



Dal sorgente all'eseguibile I programmi Assembly

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
alberto.borghese@unimi.it

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



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

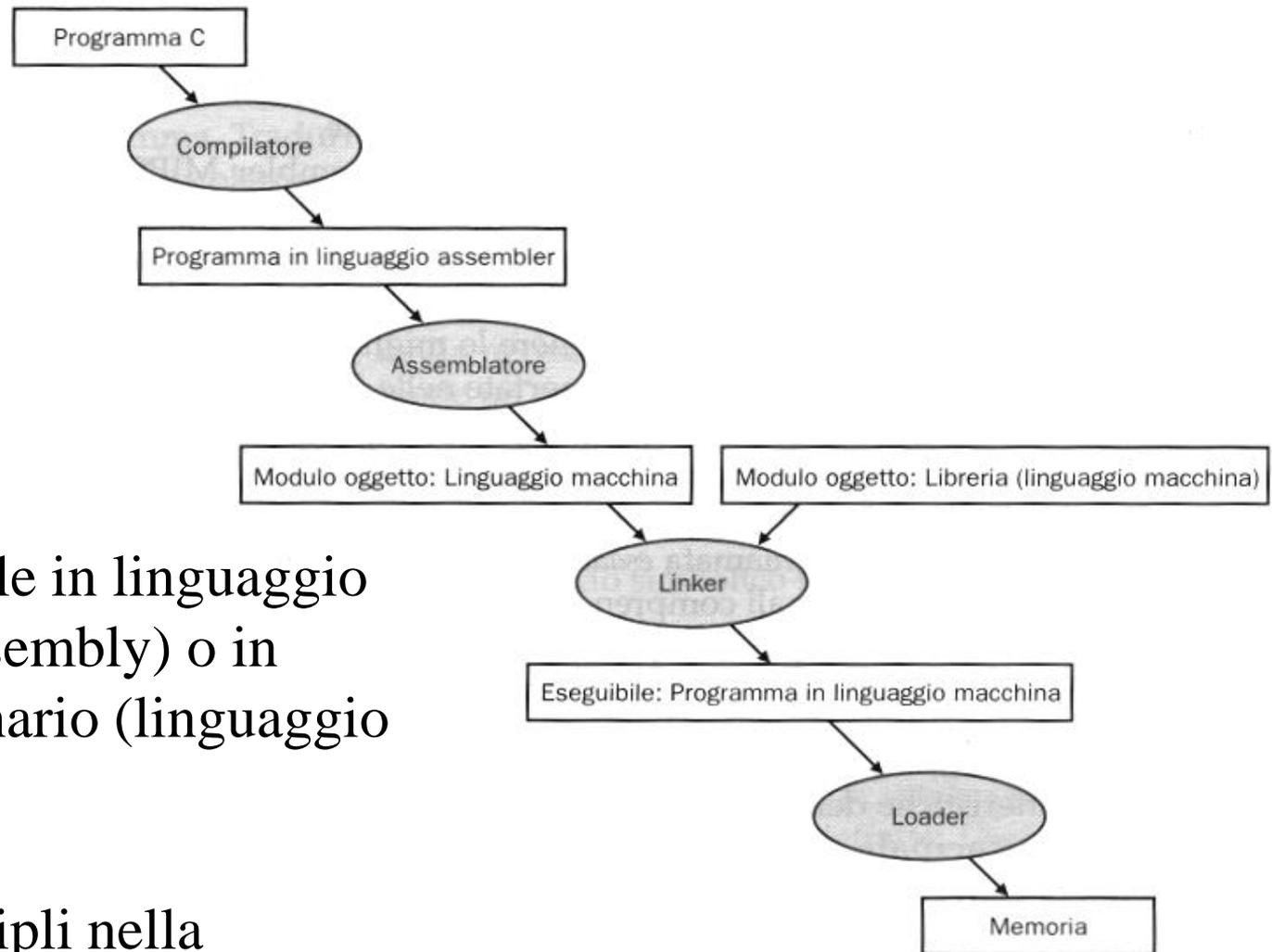
Devono contenere tutte le informazioni necessarie ad eseguire il ciclo di esecuzione dell'istruzione. Registri, comandi,

