

Mapping problems in computation

L'analisi del problema (astrazione, formalizzazione)

Data una richiesta "cercare consapevolmente delle azioni appropriate per raggiungere uno scopo chiaramente concepito, ma non immediatamente perseguibile" (Mathematical discovery, 1961)

a) determinare la somma dei numeri interi da 1 a 100

dati i numeri $a, b \in \mathbb{Z}$ (interi) e l'insieme $T = \{x \in \mathbb{Z} \wedge x \in [a, b]\}$ determinare la somma dei numeri appartenenti a T.

b) determinare lo spostamento di una massa lanciata alla velocità di 30 Km/h con un angolo di 25° rispetto al suolo in presenza di gravità

$v_0 = 30$ α (gradi) rispetto al suolo = 25



Algoritmo risolvete

Soluzione anche informale e indipendente dal calcolatore

a) Algoritmo iterativo:

somma = 0; per x che va da "a" a "b" ripeti somma = somma + x

Algoritmo di GAUSS

$$a+b = (a+1)+(b-1) = \dots (a+2)+(b-2)$$

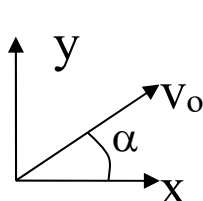
=> somma = $(a+b) \cdot (b-a+1) / 2$ (divisione tra reali per caso dispari)

Esempio: $a=1$ e $b=100$:

$$a+b = (a+1)+(b-1) = \dots = N/2 + (N/2+1) = 101$$

$$\Rightarrow \text{somma} = 101 \cdot 50$$

b)


$$\begin{aligned} v_y(t) &= v_0 \cdot \sin(\alpha) - g \cdot t & v_x(t) &= v_0 \cdot \cos(\alpha) \\ x(t) &= v_x(t) \cdot t & y(t) &= v_{oy} \cdot t - \frac{1}{2} g \cdot t^2 \end{aligned}$$

con unità di misura congruenti (non esistono in C).

L'algoritmo implementato (il programma)

1. scelta della tecnologia adatta

- identificazione di un ESECUTORE dell'algoritmo e del suo linguaggio (macchina astratta C)

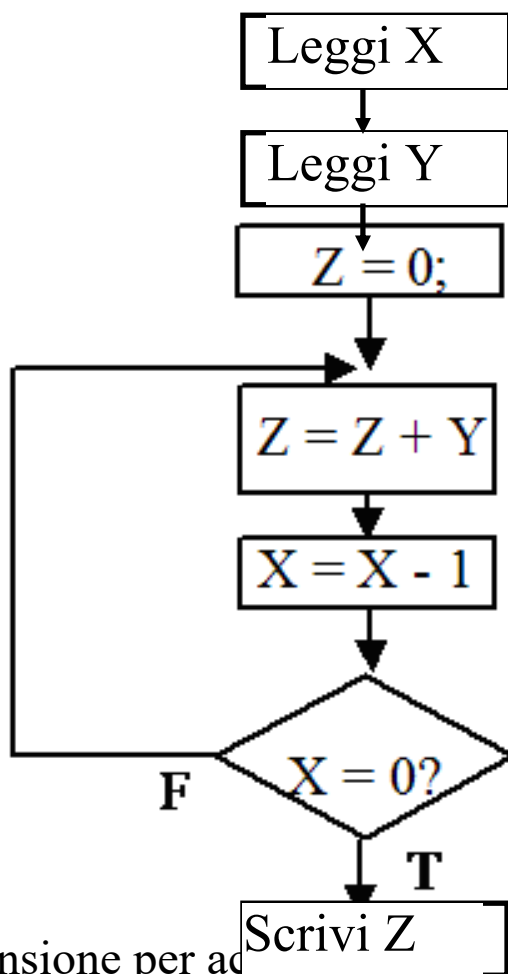
2. Implementazione imperativa dell'algoritmo risolvete: scrivere un insieme di azioni nel linguaggio che realizzano l'algoritmo (how to do) e rispettano 3 vincoli:

- ogni azione non è ambigua per l'esecutore;
- ogni azione è eseguibile dall'esecutore in un tempo finito;
- l'esecuzione ordinata delle azioni termina dopo un numero finito di passi (programmatore).

