Hunter Morera
U79718512
HW 4
10/24/18

3.1 Find cycles per loop iteration of the code snippet given. Also given is the cycles per instruction which are summarized below:

Mem LD = 3 cycles
Mem SD = 1 cycle
Int Add,Sub = 0 cycles
Branches = 1 cycle
Fadd.d = 2 cycles
Fmul.d = 4 cycles
Fdiv.d = 10 cycles

Now we will calculate the cycles per instruction of the code snippet line by line in order to calculate the total cycles per loop iteration:

Loop: Fld = 1 + 3 = 4 cycles
I0: Fmul.d = 1 + 4 = 5 cycles
I1: Fdiv.d = 1 + 10 = 11 cycles
I2: Fld = 1 + 3 = 4 cycles
I3: Fadd.d = 1 + 2 = 3 cycles
I4: Fadd.d = 1 + 2 = 3 cycles
I5: fsd = 1 + 1 = 2 cycles
I6: addi = 0 + 1 = 1 cycles
I7: addi = 0 + 1 = 1 cycles
I8: sub = 0 + 1= 1 cycles
I9: bnz = 1 + 1 = 2 cycles
Total cycles: **37 cycles**

3.2 How many cycles would the loop body in the code snippet 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. We will now re-write the code with the added stalls and new cycle number. As well I will use coloring to indicate the dependencies:

Loop: Fld: f2,0(Rx)
<stall>
<stall>
<stall>
I0: Fmul.d: f2,f0,f2
<stall>
<stall>
<stall>
<stall>
I1: Fdiv.d: f8,f2,f0

Hunter Morera
U79718512
HW 4
10/24/18

I2: Fld: f4,0(Ry)
<stall due to load>
<stall due to load>
<stall due to load>
I3: Fadd.d: f4,f0,f4
<stall due to div>
<stall due to div>
<stall due to div>
<stall due to div>
<stall due to div>
I4: Fadd.d: f10,f8,f2
I5: fsd: f4,0(Ry)
I6: addi: Rx,Rx,8
I7: addi: Ry,Ry,8
I8: sub: x20,x4,Rx
I9: bnz: x20,Loop
<stall due to bnz>
Total Cycles = **27 cycles**

3.5 Reorder the instructions to improve performance of the code, assuming that pipeline is the same as in 3.1 & 3.2, Just worry about observing true data dependencies and functional unit latencies for now. How many cycles does your reordered code take? In my opinion the easiest way to re-order the code is to take the instructions with no data dependencies and execute those during what is the stall cycles in 3.2 of Fld, seems the easiest and produces an optimized solutions, the question doesn't ask to re-order for least possible cycles, just re-order to reduce cycles, see below:

Loop: Fld: f2,0(Rx)
I6: addi: Rx,Rx,8
<stall>
<stall>
I0: Fmul.d: f2,f0,f2
I7: addi: Ry,Ry,8
<stall>
<stall>
<stall>
I1: Fdiv.d: f8,f2,f0
I2: Fld: f4,0(Ry)
I8: sub: x20,x4,Rx
<stall due to load>
<stall due to load>
I3: Fadd.d: f4,f0,f4
<stall due to div>

Hunter Morera
U79718512
HW 4
10/24/18

<stall due to div>
<stall due to div>
<stall due to div>
I4: Fadd.d: f10,f8,f2
I5: fsd: f4,0(Ry)
I9: bnz: x20,Loop
<stall due to bnz>
Total Cycles: **23 Cycles**

3.6

B. Hand-unroll two iterations of the loop in your reordered code from Exercise 3.5. Assume for simplicity that we can use the same registers between iterations without any issues. Second loop code will be highlighted:

Loop: Fld: f2,0(Rx)
Fld: f2,0(Rx)
I6: addi: Rx,Rx,8
I6: addi: Rx,Rx,8
<stall>
I0: Fmul.d: f2,f0,f2
I0: Fmul.d: f2,f0,f2
I7: addi: Ry,Ry,16
<stall>
I1: Fdiv.d: f8,f2,f0
I1: Fdiv.d: f8,f2,f0
I2: Fld: f4,0(Ry)
I2: Fld: f4,0(Ry)
I8: sub: x20,x4,Rx
I8: sub: x20,x4,Rx
I3: Fadd.d: f4,f0,f4
I3: Fadd.d: f4,f0,f4
<stall due to div>
<stall due to div>
I4: Fadd.d: f10,f8,f2
I4: Fadd.d: f10,f8,f2
I5: fsd: f4,0(Ry)
I5: fsd: f4,0(Ry)
I9: bnz: x20,Loop
<stall due to bnz>
Total Cycles: **25 Cycles**

Hunter Morera
U79718512
HW 4
10/24/18

C. In my solution above I have chosen to use yellow highlighting to indicate my second iterations instructions as it is easier to read than the green suggested in the textbook. I can now calculate my speedup that I obtained for my unrolling of the loop:

Original Implementation: 23 total cycles/iteration

Unrolled Implementation: 25 total cycles for 2 iterations = 25/2 = 12.5 total cycles/iteration

Therefore our speedup on this problem would be 23/12.5 = **1.84 speedup** by unrolling the loop in part B.

