

# Calcolatori Elettronici

## Esercitazioni Assembler

M. Sonza Reorda – M. Monetti

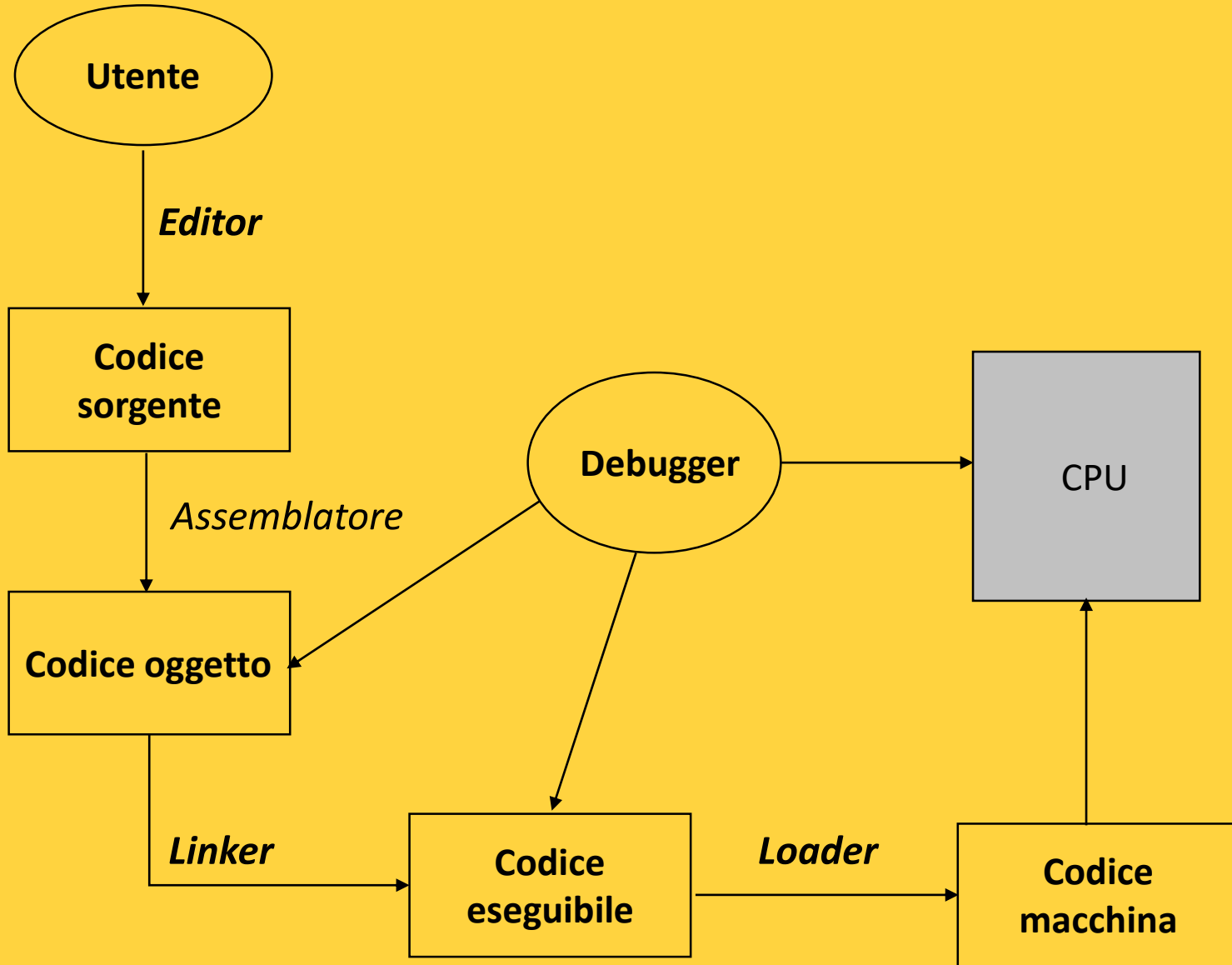
AA 2023/2024

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

Politecnico di Torino

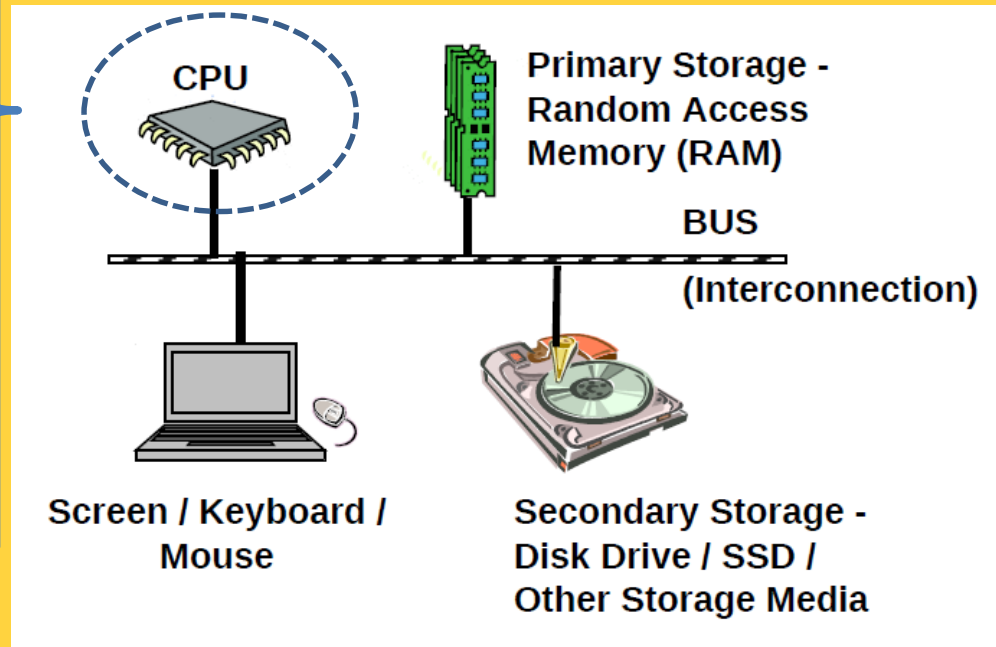
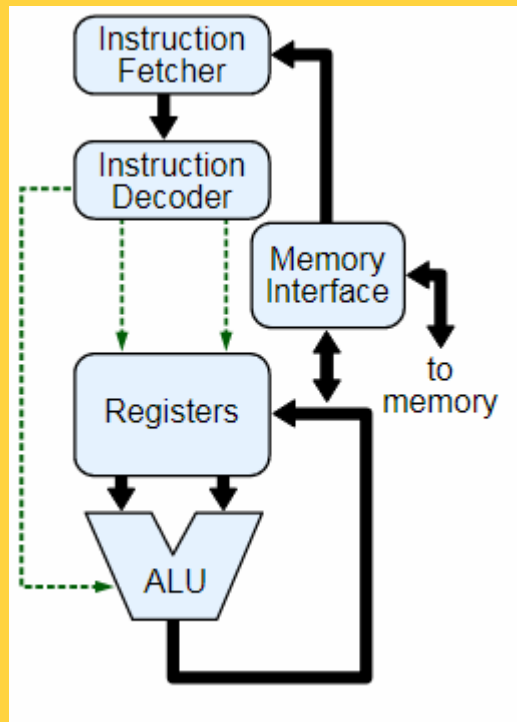
Dipartimento di Automatica e Informatica

