Linguaggio macchina

Capitolo 10

Linguaggio macchina

- Insieme delle istruzioni (instruction set) che la CPU può eseguire

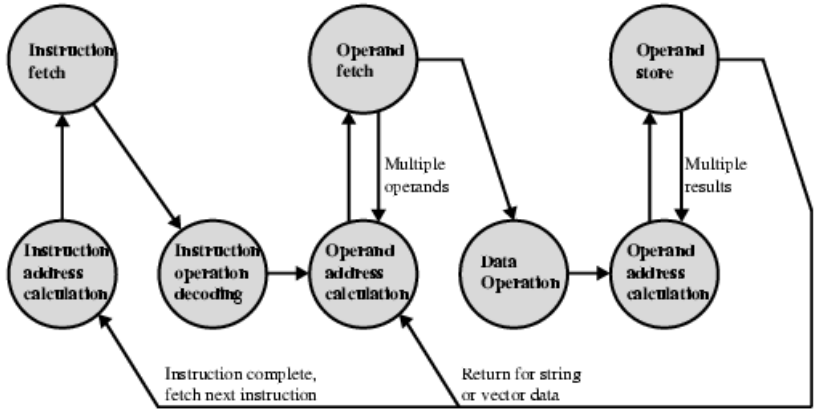
Elementi di un'istruzione macchina

- Codice operativo
  - Specifica l'operazione da eseguire
- Riferimento all'operando sorgente
  - Specifica l'operando che rappresenta l'input dell'operazione
- Riferimento all'operando risultato
  - Dove mettere il risultato
- Riferimento all'istruzione successiva

Dove sono gli operandi?

- Memoria centrale (o virtuale)
  - Si deve fornire l'indirizzo
- Registri della CPU
  - Ognuno ha un numero che lo identifica
- Dispositivi di I/O
  - Numero modulo o indirizzo di M

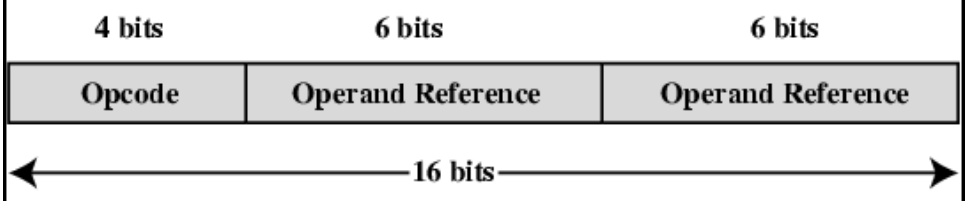
Ciclo della CPU



Rappresentazione delle istruzioni

- Istruzione = sequenza di bit
  - Divisa in campi
- Spesso viene usata una rappresentazione simbolica delle configurazioni di bit
  - es.: ADD, SUB, LOAD
- Anche gli operandi hanno una rappresentazione simbolica
  - Es.: ADD A,B

Esempio del formato di un'istruzione



Tipi di istruzioni

- Elaborazione dati
  - Istruzione aritmetiche e logiche, di solito sui registri della CPU
- Immagazzinamento dei dati in M o viceversa
- Trasferimento dei dati (I/O)
- Controllo del flusso del programma
  - Salto con o senza test

Quanti indirizzi sono necessari in una istruzione?

- Indirizzi necessari:
  - Un indirizzo per ogni operando (1 o 2)
  - Uno per il risultato
  - Indirizzo istruzione successiva
- Quindi al massimo quattro indirizzi
  - Ma molto raro, e sarebbe molto dispendioso
- Di solito 1, 2 o 3 per gli operandi/risultati

Numero di indirizzi

- 1 indirizzo
  - il secondo indirizzo è implicito
  - di solito si tratta di un registro (accumulatore)
  - situazione tipica nei primi calcolatori

Numero di indirizzi

- Zero indirizzi
  - tutti gli indirizzi sono impliciti
  - utilizza una pila (stack)
    - Ad esempio, c = a + b è realizzato come segue

```
push a
push b
add
pop c
```

