Calcolatori Elettronici
Esercitazioni Assembler

M. Sonza Reorda – M. Monetti
M. Rebaudengo – R. Ferrero
L. Sterpone – M. Grosso
renato.ferrero@polito.it

Politecnico di Torino
Dipartimento di Automatica e Informatica

Informazioni generali

- Esercitazioni assistite: 2 squadre
  - Mercoledì h. 11.30-13.00 (cognomi fino ad H)
  - Venerdì h. 10.00-11.30 (cognomi da I in poi)
- 1° laboratorio:
  - 26/03: squadra 2
  - 31/03: squadra 1
- Laboratori successivi: tutte le settimane, a partire dal 14/04
  - prima squadra 1, poi squadra 2.

Organizzazione delle esercitazioni

- All’inizio della settimana saranno caricati:
  - il pdf con il testo dell’esercitazione
  - un breve video introduttivo all’esercitazione
- Durante la settimana si tengono le virtual classroom
  - anche se lo studente riceve gli inviti per entrambe le virtual classrom, ne deve seguire solo una (la suddivisione è in base al proprio cognome, vedi slide precedente).
- Al termine della settimana sarà caricato:
  - il pdf con la soluzione dell’esercitazione

Interazione con docente e borsisti

- Domande e risposte in tempo reale durante le esercitazioni (virtual classroom)
- Domande "semplici" sul gruppo Telegram del corso
- Domande e richieste di consulenza su Slack
  - workspace: "Calcolatori Elettronici – corso 2"

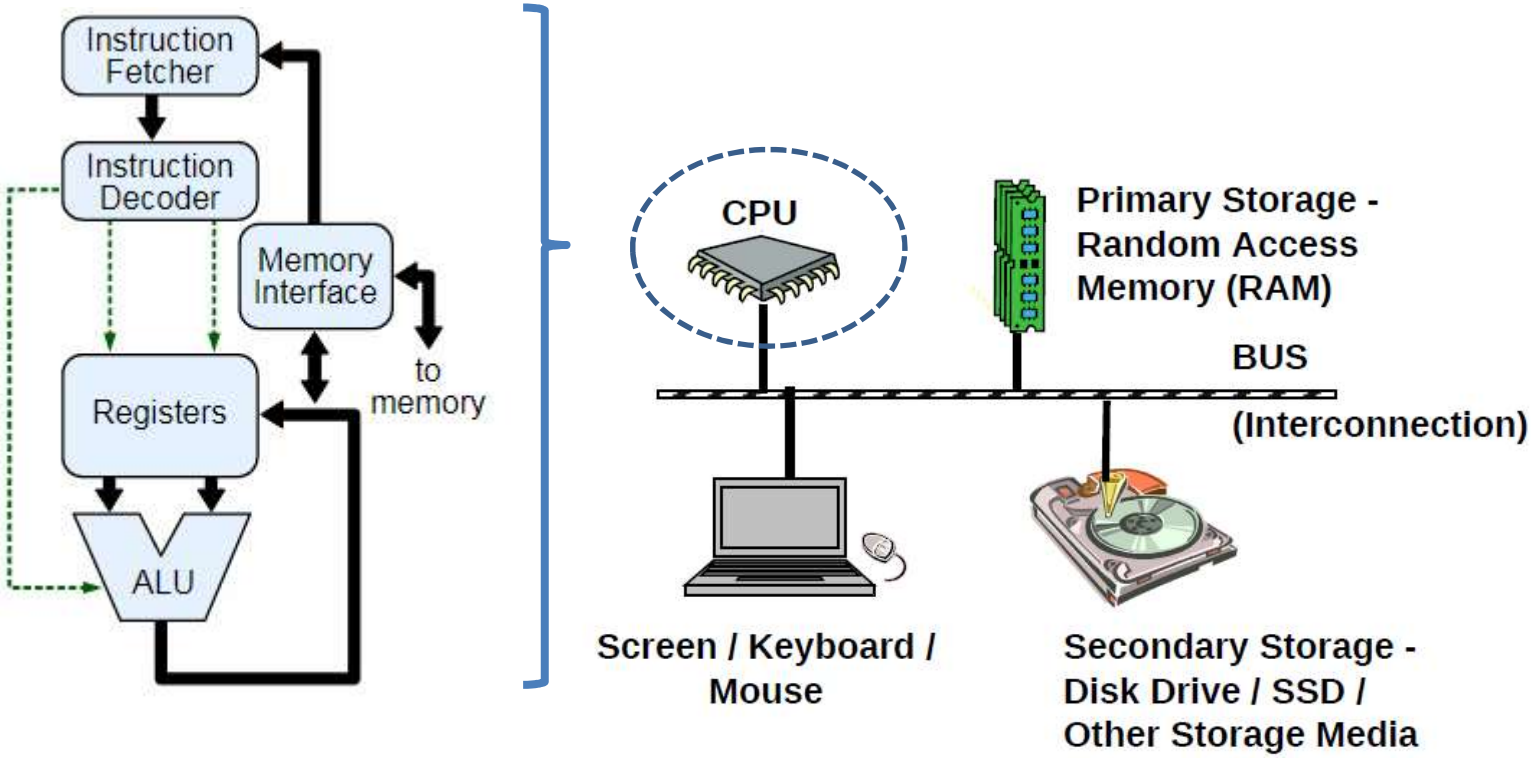
https://join.slack.com/t/calcolatoriel-l2h9254/shared_invite/zt-n8i1yo92-sB_ymmZ6OmOGyXLNYiuIPw

