Virtual Campus, COMSATS Institute of Information and Technology

**COMPUTER ORGANIZATION**

**&**

**ASSEMBLY LANGUAGE**

**LAB # 2**

**Introduction to Assembler (PCSpim)**



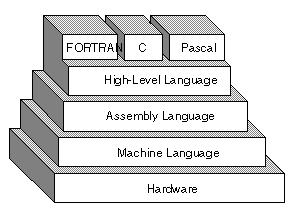
**What is Assembly?**

An assembly language is a low-level programming language for computers, microprocessors

(µP), microcontrollers (µC), DSP and other programmable devices in which each statement corresponds to a single machine language instruction. An assembly language is thus specific to certain physical (or virtual) computer architecture, in contrast to most high-level programming languages, which, ideally, are portable.

Assembly language allows the use of symbolic representation for machine operation codes (usually called **mnemonics**), memory locations, registers and other parts of an instruction.

A utility program called an assembler is used to translate assembly language statements into the target computer's machine code.



Machine languages consist entirely of numbers and are almost impossible for humans to read and write. Assembly languages have the same structure and set of commands as machine languages, but they enable a programmer to use names instead of numbers.

Each type of CPU has its own machine language and assembly language, so an assembly

language program written for one type of CPU won't run on another. In the early days of programming, all programs were written in assembly language. Now, most programs are written in a high-level language such as JAVA or C.

Programmers still use assembly language when speed is essential or when they need to perform an operation that isn't possible in a high-level language for example hardware access.

**Why Assembly?**

* Makes you better programmer
* Used with real time system
* Talk directly to hardware
* DLL (Device Drivers) written in assembly language
* Take best out of your µP/µC
* Performance of the languages like java and C/C++ can be improve as the compilers are written in assembly and some part of JVM
* Faster than High Level Languages HLL
* Take less space in memory and Hard disk than HLL codes

**ASM code is not portable**

Assembly is platform dependent (OS/CPU Architecture or Hardware) .