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

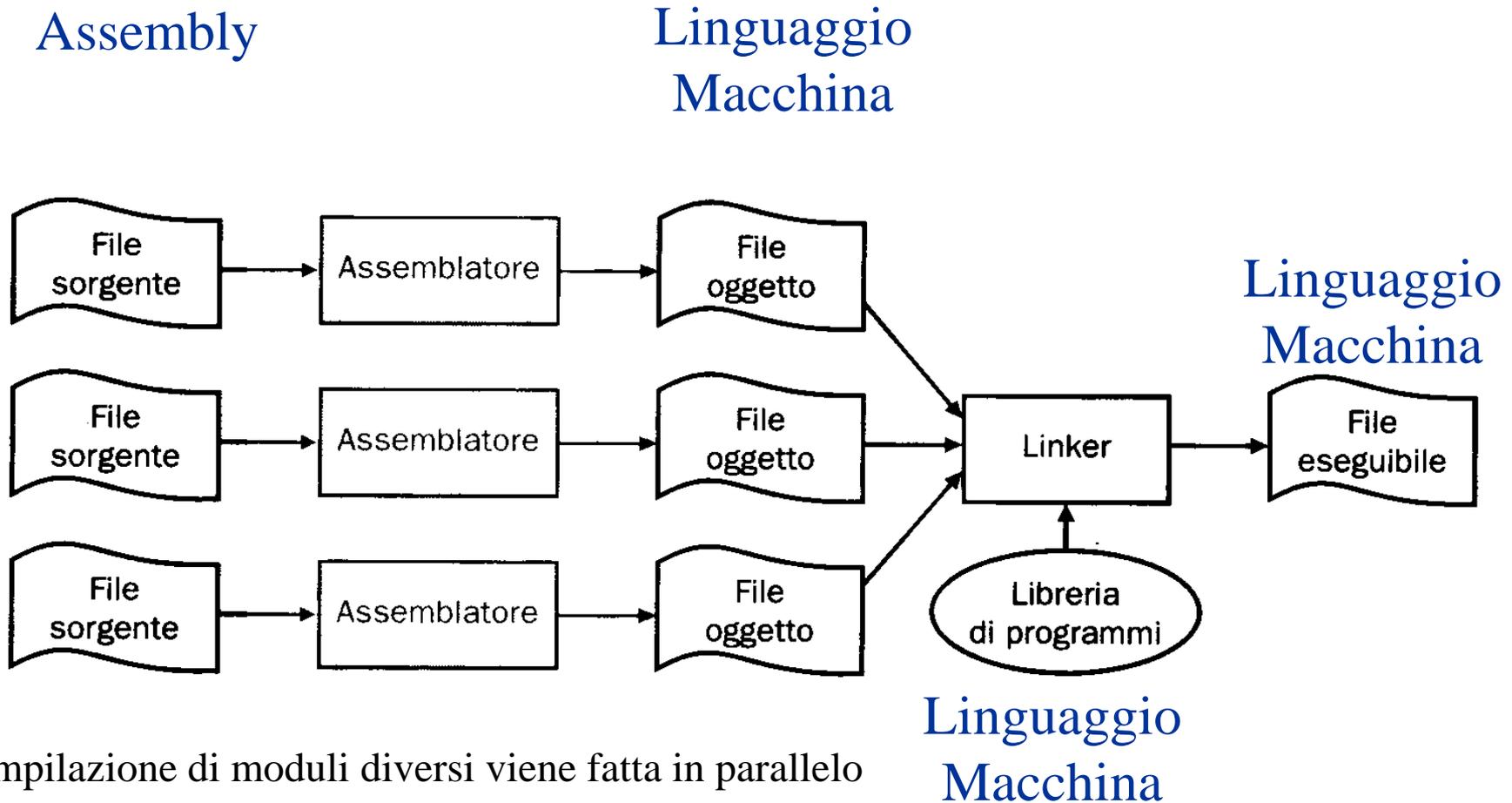


ISA esprimibile in linguaggio simbolico (assembly) o in linguaggio binario (linguaggio macchina).

Passaggi multipli nella traduzione.

