BRANCHING OPERATIONS

LAB # 02

Fall 2023

CSE-304L Computer Organization and Architecture Lab

Submitted by: **Muhammad Shahab**

Registration No.: **21PWCSE2074**

Class Section: **C**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Submitted to:

Dr. Bilal Habib

Date:

14th October 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

ASSESSMENT RUBRICS COA LABS

LAB REPORT ASSESSMENT				
Criteria	**Excellent**	**Average**	**Nill**	**Marks Obtained**
1. Objectives of Lab	All objectives of lab are properly covered [Marks 10]	Objectives of lab are partially covered [Marks 5]	Objectives of lab are not shown [Marks 0]	
2. MIPS instructions with Comments and proper indentations.	All the instructions are well written with comments explaining the code and properly indented [Marks 20]	Some instructions are missing are poorly commented code [Marks 10]	The instructions are not properly written [Marks 0]	
3. Simulation run without error and warnings	The code is running in the simulator without any error and warnings [Marks 10]	The code is running but with some warnings or errors. [Marks 5]	The code is written but not running due to errors [Marks 0]	
4. Procedure	All the instructions are written with proper procedure [Marks 20]	Some steps are missing [Marks 10]	steps are totally missing [Marks 0]	
5. OUTPUT	Proper output of the code written in assembly [Marks 20]	Some of the outputs are missing [Marks 10]	No or wrong output [Marks 0]	
6. Conclusion	Conclusion about the lab is shown and written [Marks 20]	Conclusion about the lab is partially shown [Marks 10]	Conclusion about the lab is not shown[Marks0]	
7. Cheating			Any kind of cheating will lead to 0 Marks	

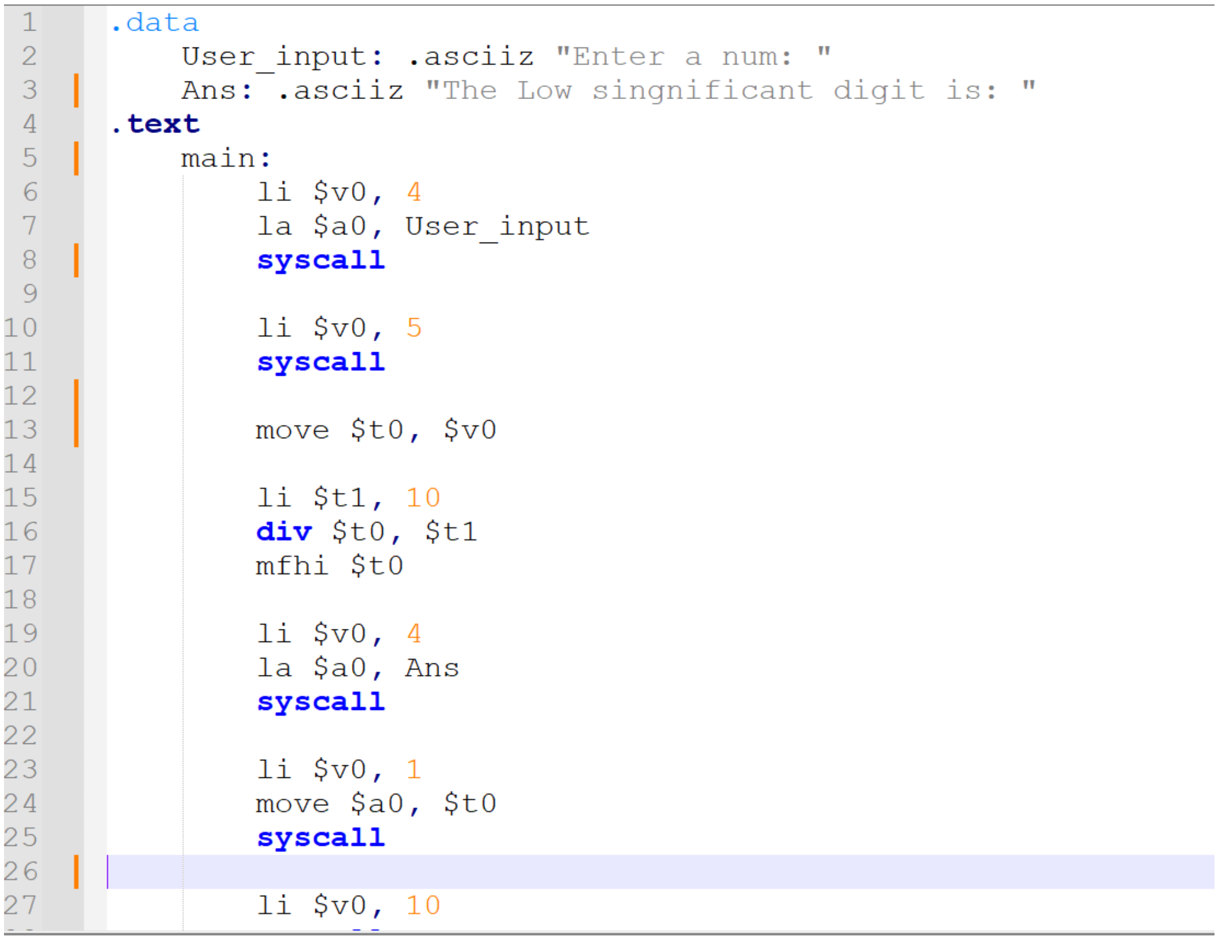
Total Marks Obtained:__________

Instructor Signature: ______________________

Task 1:

