Lab 1

Introduction to MIPS Assembly

Things you will learn in this lab

* Writing simple MIPS assembly programs
* Using Syscall
* Pre-Lab

1. Introduction to MIPS Assembly
   1. Assembly Language

Assembly language is closely related to machine language (sequence of bits). It is in human readable form. ‘Assembler’ is a program which converts the assembly language into the machine language. Every architecture has its unique assembly language.

* 1. MIPS Instruction Set

MIPS supports several instructions all of which may not be mentioned here. However, there are few instructions which form the basis for rest of the instructions. MIPS is a Load/Store architecture, so most of the instructions are register based.

|  |  |
| --- | --- |
| R-Type Instructions  a. ADD $1, $2, $3  b. SUB $1, $2, $3c.  c. AND $1, $2, $3  d. OR $1, $2, $3  g. SLT $1, $2, $3  Memory Based Instructions  a. LW $1, offset($2)  b. SW $1, offset($3) | Jump Instructions  a. J Label  b. JZ Label  d. JR $1  ADD Immediate   1. ADDI $1, $2, value   Pseudoinstruction   1. LI $1, Value ( Any 32bit integer value) 2. Move $1,$2 3. MUL $1, $t2, $t3 |

1.4Registers

The following table explains the 32 registers of MIPS for integer operations.

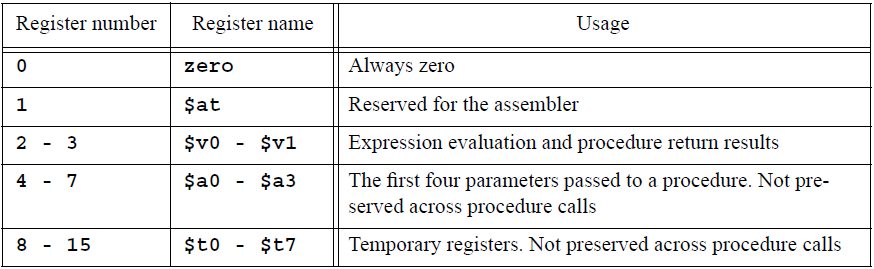




Table 1.1

1. MIPS Assembly Programming using MARS
   1. MARS Simulator

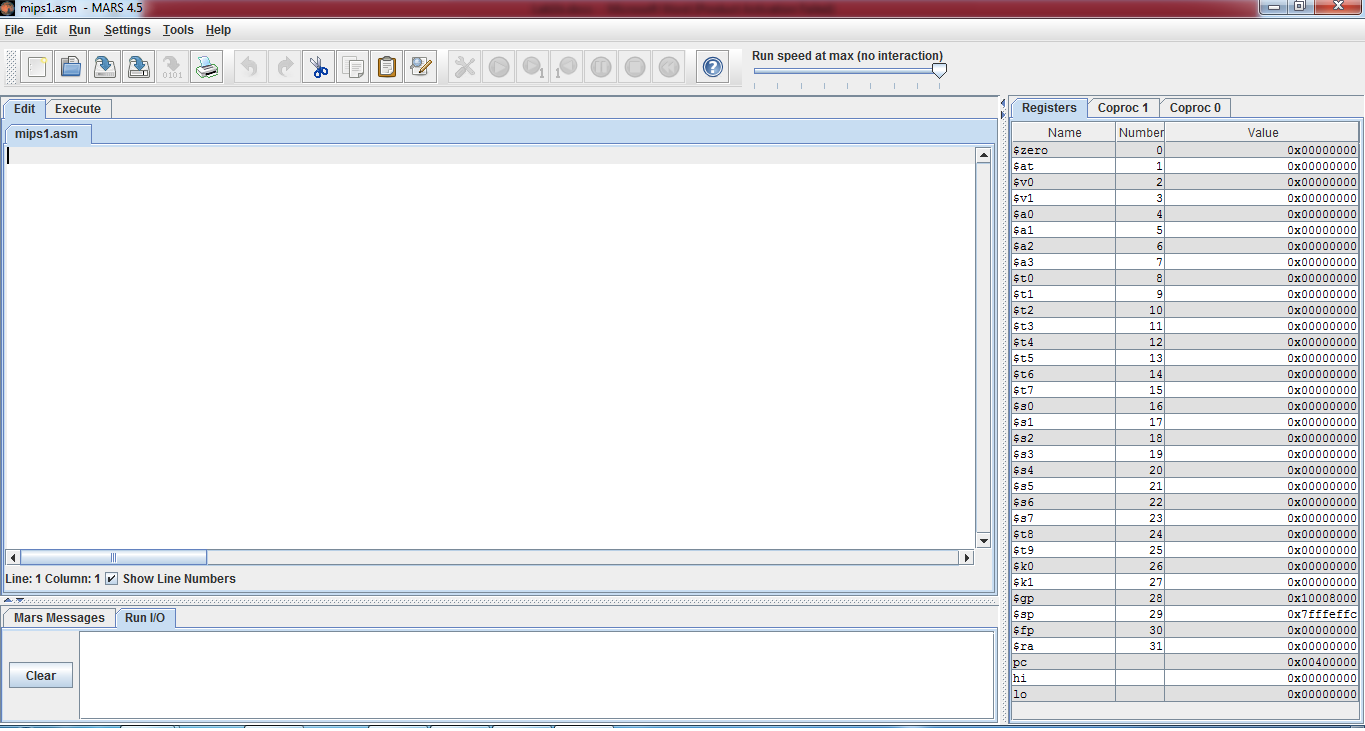
MARS is a simulator for MIPS. It has been already installed on your systems. It requires java runtime environment (JRE) to be installed on your computer. Both MARS and JRE are freely available .

