# Algoritmi e Programmi

Marta Capiluppi
marta.capiluppi@univr.it
Dipartimento di Informatica
Università di Verona

# I Dati

Ogni variabile è caratterizzata da

- Nome
- Valori
- Tipo
  - Numeri naturali o interi o reali (1, -2, 0.34)
  - Caratteri alfanumerici (A, B, ..)
  - Dati logici o booleani (Vero, Falso)

# Criteri di classificazione dei dati

- Visibilità da parte dell'utente
  - visibile (di ingresso o uscita)
  - trasparente (dati temporanei di supporto)
- Variabilità nel tempo
  - costanti
  - variabili (acquisizione dall'esterno o assegnazione)
- Struttura
  - elementari (interi, alfanumerici, booleani, …)
  - strutturati (array, matrici, …)

# Tipi

- Tipi predefiniti
  - Numerici
  - Booleani
  - Alfanumerici
  - Stringa
  - Data
- Variabili strutturate
  - Array (vettori)
  - Matrici (array multidimensionali)
  - Record (strutture complesse definite dall'utente)

# Operazioni e tipi

La stessa operazione su tipi diversi può portare a risultati diversi

Ex: +

Numerico x = 10 ➔ x+1 = 11

Stringa x = “ciao” ➔ x+”a tutti” = “ciao a tutti”

Stringa numerica x = “10” ➔ x+1 = “101”

Data x = 18.11.2010 ➔ x+1= 19.11.2010

# Variabili strutturate

Array (vettori):

- Nome
- Tipo
- Può contenere n elementi
- v[i], i=1,…,n (i=0,…,n-1)

Matrici:

- Nome
- Tipo
- Può contenere nxm elementi
- m[i,j], i=1,…,n  j=1,…,m

# Variabili strutturate

Strutture definite dall'utente:

- Nome
- Campi (componenti)
- Può comprendere componenti di diverso tipo

Ex:

Struttura prodotto {

alfanumerico nome

razionale costo

}

prodotto p.costo = 10

# Computabilità

- È sempre possibile trovare una soluzione algoritmica ad un problema?
- Esiste un esecutore automatico in grado di eseguire un algoritmo e se sì come è fatto?

Teoria della computabilità

Una funzione si dice computabile se esiste un algoritmo che la calcola

# Tesi di Church-Turing

- Risultato fondamentale:
  - *se un problema è intuitivamente calcolabile, allora esisterà una macchina di Turing (o un dispositivo equivalente, come il computer) in grado di risolverlo (cioè di calcolarlo)*
  - *La classe delle funzioni calcolabili coincide con quella delle funzioni calcolabili da una macchina di Turing.*
  - Tesi *non dimostrata*, ma dedotta dalla sostanziale equivalenza delle varie definizioni proposte per la computabilità e *mai contraddetta finora*
- Conseguenze:
  - Tutti gli esecutori sono equivalenti alla Macchina di Turing
  - Gli esecutori differiscono tra loro solo nella velocità di risoluzione dei problemi, non nella capacità di risolverli

# Macchina di Turing

- Una macchina di Turing è una macchina formale, cioè un sistema formale che può descriversi come un meccanismo ideale, ma in linea di principio realizzabile concretamente, che può trovarsi in stati ben determinati (macchina a stati), opera su stringhe in base a regole ben precise e costituisce un modello di calcolo.
- Viene largamente usata nella teoria della calcolabilità e nello studio della complessità degli algoritmi: risorse di calcolo necessarie (tempo e memoria) ➔ efficienza
- Per quel che riguarda la misurazione della risorsa tempo, data una macchina di Turing M, si dice che M opera in tempo f(n) se dato un input x di lunghezza n, la macchina M produce il risultato in f(n) passi.
- Per quel che riguarda la misurazione della risorsa spazio, data una macchina di Turing M, si dice che M opera in spazio f(n) se dato un input x di lunghezza n, la macchina M utilizza f(n) celle "temporanee" per effettuare la computazione. Per temporanee si intende che la dimensione dell'input e la dimensione dell'output non vengono conteggiate in f(n).

