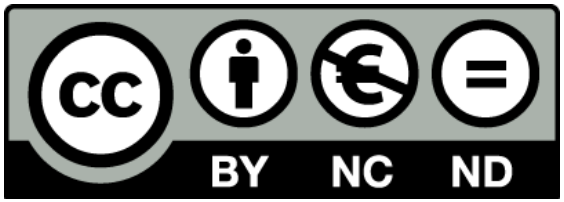
Laboratorio di Programmazione di Sistema

Programma Memorizzato

Luca Forlizzi, Ph.D.

Versione 23.1



Concetto di Programma Memorizzato

- La abstract machine di un *HLL* imperativo è un dispositivo in grado di eseguire programmi su dati
- Nei modelli computazionali degli *HLL*, è ben chiaro che i dati sono memorizzati in "contenitori" accessibili alla abstract machine
- Al contrario, tali modelli non descrivono il modo in cui il dispositivo accede alle istruzioni che formano il programma
- In linea di principio, quando si scrive un programma, si può pensare alla abstract machine di un *HLL* imperativo come ad un dispositivo che contenga al suo interno il programma da eseguire

Concetto di Programma Memorizzato

- Come sappiamo, in realtà non è così
- Uno dei pregi più importanti dei computer in uso oggi è che sono dispositivi assai flessibili in quanto eseguono programmi che hanno la forma di *software*, ovvero sono anch'essi immagazzinati in una memoria dalla quale vengono caricati per essere eseguiti
- Grazie a ciò, uno stesso computer può eseguire tanti programmi, diversissimi tra loro
- Cambiare il programma eseguito da un computer ha costo pressoché nullo

Concetto di Programma Memorizzato

- In questa presentazione discutiamo di come il concetto di *programma memorizzato* emerga a livello di *ASM*, in quanto le abstract machine di livello 4 eseguono programmi contenuti nella medesima memoria usata per i dati, esattamente come le abstract machine di livello 2
- In tutta la presentazione si usano le seguenti definizioni
  - M_4: generica abstract machine di livello 4, in grado di eseguire programmi *ASM*
  - M_2: generica abstract machine di livello 2, in grado di eseguire programmi *LM*

Concetto di Programma Memorizzato

- In una M_2, in base al modello di Von Neumann, si ha che
  - Le istruzioni sono memorizzate nella memoria principale, che è usata anche per memorizzare i dati
  - Le istruzioni sono sequenze di stringhe binarie, così come i dati
- Di norma, la memoria immagazzina bit senza attribuire loro un significato: dunque non distingue tra istruzioni e dati (così come non distingue tra tipi di dato diversi)
- Al contrario, M_2 interpreta i bit letti dalla memoria, in certi casi come istruzioni e in altri come dati
- Il concetto di programma memorizzato è presente in *ASM*, in modo simile a come è presente nel livello 2
- Questo è un ulteriore elemento che differenzia gli *ASM* dai linguaggi *HLL*

Programma Memorizzato in una *ISA*

- Ogni istruzione del *LM* è una stringa binaria, immagazzinata in un'area di memoria
- Chiamiamo *indirizzo* di un'istruzione, l'indirizzo dell'area di memoria che la contiene
- Chiamiamo *dimensione* di un'istruzione, la dimensione, espressa in byte, dell'area di memoria che contiene l'istruzione
- Se un'istruzione S_1 ha indirizzo A_{S_1} e dimensione D_{S_1} e un'istruzione S_2 ha indirizzo pari a $A_{S_1} + D_{S_1}$, allora S_2 è l'istruzione successiva di S_1 in *ordine di memorizzazione*

Programma Memorizzato in una *ISA*

- L'esecuzione di un'istruzione da parte di M_2 inizia con la lettura dell'istruzione stessa dalla memoria (fase di *fetch*)
- A tale scopo, M_2 dispone di un registro speciale chiamato *program counter* (*PC*) che contiene l'indirizzo dell'istruzione
  - Si tratta di una denominazione non particolarmente felice, ma che è ormai di uso comune; alcuni autori propongono come denominazione più appropriata *Instruction Address Register*
- Dopo aver letto i bit che formano l'istruzione, M_2 ne interpreta il significato (fase di *decode*) e mette in atto le azioni necessarie per effettuare le operazioni previste dall'istruzione (fase di *execute*)
- Durante l'esecuzione dell'istruzione, PC viene modificato in modo da contenere l'indirizzo da cui leggere la prossima istruzione, ovvero l'istruzione successiva in *ordine di esecuzione*

