



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche,
Informatiche e Matematiche

Esercitazione 4:

Programmazione Assembly

Architettura dei calcolatori [MN1-1143]

Corso di Laurea in Ing. Informatica
(D.M.270/04) [16-215]
Anno accademico 2020/2021

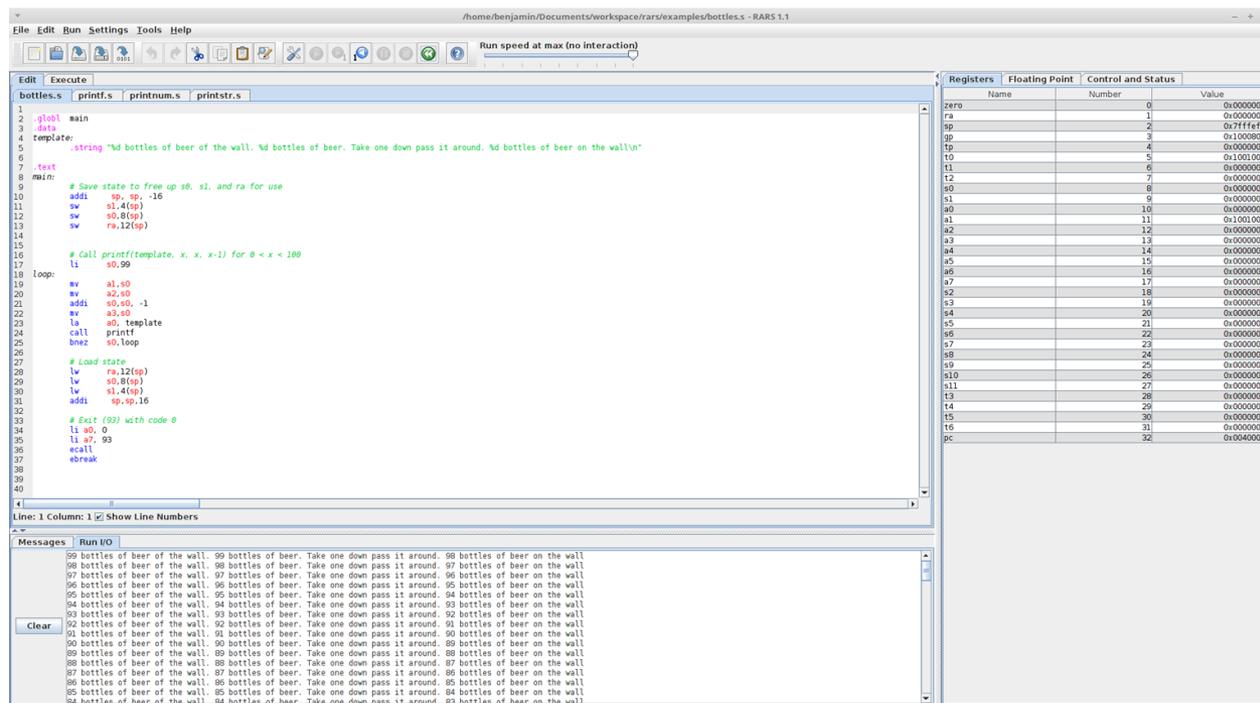
Dott. Gianluca Brilli
gianluca.brilli@unimore.it
Prof. Marko Bertogna
Marko.bertogna@unimore.it

È vietata la copia e la riproduzione dei contenuti e immagini in qualsiasi forma.

È inoltre vietata la redistribuzione e la pubblicazione dei contenuti e immagini non autorizzata espressamente dall'autore o dall'Università di Modena e Reggio Emilia.

