CSCE 692 – Chapter 3 Homework Problems

2nd Lt David Crow – 18 February 2019

(100 Points)

Due: NLT 1100 Tuesday, 26 February 2019

**Instructions:**

* Number each page (last name-pg)
* Show your work and clearly indicate your answer

**Book Problems [20 points each]: 3.1, 3.2, 3.3 (Changed from 5th Edition)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Loop: | LD | F2,0(Rx) |  | Latencies beyond single cycle |  |
| I0: | MULTD | F2,F6,F2 |  | Memory LD | +3 |
| I1: | DIVD | F8,F2,F0 |  | Memory SD | +1 |
| I2: | LD | F4,0(Ry) |  | Integer ADD, SUB | +0 |
| I3: | ADDD | F4,F0,F4 |  | Branches | +1 |
| I4: | ADDD | F10,F8,F2 |  | ADDD | +2 |
| I5: | SD | F4,0(Ry) |  | MULTD | +4 |
| I6: | ADDI | Rx,Rx,#8 |  | DIVD | +10 |
| I7: | ADDI | Ry,Ry,#8 |  |  |  |
| I8: | SUB | R20,R4,Rx |  |  |  |
| I9: | BNZ | R20,Loop |  |  |  |
| **Figure 3.47** Code and latencies for Exercises 3.1 through 3.3 | | | | |  |

3.1 [20] <3.1, 3.2> What is the baseline performance (in cycles, per loop iteration) of the code sequence in Figure 3.47 if no new instruction's execution could be initiated until the previous instruction's execution had completed? Ignore front-end fetch and decode. Assume for now that execution does not stall for lack of the next instruction, but only one instruction/cycle can be issued. Assume the branch is taken, and that there is a one-cycle branch delay slot.

Here are the cycles required for each instruction:

|  |  |  |
| --- | --- | --- |
| Instruction | Cycles | Running Total |
| LD |  |  |
| I0 |  |  |
| I1 |  |  |
| I2 |  |  |
| I3 |  |  |
| I4 |  |  |
| I5 |  |  |
| I6 |  |  |
| I7 |  |  |
| I8 |  |  |
| I9 |  |  |

We see, then, that each iteration of the loop requires .

3.2 [20] <3.1, 3.2> Think about what latency numbers really mean – they indicate the number of cycles a given function requires to produce its output, nothing more. If the overall pipeline stalls for the latency cycles of each functional unit, then you are at least guaranteed that any pair of back-to-back instructions (a "producer" followed by a "consumer") will execute correctly. But not all instruction pairs have a producer/ consumer relationship. Sometimes two adjacent instructions have nothing to do with each other. How many cycles would the loop body in the code sequence in Figure 3.48 require if the pipeline detected true data dependences and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? Show the code with <Stall > inserted where necessary to accommodate stated latencies. (Hint: An instruction with latency +2 requires two <Stall> cycles to be inserted into the code sequence. Think of it this way: A one-cycle instruction has latency 1 + 0, meaning zero extra wait states. So, latency 1 + 1 implies one stall cycle; latency 1 + N has N extra stall cycles.

We have true data dependences

* + on F2 between Loop and I0,
  + on F2 between I0 and I1,
  + on F4 between I2 and I3, and
  + on F8 between I1 and I4.

Under these dependences, our code is now the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Cycle | Instruction | Operands | Last Cycle |
| 1 | LD | F2,0(Rx) |  |
| 2 | <Stall> | |  |
| 3 | <Stall> | |  |
| 4 | <Stall> | |  |
| 5 | MULTD | F2,F6,F2 |  |
| 6 | <Stall> | |  |
| 7 | <Stall> | |  |
| 8 | <Stall> | |  |
| 9 | <Stall> | |  |
| 10 | DIVD | F8,F2,F0 |  |
| 11 | LD | F4,0(Ry) |  |
| 12 | <Stall> | |  |
| 13 | <Stall> | |  |
| 14 | <Stall> | |  |
| 15 | ADDD | F4,F0,F4 |  |
| 16 | <Stall> | |  |
| 17 | <Stall> | |  |
| 18 | <Stall> | |  |
| 19 | <Stall> | |  |
| 20 | <Stall> | |  |
| 21 | ADDD | F10,F8,F2 |  |
| 22 | SD | F4,0(Ry) |  |
| 23 | ADDI | Rx,Rx,#8 |  |
| 24 | ADDI | Ry,Ry,#8 |  |
| 25 | SUB | R20,R4,Rx |  |
| 26 | BNZ | R20,Loop |  |
| 27 | <BNZ latency> | |  |

We see here that this new pipeline requires to execute our loop.

