Architettura degli Elaboratori

Primavera 2020

Prof. Tramontana
www.dmi.unict.it/~tramonta

Hamacher Capitolo 2 Istruzioni Macchina

Obiettivi del capitolo

- Istruzioni macchina ed esecuzione programmi
- Metodi di indirizzamento per accedere alla memoria
- Linguaggio Assemblativo (Assembly) per rappresentare istruzioni macchina
- Pile e sottoprogrammi

Introduzione

- Un processore sa eseguire, ovvero riconosce, un insieme di istruzioni in **formato eseguibile** (**macchina**), per un processore tale insieme si dice Instruction Set Architecture (ISA)
- L'insieme ISA specifica
  - Il nome delle istruzioni e le operazioni corrispondentemente svolte
  - Il modo con cui si possono manipolare i dati
  - Le regole di combinazione delle varie istruzioni
- L'ISA è l'**interfaccia fra hardware e software** di un processore
- Si userà un ISA non specifico di alcun processore, somigliante a molti ISA reali
- Un programma implementato in un linguaggio ad alto livello, es. C++ o Java, è **tradotto** dal compilatore in linguaggio macchina
- Il **linguaggio Assemblativo** è una rappresentazione leggibile del **linguaggio macchina**

Memoria del calcolatore (S. 2.1)

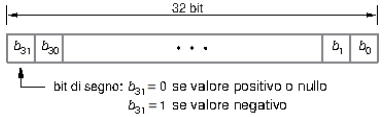
- Dati e istruzioni sono contenuti in memoria
- La memoria è costituita da celle elementari, ciascuna capace di memorizzare un bit

1	0	1	1

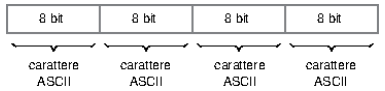
- Non è pratico usare una singola cella da sola poiché rappresenta un'informazione troppo piccola
- Si usano gruppi di celle, un gruppo di ***n*** celle da un bit si chiama **parola di memoria** (memory **word**). Dove n è la dimensione della parola
- Si usano parole di lunghezze da 16 a **64 bit** (potenze di 2, tipicamente)
- La memoria è formata da una successione di parole

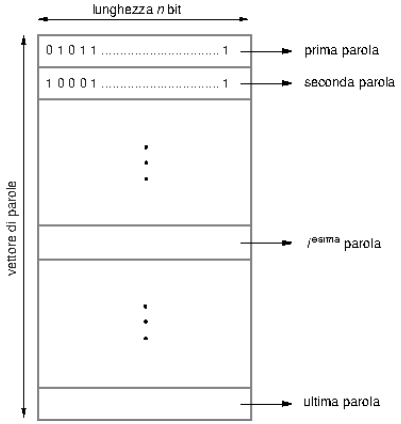
Organizzazione della memoria

- Le parole di n bit ciascuna sono memorizzate una dopo l'altra
- Una parola di memoria di 32 bit può contenere un numero di lunghezza 32 bit



- Oppure, una parola di 32 bit può contenere 4 caratteri ASCII ciascuno di 8 bit, ovvero 1 byte



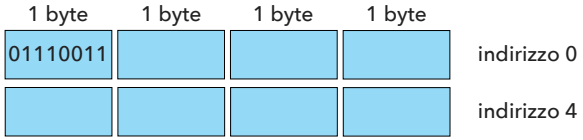


Indirizzamento

- Per leggere un dato dalla memoria (o per scriverlo) è necessario identificare la parola di memoria tramite un **indirizzo** (che costituisce la posizione della parola in memoria)
- Un indirizzo è un numero naturale da 0 a 2^m -1 (con m>0), quindi occorrono ***m bit*** per rappresentare un indirizzo
- La quantità totale di indirizzi 2^m si dice **spazio di indirizzamento** (**addressing space**)
- Con m = 24, ovvero si usano 24 bit per rappresentare un indirizzo, si possono indirizzare 2^{24} parole di memoria
- $2^{24} = 2^4 * 2^{20} =$ 16 M parole (16 Mega parole)

