

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"

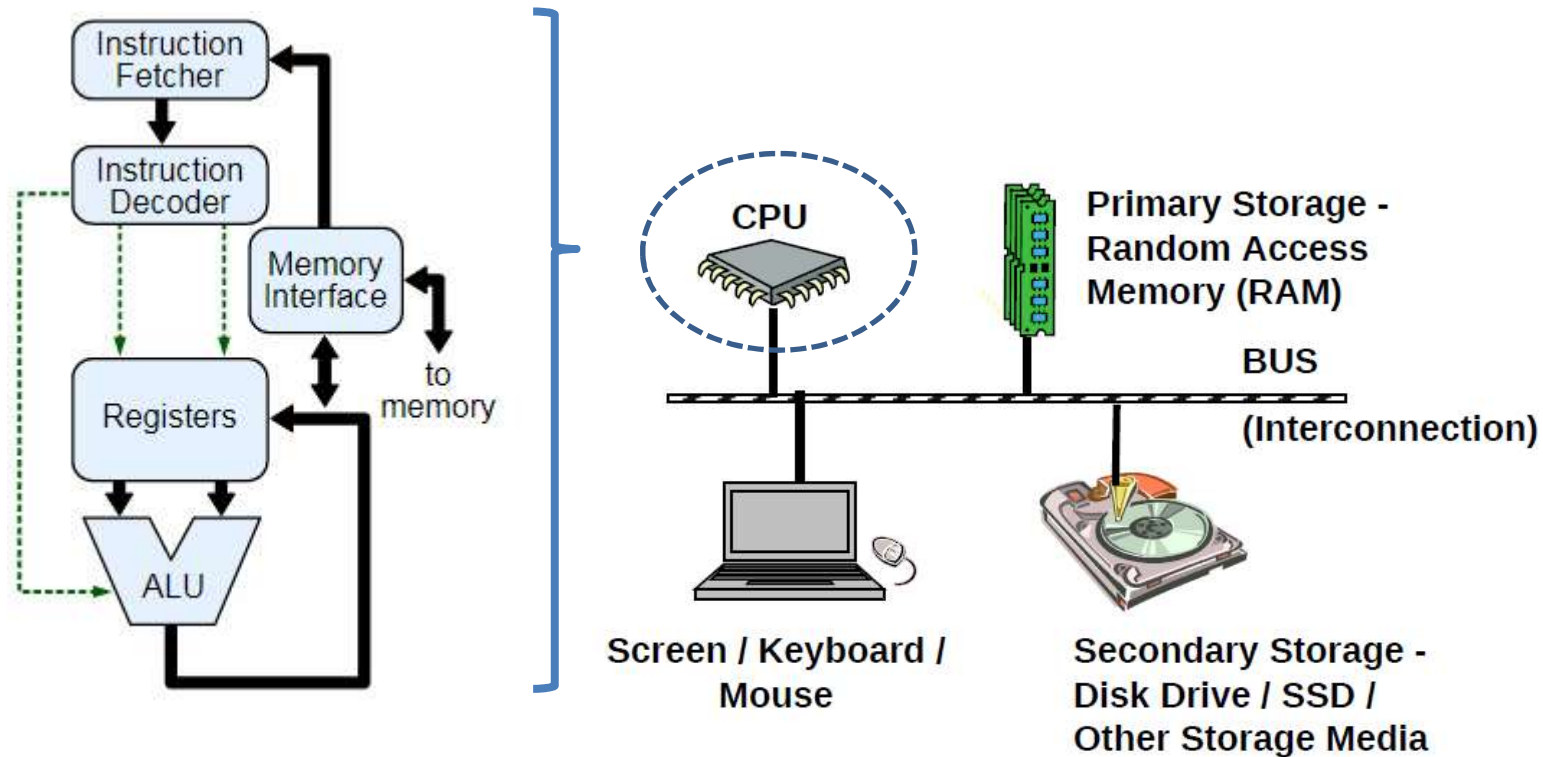
https://join.slack.com/t/calcolatoriel-l2h9254/shared_invite/zt-n8i1yo92-sB_ymmZ6OmOGyXLNYiuIPw