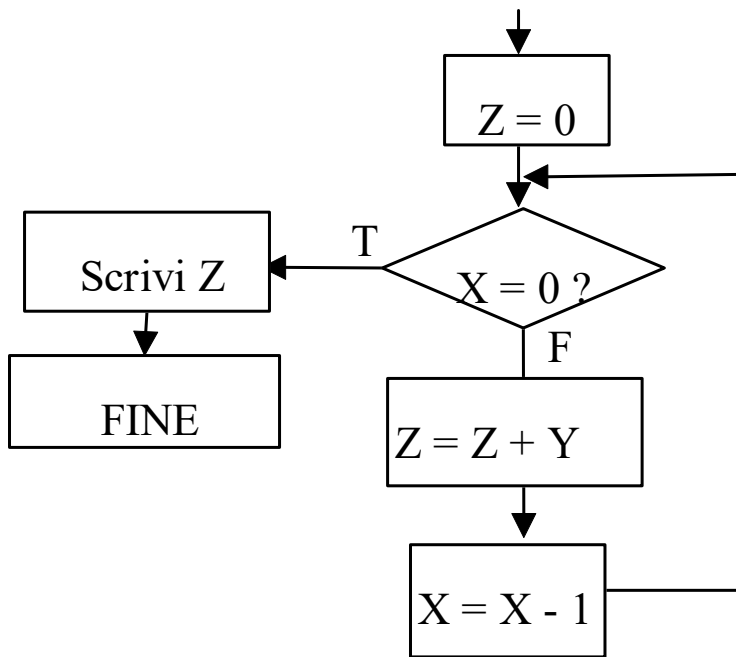
Problema:

Dati due numeri naturali X e Y eseguire la moltiplicazione $Z=X*Y$ con $X>0$ e $Y\in\mathbb{Z}$. Limite su X non controllato.

Linguaggio esecutore: diagrammi a blocchi



Estensione per ad Scrivi Z X=0



Ulteriore estensione per $X \in Z$.

Osservazioni:

- esistono più algoritmi risolvanti per un problema
- esistono più implementazioni per uno stesso algoritmo risolvante
 - stile – art of programming
 - azioni e strutture dati del linguaggio – vedremo poi
- l'esecutore non verifica la correttezza di un'implementazione



programma funzionante non implica programma corretto



Importanza testing dei programmi – engineering methodology
(α e β test, bugs report...)

Il calcolatore come esecutore di algoritmi memorizzati

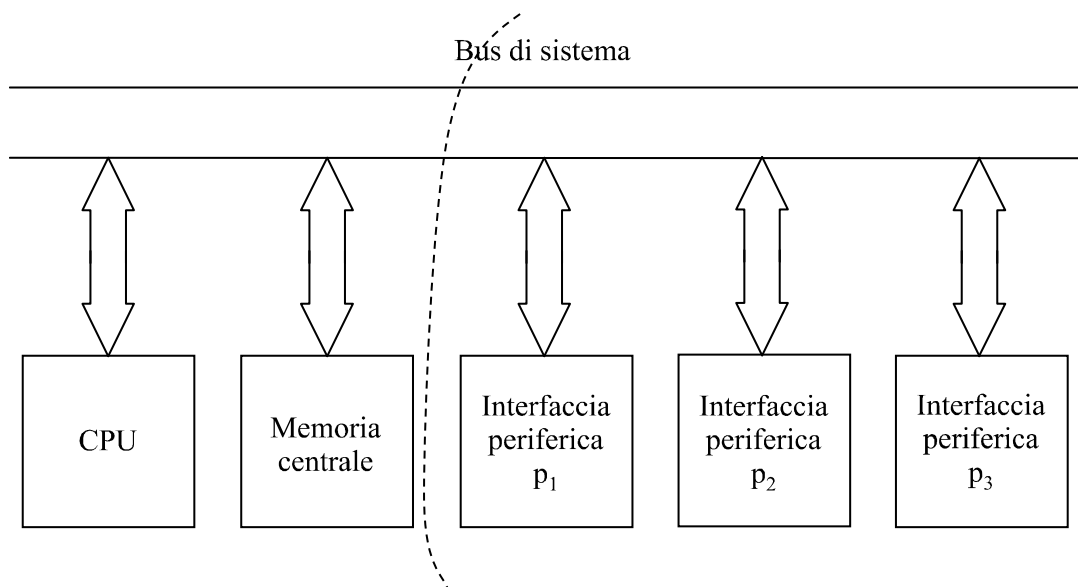
Dal calcolatore dedicato ad un'applicazione (calcolatrice)

Al calcolatore che:

- esegue programmi che vengono memorizzati
- programmi costruiti con un numero fisso di istruzioni primitive

1940-52 Calcolatore elettromeccanico MARK I (IBM). Calcolatore elettronico ENIAC(Univ. Pennsylvania-1946) a valvole, lung. 30 mt., 150KW, programma interno cablato nell'hardware (caricamento lento), bassa affidabilità. IAS machine di Von Neumann-Univ. Princeton (1KRAM, t.a. 25 μ s.), Whirlwind-Bell (2KRAM nuclei magnetici t.a. 25 μ s.)...

