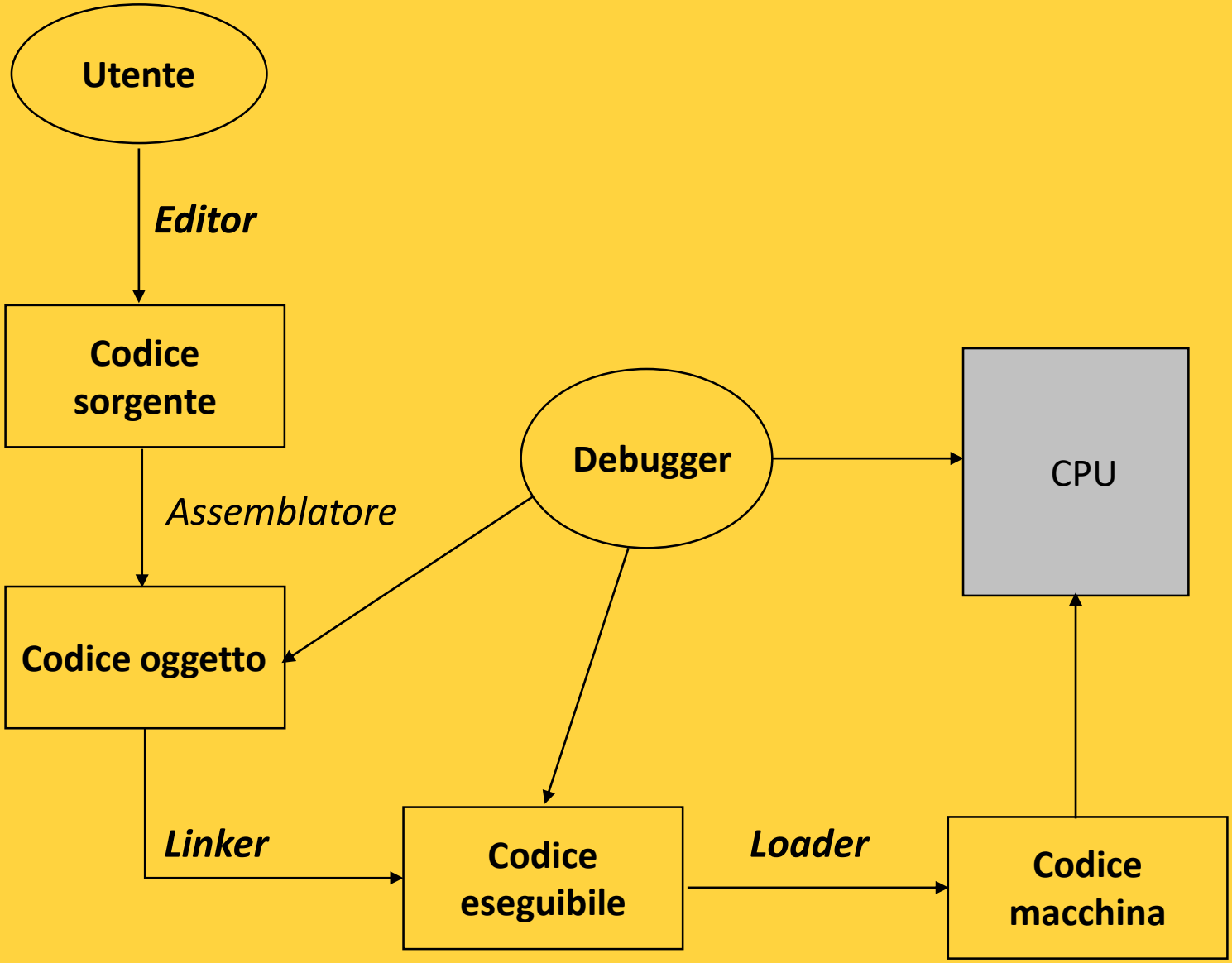
Calcolatori Elettronici
Esercitazioni Assembler

M. Sonza Reorda – M. Monetti
AA 2023/2024
massimo.monetti@polito.it

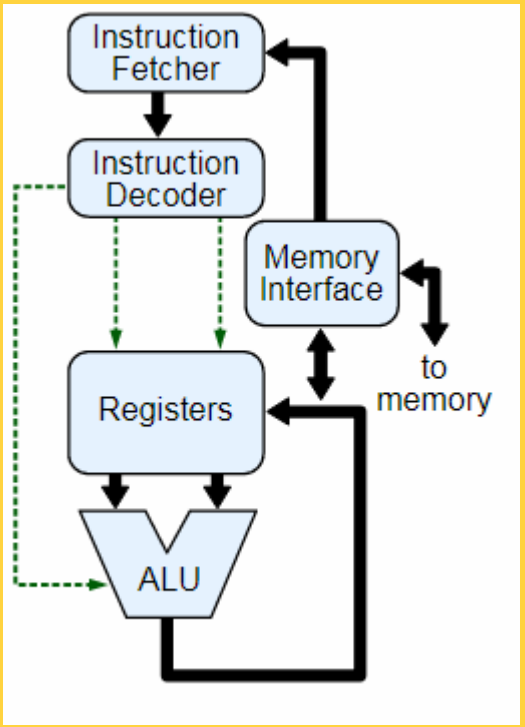
Politecnico di Torino
Dipartimento di Automatica e Informatica

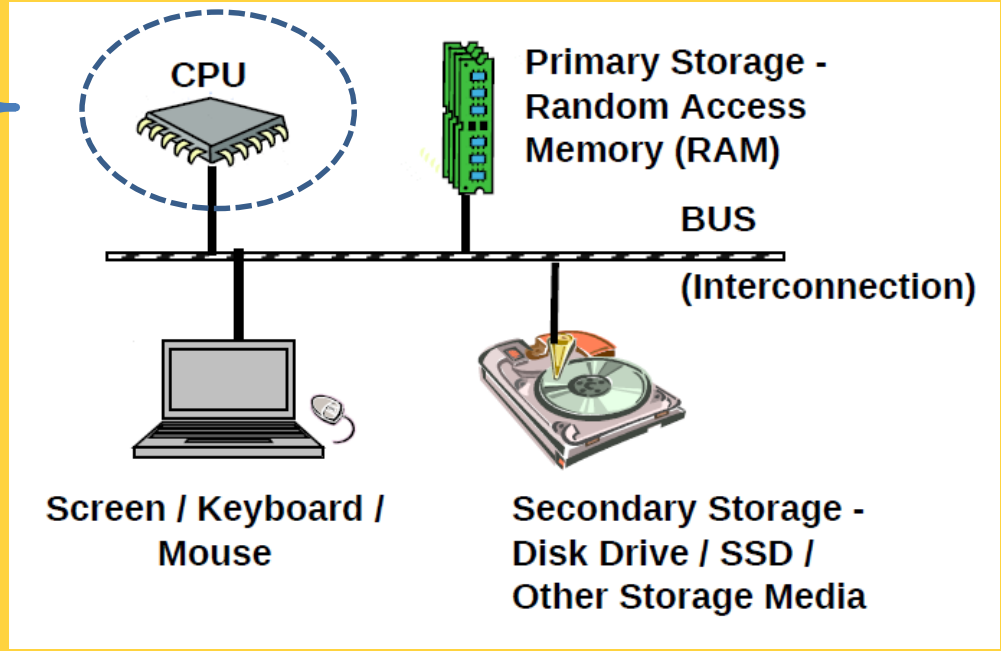
Ciclo di vita di un programma



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

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++

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

                .text
                .globl main
                .ent main
main:           lb $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 procedure

