

# Calcolatori Elettronici

## Esercitazioni Assembler

L. Sterpone – M. Grosso  
M. Sonza Reorda – M. Monetti  
[michelangelo.grosso@polito.it](mailto:michelangelo.grosso@polito.it)

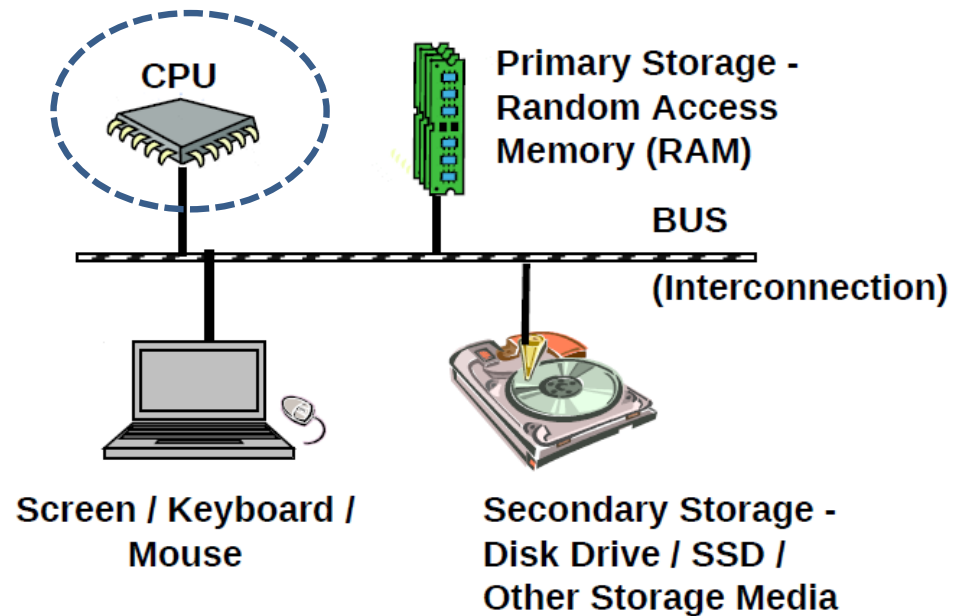
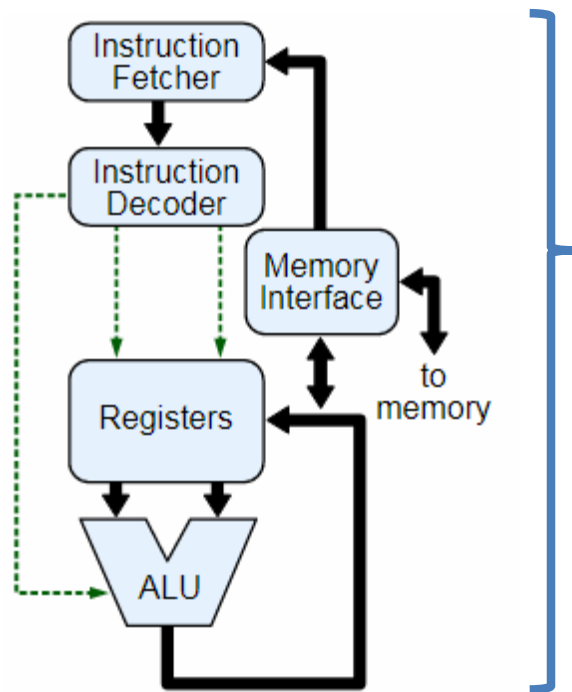
Politecnico di Torino  
Dipartimento di Automatica e Informatica

# Informazioni generali

- LABINF
  - 1° piano, ingresso da C.so Castelfidardo lato ovest
- Esercitazioni assistite: 2 squadre
  - Martedì h. 13.00-14.30 (N – Q)
  - Giovedì h. 16.00-17.30 (R – Z)
- 1° laboratorio: 26-28 marzo

