QUIZ 4 Introduction to Computer Systems COMP 273
Mon. March 14, 2016 Professor Michael Langer

LASTNAME: _____ FIRSTNAME: _____ ID: _____ GRADE: _____/4

Instructions:

You are allowed one crib sheet. Once you finish, turn your paper over and wait for the end of the quiz.

Questions:

1. (**2 points**)

   Draw the datapath for the MIPS instruction:

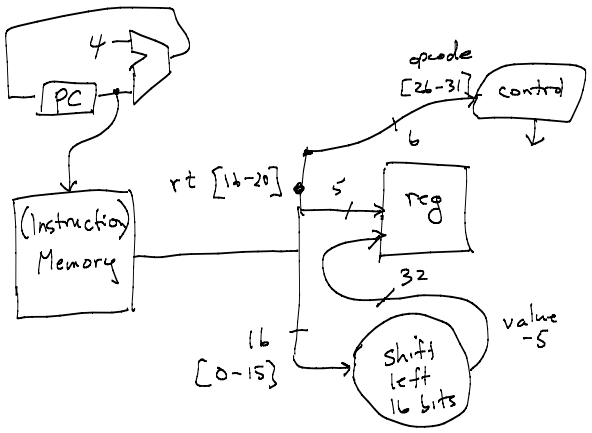
```
lui    $s0,  -5      #  load upper immediate
```

   which puts the immediate value in the upper two bytes of the word, and puts 0 into the lower two bytes.

   Your solution must show only parts of the datapath that are used by the instruction. Indicate where are the arguments `$s0` and `-5`, and what is the bit width of all lines.

Solution

I was expected a sketch such as the following.



Note that this is an I format instruction although the question didn't explicitly ask for this.

Grading scheme:

We gave 1 point for taking the 16 bit immediate argument and doing something to get it into the upper two bytes of a word, for example, "shift left (16 bits)". If however you wrote "sign extend", then you didn't get the point. Sign extending leaves the immediately argument in the lower two bytes of the word.

For the second point, you needed to write this 32 bit value directly into a register. Alternatively, you could run it through the ALU, such that the other input to the ALU was 0 and you were performing a sum, and then write the output of the ALU into a register. Some students put a (data) Memory element in the data path (MEM stage in pipelining). This is incorrect: this instruction does not use the data part of Memory. No second point if you did that.

2. (**2 points**)

(a) Give an example of a data hazard involving `$s0` in the instruction

```
sw  $s0,  12( $s4)
```

by adding one MIPS instruction, either before or after the above instruction.
Briefly explain why this is a hazard, using the pipeline stages indicated below.

add	IF	ID	ALU	MEM	WB	
sw		IF	ID	ALU	MEM	WB

SOLUTION:

```
add  $s0,  $s1,  $s2   <---   insert here
sw   $s0,  12( $s4)
```

The inserted `add` instruction creates a data hazard because `$s0` isn't written by `add` until the end of `add`'s WB stage, which occurs in the same clock cycle as `sw`'s MEM stage But `sw` needs that value *during* the MEM stage.
Other solutions were possible e.g.

```
lw   $s0,  12( $s4)
sw   $s0,  12( $s4)
```

This creates a data hazard for the same reason as in the previous example, namely the `lw` wouldn't update the `$s0` register until the WB stage, but `sw` needs the new value in `$s0` earlier.

(b) Briefly describe how this hazard can be avoided. For full points, minimize the number of extra clock cycles caused by the hazard.

SOLUTION (for the `add` example above) :

It is unnecessary to insert a `nop`. Instead, the result of the `add` which is computed in `add`'s ALU stage can be forwarded, so that it is available to `sw` in its MEM stage which is where it is needed.

[Some students put the `add` *after* the `sw`.

```
sw   $s0,  12( $s4)
add  $s0,  $s0,  $s2   <---  some students put an "add" here
```

This is incorrect. It does not create a data hazard, since the `sw` instruction does not modify the `$s0` register. By the time the `add` instruction writes back to the register (WB stage), the `sw` instruction is done.]