Dipartimento di Informatica e Sistemistica
Antonio Ruberti

“Sapienza” Università di Roma

Sistemi di Elaborazione

Corso di Fondamenti di Informatica

Laurea in Ingegneria Informatica

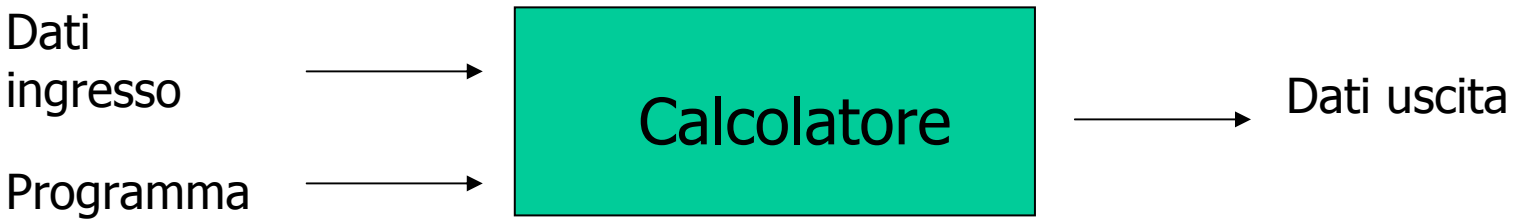
(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)

Anno Accademico 2007/2008

Prof. Paolo Romano

Si ringrazia il Prof. Alberto Finzi per aver reso
disponibile il materiale didattico sul quale si basano queste slides

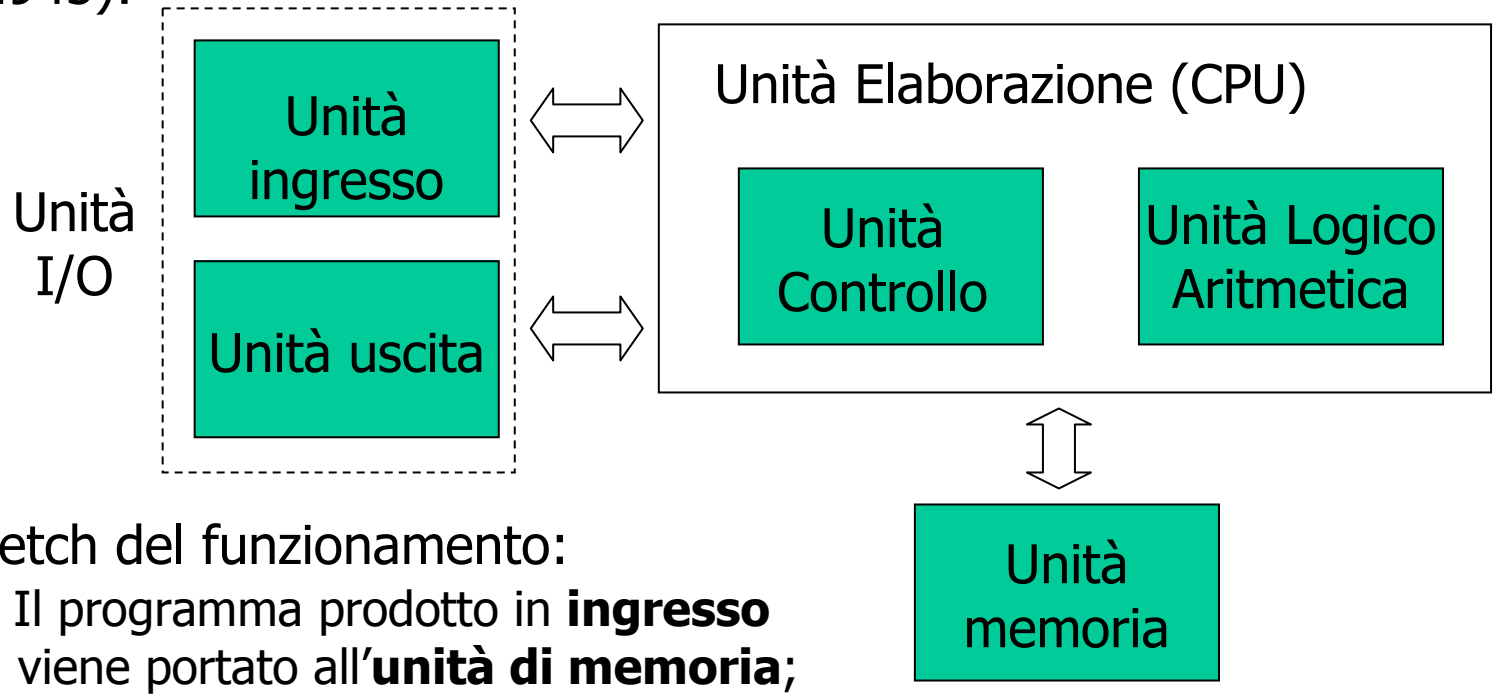
Struttura della Macchina Calcolatrice



Funzionalità richieste:	Componenti richieste:
1. Ingresso dei dati/programmi	1. Unità di ingresso (es. tastiera)
2. Uscita dei dati	2. Unità di uscita (es. video)
3. Elaborazione dei dati:	3. Unità di elaborazione (processore)
4. Memoria dei dati e programmi	4. Unità di memoria
3. Elaborazione dei dati:	3. Unità di Elaborazione:
3.1. Esecuzione operazioni	3.1. Unità Aritmetico-Logica (ALU)
3.2. Controllo dell'esecuzione	3.2. Unità di Controllo (CU)

Architettura di von Neumann

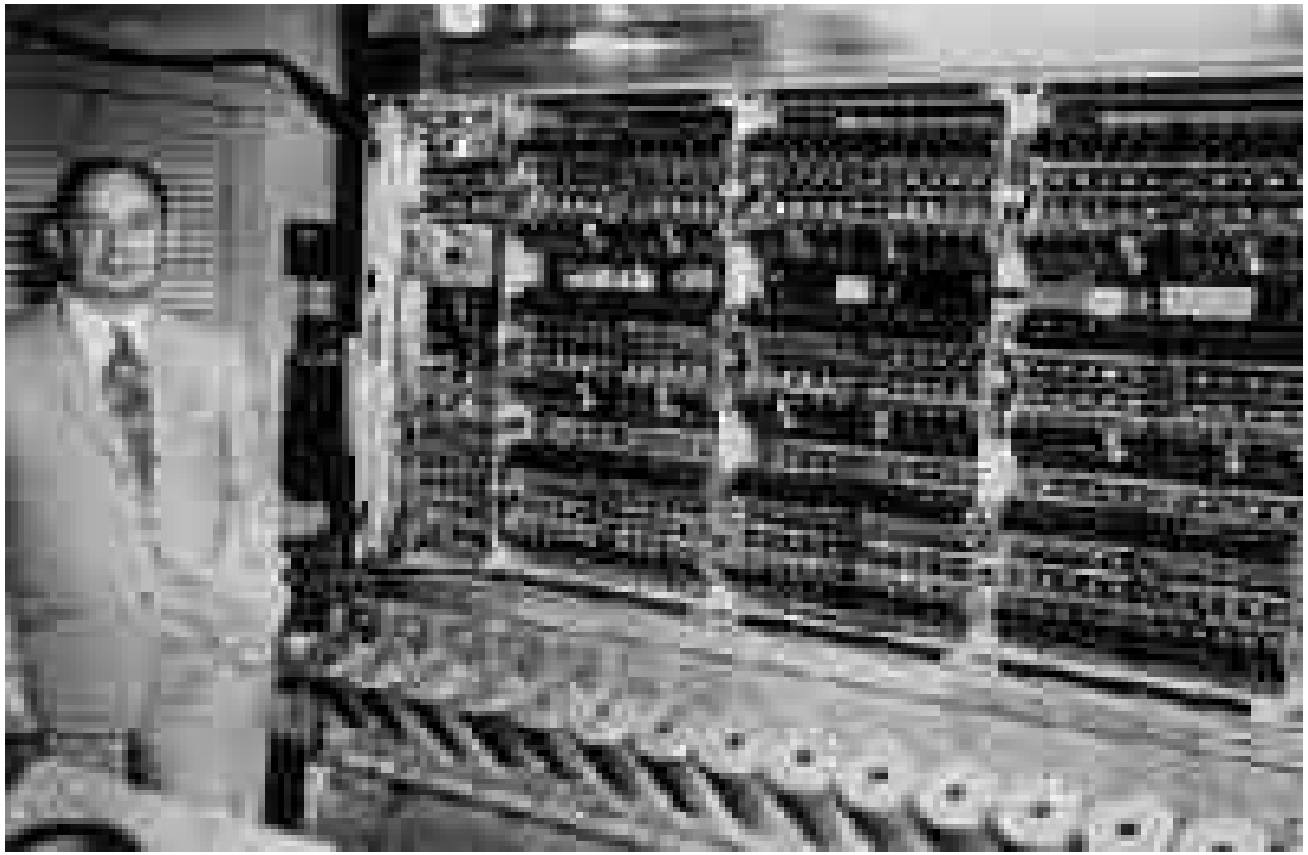
Le precedenti unità funzionali compongono l'architettura di Von Neumann (1945):



Sketch del funzionamento:

1. Il programma prodotto in **ingresso** viene portato all'**unità di memoria**;
2. un'istruzione alla volta viene prelevata dalla **memoria** mandata in esecuzione (operazioni logico-aritmetiche alla **ALU)**.

