CSC3050 Project 3

May 1, 2022

Build a 5-stage pipelined CPU

Student ID: 120090266

Student Name: Feng Yutong

1 Overview

The task of this assignment is to implement a 5-stage pipelined CPU. Basically, it supports instruction fetch (IF), instruction decode and register read (ID), excute or calculated address (EX), access data memory (MEM), write back to register (WB). To solve all the harzard required in the assignment, the program did some change on the orignal diagram. The new diagram is as followed (Note that the parameter names are the same as to the program)

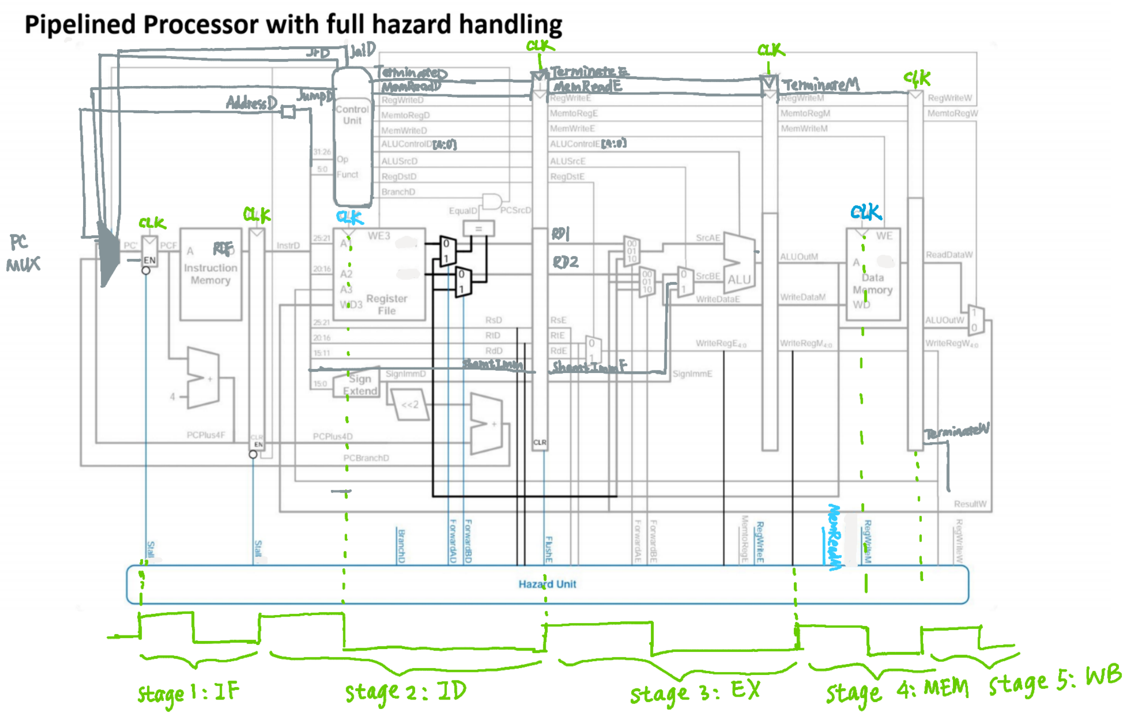

Figure 1: Diagram for CPU

2 High Level Implementation Ideas

Since this program builds a 5-stage pipelined cpu, but the diagram has 6 CLK. Green CLK are set positve-edge triggering while blue CLK are set negative-edge triggering. Thus, 5-stage is satisfied and RAW is avoided.

2.1 Stage 1: IF

New PC address will be updated according to branch, j, jal, jr instructions by multiplexers. The CPU read one instruction in the instruction memory with the new given PC address.

2.2 Stage 2: ID

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 first half stage, WB is performing (More details see details section). In the second half stage, read value of register and handle jr, branch hazard.

2.3 Stage 3: EX

ALU will handle arithmetic, logical, shifting, and conditional branch instructions. This stage also handles EX/MEM data hazard.

2.4 Stage 4: MEM

Data will be fetched from or stored to the data memory by data transfer instructions.

2.5 Stage 5: WB

Data will be written back to registers if needed.

3 Implement Details

3.1 Stage 1: IF

Update PC according to JalD, JumpD, PcSrcD (branch) and JrD by multiplexer (see code).

```
if(JumpD == 1'b1 || JalD == 1'b1) PC = AddressD;
else if(JrD == 1'b1) PC = e1;
else begin
        case(PCSrcD)
                1'b1: PC = PCBranchD;
                default: PC = PCPlus4F;
        endcase
end
```

3.2 Stage 2: ID

Register file simultaneously write and read Since instructions, writeback data and fetching register value are updated simultaneously on the positve-edge of CLK. Thus, the program writing data back to register on the negative-edge can handle the hazard about register file simultaneously write and read.

Branch & Jr harzard If branch condition is satisfied or instruction is jr, PC will be updated with PCBranchD or e1. Next ID stage will be flushed.

Two flags ForwardAD/ForwardBD are used to indicate data harzard type (see code). Harzard unit is triggered on the negative edge since we should first write back data.

