CS 232 – Computer Architecture I
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department

Lab 1 — SPIM

## 1 Objectives

Following completion of this lab you should be able to:

- Determine the value stored in a register in SPIM.
- Determine the contents of memory in SPIM.
- Determine the contents of the stack in SPIM.
- Determine the location and contents of static data in SPIM.
- Run MIPS programs using SPIM.
- Set break points and use them for debugging in SPIM.
- Use the pseudo-instructions `li` (Load an Immediate value into a register) and `move` (copy a register) in a program.
- Use the R-type instructions `add` and `sub` in a program.
- Use the I-type instructions `addi`, `lw` and `sw` in a program.
- Use the SPIM system calls for printing integers, reading integers, and exiting.
- Read, modify, and write a program in MIPS assembly language and document it properly.

## 2 General

- Your grade for this lab will be based on the files that you turn in electronically as well as your answers to some questions.
- You must remain in lab until you complete the lab exercise or the end of the lab period.
- Bring your text book to labs.
- Read the lab instructions completely before beginning.
- Don't hesitate to ask for help.

## 3 Installing SPIM

SPIM is available for MS Windows[1] and dos. The source code is also available if you'd like to run it on another architecture or OS. Both general and windoze specific documentation are available. Appendix A of your book is also available as a pdf

[1]Hereafter referred to as windoze.

**You don't want to use this version!**
The latest version of SPIM can be obtained from ftp://ftp.cs.wisc.edu/pub/spim/, however it does not manage the symbol table properly and global symbols are not included. The result of this is that you cannot set breakpoints to global symbols. The source code is available for the windoze version and **BONUS POINTS** will be awarded to anyone who fixes the symbol table bug.
**You don't want to use this version!**

Download and install SPIM. You are welcome to use any OS you like. However, I will assume that if you are not using windoze, you know what you are doing. The following steps are aimed at windoze users.

- Download the windoze version of SPIM to a convenient place on your laptop.
- The windoze version is a self extracting zip. Simply double-click to install it.
- PCSPIM (the windoze version of SPIM) can be launched by double-clicking it.

## 4 Running a Program

Examine the file `p1.asm` using your favorite editor. It contains a complete program written in MIPS assembly language. It simply places three numbers into three registers, then exits. You may need to use "binary transfer mode" to download `p1.asm` so that proper line termination is preserved.

To run program `pl.asm`, double-click PCSPIM. PCSPIM uses a point-and-click interface: you just click on the commands to make SPIM do what you want. The commands are summarized in windoze specific documentation listed in Section **??**.

Once you are in SPIM, do the following:

1. Click *File:Open* and select `p1.asm` to load the file. If there are syntax errors, messages will appear at this point. If you get an error on the first non-comment line, there is likely a line termination problem (CR-LF vs LF). If you get an error on the last line of the file, the last line is probably missing the final newline. Correct the program and reload it using *Simulator:Reload*. Occasionally, it may be necessary to completely exit PCSPIM and restart following errors.
2. Click *Simulator:Breakpoints...*, enter `main`, and click *Add* and then *Close* to set a breakpoint at the label main. We want to skip some initialization code that is executed before our program runs.
3. Click *Simulator:Go*, *Ok*, and *No* to run the simulator upto the breakpoint.
4. Press F10 three times to step through the first three lines of the program.
5. Click *Window:Registers* to display register values. Are the values what you expect? Be sure that you **understand** and can explain what is going on.
6. Click *File:Exit* to exit SPIM.

## 5 Writing a Program

Copy the code in file `p1.asm` into `p2.asm`[2]. Modify `p2.asm` to compute 27 + 380 - 401 and store the result in $5 – you simply need to add two more lines of code. Load `p2.asm` into SPIM and test it.

## 6 Register Conventions

As we will see in the future labs, there are certain conventions for register use. To assist this use, MIPS permits registers to be referred to by symbolic names. For example, register $v0 was used in program `p1.asm`. These conventions will be presented in class. For now, we will restrict ourselves to using the eight registers labeled $t0, $tl, $t2 ... $t7. The MIPS registers are summarized on page A-23 of your text.

