

# Calcolatori Elettronici

## Esercitazioni Assembler

M. Sonza Reorda – M. Monetti

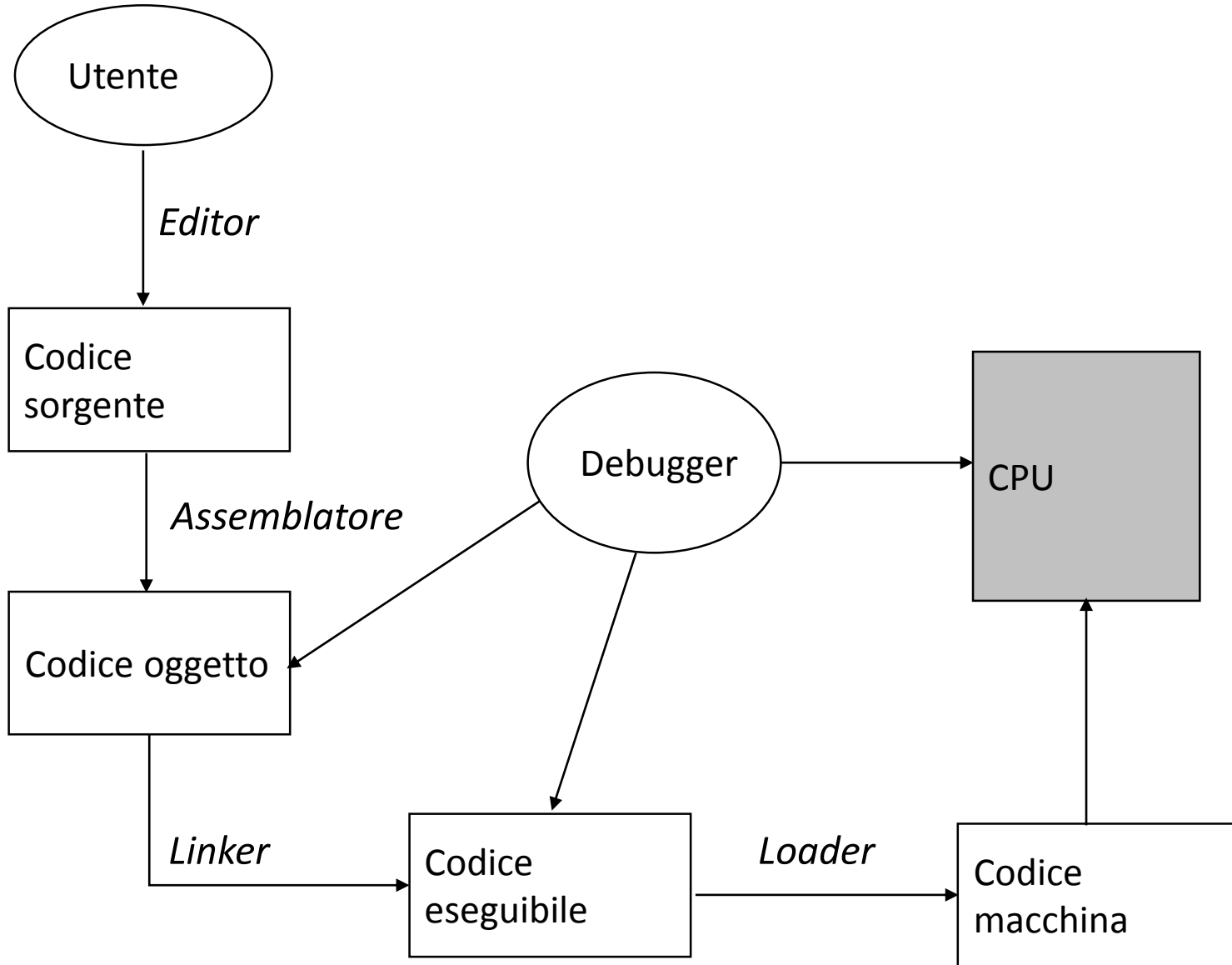
AA 2021/2022

[massimo.monetti@polito.it](mailto:massimo.monetti@polito.it)

Politecnico di Torino

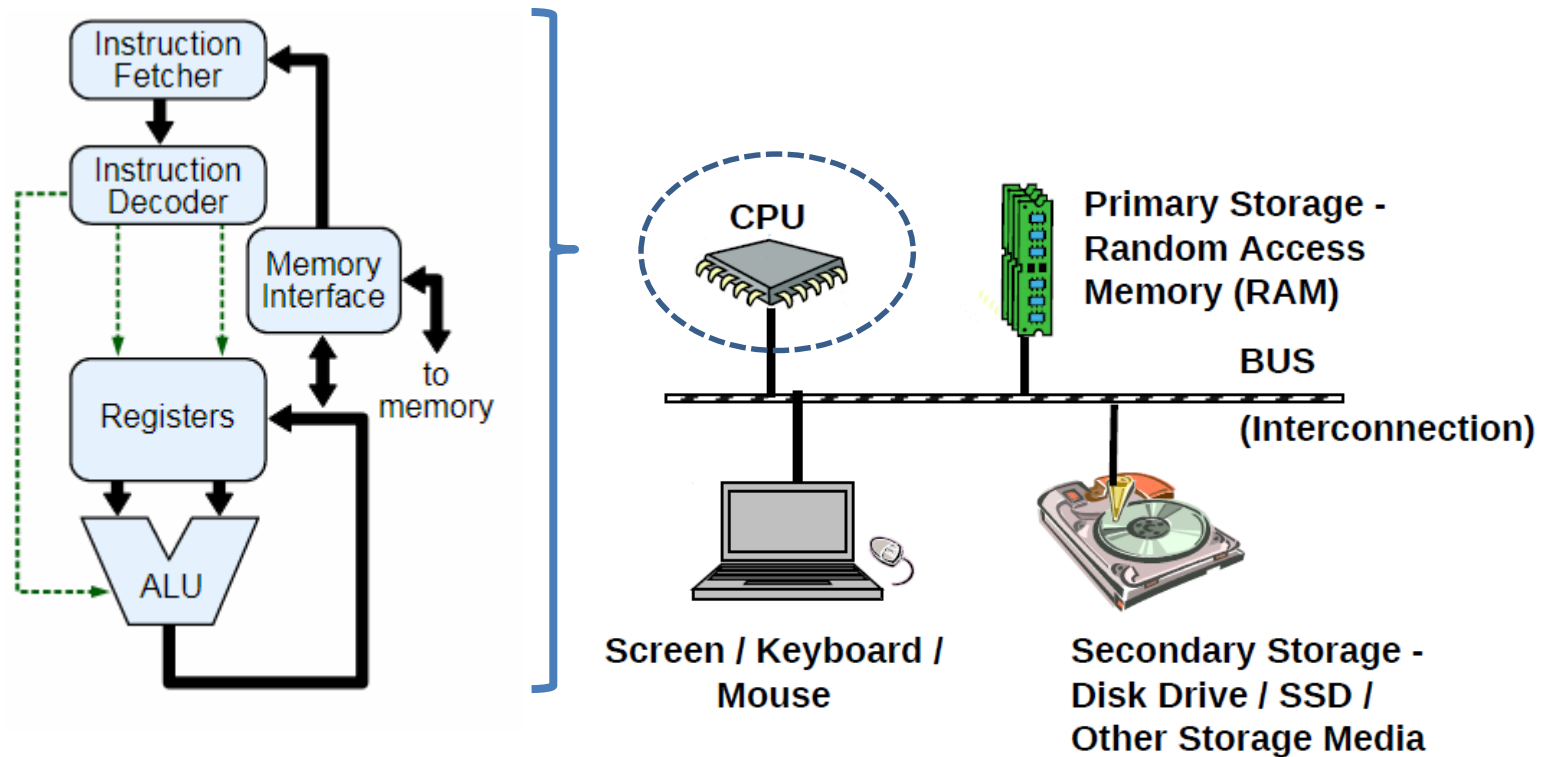
Dipartimento di Automatica e Informatica

