Introduction to SPIM Simulator

Jayaraj P B

SPIM Simulator

- SPIM is a software simulator that runs programs written for MIPS processors
- SPIM's name is just MIPS spelled backwards
- SPIM can read and immediately execute MIPS assembly language files or MIPS executable files
- SPIM contains a debugger and provides a few operating system-like services

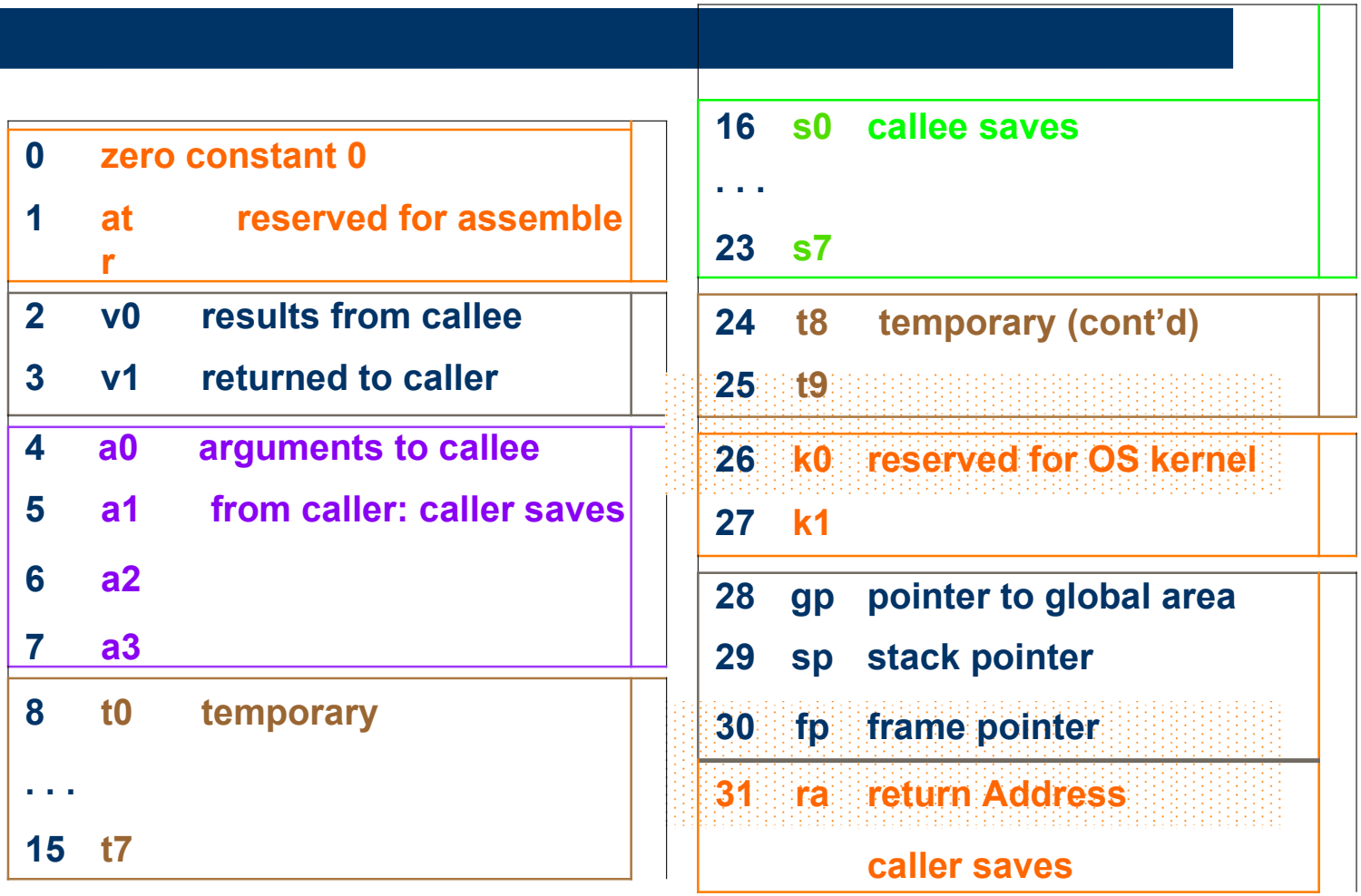
MIPS Processors

- MIPS is a load-store architecture, which means that only load and store instructions access memory
- Computation instructions operate only on values in registers

