Computer Architecture Homework Assignment #1

Grading Policy

TA in charge: Myeongjae Jang
E-mail: myeongjae0409@kaist.ac.kr
Office: N1 - #922

I. Solution and Grading Policy

Q1.

A. There are six data dependencies as below. Various ways to write data dependencies are allowed, but there have to be 3 information like the example in the question. Below solutions show only these 3 information for each data dependencies.

#1. R1	LD	DADDI	(Example)
#2. R1	DADDI	SD	
#3. R2	LD	DADDI	
#4. R2	SD	DADDI	
#5. R2	DADDI	DSUB	
#6. R4	DSUB	BNEZ	

For #3 and #4, they are not *true data dependencies*. In this question, there are not any detailed explanation about data dependency. Therefore, only four *true data dependencies* also can be an answer without #3 and #4. However, if you wrote one of #3 and #4, it is considered that you tried to find all of data dependencies and missing one can be dealt as fault.
With the number of missing or incorrect data dependencies, you will get points as below.

There are not any missing or incorrect data dependencies. **(8 points)**
There are 1~2 missing or incorrect data dependencies. **(6 points)**
There are 3~4 missing or incorrect data dependencies. **(4 points)**
There are 5 missing or incorrect data dependencies. **(2 points)**
There are 6 or more missing or incorrect data dependencies. **(0 point)**

B. In *Figure C.6*, DADD and OR show “forwarding” through the register file. It means that true data dependency can be solved on WB stage. Therefore, the pipeline timing chart can be implemented as below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD R1, 0(R2)	F	D	X	M	W													
DADDI R1, R1, #1		F	s	s	D	X	M	W										
SD 0(R2), R1					F	s	s	D	X	M	W							
DADDI R2, R2, #4								F	D	X	M	W						
DSUB R4, R3, R2									F	s	s	D	X	M	W			
BNEZ R4, Loop												F	s	s	D	X	M	W
LD R1, 0(R2)																	F	D

(F = IF, D = ID, X = EX, M = MEM, W = WB, and s = Stall)

The result of the conditional branch, BNEZ can be known after EX stage. For the iteration, it takes 16 cycles. For the last iteration, 2 cycles are needed to finish the last instruction without pipeline overlap.
Since the initial value of R3 is R2 + 396, there are 99 iterations. Finally, total cycles are calculated as follows.

(99 iterations) * (16 cycles) + (2 cycles for the last instruction)
= 1586 cycles.

Lecture note #5 says that branches are resolved in MEM stage. So, it is also able to be another answer. With branches resolved in MEM stage, each iteration takes 17 cycles, and the last instruction needs 1 cycle without pipeline overlap. Total cycles are calculated as follows.

(99 iterations) * (17 cycles) + (1 cycle for the last instruction)
= 1684 cycles.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
LD R1, 0(R2)	F	D	X	M	W													
DADDI R1, R1, #1		F	s	s	D	X	M	W										
SD 0(R2), R1					F	s	s	D	X	M	W							
DADDI R2, R2, #4								F	D	X	M	W						
DSUB R4, R3, R2									F	s	s	D	X	M	W			
BNEZ R4, Loop												F	s	s	D	X	M	W
LD R1, 0(R2)																		F

At the last, the question announces forwarding or bypassing hardware only. It assumes that the branch is handled by flushing the pipeline, but “flushing” is ambiguous and can be interpreted as several meanings. Therefore, it can be another answer if you resolve branch target in ID stage. Then, each iteration takes 15 cycles, and the last instruction needs 3 cycles without pipeline overlap. Total cycles are calculated as follows.

(99 iterations) * (15 cycles) + (3 cycles for the last instruction)
= 1488 cycles.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`LD R1, 0(R2)`	F	D	X	M	W													
`DADDI R1, R1, #1`		F	s	s	D	X	M	W										
`SD 0(R2), R1`					F	s	s	D	X	M	W							
`DADDI R2, R2, #4`								F	D	X	M	W						
`DSUB R4, R3, R2`									F	s	s	D	X	M	W			
`BNEZ R4, Loop`												F	s	s	D	X	M	W
`LD R1, 0(R2)`																F	D	X

Scores are given as follows. **(8 points total)**

Pipeline timing chart does not have any faults. **(+3 points)**
Each iteration needs (16, 17, 15) cycles. **(+2 points)**
The last instruction needs (2, 1, 3) more cycles. **(+2 points)**
The total cycles are (1586, 1684, 1488). **(+1 point)**

