Architettura degli Elaboratori

System calls, Procedure

SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Checco

alessandro.checco@uniroma1.it

alessandrochecco.github.io

Special thanks and credits:

Andrea Sterbini, Iacopo Masi,

Claudio di Ciccio

[S&PdC] 2.1 – 2.8

Argomenti

Argomenti della lezione

- Recap su Vettori e Matrici
- System calls
- Procedure (subroutines usando jal)
- Esercitazione

Scaricate il **simulatore MARS** da

http://courses.missouristate.edu/kenvollmar/mars/index.htm

- è scritto in Java (**gira su qualsiasi OS**)
- è **estensibile** (diversi plug-in per esaminare il comportamento della CPU e della Memoria)
- è un IDE integrato:
  - **Editor con evidenziazione della sintassi** assembly ed **autocompletamento** delle istruzioni
  - **Help completo** (istruzioni, syscall …)
  - **Simulatore** con possibilità di **esecuzione passo-passo** e uso di **breakpoint**
  - Permette l'**esecuzione da riga di comando** e la **compilazione di più file**

Vettori e Matrici

SAPIENZA
UNIVERSITÀ DI ROMA

Matrici: vettori di vettori

Una matrice **M × N** è altro una successione di M vettori, ciascuno di N elementi

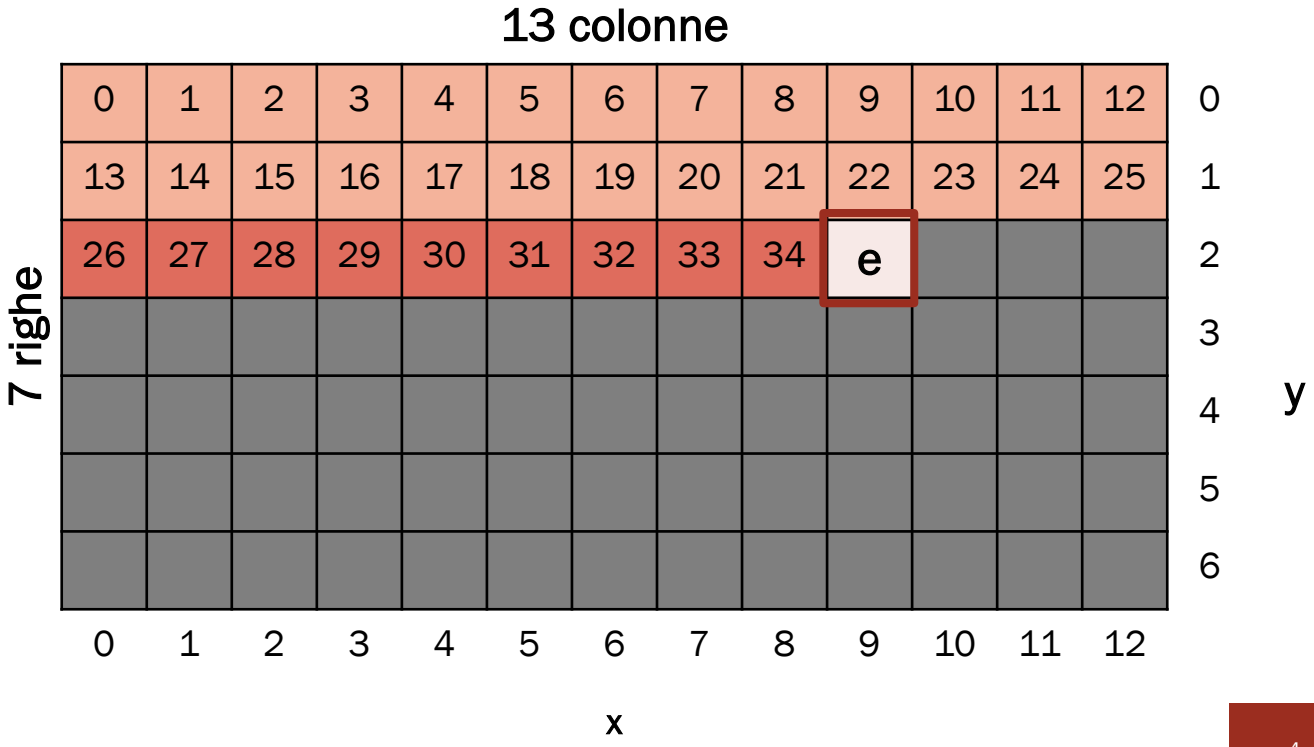
- il numero di elementi totali è: **M × N**
- la dimensione totale in byte è: **M × N × dimensione_elemento**
- la si definisce staticamente come un vettore contenente **M x N** elementi uguali

```
Matrice:    .word   0:91        # spazio per una matrice 7 × 13 word
```

