CSE338

SPIM:MIPS Simulator

Zuhal Öztürk

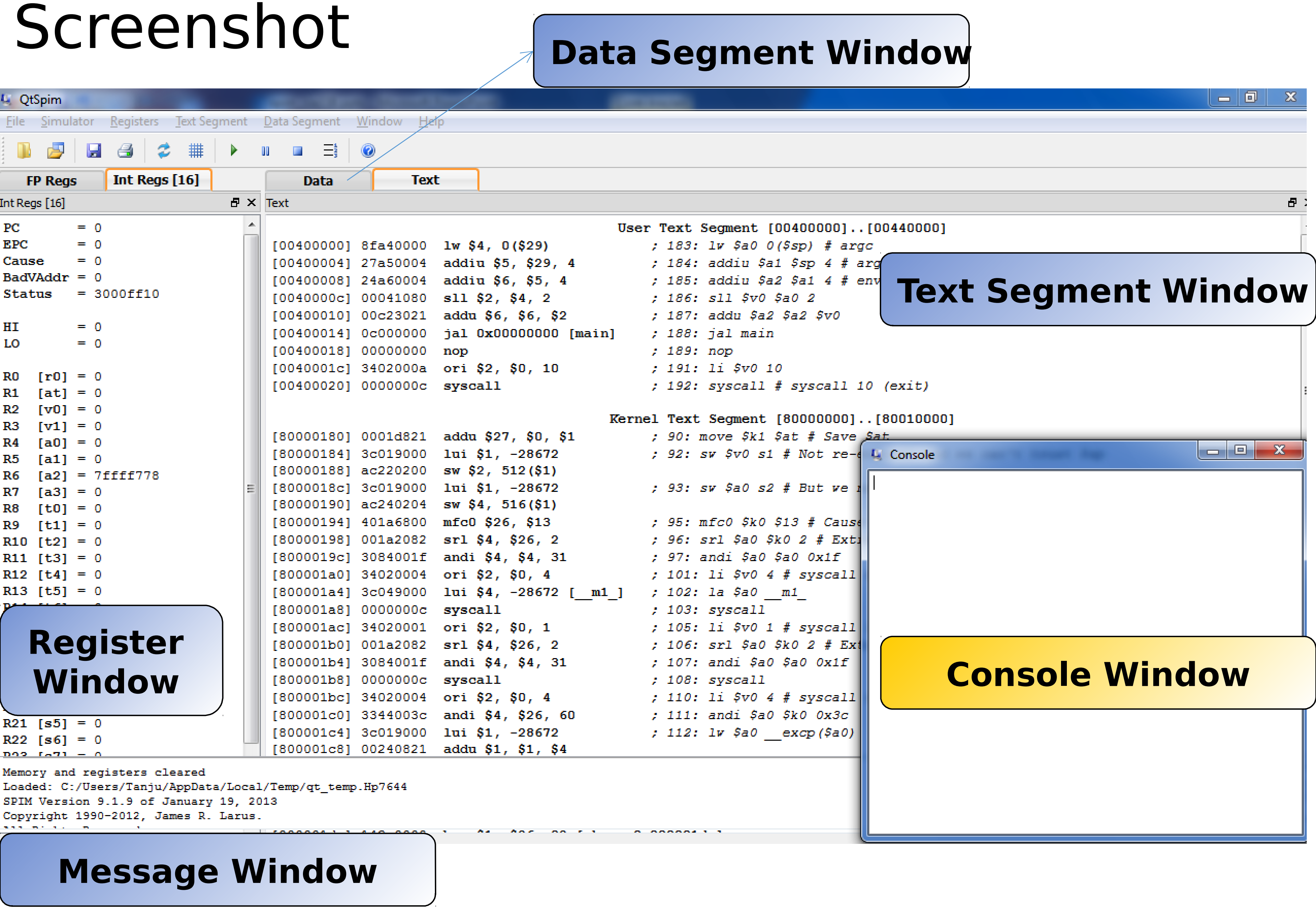
SPIM

- SPIM is a self-contained simulator that will run a MIPS32 assembly program and display the processor's registers and memory.
- SPIM reads and executes programs written in assembly language for a MIPS computer.
- Simulates MIPS-32 architecture
  - Fixed memory mapping
  - No cache structure
- Does not execute binary (compiled) programs.
- To simplify programming, *SPIM* provides a simple debugger and small set of operating system services.
- Slower than real computer, but low cost