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

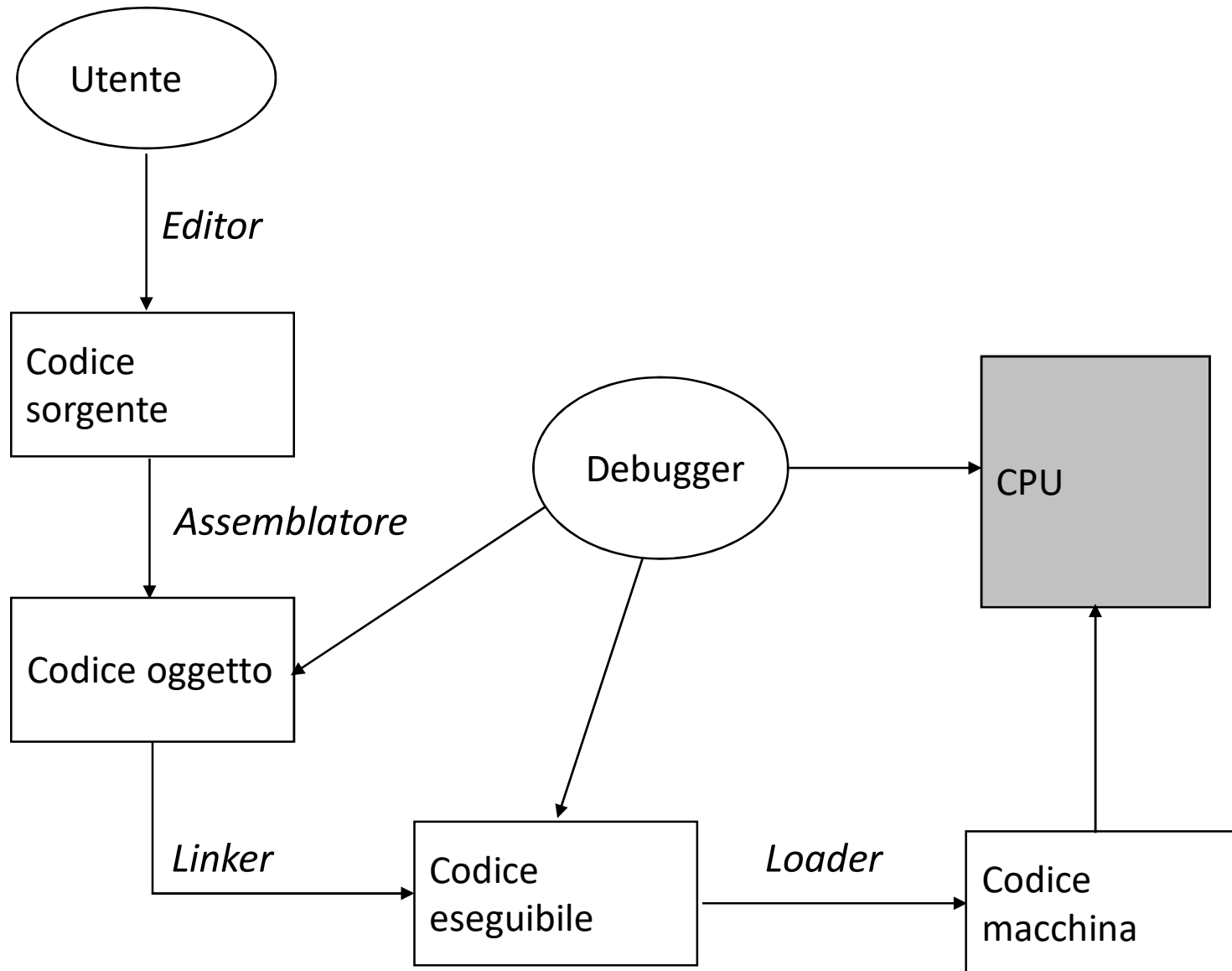
- Schema dal punto di vista del programmatore in linguaggio Assembly



Architettura MIPS32 - Registri

Nome	Numero	Uso
\$zero	\$0	il valore costante 0
\$at	\$1	<i>riservato per assembler</i>
\$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	<i>riservato per sistema operativo</i>
\$gp	\$28	<i>riservato per puntatore al segmento di variabili globali</i>
\$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

