

# Spim Download

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

Latest version of Spim is QtSpim.

QtSpim is available on Microsoft Windows, Mac OS X and Linux environment.

Downloading previous version is possible, but using QtSpim is preferred.

Go to url above and download appropriate file for your OS.

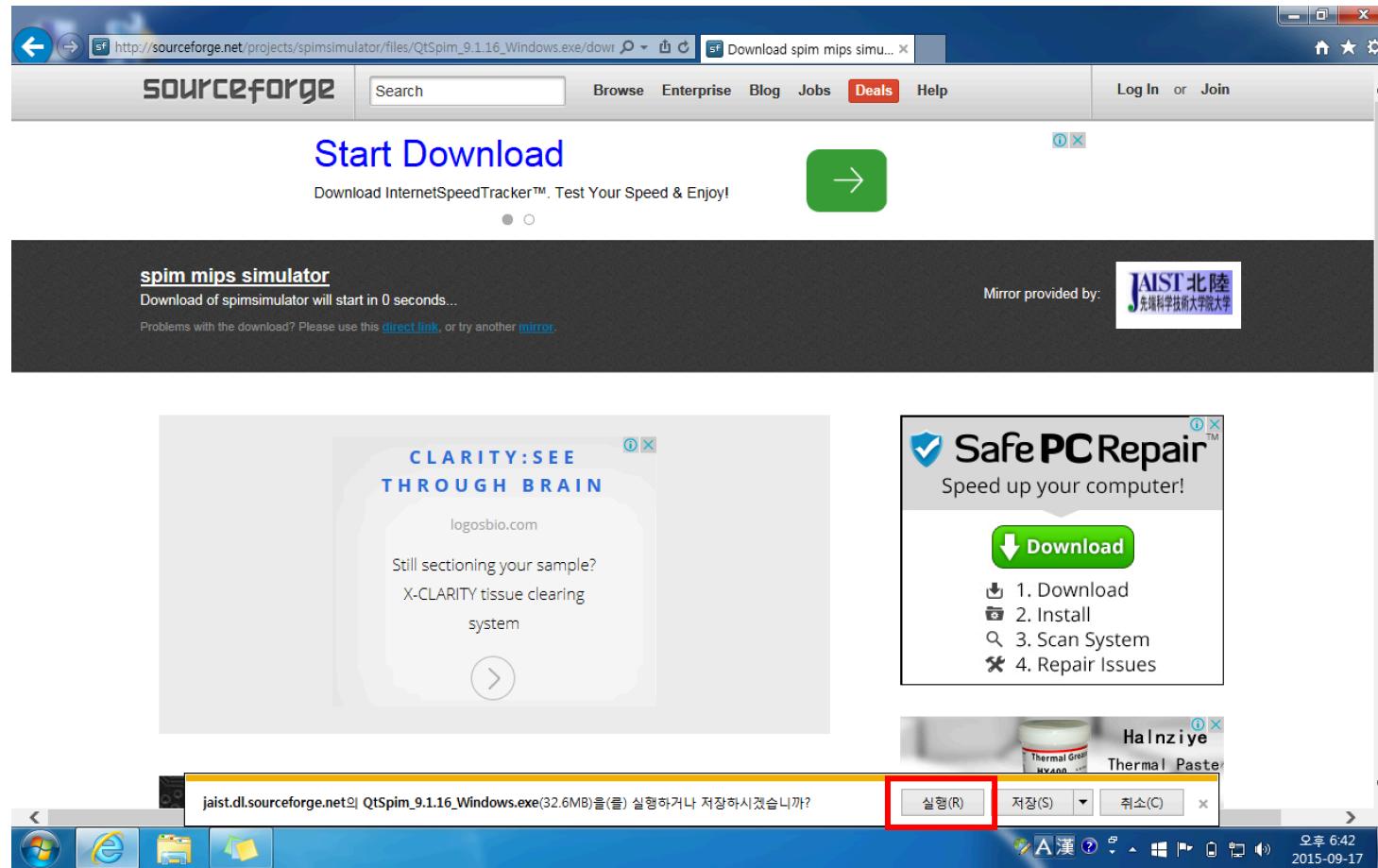
- Linux - qtspim\_9.1.16\_linux32.deb (32-bit), qtspim\_9.1.16\_linux64.deb (64-bit)
- Mac OS X - QtSpim\_9.1.16\_mac.mpkg.zip
- Windows - QtSpim\_9.1.16\_Windows.exe

# Download and Install

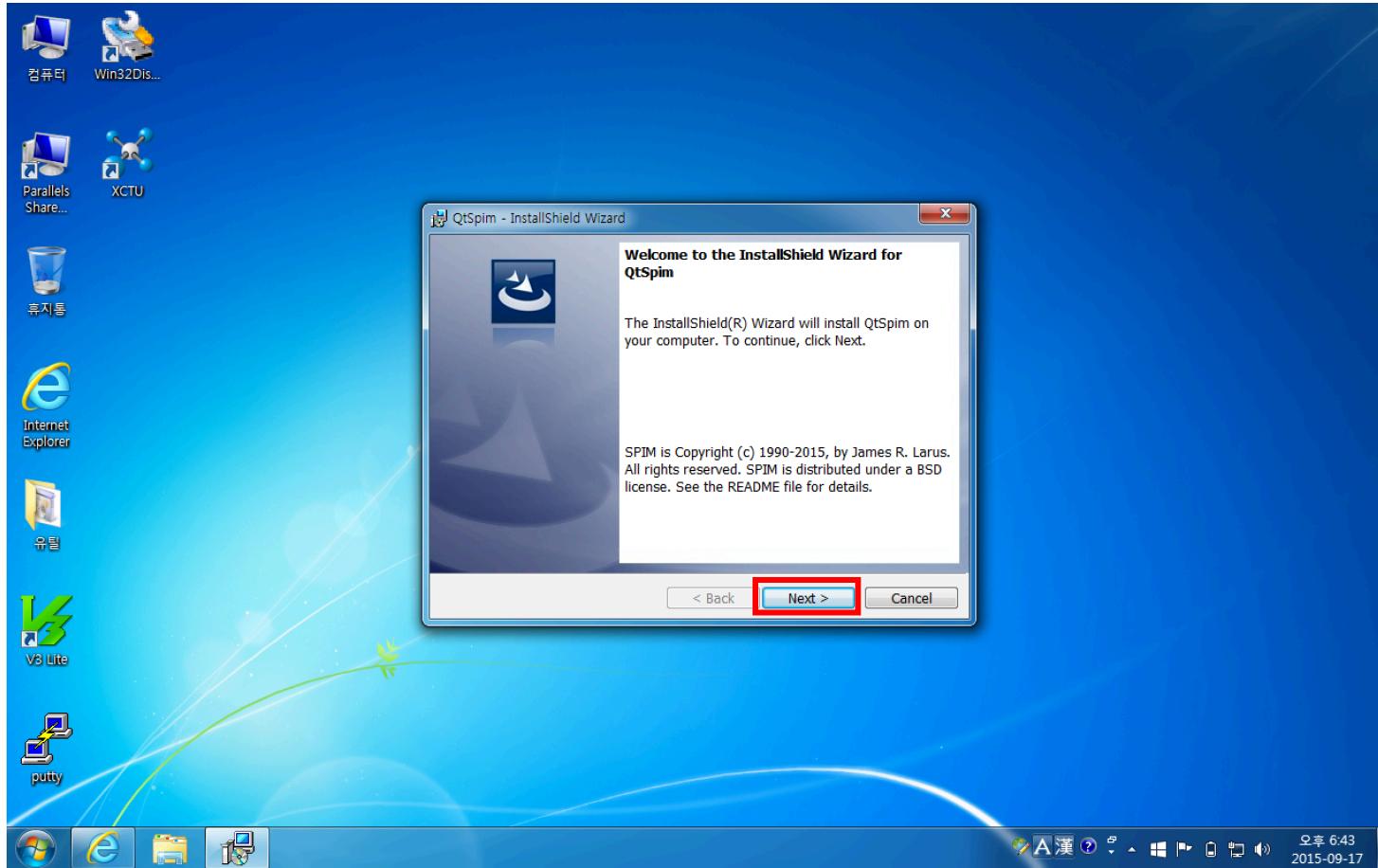
The screenshot shows a web browser window with the URL <http://sourceforge.net/projects/spimsimulator/files/>. The page title is "spim mips simulator - Bro...". The main content area displays a banner for "Open Source ESB By WSO2." followed by a sub-banner for "High Performance, Low Footprint SOA and Integration Middleware." Below this, there is a green button with a white arrow pointing right. To the right of the button, there is an advertisement for "MCSE Server Infrastructure 5-Course Bundle" with a "92% OFF >" discount offer. The main navigation bar includes links for "Browse", "Enterprise", "Blog", "Jobs", "Deals", "Help", "Log In", and "Join". Below the navigation bar, there are links for "SOLUTION CENTERS", "Go Parallel", "Resources", and "Newsletters". The main content area is titled "spim mips simulator" and is brought to you by "jameslarus". A navigation menu below the title includes "Summary", "Files", "Reviews", "Support", "Wiki", "Code", and "Tickets". A message at the top of the file list says "Looking for the latest version? Download [QtSpim\\_9.1.12\\_Windows.exe \(31.5 MB\)](#)". The file list table has columns for "Name", "Modified", "Size", and "Downloads / Week". The file "QtSpim\_9.1.16\_Windows.exe" is highlighted with a red border. The file list also includes "qtspim\_9.1.16\_linux32.deb", "QtSpim\_9.1.16\_mac.mpkg.zip", "qtspim\_9.1.16\_linux64.deb", "QtSpim\_9.1.15\_mac.mpkg.zip", "QtSpim\_9.1.13.mac.mpkg.zip", and "QtSpim\_9.1.12\_Windows.exe". On the right side of the page, there is a sidebar for "go parallel" with the tagline "Translating Multicore Power into Application Performance". It features a section titled "Stay connected and informed on parallel development via Go Parallel, where you'll find viewpoints, software tools, and educational information to help your software development work shine." Below this, there is a section titled "most recent posts:" with a note about an error retrieving the feed.

