Fondamenti di Informatica

Ingegneria Clinica

Lezione 02/12/2010

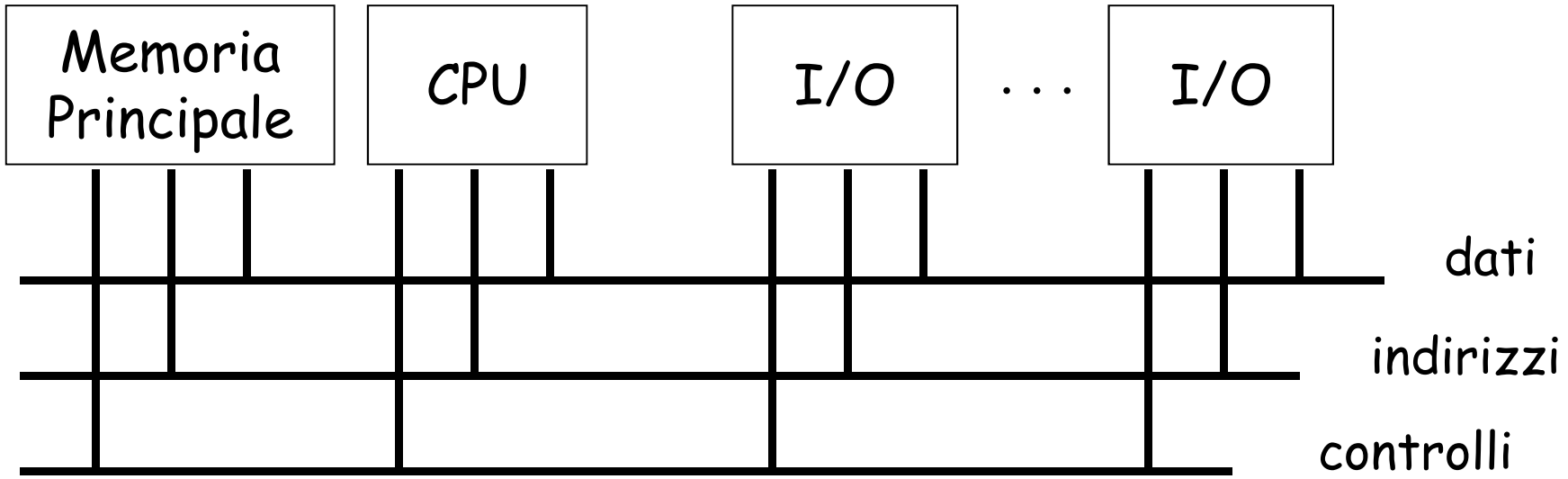
Prof. Raffaele Nicolussi

FUB - Fondazione Ugo Bordoni

Via del Policlinico, 147 - 00161 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**	**Lunedì**	**14:00 – 17:30**
Ricevimento:	**Per appuntamento**	**in FUB, per email, per telefono**
Sito web:	**http://w3.uniroma1.it/IngClinFondinf**	

Interazioni tra le unità



- Le unità sono connesse attraverso linee di comunicazione specifiche (bus) per dati (data bus), controlli (control bus) e indirizzi (address bus)
- Le interazioni tra le unità possono avvenire secondo varie modalità
- Modalità elementare: ogni trasferimento di dati è attivato e controllato esclusivamente da programma

Sistemi Operativi

Un sistema operativo (operating system) è una collezione di programmi per la gestione delle risorse presenti in un calcolatore

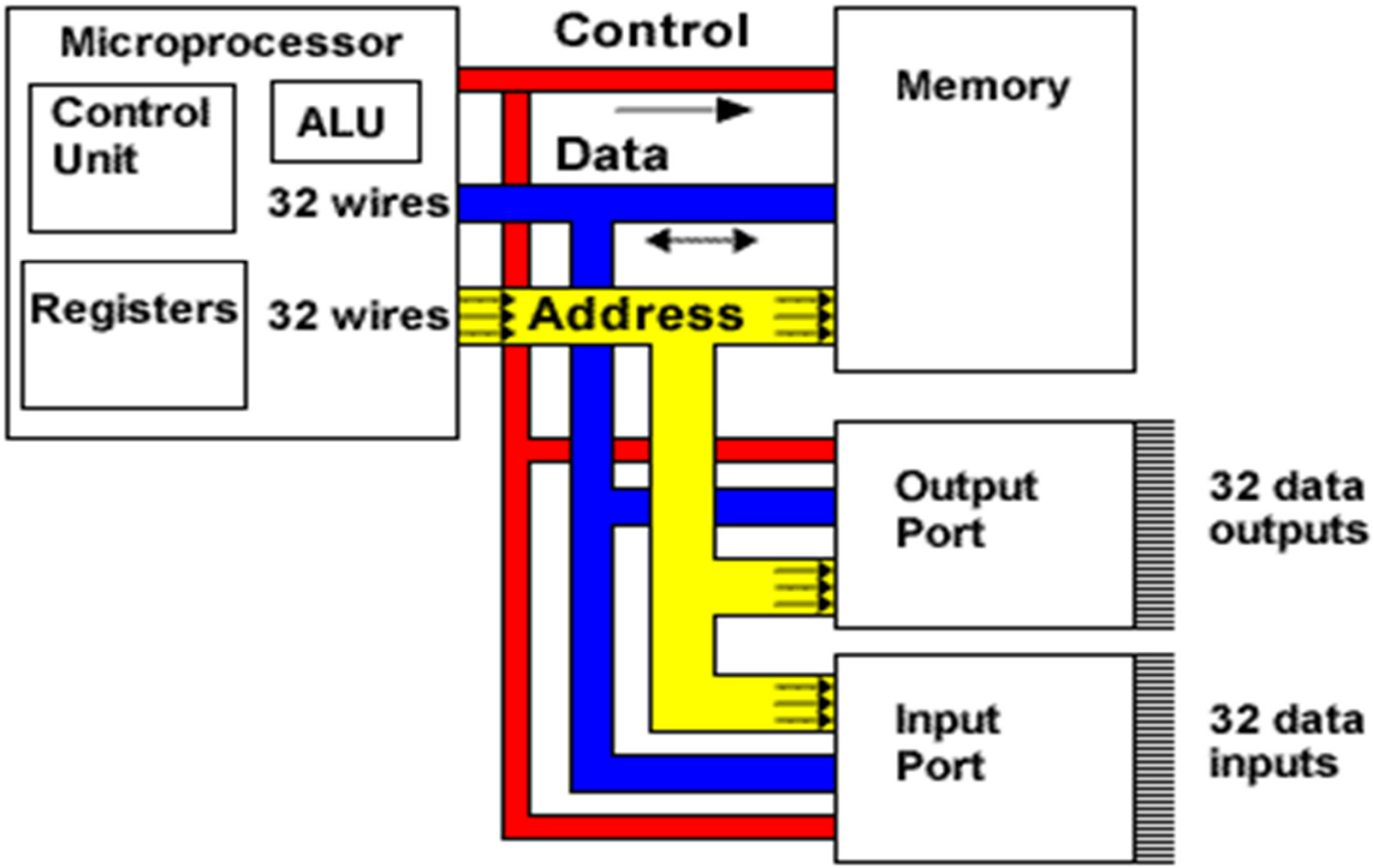
Due gli obiettivi fondamentali:

- gestire in modo efficiente l'elaboratore e le sue periferiche
- creare un ambiente virtuale per facilitare l'interazione uomo macchina

Il sistema operativo mette a disposizione degli altri programmi una serie di funzionalità, semplificando così l'uso del sistema

- Il codice macchina eseguito dalla CPU si presenta come una successione di porzioni del sistema operativo e porzioni degli altri programmi
- Per esempio, grazie al sistema operativo, la memoria secondaria appare come una collezione di file opportunamente strutturata, gestibile con semplici comandi
- Il sistema operativo fornisce un'interfaccia verso l'utente (linguaggio a comandi o interfaccia grafica per l'interazione col sistema)

Struttura del bus



Programmazione dei calcolatori

- L'uso del linguaggio macchina **è normalmente evitato**
- Nella programmazione *reale* si fa uso di linguaggi assemblativi e di linguaggi di alto livello
- Questi due approcci si distinguono essenzialmente per **il livello di astrazione in cui i linguaggi si collocano rispetto alla macchina fisica**
- Un tipico flusso di progetto di un programma inizia con la sua scrittura e finisce con la sua verifica
- In tutte le fasi del flusso di progetto, il programmatore può tipicamente contare su un ambiente/sistema di sviluppo (development environment) costituito da una opportuna collezione di programmi (e da una macchina fisica su cui eseguirli)

Algoritmi

- Definire un problema significa specificare i dati su cui si deve operare ed il risultato che si deve produrre
- Con il termine algoritmo (metodo risolutivo di un problema) si indica una sequenza ordinata di operazioni, la cui esecuzione nell'ordine specificato produce la soluzione di un problema definito
- La soluzione desiderata deve essere allora fornita in un numero finito di passi per qualsiasi configurazione ammissibile dei dati di input
- L'efficienza di un algoritmo è normalmente misurata dalle risorse di tempo e memoria necessarie per la sua esecuzione

Linguaggi Assemblativi (1)

- Un linguaggio assemblativo (assembly language), o *assembler*, è un linguaggio di programmazione per **una specifica CPU**