Copy the code from file `p2.asm` into `p3.asm`. Convert all the numbered registers to t-registers. Load `p3.asm` into SPIM and test it.

## 7 Register Use

Registers are a precious commodity—they are fast, but few are available. Therefore, we must minimize our use of registers.

Copy the code from file `p3.asm` into `p4.asm`. Modify `p4.asm` to use as few registers as possible. Do not modify the last two lines. These perform the system call to exit the program. Use only t-registers, but as few as possible. Load `p4.asm` into SPIM and test it.

## 8 Printing Integers

SPIM provides a small set system calls[3] to make input and output easier. For example, there is a mechanism for printing an integer. The system call code is placed into register $v0 and then the syscall pseudo-instruction is executed. The integer to be printed is stored in $a0. For example, the following MIPS code will print the integer in register $t3.

```
move $a0, $t3           # Coy the value of $t3 into $a0
li   $v0, 1             # 1 is the system call code ...
syscall                 # ... for printing an integer.
```

Copy the code from file `p4.asm` into `p5.asm`. Modify `p5.asm` to print the value computed. Load `p5.asm` into SPIM and test it, but this time without a breakpoint and without stepping - just

[2] `.asm` files require a newline character after every line of code, including the last one. This brittle aspect of the parser usually shows up when copy/pasting code.

[3] See page A-49 of your text for a complete list of system calls.

*Simulator:Go* (F5). Click *Window:Console* and you should see the answer (6). We will make the output prettier in the next lab when we see how to print strings.

## 9 Reading Integers

SPIM also provides a mechanism for reading an integer. Use system call code 5, instead of code 1. When the syscall is executed, the system will pause for input. The integer you type (terminated with a RETURN) will be placed into register $v0. You can move that value into whatever register you choose. For example the following code reads and integer and stores it in register $t0.

```
li   $v0, 5              # 5 is the system call code ...
syscall                  # ... for reading an integer.
move $t0, $v0            # Copy the value read to $t3
```

Copy the code from file `p5.asm` into `p6.asm`. Modify `p6.asm` to read[4] two integers and print their sum. Load `p6.asm` into SPIM and test it without a breakpoint and without stepping.

## 10 Memory Access

Download and carefully study `main.asm`. Note the coding style and documentation. From this lab forward, you will be expected to follow a similar coding style and to similarly document all your work. This program illustrates the following concepts:

1. The data associated with a MIPS program is put into a data segment (preceded by `.data`) and the executable statements are placed in a code segment (preceded by `.text`).
2. `.word` and `.asciiz` are assembler directives. See page A-51 and A-52 of your text for explanations. See also page A-14 and A-15 for a more in-depth explanation of the `.asciiz` directive.
3. The instruction `la` loads an address into a register. When using this instruction, you will supply a symbolic address (e.g. V:).
4. The instruction `sll $t0, $t0, 2` multiplies the contents of register $t0 by four. `sll` stands for shift left logical. Verify that shifting the contents of a register twice to the left (the new locations vacated to the right are filled in with 0's) indeed multiplies its contents by four.
5. You can supply a symbolic base address when using the `lw` and `sw` instructions.

Load and run `main.asm`. Does it behave correctly?

[4]There is no prompt for the two inputs. You must type those inputs on two separate lines.

Comment out the instruction `sll $t0, $t0, 2`. Load and run the program again. Does it behave correctly? Make sure that you **understand** the behavior of the program in these two cases.

Modify the program to do the following:

1. Prompt the user for an integer i between 1 and 10.
2. Print out the ith element of the array[5] V.

## 11 Turning It In

When the lab is complete turnin the answers to the lab questions and upload `p6.asm` and `main.asm` to the *afs* file system at:

`/Class/cs/cs232/turnin/`<*section*>`/`<*username*>`/lab1/`.

You will need your *afs* password and *kerberos*, the `afs client` and *aklog* from the new laptop suite. You

[5]It is important to note that our arrays are zero-based. V(2) refers to the third element of the array V, not the second. However when 2 is supplied as input, the second element is expected.