



Dal sorgente all'eseguibile I programmi Assembly

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borgese@dsi.unimi.it
Università degli Studi di Milano

Riferimenti sul Patterson: Cap. 2.10 + Appendice A, tranne A.7



Sommario

Dal linguaggio ad alto livello al codice in memoria

Lo SPIM e gli elementi di un programma Assembly

Esempi di programmi Assembly: costanti e ricorsione



Le istruzioni in linguaggio macchina



- Linguaggio di programmazione direttamente comprensibile dalla macchina
 - Le parole di memoria sono interpretate come *istruzioni*
 - Vocabolario è *l'insieme delle istruzioni (instruction set)*

Programma in
linguaggio ad alto livello
(C)

```
a = a + c  
b = b + a  
var = m [a]
```



Programma in linguaggio
macchina

```
011100010101010  
000110101000111  
000010000010000  
001000100010000
```



Le istruzioni di un'ISA



Devono contenere tutte le informazioni necessarie ad eseguire il ciclo di esecuzione dell'istruzione. Registri, comandi,

Ogni architettura di processore ha il suo linguaggio macchina

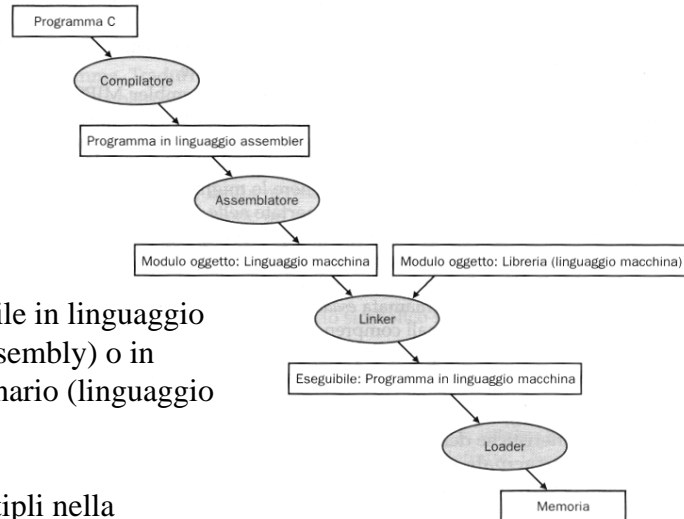
- Architettura definita dall'insieme delle istruzioni elementari.
 - **ISA (Instruction Set Architecture)**
- Due processori con lo stesso linguaggio macchina hanno la stessa architettura delle istruzioni anche se le implementazioni hardware possono essere diverse.
- Consente al SW di accedere direttamente all'hardware di un calcolatore



Dai simboli ai numeri binari

ISA esprimibile in linguaggio simbolico (assembly) o in linguaggio binario (linguaggio macchina).

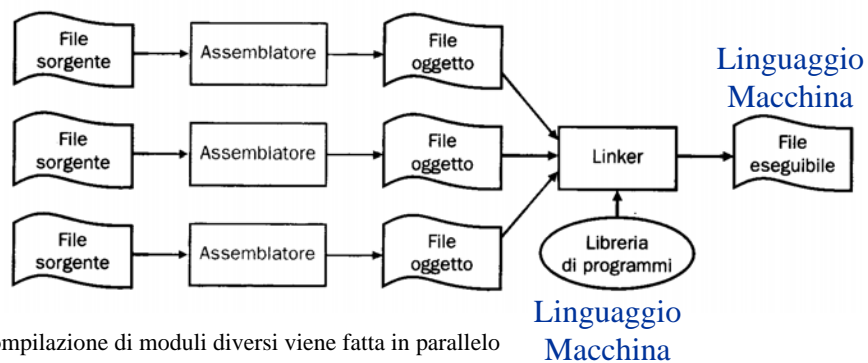
Passaggi multipli nella traduzione.



Dall'assembly all'eseguibile

Assembly

Linguaggio
Macchina



Compilazione di moduli diversi viene fatta in parallelo

Linguaggio
Macchina



Il compilatore



Dal codice sorgente C all'assembly (dipende dall'ISA dell'HW)

Il numero di linee aumenta notevolmente

Le strutture degli oggetti vengono tradotte in codice che gestisce la memoria riservata all'oggetto (base + offset).



L'assemblatore



Adatta il codice Assembly all'ISA (Assembly) dell'Architettura e quindi codifica in linguaggio macchina.