- Le sue istruzioni corrispondono *direttamente* ad istruzioni macchina della CPU e sono rappresentate da codici *mnemonici* che ne semplificano l'interpretazione

- Esempio
  - l'istruzione
    - "000000 00101 01111 10000 00000 100000"
    - istruzione aritmetica somma i contenuti dei registri 5 e 15 e memorizza il risultato nel registro 16
  - diventa
    - add $s0, $a1, $t7

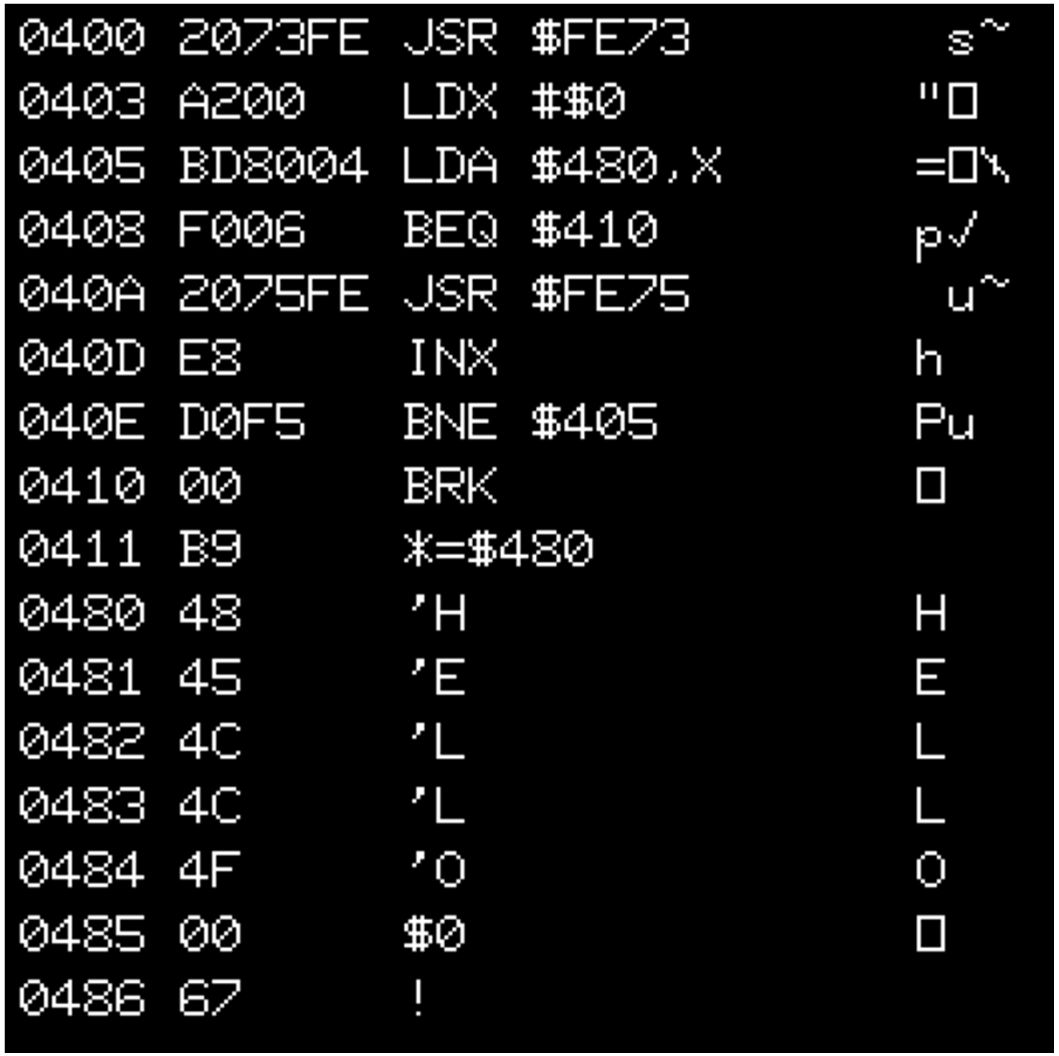
Linguaggi Assemblativi (2)

- Un assemblatore (assembler) è un programma capace di tradurre un programma in assembler (codice sorgente (source code)) *in un* corrispondente programma in linguaggio macchina (codice oggetto (object code))

- L' assemblatore, quindi, ha funzionalità simili al compilatore

- Per poter essere *eseguito* su una determinata macchina, il codice oggetto deve essere trasformato in codice eseguibile (executable code)

Esempio di assembler



Esempio CPU didattica : somma di due interi

- Assumiamo che gli interi occupino 2 byte e si trovino in memoria centrale, agli indirizzi **1000** e **1002**. Il risultato deve essere memorizzato in **1004**
  - Usiamo la "CPU didattica" composta da due registri (R0 ed R1) ed in grado di eseguire la somma usando R0 come accumulatore
  - I registri, come già accennato, sono zone di memoria "particolari" in cui memorizzare dei dati. Sono una sorta di "variabili" della CPU
- **Codice Assembler**

```
mov R0,[1000]
mov R1,[1002]
add R0, R1, R0
mov [1004],R0
```

Linguaggi di alto livello

- Un linguaggio di alto livello (high level language) permette l'uso di **istruzioni molto potenti** (il linguaggio C, presentato nel corso, è di questo tipo)

- E' progettato per essere facilmente utilizzabile dal programmatore, anche per la descrizione di algoritmi molto complessi

- A differenza di un linguaggio assemblativo, un linguaggio di alto livello è notevolmente *lontano* dalla macchina fisica su cui lo si vuole eseguire (programmazione ad alto livello di astrazione)

- (*Un approccio possibile per*) l'esecuzione su una specifica macchina di un programma in un linguaggio di alto livello richiede la *traduzione* di questo *in un* corrispondente programma nel linguaggio macchina specifico (caso tipico per il C)

Compilatori

- Un compilatore (compiler) è un programma capace di tradurre un programma espresso in uno specifico linguaggio di alto livello (codice sorgente) in un programma espresso in uno specifico linguaggio macchina (codice oggetto)

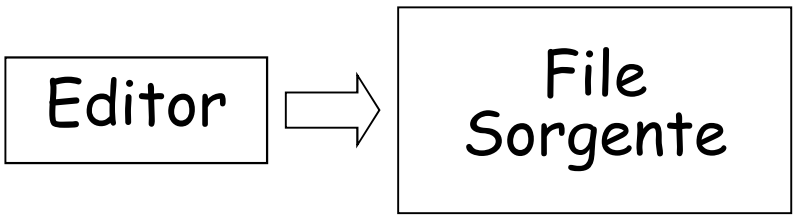
- Tipicamente, data la potenza dei linguaggi di alto livello, una singola istruzione di un linguaggio di alto livello corrisponde ad una sequenza di molte istruzioni macchina

- La disponibilità di compilare uno stesso linguaggio di alto livello per diversi linguaggi macchina consente al programmatore di scrivere programmi *portabili*

- Per poter essere *eseguito* su una determinata macchina, il codice oggetto deve essere trasformato in codice eseguibile

Progetto: Scrittura

- Per la generazione di codice sorgente viene tipicamente utilizzato un programma detto editore (editor)
- Le funzionalità essenziali di un editor sono quelle richieste nella scrittura di un *semplice* testo (un ambiente di sviluppo contiene tipicamente un editor con funzionalità extra che semplificano la preparazione del testo nel linguaggio specifico)
- Il testo prodotto viene registrato su memoria secondaria (disco) (generazione del file sorgente (source file))
- Il file sorgente è usato come input nella fase successiva



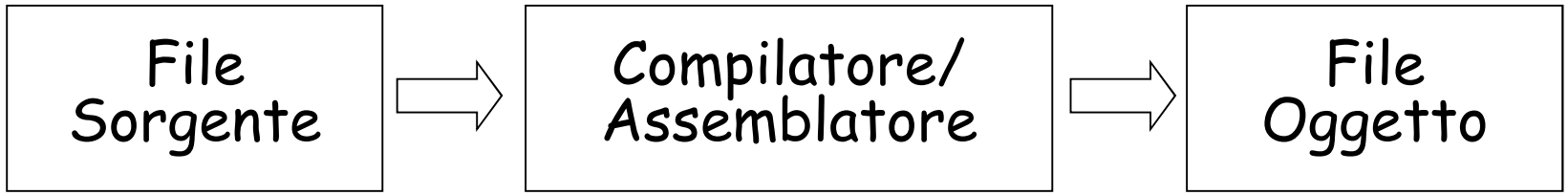
Progetto: Compilazione

- Il codice contenuto nel file sorgente viene trasformato con un compilatore in codice oggetto (viene generato un file oggetto (object file))

- Eventuali errori sintattici vengono evidenziati (violazioni delle regole di scrittura del codice sorgente)

- Il processo di compilazione prevede l'interpretazione di istruzioni *speciali* presenti nel codice sorgente dette direttive di compilazione (in linguaggio C, questo lavoro è a carico di un programma detto preprocessore (preprocessor), invocato direttamente dal compilatore)