3.3 [20] <3.7> Consider a multiple-issue design. Suppose you have two execution pipelines, each capable of beginning execution of one instruction per cycle, and enough fetch/decode bandwidth in the front end so that it will not stall your execution. Assume results can be immediately forwarded from one execution unit to another, or to itself. Further assume that the only reason an execution pipeline would stall is to observe a true data dependency. Now how many cycles does the loop require?

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cycle | Execution Pipeline 1 | | | Execution Pipeline 2 | | |
| Instruction | Operands | Last Cycle | Instruction | Operands | Last Cycle |
| 1 | LD | F2,0(Rx) |  |  |  |  |
| 2 | <Stall> | |  |  |  |  |
| 3 | <Stall> | |  |  |  |  |
| 4 | <Stall> | |  |  |  |  |
| 5 | MULTD | F2,F6,F2 |  |  |  |  |
| 6 | <Stall> | |  |  |  |  |
| 7 | <Stall> | |  |  |  |  |
| 8 | <Stall> | |  |  |  |  |
| 9 | <Stall> | |  |  |  |  |
| 10 | DIVD | F8,F2,F0 |  | LD | F4,0(Ry) |  |
| 11 | <Stall> | |  | <Stall> | |  |
| 12 | <Stall> | |  | <Stall> | |  |
| 13 | <Stall> | |  | <Stall> | |  |
| 14 | <Stall> | |  | ADDD | F4,F0,F4 |  |
| 15 | <Stall> | |  |  |  |  |
| 16 | <Stall> | |  |  |  |  |
| 17 | <Stall> | |  |  |  |  |
| 18 | <Stall> | |  |  |  |  |
| 19 | <Stall> | |  |  |  |  |
| 20 | <Stall> | |  |  |  |  |
| 21 | ADDD | F10,F8,F2 |  | SD | F4,0(Ry) |  |
| 22 | ADDI | Rx,Rx,#8 |  | ADDI | Ry,Ry,#8 |  |
| 23 | SUB | R20,R4,Rx |  |  |  |  |
| 24 | BNZ | R20,Loop |  |  |  |  |
| 25 | <BNZ latency> | |  |  |  |  |

Clearly, this loop requires . Unfortunately, the true data dependences that exist in the loop ensure that our second pipeline doesn’t benefit us much.

**Additional Problems [20 points each]: 1, 2**

Problem 1 [20] Here is an unusual loop. List all the dependences (output, anti, and true) in the following code fragment. Indicate whether the true dependences are loop-carried or not. Finally, rewrite the loop so that it can be executed in parallel (i.e., try to remove the loop dependences).

for (i=1; i<100; i=i+1) {

a[i] = b[i] + c[i]; /\* S1 \*/   
 b[i] = a[i] + d[i]; /\* S2 \*/

a[i+1] = a[i] + e[i]; /\* S3 \*/

/\* in the next iteration, relative to the current i value

a[i+1] = b[i+1] + c[i+1];

b[i+1] = a[i+1] + d[i+1];

a[i+2] = a[i+1] + e[i+1];

\*/

}

Although the problem instructs us to indicate only whether the *true* dependences are loop-carried or not, I am going to indicate whether *all* dependences are loop-carried or not.

We have true dependences

* + (not loop-carried) on a[i] between S1 and S2,
  + (not loop-carried) on a[i] between S1 and S3,

We have anti-dependences

* (not loop-carried) on b[i] between S1 and S2.

We have output dependences

* (loop-carried) on a[i+1] between S3 and S1.

Because our only loop-carried dependence is the WAW between S3 and S1, we know that a[i+1] is immediately overwritten in the next iteration. This means we can move S3 outside of the loop in the following way:

for (i=1; i<100; i=i+1) {

a[i] = b[i] + c[i]; /\* S1 \*/   
 b[i] = a[i] + d[i]; /\* S2 \*/

}

a[100] = a[99] + e[99]; /\* S3 \*/

Problem 2 [20] Using the code sequence shown below, show the complete schedule and bookkeeping in the reservation stations/register result status using the Tomasulo approach (print out as many copies of “TomasuloTemplateHW3.pdf” as you need). Show the schedule (i.e., fill in) from the form provided for each clock cycle until complete.

L.D F1, 0(R1)

MUL.D F7, F1, F5

ADD.D F1, F1, F5

ADD.D F2, F1, F7

DADDI R1, R2, 8

S.D F2, 0(R1)

Assume the following clock cycle required for each “exec” functional unit:

Loads/Stores 2 clock cycles

FP Mult 7 clock cycles

FP Add 2 clock cycles

Any Integer op 1 clock cycles

Note the DADDI (integer adder) is not normally shown in any Tomasulo unit or bookkeeping. For the purposes of this exercise we include it in our bookkeeping as far as the instruction status but do not worry about placing it in a reservation station. We also assume we have a separate CDB for integer and floating-point registers.

Okay.