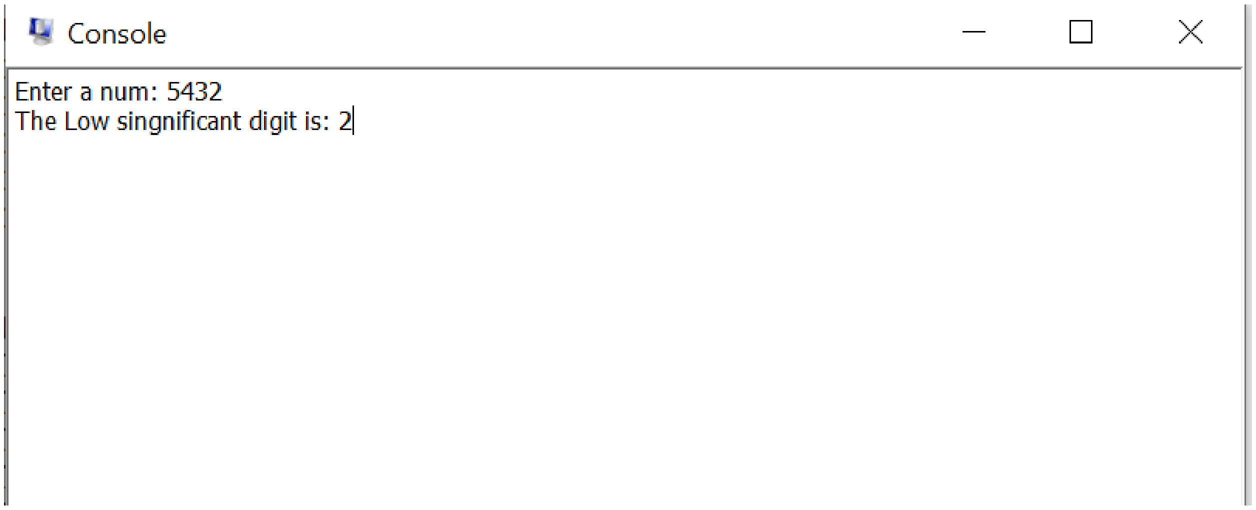
Enter a number 5432 from user and then display the last digit in the console. (hint: use mfhi).

Code:



```
.data
    User_input: .asciiz "Enter a num: "
    Ans: .asciiz "The Low singnificant digit is: "
.text
    main:
        li $v0, 4
        la $a0, User_input
        syscall

        li $v0, 5
        syscall

        move $t0, $v0

        li $t1, 10
        div $t0, $t1
        mfhi $t0

        li $v0, 4
        la $a0, Ans
        syscall

        li $v0, 1
        move $a0, $t0
        syscall

        li $v0, 10
```

Output:

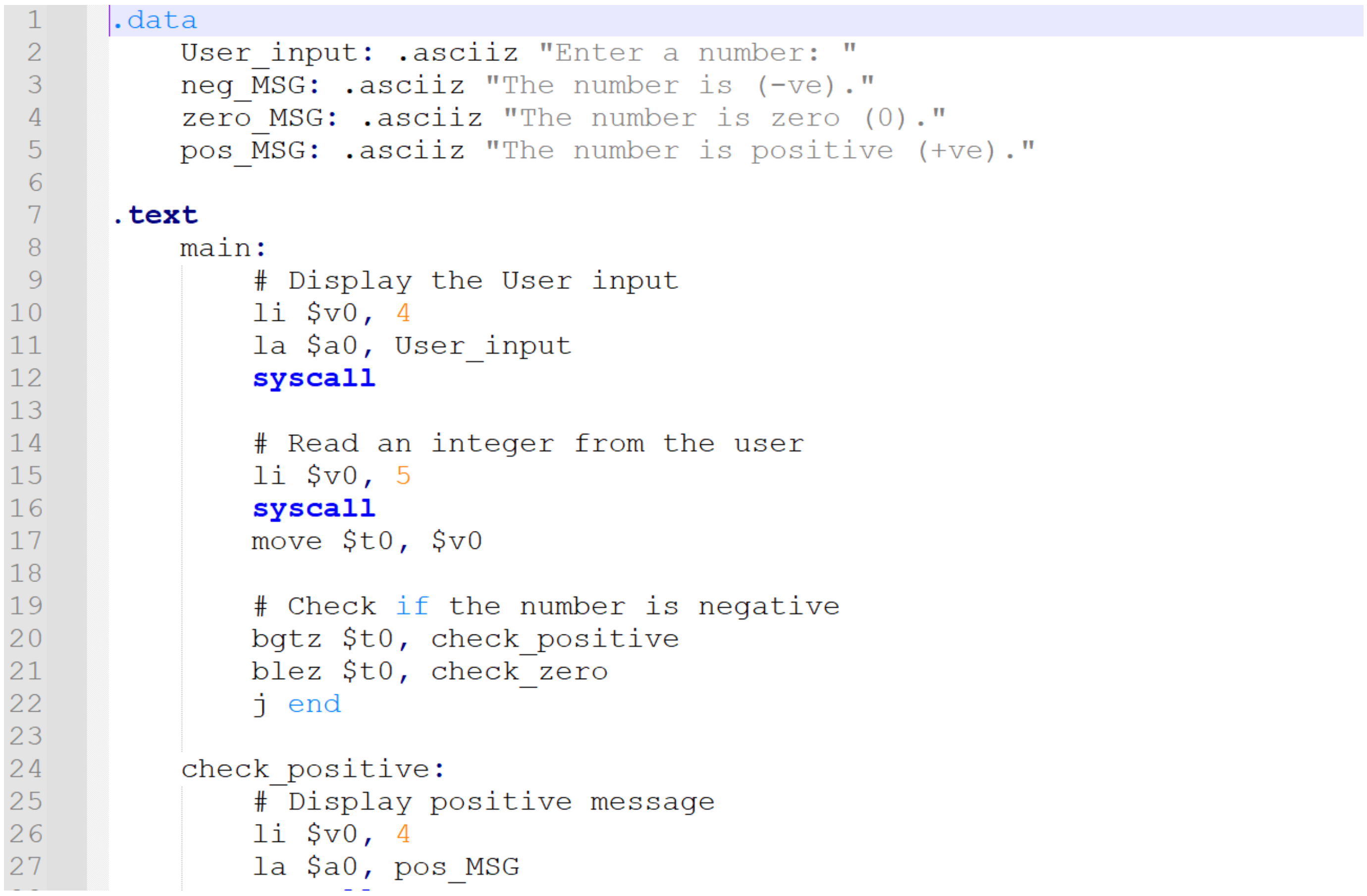


```
Enter a num: 5432
The Low singnificant digit is: 2
```

Task 2:

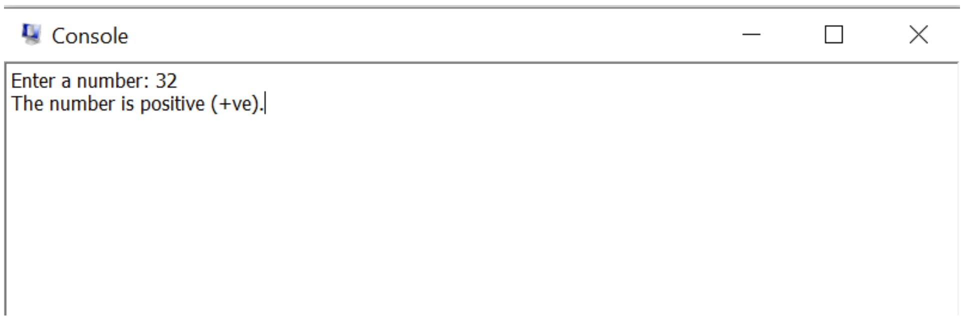
Check whether a number input by user is negative or equal to zero or greater then zero using branching (Use bgt or ble).

Code:



```
.data
    User_input: .asciiz "Enter a number: "
    neg_MSG: .asciiz "The number is (-ve)."
    zero_MSG: .asciiz "The number is zero (0)."
    pos_MSG: .asciiz "The number is positive (+ve)."

.text
    main:
        # Display the User input
        li $v0, 4
        la $a0, User_input
        syscall

        # Read an integer from the user
        li $v0, 5
        syscall
        move $t0, $v0

        # Check if the number is negative
        bgtz $t0, check_positive
        blez $t0, check_zero
        j end

    check_positive:
        # Display positive message
        li $v0, 4
        la $a0, pos_MSG
```

Output:

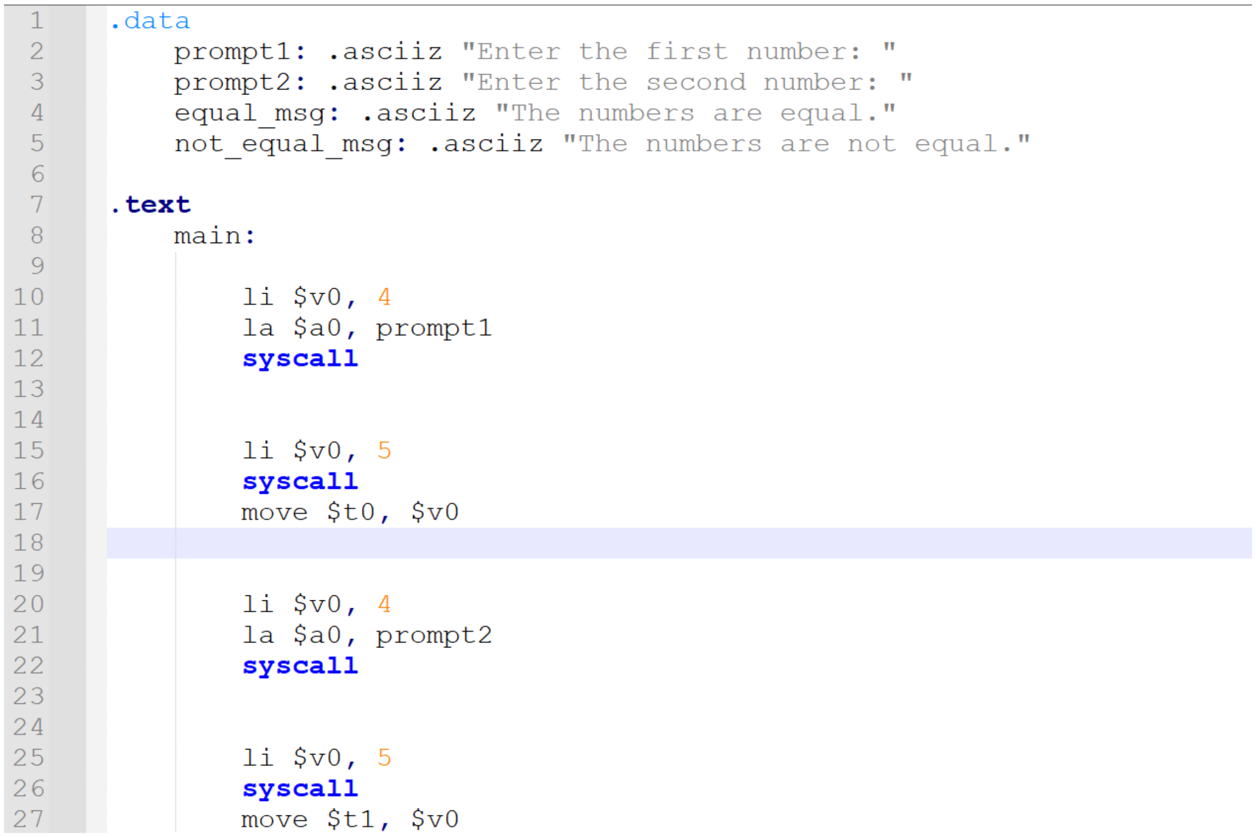


```
Enter a number: 32
The number is positive (+ve).
```

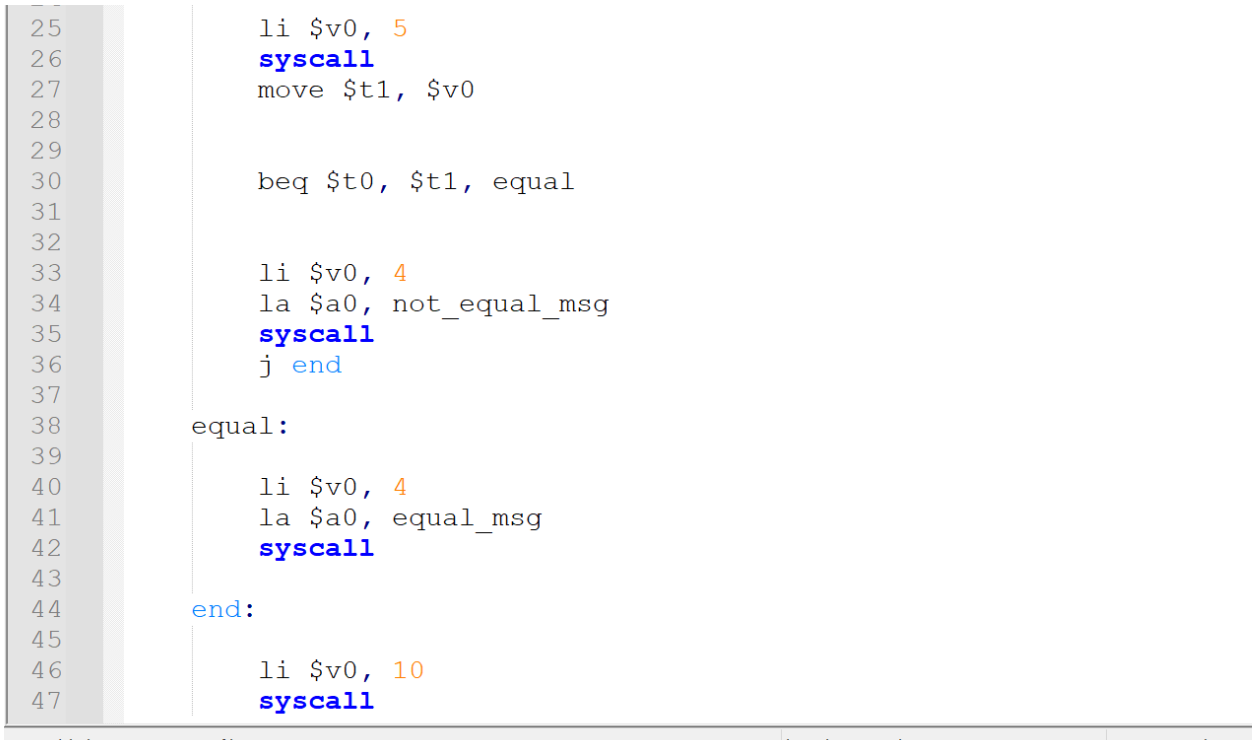
Task 3:

Check using branch whether the number input by user are equal or not (Use beq).

Code:

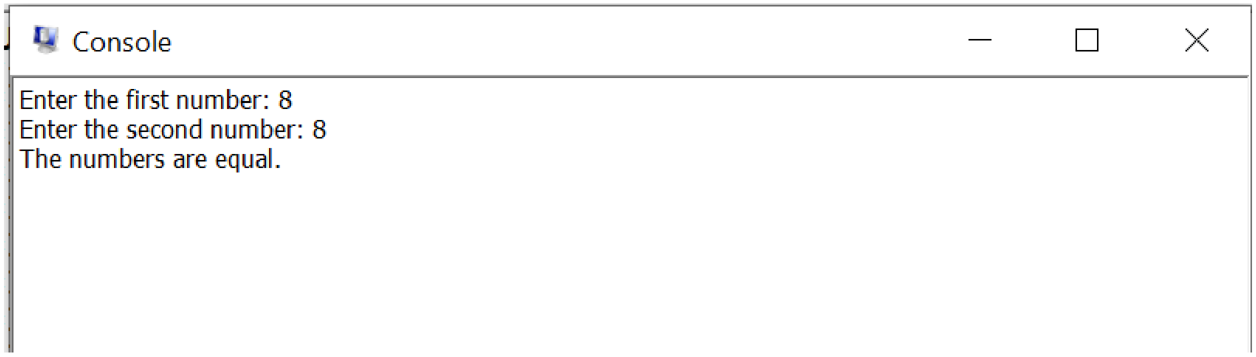


```
.data
    prompt1: .asciiz "Enter the first number: "
    prompt2: .asciiz "Enter the second number: "
    equal_msg: .asciiz "The numbers are equal."
    not_equal_msg: .asciiz "The numbers are not equal."

.text
    main:

        li $v0, 4
        la $a0, prompt1
        syscall


        li $v0, 5
        syscall
        move $t0, $v0


        li $v0, 4
        la $a0, prompt2
        syscall


        li $v0, 5
        syscall
        move $t1, $v0
```



```
        li $v0, 5
        syscall
        move $t1, $v0


        beq $t0, $t1, equal


        li $v0, 4
        la $a0, not_equal_msg
        syscall
        j end

    equal:

        li $v0, 4
        la $a0, equal_msg
        syscall

    end:

        li $v0, 10
        syscall
```

Output:



Console

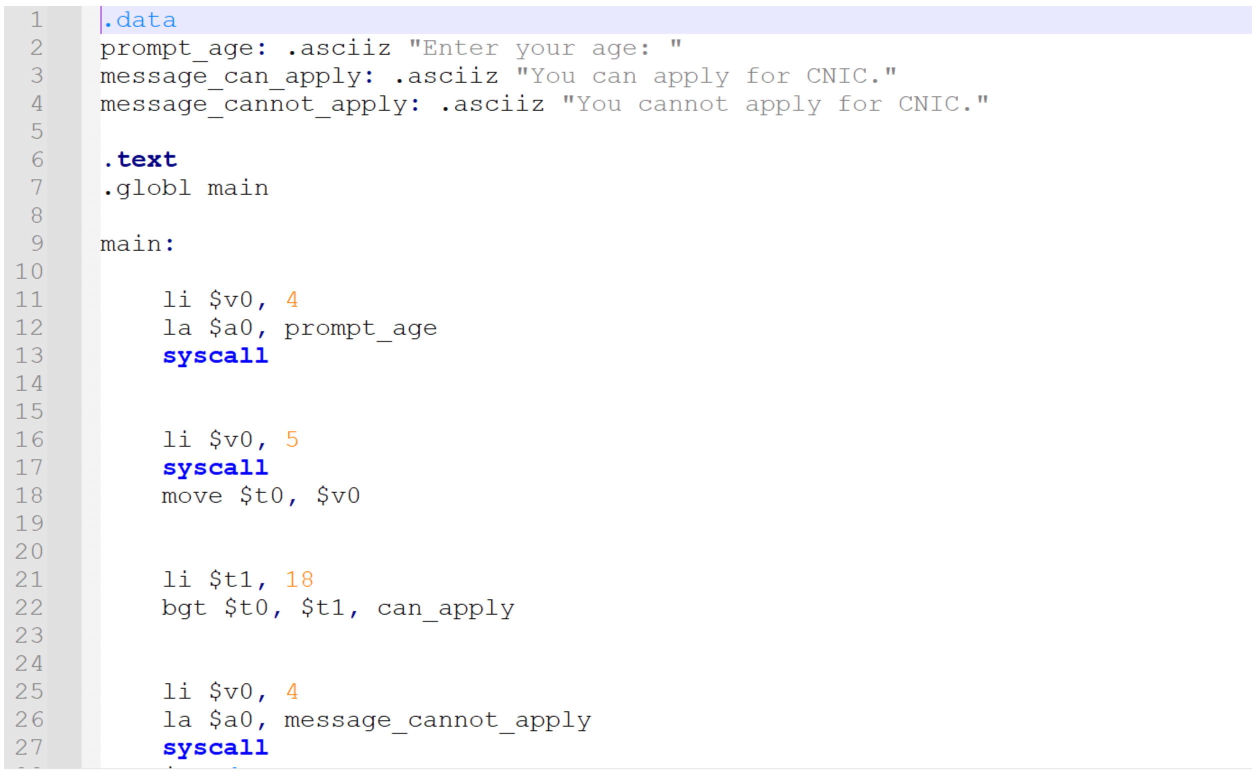
```
Enter the first number: 8
Enter the second number: 8
The numbers are equal.
```