Von Neumann con l'EDVAC (1952)



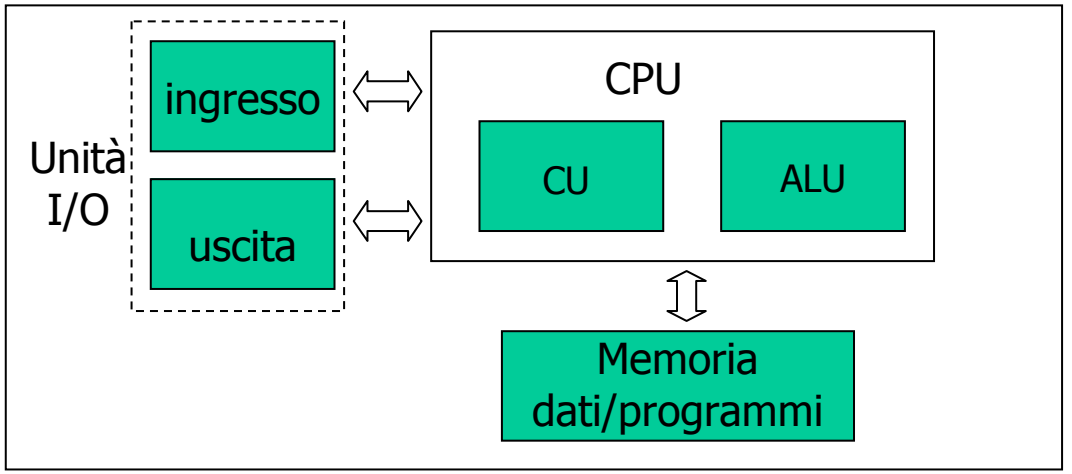
"First Draft of a report to the EDVAC", 1945
Nel quale viene presentata l'architettura di Von Neumann

Punto principale: programma in memoria trattato come dato (stored-program computer), può essere modificato dalla macchina (ma già Mauchly, Eckert e Babbage!)

Macchina von Neumann (Princeton) vs. Macchina Harvard

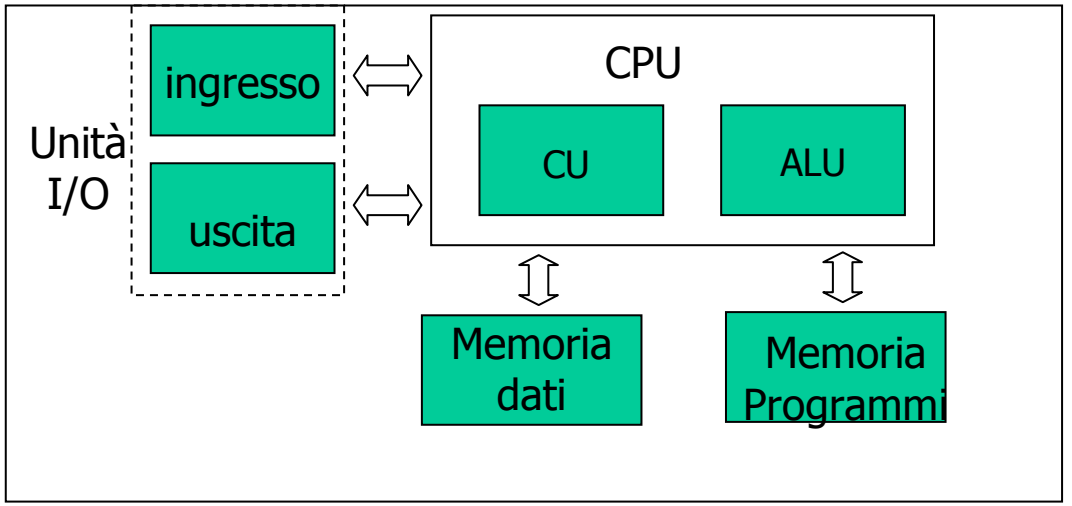
Von Neumann (1945):
unica memoria per dati & programmi

(EDVAC)
Programma in memoria, modificabile

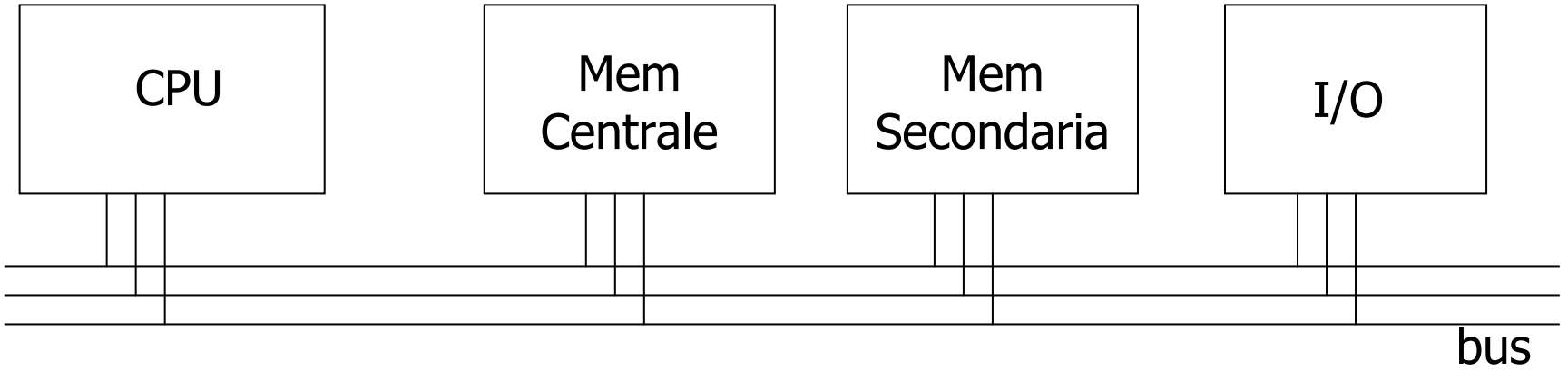


Harvard (1944):
memoria separata per dati & programmi

(Harvard Mark I)
Programma su nastro, non modificabile



Architettura Calcolatore



CPU: CU**,** ALU, Registri

I/O: input, output

Memoria Principale: memoria di lavoro ad accesso rapido

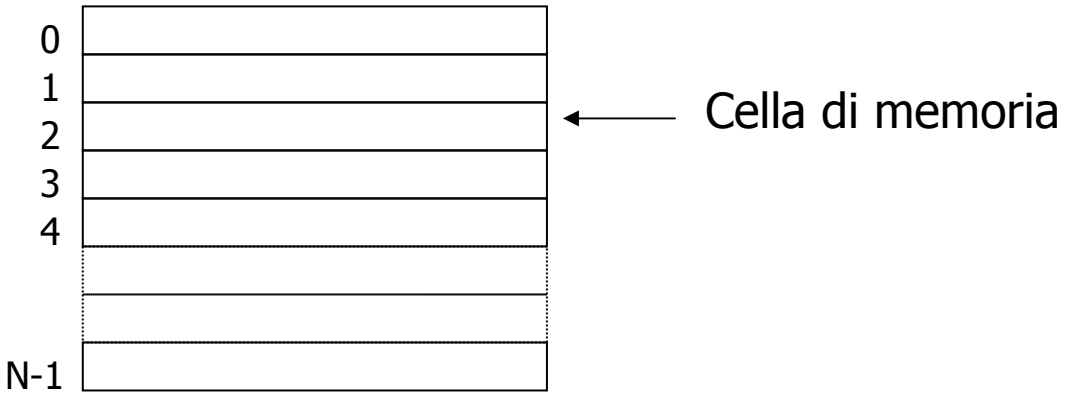
Memoria secondaria: contiene informazioni a lungo termine (*memoria non volatile*, conserva informazione anche dopo lo spegnimento del computer, es. Disco Rigido, CDRom, floppy disk)

Bus: comunicazione tra le unita' (dati/indirizzi/comandi)

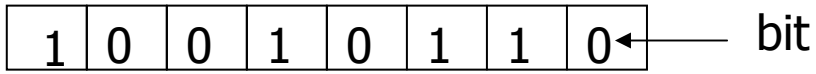
Memoria Centrale

In memoria centrale (RAM) vengono rappresentati dati e programmi.