# Il calcolatore

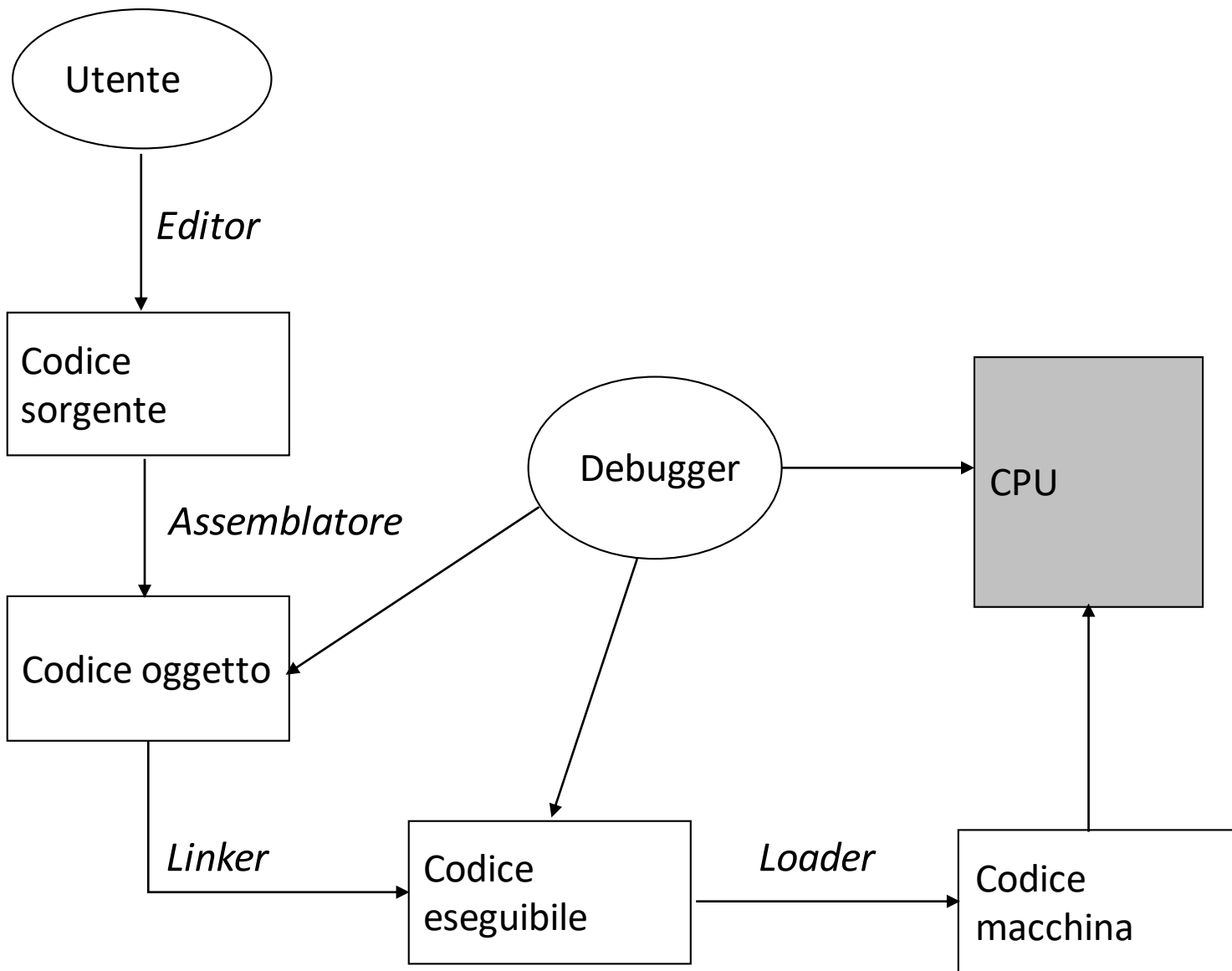
- Schema dal punto di vista del programmatore in linguaggio Assembly



