UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

# Assembly e catena programmativa (parte 3)

**Prof.ssa Giulia Cisotto**

giulia.cisotto@units.it

Trieste, 12 marzo 2026

# AGENDA DI OGGI

- Assembly (parte 3)
- Catena programmativa (parte 3)
- Procedure e Linker

RECAP

## Tipica struttura programma Assembly

```
# Title:                                Filename:
# Author:                               Date:
# Description:
# Input:
# Output:
################# Data segment #####################
.data
 . . .
################# Code segment #####################
.text
.globl main
main:                                   # main program entry
 . . .
li $v0, 10                              # Exit program
syscall
```

## Nomi riservati

| Nome Simbolico | Numero | Uso |
|---|---|---|
| $zero | 0 | Costante 0 |
| $at | 1 | Assembler temporary |
| $v0-$v1 | 2-3 | Functions and expressions evaluation |
| $a0-$a3 | 4-7 | Arguments |
| $t0-$t7 | 8-15 | Temporaries |
| $s0-$s7 | 16-23 | Saved Temporaries |
| $t8-$t9 | 24-25 | Temporaries |
| $k0-$k1 | 26-27 | Reserved for OS kernel |
| $gp | 28 | Global pointer |
| $sp | 29 | Stack pointer |
| $fp | 30 | Frame pointer |
| $ra | 31 | Return address |

**Direttive** all'assemblatore : .data, .text, .space, .ascii/.asciiz, …

**Istruzioni per operazioni aritmetiche e logiche:**

```
add rd, rs, rt
addi rd, rs, imm
and rd, rs, rt
:
```

**Pseudo-istruzioni, as esempio:** `li $s0, 0x12345678`

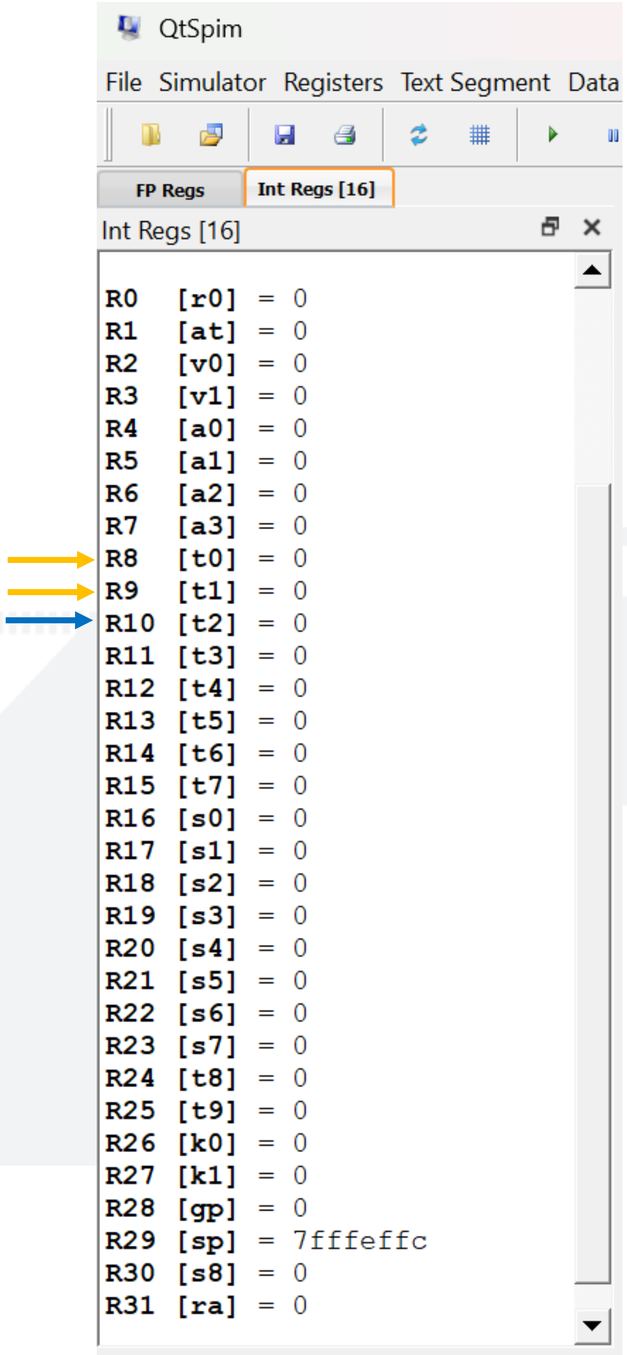
## SYS-CALL: chiamata al sistema (operativo)

| 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 | | |
| print_char | 11 | $a0 = char | |
| read_char | 12 | | char (in $a0) |
| open | 13 | $a0 = filename (string), $a1 = flags, $a2 = mode | file descriptor (in $a0) |
| read | 14 | $a0 = file descriptor, $a1 = buffer, $a2 = length | num chars read (in $a0) |
| write | 15 | $a0 = file descriptor, $a1 = buffer, $a2 = length | num chars written (in $a0) |
| close | 16 | $a0 = file descriptor | |
| exit2 | 17 | $a0 = result | |