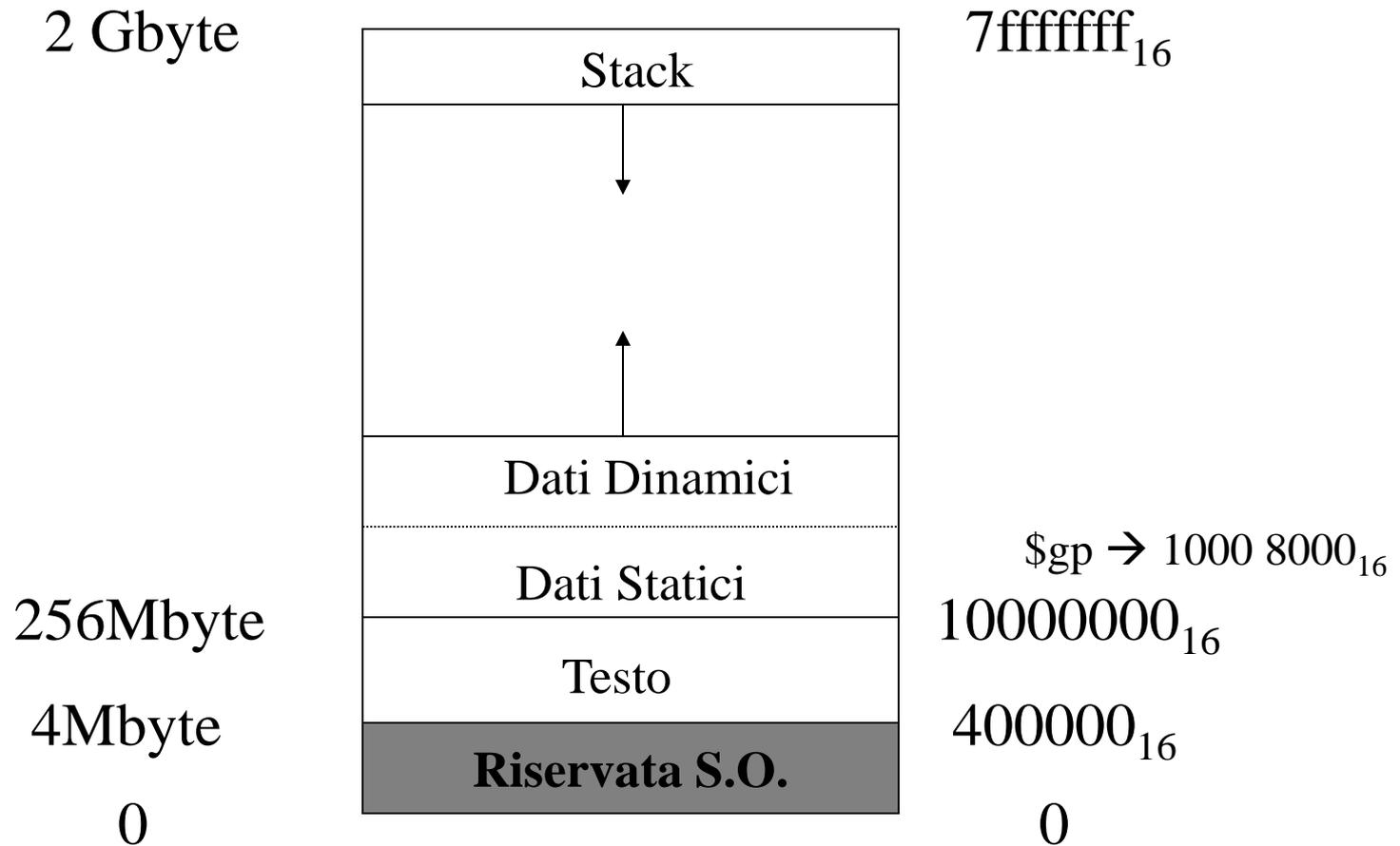


Dall'assembly all'eseguibile





Organizzazione logica della memoria





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 $+2^{15} - 1$).



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

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

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

Base address: $\$gp = 0x1000\ 0000 + 0x8000 = 0x1000\ 8000$ 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: $0x1000\ 0010 = 0x1000\ 8000 - 0x8000 + 0x10$



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.



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).



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 rilocalizzazione.

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

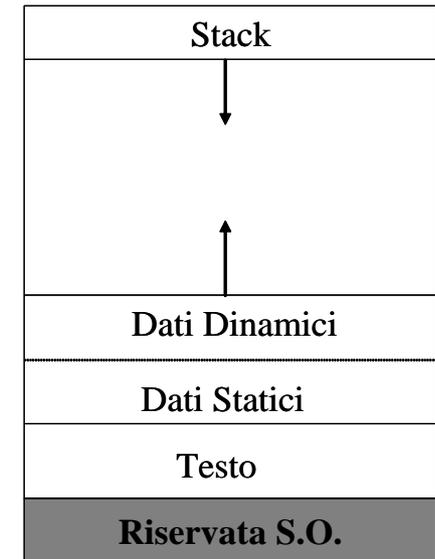
Quali sono i simboli da risolvere?

- 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 (rilocalizzazione).





