

Il processore e le istruzioni macchina

Istruzioni macchina

- I circuiti della CPU non sono in grado di eseguire direttamente le istruzioni dei linguaggi di alto livello (come il C o il C++); essi sono in grado di interpretare ed eseguire soltanto un numero molto limitato di istruzioni semplici
- Queste istruzioni "di base", comprensibili direttamente dall'hardware, sono dette **istruzioni macchina** e, per poter essere eseguite dal processore, devono essere memorizzate in formato binario nella memoria principale
- Le istruzioni macchina, quindi, sono **sequenze di bit (0/1)** e rappresentano le uniche istruzioni "comprensibili" alla CPU
- L'insieme delle istruzioni macchina forma il **linguaggio macchina** del processore
- Ogni famiglia di processori possiede il proprio linguaggio macchina

Istruzioni macchina

Le istruzioni macchina permettono di compiere semplici operazioni tra registri, o tra registri e memoria principale

Esempi di istruzioni tipiche:

- Istruzione di **LOAD**: trasferisce una quantità fissa di dati (tipicamente 8, 16, 32, 64 bit) dalla memoria principale a un registro
- Istruzione di **STORE**: trasferisce il contenuto di un registro dalla CPU alla memoria principale
- Somma il contenuto di due registri e porre il risultato in un registro
- Sottrarre, moltiplicare, dividere, calcolare l'AND, l'OR, ecc.

Linguaggio assembly

- Programmare direttamente in linguaggio macchina è estremamente faticoso (un programma in linguaggio macchina è un insieme di stringhe binarie ...)
- Per ovviare a questo problema, ad ogni istruzione in linguaggio macchina viene associata una forma simbolica (più facilmente memorizzabile da parte del programmatore) detta **istruzione assembly**
- I programmi scritti in tale linguaggio sono detti programmi in linguaggio assembler (assembly language)
- Il linguaggio assembly rappresenta quindi una **forma simbolica** del linguaggio macchina (che è invece esclusivamente binario)
- Esiste una corrispondenza "uno-a-uno" fra istruzioni macchina e istruzioni assembly

Esempio di linguaggio assembly

- Prima di studiare un linguaggio assembly reale, vediamo un semplice linguaggio assembly per una CPU "di fantasia"

Istruzione macchina	Significato	Istruzione assembly
0000111111101000	copia nel registro R4 il contenuto della cella di memoria con indirizzo 1000	LOAD R4, [1000]
0000111111101010	copia nella cella di memoria con indirizzo 1004 il contenuto del registro R2	STORE [1004],R2
0010010010000000	Esegui la somma fra il contenuto di R1 e R3, e poni il risultato nel registro R2	ADD R2,R1,R3

Sintassi dei linguaggi assembly

- Ogni linguaggio assembly utilizza determinate convenzioni. Nel nostro esempio, ogni istruzione è composta da:
 - un nome **mnemonico** che indica il tipo di operazione (a volte detto **operation code**, abbreviato **opcode**)
 - una lista di **operandi** separati dal carattere virgola; gli operandi rappresentano le "entità" coinvolte nell'operazione
- Nel nostro esempio, gli operandi sono registri (indicati con **R1,R2,..**) o locazioni di memoria (indicate con **[indirizzo]**)
- Nel nostro esempio, LOAD e STORE sono **istruzioni a due operandi**, mentre ADD è un'**istruzione a tre operandi** (ciò significa che è obbligatorio fornire SEMPRE tre operandi)
- Nota: è possibile interpretare le istruzioni come "comandi" e gli operandi come i parametri (obbligatori) del comando

Sintassi dei linguaggi assembly

- Il nostro linguaggio segue la **convenzione "destinazione-sorgente"**: se un'istruzione prevede più operandi, il primo di essi rappresenta la destinazione e i successivi rappresentano la/le sorgenti

- Esempio **SUB R4, R1, R5** significato: "R4 <-- R1 - R5"