If ForwardAD/ForwardBD = 2'b10, RD1 will be updated with previous ALU result. If ForwardAD/ForwardBD = 2'b01 RD2 will be updated with data memory or an earlier ALU result.

```
ForwardAD = ((RegWriteE==1'b1)&&(WriteRegE!=5'b0)&&(WriteRegE==RsD))
? 2'b10 : 2'b00;
ForwardBD = ((RegWriteE==1'b1)&&(WriteRegE!=5'b0)&&(WriteRegE==RtD))
? 2'b10 : 2'b00;
if(ForwardAD!=2'b10)
        ForwardAD = ((RegWriteM==1'b1)&&(WriteRegM!=5'b0)
        &&(WriteRegM==RsD)) ? 2'b01 : 2'b00;
if(ForwardBE!=2'b10)
        ForwardBD = ((RegWriteM==1'b1)&&(WriteRegM!=5'b0)
        &&(WriteRegM==RtD)) ? 2'b01 : 2'b00;
```

3.3 Stage 3: EX

ALU operation Instructions are classified to the same ALUControl if the ALU function is the same and there's no special modification. ALUSrc determines using Sign extent immediate or register value. RegDst determines which registers to write. More details can be seen in Table 1.

Data Forward Two flags ForwardAE/ForwardBE are used to indicate data harzard type (see code). Since data forwarding is affected by ResultW, harzard unit and excuation is triggered on the negative edge so that it can use latest data and has no intereference on other operations.

ForwardAE/ForwardBE = 2'b10 for MEM_to_EX data harzard and ScrAE/SrcBE will be updated with previous ALU result. ForwardAE/ForwardBE = 2'b01 for WB_to_EX data hazard and ScrAE/SrcBE will be updated with data memory or an earlier ALU result. The priority of MEM_to_EX is higher than WB_to_EX.

```
ForwardAE = ((RegWriteM==1'b1)&&(WriteRegM!=5'b0)&&(WriteRegM==RsE))
? 2'b10 : 2'b00;
ForwardBE = ((RegWriteM==1'b1)&&(WriteRegM!=5'b0)&&(WriteRegM==RtE))
? 2'b10 : 2'b00;
```

ALUctr	instructions	Mark
00000	add,addi,lw,sw	
00001	addu,addiu	
00010	sub, beq	
00011	subu	
00100	srl	reg_A=ShammtImmE
00101	srlv	
00110	sra	reg_A=ShammtImmE
00111	srav	
01000	and	
10010	xori	redo zero extend of immediate
10100	jr	set JrD = 1 in ID stage to trigger changing PC
01001	nor	
01010	or	
01011	xor	
01100	slt	
01101	jal	add, but reg_A=PC+4, reg_B=0
01110	ori	redo zero extend of immediate
01111	bne	set zero signal to the opposite for debug convenience
10000	sll	reg_A=ShammtImmE
10001	sllv	
10011	andi	redo zero extend of immediate

Table 1: ALU

```
Stall = (MemReadE==1'b1&&((RtE==RtD)||(RtE==RsD)))
? 1'b1 : 1'b0;
if(ForwardAE!=2'b10)
        ForwardAE = ((RegWriteW==1'b1)&&(WriteRegW!=5'b0)&&
(WriteRegW==RsE)) ? 2'b01 : 2'b00;
if(ForwardBE!=2'b10)
        ForwardBE = ((RegWriteW==1'b1)&&(WriteRegW!=5'b0)&&
(WriteRegW==RtE)) ? 2'b01 : 2'b00;
```

lw Stall When lw encounters data hazard, Stall will be updated (see code). A NOP is inserted by flushing next EX stage and stalling IF, ID stage.

```
Stall = (MemReadE==1'b1&&((RtE==RtD)||(RtE==RsD))) ? 1'b1 : 1'b0;
```

3.4 Stage 4: MEM

The lw, sw operation are done in this part according to MemWriteM signal.

3.5 Stage 5: WB

If RegWriteW signal is true, MemtoRegW selects the written data while WriteRegW selects the written registers.

3.6 Other tricks

1. Each stage is triggered by clock instead of module. 2. For combinational logic triggering, parmaters are carefully chosen to differentiate the performing order. 3. To deal with initial value x(don't care) in the first several cycles (some pitfalls, (1′bx==1′b0) == 1′bx, (1′bx==1′b0) == 1′bx, (1′bx==1′bx) == 1′bx, x in multiplexer selects x data), parameter default in switch(case) or using XXX != 1′b0 to replace XXX == 1′b1 can help a lot. 4. Adding a termination signal. When instruction 32′hffff_ffff, is fetched, true terminate signal will be transfered stage by utill it reaches WB stage. Then the program terminates.

4 Test

4.1 Performance Analysis

The program sucessfully handles 8 test cases. All harzard are handled as the PPT does.

Test Case	Number of Instruction	Number of Clock Cycles	CPI
Test 1	53	57	1.075
Test 2	12	16	1.333
Test 3	14	19	1.357
Test 4	14	18	1.286
Test 5	25	180	7.200
Test 6	52	55	1.058
Test 7	45	46	1.022
Test 8	25	29	1.160

Table 2: Performance Analysis

4.2 Compile & Run

The program read instructions from **instructions.bin** file and output data memory to the screen and a file named **DATA_RAM**. **This program does not use two provided modules, it only preserve the name of DATA_RAM for main memory and DATA for instruction memory** To run the program, move to the folder of CPU.v and enter **make** in the terminal to run the program. You can use **diff** to compare the expected output and the result.