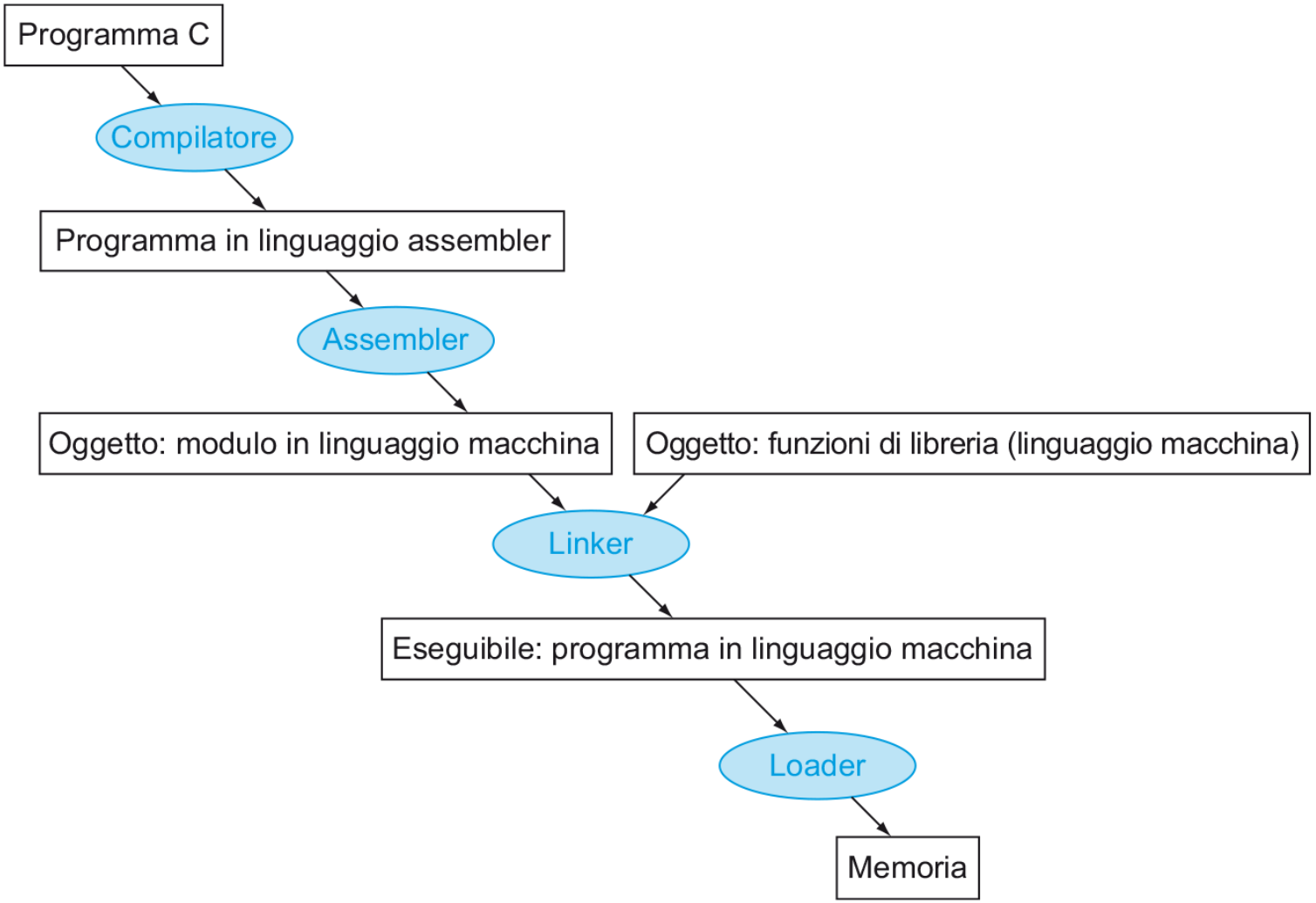
POLITECNICO
MILANO 1863

architettura dei calcolatori e sistemi operativi

assemblatore e collegatore (linker)
RISC V – capitolo 2 P&H

flusso di compilazione e assemblaggio

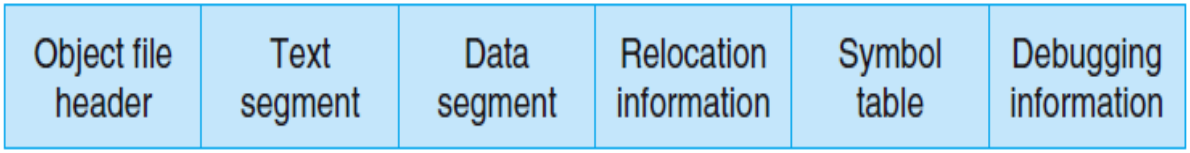


processo di assemblaggio

- si applica al programma modulo per modulo e per ciascun ***modulo (file) sorgente*** costruisce il corrispondente ***modulo (file) oggetto***
- esamina, riga per riga, il codice sorgente assembler di un modulo e ne traduce le istruzioni simboliche nel corrispondente ***formato di linguaggio macchina***
  - i codici mnemonici sono tradotti nei corrispondenti codici binari
  - i riferimenti ai registri nei corrispondenti “numeri” di registro
  - i ***riferimenti simbolici*** del modulo (identificatori di variabili, etichette di salto e nomi di funzioni) sono tradotti – se possibile – negli indirizzi binari corrispondenti; per questa operazione l’assemblatore genera la ***tabella dei simboli*** del modulo
- il segmento di ciascun modulo è assemblato in termini di indirizzi virtuali rilocabili (tutti i segmenti partono da indirizzo 0)

assemblaggio

1. traduzione delle pseudoistruzioni in istruzioni RISC V
   - p.es. pseudoistruzione **`move rd,rs1`** viene tradotta in **`addi rd, rs1, 0`**
   - p.es. pseudoistruzione **`bgt rs1, rs2, spi12`** (*branch greater than*) in **`blt rs2, rs1, spi12`**
2. definizione degli indirizzi corrispondenti a tutte le etichette e inserimento nella tabella dei simboli (una riga per ciascun simbolo a cui corrisponde un indirizzo)
   - possono rimanere dei riferimenti non risolti (definiti in un altro file)
3. generazione del file oggetto



assemblatore – traduzione delle etichette

```
.text        # segmento testo (codice)

.globl MAIN # simbolo globale

MAIN: addi sp, sp, -64
      sd   ra, 24(sp)
      sd   a0, 48(sp)
      sd   zero, 32(sp)
      sd   zero, 40(sp)
LOOP: ld   t6, 40(sp)
      sub  t3, t6, t5
      ld   t6, 32(sp)
      add  t6, t5, t4