# Ciclo di vita di un programma



# Il calcolatore

Schema dal punto di vista del programmatore in linguaggio Assembly



# Architettura MIPS - Registri

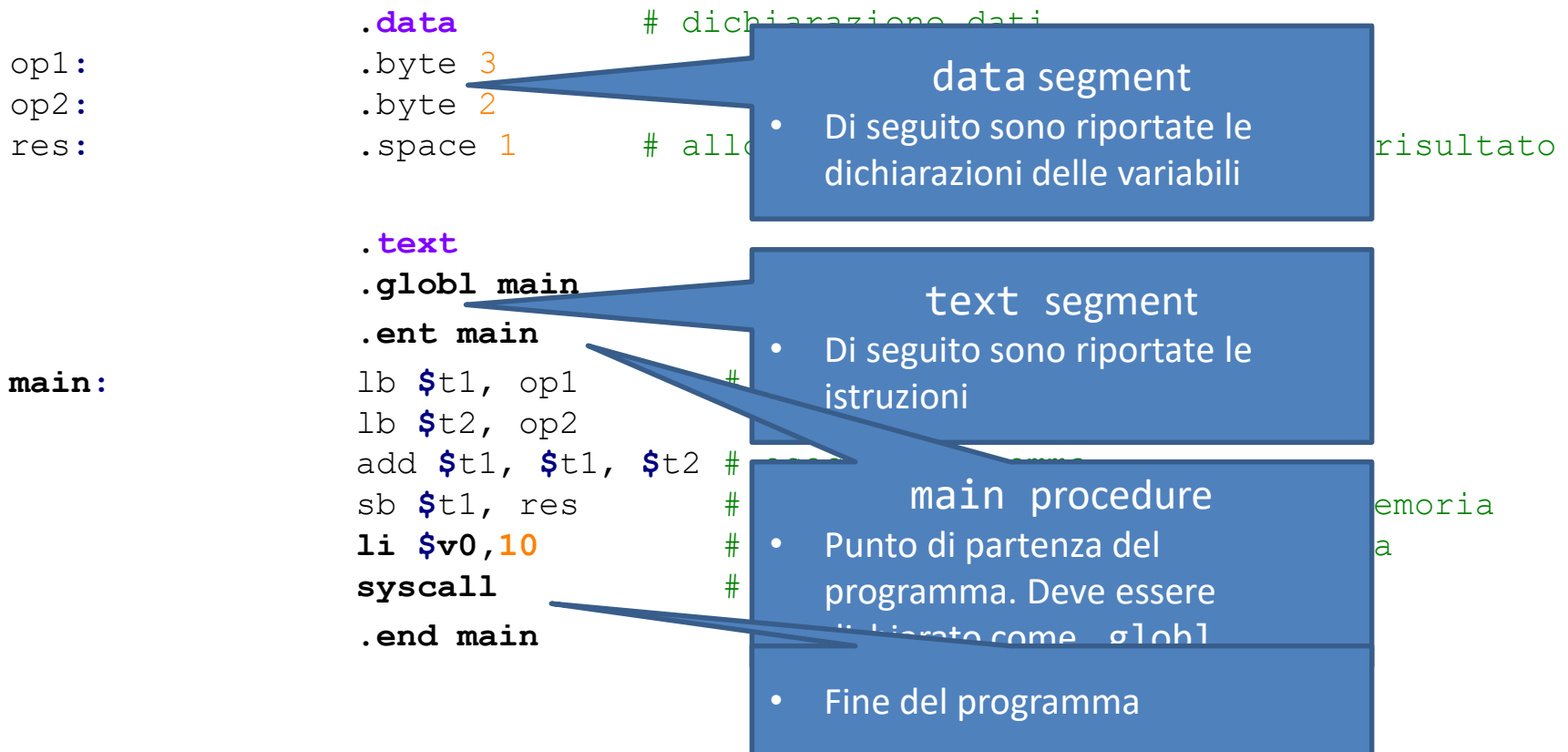
Register Name= \$ + valore alfanumerico

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

# 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++



# 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.

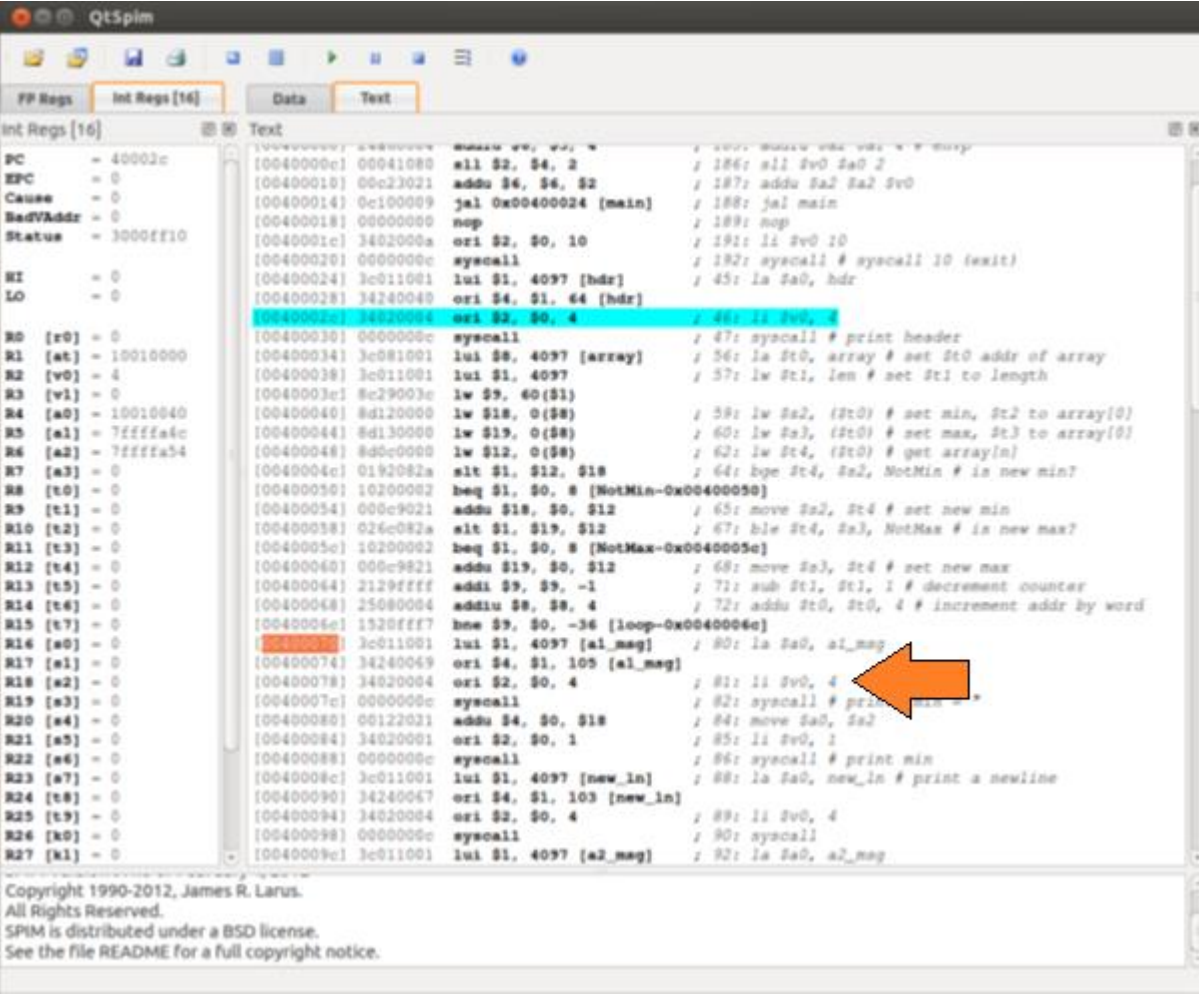
## Instruction Function

- 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  $Rd = Rs \& Rt$
- 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)  $Rt = \text{byte at } M[\text{Address} + Rs]$  (sign extended)
- sb Rt, Address(Rs)  $\text{Byte at } M[\text{Address} + Rs] = Rt$  (sign extended)
- lw Rt, Address(Rs)  $Rt = \text{word at } M[\text{Address} + Rs]$
- sw Rt, Address(Rs)  $\text{Word at } M[\text{Address} + Rs] = Rt$

# Istruzioni e pseudo-istruzioni

ori \$2, \$0, 4

; 46: li \$v0, 4



QtSpim

FP Regs Int Regs [16] Data Text

Int Regs [16]

PC = 40003c  
EPC = 0  
Cause = 0  
BadVAddr = 0  
Status = 3000ff10  
HI = 0  
LO = 0

R0 [r0] = 0  
R1 [at] = 10010000  
R2 [v0] = 4  
R3 [v1] = 0  
R4 [a0] = 10010040  
R5 [a1] = 7ffffa4c  
R6 [a2] = 7ffffa54  
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 [a0] = 0  
R17 [a1] = 0  
R18 [a2] = 0  
R19 [a3] = 0  
R20 [a4] = 0  
R21 [a5] = 0  
R22 [a6] = 0  
R23 [a7] = 0  
R24 [t8] = 0  
R25 [t9] = 0  
R26 [k0] = 0  
R27 [k1] = 0

