esercizi

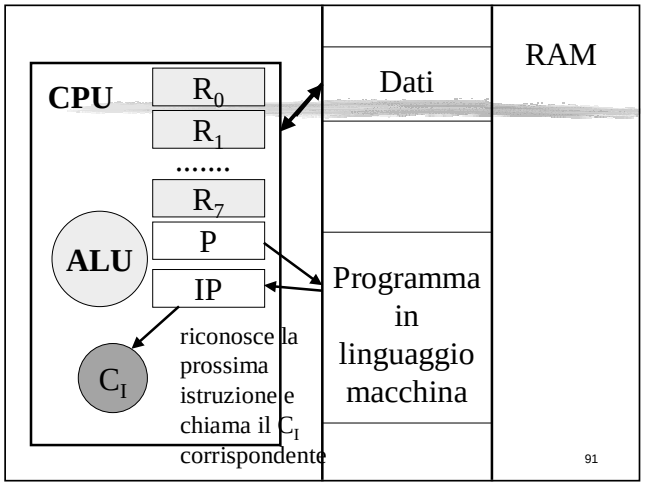
•dare la rappresentazione in virgola mobile normalizzata dei valori 0.5, 1.5 e 10.543 avendo 8 bits per l'esponente e 8 per la mantissa.

89

LINGUAGGIO MACCHINA e ASSEMBLER

Useremo il linguaggio macchina di una CPU "MINIMA" ed il corrispondente linguaggio Assembler "MINIMO".

90



91

4 tipi di istruzioni macchina:

1) di trasferimento tra RAM e registri della CPU

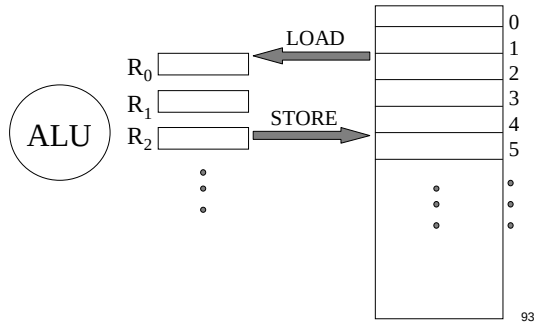
2) aritmetiche: somma,differenza, moltiplicazione, e divisione

3) di input/output

4) di confronto e salto e di stop

92

Istruzioni di trasferimento: registri ⇔ RAM



93

Formato:

in binario!

codice-op	n. registro	indirizzo parola RAM

8 bit 4 bit 20 bit

1 parola

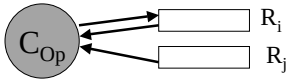
Codici:

LOAD	**00000000**
STORE	**00000001**

94

ARITMETICHE

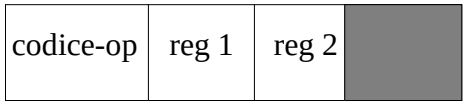
eseguono somma, differenza, moltiplicazione e divisione usando i registri come operandi



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

95

FORMATO:

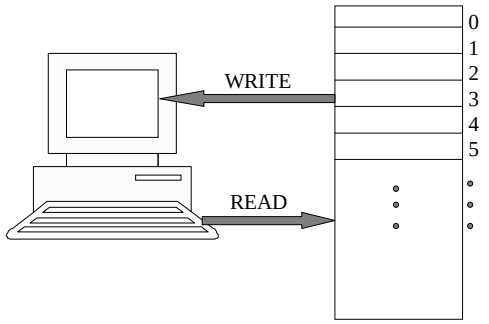


8 bit 4 bit 4 bit inutile

1 parola

96

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



97

Formato:

in binario!

codice-op	n. unità	indirizzo parola RAM

8 bit 4 bit 20 bit

1 parola

Codici:

```
READ   00010000
WRITE  00010001
```

Unità:

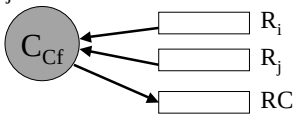
```
STINP  0000 (tastiera)
STOUT  0001 (video)
```

