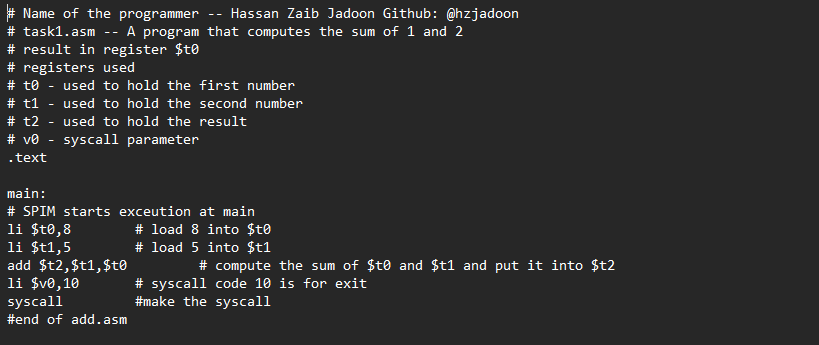
**LAB NO. 02: MIPS ASSEMBLY**

**Objectives:**The objective of this lab is to introduce the fundamentals of MIPS assembly language programming, covering basic syntax, instructions, and the use of registers. In this will learn to implement simple programs using arithmetic operations, data movement, and control flow in MIPS, gaining hands-on experience with low-level programming concepts.

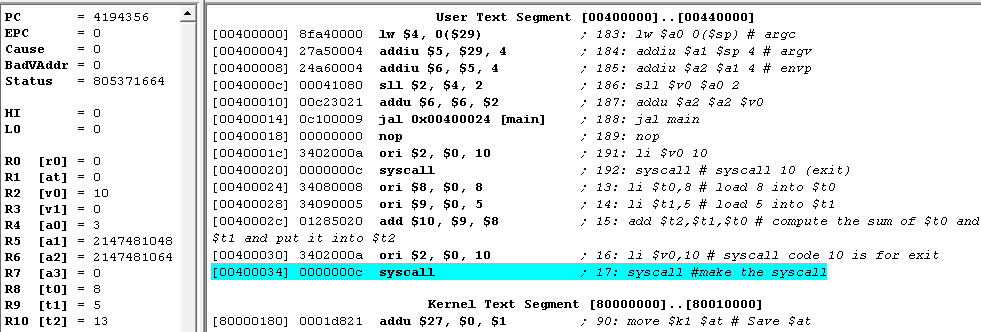
**SPIM syscalls:**

SPIM provides basic services using the syscall instruction. To request a service, load the system call code into $v0 and arguments into $a0 to $a3 (or $f12 for floating point values). Results return in $v0 (or $f0 for floating points). To display a string, load the string's address into $a0, set $v0 to 4, and run syscall. The remaining tasks are defining a string and finding its address.

**Task 1:**



**Output:**

****

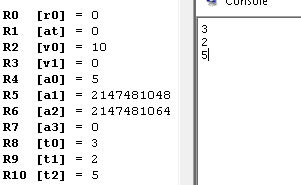
**Task 2:**

Write a program in MIPS assembly language, task2.asm, that computes and prints the sum of two numbers specified at runtime by the user.

**A screenshot of a computer program

Description automatically generated**

**Output:**

****

**Data and Text Segment:**

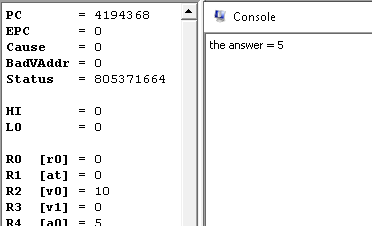
The string "Hello World" should not be part of the executable part of the program (which contains all of the instructions to execute), which is called the text segment of the program. Instead, the string should be part of the data used by the program, which is, by convention, stored in the data segment. The MIPS assembler allows the programmer to specify which segment to store each item in a program by the use of several assembler directives. To put something in the data segment, all we need to do is to put a .data before we define it. Everything between a .data directive and the next .text directive (or the end of the file) is put into the data segment. Note that by default, the assembler starts in the text segment, which is why our earlier programs worked properly even though we didn’t explicitly mention which segment to use. In general, however, it is a good idea to include segment directives in your code, and we will do so from this point on.

