EXPERIMENT 7

**Aim**: Study and Simulate (Stepwise) Experiments of MIPS Programming available at MIPS simulator. Modify the given programs to implement

1) Add 10 nos.

2) to print message "Hello MIPS"

3) To Reverse the input string (e.g."ABC" - "CBA").

**Submission Sheet**

| **SAP ID** | **Name of Student** | **Date of Experiment** | **Date of Submission** | **Remarks** |
| --- | --- | --- | --- | --- |
| 60004190057 | Junaid Girkar |  |  |  |

**Theory**:

MIPS assembly language simply refers to the assembly language of the MIPS processor. The term MIPS is an acronym for Microprocessor without Interlocked Pipeline Stages. It is a reduced-instruction set architecture developed by an organization called MIPS Technologies. The MIPS assembly language is a very useful language to learn because many embedded systems run on the MIPS processor.

**MIPS Architecture**

**Data Types**

1. All the instructions in MIPS are 32 bits.

2. A byte in the MIPS architecture represents 8 bits; a halfword represents 2 bytes (16 bits) and a word represents 4 bytes (32 bits).

3. Each character used in the MIPS architecture requires 1 byte of storage. Each integer used requires 4 bytes of storage.

**Literals**

In the MIPS architecture, literals represent all numbers (e.g. 5), characters enclosed in single quotes (e.g. ‘g’) and strings enclosed in double quotes (e.g. “Deadpool”).

Code 1: To add 10 numbers

**Registers**

MIPS architecture uses 32 general-purpose registers. Each register in this architecture is preceded by ‘$’ in the assembly language instruction. You can address these registers in one of two ways. Either use the register’s number (that is, from $0 to $31), or the register’s name (for example, $t1).

**MIPS Structure**

**General structure of a program created using the MIPS assembly language**

A typical program created using the MIPS assembly language has two main parts. They are the data declaration section of the program and the code section of the program.

**Data declaration section of a MIPS assembly language program**

The data declaration section of the program is the part of the program identified with the assembler directive .data. This is the part of the program in which all the variables to be used in the program are created and defined. It is also the part of the program where storage is allocated in the main memory (RAM). The MIPS assembly language program declares variables as follows: name: .storage\_type value(s).

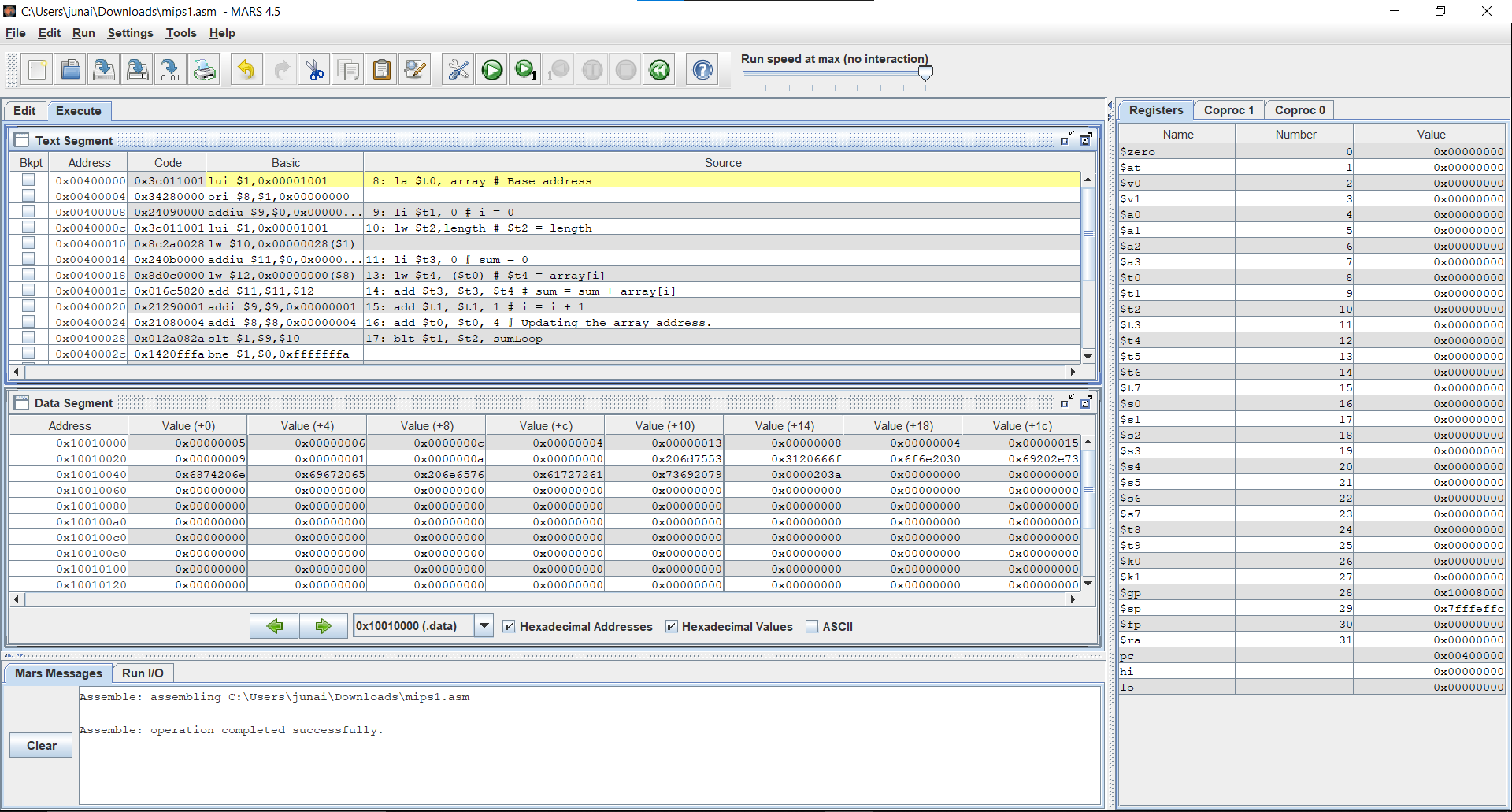
The “name” refers to the name of the variable being created. The “storage\_type” refers to the type of data that the variable is meant to store. The “value(s)” refers to the information to be stored in the variable being created.