# Macchina di Turing

- Primo modello di esecutore automatico
  - Composta da un nastro infinito ed una unità di controllo con stato che può scrivere, leggere e cancellare simboli sul nastro
- Idea fondamentale:
  - La macchina opera eseguendo istruzioni del tipo “se è vero che… allora esegui…”
  - Set di istruzioni minimo e completo
- Un’evoluzione della macchina consiste in una sequenza di sue possibili "configurazioni” (stato interno attuale, contenuto del nastro eposizione sul nastro della testina di I/O). L'evoluzione ad un certo punto si arresta se non si trova nessuna istruzione in grado di farla proseguire. Si può avere un arresto in una configurazione "utile" dal punto di vista del problema che si vuole risolvere (quello che si trova registrato sul nastro rappresenta il risultato dell'elaborazione). Si può avere però anche un arresto "inutile" che va considerato come una conclusione erronea dell'elaborazione. Può anche accadere che un'evoluzione non abbia mai fine.

# Automi

Un automa a stati finiti deterministico è una tupla:

$$A=(\Sigma,S,\delta,s_0,F)$$

- $\Sigma=\{a_0,a_1,\ldots,a_n\}$ insieme finito di simboli (alfabeto)
- $S=\{s_0,s_1,\ldots,s_m\}$ insieme finito di stati
- $\delta=S \times \Sigma \rightarrow S$ funzione di transizione
- $s_0 \in S$ stato iniziale
- $F \subseteq S$ insieme di stati finali

Una configurazione di A è (s,x) con s stato e x stringa dell'alfabeto

Funzione di transizione: $(s,x) \rightarrow (s',y)$ se

- esiste $a \in \Sigma$ t.c. $x = ay$
- $\delta(s,a) = s'$

# Linguaggio di Programmazione

- Notazione con cui è possibile descrivere gli algoritmi
- Programma: rappresentazione di un algoritmo in un particolare linguaggio di programmazione
- Ogni linguaggio è caratterizzato da una sintassi e da una semantica

# Linguaggi e Automi

L’insieme delle stringhe riconosciute da un automa
A forma un linguaggio formale

Linguaggio riconosciuto (o accettato) da A

# Sintassi e semantica

- ***Sintassi***: l'insieme delle regole che specificano come comporre istruzioni ben formate;
- ***Semantica:*** di ogni istruzione ben formata specifica il significato, e quindi la successione delle operazioni che vengono compiute allorché l'istruzione viene eseguita.

# Classificazione

- In base all'astrazione:
  - Linguaggi di basso livello (vicini all'hardware):
    - I generazione: linguaggi macchina (sequenze di bit)
    - II generazione: linguaggi assemblativi (uso di codici mnemonici per le istruzioni)
  - Linguaggi di alto livello (vicini all'utente):
    - III generazione: linguaggi imperativi e procedurali di uso generale
    - IV generazione: linguaggi per specifici ambiti applicativi
    - V generazione: linguaggi di descrizione dei problemi orientati alla risoluzione automatica

# Il linguaggio macchina

- Il linguaggio macchina è direttamente eseguibile dall'elaboratore, senza nessuna traduzione.
  - Istruzioni ed operandi relativi al programma in esecuzione sono caricati in memoria e quindi sono memorizzati in forma *binaria*.
  - Vincolo: conoscenza dei metodi di rappresentazione delle informazioni utilizzati.

# Il linguaggio Assembler

- Le istruzioni corrispondono univocamente a quelle macchina, ma vengono espresse tramite nomi simbolici (parole chiave).
- Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina (assemblatore).
- Vincolo: necessità di conoscere in dettaglio le caratteristiche della macchina (registri, dimensione dei dati, set di istruzioni)
- Anche semplici algoritmi implicano la specifica di molte istruzioni