Text

```
00400000: 00041080 all $2, $4, 2 ; 186: all $v0 $a0 2
00400010: 00c23021 addu $4, $4, $2 ; 187: addu $a2 $a2 $v0
00400014: 0c100009 jal 0x00400024 [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)
00400024: 3c011001 lui $1, 4097 [hdr] ; 45: la $a0, hdr
00400028: 34240040 ori $4, $1, 64 [hdr]
0040002e: 00000004 ori $2, $0, 4 ; 46: li $v0, 4
00400030: 0000000c syscall ; 47: syscall # print header
00400034: 3c081001 lui $8, 4097 [array] ; 54: la $t0, array # set $t0 addr of array
00400038: 3c011001 lui $1, 4097 ; 57: lw $t1, len # set $t1 to length
0040003c: 8c29003c lw $9, 40($1)
00400040: 8d120000 lw $18, 0($8) ; 59: lw $a2, ($t0) # set min, $t2 to array[0]
00400044: 8d130000 lw $19, 0($8) ; 60: lw $a3, ($t0) # set max, $t3 to array[0]
00400048: 8d0c0000 lw $12, 0($8) ; 62: lw $t4, ($t0) # get array[n]
0040004c: 0192082a slt $1, $12, $18 ; 64: bge $t4, $a2, NotMin # is new min?
00400050: 10200002 beq $1, $0, 8 [NotMin-0x00400050]
00400054: 00c90211 addu $18, $0, $12 ; 65: move $a2, $t4 # set new min
00400058: 026c082a slt $1, $19, $12 ; 67: ble $t4, $a3, NotMax # is new max?
0040005c: 10200002 beq $1, $0, 8 [NotMax-0x0040005c]
00400060: 00c98211 addu $19, $0, $12 ; 68: move $a3, $t4 # set new max
00400064: 2129ffff addi $9, $9, -1 ; 71: sub $t1, $t1, 1 # decrement counter
00400068: 25080004 addiu $8, $8, 4 ; 72: addu $t0, $t0, 4 # increment addr by word
0040006c: 1528fff7 bne $9, $0, -36 [loop-0x0040006c]
00400070: 3c011001 lui $1, 4097 [a1_msg] ; 80: la $a0, a1_msg
00400074: 34240049 ori $4, $1, 105 [a1_msg]
00400078: 34020004 ori $2, $0, 4 ; 81: li $v0, 4
0040007c: 0000000c syscall ; 82: syscall # print string "a1 = "
00400080: 00122021 addu $4, $0, $18 ; 84: move $a0, $a2
00400084: 34020001 ori $2, $0, 1 ; 85: li $v0, 1
00400088: 0000000c syscall ; 86: syscall # print min
0040008c: 3c011001 lui $1, 4097 [new_ln] ; 88: la $a0, new_ln # print a newline
00400090: 34240047 ori $4, $1, 103 [new_ln]
00400094: 34020004 ori $2, $0, 4 ; 89: li $v0, 4
00400098: 0000000c syscall ; 90: syscall
0040009c: 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 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
<b>li</b> <i>Rdest, imm</i>	Load specified immediate value into destination register.
<b>la</b> <i>Rdest, mem</i>	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
<b>l&lt;type&gt;</b> <i>Rdest, mem</i>	Load value from memory location into destination register.
<b>s&lt;type&gt;</b> <i>Rsrc, mem</i>	Store contents of source register into memory location.

Instruction	Description
<b>move</b> <i>Rdest, RSrc</i>	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
<b>beq</b> <Rsrc>, <Src>, <label>	Branch to label if <Rscr> and <Scr> are equal
<b>bne</b> <Rsrc>, <Src>, <label>	Branch to label if <Rscr> and <Scr> are not equal
