

CENG3420

Lab 1-1: MIPS assembly language programming

Haoyu Yang

Department of Computer Science and Engineering
The Chinese University of Hong Kong

hyyang@cse.cuhk.edu.hk

Spring 2019



香港中文大學
The Chinese University of Hong Kong

Overview

SPIM

Assembly Programing

System Service in SPIM

Lab Assignment



Overview

SPIM

Assembly Programing

System Service in SPIM

Lab Assignment



What is SPIM

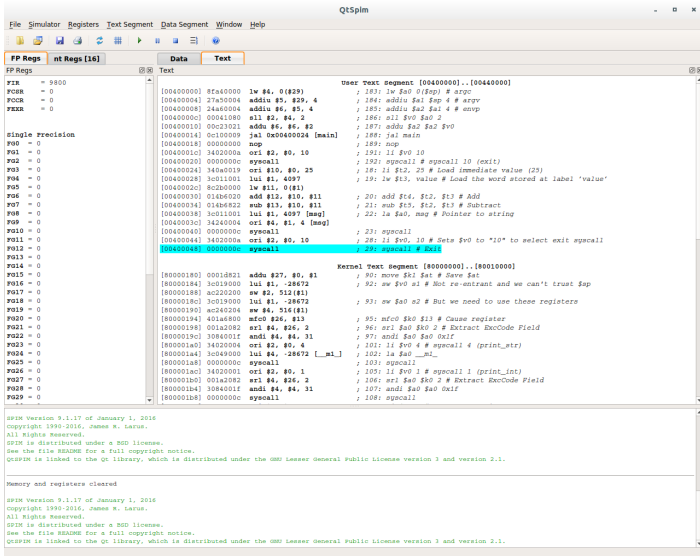
- ▶ **SPIM is a MIPS32 simulator.**
- ▶ *Spim* is a self-contained simulator that runs MIPS32 programs.
- ▶ It reads and executes assembly language programs written for this processor.
- ▶ *Spim* also provides a simple debugger and minimal set of operating system services.
- ▶ *Spim* does not execute binary (compiled) programs.

Download it here:

<http://sourceforge.net/projects/spimsimulator/files/>



SPIM Overview



The screenshot shows the SPIM simulator window titled "Qt5pim". The interface includes a menu bar (File, Simulator, Registers, Text Segment, Data Segment, Window, Help) and a toolbar. The main window is divided into several panes:

- FP Regs:** Lists floating-point registers (F0-F12) and their values, mostly zero.
- nt Regs [16]:** Lists integer registers (R0-R15) and their values, mostly zero.
- Data:** Shows memory data segments.
- Text:** Displays assembly code for the "user text segment" and "Kernel Text Segment".

The assembly code includes instructions such as `lw $a0, 0($sp)`, `addiu $s, $29, 4`, `syscall`, and `sw $a0, 0($sp)`. The "Kernel Text Segment" contains instructions for saving and restoring registers, such as `move $k1, $at` and `sw $v0, $1`.

At the bottom of the window, there are two copyright notices for SPIM Version 9.1.17 of January 1, 2016, by James M. Larus, and a notice about the GNU Lesser General Public License.

What SPIM looks like.



Register Pane and Memory Panel

The screenshot displays the QtSpim MIPS simulator interface. The top menu bar includes File, Simulator, Registers, Text Segment, Data Segment, Window, and Help. Below the menu is a toolbar with various icons. The main window is divided into three panes:

- Register pane:** Located on the left, it shows the status of various registers. The 'FP Regs' section is expanded to show 'nt Regs [16]'. The 'Data' and 'Text' tabs are selected. The register list includes \$12 through \$29, with \$29 highlighted in blue.
- Memory panel:** Located on the right, it displays the assembly code for the 'user text segment [00400000]..[00440000]'. The instruction at address 00400044 is highlighted in blue: `00400044: 0000000c syscall`.
- Message panel:** Located at the bottom, it shows the output of the simulator. It includes the SPIM version (9.1.17), copyright information (1990-2016, James M. Larus), and a message stating 'Memory and registers cleared'.

There's also a console window.



Operations

- ▶ Load a source file: File → Reinitialize and Load File
- ▶ Run the code: F5 or Press the green triangle button
- ▶ Single stepping: F10
- ▶ Breakpoint: in Text panel, right click on an address to set a breakpoint there.



Overview

SPIM

Assembly Programing

System Service in SPIM

Lab Assignment



Registers

- ▶ 32 general-purpose registers
- ▶ register preceded by \$ in assembly language instruction
- ▶ two formats for addressing:
 - ▶ using register number e.g. \$0 through \$31
 - ▶ using equivalent names e.g. \$t1, \$sp
- ▶ special registers Lo and Hi used to store result of multiplication and division
 - ▶ not directly addressable; contents accessed with special instruction `mghi` (“move from Hi”) and `mflr` (“move from Lo”)



Register Names and Descriptions

Name	Register Number	Usage	Preserve on call?
\$zero	0	constant 0 (hardware)	n.a.
\$at	1	reserved for assembler	n.a.
\$v0 - \$v1	2-3	returned values	no
\$a0 - \$a3	4-7	arguments	yes
\$t0 - \$t7	8-15	temporaries	no
\$s0 - \$s7	16-23	saved values	yes
\$t8 - \$t9	24-25	temporaries	no
\$gp	28	global pointer	yes
\$sp	29	stack pointer	yes
\$fp	30	frame pointer	yes
\$ra	31	return addr (hardware)	yes



Data Types and Literals

Data types:

- ▶ Instructions are all 32 bits
- ▶ byte(8 bits), halfword (2 bytes), word (4 bytes)
- ▶ a character requires 1 byte of storage
- ▶ an integer requires 1 word (4 bytes) of storage
- ▶ Data types: `.asciiz` for string, `.word` for int, ...

