ALU will recognize each operation including branch and jump according to the ALU control signal in my design. If jump or branch occurs, the branch signal from ALU stage will go high to accomplish the branch or jump.
The ID stage(Register File) is designed to handle the WB-ID hazard, whenever the write back address is detected to be the same as the read address. In addition, there are three other types of hazards including hazardA, hazardB and hazardC. Both hazardA and hazardB are detected in ALU stage and are EX-ID hazard. The difference is that, hazardB needs to stall which is designed for load instruction hazard while hazardA doesn't need to stalling since it's designed for other instruction without memory read. Lastly, hazardC is detected and handled in Main Memory stage as it is designed for the MEM-ID hazard.

C. ALU Control Details

In the ID stage, control unit will recognize the instruction according to opcode and funcode, and output ALU control signal from the control unit will directly decide the operation done by the ALU. There are nineteen ALU control signal as follow:

Instructions	Operation of ALU	ALU control signal
Add addi addu addui lw sw	add	00000
sub subu	sub	00001
and andi	and	00010
nor	nor	00011
ori or	or	00100
xor xori	xor	00101
sll	sll	00110
sllv	sllv	00111
srl	srl	01000
srlv	srlv	01001
sra	sra	01010
srav	srav	01011
slt	slt	01100
beq	beq	01101
bne	bne	01110
j	j	01111
jr	jr	10000
jal	jal	10001
Cannot be recognized	None	10010

The "None" operation will be used when "flush" happens. When it needs branch or jump, the Instruction Ram will send an unrecognizable instruction to Register File, so

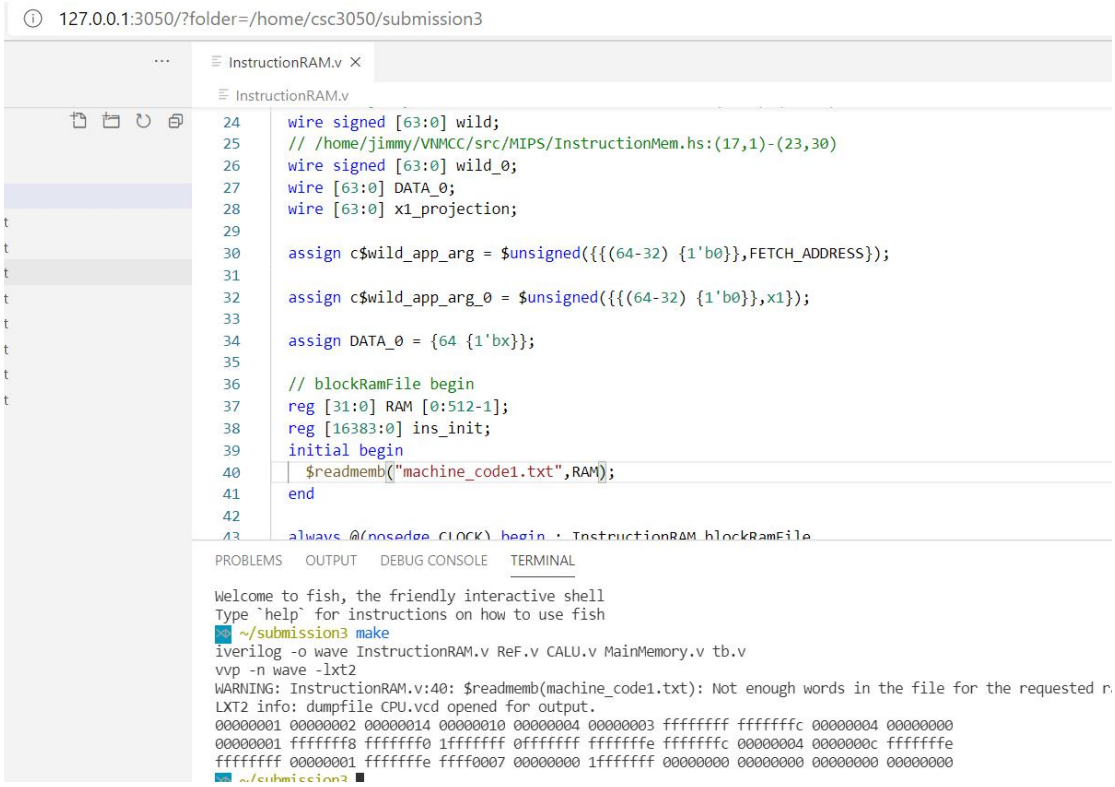
that nothing will be done by the ALU, and both the RegWrite and MemWrite will be disabled.