Name	Modified	Size	Downloads / Week
qtspim_9.1.16_linux32.deb	2015-08-26	29.7 MB	84
QtSpim_9.1.16_mac.mpkg.zip	2015-08-26	28.7 MB	264
qtspim_9.1.16_linux64.deb	2015-08-26	28.1 MB	92
<b>QtSpim_9.1.16_Windows.exe</b>	2015-08-26	34.3 MB	743
QtSpim_9.1.15_mac.mpkg.zip	2015-04-26	28.7 MB	134
QtSpim_9.1.13.mac.mpkg.zip	2014-02-05	33.1 MB	4
QtSpim_9.1.12_Windows.exe	2013-12-17	31.5 MB	454

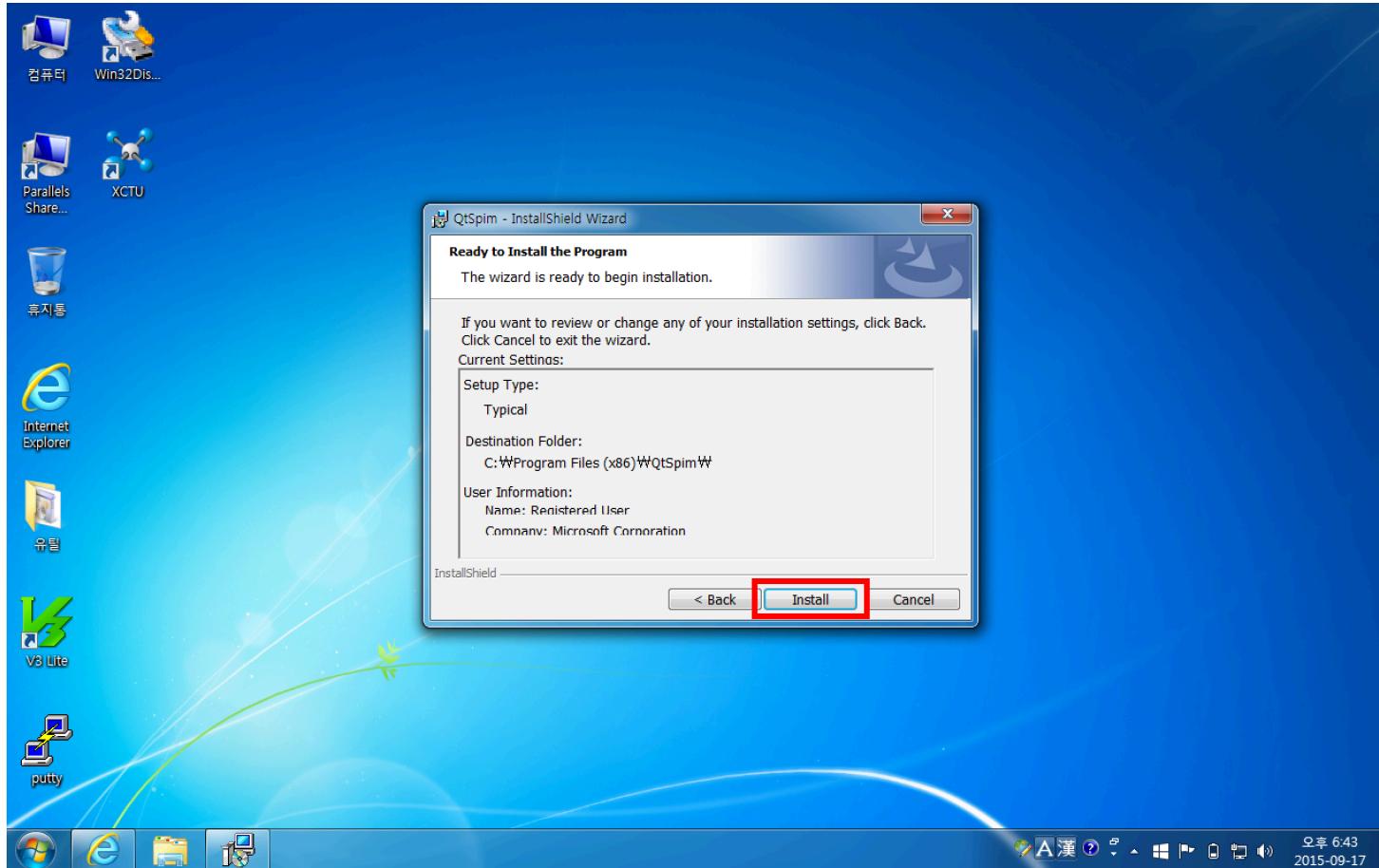
# Download and Install



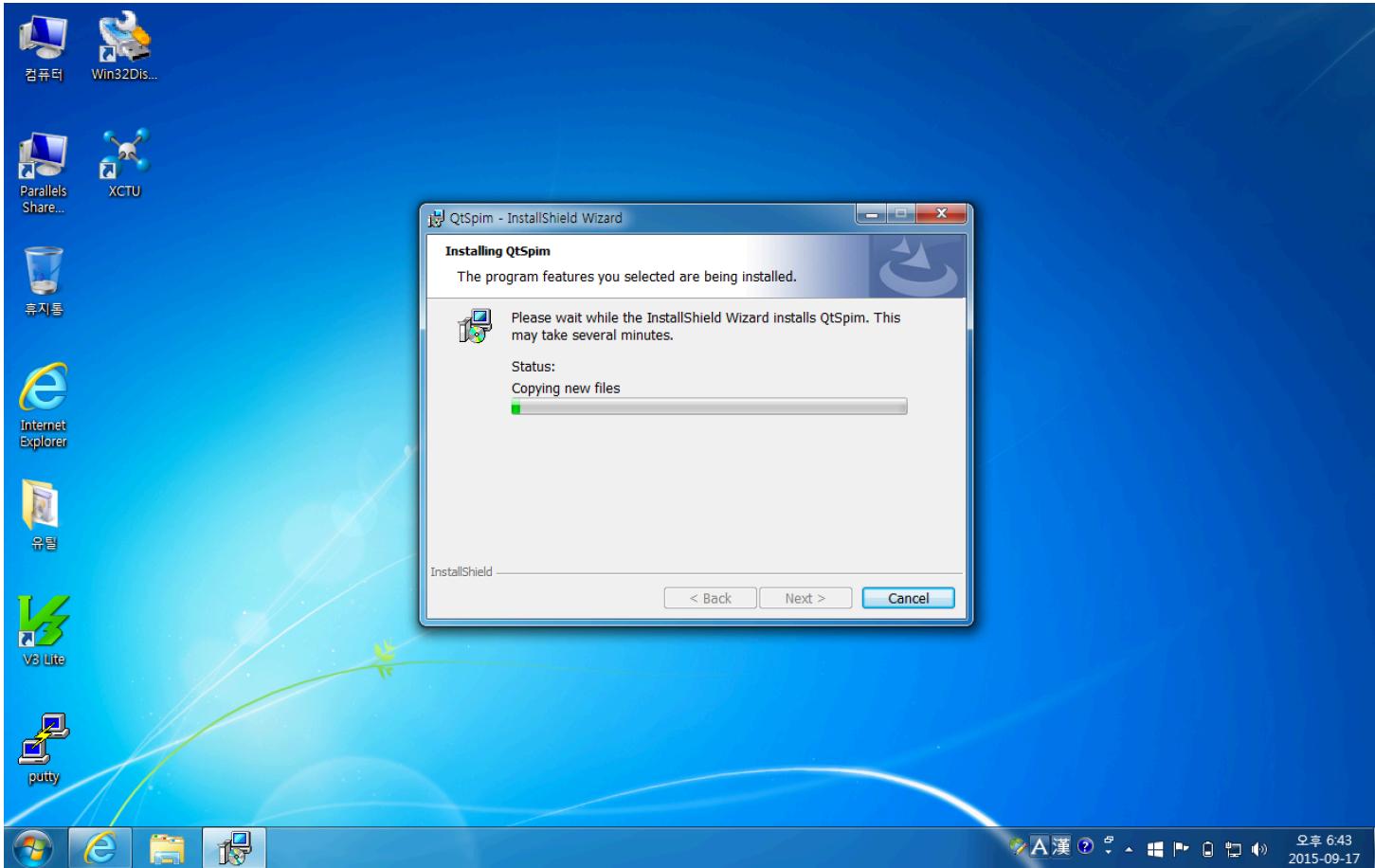