Esempi di adattamento del codice Assembly:

Sviluppo delle pseudo-istruzioni:

move \$t0, \$t1 → add \$t0, \$zero, \$t1?

blt (branch on less than) → slt + bne

far branch → branch + jump

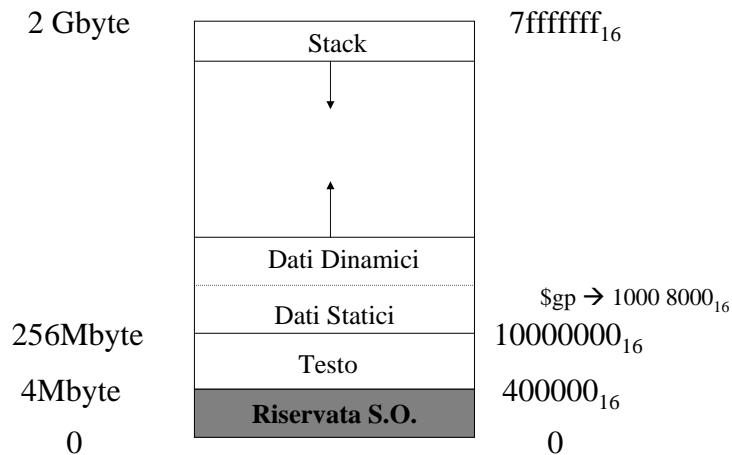
Conversione delle costanti da una qualsiasi base in base Hex.

Traduzione in linguaggio macchina (creazione del [file oggetto](#)).

Compilatore ed assemblatore possono essere uniti in un'unica fase.



Organizzazione logica della memoria



A.A. 2007-2008

9/46

<http://homes.dsi.unimi.it/~borghese>



MIPS: Software conventions for Registers



0	zero constant 0	16	s0 callee saves
1	at reserved for assembler	...	(caller can clobber)
2	v0 expression evaluation &	23	s7
3	v1 function results	24	t8 temporary (cont'd)
4	a0 arguments	25	t9
5	a1	26	k0 reserved for OS kernel
6	a2	27	k1
7	a3	28	gp Pointer to global area
8	t0 temporary: caller saves	29	sp Stack pointer
...	(callee can clobber)	30	fp frame pointer (s8)
15	t7	31	ra Return Address (HW)

$\$gp$ punta a metà del primo segmento dati: da 256,000 kByte a 256,064kByte, cioè all'indirizzo (256,032) Kbyte. Accesso alla memoria come *lw rt, Offset(\$gp)*.
(NB Offset è espresso in complemento a 2 su 16 bit ed è compreso tra $-2^{15}-1$ e $+2^{15}-1$).

A.A. 2007-2008

10/46

<http://homes.dsi.unimi.it/~borghese>



Impostazione corretta del \$gp e dell'offset



La memoria viene letta/scritta con indirizzamento mediante $\text{Indirizzo_base} + \text{offset}$
(ad esempio `lw $t0, Offset(<Registro_contenente_indirizzo_base>)`)

Il \$gp punta solitamente ad un indirizzo 32kbyte sopra il limite inferiore del segmento dati:

Segmento dati: 256Mbyte = 0x1000 0000 byte.

Offset: 32Kbyte = 0x8000byte

$\$gp = 0x1000\ 0000 + 0x8000 = 0x1000\ 8000\ \text{byte.}$

Indirizzo della prima posizione in memoria ($M_{min} = 256\text{Mbyte}$) riservabile ai dati, espressa in funzione di \$gp (base + offset):

Base address: $\$gp = 0x1000\ 0000 + 0x8000 = 0x1000\ 8000\ \text{byte.}$

Offset: -32Kbyte = numero su 16 bit con segno = 0x8000.

$M_{min} = 0x8000(\$gp)$

Ogni altro indirizzo può essere facilmente individuato sommando lo spiazzamento relativo a $M_{min} = 0x1000\ 0000$ e quindi riducendo lo spiazzamenti rispetto a \$gp.

Esempio:

Indirizzo **256,000,016 byte** => 256,032Kbyte - 32Kbyte + 16byte.

In Hex: $0x1000\ 0010 = 0x1000\ 8000 - 0x8000 + 0x10$

A.A. 2007-2008

(la somma viene effettuata modulo 64Kbyte)

<http://homes.dsi.unimi.it/~borghese>



L'assemblatore: i file oggetto



L'assemblaggio produce:

- L'insieme delle istruzioni in linguaggio macchina
- I dati statici
- Le informazioni necessarie per inserire le istruzioni in memoria correttamente.

Un file oggetto è così costituito:

- Header. Posizione e dimensione dei vari pezzi che costituiscono il file oggetto.
- Segmento testo. Contiene le istruzioni.
- Segmento dati statici. Contiene i dati relativi al file oggetto.
- Informazione di rilocazione. Identifica istruzioni, etichette e dati che dipendono dall'indirizzo a partire dal quale viene caricato il programma in memoria.
- La tabella dei simboli. Contiene le etichette che non sono definite (ad esempio riferimenti esterni, di altri moduli oggetto o librerie).
- Informazioni di debug. Consente di associare ai costrutti Assembly i costrutti C (la traduzione non è uno a uno).

A.A. 2007-2008

12/46

<http://homes.dsi.unimi.it/~borghese>



Il linker



Consente di fare ricompilare ed assemblare solo i moduli che vengono modificati (*Rebuild*).

Il linker è costituito da 3 step:

1. Disporre i moduli di codice ed i dati (statici).
2. Identificare gli indirizzi dei dati e delle istruzioni di salto "critiche".
3. Risolvere le etichette interne ai moduli ed esterne (trovare la corrispondenza). Questo passo è equivalente a compilare una tabella di rilocazione.

