Università di Firenze
Laurea Triennale in Informatica
Corso di Architettura degli Elaboratori
A.A. 2018/2019

Programmare in ambiente QtSpim

Paolo Lollini
Dipartimento di Matematica e Informatica

lollini@unifi.it

Useful links

- **Programmed Introduction to MIPS Assembly Language**
  - Central Connecticut State University - QtSpim Edition, August 2015
  - https://chortle.ccsu.edu/AssemblyTutorial/index.html
- **MIPS Assembly Language using QtSpim**
  - Ed Jorgensen - Version 1.0 - January 2013
  - http://www.egr.unlv.edu/~ed/MIPStextSMv11.pdf

Il simulatore SPIM (QtSpim)

- **Spim is a self-contained simulator that runs MIPS32 programs. Spim implements almost the entire MIPS32 assembler-extended instruction set.**
- **The newest version of Spim is called QtSpim, and it runs on Microsoft Windows, Mac OS X, and Linux—the same source code and the same user interface on all three platforms!**
  - accetta un programma scritto in linguaggio assembly, completo di direttive e di pseudoistruzioni
  - dispongono di un debugger che permette di inserire breakpoint e di eseguire a singoli step

- **Fedele all'architettura originaria, anche se:**
  - usa quale ordinamento dei byte (little-endian o big-endian) quello proprio dell'hardware su cui gira (su un Intel 80x86 è little-endian, mentre su Mac è big-endian)
  - non rispetta i tempi di esecuzione, inclusi i ritardi delle istruzioni di moltiplicazione e divisione, di quelle floating-point e ogni latenza di memoria

Sintassi dell'assembly di QtSpim

- **Il programma è organizzato in linee**
- **Ogni linea non bianca può essere una (pseudo)istruzione, una direttiva o un commento**
- **Spazi, tab e virgole (,) fungono da separatori**
- **Gli identificatori sono sequenze di caratteri alfanumerici (lettere, cifre, $), underbar (_) e dot (.) che non cominciano con una cifra**
- **Ogni linea può contenere un commento che parte dal carattere # e si estende fino alla fine della linea**

Sintassi dell'assembly di QtSpim

- **Lettere minuscole e maiuscole non sono equivalenti**
- **Le etichette (label) si definiscono scrivendole all'inizio della linea, seguite da due punti (:)**
- **I mnemonici di istruzioni, pseudoistruzioni, direttive e registri (`$0-$31, $zero, $at`, ...) sono riservati**
- **I numeri si indicano in decimale (default) o, se preceduti da `0x` in esadecimale**
- **Le stringhe si racchiudono tra doppi apici ("..."), e:**
  - a capo (newline) \n
  - tabulazione (tab) \t
  - virgolette (quote) \"