Ambiente di lavoro RARS

RARS: RISC-V Assembler and Runtime Simulator



Ambiente di lavoro

RARS – Componenti fondamentali

The screenshot displays the RARS development environment. The main window is titled "C:\Users\Gianb\Documents\Didattica\test.asm - RARS 1.4". It features a menu bar (File, Edit, Run, Settings, Tools, Help) and a toolbar with various icons. The central pane is an assembly editor showing the following code:

```
1 .globl main
2
3 .data
4 msg1: .string "Hello World "
5 .text
6 main:
7     # Write to a filedescriptor from a buffer
8     li a0, 1
9     la a1, msg1
10    li a2, 12
11    li a7, 64
12    ecall
13
14    # print int
15    li a0, 27
16    li a7, 1
17    ecall
18
19    # exit with code 0 ( return 0 )
20    li a0, 0
21    li a7, 93
```

Below the editor is a "Messages" window with a "Run I/O" tab, showing the output:

```
Hello World 27
-- program is finished running (dropped off bottom) --
27
-- program is finished running (dropped off bottom) --
```

On the right side, there is a "Registers" window with tabs for "Registers", "Floating Point", and "Control and Status". The "Registers" tab is active, showing a table of registers:

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffffc0
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400000

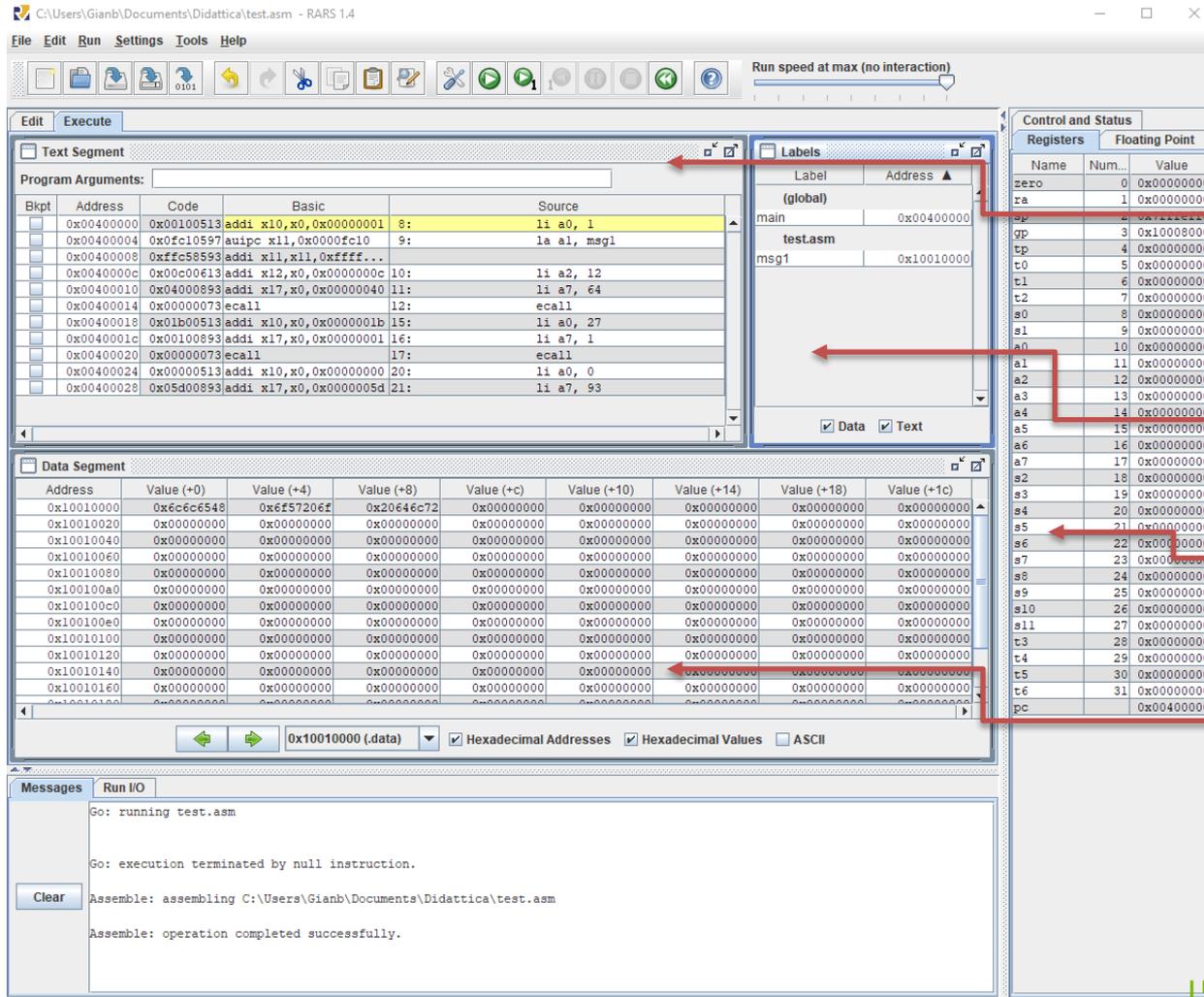
Red arrows point from the text annotations to the assembly editor, the registers window, and the terminal output window.

