

# 計算機組織與結構 – HOMEWORK1

撰寫MIPS程式 (共題，100分，滿分100分)

請於2025/12/02前上傳至M數位園區作業區繳交

請安裝QtSpim (<http://spimsimulator.sourceforge.net/>) 模擬器，並請詳細參考課本第二章及附錄A的介紹，於QtSpim模擬器環境下，撰寫一完整的MIPS核心指令集版本的程式。(需貼完整程式碼，截圖呈現結果並文字說明。)

(1) 實作第二章2.7小節範例if-then-else，請自行完成變數設定，觀察暫存器及記憶體狀態並說明程式之運作。(50分)

(2) 實作第二章2.7小節範例while迴圈，請自行完成變數設定，觀察暫存器及記憶體狀態並說明程式之運作。(50分)

Asian Edition

計算機組織與設計  
Computer Organization and Design

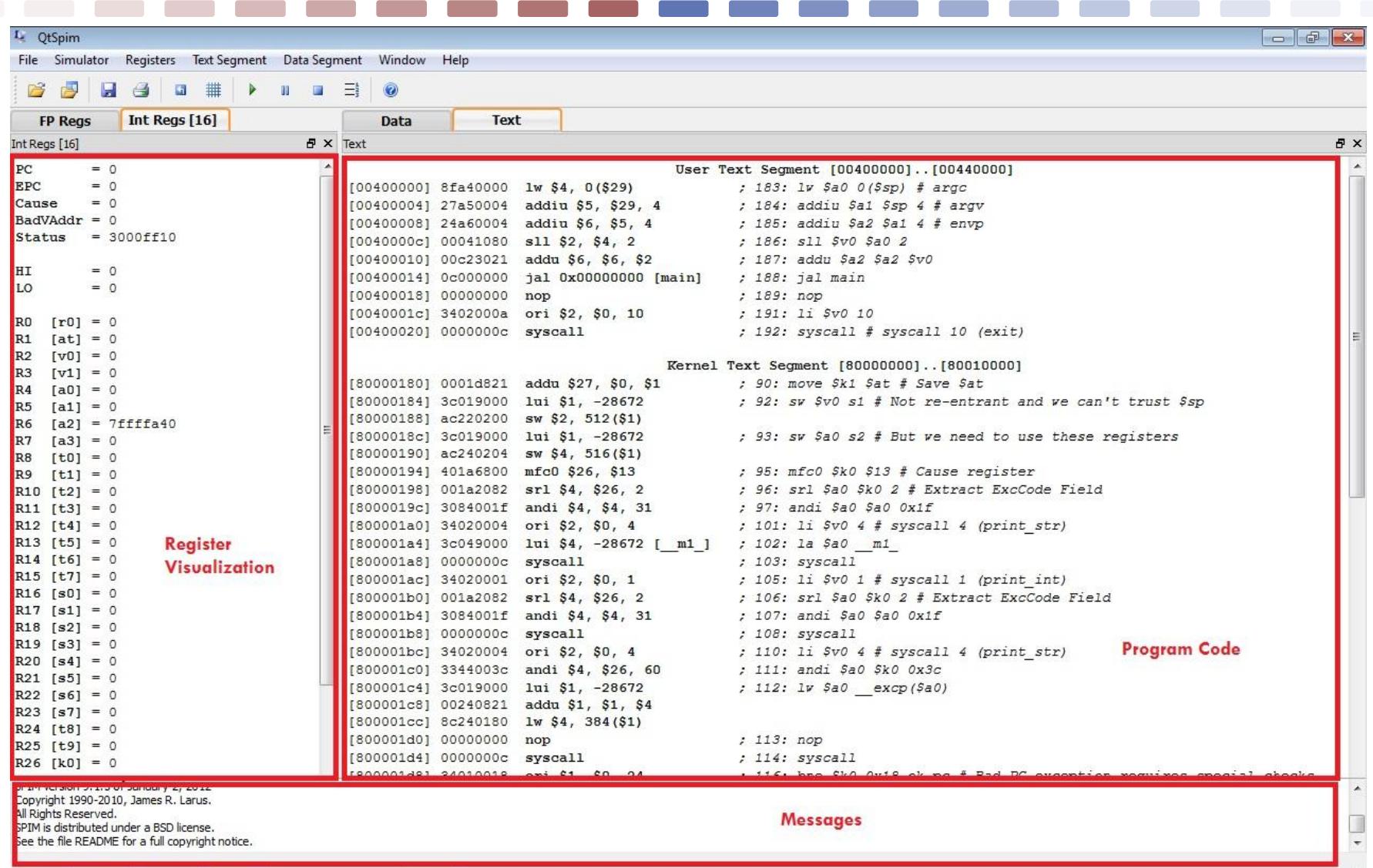


# QtSpim

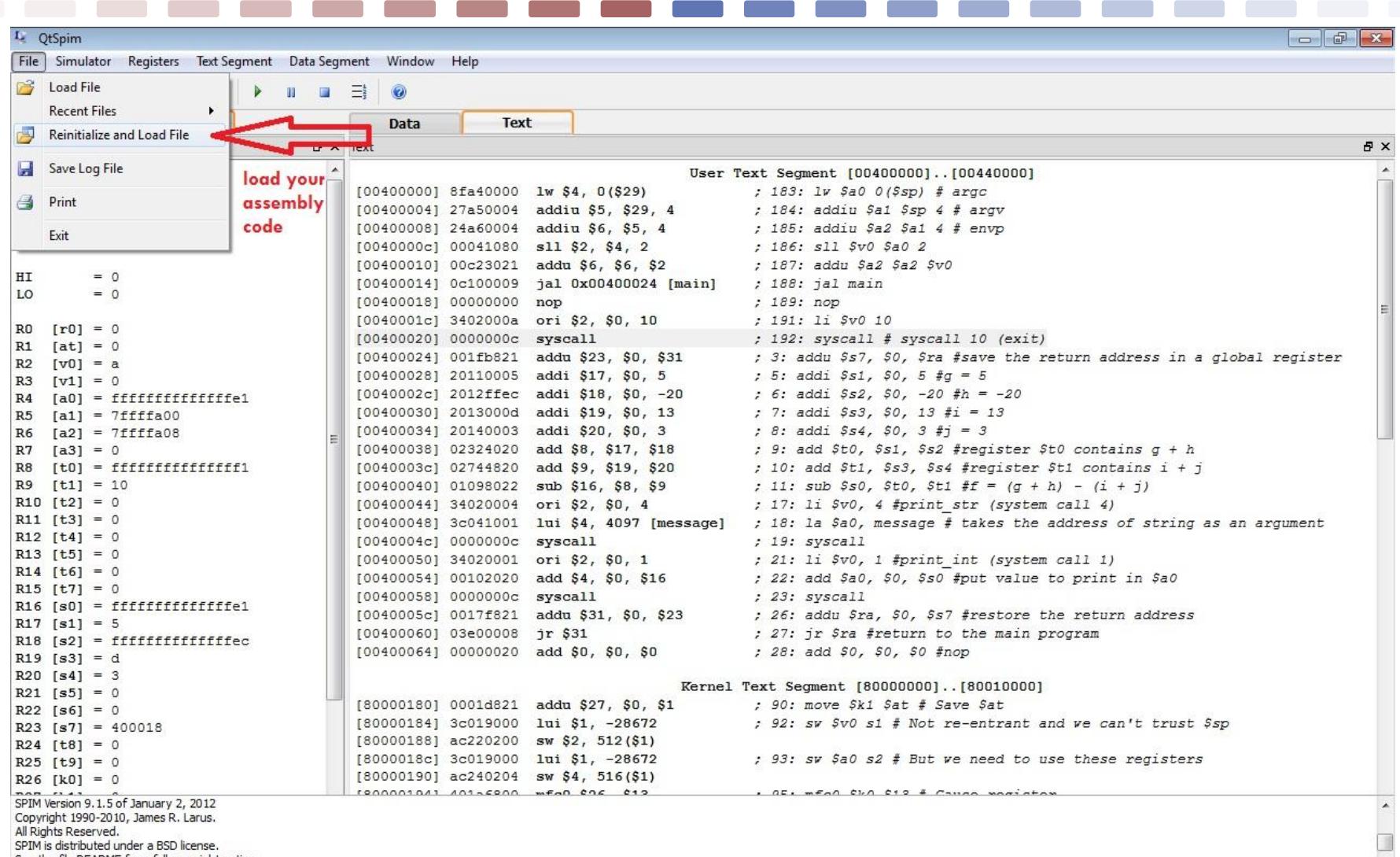
- *spim* is a simulator that runs MIPS32 programs
- It's been around for more than 20 years (improving over time).
- QtSpim is a new interface for *spim* built on the Qt UI framework which supports various platforms (Windows, Mac, Linux)
- It reads and executes assembly language programs.
- It contains a simple debugger