      sd   t6, 32(sp)
      addi t0, t6, 1
      sd   t0, 240(sp)
      li   t1, 100
      ble  t0, t1, loop
      la   a2, STR
      ld   a3, 32(sp)
      jal  PRINTF
      move a0, zero
      ld   ra, 24(sp)
      addi sp, sp, 64
      ret

.data        # segmento dati

.align 0     # allinea a byte
STR: .asciiz "Sum from 0 .. 100 is %d\n"
```

etichetta locale: **LOOP** e **STR**, visibili solo in questo file (modulo)
etichetta globale (esterna): **MAIN**, visibile all'esterno
riferimento non risolto: **PRINTF** (procedura di libreria per stampare a terminale)

nota: l'etichetta **STR** viene utilizzata (citata) prima di essere definita

l'assemblatore lavora in due passi:

1. individua tutte le etichette e le inserisce nella tabella dei simboli
2. traduce le istruzioni assembler in linguaggio macchina utilizzando le informazioni nella tabella dei simboli e considerando che il programma inizia all'indirizzo 0

processo di assemblaggio – primo passo

nel primo passo l'assemblatore non traduce nessuna istruzione ma costruisce la ***tabella dei simboli del modulo***

i riferimenti simbolici inseriti nella tabella possono essere

- simboli associati a direttive dell'assemblatore che definiscono costanti simboliche (.***eqv***) – nella tabella si crea la coppia <*simbolo*, *valore*>
- etichette che definiscono variabili del segmento dati – nella tabella si crea la coppia <*simbolo*, *indirizzo*>
- etichette che contrassegnano istruzioni destinazioni di salto – nella tabella si crea la coppia <*simbolo*, *indirizzo*>

i ***valori degli indirizzi*** inseriti sono quelli ***rilocabili*** rispetto al segmento considerato (segmento testo T o segmento dati D)

processo di assemblaggio – secondo passo

è la fase di traduzione vera e propria: usa la tabella dei simboli del modulo e genera – oltre alla traduzione – la ***tabella di rilocazione*** del modulo

un'istruzione ***è tradotta in modo «incompleto»***, e deve quindi essere elaborata anche da parte del collegatore (*linker*), se

