Exercise 1. For the following code sequence in MIPS,

- Indicate the dependences
- Indicate the potential hazards and types
- Provide your hazard resolution methods and show how many extra clock cycles you have to pay.

```
sub $2, $1,$3    # Register $2 written by sub
and $12,$2,$5    # 1st operand($2) depends on sub
or  $13,$6,$2    # 2nd operand($2) depends on sub
add $14,$2,$2    # 1st($2) & 2nd($2) depend on sub
sw  $15,100($2) # Base ($2) depends on sub
```

		CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9
I_1	sub	IF	DR	ALU	idle	WB($2)				
I_2	and		IF	DR($2)	ALU	idle	WB			
I_3	or			IF	DR($2)	ALU	idle	WB		
I_4	add				IF	DR($2)	ALU	idle	WB	
I_5	sw					IF	DR($2)	ALU	MEM	idle

Answer:

Dependences and potential hazards:

(1) 1st operand($2) of `and` in I_2 depends on `sub` in I_1; Read before Write; Data hazard

(2) 2nd operand($2) `or` in I_3 depends on `sub` in I_1; Read before Write; Data hazard

(3) 1st($2) & 2nd($2) `add` in I_4 depend on `sub` in I_1; Read before Write; Structural hazard

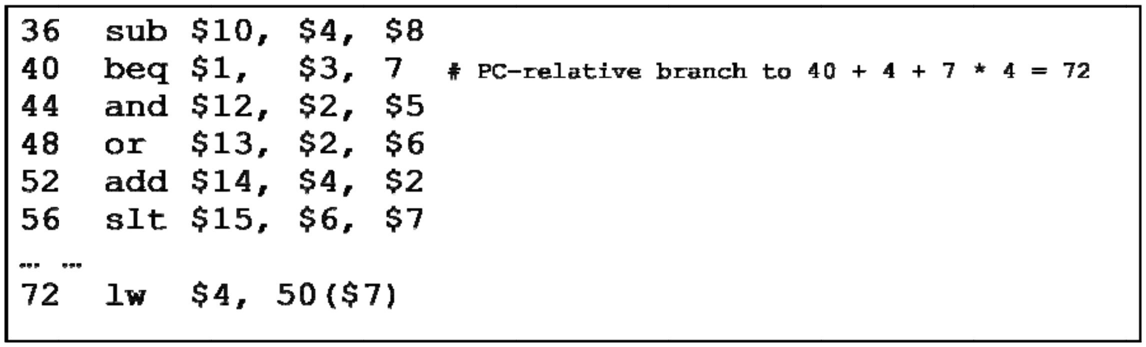
(4) Base ($2) of `sw` in I_5 depends on `sub` in I_1; Read before Write; (OK)

Resolution method 1: stall the pipeline for two cycles after sub (or equivalently insert two nops). Need to pay 2 extra clock cycles.

		CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9			
I_1	sub	IF	DR	ALU	idle	WB($2)							
	nop												
	nop												
I_2	and					IF	DR($2)	ALU	idle	WB			
I_3	or						IF	DR($2)	ALU	idle	WB		
I_4	add							IF	DR($2)	ALU	idle	WB	
I_5	sw								IF	DR($2)	ALU	MEM	idle

Resolution method 2: Use hardware support for register read and write to be done in one cycle to resolve the structural hazard; Use hardware hazard detection and forwarding units (ALU-ALU) to resolve the data hazards. No extra clock cycles to pay. However, the clock cycle time is longer in a pipeline system with hardware support for hazard detection and forwarding units.

Exercise 2: Show what happens when the branch is taken in this instruction sequence, assuming the pipeline is optimized for branches that are not taken and that we moved the branch execution to the ID stage. The numbers to the left of the instruction (40, 44, . . .) are the addresses of the instructions.



```
36   sub $10, $4, $8
40   beq $1,  $3, 7    # PC-relative branch to 40 + 4 + 7 * 4 = 72
44   and $12, $2, $5
48   or  $13, $2, $6
52   add $14, $4, $2
56   slt $15, $6, $7
...  ...
72   lw  $4, 50($7)
```

Answer:

With branch execution in the ID stage, when a branch is taken, there is one cycle delay (branch delay slot) to have the branch condition to be determined, and the instruction after the branch will be executed; When the branch is taken in this instruction sequence, the following sequence of instructions will be executed:

```
sub $10, $4, $8
beq $1,  $3, 7
and $12, $2, $5
lw  $4,  50 ($7)
```

This is not correct, because the instruction "`and $12, $2, $5`" should not be executed when the branch is taken. To avoid this mistake, a nop should be inserted after `beq,` and the sequence of instructions is executed when the branch is taken will be:

```
sub $10, $4, $8
beq $1,  $3, 7
nop
lw  $4,  50 ($7)
```