Canali Slack

- Un canale per ogni esercitazione: *laboratorioX*
  - domande su testo e soluzione degli esercizi
  - è possibile allegare un file di testo con la propria soluzione e ricevere spiegazioni
- Un canale per consulenze: *consulenze*
  - ogni settimana è indicato l'orario della consulenza, assieme alle modalità (ad esempio link Zoom)
  - per prenotarsi occorre accettare l'invito
  - se non si è interessati, non fare nulla

Il calcolatore

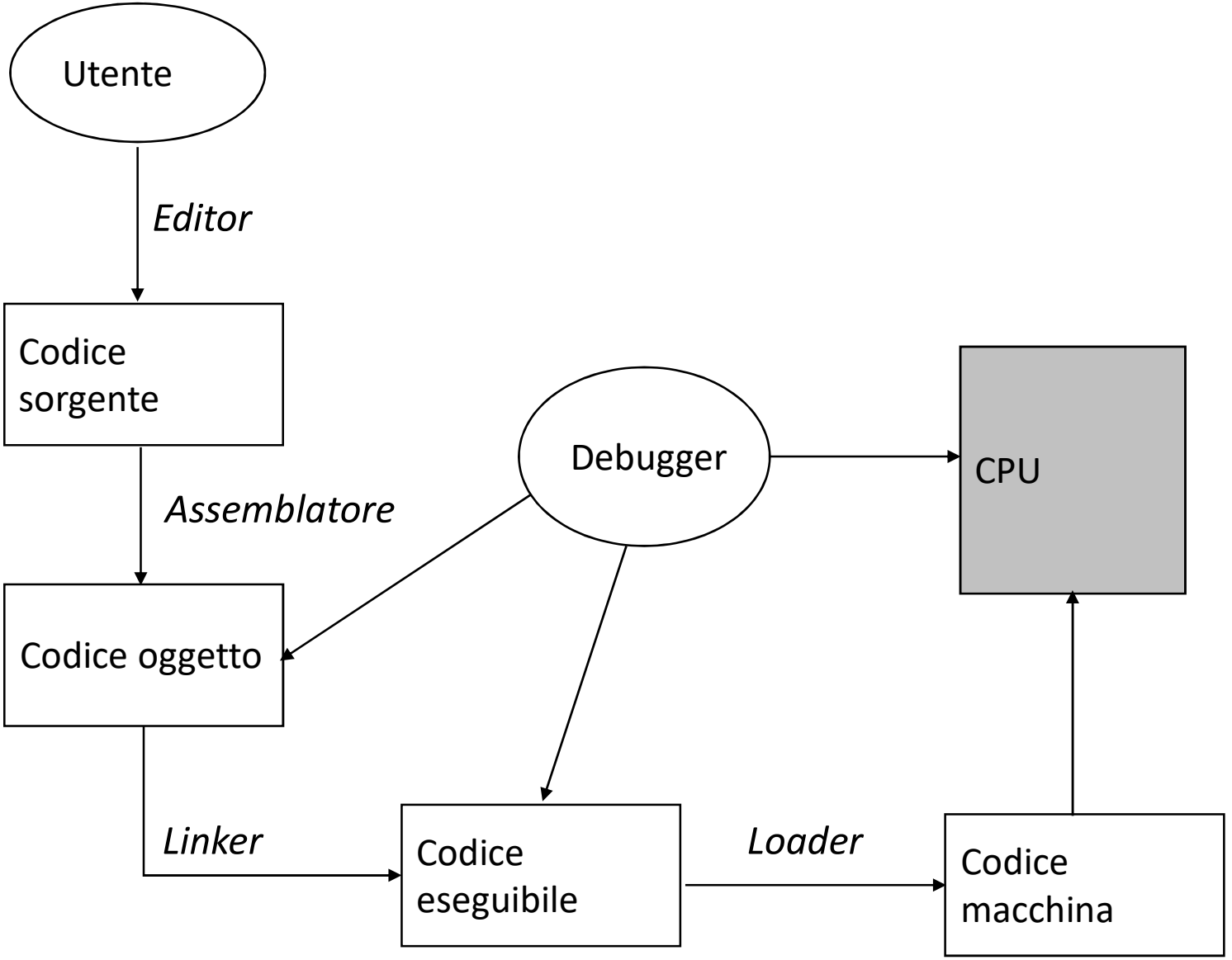
- Schema dal punto di vista del programmatore in linguaggio Assembly



Architettura MIPS32 - Registri

Nome	Numero	Uso
`$zero`	$0	il valore costante 0
`$at`	$1	*riservato per assemblatore*
`$v0-$v1`	$2-$3	valori di ritorno della funzione
`$a0-$a3`	$4-$7	argomenti della funzione
`$t0-$t7`	$8-$15	valori temporanei
`$s0-$s7`	$16-$23	variabili salvate
`$t8-$t9`	$24-$25	altri valori temporanei
`$k0-$k1`	$26-$27	*riservato per sistema operativo*
`$gp`	$28	*riservato per puntatore al segmento di variabili globali*
`$sp`	$29	stack pointer
`$fp`	$30	frame pointer
`$ra`	$31	indirizzo di ritorno della funzione

Ciclo di vita di un programma



Editor

- Qualunque editor di testo va bene
  - Un editor di documenti (come Microsoft Word) non va bene perché aggiunge dati sulla formattazione non comprensibili da QtSpim.
- Suggerimenti:
  - per Windows: Notepad++ https://notepad-plus-plus.org/downloads/
  - per ogni sistema operativo: Atom https://atom.io/

Colorazione della sintassi

- Un editor di testo può visualizzare un testo con diversi colori in base a regole sintattiche.
- Il vantaggio è il miglioramento della leggibilità del codice sorgente.
- Notepad++ e Atom possono essere configurati per supportare l'assembly MIPS.
  - Questo è un passaggio opzionale, esclusivamente per la colorazione della sintassi secondo le regole del MIPS.