# Architettura MIPS32 - Registri

Name	Register Number	Usage
<b>\$0</b>	0	the constant value 0
<b>\$at</b>	1	assembler temporary
<b>\$v0-\$v1</b>	2-3	function return values
<b>\$a0-\$a3</b>	4-7	function arguments
<b>\$t0-\$t7</b>	8-15	temporaries
<b>\$s0-\$s7</b>	16-23	saved variables
<b>\$t8-\$t9</b>	24-25	more temporaries
<b>\$k0-\$k1</b>	26-27	OS temporaries
<b>\$gp</b>	28	global pointer
<b>\$sp</b>	29	stack pointer
<b>\$fp</b>	30	frame pointer
<b>\$ra</b>	31	function return address

# Ciclo di vita di un programma





# QtSpim

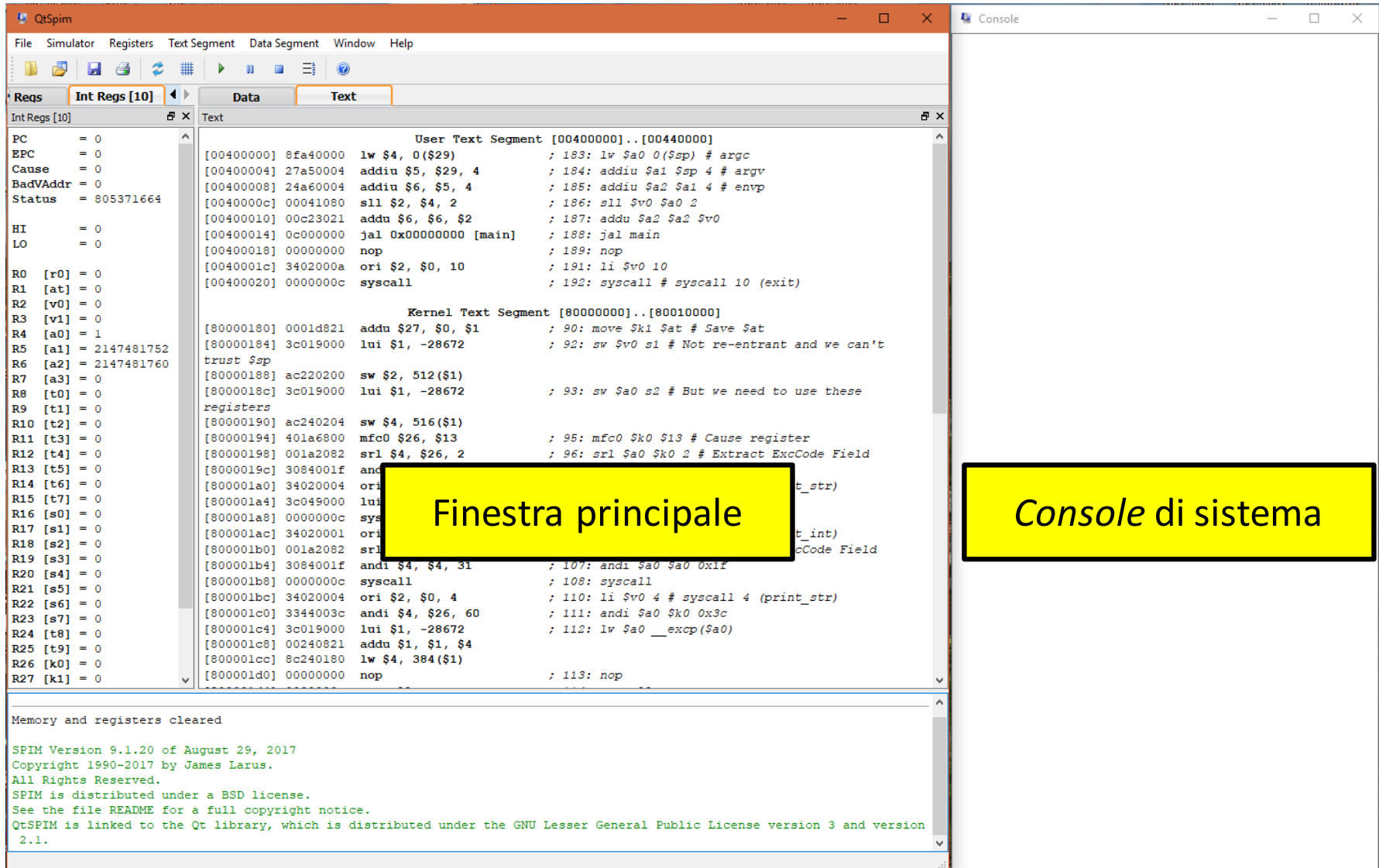
- Simulatore di programmi per MIPS32
  - Legge ed esegue programmi scritti nel linguaggio assembly di questo processore
  - Include un semplice *debugger* e un insieme minimo di servizi del sistema operativo
  - Non è possibile eseguire programmi compilati (binario)
- È compatibile con (quasi) l'intero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