# Ciclo di vita di un programma



# Il calcolatore

Schema dal punto di vista del programmatore in linguaggio Assembly



# Architettura MIPS - Registri

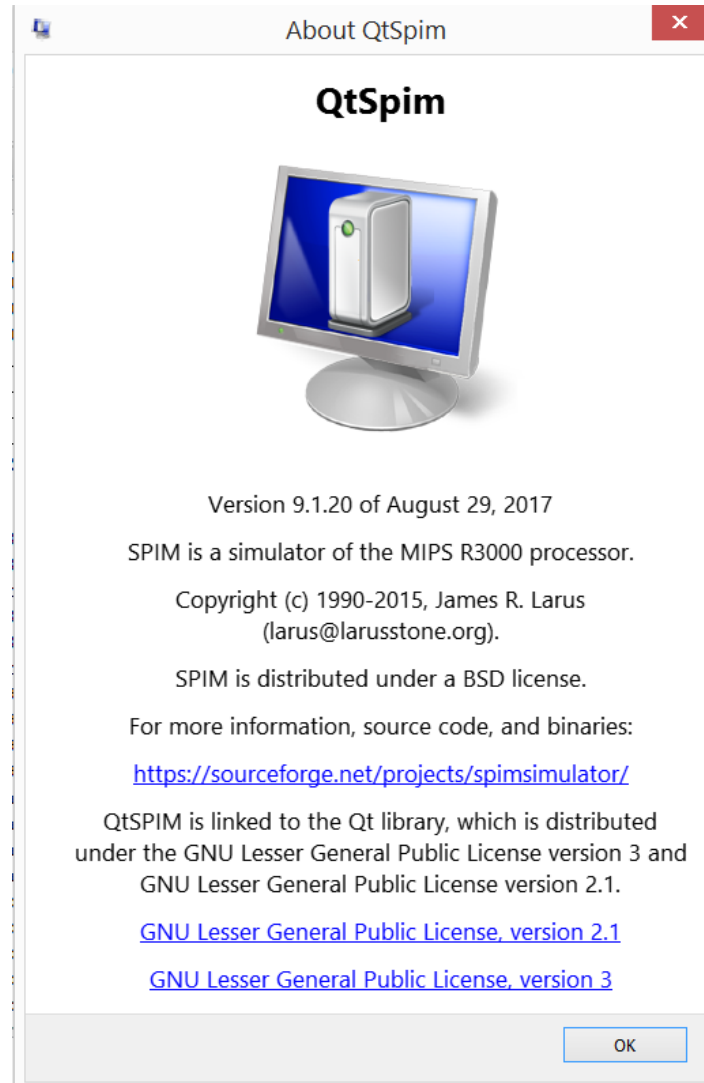
Register Name= \$ + valore alfanumerico

Register Number= \$ + valore numerico

Register Name	Register Number	Register Usage
\$zero	\$0	Hardware set to 0
\$at	\$1	Assembler temporary
\$v0 - \$v1	\$2 - \$3	Function result (low/high)
\$a0 - \$a3	\$4 - \$7	Argument Register 1
\$t0 - \$t7	\$8 - \$15	Temporary registers
\$s0 - \$s7	\$16 - \$23	Saved registers
\$t8 - \$t9	\$24 - \$25	Temporary registers
\$k0 - \$k1	\$26 - \$27	Reserved for OS kernel
\$gp	\$28	Global pointer
\$sp	\$29	Stack pointer
\$fp	\$30	Frame pointer
\$ra	\$31	Return address