**C Code:** **Assembly Code**

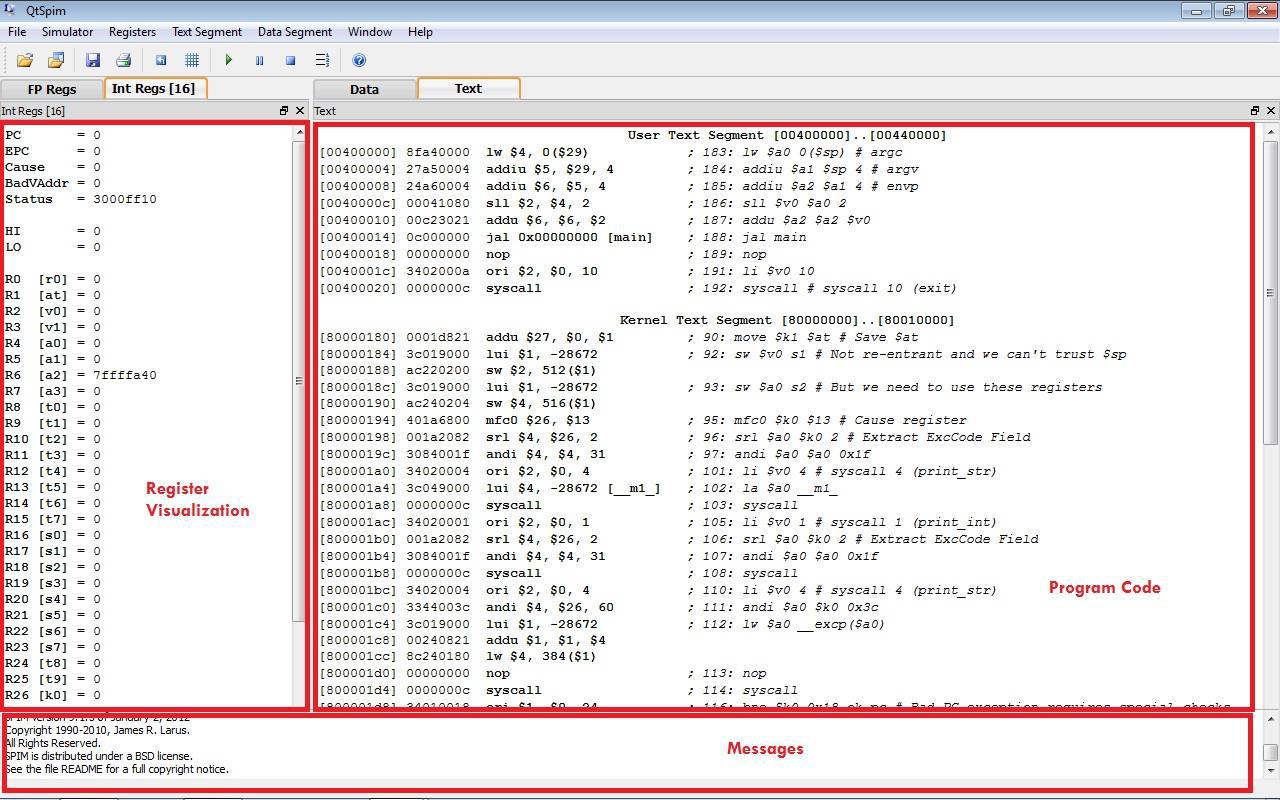
**A=b+c** **add a , b , c**

**Assembler**

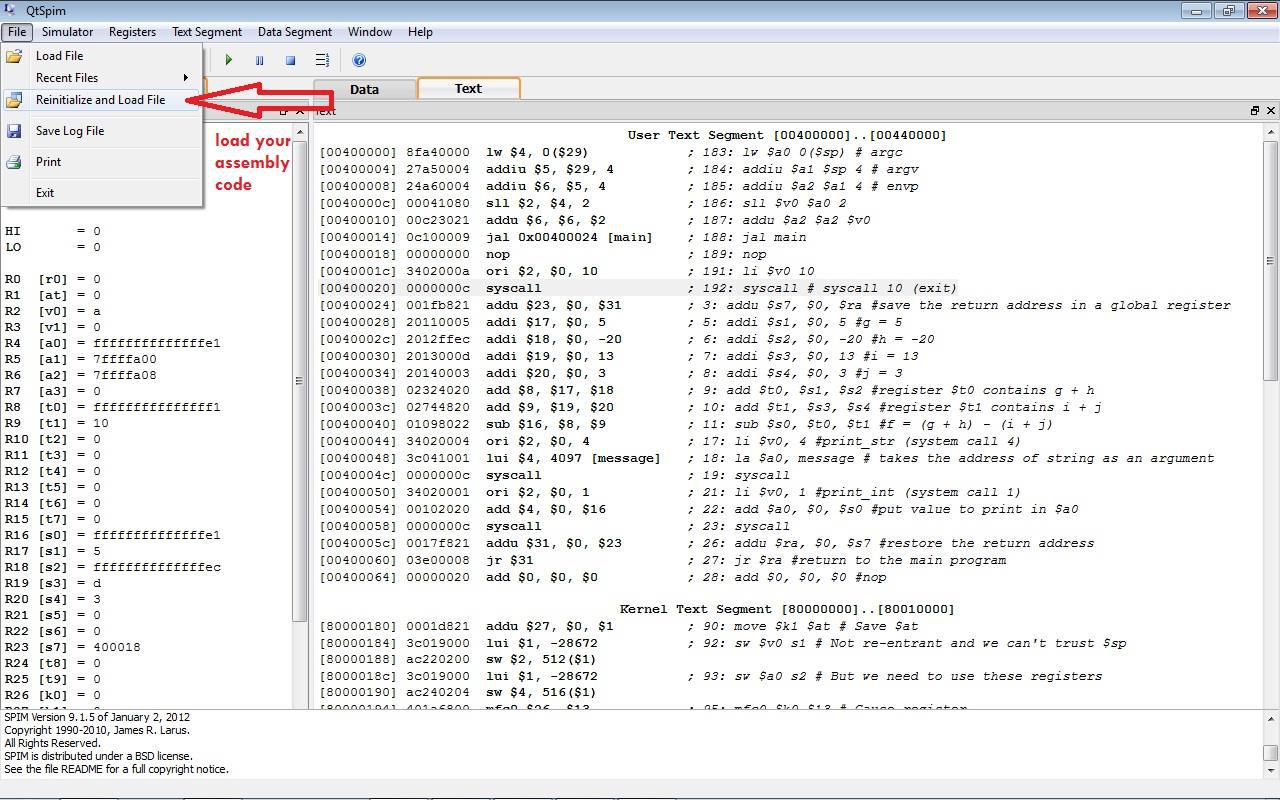
A  [program](http://www.webopedia.com/TERM/P/program.html) that translates programs from  [assembly language](http://www.webopedia.com/TERM/A/assembly_language.html) to  [machine languag](http://www.webopedia.com/TERM/M/machine_language.html)e Many assembler are available like nasm , masm , masm32 , tasm and MIPS etc For this lab we will use MIPS