Nei passi 2 e 3, il linker utilizza le informazioni di rilocazione degli oggetti e le tabelle dei simboli.

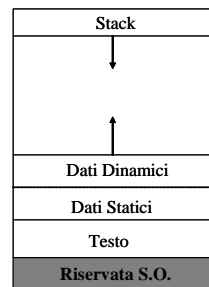
Quali sono i simboli da risolvere?

- Etichette di salto (branch o jump)
- Indirizzo dei dati (e.g. A[0]).

Dopo avere risolto tutte le etichette (sostituito le corrispondenze), occorre trovare gli indirizzi assoluti associati alle etichette.

Vengono cioè **rilocati** (riposizionati) gli oggetti al loro indirizzo finale.

Viene creato il file eseguibile. Ha lo stesso formato del file oggetto ma non ha riferimenti non risolti e gli indirizzi sono assoluti (rilocazione).



Esempio



Analizziamo due procedure:

Proc A

```

0: Proc_A:    lw $a0, 8000($gp)
4:           jal B
8:           add $t2, $t1, $t0
           ...

```

Proc B

```

0: Proc_B:    sw $a1, 8000($gp)
4:           jal A
           ...

```

NB In questo caso \$gp punta a metà del primo segmento di 64Kbyte dell'area dati (256 Mbyte + 32Kbyte)

Header Proc A

Text Size 100hex (=256byte)
Data Size 20hex (=32 byte)

...

Header Proc B

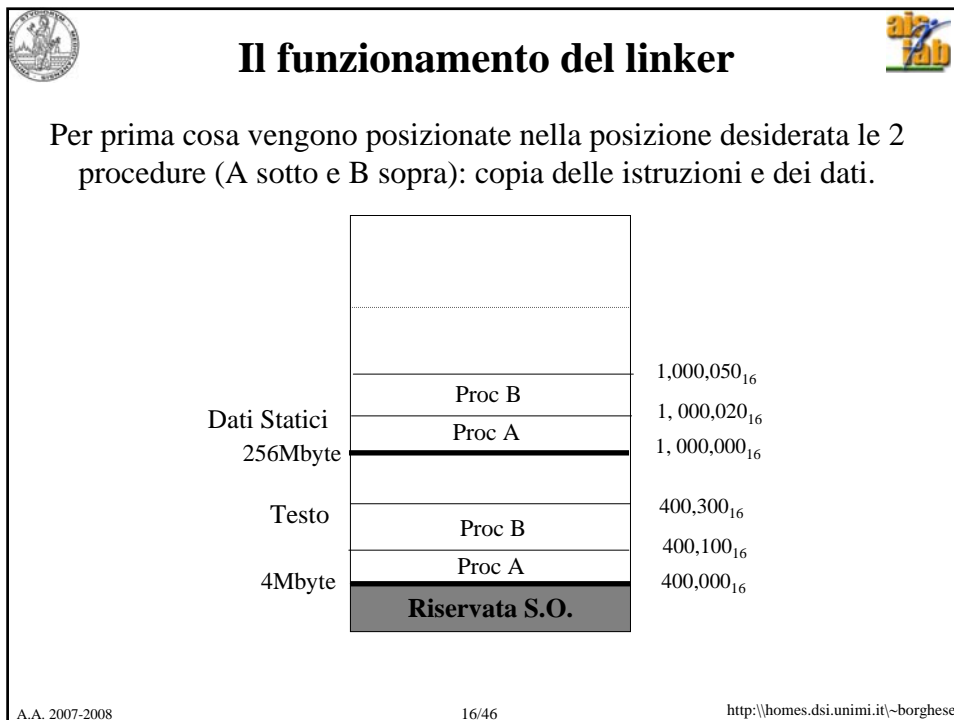
Text Size 200hex (=512byte)
Data Size 32hex (=48byte)

Compilazione

Object file header			
	Name	Procedure A	
	Text Size	100 _{hex} (=256byte)	
	Data Size	20 _{hex} (=32 byte)	
Text Segment	Address	Instruction	
	0	lw \$a0, 0(\$gp)	
	4	jal Proc_B	
		add \$t2, \$t1, \$t0	
	
Data Segment	Address	Instruction	
	0	(X)	
	
Relocation info	Address	Instruction type	Dependency
	0	lw	X
	4	jal	Proc_B
Symbol table	Label	Address	
	X	--	
	Proc_B	--	

Object file header			
	Name	Procedure B	
	Text Size	200 _{hex} (=512byte)	
	Data Size	30 _{hex} (=48 byte)	
Text Segment	Address	Instruction	
	0	sw \$a1, 0(\$gp)	
	4	jal Proc_A	
	
Data Segment	Address	Instruction	
	0	(Y)	
	
Relocation info	Address	Instruction type	Dependency
	0	sw	Y
	4	jal	Proc_A
Symbol table	Label	Address	
	Y	--	
	Proc_A	--	

A.A. 2007-2008 15/46 http://homes.dsi.unimi.it/~borgnese





Calcoli degli indirizzi testo



Segmento testo:

Inizia dopo il segmento riservato al S.O., indirizzo $0x400\ 000 = 0100\ 0000\ 0000\ 0000\ 0000\ 0000$ binario = $1 \times 2^{22} = 4\text{Mbyte}$.