# QtSpim





# 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)



# QtSpim

- 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>

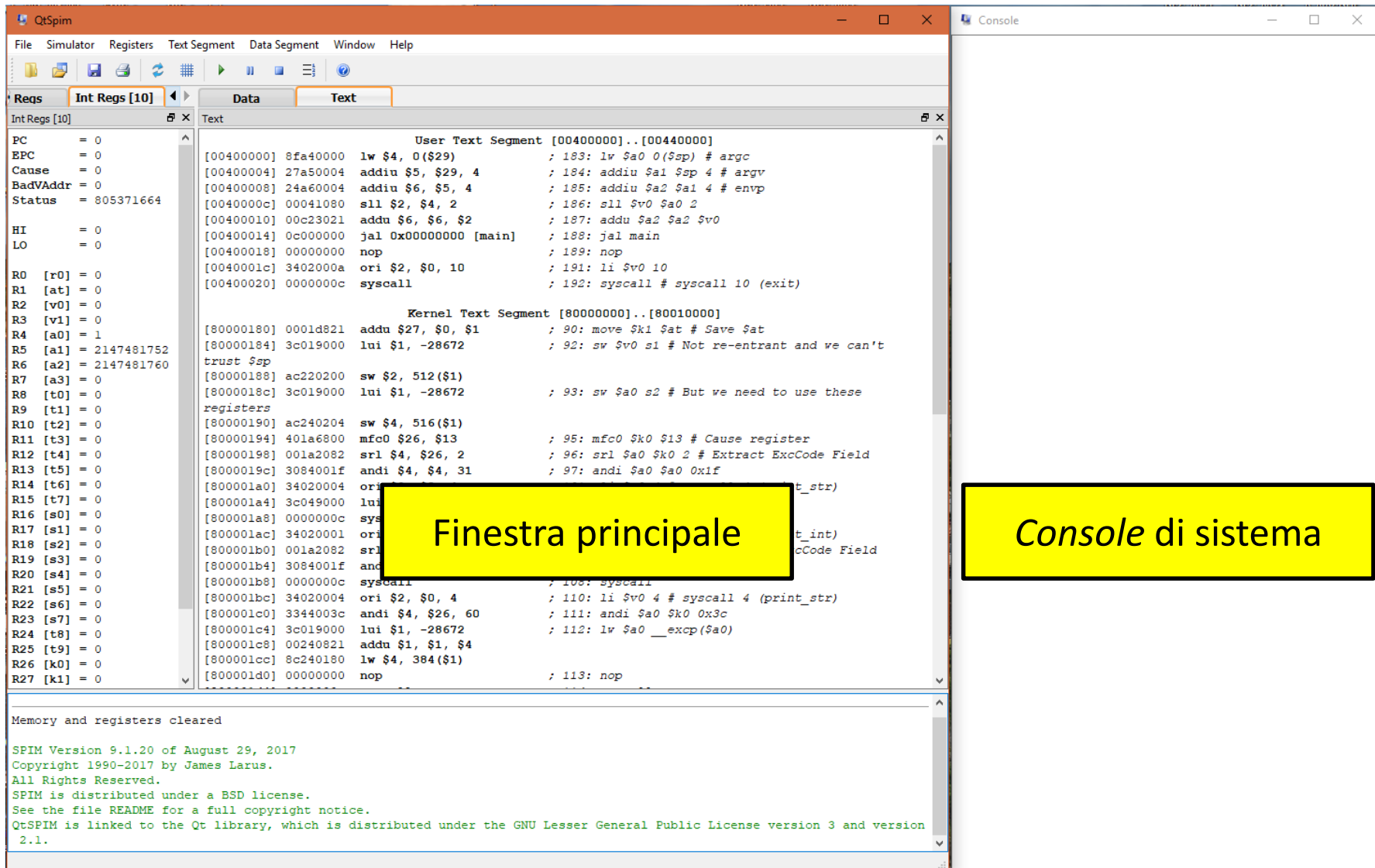


# QtSpim

- 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/>
- Versione in uso [QtSpim 9.1.20 Windows.msi](#)
- Esiste una versione successiva (9.1.23) che si può facoltativamente utilizzare
- Per qualsiasi problema, è possibile
  - Rivolgersi all'esercitatore o ai borsisti in laboratorio
  - Contattare l'esercitatore via email.



# Interfaccia di QtSpim



# Finestra principale

The screenshot shows the QtSpim MIPS simulator interface. The top menu bar includes File, Simulator, Registers, Text Segment, Data Segment, Window, and Help. Below the menu is a toolbar with icons for file operations, simulation control, and viewing options. The main window is divided into several panes: 'Regs' (Registers), 'Int Regs [10]' (Integer Registers), 'Data', and 'Text'. The 'Int Regs [10]' pane shows a list of registers (R0-R18) with their current values. The 'Text' pane displays assembly code with addresses and comments. Two callout boxes provide additional information:

**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**

- Istruzione disassemblata
- Istruzione sorgente e commenti

# Finestra principale

```
R4 [a0] = 1      [80000180] 0001d821 addu $27, $0, $1      ; 90: move $k1 $at # Save $at
R5 [a1] = 2147481752 [80000184] 3c019000 lui $1, -28672      ; 92: sw $v0 $1 # Not re-entrant and we can't
R6 [a2] = 2147481760 trust $sp
R7 [a3] = 0      [80000188] ac220200 sw $2, 512($1)
R8 [t0] = 0      [8000018c] 3c019000 lui $1, -28672      ; 93: sw $a0 $2 # But we need to use these
R9 [t1] = 0      registers
R10 [t2] = 0     [80000190] ac240204 sw $4, 516($1)
R11 [t3] = 0     [80000194] 401a6800 mfc0 $26, $13      ; 95: mfc0 $k0 $13 # Cause register
R12 [t4] = 0     [80000198] 001a2082 srl $4, $26, 2      ; 96: srl $a0 $k0 2 # Extract ExcCode Field
R13 [t5] = 0     [8000019c] 3084001f andi $4, $4, 31      ; 97: andi $a0 $a0 0x1f
R14 [t6] = 0     [800001a0] 34020004 ori $2, $0, 4      ; 101: li $v0 4 # syscall 4 (print_str)
R15 [t7] = 0     [800001a4] 3c049000 lui $4, -28672 [__m1_] ; 102: la $a0 __m1_
R16 [s0] = 0     [800001a8] 0000000c syscall      ; 103: syscall
R17 [s1] = 0     [800001ac] 34020001 ori $2, $0, 1      ; 105: li $v0 1 # syscall 1 (print_int)
R18 [s2] = 0     [800001b0] 001a2082 srl $4, $26, 2      ; 106: srl $a0 $k0 2 # Extract ExcCode Field
R19 [s3] = 0     [800001b4] 3084001f andi $4, $4, 31      ; 107: andi $a0 $a0 0x1f
R20 [s4] = 0     [800001b8] 0000000c syscall      ; 108: syscall
R21 [s5] = 0     [800001bc] 34020004 ori $2, $0, 4      ; 110: li $v0 4 # syscall 4 (print_str)
R22 [s6] = 0     [800001c0] 3344003c andi $4, $26, 60      ; 111: andi $a0 $k0 0x3c
R23 [s7] = 0     [800001c4] 3c019000 lui $1, -28672      ; 112: lw $a0 __excp($a0)
R24 [t8] = 0     [800001c8] 00240821 addu $1, $1, $4
R25 [t9] = 0     [800001cc] 8c240180 lw $4, 0($1)
R26 [k0] = 0     [800001d0] 00000000 nop
R27 [k1] = 0
```

Console informativa

- Messaggi di informazione e di errore

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.