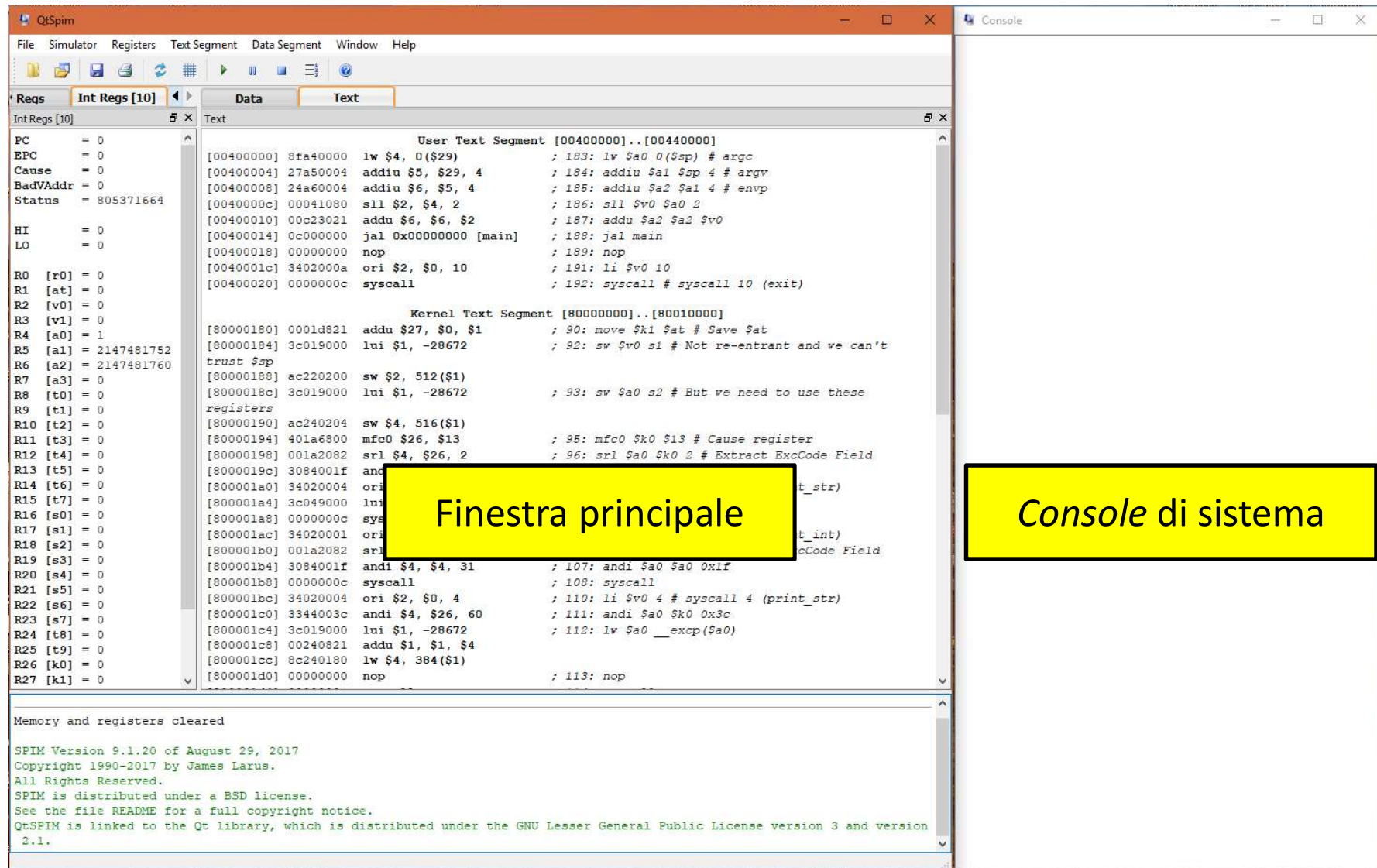
- Installare il pacchetto "language-mips" scaricabile da <https://atom.io/packages/language-mips>
- La colorazione della sintassi è applicata se l'estensione del file è .s o .asm



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 Pseudolstruzioni)
 - 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

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

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) ; 93: sw $a0 $2 # But we need to use these
[8000018c] 3c019000 lui $1, -28672 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 interface with several panes. The 'Int Regs [10]' pane on the left lists integer registers (PC, EPC, Cause, BadVAddr, Status, HI, LO, R0-R18) with their current values. The 'Text' pane on the right displays disassembled code for the 'User Text Segment' and 'Kernel Text Segment'. The 'Registers' pane at the top contains icons for various operations. Callouts provide additional context for these elements.