- il riferimento simbolico presente in essa è relativo a variabili del segmento ***.data***; p.es. la pseudoistruzione **`la`** viene espansa tramite **`auipc`** e **`addi`** con immediati convenzionali a **`0`**, che andranno poi calcolati da collegatore
- il riferimento è relativo a simboli non (ancora) presenti nella tabella dei simboli del modulo: il simbolo viene posto a **`0`** per convenzione e andrà poi calcolato
- il riferimento simbolico presente in essa è relativo a simboli su cui agisce un *modificatore*; p.es. la pseudostruzione **`li`** con il valore espresso tramite un'etichetta, viene espansa tramite **`lui`** e **`addi`** con immediati convenzionali a **`0`**, che andranno poi calcolati

in corrispondenza di ogni traduzione *incompleta* viene creato un elemento nella tabella di rilocazione, nella forma

< indirizzo rilocabile istruzione, codice op. istruzione, simbolo da risolvere (con modificatore)>

esempio

sorgente procedura B:

```
.data
Y:   .dword 0
.text
B:   bne  a2, zero, E
     la   t0, Y
     sd   a1, 0(t0)
E:   li   t1, W
```

per l'istruzione di salto in formato B che ha simbolo locale al modulo, il simbolo viene risolto subito come
(VS_REL – IADDR_REL) / 2

nell'esempio
bne a2, zero, E

il simbolo **E** viene tradotto (hex) come (10 – 0) / 2 = **8**

oggetto procedura B:
(passo 1 assemblatore)

dimensione testo: ----
dimensione dati: ----
testo:

```
0    bne
4    auipc      la
8    addi
C    sd
10   lui        li
14   addi
```

dati:

```
0    0
```

tabella dei simboli:

```
B    0     T
E    10    T
Y    0     D
```

oggetto procedura B:
(passo 2 assemblatore)

dimensione testo: 0x 18
dimensione dati: 0x 8
testo:

```
0    bne    a2, zero, 8
4    auipc  t0, 0 0 0 0 0
8    addi   t0, t0, 0 0 0
C    sd     a1, 0(t0)
10   lui    t1, 0 0 0 0 0
14   addi   t1, t1, 0 0 0
```

dati:

```
0    0
```

tabella dei simboli:

```
B    0     T
E    10    T
Y    0     D
```

tabella di rilocazione:

```
4    auipc  %pcrel_hi (Y)
8    addi   %pcrel_lo (Y)
10   lui    %hi (W)
14   addi   %lo (W)
```

formato oggetto

- **intestazione**, che descrive le dimensioni del testo e dei dati del modulo
- **segmento testo**, che contiene il codice in linguaggio macchina delle procedure del file sorgente; queste procedure potrebbero essere non eseguibili a causa di riferimenti non risolti
- **segmento dati**, che contiene una rappresentazione binaria dei dati definiti nel file sorgente; anche i dati potrebbero essere incompleti, a causa di riferimenti non risolti a etichette definite in altri file
- **informazioni di rilocazione**, che identificano le istruzioni e le parole di dati che dipendono da **indirizzi assoluti**; la posizione di queste istruzioni e dati deve essere modificata se parti del programma vengono spostate in memoria
- **tabella dei simboli**, che associa un indirizzo alle etichette esterne contenute nel file sorgente e contiene l'elenco dei riferimenti non risolti
- **informazioni di debug**, che non consideriamo ulteriormente

il collegatore *mette insieme* i diversi moduli oggetto che vanno a costituire l'eseguibile