MIPS Registers

Name	Number	Usage
$zero	0	constant 0
$at	1	reserved for assembler
$v0~$v1	2~3	return value of a function
$a0~$a3	4~7	arguments
$t0~$t7	8~15	temporary (not preserved across call)
$s0~$s7	16~23	saved temporary (preserved across call)
$t8~$t9	24~25	temporary (not preserved across call)
$k0~$k1	26~27	reserved for OS kernel
$gp	28	pointer to global area
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

MIPS: Software Conventions for Registers



Pseudoinstructions

- Here's a list of useful pseudo-instructions.
- **mov $t0, $t1:** Copy contents of register t1 to register t0.
- **li $s0, immed:** Load immediate into to register **s0**.
  - The way this is translated depends on whether **immed** is 16 bits or 32 bits.
- **la $s0, addr:** Load address into to register **s0**.
- **lw $t0, address:** Load a word at address into register t0
- Similar pseudo-instructions exist for **sw**, etc.

Pseudoinstructions

- **Translating Some Pseudoinstructions**
- **mov $t0, $s0** addi $t0, $s0, 0
- **li $rs, small** addi $rs, $zero, small
- **li $rs, big** lui $rs, upper(big) ori $rs, $rs, lower(big) lui
- **la $rs, big** $rs, upper(big) ori $rs, $rs, lower(big)

- where **small** means a quantity that can be represented using 16 bits, and **big** means a 32 bit quantity. **upper(big)** is the upper 16 bits of a 32 bit quantity. **lower(big)** is the lower 16 bits of the 32 bit quantity.
- **upper(big)** and **lower(big)** are not real instructions. If you were to do the translation, you'd have to break it up yourself to figure out those quantities.