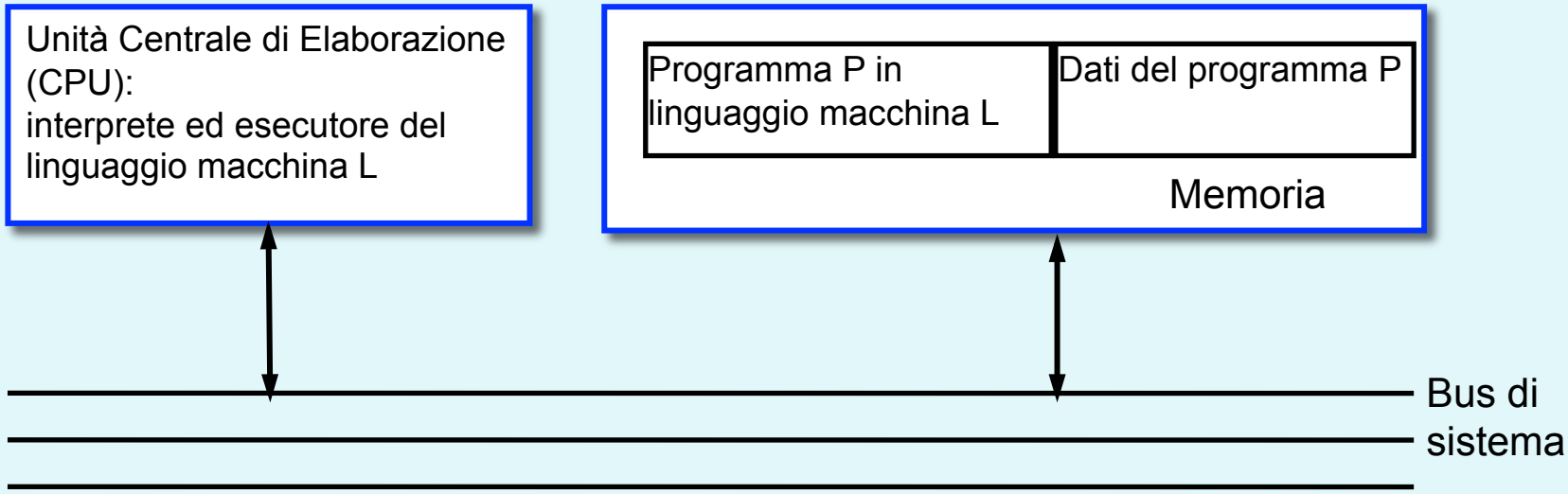
# Esempio

| Cella di memoria | Istruzione | Operatore |
|---|---|---|
| 0 | READ | 8 |
| 1 | READ | 9 |
| 2 | LOADA | 8 |
| 3 | LOADB | 9 |
| 4 | MUL | |
| 5 | STOREA | 8 |
| 6 | WRITE | 8 |
| 7 | HALT | |
| 8 | INT | |
| 9 | INT | |

| Cella di memoria | Contenuto |
|---|---|
| 0 | 0100000000001000 |
| 1 | 0100000000001001 |
| 2 | 0000000000001000 |
| 3 | 0001000000001001 |
| 4 | 1000000000000000 |
| 5 | 0010000000001000 |
| 6 | 0101000000001000 |
| 7 | 1101000000000000 |
| 8 | 0000000000000000 |
| 9 | 0000000000000000 |

Moltiplicazione tra 2 numeri interi: 2 registri, A e B; prima vengono salvate in memoria le istruzioni, poi gli operatori
NB: nella codifica il calcolatore deve conoscere il tipo degli operatori!