SPIM Distributions

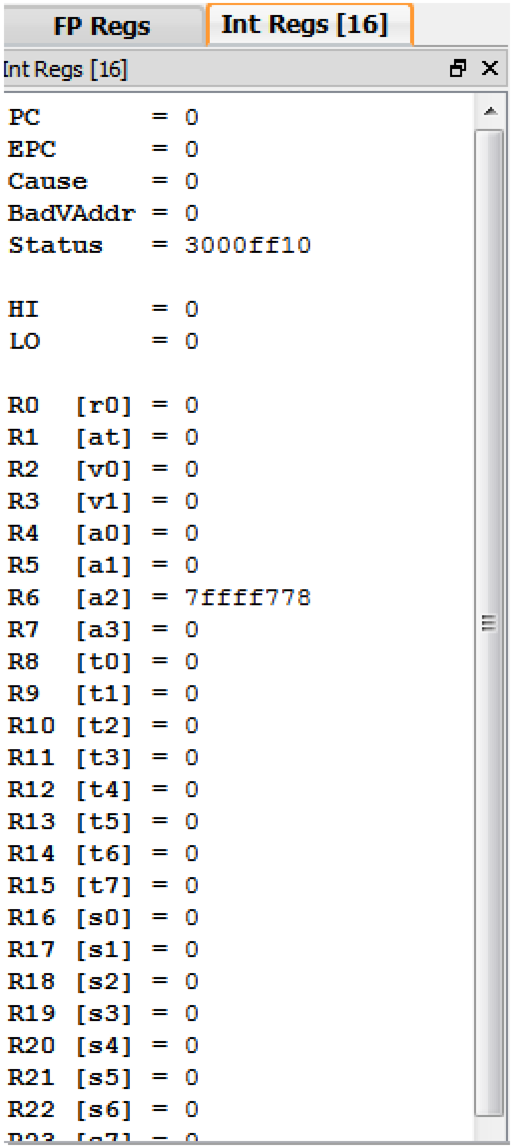
- The homepage of SPIM:
  - http://spimsimulator.sourceforge.net/
- Platform - Unix, Linux, Mac OS X, and Microsoft Windows
- Includes both command line (spim) and user interface version
- Download QtSpim for your platform from the SPIM website:
  - Unzip it
  - Run Setup.exe
- The features in the window look slightly different on Microsoft Windows than on Linux or Mac OSX, but all the menus and buttons are in the same place and work the same way

Screenshot


Data Segment Window
QtSpim
File Simulator Registers Text Segment Data Segment Window Help
FP Regs
Int Regs [16]
Data
Text
Int Regs [16]
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 0
R5 [a1] = 0
R6 [a2] = 7ffff778
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R21 [s5] = 0
R22 [s6] = 0
Text
User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # arg
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # env
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c000000 jal 0x00000000 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
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 s1 # Not re-
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672 ; 93: sw $a0 s2 # But we
[80000190] ac240204 sw $4, 516($1)
[80000194] 401a6800 mfc0 $26, $13 ; 95: mfc0 $k0 $13 # Cause
[80000198] 001a2082 srl $4, $26, 2 ; 96: srl $a0 $k0 2 # Ext
[8000019c] 3084001f andi $4, $4, 31 ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004 ori $2, $0, 4 ; 101: li $v0 4 # syscall
[800001a4] 3c049000 lui $4, -28672 [__m1_] ; 102: la $a0 __m1_
[800001a8] 0000000c syscall ; 103: syscall
[800001ac] 34020001 ori $2, $0, 1 ; 105: li $v0 1 # syscall
[800001b0] 001a2082 srl $4, $26, 2 ; 106: srl $a0 $k0 2 # Ext
[800001b4] 3084001f andi $4, $4, 31 ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c syscall ; 108: syscall
[800001bc] 34020004 ori $2, $0, 4 ; 110: li $v0 4 # syscall
[800001c0] 3344003c andi $4, $26, 60 ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000 lui $1, -28672 ; 112: lw $a0 __excp($a0)
[800001c8] 00240821 addu $1, $1, $4
Text Segment Window
Console
Console Window
Register Window
Memory and registers cleared
Loaded: C:/Users/Tanju/AppData/Local/Temp/qt_temp.Hp7644
SPIM Version 9.1.9 of January 19, 2013
Copyright 1990-2012, James R. Larus.
Message Window

