Lab: **MIPS Assembly Programming**

• What is MIPS?

MIPS refer to Million Instruction Per Second assembly simulator, It is useful to understand Assembly instructions. MIPS is called a 32-bit architecture because it operates on 32-bit data. Computers only understand 1's and 0's (Machine language: binary representation of instructions). MIPS has 32-bit instructions, 32-bit data, 32 registers and possibly also 32-bit addresses. A 64-bit version of MIPS also exists.

Get start using SPIM:

The newest version of SPIM is called QtSPIM, and unlike all of the other version, it runs on Microsoft Windows, Mac OS X, and Linux—the same source code and the same user interface on all three platforms. QtSPIM is the version of SPIM that currently being actively maintained.

- **Components of an assembly program**

Lexical category	Example(s)
Comment	`# do the thing`
Assembler directive	`.data`, `.asciiz`, `.global`
Operation mnemonic	`add`, `addi`, `lw`, `bne`
Register name	`$10`, `$t2`
Address label (decl)	`hello:`, `length:`, `loop:`
Address label (use)	`hello`, `length`, `loop`
Integer constant	`16`, `-8`, `0xA4`
String constant	`"Hello, world!\n"`
Character constant	`'H'`, `'?'`, `'\n'`

To download SPIM simulator follow the following link:
http://e.informer.com/pages.cs.wisc.edu/~larus%2Fspim.html
after installing SPIM simulator correctly the following window will appear:

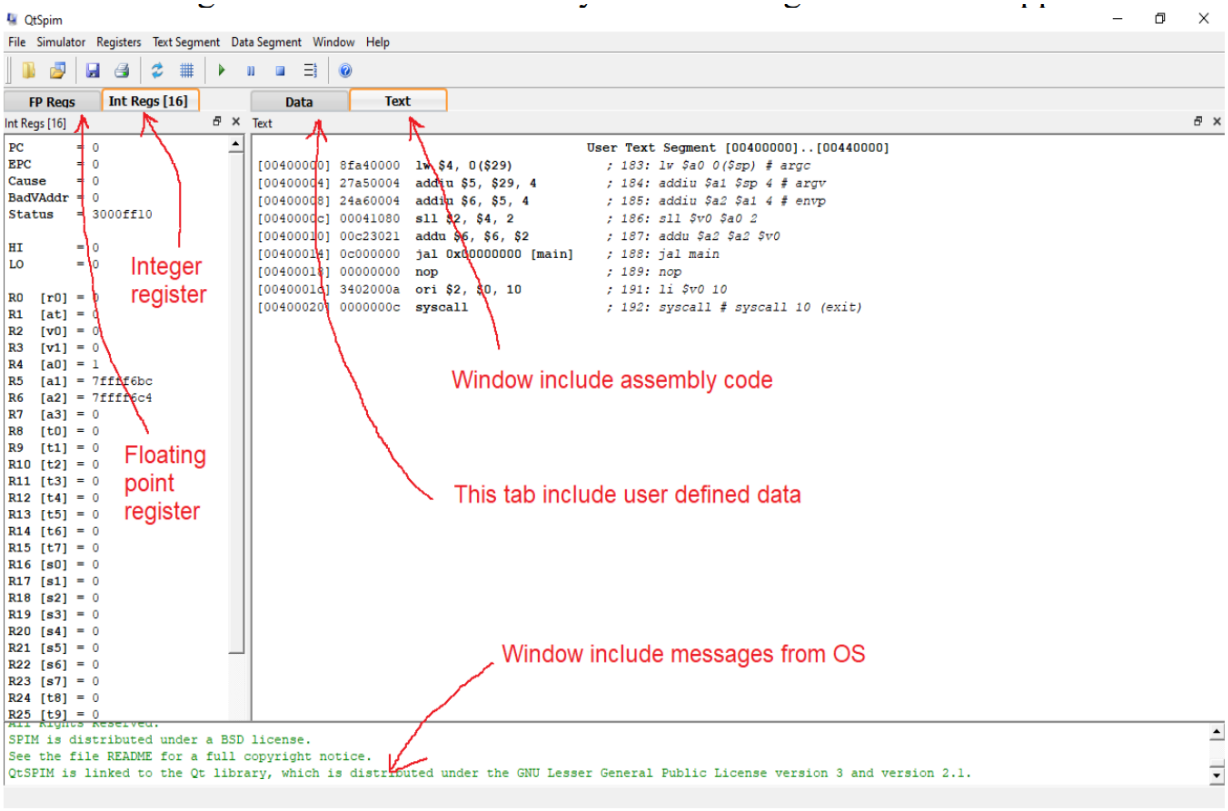

Figure1: SPIM simulator Graphical User Interface GUI

Output can be displayed in separate window called console.
To load assembly program it should include data segment and code (text) segment as showing below:

```
# Author:       your name
# Date:         current date
# Description: high-level description of your program

.data
```