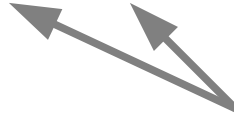
**Opcode o
mnemonico**



**destinazione
(luogo in cui
verrà salvato
il risultato)**

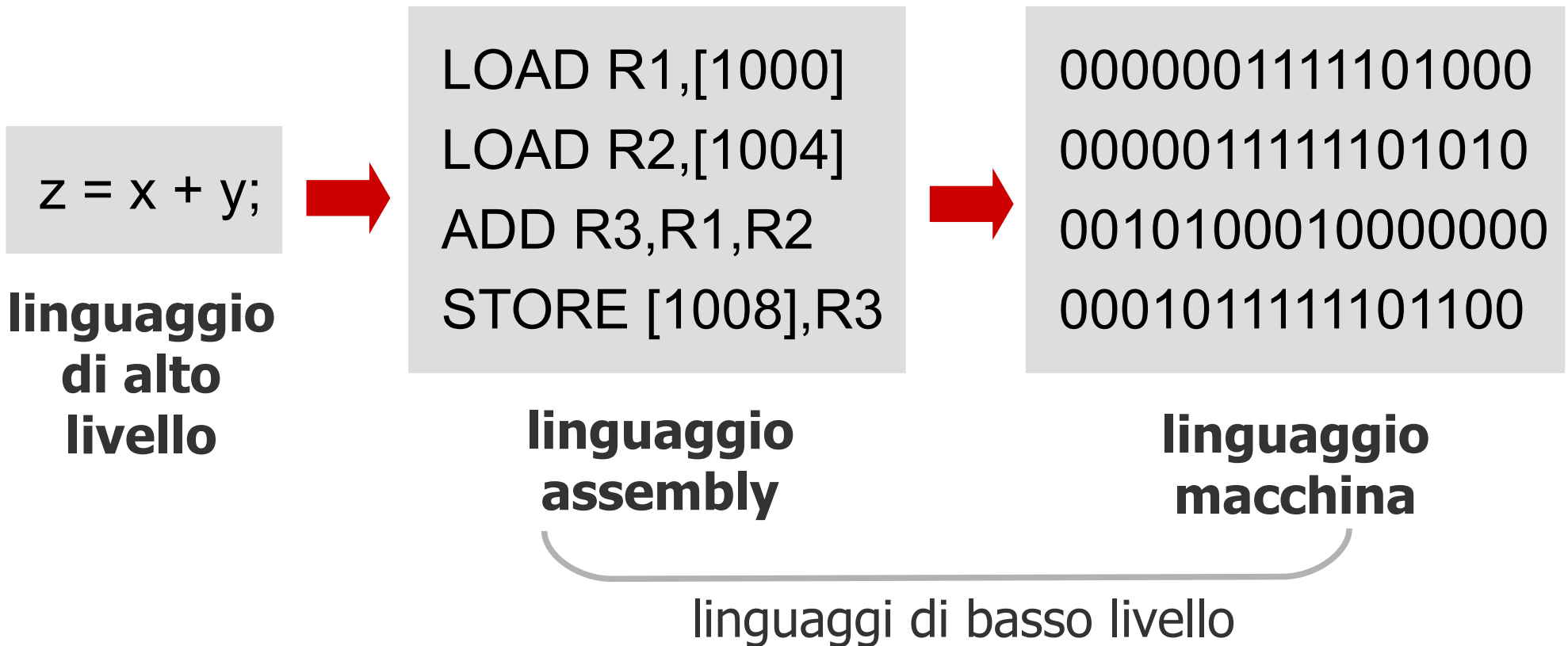


sorgenti



Un esempio di traduzione

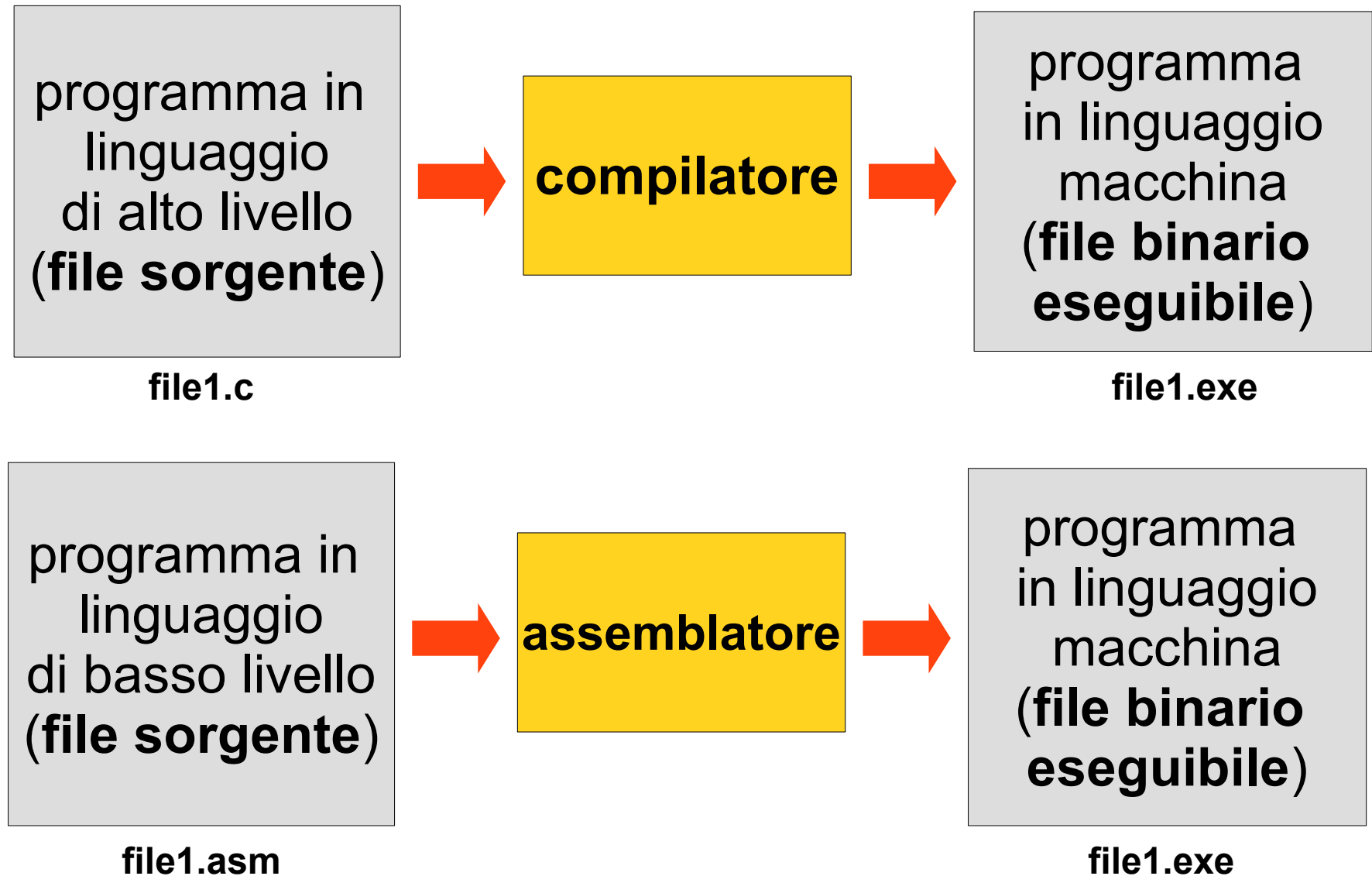
- Traduzione da linguaggio di alto livello (C) ad assembly
- Supponiamo che le variabili **x**, **y** e **z** siano di tipo **int** e si trovino agli indirizzi di memoria 1000, 1004 e 1008