Struttura di un programma assembly

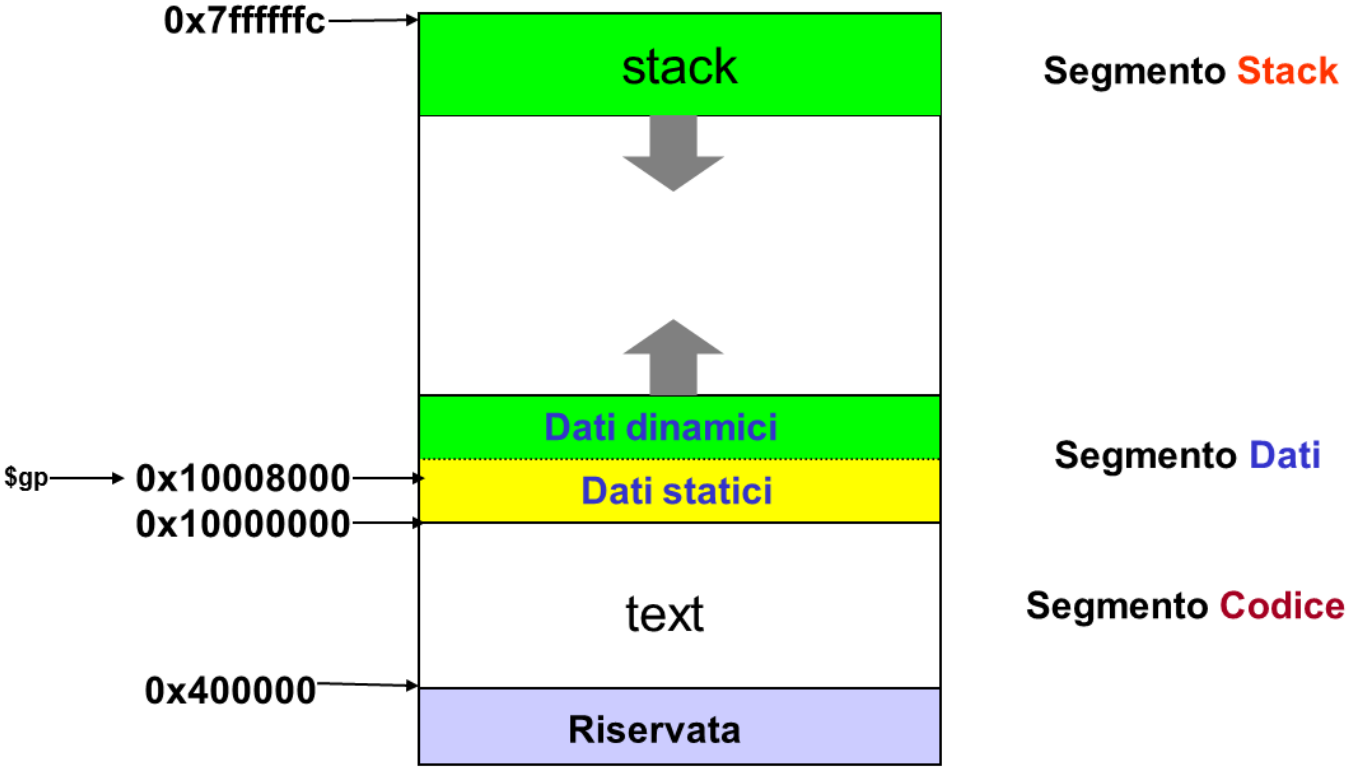
```
# Title:                                    Filename:
# Author:                                   Date:
# Description:
# Input:
# Output:
################## Data segment ########################
.data
 . . .
################## Code segment ########################
.text
.globl main
main:                                       # main program entry
 . . .
li $v0, 10                                  # Exit program
syscall
```

Direttive

- **Le principali direttive all'assemblatore sono:**
  - `.data, .kdata`
  - `.text, .ktext`
  - `.space`
  - `.ascii, .asciiz, .byte, .double, .float, .half, .word`
  - `.align`

.data

- **`.data`: introduce dati da caricare nel segmento dati**
  - I dati sono caricati nel segmento dati a partire dall'indirizzo 0x10000000 (= 2^{28}) in modo da poter essere agevolmente indirizzati tramite $gp



.text

- `.text` **introduce istruzioni e word da caricare nel segmento testo (da** `0x400000`**)**

- **Es:**

```
.text
  .globl main:
    move $t1,$zero
    ...
```

.space

- `.space n` **alloca spazio per n byte nel segmento corrente**
- **usato nel segmento dati per dichiarare array ed altre strutture dati.**

  Es.: `buffer: .space 100`

.ascii, .asciiz

- **`.ascii` *`stringa`*** **Carica *stringa* in memoria, senza terminarla con *nul***
- **`.asciiz` *`stringa`*** **Carica stringa in memoria, terminandola con *nul***
- **Es:**

```
.asciiz "Immettere dati:\n"
```

.byte, .half, .word, .float, .double

- **Consentono di caricare più numeri nei formati byte, halfword, word, float e double in posizioni consecutive di memoria**
  - `.byte b1, ..., bn`
  - `.half h1, ..., hn`
  - `.word w1, ..., wn`
  - `.float f1, ..., fn`
  - `.double d1, ..., dn`
- **Gli interi sono esprimibili in decimale o esadecimale**
- **I float e double vanno espressi in decimale, con "." obbligatorio dopo parte intera (esponente "e" opzionale)**
  - **Es:** `pi: .float 3.14`

Etichetta `__start`

- **E' l'etichetta che, per default, specifica la prima istruzione eseguibile del programma**
- **E' di solito definita nell'*exception handler* (codice di start-up, caricato all'inizio del zona text della memoria), dove è associata al codice**

```
__start:
. . .
jal main
li $v0 10     # syscall 10 (exit)
syscall
```

- **In questo caso è sufficiente che il programma contenga la procedura `main`.**

Chiamate di sistema (system calls)