La memoria è una sequenza di **celle di memoria** (dette ***parole***) tutte della medesima dimensione:



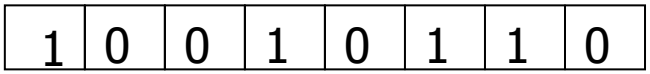
Ogni cella è una sequenza di **bit** (*binary digit*)



La lunghezza della parola dipende dalla macchina ed è un multiplo di **8 bit (byte)**: 8, 16, 32, 64 …

Le celle sono numerate da 0 a 2^n. Il numero che identifica univocamente la cella è il suo **indirizzo.**

671



Rappresentazione dell'Informazione

Dato (numero, carattere etc.) ⇨ Una o piu' celle di memoria (parola) ⇨ Dato codificato come sequenza di bit

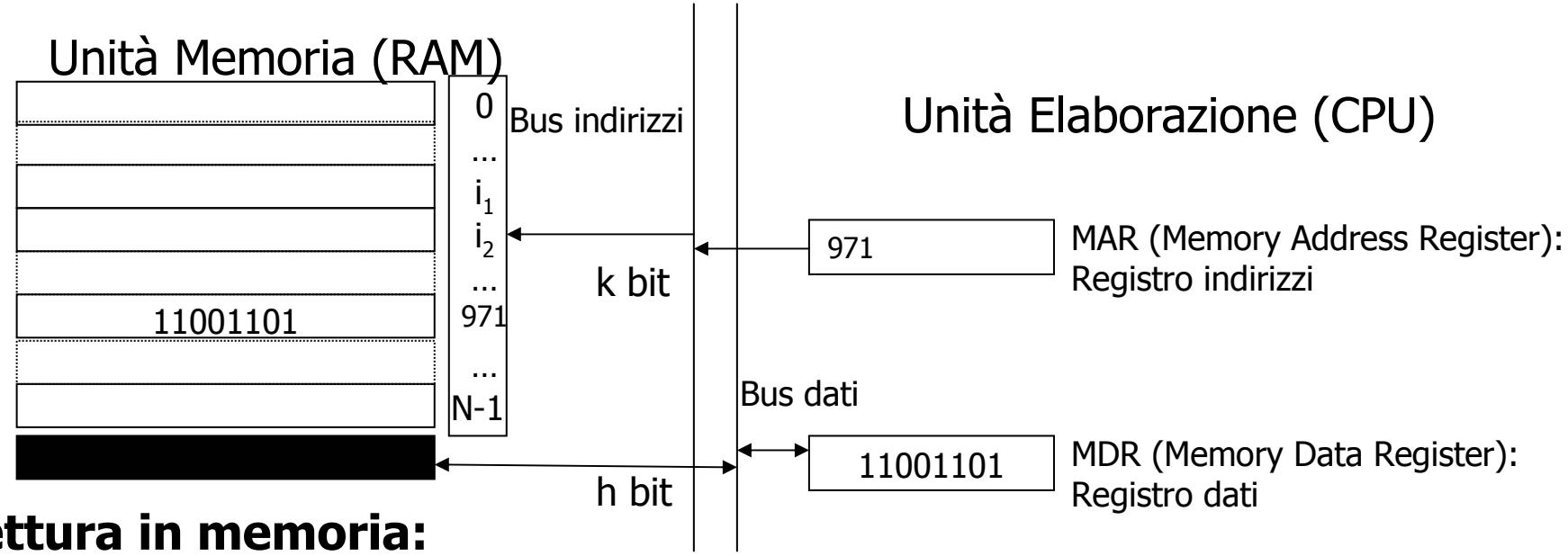
Sequenze di 1 bit: [1],[0]
Sequenze di 2 bit: [0,0], [0,1],[1,0],[1,1]
...
Sequenze di n bit: 2^n ennuple $[b_1, b_2, \ldots, b_n]$
Quanti dati con 8 bit (byte)?

Esempio: 2^8 numeri (es. da 0 a $2^8 - 1$); 2^8 caratteri (es. a = [0,0,1,1,0,1,0,1]);
2^8 indirizzi memoria, etc.

Dimensione memoria: misurata in byte

Kilobyte	$2^{10} \sim 10^3$ byte
Megabyte	$2^{20} \sim 10^6$ byte
Gigabyte	$2^{30} \sim 10^9$ byte
Terabyte	$2^{40} \sim 10^{12}$ byte

Accesso in Memoria Centrale



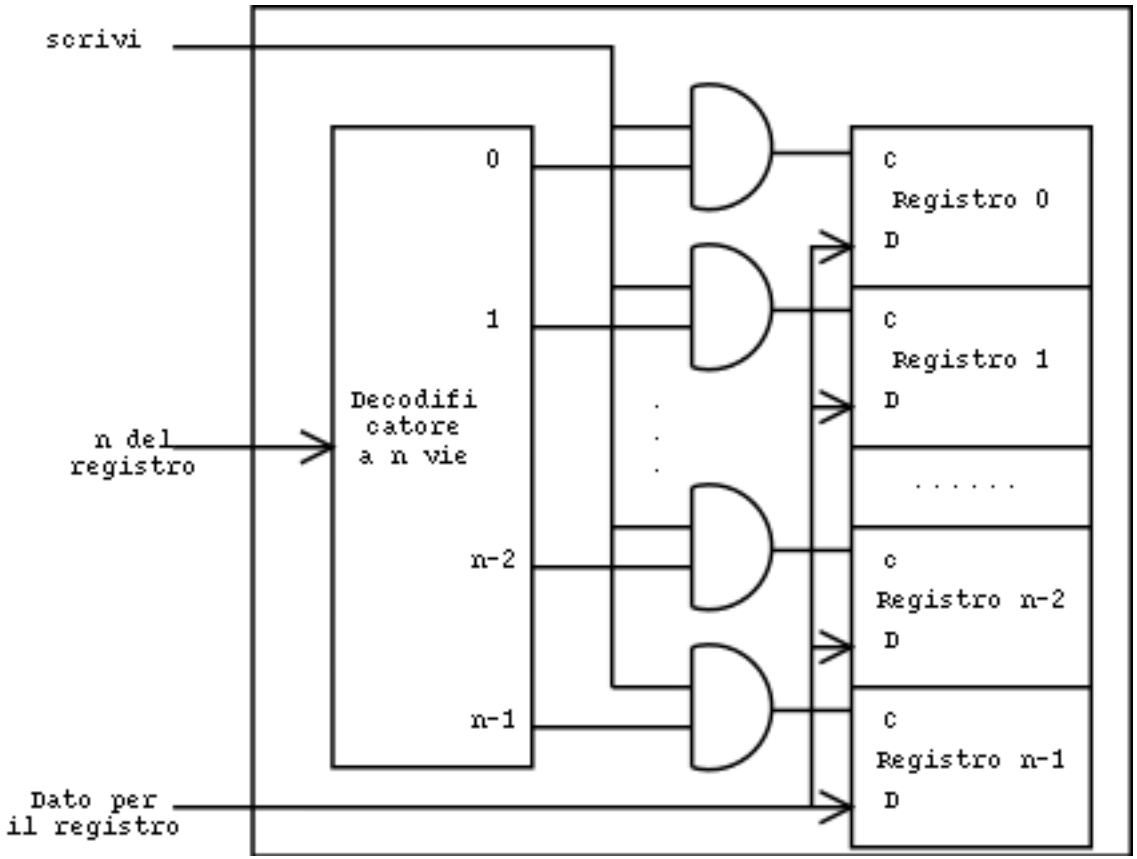
Lettura in memoria:

1. CPU scrive indirizzo di memoria nel registro indirizzi;
2. Operazione di accesso;
3. Valore della cella disponibile nel registro dati.