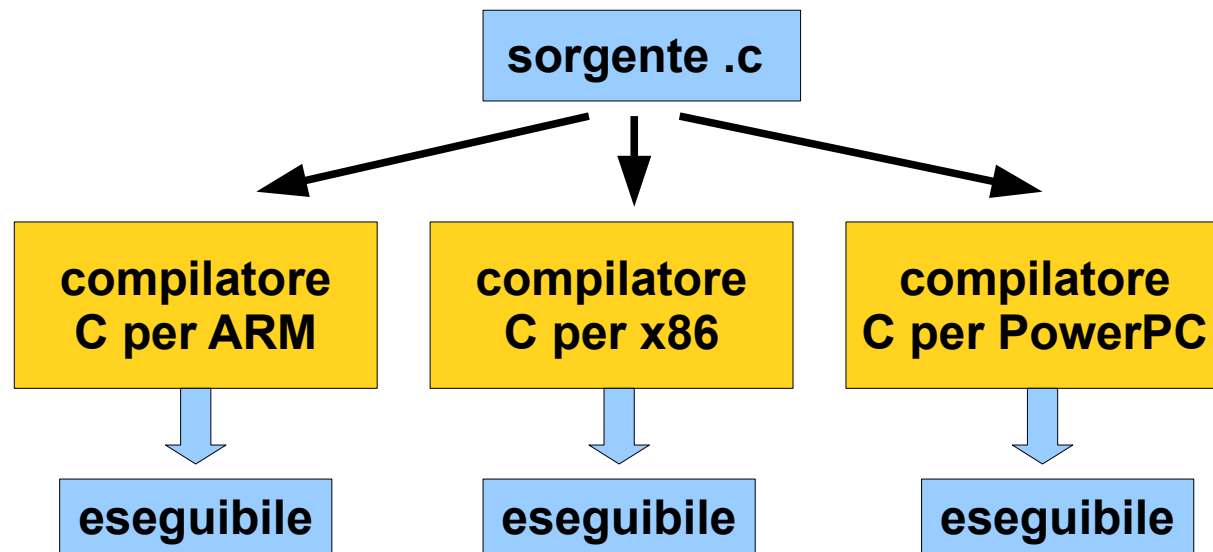
Compilatori e assembleri

- L'operazione di traduzione da un linguaggio di alto livello a un linguaggio di basso livello è detta **compilazione**
- La traduzione da linguaggio assembly a linguaggio macchina è detta assemblaggio, e il software che la esegue è detto **assemblatore** (assembler)
- L'assemblatore riceve in input un file di testo contenente il programma assembly (cioè il file sorgente, avente estensione tipica .asm o .s) e produce un file binario in cui ogni istruzione assembly è stata sostituita con la corrispondente istruzione macchina
- Nota: ogni istruzione di alto livello, per essere tradotta, richiede in genere più istruzioni assembly, mentre un'istruzione assembly si traduce in una sola istruzione macchina

Compilatori e assembleri



Compilatori e architetture



- Poichè ogni famiglia di processori possiede un diverso linguaggio macchina, il compilatore deve conoscere non solo il linguaggio di alto livello, ma anche il linguaggio macchina del processore "target" (cioè il processore su cui il programma dovrà essere eseguito)
- Esempio: se si vuole produrre un programma eseguibile per un processore ARM usando il linguaggio C, allora si dovrà utilizzare un "compilatore C per architettura ARM"
- Nota: sono disponibili compilatori C per moltissime architetture

Esercizio: progetto di un linguaggio macchina

Vogliamo progettare il linguaggio macchina di un'ipotetica CPU avente le seguenti caratteristiche:

- La CPU è dotata di **4 registri a 8 bit**
- La massima quantità di **memoria principale** supportata dalla CPU è **1KB (= 2^{10} byte)**
- La CPU deve essere in grado di interpretare le **12 istruzioni** riportate nella tabella seguente
- Esempio: **LOAD** <reg> , [<ind>]

dove <reg> indica uno dei 4 registri (R1, R2, R3, R4) e <ind> indica un indirizzo di memoria (un numero tra 0..1023)

Insieme di istruzioni (ISA)

