CS 21 Laboratory 1: SPIM Familiarization and Our First Assembly Language Program

University of the Philippines - Diliman
College of Engineering
Department of Computer Science

1 Launching the SPIM application

1. Bring up the Unix console
2. Make sure that you are in the home directory (using the **cd** command). Create a folder named after your last name using the **mkdir** command. Go to that folder.
3. Type "xspim" (sans the quotation marks) and press the ENTER key

A window like the one in Figure 1 should appear.

Notice that there are three subwindows.
The topmost subwindow shows the values contained in the processor's registers. Registers are the processor's local memory spaces.
MIPS is a 32-bit processor, therefore, each register has 32 bits.

2 Our first assembly language program

Using a text editor(vi or Kate or whatever), create a file named "myfirstprogram.asm". Make sure to save it in your folder.

2.1 Writing comments

Comments are very important when programming in assembly language. Assembly language programs are much harder to debug (especially if the one doing the debugging is NOT the programmer) than high-level language programs, hence the need for more thorough documentation.

Any text between a **pound sign (#)** and the next newline is considered a comment.

Code Block 1 Comments in an assembly language

```
# CS 21 THXY -- S1 AY10-11
# Wilson M. Tan -- 03/08/10
# myfirstprogram.asm -- a simple program where we learn the uber basics
```

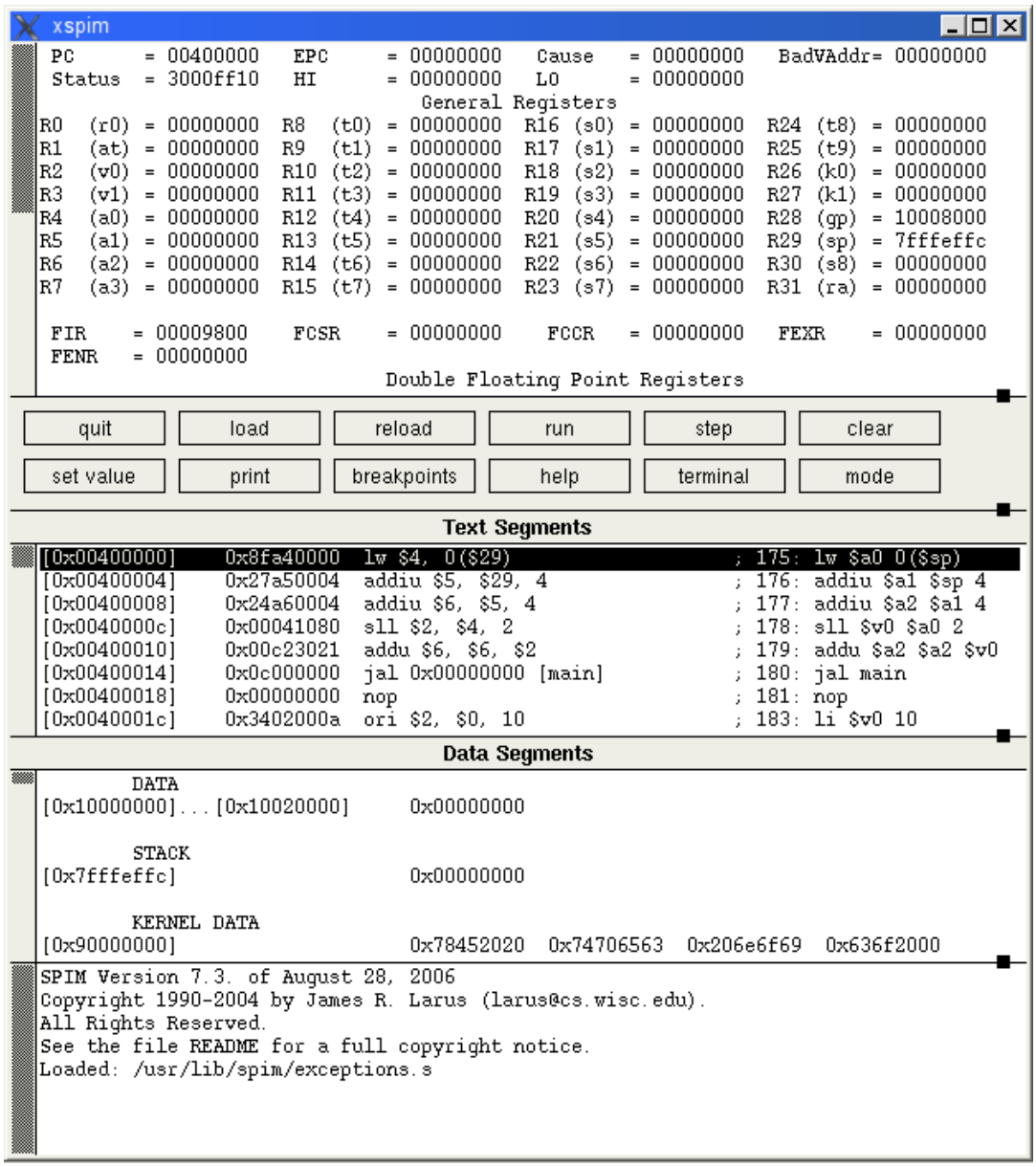

Figure 1: The xspim window.