# Download and Install



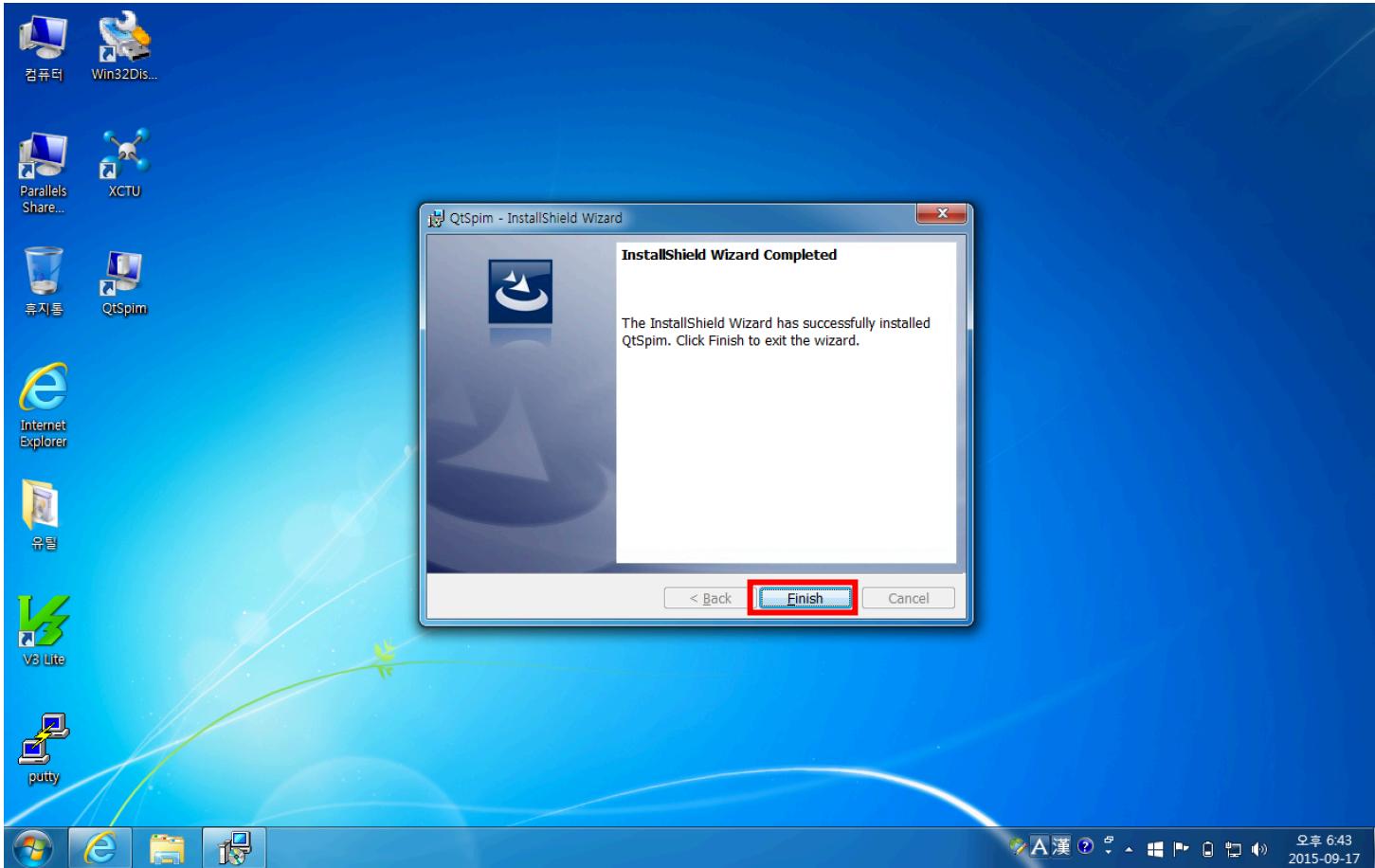
# Download and Install



# Download and Install



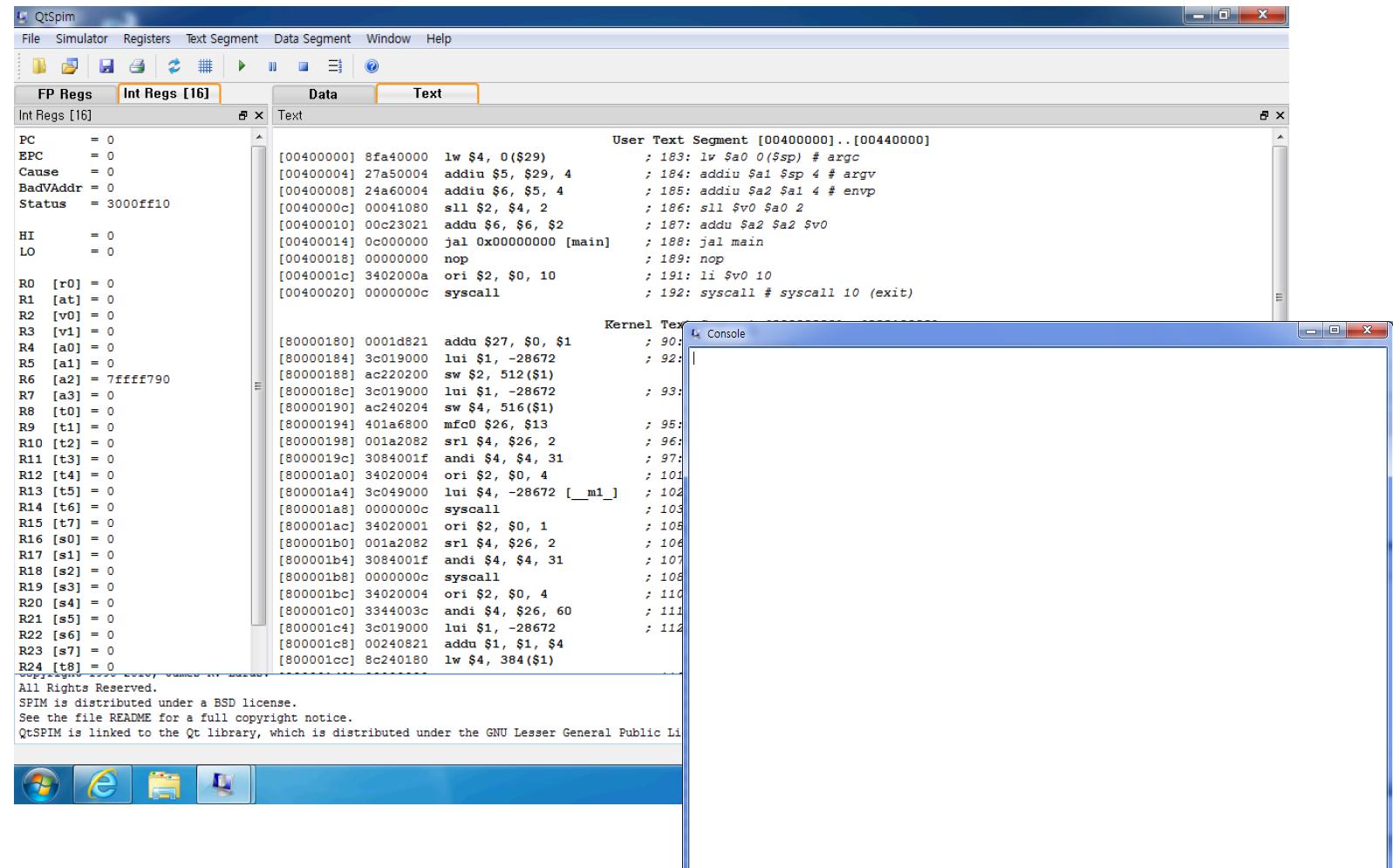
# Download and Install



# First screen of QtSpim

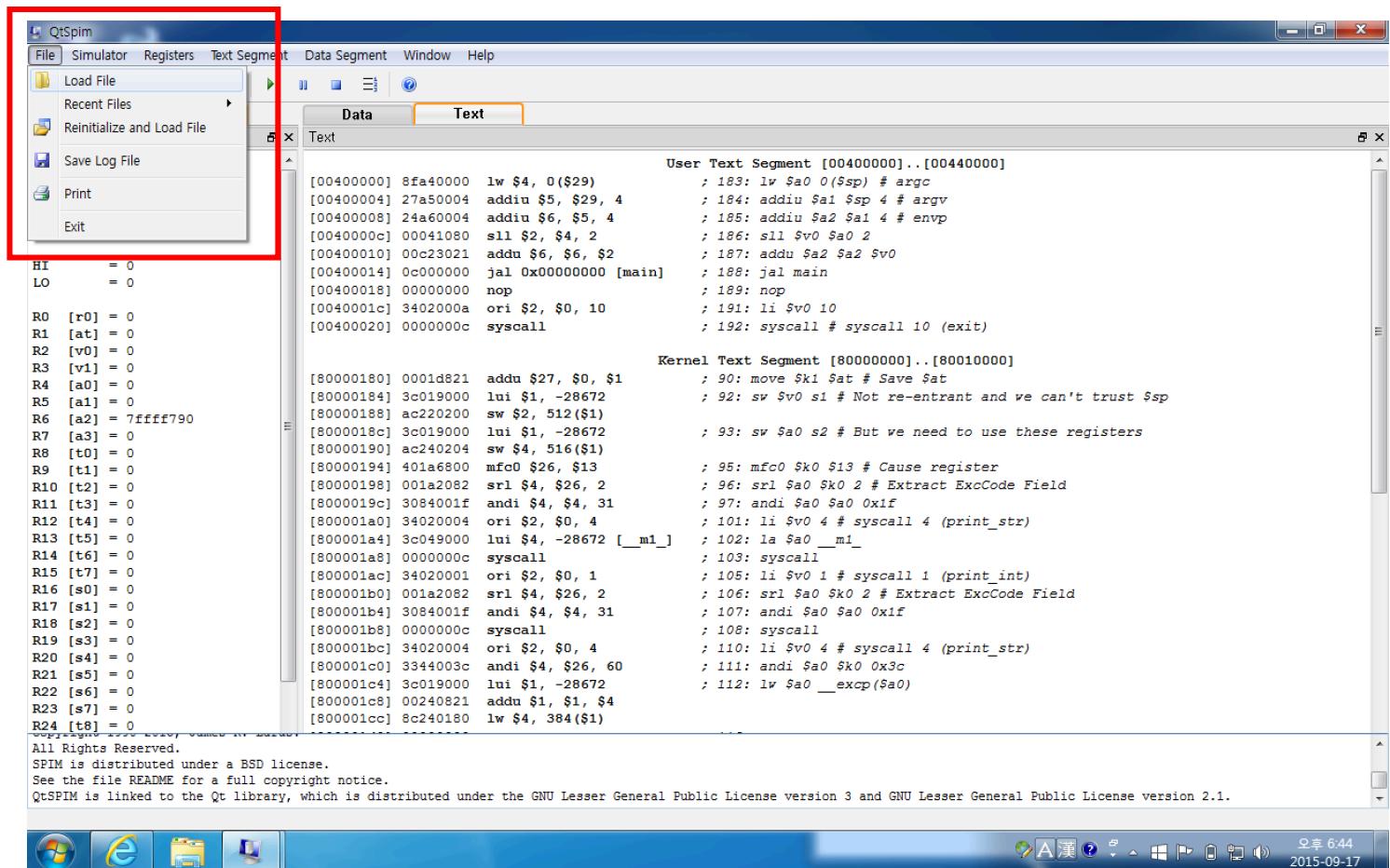
A window such as right picture and console window will be shown together.

You can check the result with console window.



# Loading assembly language file

Go to ‘File’ → ‘Load File’, then load an assembly language file.  