Mnemonico	Sintassi	Esempio	Significato
LOAD	LOAD <reg> , [<ind>]	LOAD R1, [1000]	$R1 \leftarrow \text{RAM}[1000]$
STORE	STORE [<ind>] , <reg>	STORE [1000], R3	$\text{RAM}[1000] \leftarrow R3$
ADD	ADD <reg> , <reg> , <reg>	ADD R4,R1,R3	$R4 \leftarrow R1 + R3$
SUB	SUB <reg> , <reg> , <reg>	SUB R4,R1,R3	$R4 \leftarrow R1 - R3$
MUL	MUL <reg> , <reg> , <reg>	MUL R4,R1,R3	$R4 \leftarrow R1 * R3$
DIV	DIV <reg> , <reg> , <reg>	DIV R4,R1,R3	$R4 \leftarrow R1 / R3$
OR	OR <reg> , <reg> , <reg>	OR R4,R1,R3	$R4 \leftarrow (R1 \text{ OR } R3)$
AND	AND <reg> , <reg> , <reg>	AND R4,R1,R3	$R4 \leftarrow (R1 \text{ AND } R3)$
NOT	NOT <reg>	NOT R1	$R1 \leftarrow \text{NOT } R1$
LOADI	LOADI <reg> , <val>	LOADI R1, 5	$R1 \leftarrow 5$
JUMP	JUMP <ind>	JUMP 1000	Salta a indirizzo 1000
JUMPZ	JUMPZ <ind>	JUMPZ 1000	Se l'ultima operazione della ALU ha prodotto 0, salta a indirizzo 1000

Insieme di istruzioni (ISA)

■ Note:

- **LOADI** carica un valore **immediato** (cioè una costante) in un registro
- **JUMP** è una istruzione di **salto incondizionato**: l'istruzione ha l'effetto di impostare il program counter (PC) con l'indirizzo fornito come argomento.

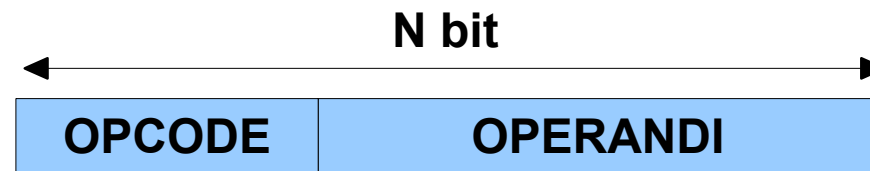
Esempio: l'istruzione "JUMP 1000" carica il valore 1000 nel PC, e quindi ha l'effetto di "saltare" all'istruzione che si trova all'indirizzo 1000

- **JUMPZ** è un **salto condizionato**: esegue il salto solo se l'ultima operazione effettuata dalla ALU ha prodotto esito 0.

Esercizio: progetto di un linguaggio macchina

Soluzione

- Dobbiamo stabilire come rappresentare un'istruzione tramite una sequenza di bit (cioè come codificare in binario l'istruzione)
- Per semplicità, supponiamo di utilizzare **istruzioni a lunghezza fissa**: ogni istruzione è rappresentata con lo stesso numero di bit (N)
- Idea: usiamo un gruppo di bit per codificare l'operazione (LOAD,STORE,ADD,...) e i bit restanti per codificare gli operandi



- Quanti bit ci occorrono? $N = ?$

Esercizio: progetto di un linguaggio macchina

- 4 bit per codificare le 12 istruzioni ($2^4=16 < 12$)

LOAD=0000 **STORE**=0001 **ADD**=0010 ...

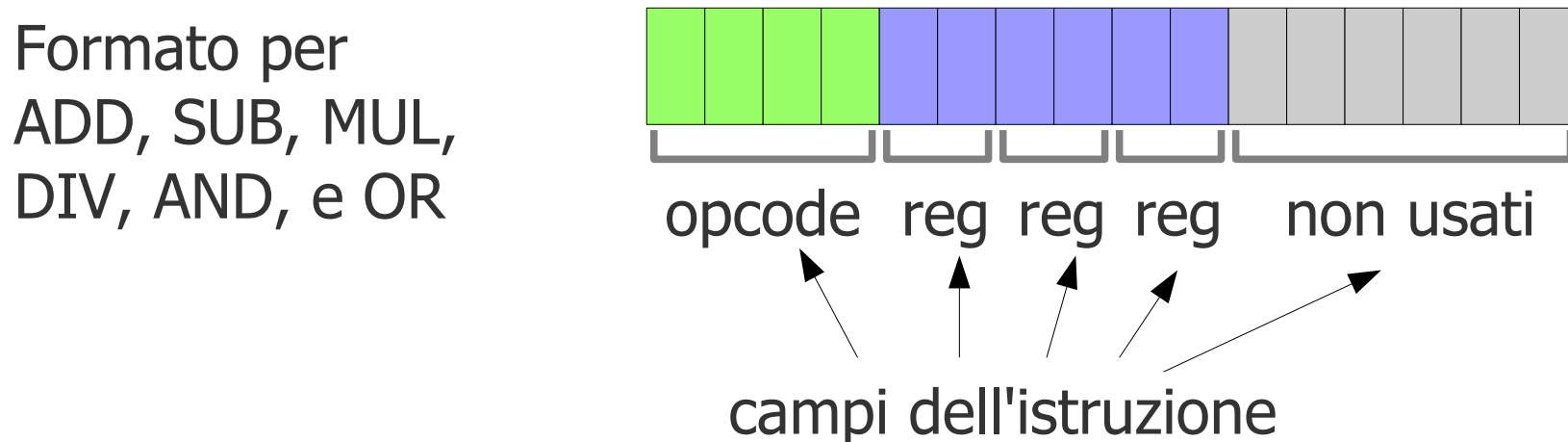
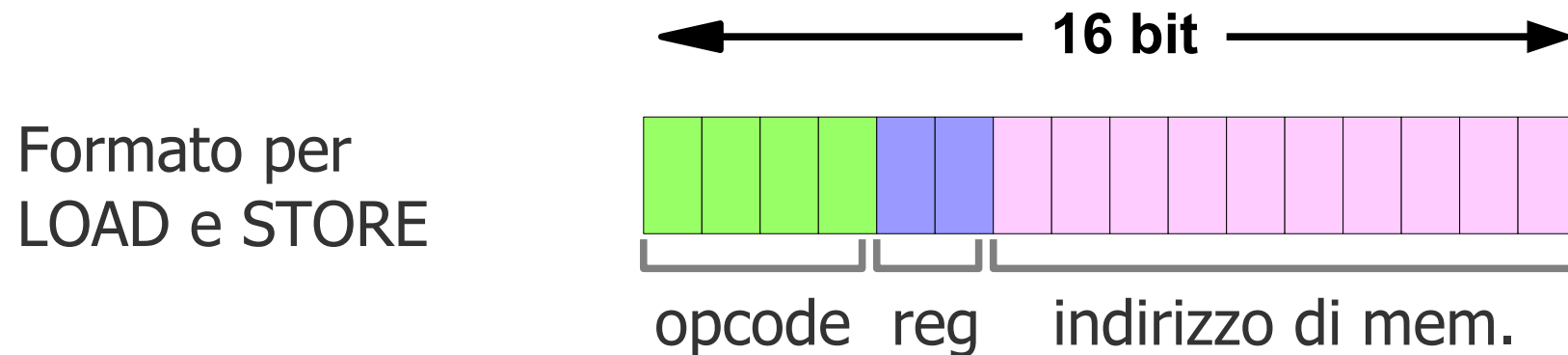
- 2 bit per codificare i 4 registri