3.11 Assume a five-stage single-pipeline microarchitecture (fetch, decode, execute, memory, write-back) and the code in Figure 3.53. All ops are one cycle except LW and SW, which are 1+2 cycles, and branches, which are 1 +1 cycles. There is no forwarding. Show the phases of each instruction per clock cycle for one iteration of the loop. To solve this let's first assume that the Decode stage does not access any registers and thus Decode can be to evaluate if there are dependencies. Also note the following F- Fetch, D- Decode, E- Execute, M- Memory, W- Write Back, and S- Stall, the pipeline diagram would be as follows:

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
lw x1,0(x2)	F	D	E	M	M	M	W																	
addi x1,x1, 1		F	D	S	S	S	S	E	M	W														
sw x1,0(x2)			F	S	S	S	S	D	S	S	E	M	M	M	W									
addi x2,x2,4								F	S	S	D	E	S	S	M	W								
sub x4,x3,x2											F	D	S	S	S	S	E	M	W					
bnz x4,Loop												F	S	S	S	S	D	S	S	E	E	M	W	
lw x1,0(x2)																						F	D	..

A. How many clock cycles per loop iteration are lost to branch overhead?

If the next iteration of the loop could occur immediately after the sub instruction, it could begin on cycle 12, however with addition of the branch instruction we can see that it is not able to begin until cycle 21, therefore we lost 22-12 = **10 cycles** per iteration to branch overhead.

Hunter Morera
U79718512
HW 4
10/24/18

B. Assume a static branch predictor, capable of recognizing a backward branch in the Decode stage. Now how many clock cycles are wasted on branch overhead?

If the next iteration of the loop could occur immediately after the sub instruction, it could begin on cycle 12, however with the static branch predictor we could begin the fetch cycles of the next iteration right as we are decoding the branch instruction in cycle 17, therefore we lost 17-12 = **5 cycles** per iteration to branch overhead.

C. Assume a dynamic branch predictor. How many cycles are lost on a correct prediction?

When we use a dynamic branch prediction it has the ability to change as the program changes. The dynamic predictor will predict branch taken if it was last time, or not taken if it was not last time. This means the decision will occur during the same cycle as the instruction is fetched in. This means that **no cycles** are lost to branch prediction using a this type of structure with a dynamic branch predictor.

Problem 2

The following series of branch outcomes occurs for a single branch in a program. (T means the branch is taken, N means the branch is not taken).

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
outcome	T	T	N	T	N	T	T	T	T	N	T	T	N

Please see the answer to the questions regarding this table below, they have been put on a new page for continuity:

Hunter Morera
U79718512
HW 4
10/24/18

A. Assume that we are trying to predict this sequence with a Branch History Table (BHT) using a 1-bit prediction. The counters of the BHT are initialized to the N state. Which of the branches would be mispredicted? Show their indices. Assume that N = 0 and T = 1

Index	Beginning Value	Ending Value	Outcome
1	0	1	Miss
2	1	1	Hit
3	1	0	Miss
4	0	1	Miss
5	1	0	Miss
6	0	1	Miss
7	1	1	Hit
8	1	1	Hit
9	1	1	Hit
10	1	0	Miss
11	0	1	Miss
12	1	1	Hit
13	1	0	Miss

As we can see from the table above we have a total of 8 misses at index (1,3,4,5,6,10,11,13)

Part B is on the next page for continuity of the table.

Hunter Morera
U79718512
HW 4
10/24/18

B. Repeat the above exercise with a 2-bit predictor as shown in Figure C.15 initialized to 10.

Index	Beginning Value	Ending Value	Outcome
1	10	11	Hit
2	11	11	Hit
3	11	10	Miss
4	10	11	Hit
5	11	10	Miss
6	10	11	Hit
7	11	11	Hit
8	11	11	Hit
9	11	11	Hit
10	11	10	Miss
11	10	11	Hit
12	11	11	Hit
13	11	10	Miss

As we can see from the table above we have a total of 4 misses at index (3,5,10,13)