Procedura A. Inizia subito dopo. Indirizzo 0 della procedura è l'indirizzo $0x400\ 000$.

Procedura B. Inizia dopo la procedura A. Indirizzo 0 della procedura B è: $0x400\ 000 + 0x100$ (dimensione della procedura B) = $0x400\ 100$.

Queste osservazioni consentono di sostituire le etichette di salto a procedura (jal).



Risoluzione delle etichette sul codice



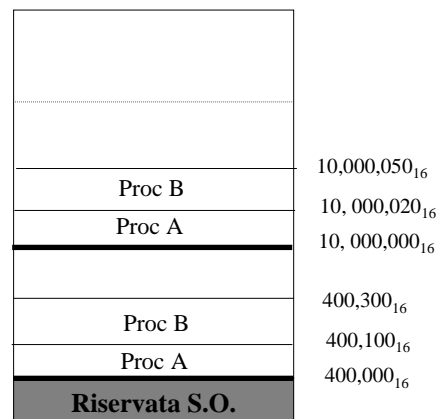
Executable file header		
	Text Size	300_{hex} (=768byte)
	Data Size	50_{hex} (=80 byte)
Text Segment	Address	Instruction
Proc A	$400,000$	lw \$a0, 8000(\$gp)
	$400,004$	jal $400,100$
	$400,008$	add \$t2, \$t1, \$t0

Proc B	$400,100$	sw \$a1, 8000(\$gp)
	$400,104$	jal $400,000$

Data Segment	Address	Data
	0	(X)
	0	(Y)

j, jr si comportano in modo analogo

Come si comportano beq e bne?





Risoluzione delle etichette sui dati



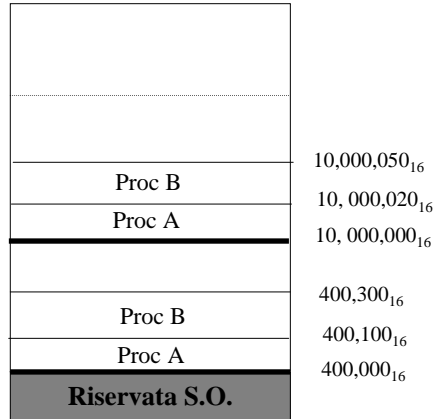
Executable file header		
	Text Size	300 _{hex} (=768byte)
	Data Size	50 _{hex} (=80 byte)
Text Segment	Address	Instruction
Proc A	400,000	lw \$a0, 8000(\$gp)
	400,004	jal 400,100
	400,008	add \$t2, \$t1, \$t0
...
Proc B	400,100	sw \$a1, 8020(\$gp)
	400,104	jal 400,000

Data Segment	Address	
	10,000,000	(X)
	10,000,020	(Y)

$\$gp = 10,008,000_{hex} = 256\text{Mbyte} + 32\text{Kbyte}$

$\$gp = 0001\ 0000\ 0000\ 0000\ 0100\ 0000\ 0000\ 0000$

Il valore di \$gp è comune a tutti i moduli.



Il loader



Dalla memoria su disco alla memoria principale (Unix).

- Lettura dello header del file eseguibile per determinare le dimensioni del segmento testo e dati (statici).
- Creazione di uno spazio in memoria sufficientemente ampio per contenere il testo ed i dati.
- Copia delle istruzioni e dei dati da disco alla memoria. In questa fase agli indirizzi assoluti dell'eseguibile viene aggiunto l'offset assegnato ai segmenti dati e testo per la presenza di altri programmi in esecuzione in memoria.
- Copia dei parametri (che devono essere passati al programma) in cima allo stack (abbassamento dello stack).
- Inizializza i registri della macchina.
- Trasferimento del programma ad una procedura (di sistema) che trasferisce i parametri dallo stack ai registri argomento e chiama (salta) alla prima istruzione della procedura main.
- Al termine del programma verrà eseguita una syscall (exit).



I problemi del linker



Viene creato un unico file eseguibile che contiene:

Le funzioni di libreria vengono inserite all'interno dell'eseguibile.

Tutte i moduli "linkati" vengono inseriti.

Il file eseguibile diventa molto grosso (static link).

Soluzione DLL (dynamic link).



Sommario



Dal linguaggio ad alto livello al codice in memoria

Lo SPIM e gli elementi di un programma Assembly

Esempi di programmi Assembly: costanti e ricorsione



Il simulatore SPIM del MIPS



• **SPIM: A MIPS R2000/R3000 Simulator : PCSPIM version 7.1** scritto da James Larus.

