Fondamenti di Informatica

Ingegneria Clinica

Lezione 06/11/2009

Prof. Raffaele Nicolussi

FUB - Fondazione Ugo Bordoni

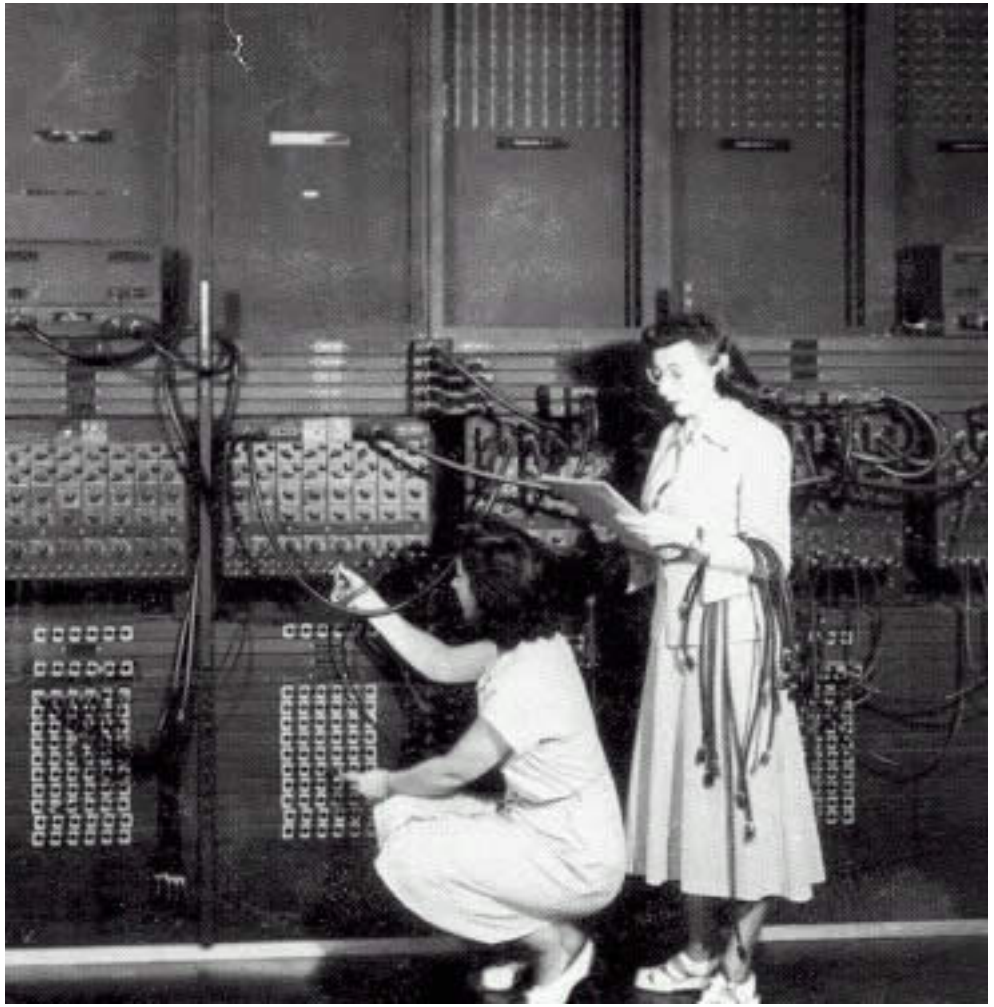
Via B. Castiglione 59 - 00142 Roma

Docente	**Raffaele Nicolussi**	*rnicolussi@fub.it* *0654803323*
Lezioni Aula 54 (ex aula 4) Via del Castro Laurenziano, 7	**Lunedì, Giovedì, Venerdì**	**12:00 – 13:30**
Esercitazioni Aula 15 Via Tiburtina, 205	**Giovedì**	**14:00 – 16.30**
Ricevimento:	**Per appuntamento**	**in FUB, per email, per telefono**
Sito web:	**http://w3.uniroma1.it/IngClinFondinf**	

