## **Computer Sciences Department**

## **University of Wisconsin-Madison**

## **CS/ECE 552 – Introduction to Computer Architecture**

## **In-Class Exercise (02/20)**

**Answers to all questions should be uploaded on Canvas.**

1. [1 point] From the textbook: (Twist on Check Yourself 4.6) A group of students were debating the efficiency of the 5-stage pipeline when one student pointed out that not all instructions are active in every stage of the pipeline. After deciding to ignore the effects of hazards, they made the following four statements. Which one(s) are correct?

1. Allowing jumps, branches, and ALU instructions to take fewer stages than the 5 required by the load instruction will increase pipeline performance under all circumstances.

2. Trying to allow some instructions to take fewer cycles does not help, since the throughput is determined by the clock cycle; the number of pipe stages per instruction affects latency, not throughput.

3. You cannot make ALU instructions take fewer cycles because of the writeback of the result, but branches and jumps can take fewer cycles, so there is some opportunity for improvement.

4. Instead of trying to make instructions take fewer cycles, we should explore making the pipeline longer, so that instructions take more cycles, but the cycles are shorter. This could improve performance.

*Given that #2 and #4 are correct, why does #4 help?*

Longer pipeline lead to small cycles so less CPI and less clock cycle. This increase throughtput and more instructions can be executed.

2. [2 points] Assume the five-stage pipeline with full forwarding (excluding MEM-to-MEM forwarding) and RF bypassing support. How many cycles does this code take to execute?

|  |  |
| --- | --- |
| **sub** | **$s4, $s2, $s3** |
| **sw** | **$s2, 8($s1)** |
| **lw** | **$s2, 0($s4)** |
| **lw** | **$s5, 4($s2)** |
| **add** | **$s1, $zero, $s5** |
| **lw** | **$s3, 0($s3)** |
| **sw** | **$s3, 0($s1)** |
| **add** | **$s2, $s1, $s2** |

18 cycles.

3. [2 points] Consider the following MIPS assembly code, assuming the five-stage pipeline with full forwarding and bypassing support:

|  |  |  |
| --- | --- | --- |
| **L1:** | **or** | **$s1, $s2, $zero** |
| **L2:** | **add** | **$s1, $s5, $s3** |
| **L3:** | **and** | **$s1, $s1, $s4** |

1. [1 point] Assume that L1 is currently in the WB stage, L2 is in the MEM stage and L3 is in the EX stage. Give the current values in the fields below of the pipeline registers ID/EX, EX/MEM and MEM/WB.

|  |  |
| --- | --- |
| ID/EX.RegisterRs | S2 |
| ID/EX.RegisterRt | S1 |
| EX/MEM.RegisterRd | S1 |
| EX/MEM.RegisterRt | S4 |
| MEM/WB.RegisterRd | S5 |

(b) [1 point] The timing diagram for this sequence of instructions is shown below. Indicate which of the conditions below were necessary in enabling EX-stage forwarding between instructions L2 and L3 (arrow in the diagram).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction \ clock cycle** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| L1 | F | D | X | M | W |  |  |  |  |
| L2 |  | F | D | X | M | W |  |  |  |
| L3 |  |  | F | D | X | M | W |  |  |

|  |  |
| --- | --- |
| **Condition** | **Enabled L2 🡪 L3 Forwarding?** |
| EX/MEM.RegisterRd == ID/EX.RegisterRs | y |
| EX/MEM.RegisterRd == ID/EX.RegisterRt | y |
| MEM/WB.RegisterRd == ID/EX.RegisterRs | n |
| MEM/WB.RegisterRd == ID/EX.RegisterRt | n |
| EX/MEM.RegisterRd != ID/EX.RegisterRs | y |
| MEM/WB.RegisterRd != ID/EX.RegisterRs | n |
| ID/EX.RegWrite == 1 | y |
| EX/MEM.RegWrite == 1 | n |
| MEM/WB.RegWrite == 1 | n |
| EX/MEM.RegWrite == 0 | n |
| MEM/WB.RegWrite == 0 | y |
| ID/EX.RegisterRd != 0 | y |
| EX/MEM.RegisterRd != 0 | y |
| MEM/WB.RegisterRd != 0 | y |

4. [5 points] Consider the following loop.

1.   
2.   
3.   
4.   
5.   
6.   
7.   
8.

(a) [1 point] Identify all data dependencies (potential data hazards) in the given code snippet. Assume the loop takes *exactly* one iteration to complete. Specify if the data dependence is RAW, WAW or WAR.

3-4, 2-4 RAW

4-5 RAW, WAR

3-6 WAR

6-8 RAW

(b) [1 point] Assume a 5-stage pipeline (IF ID EX MEM WB) without any forwarding support, but with support for a register read and write in the same cycle (i.e., RF bypassing). Also assume that branches are resolved in the ID stage and handled by stalling the pipeline. All stages take 1 cycle. Again, the loop takes one iteration to complete. Which dependencies from part (a) cause stalls? How many cycles does the loop take to execute?

4-5, 3-6, 16 cycles.

(c) [1 point] Assume that the pipeline now supports full forwarding and bypassing. Furthermore, branches are handled as predicted-not-taken. As before, the loop takes one iteration to complete. Which dependencies from part (a) still cause stalls and why? How many cycles does the loop take to execute now?

3-4

13 cycles

(d) [2 points] (Challenge) If the pipeline from part (c) instead uses a branch delay slot, how would you schedule the instructions in the loop to minimize stalls? For this part, assume the *loop takes multiple iterations to complete*. Explain your answer.

Branch predict: always predict loop will continue instead of exit the loop, so fetch the first instruction of the loop when branch instead of assume the branch will not taken and fetch the next instructions.