Code Block 1 contains comments that may appear in the "header" or beginning of an assembly language program. It contains the subject name, section, current semester, academic year, name of the programmer, date submitted, and a simple description of the program. **Such a "header comment block" would be required for all submittable codes (machine exercises and machine problems) so better familiarize yourself with it.**

2.2 4 column format

As a matter of convention, we would be using a 4 column format for our programs. This would facilitate clarity in our code. For an example, refer to Code Block 2.

Code Block 2 A code line in the 4 column format.

```
main:       li      $t1, 1      # load 1 into $t1.
```

The first column contains **labels**. Labels are symbolic names for addresses in memory. It makes it easier for us to refer to a specific portion or line from the memory. In Code Block 2, the first column contains the label "main". Labels must be followed by ":". All programs must contain the label "main", since it signifies the first instruction which the processor must execute. Take note that it is not required for all lines or instructions to have a label.

The second and third columns are dedicated to the **instruction itself**. We separate it into the operation and the operands. In Code Block 2, the second column contains **li** (shorthand for load immediate), while the third column contains **$t1, 1**.

The fourth column contains **comments** regarding the instruction.

2.3 Text and data segments

The memory we would be simulating contains two segments: **text** and **data**. The text segment contains our instructions for execution. The data segment contains data.

We could modify both using our assembly language program, but we have to tell spim which goes where. Thus, we invoke the directives ".text" and ".data". As a matter of convention we define the text segment first before the data segment. An incomplete skeletal format for our programs could thus be seen in Code Block 3.

2.4 Ending programs

We need the label "main" to tell the processor which part of the memory contains the instructions we want it to execute. *Could it know on its own when to stop? No, it can't.*

Just like in C where "return 0" indicates the end of the main function, a program's end in assembly language must also be explicitly stated. For this purpose we use "syscall code 10". Syscalls would be explained in another laboratory session. For now, simply remember that all programs must end with "syscall code 10". How do we invoke syscall

Code Block 3 Incomplete skeletal program.

```
# CS 21 THXY -- S1 AY10-11
# Wilson M. Tan -- 03/08/10
# myfirstprogram.asm -- a simple program where we learn the uber basics
.text

main: ######################
##############instructions##############
##############instructions##############

.data
##############data##############
##############data##############
```

code 10? We use two instructions in succession: **li $v0, 10** followed by **syscall**. With this, we could now complete our skeletal program. See Code Block 4.

Code Block 4 Complete skeletal program.

```
# CS 21 THXY -- S1 AY10-11
# Wilson M. Tan -- 03/08/10
# myfirstprogram.asm -- a simple program where we learn the uber basics
.text

main: ######################
##############instructions##############
##############instructions##############
li $v0, 10 #syscall code 10
syscall #syscall code 10
.data
##############data##############
##############data##############
```

Now all that is missing are the actual instructions.

2.5 The Load Immediate Instruction

The **li** or load immediate instruction is the main instruction in the MIPS ISA used in directly loading values to the registers. Its format is **li** followed by the target or recipient register, and then the value to be loaded to that register. Notice that in SPIM, we refer to registers by preceding them with the dollar ($) sign. Thus, to load the value "1" to register t1, we issue the instruction **li $t1, 1**. Used in the context of an entire program, we refer to Code Block 5.

3 Running our program

Make sure that the file is saved in the home directory. In the XSPIM window, press the "load" button and type "myfirstprogram.asm". Press the "run" button. What happens to the R9 (t1) value in the topmost subwindow?

Code Block 5 Complete program.

```
# CS 21 THXY -- S1 AY10-11
# Wilson M. Tan -- 03/08/10
# myfirstprogram.asm -- a simple program where we learn the uber basics
.text

main: li $t1, 1 #load 1 to register 1
li $v0, 10 #syscall code 10
syscall #syscall code 10

.data
###############data##############
###############data##############
```

Things to do:

1. Try loading other values to register t1.
2. Using the code from the previous item, instead of using the "run" option, use the "step" option with number of steps set to 1. Press the "step" button repeatedly while observing what is highlighted in the middle subwindow and the values at the topmost subwindow. Questions:
   - What is the step function for? What could be its practical use?
   - True or false - the instructions that we wrote or specified are ALL that the processor executes.

4 Machine Exercise

1. Load the value 9 to t1.
2. What does *li $t2, 38* do? Write complete answer on 1/2 sheet of paper.
3. Perform *li $t3, -421*. What is the value stored in t3?
4. Make the value of t4 appear as 00001985 in the subwindow.
5. Make the value of t5 appear as ABCD0000 in the subwindow.

5 Learning Checklist

In this session you should have learned...

- how to launch the SPIM/XSPIM application
- how to "force" register values via SPIM
- the basic structure of an assembly language program
- the usage of the **li** instruction
- how to run programs in SPIM, both continuously and step-by-step