Scrittura in memoria: operazione contraria

Memoria ad accesso casuale: il tempo di accesso non dipende dalla cella, viene chiamata RAM (Random Access Memory).

Accesso in Memoria Centrale



Accesso in memoria: dettaglio circuito logico

Memoria RAM

Le memorie **RAM** (Random Access Memory) sono memorie mantengono il loro contenuto finchè è presente l'alimentazione.

Esistono due tipi di memoria **RAM**:
RAM dinamica o **DRAM** (Dynamic Random Access Memory)
Economica (compatta) bassa potenza alimentazione, ma lenta e complessa. Dynamic: è necessario rigenerare i contenuti periodicamente (refresh circuit). Nei PC usata per memoria di sistema.

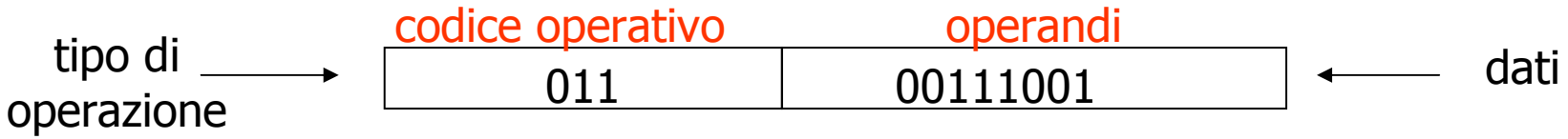
RAM statica o **SRAM** (Static Random Access Memory)
Semplice, veloce, ma costosa (poco compatta) alta potenza alimentazione. Static: il contenuto viene mantenuto finché è presente l'alimentazione. Nei PC usata per memoria cache.

Rappresentazione dell'Informazione (Programmi)

Anche i programmi sono rappresentati in memoria come sequenze di bit:

Programma ⇨ Sequenza di istruzioni ⇨ Ogni istruzione è codificata come sequenza di bit

Ogni istruzione, rapp. con una o più parole, è costituita da:

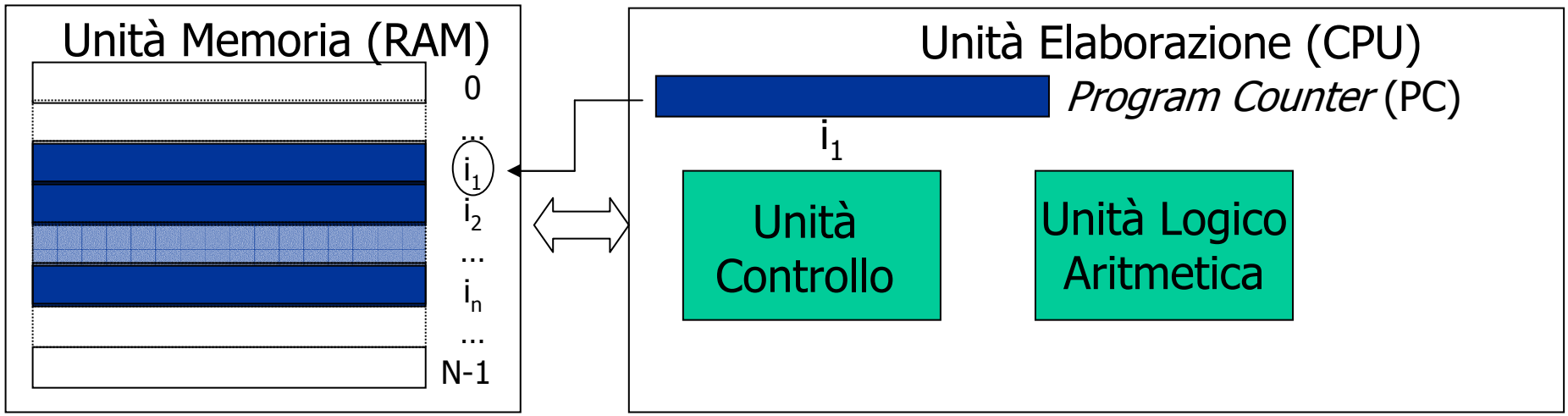


Esempio: le operazioni+,-,/,*, etc. possiamo codificarle come segue:

+ ⇨ 0	codice operativo: 000
- ⇨ 1	codice operativo: 001
/ ⇨ 2	codice operativo: 010
* ⇨ 3	codice operativo: 011
…	

Programma in Memoria

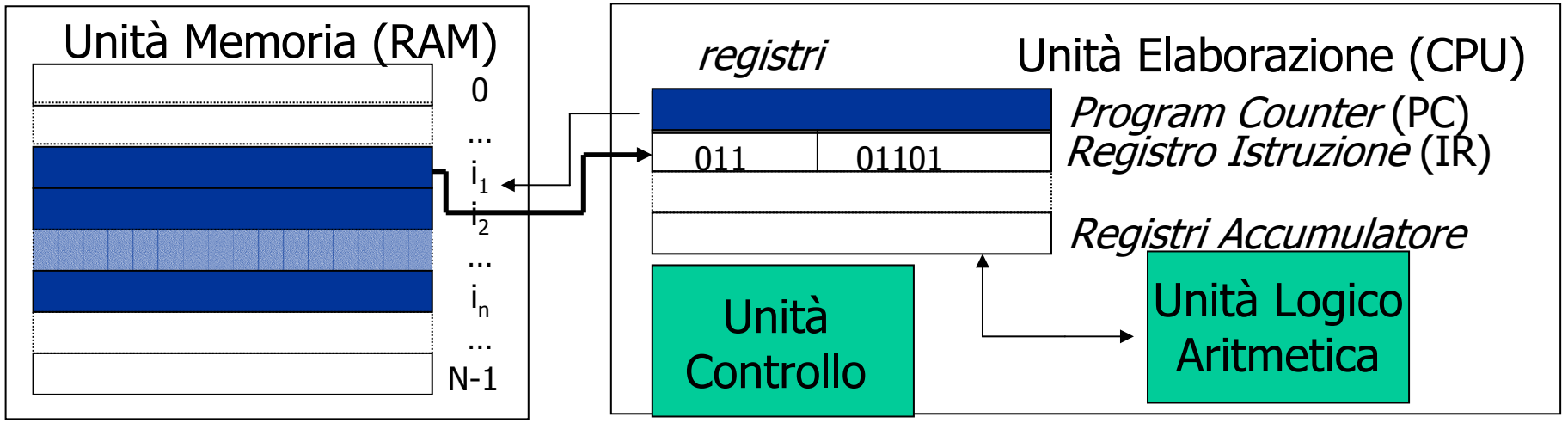
Un programma, per essere eseguito, viene caricato in memoria di lavoro:



- Le istruzioni sono caricate in celle consecutive a partire da un indirizzo iniziale;
- Una cella di memoria nella CPU (*registro PC*) registra l'indirizzo iniziale del programma.