Data segment:

- constant and variable definitions go here

```
.text
```

Text segment:

- assembly instructions go here

Figure2: Assembly program template

Procedure: A simple “Hello World” program

1. Start SPIM tool by double click installed icon shortcut. If you do not install it yet follow, Get start using SPIM section.
2. At any folder or path in your device, right click and select text document to generate assembly program
3. Copy the following assembly program in your document:

Description: A simple hello world program! :comment

	# add this stuff to the data segment
.data	# load the hello string into data memory
hello: .asciiz "Hello, world"	# now we’re in the text (code) segment
.text	# set up print string syscall
li $v0, 4	# immediate load 4 to register v0 where 4 refer to console
	# argument to print string
la $a0, hello	# load memory contents of address hello into register a0
syscall	# tell the OS to do the syscall
li $v0, 10	# set up exit syscall where 10 refer to exit
syscall	# tell the OS to do the syscall

```
# Author:       Eng Ahmed
# Date:         Oct 8, 2023
# Description:  A simple hello world program!

.data                   # add this stuff to the data segment

                        # load the hello string into data memory
hello:   .asciiz "Hello, world!"

.text                   # now we're in the text segment

        li    $v0, 4      # set up print string syscall
        la    $a0, hello  # argument to print string
        syscall           # tell the OS to do the syscall
        li    $v0, 10     # set up exit syscall
        syscall           # tell the OS to do the syscall
```

Save your program as task0.s (extension must be .s which refer to assembly file).

5. Using SPIM tool, select **Reinitialize and Load File** from **File** menu as shown in figure 3, then upload the Lab1.s assembly file you are generated in previous step.

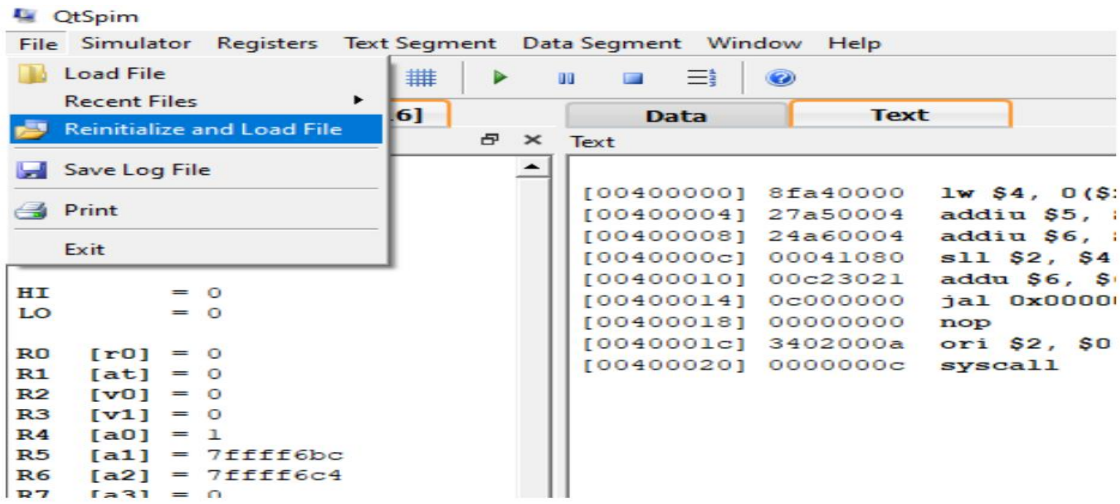

Figure3: Loading Assembly code in SPIM simulator

6. Using SPIM tool, select **Run Parameter** from **Simulator** menu, then adjust address to start running program to be the first instruction address of your code as shown in figure 4.