(extension will be .a.s or .asm)



# User file displayed

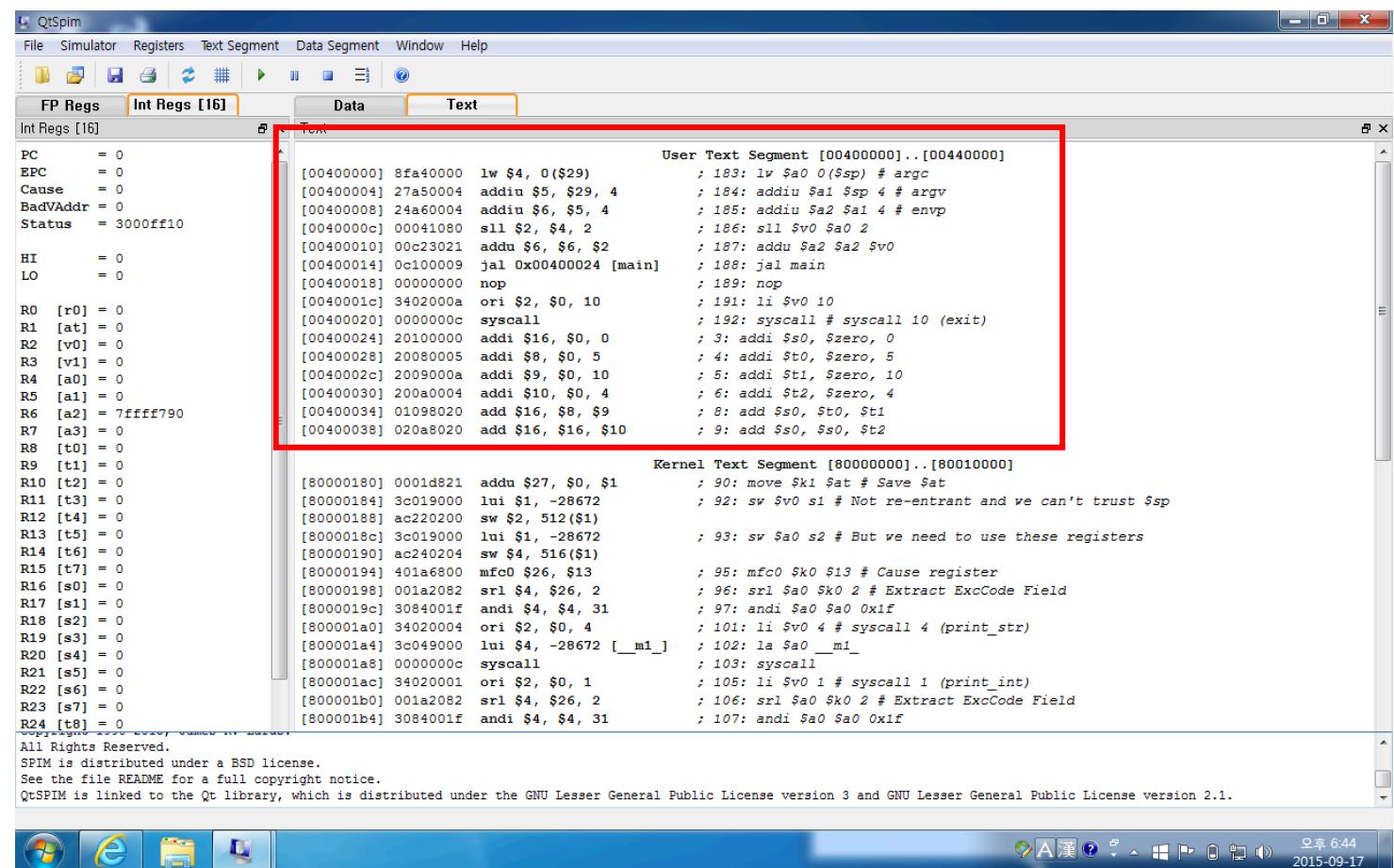
When a file is open, code will appear on the red square.

Other part show detailed procedure of transformation of assembly code to machine language.

(The code below is the source used in the example.)

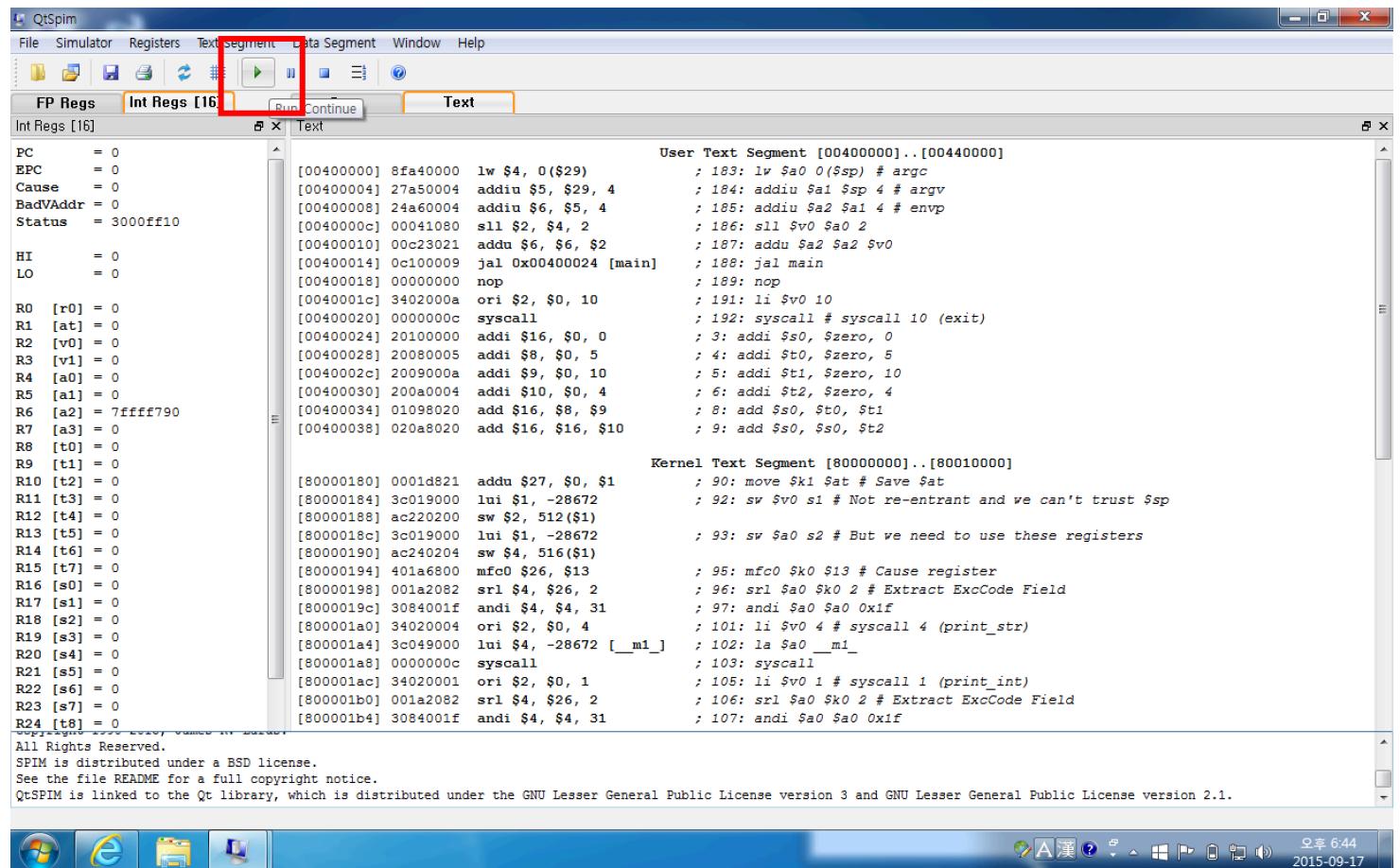
```
.text
main:
    addi $s0, $zero, 0
    addi $t0, $zero, 5
    addi $t1, $zero, 10
    addi $t2, $zero, 4

    add $s0, $t0, $t1
    add $s0, $s0, $t2
.end
```