D. The Result

The eight given sets of instructions are all tested to be successful under the environment provided by the USTF and TA. The all data of the Main Memory are displayed in hexadecimal on the terminal(every line on the terminal display 10 data), and the cycles spent to deal with the instructions is recorded according to the waveform in gtkwave(every two time units count one cycle, as I use the positive edge as the trigger). Here are the result:

Testing set	Testing result	Cycles cost
machine_code1	Fit DATA_RAM1 exactly	57 cycles
machine_code2	Fit DATA_RAM2 exactly	16 cycles
machine_code3	Fit DATA_RAM3 exactly	19 cycles
machine_code4	Fit DATA_RAM4 exactly	18 cycles
machine_code5	Fit DATA_RAM5 exactly	220 cycles
machine_code6	Fit DATA_RAM6 exactly	65 cycles
machine_code7	Fit DATA_RAM7 exactly	50 cycles
machine_code8	Fit DATA_RAM8 exactly	31 cycles

The screenshot of the first, fifth and sixth set of instruction testing result are listed below(only displayed the first 30 lines):



127.0.0.1:3050/?folder=/home/csc3050/submission3

InstructionRAM.v

```
24  wire signed [63:0] wild;
25  // /home/jimmy/VNMCC/src/MIPS/InstructionMem.hs:(17,1)-(23,30)
26  wire signed [63:0] wild_0;
27  wire [63:0] DATA_0;
28  wire [63:0] x1_projection;
29
30  assign c$wild_app_arg = $unsigned({{(64-32) {1'b0}},FETCH_ADDRESS});
31
32  assign c$wild_app_arg_0 = $unsigned({{(64-32) {1'b0}},x1});
33
34  assign DATA_0 = {64 {1'bx}};
35
36  // blockRamFile begin
37  reg [31:0] RAM [0:512-1];
38  reg [16383:0] ins_init;
39  initial begin
40    $readmemb("machine_code5.txt",RAM);
41  end
42
43  always @(posedge CLOCK) begin : InstructionRAM_blockRamFile
44    if (IRstall == 1'b1) begin
45
46    end
47    else begin
48    if (1'b0 & ENABLE) begin
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
~/submission3 make
iverilog -o wave InstructionRAM.v ReF.v CALU.v MainMemory.v tb.v
vvp -n wave -lxt2
WARNING: InstructionRAM.v:40: $readmemb(machine_code5.txt): Not enough words in the file for the requested range [0
LXT2 info: dumpfile CPU.vcd opened for output.
00000000 00000000 fffffffb 00000017 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
~/submission3
```

127.0.0.1:3050/?folder=/home/csc3050/submission3

InstructionRAM.v

```
24  wire signed [63:0] wild;
25  // /home/jimmy/VNMCC/src/MIPS/InstructionMem.hs:(17,1)-(23,30)
26  wire signed [63:0] wild_0;
27  wire [63:0] DATA_0;
28  wire [63:0] x1_projection;
29
30  assign c$wild_app_arg = $unsigned({{(64-32) {1'b0}},FETCH_ADDRESS});
31
32  assign c$wild_app_arg_0 = $unsigned({{(64-32) {1'b0}},x1});
33
34  assign DATA_0 = {64 {1'bx}};
35
36  // blockRamFile begin
37  reg [31:0] RAM [0:512-1];
38  reg [16383:0] ins_init;
39  initial begin
40    $readmemb("machine_code6.txt",RAM);
41  end
42
43  always @(posedge CLOCK) begin : InstructionRAM_blockRamFile
44    if (IRstall == 1'b1) begin
45
46    end
47    else begin
48    if (1'b0 & ENABLE) begin
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
~/submission3 make
iverilog -o wave InstructionRAM.v ReF.v CALU.v MainMemory.v tb.v
vvp -n wave -lxt2
WARNING: InstructionRAM.v:40: $readmemb(machine_code6.txt): Not enough words in the file for the requested range [0:51
LXT2 info: dumpfile CPU.vcd opened for output.
fffffffe fffffffc fffffff8 fffffff0 ffffffe0 ffffffc0 00000080 00000100 00000200 00000001
00000001 00000001 00000001 00000001 00000001 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

End of Report