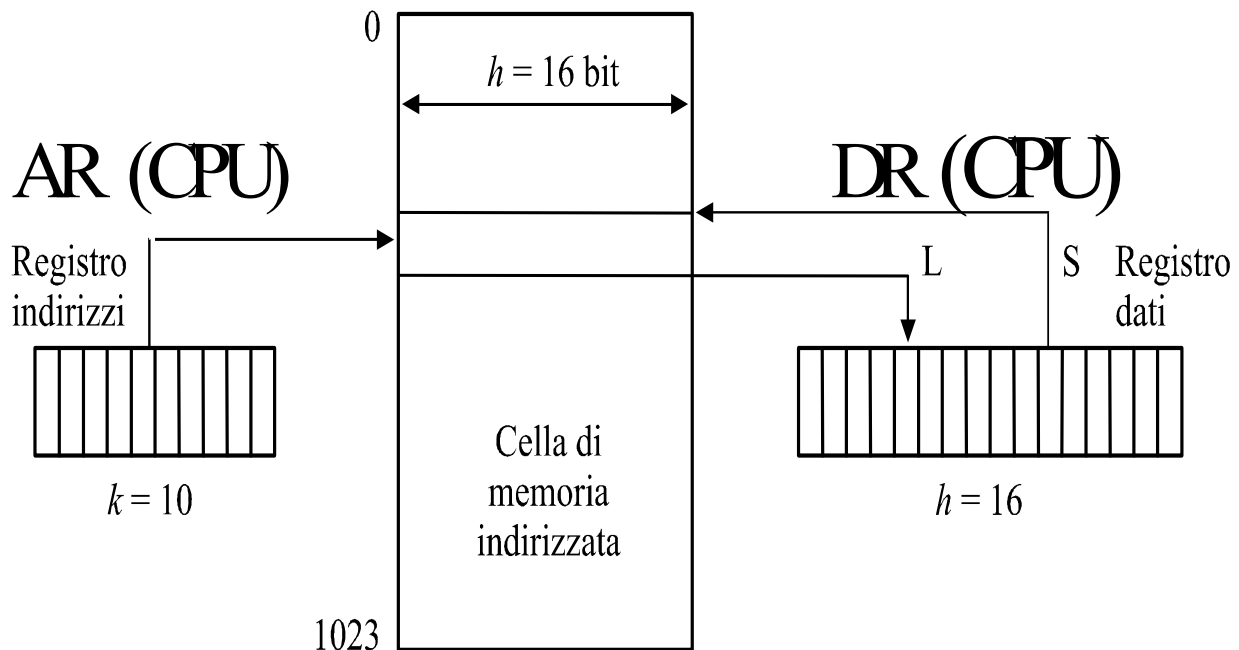
La macchina astratta di Von Neumann – 1946



CPU: unità aritmetico/logiche e di controllo, PC

Memoria centrale

Tecnologia elettronica a transistor (t.a. $\cong 9 - 11$ nS)



Memoria da 1024 parole da 16 bit ciascuna

Ogni parola ha un indirizzo: da 0 a 1023 \Rightarrow numero di bit del registro indirizzi (AR) = $\log_2 1024$

Ogni parola deve essere trasferita nella CPU per utilizzarne o modificarne il contenuto tramite il registro dati (DR).

Bus di sistema diviso in

- bus dati trasmette valori tra il registro dati (DR) della CPU e la memoria centrale/periferiche
- bus indirizzi trasmette indirizzi del registro indirizzi (AR) della CPU per individuare il componente (memoria centrale/periferica) e lo specifico sottocomponente (parola di memoria/ registro)
- bus controlli trasferisce informazione di controllo tra CPU e altri componenti

Dal linguaggio a blocchi al linguaggio assembly

Istruzioni:

[W:] ADD X,Y : somma in Y

[W:] ADD #X, Y : somma in Y

[W:] MOV X, Y: copia in Y

[W:] MOV #X, Y: copia in Y

[W:] BR I: passa ad eseguire l'istruzione in parola con etichetta I

[W:] BREQ X, I: se contenuto di X =0 allora SALTA a istruzione I

[W:] EXIT termina esecuzione programma

[W:] READ X: carica valore letto da tastiera in X

[W:] WRITE X: scrive su video valore in X

dove #X identifica il valore del simbolo X e X il valore contenuto nella parola di memoria con etichetta X

Direttive (pseudoistruzioni)

[X:] RES N Alloca N parole consecutive in memoria (RAM) e associa l'indirizzo simbolico X(etichetta) alla prima parola

END [X] indica la fine del programma e la posizione della prima istruzione da eseguire