Register Window

- It shows the values of all registers in the MIPS CPU and FPU
- This display is updated whenever your program stops running
- Displays PC, Stack Pointer, Frame Pointer, etc.
- Very important when debugging your code
- Registers are displayed as hexadecimal by default (but can be converted to the binary or decimal)



Text segment

- It displays instructions from both your program and the system code that is loaded automatically when QtSpim starts running.

Data | **Text**

Text

```
                                    User Text Segment [00400000]..[00440000]
[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] 0c000000  jal 0x00000000 [main]    ; 188: jal main
[00400018] 00000000  nop                      ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10           ; 191: li $v0 10
[00400020] 0000000c  syscall                  ; 192: syscall # syscall 10 (exit)

                                   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 s1 # Not re-entrant and we can't trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672           ; 93: sw $a0 s2 # But we need to use these registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13            ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2           ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31          ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4            ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]   ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                  ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1            ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2           ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31          ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                  ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4            ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60         ; 111: andi $a0 $k0 0x3c
```

Text segment

- Your instructions are displayed here
- From left to right:
  - Address where the instruction is stored
  - Binary machine code for the instruction
  - Assembly instruction (with registers represented as numbers)
  - Line number in your assembly source
  - Assembly instruction from your source (with registers as $s0, $v0, etc)
- Pseudo Instructions will be converted to one or more assembly instructions

One line on text segment

[0x00400000] 0x8fa40000 lw $4, 0($29) ;   89: lw $a0, 0($sp)

hexadecimal memory address of the instruction

instruction's numerical encoding in hexadecimal

instruction's mnemonic description

actual line from your assembly file
that produced the instruction

Data and stack segments

- It displays the data loaded into your program's memory and the data on the program's stack.
- Addresses are Byte Addressed
- Data is stored in words
- Data is displayed as hexadecimal by default (but can be converted to the binary or decimal)

```
Data    Text
Data
User data segment [10000000]..[10040000]
[10000000]..[1003ffff]  00000000


User Stack [7ffff770]..[80000000]
[7ffff770]    00000000  00000000  7fffffe1  7fffffba    . . . . . . . . . . . . . . . .
[7ffff780]    7fffffa1  7fffff67  7fffff36  7fffff22    . . . . g . . . 6 . . . " . . .
[7ffff790]    7ffffefe  7ffffeea  7ffffedd  7ffffec7    . . . . . . . . . . . . . . . .
[7ffff7a0]    7ffffea2  7ffffe78  7ffffe63  7ffffe4c    . . . . x . . . c . . . L . . .
[7ffff7b0]    7ffffe3e  7ffffabf  7ffffa81  7ffffa66    > . . . . . . . . . . . f . . .
[7ffff7c0]    7ffffa22  7ffffa10  7ffff9f8  7ffff9dd    " . . . . . . . . . . . . . . .
[7ffff7d0]    7ffff9bf  7ffff97e  7ffff967  7ffff932    . . . . ~ . . . g . . . 2 . . .
[7ffff7e0]    7ffff91e  7ffff90f  7ffff8f9  7ffff8d2    . . . . . . . . . . . . . . . .
[7ffff7f0]    7ffff8ac  7ffff89a  7ffff88b  7ffff870    . . . . . . . . . . . . p . . .
[7ffff800]    7ffff826  7ffff814  00000000  00000000    & . . . . . . . . . . . . . . .
[7ffff810]    00000000  646e6977  433d7269  69575c3a    . . . . w i n d i r = C : \ W i
[7ffff820]    776f646e  53560073  4f433039  4f544e4d    n d o w s . V S 9 0 C O M N T O
[7ffff830]    3d534c4f  505c3a43  72676f72  46206d61    O L S = C : \ P r o g r a m   F
[7ffff840]    73656c69  63694d5c  6f736f72  56207466    i l e s \ M i c r o s o f t   V
[7ffff850]    61757369  7453206c  6f696475  302e3920    i s u a l   S t u d i o   9 . 0
[7ffff860]    6d6f435c  376e6f6d  6f6f545c  005c736c    \ C o m m o n 7 \ T o o l s \ .
[7ffff870]    52455355  464f5250  3d454c49  555c3a43    U S E R P R O F I L E = C : \ U
[7ffff880]    73726573  6e61545c  5500756a  4e524553    s e r s \ T a n j u . U S E R N
[7ffff890]    3d454d41  6a6e6154  53550075  4f445245    A M E = T a n j u . U S E R D O
[7ffff8a0]    4e49414d  6e654c3d  006f766f  3d504d54    M A I N = L e n o v o . T M P =
[7ffff8b0]    555c3a43  73726573  6e61545c  415c756a    C : \ U s e r s \ T a n j u \ A
[7ffff8c0]    61447070  4c5c6174  6c61636f  6d65545c    p p D a t a \ L o c a l \ T e m
```

