

# CENG3420

## Lab 1-1: MIPS assembly language programming

**Bei Yu**

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

[byu@cse.cuhk.edu.hk](mailto:byu@cse.cuhk.edu.hk)

Spring 2018



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

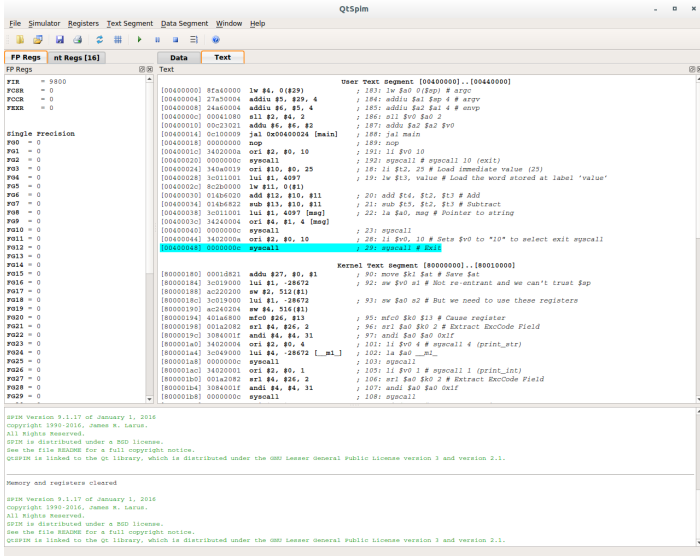
- ▶ **SPIM is a MIPS32 simulator.**
- ▶ *Spim* is a self-contained simulator that runs MIPS32 programs.
- ▶ It reads and executes assembly language programs written for this processor.
- ▶ *Spim* also provides a simple debugger and minimal set of operating system services.
- ▶ *Spim* does not execute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

Download it here:

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



# SPIM Overview



The screenshot shows the QtSpim MIPS simulator interface. The title bar reads "QtSpim". The menu bar includes "File", "Simulator", "Registers", "Text Segment", "Data Segment", "Window", and "Help". Below the menu bar is a toolbar with various icons. The main window is divided into two panes: "FP Regs" and "nt Regs [16]". The "FP Regs" pane shows floating-point registers F0-F14, all set to 0. The "nt Regs [16]" pane shows integer registers \$0-\$31, with \$0-\$14 set to 0 and \$15-\$31 containing assembly instructions. The assembly code is displayed in a text area, showing instructions like "lw \$4, 0(\$29)", "addiu \$5, \$29, 4", "addiu \$6, \$5, 4", "ori \$2, \$0, 10", "syscall", "sw \$0, 10(\$0)", "move \$k1, \$at", "sw \$0, \$1", "nop", "sw \$0, \$2", "sw \$0, \$3", "sw \$0, \$4", "sw \$0, \$5", "sw \$0, \$6", "sw \$0, \$7", "sw \$0, \$8", "sw \$0, \$9", "sw \$0, \$10", "sw \$0, \$11", "sw \$0, \$12", "sw \$0, \$13", "sw \$0, \$14", "sw \$0, \$15", "sw \$0, \$16", "sw \$0, \$17", "sw \$0, \$18", "sw \$0, \$19", "sw \$0, \$20", "sw \$0, \$21", "sw \$0, \$22", "sw \$0, \$23", "sw \$0, \$24", "sw \$0, \$25", "sw \$0, \$26", "sw \$0, \$27", "sw \$0, \$28", "sw \$0, \$29", "sw \$0, \$30", "sw \$0, \$31". The registers \$15-\$31 are highlighted in blue. Below the assembly code, there is a section for "Kernel Text Segment (80000000)..[80100000]" and "User Text Segment [00400000]..[00440000]". The bottom of the window contains copyright information for SPIM Version 9.1.17 of January 1, 2016, and a notice about the GNU Lesser General Public License version 3 and version 2.1.

What SPIM looks like.