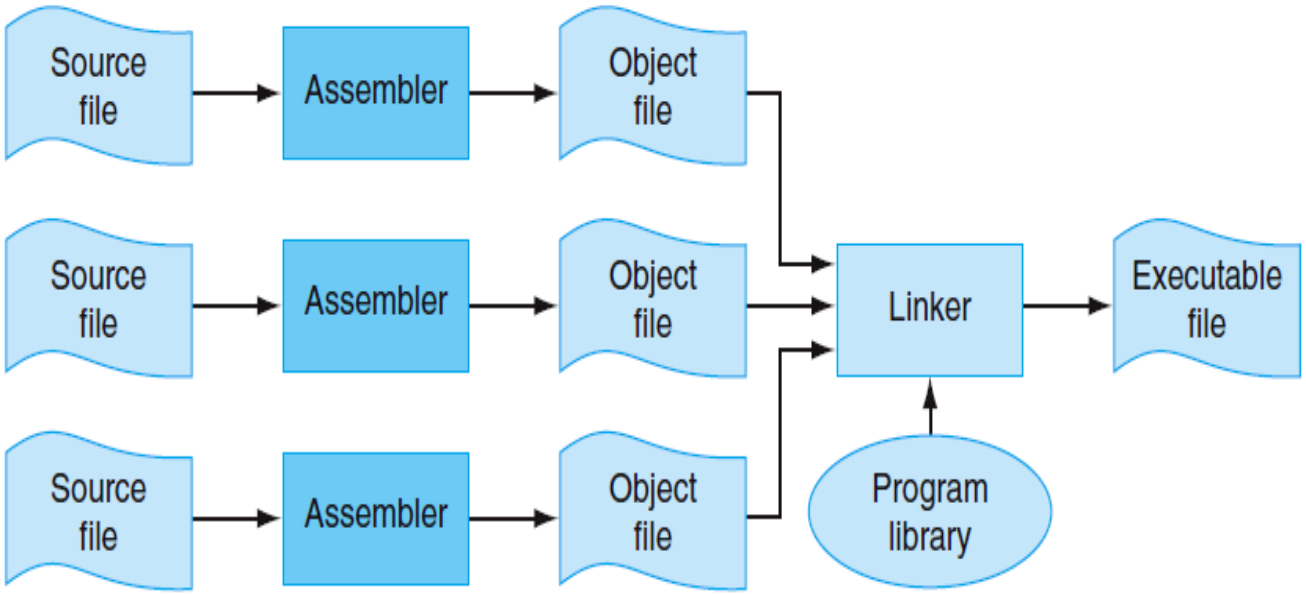
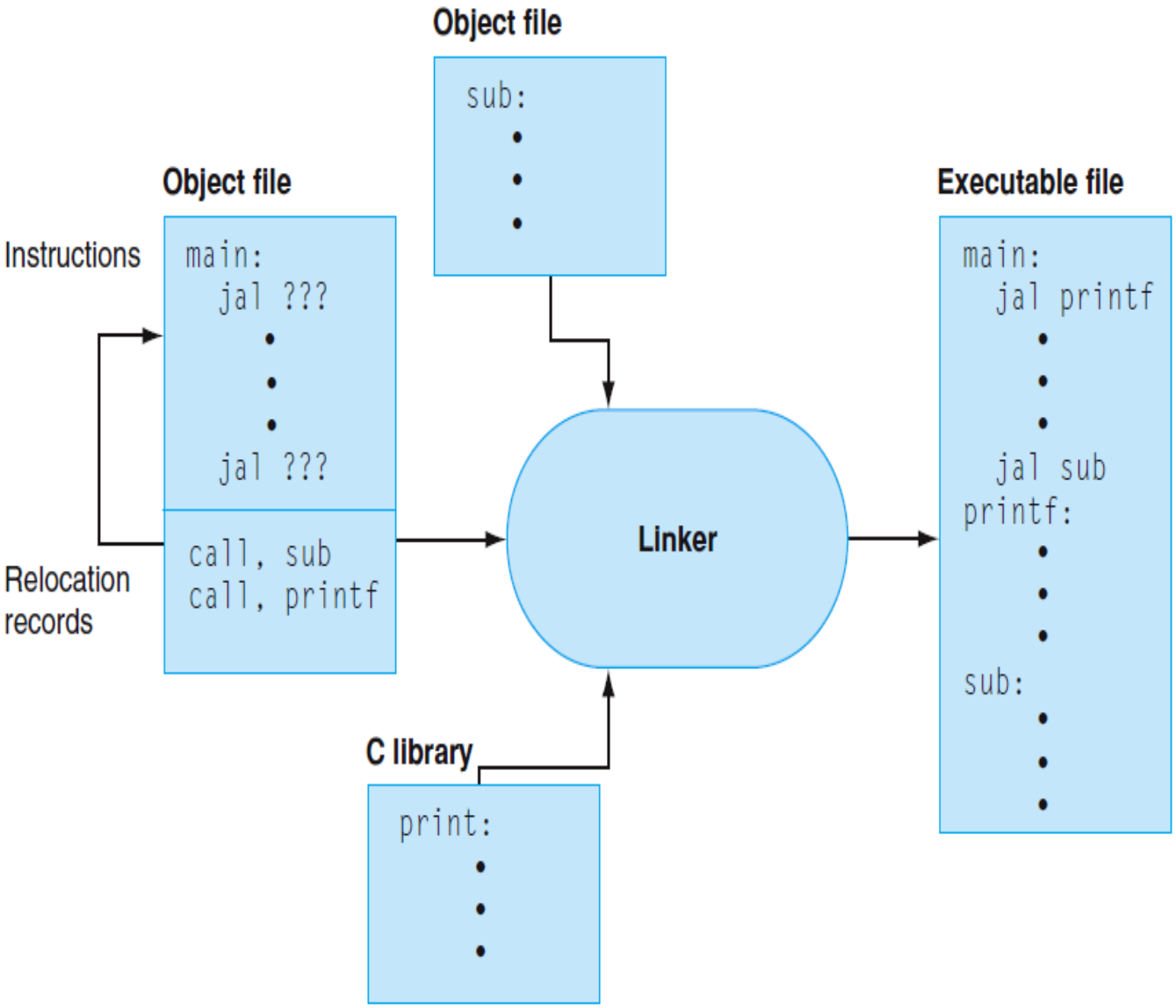