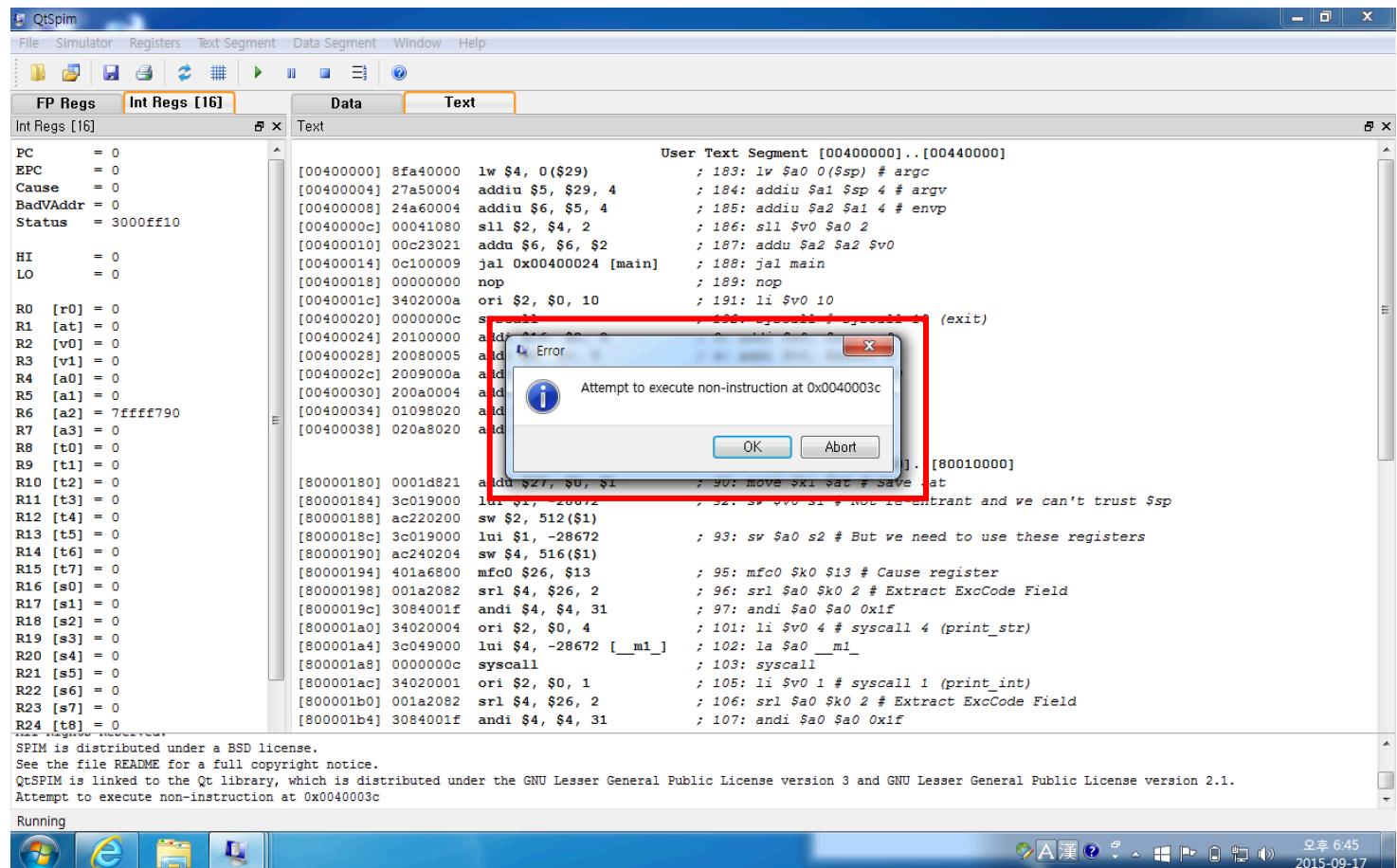
# Execution

When F5 or button in red box is pressed, assembly code will be executed.



# Message

When assembly code is executed, message on right side will be popped up. Then just press 'OK' button.



# Checking register value

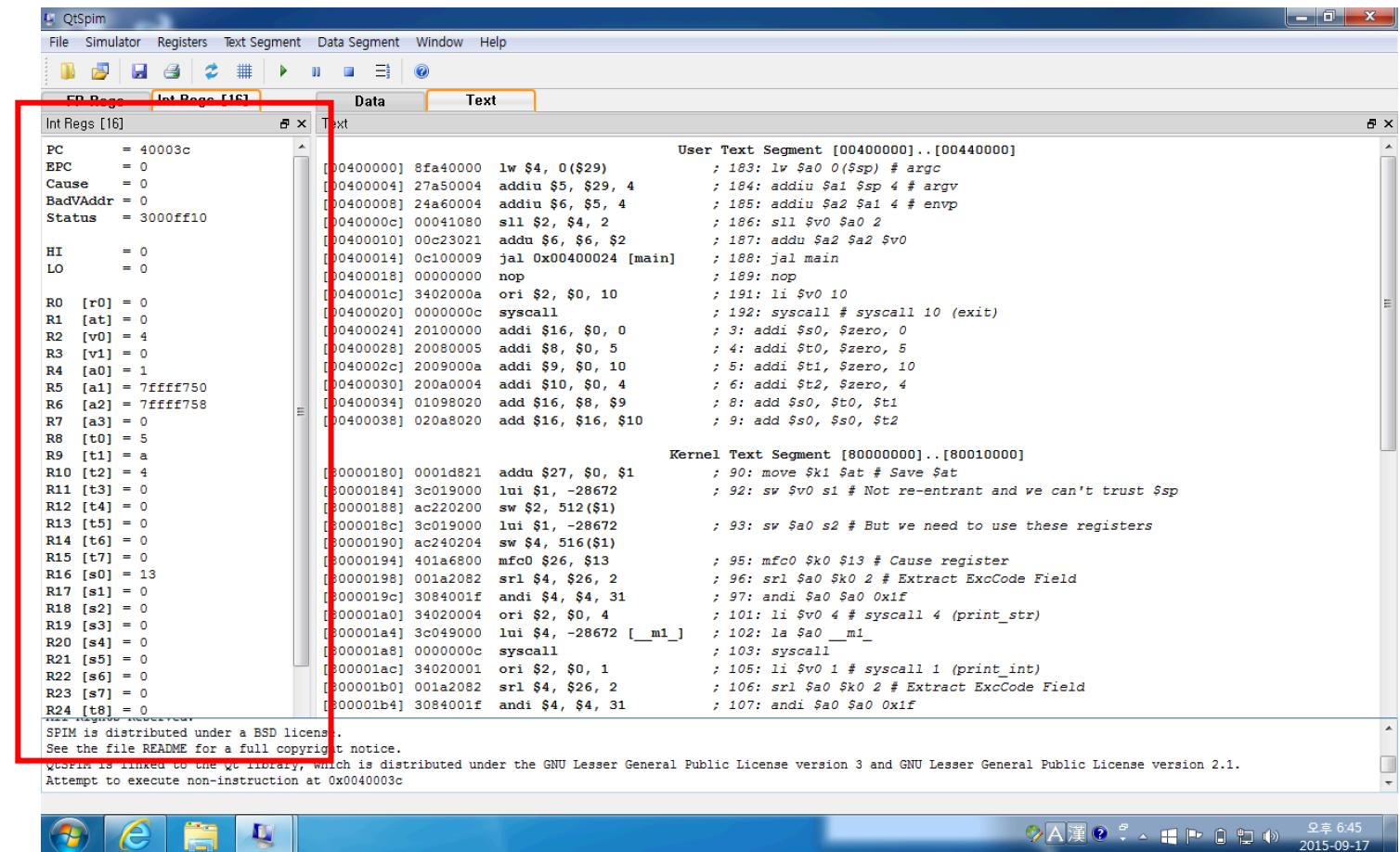
After execution of code, you can check each register value on the left side of window.

You can check the result of below source code.