L'elemento **e**, di coordinate **x=9, y=2** si trova ad una distanza di:

- **2** righe
- più **9** elementi

dall'inizio, ovvero ad un offset di **2×13+9 = 35 word** cioè **35×4 = 140 byte**



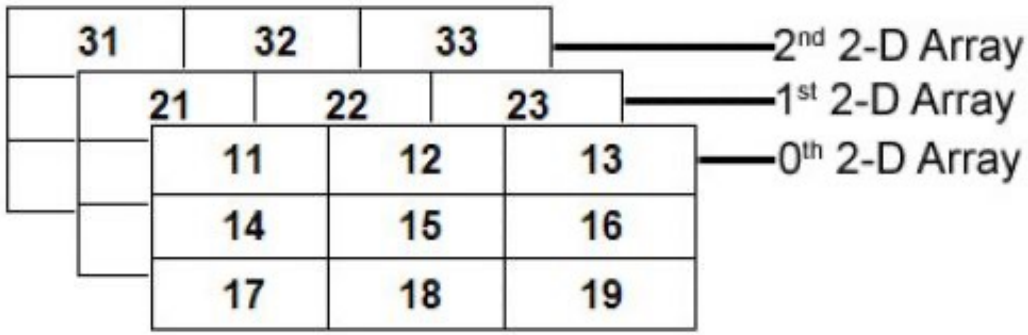
Matrici 3D

Questo è esattamente il calcolo svolto dal compilatore C, per esempio

scrivere `matrice[x][y]`

per una matrice di interi definita `int matrice[NUM_RIGHE][NUM_COL]`

equivale a usare l'indirizzo `matrice + (y*NUM_COL+x)*sizeof(int)`



Matrici a 3 dimensioni

Una matrice 3D di **dimensioni** **M x N x P** è una successione di **P** matrici 2D grandi **M x N**

L'elemento a coordinate **x, y, z** è preceduto da:

z «strati» (matrici **M x N** formate da M · N elementi)

y «righe» di M elementi sullo stesso strato

x «elementi» sulla stessa riga e strato

Quindi l'elemento si trova a **z * (M * N) + y * N + x** elementi dall'inizio della matrice 3D e la sua posizione in memoria è **indirizzo_matrice + (z * (M * N) + y * N + x) * dim_el.**

Recap

Matrici: Indirizzo + offset di riga dall'indirizzo fornisce l'elemento che vogliamo ottenere

$$(x, y) \leftarrow \underline{ind} + y \cdot X + x$$

$X = \#cols$

Matrici 3D: simili a vettori ma dobbiamo tenere traccia degli strati anche.