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

- Fine del programma

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 istruzione 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 **mod** Rt (integer division, signed)
- divu Rs, Rt lo = Rs/Rt, hi = Rs **mod** 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 **mod** Rt (signed)

remu Rd, Rs, Rt Rd = Rs **mod** 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 • Rt
- andi Rd, Rs, Imm Rd = Rs • 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
- xori Rd, Rs, Imm Rd = Rs $\oplus$ Imm
- sll Rd, Rt, Sa Rd = Rt left shifted by Sa bits
- sllv Rd, Rs, Rt Rd = Rt left shifted by Rs bits
- srl Rd, Rs, Sa Rd = Rt right shifted by Sa bits
- srlv Rd, Rs, Rt Rd = Rt right shifted by Rs bits

move Rd, Rs Rd = Rs

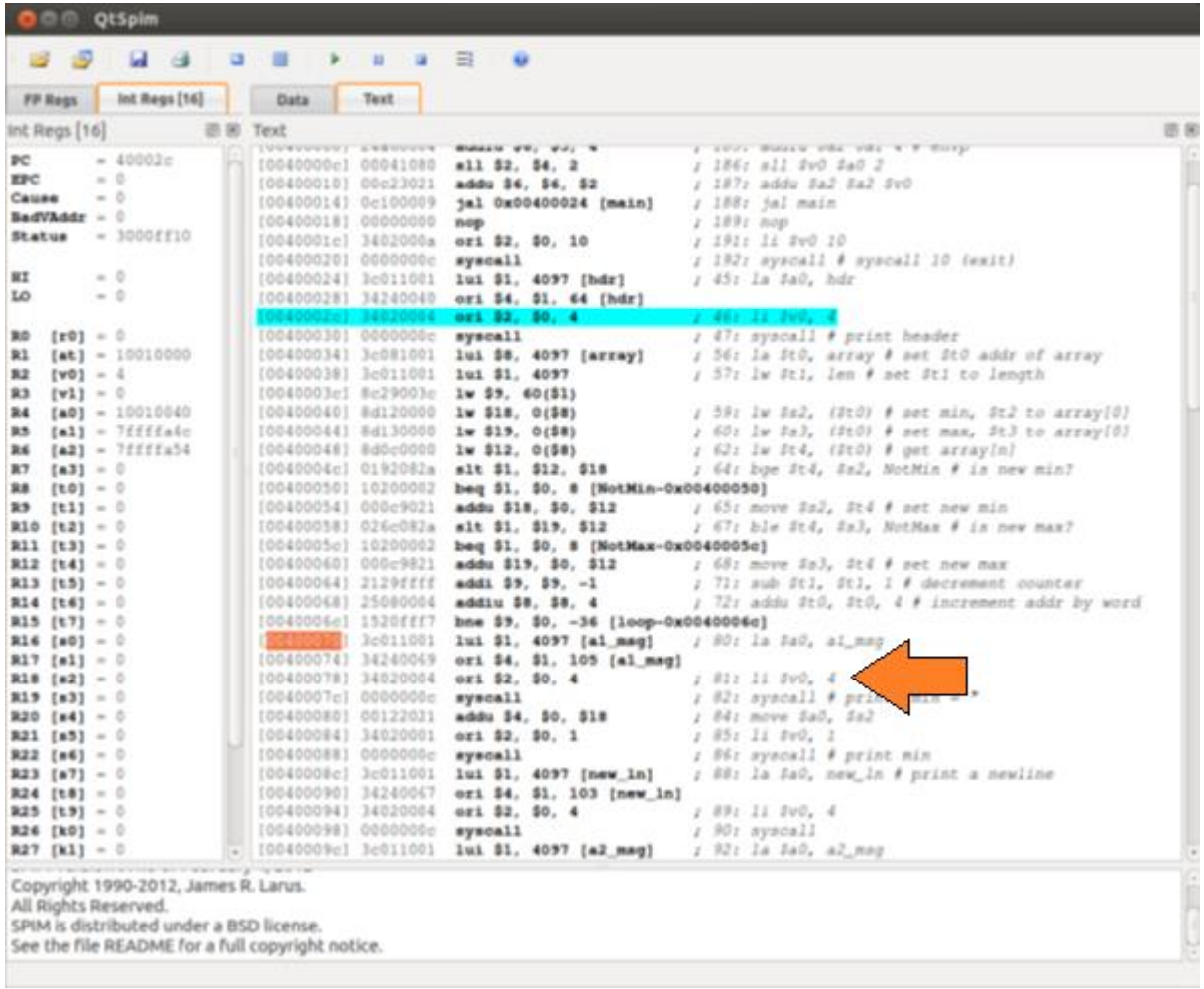
- mfhi Rd Rd = hi
- mflo Rd Rd = lo

li Rd, Imm Rd = Imm

- lui Rt, Imm Rt[31:16] = Imm, Rt[15:0] = 0
- lb Rt, Address(Rs) Rt = byte at M[Address + Rs] (sign extended)
- sb Rt, Address(Rs) Byte at M[Address + Rs] = Rt (sign extended)
- lw Rt, Address(Rs) Rt = word at M[Address + Rs]
- sw Rt, Address(Rs) Word at M[Address + Rs] = Rt

Istruzioni e pseudo-istruzioni

ori $2, $0, 4 ; *46: li $v0, 4*



Istruzioni per iniziare

Load Immediate - LI (*Rdest, imm)*

Load Address - LA (*Rdest, mem)*

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

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

MOVE *Rdest, Rsrc*

BEQ *Rsrc1, Rsrc2, label*

J *label*

ADD *Rdest, Rsrc2, Rsrc1*

SUB *Rdest, Rsrc2, Rsrc1*

Istruzioni per iniziare

Load Immediate **- LI (***Rdest, imm)*

Load Address **- LA (***Rdest, mem)*

Instruction	Description
`li Rdest, imm`	Load specified immediate value into destination register.
`la Rdest, mem`	Load address of memory location into destination register.

```
DIM=5
                .data
bVet:           .space  5
                .text
                .globl main
                .ent main
main:
                la $t0, bVet      # puntatore a inizio del vettore
                li $t1, 0         # contatore
```

Istruzioni per iniziare

Load Word/Byte - (LW,LB) (*Rdest, (address))*

Store Word/Byte - (SW,SB) (*Rsrc, (address)*)

Move - MOVE (*Rdest, Rsrc)*

Instruction	Description
`l<type>  Rdest, mem`	Load value from memory location into destination register.
`s<type>  Rsrc, mem`	Store contents of source register into memory location.

Instruction	Description
`move  Rdest, RSrc`	Copy contents of integer source register into integer destination register.