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
[80000184] 3c019000 lui $1, -28672
trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672
registers
[80000190] ac240204 sw $4, 516($1)
[80000194] 401a6800 mfc0 $26, $13
[80000198] 001a2082 srl $4, $26, 2
[8000019c] 3084001f andi $4, $4, 31
[800001a0] 34020004 ori $2, $0, 4
[800001a4] 3c049000 lui $4, -28672 [__m1_]
[800001a8] 0000000c syscall
[800001ac] 34020001 ori $2, $0, 1
[800001b0] 001a2082 srl $4, $26, 2
[800001b4] 3084001f andi $4, $4, 31
[800001b8] 0000000c syscall
[800001bc] 34020004 ori $2, $0, 4
[800001c0] 3344003c andi $4, $26, 60
[800001c4] 3c019000 lui $1, -28672
[800001c8] 00240821 addu $1, $1, $4
[800001cc] 8c240180 lw $4, 8c240180($1)
[800001d0] 00000000 nop

; 90: move $k1 $at # Save $at
; 92: sw $v0 $1 # Not re-entrant and we can't
; 93: sw $a0 $2 # But we need to use these
; 95: mfc0 $k0 $13 # Cause register
; 96: srl $a0 $k0 2 # Extract ExcCode Field
; 97: andi $a0 $a0 0x1f
; 101: li $v0 4 # syscall 4 (print_str)
; 102: la $a0 __m1_
; 103: syscall
; 105: li $v0 1 # syscall 1 (print_int)
; 106: srl $a0 $k0 2 # Extract ExcCode Field
; 107: andi $a0 $a0 0x1f
; 108: syscall
; 110: li $v0 4 # syscall 4 (print_str)
; 111: andi $a0 $k0 0x3c
; 112: lw $a0 __excp($a0)

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

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

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 7ffffe36 7ffffe19 k . . . C . . . 6
[7ffff8d0] 7ffffde8 7ffffdb3 7ffffdbb
[7ffff8e0] 7ffffd7d 7ffffdcd 7ffffcld } . . . v . . . 8
[7ffff8f0] 7ffffc00 7ffffb8e
[7ffff900] 7ffffb73 7ffffb8e
[7ffff910] 7ffffb73 7ffffb8e
[7ffff920] 7ffffb73 7ffffb8e
[7ffff930] 7ffffb73 7ffffb8e
[7ffff940] 00000001 7ffff946 00000000
[7ffff950] 67
[7ffff960] 6f
[7ffff970] 3a
[7ffff980] 4f
[7ffff990] 69
[7ffff9a0] 4e
[7ffff9b0] 6f
[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 5 . 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 dal processore. MS-Windows usa 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`

```

                                .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`

```
op1:      .byte 3
op2:      .byte 2
res:      .space 1    # allocazione di memoria per il risultato

.text
.globl main
.ent main

main:     lz $t1, op1
          lb $t2, op2
          add $t1, $t1, $t2 # calcolo della somma
          sb $t1, res      # salvataggio del risultato in memoria
          li $v0, 10        # preparazione del ritorno 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

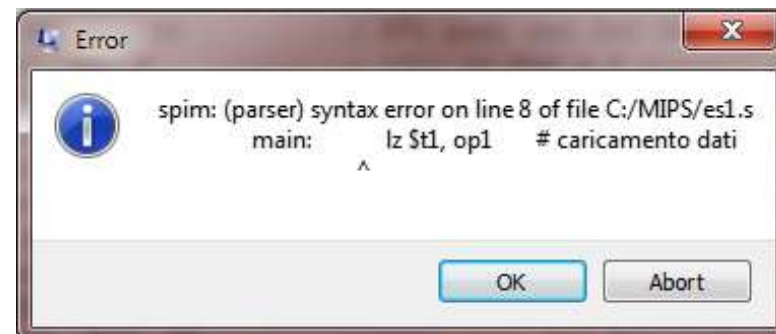
Caricamento del codice

- 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:

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

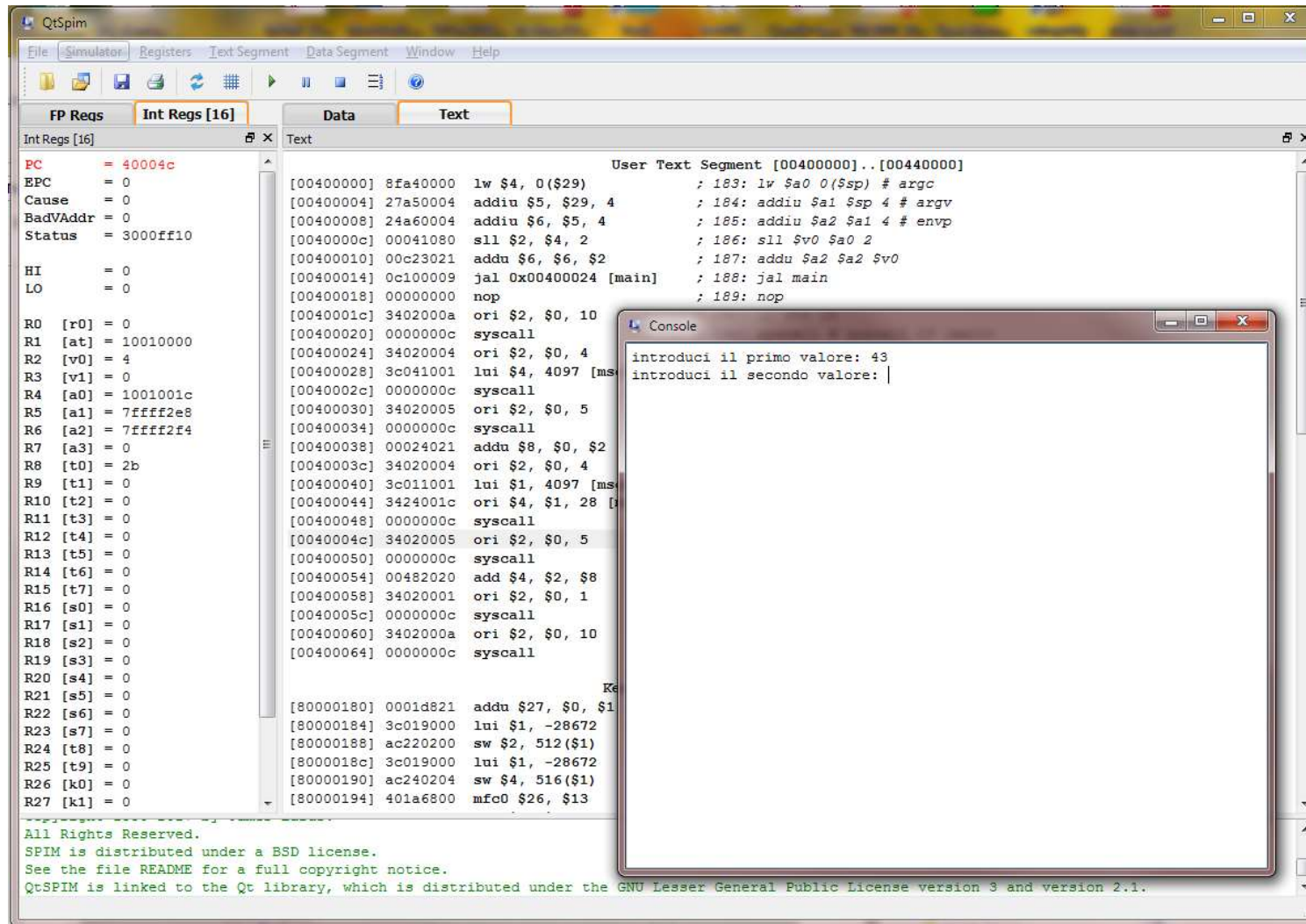
- Variabili dopo il salvataggio:

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

Input/Output da *console*

```
msg1:      .data
           .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: .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.