

## Architetture dei Calcolatori

### Assemblatore, Linker e Loader

Prof. Francesco Lo Presti

## Compilazione

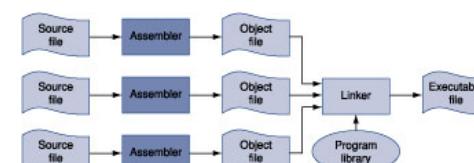
- Nella prima fase, il programma ad alto livello viene tradotto nel linguaggio assembler utilizzando un apposito programma detto **compilatore**
- Dopo la fase di compilazione, il programma scritto il linguaggio assembler viene tradotto in linguaggio macchina utilizzando un apposito programma detto **assemblatore** (assembler)
- Spesso con il termine compilazione si indica l'intero processo di traduzione da linguaggio ad alto livello a linguaggio macchina (essendo l'assemblatore spesso integrato con il compilatore)

## Il processo di compilazione



## Il linker

- Il **linker** (*link editor* o *collegatore*) ha il compito di collegare tra loro vari moduli che compongono lo stesso programma
  - Programma sorgente suddiviso in più file che vengono compilati separatamente creando diversi file oggetto
  - Utilizzo di funzioni di libreria
- Il linker collega tra loro i file contenenti il codice oggetto dei vari moduli che costituiscono il programma, unendovi anche il codice delle funzioni di libreria utilizzate, producendo un file contenente il codice eseguibile, memorizzato su disco



## Esempio

```
.globl main
main:...
jal proc
...
.globl proc
proc:    addi $sp, $sp, -20
...
beq $s0, $zero, ret
addi $a0, $s0, -1
addi $a1, $s1, 0
addi $a2, $s2, 0
jal proc
addi $s3, $v0, 0
addi $a0, $s0, 0
addi $a1, $s1, 0
addi $a2, $s2, 0
jal is_in
beq $v0, $zero, else2
sll $s3, $s3, 1
addi $s3, $s3, 1
j ret
else2:  addi $s3, $s3, 2
ret:    addi $v0, $s3, 0
...
addi $sp, $sp, 20
jr $ra
```

```
.globl is_in
is_in:   addi $t0, $zero, 0
         addi $t1, $zero, 0
while:   slt $t2, $t0, $a2
         beq $t2, $zero, end
         sll $t3, $t0, 2
         add $t3, $a1, $t3
         lw $t4, 0($t3)
         bne $t4, $a0, inc
         addi $t1, $zero, 1
         j end
inc:    addi $t0, $t0, 1
         j while
end:   addi $v0, $t1, 0
        jr $ra
```

Assembler, Linker & Loader

5

## L'assemblatore

### □ Traduce un modulo in assembler in un file oggetto

#### ○ Una combinazione di

✓ Istruzioni in linguaggio macchina

✓ Dati

✓ Informazioni per la corretta collocazione del modulo in memoria:  
Rilocazione

#### ○ Esempio

✓ Per poter tradurre in linguaggio macchina le istruzioni di salto, l'assemblatore deve poter determinare gli indirizzi corrispondenti alle etichette usate nei salti (*tabella dei simboli*)

✓ *Tabella dei simboli* contiene le corrispondenze tra nomi delle etichette (simboli) ed indirizzi di memoria

### □ Il file oggetto prodotto dall'assemblatore assume che il modulo parta dalla locazione di memoria 0 (indirizzi relativi)

#### ○ Sarà il linker a gestire la rilocazione

✓ Istruzioni dipendenti da un indirizzo assoluto

Assembler, Linker & Loader

6

## Il processo di assemblaggio

- Il processo di traduzione in linguaggio macchina è semplice, salvo per le istruzioni con riferimenti a etichette locali non ancora dichiarate (*riferimenti in avanti*: l'etichetta è usata prima di essere definita)
- Assemblatore a due passate
  - *Prima passata*: calcola la lunghezza delle istruzioni e genera una tabella dei simboli, contenente l'associazione tra i nomi delle etichette ed i corrispondenti indirizzi di memoria
  - *Seconda passata*: tutti i riferimenti locali sono noti: usando la tabella dei simboli, può generare il codice macchina
- Assemblatore ad una passata
  - Traduce in codice binario tutte le istruzioni tranne quelle con riferimenti in avanti (tecnica del *backpatching*)
  - Mette queste istruzioni in una tabella dei simboli irrisolti
  - Alla fine traduce anche queste
  - Maggiore velocità ma intera rappresentazione binaria in memoria