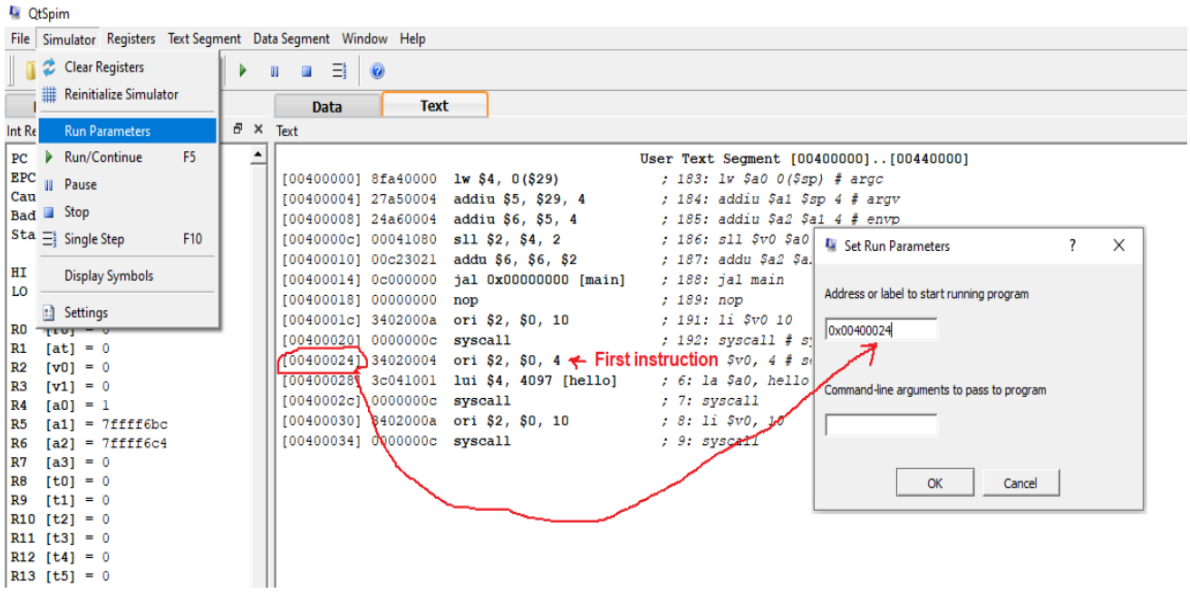

Figure 4: Adjust address of first instruction to run

7. Run the program by pressing Run button, result will appear in console window as shown in figure 5.

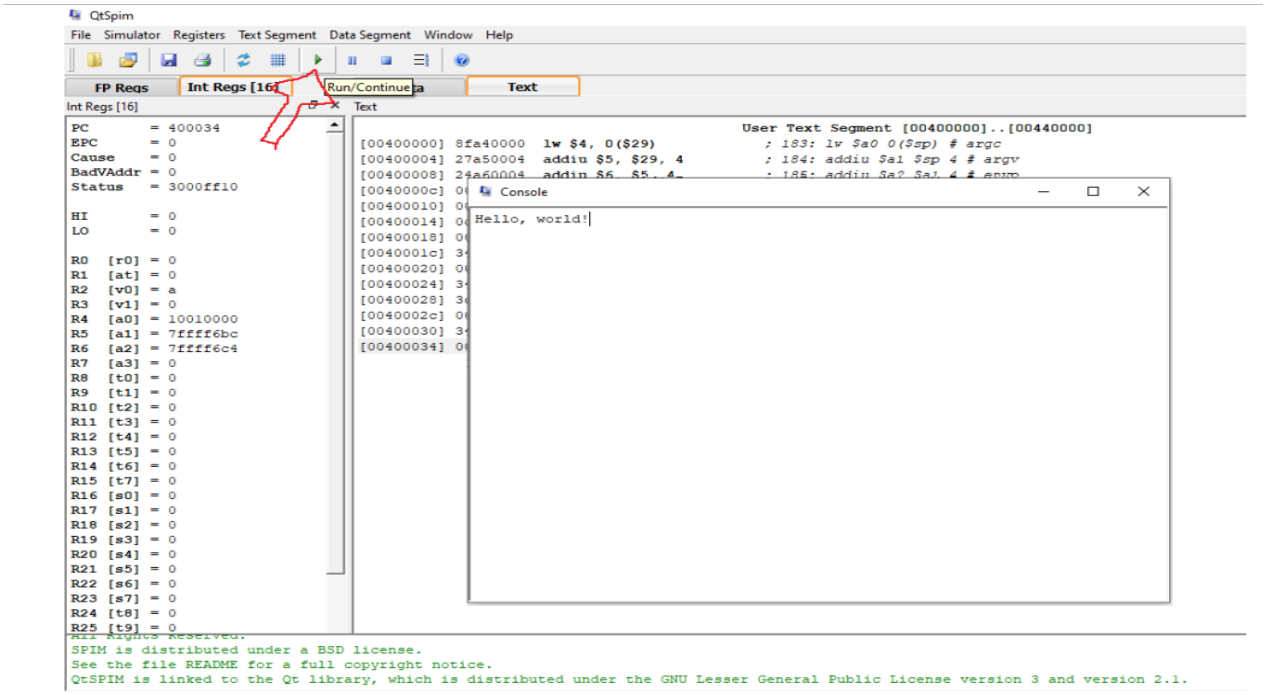

Figure 5: Hello world program result.

Note : to print multiple messages

- Use \n ,\t ,….
- Use multiple messages and print each of them.

Part 2 : Memory Access instructions:

We can access memory to perform one of two operations; to read (load) data from memory, or to write (store) data in memory.

Load instructions :

Instruction	Description	Example
li $reg, value	**Used to load immediate** value **to register** reg	**li $a0, 4**
la $reg, Label	**Used to load data from memory location labeled by** label **to register** reg	**la $a0, variable**
lw $reg, label **lw $reg, Label+I**	**Used to load word (32 bits) from memory location labeled by** label **to register** reg **or memory location with address = label +I**	**lw $t0, var** **lw $t0, var+4**
lb $reg, label **lbu $reg, label**	**Used to load byte (8 bits) from memory location labeled by** label **to register** reg. **lbu used to load unsigned byte**	**lb $t0, var**
lh $reg, label **lhu $reg, label**	**Used to load half word (16 bits) from memory location labeled by** label **to register** reg. **lhu used to load unsigned half word**	**lh $t0, var**

Store Instructions:

Instruction Description Example	Instruction Description Example	Instruction Description Example
sw $reg, label	sw $reg, label	sw $reg, label
sw $reg, Label+I	sw $reg, Label+I	sw $reg, Label+I
Used to store word (32 bits) in memory	Used to store word (32 bits) in memory	Used to store word (32 bits) in memory

If load or store applied in half word or byte it transfers lowest order that mean machine in MIPS is Little endian. Second operand may be as in load instruction. move instruction is used to transfer data from register to register:
move $a0, $t0 a0← t0

Arithmetic and logic operation instructions:

Instruction Description Example	Instruction Description Example	Instruction Description Example
add $reg1, $reg2, $reg3	Add contents of reg2 to reg3, result in reg1	add $a0, $a1, $a2
sub $reg1, $reg2, $reg3	Subtract contents of reg3 from reg2, result in reg1	sub $a0, $a1, $a2
mult $reg1, $reg2 mflo $reg3 mfhi $reg4	Multiply contents of reg1 times reg2, result in lo and hi egisters (lo contains least significant word, hi contains most significant word) mflo: used to move word from lo to reg3 mfhi: used to move word from hi to reg4	**mult $a1, $a2** **mflo $t0** **mfhi $t1**
div $reg2, $reg3	Integer divide contents of reg2 over reg3, result in lo and reminder in hi.	div $a1, $a2 mflo $t0
and $reg1, $reg2, $reg3	Logic AND contents of reg2 with reg3, result in reg1	and $a0, $a1, $a2
or $reg1, $reg2, $reg3	Logic OR contents of reg2 with reg3, result in reg1	or $a0, $a1, $a2
xor $reg1, $reg2, $reg3	Logic XOR contents of reg2 with reg3, result in reg1	xor $a0, $a1, $a2
nand $reg1, $reg2, $reg3	Logic NAND contents of reg2 with reg3, result in reg1	nand $a0, $a1, $a2
nor $reg1, $reg2, $reg3	Logic NOR contents of reg2 with reg3, result in reg1	nor $a0, $a1, $a2

Input data from user:

MIPS architecture allows user to input data via console using system call with service number 5 as shown below:

```
li $v0, 5       # adjust service of OS call to read input from user
syscall         # apply system call to perform input operation
move $s0, v0    # transfer user input from register v0 to register s0
```

Hint: to output result in console there are two service providers numbered by 4 and 1.
Service 4: is used to output string results
Service 1: is used to output integer values
Service 10: is used to exit the program