*Editor del nostro
ambiente
di
sviluppo.*

*Terminale
per
l'input/output.*

Ambiente di lavoro

RARS – Componenti fondamentali



Segmento *text* disassemblato.

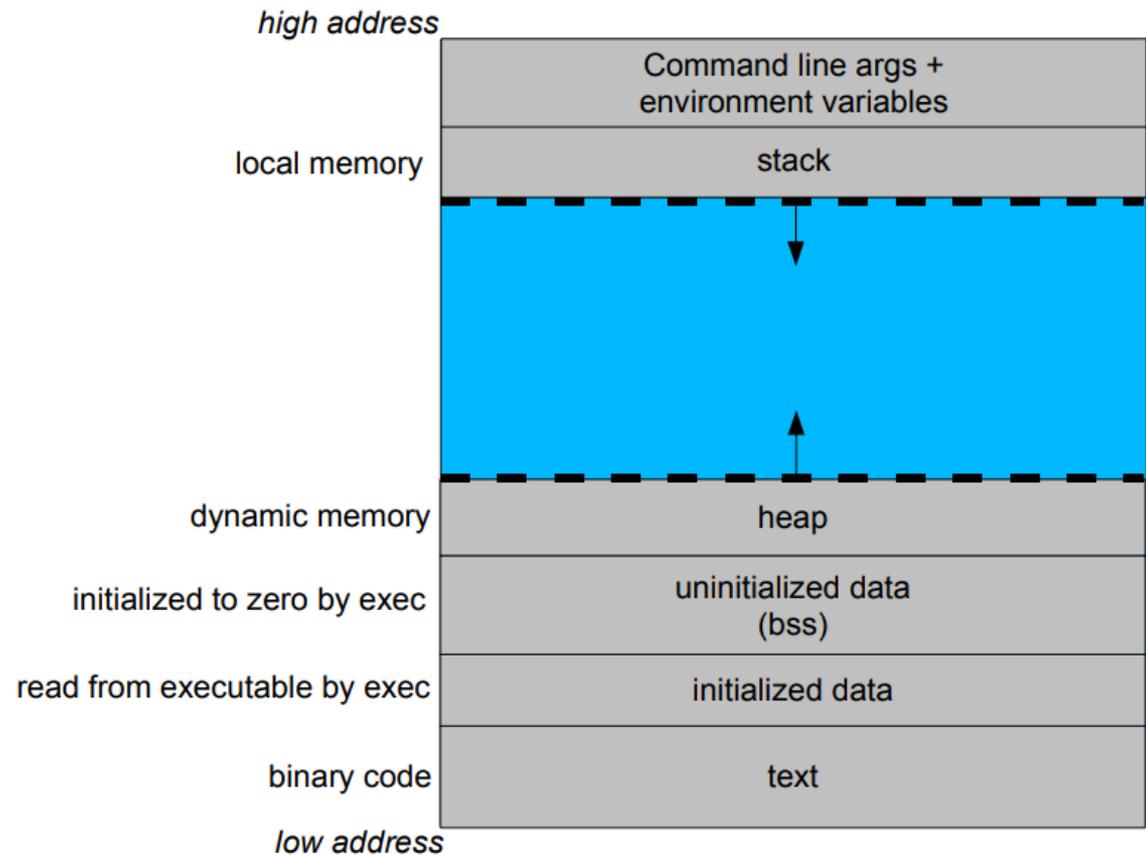
Symbol *table*, mapping tra label e rispettivo indirizzo.