Istruzioni per iniziare

Load Word/Byte - (LW,LB) (*Rdest, (address))*

Store Word/Byte - (SW,SB) (*Rsrc, (address)*)

Move - (*Rdest, Rsrc)*

```
.data
num:        .word  0
wnum:       .word  42
hnum:       .half  73
bnum:       .byte  7
hans:       .half  0
            .text
            .globl main
            .ent main
main:
            li $t0, 27
            sw $t0, num
            lw $t0, wnum
            lh $t0, hnum
            sh $t0, hans
            move $t1, $t0
```

Istruzioni per iniziare

Branch **- BEQ-BNE-BLT-BLE** *Rsrc1, Rsrc2, label*

Instruction	Description
beq <Rsrc>, <Src>, <label>	Branch to label if <Rscr> and <Scr> are equal
bne <Rsrc>, <Src>, <label>	Branch to label if <Rscr> and <Scr> are not equal
blt <Rsrc>, <Src>, <label>	Branch to label if <Rscr> is less then <Scr>
ble <Rsrc>, <Src>, <label>	Branch to label if <Rscr> is less then or equal to <Scr>

```
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
```

Istruzioni per iniziare

Jump (Branch) **- J (B)** *label*

Instruction	Description
`j <label>`	Unconditionally branch to the specified label.

```
        li  $t1, 0
ciclo1: add $t1, $t1, 1
        add $t0, $t0, 1
        sub $t2, $t1, $t0
        beq $t1, DIM, ciclo2
        j ciclo1
```

Istruzioni per iniziare

Somma (Signed e Unsigned) - ADD *Rdest, Rsrc2, Rsrc1*

Sottrazione (Signed e Unsigned) - SUB *Rdest, Rsrc2, Rsrc1*

Instruction		Description
`add`	`Rdest, Rsrc, Src`	Signed addition Rdest = Rsrc + Src or Imm
`addu`	`Rdest, Rsrc, Src`	Unsigned addition Rdest = Rsrc + Src or Imm
`sub`	`Rdest, Rsrc, Src`	Signed subtraction Rdest = Rsrc – Src or Imm
`subu`	`Rdest, Rsrc, Src`	Unsigned subtraction Rdest = Rsrc – Src or Imm

Queste istruzioni operano su 32-bit, anche se valori di tipo byte o halfword vengono memorizzati nei registri

I valori sono trattati come interi UNSIGNED, non come interi in complemento a 2. Non esiste la gestione OVERFLOW (Trap)

Istruzioni per iniziare

Somma (Signed e Unsigned) - ADD *Rdest, Rsrc2, Rsrc1*

Sottrazione (Signed e Unsigned) - SUB *Rdest, Rsrc2, Rsrc1*

```
        li  $t1, 0
ciclo1: add $t1, $t1, 1
        add $t0, $t0, 1
        sub $t2, $t1, $t0
        bne $t1, DIM, ciclo1      # itera 5 volte
```

Istruzioni di memoria

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

Register *(Rdest)*  **Address**

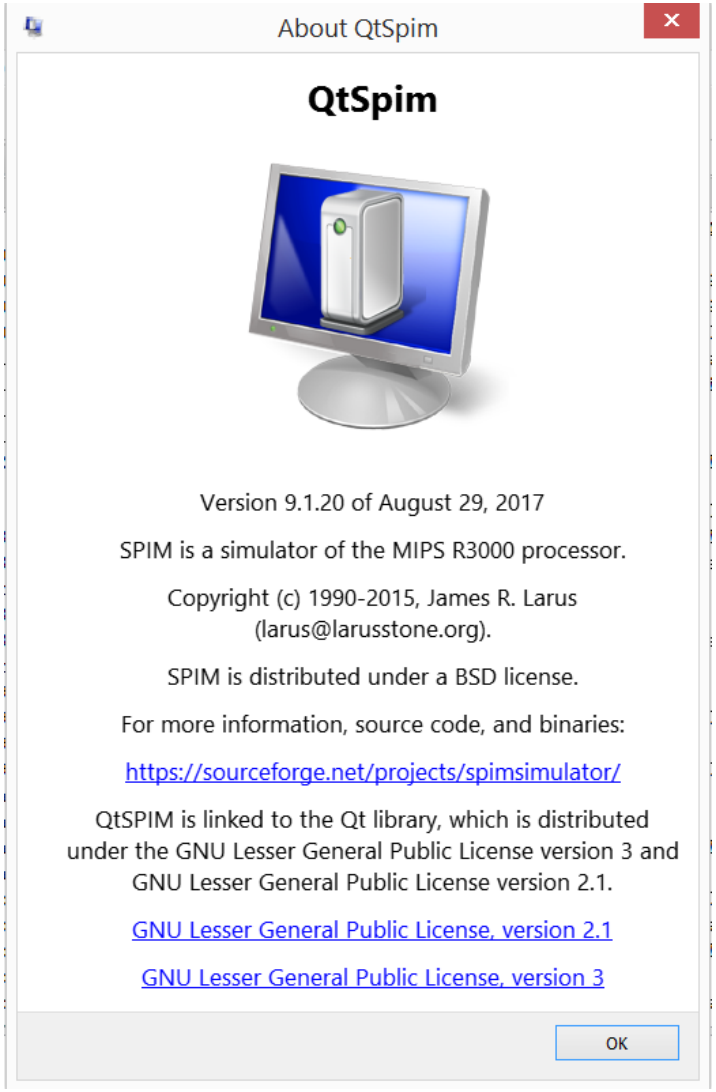
STORE (SW,SB) (*Rsrc, address*)

