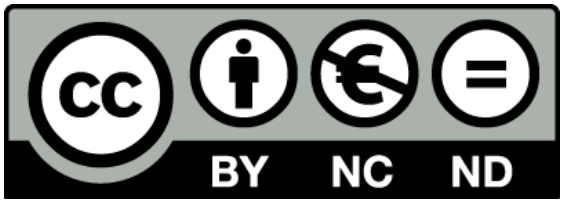
Laboratorio di Programmazione di Sistema

Organizzazione dei Dati in Memoria 2

Luca Forlizzi, Ph.D.

Versione 20.1



Indirizzi di Memoria e Label

- Per un umano che crea un programma, dover leggere, scrivere e ricordare gli indirizzi delle parole di memoria che contengono i dati del programma è molto scomodo, in quanto ogni indirizzo è una stringa binaria
- Nel *LM*, purtroppo non vi sono alternative
- Gli *ASM* permettono di lavorare in maniera più comoda, usando istruzioni estremamente simili a quelle del *LM*
- Anche gli indirizzi possono essere scritti in forme più comode, utilizzando
  - Codifiche decimale e esadecimale per rappresentare numeri
  - *Label*

Indirizzi di Memoria e Label

- Le label, che abbiamo già usato nelle lezioni precedenti, sono stringhe alfanumeriche, molto più facili da utilizzare per un umano
- Ogni label è *legata* ad un indirizzo di memoria, ovvero è una rappresentazione simbolica di tale indirizzo
- In fase di traduzione, l'assembler trasforma ogni label nell'indirizzo ad essa legato
- Poiché ogni label è rappresenta un indirizzo e ogni indirizzo può essere interpretato come intero positivo, anche le label possono essere interpretate come interi positivi
- Aquistano quindi significato alcune operazioni aritmetiche che coinvolgono label: ad esempio se `my_lab` è una label, l'espressione `my_lab+1` rappresenta l'indirizzo successivo di quello del byte rappresentato dalla label

Indirizzi di Memoria e Label

- Le considerazioni precedenti portano alla luce alcune questioni, molto importanti nella programmazione *ASM*, relative a label ed indirizzi
  - Data una definizione di label, in base a quale regola viene legata ad un indirizzo?
  - In quale momento del processo di traduzione avviene tale legame?
  - Più in generale, se il testo di un programma utilizza solo label per identificare le parole di memoria, come, quando e da cosa vengono scelti gli indirizzi delle parole di memoria effettivamente utilizzate?

Indirizzi di Memoria e Label

- Le risposte sono complesse e variano in base ad alcune caratteristiche dei livelli 2, 3 e 4 di un computer
  - Modello di memoria piatta o segmentata
  - Struttura dell'eventuale sistema operativo (SO)
  - Quali sono gli indirizzi validi e quali, pur essendo validi, sono riservati (mappati su porte o registri, riservati al SO)
  - Eventuale possibilità che più programmi siano in esecuzione (o comunque presenti in memoria) allo stesso tempo (*multi-tasking*)
  - Eventuale virtualizzazione della memoria
  - Eventuale simultaneità di traduzione e caricamento di un programma *ASM* (*Assemble-go*)
  - Eventuale possibilità di *traduzione separata* di parti di un programma *ASM*

Allocazione della Memoria

- Il procedimento generale con cui vengono scelte ed riservate le parole di memoria utilizzate da un programma è chiamato *allocazione della memoria*
- Nei programmi *ASM*, l'allocazione di memoria stabilisce anche il legame tra label e indirizzi
- Nei *HLL*, l'allocazione di memoria comprende la *allocazione delle variabili* ovvero la scelta delle parole di memoria utilizzate per memorizzare ogni variabile di un programma

Allocazione della Memoria

- In un ambiente operativo in cui più programmi possono risiedere in memoria (ed eventualmente essere eseguiti) allo stesso tempo, allocare memoria per un programma vuol dire anche riservare una certa quantità di memoria per tale programma: le parole di memoria allocate ad un programma non possono essere usate da altri programmi
- L'allocazione della memoria può essere
  - *Statica*
  - *Dinamica*