98

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
```

99

FORMATO:

codice-op	reg 1	reg 2	

8 bit 4 bit 4 bit inutile

1 parola

100

Salto

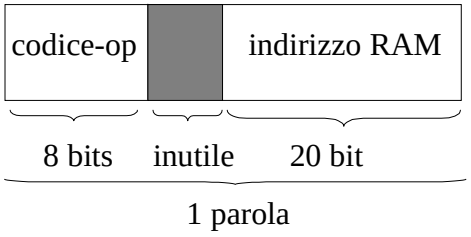
istruzioni che permettono di saltare ad un'altra istruzione del programma a seconda del contenuto di RC (cioè a seconda del risultato di un confronto)

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

Anche salto incondizionato!

101

FORMATO:



102

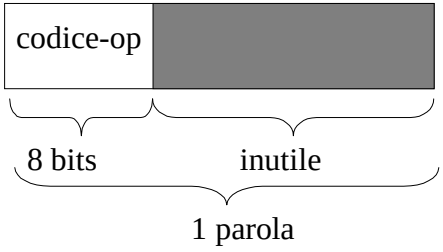
STOP

termina il programma

Codice: `STOP  10000001`

103

FORMATO:



104

esempio

scriviamo un programma macchina che:

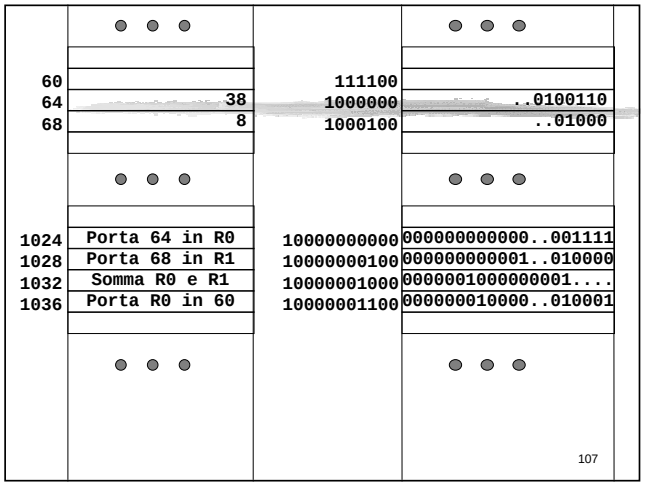
- trasferisce il contenuto di 2 parole di indirizzo 64 e 68 della RAM nei registri R_0 ed R_1
- li somma
- trasferisce la somma nella parola di indirizzo 60 della RAM

105

codici delle operazioni

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

106



107

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

108

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)
- avanzate: tipi dei dati **INT** e **FLOAT**

109

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**

110

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

111

esempio

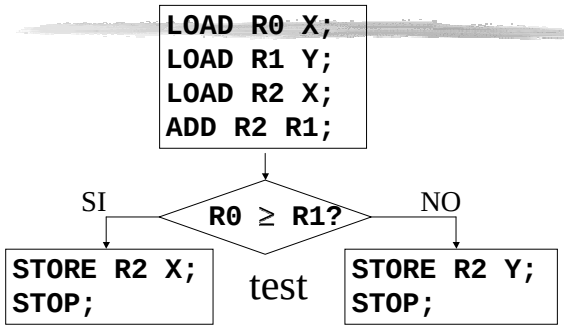
carica due valori dalla RAM, li somma e mette il risultato al posto del maggiore dei 2 numeri sommati (nel caso siano uguali, non importa in quale dei due si mette la somma)

112

```
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;
```

113

flowchart



114

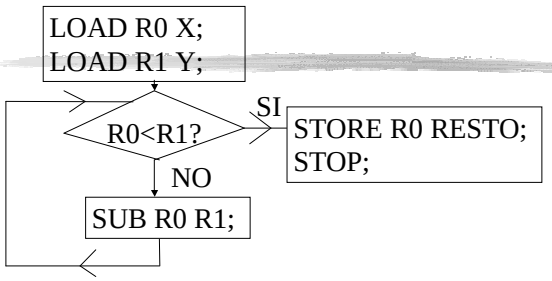
esempio

calcolare il resto di due numeri usando solo la sottrazione

115

```
X: int 356;
Y: int 23;
RESTO: int ;
LOAD R0 X;
LOAD R1 Y;
ciclo: COMP R0 R1;
BRLT fine;
SUB R0 R1;
BRANCH ciclo;
fine: STORE R0 RESTO;
STOP;
```

116



ciclo o iterazione

117

Esempio

Leggere un reale x ed un intero positivo n e calcolare la potenza x^n

Esempio potenza

118


```
X: FLOAT ;
N: INT ;
Ris: FLOAT ;
Uno : INT 1;
Unofl: FLOAT 1.0;
READ STINP X;
READ STINP N;
LOAD R0 Uno;
SUB R0 R0;
LOAD R1 Uno;
LOAD R2 X;
LOAD R3 N;
LOAD R4 Unofl;
```

R0 = 0 intero
R1 = 1 intero
R2 = X reale
R3 = N intero
R4 = 1 reale

119

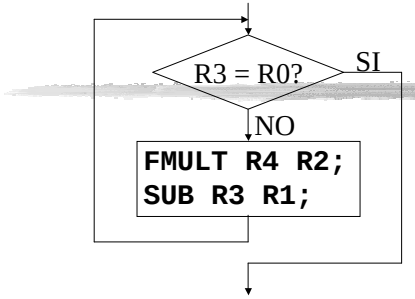
R0 = 0 intero
R1 = 1 intero
R2 = X reale
R3 = N intero
R4 = 1 reale