Numero di indirizzi

- Meno indirizzi ➔ istruzioni più elementari (e più corte), quindi CPU meno complessa
  - Però più istruzioni per lo stesso programma ➔ tempo di esecuzione più lungo
- Più indirizzi ➔ istruzioni più complesse
- Indirizzo di M o registro: meno bit per indicare un registro
- RISC (Reduced Instruction Set Computer) verso CISC (Complex Istruction Set Computer)
  - Pentium sono CISC, POwerPC (Apple, IBM, Motorola) sono RISC

Tipi degli operandi

- Indirizzi (interi senza segno)
- Numeri
  - Limite al modulo
  - Limite alla precisione
- Caratteri
- Dati logici

Numeri

- Interi (virgola fissa)
- Virgola mobile
- Quando ci sono soprattutto operazioni di I/0, si usano i decimali impaccati
  - Cifra decimale = 4 bit (0=0000, 1=0001, 2=0010, ..., 8=1000, 9=1001)
  - Inefficiente: solo 10 delle 16 configurazioni vengono usate
  - Es.: 246 = 0010 0100 0110
  - Più lungha della notazione binaria, ma evita la conversione

Caratteri

- Codice ASCII (American Standard Code for Information Exchange)
- Carattere = 7 bit ➔ 128 caratteri in totale
- Caratteri alfabetici + caratteri di controllo
- Di solito 8 bit: un bit per controllo di errori di trasmissione (controllo di parità)
  - Settato in modo che il numero totale di bit a 1 sia sempre pari (o sempre dispari)
  - Es.: 00011100 ➔ ottavo bit a 1
  - Se si ricevono 8 bit con n.ro dispari di 1, c'è stato un errore di trasmissione

Dati logici

- n bit, invece che un singolo dato
- Per manipolare i bit separatamente

Progettare un insieme di istruzioni

- Repertorio
  - quante e quali operazioni
- Tipi di dato
  - su quali dati
- Formato
  - lunghezza, numero indirizzi, dimensione campi, ...
- Registri
  - numero dei registri della CPU indirizzabili dalle istruzioni
- Indirizzamento
  - modo di specificare gli indirizzi degli operandi

Esempio di un linguaggio macchina

- È un esempio non reale
- Molto semplice
- Serve per far vedere i tipi pricipali di istruzioni
- Non lo useremo in laboratorio

Istruzioni per trasferimento dati

- In realtà, non è un trasferimento ma una copia
- LOAD: da memoria a registro
- STORE: da registro a memoria
- Anche input/output

Istruzioni logico/aritmetiche

- Operazioni aritmetiche: somma, ...
- Operazioni logiche: and, or, xor, anche shift e rotate

Istruzioni di controllo

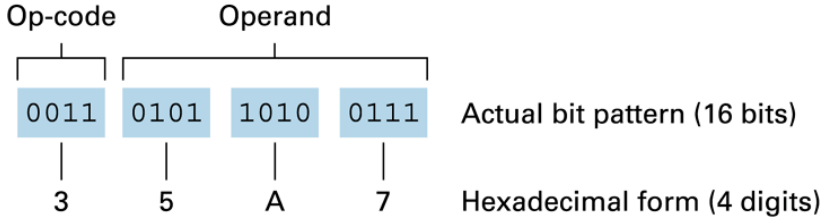
- Regolano l'esecuzione del programma
- Es.: stop
- Anche istruzioni di salto: se l'istruzione da eseguire non è la successiva nella lista
- Salto condizionato o no
- Es.: salta al passo 5, o salta al passo 5 se il valore ottenuto è 0