Stato dei registri della CPU in tempo reale.

Segmento *data* disassemblato.

Alcuni Concetti Fondamentali

Segmentazione della memoria



Alcuni Concetti Fondamentali

Registri e ABI name

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5–7	t0–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Alcuni Concetti Fondamentali

Environment Calls

- *Particolari chiamate a funzione che vengono utilizzate dal programma per richiedere un servizio al kernel o all'environment.*
- *Esempio:*

Name	Call Number (a7)	Description	Inputs	Outputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints an floating point number	fa0 = float to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A

- *Lista completa:* <https://github.com/TheThirdOne/rars/wiki/Environment-Calls>

Alcuni Concetti Fondamentali

Environment Calls

→ Esempio di stampa di un intero tramite environment call:

Name	Call Number (a7)	Description	Inputs	Outputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints an floating point number	fa0 = float to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A

li
a
0, 15
li
a
7, 1
ecall

→ Caricare il call number dentro al registro a7 ed i parametri di input all'interno dei registri a0, a1, ...

→ Successivamente utilizzare l'istruzione ecall per avviare la environment call.

Struttura di un Programma ASM

.globl main

.data

msg1: .string "Hello World "

.text

main:

Write to a filedescriptor from a buffer

li a0, 1

la a1, msg1

li a2, 12

li a7, 64

ecall

print int

li a0, 27

li a7, 1

ecall

exit with code 0 (return 0)

li a0, 0

li a7, 93

ecall

→ *Identifica "main" come simbolo globale.*

Struttura di un Programma ASM

```
.globl main

.data
    msg1: .string "Hello World "

.text
    main:
        # Write to a filedescriptor from a buffer
        li a0, 1
        la a1, msg1
        li a2, 12
        li a7, 64
        ecall

        # print int
        li a0, 27
        li a7, 1
        ecall

        # exit with code 0 ( return 0 )
        li a0, 0
        li a7, 93
        ecall
```



→ *Identifica l'inizio del segmento "dati" nel quale vengono posizionate variabili globali in lettura/scrittura.*

Struttura di un Programma ASM

```
.globl main

.data
    msg1: .string "Hello World "

.text
    main:
        # Write to a filedescriptor from a buffer
        li a0, 1
        la a1, msg1
        li a2, 12
        li a7, 64
        ecall

        # print int
        li a0, 27
        li a7, 1
        ecall

        # exit with code 0 ( return 0 )
        li a0, 0
        li a7, 93
        ecall
```



→ *Identifica l'inizio del segmento "text" il quale contiene le istruzioni del nostro programma.*

Struttura di un Programma ASM

```
.globl main

.data
    msg1: .string "Hello World "

.text
    main:
        # Write to a filedescriptor from a buffer
        li a0, 1
        la a1, msg1
        li a2, 12
        li a7, 64
        ecall

        # print int
        li a0, 27
        li a7, 1
        ecall

        # exit with code 0 ( return 0 )
        li a0, 0
        li a7, 93
        ecall
```

→ *Chiamata d'ambiente "exit", identificata dal codice numerico "93". Corrisponde all'istruzione "return 0" del linguaggio C/C++.*

Esercizio 01

Loop tramite istruzioni di branching

→ *Provare a tradurre in Assembly il seguente frammento di codice C:*

```
int main () {  
    int n = 10;  
    for (int i = 0; i < n; ++i) {  
        printf("i: %d\n", i);  
    }  
    return 0;  
}
```

Esercizio 02

Chiamate a funzione

→ *Provare a tradurre in Assembly il seguente frammento di codice C:*

```
int sum(int n) {
    int sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += i;
    }
    return sum;
}
int main () {
    printf("sum: %d\n", sum(10));
    return 0;
}
```

Accesso alla Memoria

Segmento Dati e Stack

→ Esempio di Lettura di un vettore salvato nel segmento dati:

.data

vett: .word 1, 2, 3, 4

la	t0,	→ accesso all'indirizzo di memoria.
	vett	→ accesso agli elementi a partire
lw	t0,	dall'indirizzo precedente:
	0(t0)	→ t0 = vett[0]
lw	t1,	→ t1 = vett[1]
	4(t0)	
...		