Figura A.1.1. **Il processo che produce un file eseguibile.** Un assemblatore traduce il file contenente codice assembler in un file oggetto, il quale viene collegato ad altri file e a funzioni di libreria per produrre un file eseguibile.

il collegatore (*linker*)



indirizzamento dei dati statici

- il segmento dati inizia all'indirizzo (a 64 bit)
  `0x 0000 0000 1000 0000`
- quindi le istruzioni di accesso alla memoria non possono fare riferimento direttamente agli oggetti in esso contenuti, poiché esse hanno immediati da 12 bit
- esempio: per caricare nel registro *a2* la parola doppia memoriazzata all'indirizzo `0x 0000 0000 1001 0020` occorrono due istruzioni

```
# carica i 20 bit più significativi nella metà
inferiore del registro
lui s0, 0x 10010      # 5 cifre hex = 20 bit
# costruisce l'indirizzo effettivo sommando
lo spiazzamento
ld  a2, 0x 020(s0)    # 3 cifre hex = 12 bit
```

indirizzamento dati statici

- la suddivisione dell'indirizzo nelle sue componenti per poter fare accesso a un dato in memoria, è ottenuta *in modo automatico* tramite assemblatore e collegatore facendo uso della pseudo istruzione **`la`**

- gli accessi a tutte le variabili, scalari o vettori, nel segmento dati statici sono quindi ottenuti come

```
la rd, VAR          # per memorizzare l'indirizzo della variabile in un registro,
                    # con VAR etichetta simbolica che lo rappresenta

ld  oppure sd       # con il registro base appena caricato tramite la
```

- la traduzione di **`la`** è

```
auipc rd, %pcrel_hi (VAR)
addi  rd, rd, %pcrel_lo (VAR)
```

e il calcolo fatto dal collegatore è (i.e. = indirizzo effettivo)

- **`delta_VAR = i.e. VAR – PC di auipc`** per relativizzare **`VAR`** al PC, e
- **`%hi (delta_VAR)`** e **`%lo (delta_VAR)`** per calcolare le due componenti da 20 it e 12 bit, con correzione per bit più significativo della parte low

simboli rilocabili – locali ed esterni

in fase di collegamento gli indirizzi definiti all'interno di un modulo possono cambiare se la base dello spazio di indirizzamento virtuale del modulo viene modificata (**rilocazione del modulo**)

pertanto, tutte le etichette che corrispondono a indirizzi assoluti all'interno del modulo (in pratica tutte eccetto quelle delle direttive ***.eqv***), costituiscono simboli il cui valore può cambiare al momento del collegamento

un simbolo usato in un'istruzione di un modulo è **locale** se è definito nello stesso modulo, **esterno** in caso contrario

l'assemblatore non traduce completamente le istruzioni nelle quali si fa riferimento a:

- un simbolo esterno, perché non ne conosce il valore
- un simbolo rilocabile (interno o esterno), perché il valore del simbolo cambierà in fase di collegamento, ma con l'eccezione seguente:
  - le istruzioni di salto che fanno riferimento a simboli locali che vengono relativizzati rispetto al PC possono essere tradotte completamente, perché la distanza dal PC non cambia quando il modulo viene rilocato (si dice che sono *autorilocanti*)

esempio di riferimento

sorgente procedura A:

```
.data
X:   .dword  128
W:   .dword  0x 12345678
.text
A:   la   t0, X
     ld   a2, 0(t0)
     beq  a2, zero, E
     jal  B
```

sorgente procedura B:

```
.data
Y:   .dword  0
.text
B:   bne  a2, zero, E
     la   t0, Y
     sd   a1, 0(t0)
E:   li   t1, W
```

oggetto procedura A:

dimensione testo: 0x 14

dimensione dati: 0x 10

testo:

```
0    auipc t0, 0 0 0 0 0
4    addi  t0, t0, 0 0 0
8    ld    a2, 0(t0)
C    beq   a2, zero, 0 0 0
10   jal   ra, 0 0 0 0 0
```

dati:

```
0    0x 80
8    0x 12345678
```

tabella simboli:

```
A    0      T
X    0      D
W    8      D
```

tabella rilocazione:

```
0    auipc %pcrel_hi (X)
4    addi  %pcrel_lo (X)
8    beq   %pcrel (E) / 2
10   jal   %pcrel (B) / 2
```

oggetto procedura B:

dimensione testo: 0x 18

dimensione dati: 0x 8

testo:

```
0    bne   a2, zero, 8
4    auipc t0, 0 0 0 0 0
8    addi  t0, t0, 0 0 0
C    sd    a1, 0(t0)
10   lui   t1, 0 0 0 0 0
14   addi  t1, t1, 0 0 0
```

dati:

```
0    0x 0
```

tabella dei simboli:

```
B    0      T
E    10     T
Y    0      D
```

tabella di rilocazione:

```
4    auipc %pcrel_hi (Y)
8    addi  %pcrel_lo (Y)
10   lui   %hi (W)
14   addi  %lo (W)
```

commenti relativi alla tabella di rilocazione

per ogni modulo questa tabella contiene tre valori in ogni riga

1. indirizzo dell'istruzione che va modificata dal collegatore
2. tipo di istruzione
3. simbolo il cui valore va inserito nell'istruzione come immediato da 20 bit o 12 bit, corredato dal suo modificatore

spiegazione della riga della tabella di rilocazione

- la riga 0 del modulo B è stata completata dall'assemblatore perché contiene un salto condizionale a un simbolo locale
- le righe 0 e 4 della procedura A, e 4 e 8 della procedura B, vengono risolte più comodamente dal collegatore *perché le basi di codice e dati sono diverse* (anche se i simboli coinvolti sono locali rispettivamente ad A e B, e i modificatori relativizzano rispetto al PC)
- le altre modifiche di istruzioni sono dovute al riferimento a simboli esterni

processo di collegamento (*linking*)

il collegatore ha il compito di generare a partire dai diversi moduli un ***solo programma binario eseguibile*** (in formato rilocabile)

➡ un ***solo spazio di indirizzamento*** per tutto il programma
(spazio di indirizzamento virtuale del programma)

in base

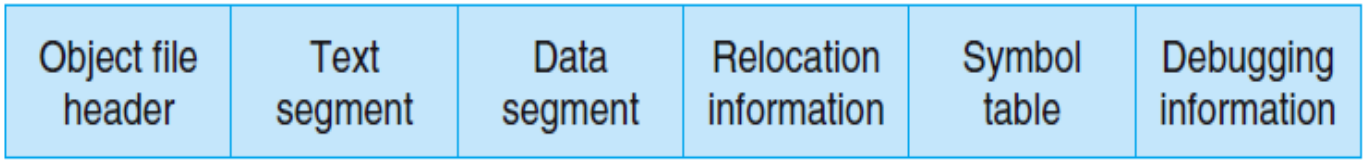
- alla lunghezza del segmento testo e del segmento dati di ciascun modulo tradotto
- agli indirizzi di impianto

il collegatore calcola gli indirizzi dei riferimenti non risolti e completa la traduzione delle istruzioni presenti nelle tabelle di rilocazione

operazioni svolte dal collegatore

1. determinare lo posizione in memoria, cioè l'indirizzo iniziale o di base, delle sezioni codice e dati dei diversi moduli (vedi esempio precedente)
2. determinare il nuovo valore di tutti gli indirizzi simbolici che risultano modificati dallo spostamento della base (creazione di una ***tabella dei simboli globale***)
3. correggere in tutti i moduli i riferimenti a indirizzi simbolici che sono stati modificati, in base alle tabelle di rilocazione

per fare questo, i moduli oggetto devono contenere una serie di informazioni aggiuntive, oltre al codice