# 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 7ffffef2 7ffffe36 7ffffe19 k . . . C . . . 6 . . . . .  
[7ffff8d0] 7ffffde8 7ffffdb3 7ffffdb3 7ffffd8b . . . . .  
[7ffff8e0] 7ffffd7d 7ffffd7d 7ffffd7d 7ffffd7d } . . . v . . . 8 . . . . .  
[7ffff8f0] 7ffffc00 7ffffb8e 7ffffb8e 7ffffb8e . . . . .  
[7ffff900] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 s . . . O . . . f . . . . .  
[7ffff910] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 . . . . .  
[7ffff920] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 . . . . .  
[7ffff930] 7ffffb73 7ffffb73 7ffffb73 7ffffb73 . . . . .  
[7ffff940] 00000000 00000000 00000000 00000000 . . . . .  
[7ffff950] 67 67 67 67 . . . . .  
[7ffff960] 6f 6f 6f 6f . . . . .  
[7ffff970] 3a 3a 3a 3a . . . . .  
[7ffff980] 4f 4f 4f 4f . . . . .  
[7ffff990] 69 69 69 69 . . . . .  
[7ffff9a0] 4e 4e 4e 4e . . . . .  
[7ffff9b0] 6f 6f 6f 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 da proc. / MS-Windows: little-endian)
- Contenuto memoria in ASCII

# Template

# Name and general description of program

# -----

# Data declarations go in this section.

**.data**

# program specific data declarations

# -----

# Program code goes in this section.

**.text**

**.globl main**

**.ent main**

**main:**

# ----

# >>>> your program code goes here.

# ----

# Done, terminate program.

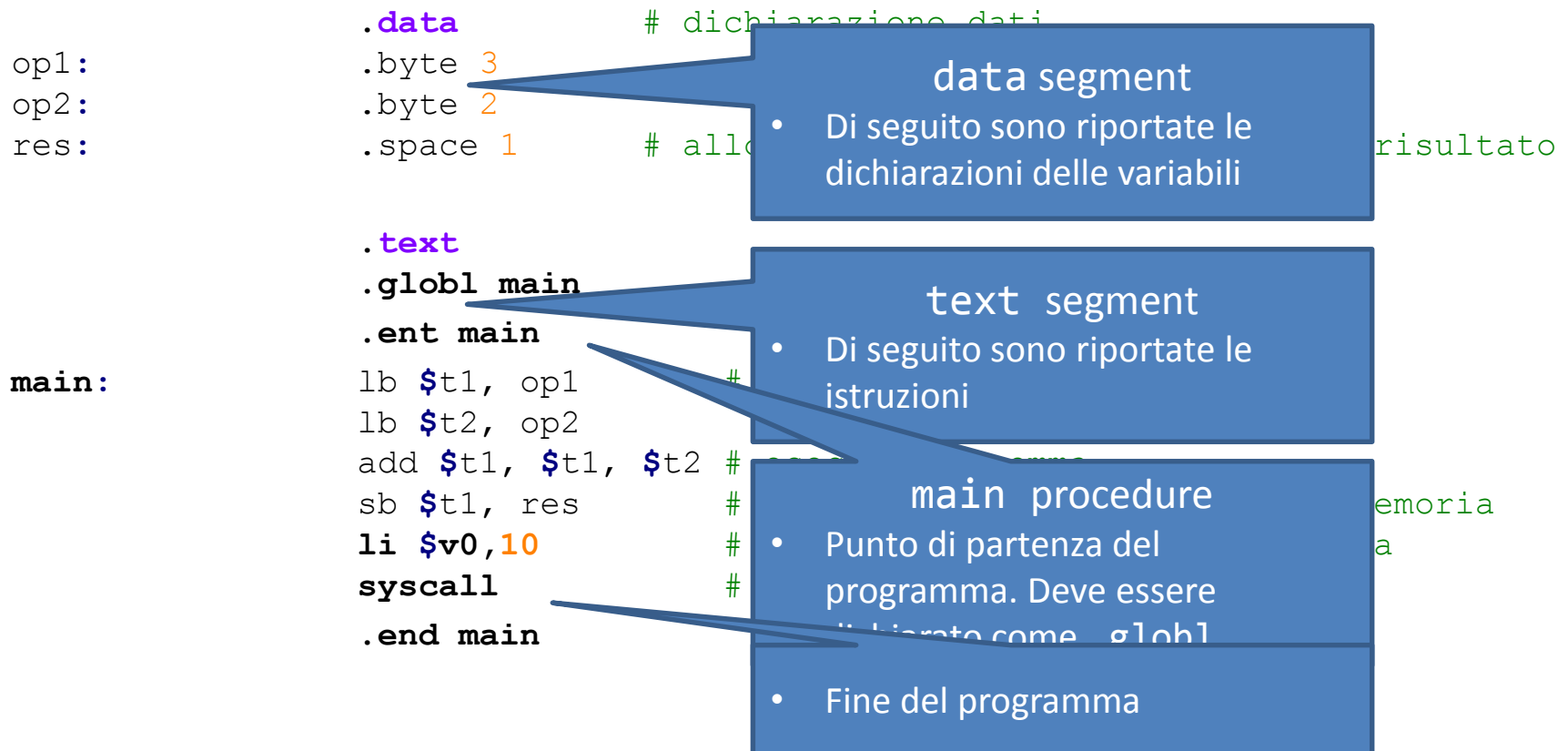
**li \$v0, 10**

**Syscall**

**.end main**

# Codice di esempio

- Il codice può essere introdotto con un qualsiasi editor di testo, e salvato in un file con estensione **.a**, **.s** oppure **.asm**
  - Editor consigliato: notepad++



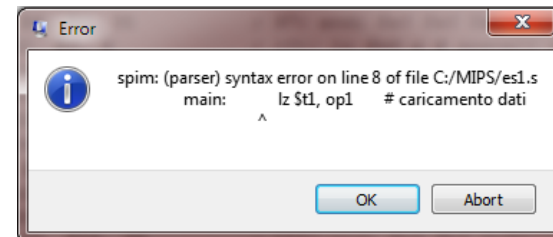
# Caricamento del codice

- 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



# Tipi dato e dimensioni

I data types base sono : **integer**, **floating-point**, e **characters**.

L'architettura MIPS utilizza le seguenti dimensioni di data/memory :

- Byte (**8 bit**)
- Halfword (semplicemente *half*) (**16 bit**)
- Word (**32 bit**)

**Floating-point** ha dimensioni di un word (32-bit) o un double word (64-bit).

**Character** ha tipicamente dimensioni di 1 byte e una stringa è una serie di byte in sequenza



# Istruzioni e pseudo-istruzioni

Una istruzione nativa (bare-instruction) è un'istruzione che viene eseguita dalla CPU.