# Register Panel and Memory Panel

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs [16] int Regs [16] Data Text

FP Regs

FP Regs

FP0 = 0  
FP1 = 0  
FP2 = 0  
FP3 = 0  
FP4 = 0  
FP5 = 0  
FP6 = 0  
FP7 = 0  
FP8 = 0  
FP9 = 0  
FP10 = 0  
FP11 = 0  
FP12 = 0  
FP13 = 0  
FP14 = 0  
FP15 = 0  
FP16 = 0  
FP17 = 0  
FP18 = 0  
FP19 = 0  
FP20 = 0  
FP21 = 0  
FP22 = 0  
FP23 = 0  
FP24 = 0  
FP25 = 0  
FP26 = 0  
FP27 = 0  
FP28 = 0  
FP29 = 0

Text

```
[00400000] 8fa40000 lw $4, 0($29)           User Text Segment [00400000]..[00400000]
[00400004] 27a50004 addiu $5, $29, 4           ; 183: lw $a0 0($sp) # argc
[00400008] 24a60004 addiu $6, $5, 4           ; 184: addiu $a1 $sp 4 # argv
[0040000c] 00e41080 ori $2, $4, 2           ; 185: addiu $a2 $a1 4 # envp
[00400010] 00c23021 addu $6, $6, $2           ; 186: aii $v0 $a2 3
[00400014] 0e100009 jal 0x00400024 [main]          ; 187: addu $a2 $a2 $v0
[00400018] 00000000 nop                               ; 188: jal main
[0040001c] 3402000a ori $2, $0, 10          ; 189: nop
[00400020] 0000000c syscall                  ; 190: li $v0, 10
[00400024] 34e00019 ori $10, $0, 25         ; 191: li $v0, 10 (exit)
[00400028] 3c011001 lui $1, 4097           ; 192: li $t2, 25 # Load immediate value (25)
[0040002c] 8c2b0000 lw $11, 0($t1)          ; 193: lw $t3, value # Load the word stored at label 'value'
[00400030] 014b6020 add $12, $10, $11          ; 201: add $t4, $t2, $t3 # Add
[00400034] 014b4022 sub $13, $10, $11          ; 21: sub $t5, $t2, $t3 # Subtract
[00400038] 3c011001 lui $1, 4097 [msg]      ; 22: la $a0, msg # Pointer to string
[0040003c] 34240004 ori $4, $1, 4 [msg]
[00400040] 0000000c syscall                  ; 23: syscall
[00400044] 3402000a ori $2, $0, 10          ; 28: li $v0, 10 # Sets $v0 to "10" to select exit syscall
[00400048] 0000000c syscall                  ; 29: syscall # Exit

Kernel Text Segment [80000000]..[80010000]
[80001800] 0001d021 addu $27, $0, $1          ; 90: move $k1 $a1 # Save $a1
[80001804] 3c019000 lui $1, -28672         ; 92: sw $v0 $1 # Noc re-entrant and we can't trust $sp
[80001808] ac220200 sw $2, 512($1)          ; 93: sw $a0 $2 # But we need to use these registers
[8000180c] 3c019000 lui $1, -28672         ; 94: sw $a0 $2 # But we need to use these registers
[80001810] ac240204 sw $6, 516($1)          ; 95: mfc0 $k0 $13 # Cause register
[80001814] 401a0000 mfc0 $26, $13          ; 96: srl $a0 $k0 2 # Extract EnoCode Field
[80001818] 001a2082 srl $4, $26, 2         ; 97: andi $a0 $a0 0x1f
[8000181c] 3084001f andi $4, $4, $1         ; 101: li $v0 4 # syscall 4 (print_str)
[80001820] 3402000a ori $2, $0, 4          ; 102: la $a0 _ml_
[80001824] 3c040000 lui $4, -28672 [_ml_]
[80001828] 0000000c syscall                  ; 103: syscall
[8000182c] 34020001 ori $2, $0, 1           ; 105: li $v0 1 # syscall 1 (print_int)
[80001830] 001a2082 srl $4, $26, 2         ; 106: srl $a0 $a0 2 # Extract EnoCode Field
[80001834] 3084001f andi $4, $4, $1         ; 107: andi $a0 $a0 0x1f
[80001838] 0000000c syscall                  ; 108: syscall
```

Register panel

Memory panel

SPIM Version 9.1.17 of January 1, 2016  
Copyright 1993-2016, James W. Larus.  
All Rights Reserved.  
SPIM is distributed under a BSD license.  
See the file README for a full copyright notice.  
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

Message panel

Memory and registers cleared

SPIM Version 9.1.17 of January 1, 2016  
Copyright 1993-2016, James W. Larus.  
All Rights Reserved.  
SPIM is distributed under a BSD license.  
See the file README for a full copyright notice.  
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

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 `mfl0` (“move from Lo”)



# Register Names and Descriptions

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



# 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:  .asciiz "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.