Figure 1.1

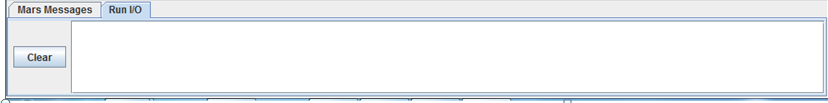


Figure 1.2(I/O Panel)

The side panel shows the contents of the register and the one at the bottom is for input/output.

To begin with MARS, let us write an assembly language program that computes the sum of 1 & 2 and stores the result in $t0. Having ‘comments’ in a code is always helpful for others in understanding what the code does. So, make sure you always put all the required comments in your code and use ‘#’ sign to begin your comments.

Open new file and write the folloeing code in it.

# Your Name Reg No Date

# add.asm-- A program that computes the sum of 1 and 2,

# leaving the result in register $t0.

# Registers used:

# t0 - used to hold the result.

# t1 - used to hold the constant 1.

li $t1, 1 # load 1 into $t1.

addi $t0, $t1, 2 # $t0 = $t1 + 2.

#see the value in register panel

# end of add.asm

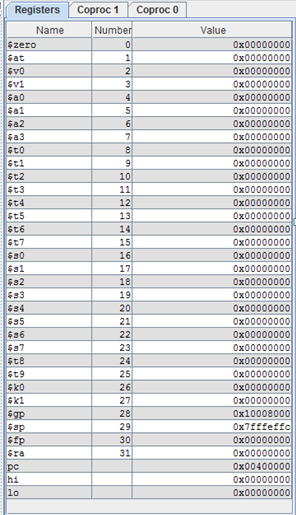


Figure 1.3 (Registers)

* 1. Completing the Program

These two instructions perform the calculation that we want, but they do not form a complete program. Much like C, an assembly language program must contain some additional information that tells the assembler where the program begins and ends. The exact form of this information varies from assembler to assembler (note that there may be more than one assembler for a given architecture, and there are several for the MIPS architecture)

* + 1. Labels

Labels are used to tag the address of a particular instruction. If it is required to jump back (for implementing loops) to a particular instruction, or jump forward (for implementing if -else); these are mentioned in the jump instructions

# Your Name Reg No Date

# add.asm-- A program that computes the sum of 1 and 2,

# leaving the result in register $t0.

# Registers used:

# t0 - used to hold the result.

# t1 - used to hold the constant 1.

main:

li $t1, 1 # load 1 into $t1.

addi $t0, $t1, 2 # $t0 = $t1 + 2

Note that the MARS assembler does not permit the names of instructions to be used as labels. Therefore, a label named ‘add’ is not allowed, since there is an instruction of the same name.

* + 1. Syscalls

Since you will practice MIPS assembly programming on a MIPS simulator which runs on different processor (Intel, most probably), running an operating system ( Windows); you can neither get input yourself from the user nor display result on screen.

Syscall is used to get the job done in this matter. Before using syscall, system call code should be moved to register v0 as listed in second column of Table 1.2. The table describes the arguments required and result registers.