• <http://www.cs.wisc.edu/~larus/spim.html>

• Piattaforme:

- Unix or Linux system
- Microsoft Windows (Windows 98, NT, 2000, XP)
- Microsoft DOS



SPIM interface



The screenshot shows the PCSPIM simulator window with the following content:

```

PC = 00000000  EPC = 00000000  Cause = 00000000  BadVAddr= 00000000
Status = 00000000  HI = 00000000  LO = 00000000

General Registers
R0 (r0) = 00000000  R8 (t0) = 00000000  R16 (s0) = 00000000  R24 (t8) = 00000000
R1 (a1) = 00000000  R9 (t1) = 00000000  R17 (s1) = 00000000  R25 (t9) = 00000000
R2 (v0) = 00000000  R10 (t2) = 00000000  R18 (s2) = 00000000  R26 (k0) = 00000000
R3 (v1) = 00000000  R11 (t3) = 00000000  R19 (s3) = 00000000  R27 (k1) = 00000000
R4 (a0) = 00000000  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 10008000

[0x00400000] 0x34090002 ori $9, $0, 2           ; 13: li   $t1, 2 # N interi di cui calcolare la s
[0x00400004] 0x340e0000 ori $14, $0, 0        ; 15: li   $t6, 0 # t6 e' indice di ciclo indice c
[0x00400008] 0x34180000 ori $24, $0, 0        ; 16: li   $t8, 0 # t8 contiene la somma dei quad
[0x0040000c] 0x01e00118 mult $14, $14         ; 18: mul  $t7, $t6 $t6 # t7 = t6 x t6
[0x00400010] 0x00007812 mflo $15
[0x00400014] 0x030fc021 addu $24, $24, $15    ; 19: addu $t8, $t8, $t7 # t9 = t8 + t7
[0x00400018] 0x21ce0001 addi $14, $14, 1     ; 20: addi $t6, $t6, 1
[0x0040001c] 0x012e082e slt $1, $9, $14      ; 21: ble  $t6, $t1, Loop # if t6 < N stay in loop

DATA
[0x10000000]...[0x1000ffff] 0x00000000
[0x1000ffff] 0x00000000
[0x10010000] 0x7320614c 0x20616d6f 0x30206164 0x4e206120
[0x10010010] 0x6c617620 0x00002065 0x00000000 0x00000000
[0x10010020]...[0x10040000] 0x00000000

All Rights Reserved.
DOS and Windows ports by David A. Carley (dac@cs.wisc.edu).
Copyright 1997 by Morgan Kaufmann Publishers, Inc.
See the file README for a full copyright notice.
Memory and registers have been cleared, and the simulator reinitialized.
C:\_WWW_page\Borghese\Teaching\Architettura\Asm\squared.asm has been successfully loaded

For Help, press F1
PC=0x00000000 EPC=0x00000000 Cause=0x00000000
  
```



System call



- Sono disponibili delle **chiamate di sistema (system call)** predefinite che implementano particolari servizi (ad esempio: stampa a video)
- Ogni system call ha:
 - un codice
 - degli argomenti (opzionali)
 - dei valori di ritorno (opzionali)

CodOp	rs	rt	rd	shamt	funct
0	0	0	0	0	0xC
6 bit					6 bit

Il codice della funzione di sistema richiesta è memorizzato **sempre** nel registro **\$v0**.
Le syscall hanno tutte la stessa stringa in linguaggio macchina; il sistema operativo capisce cosa deve fare dal valore di \$v0.

A.A. 2007-2008

25/46

<http://homes.dsi.unimi.it/~borgnese>



System call



Nome	Codice (\$v0)	Argomenti	Risultato
print_int	1	\$a0	
print_float	2	\$f12	
print_double	3	\$f12	
print_string	4	\$a0	
read_int	5		\$v0
read_float	6		\$f0
read_double	7		\$f0
read_string	8	\$a0,\$a1	
sbrk	9	\$a0	\$v0
exit	10		

- Per richiedere un servizio ad una chiamata di sistema (**syscall**) occorre:
 - Caricare il **codice** della **syscall** nel registro **\$v0**
 - Caricare gli **argomenti** nei registri **\$a0 - \$a3** (oppure nei registri **\$f12 - \$f15** nel caso di valori in virgola mobile)
 - Eseguire **syscall**
 - L'eventuale **valore di ritorno** è caricato nel registro **\$v0 (\$f0)**

A.A. 2007-2008

26/46

<http://homes.dsi.unimi.it/~borgnese>



System call



- **print_int**: stampa sulla console il numero intero che le viene passato come argomento;
- **print_float**: stampa sulla console il numero in virgola mobile con singola precisione che le viene passato come argomento;
- **print_double**: stampa sulla console il numero in virgola mobile con doppia precisione che le viene passato come argomento;
- **print_string**: stampa sulla console la stringa che le è stata passata come argomento terminandola con il carattere **Null**;
- **read_int**: legge una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati);
- **read_float**: leggono una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati);
- **read_double**: leggono una linea in ingresso fino al carattere a capo incluso (i caratteri che seguono il numero sono ignorati);
- **read_string**: legge una stringa di caratteri di lunghezza **\$a1** da una linea in ingresso scrivendoli in un buffer (**\$a0**) e terminando la stringa con il carattere **Null** (se ci sono meno caratteri sulla linea corrente, li legge fino al carattere a capo incluso e termina la stringa con il carattere **Null**);
- **sbrk** restituisce il puntatore (indirizzo) ad un blocco di memoria;
- **exit** interrompe l'esecuzione di un programma;

A.A. 2007-2008

27/46

<http://homes.dsi.unimi.it/~borgnese>



Direttive



- Le direttive (data layout directives) danno delle indicazioni all'assemblatore sul contenuto di un file (istruzioni, strutture dati, ecc.)
- Sintatticamente le direttive iniziano tutte con il carattere ":"

I segmenti

.data <addr>

Gli elementi successivi sono memorizzati nel segmento dati a partire dall'indirizzo **addr**, **facoltativo**

.text <addr>

Memorizza gli elementi successivi nel segmento testo dell'utente a partire dall'indirizzo **addr**. (Questi elementi possono essere **solo istruzioni** o **parole**).