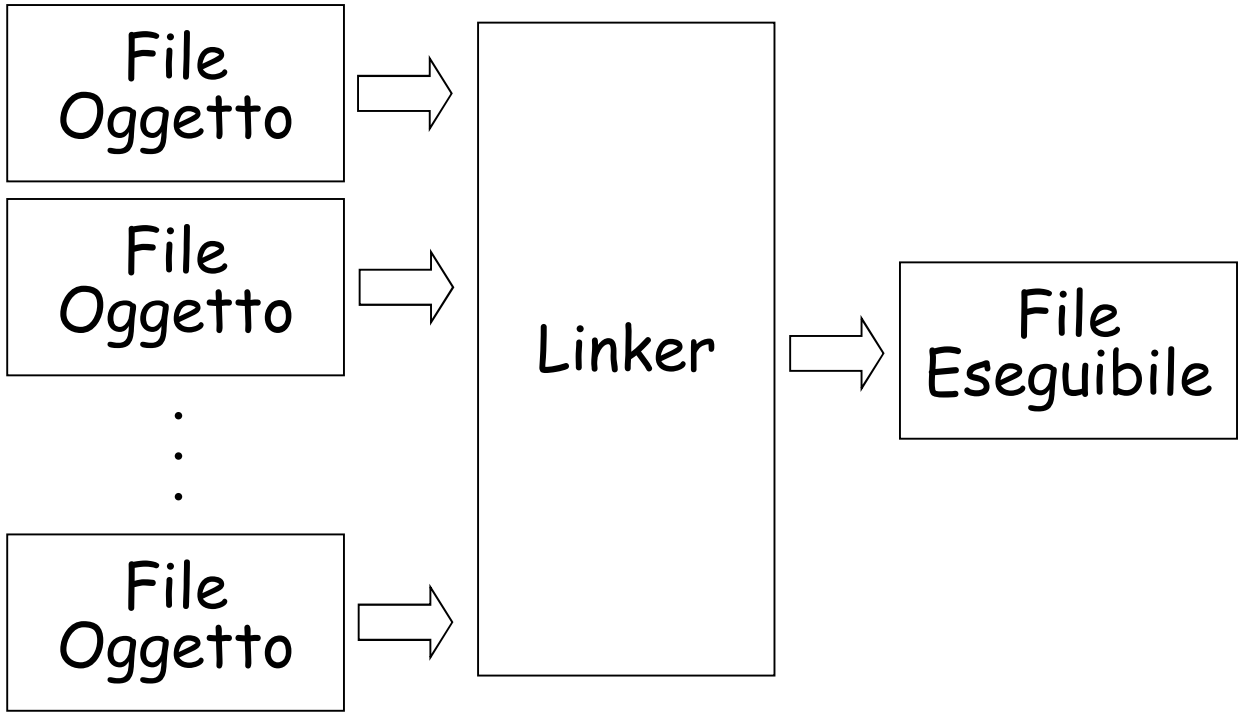
- Il file oggetto è usato come input nella fase successiva



Progetto: Collegamento

- Il programma finale in linguaggio macchina si compone tipicamente di varie porzioni di codice macchina presenti in più file oggetto

- La generazione del codice eseguibile avviene *collegando* tutti i file oggetto necessari tramite un programma detto collegatore (linker) (viene generato un file eseguibile (excutable file))

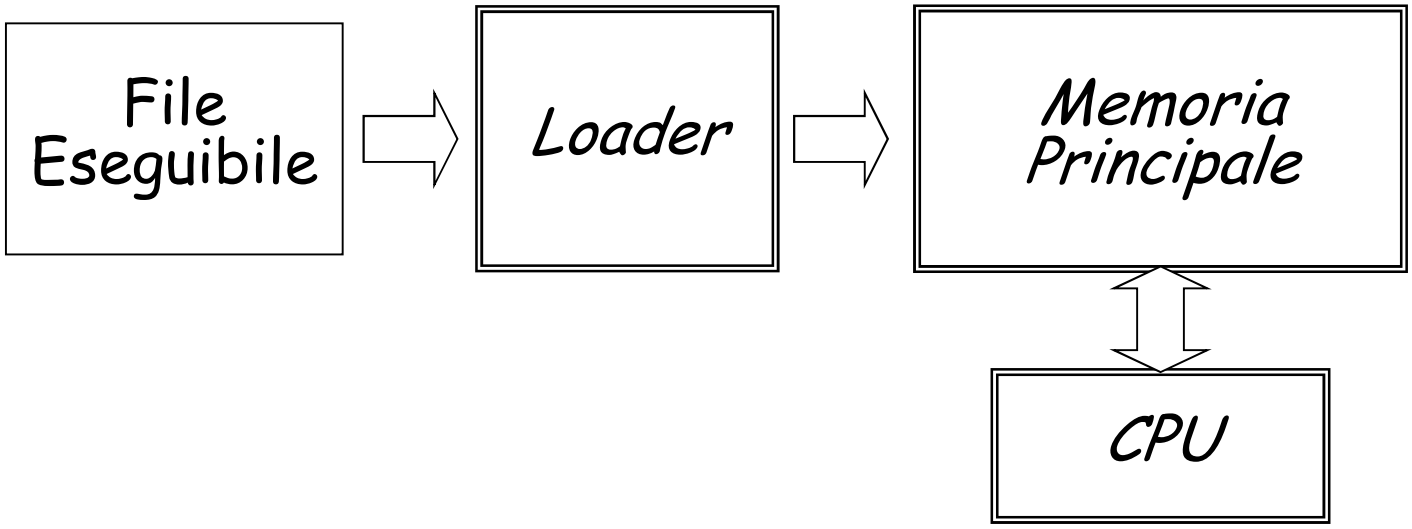


Esecuzione del programma

- L'esecuzione del programma richiede il *caricamento* nella memoria principale del codice macchina contenuto nel file eseguibile e l'*attivazione* della CPU al prelievo della prima istruzione da eseguire

- Questi compiti sono svolti da un programma detto caricatore (loader)

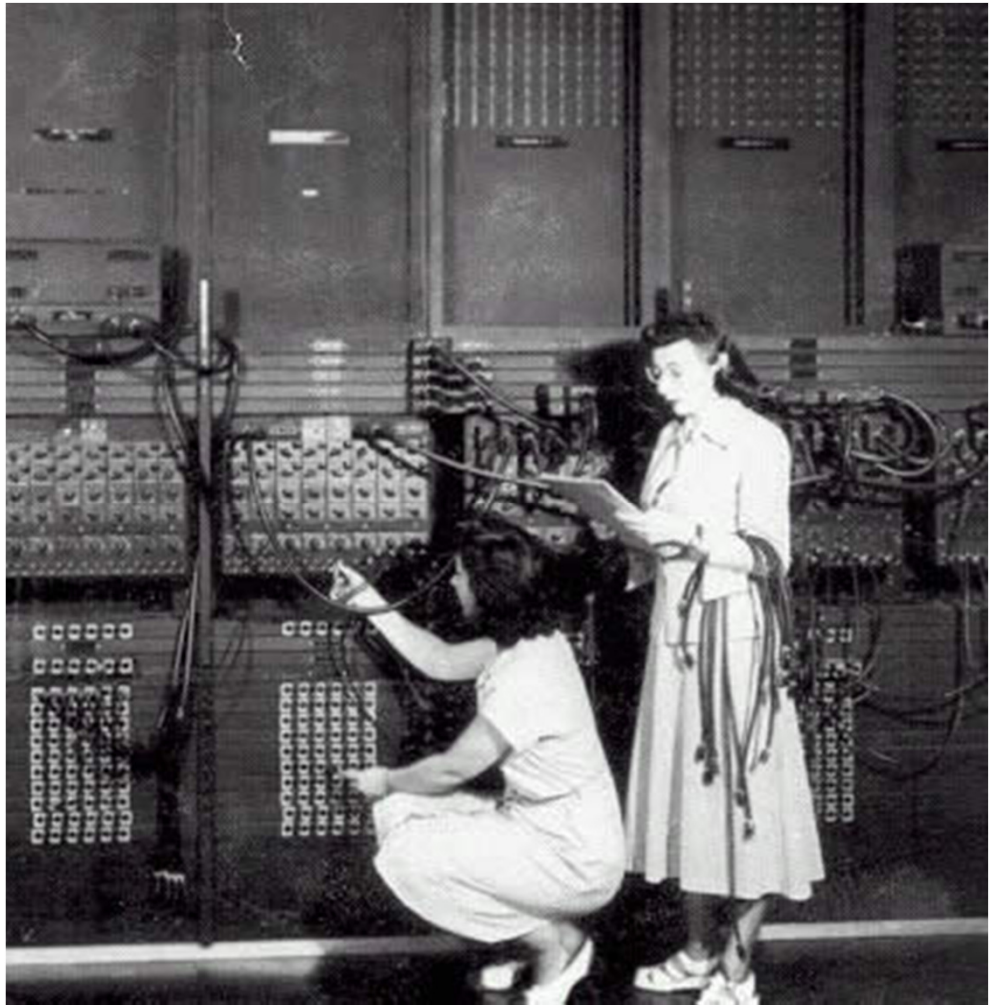
- Il loader è una delle funzionalità offerte dal sistema operativo