**Code section of the MIPS assembly language program**

The code section of the program is the part of the program in which the instructions to be executed by the program are written. It is placed in the section of the program identified with the assembler directive .text. The starting point for the code section of the program is marked with the label “main” and the ending point for the code section of the program is marked with an exit system call. This section of a MIPS assembly language program typically involves the manipulation of registers and the performance of arithmetic operations.

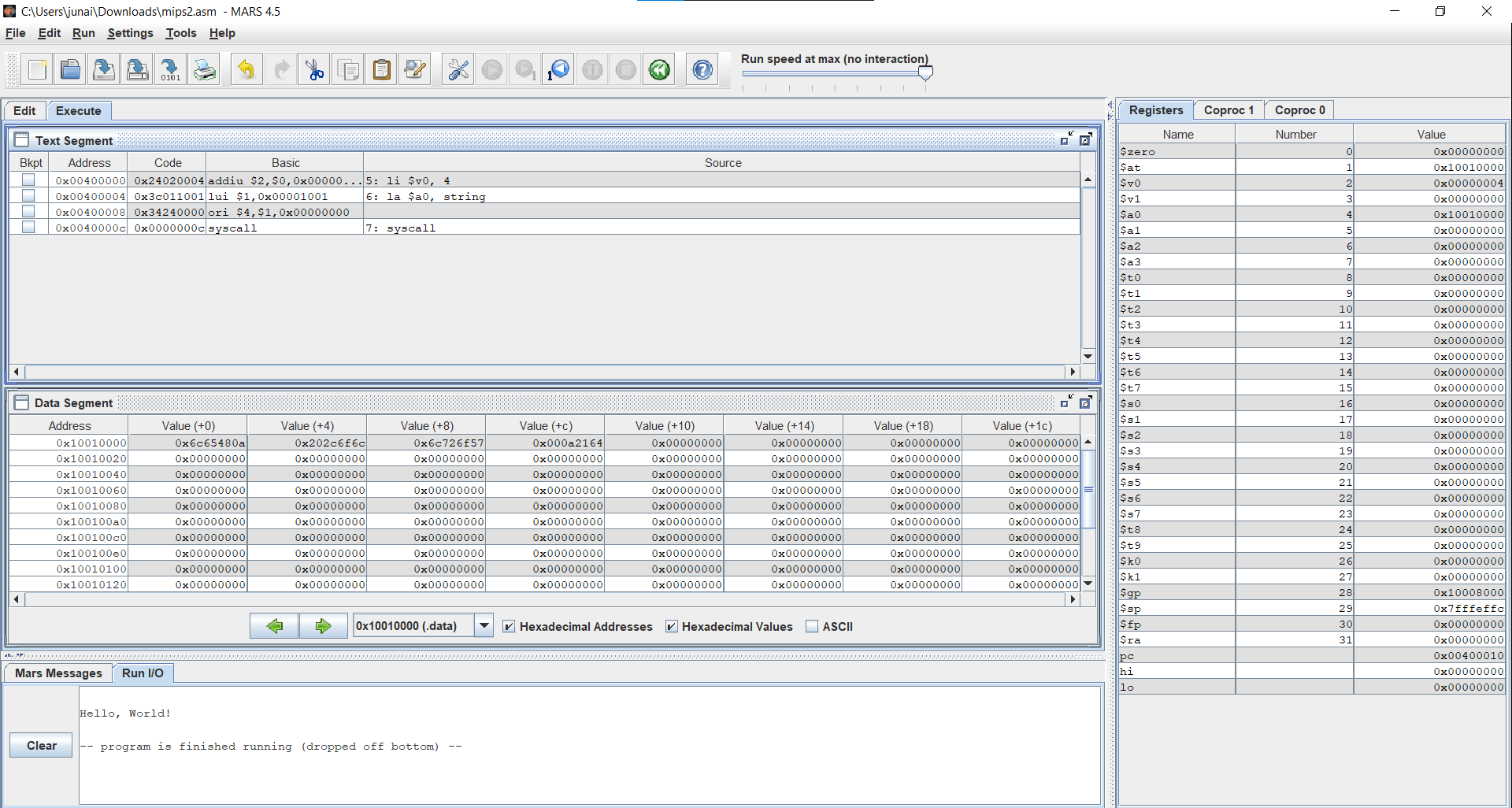
**CODE 1: Addition of 10 numbers**

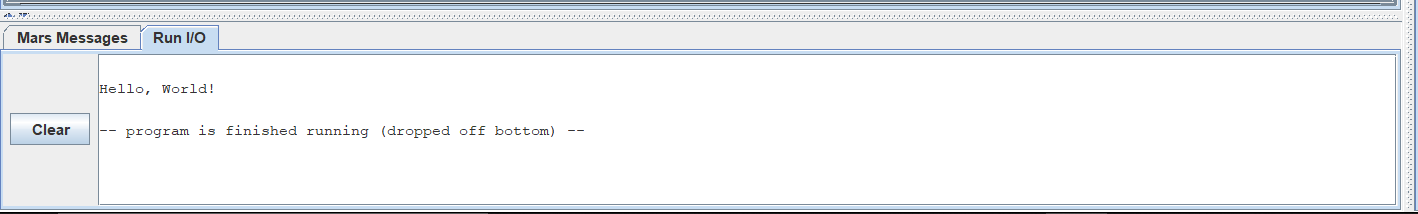
| .data array: .word 5,6,12,4,19,8,4,21,9,1 length: .word 10 sum: .word 0 myMessage: .ascii "Sum of 10 nos. in the given array is: " .text main: la $t0, array # Base address li $t1, 0 # i = 0 lw $t2,length # $t2 = length li $t3, 0 # sum = 0 sumLoop: lw $t4, ($t0) # $t4 = array[i] add $t3, $t3, $t4 # sum = sum + array[i] add $t1, $t1, 1 # i = i + 1 add $t0, $t0, 4 # Updating the array address. blt $t1, $t2, sumLoop sw $t3, sum li $v0, 4 la $a0, myMessage syscall li $v0, 1 move $a0, $t3 syscall |
| --- |



**Code 2: To print Hello World**

| .data string: .asciiz "\nHello, World!\n" .text main: li $v0, 4 la $a0, string syscall |
| --- |