**FIGURE A.9.1 System services.**

# ESEMPIO DI PROGRAMMA ASSEMBLY (1)

```
.text
main:
    li $t0, 0xCC          # A = 11001100
    li $t1, 0xAA          # B = 10101010
    and $t2, $t0, $t1     # $t2 = A AND B = 10001000
```



# ASSEMBLY E SISTEMA OPERATIVO

Il sistema operativo (SO) è un insieme di programmi che:

- stanno in un'area protetta della memoria (*kernel*)
- svolgono funzioni di utilità generale (in particolare, I/O) richiamabili dai programmi utente.

Il simulatore QtSpim fornisce alcune funzioni elementari che simulano alcune funzionalità base del SO richiamabili attraverso il meccanismo di **syscall**, concettualmente analogo a una chiamata a procedura

# PROGRAMMARE IN ASSEMBLY

**Chiamate a procedure di I/O del sistema operativo**

Modalità d'uso:

– mettere il/i parametro/i nei registri ($a0, $a1)

– specificare il tipo di system call, scrivendo un codice opportuno nel registro $v0

– syscall

**Esempio**: stampa dell'intero con segno in $t2

```
move $a0, $t2      # $a0=$t2
li $v0, 1          # codice 1 in $v0
syscall            # chiamata di sistema
```

# SYSCALL

## SYS-CALL: chiamata al sistema (operativo)

**Analogo a una chiamata a procedura:**

- Convenzioni per le syscall: Tabella a pag. A43 (Appendice A)
- Impostare i parametri nei registri $a0-$a3 (come da tabella)
- Impostare nel registro $v0 il codice della chiamata

**Syscall essenziali:**

- exit ($v0 ← 10): uscita dalla procedura
- exit2 ($v0 ← 17): terminazione del programma!
- read_int ($v0 ← 5): leggere un intero scritto dall'utente
- print_int ($v0 ← 1). ! parametro passato per valore
- read_string ($v0 ← 8): leggere una stringa scritta dall'utente
- print_string ($v0 ← 4). ! parametro passato per indirizzo

| 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 | | |
| print_char | 11 | $a0 = char | |
| read_char | 12 | | char (in $a0) |
| open | 13 | $a0 = filename (string), $a1 = flags, $a2 = mode | file descriptor (in $a0) |
| read | 14 | $a0 = file descriptor, $a1 = buffer, $a2 = length | num chars read (in $a0) |
| write | 15 | $a0 = file descriptor, $a1 = buffer, $a2 = length | num chars written (in $a0) |
| close | 16 | $a0 = file descriptor | |
| exit2 | 17 | $a0 = result | |

**FIGURE A.9.1 System services.**

**Esempio**

```
.data
msg: .asciiz "ciao"

.text
la $a0, msg
li $v0, 4
syscall
```

# ESEMPIO DI PROGRAMMA ASSEMBLY (2)

```
.data
vet: .word 2, 4, 6, 8, 10  # vettore di 5 interi

.text
.globl main

main:
    la $t0, vet         # Carica in $t0 l'indirizzo base del vettore
    lw $t1, 0($t0)      # Carica primo elemento (vet[0]=2) in $t1
    lw $t2, 16($t0)     # Carica terzo elemento (vet[2]=10) in $t2
    add $t3, $t1, $t2 # Somma $t1+$t2 → risultato in $t3
                       # risultato finale in $t3 (=12)
```