Procedura per Notepad++

- Notepad++ usa un file xml per descrivere le regole sintattiche del linguaggio: https://github.com/notepad-plus-plus/userDefinedLanguages/blob/master/udl-list.md
- Per MIPS, scaricare *ASM for MIPS R2000*
- Copiare il file xml nella directory: Language -> User Defined Language -> Open User Defined Language folder; poi riavviare Notepad ++.

Linguaggio predefinito

- La colorazione della sintassi definita in *ASM for MIPS R2000* è applicata in automatico solo se l'estensione del file è .s
- Si può applicare la colorazione della sintassi anche a file con altre estensioni (ad esempio .asm o .a):
  - manualmente: nel menu Language selezionare il linguaggio Assembly (MIPS)
  - in automatico: nel file .xml di *ASM for MIPS R2000* cambiare

    <UserLang name="Assembly (MIPS)" ext="s">

    aggiungendo le estensioni desiderate, ad esempio:

    <UserLang name="Assembly (MIPS)" ext="s asm a">

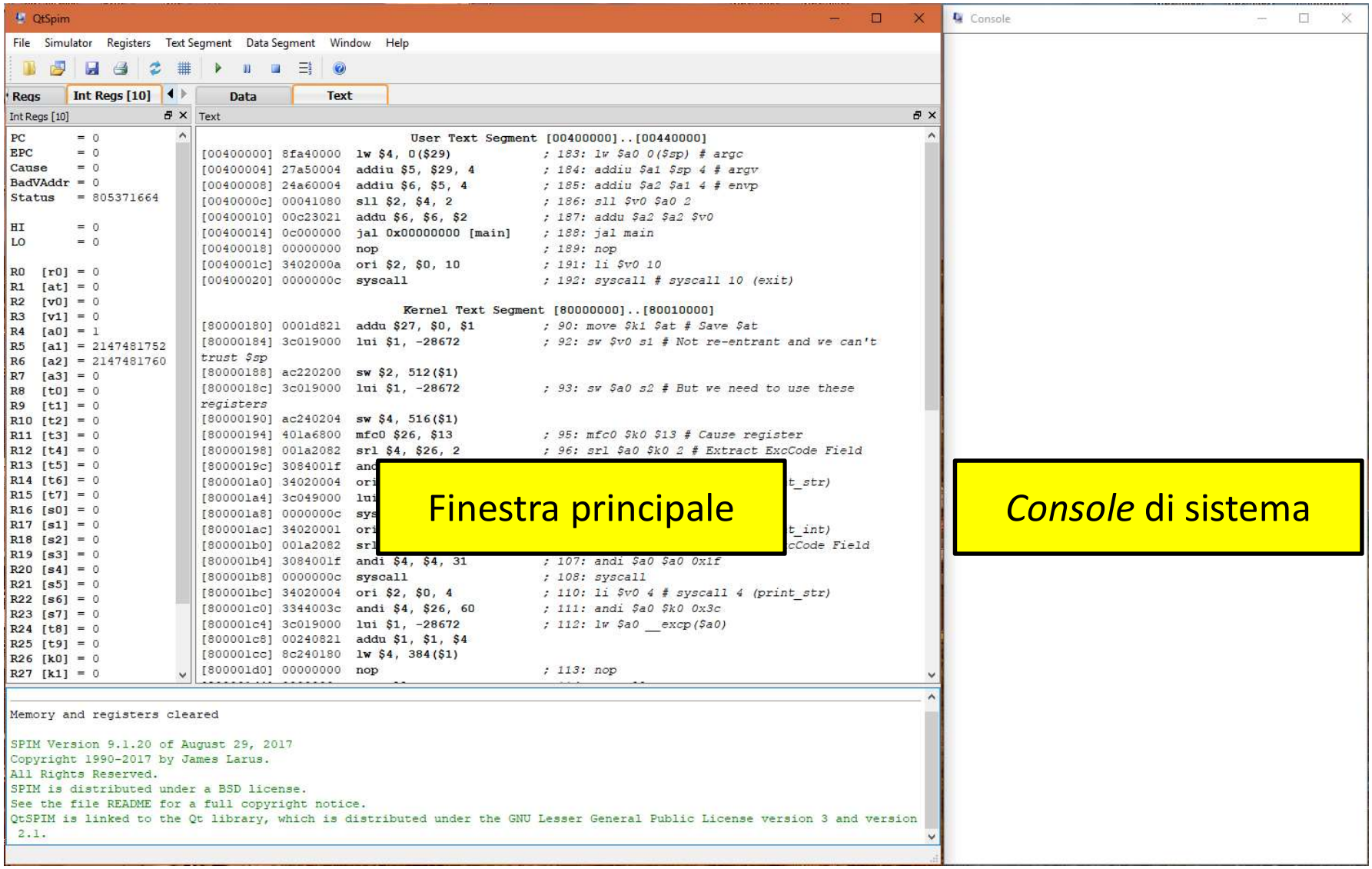
Procedura per Atom

- Installare il pacchetto "language-mips" scaricabile da https://atom.io/packages/language-mips
- La colorazione della sintassi è applicata se l’estensione del file è .s o .asm