Elaborazione Istruzioni

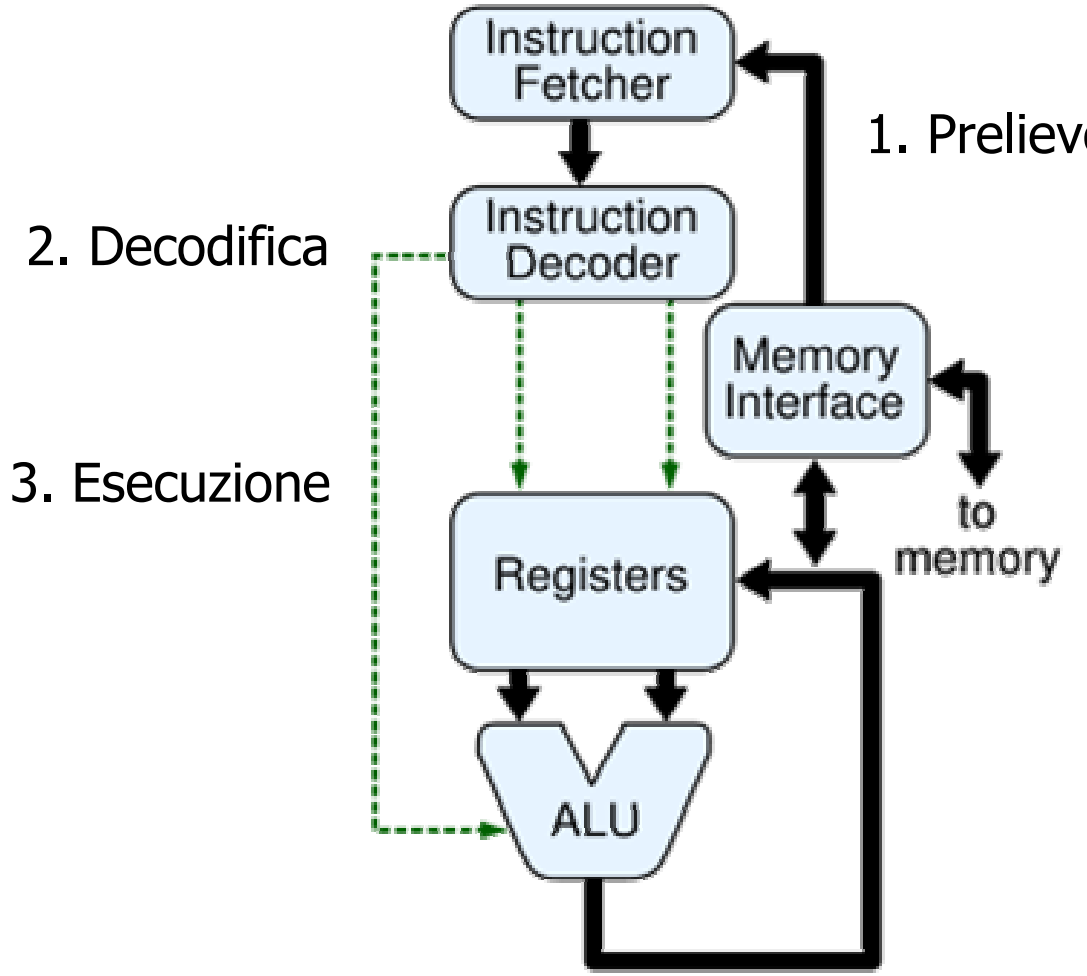
La CPU interpreta le istruzioni che compongono il programma e le esegue.
La CPU è costituta da 3 componenti principali: **CU, ALU, registri**.



CU esegue il ciclo: *prelievo – decodifica - esecuzione*

1. *prelievo* (*fetch*): istruzione puntata da **PC** caricata nel registro istruzione (**IR**);
2. *decodifica*: dal codice operativo viene individuata l'operazione da eseguire;
3. *esecuzione*: l'operazione viene eseguita. Utilizzati:
   - *Registri Accumulatori*: usati dalla ALU per operandi e risultati.
   - *Registro dei Flag:* memorizza informazioni sull'ultima operazione eseguita (carry, zero, segno,overflow, etc.)

Schema Funzionale CPU



Linguaggio Macchina

Alla CPU è associato un linguaggio di programmazione costituito da un set di istruzioni codificate in binario: *Linguaggio Macchina*.

Categorie di istruzioni:

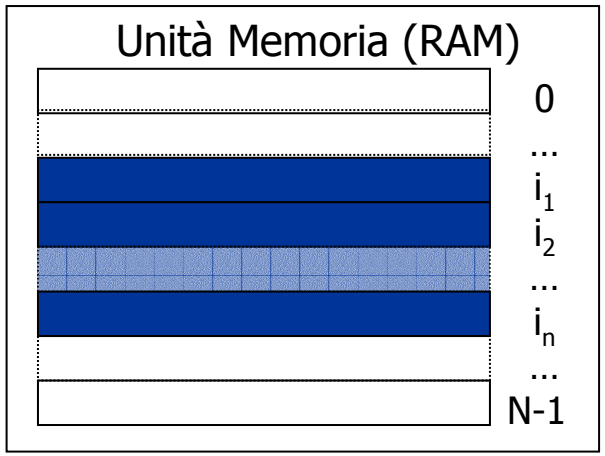
- Ingresso dati: comanda il trasferimento di dati da Unità Ingresso a Memoria
- Uscita Dati: comanda il trasferimento di dati da Memoria a Unità Uscita
- Trasferimento Dati: comanda trasferimento dati Registro-Registro, Registro- Memoria (R1 -> R2; R1 -> (R2))
- Operazione Aritmetica o Logica: comanda trasferimento dati **ALU** ed esecuzione dell'operazione (es. R1 + ACC -> ACC)
- Salto: aggiornamento del **PC** in modo opportuno (es. #7 -> PC)

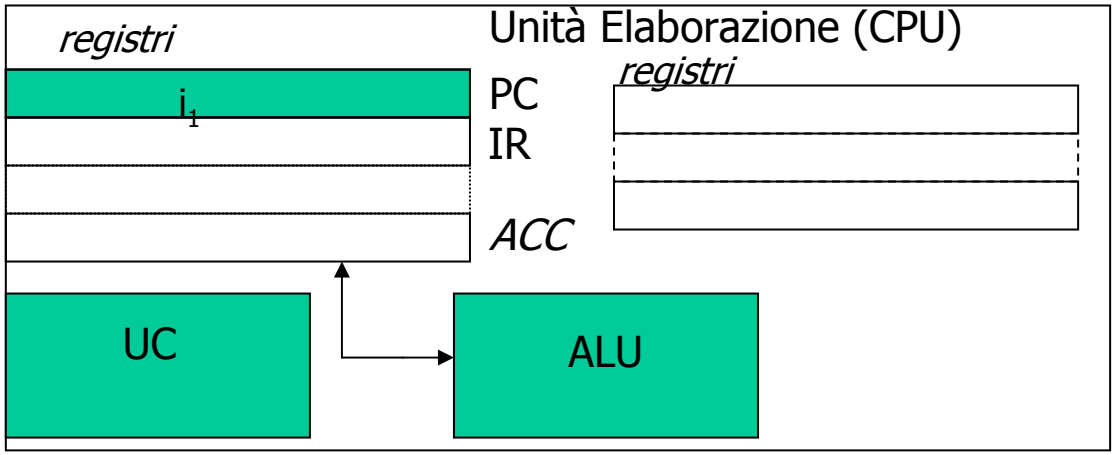
Esecuzione Programma

Un programma è una *sequenza di istruzioni* da eseguire in sequenza a meno che non vi sia un salto. L'esecuzione avviene con il ciclo:

ciclo {

1. **Fetch**: carica istruzione indirizzata da PC e metti in IR:
   (PC) -> IR;
2. **Decodifica**: dal codice in IC operando risali all'operazione;
   incrementa PC: **PC = PC + 1**;
3. **Esegui** l'istruzione }



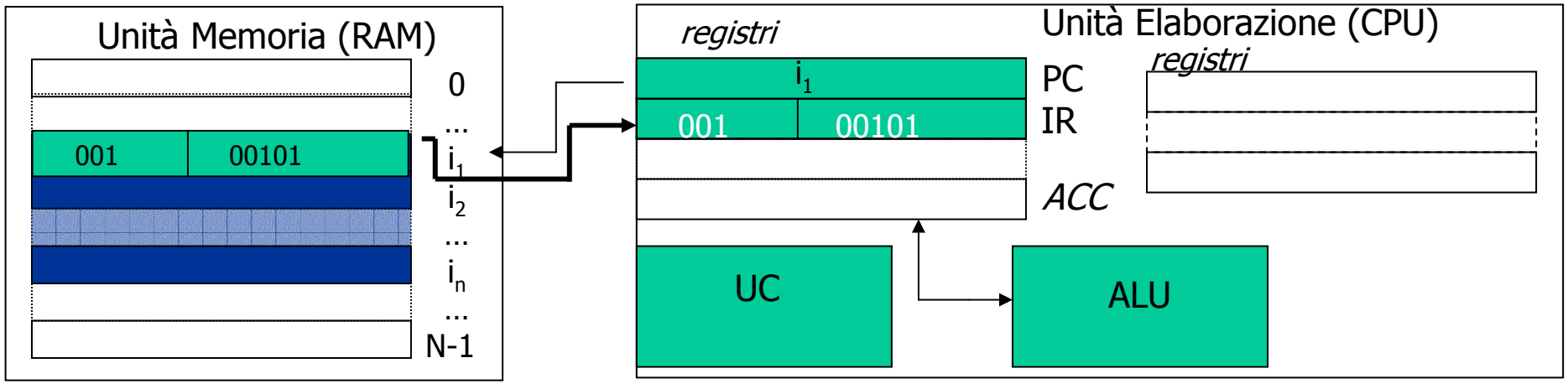


Esecuzione Programma

Un programma è una *sequenza di istruzioni* da eseguire in sequenza a meno che non vi sia un salto. L'esecuzione avviene con il ciclo:

ciclo {

1. **Fetch**: carica istruzione indirizzata da PC e metti in IR:
   (PC) -> IR;
2. **Decodifica**: dal codice in IC operando risali all'operazione;
   incrementa PC: **PC = PC + 1**;
3. **Esegui** l'istruzione }