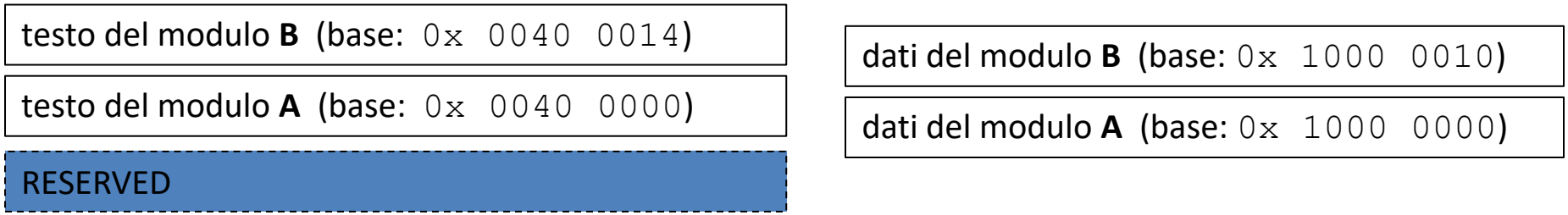


1 – determinazione della posizione in memoria dei moduli

l'assemblatore alloca la sezione testo e la sezione dati a partire dall'indirizzo base 0

- ma i moduli non possono essere caricati tutti nella stessa zona di memoria; devono invece essere caricati sequenzialmente
- inoltre devono rispettare la struttura generale della memoria

esempio di riferimento (nota: gli indirizzi sono rappresentati su 32 bit)



2 – creazione della tabella dei simboli globale

è costituita dall'unione delle tabelle dei simboli di tutti i moduli che vanno collegati, modificati (rilocati) in base all'indirizzo di base del modulo al quale appartengono

esempio di riferimento

simbolo	valore iniziale	base di rilocazione	valore finale (indirizzo effettivo o i.e.)
A	0	0040 0000	0040 0000
X	0	1000 0000	1000 0000
W	8	1000 0000	1000 0008
B	0	0040 0014	0040 0014
E	10	0040 0014	0040 0024
Y	0	1000 0010	1000 0010

3 – correzione dei riferimenti nei moduli

simbolo	i.e.
A	0040 0000
X	1000 0000
W	1000 0008
B	0040 0014
E	0040 0024
Y	1000 0010

base testo modulo **A**	base testo modulo **B**
0x 0040 0000	0x 0040 0014
base dati modulo **A**	base dati modulo **B**
0x 1000 0000	0x 1000 0010
tab. di rilocazione mod. **A**	tab. di rilocazione mod. **B**
0 **auipc** %pcrel_hi (X)	4 **auipc** %pcrel_hi (Y)
4 **addi** %pcrel_lo (X)	8 **addi** %pcrel_lo (Y)
8 **beq** %pcrel (E) / 2	10 **lui** %hi (W)
10 **jal** %pcrel (B) / 2	14 **addi** %lo (W)

	delta_IND = i.e. – PC	delta_IND
%pcrel (X)	1000 0000 – 0040 0000	0FC0 0000
%pcrel (E)	0040 0024 – 0040 0008	0000 001C
%pcrel (B)	0040 0014 – 0040 0010	0000 0004
%pcrel (Y)	1000 0010 – 0040 0018	0FBF FFF8

%pcrel_hi (X)	0FC0 0
%pcrel_lo (X)	000
%pcrel (E) / 2	00D
%pcrel (B) / 2	0 0002
%pcrel_hi (Y)	0FC0 0, cioè 0FBF F + 1
%pcrel_lo (Y)	FF8
%hi (W)	1000 0
%lo (W)	008

3 bis – correzione dei riferimenti nei moduli

siano

- ISTR un'istruzione riferita dalla tabella di rilocazione di un modulo M, con simbolo S e indirizzo IND
- IADDR l'indirizzo di una istruzione ISTR riferita dalla tabella di rilocazione di un modulo M con simbolo S
- VS il valore di S nella tabella globale dei simboli
- **%pcrel (S) = delta_VS = VS – IADDR** dell'istruzione cui si riferisce (per relativizzare VS rispetto a IADDR; in genere IADDR è il PC)
- **%hi (delta_VS)** e **%lo (delta_VS)** per calcolare le due componenti da 20 bit e 12 bit, rispettivamente, con correzione per bit più significativo della parte low

regole da applicare in base al tipo di istruzione

- ISTR è in formato **J:** inserire **%pcrel (S) / 2** su 20 bit
- ISTR è in formato **B**: inserire **%pcrel (S) / 2** su 12 bit
- ISTR è in formato **U**:
  - se **`auipc`** inserire **%pcrel_hi (S)**
  - se **`lui`** inserire **%hi (S)**
- ISTR è **`addi`** espansione di pseudoistruzione di *load*:
  - se espansione di **`la`** inserire **%pcrel_lo (S)**
  - se espansione di **`li`** inserire **%lo (S)**