SPIM messages

- This pane is used by QtSpim to write messages.
  - Loading the exception handler
  - Loading your assembly file
  - Any errors that SPIM encounters
- This is where error messages appear.

```
See the file README for a full copyright notice.
Loaded: /usr/share/spim/exceptions.s
```

MIPS Assembly Code Layout

Typical Program Layout

```
        .text           #code section
        .globl main     #starting point: must be global
main:
        # user program code
        .data           #data section
        # user program data
```

MIPS Assembler Directives

- Top-level Directives:

- **.text**
  - indicates that following items are stored in the user text segment, typically instructions

- **.data**
  - indicates that following data items are stored in the data segment

- **.globl main**
  - declare that symbol **main** is global and can be referenced from other files

Data Types

- **.word** w1, …, wn
  - store n 32-bit quantities in successive memory words
- **.half** h1, …, hn
  - store n 16-bit quantities in successive memory halfwords
- **.byte** b1, …, bn
  - store n 8-bit quantities in successive memory bytes
- **.ascii** str
  - store the string in memory but do not null-terminate it
  - strings are represented in double-quotes “str”
  - special characters, eg. \n, \t, follow C convention
- **.asciiz** str
  - store the string in memory and null-terminate it

Data Types

- **.float** f1, …, fn
  - store n floating point single precision numbers in successive memory locations
- **.double** d1, …, dn
  - store n floating point double precision numbers in successive memory locations
- **.space** n
  - reserves n successive bytes of space
- **.align** n
  - align the next datum on a 2^n byte boundary.
  - For example, **.align 2** aligns next value on a word boundary.
  - **.align 0** turns off automatic alignment of .half, .word, etc. till next .data directive

Assembler Syntax

- **Comments** in assembler files begin with a sharp-sign (#).
- **Identifers** are a sequence of alphanumeric characters, underbars (_), and dots (.) that do not begin with a number.
- **Opcodes** for instructions are reserved words that are not valid identifiers.
- **Labels** are declared by putting them at the beginning of a line followed by a colon.

Memory Usage

- Text segment
  - Program instructions
  - Starting at address 0x00400000
- Data segment
  - Data accessed by the program
  - Static: 0x10000000-0x1000ffff
  - Dynamic: Starting at address 0x10010000
- Stack segment
  - Procedure call frames
  - Starting at address 0x7fffffff