A.A. 2007-2008

28/46

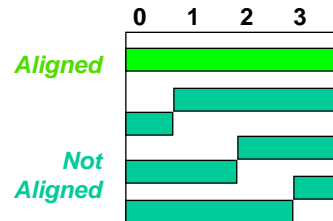
<http://homes.dsi.unimi.it/~borgnese>



Addressing Objects: Alignment



Alignment: require that objects fall on address that is multiple of their size.



In MIPS è opportuno che ogni parola (word) inizi ad un indirizzo multiplo di 4.
Questo è sempre vero per le istruzioni, può essere desiderabile per i dati.



Direttive per il segmento dati (.data)



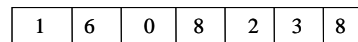
.byte b1,...,bn

Memorizza gli **n** valori **b1, ..., bn** in byte consecutivi di memoria

.word w1, ...,wn

Memorizza gli **n** valori su 32-bit **w1, ..., wn** in parole consecutive di memoria.

Esempio: .word 1, 6, 0, 8, 2, 3, 8



←→

1 word

↑
Successivo
inserimento

.half h1, ...,hn

Memorizza gli **n** valori su 16-bit **h1, ..., hn** in halfword (mezze parole) consecutive di memoria

.ascii str

Memorizza la stringa **str** terminandola con il carattere **Null** (**.ascii str** ha lo stesso effetto, ma non aggiunge alla fine il carattere **Null**)

.space n

Alloca uno spazio pari ad **n** byte nel segmento dati



Rappresentazione di un vettore in memoria



Etichetta + tipo
array + .word

```
array: .word 1, 6, 0, 8, 2, 3, 8
```

1	6	0	8	2	3	8
---	---	---	---	---	---	---



1 word

⇒ array rappresenta l'indirizzo del primo elemento

Per indirizzare gli elementi dell'array utilizzo l'istruzione:
la \$t0, array #Carico in \$t0 l'indirizzo di array[0]

lw \$s0, 0(\$t0) -> carico il primo elemento del vettore

lw \$s0, 24(\$t0) -> carico l'ultimo elemento del vettore

lw \$s0, 28(\$t0) -> carico la prima parola memorizzata dopo il vettore (il risultato è imprevedibile, può essere un errore di sistema (se la parola appartiene ad un altro programma, un numero senza significato...))



Allineamento dei dati (.data)



.align n

Allinea il dato successivo a blocchi di 2^n byte: ad esempio

- **.align 2 = .word** allinea alla parola il valore successivo
- **.align 1 = .half** allinea alla mezza parola il valore successivo
- **.align 0** elimina l'allineamento automatico delle direttive **.half**, **.word**, **.float**, e **.double** fino a quando compare la successiva direttiva **.data**



Direttive per il segmento testo (.text)



.globl sym

Dichiara **sym** come etichetta globale (ad essa è possibile fare riferimento da altri file). Tipicamente si utilizza per la procedura main (`.globl main`).

I simboli costituiscono delle etichette. Esempio:

0x400000 Proc_A:

e associano ad un indirizzo numerico (della memoria) un codice simbolico.



Sommario



Dal linguaggio ad alto livello al linguaggio Assembly

Lo SPIM e gli elementi di un programma Assembly

Esempi di programmi Assembly: costanti e ricorsione



Procedure ricorsive



- Procedure che contengono una chiamata a se stesse al loro interno => il codice della procedura viene riutilizzato più volte, ogni volta con parametri diversi.

Calcolo del fattoriale di un numero intero

```
main(int argc, char *argv[])
{
    int n;
    printf("Inserire un numero intero\n");
    scanf("%d", &n);
    printf("Fattoriale: %d\n", fact(n));
}

int fact(int m)
{
    if (m <= 1)
        return(1);
    else
        return(m*fact(m-1));
}
```

A.A. 2007-2008

35/46

<http://homes.dsi.unimi.it/~borgnese>



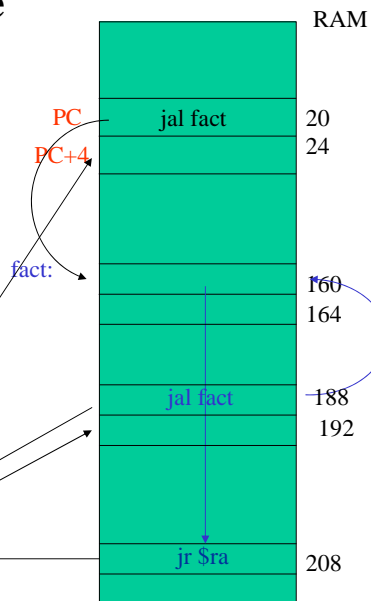
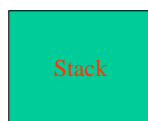
Strutture interessate dalle procedure ricorsive