Literals:

- ▶ numbers entered as is. e.g. 4
- ▶ characters enclosed in single quotes. e.g. 'b'
- ▶ strings enclosed in double quotes. e.g. "A string"



Program Structure I

- ▶ 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

- ▶ Identified with assembler directive **.data**.
- ▶ Declares variable names used in program
- ▶ Storage allocated in main memory (RAM)
- ▶ `<name>: .<datatype> <value>`



Program Structure II

Code

- ▶ placed in section of text identified with assembler directive **.text**
- ▶ contains program code (instructions)
- ▶ starting point for code e.g. execution given label **main:**
- ▶ ending point of main code should use exit system call

Comments

anything following # on a line



Program Structure III

The structure of an assembly program looks like this:

Program outline

```
# Comment giving name of program and description
# Template.s
# Bare-bones outline of MIPS assembly language program

    .globl main

    .data    # variable declarations follow this line
            # ...

    .text    # instructions follow this line

main:      # indicates start of code
           # ...

# End of program, leave a blank line afterwards
```



An Example Program

```
1      .globl main
2      .data
3 msg:  .ascii "Welcome to CENG3420.\n"
4      .text
5 main:
6      li $v0,4
7      la $a0,msg
8      syscall
9      li $v0,10
10     syscall
11
```

- ▶ `li`: load immediate
- ▶ `la`: load address
- ▶ `lw`: load word from memory



More Information

For more information about MIPS instructions and assembly programming you can refer to:

1. Lecture slides and textbook.

2. `http:`

`//www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html`



Overview

SPIM

Assembly Programing

System Service in SPIM

Lab Assignment



System calls in SPIM I

SPIM provides a small set of operating system-like services through the system call (`syscall`) instruction.

Service	System call code	Arguments	Result
<code>print_int</code>	1	<code>\$a0 = integer</code>	
<code>print_float</code>	2	<code>\$f12 = float</code>	
<code>print_double</code>	3	<code>\$f12 = double</code>	
<code>print_string</code>	4	<code>\$a0 = string</code>	
<code>read_int</code>	5		integer (in <code>\$v0</code>)
<code>read_float</code>	6		float (in <code>\$f0</code>)
<code>read_double</code>	7		double (in <code>\$f0</code>)
<code>read_string</code>	8	<code>\$a0 = buffer, \$a1 = length</code>	
<code>sbrk</code>	9	<code>\$a0 = amount</code>	address (in <code>\$v0</code>)
<code>exit</code>	10		
<code>print_char</code>	11	<code>\$a0 = char</code>	
<code>read_char</code>	12		char (in <code>\$v0</code>)
<code>open</code>	13	<code>\$a0 = filename (string), \$a1 = flags, \$a2 = mode</code>	file descriptor (in <code>\$a0</code>)
<code>read</code>	14	<code>\$a0 = file descriptor, \$a1 = buffer, \$a2 = length</code>	num chars read (in <code>\$a0</code>)
<code>write</code>	15	<code>\$a0 = file descriptor, \$a1 = buffer, \$a2 = length</code>	num chars written (in <code>\$a0</code>)
<code>close</code>	16	<code>\$a0 = file descriptor</code>	
<code>exit2</code>	17	<code>\$a0 = result</code>	



System calls in SPIM II

To request a service, a program loads the system call code into register $\$v0$ and arguments into registers $\$a0$ – $\$a3$ (or $\$f12$ for floating-point values). System calls that return values put their results in register $\$v0$ (or $\$f0$ for floating-point results). Like this example:

Using system call

```
.data
str: .asciiz "the_answer_=_ " #labels always followed by colon
.text

li    $v0, 4    # system call code for print_str
la    $a0, str  # address of string to print
syscall # print the string
li    $v0, 1    # system call code for print_int
li    $a0, 5    # integer to print
syscall # print it
```



Overview

SPIM

Assembly Programing

System Service in SPIM

Lab Assignment



Lab Assignment

Write an assembly program with the following requirements:

1. Define two variables `var1` and `var2` which have initial value 15 and 19, respectively.
2. Print `var1` and `var2`.
3. Print RAM addresses of `var1` and `var2` using `syscall`.
4. Swap `var1` and `var2` and print them.

Submission Method:

Submit the source code and report **after** the whole Lab1, onto [blackboard](#).



Some Tips

1. Variables should be declared following the `.data` identifier.
2. `<name>: .<datatype> <value>`
3. Use `la` instruction to access the RAM address of declared data.
4. Use system call to print integers.
5. Do not forget `exit` system call.