Allocazione della Memoria

- L'allocazione statica della memoria può a sua volta avvenire in due diversi momenti
  - *Al momento della traduzione*
  - *Al momento del caricamento del programma in memoria* (di solito subito prima che inizi l'esecuzione)
- Se l'allocazione statica avviene al momento della traduzione, allora un programma è vincolato ad utilizzare sempre le stesse aree di memoria ogni volta che viene eseguito
- Quando un programma ha questo vincolo, la sua traduzione in *LM* viene detta codice (eseguibile) *assoluto*

Allocazione della Memoria

- Se si desidera dotare un computer di un ambiente operativo multi-tasking, ovvero che permetta a più programmi (non progettati appositamente) di essere in esecuzione allo stesso tempo, è necessario che i programmi non siano vincolati a dover usare specifiche aree di memoria
- In caso contrario, due programmi vincolati ad usare aree di memoria che hanno byte in comune, non potrebbero essere in memoria allo stesso tempo, in quanto interferirebbero
- Dunque i programmi devono essere in grado di funzionare in modo indipendente dagli indirizzi delle aree di memoria ad essi riservate; ovvero al programmatore non è permesso richiedere che la memoria venga allocata a specifici indirizzi

Allocazione della Memoria

- Ciò implica che in ambienti operativi multi-tasking
  - L'allocazione statica della memoria (e in particolare la determinazione degli indirizzi della memoria da allocare) deve avvenire al momento del caricamento
  - Le traduzioni in *LM* dei programmi devono essere non vincolate ad utilizzare aree di memoria specifiche
  - Codice eseguibile che ha tale caratteristica, viene detto codice (eseguibile) *rilocabile*

Allocazione Statica in *ASM*

- Per i programmi *ASM*, l'allocazione della memoria statica viene fatta mediante le *sezioni*
- L'idea è semplice: ad ogni sezione corrisponde un'area di memoria allocata al programma
- Naturalmente le diverse aree allocate presenti, in un certo momento, nella memoria del computer, non devono sovrapporsi tra loro (indipendentemente dal fatto che siano riservate allo stesso programma o a programmi diversi)

Allocazione Statica in *ASM*

- Per ogni sezione l'indirizzo dell'area di memoria ad essa associata viene detto *indirizzo di inizio* (della sezione)
- Gli indirizzi di inizio delle sezioni possono essere stabiliti come parte dell'allocazione statica della memoria, ovvero:
  - *Al momento della traduzione*
  - *Al momento del caricamento del programma in memoria* (di solito subito prima che inizi l'esecuzione)

Allocazione Statica in *ASM*

- In *ASM* si alloca staticamente memoria per i dati di un programma, utilizzando delle direttive apposite chiamate *direttive di definizione dati*
- Ogni direttiva di definizione dati alloca un'area di memoria
- Una direttiva di definizione dati indica quanti byte devono essere allocati e, opzionalmente, i valori da memorizzare in tali byte al momento dell'allocazione (ovvero prima che inizi l'esecuzione del programma)
- Una direttiva di definizione dati non indica l'indirizzo di memoria dei byte da allocare: tale indirizzo viene determinato dall'indirizzo di inizio della sezione in cui compare la direttiva e da una variabile chiamata *location counter (LC)* che è associata ad ogni punto del codice sorgente

Allocazione Statica in *ASM*

- In ogni punto del codice sorgente, il valore di LC è pari al numero di byte allocati dall'inizio della sezione a cui il punto appartiene, fino al punto stesso
- Ogni direttiva di inizio sezione imposta a 0 il valore di LC
- Ogni direttiva di definizione dati aumenta LC di un valore pari alla quantità di byte da essa allocati
- Se una direttiva di definizione dati si trova in una sezione che ha indirizzo di inizio $\mathcal{I}$ e in un punto del sorgente in cui LC vale $\mathcal{LC}$, l'indirizzo dell'area di memoria allocato per tale direttiva è pari a $\mathcal{I} + \mathcal{LC}$

Allocazione Statica in *ASM*

- Il legame tra label e indirizzi avviene utilizzando gli indirizzi di inizio sezione e LC
  - Ogni label viene legata all'indirizzo che ha valore pari alla somma dell'indirizzo di inizio della sezione in cui la label è definita e del valore di LC nel punto in cui si trova la definizione della label
- Da questa regola segue che nel momento in cui viene stabilito l'indirizzo di inizio sezione, automaticamente si determinano anche:
  - L'indirizzo dell'area allocata da ciascuna direttiva di definizione dati
  - Per ciascuna label, l'indirizzo ad essa legato