Register *(Rsrc)*  **Address**

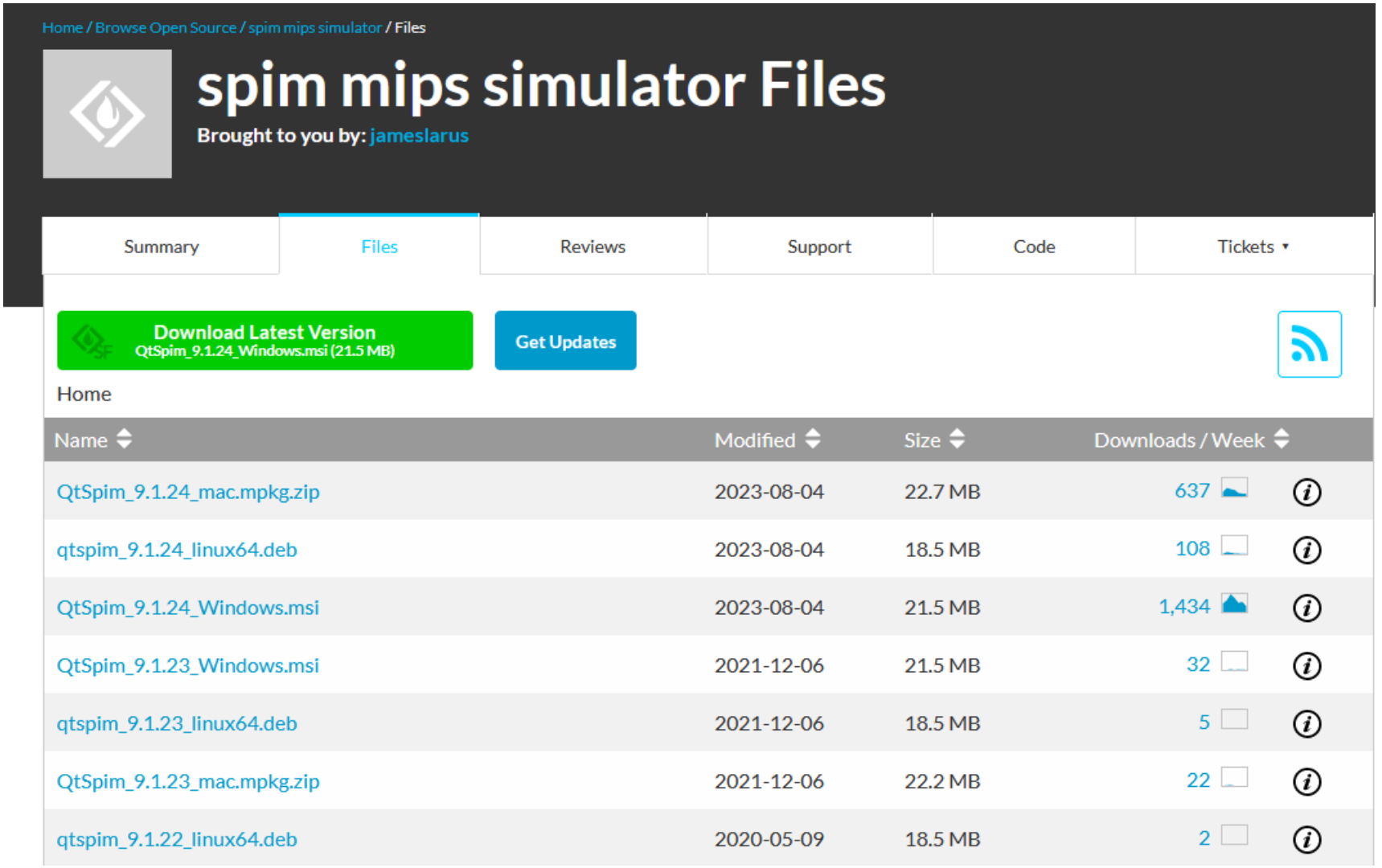
MOVE *Rdest, Rsrc*

Register *(Rdest)*  **Register** *(Rsrc)*

QtSpim


About QtSpim
QtSpim
Version 9.1.20 of August 29, 2017
SPIM is a simulator of the MIPS R3000 processor.
Copyright (c) 1990-2015, James R. Larus
(larus@larusstone.org).
SPIM is distributed under a BSD license.
For more information, source code, and binaries:
https://sourceforge.net/projects/spimsimulator/
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and GNU Lesser General Public License version 2.1.
GNU Lesser General Public License, version 2.1
GNU Lesser General Public License, version 3
OK

Download


Home / Browse Open Source / spim mips simulator / Files
spim mips simulator Files
Brought to you by: jameslarus
Summary
Files
Reviews
Support
Code
Tickets
Download Latest Version
QtSpim_9.1.24_Windows.msi (21.5 MB)
Get Updates
Home
Name
Modified
Size
Downloads / Week
QtSpim_9.1.24_mac.mpkg.zip 2023-08-04 22.7 MB 637
qtspim_9.1.24_linux64.deb 2023-08-04 18.5 MB 108
QtSpim_9.1.24_Windows.msi 2023-08-04 21.5 MB 1,434
QtSpim_9.1.23_Windows.msi 2021-12-06 21.5 MB 32
qtspim_9.1.23_linux64.deb 2021-12-06 18.5 MB 5
QtSpim_9.1.23_mac.mpkg.zip 2021-12-06 22.2 MB 22
qtspim_9.1.22_linux64.deb 2020-05-09 18.5 MB 2

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)

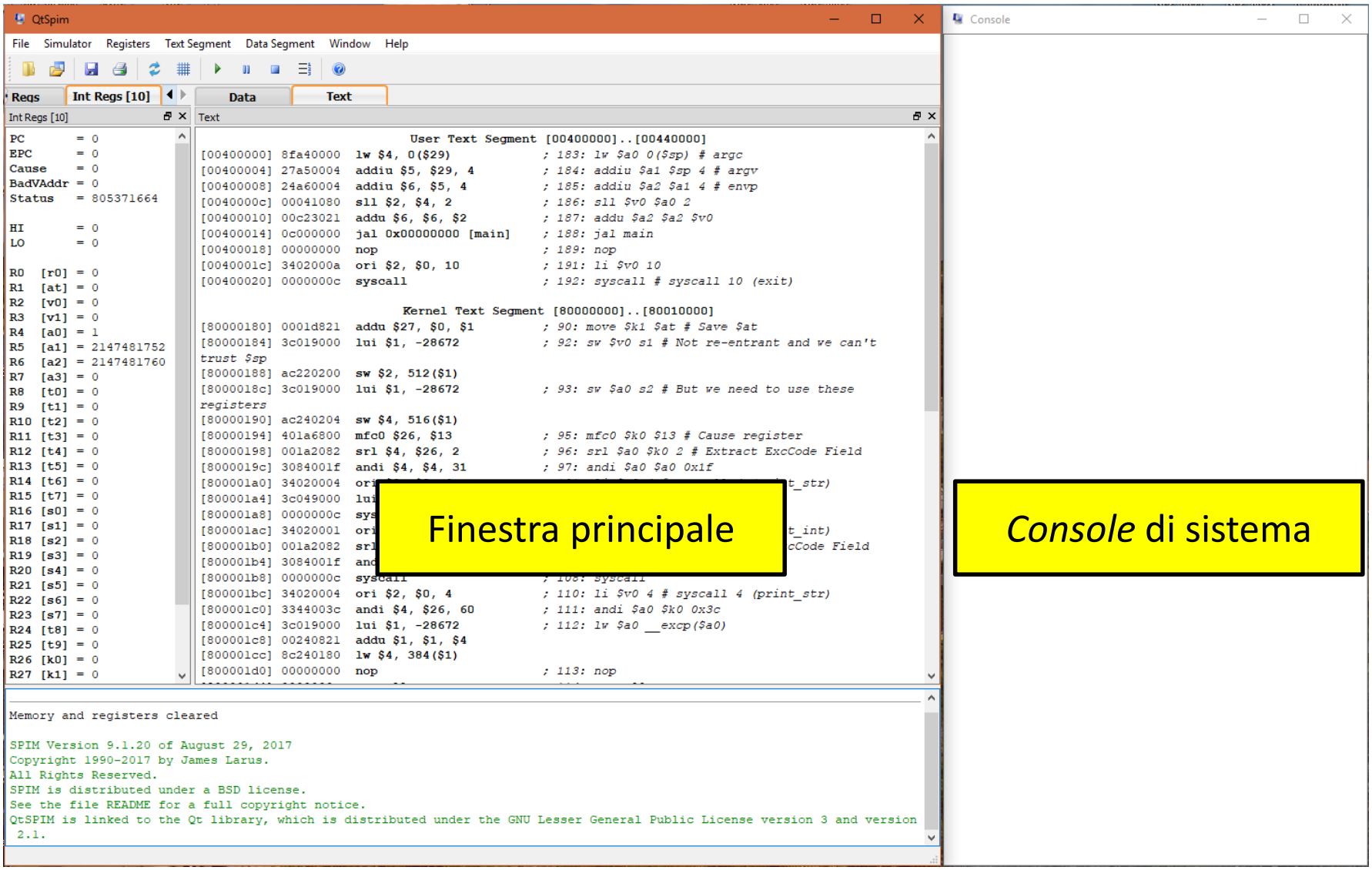
QtSpim

- È compatibile con (quasi) l'intero instruction set MIPS32 (Istruzioni e PseudoIstruzioni)
  - 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

