Computer Systems HW #1

Due: Friday, Sept 8 (4 PM in ITT 305 mailbox or under my office door, ITT 313)

Type of Instruction	**Assembly Language**	**Register Transfer Language Description**
Memory Access (Load and Store)	lw $4, Mem	$4← [Mem]
	sw $4, Mem	Mem←$4
	lw $4, 16($3)	$4← [Mem at address in $3 + 16]
	sw $4, Mem	[Mem at address in $3 + 16]← $4
Move	move $4, $2	$4← $2
	li $4, 100	$4← 100
Load Address	la $5, mem	$4← load address of mem
Arithmetic Instruction (reg. operands only)	add $4, $2, $3	$4← $2 + $3
	mul $10, $12, $8	$10← $12 * $8 (32-bit product)
	sub $4, $2, $3	$4← $2 - $3
Arithmetic with Immediates (last operand must be an integer)	addi $4, $2, 100	$4← $2 + 100
	mul $4, $2, 100	$4← $2 * 100 (32-bit product)
Conditional Branch	bgt $4, $2, LABEL	Branch to LABEL if $4 > $2
Unconditional Branch	j LABEL	Always Branch to LABEL

```
sumPos = 0;
sumNeg = 0;
for i = 0 to length-1 do
    if numbers[i] < 0 then
        sumNeg = sumNeg + numbers[i]
    else
        sumPos = sumPos + numbers[i]
    end if
end for
```

1. Write MIPS Assembly Language code for the above algorithm that sums the array's elements.

```
                .data
numbers:    .word 20, 30, 10, 40, 50, 60, 30, 25, 10, 5
length:     .word 10
sumPos:     .word 0
sumNeg:     .word 0

                .text
                 .globl main
main:
```

2. Compare zero-, one-, two-, three-address, and the load & store machines by writing programs to compute

X = A + B * C - E;

Y = A / (E * C - D);

for each of the five machines. The instructions available for use are as follows:

3 Address	2 Address	1 Address (Accumulator machine)	0 Address (Stack machine)
MOVE (X ← Y)	MOVE (X ← Y)	LOAD M	PUSH M
		STORE M	POP M
ADD (X ← Y + Z)	ADD (X ← X + Y)	ADD M	ADD
SUB (X ← Y - Z)	SUB (X ← X - Y)	SUB M	SUB
MUL (X ← Y * Z)	MUL (X ← X * Y)	MUL M	MUL
DIV (X ← Y / Z)	DIV (X ← X / Y)	DIV M	DIV
		Notes: "SUB M" performs AC = AC - M "DIV M" performs AC = AC / M	Notes: "SUB" performs POP T POP T2 T3 = T2 - T PUSH T3 "DIV" performs POP T POP T2 T3 = T2 / T PUSH T3

Load/Store Architecture - operands for arithmetic operations must be from/to registers. For example, to perform the high-level statement "Z = X - Y" we need the code:

LOAD R2, X
LOAD R3, Y
SUB R4, R2, R3
STORE R4, Z

3. Assume 8-bit opcodes, 32-bit absolute addressing, 5-bit register numbers, and 32-bit operands. Compute the number of bits needed in programs from question 1 by completing the following table.

	3 Address	2 Address	1 Address	0 Address	Load & Store
Number of bits needed to store the program					
Number of bits of data transferred to and from memory					
Total number of bits read and written while the program executes					

4. You are to assume the same 6-stage pipeline as textbook when answering these questions.
Assume that the first register in an arithmetic operation is the destination register, e.g., in "ADD R3, R2, R1" register R3 receives the result of adding registers R2 and R1.

a. What would the timing be **without** bypass-signal paths/forwarding (use "stalls" to solve the data hazard)?
(This code might require more or less that 15 cycles)

	Time →														
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
SUB R3, R2, R1	FI	DI	CO	FO	EI	WO									
LOAD R4, 4(R3)		FI													
STORE R4, 4(R5)															
ADD R6, R4, R8															
SUB R5, R6, R4															
STORE R5, 8(R7)															

(Assume that a register **cannot** be written and the new value read in the same stage.)

b. What would the timing be **with** bypass-signal paths?
(This code might require more that 15 cycles)

	Time →														
Instructions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
SUB R3, R2, R1	FI	DI	CO	FO	EI	WO									
LOAD R4, 4(R3)		FI													
STORE R4, 4(R5)															
ADD R6, R4, R8															
SUB R5, R6, R4															
STORE R5, 8(R7)															

(Assume that a register **cannot** be written and the new value read in the same stage.)

c. Draw ALL the bypass-signal paths needed for the above example.