System Calls

Service	Trap code	Input	Output
print_int	$v0 = 1	$a0 = integer to print	prints $a0 to standard output
print_float	$v0 = 2	$f12 = float to print	prints $f12 to standard output
print_double	$v0 = 3	$f12 = double to print	prints $f12 to standard output
print_string	$v0 = 4	$a0 = address of first character	prints a character string to standard output
read_int	$v0 = 5		integer read from standard input placed in $v0
read_float	$v0 = 6		float read from standard input placed in $f0
read_double	$v0 = 7		double read from standard input placed in $f0
read_string	$v0 = 8	$a0 = address to place string, $a1 = max string length	reads standard input into address in $a0
sbrk	$v0 = 9	$a0 = number of bytes required	$v0= address of allocated memory Allocates memory from the heap
exit	$v0 = 10		
print_char	$v0 = 11	$a0 = character (low 8 bits)	
read_char	$v0 = 12		$v0 = character (no line feed) echoed
file_open	$v0 = 13	$a0 = full path (zero terminated string with no line feed), $a1 = flags, $a2 = UNIX octal file mode (0644 for rw-r--r--)	$v0 = file descriptor
file_read	$v0 = 14	$a0 = file descriptor, $a1 = buffer address, $a2 = amount to read in bytes	$v0 = amount of data in buffer from file (-1 = error, 0 = end of file)
file_write	$v0 = 15	$a0 = file descriptor, $a1 = buffer address, $a2 = amount to write in bytes	$v0 = amount of data in buffer to file (-1 = error, 0 = end of file)
file_close	$v0 = 16	$a0 = file descriptor	

System Calls

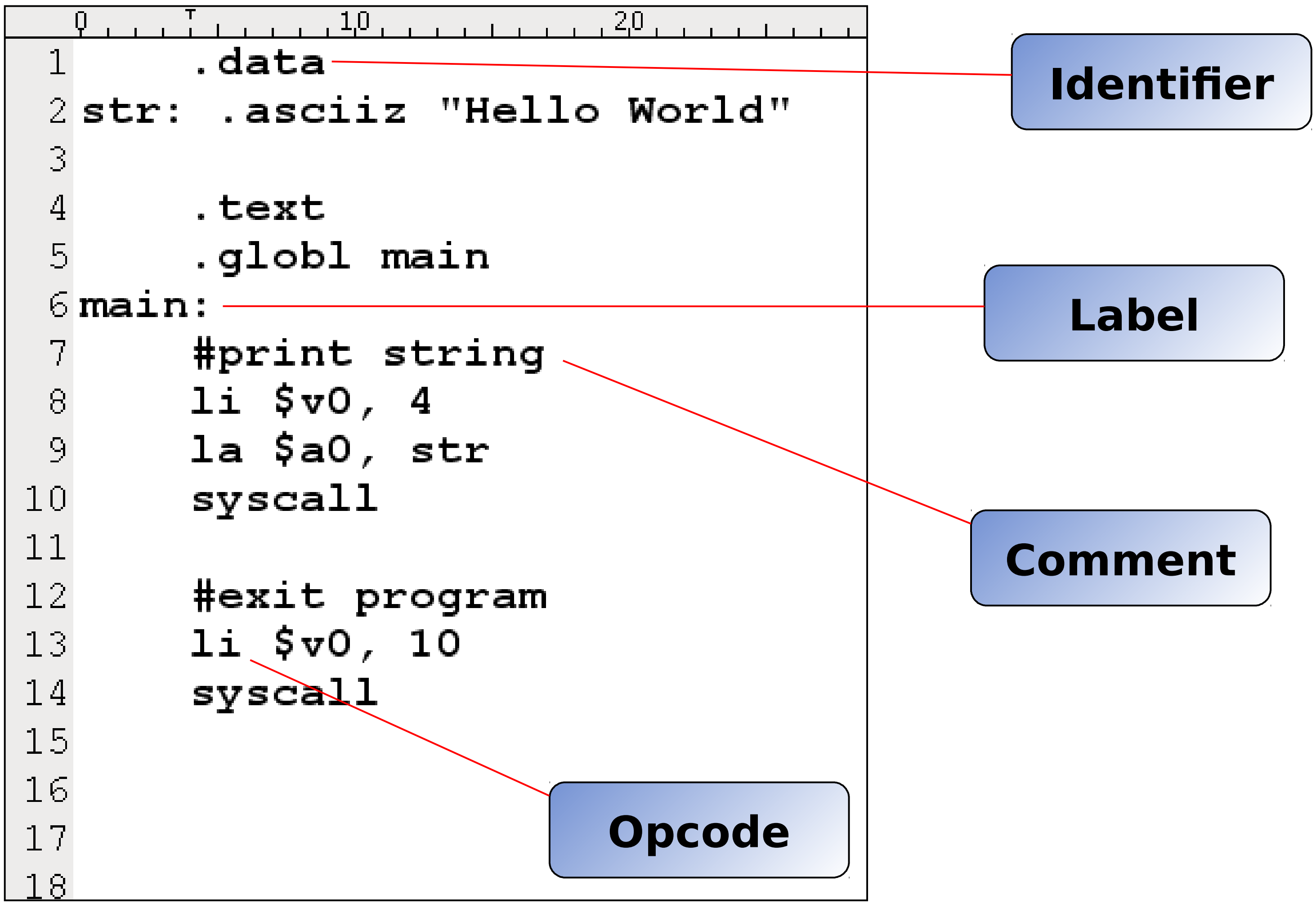
- syscall instruction
- System call code into $v0
- Arguments into $a0-$a3 or $f12
- Return values into $v0 or $f0

```
li $v0, 1 # call code 1 for print integer
li $a0, 5 # integer to print
syscall
```