**\$t0 = 5, \$t1 = a, \$t2 = 4, \$s0 = 13**

```
.text
main:
    addi $s0, $zero, 0
    addi $t0, $zero, 5
    addi $t1, $zero, 10
    addi $t2, $zero, 4

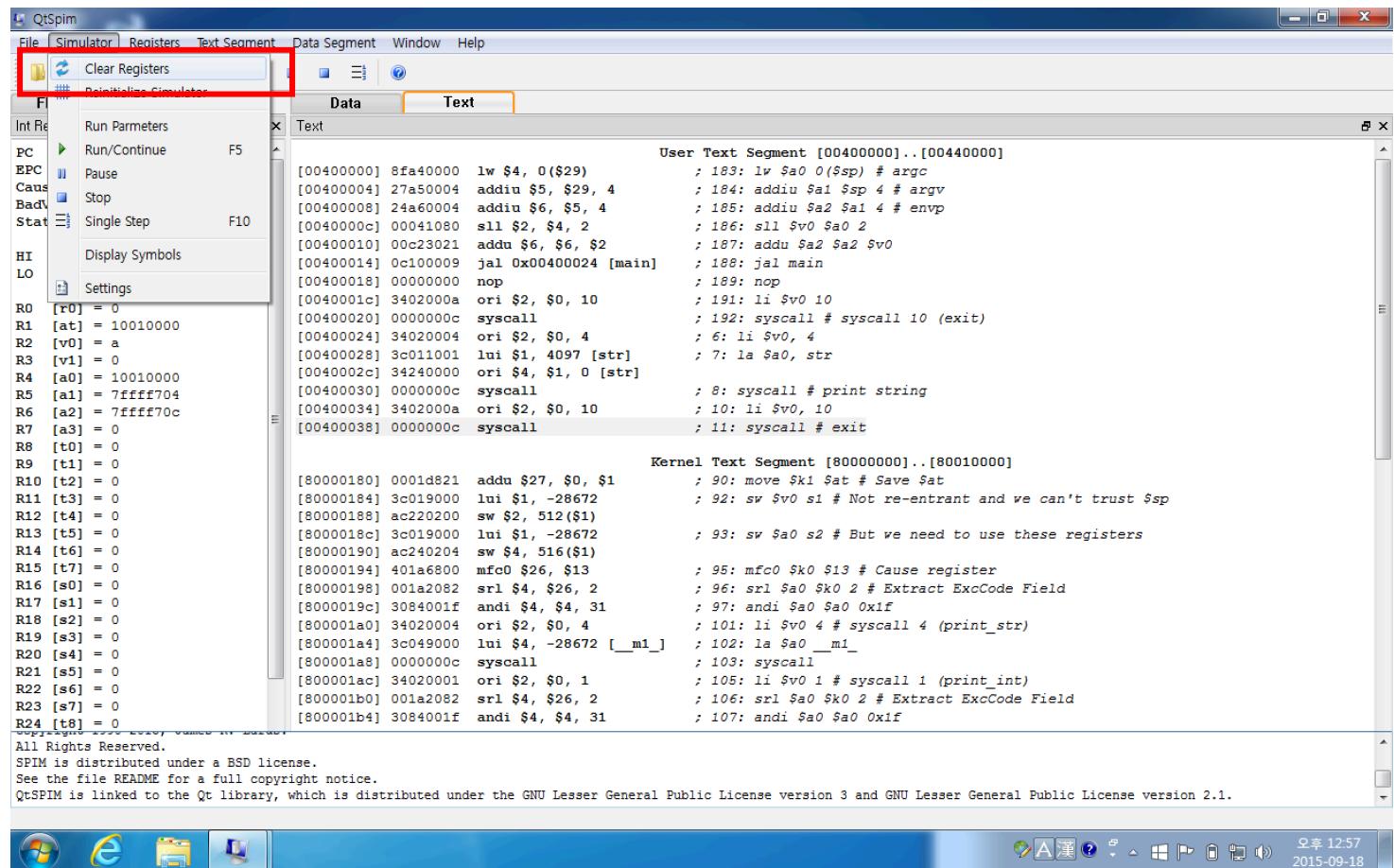
    add $s0, $t0, $t1
    add $s0, $s0, $t2
.end
```



Register values are hexadecimal numbers

# Re-Execution

If you want to re-execute source code,  
go to ‘Simulator’ →  
**‘Clear Registers’** and  
execute the source file.

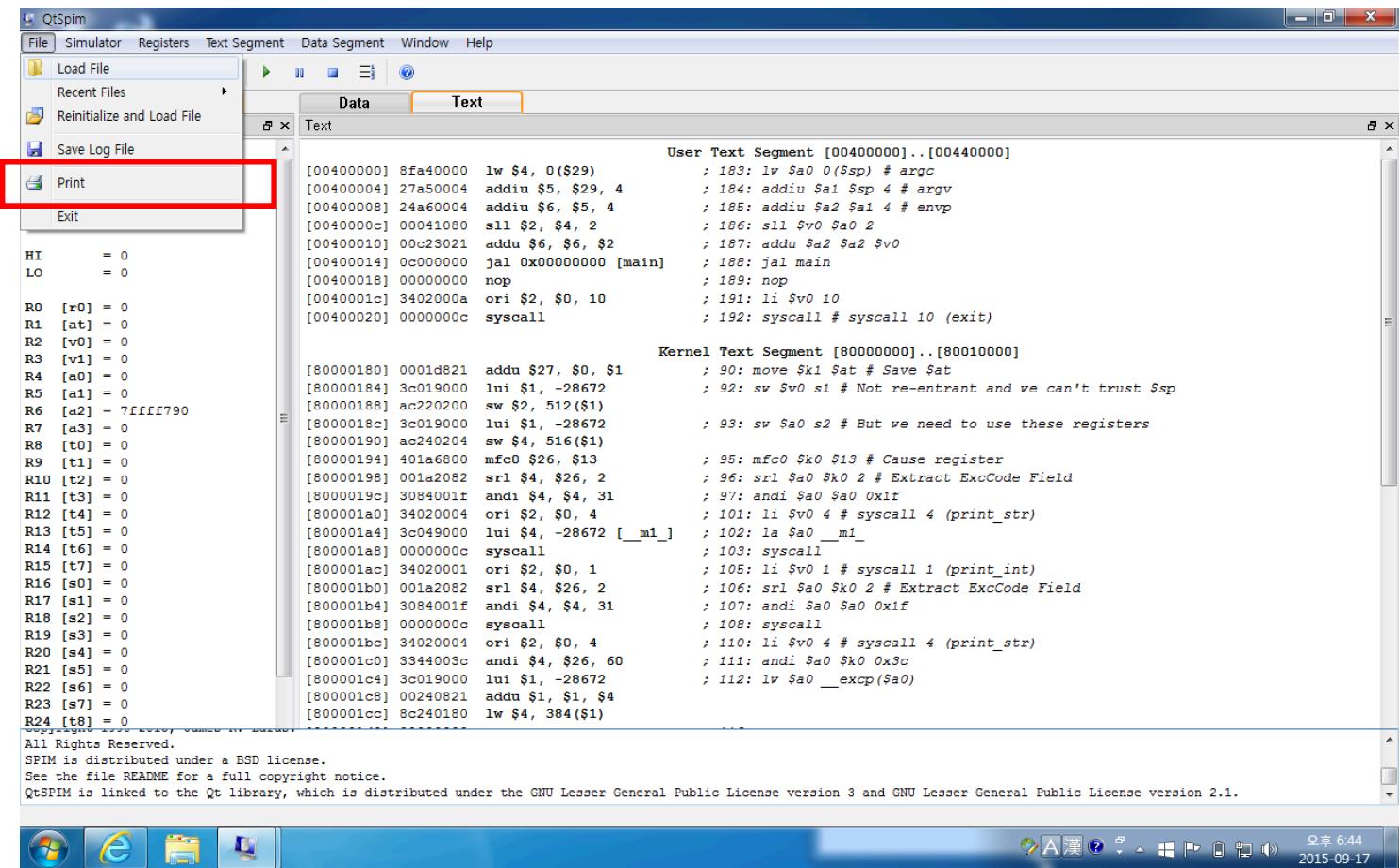


# Loading another file

If you want to load another assembly file you can do it with

**'File' → 'Reinitialize and Load File'**

Even when message below shows up, press **'File' → 'Reinitialize and Load File'**



spim: (parser) Label is defined for the second time on line 9 of file  
main:

^

# Example

The diagram illustrates the compilation process from C code to assembly code. On the left, the C code is shown:

```
#include <stdio.h>
int main(void)
{
    printf("Hello World!!!\n");
    return 0;
}
```

A blue oval highlights the part of the C code within the curly braces, with the annotation: "This part is custom expression."