Programma Memorizzato in una *ISA*

- La maggior parte delle istruzioni modificano PC aggiungendo ad esso la dimensione dell'istruzione corrente
- In questo modo l'indirizzo dell'istruzione successiva in *ordine di esecuzione* sarà pari all'indirizzo dell'istruzione corrente aumentato con la dimensione dell'istruzione corrente, ovvero sarà uguale all'indirizzo dell'istruzione successiva in *ordine di memorizzazione*
- Le istruzioni di salto del *LM*, invece, possono modificare in modo diverso PC, in base alla propria semantica, e quindi fare in modo che l'istruzione successiva in *ordine di esecuzione* abbia un indirizzo diverso da quello dell'istruzione successiva in *ordine di memorizzazione*

Programma Memorizzato in un *ASM-PM*

- Il concetto di programma memorizzato emerge al livello 4 per via della corrispondenza tra le istruzioni che formano un programma *ASM* e le istruzioni che formano la traduzione in *LM* di tale programma
- Infatti ogni istruzione *ASM* ha una traduzione costituita da una o più istruzioni *LM*
- Le istruzioni *LM* che formano la traduzione di una singola istruzione *ASM* devono essere eseguite in sequenza, quindi devono formare una sequenza in ordine di memorizzazione, ovvero devono essere immagazzinate in un'area di memoria

Programma Memorizzato in un *ASM-PM*

- Di conseguenza per ogni istruzione *ASM I* si definisce
  - L'*indirizzo*, pari all'indirizzo dell'area di memoria che contiene la traduzione di *I* in *LM*
  - La *dimensione*, pari alla dimensione dell'area di memoria che contiene la traduzione di *I* in *LM*
- Il registro PC è presente anche nelle abstract machine di livello 4 e contiene gli indirizzi delle istruzioni, a mano a mano che esse vengono eseguite
- Inoltre un programma *ASM* può accedere alla memoria e quindi in particolare alle parole di memoria che contengono le traduzioni *LM* di istruzioni *ASM*, anche istruzioni del programma stesso

Caricamento delle Istruzioni

- L'esecuzione di una istruzione *ASM I* da parte di una abstract machine di livello 4, viene realizzata facendo eseguire da una abstract machine di livello 2, la traduzione in *LM* di *I*
- Poiché la abstract machine di livello 2 legge dalla memoria le istruzioni da eseguire, la traduzione in *LM* di un'istruzione *ASM I* deve essere *caricata*, ovvero immagazzinata, in memoria
- Il *caricamento* di un programma *ASM* consiste nell'immagazzinare in memoria le traduzioni di tutte le istruzioni che formano il programma

Caricamento delle Istruzioni

- Di solito, il caricamento avviene dopo la traduzione e prima dell'avvio dell'esecuzione del programma
- Gli assembler di tipo 1, effettuano il caricamento durante oppure subito dopo la traduzione del programma
- Gli assembler di tipo 2 o 3 producono invece un file eseguibile, e il caricamento viene fatto dal loader, il quale legge tale file dalla memoria secondaria e ne trasferisce il contenuto (ovvero le istruzioni LM) in memoria principale

Caricamento delle Istruzioni

- Come abbiamo detto, in assenza di istruzioni di salto, l'ordine di esecuzione è determinato
  - in *ASM* dall'ordine testuale
  - in *LM* dall'ordine di memorizzazione
- Quindi nel caricare la traduzione in *LM* di un programma *ASM*, è necessario far coincidere l'ordine testuale delle istruzioni *ASM* con l'ordine di memorizzazione delle corrispondenti traduzioni in *LM*
- Ciò vuol dire che se I_1, I_2 sono due istruzioni di un programma *ASM* e I_2 è l'istruzione successiva di I_1 in ordine testuale, la prima delle istruzioni che formano la traduzione di I_2 deve essere l'istruzione successiva in ordine di memorizzazione dell'ultima istruzione tra quelle che formano la traduzione di I_1
- Ovvero l'indirizzo di I_2 deve essere pari a l'indirizzo di I_1 aumentato della dimensione di I_1

Caricamento delle Istruzioni

- Ricordiamo che
  - Un programma *ASM* è diviso in sezioni, alcune di dati altre di codice
  - Nel processo di caricamento, per ogni sezione di dati S_D viene allocata staticamente un'area di memoria a partire dall'indirizzo di inizio sezione: procedendo in ordine testuale, per ciascuna direttiva di allocazione dato D contenuta in S_D si alloca (e eventualmente si inizializza) una determinata quantità di byte, ad indirizzi sempre crescenti