Hello World Example - C

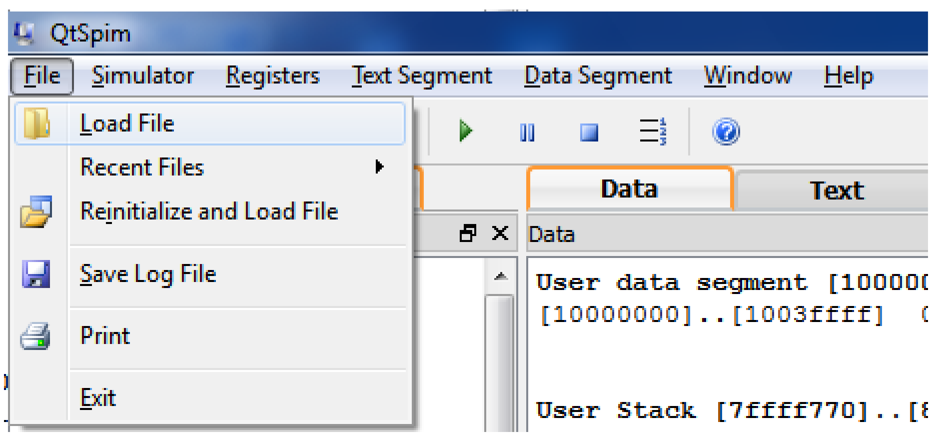
```
  int main()
{
        printf(“Hello World”);
        return 0;
}
```

Hello World Example - MIPS



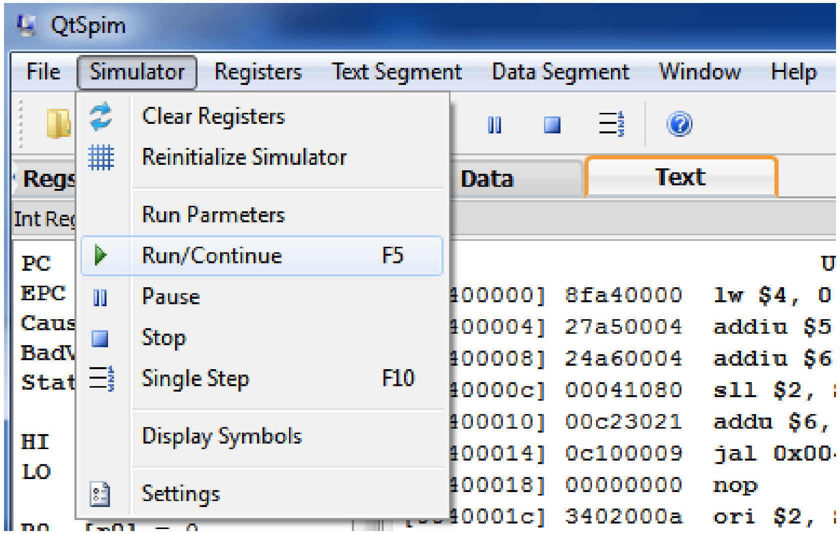
How to load program

- Your program should be stored in a file.
- Assembly code files usually have the extension ".s", as in file1.s.
- To load a file, go to the **File** menu and select **Load File**.
- The screen will change as the file is loaded, to show the instructions and data in your program.