Una pseudo-istruzione è un'istruzione che l'assemblatore, o simulatore, riconosce, ma deve poi convertire in una o più istruzioni native.

• - Indicates an actual MIPS instruction. Others are SPIM pseudoinstructions.

## Instruction Function

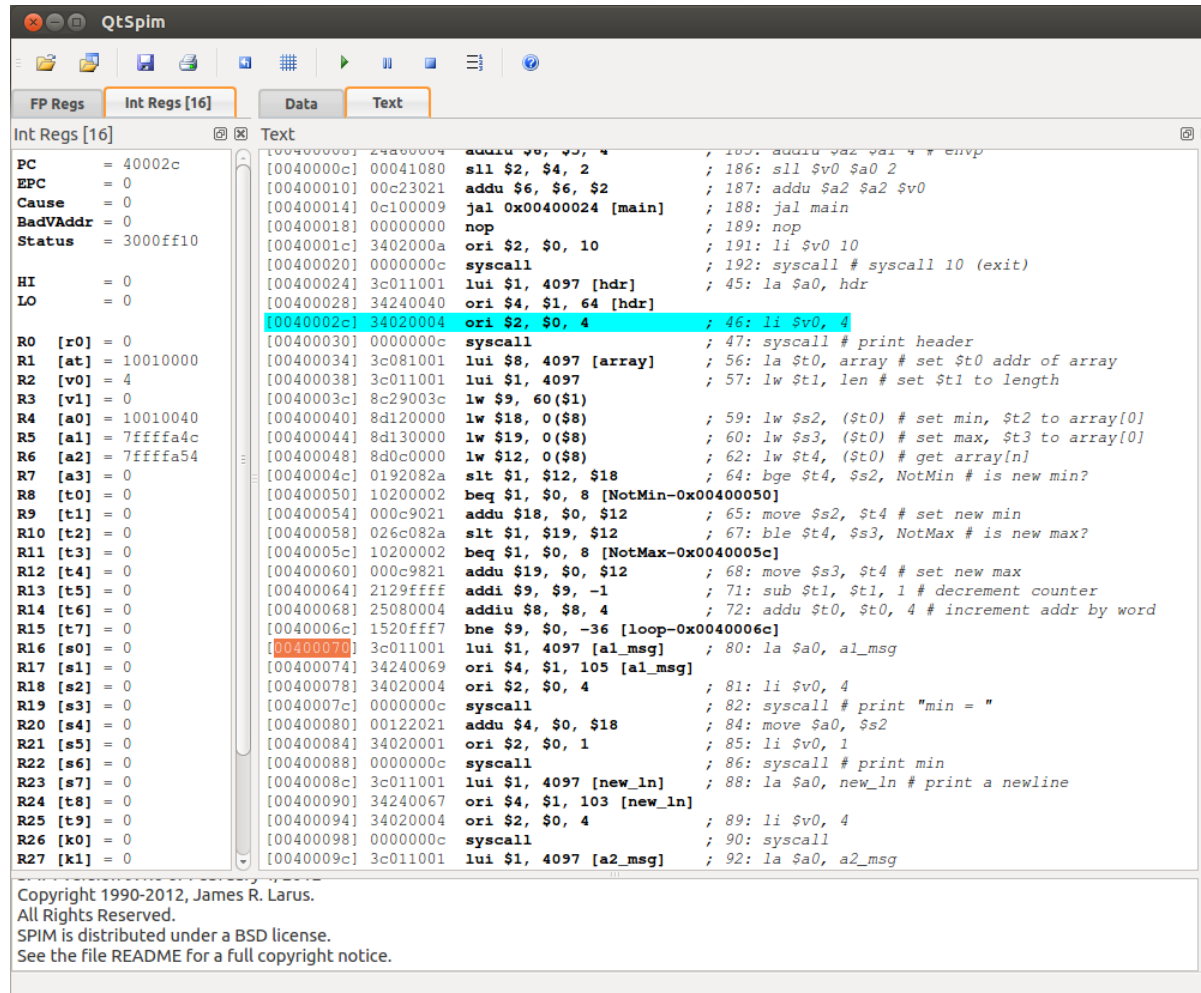
- add Rd, Rs, Rt  $Rd = Rs + Rt$  (signed)
- addu Rd, Rs, Rt  $Rd = Rs + Rt$  (unsigned)
- addi Rd, Rs, Imm  $Rd = Rs + Imm$  (signed)
- sub Rd, Rs, Rt  $Rd = Rs - Rt$  (signed)
- subu Rd, Rs, Rt  $Rd = Rs - Rt$  (unsigned)
- div Rs, Rt lo =  $Rs/Rt$ , hi =  $Rs \bmod Rt$  (integer division, signed)
- divu Rs, Rt lo =  $Rs/Rt$ , hi =  $Rs \bmod Rt$  (integer division, unsigned)
- div Rd, Rs, Rt  $Rd = Rs/Rt$  (integer division, signed)
- divu Rd, Rs, Rt  $Rd = Rs/Rt$  (integer division, unsigned)
- rem Rd, Rs, Rt  $Rd = Rs \bmod Rt$  (signed)
- remu Rd, Rs, Rt  $Rd = Rs \bmod Rt$  (unsigned)
- mul Rd, Rs, Rt  $Rd = Rs * Rt$  (signed)

- mult Rs, Rt hi, lo =  $Rs * Rt$  (signed, hi = high 32 bits, lo = low 32 bits)
- multu Rd, Rs hi, lo =  $Rs * Rt$  (unsigned, hi = high 32 bits, lo = low 32 bits)
- and Rd, Rs, Rt  $Rd = Rs \bullet Rt$
- andi Rd, Rs, Imm  $Rd = Rs \bullet Imm$
- neg Rd, Rs  $Rd = -(Rs)$
- nor Rd, Rs, Rt  $Rd = (Rs + Rt)'$
- not Rd, Rs  $Rd = (Rs)'$
- or Rd, Rs, Rt  $Rd = Rs + Rt$
- .....