Indirizzo + offset di strato + offset di colonna

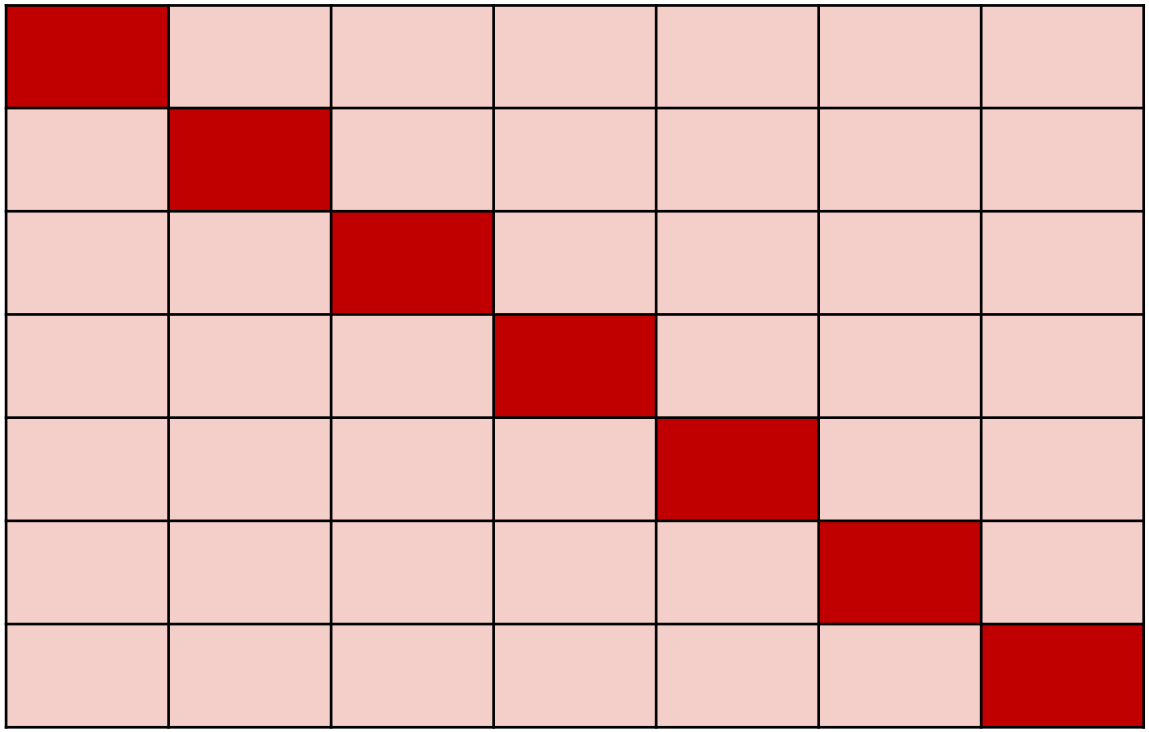
$$(x, y, z) \leftarrow ind + \left[z \cdot (X \cdot Y) + \overbrace{y \cdot X + x} \right]$$

offset strato

$X = \#cols$

$Y = \#righe$

Somma della diagonale (INEFF.)



Somma della diagonale (INEFF.)

```
.data
   matrice2D:   .word   0:400   # matrice quadrata di 20x20 word
   DIM:         .word   20      # lato della matrice
.text
main:          li  $t0, 0                  # coordinata x (colonne)
               li  $t1, 0                  # coordinata y (righe)
               li  $t2, 0                  # somma iniziale
               lw  $t3, DIM                # lato della matrice
cicloRighe:    bge $t1, $t3, fine          # se finite le righe
cicloColonne:  bge $t0, $t3, nextRiga      # se finite le colonne
               bne $t0, $t1, continua      # se x != y si continua
               mul $t4, $t1, $t3           # y*X
               add $t4, $t4, $t0           # y*X + x
               sll $t4, $t4, 2             # word => molt. per 4
               lw  $t4, matrice2D($t4)     # carico matrice2D[x][y]
               add $t2, $t4, $t2           # e lo accumulo
```

Somma della diagonale (segue)

```
continua:       addi    $t0, $t0, 1     # x += 1
                j       cicloColonne    # alla colonna successiva
nextRiga:       li      $t0, 0          # azzero x
                addi    $t1, $t1, 1     # y += 1
                j       cicloRighe      # alla riga successiva
fine:           move    $a0, $t2        # preparo la stampa
                li      $v0, 1          # syscall 1 = print int
                syscall
                li      $v0, 10         # syscall 10 = stop
                syscall
```

NOTA: questa è una versione volutamente inefficiente che scandisce tutta la matrice. Può essere resa più efficiente:

- **Usando un solo ciclo sulla coordinata x da 0 a DIM-1**
- **Usando i puntatori (indirizzi in memoria)**
- **Incrementando il puntatore di DIM+1 elementi = (DIM+1)*4 byte per passare da un elemento della diagonale al successivo**

Somma diagonale (EFFICIENTE)

```
.data
   matrice2D:   .word   0:400        # matrice di 20x20 word
   DIM:         .word   20           # lato della matrice quadrata
.text   # preparazione degli indirizzi e degli incrementi
main:   # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento (in byte)
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (byte seguente)
        la      $t0, matrice2D     # indirizzo dell'inizio
        lw      $t1, DIM           # lato della matrice
        mul     $t3, $t1, $t1      # DIM * DIM elementi
        sll     $t3, $t3, 2        # totale DIM^2 * 4 byte
        add     $t3, $t3, $t0      # indirizzo finale
        addi    $t1, $t1, 1        # DIM + 1
        sll     $t1, $t1, 2        # incremento = (DIM+1)*4
        li      $t2, 0             # somma iniziale
```

Somma della diagonale (segue)

```
        # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (subito dopo)
# ciclo che scandisce un elemento ogni DIM+1
ciclo:  bge     $t0, $t3, fine      # se è finita la matrice esco
        lw      $t4, ($t0)          # carico matrice2D[x][x]
        add     $t2, $t4, $t2       # e lo accumulo
        add     $t0, $t0, $t1       # x += (DIM+1)*4
        j       ciclo
# stampa del risultato
fine:   move    $a0, $t2            # preparo la stampa
        li      $v0, 1              # syscall 1 = print int
        syscall
        li      $v0, 10             # syscall 10 = stop
        syscall
```

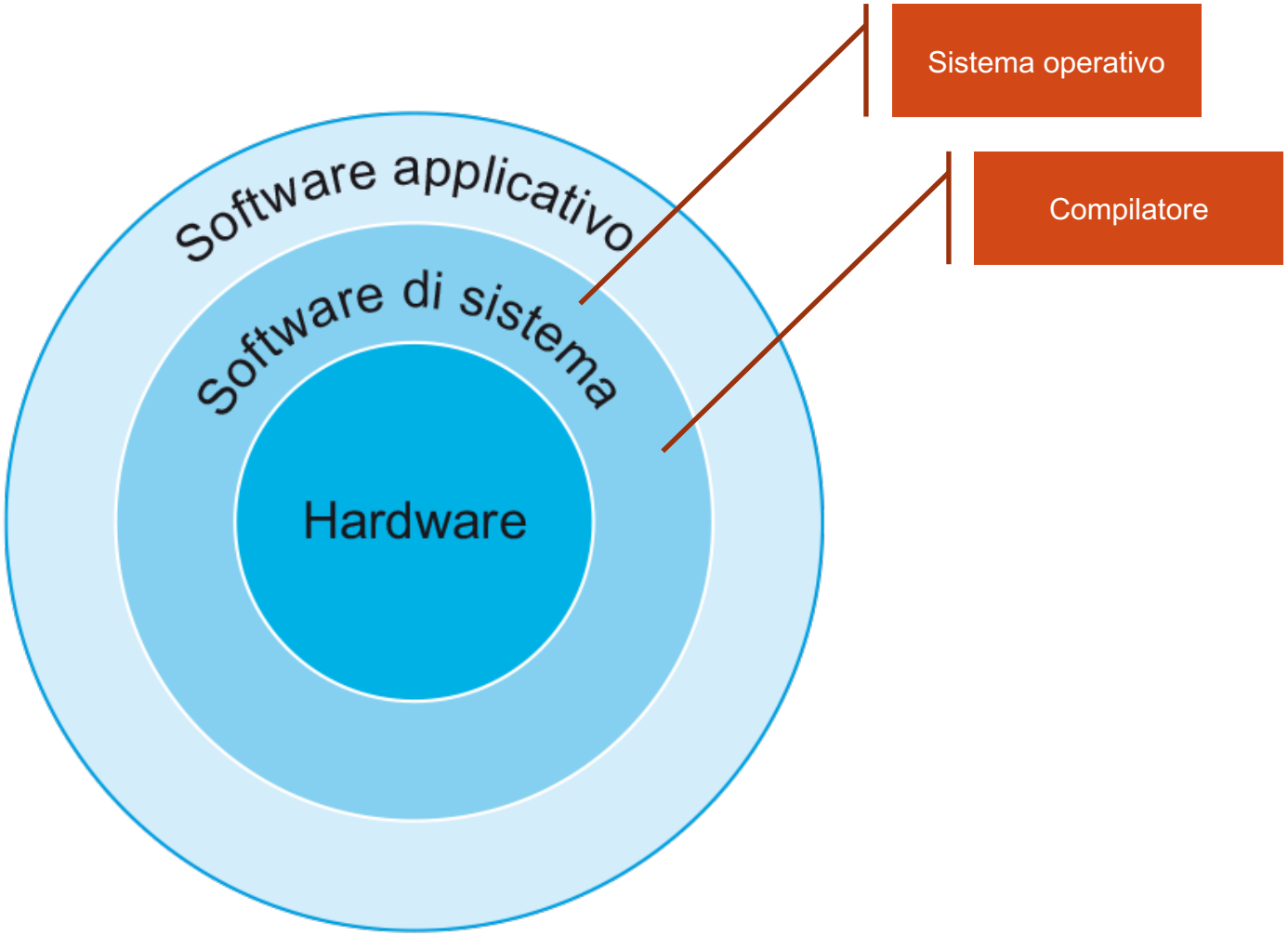
Somma della diagonale (segue)

```
        # $t0 = indirizzo dell'elemento corrente
        # $t1 = incremento di una riga + 1 elemento
        # $t2 = somma parziale
        # $t3 = indirizzo finale della matrice (subito dopo)
# ciclo che scandisce un elemento ogni DIM+1
ciclo:  bge     $t0, $t3, fine      # se è finita la matrice esco
        lw      $t4, ($t0)          # carico matrice2D[x][x]
        add     $t2, $t4, $t2       # e lo accumulo
        add     $t0, $t0, $t1       # x += (DIM+1)*4
        j       ciclo
# stampa del risultato
fine:   move    $a0, $t2            # preparo la stampa
        li      $v0, 1              # syscall 1 = print int
        syscall
        li      $v0, 10             # syscall 10 = stop
        syscall
```