# QtSpim

- QtSpim è già installato sui PC del laboratorio in ambiente MS-Windows
- Gli studenti possono inoltre installare il programma sul proprio PC, scaricandolo da <http://spimsimulator.sourceforge.net/>
- Per qualsiasi problema, è possibile
  - Rivolgersi all'esercitatore o ai borsisti in laboratorio
  - Contattare l'esercitatore via email.

# Interfaccia di QtSpim





# Finestra principale

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

Reqs Int Regs [10] Data Text

Int Regs [10]

PC = 0  
EPC = 0  
Cause = 0  
BadVAddr = 0  
Status = 805371664

HI = 0  
LO = 0

R0 [r0] = 0  
R1 [at] = 0  
R2 [v0] = 0  
R3 [v1] = 0  
R4 [a0] = 1  
R5 [a1] = 2147481752  
R6 [a2] = 2147481760  
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

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 $1 # Not re-entrant and we can't
trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672 ; 93: sw $a0 $2 # 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
```

# 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

The screenshot shows a debugger window with a menu bar (File, Registers, Text Segment, Data Segment, Window, Help) and a toolbar. The 'Registers' tab is active, displaying a list of registers (PC, EPC, Cause, BadVAddr, Status, HI, LO, R0-R18) and their values. The 'Text' tab is also visible, showing memory segments (User Text Segment, Kernel Text Segment) and their contents (hex addresses, instructions, and comments). A toolbar at the top contains icons for loading, saving, and running. A callout points to the 'Int Regs [10]' tab, and another points to the 'Text' tab.

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

```
---
R4 [a0] = 1
R5 [a1] = 2147481752
R6 [a2] = 2147481760
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

[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, 00240180
[800001d0] 00000000 nop