Implementazione (chiamato PROGRAMMA) della moltiplicazione (linearizzazione schema a blocchi)

etichette direttive/istruzioni

X: RES 1

Y: RES 1

Z: RES 1

START: READ X

READ Y

MOV #0, Z

CICLO: BREQ X, FINE

ADD Y, Z

ADD #-1, X

BR CICLO

FINE: WRITE Z

EXIT

END START (prima istruzione eseguibile)

Caricamento del programma in memoria centrale e sua esecuzione

Es. Caricamento programma dall'indirizzo 0

Etichetta	indirizzo		
X	0	0000	
Y	1	0001	
Z	2	0010	
START	3	0011	READ X
	4	0100	READ Y
	5	0101	MOV #0, Z
CICLO	6	0110	BREQ X, FINE
	7	0111	ADD Y, Z
	8	1000	ADD #-1, X
FINE	9	1001	BR CICLO
	10	1010	WRITE Z
	11	1011	EXIT
	12	1100	
	13	1101	
	14	1110

MOV #START, PC (3)

Esecuzione del programma

Esecuzione moltiplicazione con Y=4 e X=2

indirizzo corrente	Istruzione	PC dopo ese- cuz. Istruz.	X	Y	Z
			?	?	?
3	READ X	4	2	?	?
4	READ Y	5	2	4	?
5	MOV #0, Z	6	2	4	0
6	BEQ X, FINE	7	2	4	0
7	ADD Y, Z	8	2	4	4
8	ADD #-1, X	9	1	4	4
9	BR CICLO	6	1	4	4
6	BEQ X, FINE	7	1	4	4
7	ADD Y, Z	8	1	4	8
8	ADD #-1, X	9	0	4	8
9	BR CICLO	6	0	4	8
6	BEQ X, FINE	10	0	4	8
10	WRITE Z	11	0	4	8
11	EXIT	?			

Dal linguaggio simbolico al linguaggio binario della CPU

(simulazione assemblatore– collegatore/linker)

Convenzione imposta dalla CPU:
Numeri: 16 bit complemento a 2.

Codifica istruzioni su 16 bit

Codice operativo	mod. ind. operando 1	Ind. operando 1	mod. ind. operando 2	mod. ind. Operando 2
4	2	4	2	4

Istruzione	C.O.
MOV	1000
ADD	0110
READ	1111
WRITE	0111
BR	1001
BREQ	1010
EXIT	1011
JSR	...
RTS	...