C. Because full forwarding and bypassing hardware is supported, true data dependencies can be dealt with forwarding. Therefore, there are not any stall cycles except for LD stall cycle.
In this question, it uses branch prediction. It means that branch computation is done in ID stage. Register R4 which is used for branch computation has true data dependency between DSUB and BNEZ. Because branch computation is done in ID stage, it needs 1 stall cycle to wait R4 from DSUB. It predicts branch as "not taken", but it continuously makes misprediction during 99 loop iterations. It causes 1 stall cycle.
As the result, you can draw the pipeline timing chart as below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`LD R1, 0(R2)`	F	D	X	M	W													
`DADDI R1, R1, #1`		F	D	s	X	M	W											
`SD R1, 0(R2)`			F	s	D	X	M	W										
`DADDI R2, R2, #4`					F	D	X	M	W									
`DSUB R4, R3, R2`						F	D	X	M	W								
`BNEZ R4, Loop`							F	s	D	X	M	W						
(incorrect instruction)									F	s	s	s	s					
`LD R1, 0(R2)`										F	D	X	M	W				

Each iteration needs 9 cycles, and the last instruction needs 3 more cycles without pipeline overlap. Total cycles are computed as follows.

(99 iterations) * (9 cycles) + (3 cycles for the last instruction)
= 894 cycles.

Scores are given as follows. **(10 points total)**

Pipeline timing chart does not have any faults. **(+3 points)**
Each iteration needs 9 cycles. **(+3 points)**
The last instruction needs 3 more cycles. **(+3 points)**
The total cycles are 894. **(+1 point)**

D. For computation except branch prediction, it is same with the question C above. In this time, it predicts branch as taken. Therefore, it makes correct prediction during loop iteration, and there are not any additional stall cycles caused by misprediction.
The pipeline timing chart is as below.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`LD R1, 0(R2)`	F	D	X	M	W													
`DADDI R1, R1, #1`		F	D	s	X	M	W											
`SD R1, 0(R2)`			F	s	D	X	M	W										
`DADDI R2, R2, #4`					F	D	X	M	W									
`DSUB R4, R3, R2`						F	D	X	M	W								
`BNEZ R4, Loop`							F	s	D	X	M	W						
`LD R1, 0(R2)`									F	D	X	M	W					

Each iteration needs 8 cycles, and the last instruction needs 4 more cycles without pipeline overlap. Total cycles are computed as follows.

(99 iterations) * (8 cycles) + (4 cycles for the last instruction)
= 796 cycles.

Scores are given as follows. **(10 points total)**

Pipeline timing chart does not have any faults. **(+3 points)**
Each iteration needs 8 cycles. **(+3 points)**
The last instruction needs 4 more cycles. **(+3 points)**
The total cycles are 796. **(+1 point)**

Q2.

A. To know how much faster, you should calculate speedup from pipelining. You can find the explanation and formula of speedup from *Lecture note #5* or the text book Appendix page C-12 and C-13. The formula of speedup from pipelining is as follows.

(Speedup from pipelining)
= (Pipeline depth) / {1 + (Pipeline stall cycles per instruction)}.

In this question, it uses four-deep pipeline. Therefore, "Pipeline depth" is 4. Next, you should compute "Pipeline stall cycles per instruction". As the ideal case, there are not any stall cycles per instruction.

(Speedup from pipelining - Ideal)
= 4 / (1 + 0)
= 4.

With branch hazards, there are some stall cycles per instruction. Since the question ignores other pipeline stalls, you should focus on branch hazards only. At the first, unconditional branches, "Jump and calls", resolve a branch target at the end of the second cycle. Even though the first pipe stage can always be done independently, it always causes a stall since unconditional branches are always taken, and its target address is always changed. As the result, every unconditional branches make 1 stall cycle.
Secondly, conditional branches can make stalls with branch hazards. Conditional branches can resolve a branch target at the end of the third cycle. Conditional branches can be divided into two cases: "Taken" and "Not-taken". For "Not-taken" case, independently proceed instruction at the first pipe stage will be used continuously. However, it depends on the conditional branches from the second pipe stage, so it should be wait until conditional branches resolve the branch target after the first pipe stage. It makes 1 stall cycle. For "Taken" case, the branch target is changed and the first pipe stage should be repeated again like unconditional branches. Since conditional branches take one more stage to resolve than unconditional branches, they make 2 stall cycles.
Finally, we can compute "Pipeline stall cycles per instruction". Followed by the branch frequencies on the question, you can compute as follows.