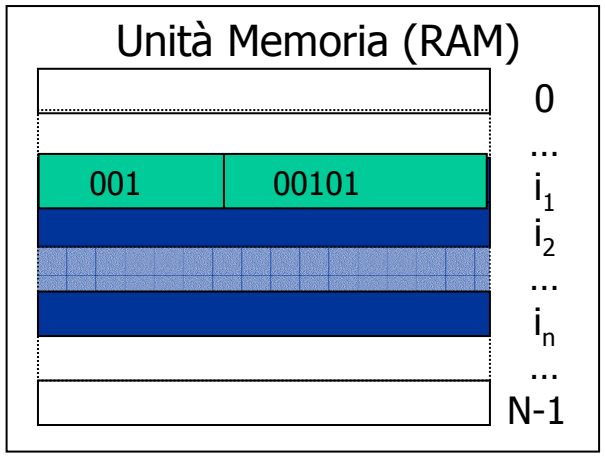


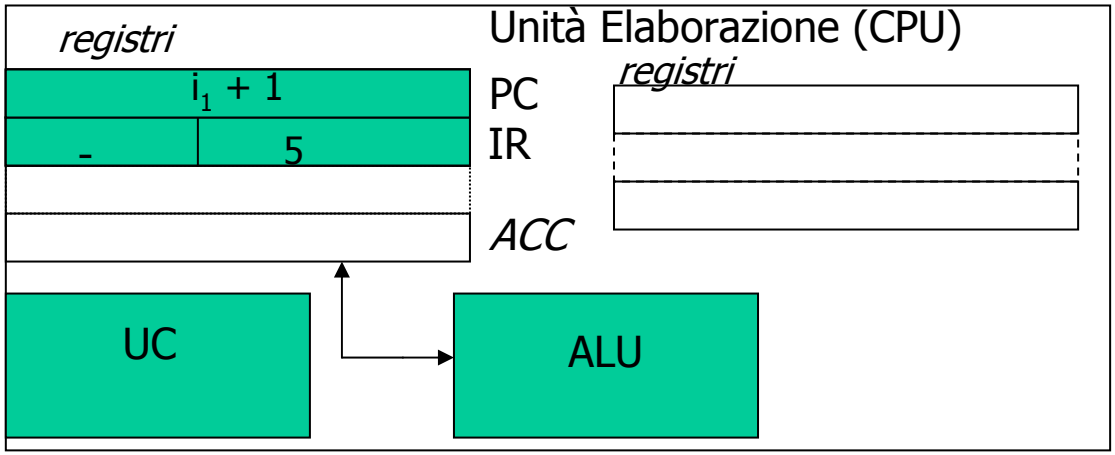
Esecuzione Programma

Un programma è una *sequenza di istruzioni* da eseguire in sequenza a meno che non vi sia un salto. L'esecuzione avviene con il ciclo:

ciclo {

1. **Fetch**: carica istruzione indirizzata da PC e metti in IR:
   (PC) -> IR;
2. **Decodifica**: dal codice in IC operando risali all'operazione;
   incrementa PC: **PC = PC + 1**;
3. **Esegui** l'istruzione }



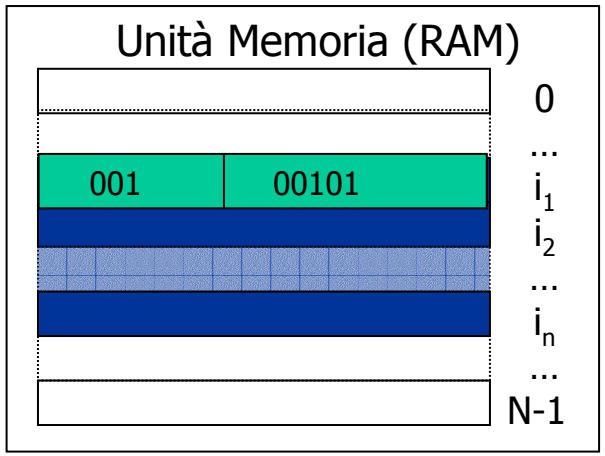


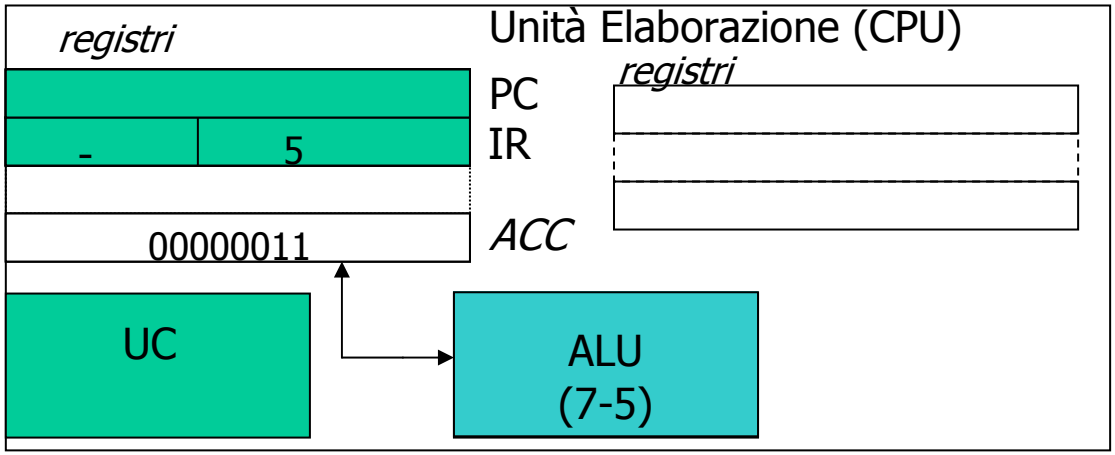
Esecuzione Programma

Un programma è una *sequenza di istruzioni* da eseguire in sequenza a meno che non vi sia un salto. L'esecuzione avviene con il ciclo:

ciclo {

1. **Fetch**: carica istruzione indirizzata da PC e metti in IR:
   (PC) -> IR;
2. **Decodifica**: dal codice in IC operando risali all'operazione;
   incrementa PC: **PC = PC + 1**;
3. **Esegui** l'istruzione }





