Assignment 5: MIPS Pipeline

This assignment is based on MIPS pipeline simulation using WinMIPS64 simulator. The simulator and its documentation/tutorial is available at http://indigo.ie/~mscott/. Since the simulator is based on windows, you will have to use something like Wine (https://www.winehq.org) to run it on ubuntu or macOS.

Submission Instructions: Make a directory named with your roll number. For each problem *i*, write your code (if required by the problem) in a file named *i*.s and include them in your submission directory. Also write answer to each question just below it and include this file in your submission. Finally submit a tar.gz i.e., [roll-no.].tar.gz (as follows) on moodle:

```
[roll-no.]
   |----1.s
   |----2.s
   :
   |----n.s
   |----a5.docx (or any other format as you like)
```

1. Write a snippet of assembly code and use the simulator timing diagram to deduce whether each of the following is pipelined and, if so, the number of stages involved in each operation.
(i) Integer multiply
(ii) integer divide
(iii) FP add
(iv) FP multiply
(v) FP divide

2. Configure the pipeline to (a) suppress forwarding and (b) enable forwarding for different snippets of code.

3. Write a 2-instruction sequence to maximize the time between the start of the first instruction and the end of the last. Repeat for a 3-instruction sequence.

4. Write a snippet of code with the potential to create a structural hazard in the

multi-cycle execution case. Is this resolved in the pipeline and, if so, how?

5. What is the time to execute a pair of multiply instructions with the second having a RAW dependency on the first, with and without forwarding?

6. Write a snippet of code to determine the branch penalty in each case – (a) branch not taken and (b) branch taken? What are the penalties in each case and what do you conclude about the stage in which the branch condition is evaluated and the stage in which the target address is computed?

7. Experiment by configuring the pipeline to support a branch delay slot. Show a snippet of code in which this feature is useful and one where this feature is not.

8. Write a snippet to illustrate a potential WAW hazard. Is this situation handled in the pipeline and, if so, how?

9. Can a WAR hazard exist in this pipeline? If so, how? If no, why not?

10. It is required to sum 12 FP numbers residing in registers f1 through f12. Write the snippet of code to perform the summation with the minimum number of clock cycles. What is the minimum time required? Assume that the final sum should be in f13 and that the contents of registers f1 through f12 should not be destroyed.

ANSWERS - 160050064

1. (i) Pipelined - 7 stages (M0 to M6)
   (ii) Not pipelined
   (iii) Pipelined - 4 stages (A0 to A3)
   (iv) Pipelined, 7 stages (M0 to M6)
   (v) Not pipelined

2. Simply checked/unchecked the 'Enable Fowarding' option in Configure tab of WinMIPS64.
When forwarding is not enabled, the 2nd instruction 'dmul r4,r2,r3' will run after 1st instruction 'dmul r3,r1,r2' has finished WB as r3 is written in 1st instruction and during this gap, 2nd instruction will be in raw state.
When forwarding is enabled, the additional hardware support gets the required r3 value to 2nd instruction earlier and hence finishes with lower clock latency cycles using forwarding.

3. Two div.d (as we know, its not pipelined and has high latency) instructions one after the other with the result of first instruction as argument of second will lead to largest time interval between two instructions (start of first and end of second).
Similarly, we can extend this concept to three instructions as doing three divisions like a=b/c, d=a/e, e=a/f.

4. The potential structural hazard in 4.s is resolved in this pipeline due to use of multiple memories.

5. With forwarding - 18 cycles
   Without forwarding - 20 cycles

6. If branch not taken - 0 penalty cycles
   If branch taken - 1 penalty cycle

8. dmul has more latency than dadd, but for correctness, dmul must write first and then the dadd should write. And, yes the pipeline indeed handles this, as a waw occurs and 2nd instruction write occurs only after first instruction in the pipeline.

9. No, a WAR hazard is not possible in this pipeline because registers are read in ID stage, and instructions pass through ID stage in program order, and give results only after leaving the ID stage. A WAR hazard is possible only when a register read is delayed.

10. Minimum number of clock cycles are 35. The solution is a more like a binary tree.