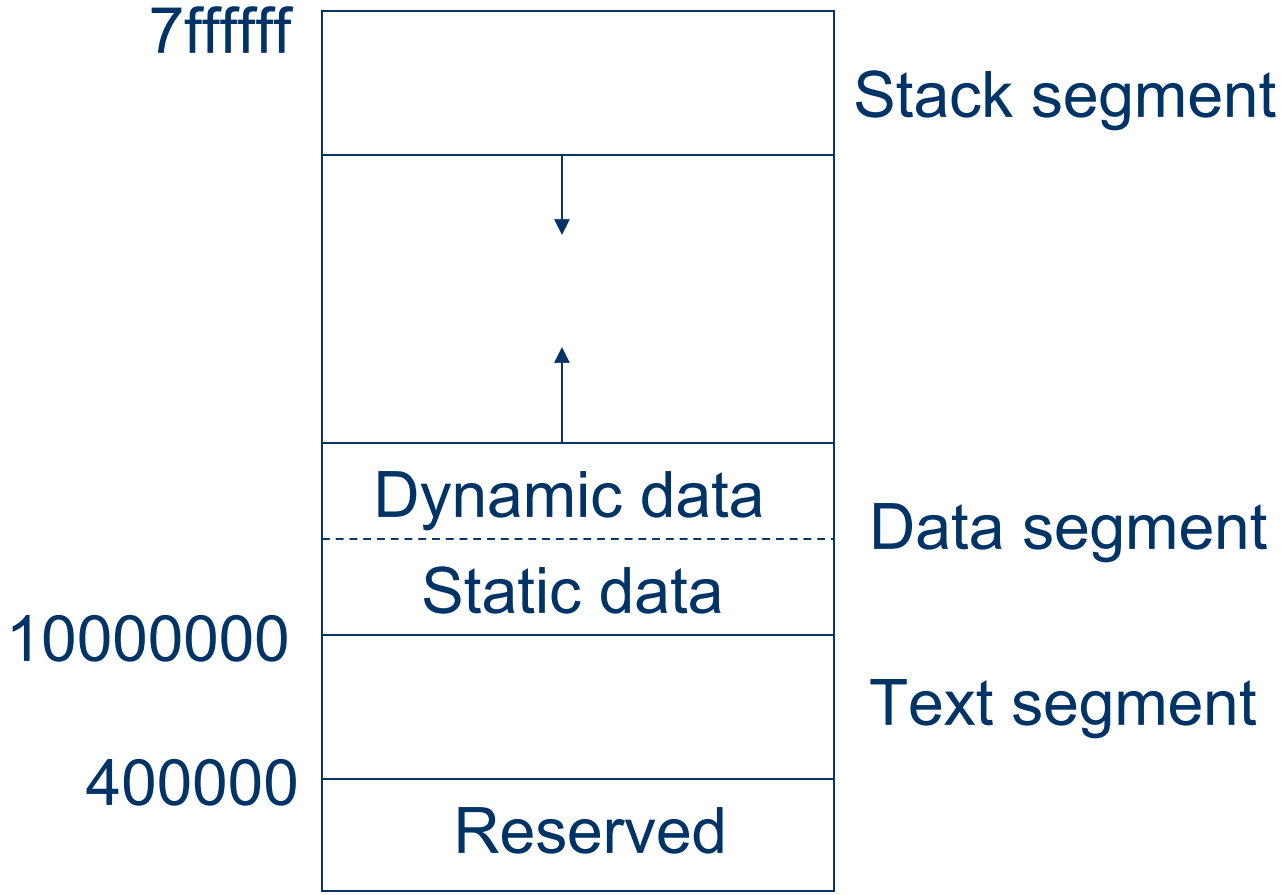
Arithmetic Instructions

add	rd, rs, rt	rd ← rs + rt
sub	rd, rs, rt	rd ← rs – rt
mul	rd, rs, rt	rd ← rs * rt
div	rd, rs, rt	rd ← rs / rt
rem	rd, rs, rt	rd ← rs % rt
neg	rd, rs	rd ← - rs

Branch Instructions

beq	rs, rt, label	branch to label if rs == rt
bne	rs, rt, label	branch to label if rs != rt
bgt	rs, rt, label	branch to label if rs > rt
bge	rs, rt, label	branch to label if rs >= rt
blt	rs, rt, label	branch to label if rs < rt
ble	rs, rt, label	branch to label if rs <= rt
b	label	branch to label

Memory Layout



MIPS Assembler Directives

- Top-level Directives:

.text

- indicates that following items are stored in the user text seg ment, typically instructions

.data

- indicates that following data items are stored in the data seg ment

.globl sym

- declare that symbol sym is global and can be referenced fro m other files

MIPS Assembler Directives

- Common Data Definitions:
  - **.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

MIPS Assembler Directives

- Common Data Definitions:
  - **.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

System Calls

- SPIM provides a small set of operating system-like services through the system call (syscall) instruction
- To request a service, a program loads the system call code into register $v0 and arguments into registers $a0~$a3
- System calls that return values put their results in register $v0

System Call Code

Service	Code (put in $v0)	Arguments	Result
print_int	1	$a0=integer	
print_float	2	$f12=float	
print_double	3	$f12=double	
print_string	4	$a0=addr. of string	
read_int	5		int 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	addr in $v0
exit	10		

QtSPIM

- QtSpim is software that will help you to simulate the execution of MIPS assembly programs.
- It does a context and syntax check while loading an assembly program.
- In addition, it adds in necessary overhead instructions as needed, and updates register and memory content as each instruction is executed.
- Download the source from the SourceForge.org link at: http://pages.cs.wisc.edu/~larus/spim.html
- Alternatively, you can go directly to: http://sourceforge.net/projects/spimsimulator/files/
- Versions for Windows, Linux, and Macs are all available

