CprE 381 Homework 5

[Note: This homework gives you some more practice with MIPS programming and thinking about testing your processor. As a happy coincidence, it will help provide a class-wide test program that I will share with you. Then the homework begins to cover processor design choices.]

1. Instruction Test
    a. Write a set of MIPS code sequence to stress test one of the MIPS instructions you are implementing for your projects (i.e., it should test the range of operand values, any exceptional behavior, and any other edge cases you can think of). There should be at least three such tests, each in their own assembly file whose name is your "<your instruction mnemonic>_<your NetID>_<test_number>.s".

    You may only use those instructions above yours in the list. In general, use as few other instructions as possible (we wouldn't want to see any false failures). In the comments, you **must** justify why you are including each specific test case and why the test has value (e.g., what edge case it covers or what common case it covers). I have included an example test file for you to use as a temple. You don't necessarily need to use as many instructions as I have for this test (part of my purpose was checking all registers could be cleared).

    Note that these files will be used to help test your implementations. As such, upload your function in a separate .s file. The instruction you are implementing is as below:

Last Name Starts	**Instruction to Stress Test**
Zh, Za, Wo	**addi**
Wi, Wa	**lui**
Vl, Ur, To	**ori**
Ti, Th	**add**
St, So	**addu**
Sn, Sm	**and**
Sk, Sh	**andi**
Sc, Se	**addiu**
Sa, Ro	**lw add**
Ri	**nor**
Re, Ra	**xor**
GNe, Na	**xori**
Mo	**or**
Me, Mc	**slt**

Ma	**slti**
Li, La	**sltiu**
Ko, Ki	**sltu**
Ji, He	**sll**
Ha, Gu	**srl**
Go	**sra**
Ga, Fu	**sllv**
El, Du	**srlv**
De, Da	**srav**
Cu, Co	**sw**
Ch	**sub**
Bus	**subu**
Bur, Bue	**beq**
Br, Bi	**bne**
Ba	**j**
Al	**jal**
Ad, Ab	**jr**

2. Processor Implementation Details

   P&H(4.2) <§4.1>. The basic single-cycle MIPS implementation in Figure 4.2 can only implement some instructions. New instructions can be added to an existing Instruction Set Architecture (ISA), but the decision whether or not to do that depends, among other things, on the cost and complexity the proposed addition introduces into the processor datapath and control. The first three problems in this exercise refer to the new instruction:

   Instruction: **lwi** rt, rd(rs)
   Interpretation: Reg[rt] = Mem[Reg[rd]+Reg[rs]]

   a. Which existing blocks (if any) can be used for this instruction?
   b. Which new functional blocks (if any) do we need for this instruction?
   c. What new signals do we need (if any) from the control unit to support this instruction?

3. Processor Cycle Time Determination

   Assume the following latencies for the logic blocks in Figure 4.17 from the textbook.

I-Mem	Adder	MUX	ALU	Reg Read	D-Mem	Sign-Extend	Shift-Left-2	Control	ALU Control	AND gate
200ps	70ps	20ps	90ps	90ps	250ps	10ps	5ps	40ps	20ps	10ps

a. Identify and quantify (i.e., give the path through the blocks and the time for that path) the worst-case path for each of the following: an arithmetic R-format instruction, a **`lw`** instruction, and a conditional branch instruction. *[Note that in the second part of your project you will actually synthesize your processor and be able to identify its critical path—you'll find out the specific delays for the components you've designed when they are mapped to an FPGA. This problem should help you develop an intuitive sense of how to reason about critical paths.]*

b. Rank the following design approaches in terms of which improve the cycle time the most. You **must** justify your ranking.
    i. Developing an "extra fast adder" that reduces the Adder and ALU latencies
    ii. Changing the ISA memory addressing mode to directly use the base register's value (i.e., no offset addition)
    iii. Designing a lower-latency control unit