CENG3420

Lab 1-1: MIPS assembly language programing

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 programing

System service in SPIM

Lab assignment

Overview

SPIM

Assembly programing

System service in SPIM

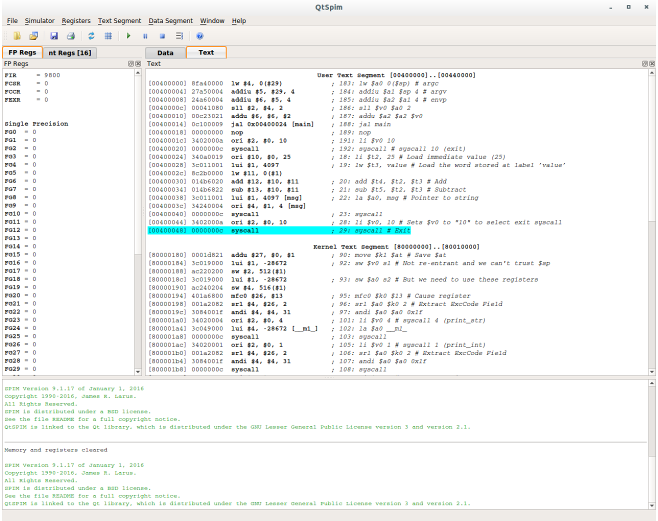
Lab assignment

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.

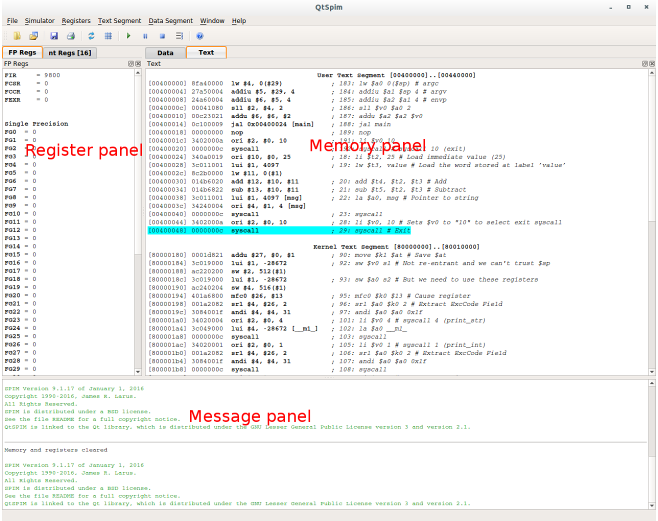
Dowload it here: `http://sourceforge.net/projects/spimsimulator/files/`

SPIM Overview



What SPIM looks like.

Register Panel and Memory Panel



There's also a console window.

Operations

- Load a source file: File $\rightarrow$ 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 `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

```
      .globl main
      .data
msg:  .asciiz "Welcome to CENG3420.\n"
      .text
main:
      li $v0,4
      la $a0,msg
      syscall
      li $v0,10
      syscall
```

- `li`: load immediate
- `la`: load address

More Information

For more information about MIPS instructions and assembly programing 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
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

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.

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.