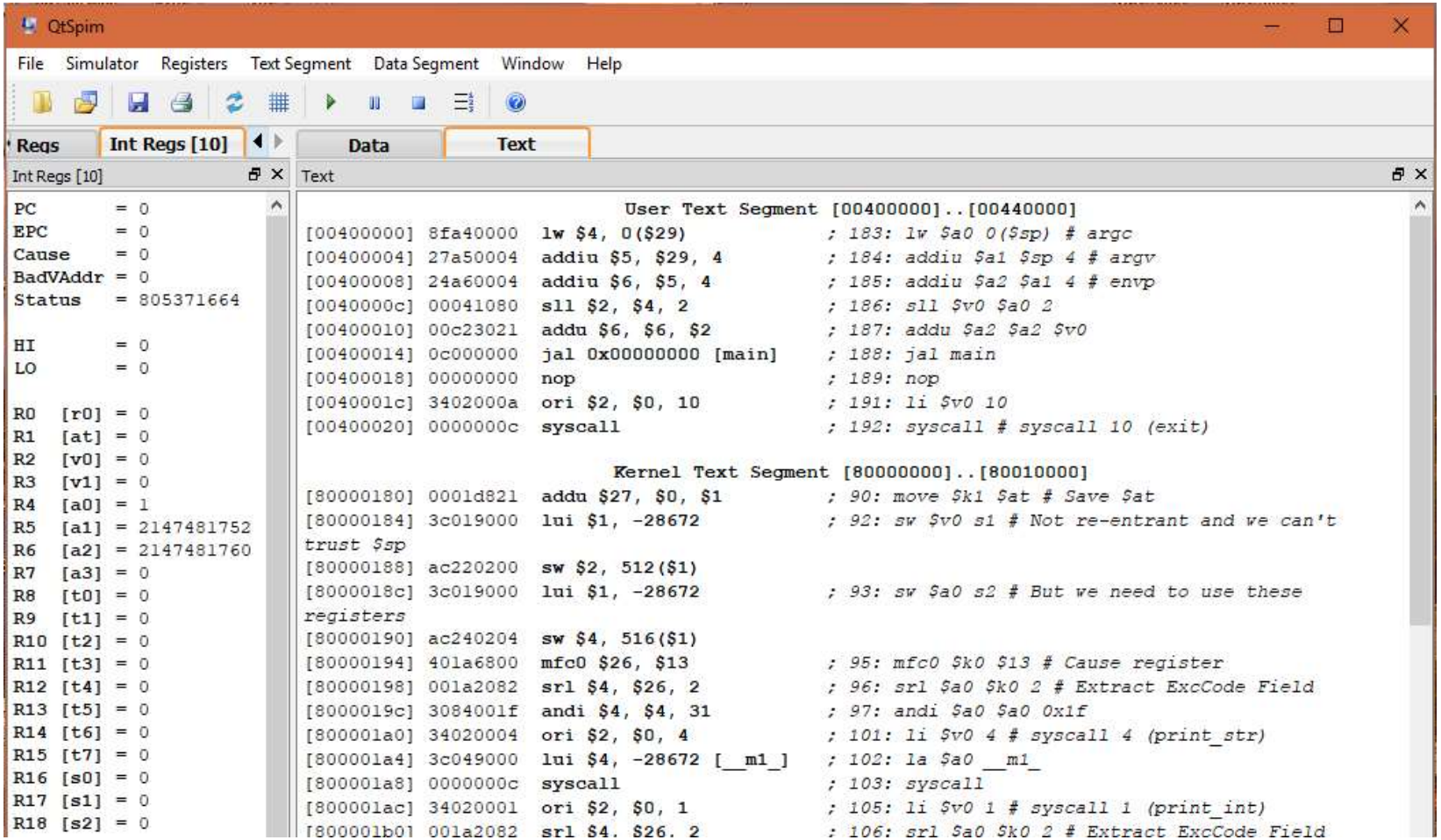
QtSpim

- Simulatore di programmi per MIPS32
  - Legge ed esegue programmi scritti nel linguaggio assembly di questo processore
  - Include un semplice *debugger* e un insieme minimo di servizi del sistema operativo
  - Non è possibile eseguire programmi compilati (binario)
- È compatibile con (quasi) l'intero *instruction set* MIPS32 (Istruzioni e PseudoIstruzioni)
  - Non include confronti e arrotondamenti *floating point*
- È gratuito e open-source, e sono disponibili versioni per MS-Windows, Mac OS X e Linux http://spimsimulator.sourceforge.net/
- Informazioni utili: http://spimsimulator.sourceforge.net/further.html