Task 4:

Write the assembly of the below C++ code:

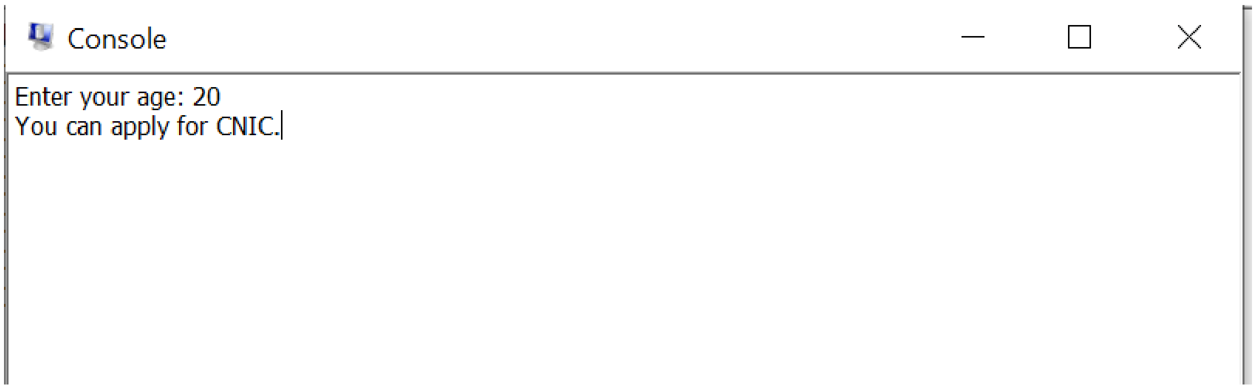
```
Int age;
Cout<<"enter your age"<<endl;
Cin>>age;
If(age > 18)
{
Cout<<"you can apply for CNIC"<<endl;
}
Else
{
Cout<<"you cannot apply for CNIC"<<endl;
}
```

Code:



```
.data
prompt_age: .asciiz "Enter your age: "
message_can_apply: .asciiz "You can apply for CNIC."
message_cannot_apply: .asciiz "You cannot apply for CNIC."

.text
.globl main

main:

    li $v0, 4
    la $a0, prompt_age
    syscall


    li $v0, 5
    syscall
    move $t0, $v0


    li $t1, 18
    bgt $t0, $t1, can_apply


    li $v0, 4
    la $a0, message_cannot_apply
    syscall
    li $v0, 4
    la $a0, message_cannot_apply
    syscall
    j end

can_apply:

    li $v0, 4
    la $a0, message_can_apply
    syscall

end:

    li $v0, 10
    syscall
```

Output:



