CSCE 692 HW Set 3 – Exercises

(Appendix C)

(100 Points)

Due: NLT 1100 Thursday, 31 January 2019

**Problems: C.1 – a** [**10**]**, b** [**20**]**, c** [**20**]**, d** [**20**]

**C.3 – a** [**7**]**, b** [**7**]**, c** [**7**]**, d** [**9**]

**Instructions:**

* Put your name at the top, and number each page (last name-pg)
* State the complete problem
* Show your work
* Clearly indicate your answer

**Notes:**

**C.1 a. Not all data dependencies result in hazards. Find ALL data dependencies.**

* **Do not list “transitive” dependencies (that cross instructions).**

**C.1 (b, c, d)**

* **Assume branch target is known in ID, but that branch outcome is not decided until the EX stage.**
* **When computing how many cycles the loop requires, it is asking for all iterations of the loop, beginning at the first instruction, and continuing until the next valid instruction following the loop is able to begin.**
* **When you have to draw a multi-cycle diagram, consider using or printing out copies of the blank template (pipeline\_matrix.xlsx) (on CANVAS, and printed below).**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | Operands | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| ld | x1, 0(x2) | IF | ID | EX | M | WB |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

C.1 [10/20/2/20/] <C.2> Using the following code fragment, answer the following questions:

Loop: ld x1,0(x2) ; load x1 from address 0+x2

addi x1,x1, 1 ; x1=x1+1

sd x1,0(x2) ; store x1 at address 0+x2

addi x2,x2, 4 ; x2=x2+4

sub x4,x3,x2 ; x4=x3-x2

bnez x4,Loop ; branch to Loop if x4!=0

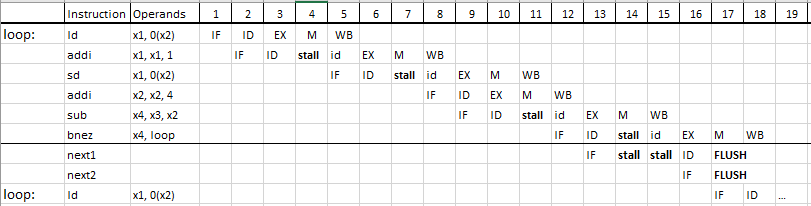
(Assume that the initial value of x3 is x2 + 396.)

99 Loops 🡪 x2 is increasing by 4 each iteration of the loop

1. [10] <C.2> Data hazards are caused by data dependences in the code. Whether a dependency causes a hazard depends on the machine implementation (i.e., number of pipeline stages). List all of the data dependences in the code above. Record the register, source instruction, and destination instruction; for example, there is a data dependency for register x1 from the ld to the addi.

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Source Instruction** | **Destination Instruction** | **Dependency Type** |
| x1 | ld | addi | WAW and RAW |
| x1 | addi | sd | WAW and WAR |
| x2 | sd | addi | WAR |
| x2 | addi | sub | RAW |
| x4 | sub | bnez | RAW |

1. [20] <C.2> Show the timing of this instruction sequence for the 5-stage RISC pipeline without any forwarding or bypassing hardware but assuming that a register read and a write in the same clock cycle “forwards” through the register file, as shown in Figure C.5. Use a pipeline timing chart like that in Figure C.8. Assume that the branch is handled by flushing the pipeline. If all memory references take 1 cycle, how many cycles does this loop take to execute?

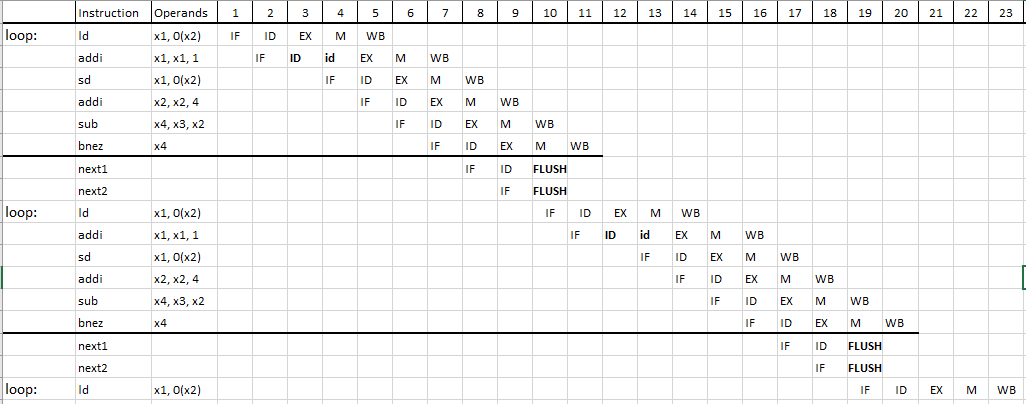


The stall in cycle 4 occurs because the processor is waiting for the result of the load, which is loaded into memory in cycle 4, and can be used in ID in cycle 5. The stall in cycle 7 occurs because x2 is being utilized for the store instruction. The stall in cycle 11 occurs because the subtract operation requires the result of the addi instruction, which is produced by the EX in cycle 10, and is put into memory in cycle 11. The stall in cycle 14 occurs as the branch instruction needs the result of the previous subtract to calculate the branch outcome. The stall in cycles 14-15 for next1 occur because the branch has not left the ID step. When the branch is taken, next1 and next2 are flushed from the pipeline, and execution continues from the start of the loop.