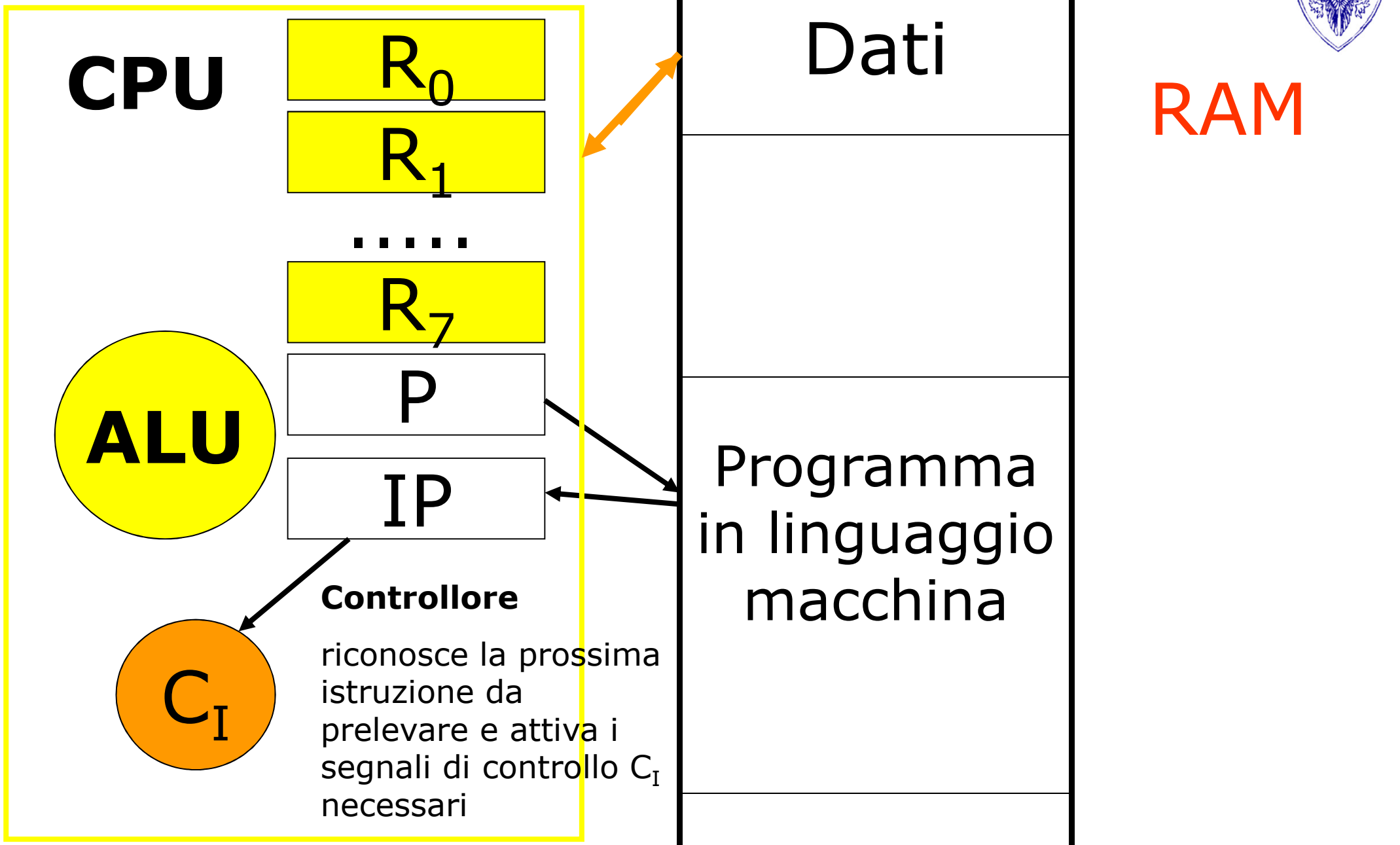
CPU didattica

- 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

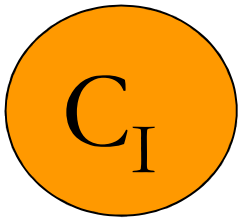


Useremo il linguaggio macchina di una CPU "MINIMA" ed il corrispondente linguaggio Assembler "MINIMO".


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

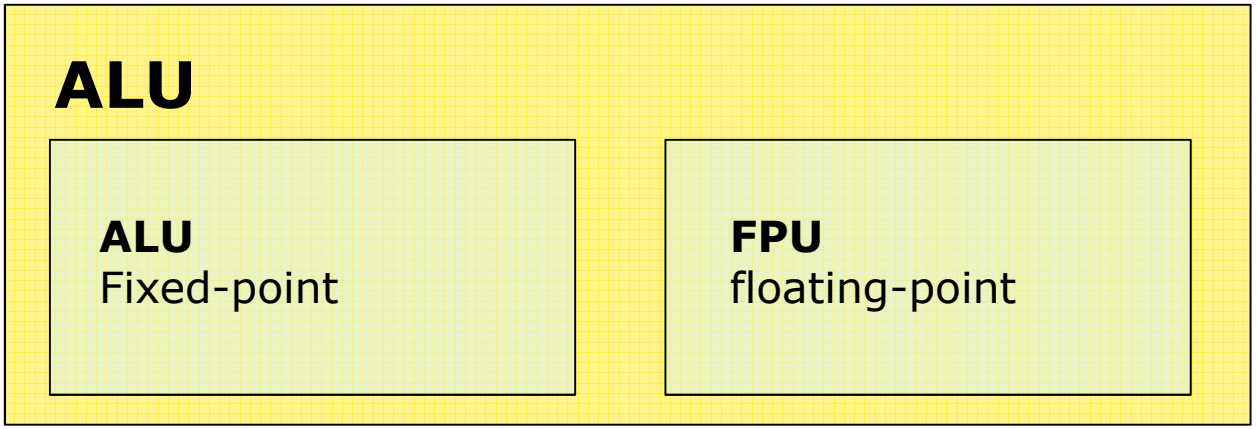
Controllore o unità di controllo

- 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

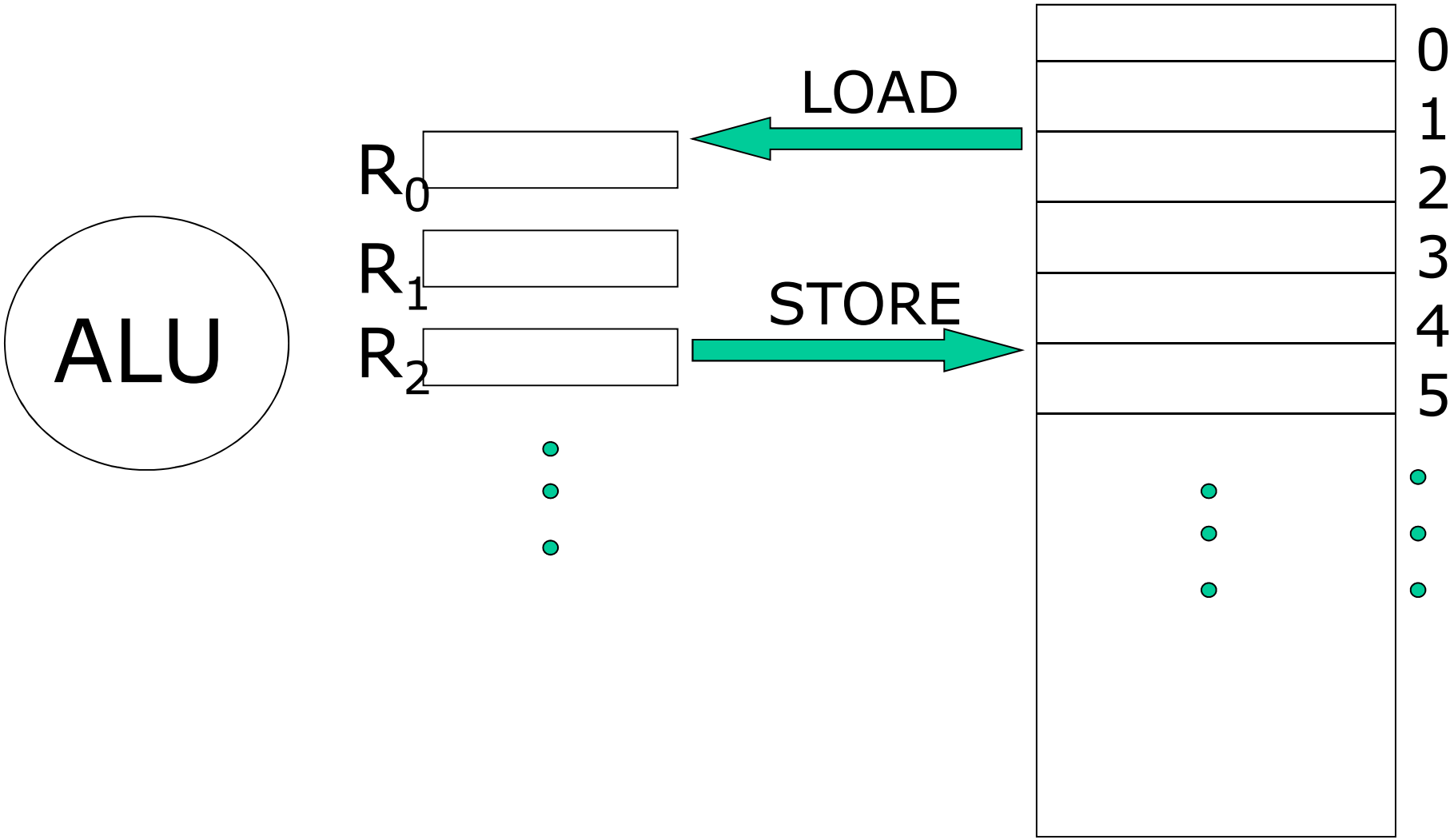
Coprocessore matematico

- 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:

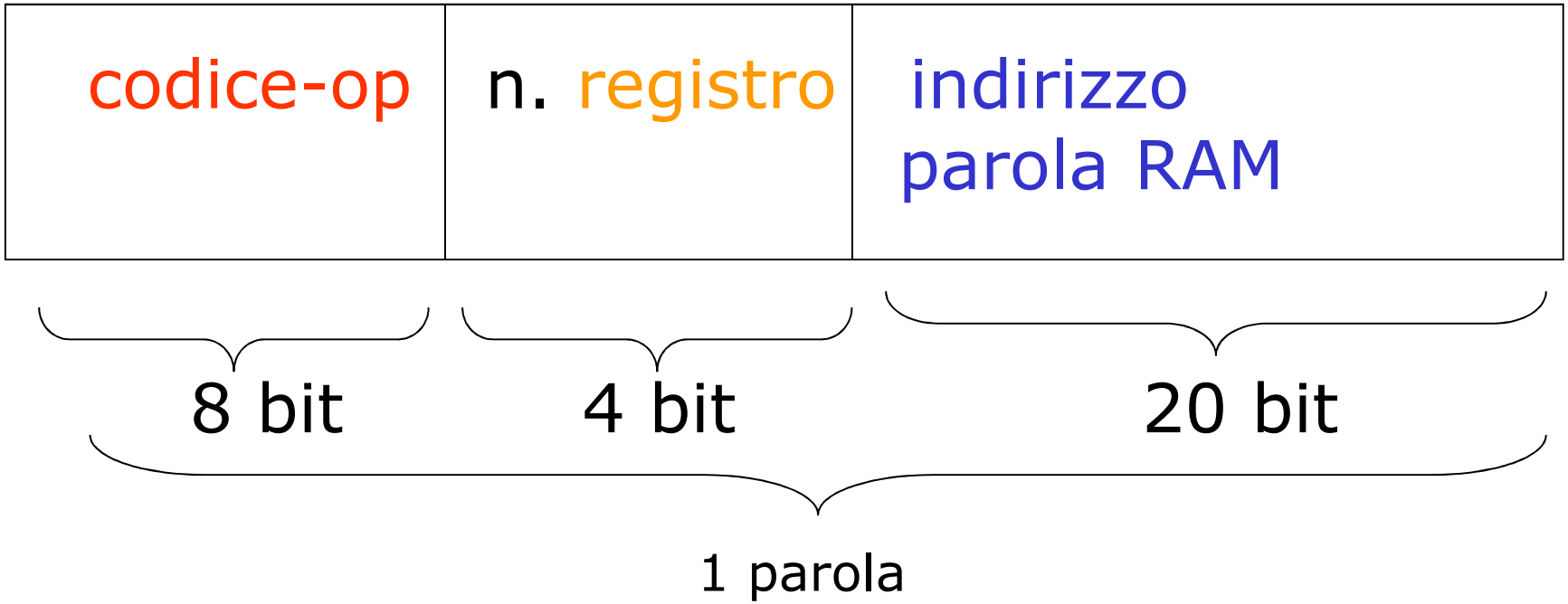
1. Trasferimento tra RAM e registri della CPU
2. Aritmetiche: somma, differenza, moltiplicazione, e divisione
3. Input/output
4. Confronto e salto e di stop

Istruzioni di trasferimento: registri ⇔ RAM



Formato:

in binario!

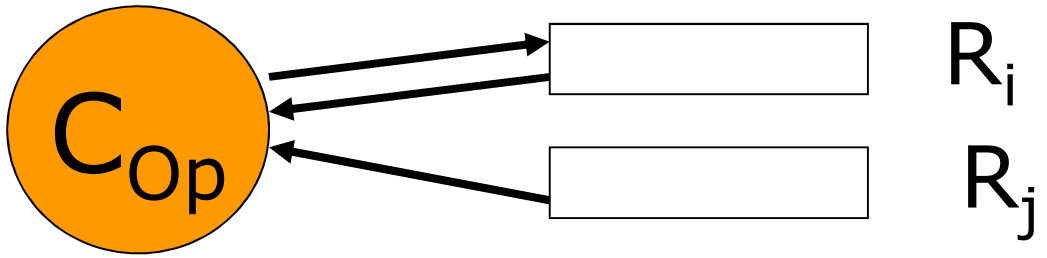


Codici:

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

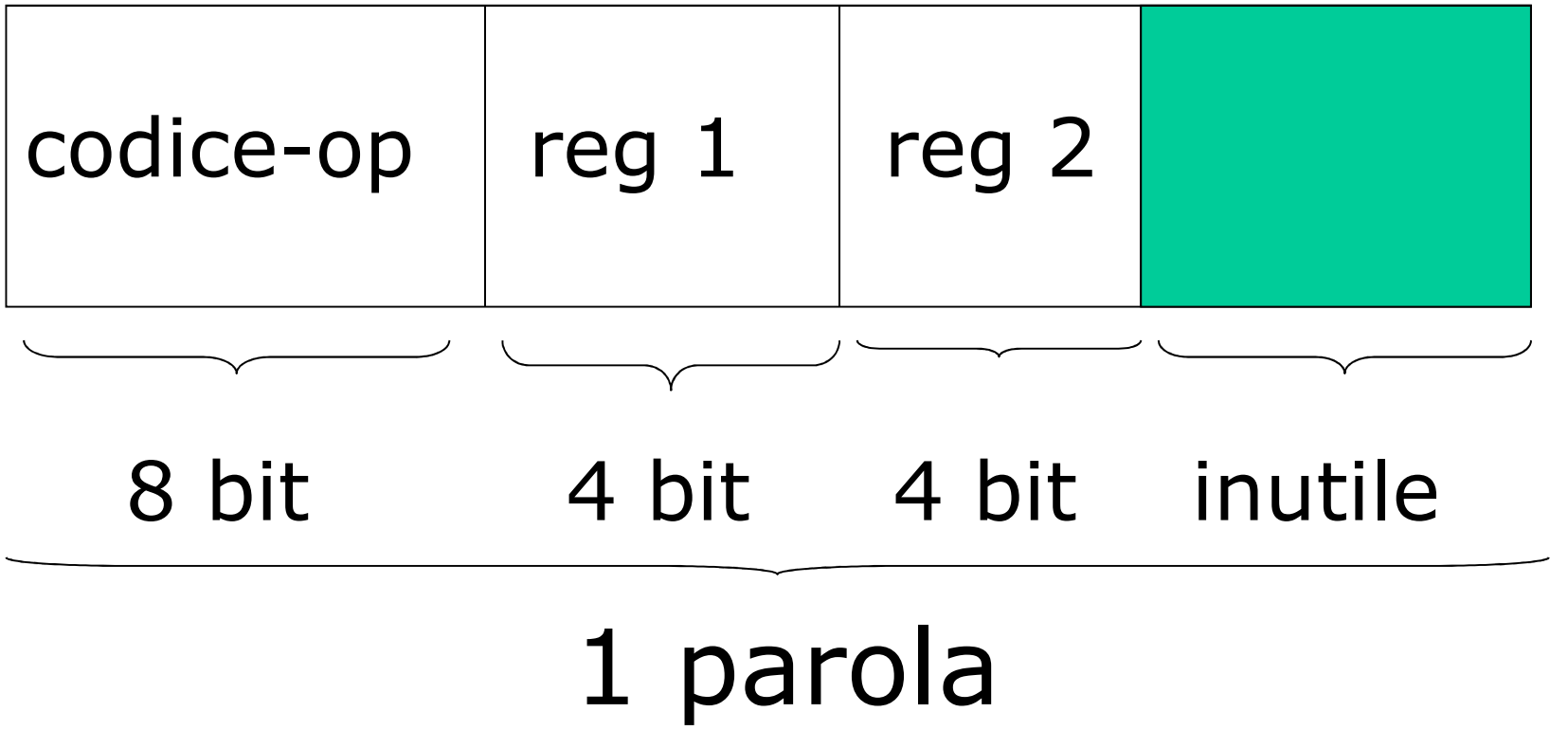
ARITMETICHE

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

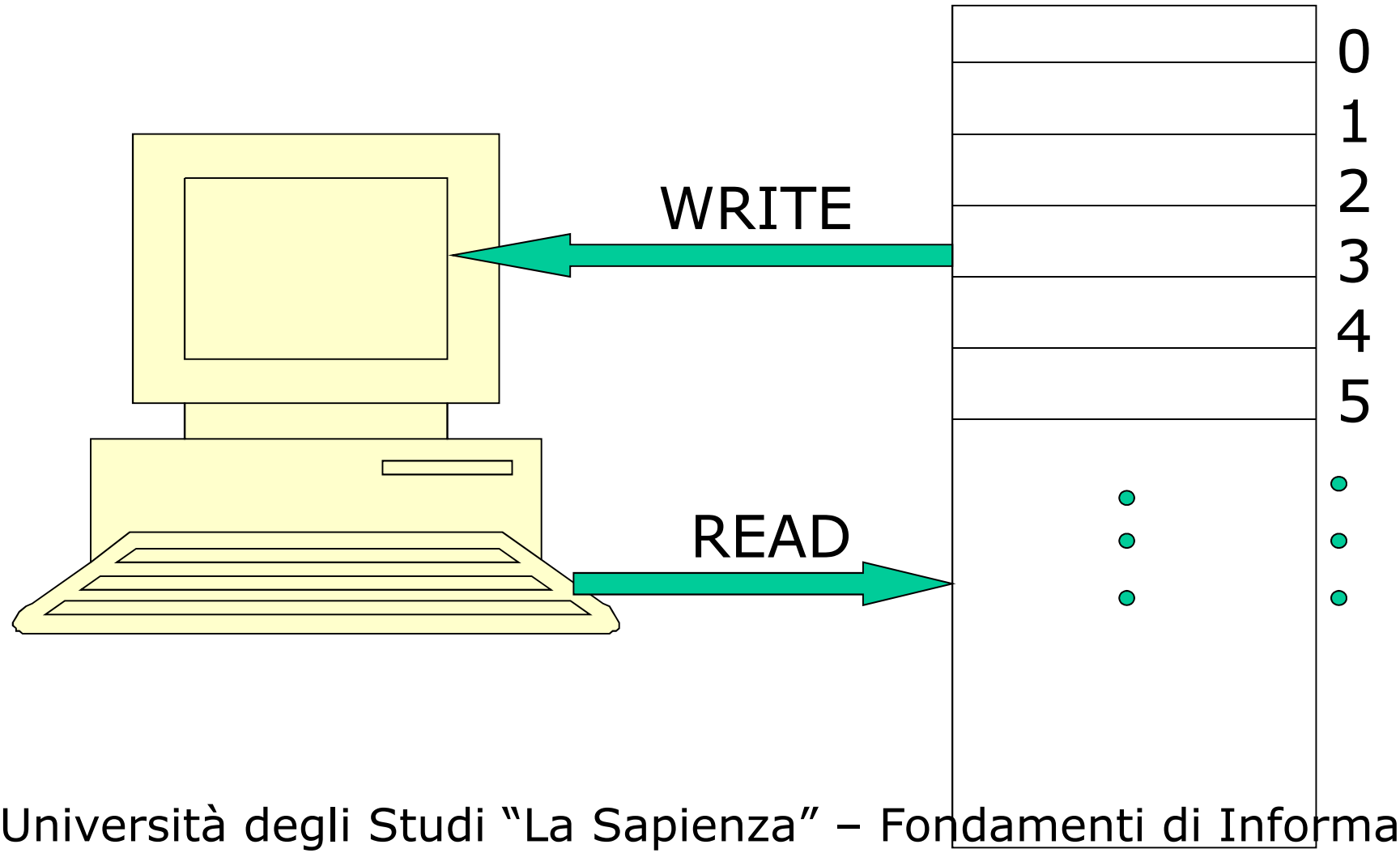


ADD	**00000010**	**FADD**	**00000011**
SUB	**00000100**	**FSUB**	**00000101**
MULT	**00000110**	**FMULT**	**00000111**
DIV	**00001000**	**FDIV**	**00001001**
		MOD	**00001010**

FORMATO:

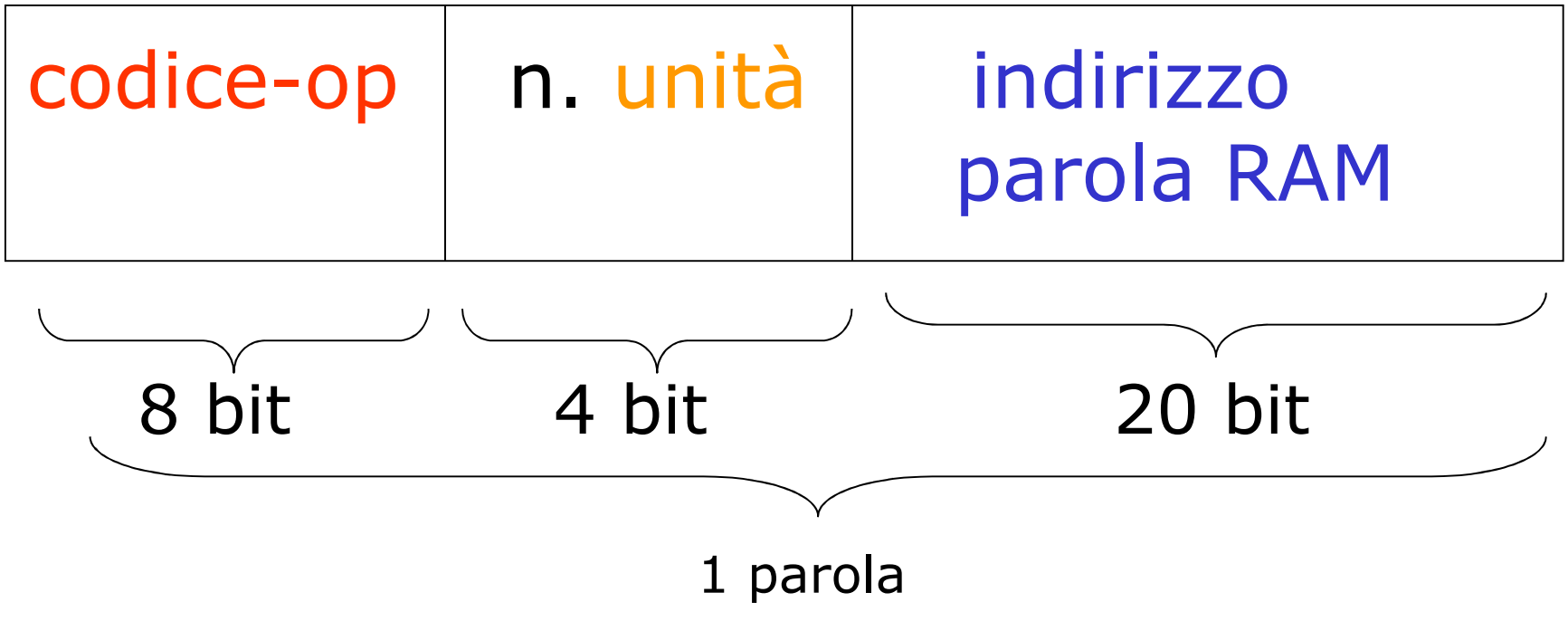


Istruzioni di input/output: unità I/O ⇔ RAM



Formato:

in binario!

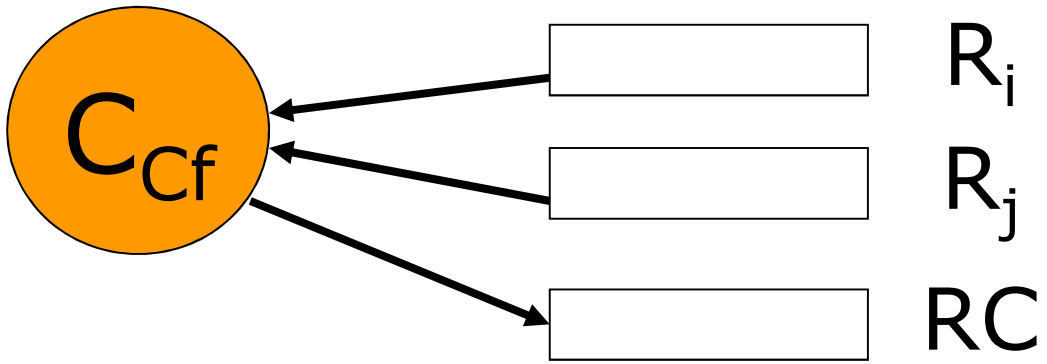


Codici:

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

Confronto

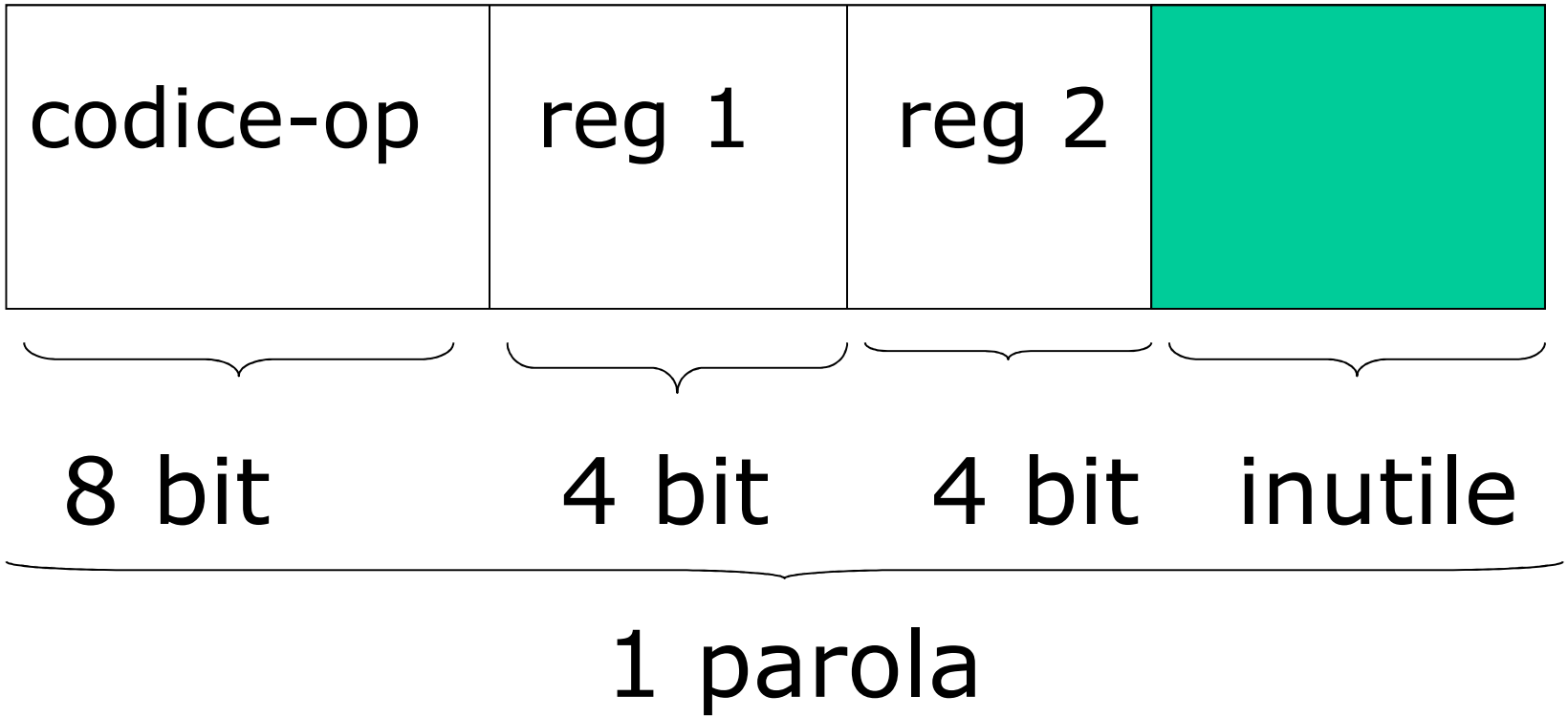
- confronta il contenuto dei registri $\mathbf{R_i}$ ed $\mathbf{R_j}$ e:
  - se $R_i < R_j$ mette -1 nel registro RC
  - se $R_i = R_j$ mette 0 in RC
  - se $R_i > R_j$ mette 1 in RC



Codici:

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

FORMATO:



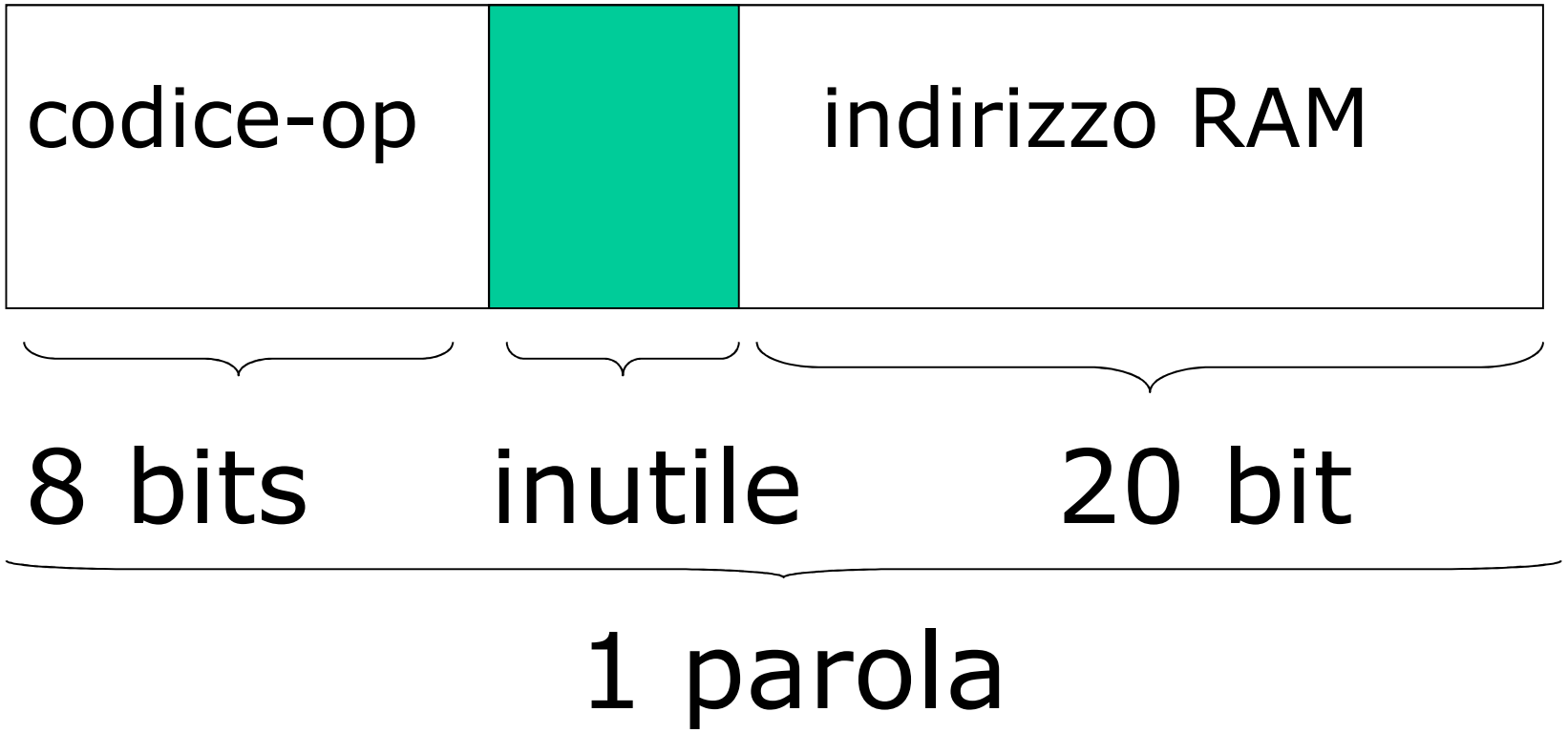
Salto

- 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:



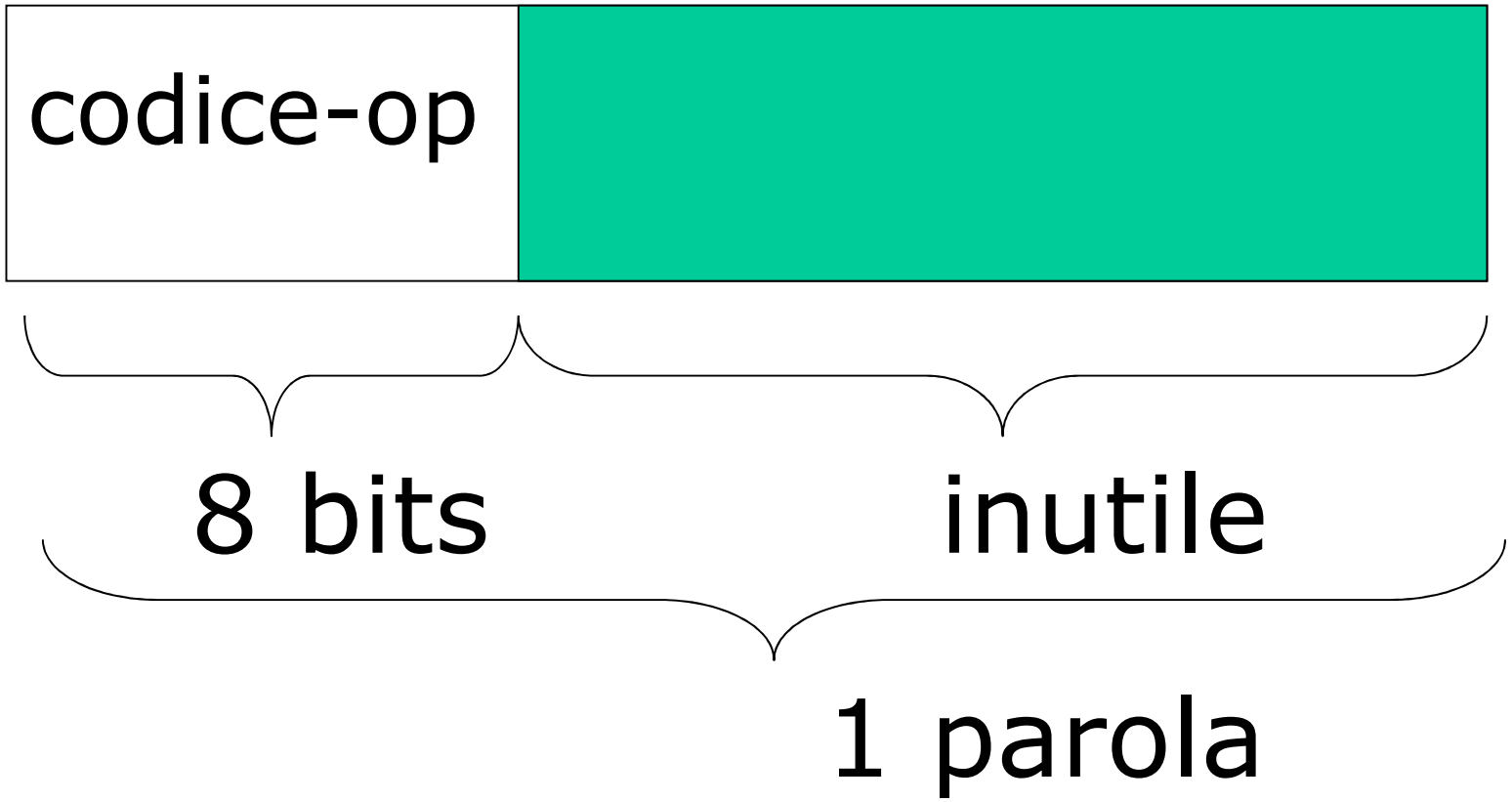
STOP

termina il programma

Codice **STOP 10000001**

:

FORMATO:



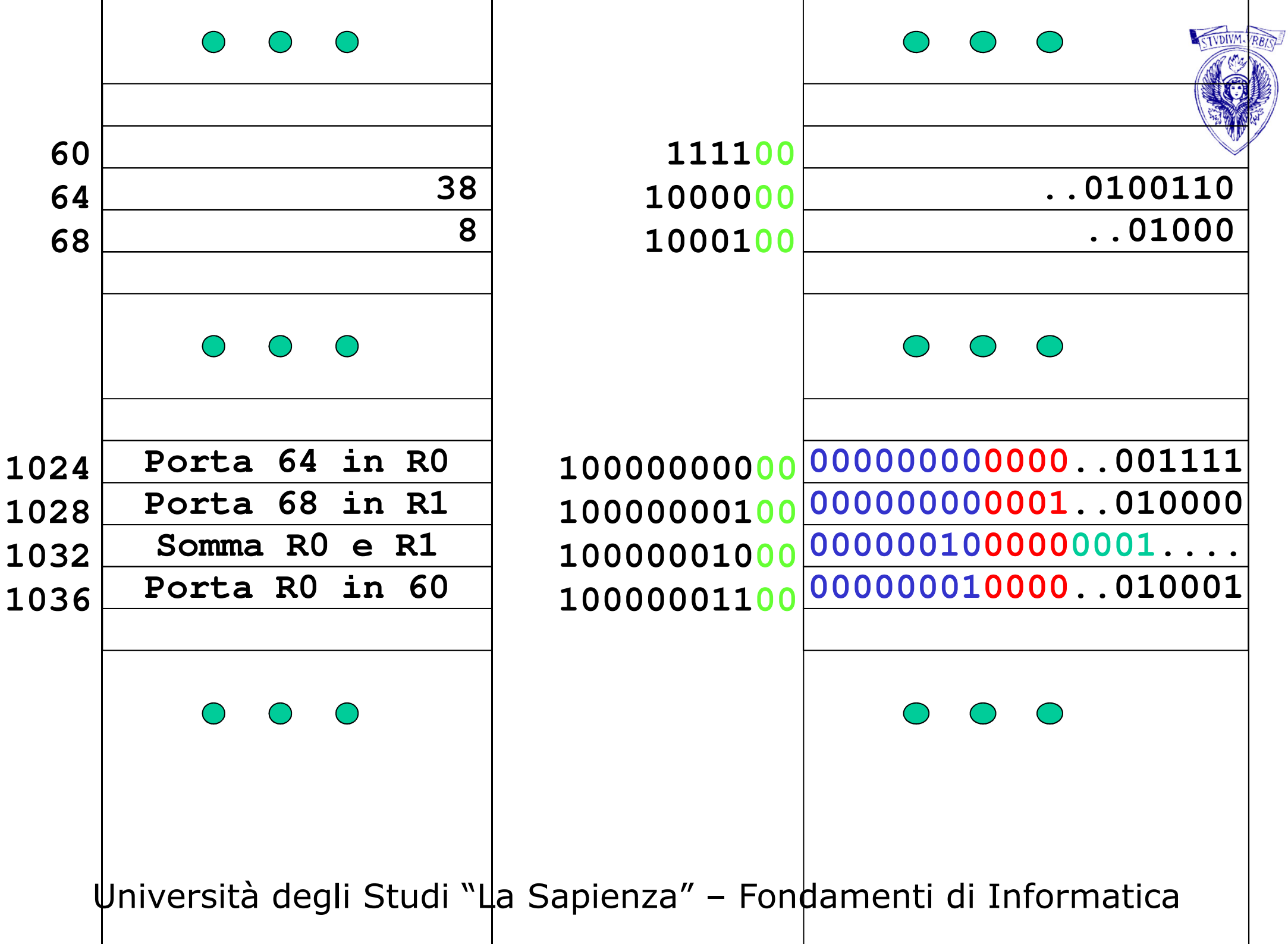
Esempio

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
1028
1032
1036
Porta 64 in R0
Porta 68 in R1
Somma R0 e R1
Porta R0 in 60
111100
1000000
1000100
1000000000
1000000100
1000001000
1000001100
..0100110
..01000
000000000000..001111
000000000001..010000
000000100000000001....
000000010000..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

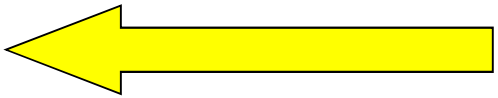
- codici mnemonici per le operazioni
- nomi mnemonici (identificatori) al posto degli indirizzi RAM per i dati (e indirizzi RAM delle istruzioni usate nei salti)
- avanzate: tipi dei dati `INT` e `FLOAT`

codice-op mnemonici:

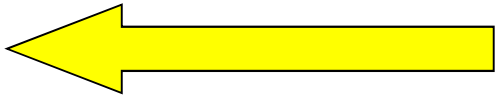
- trasferimento: **LOAD** (RAM → CPU) e **STORE** (CPU → RAM)
- aritmetiche: **ADD, SUB, DIV, MULT, MOD, FADD, FSUB, FDIV, FMULT**
- input/output: **READ** (U-INP → CPU), **WRITE** (CPU → U-OUT)
- test: **COMP, FCOMP**
- salto: **BREQ, BRGT, BRLT, BRGE, BRLE, BRANCH**
  - EQ = Equal
  - GT = Greater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



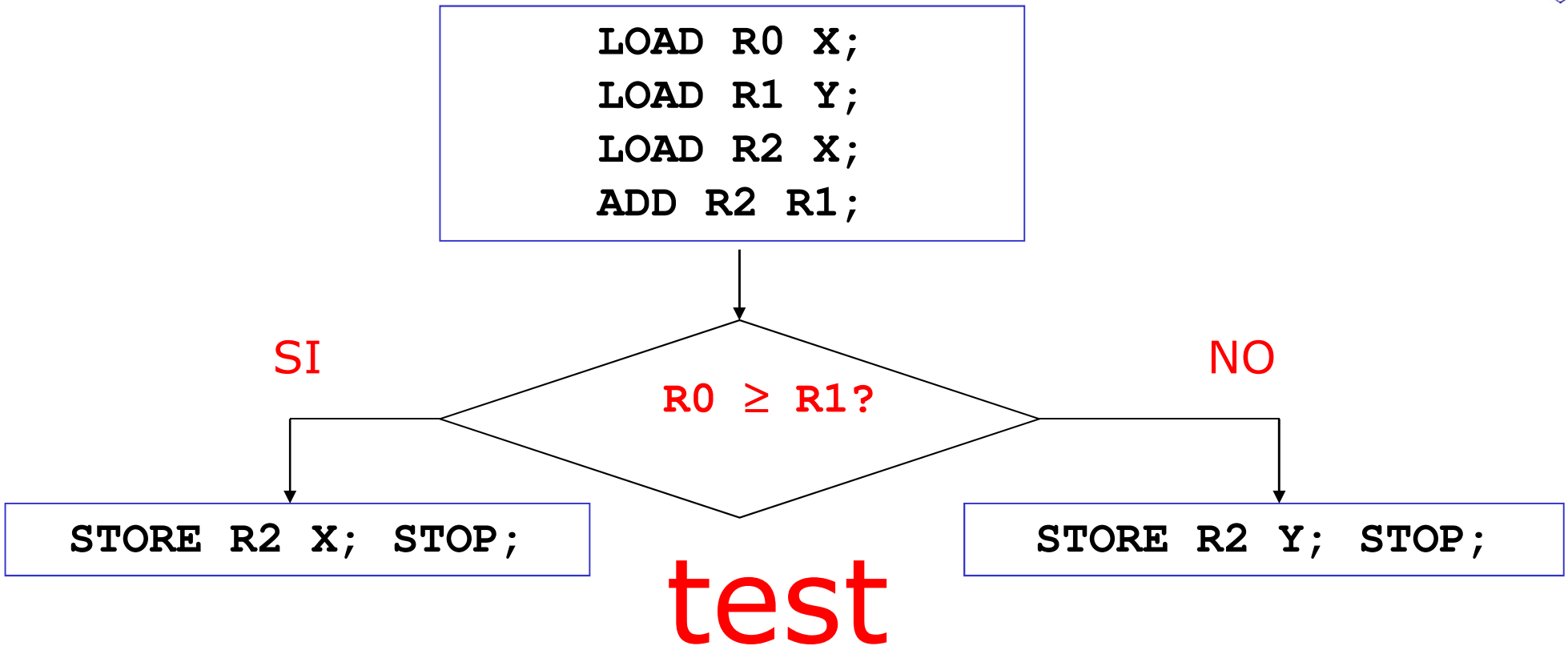
istruzioni assembler

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