



STRUTTURA METORIA > COMPORTAMENTI

La memoria primaria è formata da varie sezioni
(Dall'alto verso l'alto)

RISERVATO

Area riservata ad utilizzo interno della CPU.

Allocazione della memoria per programmi e dati nel RISC-V

- Sarà visto nel dettaglio durante la teoria

- Gli indirizzi indicati sono frutto solo di convenzioni software

- Lo stack pointer decrece verso il basso verso il segmento dati

- I segmenti di testo contengono il codice del programma

- I dati statici e dinamici sono "sopra" (ad indirizzi maggiori) il segmento testo

SP → 0000 003f ffff fff0_{hex}



0000 0000 1000 0000_{hex}

PC → 0000 0000 0040 0000_{hex}

0

TESTO

Area contenente le istruzioni in linguaggio macchina da eseguire.

DATI STATICI

Area contenente costanti, vettori ecc.

STACK

Area di memoria contenente

↓ - Dati dei record di attivazione (main, funzioni, variabili,...)

↑ - Dati allocati dinamicamente (Tramite Heap)

ASSEMBLY > COMPONENTI

COMPONENTI : Si scrivono col

ETICHETTE :

Sezioni alfabetiche usate per spostarsi nel codice.

DIRETTIVE :

Elementi che specificano modificatori sul codice.

MACRO :

Etichette il cui valore viene sostituito in fase di assemblying in linguaggio machine

RISC-V - Etichette

```
mylabel:
# Instructions here are reached using the "mylabel" label
mydata:
# Data declared here can be referenced using the "mydata" label
```

• Un'etichetta deve essere scritta come:

name:

Directive	Funzione
global <label>	Definisce un simbolo come globale
text <label>	Zona del programma, i.e., dove si trovano le istruzioni del codice
data <label>	Zona nella memoria dove ci sono i dati
word <label>	Indica che il valore in "data" deve essere rappresentato con 32-bit (word)
eq <NAME> <VALUE>	Definisce una costante, ad esempio, .eqv HOMER 4

Codice in assembly per scambiare 2 registri, usando un terzo registro come appoggio

```
# swap macro
.macro swap reg1, reg2, reg3
    add $reg3, $reg2, zero
    add $reg2, $reg1, zero
    add $reg1, $reg3, zero
.endm
```

li \$1, 10
li \$2, 20
li \$3, 30
swap \$1, \$2, \$0

* tradotto come

```
add $1, $2, zero
add $2, $1, zero
add $1, $3, zero
add $3, $1, zero
```

Assembly degli indirizzi:

add \$1, \$2, zero

11

2

STRUTTURA ➤

".globl _start" Dichiara l'etichetta _start come globale

".text" fa iniziare il programma

"exit" effettua delle call di sostanzia per terminare il programma.

".data" Contiene delle etichette con dei valori in memoria.

```

1 .globl _start
2
3 .text
4
5 _start:
6
7 #<assembler code here>
8
9
10 exit:
11 addi x17, x0, 10 # call number 10 = exit
12 ecall
13

```

Edi Execute

0_template.asm 5_countones.asm 6_for_for.asm 0_example.asm

```

4 .data
5   v1: .word 8
6   v2: .word 9
7   v3: .word 0
8
9 .text
10 _start:
11   la t0, v1      # base address of our .data segment
12   lb t1, 1(t0)    # copy the first word to t1
13   lw t2, 4(t0)    # copy the second word to t2 -
14   add t3, t1, t2  # sum the words and save it on reg:
15   sw t3, 8(t0)    # store the sum in the 3rd word sta-
16
17 print:
18   addi a0, t3, 0      # print the result
19

```

Line: 5 Column: 5 Shape Line Numbers

```

1 .globl _start
2
3 .data
4   w1: .word 1
5   w2: .word 9
6   w3: .word 5
7   w4: .word 13
8
9 .text
10
11 _start:
12   # carico i valori
13   la a1, w1      # carico l'indirizzo di w1 in a1
14   lw t1, 0(a1)    # carico il valore dell'indirizzo in a1 dentro t1
15
16   la a2, w2      # stessa cosa
17   lw t2, 0(a2)

```

NB: Per i vettori vanno separati da celle,

```

.data
vet: .word 1, 2, 3, 4, 5, 6
result: .word 0
.text

_start:
la a1, vet      # carico il valore in memoria
lw s1, 0(a1)    # di v[0]

```

```

bge s3, s2, end_loop
# then
add s0, s0, s1
addi a1, a1, 4
lw s1, 0(a1)
addi s3, s3, 1
beq zero, zero, loop
# sommo valore
# aumento l'indirizzo per passare al valore successivo
# ricarico dentro al il valore
# aumento contatore
end_loop:

```