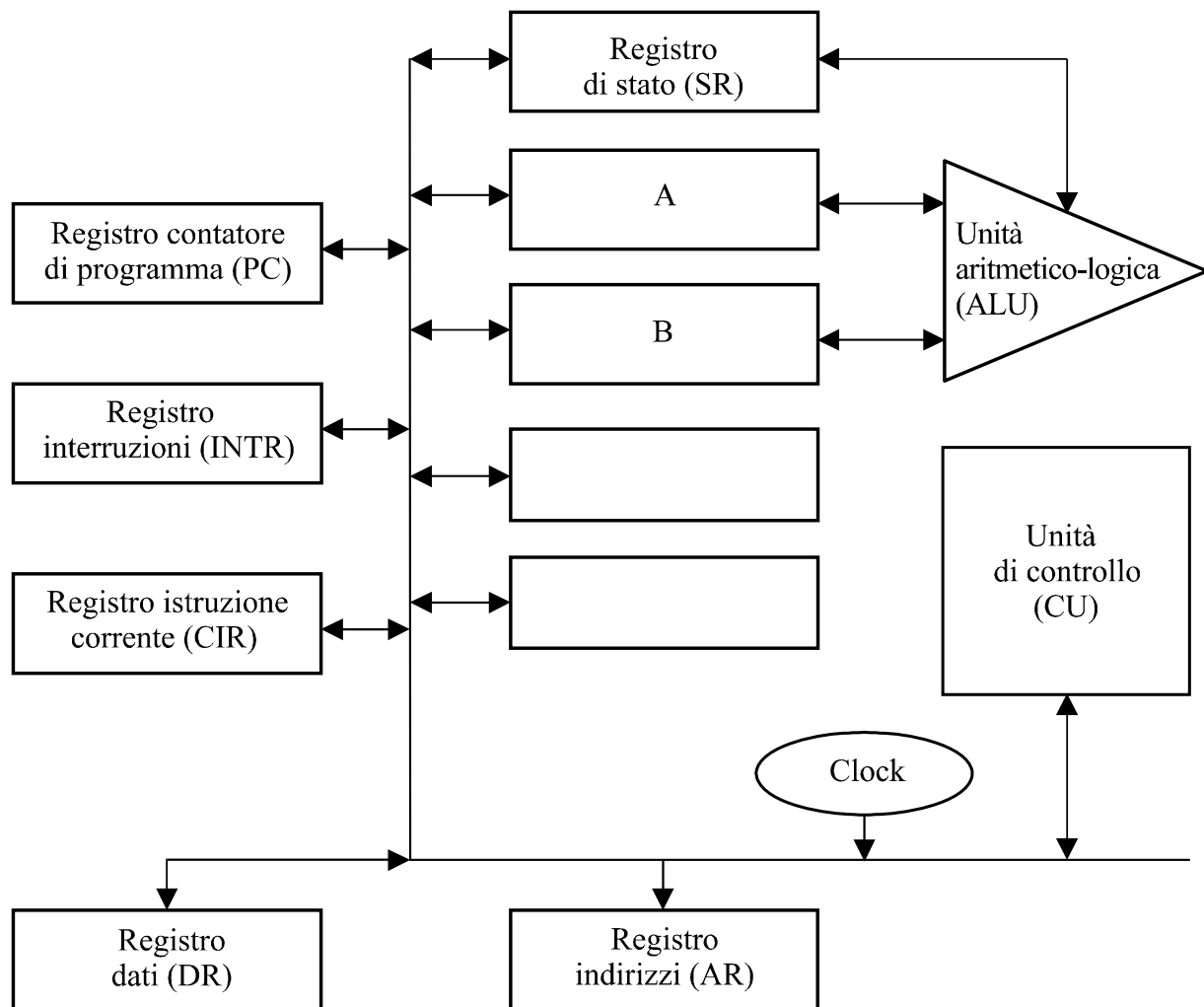
Mod. indirizzamento

X	00
#X	01

Programma in linguaggio binario (caricamento in memoria a partire dall'indirizzo 0 - PC = 0011)

0	0000	?	?	?
1	0001	?	?	?
2	0010	?	?	?
3	0011	1111	00 0000	?
4	0100	1111	00 0001	?
5	0101	1000	01 0000	00 0010
6	0110	1010	00 0000	00 1010
7	0111	0110	00 0001	00 0010
8	1000	0110	01 1111	00 0000
9	1001	1001	00 0110	?
10	1010	0111	00 0010	?
11	1011	1011	?	?
12	1100	?	?	?

L'esecuzione del programma (come lavora la CPU)



- **unità di controllo:** prelievo e decodifica istruzioni e invio dei segnali di controllo
- **unità aritmetico-logica**
- **CIR contiene istruzione in esecuzione**
- **PC, Program Counter:** indirizzo della prossima istruzione da eseguire;
- **INTR** informazioni sul funzionamento delle periferiche
- **registri A, B, ...** operandi risultato delle elaborazioni dell'ALU e per velocizzare operazioni;
- **SR, Status Register** ogni bit indica condizione sul risultato delle operazioni della ALU (segno, overflow, riporto, zero)
- **Clock** sincronizzazione operazioni (cicli/sec – Hz): $400\text{MHz} \rightarrow 400 * 10^6\text{Hz} \rightarrow 1 \text{ ciclo eseguito in } 1/(400 * 10^6\text{Hz}) = 2,5\text{ns}$

Il microprogramma (ROM della CPU)

1953 Wilkes ha l'idea della μ programmazione \Rightarrow 1964 IBM 360
While (calcolatore è acceso)

{1. (*fase di fetch*)

- a. AR = PC;
- b. lettura parola di memoria con indirizzo in AR
 - AR \Rightarrow bus indirizzi \Rightarrow memoria centrale
 - comando L \Rightarrow bus controlli \Rightarrow memoria centrale
 - exec L \Rightarrow bus dati \Rightarrow parola memoria in DR
 - OK exec \Rightarrow bus controlli \Rightarrow CPU
- c. CIR = DR

2./*decodifica codice operativo*/ (analisi primi 4 bit di CIR)

3. caso 0110(ADD): {/*esecuzione istruzione*/

- a. caso (mod. ind. operando1 (bit 5,6 CIR)=00)
 - a1. AR= ind. Operando 1 (bit 7 \Leftrightarrow 10)
 - a2 Lettura parola di memoria (passo b. fetch)
 - a3 Registro A= DR
- else ...
- b. caso (mod. ind. Operando2 (bit11,12 CIR)=00)
 - b1 AR= ind. Operando 2 (bit 13 \Leftrightarrow 16)
 - b2 Lettura parola di memoria (passo b. fetch)
 - b3 Registro B = DR
- else ...
- c. Registro B = sommatore ALU (A, B)
- d. AR \Rightarrow bus indirizzi \Rightarrow memoria centrale
- e. DR = registro B
- f. DR \Rightarrow bus dati \Rightarrow memoria centrale
- g. comando S \Rightarrow bus controlli \Rightarrow memoria centrale \Rightarrow exec S
- h. OK esecuzione \Rightarrow bus controlli \Rightarrow CPU
- i. PC=PC + 1

caso 1001(BR):

caso (mod. ind. Operando1 (bit5,6 CIR)=00)

PC = ind. Operando 1 (bit 7 \Leftrightarrow 10)


else ...