```
Ciclo: COMP R3 R0;
  BREQ Esci;
  FMULT R4 R2;
  SUB R3 R1;
  BRANCH Ciclo;
Esci: STORE R4 Ris;
  WRITE STOUT Ris;
  STOP;
```

$R4 = X^{N-R3}$

$R4 = X^{N}$

120



ciclo o iterazione

121

Nel programma precedente, per calcolare x^n, il ciclo viene ripetuto n volte.

Il tempo calcolo richiesto aumenterà proporzionalmente con l'aumentare di n.

Diciamo che il programma ha complessità tempo O(n).

Vediamo un altro modo per calcolare x^n.

122

Esempio potenza1

```
X: FLOAT ;
N: INT ;
Ris: FLOAT ;
Due : INT 2;
Unofl: FLOAT 1.0;
READ STINP X;
READ STINP N;
LOAD R0 Due;
SUB R0 R0;
LOAD R1 Due;
LOAD R2 X;
LOAD R3 N;
LOAD R4 Unofl;
LOAD R5 Due;
```

R0 = 0 intero
R1 = 2 intero
R2 = X reale
R3 = N intero
R4 = 1 reale
R5 = 2 intero

123

R0 = 0 intero
R1 = 2 intero
R2 = X reale
R3 = N intero
R4 = 1 reale
R5 = 1 reale

```
Ciclo: COMP R3 R0;
  BREQ Esci;
  SUB R5 R5;
  ADD R5 R3;
  MOD R5 R1;
  COMP R5 R0;
  BREQ Pari;
  FMULT R4 R2;
Pari: FMULT R2 R2;
  DIV R3 R1;
  BRANCH Ciclo;
```

$R4 \cdot R2^{R3} = X^N$

$R4 = X^N$

124

$R4 = X^N$