System Calls

SAPIENZA
UNIVERSITÀ DI ROMA

System Calls



syscall

Richieste al sistema operativo

Input:

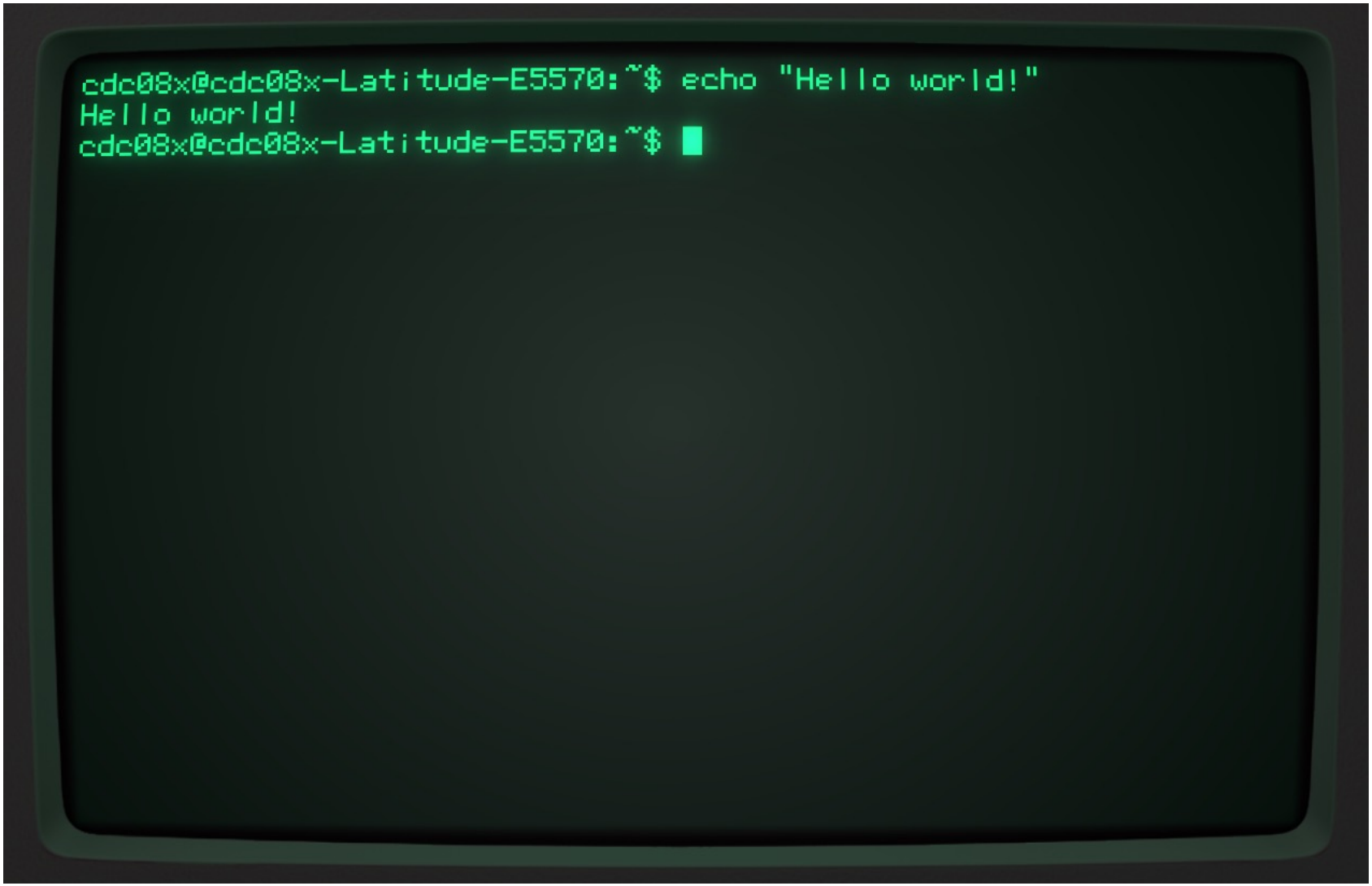
- $v0: operazione **richiesta**
- $a0..$a2,$f0: eventuali **parametri**

Output:

- $v0, $f0: eventuale **risultato**

Syscall ($v0)	Descrizione	Argomenti ($a0 ...)	Risultato ($v0 ...)
1	Stampa Intero	Intero	
4	Stampa Stringa	String Address	
5	Leggi Intero		Intero
8	Leggi Stringa	$a0 = buffer address $a1 = num chars.	
10	Fine programma		