Assembler, Linker & Loader

7

## Struttura del file oggetto

- Nei sistemi Unix, il file oggetto è composto da 6 sezioni distinte
  - Object file header (intestazione)
    - ✓ Dimensione e posizione delle altre parti del file oggetto
  - Segmento di testo
    - ✓ Codice in linguaggio macchina per le procedure nel file sorgente; possono esserci riferimenti irrisolti (istruzioni contenenti riferimenti a simboli esterni)
  - Segmento di dati
    - ✓ Rappresentazione binaria dei dati
  - Informazioni di rilocazione
    - ✓ Identificazione di istruzioni e dati da aggiornare al momento della rilocazione, in quanto dipendenti da indirizzi di memoria assoluti
  - Tabella dei simboli
    - ✓ Simboli definiti nel modulo e *riferibili dall'esterno* (etichette globali) e lista dei *riferimenti irrisolti* (ad es. per uso di funzioni di libreria)
  - Informazioni per il debugger
    - ✓ Descrizione concisa del modo in cui il programma è stato compilato

Assembler, Linker & Loader

8

## Esempio: Primo Passata

```
.globl proc
proc:    addi $sp, $sp, -20
...
beq $s0, $zero, ret
addi $a0, $s0, -1
addi $a1, $s1, 0
addi $a2, $s2, 0
jal proc
addi $s3, $v0, 0
addi $a0, $s0, 0
addi $a1, $s1, 0
addi $a2, $s2, 0
jal is_in
beq $v0, $zero, else2
sll $s3, $s3, 1
addi $s3, $s3, 1
j ret
else2: addi $s3, $s3, 2
ret:    addi $v0, $s3, 0
...
addi $sp, $sp, 20
jr $ra
```

| Tabella dei Simboli |           |
|---------------------|-----------|
| Etichetta           | Indirizzo |
| proc                | 0x00      |
| else2               | 0x60      |
| ret                 | 0x64      |

Assembler, Linker & Loader

9

## Esempio: Secondo Passata

```
.globl proc
proc:    addi $sp, $sp, -20
...
beq $s0, $zero, 14
addi $a0, $s0, -1
addi $a1, $s1, 0
addi $a2, $s2, 0
jal proc
addi $s3, $v0, 0
addi $a0, $s0, 0
addi $a1, $s1, 0
addi $a2, $s2, 0
jal is_in
beq $v0, $zero, else2
sll $s3, $s3, 1
addi $s3, $s3, 1
j ret
else2: addi $s3, $s3, 2
ret:    addi $v0, $s3, 0
...
addi $sp, $sp, 20
jr $ra
```

Generazione  
Codice  
Macchina

```
...  
addi $sp, $sp, -20  
...  
beq $s0, $zero, 14  
addi $a0, $s0, -1  
addi $a1, $s1, 0  
addi $a2, $s2, 0  
jal 0x0  
addi $s3, $v0, 0  
addi $a0, $s0, 0  
addi $a1, $s1, 0  
addi $a2, $s2, 0  
jal ???  
beq $v0, $zero, 3  
sll $s3, $s3, 1  
addi $s3, $s3, 1  
j ret  
else2: addi $s3, $s3, 2  
ret:    addi $v0, $s3, 0  
...
addi $sp, $sp, 20
jr $ra
```

| Tabella dei Simboli |           |
|---------------------|-----------|
| Etichetta           | Indirizzo |
| proc                | 0x00      |
| else2               | 0x60      |
| ret                 | 0x64      |

| Tabella dei Simboli    |           |
|------------------------|-----------|
| Etichetta              | Indirizzo |
| proc                   | 0x00      |
| is_in                  | 0x4B      |
| Relocation Information |           |
| Istr.                  | Indirizzo |
| jal                    | 0x38      |
| jal                    | 0x4B      |
| j                      | 0x5B      |