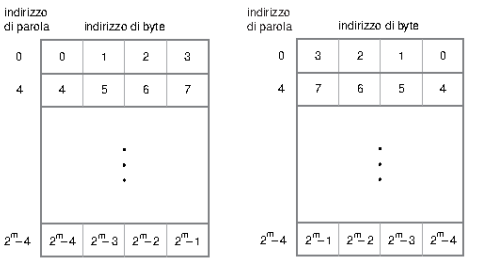
Indirizzamento per byte

- Si abbia un'organizzazione della memoria secondo parole di memoria, e si abbia una parola di 32 bit, ovvero 4 byte
- Con l'indirizzamento per byte, ogni indirizzo permette di indicare la posizione del singolo byte
- Due parole di memoria consecutive avranno indirizzi che differiscono di 4 (ovvero della lunghezza in byte della parola)
- Per la parola con indirizzo 0, i byte consecutivi avranno indirizzi 0, 1, 2, 3



Ordinamento di byte

- Nella memoria indirizzabile per byte, l'ordine dei byte di una parola ha due alternative di indirizzamento: (a) e (b)
- L'ordine di enumerazione dei bit all'interno del byte o della parola, frequentemente, presenta il **bit più significativo a sinistra** (perché è come nell'aritmetica), ovvero per un byte si ha b_7, b_6, .. b_1, b_0
- Quando una parola ha indirizzo multiplo della lunghezza della parola in byte (0, 4, 8, .. per parole di 4 byte), si dice che lo schema di indirizzamento è **allineato**



(a) schema *crescente* o *big-endian* (b) schema *decrescente* o *little-endian*

Operazioni di memoria (S. 2.2)

- Le due operazioni di base sono
  - **Lettura** (**read**), o prelievo (fetch), o caricamento (**load**), di un dato da un indirizzo di memoria e scrittura all'interno del processore (in un registro)
  - **Scrittura** (**write**), o memorizzazione, o immagazzinamento (**store**) di un dato presente in un registro del processore a un indirizzo di memoria
  - La realizzazione di queste istruzioni si affronterà dopo. Si considera adesso il linguaggio macchina, astrazione ISA, messo a disposizione dal processore, e che il programmatore usa. Questo linguaggio è l'interfaccia fra hardware e software

Istruzioni macchina di base (S. 2.3)

- Ci sono quattro tipi essenziali di istruzioni, presenti in ogni ISA
  - **Trasferimento** di dati tra memoria e registri interni al processore
  - **Operazioni aritmetiche e logiche** sui dati
  - **Controllo della sequenza** di esecuzione delle istruzioni
  - **Operazioni di I/O** per il trasferimento dei dati fra unità di I/O e registri del processore

Notazione RTN

- Una notazione o formalismo per rappresentare istruzioni è la Register Transfer Notation (RTN), e rappresenta la copia di dati fra locazioni di memoria e registri. Si usano
  - Costanti numeriche o simboliche per indirizzi di parole di memoria, es. `IND`
  - Nomi predefiniti per i registri del processore, es. `R1`, `R2`, etc.
  - Parentesi quadre per indicare il contenuto della memoria indirizzata, es. `[IND]` è il contenuto della memoria alla locazione di nome `IND`

`R2 ← [LOC]`
- Indica il trasferimento nel registro `R2` del contenuto della memoria di indirizzo `LOC`

`R4 ← [R2] + [R3]`
- Indica il trasferimento in `R4` della somma dei contenuti dei registri `R2` e `R3`
- A sinistra della freccia deve esserci un solo elemento, capace di contenere un valore, quindi un registro o una locazione di memoria. Solo la locazione di destinazione viene cambiata

Notazione simbolica

- La notazione simbolica è adatta a rappresentare il linguaggio macchina e rappresenta un linguaggio detto linguaggio macchina simbolico, o linguaggio Assemblativo (Assembly)
- Un'**istruzione** specifica un'**operazione** da eseguire e gli **operandi** coinvolti
- Si usano parole inglesi (es. `Load`, `Store`, `Add`) che indicano varie operazioni, e operandi che indicano sorgenti e destinazioni dei dati. Nei processori in commercio tali parole possono essere sostituite da codici mnemonici (es. `LD`, `ST`)

`Load R2, LOC`
- col significato di `R2 ← [LOC]`

`Add R3, R2, R1`
- col significato di `R3 ← [R2] + [R1]`

Architetture RISC E CISC

- Nella progettazione di processori e quindi delle relative ISA vi sono due approcci
- RISC: **Reduced** instruction set computer
  - Ogni istruzione occupa esattamente una parola di memoria e gli operandi necessari sono già nel processore
  - La realizzazione del circuito che esegue le istruzioni è più semplice, veloce, e può essere organizzato a stadi (pipeline)
  - Lo spazio ridotto che deve occupare una singola istruzione vincola la complessità e il numero di tipi di istruzioni disponibili
- CISC: **Complex** instruction set computer
  - Istruzioni di lunghezza variabile, che possono estendersi per più di una parola
  - Molti tipi diversi di istruzioni e istruzioni che indicano operazioni complesse

