CSE341 PROJECT REPORT

Name: Weiyang Chen
UB Person #: 50141904
Date: 04/10/2019

1. METHODS (5 POINTS)

Please explain how are you going to implement those functions in assembly language. You may list some key instructions you plan to use, or use pseudocode to illustrate your ideas. This question doesn't require any specific format. Just choose the most comfortable way to demonstrate your ideas to implement the desired functions.

Calculator Part

1. Asking user to input the first number and store to a register $s0.
2. Asking user to input the second number and store into register $s1.
3. Asking user to input an operation. Using 1-4 to instead of +, -, *, / , when user input the number, it will determine the number whether is out of bound, if less than 1 or greater than 4, will jump back to input an operation step and asking to input a valid operation between 1-4. If the input a valid number then store the number into register $s2.
4. Using li instruction to store the number 1-4 into register $t0 - $t3. So $t0 contains 1, $t1 contains 2 and so on. (These registers will use to determine the operation in the later.)
5. Using beq instruction to compare the operation number we store in the $s2 and $t0 - $t3. If $s2 equal to $t0 will jump to add number step. If $s2 equal to $t1 will jump to sub number step. If $s2 equal to $t2 will jump to multiple step. If $s2 equal to $t3 will jump to divide step.
6. In Add Step, Using add instruction to add number we inputted and stored in register $s0 and $s1 together and store into register $s3 jump to returnResult.
7. In Sub Step, Using sub instruction to sub number $s0 and $s1 and store into register $s3 then jump to returnResult.
8. In Multiple step, Using mul instruction to multiple number $0 and $s1 and store into register $s3 then jump to returnResult.
9. In Divide step, Using div instruction to divide number $0 by $s1 and sotre into register $s3 then jump to returnResult.

10. In returnResult, output the result store in the register $s3, and jump back to the beginning of the programming that restart to ask people input first number.

Hamming Distance

1. Aksing user to input first number, after user input a number, determine whether the number is between 10 – 99 , if not between 10 – 99 will return back and ask to input first number again. Until user input a valid number then store it into register $s0. (Using li to store 10 to register $t0, and 99 to register $t1 and use blt and bgt instruction to compare $t0 with $s0 and $t1 with $s0 and determine)
2. Asking user to input second number, after user input a number, determine whether the number is between 10 – 99 , if not between 10 – 99 will return back and ask to input second number again. Until user input a valid number then store it into register $s1. (Using li to store 10 to register $t0, and 99 to register $t1 and use blt and bgt instruction to compare $t0 with $s0 and $t1 with $s0 and determine)
3. Compare Front digit Integer, divide the each input number by 10 (Because in the first step we have already store 10 in the register $t0, So we can use div instruction for $s0 with $t0 and $s1 with $t0 to implement this.) to get their own front digit integer and c using bne instruction to compare them. If they are different then use addi instruction to add 1 into register $t2.
4. Compare Second digit Integer, Using front digit integer multiple by 10 and sub the original input number. Two get their own second digit Integer and use bne instruction to compare, if they are different, use addi instruction to add 1 into register $t2. And return the result.
5. Return the result store in the register $t2. And jump to the first step ask user to input the first number.

Euclidean Distance

1. Asking user input corrdinate x of Point A and store into register $s0.
2. Asking user input corrdinate y of Point A and store into register $s1.
3. Asking user input corrdinate x of Point B and store into register $s2.
4. Asking user input corrdinate y of Point B and store into register $s3.

Implement this formula

$$\mathrm{d}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathrm{x}_1 - \mathrm{y}_1)^2 + (\mathrm{x}_2 - \mathrm{y}_2)^2}$$

5. First :

   $$(\mathrm{x}_1 - \mathrm{y}_1)^2 + (\mathrm{x}_2 - \mathrm{y}_2)^2$$

   Using sub instruction to sub A'x and B'x and store number into register $t0. Same to the A'y and B'y and store number into register $t1. Using Mul instruction to square the $t0, (mul $t0, $0 , $0) and same way to square the $t1. And using add instruction to add them together into register $t2.