<b>blt</b> <Rsrc>, <Src>, <label>	Branch to label if <Rscr> is less than <Scr>
<b>ble</b> <Rsrc>, <Src>, <label>	Branch to label if <Rscr> is less than 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
<code>j &lt;label&gt;</code>	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
<b>add</b> <i>Rdest, Rsrc, Src</i>	Signed addition $Rdest = Rsrc + Src \text{ or } Imm$
<b>addu</b> <i>Rdest, Rsrc, Src</i>	Unsigned addition $Rdest = Rsrc + Src \text{ or } Imm$
<b>sub</b> <i>Rdest, Rsrc, Src</i>	Signed subtraction $Rdest = Rsrc - Src \text{ or } Imm$
<b>subu</b> <i>Rdest, Rsrc, Src</i>	Unsigned subtraction $Rdest = Rsrc - Src \text{ or } Imm$

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*)



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

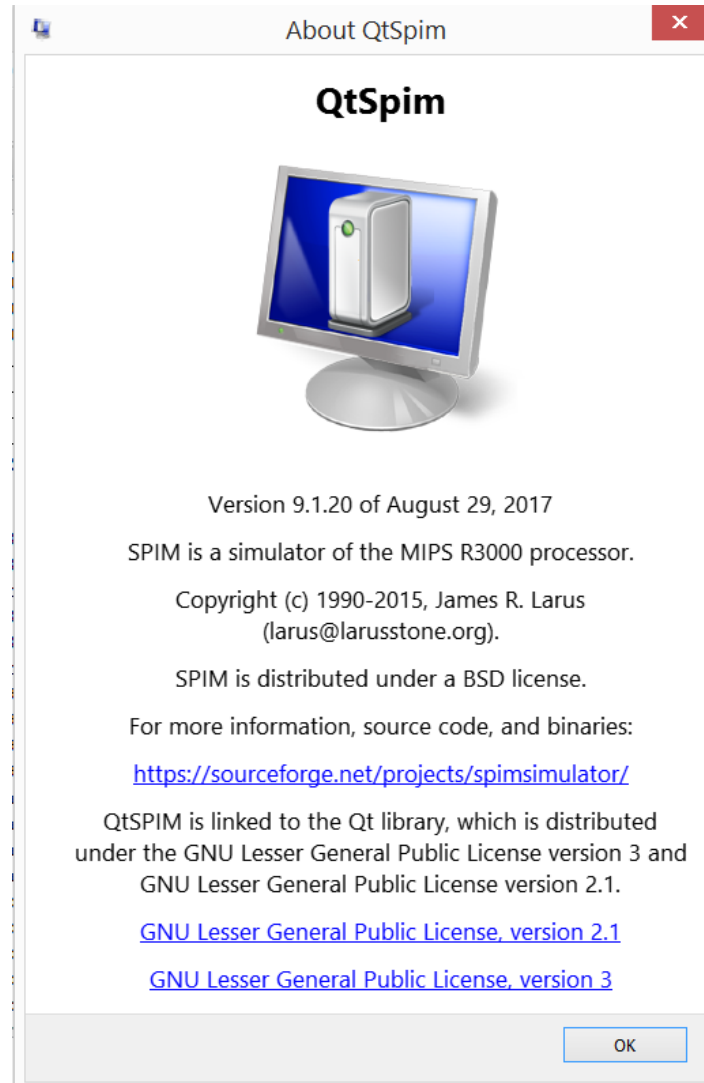


**MOVE** *Rdest*, *Rsrc*





# QtSpim



# 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 ↕
<a href="#">QtSpim_9.1.24_mac.mpkg.zip</a>	2023-08-04	22.7 MB	637  
<a href="#">qtspim_9.1.24_linux64.deb</a>	2023-08-04	18.5 MB	108  
<a href="#">QtSpim_9.1.24_Windows.msi</a>	2023-08-04	21.5 MB	1,434  
<a href="#">QtSpim_9.1.23_Windows.msi</a>	2021-12-06	21.5 MB	32  
<a href="#">qtspim_9.1.23_linux64.deb</a>	2021-12-06	18.5 MB	5  
<a href="#">QtSpim_9.1.23_mac.mpkg.zip</a>	2021-12-06	22.2 MB	22  
<a href="#">qtspim_9.1.22_linux64.deb</a>	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

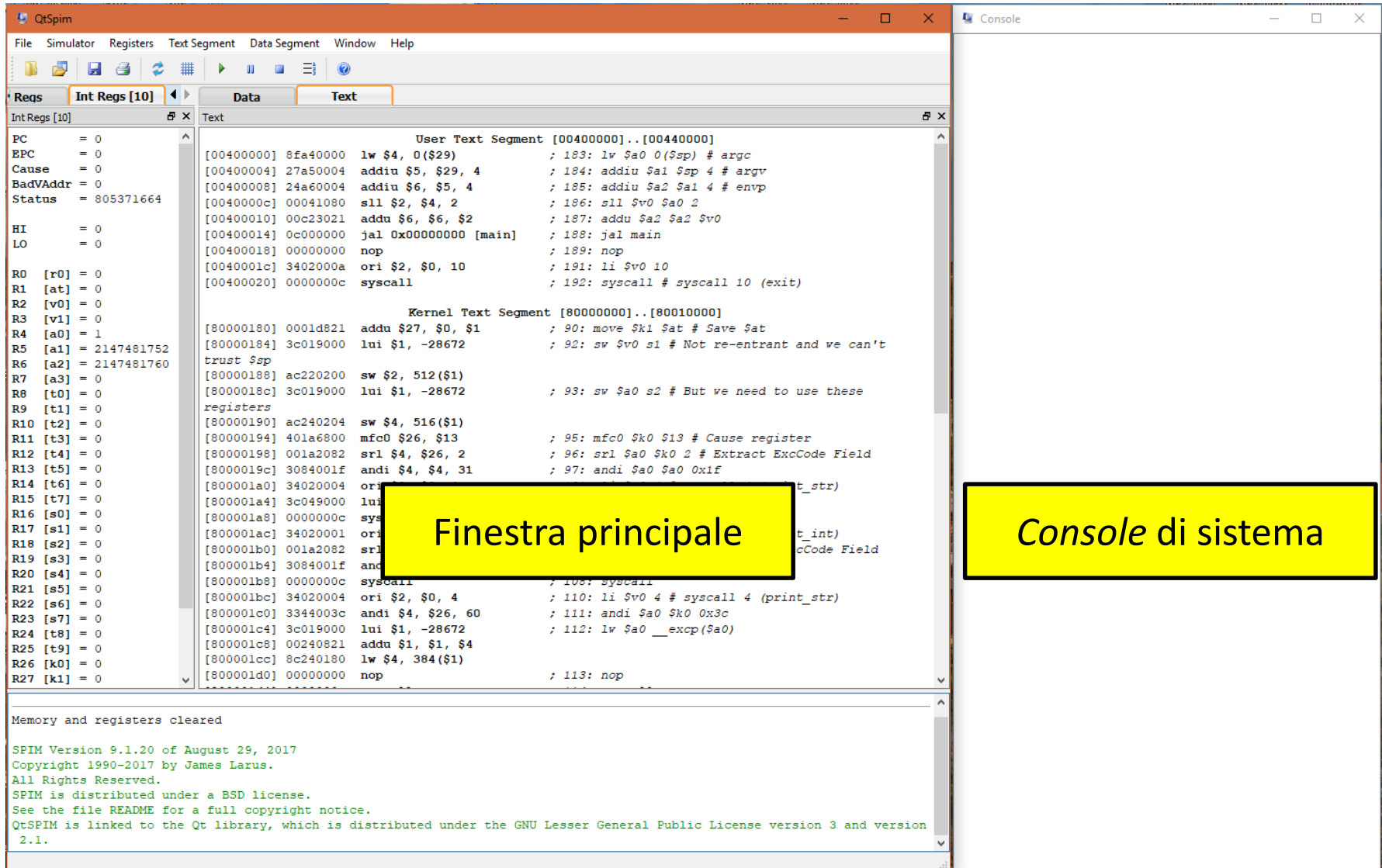
- È compatibile con (quasi) l'intero instruction set MIPS32 (Istruzioni e Pseudolstruzioni)
  - Non include confronti e arrotondamenti floating point
  - Non include la gestione delle tabelle di pagina della memoria
- È gratuito e open-source, e sono disponibili versioni per MS-Windows, Mac OS X e Linux
- 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](#) (***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

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 three panes: 'Regs' (Registers), 'Data', and 'Text'. The 'Regs' pane on the left shows system registers (PC, EPC, Cause, BadVAddr, Status, HI, LO) and general-purpose registers (R0-R18) with their current values. The 'Text' pane on the right displays the assembly code of the 'User Text Segment'. A blue callout box points to the toolbar, listing the functions of the buttons. Another blue callout box points to the 'Regs' pane, explaining the functionality of the register list. A third blue callout box points to the 'Text' pane, detailing the information shown for each instruction.

**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 Registri**

- In...
- V...
- Istruzione disassemblata
- Istruzione sorgente e commenti

**Sezione Testo**

- 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
R27 [k1] = 0     [800001d0] 00000000 nop
```