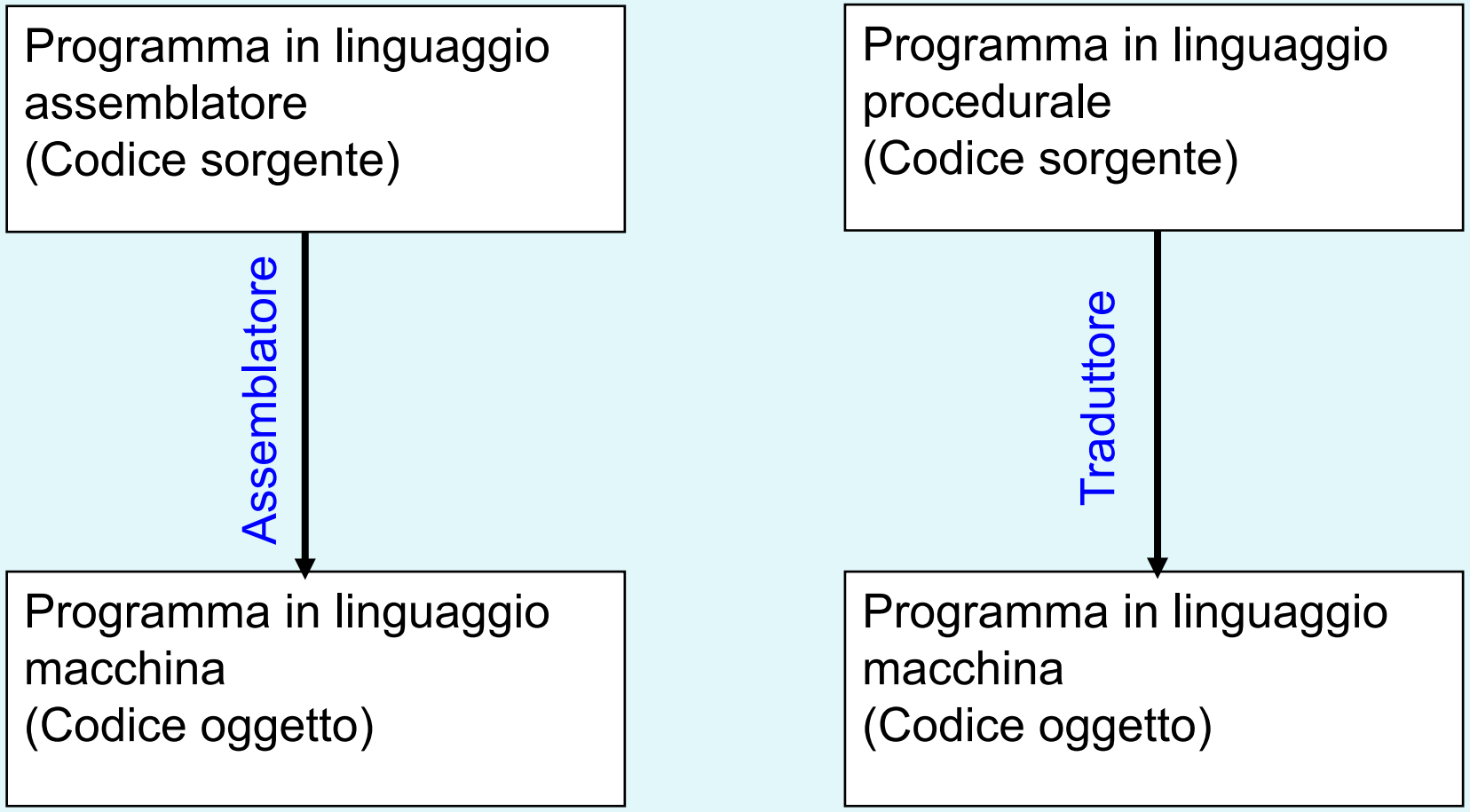
# CPU come interprete del suo linguaggio macchina



# Linguaggi di alto livello (III generazione)

- Le istruzioni esprimono una serie di azioni. Il programma prima di essere eseguito deve essere tradotto in linguaggio macchina (traduttore)
- Il programmatore può astrarre dai dettagli legati all'architettura ed esprimere i propri algoritmi in modo simbolico
- Sono indipendenti dalla macchina (astrazione)

# Linguaggi di II/III generazione



# Traduttore

Programma particolare che provvede a convertire il codice di programmi scritti in un dato linguaggio di programmazione (sorgenti), nella corrispondente rappresentazione in linguaggio macchina (eseguibili)