Reg1=00 **Reg2**=01 **Reg3**=10 **Reg4**=11

- 10 bit per codificare gli indirizzi di memoria (1KB di memoria)
- Le istruzioni LOAD e STORE richiedono quindi almeno 16 bit (4 per il codice operativo, 2 per il registro e 10 per l'indirizzo di memoria)
- Le restanti istruzioni richiedono almeno 10 bit (4 bit per l'operazione e 2 bit per ognuno dei 3 registri)
- Decidiamo quindi di codificare ogni istruzione con 16 bit

Formato delle istruzioni

- Possiamo immaginare il seguente formato per le istruzioni:

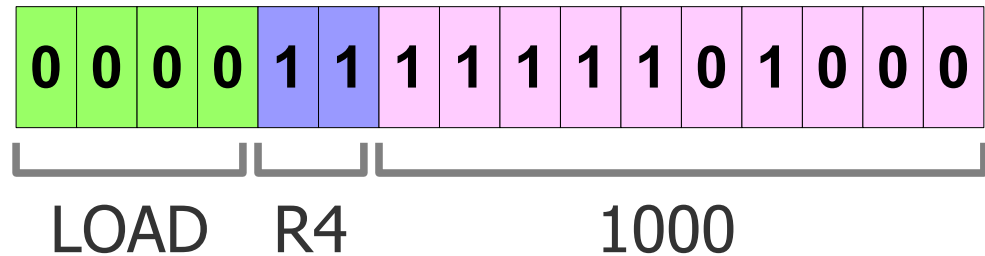


Esempio

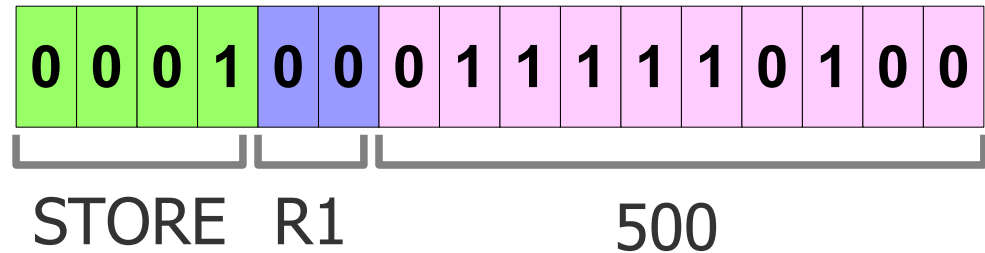
Istruzione assembly

Istruzione macchina

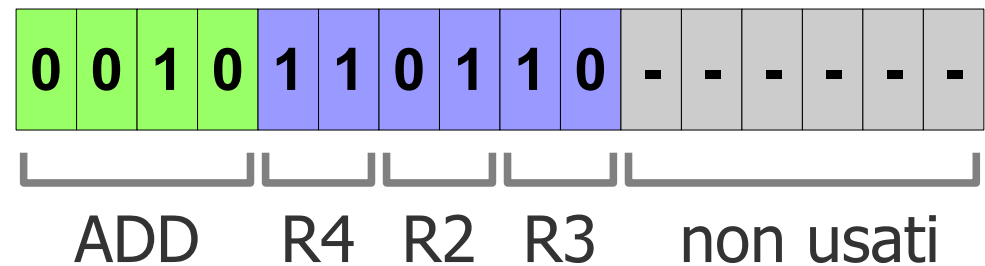
LOAD R4,[1000]



STORE [500],R1



ADD R4,R2,R3



Considerazioni sul formato delle istruzioni

- In generale, le istruzioni macchina possono avere lunghezza fissa o lunghezza variabile
- L'utilizzo di istruzioni a lunghezza variabile rende più complessa la decodifica, ma consente di avere programmi eseguibili più compatti (il programma in linguaggio macchina occupa meno memoria)
- In ogni caso, in ogni istruzione macchina è possibile identificare alcuni **campi** fondamentali: il codice dell'operazione (opcode) e gli operandi
- Nel caso di lunghezza variabile, in genere i primi bit dell'opcode consentono di stabilire la lunghezza dell'istruzione
- Esempio: le istruzioni macchina del processore 8086 hanno una lunghezza variabile da 1 a 6 byte.