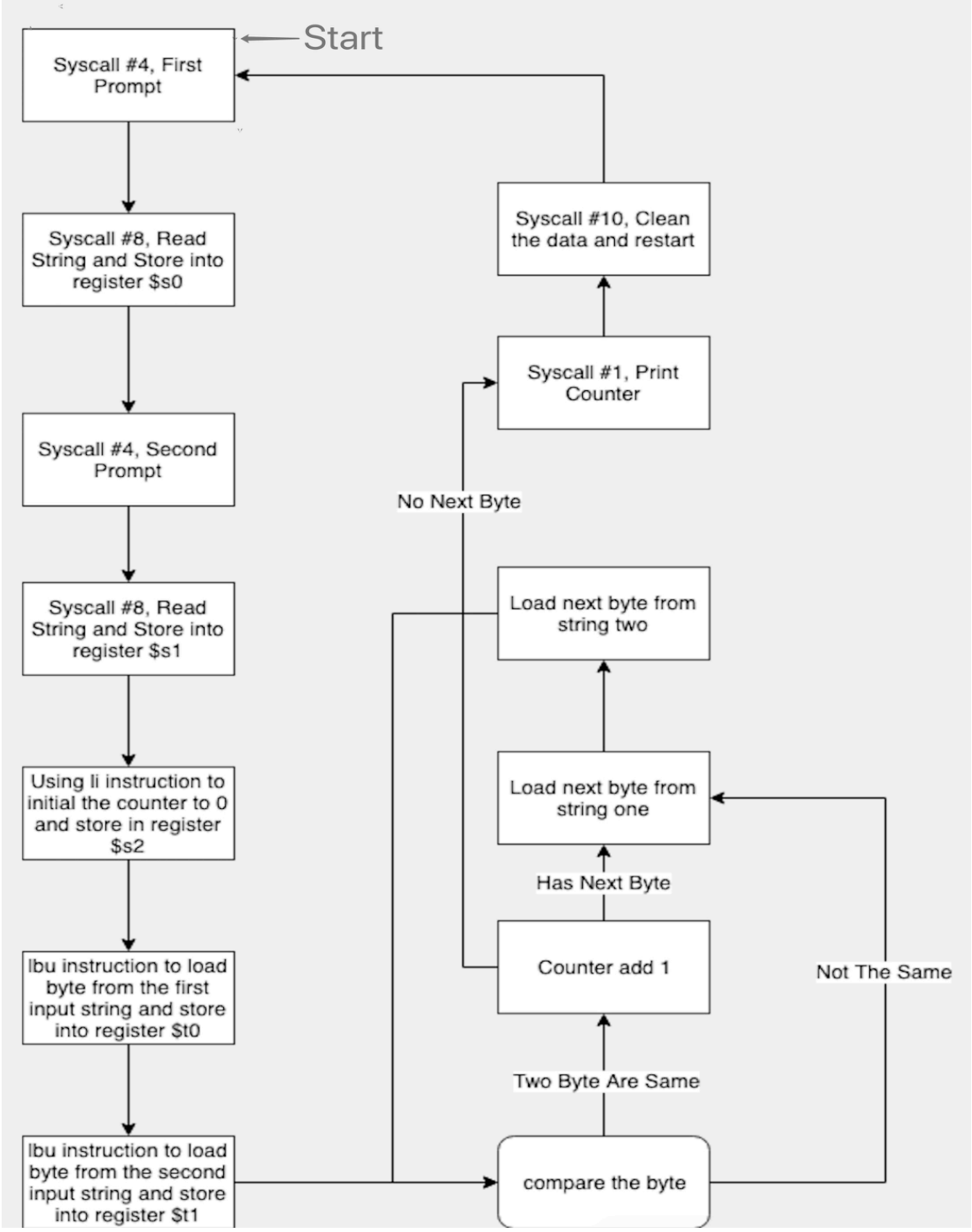
6. Second: implement square root

   Using li instruction to store 0 and 1 into register $t3 and $t4.