Each iteration through the loop increases x2 by 4 (x2 = x2 + 4). Because the loop terminates when x2 = x3, and x3 initially takes the value x3 = x2 + 396; this requires 99 iterations of the loop.

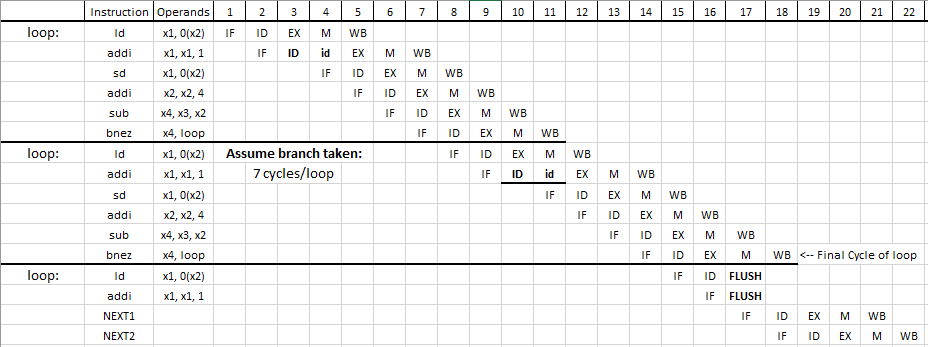
Each loop takes a total of 16 cycles: this is the amount of cycles from the IF of the first instruction of the loop until the load operation for the subsequent iteration.

The final iteration of the loop has 12 cycles because the branch outcome is not to branch, but instead to go to the next operation. Thus, useful work was done while waiting for the branch resolution.

c. [20] <C.2> Show the timing of this instruction sequence for the 5-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.8. Assume that the branch is handled by predicting it as not taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

**Stalls:** there is only one stall in each loop, it occurs in cycle 3 on the first add immediate because the instruction must wait for the load operation to load the value into x1. Thus, ID can occur when the load is in the MEM (M) phase of the pipeline.

Because the branch prediction is **not taken**, two instructions – next1 and next2 begin their operation, but are flushed from the pipeline if the branch is taken, as shown in cycle 11. This is wasted work during the loop’s iterations, but would be useful at the termination of the loop.

d. [20] <C.2> Show the timing of this instruction sequence for the 5-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.8. Assume that the branch is handled by predicting it as taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

**Stalls:** there is one stall in each loop, it occurs in cycle 3 on the first add immediate because the instruction must wait for the load operation to load the value into x1. Thus, ID can occur when the load is in the MEM (M) phase of the pipeline.

Because the branch prediction is **taken**, each time the branch is taken, the cycles used while waiting for the branch resolution produce useful work. This provides a pipelined 7 cycles/loop during the loop’s iterations. On the final loop iteration (shown above ending at cycle 18, but in practice would be much later), where the branch will not be taken, there are 9 cycles to finish the loop.

Thus, the total cycles can be calculated using the same equation from part C:

C.3 [7/7/7/9] <C.2> We begin with a computer implemented in single-cycle implementation. When the stages are split by functionality, the stages do not require exactly the same amount of time. The original machine had a clock cycle time of 7 ns. After the stages were split, the measured times were IF, 1 ns; ID, 1.5 ns; EX, 1 ns; MEM, 2 ns; and WB, 1.5 ns. The pipeline register delay is 0.1 ns.

a. [7] What is the clock cycle time of the 5-stage pipelined machine?

Clock cycle time = longest stage + register delay = 2 + .1 = 2.1 ns

b. [7] If there is a stall every 4 instructions, what is the CPI of the new machine?

CPI = 1 + % Stalls \* stall penalty = 1 + .25 \* 1 = 1.25

c. [7] What is the speedup of the pipelined machine over the single- cycle machine?

The following calculations assume a pipelined machine with a stall every 4 instructions:

d. [9] If the pipelined machine had an infinite number of stages, what would its speedup be over the single-cycle machine?

The CPI stays at 1.25, regardless of the pipeline depth because there is still 1 stall every 4 instructions. The cycle time becomes 0.1 ns, the amount of time required to access the pipeline registers.