QtSPIM

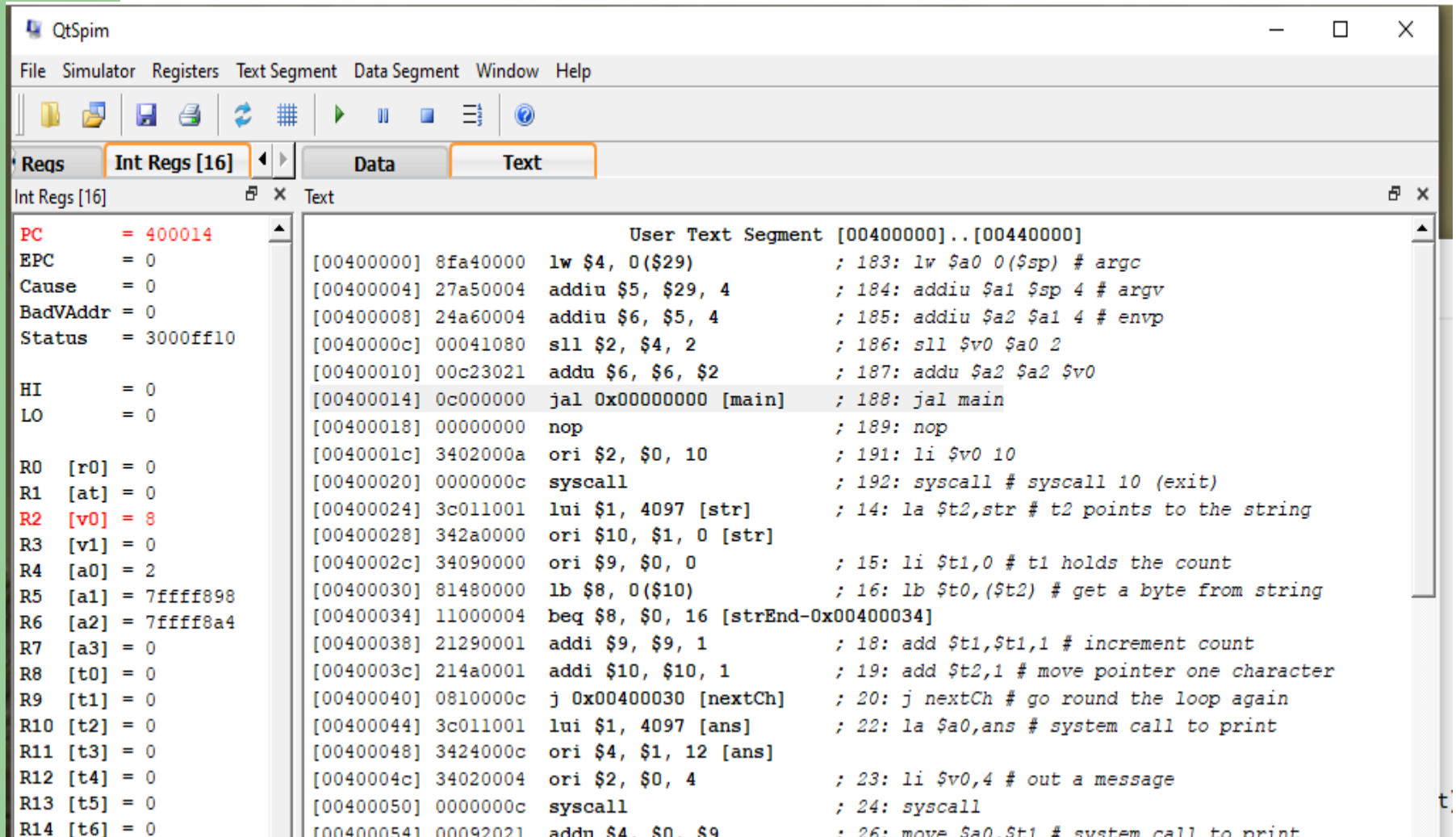
- QtSPIM window is divided into different sections:

1. The *Register tabs display the content of all registers.*
2. Buttons across the top are used to load and run a simulation
   - Functionality is described in Figure 2.
3. The *Text tab displays the MIPS instructions loaded into memory to be executed.*
   - From left-to-right, the memory address of an instruction,