Gerarchia dei Linguaggi

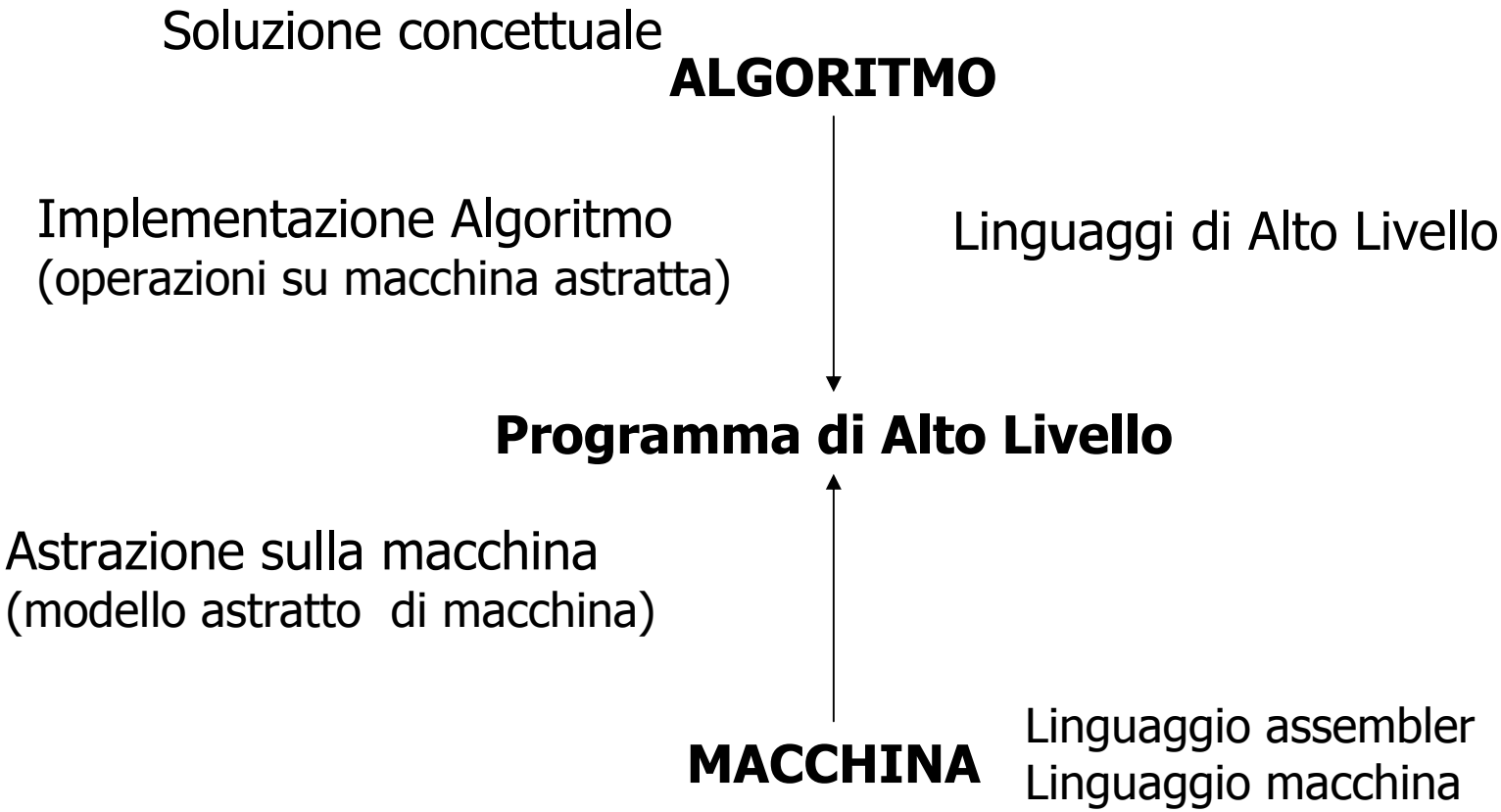
Livelli di astrazione dei linguaggi di programmazione (dal codice macchina verso l'algoritmo).

Linguaggio Macchina: linguaggio della CPU (dipende strettamente dal calcolatore), le istruzioni sono codificate in binario.

Linguaggio Assemblativo: Linguaggio Macchina tradotto in codice simbolico: stesse istruzioni scritte in forma più comprensibile. Semplifica la programmazione in linguaggio macchina.

Linguaggio ad Alto Livello: è basato su costrutti più astratti; non richiede un conoscenza dettagliata della macchina, in larga misura indipendente dalla macchina usata. Una istruzione di alto livello corrisponde a più istruzioni in linguaggio macchina. Esempi: C, Fortran, Pascal, C++, Java etc.

Gerarchia dei Linguaggi

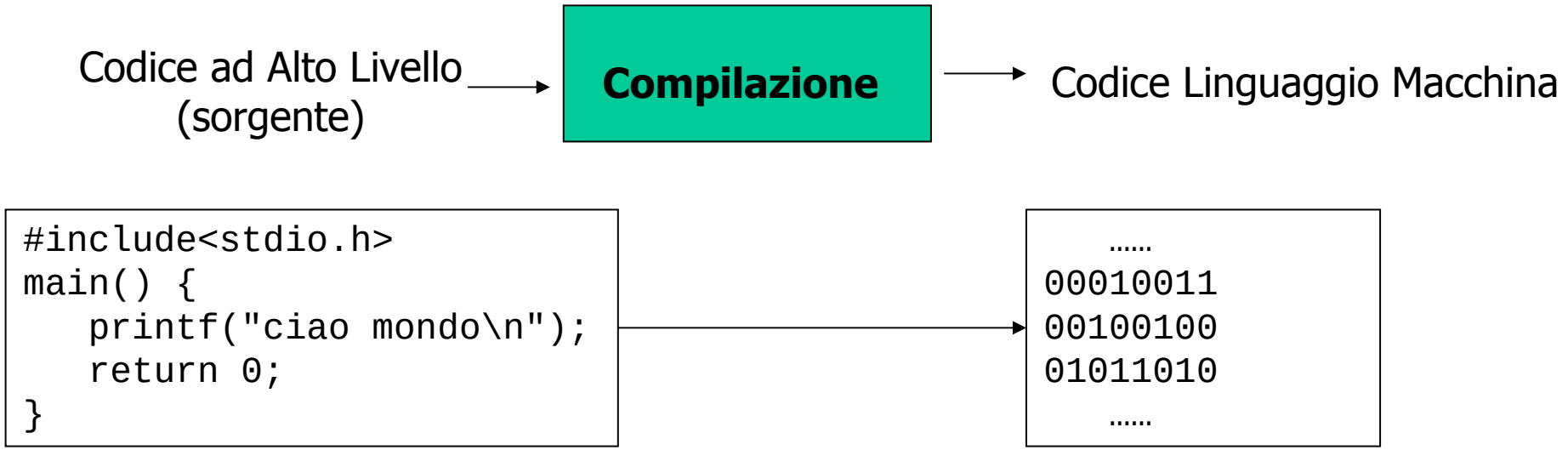


Esempio di Linguaggio Assembler

operazione	Codice assembler	significato
Caricamento di un dato	LOAD R1 X	Carica nel registro R1 il dato memorizzato in X
Somma Sottrazione	SUM R1 R2 SUB R1 R2	Somma (sottrae) i contenuti di R1 R2, il risultato è memorizzato in R1
Memorizzazione	STORE R1 X	Memorizza il contenuto di R1 nella cella denominata X
Lettura	READ X	Legge un dato memorizzandolo nella cella X
Scrittura	WRITE X	Scrive il valore contenuto nella cella X
Salto incondizionato	JUMP A	La prossima istruzione da eseguire è all'indirizzo etichettato con A
Salto condizionato	JUMPZ A	Se R1 = 0 allora la prossima istruzione da eseguire è all'indirizzo etichettato con A
Fine esecuzione	STOP	Ferma l'esecuzione

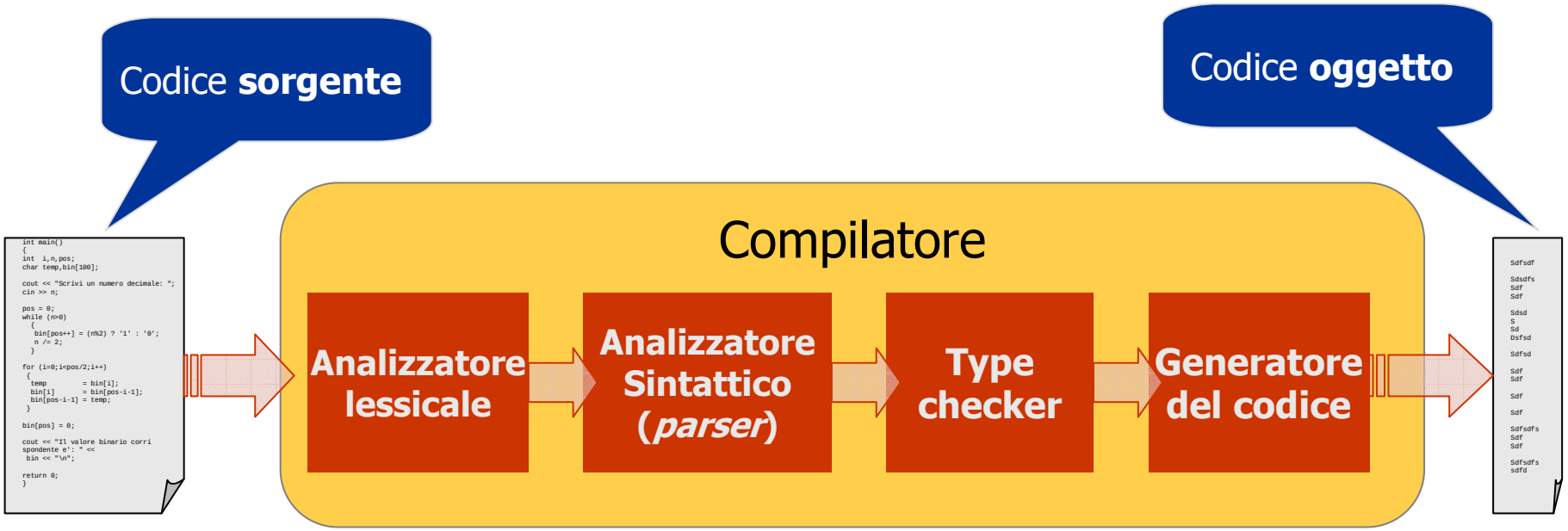
Compilazione

Compilazione: traduzione di codice da un linguaggio ad un altro.