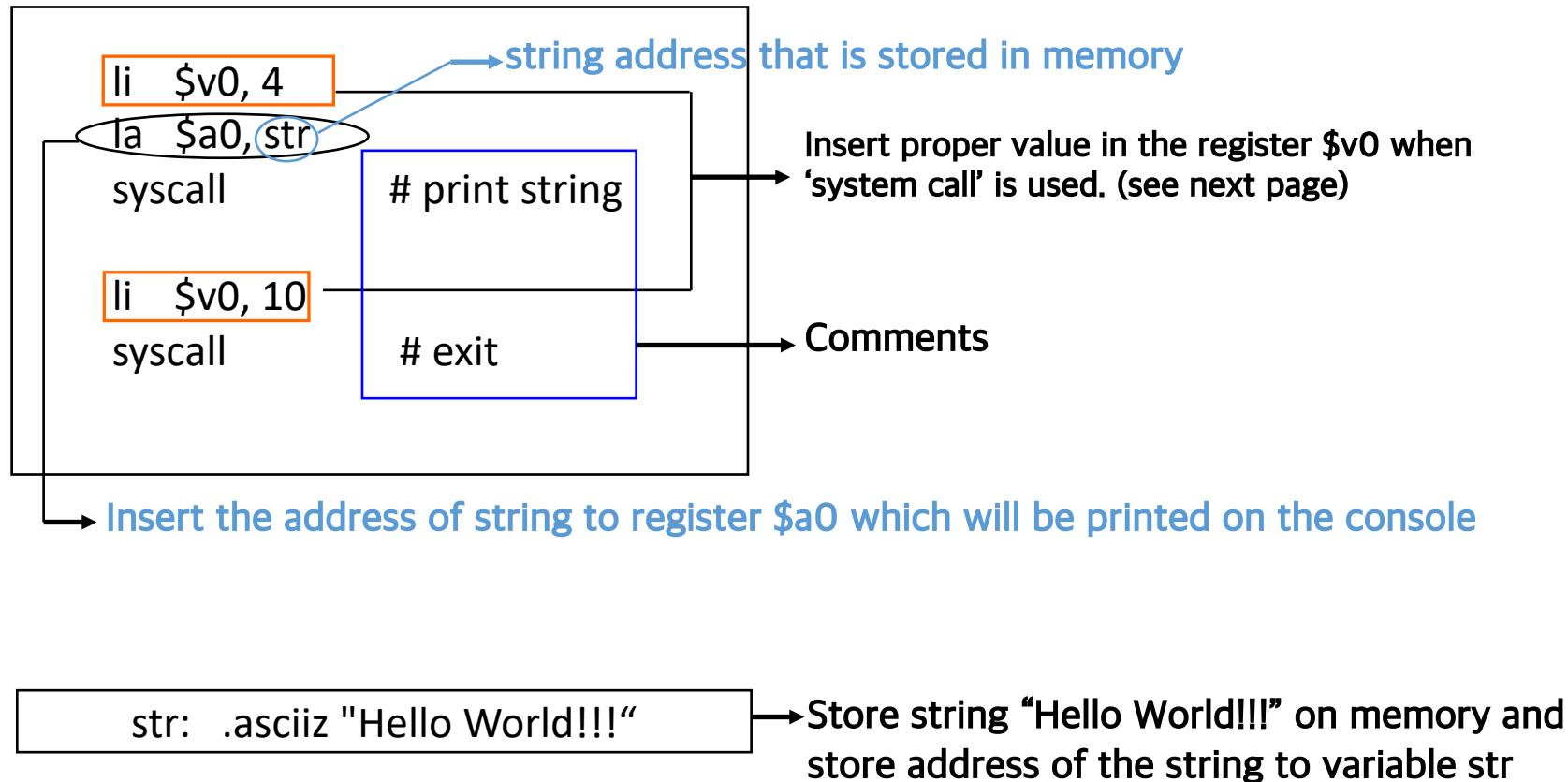
On the right, the generated assembly code is shown:

```
# test.s
# print "Hello World!!!" } Comments.
.text
.globl main
main:
    { li $v0, 4
      la $a0, str
      syscall # print string
    }
    { li $v0, 10
      syscall # exit
    }
.data
str: .asciiz "Hello World!!!"
```

Annotations explain specific parts of the assembly code:

- Comments: A brace groups the first two lines of assembly, with the annotation: "Comments."
- Indicates beginning of User text segment: An arrow points to ".text" with the annotation: "Indicates beginning of User text segment."
- Declare Main with global: An arrow points to ".globl main" with the annotation: "Declare Main with global. It could be referred by outside."
- Indicates beginning of Data segment: An arrow points to ".data" with the annotation: "Indicates beginning of Data segment"

# System call



# System Call (System services)

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	

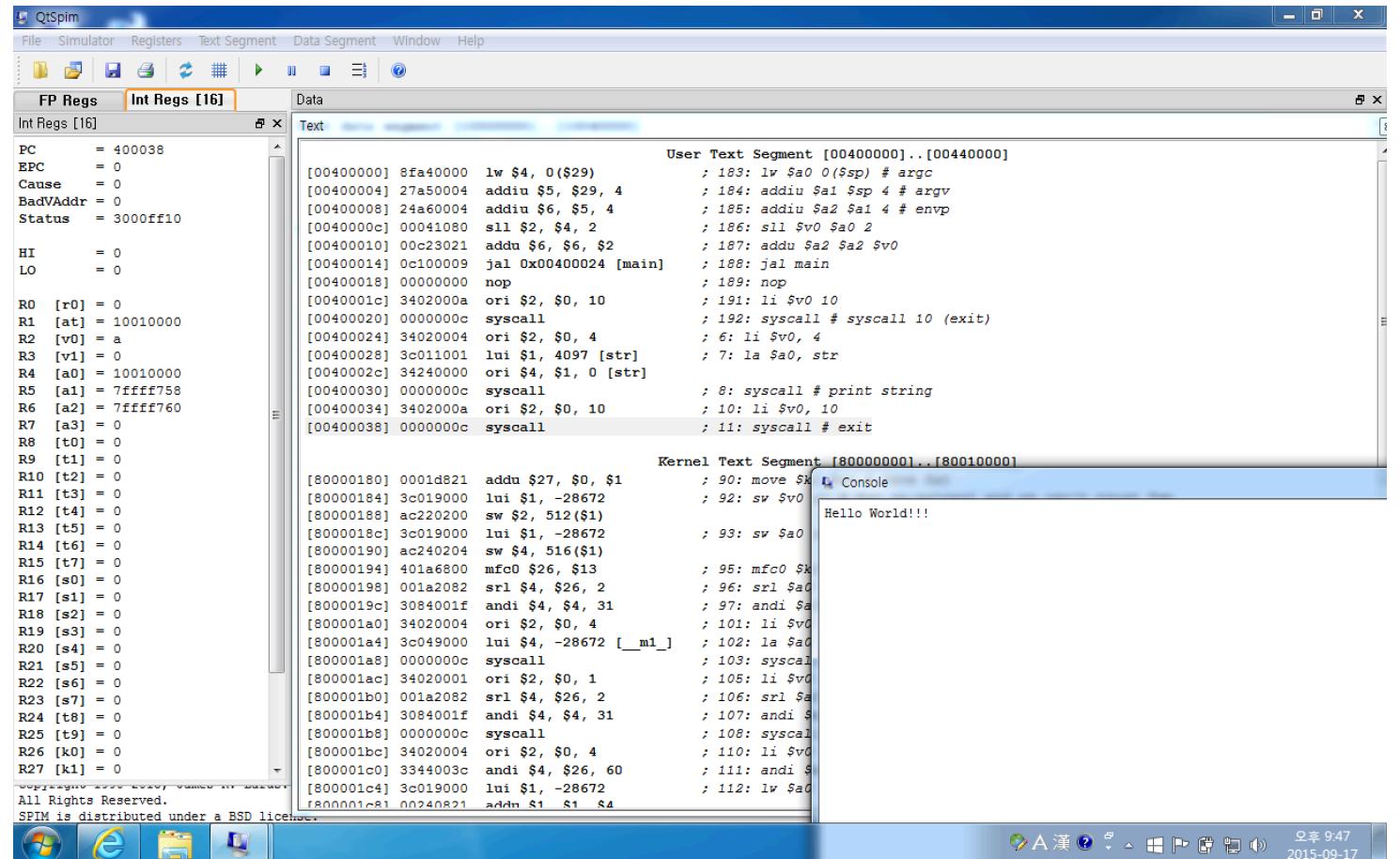
**FIGURE B.9.1 System services.**

# Previous Example

```
# test.s
# print "Hello World!!!"
.text
.globl main
main:
    li $v0, 4
    la $a0, str
    syscall    # print string

    li $v0, 10
    syscall    # exit

.data
str: .asciiz "Hello World!!!"
```



# Pseudo Assembly Instruction

- **move \$rt, \$rs**: copy contents of \$rs to \$rt
- **li \$rs, immed**: Load immediate value(immed) to \$rs
- **la \$rs, addr**: Load address(addr) to \$rs

# System Call (syscall)

- Load ‘system call code’ that you want to use on the register \$v0
- Use \$a0 ~ \$a3 to load argument (If value is floating-point, use \$f12)

The diagram illustrates the assembly code for printing strings and integers using system calls. It shows two separate sequences of assembly instructions, each ending with a `syscall` instruction.

**Print string in \$a0:**

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

**Print integer in \$a0:**

Annotations with arrows point from the `syscall` instructions to descriptive text:

- An arrow points from the first `syscall` to the text "System call code that prints string".
- An arrow points from the second `syscall` to the text "System call code that prints integer".