(Pipeline stall cycles per instruction)
= (Jump and calls) * (1 stall cycle) + (Conditional branches) * {(Taken) * (2 stall cycles) + (Not-taken) * (1 stall cycle)}
= 0.01 * 1 + 0.15 * {0.6 * 2 + (1 - 0.6) * 1}
= 0.25.

Therefore, speedup as the branch hazard case can be computed as follows.

(Speedup from pipelining - Branch hazard)
= 4 / (1 + 0.25)
= 3.2.

As the result, you can compare two cases and know how much faster.

(Speedup - Ideal) / (Speedup - Branch hazard)
= 4 / 3.2
= 1.25 times faster without any branch hazards.

Scores are given as follows. If you use another similar performance parameters (e.g. clock cycle, CPI, etc.), you can get points. However, there are not enough explanations, you will get minus points based on the grading policy. **(15 points total)**

Pipeline depth is 4. **(+1 points)**
Speedup on ideal case is 4. **(+1 point)**
Unconditional branches make 1 stall cycle. **(+3 points)**
Taken conditional branches make 2 stall cycles. **(+3 points)**
Not-taken conditional branches make 1 stall cycle. **(+3 points)**
Pipeline stall cycles per instruction is 0.25. **(+2 point)**
Speedup with branch hazards is 3.2. **(+1 points)**
The machine can be 1.25 times faster. **(+1 point)**

B. You can solve it in a similar way as in the question A. Differences are 15-deep pipeline, unconditional branches are resolved at the end of the fifth cycle, and conditional branches are resolved at the end of the tenth cycle. Followings are formulas to compute.

(Speedup from pipelining - Ideal)
= 15 / (1 + 0)
= 15.

(Pipeline stall cycles per instruction)
= (Jump and calls) * (4 stall cycle) + (Conditional branches) * {(Taken) * (9 stall cycles) + (Not-taken) * (8 stall cycle)}
= 0.01 * 4 + 0.15 * {0.6 * 9 + (1 - 0.6) * 8}
= 1.33.

(Speedup from pipelining - Branch hazard)
= 15 / (1 + 1.33)
= 6.44.

(Speedup - Ideal) / (Speedup - Branch hazard)
= 15 / 6.44
= 2.33 times faster without any branch hazards.

Scores are given as follows. **(15 points total)**

Pipeline depth is 15. **(+1 points)**
Speedup on ideal case is 15. **(+1 point)**
Unconditional branches make 4 stall cycle. **(+3 points)**
Taken conditional branches make 9 stall cycles. **(+3 points)**
Not-taken conditional branches make 8 stall cycle. **(+3 points)**
Pipeline stall cycles per instruction is 1.33. **(+2 point)**
Speedup with branch hazards is 6.44. **(+1 points)**
The machine can be 2.33 times faster. **(+1 point)**

Q3.

A. With the pipelined machine, the clock cycle time is set by the longest stage. In the question, the longest stage is MEM, and it is measured as 2 ns. Moreover, the machine has the pipeline register between neighboring two stages. Its delay is 0.1 ns. So, it is also considered for the clock cycle time.

(Clock cycle time)
= (Longest stage time) + (Pipeline register delay)
= 2 ns + 0.1 ns
= 2.1 ns.

Scores are given as follows. **(7 points total)**

Longest stage is MEM, and it takes 2 ns. **(+4 points)**
Pipeline register delay is 0.1 ns. **(+1 point)**
The clock cycle time is 2.1 ns. **(+2 points)**

B. As the ideal pipeline, one instruction is executed in one cycle. Therefore, the ideal pipelined CPI is 1.
In this question, it takes one stall cycle for every 4 instructions. It means 5 cycles are needed for every 4 instructions.
As the result, the CPI is 5 / 4 = 1.25.

Scores are given as follows. **(7 points total)**

The ideal CPI is 1. **(+2 points)**
4 instructions are executed in 5 cycles based on the ideal CPI. **(+4 point)**
The result CPI is 1.25. **(+1 points)**

C. In this question, you cannot know pipeline stall cycles per instruction because there are not any information about instruction. You should compute speedup based on the CPU time. You can find some information about CPU time from *Lecture note #2*.