- **Chiamate a procedure di I/O del sistema operativo**
- **Modalità d'uso:**
  - mettere il/i parametro/i nei registri (`$a0, $a1`)
  - specificare il tipo di system call, scrivendo un codice opportuno nel registro `$v0`
  - `syscall`
- **Esempio: stampa dell'intero con segno in `$t2`**

```
move $a0, $t2      # $a0=$t2
li $v0, 1          # codice 1 in $v0
syscall            # chiamata di sist.
```

System calls

Funzione di Servizio	Codice di chiamata di sistema	Argomenti	Risultato
print_int	1	$a0 = intero	
print_float	2	$f12 = virgola mobile, singola pr.	
print_double	3	$f12 = virgola mobile, doppia pr.	
print_string	4	$a0 = stringa	
read_int	5		Intero (in $v0)
read_float	6		Virgola mobile, singola pr. (in $v0)
read_double	7		Virgola mobile, doppia pr. (in $v0)
read_string	8	$a0 = buffer, $a1 = lunghezza	
sbrk	9	$a0 = quantità	Indirizzo (in $v0)
exit	10		
print_char	11	$a0 = carattere	
read_char	12		Carattere (in $v0)
open	13	$a0 = nome file (stringa), $a1 = flags, $a2 = modalità	Descrittore del file (in $a0)
read	14	$a0 = descrittore del file, $a1 = buffer, $a2 = lunghezza	Num. caratteri letti (in $a0)
write	15	$a0 = descrittore file, $a1 = buffer, $a2 = lunghezza	Num. caratteri scritti (in $a0)
close	16	$a0 = descrittore del file	
exit2	17	$a0 = risultato	

Esempio

```
            # This is the data segment
            .data
string0:    .asciiz "This is a text string\n"
item0:      .word 99
array0:     .word 11 22 33 44 55 66 77 88 99
```

Esempio I

```
# EXAMPLE I: Print a string on the console.
# The method is to load the address of the string into
# $a0 and then use a syscall to print the string.

        la $a0, string0     # Load the base address
                            # of string0 into $a0
        li $v0, 4           # Set $v0 to 4, this
                            # tells syscall to
                            # print the text string
                            # specified by $a0
        syscall             # Now print the text
                            # string to the console
```

Esempio II

```
# EXAMPLE II: Print an integer on the console.
# The method is to load an integer from the data
# segment and print it.

        lw $a0, item0          # Load the value of item0
                               # into $a0
        li $v0, 1              # Set $v0 to 1, this tells
                               # syscall to print the
                               # integer specified by $a0
        syscall                # Now print the integer
```

Esempio III

```
# EXAMPLE III: Read an integer from the console.

        li $v0, 5   # Set $v0 to 5, this tells
                    # syscall to read an
                    # integer from the console
        syscall     # Now read the integer.
                    # The integer is now in $v0.
```

Esempio IV

```
# EXAMPLE IV: Print an element from an integer array
#             in the data segment.
        li $t0, 3       # Set t0 to 3, the index of
                        # the element we are fetching
        li $t1, 4       # Set t1 to 4, this is the
                        # size in bytes of an element
        mul $t2, $t1, $t0   # t2 = t1 * t0, so t2 is the
                            # BYTE offset of the element
        lw $a0, array0($t2) # Load the element of the
                            # array; note that the first
                            # element in the array has
                            # an offset of zero
        li $v0, 1           # Set $v0 to 1, this tells
                            # syscall to print the
                            # integer specified by $a0
        syscall             # Now print the integer
```

Pseudoistruzioni

Esempio di pseudoistruzione con pseudoindirizzamento

Esempio V

```
# EXAMPLE V: Halt the program.
   li $v0, 10          # set $v0 to 10, this tells
                       # syscall to end execution
   syscall
```

