Assignment 4

Due date: 11:59 PM, Thursday, March 31st, 2016

Homework Problems Due at the Beginning of Class, Thursday, March 31st, 2016

Purpose:

Generate a detailed understanding of the operation of the MIPS pipeline. 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.

The VHDL model code is available from the class website.

Assignment Part I: Understanding the Model

This part does not have a submission requirement. The warm-up can be discussed with your classmates. 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.

- 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 (in bits) 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 (150 pts)

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

1. Implement data forwarding to the EX stage.
2. Add support for the load-to-use stall. For example, if a `lw` instruction produces data used by an immediately following instruction, one stall cycle should be included between the `lw` and the immediately following instruction.
3. Assume branches are predicted as not taken. Thus, the fetch stage will keep fetching instructions through PC+4 until the branch condition is resolved. Note that if the branch is taken, then the instructions which were speculatively fetched would have to be flushed.
4. We will provide test programs. In the interim create your own simple test

programs. Remember that the data path supports only those instructions supported by the data path described in the text, e.g., no support for immediate instructions.

5. Do pay attention the documentation provided with the data path.
6. For the test program,
    a. Calculate the CPI for the test program for the baseline (unmodified) data path. You will have to do this via the insertion of nops as necessary.
    b. Now calculate the reduction in CPI (Cycles Per Instruction) obtained after the modifications.

Grading Guidelines

1) A complete and correct design as evidenced by the code and writeup – **45** pts.
2) Compiles and executes: **35** pts.
3) Execution has correct output: **45** pts
4) Clarity of description and documentation (including code documentation, i.e., comments in the code): **25** 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 data path **clearly** (label your changes 1, 2, ... or A, B, ...) showing the modifications you made to the data path.
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 the required functionality.
e. You will be provided a set of test programs. For each test program provide a screen trace (exact instructions will be provided).

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

a. The complete VHDL code for the modified data path. 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.

Programming submissions should be electronic. Submissions must time stamped by the due date and time. Submissions will be via Tsquare.

Problems (50 points)

1. (**30 pts**) Consider the state of following procedure executing within the SPIM pipeline (Figure 4.51) during cycle 11. Assume that the first instruction is fetched in cycle 0! Assume full hardware support for forwarding, branches predicted as not taken, and flushing when branches are taken. The first instruction is at address 0x00400000. The data segment starts at 0x0100. Assume the data path supports the addi and bne instructions similar to add and beq respectively.

```
                .data
start:  .word 21, 22, 23,24
str:    .asciiz "CmpE"
        .align 4
        .word 24, 0x77

                .text
        .globl main

main:   addi $t3, $0, 0x4
        addi $t0, $0, 0x0100
        addi $t1, $0, 0x0200
move:   lw $t5, 0($t0)
        sw $t5, 0($t1)
        addi $t0, $t0, 4
        addi $t1, $t1, 4
        addi $t3, $t3, -1
end:    bne $t3, $zero, move

        addi $v0, $0, 0xA
        syscall
```

   a. Insert required nops to ensure this program executes correctly. Explain why it will not execute correctly.

   b. What instructions are in each stage of the pipeline after the preceding correction is applied.

      IF _____

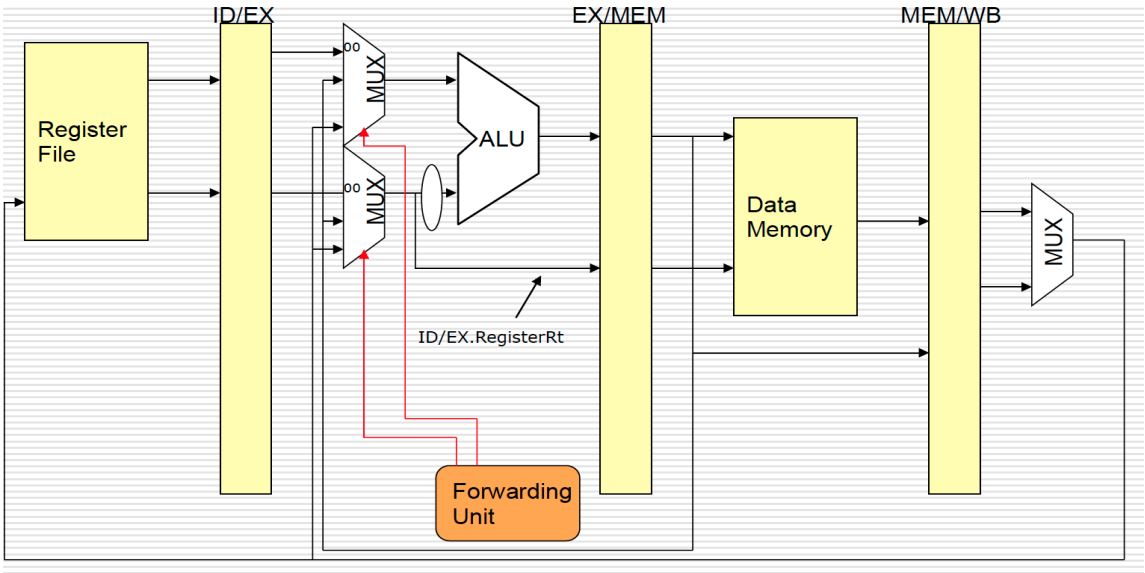
      ID _____

      EX _____

      MEM _____

      WB _____

c. What are the values of the following fields? **Mux inputs are numbered from the top to bottom starting at 0**. MuxA is the first one from the top in EX. The figure below is just clarify notation for forwarding. Use Figure 4.51.



MEM/WB.MemToReg _____

ForwardMuxA _____

ForwardMuxB _____

IF/ID.PC _____

ID/EX.RegWrite _____

EX/MEM.WriteData _____

EX/MEM.WriteRegister (Destination Address) _____

d. Reschedule the code using `nops` to minimize/eliminate stall cycles.

2. (**10 pts**) Add support for the jump instruction to the data path in Figure 4.51. To receive credit your solution should gave the following elements.
    a. Clearly show the hardware changes required of the datapath using Figure 4.51. If your drawing or handwriting is not legible you will not receive credit.
    b. Describe briefly how this would work.
        i. Operation in a few sentences
        ii. Changes to the controller and control signals.

3. (**10 pts**) Extensive analysis of the instruction stream of code targeted for the pipelined MIPS with forwarding (Figure 4.51) has produced the following statistics. To gain some performance the EX and MEM stages have been partitioned into two stage pipelines. The overall pipeline is now 7 stages. 30% of load instructions produce a load-to-use hazard.

Instruction	Frequency
Loads	22%
Stores	13%
ALU Operations	42%
Branches	18%
Jumps	5%

    a. What is the penalty in cycles for i) data hazards, and ii) control hazards.
    b. 20% of these load delay slots can be filled via instruction scheduling. Similarly, we can use scheduling can fill 60% of the branch delay slots. Compute the CPI if the base (ideal) CPI is 1.0.
    c. How would branch prediction affect the preceding analysis if a correct prediction can reduce the branch penalty to 1 cycle?

Note: No late assignments will be accepted. You must make achieve a minimum average of 50% in the assignments to pass the course.