Task 2: add 2 integer num

```
.data
prompt: .asciiz "\n Please enter an integer: "
result: .asciiz "\n The addition result = "
        .extern foobar 4

    .text
    .globl main
main:
  # ask the user to enter the first integer
  li $v0, 4
  la $a0, prompt
  syscall

  # read the input integer and save it in s0 register
  li $v0, 5
  syscall
  move $s0, $v0

  # ask the user to enter the second integer
  li $v0, 4
  la $a0, prompt
  syscall

  # read the input integer and save it in s1 register
  li $v0, 5
  syscall
```

```
move $s1, $v0


# add two integer, result in s2
add $s2, $s0, $s1


# print the result message
li $v0, 4
la $a0, result
syscall


# display result in console
li $v0, 1
move $a0, $s2
syscall


# exit the program
li $v0, 10
syscall
```

part 3: if statement

Instruction	Description	Example
beq $reg1, $reg2, Label	Branch to instruction addressed by Label if reg1 = reg2	**beq $s0, $s1, L1**
bne $reg1, $reg2, Label	Branch to instruction addressed by Label	**bne $s0, $s1, L1**

Unconditional Branching instructions:

Instruction	Description	Example
j label	Unconditional jump to instruction addressed by Label	**j L1**
jr $reg	Unconditional jump to instruction addressed by the contents of register reg	**Jr $s0**

High-Level-Language code	MIPS instructions

if (i= = j)
f=g+h;
f=f-i;
	# assume $s0 = f, $s1 = g, $s2 = h, $s3
	= i, $s4 = j
	bne $s3, $s4, L1
	add $s0, $s1, $s2
	L1: sub $s0, $s0, $s3

Task 3: A simple condition program! enter enteger number and print addition if user enter -1

```
.data

prompt: .asciiz "\n Please enter an integer: "

result: .asciiz "\n The addition result = "

        .extern foobar 4


      .text

      .globl main

main:


      # set register s1 to conditional value = -1

       li $s1, -1


      # ask the user to enter an integer value

      L2: li $v0, 4

      la $a0, prompt

      syscall


      # read the input integer and save it in s0 register

      li $v0, 5

      syscall

      move $s0, $v0
```

```
# if input integer = -1, display result and exits program
beq $s0, $s1, L1

# add integer to sum, result in s2 and return to enter another integer
add $s2, $s2, $s0
j L2

# print the result message
L1: li $v0, 4
la $a0, result
syscall

# display result in console
li $v0, 1
move $a0, $s2
syscall

# exit the program
li $v0, 10
syscall
```

part 4: Array in MIPS

Initializing an "array" of data:

This way is used when the initial values of array elements are known

Initialize array in HLL	Initialize array in MIPS
int array[8] = {3, 8, 1, 6, 11, 7, 2};	.data array: .word 3, 8, 1, 6, 11, 7, 2

Reserving space for a N integer elements array:

This way is used when the initial values of array elements are unknown

Reserve array in HLL	Reserve array in MIPS
int array[8];	.data array: .word 0, 0, 0, 0, 0, 0, 0, 0 //there is another way can be used by define data ***.space*** instead of ***.word*** and write number of byte (8*4=32), but this way may cause bad address exception

Accessing array elements:

Array elements can be accessed (reading or writing) using label or register-base:

Accessing array in HLL	Accessing array using Label in MIPS	Accessing array using Register-base in MIPS
int array[3] = {3, 8, 1};	.data array: .word 3, 8, 1	.data array: .word 3, 8, 1
x = array[1] ;	lw $s0, array+4 //x=$s0	li $t0, 4 #index 1→offset 4 lw $s0, array($t0) #let x=$s0
array[2] = y ;	sw $s1, array+8 //y=$s1	li $t0, 8 #index 2→offset 8 sw $s1, array($t0) #let y=$s1

Implement For loop:

High-Level-Language code	MIPS instructions

// add the numbers from 3 to 9
int sum = 0; //initial value sum=0
int i;
for (i = 3; i != 10; i = i++)
//initialize i=3, condition, and increment i
{
sum = sum + i; //body
}
	# assume $s0 = i, $s1 = sum
	li $s1, 0 #initialize sum=0
	li $s0, 3 #initialize i=3
	li $t0, 10 #set test value
	for: beq $s0, $t0, done #condition
	add $s1, $s1, $s0 #body
	addi $s0, $s0, 1 #increment i
	j for
	done:

MIPS allows any comparisons relation using the following instructions:

i = = 0	i ! = 0	i > 0	i < 0	i > = 0	i < = 0
#$s0=i, $t0=0 beq $s0, $t0, L1	#$s0=i, $t0=0 bne $s0, $t0, L1	#$s0=i, $t0=0 bgt $s0, $t0, L1	#$s0=i, $t0=0 blt $s0, $t0, L1	#$s0=i, $t0=0 bge $s0, $t0, L1	#$s0=i, $t0=0 ble $s0, $t0, L1

The below example shows how to perform Less Than Comparisons statement:

High-Level-Language code	MIPS instructions

// add odd number from 3 to 99
int sum = 0; //initial value sum=0
int i;
for (i = 3; i < 100; i +=2)
//initialize i=3, condition, and increment i+2
{
sum = sum + i; //body
}
	# assume $s0 = i, $s1 = sum
	li $s1, 0 #initialize sum=0
	li $s0, 3 #initialize i=3
	li $t0, 100 #set test value
	for: bge $s0, $t0, done #condition
	add $s1, $s1, $s0 #body
	addi $s0, $s0, 2 #increment i by2
	j for

Implement While loop:

High-Level-Language code	MIPS instructions

// add 2 to x till x = 10
int x = 0; //initial value
while (x != 10) //condition
{
x = x + 2; //body
}
	# assume $s1 = x
	li $s1, 0 # initial value
	li $t0, 10 # set test value
	while: beq $s1, $t0, done # condition
	addi $s1, $s1, 2 # body
	j while
	done:

Task 4:

 # Description: A program displays largest integer using array and loop!

.data

prompt: .asciiz "\n Please enter an integer: "

array: .word 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0

result: .asciiz "\n The Largest integer = "

```
        .text
        .globl main
main:
    # set register s1 to conditional value = 10 that require to enter 10 integer
        li $s1, 10
    # load first element address offset in $t1
        li $t1, 0
        # initialize array index to 0 in register t0
        li $t0, 0
        # ask the user to enter an integer value
  enter: li $v0, 4
        la $a0, prompt
        syscall
        # read the input integer and save it in array
        li $v0, 5
        syscall
        sw $v0, array($t1)
        #adjust register t1 by address of next element
        addi $t1, $t1, 4
        # increment array index by 1
        addi $t0, $t0, 1
        # if array index (t0) reach 10 (s1), go to comparison section program
        beq $t0, $s1, comp
        # if array index less than 10 return to enter another integer
        j enter
 #readjust element offset to first element in register t1, array index to 0 in t0
   comp: li $t1, 0
```

```
    li $t0, 0
    #set largest element in s2 assume that it is first element
    lw $s2, array($t1)
    #adjust register t1 by offset of next element and put it in register s3
    large: addi $t1, $t1, 4
    lw $s3, array($t1)
    # increment array index by 1
    addi $t0, $t0, 1
    # if array index (t0) reach 10 (s1), go to done to print result
    beq $t0, $s1, done
    # compare largest element s2 with next element s3
    bge $s2, $s3, large
    #if largest element in s2 less than next element in s3 replace it
    move $s2, $s3
    j large
done: #print message result
    li $v0, 4
    la $a0, result
    syscall

    # display result in console
    li $v0, 1
    move $a0, $s2
    syscall
    # exit the program
    li $v0, 10
    syscall
```

part 5: function in mips

Call and return	discription	example
jal function label (call function)	Go to function labelled name: function label	jal sum
jr $ reg (return from function)	Return from function whose address at reg	jr $ra

Task 5: use function , stack

```
# Description: A simple addition program using functions
 .data
  prompt: .asciiz "\n Please enter an integer: "
      result: .asciiz "\n The addition result = "


      .text
      .globl main
main:
      #use stack to save return address at #register $ra
      addi $sp, $sp, -4
      sw $ra, 0($sp)


      #call enter function for first numbers
      jal enter
      move $s0, $v0


      #call enter function for second number
      jal enter
      move $s1, $v0
```

```
    #call sum function
    jal sum

    # print the result message
    li $v0, 4
    la $a0, result
    syscall

    # display result in console
    li $v0, 1
    move $a0, $s2
    syscall

   # exit the program
   li $v0, 10
   syscall

enter:
   # ask the user to enter the first integer
   li $v0, 4
   la $a0, prompt
   syscall
```

```
        # read the input integer from console
        li $v0, 5
        syscall

      #return to main function
      jr $ra
sum : add $s2, $s1,$s0
     jr $ra
```