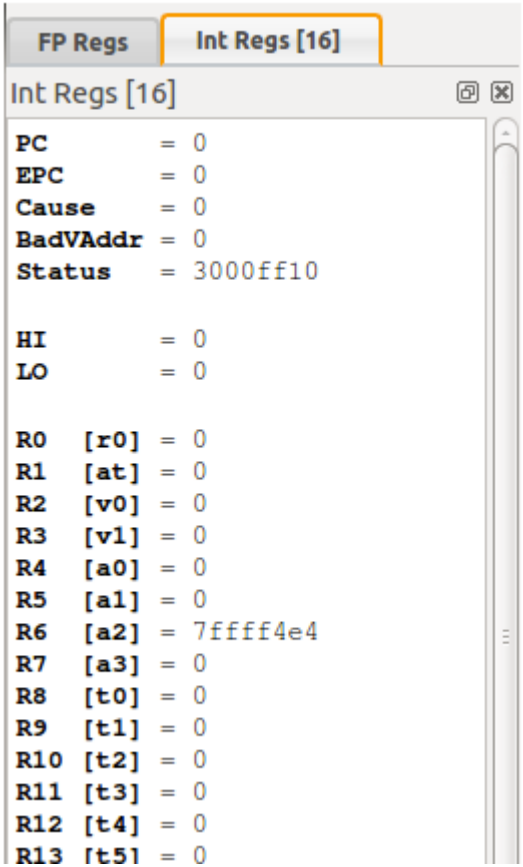
the contents of the address in hex, the actual MIPS instructions where register numbers are used, the MIPS assembly that you wrote, and any comments you made in your code are displayed.

4. The *Data tab displays memory addresses and their values in the data and stack segments* of the memory.
5. The *Information Console lists the actions performed by the simulator.*

