

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

2017 Spring



香港中文大學

The Chinese University of Hong Kong

Overview

SPIM

Assembly programming

System service in SPIM

Lab assignment



Overview

SPIM

Assembly programming

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 displays the QtSpim MIPS simulator interface. The main window is titled "QtSpim" and contains several panes. At the top, there is a menu bar with "File", "Simulator", "Registers", "Text Segment", "Data Segment", "Window", and "Help". Below the menu bar is a toolbar with various icons for simulation control. The main area is divided into two sections: "FP Regs" and "nt Regs [16]". The "FP Regs" section shows registers \$FIR through \$F23, all with a value of 0. The "nt Regs [16]" section shows registers \$0 through \$31, with \$0 through \$10 containing 0, and \$11 through \$31 containing various values. The assembly code pane shows the following code:

```
[00400000] 00400000 lw $t, 0($0) ; 181: lw $a0 0($0) # argp
[00400004] 27a50004 addiu $t, $29, 4 ; 184: addiu $a1 $a1 4 # argv
[00400008] 24a80004 addiu $t, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041000 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c31001 addu $t, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x04000024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall 1 # syscall 10 (exit)
[00400024] 340a0019 ori $10, $0, 25 ; 18: li $12, 25 # Load immediate value (25)
[00400028] 3c011001 lui $1, 4097 ; 19: li $13, value # Load the word stored at label 'value'
[0040002c] 8c280000 lw $11, 0($1) ; 20: add $14, $12, $13 # Add
[00400030] 014b6020 add $12, $10, $11 ; 21: sub $15, $12, $13 # Subtract
[00400034] 014b6822 sub $13, $10, $11 ; 22: la $a0, msg # Pointer to string
[00400038] 3c011001 lui $1, 4097 [msg] ; 23: syscall
[0040003c] 34240004 ori $4, $1, 4 [msg] ; 24: syscall
[00400040] 0000000c syscall ; 25: syscall
[00400044] 3402000a ori $2, $0, 10 ; 26: li $v0, 10 # Sets $v0 to '10' to select exit syscall
[00400048] 0000000c syscall ; 27: syscall # Exit
```

The assembly code is color-coded: User Text Segment (00400000..[00400000]), Kernel Text Segment (80000000..[80000000]), and Data Segment (00400000..[00400000]). The registers \$FIR through \$F23 are all 0. The registers \$0 through \$10 are 0. The registers \$11 through \$31 contain various values: \$11=0, \$12=25, \$13=4097, \$14=4097, \$15=4097, \$16=4097, \$17=4097, \$18=4097, \$19=4097, \$20=4097, \$21=4097, \$22=4097, \$23=4097, \$24=4097, \$25=4097, \$26=4097, \$27=4097, \$28=4097, \$29=4097, \$30=4097, \$31=4097.

SPIM version 9.1.17 of January 1, 2016
copyright 1999-2016, James R. 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.

Memory and registers cleared

SPIM version 9.1.17 of January 1, 2016
Copyright 1999-2016, James R. 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.

What SPIM looks like.



Register Panel and Memory Panel

The screenshot displays the QtSpim simulator interface with three panels highlighted in red:

- Register panel:** Located on the left side, it shows the state of various registers. The `FP Regs` section is expanded, listing registers from `FP0` to `FP29`. The `FP0` register is highlighted in blue.
- Memory panel:** Located in the center, it displays assembly code with corresponding comments. The line `00400044: 0000000c syscall` is highlighted in blue.
- Message panel:** Located at the bottom, it shows system messages and copyright information for QtSPIM version 9.1.17.

```
File Simulator Registers Text Segment Data Segment Window Help
FP Regs nt Regs [16] Data Text
FP Regs
FP0 = 9800
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

(00400000) 8fa40000 lw $t, 0($20)
(00400004) 27a50004 addiu $t, $29, 4
(00400008) 24a60004 addiu $t, $5, 4
(0040000c) 00041000 sll $2, $4, 2
(00400010) 00c33021 addu $6, $6, $2
(00400014) 0c100009 jal 0x04000024 [main]
(00400018) 00000000 nop
(0040001c) 3402000a ori $2, $0, 10
(00400020) 0000000c syscall
(00400024) 340a0019 ori $10, $0, 25
(00400028) 3c011001 lui $1, 4097
(0040002c) 8c280000 lw $11, 0($1)
(00400030) 014b6020 add $12, $10, $11
(00400034) 014b6822 sub $13, $10, $11
(00400038) 3c011001 lui $1, 4097 [msg]
(0040003c) 34240004 ori $4, $1, 4 [msg]
(00400040) 0000000c syscall
(00400044) 3402000a ori $2, $0, 10
(00400048) 0000000c syscall

User Text Segment [00400000]..[00440000]
; 181: lw $a0 0($sp) # argc
; 184: addiu $a1 $p, 4 # argv
; 185: addiu $a2 $a1, 4 # envp
; 186: sll $v0 $a0 2
; 187: addu $a2 $a2 $v0
; 188: jal main
; 189: nop
; 190: # $v0 = 4
; 191: # $v0 = 10 (exit)
; 19: li $t2, 25 # Load immediate value (25)
; 19: lw $t3, $v0 # Load the word stored at label 'value'
; 20: add $t4, $t2, $t3 # Add
; 21: sub $t5, $t2, $t3 # Subtract
; 22: la $a0, msg # Pointer to string
; 23: syscall
; 28: li $v0, 10 # Sets $v0 to "10" to select exit syscall
; 29: syscall # Exit

[Kernel Text Segment [80000000]..[80010000]
; 90: move $k1 $at # Save $at
; 92: sw $v0 $k1 # Not re-entrant and we can't trust $p
; 93: sw $a0 $k2 # But we need to use these registers
; 95: mfc0 $k0 $t3 # Cause register
; 96: srl $a0 $k0 2 # Extract DecCode Field
; 97: andi $a0 $a0 0x1f
; 101: li $v0 4 # syscall 4 (print_ptr)
; 102: la $a0 __rl_
; 103: syscall
; 105: li $v0 1 # syscall 1 (print_int)
; 106: srl $a0 $k0 2 # Extract DecCode Field
; 107: andi $a0 $a0 0x1f
; 108: syscall

QtSPIM version 9.1.17 of January 1, 2016
copyright 1999-2016, James R. Larus.
All Rights Reserved.
QtSPIM 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.

Memory and registers cleared

QtSPIM version 9.1.17 of January 1, 2016
Copyright 1999-2016, James R. Larus.
All Rights Reserved.
QtSPIM 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 programming

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 `mfhi` (“move from Hi”) and `mflo` (“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: .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



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 programming

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_is_" #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 programming

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.

Lab report should include (1) source code, (2) console output.



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.