**CODE 3: To Reverse the input string (e.g."ABC" – "CBA").**

| .data  str: .asciiz "India"  str\_msg1: .asciiz "Original string: "  str\_msg2: .asciiz "Reversed string: "  str\_nl: .asciiz "\n"  str\_len: .word 0 .text  main:  #print original string  la $a0,str\_msg1 #leading text  li $v0,4  syscall  la $a0,str #original string  li $v0,4  syscall  la $a0,str\_nl #new Line  li $v0, 4  syscall  #get length  add $t0,$zero,$zero #initialize registers when needed  add $a0,$zero,$zero  add $a1,$zero,$zero  la $a0,str #loads the address of the string  la $a1,str  getLen:  lb $t0,0($a0) #load first byte  beqz $t0,saveLen   addi $a0,$a0,1  j getLen #jump back to start of this loop  saveLen:  subu $t0,$a0,$a1 #len = address of null terminator - str address  sw $t0,str\_len  #reverse the string  add $t0,$zero,$zero #address of the beginning of str  add $t1,$zero,$zero #address of the end of str  add $t2,$zero,$zero   add $t3,$zero,$zero   revString:  #find the index of the last character before the end of the string  la $t0,str #loads the address of the start of the string  lw $t1,str\_len #loads the length of the string  addu $t1,$t0,$t1   subi $t1,$t1,1  loop:  lb $t2,0($t0) #load the first character  lb $t3,0($t1) #load the last character  ble $t1,$t0,printRev  sb $t3,0($t0)  sb $t2,0($t1)   addi $t0,$t0,1  subi $t1,$t1,1 #and loop until we reach the middle  j loop  #print the reversed version of the text  printRev:  add $a0,$zero,$zero #initialize the a0 registry  la $a0,str\_msg2 #leading text  li $v0,4  syscall  la $a0,str #reversed string  li $v0,4  syscall  li $v0,10 #exit program  syscall |
| --- |