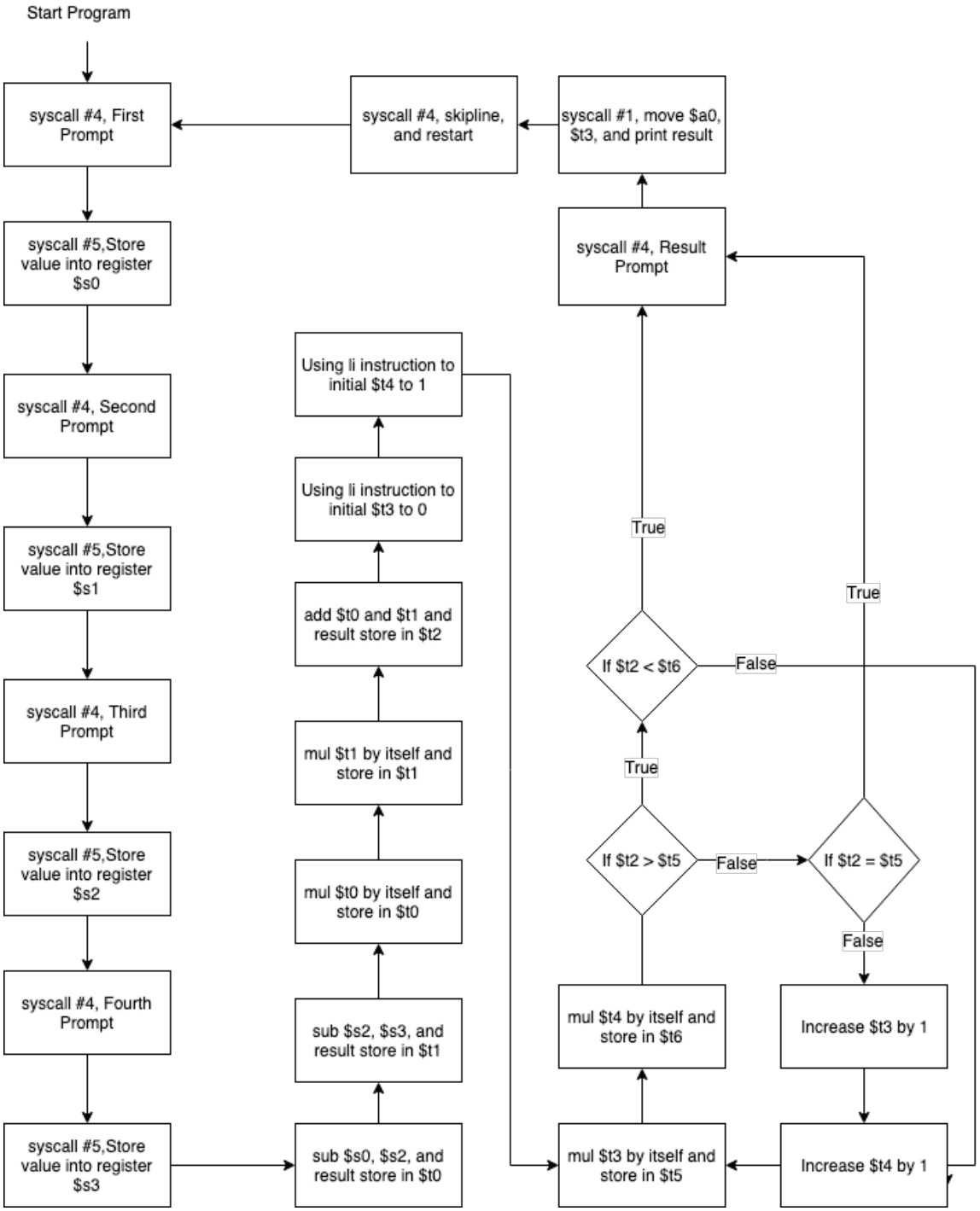
7. Using mul instruction to square $t3 and $t4 and store into register $t5 and $t6. And using bgt blt and beq to instruction to compare whether the number in the $t2 is between $t5 and $t6. If $t2 is between $t5 and $t6 then return lower bound as result which is $t3 as return result. Or if not between $t5 or $t6 then, using addi instruction to add 1 to $t3 and $t4, and repeat step 7 until return result.
8. Output the result store in the register $t3 and jump to Step 1.

2. FLOW CHART (10 POINTS)

1. Assume we no more limit the length of strings when calculating Hamming Distance. Use a flowchart or other methods to present how you will implement this assignment. (No unique solution here, but you need to at least indicate the registers and the syscalls you use)



2. Use flowchart or other methods to present how you implement radication. (No unique solution, but you need to at least indicate the registers and the syscalls you use)


Start Program
syscall #4, First Prompt
syscall #4, skipline, and restart
syscall #1, move $a0, $t3, and print result
syscall #5,Store value into register $s0
syscall #4, Result Prompt
syscall #4, Second Prompt
Using li instruction to initial $t4 to 1
Using li instruction to initial $t3 to 0
True
syscall #5,Store value into register $s1
add $t0 and $t1 and result store in $t2
True
If $t2 < $t6
False
syscall #4, Third Prompt
mul $t1 by itself and store in $t1
True
syscall #5,Store value into register $s2
mul $t0 by itself and store in $t0
If $t2 > $t5
False
If $t2 = $t5
syscall #4, Fourth Prompt
sub $s2, $s3, and result store in $t1
mul $t4 by itself and store in $t6
Increase $t3 by 1
False
syscall #5,Store value into register $s3
sub $s0, $s2, and result store in $t0
mul $t3 by itself and store in $t5
Increase $t4 by 1

3. COMMENTS FOR PROJECT (5 POINTS)

Give some evaluation of this project. (E.g., is this project difficult or easy? Whether there exists something unreasonable in this project. How to make this project helpful to you? etc.)

Answer : I think this project is easy, because you have gave us an example code. And it very easy to use the code as demo to add more function and implement the project. But if you just give the code like this, I think student will not spend more time on it. For example, some students after finishing the project still do not what is syscall. And which the different between different # of syscall. They just use the example code, copy and implement. But this project is still useful that I can write the assembly by myself, it more helpful for me to understand if statement and for loop in assembly. Implement by hand is better than just reading and watching.