## 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 7ffffef2 7ffffdb3 7ffffd8b . . . . .  
[7ffff8e0] 7ffffd7d 7ffffef2 7ffffc1d 7ffffc1d } . . . v . . . 8 . . . . .  
[7ffff8f0] 7ffffc00 7ffffef2 7ffffb8e . . . . .  
[7ffff900] 7ffffb73 7ffffef2 . . . . .  
[7ffff910] 7ffffb73 7ffffef2 . . . . .  
[7ffff920] 7ffffb73 7ffffef2 . . . . .  
[7ffff930] 7ffffb73 7ffffef2 . . . . .  
[7ffff940] 00000000 7ffffef2 . . . . .  
[7ffff950] 677ffffef2 7ffffef2 . . . . .  
[7ffff960] 6f7ffffef2 7ffffef2 . . . . .  
[7ffff970] 3a7ffffef2 7ffffef2 . . . . .  
[7ffff980] 4f7ffffef2 7ffffef2 . . . . .  
[7ffff990] 697ffffef2 7ffffef2 . . . . .  
[7ffff9a0] 4e7ffffef2 7ffffef2 . . . . .  
[7ffff9b0] 6f7ffffef2 7ffffef2 . . . . .  
[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

# Debug

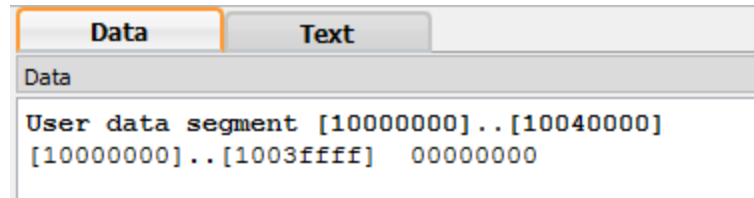
- L'esecuzione passo-passo è fondamentale per il *debug*
  - Osservare il valore di memoria e registri al termine di ogni istruzione
- È possibile inserire un *breakpoint* facendo click con il pulsante destro sull'istruzione desiderata nella sezione *Text* della finestra principale, e selezionando "*Set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