How to run program

- To start a program running after you have loaded it, go to the **Simulator** menu and click **Run/Continue**.

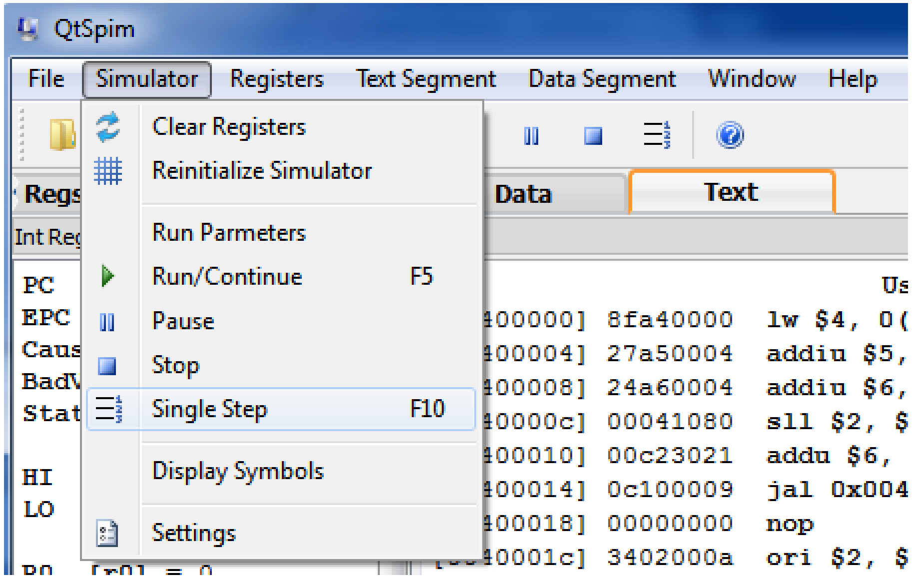


- Your program will run until it finishes or until an error occurs.

- Either way, you will see the changes that your program made to the MIPS registers and memory, and the output your program writes will appear in the Console window.

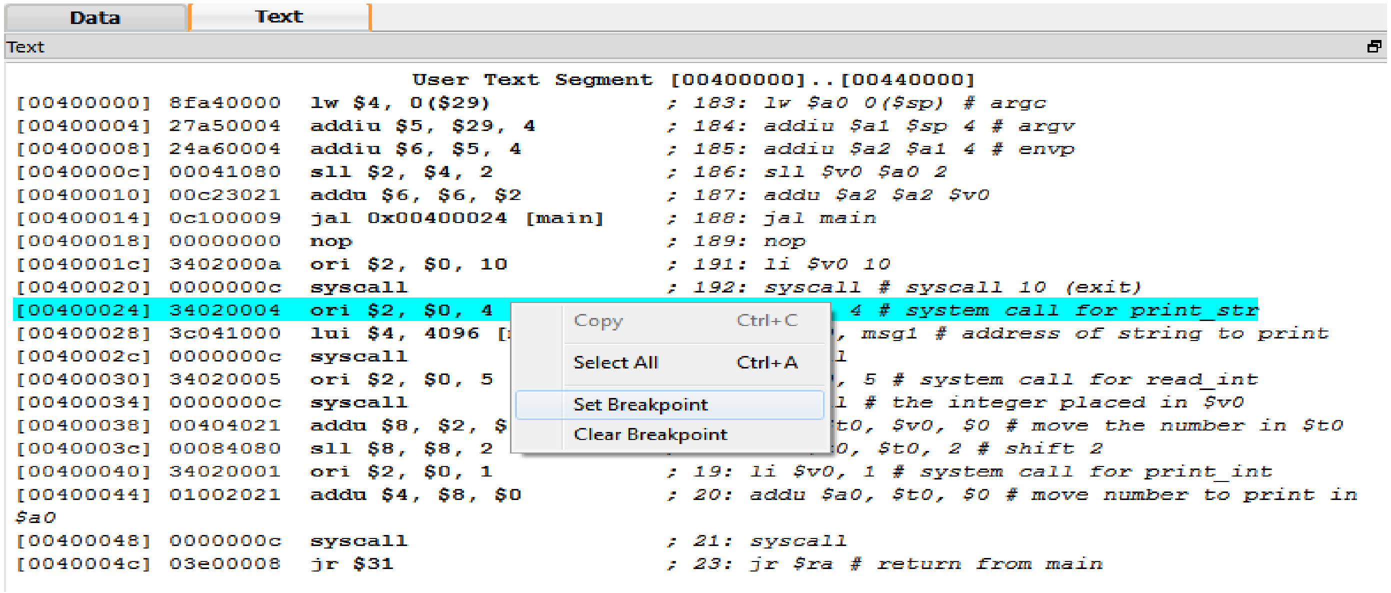
How to run program (cont.)