Riceve in ingresso N in a0.

```
160: fact: addi $sp, $sp, -8
164:      sw $a0, 0($sp)
168:      sw $ra, 4($sp)
172:      sli $t0, $a0, 2
          beq $t0, $zero, ric #if (a0 > 1) continua
                               # la ricorsione

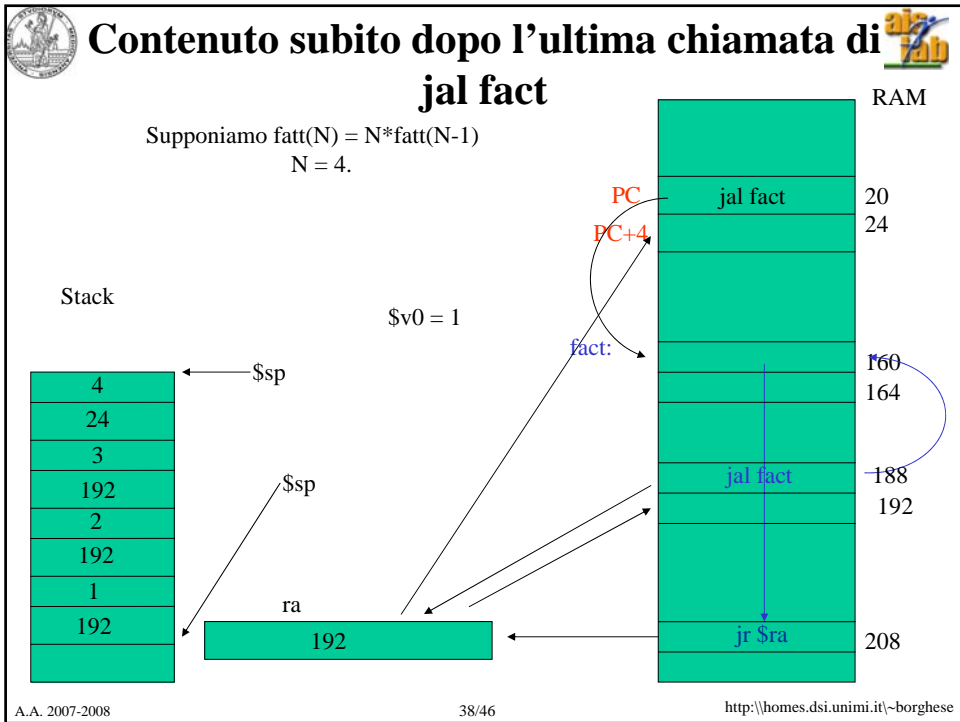
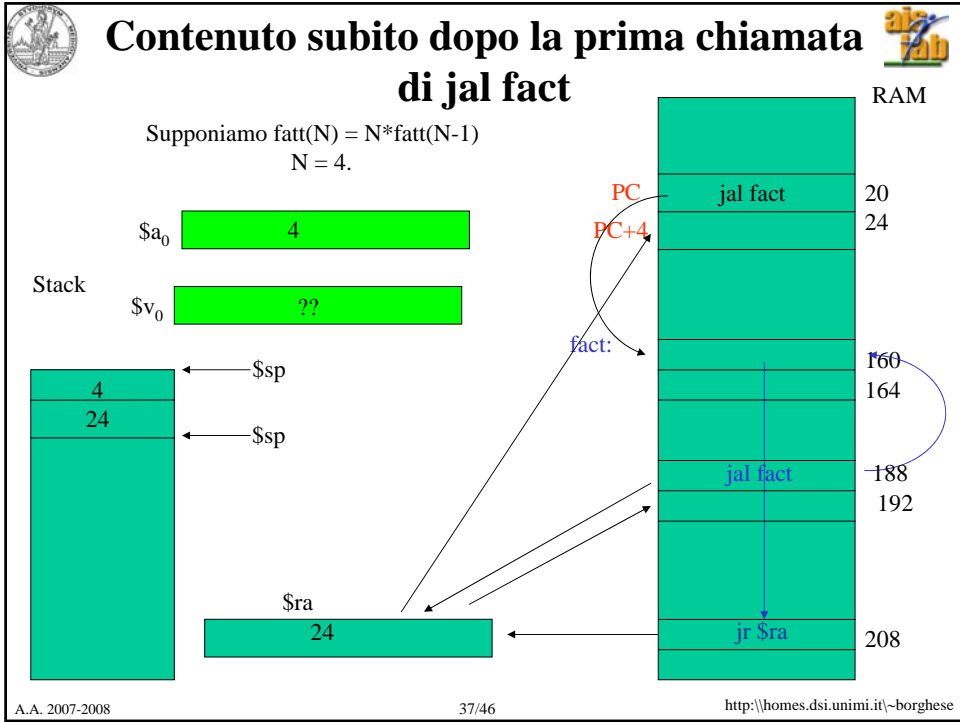
176:      li $v0, 1
180:      j end
184: ric: addi $a0, $a0, -1
188:      jal fact
192:      lw $a0, 0($sp)
196: end: mul $v0, $v0, $a0
200:      lw $ra, 4($sp)
204:      addi $sp, $sp, 8
208:      jr $ra
```