# Due tipi di traduttori (I)

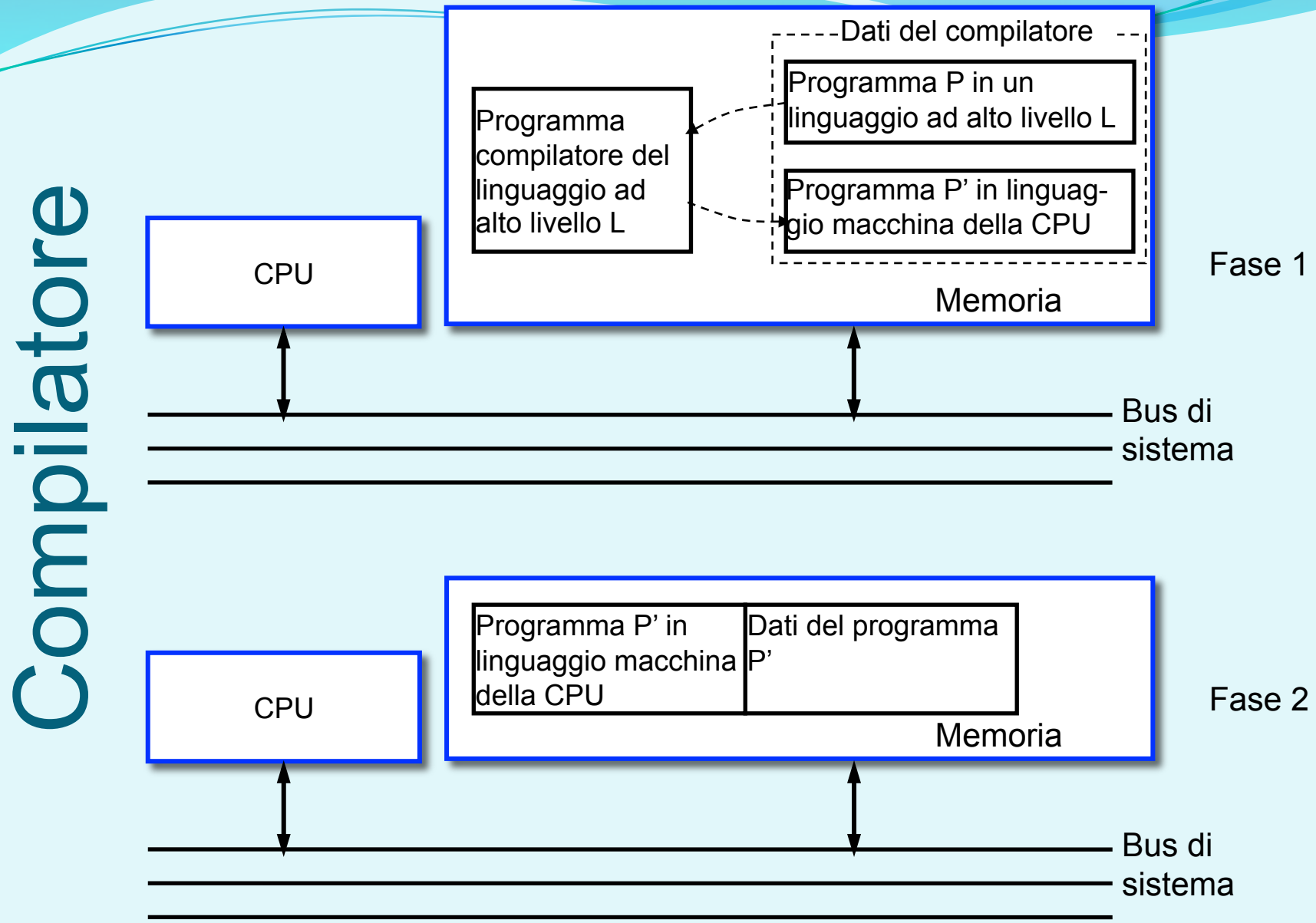
- Compilatori

  Accettano in ingresso l'intero programma e producono in uscita la rappresentazione dell'intero programma in linguaggio macchina