Caricamento delle Istruzioni

- In modo simile a quanto accade per le sezioni dati, nel processo di caricamento
  - Per ogni sezione di codice S_C si alloca un'area di memoria a partire dall'indirizzo di inizio sezione
  - Si esaminano le istruzioni di S_C in ordine testuale, e per ciascuna di esse, procedendo per indirizzi crescenti, si alloca una quantità di byte pari alla dimensione dell'istruzione e in tali byte si memorizza la traduzione dell'istruzione

Caricamento delle Istruzioni

- Anche le definizioni di label nelle sezioni di codice vengono gestite allo stesso modo di quelle presenti nelle sezioni di dati, ed è questo il motivo per cui le label si usano sia per riferirsi ad istruzioni che a dati
- Ogni definizione di label Def_L, stabilisce che la label L sia una rappresentazione simbolica di un indirizzo di memoria
  - Se Def_L non è preceduta, all'interno della sezione, da un'istruzione o da una direttiva di allocazione dato, L rappresenta l'indirizzo di inizio sezione
  - Altrimenti, L rappresenta l'indirizzo del byte successore dell'area di memoria allocata per l'istruzione o direttiva di allocazione dato che precede immediatamente Def_L

Indirizzamento delle Istruzioni

- Sia in *ASM* che in *LM* l'esecuzione di un programma è controllata, oltre che dal sequenziamento, anche dalle istruzioni di salto
- Ogni istruzione di salto J specifica un'istruzione di destinazione del salto I_{DJ}
- Quando J viene eseguita, se il salto avviene (il che può non accadere sempre se J è un salto condizionato), la abstract machine *indirizza* la prossima istruzione da eseguire I_{DJ}, ovvero
  - calcola l'indirizzo di I_{DJ}
  - copia tale indirizzo in PC
- L'indirizzamento della prossima istruzione da eseguire, viene fatto in base ad un modo di indirizzamento, in maniera analoga a quanto avviene per i dati

Indirizzamento delle Istruzioni

- Poiché la traduzione *LM* di un'istruzione *ASM* è il contenuto di un'area di memoria, tutti i modi di indirizzamento già noti per dati in memoria possono essere usati come modi di indirizzamento per istruzioni
- Si noti che nel caso di istruzioni di salto con destinazione dinamica, è necessario utilizzare un modo di indirizzamento che permetta di calcolare e modificare dinamicamente l'indirizzo dell'operando, ad esempio l'indirizzamento indiretto-registro
- Un ulteriore tipologia di modi di indirizzamento, molto comune per istruzioni, è quella dei modi di indirizzamento *PC-relativo*, detti anche *PC con offset*

Indirizzamento PC-relativo

- In un modo di indirizzamento *PC-relativo*, l'operando è una parola di memoria il cui indirizzo è la somma del contenuto di PC e di una costante intera, detta *offset*
- Quando viene eseguita un'istruzione di salto *J* che usa l'indirizzamento PC-relativo, l'indirizzo di destinazione del salto viene calcolato sommando l'offset al valore contenuto in PC nel momento in cui *J* viene eseguita
- Si noti che in molti *ASM-PM*, il contenuto di PC nel momento in cui *J* viene eseguita, è pari non all'indirizzo di *J* ma all'indirizzo di *J* aumentato di un valore costante
- Ciò accade in quanto parallelamente all'esecuzione di ogni istruzione *LM*, PC viene incrementato in modo automatico per velocizzare il calcolo dell'indirizzo dell'istruzione successiva in ordine di memorizzazione, che potrebbe essere la prossima istruzione ad essere eseguita

Indirizzamento PC-relativo

- Se si vuole che un'istruzione *J* che usa l'indirizzamento PC-relativo salti ad un'istruzione *I*, è necessario calcolare in modo corretto il valore dell'offset da utilizzare
- Tale valore deve essere calcolato a partire dalla differenza tra gli indirizzi di *I* e *J*, tenendo anche conto di quale sarà il valore di PC al momento dell'esecuzione di *J*, il che dipende dallo specifico *ASM-PM*
- Fortunatamente, la maggior parte degli *ASM* consentono di indicare l'istruzione di destinazione del salto semplicemente con una label, lasciando che sia l'assembler, in fase di traduzione, a calcolare il valore corretto dell'offset