# Start SPIM



# Load Program



SPIM Version 9.1.5 of January 2, 2012  
Copyright 1990-2010, James R. Larus.  
All Rights Reserved.  
SPIM is distributed under a BSD license.  
See the file README for a full copyright notice.

# Execute Program

The screenshot shows the QtSpim simulator interface with the following details:

- Registers:** The left pane displays the Register window with the "Int Regs [16]" tab selected. It shows the following register values:
  - PC = 40002c
  - EPC = 0
  - Cause = 0
  - BadVAddr = 0
  - Status = 3000ff10
  - HI = 0
  - LO = 0
  - R0 [r0] = 0
  - R1 [at] = 0
  - R2 [v0] = 4
  - R3 [v1] = 0
  - R4 [a0] = 1
  - R5 [a1] = 7ffff638
  - R6 [a2] = 7ffff640
  - R7 [a3] = 0
  - R8 [t0] = 0
  - R9 [t1] = 0
  - R10 [t2] = 0
  - R11 [t3] = 0
  - R12 [t4] = 0
  - R13 [t5] = 0
  - R14 [t6] = 0
  - R15 [t7] = 0
  - R16 [s0] = 0
  - R17 [s1] = 5** (highlighted with a red arrow)
  - R18 [s2] = 0
  - R19 [s3] = 0
  - R20 [s4] = 0
  - R21 [s5] = 0
  - R22 [s6] = 0
  - R23 [s7] = 400018
  - R24 [t8] = 0
  - R25 [t9] = 0
  - R26 [k0] = 0
- Text Window:** The right pane displays the assembly code and its corresponding comments. The code starts at address 40000000 and ends at 40000640. A red arrow points to the instruction at address 4000002c, labeled "your assembly code starts". Another red arrow points to the current instruction at address 4000002c, labeled "current instruction".

```
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0e100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 190: li $v0 10
[00400020] 0000000c syscall ; 191: syscall # syscall 10 (exit)
[00400024] 001fb821 addu $23, $0, $31 ; 192: addu $s7, $0, $ra # save the return address in a global register
[00400028] 20110005 addi $17, $0, 5 ; 193: addi $s1, $0, 5 #g = 5
[0040002c] 2012ffec addi $18, $0, -20 ; 194: addi $s2, $0, -20 #h = -20 (highlighted with a red box)
[00400030] 2013000d addi $19, $0, 13 ; 195: addi $s3, $0, 13 #i = 13
[00400034] 20140003 addi $20, $0, 3 ; 196: addi $s4, $0, 3 #j = 3
[00400038] 02324020 add $8, $17, $18 ; 197: add $t0, $s1, $s2 #register $t0 contains g + h
[0040003c] 02744820 add $9, $19, $20 ; 198: add $t1, $s3, $s4 #register $t1 contains i + j
[00400040] 01098022 sub $16, $8, $9 ; 199: sub $s0, $t0, $t1 #f = (g + h) - (i + j)
[00400044] 34020004 ori $2, $0, 4 ; 200: li $v0, 4 #print_str (system call 4)
[00400048] 3c041001 lui $4, 4097 [message] ; 201: la $a0, message # takes the address of string as an argument
[0040004c] 0000000c syscall ; 202: syscall
[00400050] 34020001 ori $2, $0, 1 ; 203: li $v0, 1 #print_int (system call 1)
[00400054] 00102020 add $4, $0, $16 ; 204: add $a0, $0, $s0 #put value to print in $a0
[00400058] 0000000c syscall ; 205: syscall
[0040005c] 0017f821 addu $31, $0, $23 ; 206: addu $ra, $0, $s7 #restore the return address
[00400060] 03e00008 jr $31 ; 207: jr $ra #return to the main program
[00400064] 00000020 add $0, $0, $0 ; 208: add $0, $0, $0 #nop
```