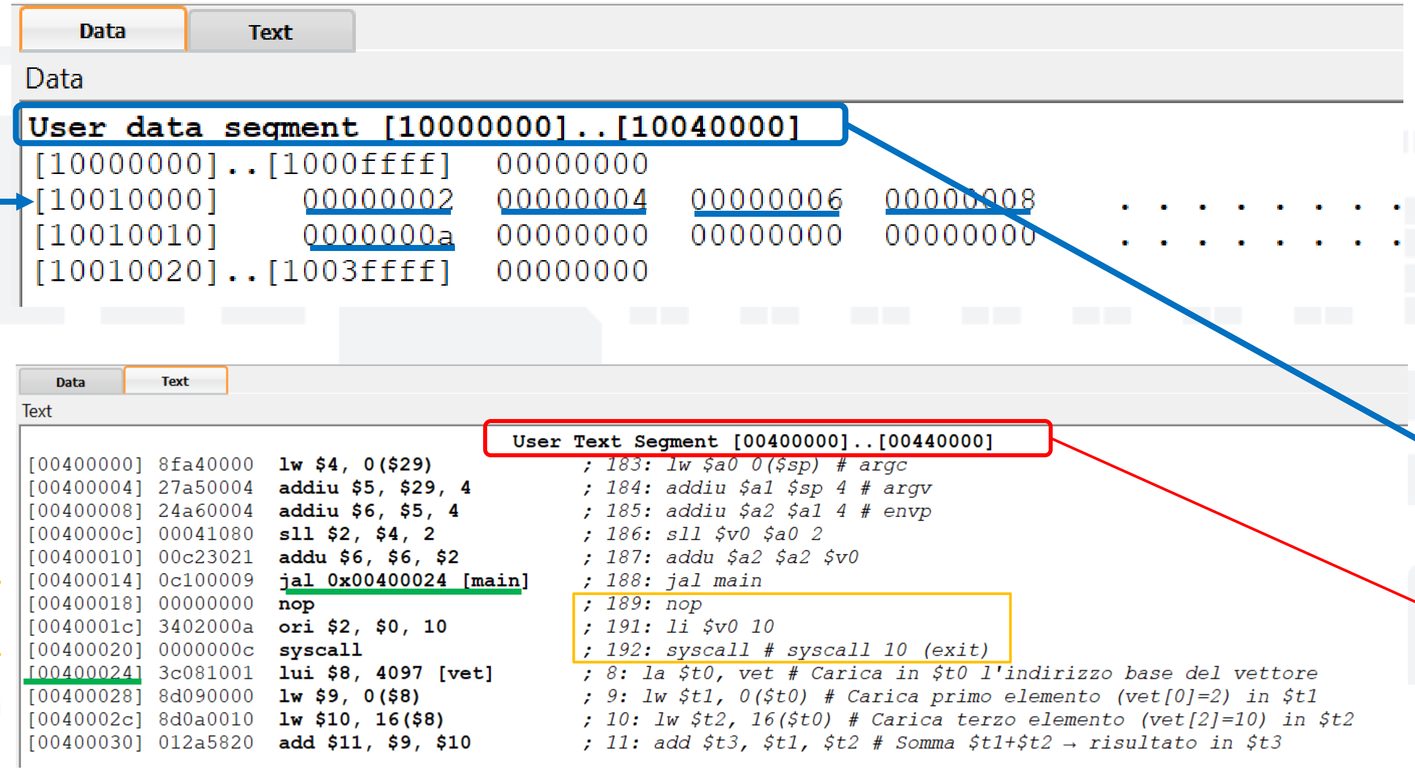
Nota. lw $t2, **16**($t0) = lw $t2, **0x10**($t0)

# ESEMPIO DI PROGRAMMA ASSEMBLY (2): NOTA

```
.data
vet: .word 2, 4, 6, 8, 10   # vettore di 5 interi

la $t0, vet                 # Carica in $t0 l'indirizzo base del vettore
```

è un'istruzione di **LOAD ADDRESS** → dà l'indirizzo dove inizia vet (10010000)



# SYSCALL: ESEMPIO

Scrivere un codice che stampa la stringa «La risposta è 5 »

```
.data
str: .asciiz “La risposta è “
numero: .word 5
```

# SYSCALL: ESEMPIO

Scrivere un codice che stampa la stringa «La risposta è 5 »

```
.data
str: .asciiz “La risposta è “
numero: .word 5

.text
.globl main
main:
li $v0, 4              #Codice della chiamata di sistema per print_str
la $a0, str            #Indirizzo stringa da stampare (passato per indirizzo!)
syscall                #Stampa la stringa
li $v0, 1              #Codice della chiamata di sistema per print_int
lw $a0, numero         #Intero da stampare (passato per valore!!!)
syscall                #Stampa l’intero
```

Chiamata al sistema per stampare la stringa

Chiamata al sistema per stampare il valore numerico

# Materiale per la lezione

- *Hennessy-Patterson, cap. 1 pp.14-16*
- *Appendix A.2, A.3, A.4, A.5*