**QtSpim**

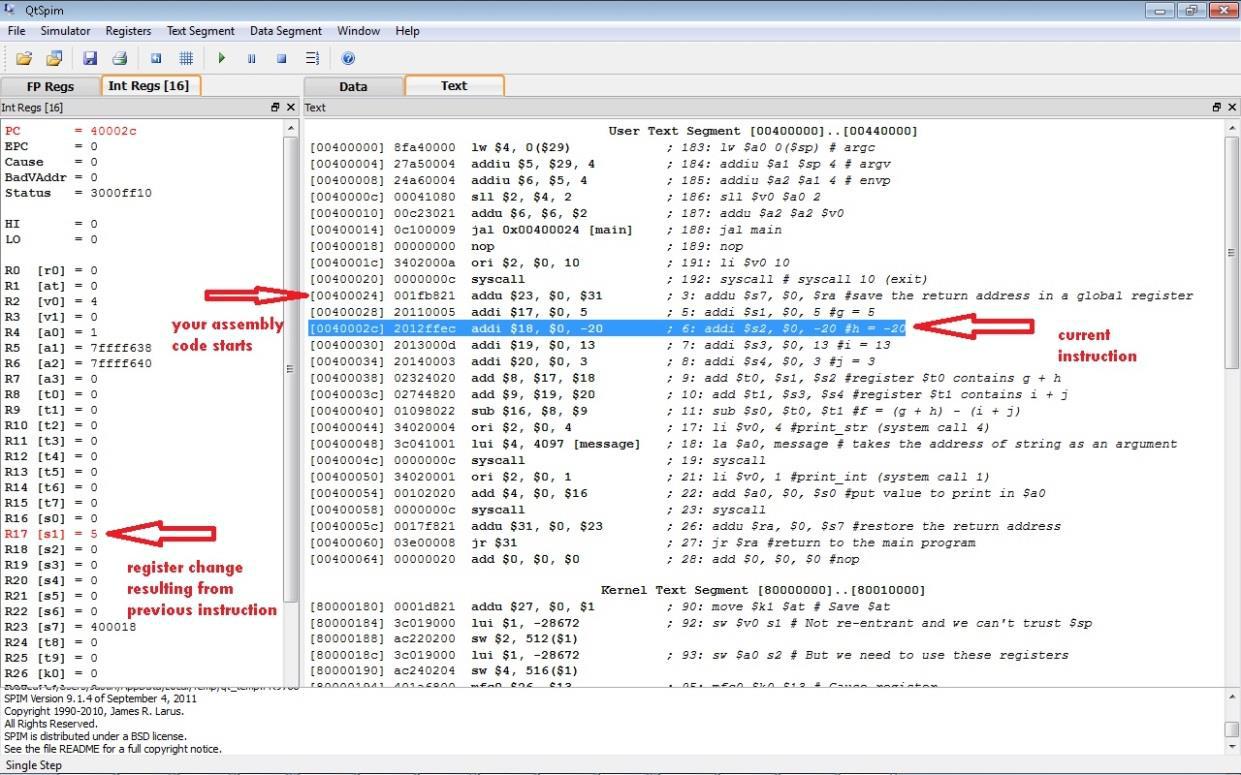
**Start Qtspim**



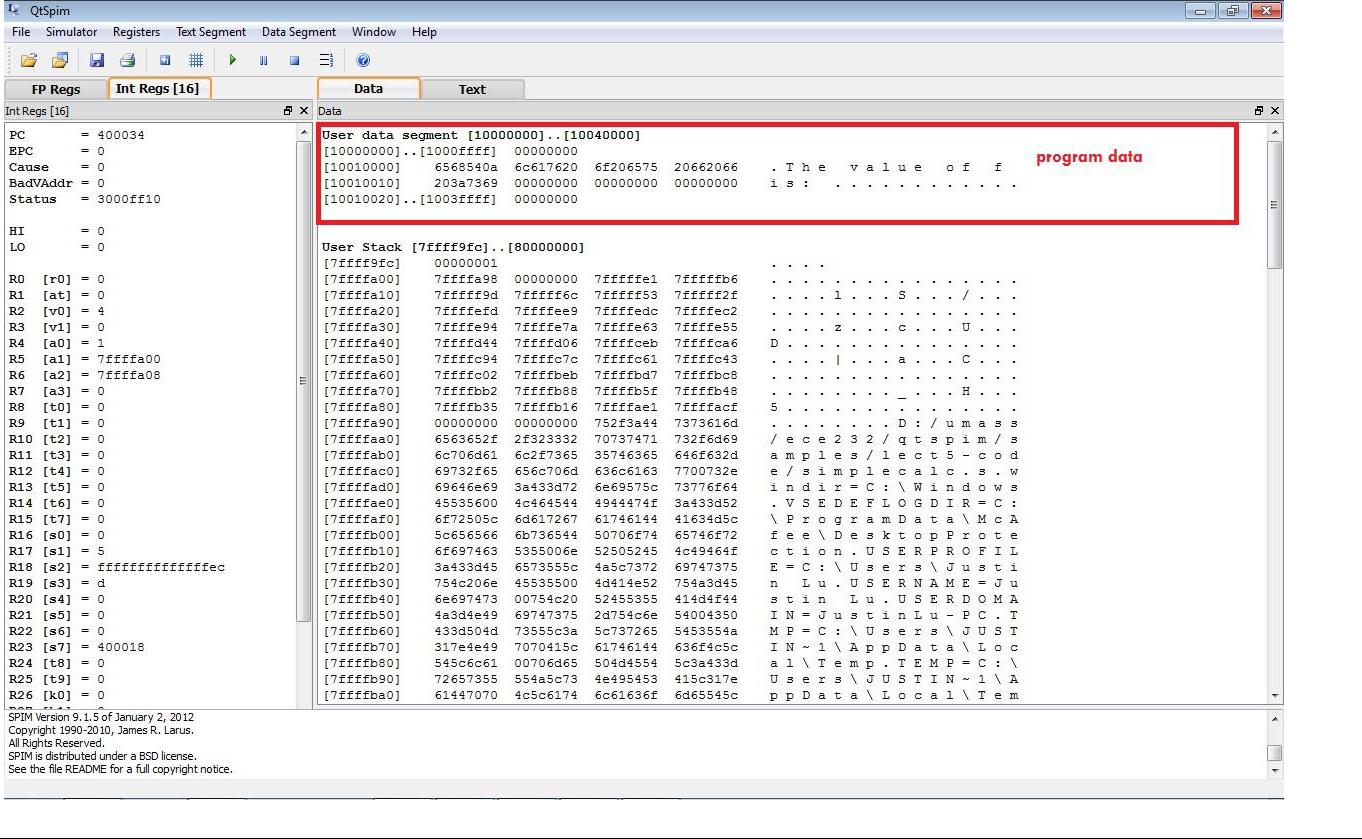
**Load Program:**



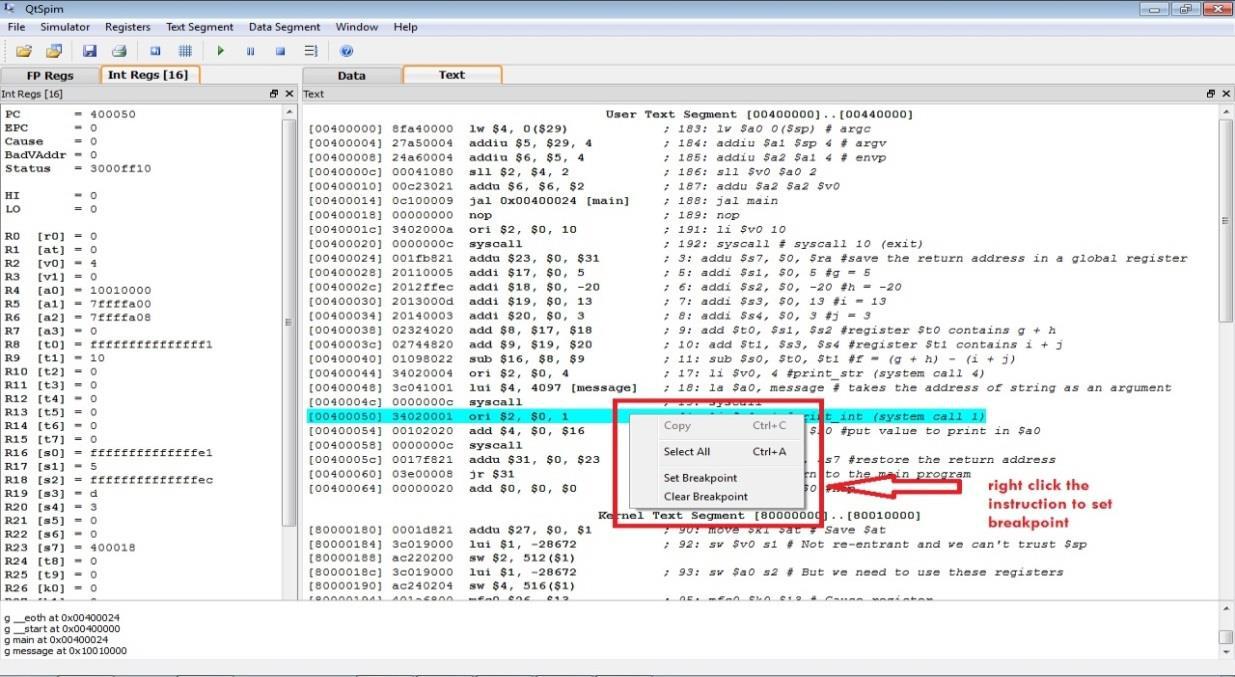
**Execute Program**



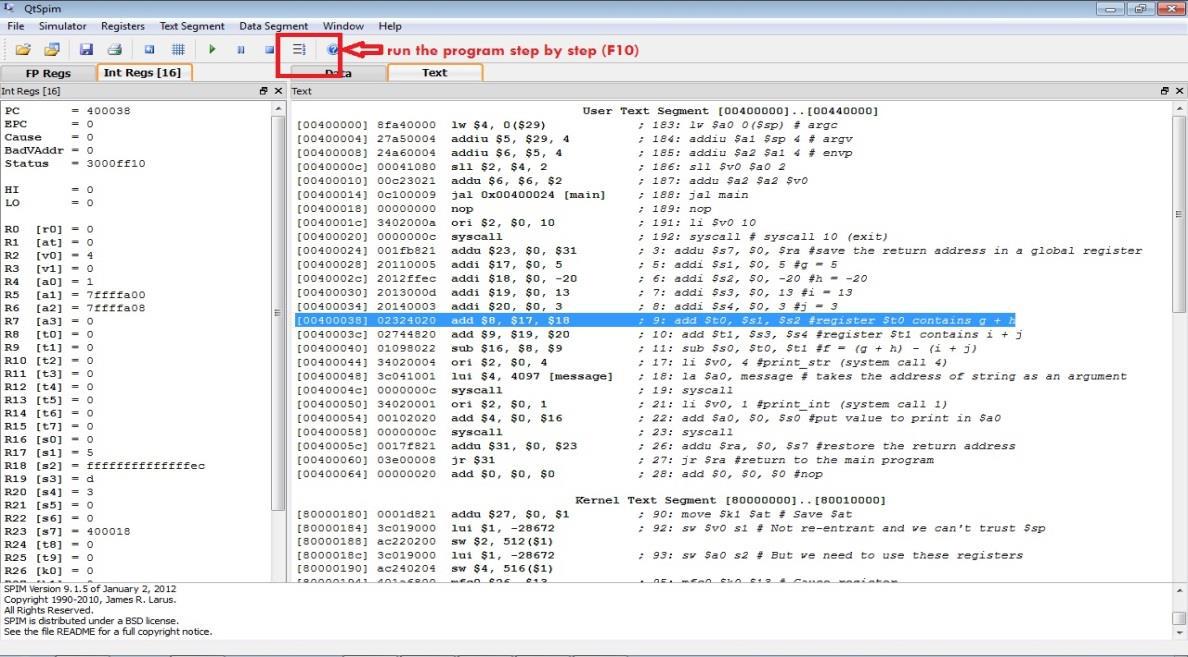
**Program data:**



**Set a Breakpoint**



**Debug by stepping line by line**



**Basics Of Mips labels**

A *label* is something to make your life simple. When you reference a piece of your program, instead of having to count lines, you can just give it a name. You use this in loops, jumps, and variable names.

Labels don't appear in your final code, they're only there for convenience, one of the few perks you'll get from the typical MIPS assembler. It also makes life easy for the assembler, because it can now easily go around relocating and linking code. Don't worry if you don't know what those are, that'll come later.