Ricordarsi di sfruttare opportunamente l'offset in base a come è definita la memoria (word in questo caso).

Esercizio 03

String Copy

→ Copiare il file sorgente di seguito, ed inserire le opportune direttive per permettere l'assemblamento.

strcpy:

```
addi    sp, sp, -8           # adjust stack for 1 dw
sd      x19, 0(sp)          # push x19
add     x19, x0, x0          # i=0
L1:
add     x5, x19, x11         # x5 = addr of y[i]
lbu    x6, 0(x5)            # x6 = y[i]
add     x7, x19, x10         # x7 = addr of x[i]
sb     x6, 0(x7)            # x[i] = y[i]
beq    x6, x0, L2           # if y[i] == 0 then exit
addi   x19, x19, 1          # i = i + 1
jal    x0, L1                # next iteration of loop
L2:
ld     x19, 0(sp)           # restore saved x19
addi   sp, sp, 8            # pop 1 dw from stack
jalr   x0, 0(x1)           # and return
```

Esercizio 04

String Copy

→ *Eeguire il programma precedente con due stringhe di prova, ad esempio:*

```
.data  
    X: .string «mario\0»  
    Y: .string «rossi\0»
```

→ *Visualizzare tramite RARS il contenuto del segmento di memoria e verificare l'effettivo funzionamento del programma.*

Ambiente di lavoro

RARS – Documentazione Disponibile

→ *Documentazione utile:*

- <https://github.com/TheThirdOne/rars/wiki/Supported-Instructions>
- <https://github.com/TheThirdOne/rars/wiki/Environment-Calls>
- <https://github.com/TheThirdOne/rars/wiki/Assembler-Directives>
- <https://github.com/TheThirdOne/rars/wiki>

Esercizio 05

Operazioni su array

- Realizzare un programma che vada a calcolare il prodotto scalare tra due array a e b , memorizzati ad esempio nel segmento dati.
- Si ricorda che il prodotto scalare può essere calcolato per mezzo della seguente formula:

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z = \vec{A} \cdot \vec{B}$$

Esercizio 06

Fattoriale ricorsivo

- Realizzare un programma in grado di calcolare il fattoriale in maniera ricorsiva:
 - $n! = n \times (n-1)!$
 - $n! = 1$ if $n = 0$ or $n = 1$

Esercizio 07

Maschere di bit

- Utilizziamo le maschere di bit per realizzare un meccanismo di controllo errori, basato sul calcolo della parità.
- In pratica quando viene trasmesso un dato, viene aggiunto un bit che è settato 1 se il numero di "1" nel dato da trasmettere è dispari (parità pari).
- Realizzare un programma che vada a settare il bit di parità per un dato a 7 bit.

7 bit di dati	Byte con bit di parità	
	Bit di parità pari	Bit di parità dispari
1101001	0 1101001	1 1101001
1111111	1 1111111	0 1111111

- Il bit più significativo del nostro byte rappresenta il bit di parità (come in figura).

Esercizio 08

Sorting

→ *Dato il seguente pseudocodice:*

```
procedure SelectionSort(a: lista dei numeri da ordinare);  
  for i = 0 to n - 1  
    posmin ← i  
    for j = (i + 1) to n  
      if a[j] < a[posmin]  
        posmin ← j  
    if posmin ≠ i  
      tmp ← a[i]  
      a[i] ← a[posmin]  
      a[posmin] ← tmp
```

→ *Che implementa l'algoritmo di ordinamento Selection Sort*

https://it.wikipedia.org/wiki/Selection_sort.

→ *Convertirlo in codice Assembly RISC-V.*