EE 2310 Homework #5 Solutions – Writing Simple Programs

Write each of the programs in this homework and run them on your SPIM emulator. DO NOT just write something in the area to the right. COMPOSE THE PROGRAMS ACCORDING TO THE DIRECTIONS AND MAKE SURE EACH RUNS PROPERLY AND GIVES THE CORRECT RESULT! Then turn in the programs when the homework is due.

1. **Write a program to do the following: Using the li instruction, load 50 in $t3, 72 in $t4, and 39 in $t5. Multiply the contents of $t3 and $t4, putting the result in $t0. Multiply the contents of $t4 and $t5, putting the result in $t1. Subtract the contents of $t1 from $t0, putting the result in $a0. Output the contents of $a0. Output the contents of $t0 and $t1 as well. Note that no data declaration is necessary. Write down the value of $a0 on your answer sheet. Remember to end your program correctly. After you have finished and the program executes correctly, add instructions to put each answer of the three on a different line (i.e., do a CR/LF between each answer).**

```
        .text
main:   li $t3,50
        li $t4,72
        li $t5,39
        mul $t0,$t3,$t4
        mul $t1,$t4,$t5
        sub $a0,$t0,$t1
        li $v0,1
        syscall
        li $a0,0x0a
        li $v0,11
        syscall
        move $a0,$t0
        li $v0,1
        syscall
        li $a0,0x0a
        li $v0,11
        syscall
        move $a0,$t1
        li $v0,1
        syscall

        li $v0,10
        syscall
```

Program is shown; red lines are carriage return/line feed.

2. **Write a program to do the following: Load the word t and u into registers and subtract u from t. Store that result in x. Load v and w into registers and subtract w from v, storing the result in y. Then multiply x times y and store the result in z. Output the numerical value of z to the console, and stop the program as usual.**

Remember: You can only perform mathematical operations on data that is in registers.

```
        .text
main:   lw $t0, t
        lw $t1,u
        sub $t2,$t0,$t1
        sw $t2,x
        lw $t4,v
        lw $t5,w
        sub $t7,$t4,$t5
        sw $t7,y
        mul $t8,$t2,$t7
        sw $t8,z

        move $a0,$t8
        li $v0,1
        syscall
        li $v0,10
        syscall

        .data
t:      .word 100
u:      .word 54
v:      .word 37
w:      .word 23
x:      .word 0
y:      .word 0
z:      .word 0
```

3. **The data declarations for the program below are done. First, use syscall 4 to output "Hello, world!" Then, remembering that data can only be manipulated in registers, subtract data2 from data1. Then add that result to data3. Divide that result by data4, then multiply the resulting quotient by data 5. Store this final result in ans, and output the result also to the console. You do not have to output a CR/LF before outputting the final result, as there is a CR/LF at the end of "Hello, world!"**

```
        .text
main:   la $a0,str
        li $v0,4
        syscall

        lw $t0,data1
        lw $t1,data2
        sub $t2,$t0,$t1
        lw $t3,data3
        add $t4,$t3,$t2
        lw $t5,data4
        div $t6,$t4,$t5
        lw $t7,data5
        mul $t8,$t6,$t7
        sw $t8,ans

        move $a0,$t8
        li $v0,1
        syscall
        li $v0,10
        syscall

        .data
str:  .asciiz "Hello, world!"
data1: .word 51
data2: .word 37
data3: .word 26
data4: .word 49
data5: .word 80
ans:    .word 0
```

4. We have not studied branch instructions yet, but they are actually quite easy to understand. In a branch instruction, a test is made for a certain condition, normally comparing two registers or perhaps determining the value of a single register. Consider the instruction "branch if less than zero:" An example is bltz $t4,next. What this branch instruction instructs the computer to do is to evaluate the contents of the register $t4. If the contents are less than zero ("ltz"), then the computer executes the instruction labeled "next" immediately after the branch. That instruction may be anywhere in the program, but if the contents of $t4 are less than 0, the computer immediately finds the instruction labeled "next" and executes it, then continues on from that point executing instructions. If the contents of $t4 are either zero or positive, the computer simply executes the instruction following the branch instruction.

Let's use that simple branch instruction to do the following: Load each of the two words declared in the data statement shown to the right into registers, and then determine if either is a negative number. If either number is negative, print it out using syscall 1. If a number is not negative (0 or positive), simply ignore it. End the program with a syscall 10. Note: you can declare words as either decimal or hexadecimal. SPIM understands both number systems.

```
        .text
main:   lw $a0,data1
        bltz $a0,print1
        j next
print1: li $v0,1
        syscall

next:   lw $a0,data2
        bltz $a0,print2
        j done

print2:  li $v0,1
        syscall
done:   li $v0,10
        syscall

        .data
data1: .word 0x6372eef5
data2: .word 0x91abccd2
```

5. **In a data statement, declare six words, aa, bb, cc, dd, ee, and ff. The words dd-ff are basically placeholders; declare each with a value of 0. The value of aa 159, the value of bb is 27, and the value of cc is 42. In the program square aa, bb, and cc, and store the results in dd, ee, and ff, respectively. Subtract ee from dd, and then subtract ff from that result. Output the value of dd – ee – ff to the console, then stop the program as usual.**

```
        .text
main:   lw $t0,aa
        mul $t0,$t0,$t0
        sw $t0,dd
        lw $t1,bb
        mul $t1,$t1,$t1
        sw $t1,ee
        lw $t2,cc
        mul $t2,$t2,$t2
        sw $t2,ff

        sub $t0,$t0,$t1
        sub $t0,$t0,$t2
        move $a0,$t0
        li $v0,1
        syscall

        li $v0,10
        syscall

        .data
aa:     .word 159
bb:     .word 27
cc:     .word 42
dd:     .word 0
ee:     .word 0
ff:     .word 0
```