## **Computer Sciences Department**

## **University of Wisconsin-Madison**

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

## **In-Class Exercise (02/27)**

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

1. [3 points] From the textbook:

1. [1 point] (Twist on Check Yourself 4.9) Which exception should be recognized first in this sequence?
2. add $1, $2, $1 # arithmetic overflow
3. XXX $1, $2, $1 # undefined instruction
4. sub $1, $2, $1 # hardware error

Given that 1 is the correct answer, what scenario would lead to either 2 or 3 being recognized first?

**1 does not have an overflow, 2 get a problem at decoding and if instruction memory fail than 3 first**

1. [1 point] (4.9) When we encounter an exception in the EX stage of our 5-stage pipeline, why do we not want to flush the instructions currently in the MEM and WB stage?

Next instruction might not have any hazard

1. [1 point] (4.9) Given the following instruction sequence that executes in the 5-stage MIPS pipeline, what is the value of the EPC register after an exception occurs in the sub instruction?

…

0x40 sub $11, $2, $4

0x44 and $12, $2, $5

0x48 or $13, $2, $6

0x4C add $1, $2, $1

0x50 slt $15, $6, $7

0x54 lw $16, 50($7)

…

0x30

2. [2 points] Consider the following MIPS assembly program:

|  |  |  |
| --- | --- | --- |
| DEST1: | addi | $s2, $zero, 16 |
|  | and | $s4, $s4, $s1 |
|  | slt | $s6, $s1, $s2 |
|  | beq | $s6, $s0, DEST2 |
| <branch delay slot> | | | |
|  | or | $s3, $s4, $s6 |
|  | add | $s1, $s1, $s3 |
| DEST2: | slt | $s7, $s2, $s5 |
|  | and | $s3, $s7, $s1 |
|  | bne | $s3, $zero, DEST1 |

In the table below, specify which of the instructions can individually be **legally removed** from its current position in the program and **placed into the branch delay slot**. And if the instruction can be moved, specify if it would introduce any wasted work in the branch-taken path and/or fall-through (i.e., branch-not-taken) path.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Can be moved?** | **Wasted work on branch-taken?** | **Wasted work on fall-through?** |
| addi $s2, $zero, 16 | N |  |  |
| and $s4, $s4, $s1 | y | y | n |
| slt $s6, $s1, $s2 | n |  |  |
| or $s3, $s4, $s6 | y | y | n |
| add $s1, $s1, $s3 | n |  |  |
| slt $s7, $s2, $s5 | y | n | n |
| and $s3, $s7, $s1 | n |  |  |

3. [2 points] Consider two different 5-stage pipeline machines (IF ID EX MEM WB). The first machine resolves branches in the ID stage, uses one branch delay slot, and can fill 75% of the delay slots with useful instructions. The second machine resolves branches in the EX stage and uses a predict-not-taken scheme. Assume that the cycle times of the machines are identical. Given that 20% of the instructions are branches, 25% of branches are taken, and that stalls are due to branches alone, which machine is faster? To get any credit, you must justify your answer.

1. 4/5\*n\*CPI + ¼\*n/5\*(CPI+1)=17/20nCPI + n/20

2. 4/5\*n\*CPI+n/20\*(CPI+2)=17/20\*CPI+1/10n

1 is 5% faster

4. [3 points] In our five-stage pipeline with full forwarding (excluding MEM-to-MEM forwarding) and RF bypassing, the conditions for **data hazard stalls** are shown below (from Chapter 4.7). Note that RegisterRd here refers to the destination register for either an R-type instruction (which comes from the instruction's Rd) or an I-type instruction (which comes from instruction's Rt).

if ( ID/EX.MemRead

and (ID/EX.RegisterRd ≠ 0)

and ((ID/EX.RegisterRd = IF/ID.RegisterRs) or (ID/EX.RegisterRd = IF/ID.RegisterRt))

) insert NOP at ID/EX; // stall cycle

Assume that we modify the pipeline such that branch decisions are now resolved at the ID stage as in COD Figure 4.65. Also assume that data forwarding from the EX stage is now available to the branch decision circuit at the ID stage. Fill in the blanks below for the new set of conditions for data hazard stalls in this modified pipeline. Note: jump instructions are still resolved in the EX stage.

if ( ( (ID/EX.MemRead or (ID/EX.Branch and **EX/MEM.**RegWrite))

and (ID/EX.RegisterRd ≠ 0)

and ((ID/EX.RegisterRd = IF/ID.RegisterRs) or (ID/EX.RegisterRd == IF/ID.RegisterRt))

)

or

( (**ID/EX.MemRead** and **ID/EX**.Branch)

and (**ID/EX.RegRd**≠ 0)

and ((**ID/EX.RegRd** == **IF/ID.RegRs**) or

(**ID/Ex.RegRd** == **IF/RegRt**))

)

) insert NOP at ID/EX; // stall cycle