**Name:**

( Note that, you can't create a label with the same name as a MIPS instruction.)

**assembler directives:**

These are a subset of the MIPS assembler directives that are used to tell the assembler something about names that are used in a MIPS assembly file:

.text # the succeeding lines contain instructions

.data # the succeeding lines contain data

.globl name # name is global symbol (visible to code in other files)

.asciiz "a string\n" # stores a null terminated string in memory

**Program Structure**

* just plain text file with data declarations, program code (name of file should end in suffix .s to be used with SPIM simulator)
* data declaration section followed by program code section

**Data Declarations**

* placed in section of program identified with assembler directive **.data**
* declares variable names used in program; storage allocated in main memory (RAM)

**Code**

* placed in section of text identified with assembler directive **.text**
* contains program code (instructions)



* starting point for code given label **main:**
* ending point of main code should use exit system call.

**Comments**

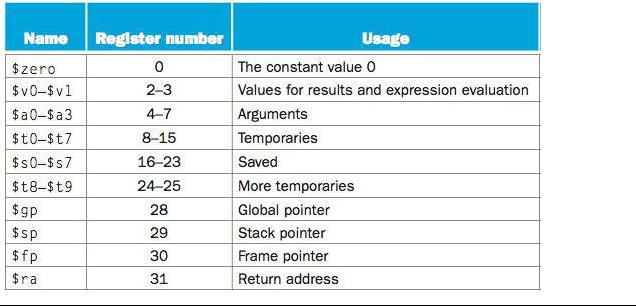
anything following # on a line

* This stuff would be considered a comment Template for a MIPS assembly language program:
* Comment giving name of program and description of function
* Template.s
* outline of MIPS assembly language program

|  |  |  |  |
| --- | --- | --- | --- |
| .data | # | variable declarations follow this line | |
|  | # ... | |  |
| .text | # | instructions follow | this line |
| main: | # indicates start of code | | (first instruction to execute) |

* + ...
* End of program, leave a blank line afterwards to make SPIM happy

**Register in MIPS Assembly Language**



**Instruction Format:**

There are 3 different types of instructions: R Instructions, I Instructions, and J Instructions. We will study these in details in coming labs.

**Arithmetic Instructions: add**

|  |  |  |
| --- | --- | --- |
| add rd, rs, rt | #rd = rs + rt | |
| **sub** |  |  |
| sub rd, rs, rt | # rd = rs - | rt |
| **mult** |  |  |
| mult rd, rs, rt | # rd = rs \* | rt |
| **div** |  |  |
| div rd, rs, rt | # rd = rs \* | rt |