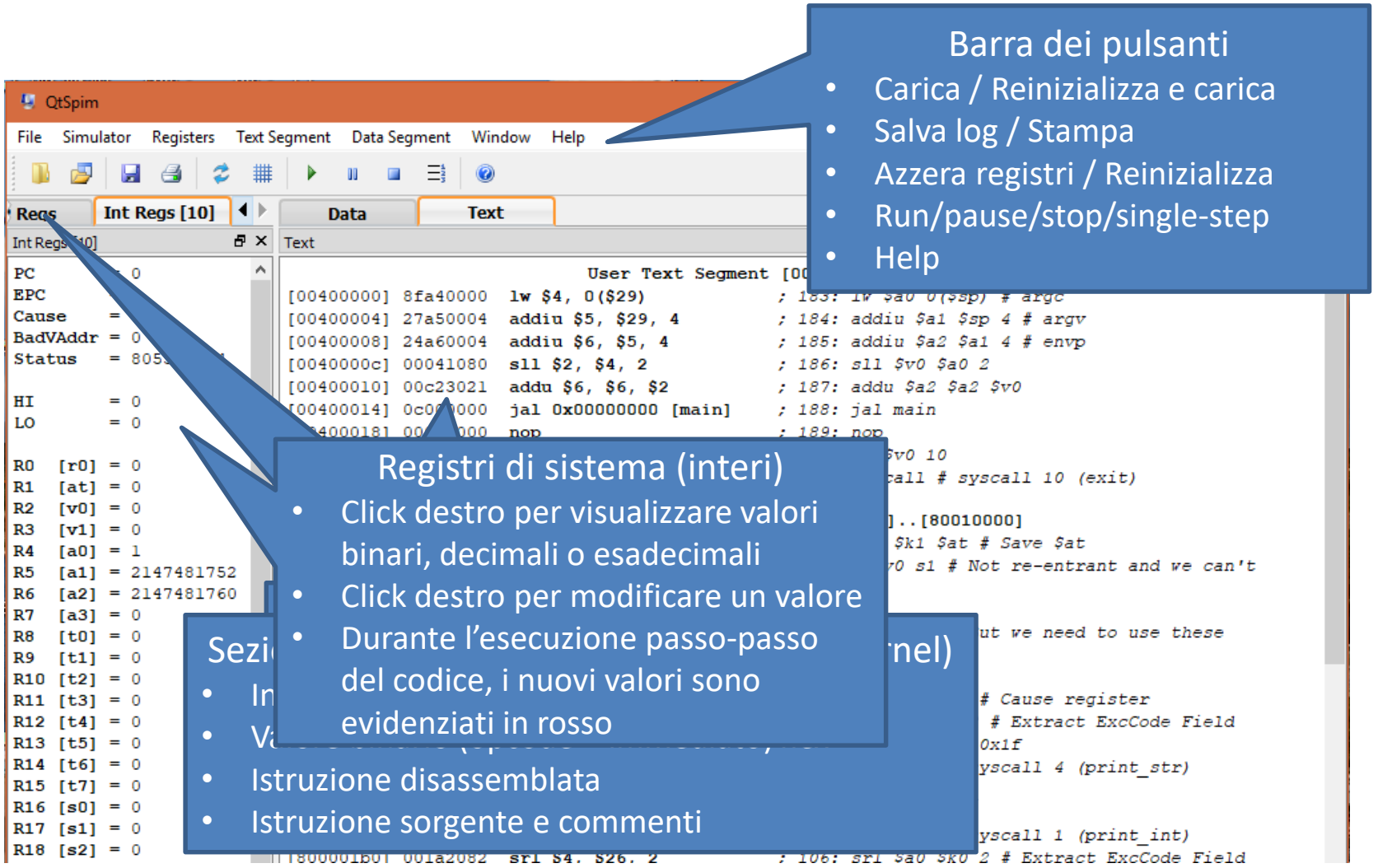
QtSpim

- 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 (***probabile disponibilità 9.1.24***)
- Per qualsiasi problema, è possibile
  - Rivolgersi all'esercitatore o ai borsisti in laboratorio
  - Contattare l'esercitatore via email.