Interfaccia di QtSpim



Finestra principale

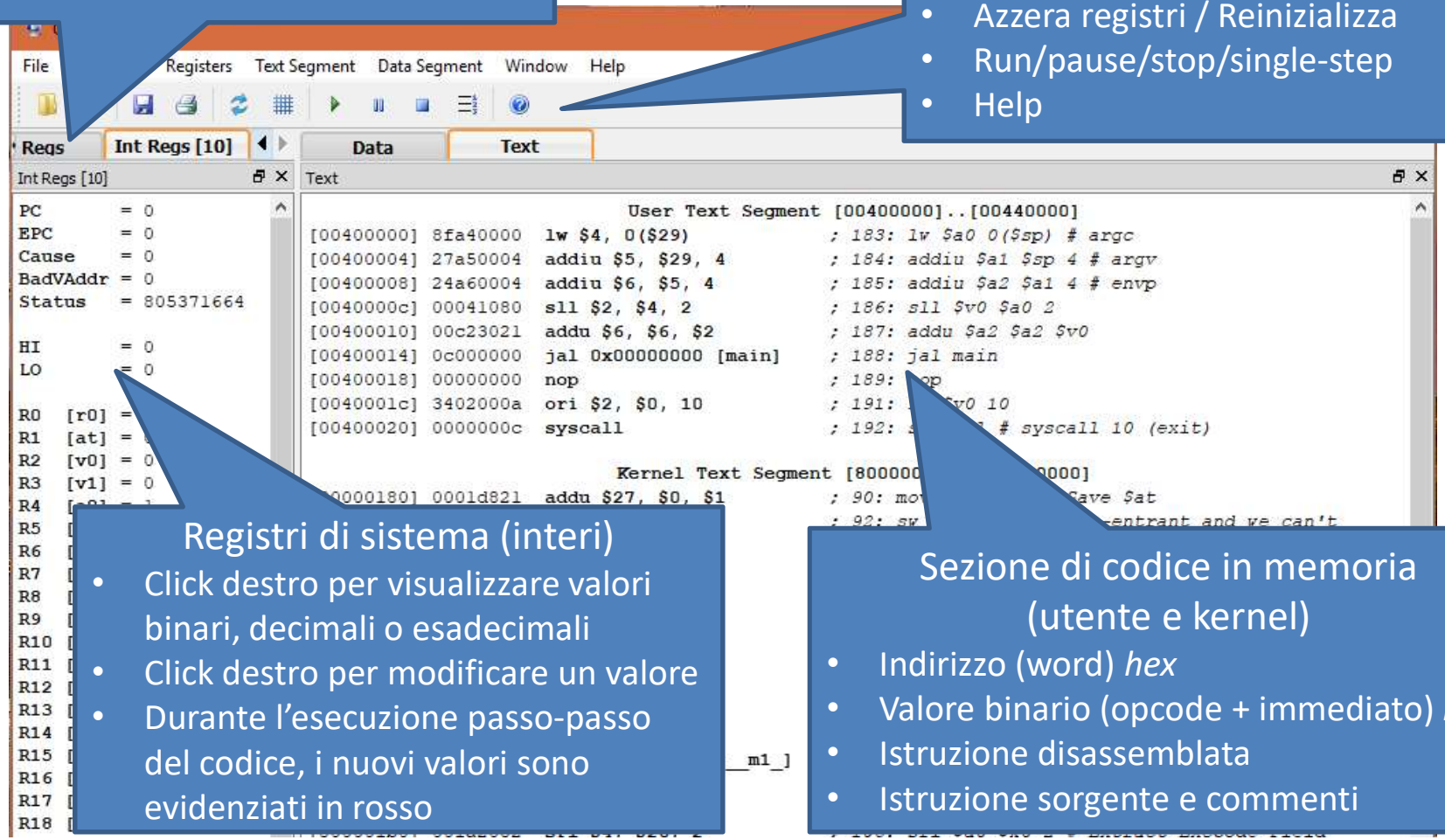


tra princ

Visualizzazione dei registri dell'unità floating point

Barra dei pulsanti

- Carica / Reinizializza e carica
- Salva log / Stampa
- Azzera registri / Reinizializza
- Run/pause/stop/single-step
- Help



Registri di sistema (interi)