Memory and registers cleared

SPIM Version 9.1.20 of August 29, 2017
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.
```

Console informativa

- Messaggi di informazione e di errore

# Finestra principale

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

Regs Int Regs [10] Data Text

Int Regs [10]

PC = 0  
EPC = 0  
Cause = 0  
BadVAddr = 0  
Status = 805371664

HI = 0  
LO = 0

R0 [r0] = 0  
R1 [at] = 0  
R2 [v0] = 0  
R3 [v1] = 0  
R4 [a0] = 1  
R5 [a1] = 2147481752  
R6 [a2] = 2147481760  
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

User data segment [10000000]..[10040000]  
[10000000]..[1000ffff] 00000000  
[10010000] 00000203 00000000 00000000 00000000 . . . . .  
[10010010]..[1003ffff] 00000000

User Stack [7ffff894]..[80000000]  
[7ffff894] 00000001 7ffff946 00000000 . . . . F . . . . .  
[7ffff8a0] 7fffffe1 7fffffb3 7fffff7c 7fffff40 . . . . . | . . . @ . . .  
[7ffff8b0] 7fffff0f 7ffffef2 7ffffece 7ffffe9c . . . . . . . . . . .  
[7ffff8c0] 7ffffe6b 7fffff36 7ffffe36 7ffffe19 k . . . C . . . 6 . . . . .  
[7ffff8d0] 7ffffde8 7ffffdb3 7ffffdb3 7ffffdb3 . . . . . . . . . . .  
[7ffff8e0] 7ffffd7d 7ffffc1d 7ffffc1d 7ffffc1d } . . . v . . . 8 . . . . .  
[7ffff8f0] 7ffffc00 7ffffb8e 7ffffb8e 7ffffb8e . . . . . . . . . . .  
[7ffff900] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 . . . . . . . . . . .  
[7ffff910] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 . . . . . . . . . . .  
[7ffff920] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 . . . . . . . . . . .  
[7ffff930] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 . . . . . . . . . . .  
[7ffff940] 00000000 00000000 00000000 00000000 . . . . . . . . . . .  
[7ffff950] 67777777 67777777 67777777 67777777 . . . . . . . . . . .  
[7ffff960] 6f777777 6f777777 6f777777 6f777777 . . . . . . . . . . .  
[7ffff970] 3a777777 3a777777 3a777777 3a777777 . . . . . . . . . . .  
[7ffff980] 4f777777 4f777777 4f777777 4f777777 . . . . . . . . . . .  
[7ffff990] 69777777 69777777 69777777 69777777 . . . . . . . . . . .  
[7ffff9a0] 4e777777 4e777777 4e777777 4e777777 . . . . . . . . . . .  
[7ffff9b0] 6f777777 6f777777 6f777777 6f777777 . . . . . . . . . . .  
[7ffff9c0] 4d414f52 50474e49 49464f52 443d454c R O A M I N G P R O F I L E = D  
[7ffff9d0] 544b5345 412d504f 4f4e4b4e 55003548 E S K T O P - A N K N O H S . U

Sezione di dati in memoria (utente, stack e kernel)

- Indirizzo (word) o intervallo di indirizzi hex
- Contenuto memoria in esadecimale (little- o big- endian dipende da proc. / MS-Windows: little-endian)
- Contenuto memoria in ASCII

# 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`
  - Editor consigliato: notepad++

