Project 3

Due Date: 11:55 pm, Wednesday July 1st, 2015.

Purpose:

Generate a detailed understanding of the operation of the MIPS pipeline including forwarding and hazard detection. This is intended to solidify your understanding of the fundamental concepts of

I. Data dependencies between instructions and how they can interact in the pipeline to produce hazards

II. Control flow and its impact on the operation of the pipeline

Assignment Part I: Understanding the Model

This part does not have a submission requirement. It is intended to help you get up to speed quickly and to familiarize yourself with the model, and compile and execute a few simple programs. Start by studying the trace and execution of the program that is already in instruction memory and determine if the program is correctly executed. Check the documentation associated with the model. The warm-up can be discussed with your classmates.

- Is this program execution correct? If not, why not?
- The datapath does not provide hardware support for correctly executing branches. How many delay slots does the pipeline have?
- What is the initial value of the stack pointer?
- What value is placed on the read output of memory when the data memory is not being read?
- If ALUSrc is undefined, what is the value of the BInput of the ALU in the execute stage.
- How large are the EX/MEM and IF/ID pipeline registers?
- If the opcode is 0x00 the effect is a `nop`. Look at *ps_control.vhd* and convince yourself you understand why this is produces a `nop`.

Assignment Part II: Pipeline Functionality (200 pts)

This assignment is comprised of the following steps. **<u>You may discuss solutions with colleagues. However the coding and testing must be performed independently!</u>**

1. Extend the data path to implement data forwarding for the supported instructions. The instructions supported correspond to those supported in the datapath in the earlier assignments.

   Your implementation should correctly execute all programs that have dependencies between register-register instructions that are supported by the datapath. Note that to operate correctly with this modification, programs must be written with:

   a) A single delay slot (`nop` instruction) when there is a load→ALU hazard, and

   b) `nop` instructions following a branch (how many?)

2. Extend the datapath with hazard detection to *automatically* insert a stall cycle in hardware in the presence of a lw instruction that loads a register that is the source register for an immediately following ALU instruction.

   Your implementation should correctly execute all programs that have dependencies between load instruction and an immediately following ALU instruction.

   The beq instruction should execute correctly if the program has the correct number of nop instructions.

Recommendations

Here are some suggestions for how to proceed. You of course can pursue other code designs.

Consider forwarding:

1. Implement the forwarding hardware in *ps_execute_vhd.* This avoids adding a new VHDL module and reduces a lot of top-level signal additions and routing that would otherwise have to be performed.
2. Implement the forwarding equations as captured on slide 50 of the course notes (also described in more detail in your text).
3. To implement the previous steps you will need signals from multiple pipeline stages routed to the execute module since you can forward from upstream in the pipeline. This means,
   a. You need to add ports *to ps_execute.vhd*
   b. Since you added ports to *ps_execute.vhd*, you will need to change the component declaration for the EX stage in *spim_pipe.vhd*.
   c. When *ps_execute.vhd* is instantiated in *spim_pipe.vhd*, you will need to connect these newly added ports to local signals that carry the information you need. Depending on how you do this, you may or may not have to declare some new signals in *spim_pipe.vhd*.
4. The documentation for the model describes steps needed for adding signals to the datapath.
5. Note that in the figure for the pipelined datapath in your text, the ALUSrc mux is missing!
6. After adding forwarding, be careful where you pickup the contents of register rt in EX to move to the next stage! You should move the forwarded value onto the MEM stage, not the value read in the register file in the ID stage.
7. Implement forwarding first and save this body of VHDL code. In case you do not get the lw hazard working, you can submit a working version of the pipeline with forwarding. Note that it does not make sense to implement the lw hazard detection logic without implementing forwarding.

Now consider load$\rightarrow$ALU hazard detection

1. To insert a stall cycle, the control signal values in the ID/EX register must be set to zero. However, I also recommend a reset of the entire ID/EX register (=0). This will prevent false hits in the forwarding logic making it easier to debug your code.
2. To implement the lw hazard you have to keep track of two successive instructions. Rather than create a new VHDL module for hazard detection, I suggest you perform this hazard check in the control module. Generate the stall

signal in *ps_control.vhd* and transmit it to wherever it is needed.

As with previous assignments, I strongly suggest first generating a design – i) specifying the functional changes to each VHDL module, ii) making changes in the figure of the full datapath, e.g., addition of new signals, and iii) write code to implement these changes.

Miscellaneous Comments:

1. You can use simple test programs for each case.
   i. A register-register hazard test case requires only 2 instructions for the simplest case. Remember to include a longer test case that requires forwarding from both MEM and WB stages.
   ii. A load → ALU hazard requires only 2 instructions.

2. You can add additional signals to the trace. Look at beginning of the Architecture for *spim_pipe.vhd* to see how some local signals are routed to the ports. Alternatively, check the ModelSim documentation.

Extra Credit (150 pts)

Extra credit is exactly that - *extra*. If you do not attempt this, it will not affect your grade in any way. The extra credit part is due at the same time with the same documentation.

1. Assuming a *branch-not-taken* instruction fetch policy, add the ability to flush the pipeline when a branch is taken and recover the proper instruction (i.e. target address). Minimize the number of nops required.
2. You should provide a proper test case and trace for this purpose.

Grading Guidelines

1) A complete and correct design as evidenced by the code and writeup
   a) Forwarding: **35** pts
   b) Stall cycle insertion: **35** pts
2) Compiles and executes
   a) Forwarding: **35** pts
   b) Stall cycle insertion: **30** pts
3) Execution has correct output: **45** pts
   a) Forwarding: **22.5** points
   b) Stall Cycle insertion: **22.5** points
4) Clarity of description and documentation (including code documentation, i.e., comments in the code): **20** pts

Submissions Instructions

Brevity and precision is valued over volume.

3. Design & Verification Report: A Single PDF file that contains the following.
   a. Your name and GTID on the top of each page
   b. A figure of the pipelined datapath clearly (label your changes 1, 2, ... or A, B, ...) showing the modifications you made to the datapath.
   c. Specify which VHDL modules you modified and how they were modified (be brief

and direct). Map these changes to the labeled changes of the figure.

d. A sequence of steps briefly describing how your code achieves this functionality.
e. A screen trace of the provided test program in *ps_fetch.vhd*.

4. Code: In a single zip file submit (lastName_firstName.zip)

a. The complete VHDL code for the modified datapath. It should be in a form whereby all that needs to be done is compiling & execution (as in the case of the model that was provided to you).

b. Documentation in Code:

i. Include your name and GTID in each module of code (even the unmodified ones)

ii. At the top of each module, comments indicating what changes you have made to each module. This can be brief.

All submissions should be electronic via Tsquare. Projects are accepted late but will be docked **10 percentage points per day**.