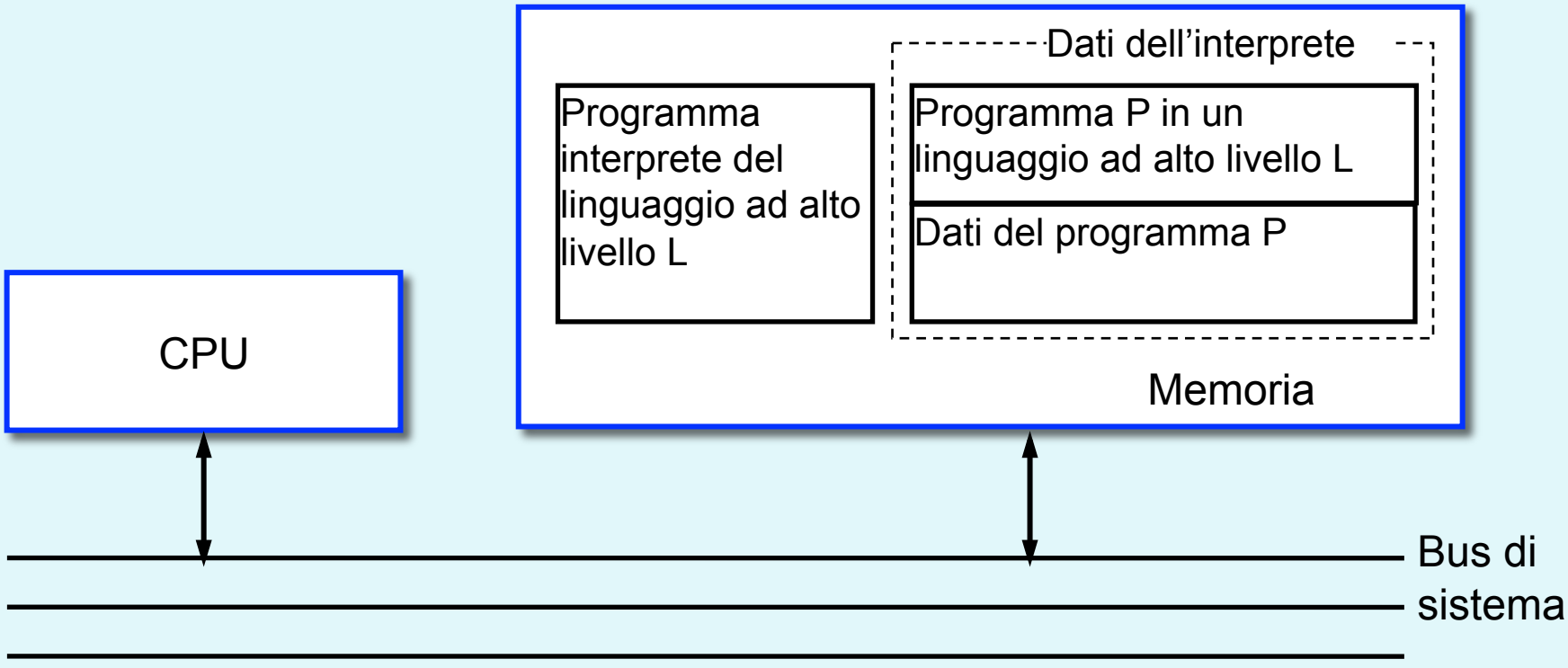
- Interpreti

  Traducono ed eseguono direttamente ciascuna istruzione del programma sorgente, istruzione per istruzione

# Compilatore



# Interprete


Dati dell'interprete
Programma interprete del linguaggio ad alto livello L
Programma P in un linguaggio ad alto livello L
Dati del programma P
Memoria
CPU
Bus di sistema

# Due tipi di traduttori II

- Compilatori

  Per ogni programma da tradurre, lo schema viene percorso una volta sola prima dell'esecuzione.

- Interpreti

  Lo schema viene attraversato tante volte quante sono le istruzioni che compongono il programma; ad ogni attivazione dell'interprete su una particolare istruzione, segue l'esecuzione dell'istruzione stessa.

NB: L'esecuzione di un programma compilato è più veloce dell'esecuzione di un programma interpretato

# Interprete vs compilatore

Quale delle due soluzioni è la migliore?

- Compilazione
  - applicazioni più veloci
  - maggior lavoro nel processo di messa a punto e manutenzione
  - OK per i prodotti commerciali a larga diffusione.
- Interpretazione
  - consente tempi di sviluppo più contenuti,
  - produce programmi meno efficienti;
  - OK in fase di prototipazione dei programmi che, una volta ultimati, venivano compilati prima del rilascio commerciale.

# Paradigmi di programmazione

- E’ possibile affrontare il problema della descrizione dei programmi in modi differenti
- Soluzioni differenti costituiscono *paradigmi di programmazione* differenti:
  - Imperativo-procedurale
  - A oggetti
  - Funzionale
  - Dichiarativo
  - Ecc.

