**PLEASE READ BEFORE DOING HAZARDS HOMEWORK**

As is typical for our homework problems, we are asking you to apply what you have learned to conditions you may not have encountered.

**After completing the problem please be sure to pay attention** to the explanations in ”Show Answer”, where Professor Albonesi has added some further content to extend your learning.

## HOMEWORK 1. HAZARDS (1/1 point)

Consider the following LC-3 code segment:

```
LDR R1, R4, #0
ADD R3, R2, R1
```

Unlike in the previous forwarding examples where an operate instruction forwarded the result, here the result is forward by a load instruction. Note the difference: the load reads R1 from memory in the MEM stage, so the value of R1 is not available until the beginning of WB, rather than the beginning of MEM with an operate instruction. (We assume in all these examples that we forward values from the outputs of the pipeline registers.)

Which of the following statements describes how the pipeline would handle forwarding in this case? (You may assume that new hardware can be added to our existing pipeline if necessary.)

- The value of R1 would be forwarded from the LDR at the beginning of MEM to the ADD in EX.
- The value of R1 would be forwarded from the LDR at the beginning of WB to the ADD in MEM.
- [✓ The value of R1 would be forwarded from the LDR at the beginning of WB to the ADD in EX, but the ADD would have to wait in EX for 1 cycle while the LDR advances to WB.] ✓
- The value of R1 would have to be forwarded early in the pipeline when the LDR is in the ID stage.

**EXPLANATION**

Since the new R1 cannot be forwarded until WB and it is needed two stages earlier in EX, neither of the first two forwarding cases can work. Instead, the ADD must be forced to wait in EX for an additional cycle while the LDR advances into WB. We call this stalling the ADD, since it is “stuck” in EX for an additional cycle. The instructions in ID and IF must also stall.

What happens to the MEM stage while the ADD stays in EX and the LDR moves to WB? The pipeline forces a NOP (which stands for "no operation" and is pronounced "no op") instruction into MEM. A NOP instruction performs no action; it does not modify the register file, the PC, or memory. Computer architects call this "forcing a bubble" into the pipeline.

Recall that in LC-3, a NOP has all 0's, which is equivalent to "branch never." Therefore, to force a bubble into MEM, the control logic can simply reset the EX/MEM pipeline register.

Help

Final Check | **Save** | Hide Answer *You have used 1 of 2 submissions*

Show Discussion | New Post

## HOMEWORK 2. HAZARDS (1 point possible)

Consider the following LC-3 code segment:

```
LDR R1, R4, #0
STR R1, R4, #1
```

This sequence copies the data at the memory location pointed to by R4 to the next location in memory.

Which of the following statements describes how the pipeline would handle forwarding in this case? (You may assume that new hardware can be added to our existing pipeline if necessary.)

- ○ R1 would get forwarded from the LDR in WB to the STR in MEM. ✓
- ○ R1 would get forwarded from the LDR in WB to the STR in MEM, but the STR instruction would need to wait in MEM for 1 additional cycle.
- ● R1 would get forwarded from the LDR in EX to the STR in ID. ✗
- ○ R1 would get forwarded from the LDR in EX to the STR in ID, but the STR instruction would need to wait in EX for 1 additional cycle.

**EXPLANATION**

When the LDR enters WB, it has the value of R1 that is needed by the STR, which has just entered MEM. The result can be forwarded to MEM at that time without requiring the STR to stall. This requires adding a 2-to-1 mux before the memory data input port used for store instructions.

Hide Answer *You have used 2 of 2 submissions*

Show Discussion

New Post

Help

You will need to reference the LC-3 Pipeline Diagram for Homework 3 and 4.

Please click here to open in a new window.

## HOMEWORK 3. HAZARDS (1/1 point)

Please reference the LC-3 Pipeline Diagram when completing this Homework.

In the non-pipelined LC-3 implementation, the Condition Code (CC) register was updated at the same time as the register file. In the pipelined case, the CC register is updated by an operate instruction at the end of EX. Why is this the case?

- To permit CC values to be forwarded to subsequent operate instructions.
- To permit CC values to be forwarded to subsequent memory instructions.
- [✓ To permit a BR instruction that immediately follows an operate instruction to update the PC in EX if the condition is met.] ✓
- To permit an LDR instruction to update the CC in EX.

**EXPLANATION**

The first two statements are incorrect. CC values are not used by operate or memory instructions.

The third statement is the reason that the CC is updated at the end of EX. In order to reduce the number of instructions that must be discarded on a taken branch, we permit operate instructions to update the CC register at the end of EX. In this way, a BR in EX can change the PC in EX if the condition is met (branch is taken), and only the instructions in IF and ID need to be discarded by turning them into NOPs.

If we waited until the end of WB to update the CC, then the BR in EX and the instructions in ID and IF would have to stall for two clock cycles, which would cause our program to run longer.

The last statement is not true, since an LDR instruction cannot update the CC until the end of MEM.

Final Check | **Save** | Hide Answer | *You have used 1 of 2 submissions*

Show Discussion

New Post

## HOMEWORK 4. HAZARDS (1/1 point)

Please reference the LC-3 Pipeline Diagram when completing this Homework.

Consider this LC-3 code segment:

Help

```
LDR R1, R4, #0
BRz NEXT
```

Which of the following statements describes how the LC-3 pipelined implementation would handle this case? (You may assume that new hardware can be added to our existing pipeline if necessary.)

- The LDR would update the CC at the end of EX and the BR would update the PC in the next clock cycle if the condition is met.
- The LDR would update the CC at the end of ID and the BR would update the PC in the next clock cycle if the condition is met.
- [● The LDR would update the CC at the end of MEM and the BR would update the PC in the next clock cycle if the condition is met. ✓]
- The LDR would update the CC at the end of WB and the BR would update the PC in the next clock cycle if the condition is met.

**EXPLANATION**

For the pipelined LC-3 implementation, a BR instruction updates the PC in EX if the condition is met. For an LDR, the CC is not updated until the end of MEM. Thus, the BR instruction in EX and the instructions in ID and IF must stall for one cycle and a bubble forced into MEM. In the cycle after the LDR updates the CC, the new CC value is available, the PC is updated if the condition is met, and the pipeline is allowed to advance.

Final Check | **Save** | Hide Answer *You have used 1 of 2 submissions*

Show Discussion New Post

edX About edX Follow Us

Help