# Istruzioni e pseudo-istruzioni

**ori \$2, \$0, 4**

**; 46: li \$v0, 4**



The screenshot shows the QtSpim MIPS simulator interface. The 'Text' tab is selected, displaying assembly code. The 'Int Regs [16]' panel on the left shows the current state of the registers. The assembly code is as follows:

```
00000000: 24000004 addiu $0, $0, 4 ; 105: addiu $a2, $a1, 4 # envp
0000000c: 00041080 sll $2, $4, 2 ; 186: sll $v0, $a0, 2
00000010: 00c23021 addu $6, $6, $2 ; 187: addu $a2, $a2, $v0
00000014: 0c100009 jal 0x00400024 [main] ; 188: jal main
00000018: 00000000 nop ; 189: nop
0000001c: 3402000a ori $2, $0, 10 ; 191: li $v0, 10
00000020: 0000000c syscall ; 192: syscall # syscall 10 (exit)
00000024: 3c011001 lui $1, 4097 [hdr] ; 45: la $a0, hdr
00000028: 34240040 ori $4, $1, 64 [hdr]
0000002c: 34020004 ori $2, $0, 4 ; 46: li $v0, 4
00000030: 0000000c syscall ; 47: syscall # print header
00000034: 3c081001 lui $8, 4097 [array] ; 56: la $t0, array # set $t0 addr of array
00000038: 3c011001 lui $1, 4097 ; 57: lw $t1, len # set $t1 to length
0000003c: 8c29003c lw $9, 60($1)
00000040: 8d120000 lw $18, 0($8) ; 59: lw $s2, ($t0) # set min, $t2 to array[0]
00000044: 8d130000 lw $19, 0($8) ; 60: lw $s3, ($t0) # set max, $t3 to array[0]
00000048: 8d0c0000 lw $12, 0($8) ; 62: lw $t4, ($t0) # get array[n]
0000004c: 0192082a slt $1, $12, $18 ; 64: bge $t4, $s2, NotMin # is new min?
00000050: 10200002 beq $1, $0, 8 [NotMin-0x00400050]
00000054: 000c9021 addu $18, $0, $12 ; 65: move $s2, $t4 # set new min
00000058: 026c082a slt $1, $19, $12 ; 67: ble $t4, $s3, NotMax # is new max?
0000005c: 10200002 beq $1, $0, 8 [NotMax-0x0040005c]
00000060: 000c9821 addu $19, $0, $12 ; 68: move $s3, $t4 # set new max
00000064: 2129ffff addi $9, $9, -1 ; 71: sub $t1, $t1, 1 # decrement counter
00000068: 25080004 addiu $8, $8, 4 ; 72: addu $t0, $t0, 4 # increment addr by word
0000006c: 1520ffff bne $9, $0, -36 [loop-0x0040006c]
00000070: 3c011001 lui $1, 4097 [a1_msg] ; 80: la $a0, a1_msg
00000074: 34240069 ori $4, $1, 105 [a1_msg]
00000078: 34020004 ori $2, $0, 4 ; 81: li $v0, 4
0000007c: 0000000c syscall ; 82: syscall # print "min = "
00000080: 00122021 addu $4, $0, $18 ; 84: move $a0, $s2
00000084: 34020001 ori $2, $0, 1 ; 85: li $v0, 1
00000088: 0000000c syscall ; 86: syscall # print min
0000008c: 3c011001 lui $1, 4097 [new_ln] ; 88: la $a0, new_ln # print a newline
00000090: 34240067 ori $4, $1, 103 [new_ln]
00000094: 34020004 ori $2, $0, 4 ; 89: li $v0, 4
00000098: 0000000c syscall ; 90: syscall
0000009c: 3c011001 lui $1, 4097 [a2_msg] ; 92: la $a0, a2_msg
```

Copyright 1990-2012, James R. Larus.  
All Rights Reserved.  
SPIM is distributed under a BSD license.  
See the file README for a full copyright notice.

# Istruzioni di memoria

**LOAD (LW, LB)** (*Rdest*, *address*)



**STORE (SW, SB)** (*Rsrc*, *address*)



**MOVE** *Rdest*, *Rsrc*



# La memoria

Area Dati corrispondente alle seguenti dichiarazioni :

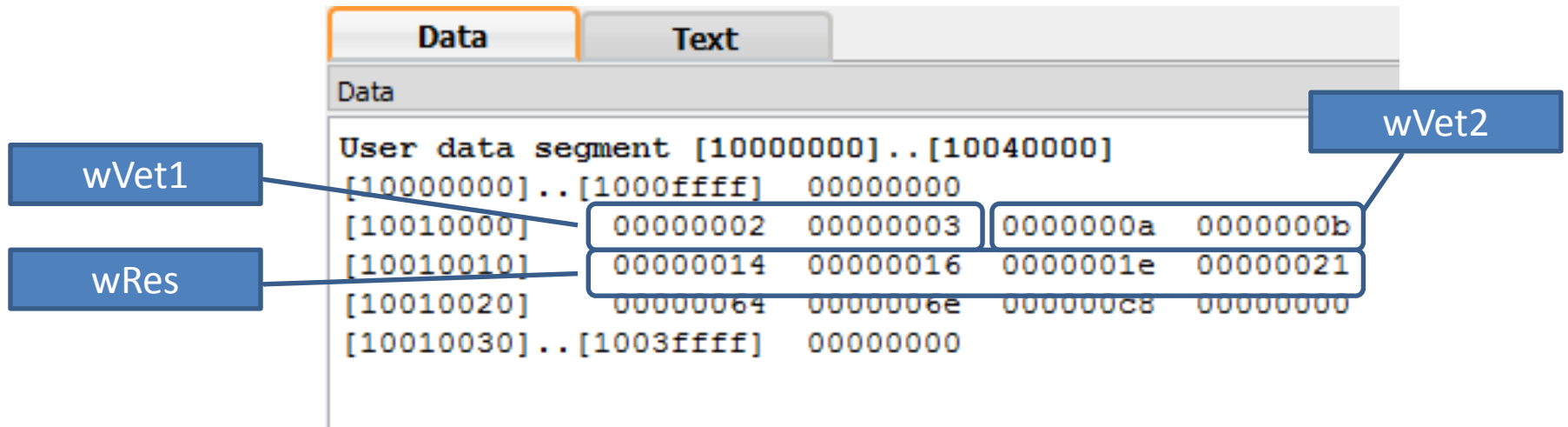
**wVet1:** .word 2, 3

**wVet2:** .word 10, 11

**wRes:** .space 8

Data	Text	
Data		
User data segment [10000000]..[10040000]		
[10000000]..[1000ffff]	00000000	
[10010000]	00000002	00000003 0000000a 0000000b
[10010010]..[1003ffff]	00000000	