Lo stavate aspettando



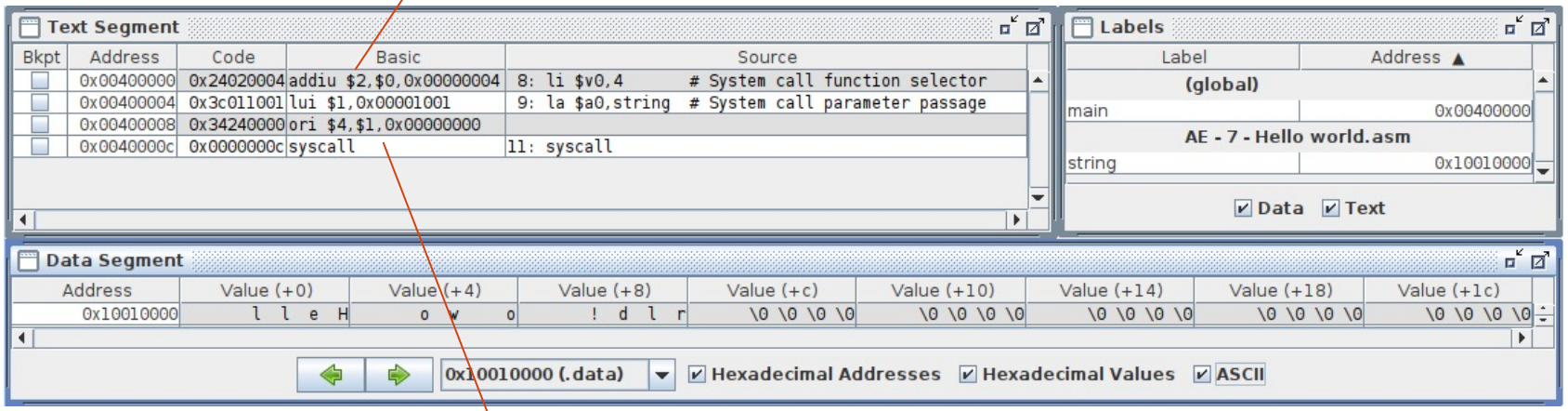
Lo stavate aspettando

```
.globl main

.data
string: .asciiz "Hello world!"

.text
main:
li $v0,4          # System call function selector
la $a0,string     # System call parameter passage

syscall
```

Esempio in MARS

Sulle pseudoistruzioni li e la

evitare complemento a 2 (estensione del segno)



indirizzo a 32bit → due istruzioni con immediate da 16bit

Codici per syscall

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 $v0)
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	

Procedure (subroutines) e Funzioni

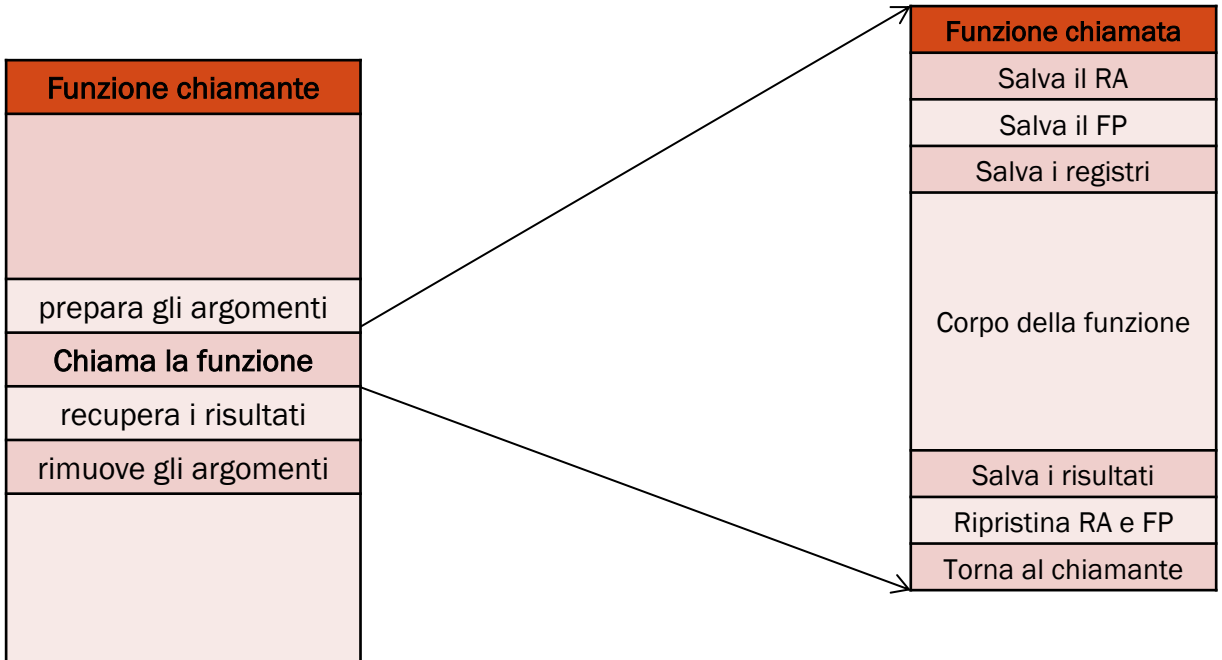
SAPIENZA
UNIVERSITÀ DI ROMA

Procedura e Funzioni

Funzioni / procedure:

Frammento di codice che riceve degli argomenti e calcola un risultato (**utile per rendere il codice riusabile e modulare**)

- ha un indirizzo di partenza
- riceve uno o più argomenti
- svolge un calcolo
- ritorna un risultato
- continua la sua esecuzione dall'istruzione seguente a quella che l'ha chiamata



Funzioni: ingredienti / 1 – Salti incondizionati

Salto condizionato

```
While: # while (i < N) {
  bge $t1,$s7,WhileEnd # IF i >= N THEN jump to WhileEnd
  lw $t2,Array($t1) # Load Array[i];
  add $s0,$s0,$t2 # s = s + Array[i];
  addi $t1,$t1,4 # i+=1;
  j While # }
WhileEnd:
```

Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

Image © Van Halen – Jump (1983) https://youtu.be/SwYN7mTi6HM

Istruzioni necessarie

Per chiamare la funzione/procedura

`jal etichetta` Jump And Link

(simile ad un salto a subroutine ma più limitato)

ricorda nel registro `$ra` la posizione dell'istruzione successiva ($ra <- PC+4)

cambia il PC per iniziare l'esecuzione del corpo della funzione (PC <- etichetta)

Per tornare e continuare l'esecuzione del programma chiamante

`jr $ra` Jump to Register

salta all'indirizzo contenuto nel registro indicato (PC <- $ra)

Come passare valori ALLA funzione (caso semplice)

`$a0`, `$a1`, `$a2`, `$a3` 4 registri per passare fino a 4 valori a 32 bit o 2 a 64 bit

e restituirli DALLA funzione

`$v0`, `$v1` 2 registri per restituire fino a 2 valori a 32 bit (o 1 a 64 bit)

nota (convenzioni):

`$t0`, `$t1`, . . . possono cambiare tra una chiamata e l'altra (temporary)

`$s0`, `$s1`, . . . non cambiano tra una chiamata e l'altra (saved)

Case Switch Program + System Calls + Subroutines

main

```
1  .globl main, case_string
2
3  .data
4    switch_case:  .word case_00, case_01
5    case_string:  .asciiz "Ran case #: "
6
7  .text
8  main:
9    jal read_int
10   move $t2,$v0
11   jal read_int
12   move $t3,$v0
13   jal read_int
14   move $t1,$v0
15   sll $t0, $t1, 2           # the index
16                             # so I have
17                             # the offse
18   lw $s1, switch_case($t0)  # load the
19   la $a0, case_string
20   jal print_string          # Print the
21   move $a0,$t1
22   jal print_rez
23   jr $s1                    # jump to t
24
25 case_00:
26   #somma
27   add $t3,$t3,$t2
28   j end_switch
29
30 case_01:
31   mul $t3,$t3,$t2
32   j end_switch
33
34 end_switch:
35   li $a0,0x0A  #10          # 10 is asci
36   jal print_char
37   move $a0,$t3
38   jal print_rez
39   j exit
```

aux

```
1  .globl print_string, read_int, print_rez, print_char, exit
2
3  .data
4    # Syscall lookup table
5    print_s:    .byte 4
6    print_i:    .byte 1
7    read_i:     .byte 5
8    print_c:    .byte 11
9    exit_s:     .byte 10
10
11 .text
12 print_string:                # cenno ad una procedura
13   lb $v0, print_s              # print the template string (4 means
14   syscall
15   jr $ra
16
17 print_rez:
18   lb $v0, print_i              # print an integer (1 means that)
19   syscall
20   jr $ra
21
22 read_int:
23   lb $v0, read_i
24   syscall
25   jr $ra
26
27 print_char:
28   lb $v0, print_c
29   syscall
30   jr $ra
31
32 exit:
33   lb $v0, exit_s            # Very important: tells the OS to exit the
34   syscall                   # if we do not add this we go in infinite l
35
```