CA Project 1 Report, Team 25

I. Members and work load

B03902043 邱綜樹	B03902005 林日能	B03902075 賴楷宗
EX_MEM.v	IF_ID.v	Control.v
MEM_WB.v	ID_EX.v	MUX3.v
ForwardingUnit.v	HazardDetection.v	MUX32.v
ALU.v	PC.v	MUX5.v
ALU_Control.v	DataMemory.v	MUX8.v
		shiftLeft2_26.v
		shiftLeft2_32.v
CPU.v		
testbench.v		
Report.pdf		

II. Implementation of pipelined CPU

There are five latches, PC.v, IF_ID.v, ID_EX.v, EX_MEM.v and MEM_WB.v, to serve as the main component of pipelined CPU. For every negative edge of clock, each latch advances its input to output concurrently, without overwriting. For Registers.v and DataMemory.v, data are written in positive edge of clock. The clock of first half of a cycle is 0, and that of second half is 1.

III. Module implementation

- ALU_Control.v

  I use case to check ALUOp code (00 = R-type, 01 = Or, 11 = Subtract, 10 = Add). R-type can then be divided by function-bits to (+, -, *, &, |) groups. The return value determine which Operation ALU would do.

- ALU.v

  From the control signal, we can do operation on two input datas (AND, OR, ADD, Subtract or Mul). Then connect the result to output.

- EX_MEM.v

  Always at negative edge connect the input to output. Two ctrl signals (MemRead & MemWrite) are derived from MEM_i this input.

- MEM_WB.v

  Always at negative edge connect the input to output.

- ForwardingUnit.v

Detect the EX hazard or MEM hazard by Rd and regWrite, then compare Rd to Rs&Rt.

```
pseudo :
if regWrite_i and RegRd_i != 0
    if EX_MEM_RegRd_i == ID_EX_RegRs
        fa_temp = 2'b10 or 2’b01    // decided by EX_MEM’s rd or MEM_WB’s rd
    if EX_MEM_RegRd_i == ID_EX_RegRt
        fb_temp = 2'b10 or 2’b01
```

- HazardDetection.v

Read in the whole instruction in ID stage. If the instruction in EXE stage has to read from memory, and the rt register in EXE stage is identical to rs or rt in ID stage, stall IF and ID for one cycle, set the control bits to be zero, in order to create a bubble in EXE stage for next cycle. Otherwise, do nothing.

- IF_ID.v

Move the instruction from IF stage to ID stage in every negative edge of clk_i. If encountered data hazard, stall for one cycle. If encountered control hazard, flush the instruction in ID stage in next cycle.

- ID_EX.v

Move the instruction from ID stage to EXE stage in every negative edge of clk_i.

- PC.v

Initially, set the program counter to be 0. Give PC value to IF stage in every negative edge of clk_i. If encountered data hazard, stall for one cycle.

- DataMemory.v

A 32-Bytes memory. Assign the memory data out when MemRead bit is 1. Write data into the memory in positive edge of clk_i, when MemWrite bit is 1.

- Control.v

Deal with different Operations by switch-case.

- MUX5.v & MUX32.v

Deal with the two cases by if-then-else.

- MUX3.v

  Deal with the three cases by if-then-else.

- shiftLeft_32.v

  Trivial, output = input << 2

- shiftLeft_26.v

  Trivial, output[28:0] = input[26:0] << 2.

- Adder.v, Sign_Extend.v, Registers.v and Instruction_Memory.v

  Same as those in homework 4.

- testbench.v

  The clock is initialized as 0. And the configuration of our CPU is printed at every negative edge.

IV. Problem & Solution

Problem 1 : If I connect A module's output to B module's input directly, there is some value lost (e.g. A'output = 10, but B's input = x)
Solution 1 : create a wire between A & B. (Even though I am not use this solution at somewhere, there aren't any value lost at that point. So it's a bit weird…)

Problem 2 : We weren't familiar with Pipelined CPU at the beginning. So the clock edge of each latch (s.t. IF/ID, ID/EX, …) and PC we don't select carefully. And the outcome answer is not correct.
Solution 2 : After some blind try (many hours' debuging QQ), we consider that every latch and PC's connection work should be done with negative edge.

Problem 3 : TA's Pdf has error in last page. There is a wire not connecting to any module but it drawn. Besides, the picture of total CPU data path is complicate.
Solution 3 : We ignore that wire and print that page out. When connecting CPU, we write down variables' name to help us think clearly.

Problem 4: This code cannot work on Mac OSX.
Solution 4: It works perfectly on Ubuntu.