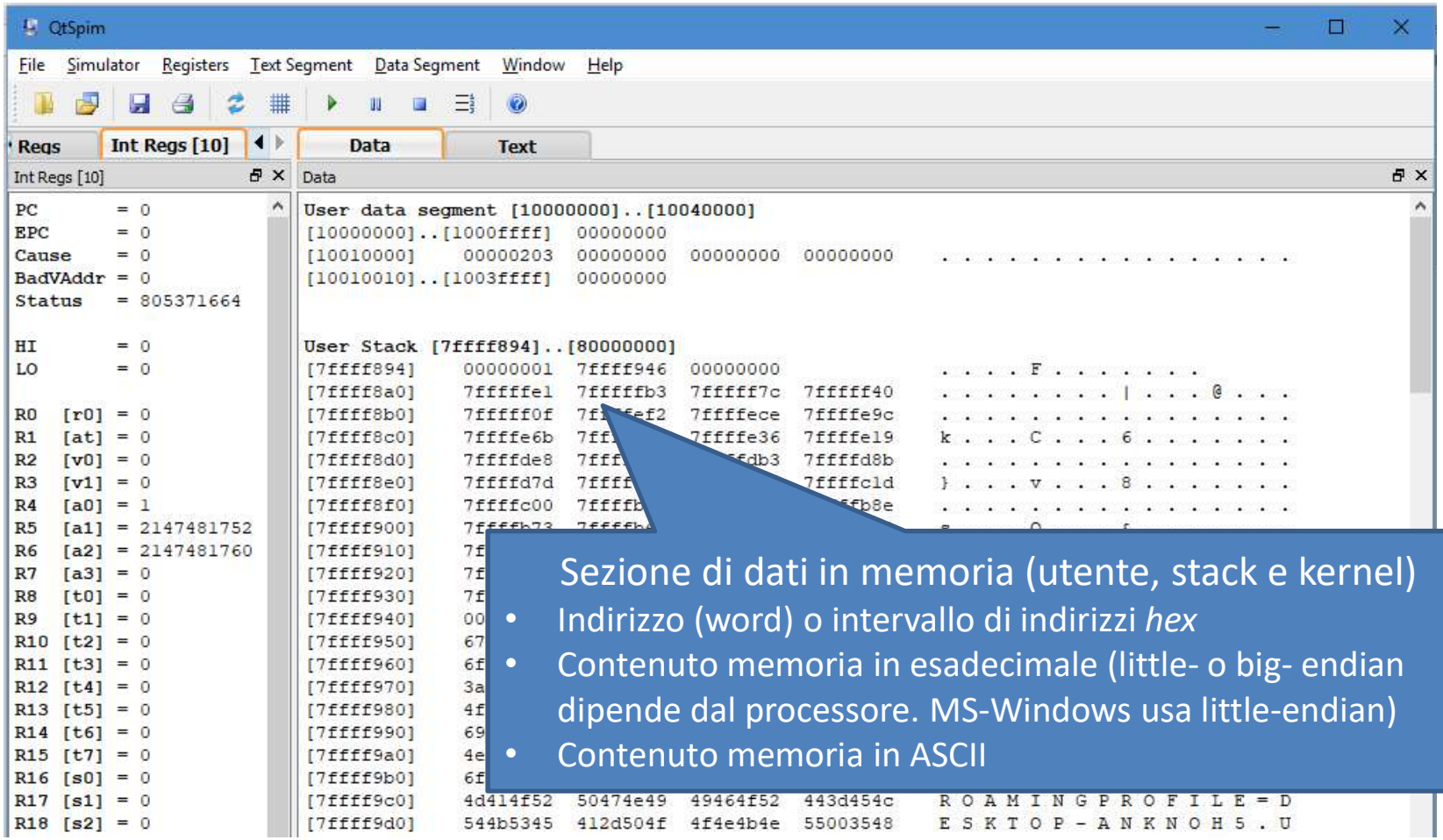
- Click destro per visualizzare valori binari, decimali o esadecimali
- Click destro per modificare un valore
- Durante l'esecuzione passo-passo del codice, i nuovi valori sono evidenziati in rosso

Sezione di codice in memoria (utente e kernel)

- Indirizzo (word) *hex*
- Valore binario (opcode + immediato) *hex*
- Istruzione disassemblata
- Istruzione sorgente e commenti

Finestra principale

Finestra principale



Codice di esempio

- Il codice può essere introdotto con un qualsiasi editor di testo, e salvato in un file con estensione `.a`, `.s` o `.asm`

```
                .data           # dichiarazione dati
op1:            .byte 3
op2:            .byte 2
res:            .space 1        # allocazione spazio in memoria per risultato

                .text
                .globl main
                .ent main
main:           lz $t1, op1         # caricamento dati
                lb $t2, op2
                add $t1, $t1, $t2 # esecuzione somma
                sb $t1, res         # salvataggio del risultato in memoria
                li $v0, 10          # codice per uscita dal programma
                syscall             # fine
                .end main
```

Codice di esempio

- Il codice può essere introdotto con un qualsiasi editor di testo, e salvato in un file con estensione `.asm`

```
                .data
op1:            .byte 3
op2:            .byte 2
res:            .space 1        # allo                    risultato

                .text
                .globl main
                .ent main
main:           lz $t1, op1
                lb $t2, op2
                add $t1, $t1, $t2 #
                sb $t1, res       #                         emoria
                li $v0,10         #                         a
                syscall           #
                .end main
```

data segment

- Di seguito sono riportate le dichiarazioni delle variabili

text segment

- Di seguito sono riportate le istruzioni

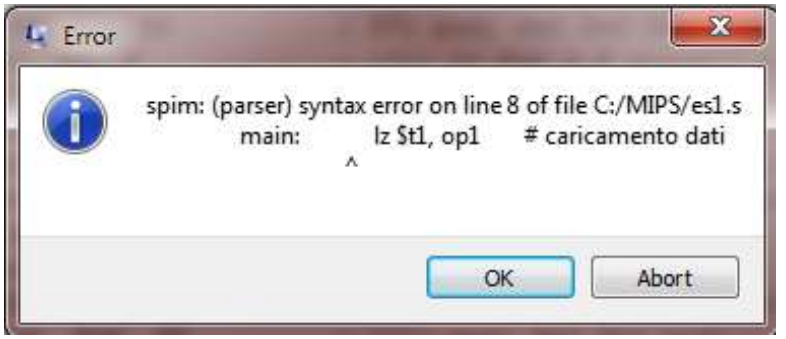
main segment

- Punto di partenza del programma. Deve essere dichiarato come `.globl`

- Fine del programma

Caricamento del codice

- In QtSpim, dal menu *File* selezionare "*Reinitialize and Load File*", quindi selezionare il codice salvato precedentemente
  - In alternativa, premere il pulsante 
  - Eventuali errori di sintassi sono segnalati e richiedono la correzione del codice



