Name: Prashant Serai lastname.#: serai.1
Assignment 1 - Computer Architecture CSE 6421

Q1.

a.

I1	Loop:	LD	R1,0(R2)	;load R1 from address 0+R2
I2		DADDI	R1,R1,#1	;R1=R1+1
I3		SD	R1,0,(R2)	;store R1 at address 0+R2
I4		DADDI	R2,R2,#4	;R2=R2+4
I5		DSUB	R4,R3,R2	;R4=R3-R2
I6		BNEZ	R4,Loop	;branch to Loop if R4!=0

Dependencies:

1. Data dependence – R1 register – between I1 and I2
2. Data dependence – R1 register – between I2 and I3
3. Name dependence – R2 register – between I3, I1 and I4
4. Data dependence – R2 register – between I4 and I5
5. Data dependence – R4 register – between I5 and I6
6. Control dependence – between I6 and I1,I2,I3,I4,I5

b.

The timing diagram is as follows:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
I1	IF	ID	EX	MEM	WB													
I2		IF	Stall	Stall	ID	EX	MEM	WB										
I3			Stall	Stall	IF	Stall	Stall	ID	EX	MEM	WB							
I4						Stall	Stall	IF	ID	EX	MEM	WB						
I5									IF	Stall	Stall	ID	EX	MEM	WB			
I6										Stall	Stall	IF	Stall	Stall	ID	EX	MEM	WB
I1-2													Stall	Stall	Stall	Stall	IF	ID

The loop executes (396/4) i.e. 99 times. The number of cycles consumed per loop is taken to be ‘n’ when, if the loop starts at the ‘i’th cycle, then the next instruction or the next loop iteration can start at the ‘i+n’th cycle. Assuming the definition above, the number of cycles consumed per loop iteration here is (17-1) i.e. 16, and hence the total number of cycles consumed is (16 x 99) i.e. 1584 cycles.

c.

The timing diagram for all but the last iteration of the loop, is as follows:

	1	2	3	4	5	6	7	8	9	10	11	12
I1	IF	ID	EX	MEM	WB							
I2		IF	ID	Stall	EX	MEM	WB					
I3			IF	Stall	ID	EX	MEM	WB				
I4				Stall	IF	ID	EX	MEM	WB			
I5						IF	ID	EX	MEM	WB		
I6							IF	Stall	ID	EX	MEM	WB
I7 (Wrong)								Stall	IF	…	…	…
I1-2										IF	ID	EX

The loop executes (396/4) i.e. 99 times. The number of cycles consumed per loop is taken to be 'n' when, if the loop starts at the 'i'th cycle, then the next instruction or the next loop iteration can start at the 'i+n'th cycle. Assuming the definition above, the number of cycles consumed per loop iteration here is (10-1) i.e. 9, for the first 98 iterations WHEN the loop is taken, but the prediction says not taken. For the last iteration, the prediction would be correct, so the I7 will no longer be a wrong instruction and the remaining instructions will continue as per the rest of the program; hence for the last iteration only (9-1) i.e. 8 cycles are consumed. Hence the total number of cycles consumed is ((9 x 98) + 8) i.e. 890 cycles.

Q2.

a.

Loop	+4
I0	+12
I1	+5
I2	+4
I3	+1
I4	+1
I5	+0
I6	+0
I7	+1
I8	+0
I9	+1
TOTAL	+29
GRAND TOTAL	11+29 = 40

Even though the branch is specified to be taken, the latency due to branch instruction is given to be a fixed value of '+1'. Given this and the other information in the question, the above table

shows the additional latencies instruction wise, and the TOTAL comes out to be '+29'. Since there are 11 instructions (Loop and I0 through I9), their basic execution time of one cycle will sum up to 11, and hence the GRAND TOTAL i.e. the baseline performance (in cycles, per loop iteration) comes out to be 40.

b.

The question seems to say that the pipeline will stall only on True DATA Dependencies, and a Branch is essentially a control dependency, so the timing is expected to be as follows:

1	Loop
2	<stall>
3	<stall>
4	<stall>
5	<stall>
6	I0
7	I1
8	I2
9	<stall>
10	<stall>
11	<stall>
12	<stall>
13	I3
14	<stall>
15	<stall>
16	<stall>
17	<stall>
18	<stall>
19	I4
20	I5
21	I6
22	I7
23	I8
24	I9

Accordingly, the number of cycles per loop iteration shall be 24.

NOTE: Here the fact that I9 is a Branch and may cause a stall is ignored because, the question seems to say that the pipeline will stall only on True DATA Dependencies, and a Branch is essentially a control dependency. Moreover the Branch target is the "Loop" which does not have any true data dependency with an incomplete instruction. If however, 1 stall occurs due to the Branch, the cycles/loop iteration will come out to be 25.

c.

Cycle	Pipe 1	Pipe 2
1	Loop	<stall>
2	<stall>	<stall>
3	<stall>	<stall>
4	<stall>	<stall>
5	<stall>	<stall>
6	I0	I1
7	I2	<stall>
8	<stall>	<stall>
9	<stall>	<stall>
10	<stall>	<stall>
11	<stall>	<stall>
12	I3	<stall>
13	<stall>	<stall>
14	<stall>	<stall>
15	<stall>	<stall>
16	<stall>	<stall>
17	<stall>	<stall>
18	<stall>	<stall>
19	I4	I5
20	I6	<stall>
21	I7	I8
22	I9	<stall>

ASSUMPTION:
The result of execution of an instruction at the ‘i’th cycle is only available towards the end of the cycle, and hence any instruction to be executed which must use the said result, would need to be executed only on the ‘i+1’th cycle or later.

The abovementioned assumption particularly causes a stall in Pipeline 2 on Cycle 20, as the Instruction I7 i.e. a store instruction calculates its address based on the value of Ry, which is computed in I6. This may be avoidable if the offset in I7 is modified, but no mention is made of such capability being available in the hardware, hence it is taken that a stall must indeed occur at that point.

Thus, with the above assumption, the number of cycles per loop iteration will come out to be 22.

NOTE:
Here the fact that I9 is a Branch and may cause a stall is ignored because, the question seems to say that the pipeline will stall only on True DATA Dependencies, and a Branch is essentially a control dependency. Moreover the Branch target is the “Loop” which does not have any true data dependency with an incomplete instruction. If however, 1 stall occurs due to the Branch, the cycles/loop iteration will come out to be 23.

PS: It is also assumed that, when I9 is executing in Pipeline 1, a stall must indeed occur in Pipeline 2, as even to predict that the branch is taken, and accordingly fetch the corresponding instruction from the target, the branch must at least execute for a period of 1 cycle. It is possible that this assumption may be untrue, which will not affect the time taken for execution of the first iteration of the loop, but may affect the other iterations.

Q3.

It seems that the problem expects us to apply a logic similar to that of Reservation Stations, where the renaming is done accordingly to a 1 to 1 correspondence table that is updated to hold the latest corresponding Temporary register.
Accordingly, the resulting code will come out to be:

```
LD      T9, 0(Rx)
MULTD   T10, F0, F2
DIVD    T11, T9, T10
LD      T12, 0(Ry)
ADDD    T13, F0, T12
SUBD    T14, T11, T13
SD      T14, 0 (Ry)
```

[Due to the way the SD instruction is written, F8 is only a source register, so it is only renamed to T14, and not a new register]