Indirizzamento PC-relativo

- In alcuni *ASM-PM*, l'indirizzamento PC-relativo può essere usato anche per accedere a dati
- Ad esempio in MC68000 il modo di indirizzamento PC-relativo può essere usato da molte istruzioni di trasferimento dati e aritmetico-logiche (ad esempio da `move` o da `add`), ma solo per gli operandi che non vengono modificati dall'istruzione

Indirizzamento delle Istruzioni in MIPS32-MARS

- L'istruzione `b` e tutte le istruzioni di salto condizionato usano il modo di indirizzamento PC-relativo
  - L'offset è una stringa binaria di 18 cifre, interpretata in complemento a 2
  - L'indirizzo di destinazione del salto è pari alla somma tra l'offset e l'indirizzo dell'istruzione di salto aumentato di 4
  - Quindi la distanza massima tra l'indirizzo dell'istruzione di salto e quello di destinazione del salto è circa 128 Kbytes
- In MARS, l'indirizzo dell'istruzione di destinazione del salto deve essere obbligatoriamente espresso da una label
- In fase di traduzione, MARS calcola automaticamente il valore dell'offset, a partire dalla label

Indirizzamento delle Istruzioni in MIPS32-MARS

- L'istruzione `j` utilizza un modo di indirizzamento diretto-memoria
- In tale istruzione, l'indirizzo dell'istruzione di destinazione del salto deve essere obbligatoriamente espresso da una label, ed è soggetto ad un vincolo: esso deve avere le 4 cifre binarie più significative uguali a quelle del contenuto di PC al momento dell'esecuzione di `j`
- Si noti che tale vincolo è assente nelle istruzioni di trasferimento dati che usano l'indirizzamento diretto-memoria: questa differenza dipende dal fatto che le istruzioni di salto vengono tradotte in *LM* in modo diverso dalle istruzioni di trasferimento dati

Indirizzamento delle Istruzioni in MIPS32-MARS

- MIPS32 dispone anche di un'istruzione di salto incondizionato con destinazione *dinamica*
- Si tratta dell'istruzione `jr`, che ha come unico operando uno dei GPR
- Tale istruzione usa il modo di indirizzamento indiretto-registro: l'indirizzo di destinazione del salto è il contenuto del registro

Indirizzamento delle Istruzioni in MC68000-ASM1

- L'istruzione `bra` e le istruzioni di salto condizionato usano il modo di indirizzamento PC-relativo
  - L'offset è una stringa binaria di 16 cifre, interpretata in complemento a 2
  - L'indirizzo di destinazione del salto è pari alla somma tra l'offset e l'indirizzo dell'istruzione di salto aumentato di 2
  - Quindi la distanza massima tra l'indirizzo dell'istruzione di salto e quello di destinazione del salto è circa 32 Kbytes
- In ASM1, l'operando di tali istruzioni può essere
  - Una costante numerica k, che viene usata direttamente come valore dell'offset
  - Una label, che rappresenta l'indirizzo di destinazione del salto desiderato dal programmatore: in questo caso l'assembler calcola automaticamente il valore dell'offset, a partire dalla label

Indirizzamento delle Istruzioni in MC68000-ASM1

- L'istruzione `jmp` può usare diversi modi di indirizzamento per dati in memoria
- In LPS ci limitiamo a considerare i più usati, ovvero diretto-memoria e indiretto-registro
- Tramite l'indirizzamento diretto-memoria, si può specificare staticamente l'intero indirizzo di destinazione del salto
- Quindi `jmp` può saltare, incondizionatamente, ad un'istruzione che può trovarsi ad un qualunque indirizzo, anche lontano
- Usando l'indirizzamento indiretto-registro, si specifica che l'indirizzo di destinazione del salto è il contenuto di un registro indirizzi
- Quindi `jmp`, usata con tale modo di indirizzamento, effettua un salto incondizionato con destinazione dinamica ad un qualunque indirizzo, anche lontano

Istruzioni come Dati

- Il fatto che le istruzioni siano memorizzate e abbiano indirizzi, rende possibile scrivere in *ASM* programmi che utilizzano come dati gli indirizzi delle istruzioni o addirittura le stesse traduzioni *LM* delle istruzioni del programma
- Ciò permette di realizzare
  - salti con destinazione dinamica
  - salti ad istruzioni i cui indirizzi sono calcolati dinamicamente con operazioni aritmetiche
  - programmi che copiano e modificano altri programmi
  - programmi che si auto-modificano