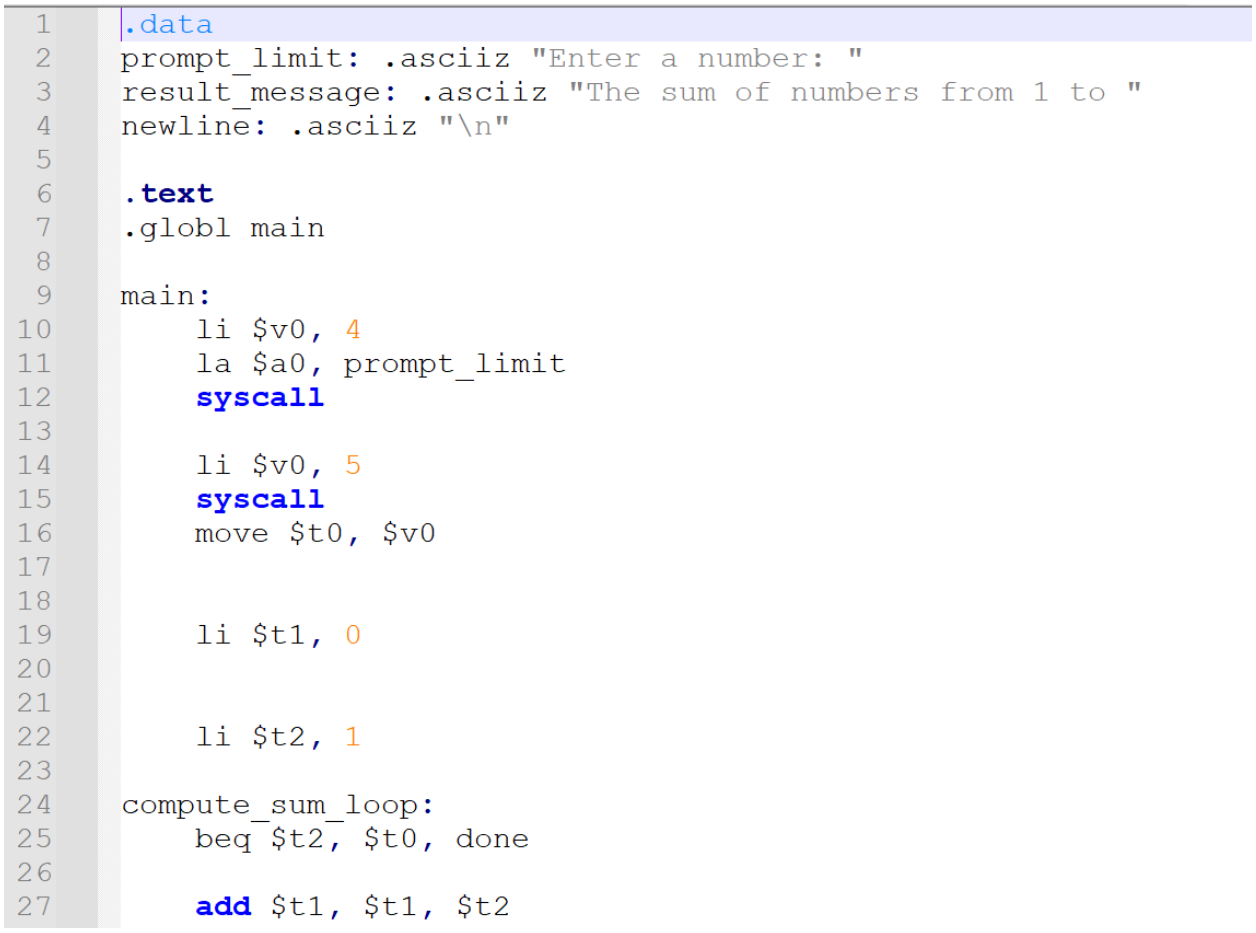
```
Enter your age: 20
You can apply for CNIC.
```

Task 5:

