Computer Architecture, Fall 2017

Project 1 report

DUE DATE: DECEMBER 11, 2017

1 Members & Team Work

- b03902125 郭曉嵐
  - Debugging.
  - Fixing some data hazard error.
- b03902127 林映廷
  - Linking the module with *CPU.v*.
  - Handling with *flush* and *stall* in *HazardDetection.v*.
- b03902129 陳鵬宇
  - Linking the module with *CPU.v*.
  - The implementation of *Forwading.v*.
  - The implementation of each sub module.

2 Implementation of the Pipeline CPU

We use *testbench.v* to see the output; *CPU.v* to build the whole datapath.

The *CPU.v* is the core of the whole program, which takes two inputs *clk_i* since we change the value when *positive clock edge* and *start_i* since we need to know when the program is starting, so we set the *start_i* to 1 when the clock time begins for one-fourth of the *CYCLE_TIME* to ensure that the start signal is valid at the right time.

- **testbench**: Do the following by specs:
  - Initialize storage units.
  - Load ”instruction.txt” into instruction memory.
  - Create clock signal.
  - Output cycle count in each cycle.
  - Output Register File & Data Memory in each cycle.
  - Print result to ”output.txt”.
  - Handle when to *stall* or *flush*.
- **CPU**: Declare the *wire* and *reg* type variables in *CPU.v* since there may be an output of a module connecting to several inputs of other modules. If we don’t declare these variables globally, the codes will look tedious and hard to read. And classify the registers as four types:
  - IF/ID registers
  - ID/EX registers
  - EX/MEM registers
  - MEM/WB registers

3 Implementation and Explanation of Each Module

IF stage

- **Instruction_Memory**: Save the instructions.
- **PC**: *PC* module:
  - if *hazard* doesn't occur and *start_i* is set, updates the PC value.
  - else if *start_i* is not set, clears the *PC value* to zeros.
- **Add_PC**: Add the *instAddr* for 4.
- **MUX_Branch**: Determine whether *branch* to new address or not.
- **MUX_Jump**: Determine whether *jump* to new address or not.

ID stage

- **Registers**: Save the data of registers.
- **ADD**: Add the *shifted address* and *IFID_pc* together.
- **ShiftJump**: Shift the *jump address* left by two bits, and concatenate with two zeros.
- **ShiftLeft2**: Shift the *address* left by two bits.
- **Sign_Extend**: Extend the 16-bits *address* to 32 bits by filling in the $[31:16]$ bits as the first bit of the input address, i.e., *address*[15].
- **Equal**: Determine whether *RS_data* and *RT_data* are equal or not.
- **Control**: Set the following control units to 0 or 1 by the 6-bits *op_i*$[5:0]$.
  - $RegWrite$
  - $MemtoReg$
  - $Branch$
  - $MemRead$
  - $MemWrite$
  - $RegDst$
  - $ALUOp$
  - $ALUSrc$
  - $Jump$
- **MUX_CtrlUnit**: Determine whether to *stall* the pipeline or not.
- **MUX_RS**: Determine whether forwarding the *RSdata* by the *Forwarding* module.
- **MUX_RT**: Determine whether forwarding the *RTdata* by the *Forwarding* module.
- **HazardDetection**: Detect the hazard.
  - if *MemRead_i* is set, it means that *lw* may incurs some hazard, thus we have to *stall* the pipeline for one clock cycle but do not *flush* the pipeline. Also we set the *pc_hazard_o* to 0.
  - else if the control units *Jump_i* or *Branch_i* & *Equal_i* are set, we not only *stall* the pipeline but *flush* it. Also we set the *pc_hazard_o* to 1.
  - else we continue our stages as before and set *stall_o, flush_o* and *pc_hazard_o* to 0.

EX stage

- **ALU**: Do the ALU operation with two data, and also assign the *zero* value to 0 or 1.
- **ALU_Controlt**: Determine the *ALU operation* by the 6-bits *function* field.
- **MUX_RegisterRd**: Decide the *RegisterRd* between $inst[20:16]$ and $inst[15:11]$.
- **MUX_ForwardA**: Determine the input of *ALU.data1_i* by the *Forwarding* module.
- **MUX_ForwardB**: Determine the input of *tmpALUdata2* by the *Forwarding* module.
- **MUX_ALUSrc2**: Determine whether the input of *ALU.data2_i* is *tmpALUdata2* or the *sign-extened data*.
- **Forwarding**: Determine the value of *ForwardA* and *ForwardB* by the following two data hazard:
  1. *EX hazard*:
     - if (EX/MEM.RegWrite
       and (EX/MEM.RegisterRd $\neq$ 0)
       and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10
     - if (EX/MEM.RegWrite
       and (EX/MEM.RegisterRd $\neq$ 0)
       and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
  2. *MEM hazard*:
     - if (MEM/WB.RegWrite
       and (MEM/WB.RegisterRd $\neq$ 0)
       and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01
     - if (MEM/WB.RegWrite
       and (MEM/WB.RegisterRd $\neq$ 0)
       and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01

MEM stage

- **Data_Memory**: Save the data of the memory.

WB stage

- **MUX_RegWriteData**: Determine the data written to the register is *MEMWB_ALUResult* or *MEMWB_ReadData*.

4 Problems and Solution of the Project

To implement a such huge five stages pipeline is not easy, we have to carefully connect each input and output in a correct manner. Because we are not that familiar with *verilog*, we *Google* a lot to *fetch* the critical information we need.

On the one hand, we took a long time to figure out the difference between ”<=” and ”=”. We use ”<=” in a common situation and sequential logic and ”=” in a combinational logic. On the other hand, we also took some time to know the difference between *reg* and *wire*. Following are the explanations of them we Googled:

- **Wire**
  Wires are used for connecting different elements. They can be treated as physical wires. They can be read or *assigned*. No values get stored in them. They need to be driven by either continuous assign statement or from a port of a module.
- **Reg**
  Contrary to their name, regs don’t necessarily correspond to physical registers. They represent data storage elements in Verilog/SystemVerilog. They retain their value till next value is assigned to them (not through assign statement).

It is also very difficult to debug because there are *twenty* .v files in the folder. Our eyes really suffered a lot. We inserted the following codes to generate the .vcd file for conveniently debugging:

```
$dumpfile(”project.vcd”);
$dumpvars
```

By tracing the logic of the .vcd file, it can significantly increase the speed of debugging.