(CPU time)
= (Instruction count) * (CPI) * (Clock cycle time).

Since there are not any mentions about instructions, "Instruction count" will be expressed as IC.
First of all, the CPU time of the single-cycle machine is computed as follows.

(CPU time - Single)
= IC * 1 * 7 = 7 * IC.

Next, the CPU time of the new pipelined machine is computed as follows. The CPI and clock cycle time of the new machine were computed in the question A and B.

(CPU time - New)
= IC * 1.25 * 2.1 = 2.625 * IC.

As the result, the speedup of the new machine is

(Speedup)
= (CPU time - Single) / (CPU time - New)
= (7 * IC) / (2.625 * IC)
= 2.67.

Therefore, the new machine is 2.67 times faster than the single-cycle machine. There are not any detailed information which the new machine is based on the question B. If you did not consider a stall for every 4 instructions, followings can be the answer.

(CPU time - New)
= IC * 1 * 2.1 = 2.1 * IC.

(Speedup)
= (CPU time - Single) / (CPU time - New)
= (7 * IC) / (2.1 * IC)
= 3.33.

Therefore, the new machine is 3.33 times faster than the single-cycle machine. Scores are given as follows. IC can be omitted. **(10 points total)**

CPU time of the single-cycle machine is 7 * IC. **(+3 points)**
CPU time of the new pipelined machine is (2.625, 2.1) * IC. **(+3 point)**
The speedup is (2.67, 3.33). **(+4 points)**

D. Originally, one instruction is executed in 7 ns in the single-cycle machine. If the machine is changed as an infinite number of stages, it should execute the infinite number of stages in 7 ns. It means that the execution time for each stage converges to 0 ns. There is only pipeline register delay, 0.1 ns. As the result, the clock cycle time of the new machine is 0.1 ns.

For the CPI, there can be a several possible solutions.
1) If you focus on the cycle time, each cycle is executed in 0 ns as calculated above. Then, you can ignore a stall cycle for every 4 instructions. As the result, CPI can be 1.
2) If you focus on the mathematical meaning of the ideal CPI, it has several possibilities. Let's assume a 5-stage pipeline machine. The ideal CPI can be calculated as below.

$$\lim_{n\to\infty} \frac{(n\ \text{instructions}) + (4\ \text{stages without pipeline overlap})}{(n\ \text{instructions})} = 1$$

You can add a stall cycle for every 4 instructions.

$$\lim_{n\to\infty} \frac{(n\ \text{instructions}) + (4\ \text{stages for init.}) + \left(\frac{n}{4}\ \text{stalls}\right)}{(n\ \text{instructions})}$$
$$= \lim_{n\to\infty} \frac{\frac{5}{4}n}{n} = \frac{5}{4}$$

However, if you focus on "the infinite number of stages", it can be different.

$$\lim_{n,k\to\infty} \frac{(n\ \text{instructions}) + (k\ \text{stages for init.}) + \left(\frac{n}{4}\ \text{stalls}\right)}{(n\ \text{instructions})}$$

In that time, you cannot compute the CPI because you do not know which one diverges faster, *n* or *k*. If *n* diverges faster than *k*, the CPI can be still 1.25. But if *k* diverges faster than *n*, the CPI can be infinity.

Remaining solutions are about cases when IPC is 1 or 1.25.
Finally, you can compute speedup like the question C.

(CPU time - Inf)
= IC * 1 * 0.1 = 0.1 * IC.

(CPU time - Inf)
= IC * 1.25 * 0.1 = 0.125 * IC.

(Speedup)
= (CPU time - Single) / (CPU time - Inf)
= (7 * IC) / (0.1 * IC)
= 70.
(Speedup)
= (CPU time -Single) / (CPU time - Inf)
= (7 * IC) / (0.125 * IC)
= 56.

Scores are given as follows. Because of ambiguity of the question as above, there can be several possible answers. Therefore, if you give a possible answer and explain appropriately, you will get points. **(10 points total)**

The clock cycle of the infinite stage machine is 0.1 ns. **(+3 points)**
The CPI of the infinite stage machine is (1, 1.25, inf.). **(+3 point)**
The speedup is (70, 56, 0 or cannot compute). **(+4 points)**