1. What is the difference between a dependence and a hazard?

   A dependence is a property of the instructions in a program. For example, a true dependence arises when one instruction uses the value produced by an earlier instruction.
   A hazard is a potential problem in a pipeline that may arise from a dependence. For example, a true data dependence causes a hazard when the value produced by the earlier instruction is not yet available in the register file when the dependent instruction attempts to fetch it.

2. What is the difference between a data hazard and a structural hazard?

   Data hazard: instructions depends on result of prior instruction still in the pipeline (i.e. two instructions use the same value or storage)

   Structural hazard: hardware cannot support certain combinations of instructions (two instructions in the pipeline require the same resource)

3. Are WAW hazards possible in a simple 5-stage pipeline? Explain your answer.

   No, WAW hazards do not affect a simple 5-stage pipeline since all instructions are executed in order and the instructions have single-cycle execute stage. The WAW hazards cause issues if compiler reschedule the instructions or if the instructions have multiple execute cycles.

4. What is the significance of the phrase "Assume that the Writeback stage writes to the register file (RF) in the first half of a cycle, while the Decode stage reads form the register file in the second half of a cycle" from Section 3? Give an example of how the register file access order can affect the number of RAW stall cycle.

   If the assumption does not hold, the stalling cycle will increase when a dependency occurs. Let us look at an example:

   Assume that the Writeback stage writes to the register file (RF) in the first half of a cycle, while the Decode stage reads form the register file in the second half of a cycle

	Cycles														
Instr	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
ADD r1, r2 -> r3	F	D	X	M	W										
LW #0(r3) -> r4		F	d*	d*	D	X	M	W							
ADDI r3, 1 -> r6			p*	p*	F	D	X	M	W						
SW r3-> #0(r7)						F	D	X	M	W					

   At cycle 5, LW instruction get the results from ADD instruction since the write happens in the first half of the cycle, while the decode stage reads from RF in the second half of the cycle. The number of RAW stall cycle in this case is 2.

However, if you assume the write results are available at the end of the Writeback stage

	Cycles														
Instr	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
ADD r1, r2 -> r3	F	D	X	M	W										
LW #0(r3) -> r4		F	d*	d*	d*	D	X	M	W						
ADDI r3, 1 -> r6			p*	p*	p*	F	D	X	M	W					
SW r3-> #0(r7)							F	D	X	M	W				

The LW instruction get the correct results from RF in R6 thus increasing the number of RAW stall cycle by 1.

5. Provide a high-level algorithmic solution for each question of Section3, along with pipeline diagrams. Think of a series of instructions and specify the necessary conditions for a hazard to occur (e.g., data dependence, distance of instructions in the pipeline). Provide examples of all possible stall scenarios; for example, the conditions will be different for a one-cycle stall and a two-cycle stall.

High-level algorithm:
3.1:
Initialize a reg_used array with size MD_REG_TOTAL to -3. This is used to denote the current instruction cycle number where the registers are used.
At the end of every iteration store the current instruction cycle number of the destination registers.
For detecting hazards there are 2 possible cases:
1) If the adjacent instructions have dependencies (i.e. sim_num_insn – reg_used[r_in[]] == 1), a 2-cycle-stall is detected. #1-cycle-stall++ and reinitialize the reg_used[] to -3. (If a 2-cycle-stall and a 1-cycle-stall occur at the same time, it counts as a 2-cycle-stall.)
2) If a 1-cycle-stall occurs (i.e. sim_num_insn – reg_used[r_in[]] == 2), #2-cycle-stall++ and reinitialize the reg_used[] to -3.

	Cycles												Comments
Instr	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	
ADD r1, r2 -> r3	F	D	X	M	W								
LW #0(r3) -> r4		F	d*	d*	D	X	M	W					2-cycle-stall
ADDI r3, 1 -> r6					F	D	X	M	W				
SW r4-> #0(r7)						F	d*	D	X	M	W	W	1-cycle-stall

The number of RAW hazards is the sum of 1-cycle-stall and 2-cycle-stall and the CPI is given by:

CPI = 1 + (#1-cycle-stall / # instructions + 2 * #2-cycle-stall / #instructions)

3.2:
Initialize a reg_used array with size MD_REG_TOTAL to -2. This is used to denote the current instruction cycle number where the registers are used.
At the end of every iteration store the current instruction cycle number of the destination registers. There are 2 cases:

1) If the instruction is a store instruction, store sim_num_instruction + 1. (The reason is that for a store instruction the results are ready at Writeback stage while for an arithmetic instruction the results are ready at memory stage)
2) Else, store sim_num_instruction

For detecting hazards there are two cases:

1) I-cycle-stall if there is a dependency between an arithmetic instruction and a non-store instruction or an arithmetic instruction and a store instruction's second source register
2) 2-cycle-stall if there is a dependency between a load instruction and a non-store instruction or a store instruction's second source register

	Cycles												Comments
Instr	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	
ADD r1, r2 -> r3	F	D	X1	X2	M	W							
LW #0(r3) -> r4		F	d*	D	X1	X2	M	W					1-cycle-stall
ADD r4, r5 -> r6				F	d*	d*	D	X1	X2	M	W		2-cycle-stall

The number of RAW hazards is the sum of 1-cycle-stall and 2-cycle-stall and the CPI is given by:

CPI = 1 + (#1-cycle-stall / # instructions + 2 * #2-cycle-stall / #instructions)