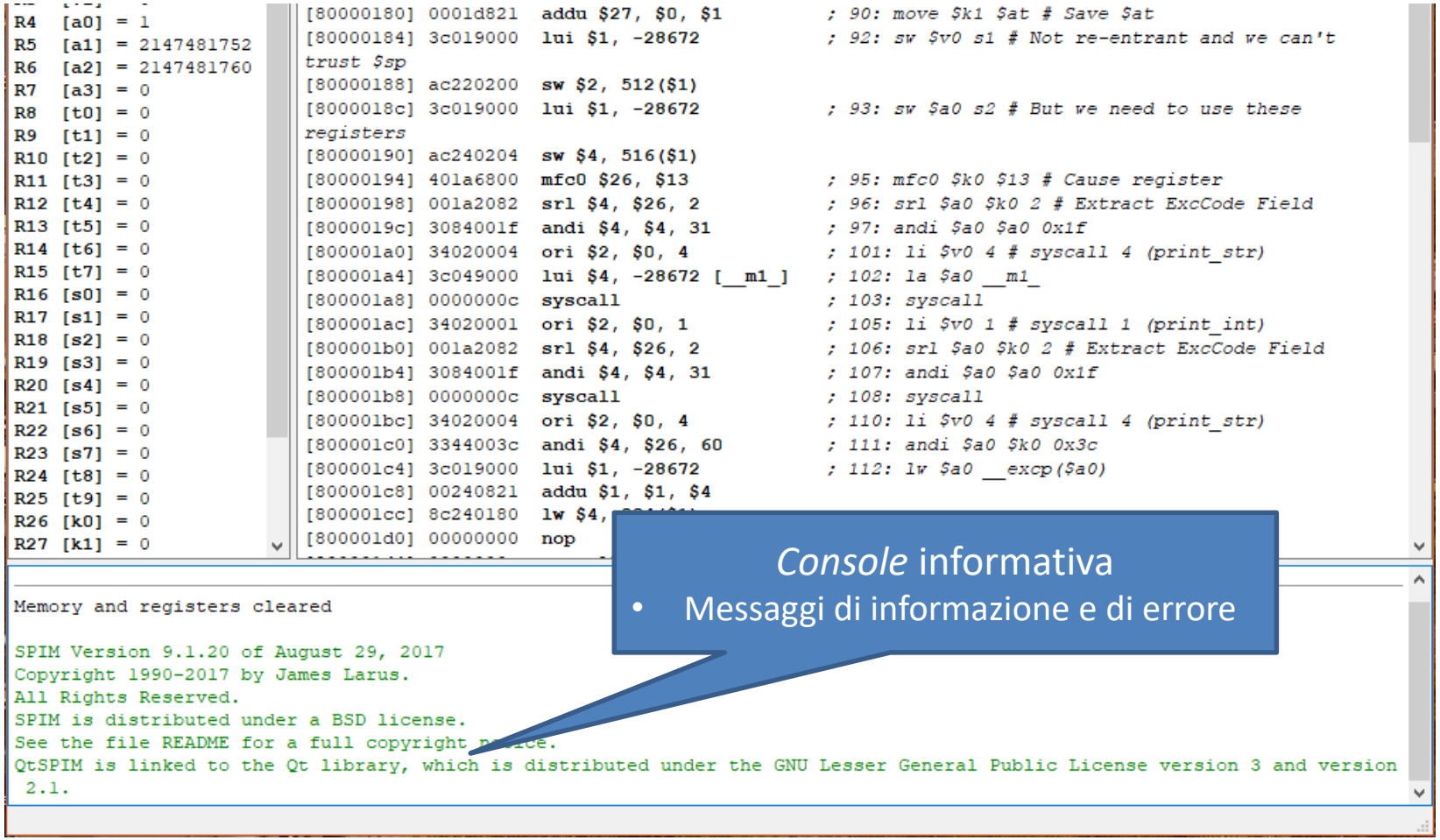
Interfaccia di QtSpim



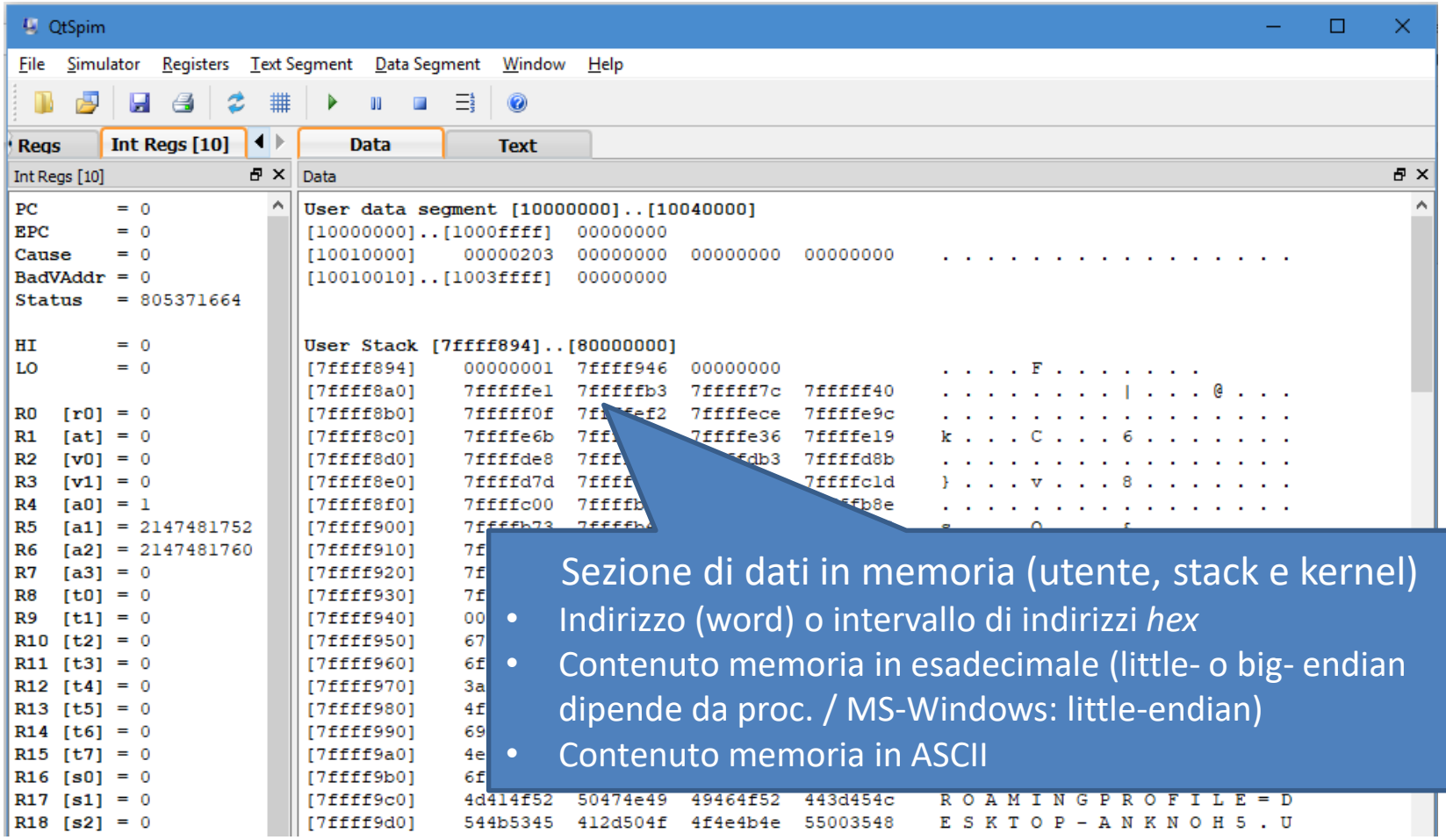
Finestra principale



Finestra principale



Finestra principale



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*”

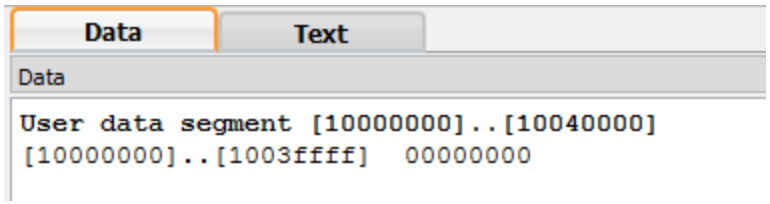
- Ogni volta che il codice viene modificato, è necessario ripartire da “*Reinitialize and Load File*”

Debug [cont.]