Registri con funzione speciale

Ogni CPU contiene, oltre ai registri di uso generale, anche alcuni registri con funzioni "speciali":

- Registro **Program Counter (PC)**: contiene l'indirizzo di memoria della prossima istruzione da eseguire
- Registro **Instruction Register (IR)**: contiene la codifica binaria dell'istruzione in corso di esecuzione (cioè contiene l'istruzione corrente, in linguaggio macchina)
- Registro dei **flag**: contiene informazioni sullo stato della CPU e sull'ultima operazione eseguita dalla ALU; è detto anche **registro di stato** o **PSW (program status word)**

Il ciclo fetch/decode/execute

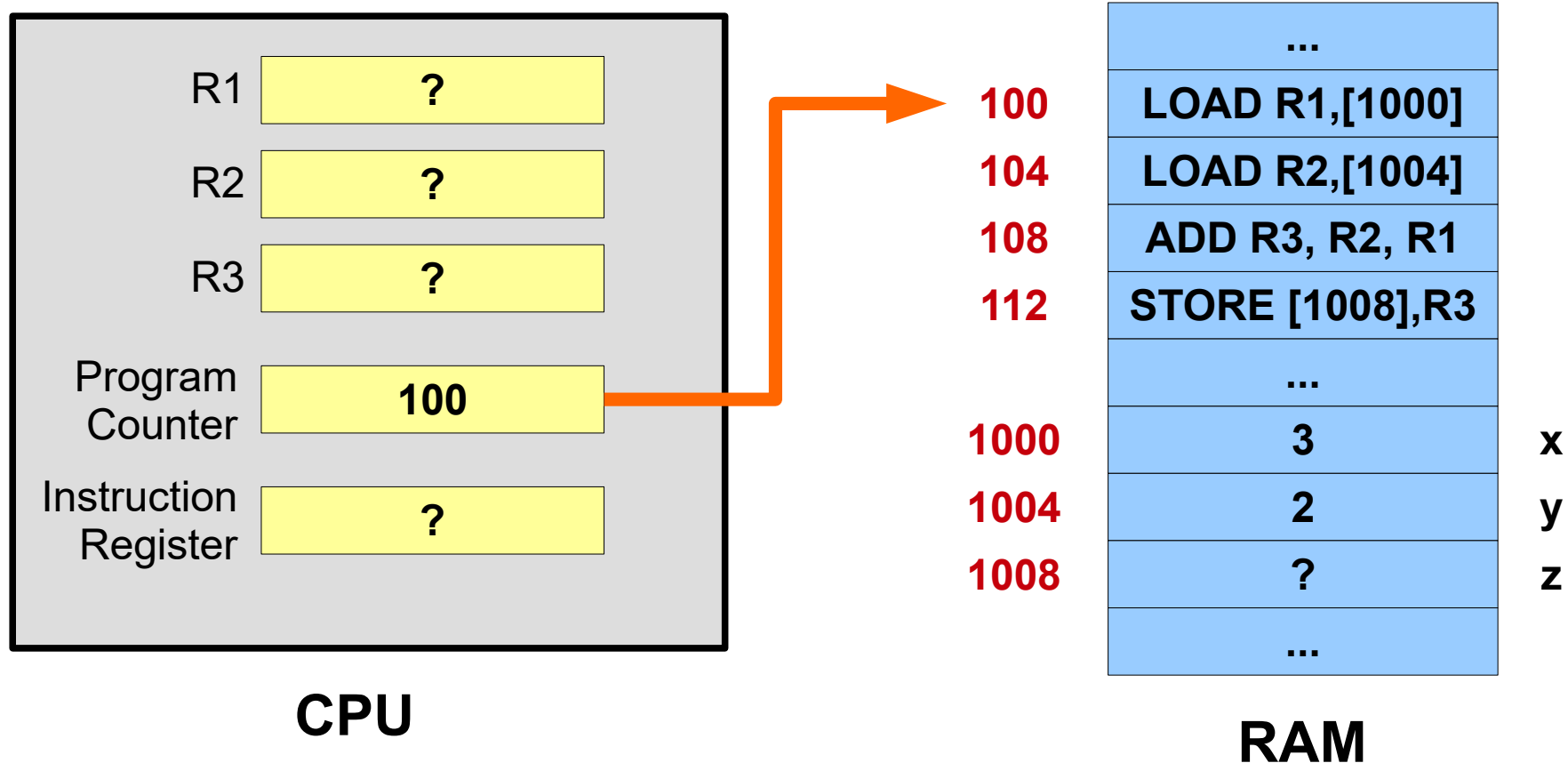
- Ogni CPU, dal momento in cui viene accesa al momento in cui viene spenta, esegue ciclicamente i seguenti passi:

- 
- 1) Fetch:** preleva l'istruzione il cui indirizzo è memorizzato nel registro PC, e la pone nel registro IR
 - 2) Decode:** decodifica l'istruzione (cioè analizza l'istruzione macchina, determina il tipo di istruzione, la sua lunghezza*, gli operandi,...). Il program counter è incrementato sommandovi la lunghezza dell'istruzione appena analizzata, in modo che esso contenga ora l'indirizzo della prossima istruzione da eseguire
 - 3) Execute:** esegue l'istruzione e torna al punto 1