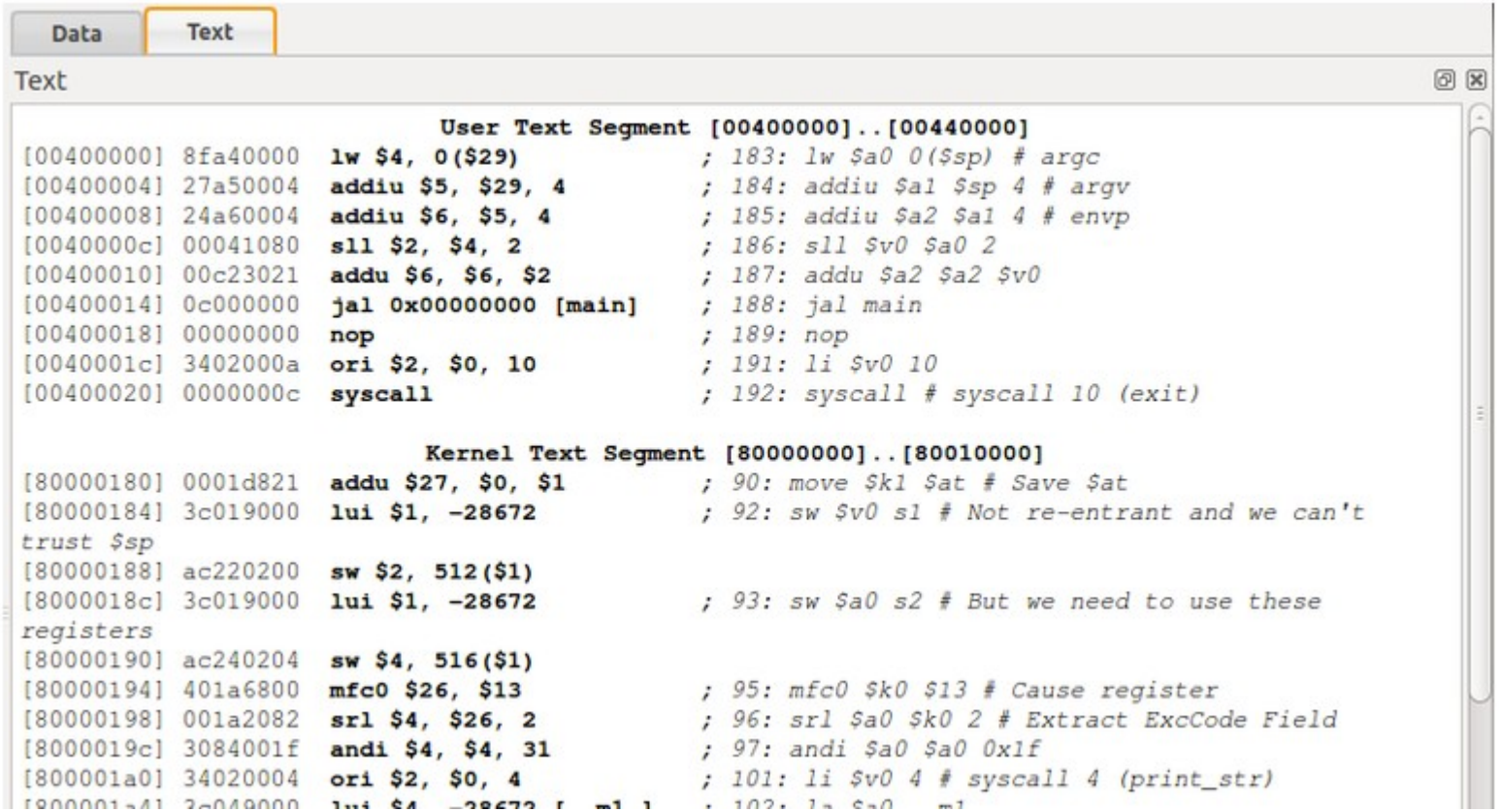
QtSpim simulator - screenshot



Register panel



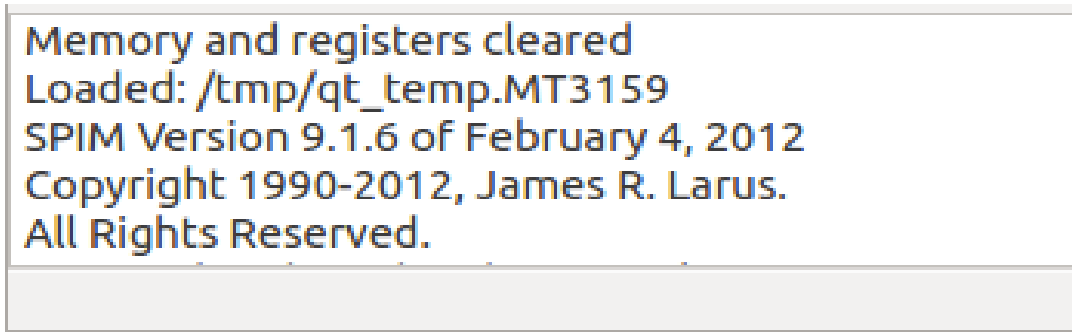
Memory panel: text panel



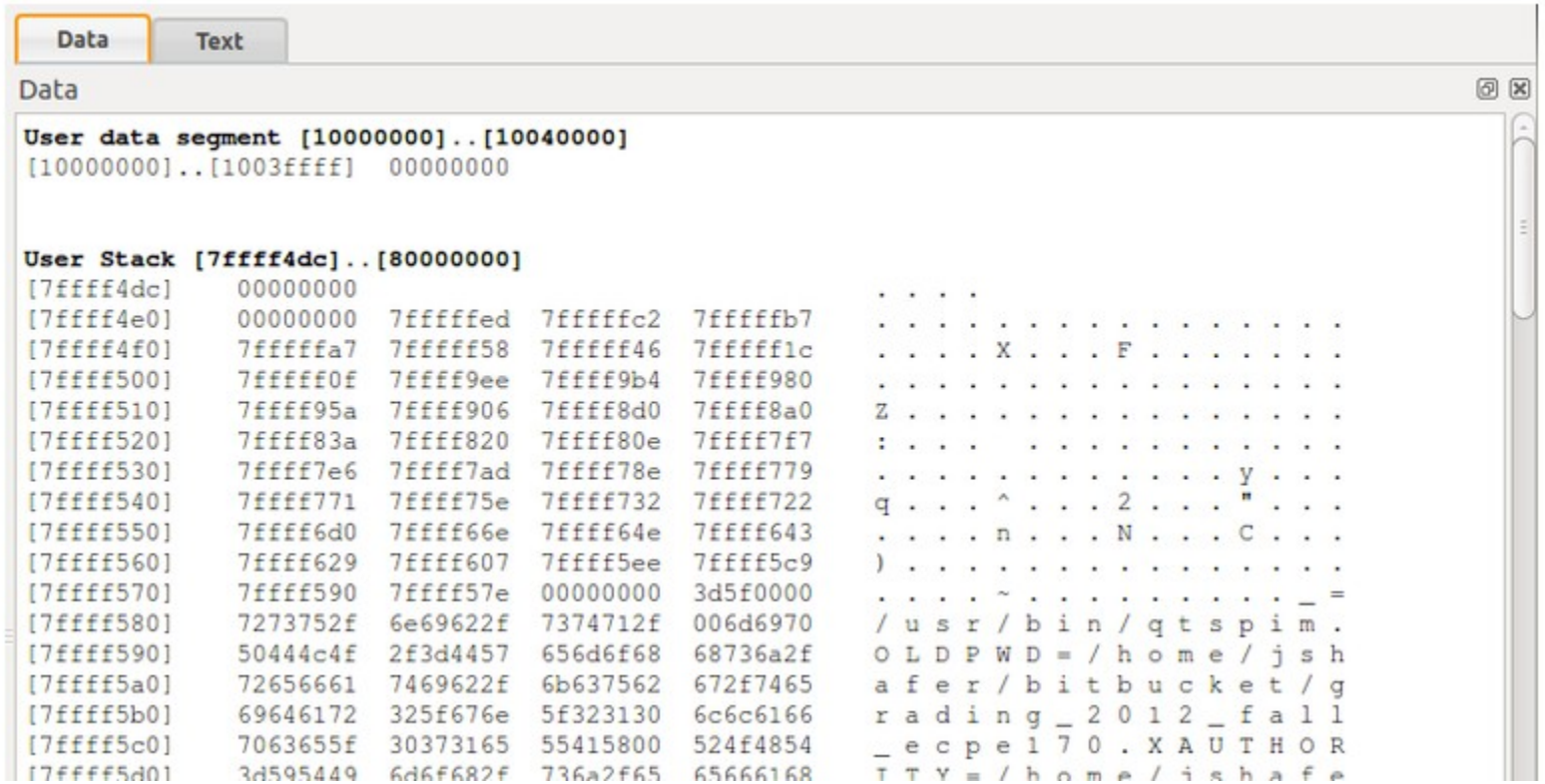
Message panel

Messages Panel

The Messages panel displays messages from QtSPIM to the user.



Memory panel :data segment



Program template

```
            .data
              # data segment
            .text
            .globl main
main:
            # your code will come here

EXIT:  li $v0,10
            syscall
```

Program for reading a string

```
.data
theString:    .space 64
    .text
main:
    li      $v0, 8
    la      $a0, theString
    li      $a1, 64
    syscall
    jr      $ra
```