- If your program does not work correctly, there are several things you can do.
- The easiest is to single step between instructions, which lets you see the changes each instructions makes, one at a time.
- This command is also on the **Simulator** menu and is named **Single Step**.



How to debug program

- You set a breakpoint by right-clicking on the instruction where you want to stop, and selecting **Set Breakpoint**.



- When you are done with the breakpoint, you can remove it by selecting **Clear Breakpoint** instead.

How to reload program

- Two methods
  - quit and reload spim
    - Click on the *quit* button
  - Click on **Reinitialize and Load File** command on **File** menu
    - It first clears all changes made by a program, including deleting all of its instructions, and then reloads the last file.

SPIM example 1: add two numbers

```
#     $t2 - used to hold the sum of the $t0 and $t1.
#     $v0 - syscall number, and syscall return value.
#     $a0 - syscall input parameter.

      .text                   # Code area starts here
main:
      li  $v0, 5      # read number into $v0
      syscall         # make the syscall read_int
      move    $t0, $v0    # move the number read into $t0

      li  $v0, 5      # read second number into $v0
      syscall         # make the syscall read_int
      move    $t1, $v0    # move the number read into $t1

      add $t2, $t0, $t1

      move    $a0, $t2    # move the number to print into $a0
      li  $v0, 1      # load syscall print_int into $v0
      syscall         #

      li  $v0, 10     # syscall code 10 is for exit
      syscall         #
# end of main
```

Assembler directive starts with a dot

Special SPIM instruction: system call

SPIM example 2: sum N numbers

```
# Input: number of inputs, n, and n integers;   Output: Sum of integers
            .data                    # Data memory area.
prmpt1:     .asciiz "How many inputs? "
prmpt2:     .asciiz "Next input: "
sumtext:    .asciiz "The sum is "
            .text                  # Code area starts here
main:       li   $v0, 4        # Syscall to print prompt string
            la   $a0, prmpt1         # li and la are pseudo instr.
            syscall
            li   $v0, 5            # Syscall to read an integer
            syscall
            move $t0, $v0          # n stored in $t0

            li   $t1, 0                # sum will be stored in $t1
while:      blez $t0, endwhile       # (pseudo instruction)
            li   $v0, 4        # syscal to print string
            la   $a0, prmpt2
            syscall
            li   $v0, 5
            syscall
            add  $t1, $t1, $v0     # Increase sum by new input
            sub  $t0, $t0, 1       # Decrement n
            j    while

endwhile:   li   $v0, 4            # syscal to print string
            la   $a0, sumtext
            syscall
            move $a0, $t1             # Syscall to print an integer
            li   $v0, 1
            syscall
            li   $v0, 10              # Syscall to exit
            syscall
```

References

- Patterson and Hennessy, *Computer Organization and Design, 4th edition*