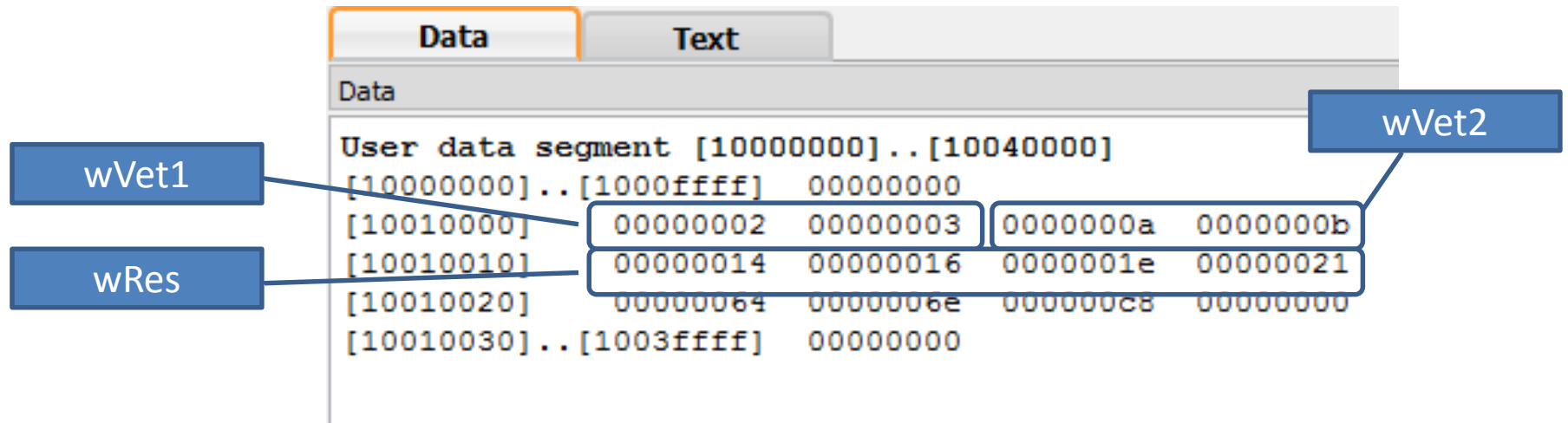
# La memoria



## LITTLE ENDIAN

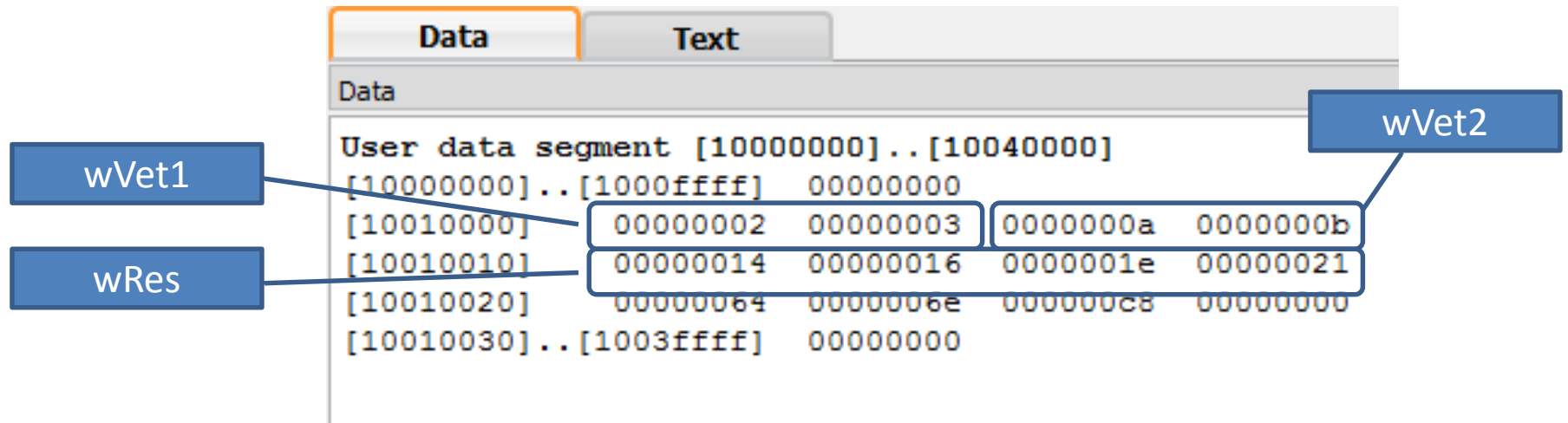
La memorizzazione parte dal byte meno significativo per finire col più significativo, per indirizzo di memoria crescente.

# La memoria



wVet1 (1)	10010000	02
	10010001	00
	10010002	00
	10010003	00
wVet1(2)	10010004	03
	10010005	00
	10010006	00
	10010007	00

# La memoria



wVet2(1)	10010008	0a
	10010009	00
	1001000A	00
	1001000B	00
wVet2(2)	1001000C	0b
	1001000D	00
	1001000E	00
	1001000F	00

