ICS 152, Problem Set 4

- Please show your work.
- Bottom line answers without proper explanation are worth **zero** points.

1. The *sub* and *lw* instructions have a hazard since the value stored in register $3 is updated by the *add* instruction. Fortunately, all this hazards can be resolved using forwarding. The last *add* instruction suffers from a load-use hazard and cannot be resolved without incurring a stall.

2. Freebie

3. At cycle 5, right before the instructions are executed:

	1	2	3	4	5
lw $5, 40($2)	IF	ID	EX	MEM	WB
add $6, $3, $2		IF	ID	EX	NOP
or $7, $2, $1			IF	ID	EX
and $8, $4, $3				IF	ID
sub $9, $2, $1					IF

Only *lw* and *or* require control (IF and ID controls are set every clock cycle).
WB: memtoReg = 1, RegWrite = 1
EX: RegDst = 1, ALUSrc = 0, ALUOp1 = 1, ALUOp2 = 0

No value has been stored at the beginning of the cycle, but at the end of it registers $5 will hold the value 1009.

4. (a) *add $1, $2, $3*
sw $4, 0($1)

This is an EX/MEM data hazard: the *sw* instruction needs the value of $1 in the EX stage (while computing the effective address), and the previous instruction (*add*) is in the MEM stage.

(b) *add $1, $2, $3*
sw $1, 0($4)

This is also an EX/MEM data hazard. The *sw* instruction needs the value of $1 in the MEM stage (when it will write it to memory), and the previous instruction (*add*) is in the WB stage. One might think that this is a MEM/WB data hazard, but we can resolve the hazard in the EX stage of the *sw* instruction because *sw* is an I-type instruction (the ALUSrc signal is set to 1, and the correct value of register $1 from EX/MEM.RegisterRd bypasses the ALU and feeds into the EX/MEM.RegisterRt register at the end of the cycle).

(c) *lw $1, 0($2)*
sw $1, 0($4)

- Please show your work.
- Bottom line answers without proper explanation are worth **zero** points.

This is a MEM/WB load-use hazard. It can be resolved by reading the value of register $1 from MEM/WB.RegisterRt and forwarding it to the "write-data" port.

(d) *lw $1, 0($2)*
sw $4, 0($1)

(e) *lw $1, 0($2)*
add $1, $1, $2

Both (d and e) are load-use hazards and cannot be resolved without incurring a stall. If a stall is taken, the hazard can be avoided using the same technique used for dealing with the MEM/WB data hazard

5. *subnz* is essentially a *sub* (the fact that the destination register is updated only when the difference is non-zero plays no role in determining whether or not there is a hazard and how to handle it). Hence, the solution is exactly what you would get for a regular *sub* instruction.

(a) When forwarding is not implemented, any instruction that uses the result of the *subnz* instruction before it completes its WB stage will cause a data hazard. *e.g.*
subnz $s0, $s1, $s2
addi $s3, $s0, 5

Pipeline Diagram without forwarding

Instruction	1	2	3	4	5	6	7	8
subnz	IF	ID	EX	NOP	WB			
addi		IF	ID	ID	ID	EX	NOP	WB

(b) Yes, forwarding can eliminate all data hazards. The hazard detection conditions are exactly the same for *sub*. *e.g.*
if (EX/MEM.Regwrite == 1 and EX/MEM.RegisterRd == ID/EX.RegisterRs)
{ do_forwarding }

ICS 152, Problem Set 4

- Please show your work.
- Bottom line answers without proper explanation are worth **zero** points.

6. a. Always taken

Branch	Accuracy (%)
1	100
2	0
3	50
4	80
5	5/7 = 71

b. Always not taken

Branch	Accuracy (%)
1	0
2	100
3	50
4	20
5	2/7 = 28

c. 1-bit predictor initialized to predict taken

Branch	Accuracy (%)
1	100
2	75
3	1/6 = 16.666
4	60
5	3/7 = 42

d. 2-bit predictor weakly initialized to predict taken

Branch	Accuracy (%)
1	100
2	75
3	50
4	80
5	71

7.
```
lw $2, 100($6)
lw $3, 200($7)
add $4, $2, $3
add $6, $3, $5
sub $8, $4, $6
lw $7, 300($8)
beq $7, $8, loop
```