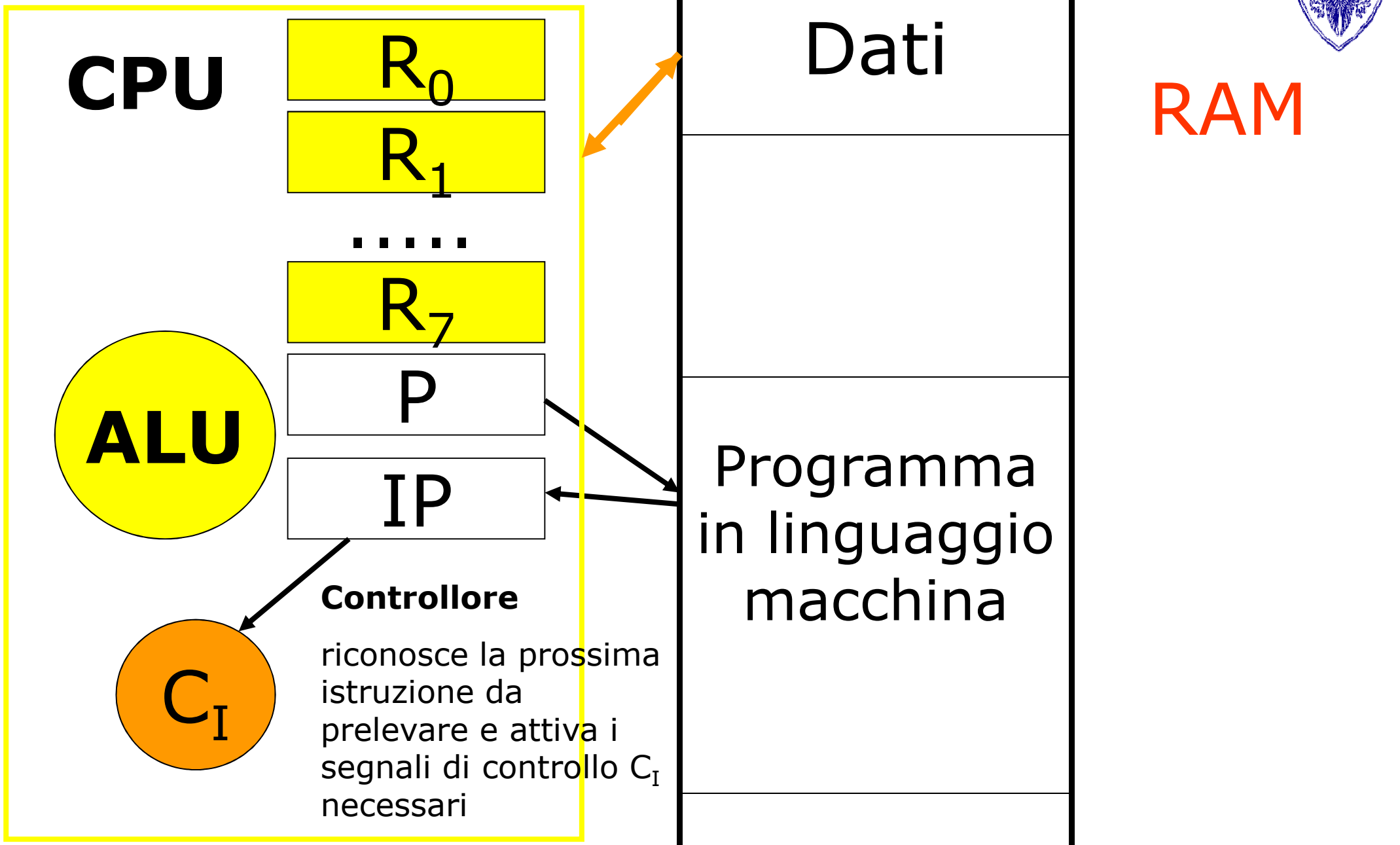
CPU didattica

- Linguaggio macchina elementare
- Istruzioni di trasferimento, aritmetiche, salto e salto condizionato
- Esempio di programma
- Segmento di codice e dati
- Allocazione in memoria di programma e dati
- Le etichette
- Es. somma di due numeri con confronto finale
- Es. potenza di due numeri
- Es. versione più efficiente della potenza

LINGUAGGIO MACCHINA e ASSEMBLER

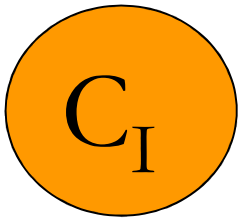


Useremo il linguaggio macchina di una CPU "MINIMA" ed il corrispondente linguaggio Assembler "MINIMO".


CPU
R_0
R_1
.....
R_7
P
IP
ALU
C_I
Controllore
riconosce la prossima istruzione da prelevare e attiva i segnali di controllo C_I necessari
Dati
Programma in linguaggio macchina
RAM

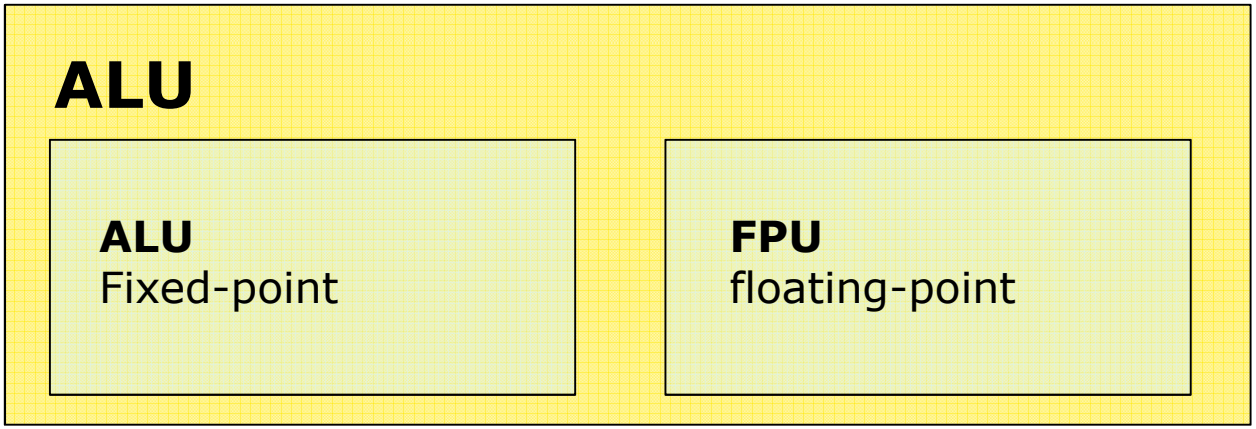
Controllore o unità di controllo

- E' l'unità che si occupa di controllare tutti i moduli della CPU attraverso segnali di controllo
- Compiti della control unit:
  - **Instruction sequencing** : stabilire quale istruzione deve essere prelevata dalla memoria
  - **Instruction interpretation**: attivare i segnali di controllo necessari per l'esecuzione di una istruzione



ALU

- Contiene tutti i circuiti necessari per l'esecuzione di operazioni aritmetiche e logiche
- E' divisa in
  - **ALU fixed-point** : dedicata alle operazioni su numeri interi
  - **FPU floating-point** : dedicata alle operazioni su numeri con la virgola



Floating-point unit

- Due possibilità
  - La FPU è integrata nel calcolatore
  - La FPU risiede nel coprocessore matematico