# SysCall

Principali SysCall con cui gestire I/O

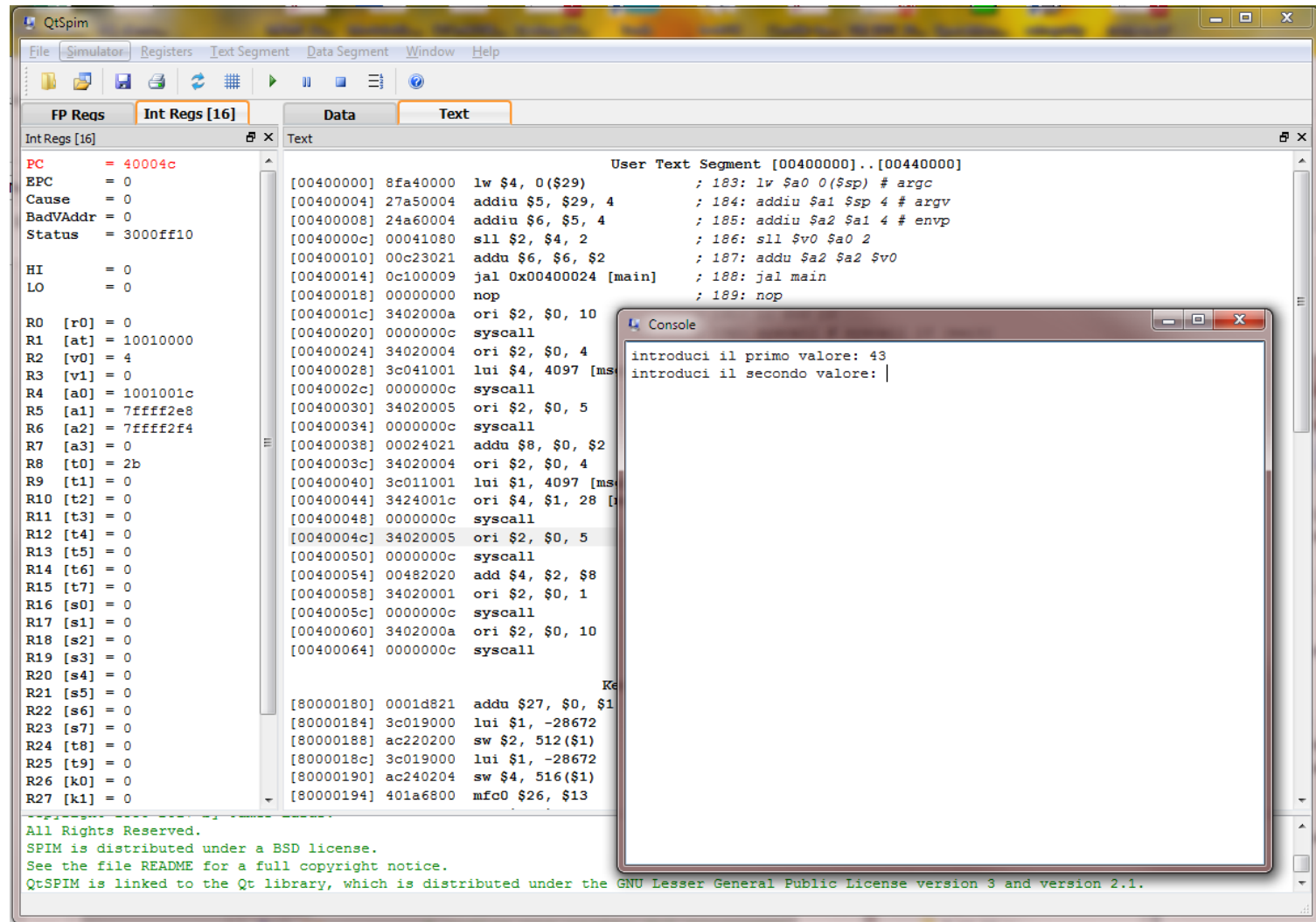
Service	System Call Code	Arguments	Result
Print_int	1	\$a0 = integer	
Print_float	2	\$f12 = float	
Print_double	3	\$f12 = double	
Print_string	4	\$a0 = string	
Read_int	5		Integer (in \$v0)
Read_float	6		Float (in \$f0)
Read_double	7		Double (in \$f0)
Read_string	8	\$a0 = buffer, \$a1 = length	
Sbrk	9	\$a0 = amount	Address (in \$v0)
exit	10		



# 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
```

# Debug

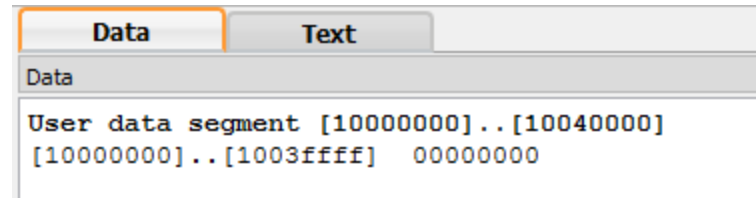
- L'esecuzione passo-passo è fondamentale per il *debug*
  - Osservare il valore di memoria e registri al termine di ogni istruzione
- È possibile inserire un *breakpoint* facendo click con il pulsante destro sull'istruzione desiderata nella sezione *Text* della finestra principale, e selezionando "*Set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

# Debug [cont.]

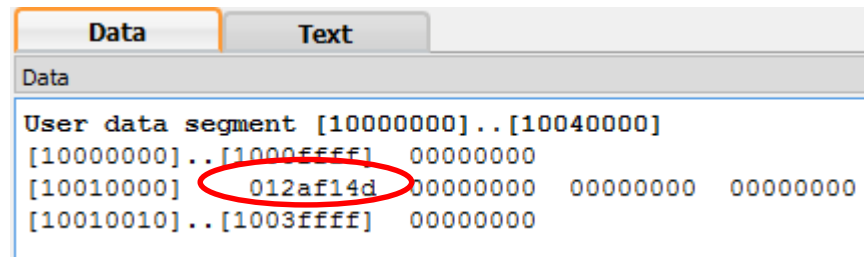
Esempio: esecuzione dell'istruzione di memorizzazione

```
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: syscall # syscall 10 (exit)
[00400024] 3c01012a lui $1, 298 ; 8: li $t0, 19591501 # variabile = 19591501 (012A F14D hex)
[00400028] 3428f14d ori $8, $1, -3763
[0040002c] 3c011001 lui $1, 4097 ; 9: sw $t0, variabile
[00400030] ac280000 sw $8, 0($1)
[00400034] 3402000a ori $2, $0, 10 ; 11: li $v0, 10
[00400038] 0000000c syscall ; 12: syscall
```

Variabili prima del salvataggio:



Variabili dopo il salvataggio:



# Esempio codice : Ricerca del carattere minimo

DIM=5

```
                .data
bVet:           .space 5
bRes:           .space 1
message_in :    .asciiz  "Inserire caratteri :"
message_out:    .ascii   "\nValore Minimo : "

                .text
                .globl main
                .ent main

main:

                la $t0, bVet                # puntatore a inizio del vettore
                li $t1, 0                   # contatore

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

# Esempio codice : 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, DIM, ciclo1     # itera 5 volte

        la  $t0, bVet
        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, DIM, ciclo2
```

## Esempio codice : 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
```



# ASCII Table

## ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

# Debug

Esempio: esecuzione dell'istruzione di memorizzazione

Data	
User data segment [10000000]..[10040000]	
[10000000]..[1000ffff]	00000000
[10010000]	34333231 6e490035 69726573 63206572 1 2 3 4 5 . I n s e r i r e c
[10010010]	74617261 69726574 203e3e20 560a000a a r a t t e r i > > . . . V
[10010020]	726f6c61 694d2065 6f6d696e 00203a20 a l o r e M i n i m o : .
[10010030]..[1003ffff]	00000000