# Paradigma imperativo/procedurale

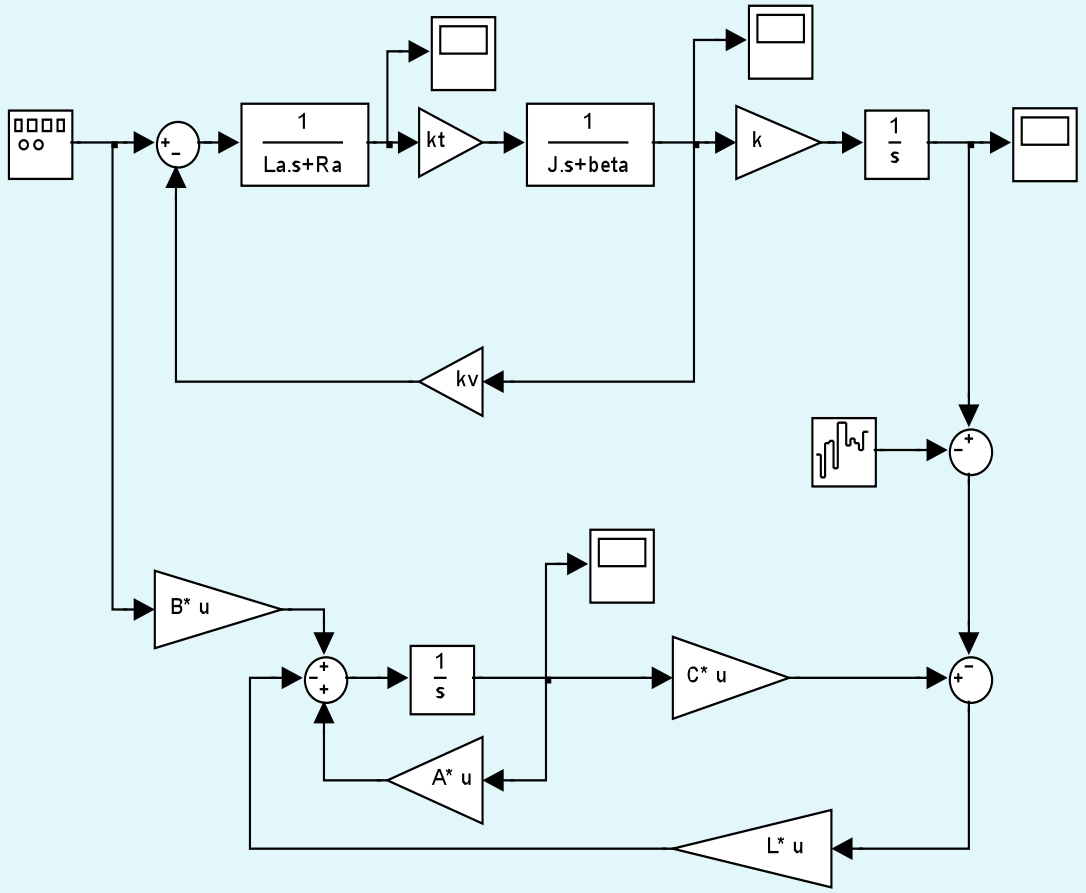
- Idea generale: descrivere al calcolatore i passi di risoluzione del problema
- Sottoprogrammi!
- Codifica un algoritmo in un linguaggio formale interpretabile dalla macchina
  - Passi elementari: indipendenti dal singolo calcolatore (astrazione)
  - Il linguaggio deve definire istruzioni e strutture dati
  - Passo di traduzione verso il linguaggio macchina
- Molto diffuso
  - Esempi: Fortran, Basic, Pascal, C (programmazione strutturata, evoluzione)

# Paradigma ad oggetti

- Idea: imitare la realtà
  - Il programma si definisce come sequenze di interazioni tra oggetti dotati di un comportamento
  - Gli oggetti reagiscono alle interazioni risolvendo la loro parte di problema
  - E' possibile usare oggetti senza conoscerne la struttura ed il funzionamento interni
  - Linguaggi Object Oriented e Object Based
- Molto diffuso (adatto per progetti ampi)
  - Simula, Smalltalk, C++, Object Pascal, Java…

# Paradigma funzionale

- Idea generale: unificare dati e istruzioni, considerando un programma come una serie di funzioni innestate
- Esempi
  - Lisp: un unico costrutto, detto *lista*
    - *print 12 + 23* diventa *(print (+ 12 23))*
  - LabView, Simulink: linguaggi grafici