# Debug [cont.]

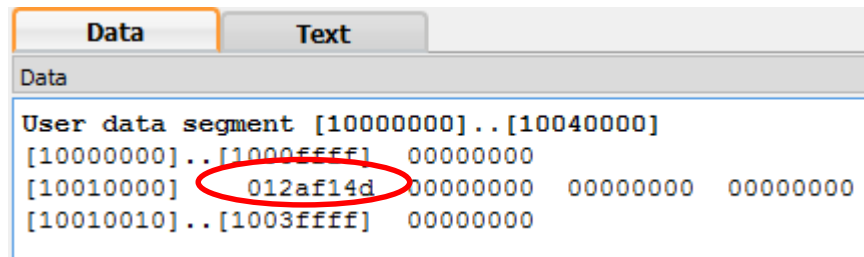
Esempio: esecuzione dell'istruzione di memorizzazione

```
[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:



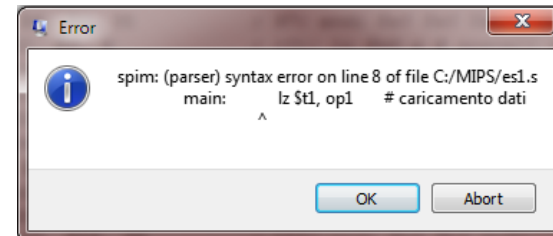
# Caricamento del codice

- 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 :

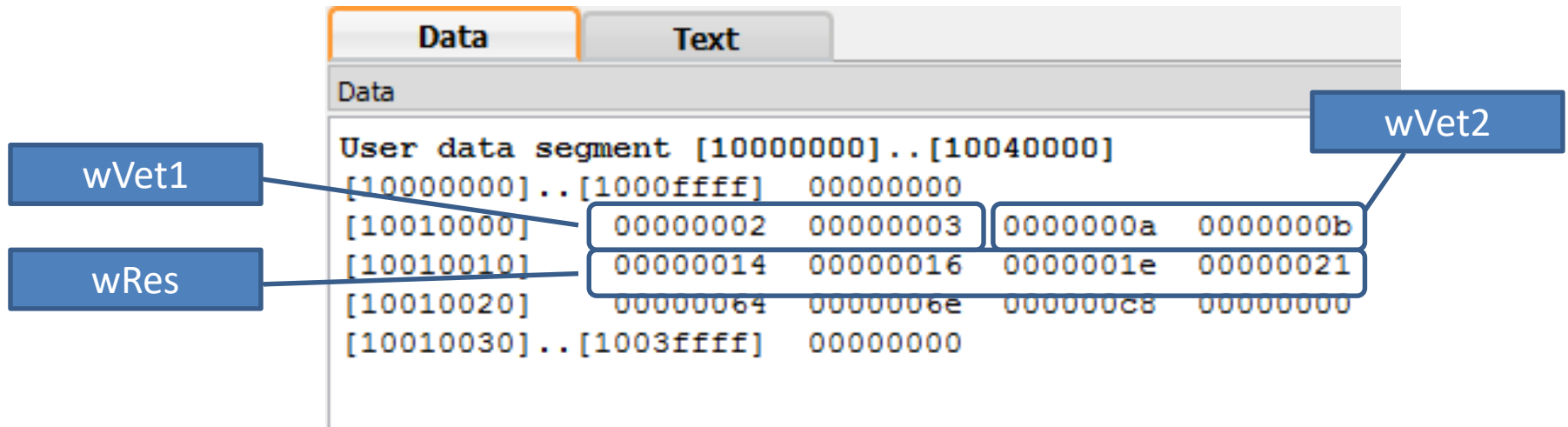
**wVet1:** .word 2, 3

**wVet2:** .word 10, 11

**wRes:** .space 16 (*.space* riserva in memoria **16 bytes**)

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