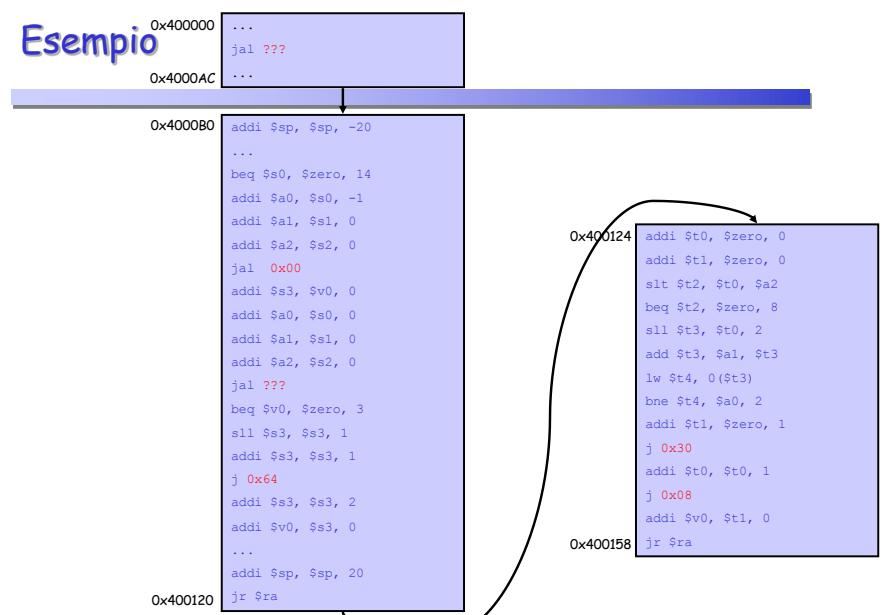
Assembler, Linker & Loader

10

## Il linker

- ❑ Ciascun modulo ha il suo spazio degli indirizzi
  - Quando i moduli vengono collegati occorre **traslare** i loro spazi degli indirizzi
- ❑ Occorre risolvere tutti i riferimenti esterni - tramite etichette globali - ossia le chiamate tra moduli e riferimenti a variabili globali
- ❑ Compiti del linker (a partire dalla tabella dei simboli e dalle informazioni di rilocazione)
  - Porre simbolicamente in memoria istruzioni e dati
  - Determinare gli indirizzi dei dati e delle etichette dei salti
  - Unire i riferimenti interni ed esterni alle procedure
- ❑ Il linker produce un file oggetto che può essere eseguito sulla macchina
  - Stesso formato del file oggetto ma con tutti i riferimenti risolti e senza informazioni di rilocazione

## Esempio

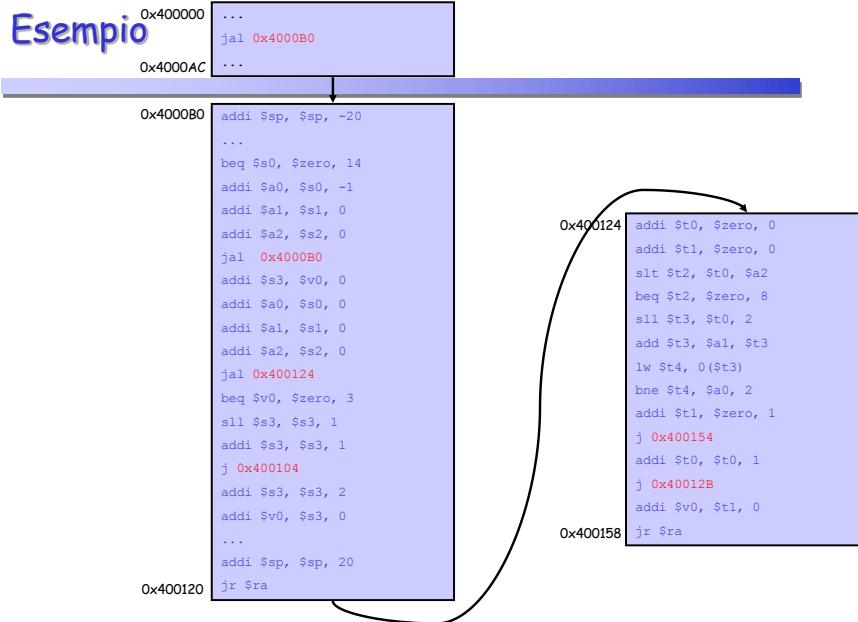


Assembler, Linker & Loader

12

Assembler, Linker & Loader

11



## Istruzioni MIPS da Rilocare

- ❑ Indirizzi relativi al PC (istruzioni beq e bne): da non rilocare
  - L'indirizzamento relativo al PC è preservato anche se il codice viene spostato
- ❑ Indirizzi assoluti (istruzioni j e jal): da rilocare sempre
- ❑ Da rilocare anche istruzioni lw e sw che utilizzano il registro \$gp per indirizzare dati statici
- ❑ Riferimenti esterni (tipicamente istruzione jal): da rilocare sempre

## Il loader

- ❑ Una volta che il linker ha creato il file eseguibile, esso viene di solito memorizzato su disco
- ❑ All'atto dell'esecuzione, il S.O. lo carica dal disco e ne avvia l'esecuzione
- ❑ Il loader si occupa di far partire il programma
  - Legge l'intestazione per determinare la dimensione dei segmenti testo e dati
  - Crea uno spazio di memoria sufficiente per testo e dati
  - Copia istruzioni e dati in memoria
  - Copia nello stack i parametri (se presenti) passati al main
  - Inizializza i registri e lo stack pointer
  - Salta ad una procedura di inizializzazione che copia i parametri nei registri appositi dallo stack e poi invoca la procedura di inizio del programma (`main()` in C)

## Librerie collegate dinamicamente

- ❑ Svantaggi del collegamento statico (prima dell'esecuzione del programma)
  - Le routine della libreria fanno parte del codice eseguibile
    - ✓ Uso di vecchie versioni della libreria anche se disponibili nuove versioni
  - Caricamento dell'intera libreria anche se usata parzialmente
- ❑ Librerie collegate dinamicamente (DLL)
  - Le routine della libreria non sono collegate e caricate finché il programma non viene eseguito
  - Ogni routine viene collegata solo dopo essere invocata (*lazy procedure linkage*)