Divisione di due valori in memoria

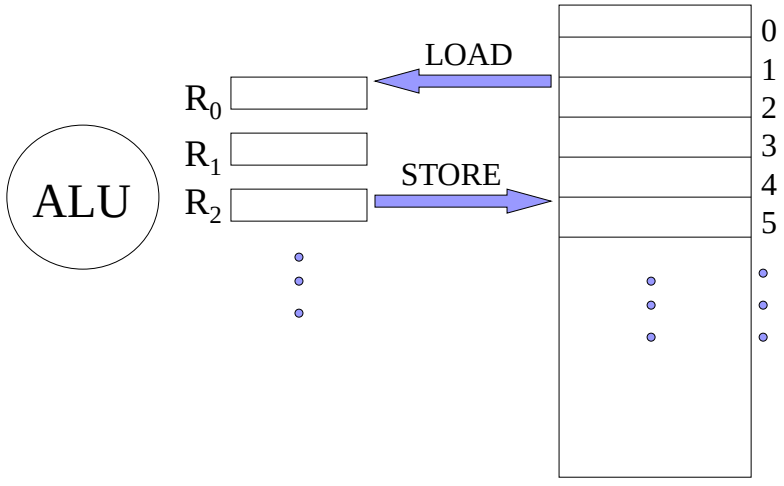
1. Carica in un registro un valore in memoria (LOAD)
2. Carica in un altro registro un altro valore in memoria (LOAD)
3. Se questo secondo valore è 0, salta al passo 6 (salto condizionato)
4. Dividi il contenuto del primo registro per quello del secondo registro e metti il risultato in un terzo registro (op. aritmetica)
5. Archivia il contenuto del terzo registro in memoria (STORE)
6. STOP

Istruzione macchina

- Due parti (campi):
- Campo codice operativo: quale operazione eseguire
- Campo operando: diverso a seconda dell'operazione

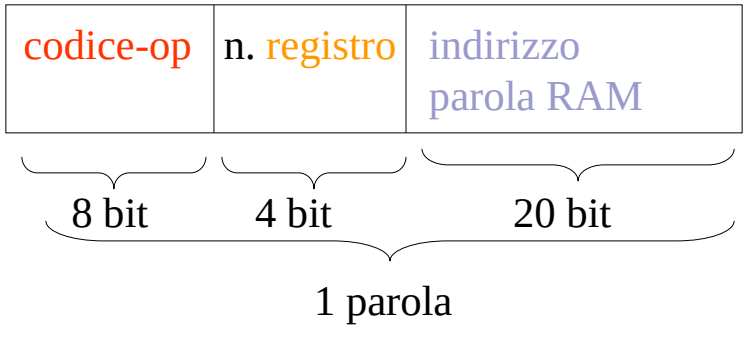


Istruzioni di trasferimento: registri ⇔ RAM



Formato:

in binario!

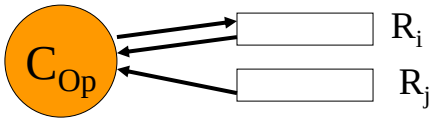


Codici:

LOAD	00000000
STORE	00000001

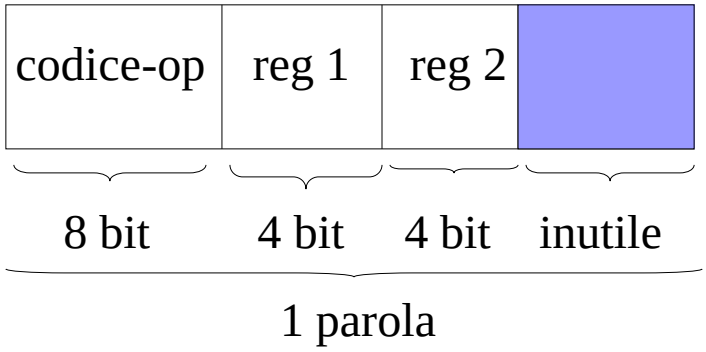
ARITMETICHE

eseguono somma, differenza, moltiplicazione e divisione usando i registri come operandi