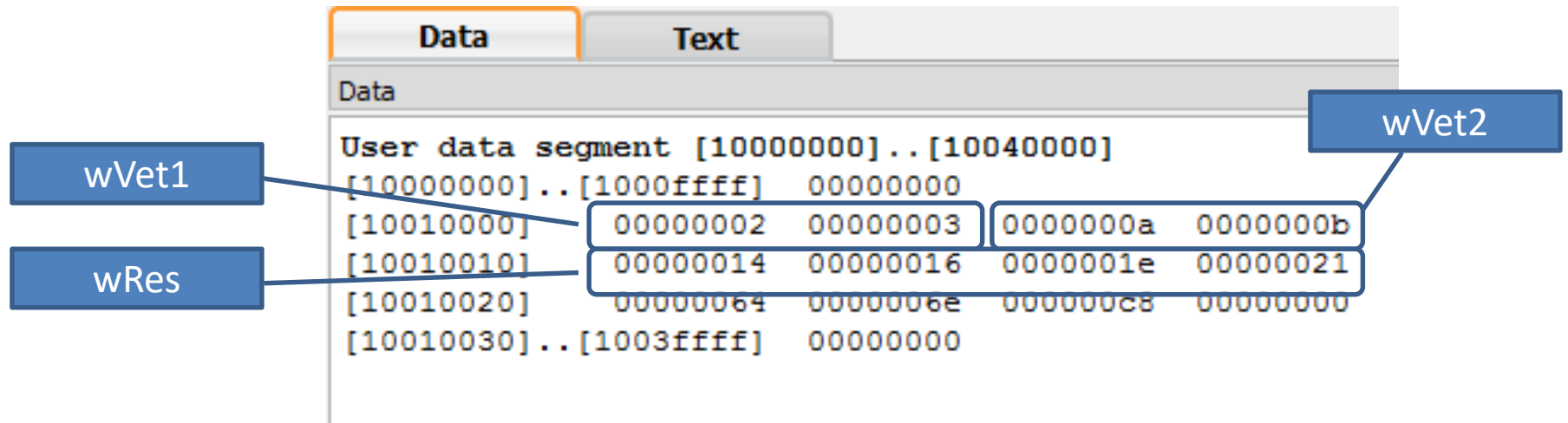
# La memoria



## LITTLE ENDIAN

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

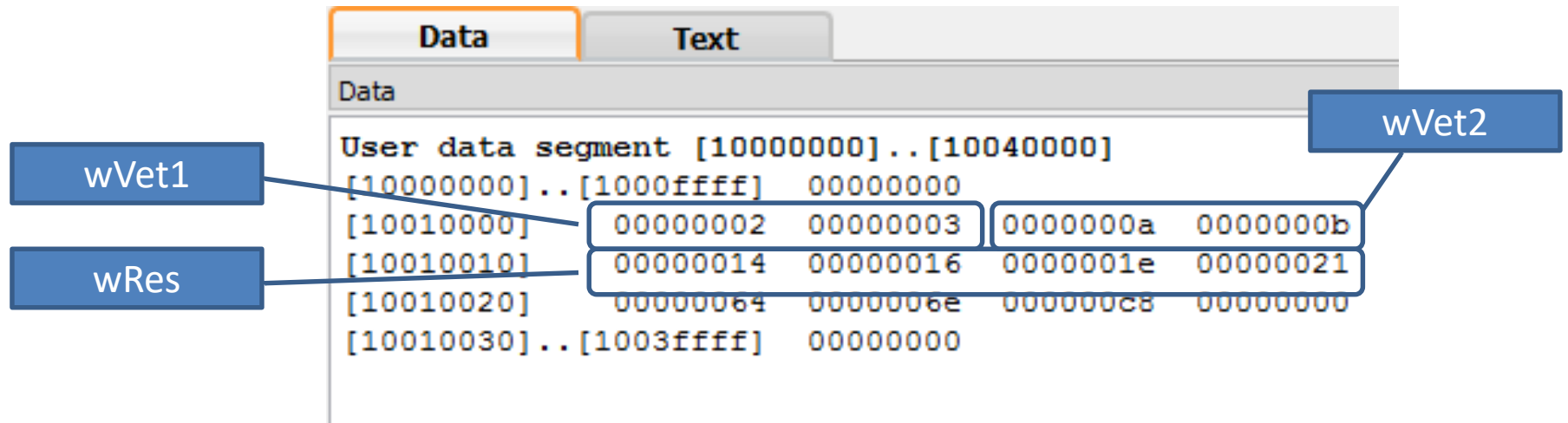
# La memoria



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

SysCall con cui gestire I/O

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

# SysCall

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

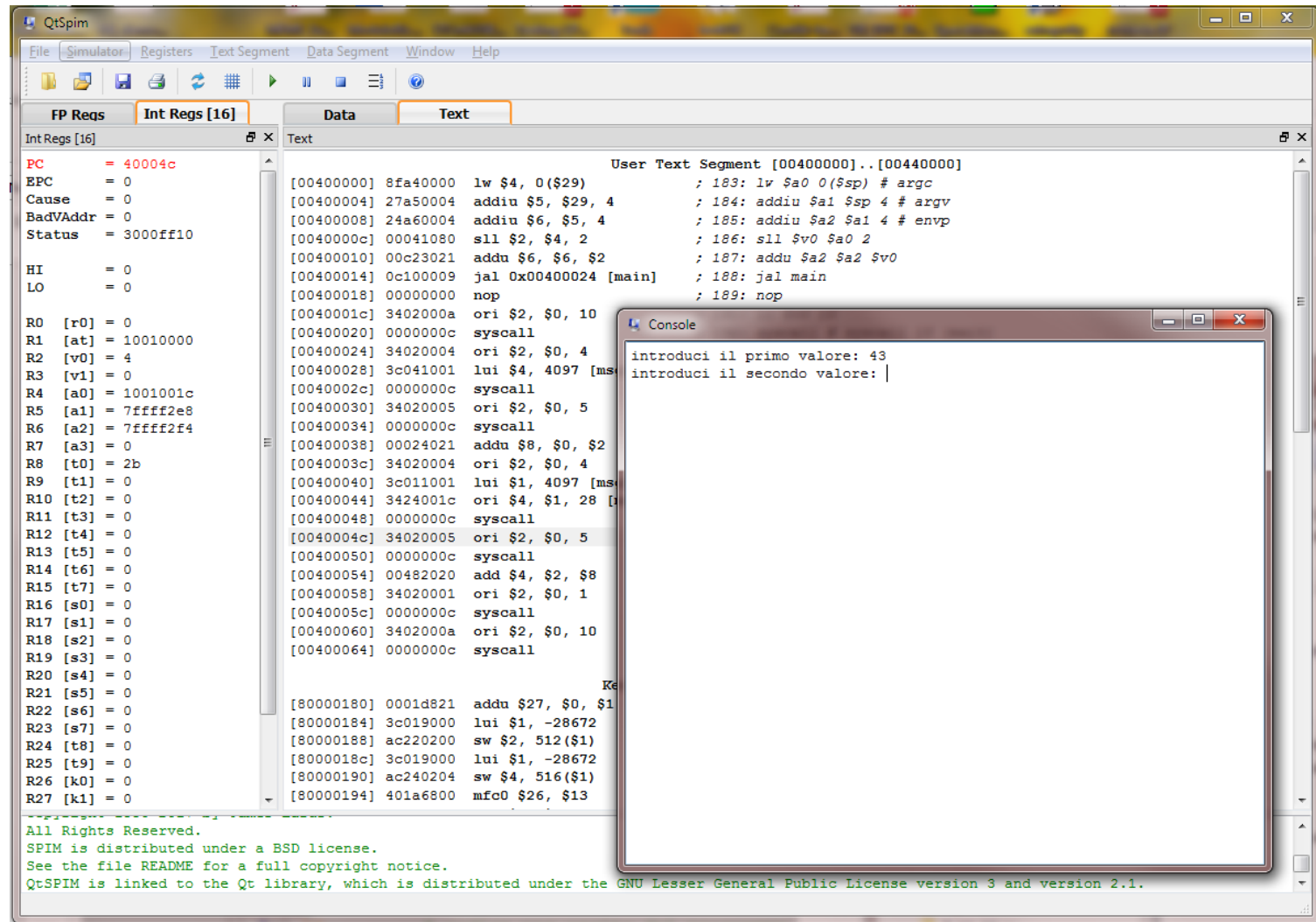
```
.data
msg1:      .asciiz "introduci il primo valore: "
msg2:      .asciiz "introduci il secondo valore: "

