HW #8

CSEE W3827 - Fundamentals of Computer Systems Spring 2022

Prof. Rubenstein
Due 4/15/2022, 5pm

Topics: Pipelining

Note that this homework has 5 problems and is 3 pages long.

1. For each instruction below, each parameter "holds" a value that is either retrieved during the processing of the instruction, or computed during processing and/or stored during processing. For instance, the instruction add $s1, $s2, $s3 has three parameters: $s1, $s2, $s3. $s2 and $s3 have values that are retrieved and $s1 has a value that is stored.

   Indicate for each parameter of each instruction below in what stage the values are retrieved, in what stage they are computed, and in what stage they are stored, when applicable. For registers we are interested in the values in the registers, not the register numbers themselves.

   For instance, for add $s1, $s2, $s3: the values in $s2 and $s3 are retrieved during the ID stage, the value to be stored in $s1 is computed in the EX stage, and stored in the WB stage.

   (a) addi $s1, $s2, 100

   (b) lw $s1, 400($s2)

   (c) sw $s1, 400($s2)

   (d) beq $s1, $s2, LABEL

2. Suppose the MIPS architecture were modified so that lw and sw commands did not use a constant offset from the memory address, i.e., instructions were just of the form:

   ```
   lw $s1, $s2
   ```

   How could one shorten the pipeline? How would data dependencies change with these instructions as a result?

3. Consider the following two sequences of instructions:

i.	lw $s1, 40($s6)
	add $s6, $s2, $s2
	sw $s6, 50($s1)

ii.	lw $s5, -16($s5)
	sw $s5, -16($s5)
	add $s5, $s5, $s5

   Separately for i and ii:

   (a) Identify all data dependencies (whether or not they may cause a hazard or stall).

   (b) Assume there is no forwarding in this pipelined processor. Indicate hazards and add nop instructions to eliminate them.

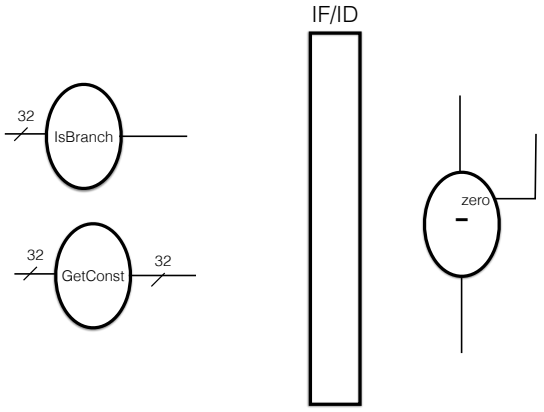
   (c) Assume there is full forwarding. Indicate hazards and add nop instructions to eliminate them.

4. For the following code, identify all data dependencies. Which dependencies can be resolved via forwarding? Which will cause a stall?

```
add $s3, $s4, $s2
sub $s5, $s3, $s1
lw $s6, 100($s3)
add $s7, $s3, $s6
```

5. Suppose we wish the architecture to implement branch prediction that assumes that branch instruction conditionals evaluate to true. You may assume the existence of two special circuits that take in as input the instruction and can return a result quickly enough that they can be applied within the IF stage:

   - IsBranch looks at the OpCode and returns 1 if the instruction is a branch instruction. For the purposes of this problem, you may assume that the only branch instruction we are dealing with is beq.
   - GetConst takes in the instruction and returns as a 32-bit signed binary what is potentially the 16-bit constant (i.e., it simply pulls out the 16 lowest order bits of the instruction and sign-extends the result).



Note that the conditional itself cannot be evaluated in the IF stage because it still requires the comparison of values that are stored in the register file.

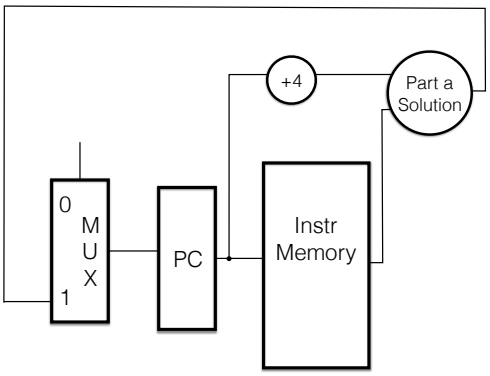
(a) Draw the circuitry that, given a current value of the PC and instruction memory, uses GetConst and IsBranch to the PC to the next instruction it should pull from instruction memory (i.e., PC$+4$ when not a branch instruction, and the appropriate value when it is a branch instruction). In addition to the PC, instruction memory, GetConst and IsBranch circuits, you may also use:

   - A shifter (that can shift left by 2)
   - Another adder.
   - MUXes

(b) The architecture depicted on slide 204 of (the most recently revised version of) Lecture 12 uses branch prediction but assumes the conditional is false (i.e., the next instruction is at PC $+4$) Aside from the change made in part (a), what else needs to be altered such that the pipeline can "recover" when the conditional turns out to be false?

   i. What additional/new information needs to be passed along the various stages (hint: in case the conditional turns out to be false)?

ii. To implement this branch prediction, suppose in addition to your solution in part (a) that computes the address of a branch (assuming the conditional is true), an additional MUX is added to set the PC for the case that the conditional winds up being false. Using the figure below as the parts of the architecture that need modification, draw the necessary additions to the pipeline that properly sets the PC when the conditional is noted to be false.



Note that in your solution the $+4$ circuit might have been embedded inside your part (a) - it has been pulled out here for a reason...