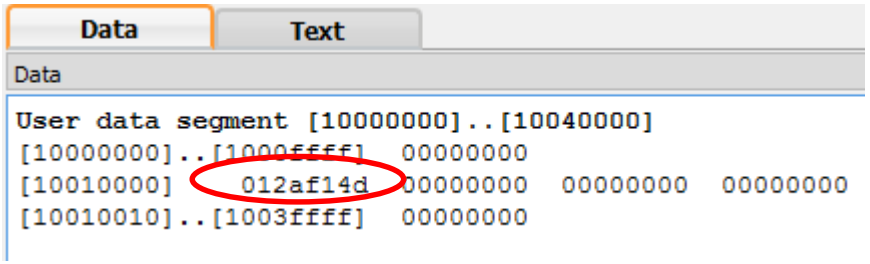
Esempio: esecuzione dell'istruzione di memorizzazione

```
[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:

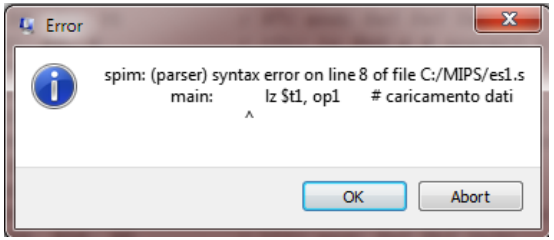


Variabili dopo il salvataggio:



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

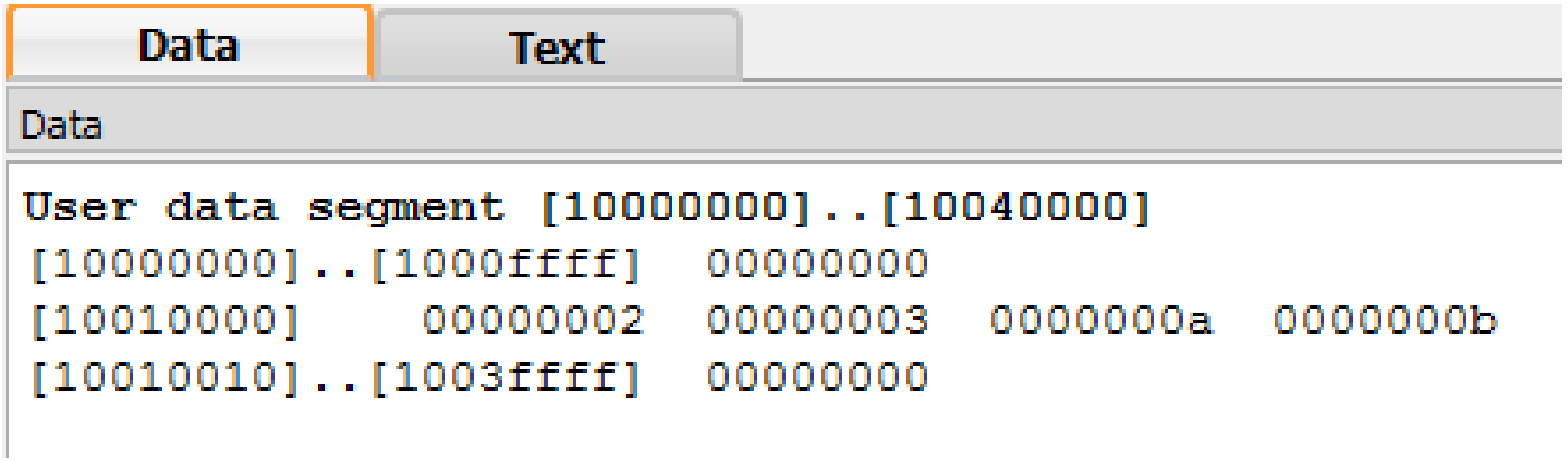


- Quando il codice è correttamente caricato, è possibile agire sugli opportuni pulsanti per eseguirlo 

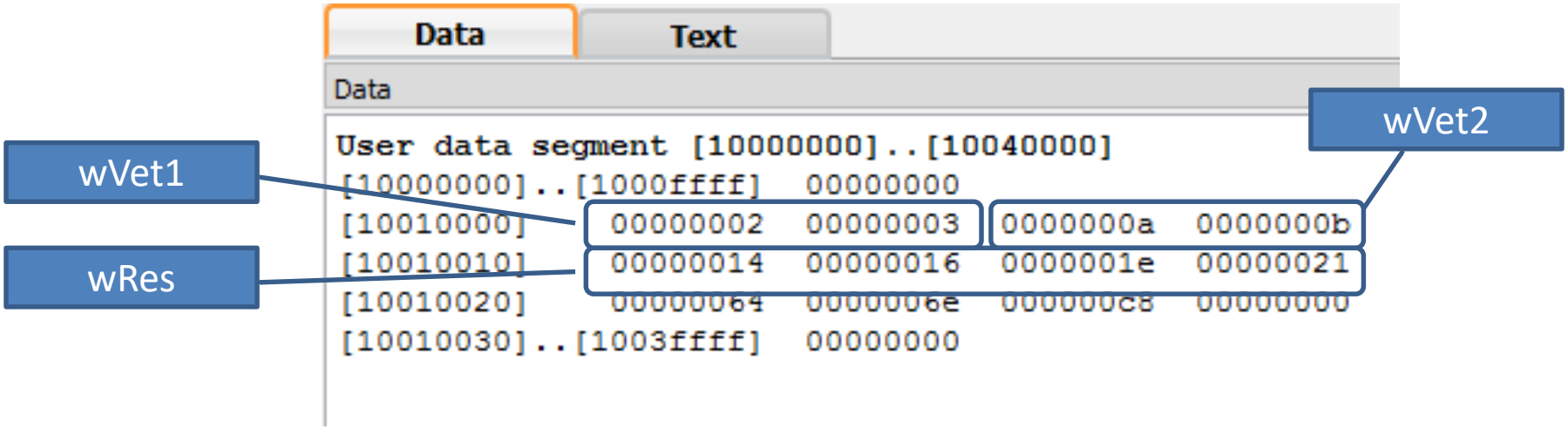
La memoria

Area Dati corrispondente alle seguenti dichiarazioni :

wVet1: .word 2, 3
wVet2: .word 10, 11
wRes: .space 16 (.*space* riserva in memoria **16 bytes**)



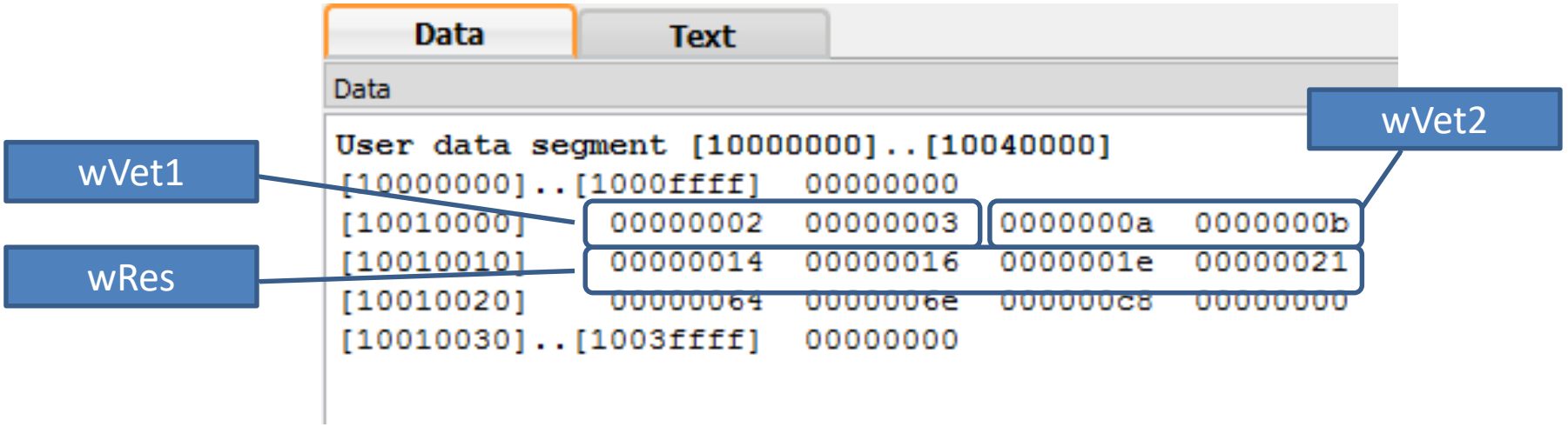
La memoria



LITTLE ENDIAN

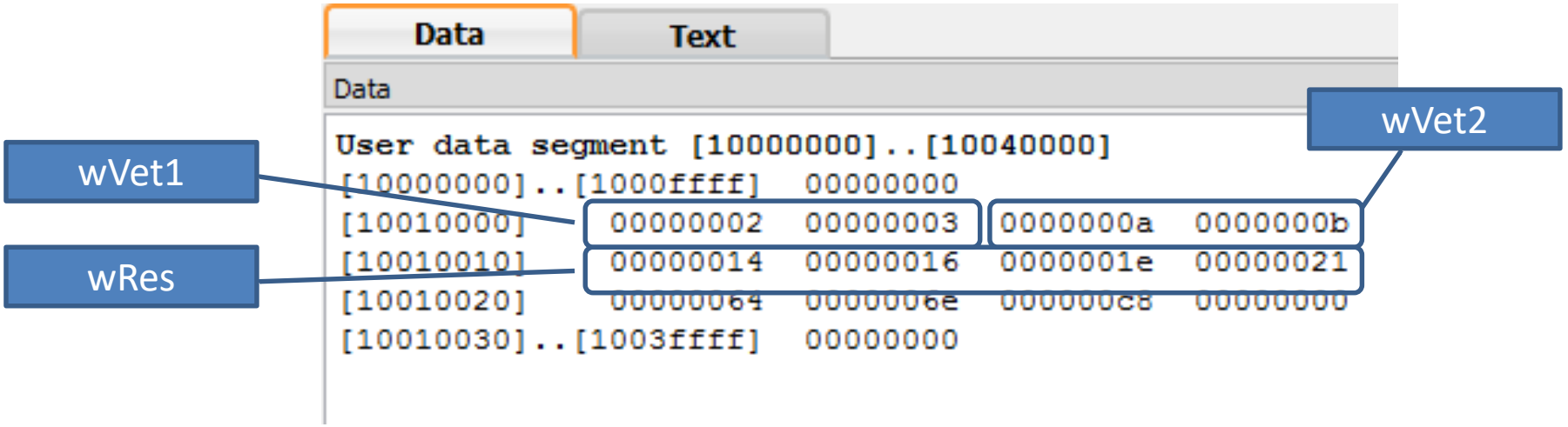
La memorizzazione parte dal byte meno significativo per finire col più significativo, per indirizzo di memoria crescente.

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

SysCall con cui gestire I/O

Service Name	Call Code	Input	Output
Print Integer (32-bit)	1	**$a0** → integer to be printed	
Print Float (32-bit)	2	**$f12** → 32-bit floating-point value to be printed	
Print Double (64-bit)	3	**$f12** → 64-bit floating-point value to be printed	
Print String	4	**$a0** → starting address of NULL terminated string to be printed	
Read Integer (32-bit)	5		**$v0** → 32-bit integer entered by user
Read Float (32-bit)	6		**$f0** → 32-bit floating-point value entered by user

SysCall

Service Name	Call Code	Input	Output
Read Double (64-bit)	7		**$f0** → 64-bit floating-point value entered by user
Read String	8	**$a0** → starting address of buffer (of where to store character entered by user) **$a1** → length of buffer	
Allocate Memory	9	**$a0** → number of bytes to allocate	**$v0** → starting address of allocated memory
Terminate	10		
Print Character	11	**$a0** → character to be printed	
Read Character	12		**$v0** → character entered by user
File Open	13	**$a0** → file name string, NULL terminated **$a1** → access flags **$a2** → file mode, (UNIX style)	**$v0** → file descriptor
File Read	14	**$a0** → file descriptor **$a1** → buffer starting address **$a2** → number of bytes to read	**$v0** → number of bytes actually read from file (-1 = error, 0 = end of file)
File Write	15	**$a0** → file descriptor **$a1** → buffer starting address **$a2** → number of bytes to read	**$v0** → number of bytes actually written to file (-1 = error, 0 = end of file)
File Close	16	**$a0** → file descriptor	

CODICE di ESEMPIO

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
```