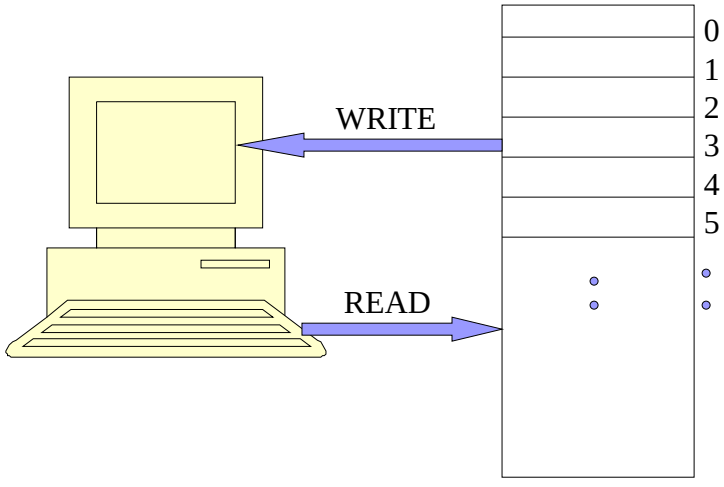


ADD	00000010	FADD	00000011
SUB	00000100	FSUB	00000101
MULT	00000110	FMULT	00000111
DIV	00001000	FDIV	00001001
MOD	00001010		

FORMATO:

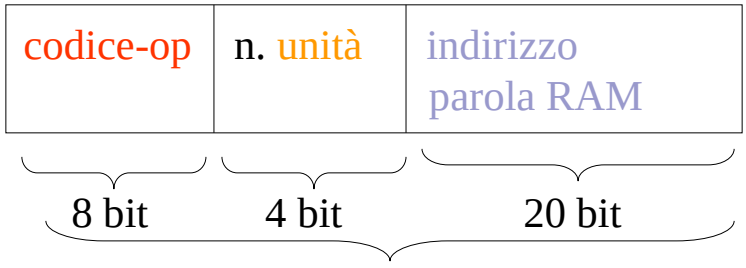


Istruzioni di input/output:
unità I/O ⇔ RAM



Formato:

in binario!



Codici:

```
READ    00010000
WRITE   00010001
```

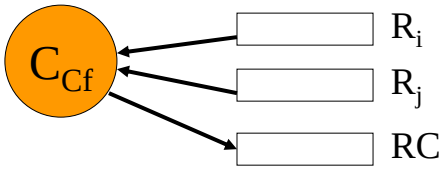
Unità:

```
STINP   0000 (tastiera)
STOUT   0001 (video)
```

Confronto

paragona il contenuto di 2 registri R_i ed R_j e:

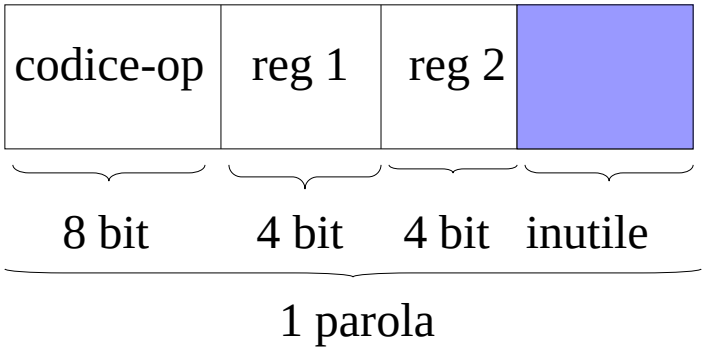
- se $R_i < R_j$ mette -1 nel registro RC
- se $R_i = R_j$ mette 0 in RC
- se $R_i > R_j$ mette 1 in RC



Codici:

```
COMP    00100000
FCOMP   00100001
```

FORMATO:



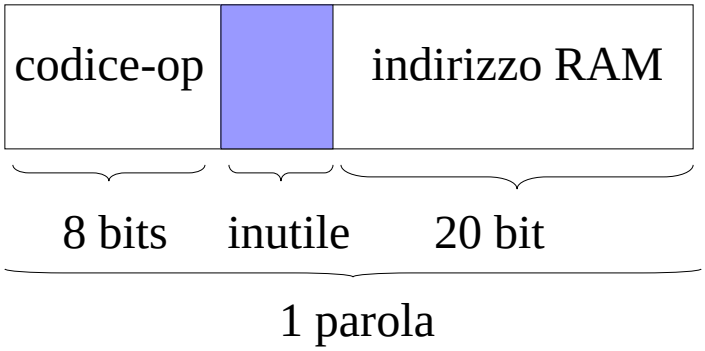
Salto

istruzioni che permettono di saltare ad un'altra istruzione del programma a seconda del contenuto di RC (cioè a seconda del risultato di un confronto)