Istruzioni RISC

- Ogni istruzione occupa una sola parola di memoria
- Si dice che si ha un'architettura load/store
  - L'accesso a operandi in memoria avviene solo tramite le istruzioni Load e Store
  - Gli operandi di un'istruzione aritmetica o logica sono nei registri del processore, o in modo immediato dentro la parola dell'istruzione
- Formati di istruzioni (d = destinazione, s = sorgente)

```
Load d, s
Store s, d
Add d, s1, s2
```

- La destinazione `d` dell'istruzione `Load` è un registro del processore e la sorgente `s` è una locazione di memoria, mentre per `Store` la `s` è un registro e la `d` è la locazione di memoria. Quindi gli **operandi** indicano **prima i registri** e **dopo gli indirizzi**
- L'istruzione `Add` ha tre operandi; destinazione `d` e sorgenti, `s1` e `s2`, possono essere solo registri

Operazione di addizione

- Si consideri l'operazione aritmetica di addizione, espressa con un linguaggio ad alto livello

```
C = A + B
```

- I valori di due variabili, `A` e `B`, sono sommati e il risultato è assegnato alla variabile `C`
- La notazione RTN sarà

```
C ← [A] + [B]
```

- Per essere effettuata da semplici istruzioni macchina avremo, ovvero con la notazione simbolica si ha

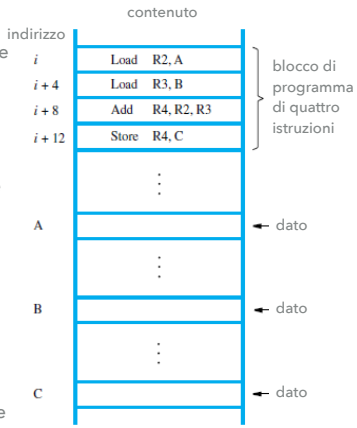
```
Load R2, A
Load R3, B
Add R4, R2, R3
Store R4, C
```


Esecuzione di istruzioni

- Le istruzioni del programma occupano ciascuna una parola di memoria di lunghezza 32 bit e la memoria è indirizzabile a byte
- Le istruzioni sono contenute in parole consecutive, la prima all'indirizzo i
- Nel registro **PC**, contatore di programma, si trova l'indirizzo della prossima istruzione da eseguire, ovvero i nel caso iniziale
- I circuiti di controllo prelevano l'istruzione puntata da PC e aggiornano il PC (per l'esecuzione sequenziale incrementano PC di 4). Questa è la **prima fase** detta di **prelievo** (**fetch**)

```
IR ← [[PC]]
```

- Viene letta l'istruzione presente all'indirizzo contenuto nel registro PC e scritta in **IR**, registro di istruzione
- La **seconda fase** è detta di **esecuzione**: il contenuto di IR viene decodificato e l'istruzione viene eseguita. L'esecuzione può comportare vari calcoli e attività ausiliarie



Esecuzione di n somme

- Si abbia una lista di n numeri interi (n>1). Gli indirizzi delle parole da sommare sono indicati in modo simbolico NUM1, NUM2, ..
- Le istruzioni calcolano la somma progressivamente e la accumulano nel registro R2. Al termine la somma viene memorizzata all'indirizzo SOMMA

```
Load R2, NUM1
Load R3, NUM2
Add R2, R2, R3
Load R3, NUM3
Add R2, R2, R3
..
Load R3, NUMn
Add R2, R2, R3
Store R2, SOMMA
```


Esecuzione con salto

- Anziché scrivere n volte la somma, si ricorre a un ciclo. Si carica un numero dalla memoria, si somma con l'accumulatore, si ripete finché vi sono numeri
  - N, CICLO e SOMMA sono costanti (indirizzi di memoria) che permettono di accedere al valore corrispondente in memoria
  - R2 è usato come contatore del numero di passate del ciclo
  - R3 è usato come accumulatore
  - R5 contiene il numero caricato dalla memoria
  - Un valore immediato fornito nell'istruzione viene preceduto dal simbolo #

```
      Load R2, N
      Clear R3
CICLO: Determina l'indirizzo del prossimo numero
      Carica il prossimo numero in R5
      Add R3, R3, R5
      Subtract R2, R2, #1
      Branch_if_[R2]>0 CICLO
      Store R3, SOMMA
```