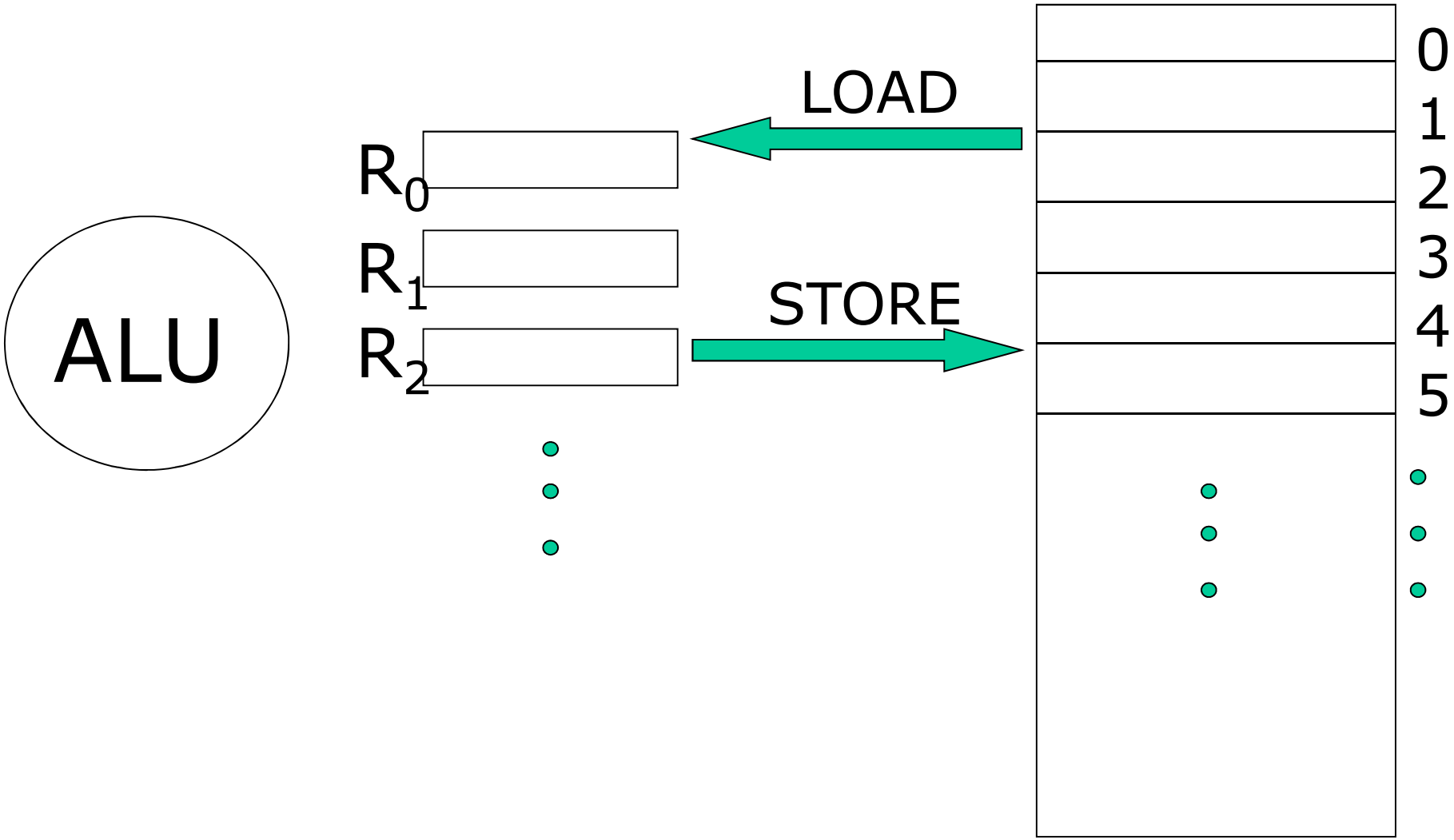
Coprocessore matematico

- Esegue istruzioni in virgola mobile
- E' un modulo separato dalla CPU
- Nella famiglia Intel, fino all'introduzione dell'80486, il coprocessore era fisicamente staccato dalla CPU e connesso al bus di sistema
- Per ogni CPU era disponibile un coprocessore ad-hoc:
  - 8086 -> 8087
  - 80386 -> 80387

4 tipi di istruzioni macchina:

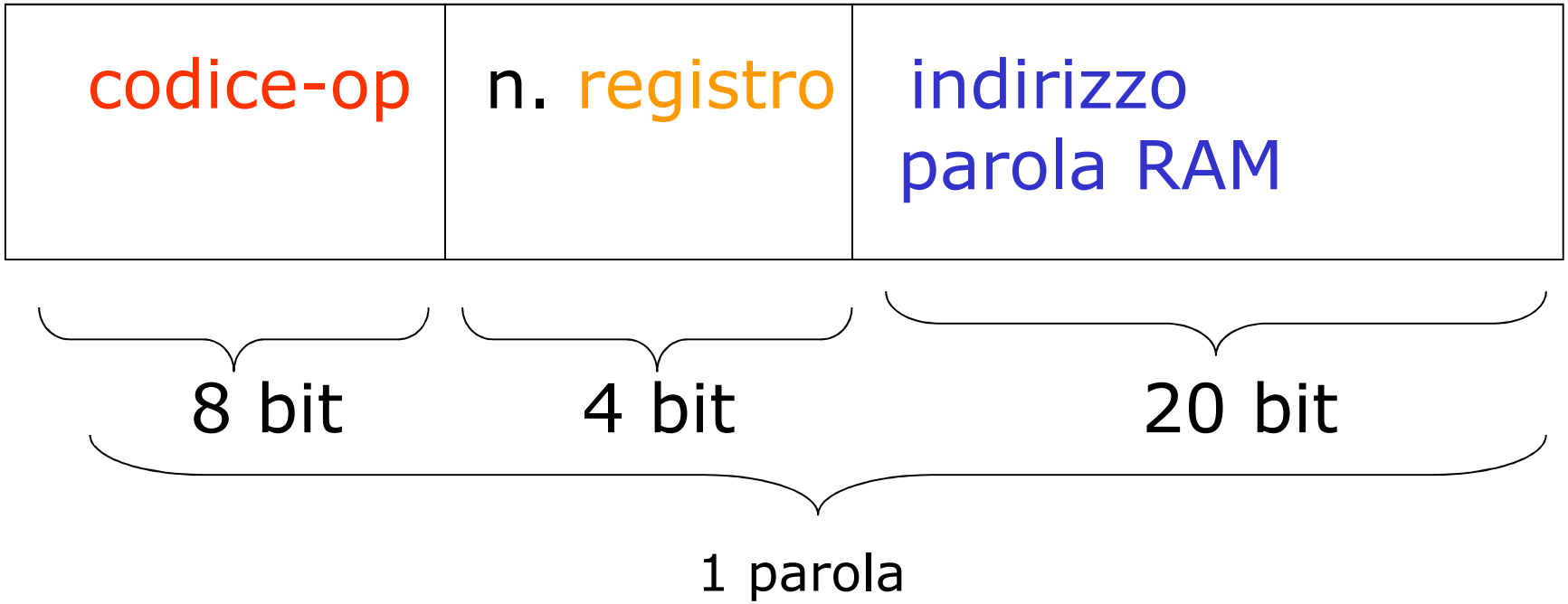
1. Trasferimento tra RAM e registri della CPU
2. Aritmetiche: somma, differenza, moltiplicazione, e divisione
3. Input/output
4. Confronto e salto e di stop

Istruzioni di trasferimento: registri ⇔ RAM



Formato:

in binario!

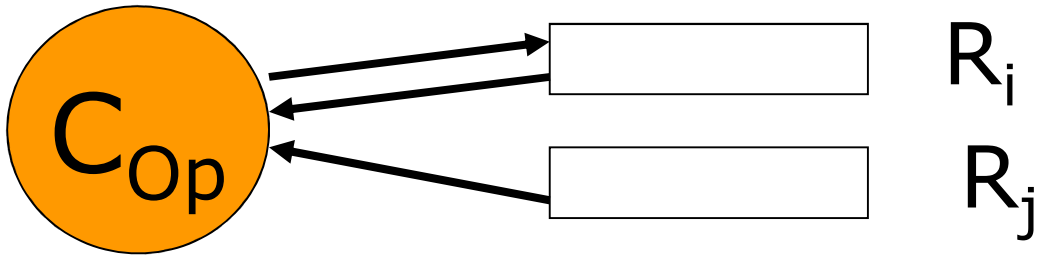


Codici:

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

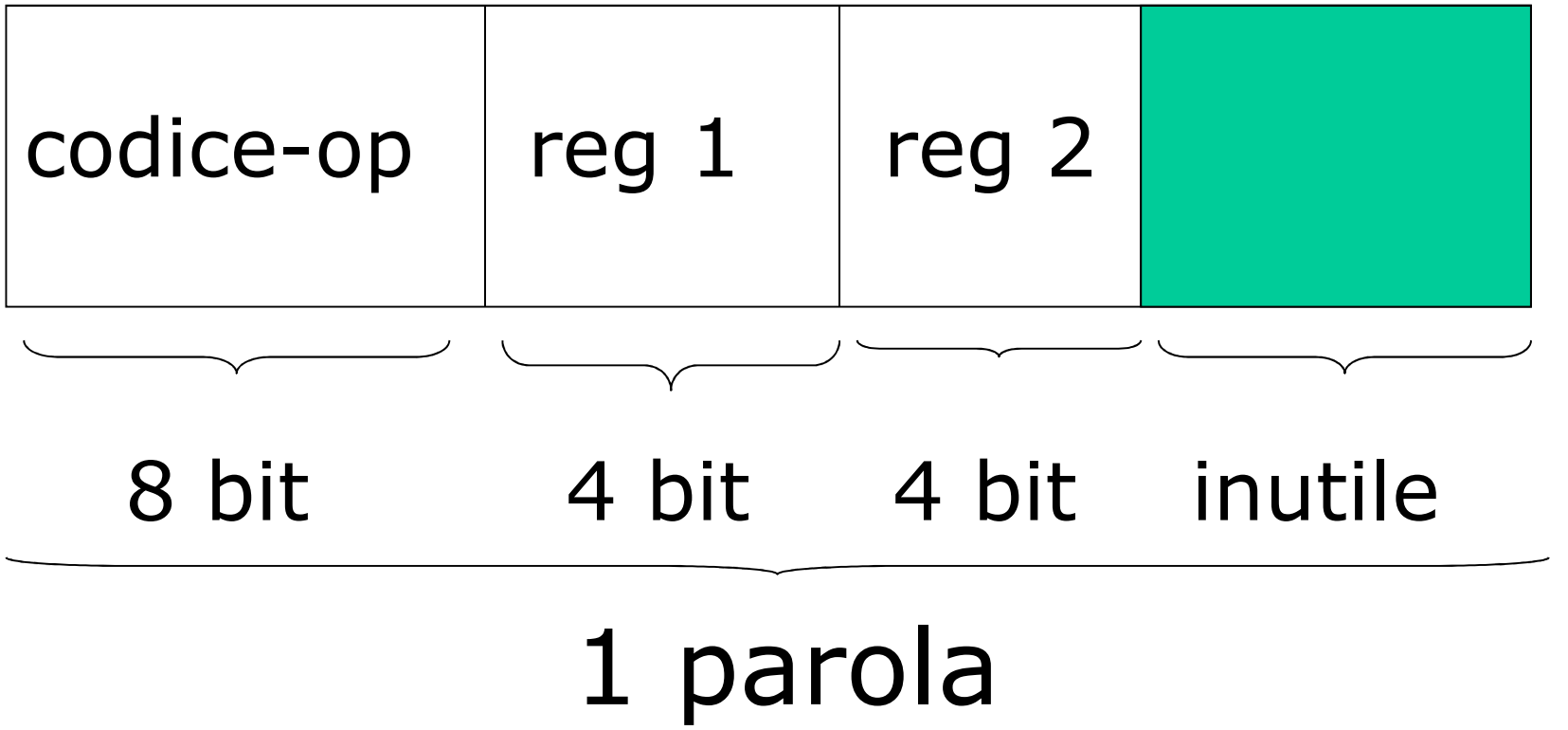
ARITMETICHE

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

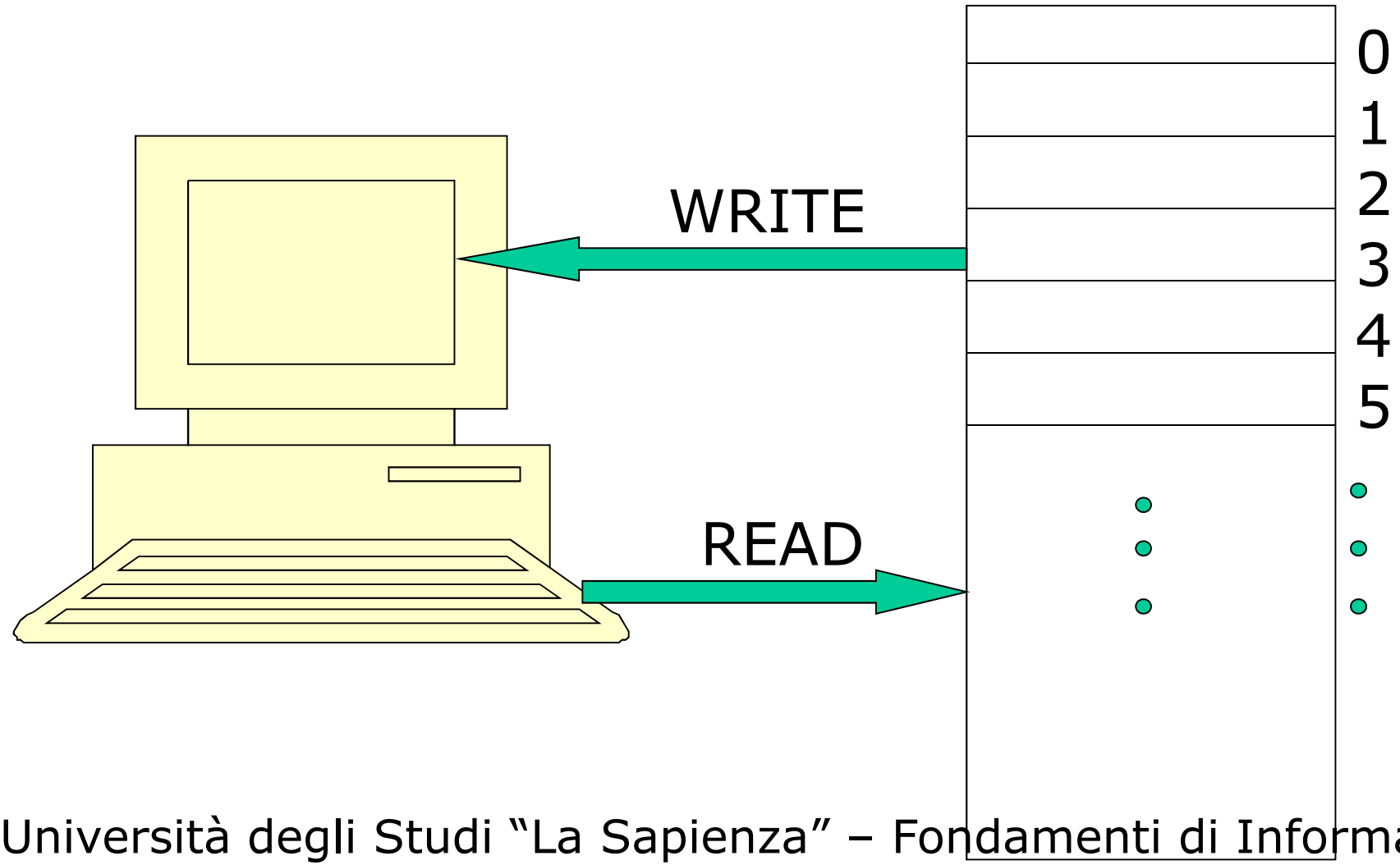


ADD	**00000010**	**FADD**	**00000011**
SUB	**00000100**	**FSUB**	**00000101**
MULT	**00000110**	**FMULT**	**00000111**