indirizzo	istruzione completa	note
segmento codice		
0040 0000	**auipc** t0, **0 F C 0 0**	%pcrel_hi (X)
0040 0004	**addi** t0, t0, **0 0 0**	%pcrel_lo (X)
0040 0008	**ld** a2, 0(t0)	
0040 000C	**beq** a2, zero, **0 0 D**	%pcrel (E) / 2
0040 0010	**jal** ra, **0 0 0 0 2**	%pcrel (B) / 2
0040 0014	**bne** a2, zero, **8**	
0040 0018	**auipc** t0, **0 F C 0 0**	%pcrel_hi (Y)
0040 001C	**addi** t0, t0, **F F 8**	%pcrel_lo (Y)
0040 0020	**sd** a1, 0(t0)	
0040 0024	**lui** t1, **1 0 0 0 0**	%hi (W)
0040 0028	**addi** t1, t1, **0 0 8**	%lo (W)
segmento dati		
1000 0000	128	
1000 0008	0x 1234 5678	
1000 0010	0	

modulo **A**

modulo **B**

caricamento ed esecuzione

nei sistemi UNIX, è il kernel del sistema operativo che carica un programma nella memoria principale e ne lancia l'esecuzione

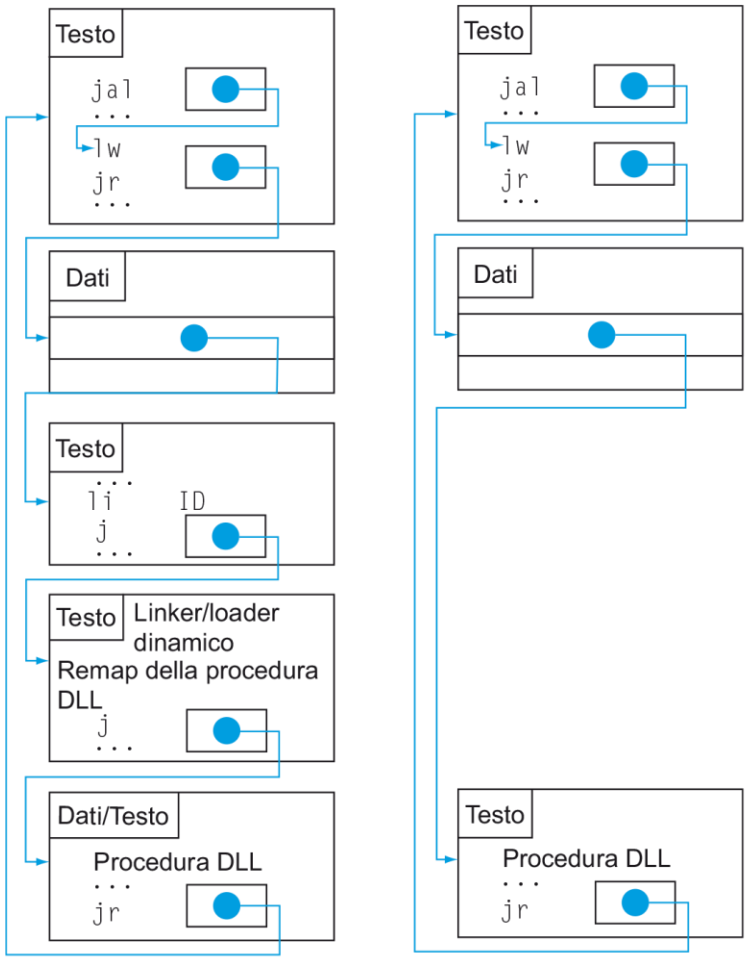
operazioni

1. leggere l'intestazione del file eseguibile per determinare le dimensioni dei segmenti testo e dati
2. creare un nuovo spazio di indirizzamento per il programma; questo spazio è abbastanza grande da contenere i segmenti di testo e dei dati, nonché un segmento per la pila
3. copiare le istruzioni e i dati del file eseguibile in memoria all'interno del nuovo spazio di indirizzamento

caricamento ed esecuzione

4. copiare nella pila gli argomenti passati al programma
5. inizializzare i registri dell'architettura; in generale, il contenuto della maggior parte dei registri viene cancellato, tranne quello del registro puntatore alla pila, al quale viene assegnato l'indirizzo della prima locazione libera della pila
6. saltare a una procedura di avvio che copia gli argomenti del programma dalla pila ai registri, per poi chiamare la procedura **`main`** del programma; quando la procedura **`main`** termina, la procedura di avvio conclude il programma tramite la chiamata di sistema **`exit`**

librerie a caricamento dinamico



(a) Prima chiamata a una procedura di una DLL

(b) Chiamate successive alla procedura della DLL