CS613 Spring 2012 Exam 2 NAME __solutions_____

1. For the MIPS pipeline we studied, is there a data hazard in the following code segment? What type?

```
lw    $s0, 4($s1)
add   $t1, $s1, $s1
sw    $s0, 8($s1)
```

The lw instruction writes $s0, the sw instruction reads it, so this is a WAR hazard.

2. Discuss how the way we do forwarding for the sequence:

```
add r1,r2,r3
sub r4,r1,r3
```

is different from the way we do forwarding for a "Load-Use" sequence.

The key thing to note is that while we can forwarding for the add-sub sequence and eliminate all stalls, we cannot do so for a Load-Use sequence.

3. Show how static scheduling could be used to eliminate Load-Use hazards in the following code.

```
LW    $s0,0($t0)
LW    $s1,0($t1)
ADD   $s2,$s0,$s1
SW    $t2,0($s2)
LW    $s3,0($t3)
LW    $s4,0($t4)
```

Here is one solution, assuming we have forwarding:

```
LW    $s0,0($t0)
LW    $s1,0($t1)
LW    $s3,0($t3)
ADD   $s2,$s0,$s1
LW    $s4,0($t4)
SW    $t2,0($s2)
```

4. Given an arbitrary branch instruction in a program, it is slightly more likely that the branch will be taken than that it will "fall through". Yet, the MIPS architects found that a "Predict Not Taken" branch prediction strategy results in higher throughput than "Predict Taken". Why is this?

"Predict not taken" only need the address of the next instruction in program order. That address is available in the IF cycle, so the next instruction can be loaded without a stall. If we did "Predict Taken", we would have to calculate the "taken" address (in the EX stage).

5. Explain what it means to say that two instructions are "parallel".

The key idea is that the instructions can be executed in either order and they can be overlapped to any extent without changing the program results.

You can also say: "they can execute simultaneously in a pipeline of arbitrary depth without causing any stalls (assuming no structural hazards)"

6. What type of hazard is associated with the following dependency types?
   a. Data dependence
   RAW

   b. Anti-dependence
   WAR

   c. Output dependence
   WAW

7. Use register renaming to eliminate the dependencies in the following code segment:

```
add     r4,r1,r3
sub     r1,r2,r3
add     r6,r1,r7
```

There are two dependencies in this sequence. You can rename r1 in the second and third instructions and eliminate the first dependency, but you can't eliminate the second one with register renaming alone. To get full credit, I expected you to notice the second dependency as well as the first.

8. Assume that a Branch statement in a program has the following history:

   Taken
   Taken
   Not taken
   Taken
   Not taken

   What would the next prediction be if you were using:
   a. A 1-bit BHT
   Not Taken

b. A 2-bit BHT
Taken

9. Unroll the following loop 1 time. Use register renaming to eliminate the dependencies that are introduced by unrolling:

```
UAH:  lw    $s3, 0($t4)
      addi  $t4, $t4, -4
      sub   $s3, $s3, $s2
      sw    $s3, 4($t4)
      bne   $s1, $zero, UAH
```

This is the same as the example presented in the lecture slides.

10. Explain how the "Issue" stage of the Dynamic Scheduling approach we studied is different from the "Instruction Decode" stage of the conventional pipeline processor we studied.

The Issue stage includes the "Issue" operation, whereas ID does not.

11. For Tomasulo's approach:
    a. Explain how Reservation Stations are related to Register Renaming.

       When an instruction is executed, the register references are replaced with references to Reservation Stations. Given enough Reservation Stations, this serves to "rename" the registers.

    b. Explain why the CDB is referred to as a "come from" bus.

       Data on the CDB is tagged to identify the source of the data, not the destination, so it tells where the data "comes from".

12. Explain the purpose(s) of the ROB in Tomasulo's approach.

The ROB holds executed instructions until they are committed.

13. Explain what we mean by the terms "speculative" and "commit".

An instruction is speculative if it is executed because of a branch prediction, and the true branch outcome is not yet available. The instruction is committed when the outcome is known (to be the same as the prediction).

14. Explain the differences between Fine-Grained and Coarse-Grained Multithreading

Fine-grained MT swaps thread at the end of each instruction. It requires extremely fast thread switching to sustain an acceptable throughput. It hides short stalls as well as long stalls. Coarse-Grained MT swaps threads only when a long stall occurs. Consequently,

it cannot hide short stalls.