DIV	**00001000**	**FDIV**	**00001001**
		MOD	**00001010**

FORMATO:

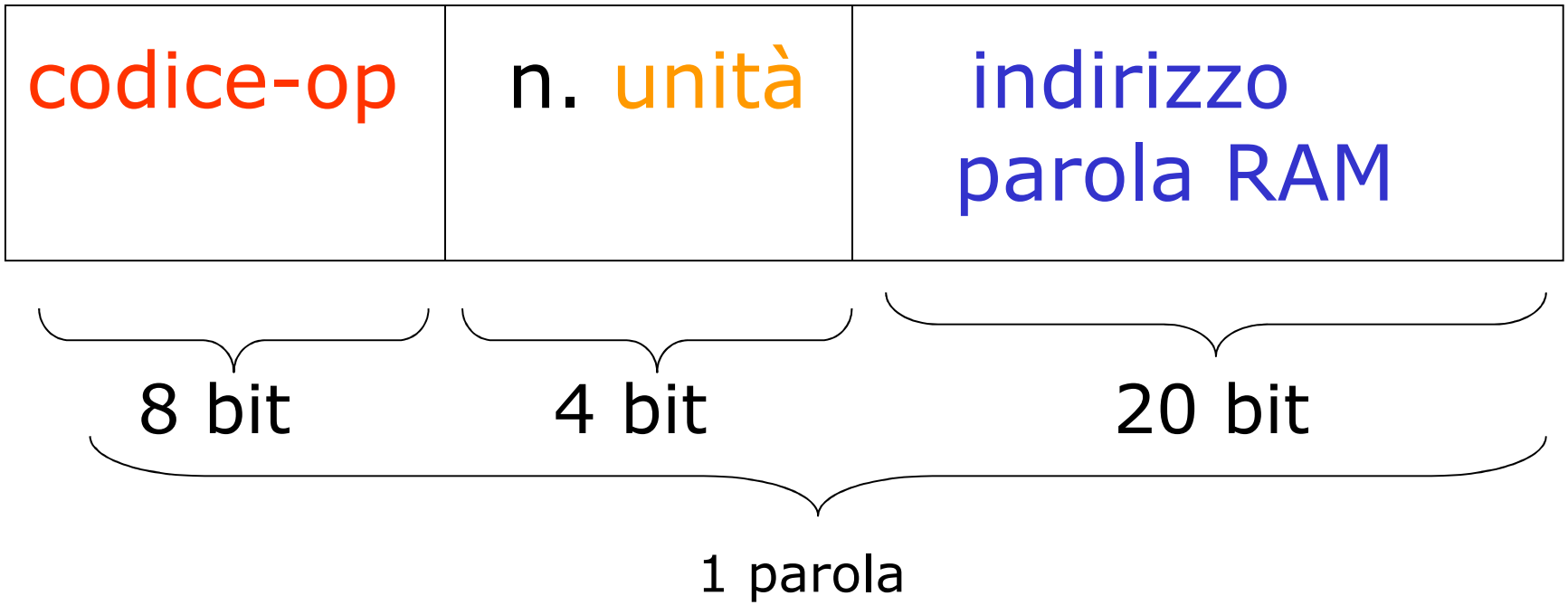


Istruzioni di input/output: unità I/O ⇔ RAM



Formato:

in binario!

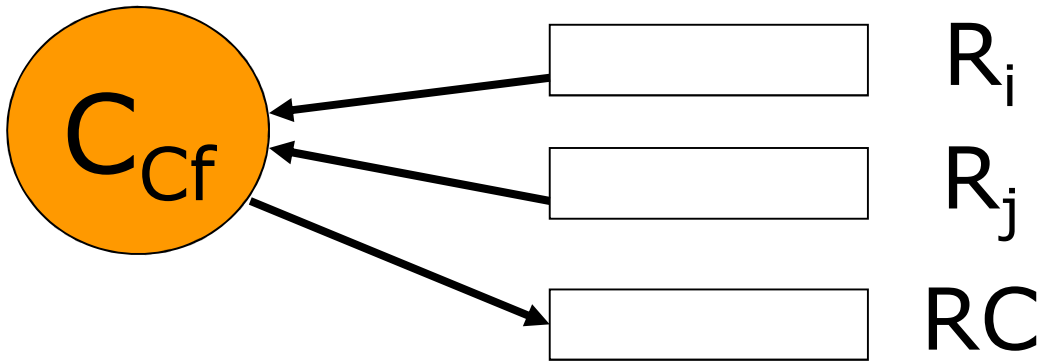


Codici:

READ	**00010000**	**STINP**	**0000 (tastiera)**
WRITE	**00010001**	**STOUT**	**0001 (video)**