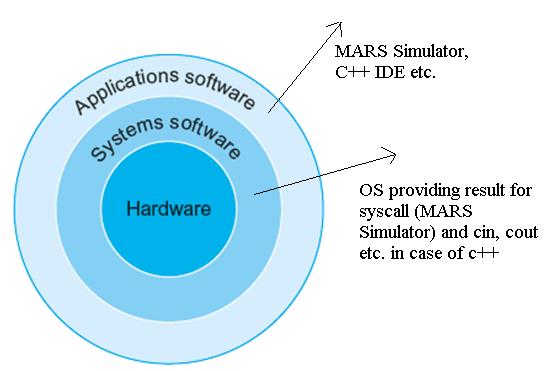


Figure 1.4

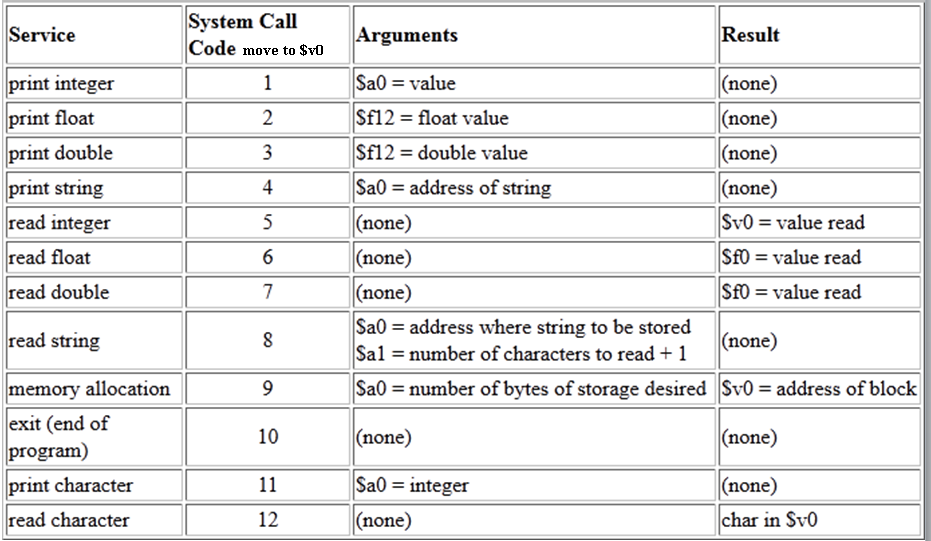


Table 1.2

Getting input from user

li $v0, 5 # Call code for input is 5

syscall # make the syscall.

move $t0, $v0 # Value is read in $v0 by syscall

# Moving the value to register $t0

Displaying value in $t4

move $a0, $t4 # Because argument should be in $a0

li $v0, 1 # Call code for displaying integer is 1

syscall # make the syscall.

Exit from execution at the end of code

li $v0, 10 # syscall code 10 is for exit.

syscall # make the syscall.

* In-Lab
  + 1. Example code

*# Your Name Reg No. Date*

***# add2.asm-- A program that computes and prints the sum***

***# of two numbers specified at runtime by the user.***

*# Registers used:*

*# $t0 - used to hold the first number.*

*# $t1 - used to hold the second number.*

*# $t2 - used to hold the sum of the $t1 and $t2.*

*# $v0 - syscall parameter and return value.*

*# $a0 - syscall parameter.*

*main:*

*## Get first number from user, put into $t0.*

*li $v0, 5 # load syscall read\_int into $v0.*

*syscall # make the syscall.*

*move $t0, $v0 # move the number read into $t0.*

*## Get second number from user, put into $t1.*

*li $v0, 5 # load syscall read\_int into $v0.*

*syscall # make the syscall.*

*move $t1, $v0 # move the number read into $t1.*

*add $t2, $t0, $t1 # compute the sum.*

*## Print out $t2.*

*move $a0, $t2 # move the number to print into $a0.*

*li $v0, 1 # load syscall print\_int into $v0.*

*syscall # make the syscall*

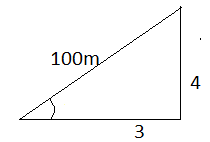
*li $v0, 10 # syscall code 10 is for exit.*

*syscall # make the syscall.*

*# end of add2.asm.*

**Tasks:**

* Write a program which inputs the bill amount and the amount paid by the customer. It calculates the amount to be returned to the customer and displays it.
* A store is offering 20% discount on all purchases. Write a program which inputs total purchase bill in rupees, total amount paid by the customer and calculates the amount to be returned to the customer.
* Write a program that inputs Marks of QA (out of 25), Sessional1 (out of 10), Sessional2 (out of 15), Terminal (out of 50); then calculates and displays total marks.
* Post-Lab
* Write a program that calculates the lengths of base and perpendicular for the following triangle where the ratio of length of perpendicular to base is 4:3



.

**Note:**

* **Your lab report, a .doc file, should contain properly commented Post-Lab task code, MARS screenshot for each code which clearly show the register contents after execution and I/O panel, and critical analysis.**
* **The report must have a title page.**
* **Name the .doc file RegNo.docx; eg SP14-BCE-99.docx**
* **Upload it on portal**
* **Deadline for Lab Report submission is before start of the next Lab.**