.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

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

# ASCII Table

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	À
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a	129	81	ù	161	A1	â	193	C1	Á
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	Â
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	Ã
4	4	End of xmit	EOT	CTRL-D	36	24	\$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	Ä
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e	133	85	å	165	A5	Ñ	197	C5	Å
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f	134	86	ä	166	A6	*	198	C6	Æ
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	°	199	C7	Ç
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h	136	88	è	168	A8	¿	200	C8	È
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i	137	89	é	169	A9	¬	201	C9	É
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	Ê
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	Ë
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	Ì
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m	141	8D	ï	173	AD	½	205	CD	Í
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	¼	206	CE	Î
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	¾	207	CF	Ï
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p	144	90	E	176	B0	¾	208	D0	Ï
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	¾	209	D1	Ï
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	¾	210	D2	Ï
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s	147	93	ø	179	B3	¾	211	D3	Ï
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t	148	94	ó	180	B4	¾	212	D4	Ï
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u	149	95	ö	181	B5	¾	213	D5	Ï
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	¾	214	D6	Ï
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w	151	97	ü	183	B7	¾	215	D7	Ï
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x	152	98	ý	184	B8	¾	216	D8	Ï
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y	153	99	ÿ	185	B9	¾	217	D9	Ï
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	¾	218	DA	Ï
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{	155	9B	ý	187	BB	¾	219	DB	Ï
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C		156	9C	£	188	BC	¾	220	DC	Ï
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	¾	221	DD	Ï
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~	158	9E	₣	190	BE	¾	222	DE	Ï
31	1F	Unit separator	US	CTRL-`	63	3F	?	95	5F	_	127	7F	DEL	159	9F	ƒ	191	BF	¾	223	DF	Ï