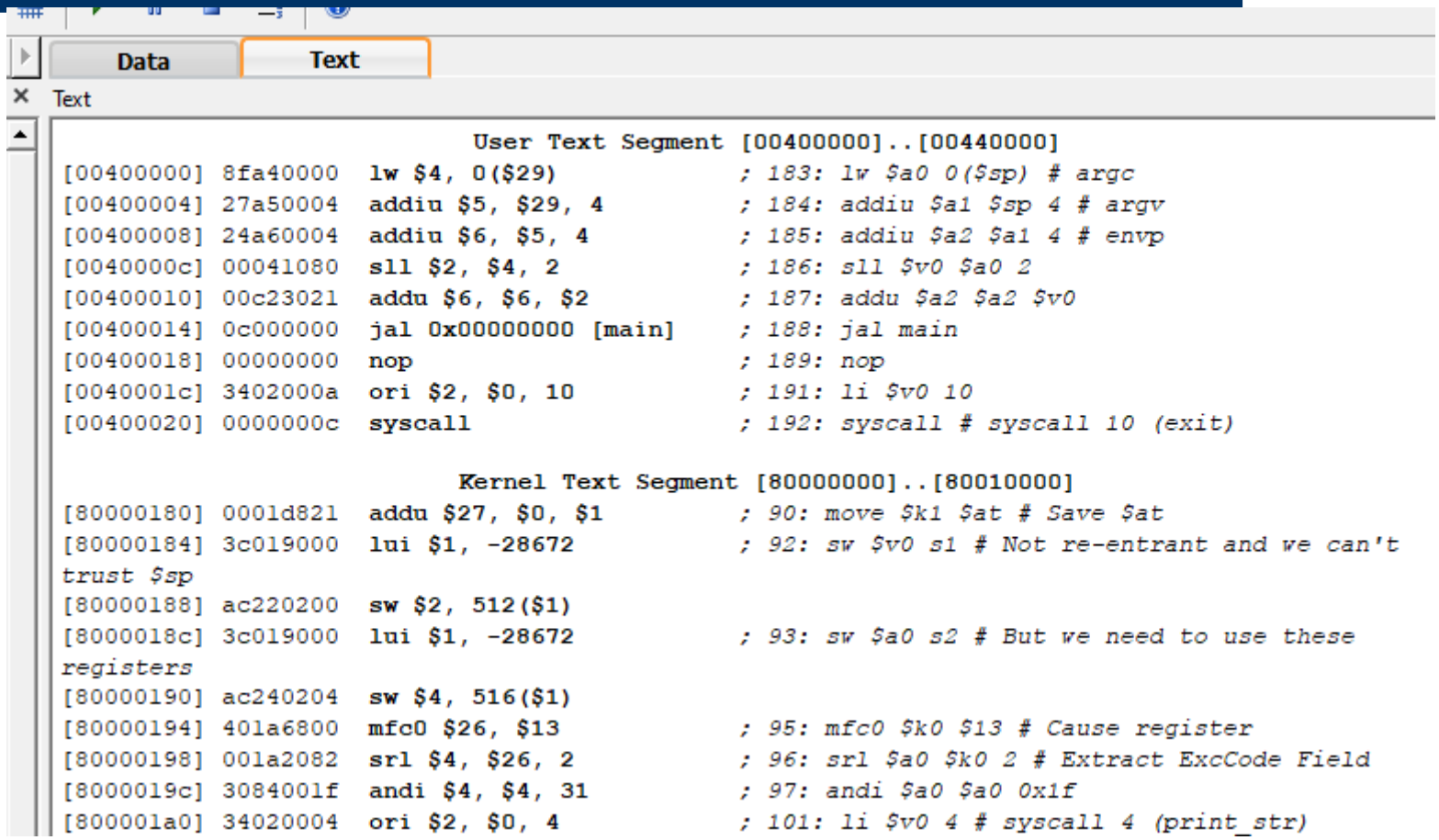
```
        .text
        .globl main

main:
        li $t2, 25              # Load immediate value (25)
        lw $t3, value           # Load the word stored in val
        add $t4, $t2, $t3       # Add
        sub $t5, $t2, $t3       # Subtract
        sw $t5, Z               #Store the answer in Z (decla

        li $v0, 10 # Sets $v0 to "10" to select exit syscall
        syscall # Exit


        .data
value:  .word 12
Z:      .word 0
```

User text segment



QtSPIM Program Example

- A Simple Program

```
#sample example 'add two numbers'

.text                          # text section
.globl main                    # call main by SPIM

main:   la $t0, value          # load address 'value' into $t0
        lw $t1, 0($t0)         # load word 0(value) into $t1
        lw $t2, 4($t0)         # load word 4(value) into $t2
        add $t3, $t1, $t2      # add two numbers into $t3
        sw $t3, 8($t0)         # store word $t3 into 8($t0)

.data                          # data section
value:  .word 10, 20, 0          data for addition
                               #
```

QtSPIM Example Program

```
## Program adds 10 and 11

    .text                       # text section
    .globl    main              # call main by SPIM

main:
    ori       $8,$0,0xA         # load "10" into register 8
    ori       $9,$0,0xB         # load "11" into register 9
    add       $10,$8,$9         # add registers 8 and 9, put result
                                # in register 10
```

QtSPIM Example Program: swap2memoryWords.asm

```
## Program to swap two memory words

    .data               # load data
    .word 7
    .word 3

    .text
    .globl main

   main:
   lui $s0, 0x1001 # load data area start address 0x10010000

   lw  $s1, 0($s0)
   lw  $s2, 4($s0)
   sw  $s2, 0($s0)
   sw  $s1, 4($s0)
```