Confronto

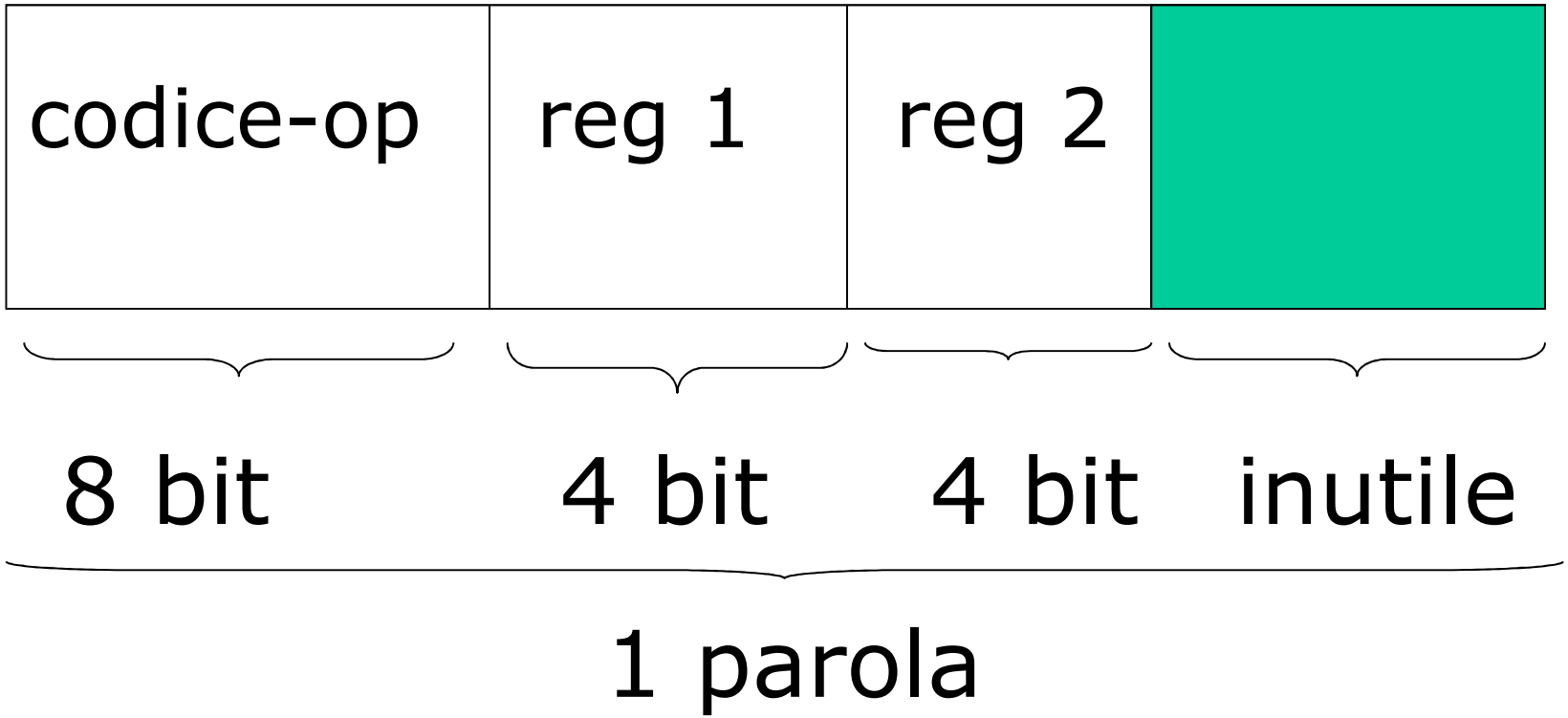
- confronta il contenuto dei registri $\mathbf{R_i}$ ed $\mathbf{R_j}$ e:
  - se $R_i < R_j$ mette -1 nel registro RC
  - se $R_i = R_j$ mette 0 in RC
  - se $R_i > R_j$ mette 1 in RC



Codici:

COMP	**00100000**
FCOMP	**00100001**

FORMATO:



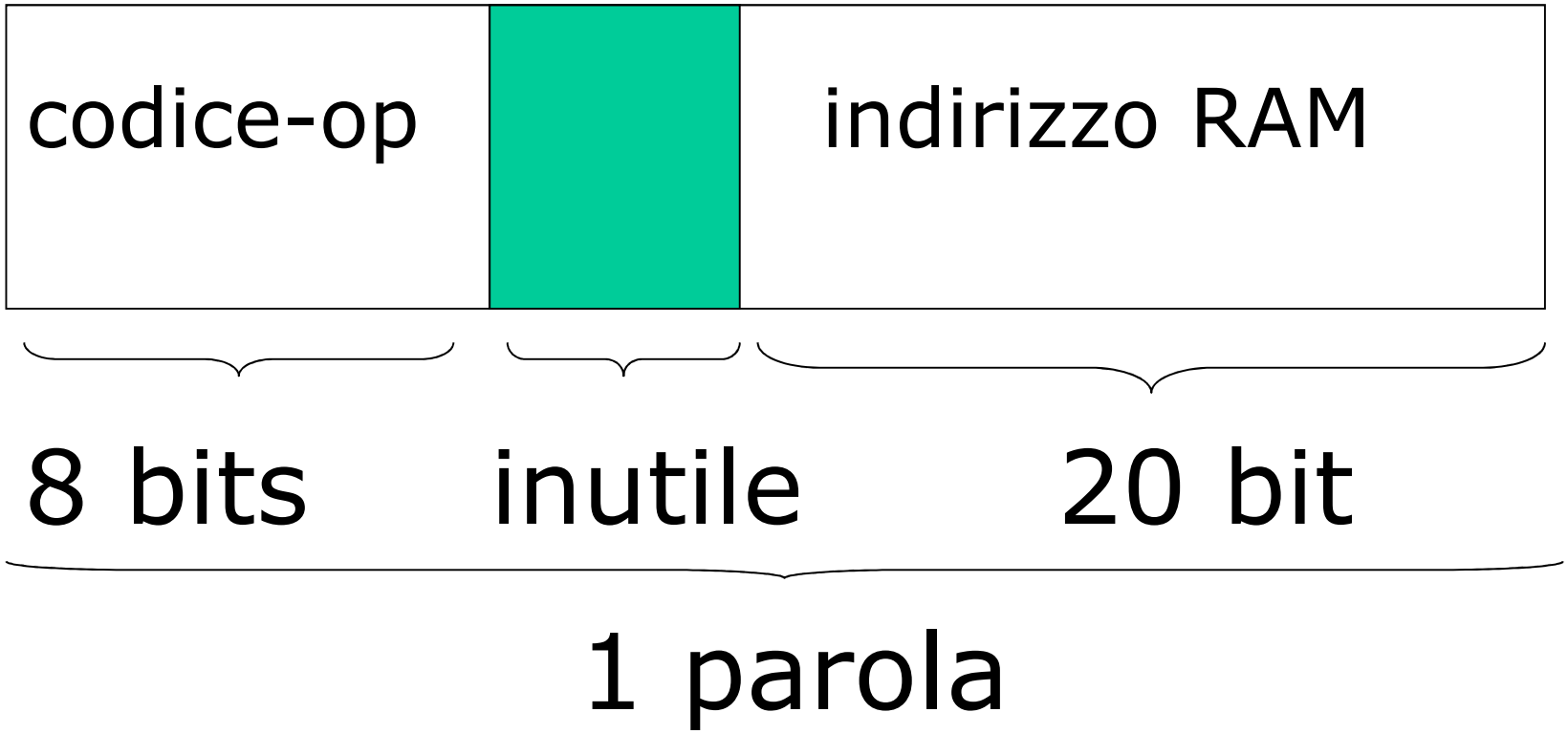
Salto

- 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

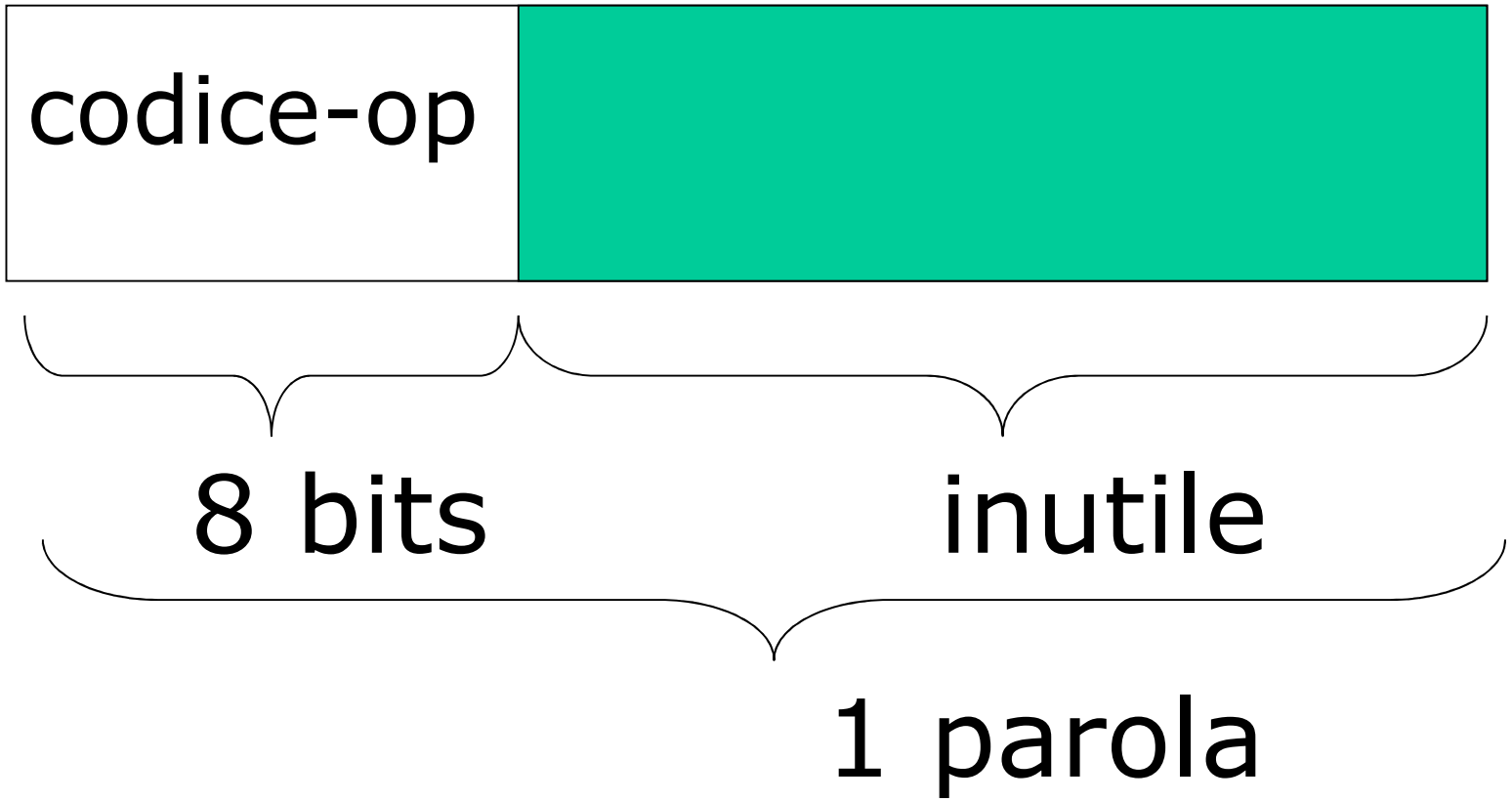
termina il programma

Codice

```
STOP    10000001
```

:

FORMATO:



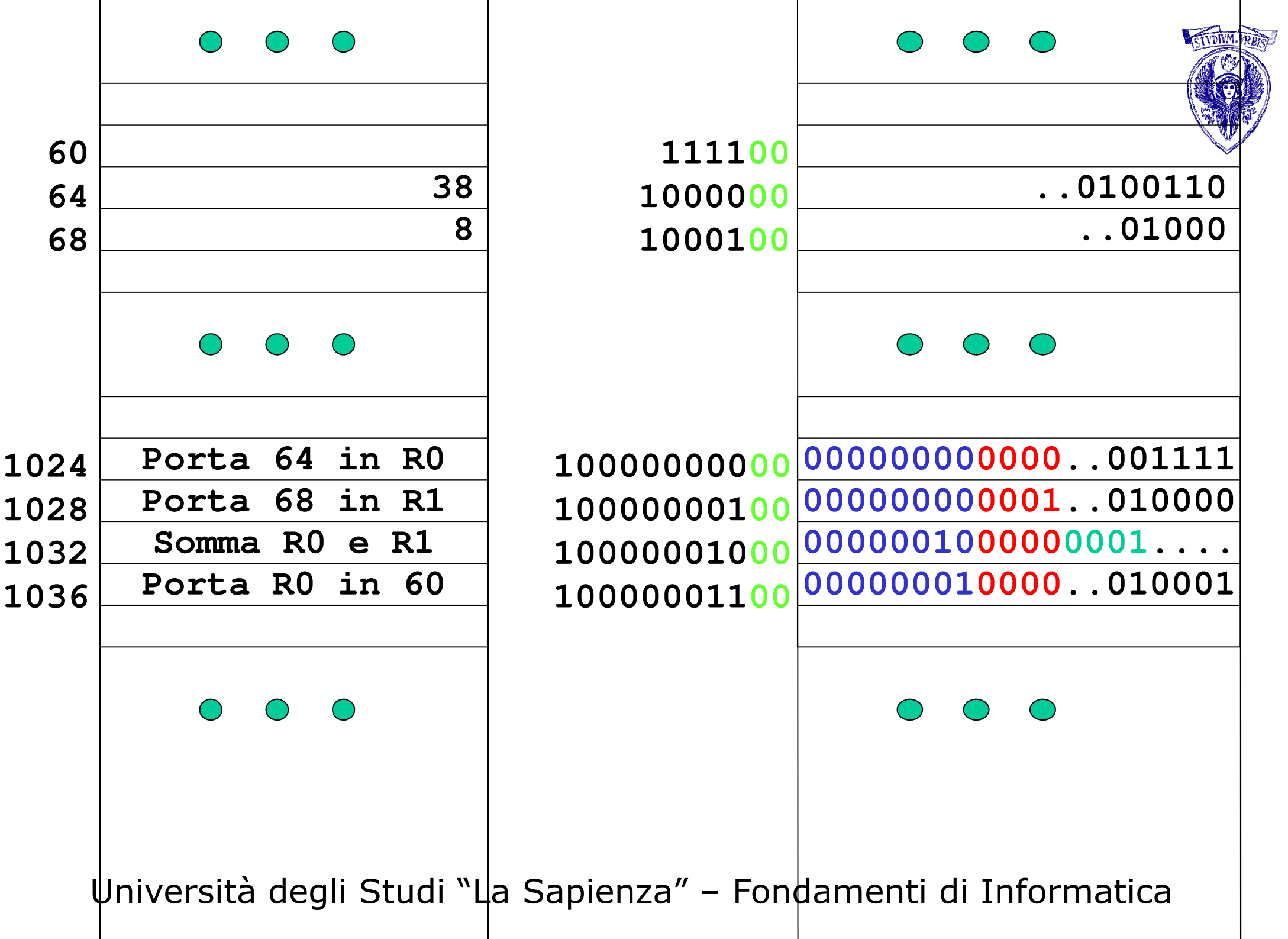
Esempio

scriviamo un programma macchina che:

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

codici delle operazioni

1. trasferimento RAM → CPU:
   00000000
2. trasferimento CPU → RAM:
   00000001
3. somma : **00000010**


60
64
68
38
8
1024
Porta 64 in R0
1028
Porta 68 in R1
1032
Somma R0 e R1
1036
Porta R0 in 60
1111 00
1000 00
1000 100
..0100110
..01000
1000000000
0000000000000..001111
1000000100
0000000000001..010000
1000001000
0000001000000001....
1000001100
0000000100000..010001

svantaggi del linguaggio macchina:

I programmi in binario sono difficili da scrivere, capire e cambiare

Il programmatore deve occuparsi di gestire la RAM

operazione difficile ed inefficiente

Soluzione → Assembler

Novità dell'Assembler

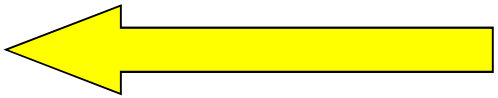
- codici mnemonici per le operazioni
- nomi mnemonici (identificatori) al posto degli indirizzi RAM per i dati (e indirizzi RAM delle istruzioni usate nei salti)
- avanzate: tipi dei dati `INT` e `FLOAT`

codice-op mnemonici:

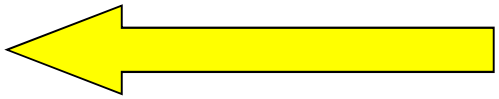
- trasferimento: **LOAD** (RAM → CPU) e **STORE** (CPU → RAM)
- aritmetiche: **ADD, SUB, DIV, MULT, MOD, FADD, FSUB, FDIV, FMULT**
- input/output: **READ** (U-INP → CPU), **WRITE** (CPU → U-OUT)
- test: **COMP, FCOMP**
- salto: **BREQ, BRGT, BRLT, BRGE, BRLE, BRANCH**
  - EQ = Equal
  - GT = Greater than
  - LT = Lesser than
  - GE = Greater equal
  - LE = Lesser Equal
- terminazione: **STOP**

Somma di due numeri in Assembler

```
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



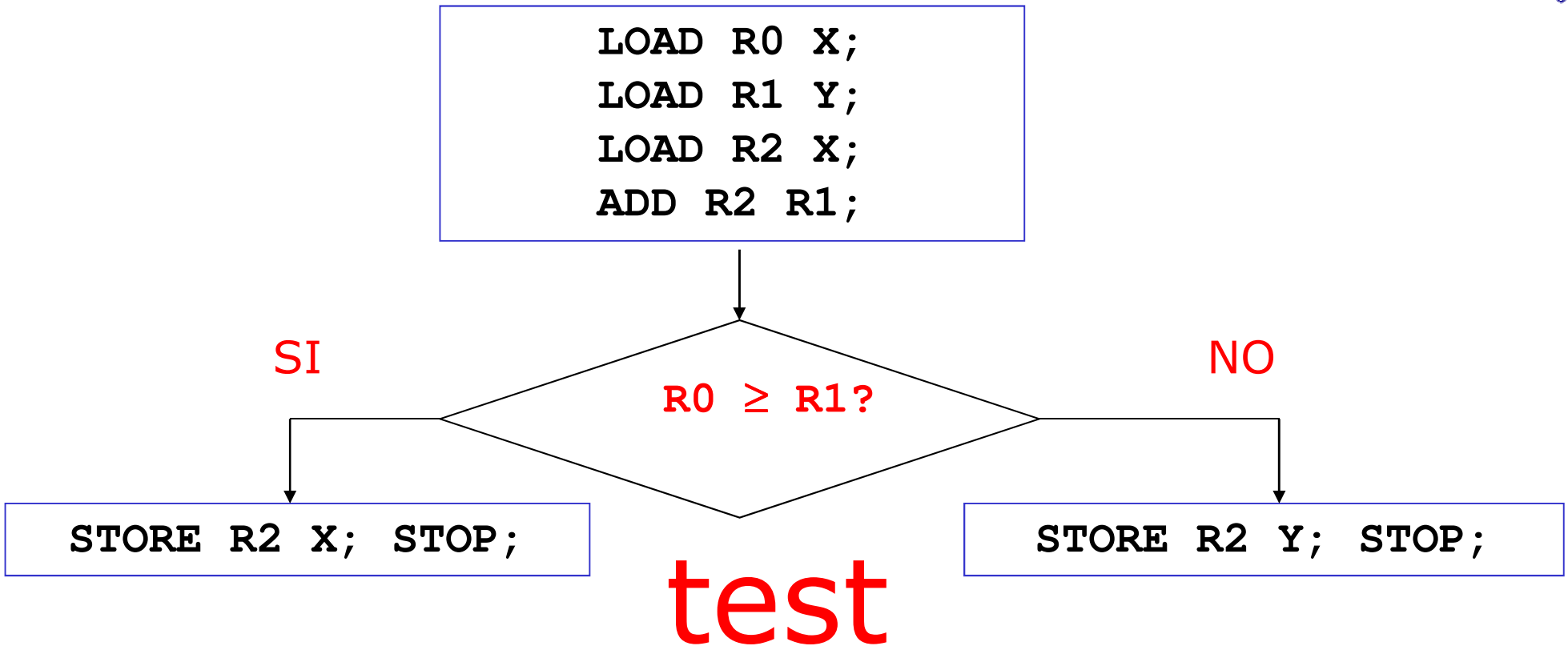
istruzioni assembler

esempio

- carica due valori dalla RAM
- li somma
- 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;
COMPARE 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;
STORE R2 Y; STOP;
test