1. Identify the write-read, write-write, and read-write dependencies in the instruction sequence below by entering each line pair with a dependency in the correct column of the table to the right. For example, if L1 and L4 had a write-write dependency (which they don't), you would enter L1-L4 in the column labeled "write-write".

```
L1:   R1 = 100

L2:   R1 = R2 + R4

L3:   R2 = R4 - 25

L4:   R4 = R1 + R3

L5:   R1 = R1 + 30
```

write-read	write-write	read-write
*L2 – L4	*L1 – L2	L2 – L3
*L2 – L5	*L2 – L5	L2 – L4
L1 – L4	L1 – L5	L3 – L4
L1 – L5		L4 – L5

2. Rename the registers from problem 1 to prevent dependency problems. Identify references to initial register values using the subscript 'a' to the register reference.

```
L1:   R1b = 100

L2:   R1c = R2a + R4a

L3:   R2b = R4a - 25

L4:   R4b = R1c + R3a

L5:   R1d = R1c + 30
```

Questions 3 and 4 are based on the "in-order issue/in-order completion" execution sequence shown in the figure below.

Decode		Execute			Write		Cycle
I1	I2						1
	I2			I1			2
	I2			I1			3
I3	I4		I2				4
I5	I6		I4	I3	I1	I2	5
I5	I6			I3			6
		I5	I6		I3	I4	7
		I5					8
					I5	I6	9

3. Identify the most likely reason why I2 could not enter the execute stage until the 4th cycle. Will "in-order issue/out-of-order completion" or "out-of-order issue/out-of-order completion" fix this? If so, which?

Since I2 and I1 are in different columns of the execution unit, it is unlikely that there is a resource conflict, i.e., I2 is not waiting for I1 to finish using one of the CPU's resources. What is far more likely is that there is a true data dependency here. In other words, the result of I1 is needed to execute I2.

True data dependencies cannot be fixed using out-of-order issue or out-of-order completion. Therefore, there will be no speed up of I2 with respect to I1 by changing the issue sequence or output sequence.

4. Identify the reason why I6 could not enter the write stage until the 9th cycle. Will "in-order issue/out-of-order completion" or "out-of-order issue/out-of-order completion" fix this? If so, which?

The in-order completion requirements of the system require I5 to be completed before I6 can be written. The pair went into the pipe together and they must come out together.

This is not the case for out-of-order completion, so either "in-order issue/out-of-order completion" or "out-of-order issue/out-of-order completion" will fix this.

As a side note, let's look at what happens to the execution of these instructions if we do pass this through an "out-of-order issue/out-of-order completion" machine. Assume that the only true data dependency we have is between I1 and I2. and therefore, I2 must stay in the instruction window until I1 is finished.

Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	I1, I2			I1			2
I5	I6	I2, I3, I4			I1			3
		I4, I5, I6	I5	I2	I3	I1		4
		I6	I5	I4	I3	I2		5
				I6		I3	I4	6
						I5	I6	7

The first benefit of going to "out-of-order issue/out-of-order completion" is the ability to get instructions into the execution stage as soon as the resource becomes available. The decode unit takes one cycle to pull in a pair of instructions then pass them to the window. As long as the window has room for the instruction, nothing should hold up the decode stage.

The instructions must stay in the window until either the dependency is resolved (e.g., I1 – I2) or until the execute resource frees up allowing the stage to execute.

- I2 waits in the window until I1 is completed to satisfy the dependency
- I3 comes out of the window as soon as I1 is finished with the resource that I3 needs
- I4 comes out of the window as soon as I2 is finished with the resource that I4 needs
- I5 doesn't need to stay in the window because its resource is free as soon as it's done with decoding
- I6 has to wait until I4 is finished with the resource that I6 needs

The out-of-order completion allows instructions to enter the write cycle as soon as they are completed. The only hiccup in this process is that I5, I4, and I3 are all completed at the same time, but since there are only two write pipes, I5 must wait until cycle 7 to be written.

In the end, "out-of-order issue/out-of-order completion" saves two cycles. This reduces the execution time by 2 cycles/9 cycles = 22%.