A.A. 2007-2008

36/46

<http://homes.dsi.unimi.it/~borgnese>





Esempio, programma per il calcolo del fattoriale in Assembly – parte dichiarativa



```
# Programma che calcola il fattoriale iterativamente
.data
prompt: .asciiz "Inserisci un numero intero"
output: .ascii "Il fattoriale è:"
.text
.globl main
main:
    li $v0, 4          # $v0 ← codice della print_string
    la $a0, prompt    # $a0 ← indirizzo della stringa
    syscall           # stampa la stringa

    li $v0, 5          # $v0 ← codice della read_int
    syscall           # legge l'intero e lo carica in
    $v0

    move $s0, $v0     # Per pulizia, porta l'intero in
                    # un registro di varibili
```

A.A. 2007-2008

39/46

<http://homes.dsi.unimi.it/~borgnese>



Esempio, fattoriale – stampa risultato



```
# stampa il risultato

    move $a0, $v0     # $a0 ← $v0
    li $v0, 1         # $v0 ← codice della print_int
    syscall           # stampa l'intero in input

    li $v0, 4         # $v0 ← codice della print_string
    la $a0, output    # $a0 ← indirizzo della stringa
    syscall           # stampa la stringa

    li $v0, 1         # $v0 ← codice della print_int
    move $a0, $t1     # $a0 ← n!
    syscall           # stampa n!

    li $v0, 10        # $v0 ← codice della exit
    syscall           # esce dal programma
```

A.A. 2007-2008

40/46

<http://homes.dsi.unimi.it/~borgnese>



Programma di caricamento di costanti



```
# Somma

.text          #Definizione segmento codice
.globl main    #Definizione del main

main:
    li $t1,10  # carica il valore decimale 10 nel reg. $t1
    li $t2,15  # carica il valore decimale 15 nel reg. $t2

    add $a0,$t2,$t1 # $a0 ← $t2 + $t1

print_result:
    li $v0, 1 # stampa risultato (10 + 15 = 25)
    syscall
```



Programma di somma di costanti e variabili



```
# L'accesso immediato è usato anche dalle operazioni
# aritmetiche

.text          #Definizione segmento codice
.globl main

main:
    li $t1,10  # $t1 ← 10
    addi $a0,$t1,15 # $a0 ← $t1 + 15

# stampa risultato
print_result:
    li $v0,1
    syscall
```



Programma di stampa



```
#Programma che stampa: la risposta è 5
.data
str: .asciiz "la risposta è "
.text
.globl main

Main:
li $v0, 4          # $v0 ← codice della print_string
la $a0, str        # $a0 ← indirizzo della stringa
syscall           # stampa della stringa

li $v0, 1          # $v0 ← codice della print_integer
li $a0, 5          # $a0 ← intero da stampare
syscall           # stampa dell'intero

li $v0, 10         # $v0 ← codice della exit
syscall           # esce dal programma
```

A.A. 2007-2008

43/46

<http://homes.dsi.unimi.it/~borgnese>



Programma di lettura scrittura di numeri



```
#Programma che legge e stampa un intero

.data
prompt:.asciiz "Dammi un intero: "

.text
.globl main

main:
li $v0, 4          # $v0 ← codice della print_string
la $a0, prompt     # $a0 ← indirizzo della stringa
syscall           # stampa la stringa

li $v0, 5          # $v0 ← codice della read_int
syscall           # legge un intero e lo carica in $v0

li $v0, 10         # $v0 ← codice della exit
syscall           # esce dal programma
```

A.A. 2007-2008

44/46

<http://homes.dsi.unimi.it/~borgnese>



Programma che somma i quadrati dei primi N numeri



```
# N e' memorizzato in t1.

.data
str:      .asciiz "La soma da 0 a N vale "

.text
.globl main

main:

    li     $t1, 7      # N=7, interi di cui calcolare la somma dei quadrati
    li     $t6, 0      # t6 e' indice di ciclo
    li     $t8, 0      # t8 contiene la somma dei quadrati

Loop:
    mult   $t7, $t6, $t6    # t7 = t6 x t6
    addu   $t8, $t8, $t7    # t8 = t8 + t7
    addi   $t6, $t6, 1      # t6++
    blt    $t6, $t1, Loop   # if t6 < N stay in loop

    la     $a0, str
    li     $v0, 4           #print
    syscall

    li     $v0, 1           #print
    add    $a0, $t8, $zero
    syscall

    li     $v0, 10          # $v0 codice della exit
    syscall                 # esce dal programma
```



Sommario



Dal linguaggio ad alto livello al linguaggio Assembly

Lo SPIM e gli elementi di un programma Assembly

Esempi di programmi Assembly: costanti e ricorsione