}/*while*/

Dalla correttezza all'efficienza

- tempo esecuzione singola μ istruzione = $1/f$ (frequenza di clock - 400MHz)
- tempo esecuzione singola istruzione = $n^\circ \mu$ istruz. richieste/ f (pipeline, coprocessori)
- $n^\circ \mu$ istruz. richieste \leftarrow ALU specializzate
- tempo di esecuzione di un programma = $f(n^\circ \text{ istruzioni, flusso eseguito, compilatore, architettura hdw/soft del sistema - multiprocessori, carico del sistema, colli bottiglia,...})$

benchmarks (trans. per sec./ SPEC 95 (int/real)/MIPS)



Valutazione della complessità computazionale (polinomiale o NP) con riferimento ad una risorsa da misurare (tempo CPU, RAM, disk ...)

Nel corso di fondamenti di informatica non sono considerate le prestazioni degli algoritmi (tranne analisi casi limite)

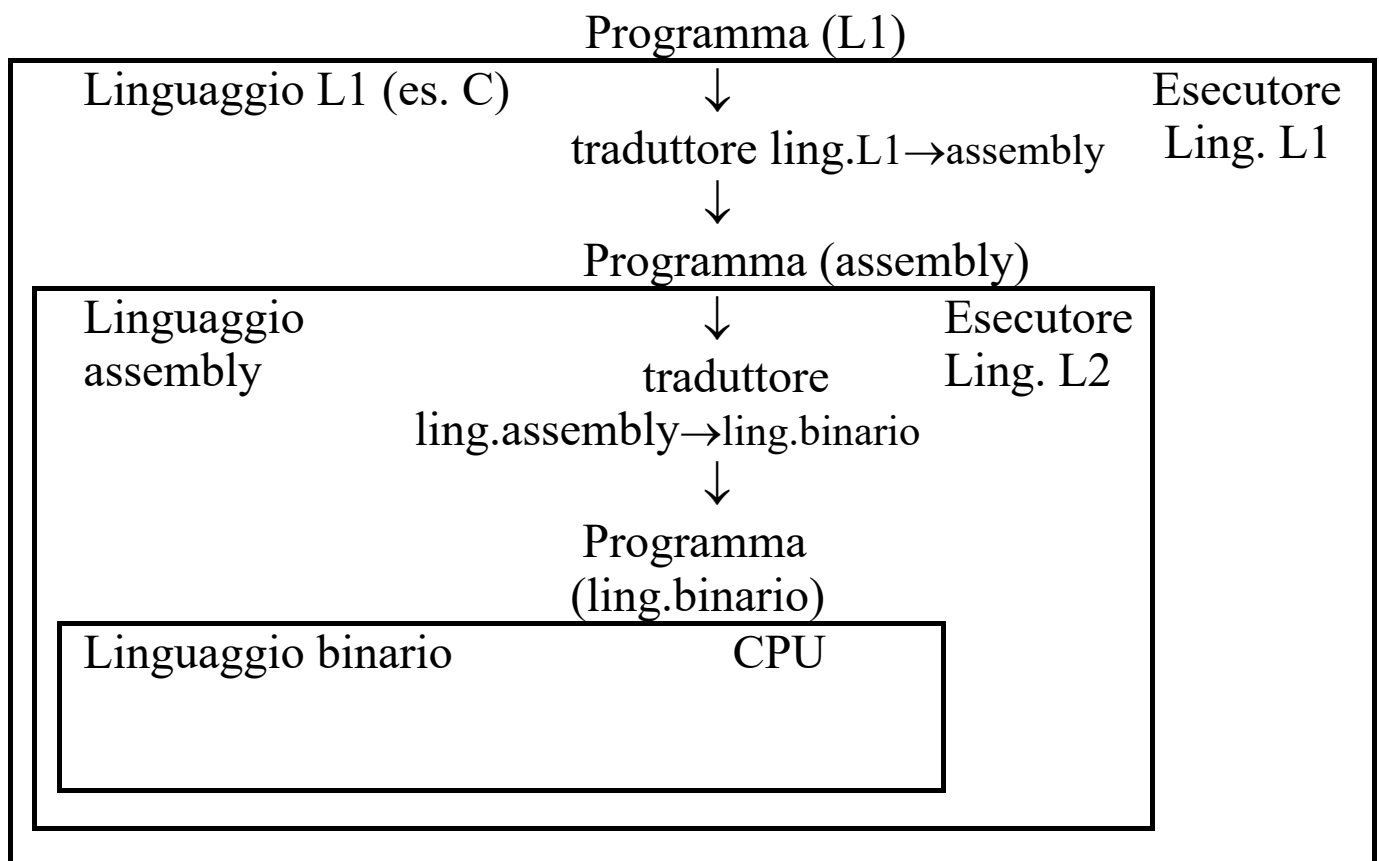
L'algoritmo della moltiplicazione per l'esecutore C (macchina astratta che comprende il linguaggio C)

```
#include <stdio.h>
int X, Y, Z;
main()
{ scanf ("%d", &X); scanf ("%d", &Y);    Z = 0;
  while (X > 0)
    {Z=Z+Y;  X=X-1; }
  printf("%d",Z);
}
```