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)

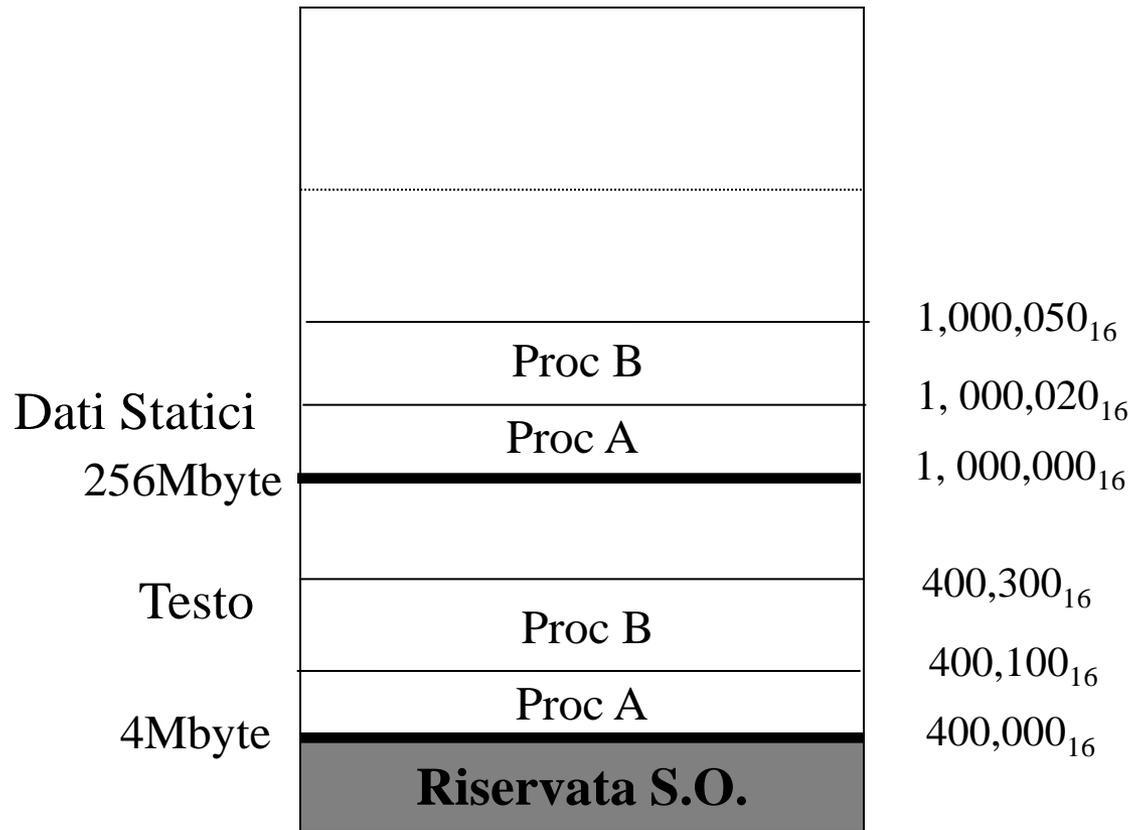
Object file header			
	Name	Procedure A	
	Text Size	100 _{hex} (=256byte)	
	Data Size	20 _{hex} (=32 byte)	
Text Segment	Address	Instruction	
	0	lw \$a0, 0(\$gp)	
	4	jal Proc_B	
		add \$t2, \$t1, \$t0	
	
Data Segment	0	(X)	
	
Relocation info	Address	Instruction type	Dependency
	0	lw	X
	4	jal	Proc_B
Symbol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



Il funzionamento del linker

Per prima cosa vengono posizionate nella posizione desiderata le 2 procedure (A sotto e B sopra): copia delle istruzioni e dei dati.