BRLT	**01000001**	**BRNE**	**01000100**
BRLE	**01000010**	**BRGE**	**01000110**
BREQ	**01000011**	**BRGT**	**01000101**
	BRANCH	**10000000**	

Anche salto incondizionato!

FORMATO:

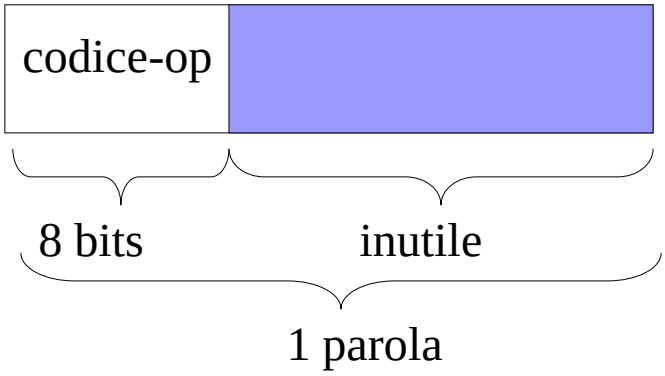


STOP

termina il programma

Codice: `STOP 10000001`

FORMATO:



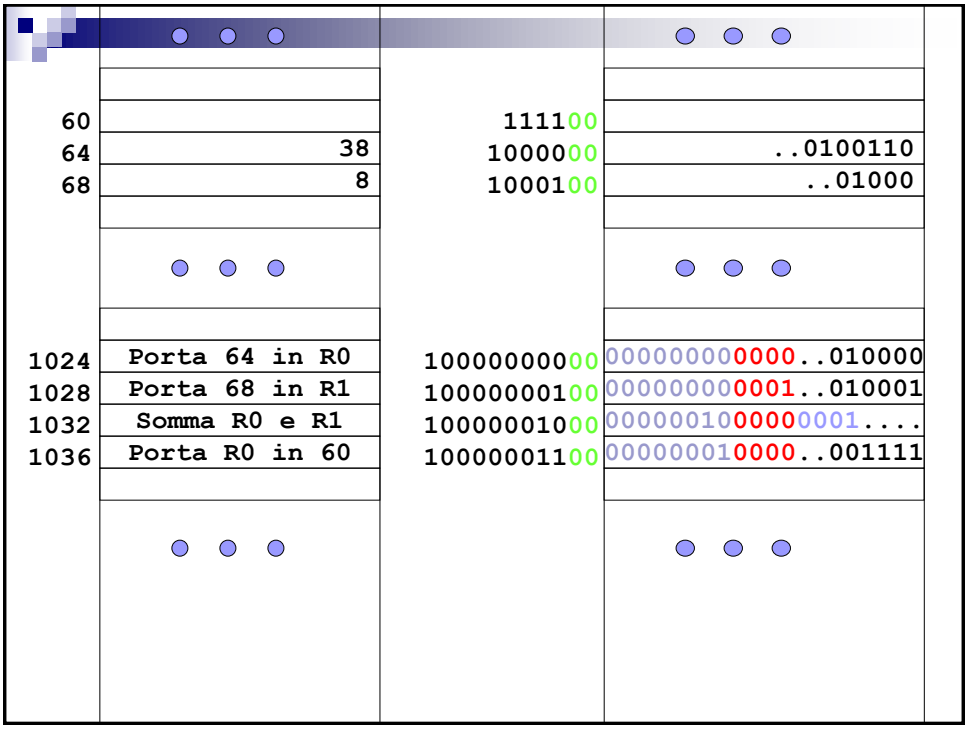
Esempio

Scriviamo un programma macchina che:

- trasferisce il contenuto di 2 parole di indirizzo 64 e 68 della RAM nei registri R_0 ed R_1
- li somma
- trasferisce la somma nella parola di indirizzo 60 della RAM

Codici delle operazioni

- trasferimento RAM → CPU: **00000000**
- trasferimento CPU → RAM: **00000001**
- somma : **00000010**



Svantaggi del linguaggio macchina:

- programmi in binario sono difficili da scrivere, capire e cambiare
- il programmatore deve occuparsi di gestire la RAM: difficile ed inefficiente

primo passo → Assembler

Novità dell'Assembler

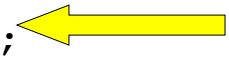
- codici mnemonici per le operazioni
- nomi mnemonici (identificatori) al posto degli indirizzi RAM per i dati (e indirizzi RAM delle istruzioni usate nei salti)
- tipi dei dati **`INT`** e **`FLOAT`**

Codice-op mnemonici:

- trasferimento: **LOAD** (RAM → CPU) e **STORE** (CPU → RAM)
- aritmetiche: **ADD,SUB,DIV,MULT,MOD, FADD,FSUB,FDIV,FMULT**
- input/output: **READ** (U-INP → CPU), **WRITE** (CPU → U-OUT)
- test: **COMP, FCOMP**
- salto: **BREQ,BRGT,BRLT,BRGE,BRLE, BRANCH**
- terminazione: **STOP**

Stesso esempio del linguaggio macchina

```
Z : INT ;
X : INT 38;
Y : INT 8;
LOAD R0 X;
LOAD R1 Y;
ADD  R0 R1;
STORE R0 Z;
```



dichiarazioni degli identificatori dei dati

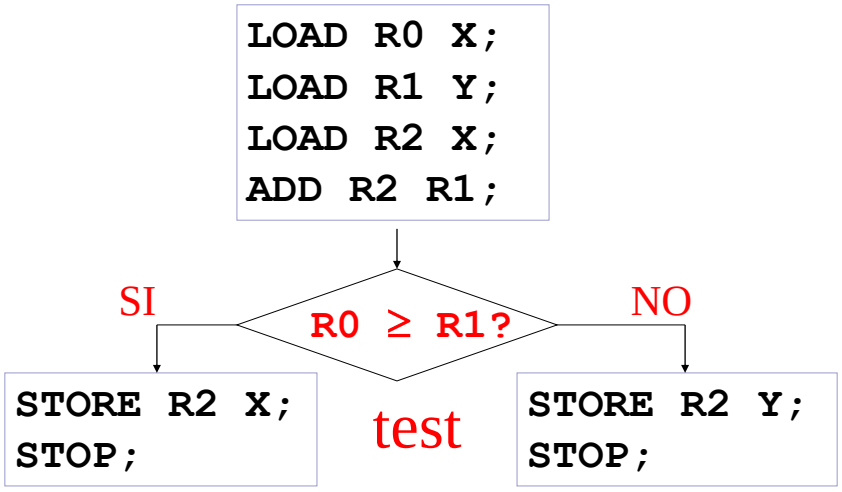
istruzioni assembler

Esempio

carica due valori dalla RAM, li somma e mette il risultato al posto del maggiore dei 2 numeri sommati (nel caso siano uguali, non importa in quale dei due si mette la somma)

```
X:  INT 38;
Y:  INT 8;
      LOAD R0 X;
      LOAD R1 Y;
      LOAD R2 X;
      ADD R2 R1;
      COMP R0 R1;
      BRGE pippo;
      STORE R2 Y;
      STOP;
pippo: STORE R2 X;
      STOP;
```

Flowchart


LOAD R0 X;
LOAD R1 Y;
LOAD R2 X;
ADD R2 R1;
R0 ≥ R1?
SI
NO
STORE R2 X;
STOP;
test
STORE R2 Y;
STOP;