- Quando il codice è correttamente caricato, è possibile agire sugli opportuni pulsanti  per eseguirlo

Debug

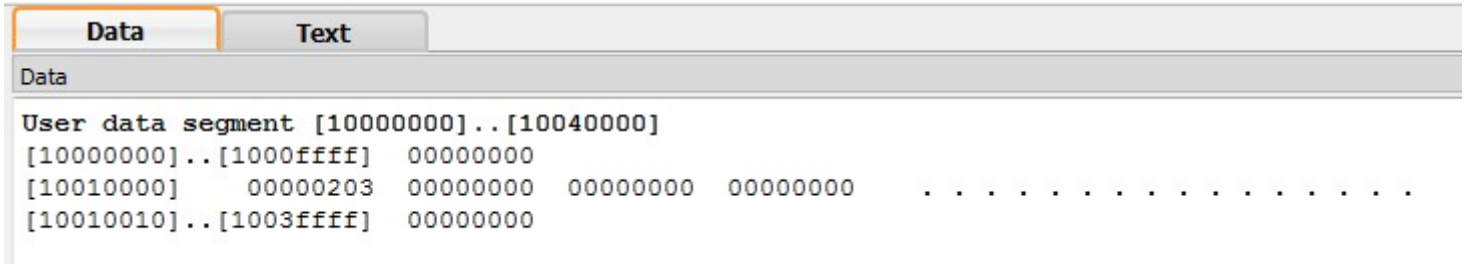
- L’esecuzione passo-passo è fondamentale per il *debug*
  - Osservare il valore di memoria e registri al termine di ogni istruzione
- È possibile inserire un *breakpoint* facendo click con il pulsante destro sull’istruzione desiderata nella sezione *Text* della finestra principale, e selezionando “*Set Breakpoint*”
- Ogni volta che il codice viene modificato, è necessario ripartire da “*Reinitialize and Load File*”

Debug [cont.]

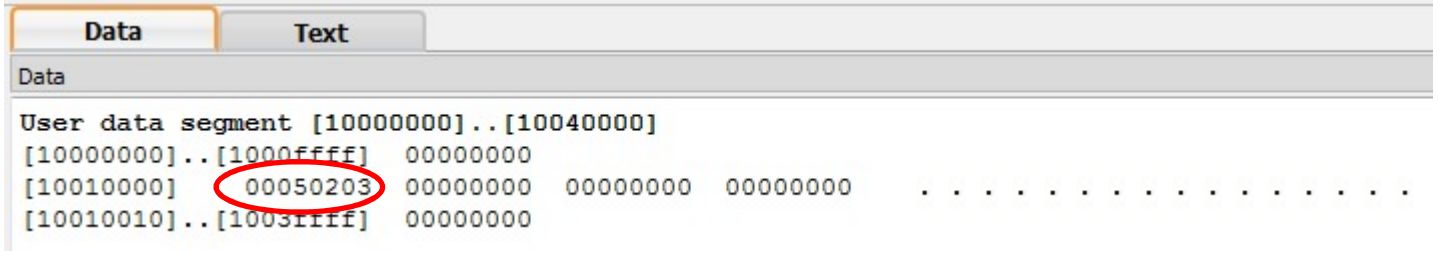
- Esempio: esecuzione dell'istruzione di memorizzazione

```
[00400034] 012a4820  add $9, $9, $10          ; 10: add $t1, $t1, $t2 # esecuzione somma
[00400038] 3c011001  lui $1, 4097             ; 11: sb $t1, res # salvataggio del risultato in memoria
[0040003c] a0290002  sb $9, 2($1)
[00400040] 3402000a  ori $2, $0, 10           ; 12: li $v0,10 # codice per uscita dal programma
[00400044] 0000000c  syscall                  ; 13: syscall # fine
```

  - Variabili prima del salvataggio:



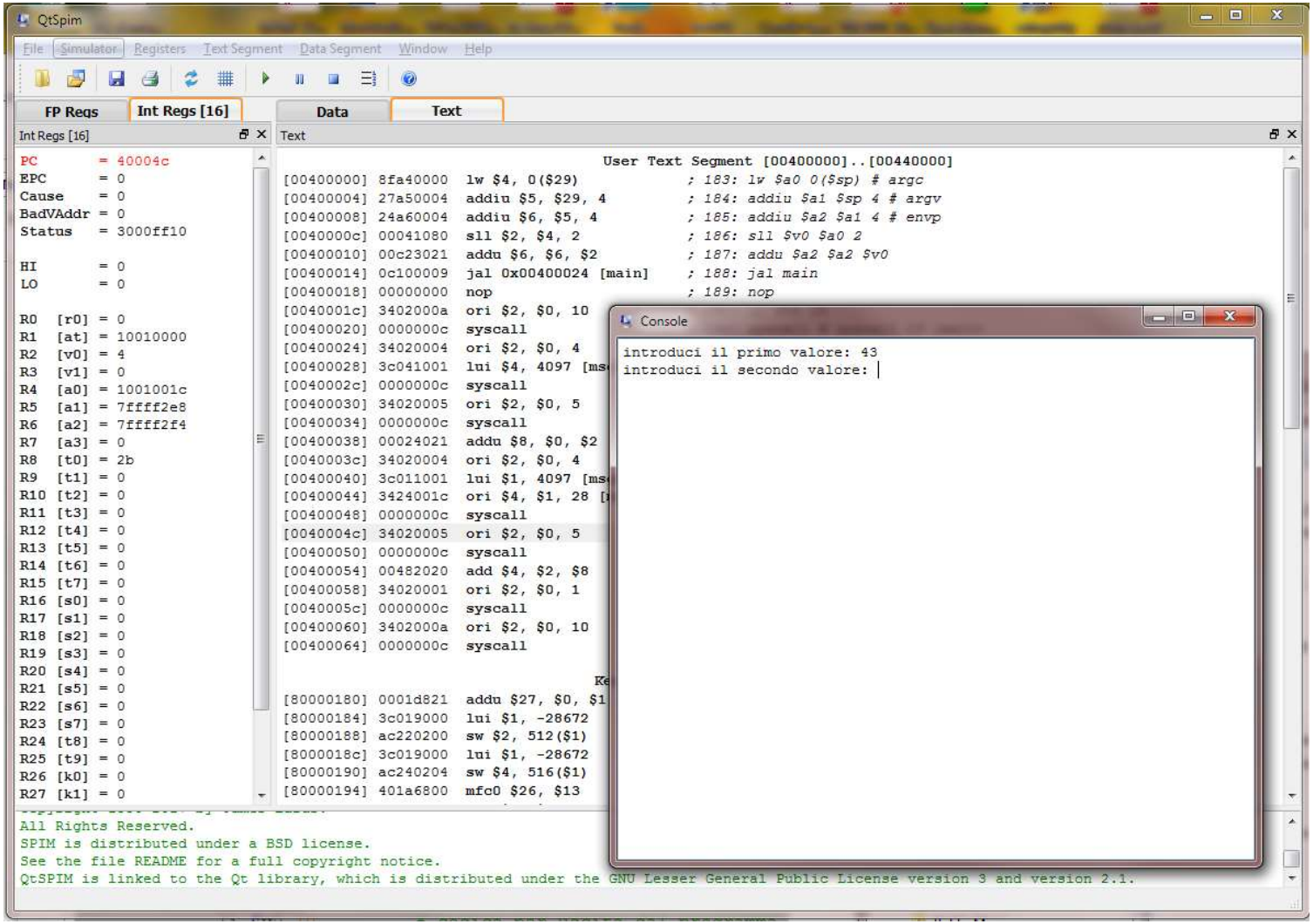
  - Variabili dopo il salvataggio:



Input/Output da *console*

```
            .data
msg1:       .asciiz "introduci il primo valore: "
msg2:       .asciiz "introduci il secondo valore: "
            .text
            .globl main
            .ent main
main:       li $v0, 4           # syscall 4 (print_str)
            la $a0, msg1        # argomento: stringa
            syscall             # stampa la stringa
            li $v0, 5           # syscall 5 (read_int)
            syscall
            move $t0, $v0       # primo operando
            li $v0, 4
            la $a0, msg2
            syscall
            li $v0, 5
            syscall
            add $a0, $v0, $t0 # somma degli operandi
            li $v0, 1           # syscall 1 (print_int)
            syscall
            li $v0,10           # codice per uscita dal programma
            syscall             # fine
            .end main
```

Input/Output da *console* [cont.]



Scrittura di un valore in una cella di memoria

```
            .data
variabile:  .space 4    # int variabile
            .text
            .globl main
            .ent main
main:

            li  $t0, 19591501       # variabile = 19591501 (012A F14D hex)
            sw  $t0, variabile

            li $v0, 10
            syscall

            .end main
```

Ricerca del carattere minimo

```
            .data
wVet:       .space  5
wRes:       .space  1
message_in:  .asciiz   "Inserire caratteri\n"
message_out: .asciiz   "\nValore Minimo : "

            .text
            .globl main
            .ent main
main:
            la $t0, wVet                    # puntatore a inizio del vettore
            li $t1, 0                       # contatore

            la $a0, message_in              # indirizzo della stringa
            li $v0, 4                       # system call per stampare stringa
            syscall
```

Ricerca del carattere minimo [cont.]

```
ciclo1: li  $v0, 12              # legge 1 char
        syscall                  # system call (risultato in $v0)
        sb  $v0, ($t0)
        add $t1, $t1, 1
        add $t0, $t0, 1
        bne $t1, 5, ciclo1       # itera 5 volte

        la  $t0, wVet
        li  $t1, 0               # contatore
        lb  $t2, ($t0)           # in $t2 memorizzo MIN iniziale

ciclo2: lb  $t3, ($t0)
        bgt $t3, $t2, salta      # salta se NON deve aggiornare MIN
        lb  $t2, ($t0)           # aggiorna MIN
salta:  add $t1, $t1, 1
        add $t0, $t0, 1
        bne $t1, 5, ciclo2
```

Ricerca del carattere minimo [cont.]

```
la $a0, message_out
li $v0, 4
syscall

li    $v0, 11         # stampa 1 char
move  $a0, $t2
syscall

li $v0, 10
syscall
.end main
```

Laboratorio:

- Si prendano gli esempi di codice presentati in precedenza e quelli nell' "Introduzione al linguaggio MIPS" (ASSEM_00.pdf).
- Si richiede di inserire tali esempi di codice in QtSpim, compilarli, eseguirli e analizzarne il comportamento in modalità di debug.