```

                                .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`
  - Editor consigliato: notepad

```
op1:
op2:
res:

.data # dichiarazione variabili
.byte 3
.byte 2
.space 1 # allocazione memoria

.text # codice per uscita dal programma
.globl main
.ent main
main:
    lz $t1, op1
    lb $t2, op2
    add $t1, $t1, $t2 # risultato
    sb $t1, res # memoria
    li $v0, 10
    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

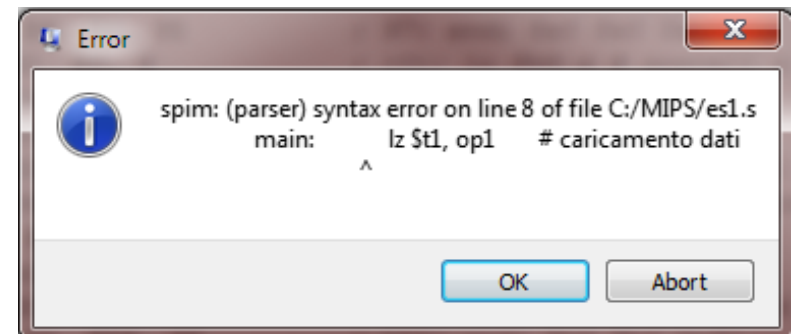
# Caricamento del codice

- 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:

Data	Text
Data	
User data segment [10000000]..[10040000]	
[10000000]..[1000ffff]	00000000
[10010000]	00000203 00000000 00000000 . . . . .
[10010010]..[1003ffff]	00000000

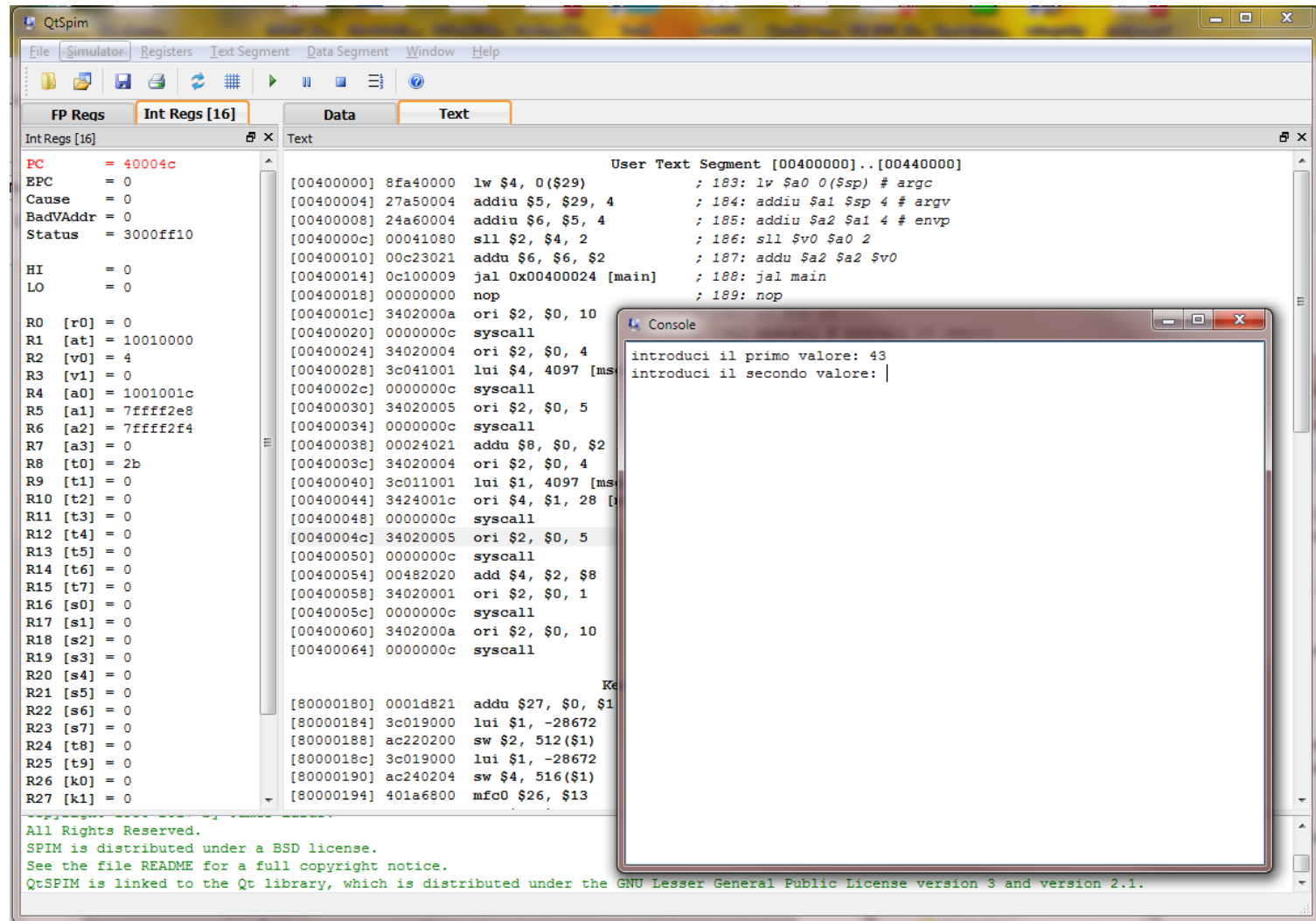
- Variabili dopo il salvataggio:

Data	Text
Data	
User data segment [10000000]..[10040000]	
[10000000]..[1000ffff]	00000000
[10010000]	00050203 00000000 00000000 00000000 . . . . .
[10010010]..[1003ffff]	00000000

# 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

```
variabile:  .data
            .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: .ascii  "\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 - stampa 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.