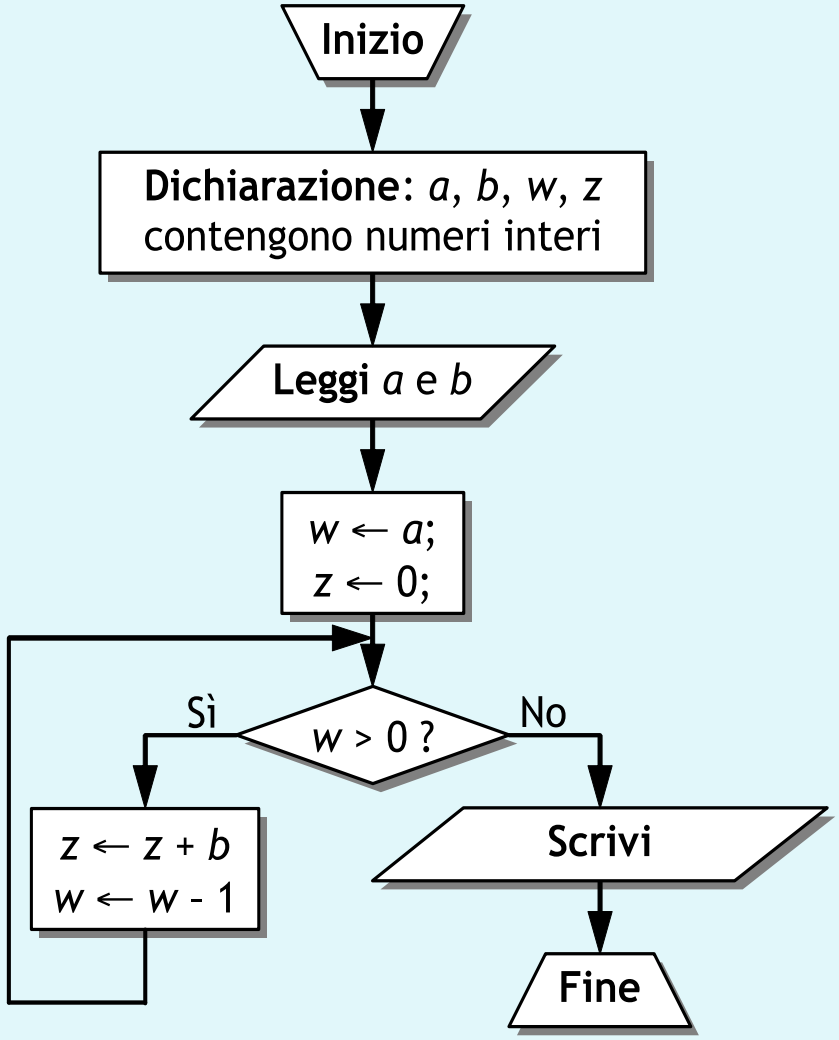


# Paradigma dichiarativo

- Idea: descrivere le caratteristiche della soluzione con *dichiarazioni* che le descrivono piuttosto che il procedimento per ottenerla
  - Il programma viene eseguito da un risolutore di problemi che determina la strategia migliore
  - Paradigma utilizzabile in campi specifici
- Esempi: Prolog, SQL

# Prodotto di due numeri naturali

```
Dati
   a, b interi
positivi
   w, z interi
Risoluzione
   leggi a e b
   z ← 0
   w ← a
   finché w > 0 ripeti
      z ← z + b
      w ← w – 1
   fine ciclo
   scrivi z
   fine
```



# Prodotto di due numeri naturali

## Algoritmo

**Dati**
a, b interi positivi
w, z interi
**Risoluzione**
leggi a e b
z ← 0
w ← a
finché w > 0 ripeti
  z ← z + b
  w ← w – 1
fine ciclo
scrivi z
fine

## Programma in C

```
main() { /* prodotto C
*/
  unsigned int a, b;
  int w, z;

  scanf("%d
%d",&a,&b);
  z = 0;
  w = a;
  while (w > 0) {
    z = z + b;
    w = w – 1;
  }
  printf("%d", z);
}
```

# Parti fondamentali di un programma

- *Identificazione* del programma
- *Dichiarazione* delle variabili utilizzate, di cui sono indicati tipo e nome
- Specificazione della parte *esecutiva* del programma, detta anche *corpo del programma*