* ricorda che, in generale, istruzioni diverse possono avere lunghezza diversa

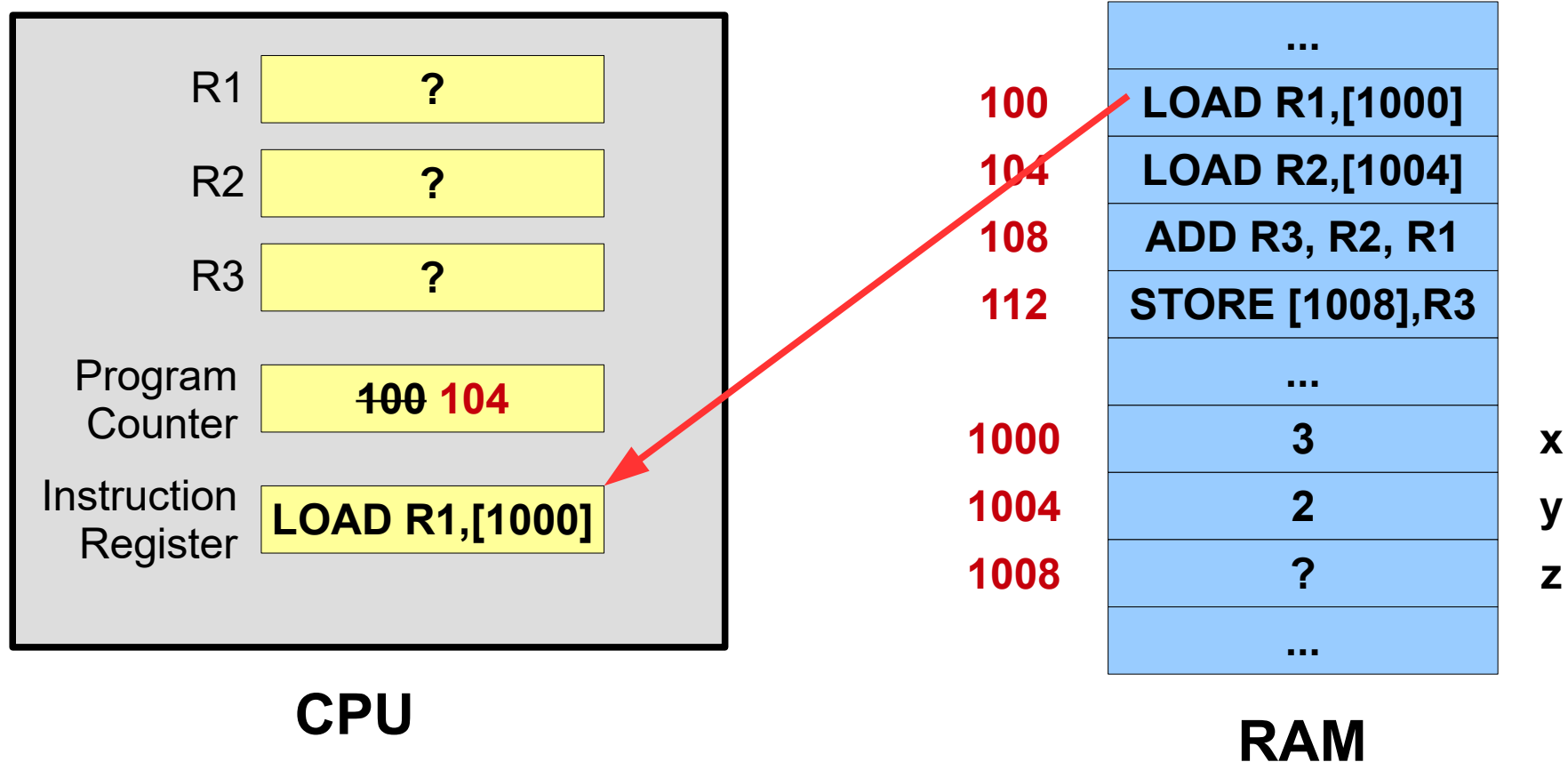
Esempio di ciclo fetch/decode/execute

Situazione iniziale: il program counter "punta" alla prima istruzione del programma