```
Esci: STORE R4 Ris;
  WRITE STOUT Ris;
  STOP;
```

125

Quante volte viene ripetuto il ciclo nel programma precedente?

All'inizio R3 = n, ad ogni iterazione R3 è diviso per 2, ci si ferma quando R3 = 0.

Quindi il ciclo viene ripetuto $\log_2 n$ volte.

Il tempo calcolo richiesto aumenterà proporzionalmente a $\log_2 n$. Il programma ha complessità tempo $O(\log_2 n)$.

126

Come aumenta il tempo al variare di n.

$f(n)$	n=10	n=100	n=1000	n=10^6	n=10^9
$\log_2 n$	3 μs	6 μs	9 μs	18 μs	27 μs
n	10 μs	100 μs	1 ms	1 s	17 m
n^2	100 μs	10 ms	1 s	278 ore	>$3 \cdot 10^4$ anni
n^3	1 ms	1 s	17 m	>$3 \cdot 10^7$ anni	
2^n	1 ms	>10^{16} anni			
10^n	17 min	>10^{92} anni			

127

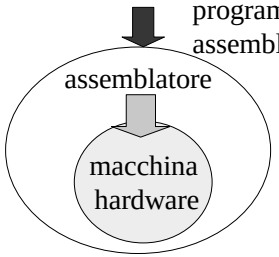
Massimo N che richiede tempo ≤ 1 sec.

$f(n)$	A	B
$\log_2 n$	N=$2^{1000000}$	N=$2^{1000000000}$
n	N=10^6	N= 10^9
n^2	N=1000	N=$3 \cdot 10^4$
n^3	N=100	N=1000
2^n	N=20	N=30
10^n	N=6	N=9

128

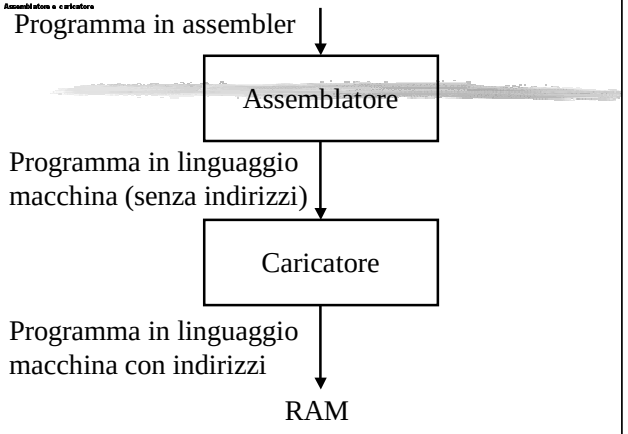
La CPU non "capisce" l'assembler !!

il programma assembler deve essere tradotto in un programma macchina

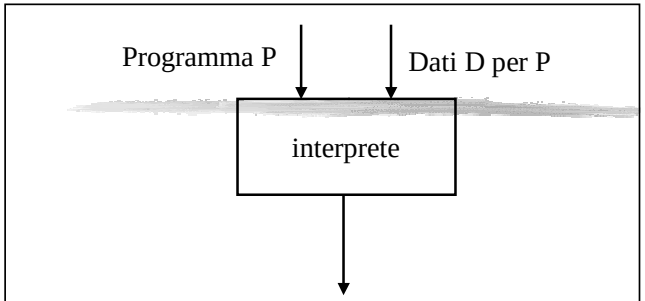


129

Assemblatore e caricatore



130



Gli stessi risultati che si otterrebbero eseguendo il programma P con i dati D

131

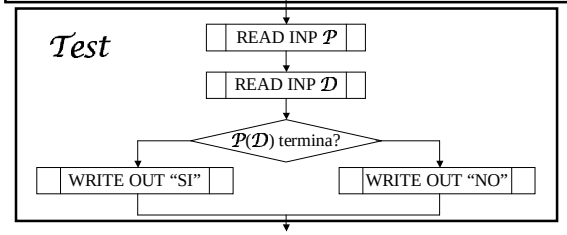
OSSERVAZIONE

Il programma *assemblatore* legge un programma in assembler e il programma *interprete* legge sia un programma che i dati per tale programma.

Esistono quindi programmi che hanno dei programmi come dati.

132

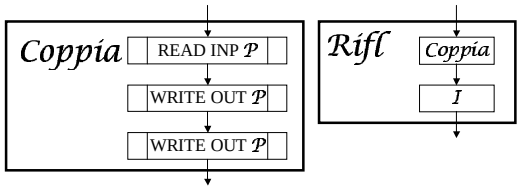
Ha senso quindi cercare un programma *Test* che sia in grado di leggere un qualsiasi programma $\mathcal{P}$ e i relativi dati $\mathcal{D}$ e dire se, eseguendo il programma $\mathcal{P}$ con i dati $\mathcal{D}$, esso termina oppure entra in un ciclo infinito e quindi non termina ?



133

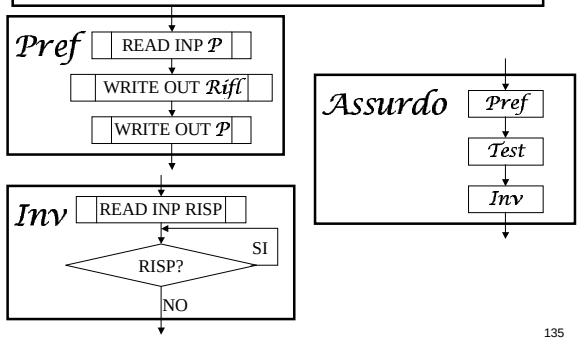
Il programma *Coppia* legge un programma $\mathcal{P}$ e stampa due copie dello stesso programma $\mathcal{P}$.

Componendolo con un interprete I otteniamo il programma *Rifl* che con input un programma $\mathcal{P}$ lo esegue con dati il programma stesso $\mathcal{P}$.



134

Usando il programma ipotetico *Test* ed il programma *Rifl* costruiamo un programma *Assurdo* nel modo seguente.

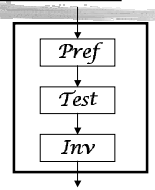


135

Il programma *Assurdo*, con input un qualsiasi programma $\mathcal{P}$, si comporta nel modo seguente:

Se $\mathcal{P}$ con input $\mathcal{P}$ non termina allora *Assurdo* con input $\mathcal{P}$ termina.

Se $\mathcal{P}$ con input $\mathcal{P}$ termina allora *Assurdo* con input $\mathcal{P}$ non termina.



Se *Assurdo* con input *Assurdo* non termina allora *Assurdo* con input *Assurdo* termina.

Se *Assurdo* con input *Assurdo* termina allora *Assurdo* con input *Assurdo* non termina.

136

domande d'esame:

•quali sono le novità principali dell'Assembler rispetto al linguaggio macchina?

•in un programma assembler, perchè si attaccano etichette a certe istruzioni?

137

•come si chiama in Assembler l'istruzione che trasferisce una parola dalla RAM ad un registro della CPU? E quella che compie il trasferimento inverso?

•In assembler a cosa servono gli identificatori o variabili?

138

Informatica di base – 2a parte
A.A. 2002/2003

Sistemi operativi ed applicativi

Sommario degli argomenti

- Sistemi operativi: DOS, Unix/Linux,Windows
- Word processors: Word
- Fogli elettronici: Excel
- Sistemi per la gestione di basi di dati: Access
- Reti: TCP/IP, Internet, ftp, telnet, posta elettronica
- WWW: http, Netscape, HTML, motori di ricerca

140