# System Call (syscall)

This program receives string from user and prints this string.

```
.data
buffer: .space 20
str1: .asciiz "Enter string(max 20 chars): "
str2: .asciiz "You wrote:\n"

.text
.globl main
main:
    la $a0, str1      # load and print str1
    li $v0, 4          # print_string system call code
    syscall

    li $v0, 8          # read_string system call code
    la $a0, buffer     # load the byte space for string
    move $t0, $a0        # save string to $t0
    syscall

    la $a0, str2      # load and print str2
    li $v0, 4          # print_string system call code
    syscall

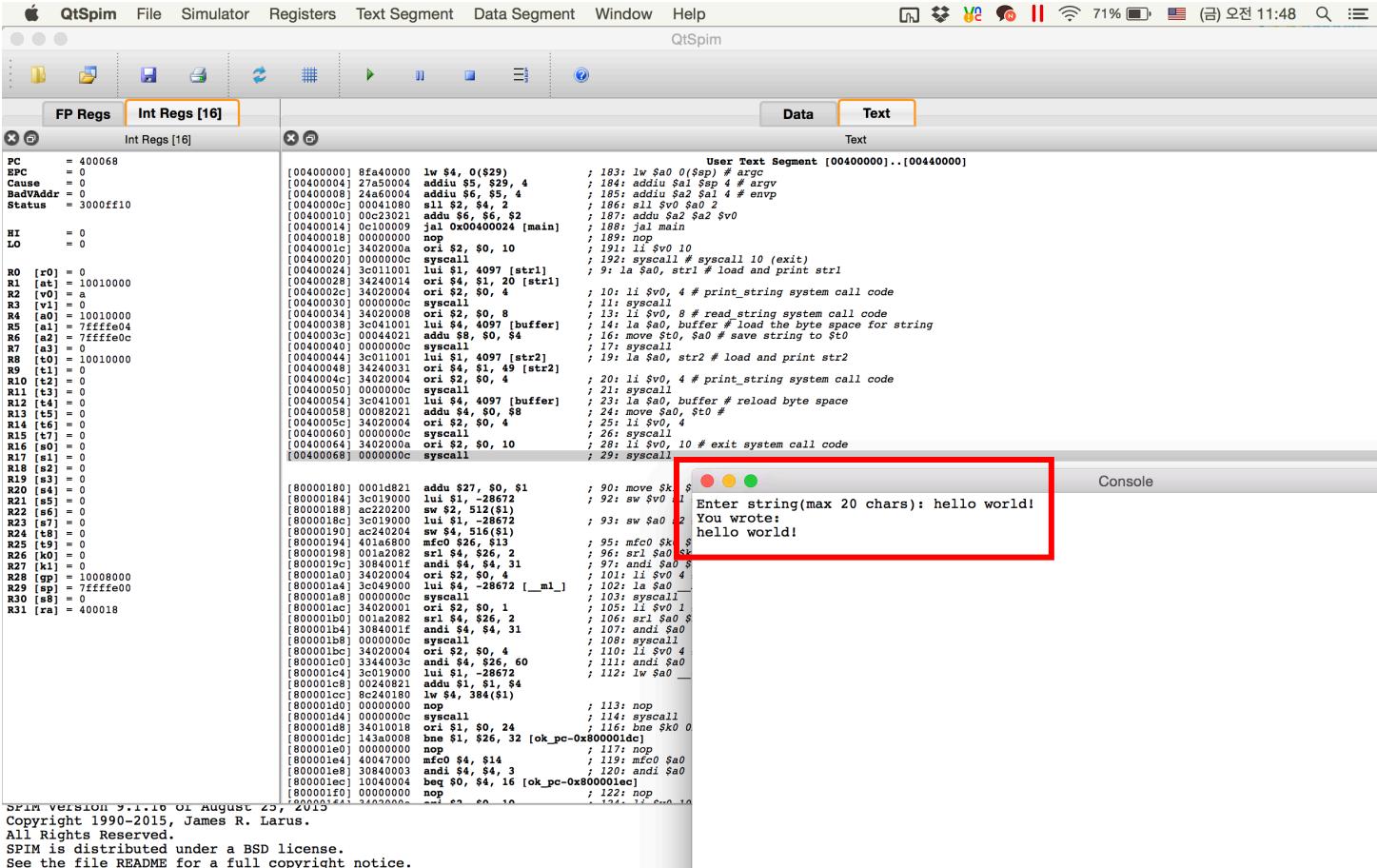
    la $a0, buffer     # reload byte space
    move $a0, $t0        # load string to $a0
    li $v0, 4
    syscall

    li $v0, 10         # exit system call code
    syscall
```

Declare the variables which will be used.

Code that receive string.

# System Call (syscall)



# Breakpoint

- When an appropriate instruction is determined, move the cursor to the instruction address and right-click. The right-click will display the breakpoint menu as shown in the image below.

# Breakpoint

QtSpim

FP Regs Int Regs [16] Data Text

	Int Regs [16]	Text
PC	- 40002c	[00400000] 24000001 addiu \$v0, \$v0, 4 , 100: addiu \$v0, \$v0, 4 # envp
EPC	- 0	[0040000c] 00041080 sll \$2, \$4, 2 ; 106: sll \$v0, \$a0, 2
Cause	= 0	[00400010] 00c23021 addu \$6, \$6, \$2 ; 107: addu \$a2, \$a2, \$v0
BadVAddr	- 0	[00400014] 0c100009 jal 0x00400024 [main] ; 108: jal main
Status	= 3000Ef10	[00400018] 00000000 nop ; 109: nop
HI	= 0	[0040001c] 3402000a ori \$2, \$0, 10 ; 110: li \$v0, 10
LO	= 0	[00400020] 0000000c syscall ; 111: syscall # syscall 10 (exit)
		[00400024] 3c011001 lui \$1, 4097 [hdr] ; 112: la \$a0, hdr
		[00400028] 34240000 ori \$4, \$1, 64 [hdr] ; 113: la \$a0, hdr
R0 [r0]	- 0	[0040002c] 34020004 ori \$2, \$0, 4 ; 114: li \$v0, 4
R1 [at]	= 10010000	[00400030] 0000000c syscall ; 115: syscall # print header
R2 [v0]	- 4	[00400034] 3c081001 lui \$8, 4097 [array] ; 116: la \$t0, array # set \$t0 addr of array
R3 [v1]	= 0	[00400038] 3c011001 lui \$1, 4097 ; 117: la \$t1, len # set \$t1 to length
R4 [a0]	- 10010040	[0040003c] 8c29000c lw \$9, 60(\$1) ; 118: lw \$s2, (\$t0) # set min, \$t2 to array[0]
R5 [a1]	- 7fffffa4c	[00400040] 8d120000 lw \$18, 0(\$8) ; 119: lw \$s3, (\$t0) # set max, \$t3 to array[0]
R6 [a2]	- 7fffffa54	[00400044] 8d130000 lw \$19, 0(\$8) ; 120: lw \$t4, (\$t0) # get array[n]
R7 [a3]	- 0	[00400048] 8d0c0000 lw \$12, 0(\$8) ; 121: bge \$t4, \$s2, NotMin # is new min?
R8 [t0]	= 0	[00400050] 10200002 beq \$1, \$0, 8 [NotMin-0x00400050] ; 122: beq \$t4, \$s2, NotMax # is new max?
R9 [t1]	- 0	[00400054] 000c9021 addu \$18, \$0, \$12 ; 123: move \$s2, \$t4 # set new min
R10 [t2]	- 0	[00400058] 026c082a slt \$1, \$19, \$12 ; 124: ble \$t4, \$s3, NotMax # is new max?
R11 [t3]	- 0	[0040005c] 10200002 beq \$1, \$0, 8 [NotMax-0x0040005c] ; 125: bne \$t4, \$s3, loop
R12 [t4]	- 0	[00400060] 000c9821 addu \$19, \$0, \$12 ; 126: move \$s3, \$t4 # set new max
R13 [t5]	- 0	[00400064] 2129FFFF addi \$9, \$9, -1 ; 127: sub \$t1, \$t1, 1 # decrement counter
R14 [t6]	- 0	[00400068] 25080004 addiu \$8, \$8, 4 ; 128: addu \$t0, \$t0, 4 # increment addr by word
R15 [t7]	- 0	[0040006c] 1520FFF7 bne \$9, \$0, -36 [loop-0x0040006c] ; 129: bne \$t4, \$s3, loop
R16 [s0]	= 0	[00400070] 3c011001 lui \$1, 4097 [al_msg] ; 130: la \$a0, al_msg
R17 [s1]	- 0	[0] Copy Ctrl+C , \$1, 105 [al_msg] ; 131: li \$v0, 4
R18 [s2]	= 0	[0] , \$0, 4 ; 132: syscall # print "min = "
R19 [s3]	- 0	[0] Select All Ctrl+A 1 ; 133: move \$a0, \$s2
R20 [s4]	- 0	[0] , \$0, \$18 ; 134: move \$a0, \$s2
R21 [s5]	= 0	[0] Set Breakpoint 1 ; 135: li \$v0, 1
R22 [s6]	- 0	[0] Clear Breakpoint 1 ; 136: syscall # print min
R23 [s7]	= 0	[0040008c] 3c011001 lui \$1, 4097 [new_ln] ; 137: la \$a0, new_ln # print a newline
R24 [t8]	- 0	[00400090] 34240067 ori \$4, \$1, 103 [new_ln] ; 138: li \$v0, 4
R25 [t9]	- 0	[00400094] 34020004 ori \$2, \$0, 4 ; 139: li \$v0, 4
R26 [k0]	= 0	[00400098] 0000000c syscall ; 140: syscall
R27 [k1]	- 0	[0040009c] 3c011001 lui \$1, 4097 [a2_msg] ; 141: la \$a0, a2_msg