Esempio di ciclo fetch/decode/execute

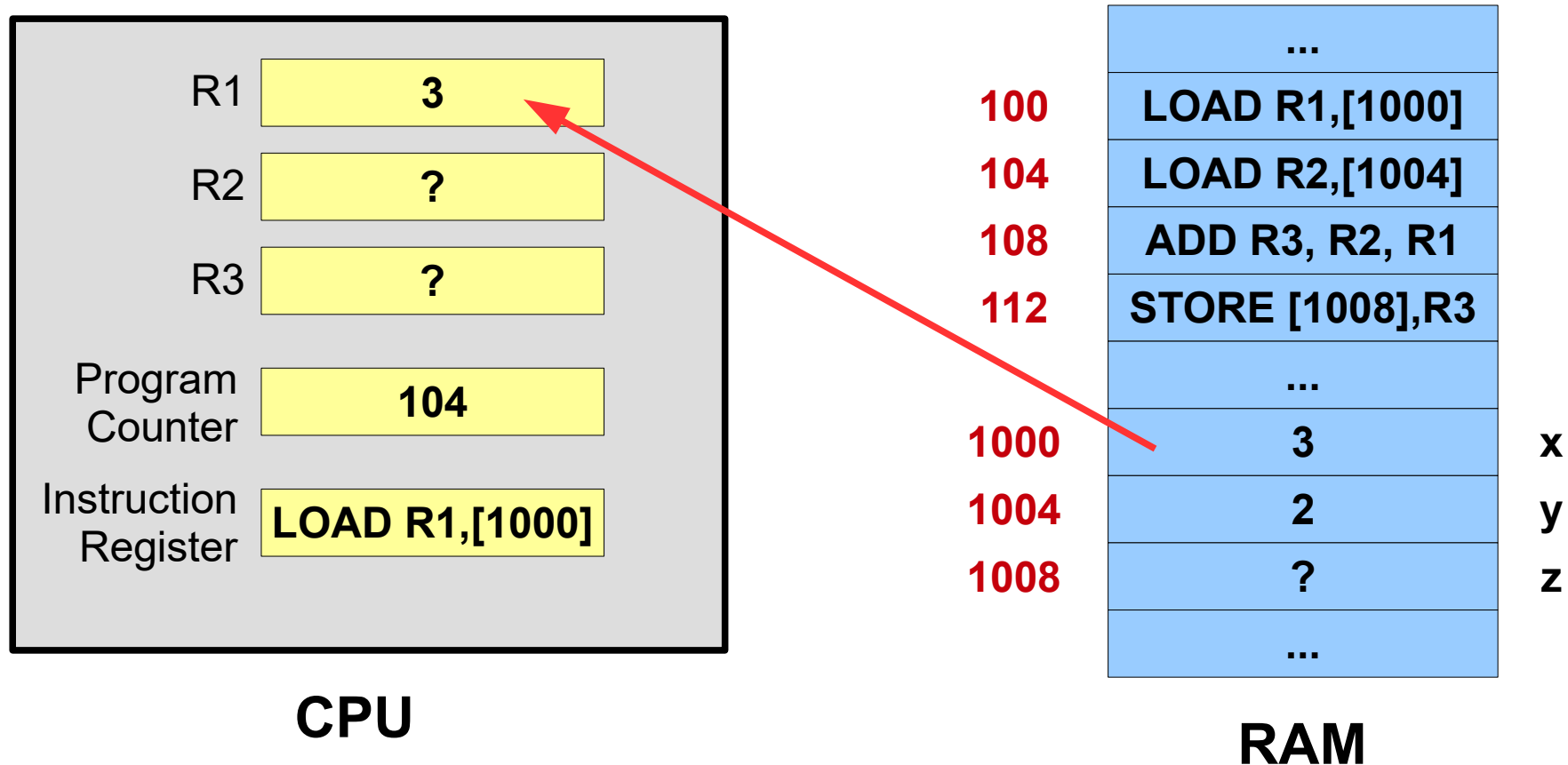
Fase di fetch: l'istruzione all'indirizzo 100 è prelevata dalla memoria e portata nel registro IR



Fase di decode: l'istruzione è analizzata. Poichè la sua lunghezza è 4 byte, il valore del PC viene incrementato di 4 unità passando così da 100 a 104

Esempio di ciclo fetch/decode/execute

Fase di execute: il valore 3 è copiato dalla RAM al registro



Ora il ciclo ricomincia:

fetch dell'istruzione all'indirizzo 104, sua decodifica, sua esecuzione,
fetch dell'istruzione all'indirizzo 108, sua decodifica, sua esecuzione,
e così via...

ISA = Instruction Set Architecture

- L'insieme delle istruzioni macchina disponibili per un dato processore⁽¹⁾ è detto **ISA** (Instruction Set Architecture, ovvero *architettura dell'insieme di istruzioni*)
- A volte ci si riferisce all'**ISA** come all'**architettura** di un computer
- Se due modelli di processori riconoscono lo stesso insieme di istruzioni macchina, si dice che essi hanno lo stesso *ISA* (c la stessa *architettura*)
- Esempio: ISA x86, ISA x64, ecc...
- Un insieme di processori con lo stesso ISA costituisce una famiglia di processori

⁽¹⁾ e di tutte le altre informazioni che occorre conoscere per scrivere programmi "macchina" (come ad esempio, il numero di registri disponibili, il modello di memoria, ecc..)

CISC e RISC

- **CISC** = Complex Instruction Set Computer
- **RISC** = Reduced Instruction Set Computer
- I processori dotati di un ISA con un elevato numero di istruzioni (di cui alcune "complesse" e non indispensabili) sono detti processori CISC
- I processori RISC sono invece dotati di un minor numero di istruzioni (tutte "semplici")
- In generale, un processore RISC richiede più istruzioni per eseguire un programma rispetto a un processore CISC, ma le istruzioni RISC sono eseguite in modo più veloce (minor numero di cicli di clock per istruzione)
- La tendenza attuale è quella di progettare CPU di tipo RISC
- Architettura ARM (**ARM = Advanced RISC Machine**)

Istruzioni memoria-memoria e memoria-registro

- Un processore potrebbe essere dotato di un'istruzione che preleva due valori dalla memoria principale e ripone in risultato in memoria principale.

Esempio ADDMEM [100], [200], [300]

- Le istruzioni che prelevano uno o più dati dalla memoria e ripongono il risultato in memoria sono dette **istruzioni memoria-memoria**
- Le istruzioni memoria-memoria sono più complesse delle istruzioni **registro-memoria** (o memoria-registro)
- Nella progettazione di processori RISC si cerca di far sì che le uniche istruzioni che accedono alla memoria principale siano le istruzioni di LOAD e STORE (e non si inseriscono istruzioni memoria-memoria)