o forse meglio

```
#include <stdio.h>
int X, Y, Z;
main()
{ scanf ("%d", &X); scanf ("%d", &Y);
  Z = X*Y;
  printf("%d",Z);
}
```

La “Matrioška” delle macchine astratte



Costruire macchine astratte con diverso potere espressivo

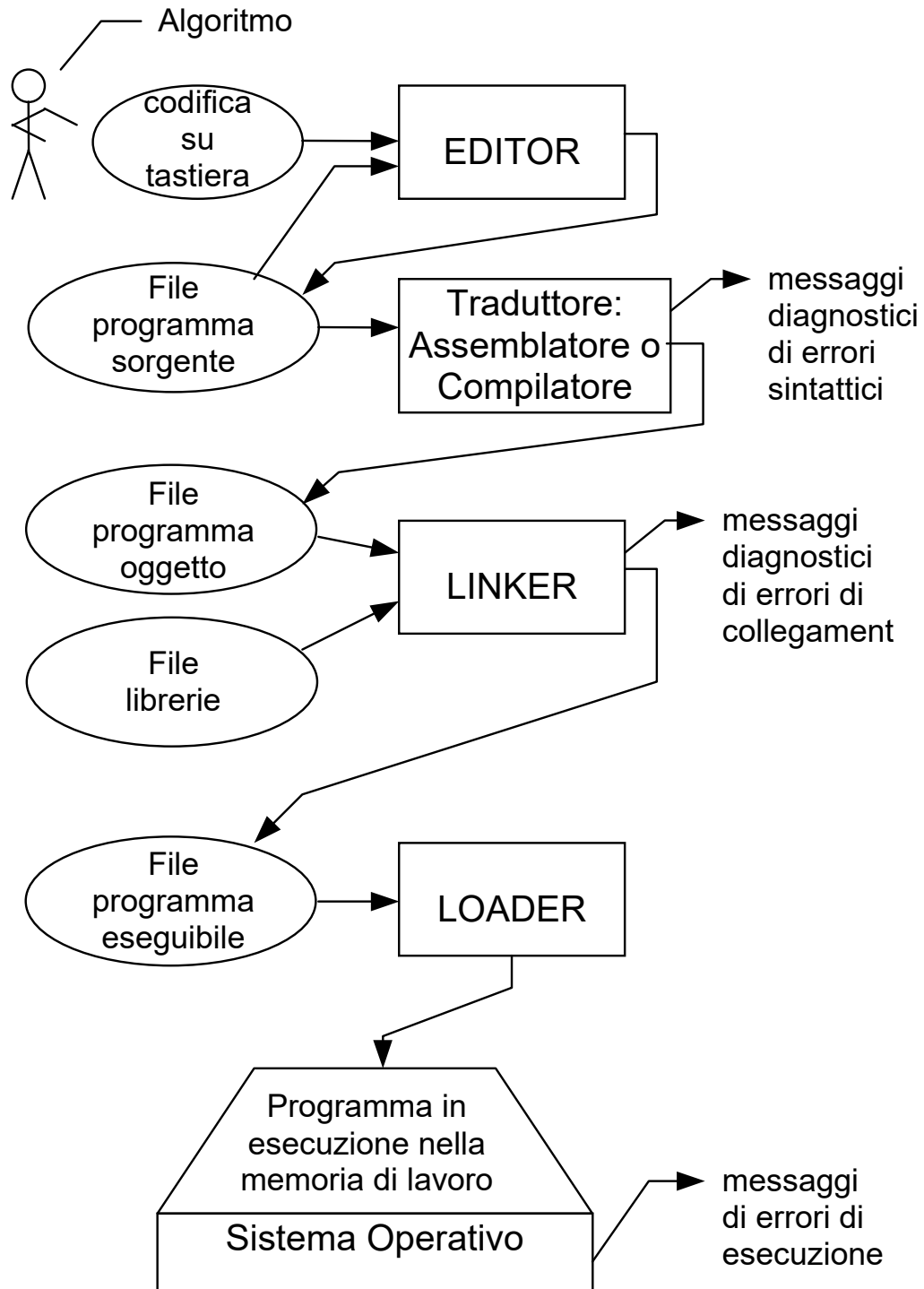
- Per avere strutture dati + adatte ad un contesto disciplinare.
- Per avere istruzioni più potenti per operare sui dati creati
- Per avere nuovi meccanismi per il controllo dell'esecuzione



Maggior facilità nella scrittura dei programmi
Maggior efficacia ed efficienza in manutenzione
Minor efficienza in esecuzione?