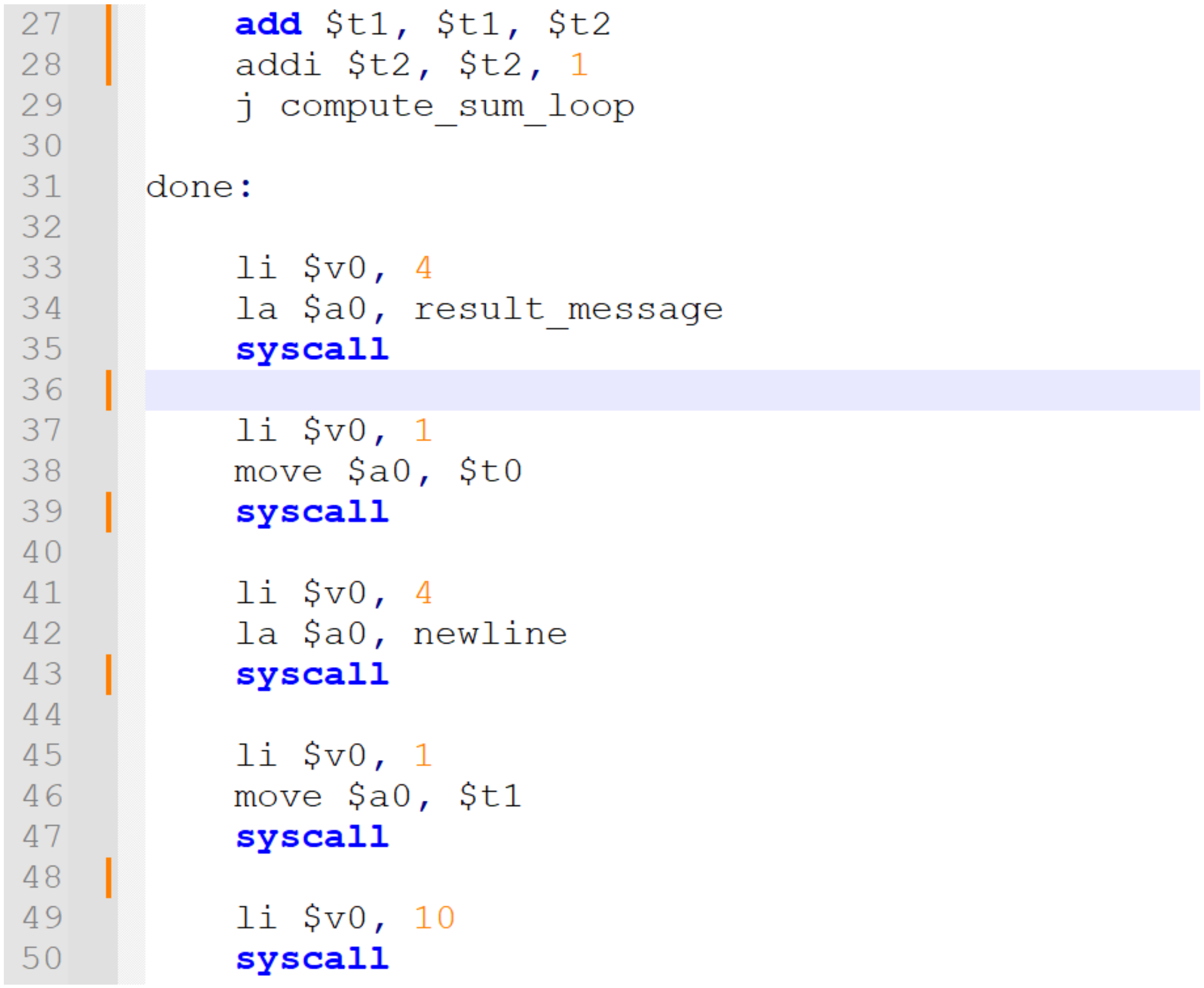
Write a program which take a limit from user and compute the sum of numbers from 0 to the limit (Use bqe, add, addi, and J (jump)). Below is the C++ language code:

```
Int limit;

Int sum;

Cout<<"Enter a number"<<endl;

Cin>>limit;

   for (int i = 1; i <= limit; ++i) {
      sum += i;
   }
   Cout<<"sum of numbers from 1 to <<limit<<"is"<<sum<<endl;
```

Code:

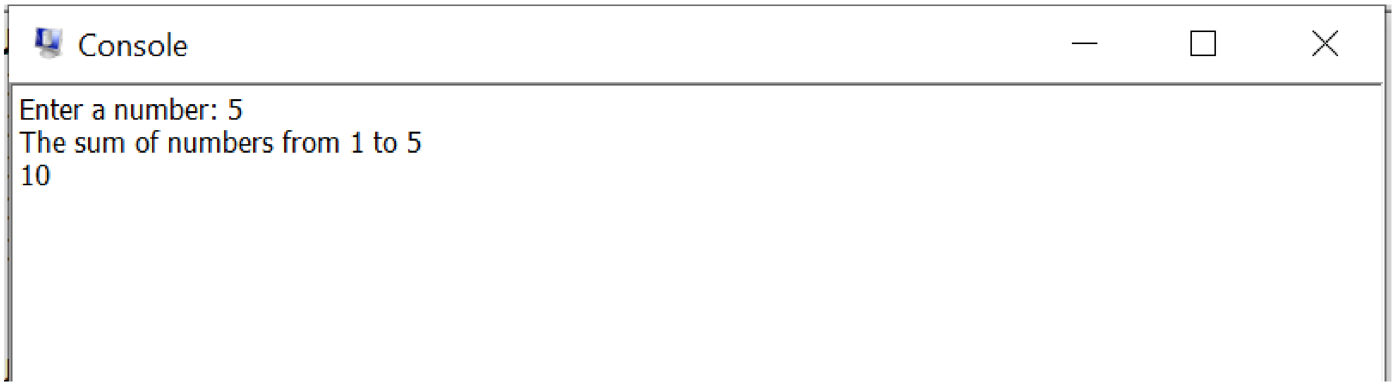


```
.data
prompt_limit: .asciiz "Enter a number: "
result_message: .asciiz "The sum of numbers from 1 to "
newline: .asciiz "\n"

.text
.globl main

main:
    li $v0, 4
    la $a0, prompt_limit
    syscall

    li $v0, 5
    syscall
    move $t0, $v0


    li $t1, 0


    li $t2, 1

compute_sum_loop:
    beq $t2, $t0, done

    add $t1, $t1, $t2
```



```
    add $t1, $t1, $t2
    addi $t2, $t2, 1
    j compute_sum_loop

done:

    li $v0, 4
    la $a0, result_message
    syscall

    li $v0, 1
    move $a0, $t0
    syscall

    li $v0, 4
    la $a0, newline
    syscall

    li $v0, 1
    move $a0, $t1
    syscall

    li $v0, 10
    syscall
```

Output:



```
Enter a number: 5
The sum of numbers from 1 to 5
10
```