Kernel Text Segment [80000000]..[80010000]

```
[80000180] 0001d821 addu $27, $0, $1 ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672 ; 92: sw $v0 $1 # Not re-entrant and we can't trust $sp
[80000188] ac220200 sw $2, 512($1) ; 93: sw $a0 $2 # But we need to use these registers
[8000018c] 3c019000 lui $1, -28672
[80000190] ac240204 sw $4, 516($1)
[80000194] 1015f800 mfc0 $0, $12 # Cause exception
```

SPIM Version 9.1.4 of September 4, 2011

Copyright 1990-2010, James R. Larus.

All Rights Reserved.

SPIM is distributed under a BSD license.

See the file README for a full copyright notice.

Single Step

# Program data

The screenshot shows the QtSpim simulator interface. The top menu bar includes File, Simulator, Registers, Text Segment, Data Segment, Window, and Help. The toolbar below has icons for file operations like Open, Save, and Print. The main window has tabs for FP Regs, Int Regs [16], and Data/Text. The Data tab is active, displaying memory dump sections:

- User data segment [10000000]..[10040000]**:  
[10000000]..[1000ffff] 00000000  
[10010000] 6568540a 6c617620 6f206575 20662066 . T h e v a l u e o f f  
[10010010] 203a7369 00000000 00000000 00000000 i s : . . . . .  
[10010020]..[1003ffff] 00000000
- User Stack [7ffff9fc]..[80000000]**:  
[7ffff9fc] 00000001  
[7ffffa00] 7ffffa98 00000000 7fffffe1 7fffffb6 . . . . .  
[7ffffa10] 7ffff9d 7fffff6c 7fffff53 7fffff2f . . . . 1 . . . S . . . / . . .  
[7ffffa20] 7ffffefd 7ffffee9 7fffffec 7fffffec2 . . . . .  
[7ffffa30] 7ffffe94 7ffffe7a 7ffffe63 7ffffe55 . . . . z . . . c . . . U . . .  
[7ffffa40] 7fffffd44 7fffffd06 7fffffcbe 7ffffca6 D . . . . .  
[7ffffa50] 7fffffc94 7ffffc7c 7ffffc61 7ffffc43 . . . . | . . . a . . . C . . .  
[7ffffa60] 7fffffc02 7ffffbeb 7ffffbd7 7ffffbca8 . . . . .  
[7ffffa70] 7ffffbb2 7ffffb88 7ffffb5f 7ffffb48 . . . . . \_ . . . H . . .  
[7ffffa80] 7ffffb35 7ffffb16 7fffffae1 7fffffacf 5 . . . . . . . . .  
[7ffffa90] 00000000 00000000 752f3a44 7373616d . . . . . D : / u m a s s  
[7ffffaa0] 6563652f 2f323332 70737471 732f6d69 / e c e 2 3 2 / q t s p i m / s  
[7ffffab0] 6c706d61 6c2f7365 35746365 646f632d a m p l e s / l e c t 5 - c o d  
[7ffffac0] 69732f65 656c706d 636c6163 7700732e e / s i m p l e c a l c . s . w  
[7ffffad0] 69646e69 3a433d72 6669575c 73776f64 i n d i r = C : \ W i n d o w s  
[7ffffae0] 45535600 4c464544 494474f4 3a433d52 . V S E D E F L O G D I R = C :  
[7ffffaf0] 6f72505c 6d617267 61746144 41634d5c \ P r o g r a m D a t a \ M c A  
[7fffffb00] 5c656566 6b736544 50706f74 65746f72 f e e \ D e s k t o p P r o t e  
[7fffffb10] 6f697463 5355006e 52505245 4c49464f c t i o n . U S E R P R O F I L  
[7fffffb20] 3a433d45 6573555c 4a5c7372 69747375 E = C : \ U s e r s \ J u s t i  
[7fffffb30] 754c206e 45535500 4d414e52 754a3d45 n L u . U S E R N A M E = J u  
[7fffffb40] 6e697473 00754c20 52455355 414d4f44 s t i n L u . U S E R D O M A  
[7fffffb50] 4a3d4e49 69747375 2d754c6e 54004350 I N = J u s t i n L u - P C . T  
[7fffffb60] 433d504d 73555c3a 5c737265 5453554a M P = C : \ U s e r s \ J U S T  
[7fffffb70] 317e4e49 7070415c 61746144 636f4c5c I N ~ 1 \ A p p D a t a \ L o c  
[7fffffb80] 545c6c61 00706d65 504d4554 5c3a433d a l \ T e m p . T E M P = C : \  
[7fffffb90] 72657355 554a5c73 4e495453 415c317e U s e r s \ J U S T I N ~ 1 \ A  
[7ffffbba0] 61447070 4c5c6174 6c61636f 6d65545c p p D a t a \ L o c a l \ T e m