Allocazione Statica in *ASM*

- Se gli indirizzi di inizio sezione vengono stabiliti al momento del caricamento, allora al momento della traduzione non sono ancora definiti gli indirizzi delle aree allocate dalle direttive di definizione dati e i legami tra label e indirizzi
- Tuttavia, date due direttive di definizione dati che appartengono alla stessa sezione, già al momento della traduzione risulta definita la distanza tra gli indirizzi delle aree allocate dalle due direttive: essa è pari al valore assoluto della differenza tra i valori di LC nei punti del sorgente in cui si trovano le due direttive
- In modo analogo, date due label definite nella stessa sezione, al momento della traduzione è definita la distanza tra gli indirizzi legati alle label

Esempio Allocazione Statica in MARS

- Nel seguito è mostrato un breve esempio di sezione dati in MARS, con evidenziati i valori di LC e il calcolo degli indirizzi

```
# inizio sez. dati, LC = 0
  .data 0x10010000
lab3: .word -2, 3, 5   # lab3 = 0x10010000
                       # in questo punto LC = 12
      .word 0, 2       # 2 word da 0x1001000C, LC = 20
lab4: .half -2         # una halfword da lab4 = 0x10010014
                       # in questo punto LC = 22
      .half 6, 7       # 2 halfword da 0x10010016, LC = 26
```

Allocazione e Assembler

- Come abbiamo detto vi sono diverse caratteristiche dell'ambiente operativo che influiscono sulle modalità di allocazione statica
- Le modalità di allocazione statica, a loro volta influiscono su alcune caratteristiche degli assembler
- Nel caso in cui gli indirizzi di inizio delle sezioni vengono stabiliti al momento della traduzione, allora un programma è vincolato ad utilizzare sempre le stesse aree di memoria ogni volta che viene eseguito, ovvero la sua traduzione in LM è codice assoluto

Allocazione e Assembler

- Si noti che se gli indirizzi di inizio delle sezioni sono stabiliti al momento della traduzione, allora l'assembler conosce esattamente
  - Quali aree di memoria allocare
  - Quali sono gli indirizzi corrispondenti alle label
- Pertanto è anche in grado di caricare in memoria il codice assoluto prodotto
- Si chiama *Assemble-go* la modalità operativa degli assembler che, dopo aver tradotto il codice sorgente, si occupano anche di caricare in memoria il codice assoluto prodotto

Allocazione e Assembler

- Se un assembler non opera in modalità Assemble-go, allora produce codice rilocabile
  - La traduzione in *LM* del programma non contiene gli indirizzi di memoria da allocare e a cui accedere
  - Per ogni parola di memoria allocata staticamente e usata dal programma, il codice rilocabile contiene solo la distanza di tale parola dall'inizio della sezione in cui essa è contenuta
  - Prima o durante l'esecuzione, l'indirizzo di inizio sezione viene sommato a tale distanza, ottenendo il corretto indirizzo che consente di accedere alla parola di memoria

Allocazione e Assembler

- Se non diversamente specificato, coerentemente con il modello a 3 livelli che non prevede l'esistenza di un sistema operativo, nelle prossime lezioni di LAE assumiamo
  - Flat memory model
  - Il programma *ASM* è l'unico programma presente nel computer
  - Memoria non virtualizzata
  - Un programma viene sempre tradotto tutto insieme
- In queste condizioni, vincolare i programmi ad utilizzare sempre le stesse aree di memoria non crea alcun problema
- Ciò rende possibile operare con assembler in modo *Assemble-go*, semplificandone l'utilizzo

Allocazione e Assembler

- MARS opera sempre in modo *Assemble-go*: effettua l'allocazione statica durante la traduzione e permette che sia il programmatore a stabilire gli indirizzi di inizio sezione
- ASM-One può essere usato in modo *Assemble-go*, come abbiamo fatto nelle precedenti lezioni di LPS, oppure per produrre codice rilocabile in grado di funzionare nel sistema operativo AmigaOS, che supporta il multi-tasking dei programmi
- Per ulteriori informazioni sulle direttive di inizio sezione e le direttive di definizione dei dati si rimanda alla documentazione di MARS e a quella di ASM-One