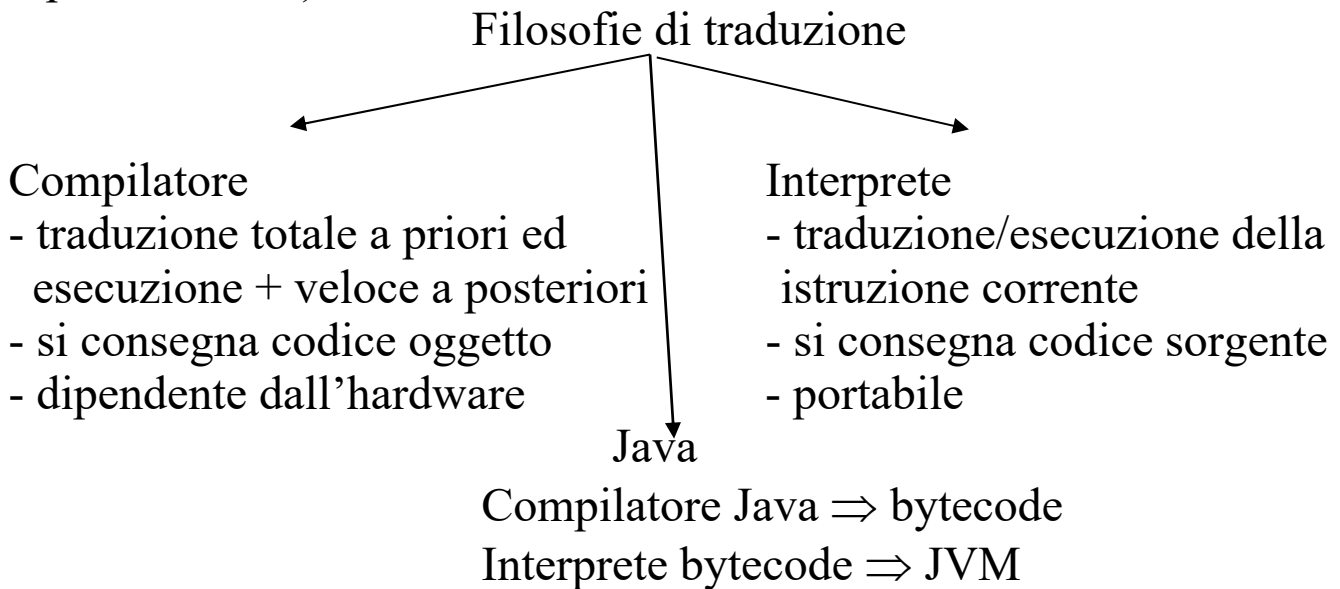
AMBIENTE DI SVILUPPO DI UN PROGRAMMA

IDE (Integrated Desktop Environment) x la creazione di un eseguibile



Pre (compilatore): \Rightarrow programma in ling. Binario (codice oggetto)

- analisi lessicale (simboli dell'alfabeto: while, identificatori), analisi grammaticale (rispetto regole) e analisi contestuale (identificatori non definiti);
- segnala eventuali errori sintattici (variabili non dichiarate, ma usate; parole chiave usate come variabili, conflitti nell'uso dei tipi o nei parametri, ...)



Linker: codice oggetto programma e librerie \Rightarrow eseguibile

- segnala eventuali corrispondenze non trovate

Loader (S.O.): carica l'eseguibile in memoria e ne attiva l'esecuzione.

Esecuzione e testing : segnala errori in esecuzione (*run-time*):

- Calcoli con risultati scorretti \Rightarrow Ad es. Overflow.
Calcoli impossibili \Rightarrow Divisione per zero, logaritmo di numero negativo, radice quadrata di un negativo, ecc.
- Errori logici intercettati in fase α/β test o dall'utente
 - assenza di controllo sui valori di input ammissibili (salario negativo);
 - possibili sequenze di input non considerate nell'analisi;
 - algoritmo (risolvente o implementato) sbagliato (es. moltiplicazione per valori di $X \geq 0$).