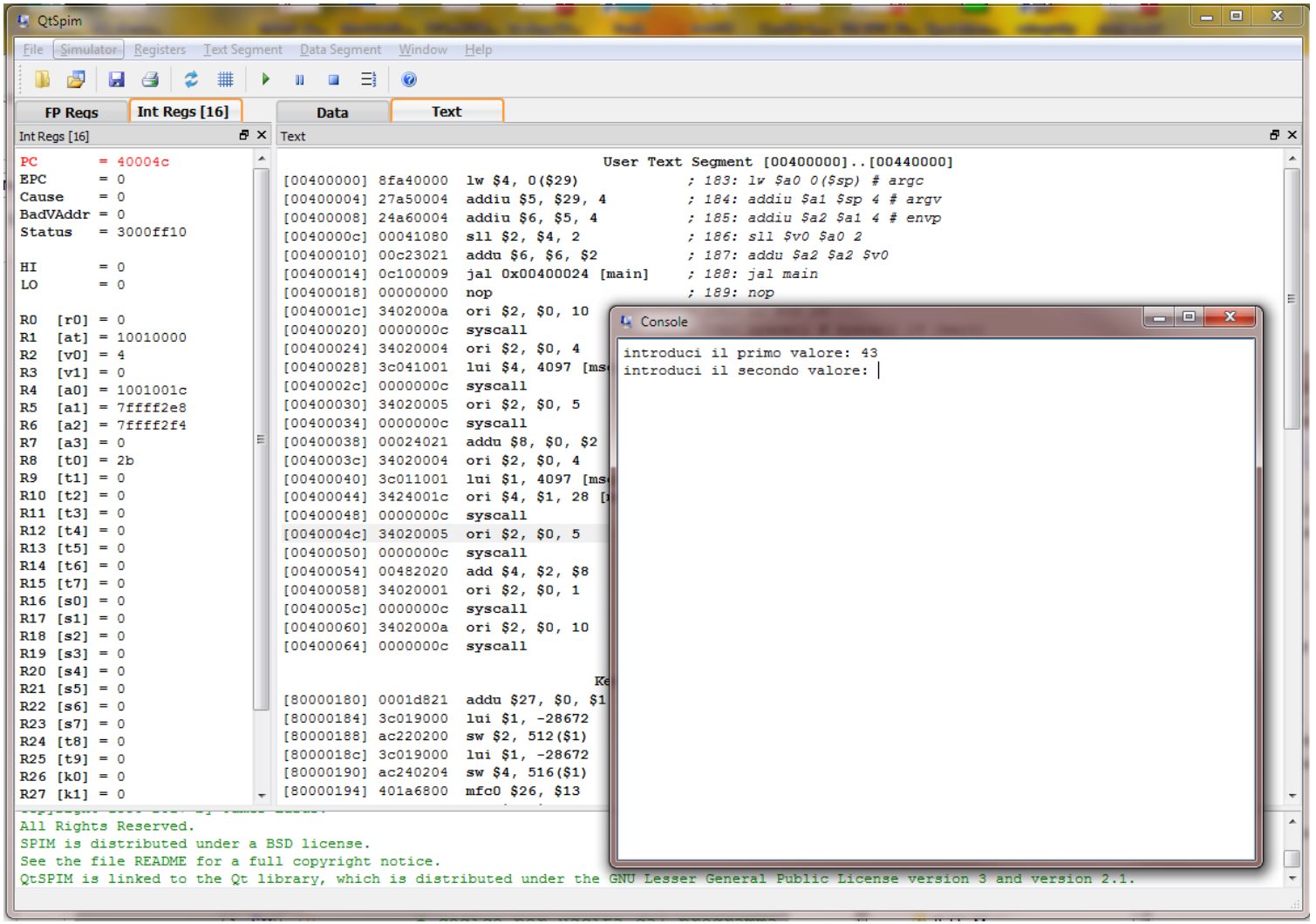
Scrittura di un valore in una cella di memoria

```
                .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
```

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.]



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]

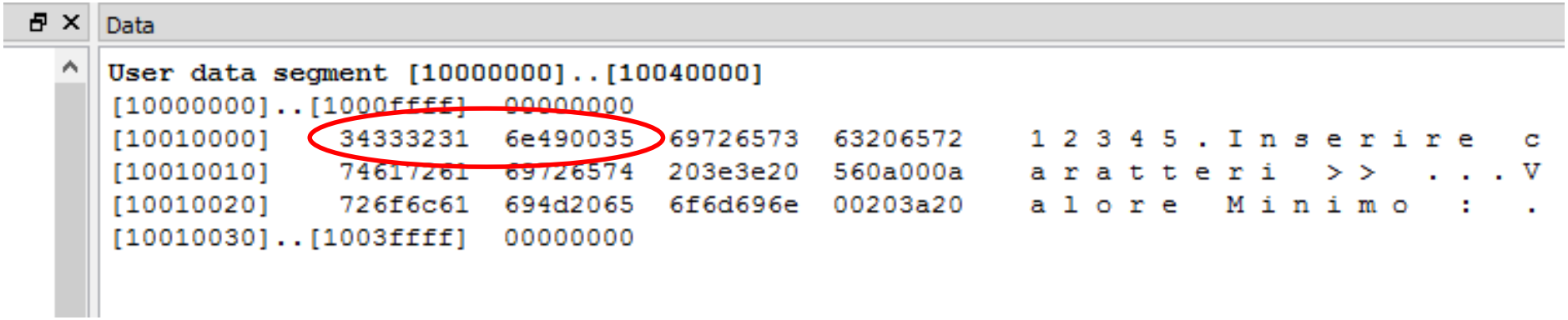
```
ciclo1: li  $v0, 12                  # legge 1 char
        syscall                      # system call (risultato in $v0)
        sb  $v0, ($t0)
        add $t1, $t1, 1
        add $t0, $t0, 1
        bne $t1, DIM, ciclo1         # itera 5 volte

        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
```

Debug

Esempio: esecuzione dell'istruzione di memorizzazione



ASCII Table

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	Ô	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	ı	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	Pts	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	