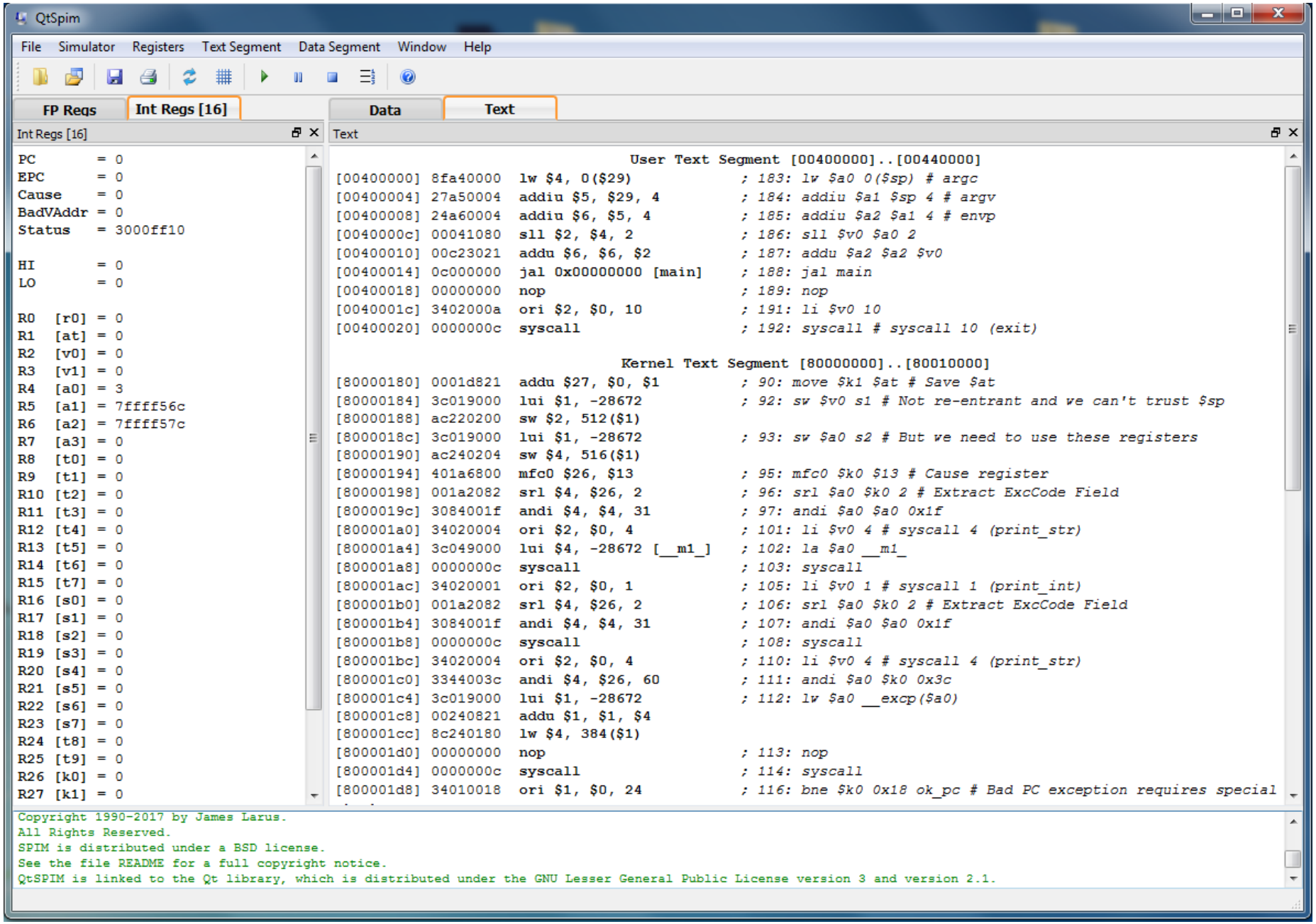
Caratteristiche essenziali di QtSpim

Download:

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

- **clear : pulisce registri e memoria**
- **load : carica il programma**
- **run : esegue il programma**
- **step : esecuzione single-step**
- **breakpoint : set/delete breakpoints**

- **Nota:**
  - Codice di start-up è pre-caricato (*exceptions.s*).
  - Pseudo-istruzioni sono espanse in istruzioni reali.


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] = 3
R5 [a1] = 7ffff56c
R6 [a2] = 7ffff57c
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] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 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 # 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
[800001c4] 3c019000 lui $1, -28672 ; 112: lw $a0 __excp($a0)
[800001c8] 00240821 addu $1, $1, $4
[800001cc] 8c240180 lw $4, 384($1)
[800001d0] 00000000 nop ; 113: nop
[800001d4] 0000000c syscall ; 114: syscall
[800001d8] 34010018 ori $1, $0, 24 ; 116: bne $k0 0x18 ok_pc # Bad PC exception requires special
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.