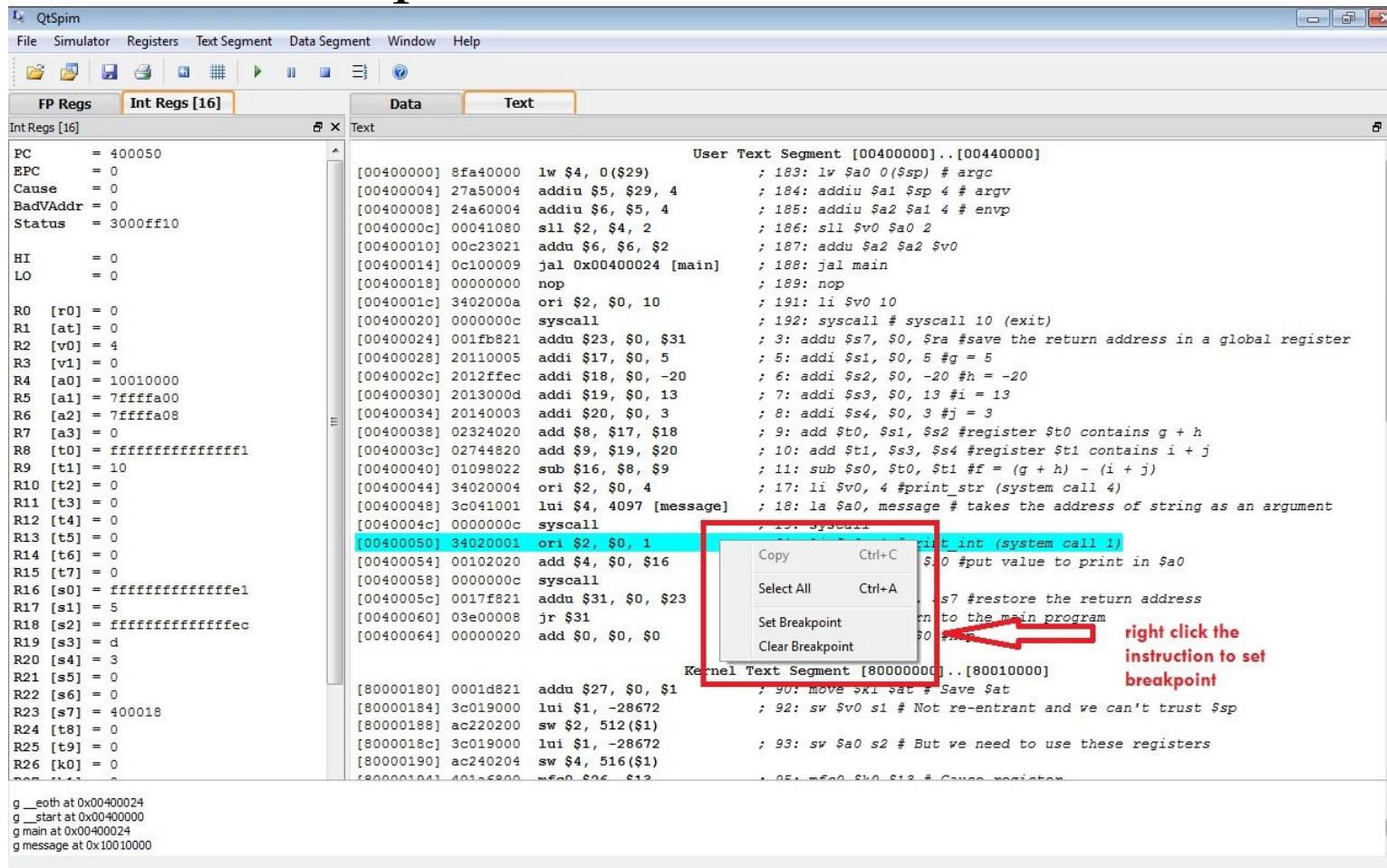
A red box highlights the text "User data segment" and "User Stack" sections. To the right of the "User data segment" section, the text "program data" is written in red.

At the bottom left, there is a copyright notice for SPIM Version 9.1.5:

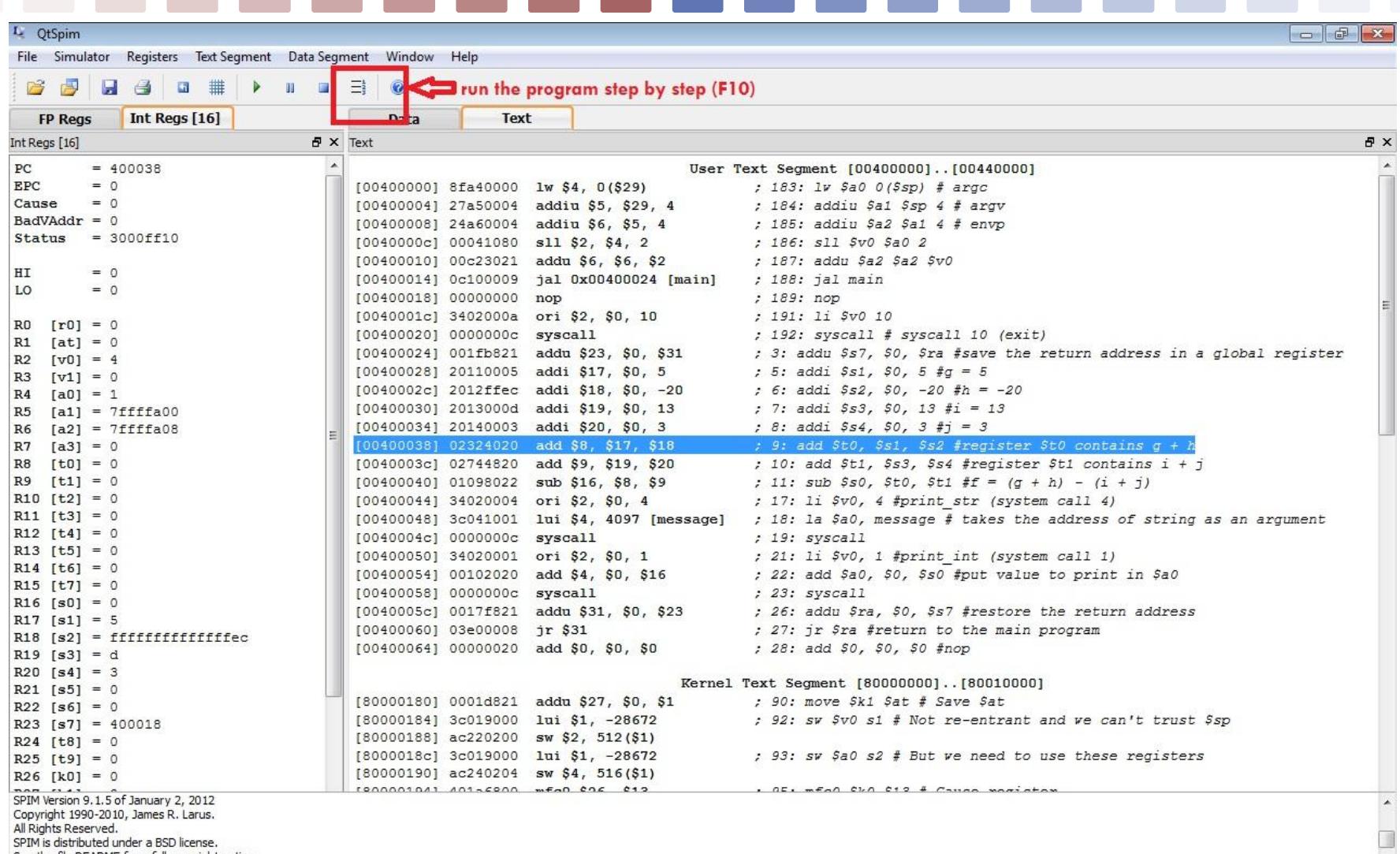
SPIM Version 9.1.5 of January 2, 2012  
Copyright 1990-2010, James R. Larus.  
All Rights Reserved.  
SPIM is distributed under a BSD license.  
See the file README for a full copyright notice.

# Set a break point

## Set a break point at the conditional instruction



# Debug by stepping your code line by line



The screenshot shows the QtSpim 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. A red box highlights the step-by-step execution icon (a blue circle with a white arrow) in the toolbar, with the text "run the program step by step (F10)" overlaid in red. The main window displays the assembly code in two panes: User Text Segment and Kernel Text Segment. The User Text Segment starts at address 00400000 and contains instructions like lw, addiu, sll, addu, jal, li, syscall, sub, addi, lui, jr, and move. The Kernel Text Segment starts at address 80000000 and contains instructions like addu, lui, sw, and move. On the left side, there are tabs for FP Regs, Int Regs [16], and Int Regs [16] (highlighted in orange). The bottom left corner shows the SPIM version information: SPIM Version 9.1.5 of January 2, 2012, Copyright 1990-2010, James R. Larus, All Rights Reserved, SPIM is distributed under a BSD license, See the file README for a full copyright notice.

# MIPS Hello World

```
# Hello, World!
```

```
.data ## Data declaration section
```

```
## String to be printed:
```

```
out_string: .asciiz "\nHello, World!\n"
```

```
.text ## Assembly language instructions go in text segment
```

```
main: ## Start of code section
```

```
    li $v0, 4 # system call code for printing string = 4
```

```
    la $a0, out_string # load address of string to be printed into $a0
```

```
    syscall # call operating system to perform operation
```

```
                # specified in $v0
```

```
                # syscall takes its arguments from $a0, $a1, ...
```

```
    li $v0, 10 # terminate program
```

```
    syscall
```