**Task 3:**

**A screenshot of a computer program

Description automatically generated**

**Output:**

****

**Task 4:**

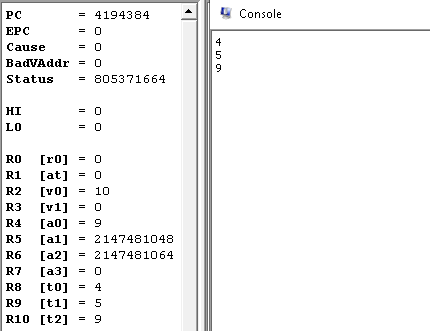
Write a program to take 2 numbers from the user, compute their sum and display the sum result on console

**Code:**

A screenshot of a computer

Description automatically generated

**Output:**

****

**Task 5:**

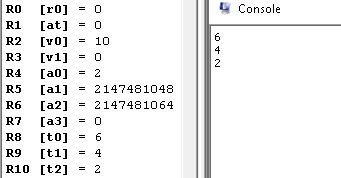
Write a program to take 2 numbers from the user, compute their difference and display the difference result on console

**Code:**

**A screenshot of a computer

Description automatically generated**

**Output:**



**Task 6:**

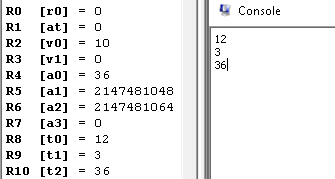
Write a program to take 2 numbers from the user, compute their product using mul() and display the product result on console

**Code:**

**A screenshot of a computer

Description automatically generated**

**Output:**

****

**Task 7:**

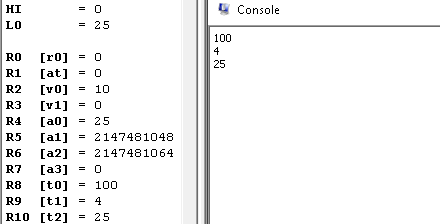
Write a program to take 2 numbers from the user, compute their division and display the division result on console

**Code:**

A screenshot of a computer

Description automatically generated

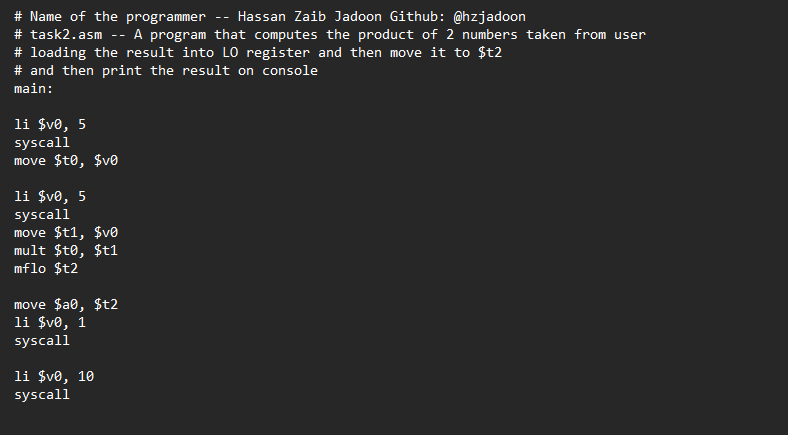
**Output:**

****

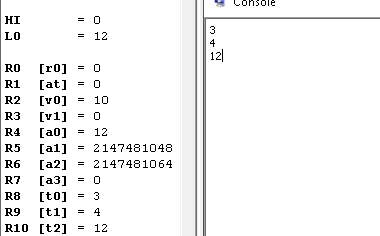
**Task 8:**

Write a program to take 2 numbers from the user, compute their product using mult() and display the product result on console

**Code:**

****

**Output:**

****