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



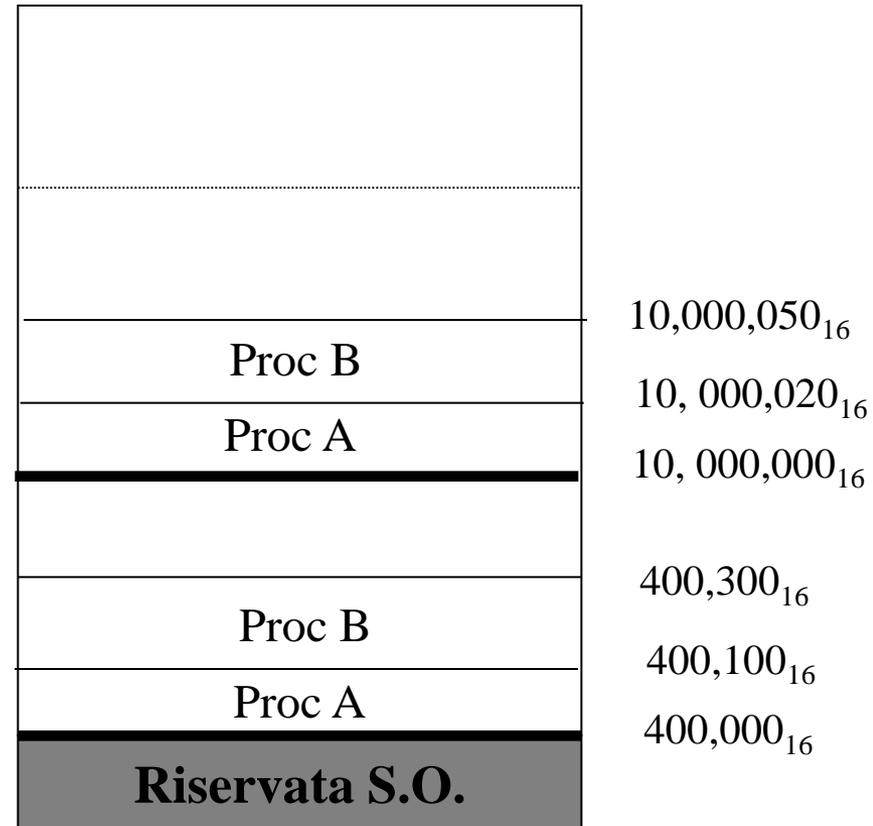
Executable file header		
	Text Size	300 _{hex} (=768byte)
	Data Size	50 _{hex} (=80 byte)
Text Segment	Address	Instruction
Proc A	400,000	lw \$a0, 8000(\$gp)
	400,004	jal 400,100
	400,008	add \$t2, \$t1, \$t0

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

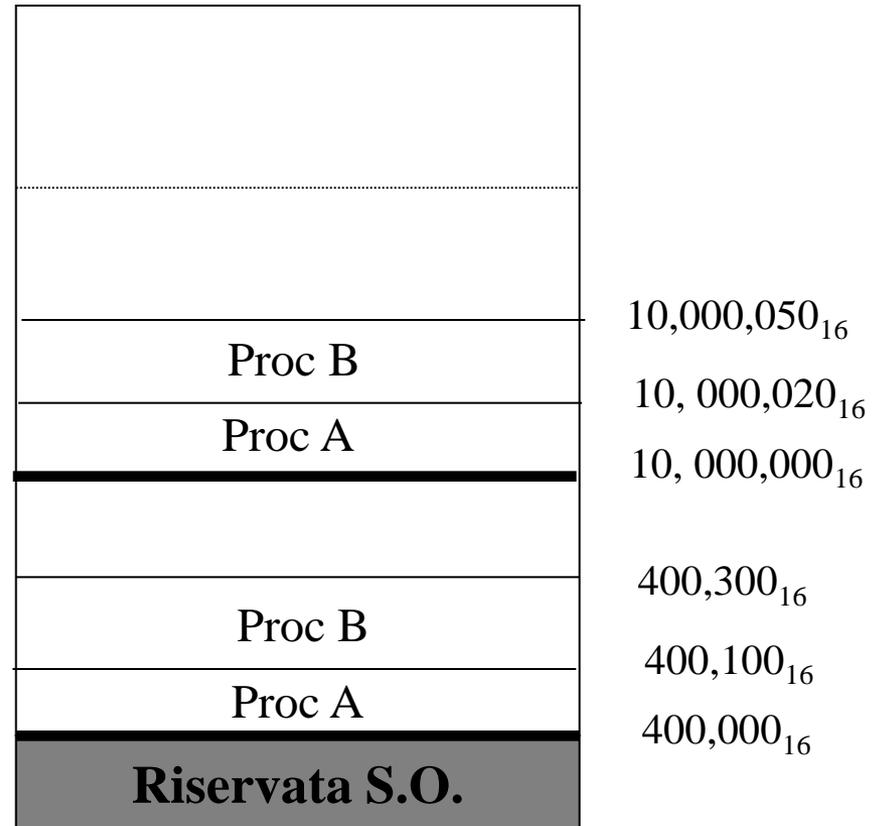
Proc B	400,100	sw \$a1, 8020(\$gp)
	400,104	jal 400,000

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

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

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

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.

Tutti i moduli "linkati" vengono inseriti.

Il file eseguibile diventa molto grosso (static link).

Soluzione DLL (dynamic link).