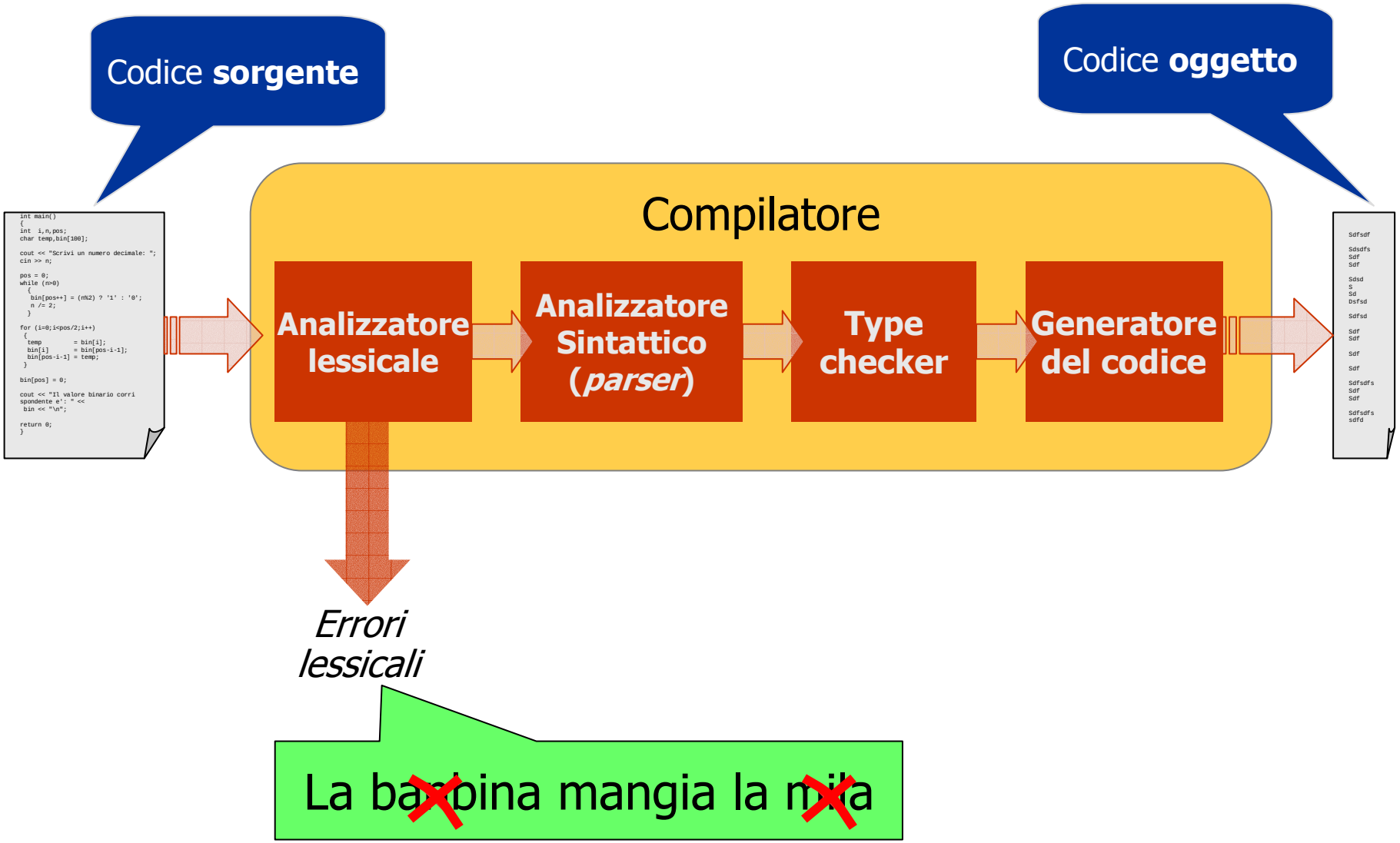


Assemblaggio: traduzione da *Codice Assemblativo* a *Codice Macchina*.

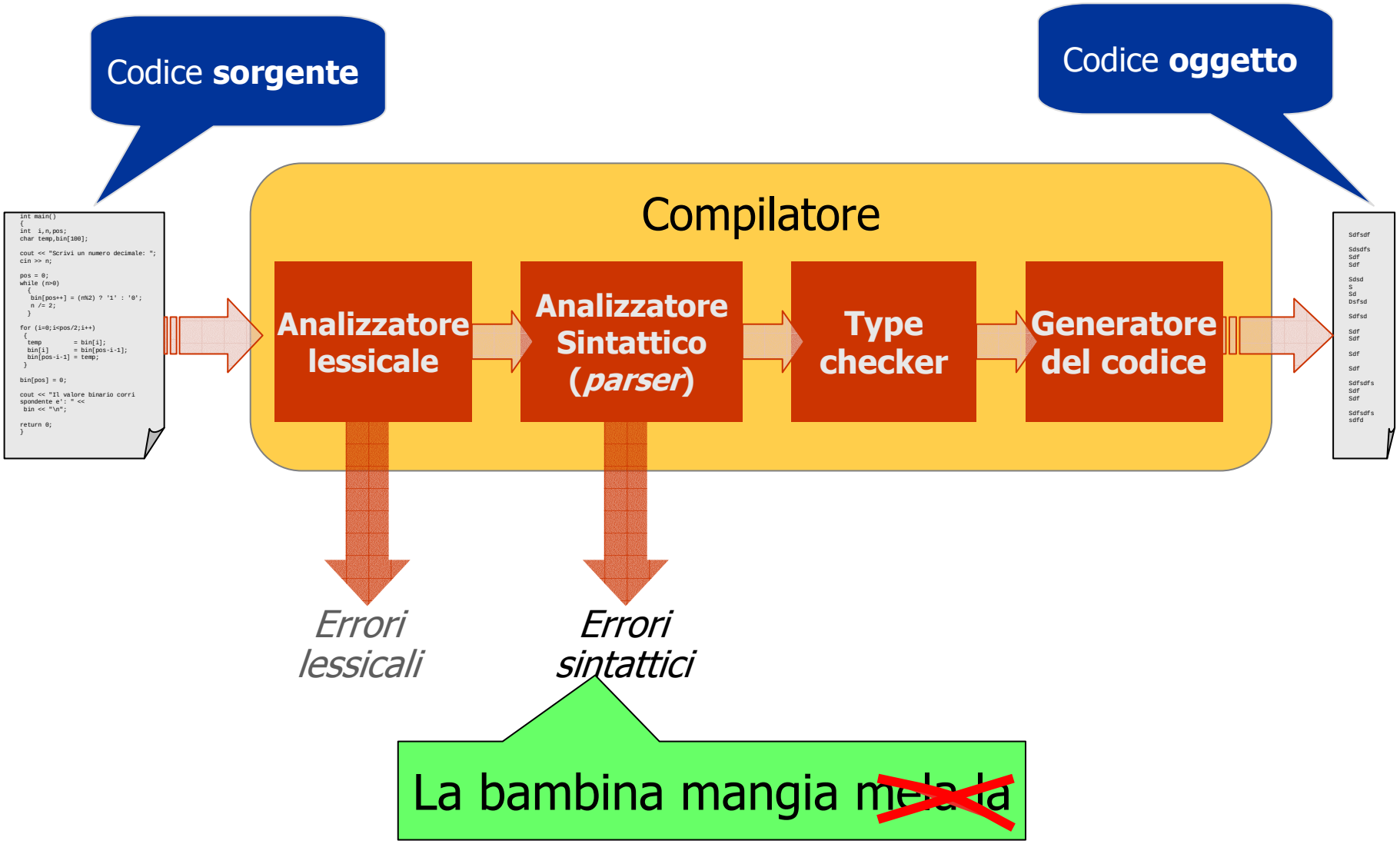
Il Compilatore



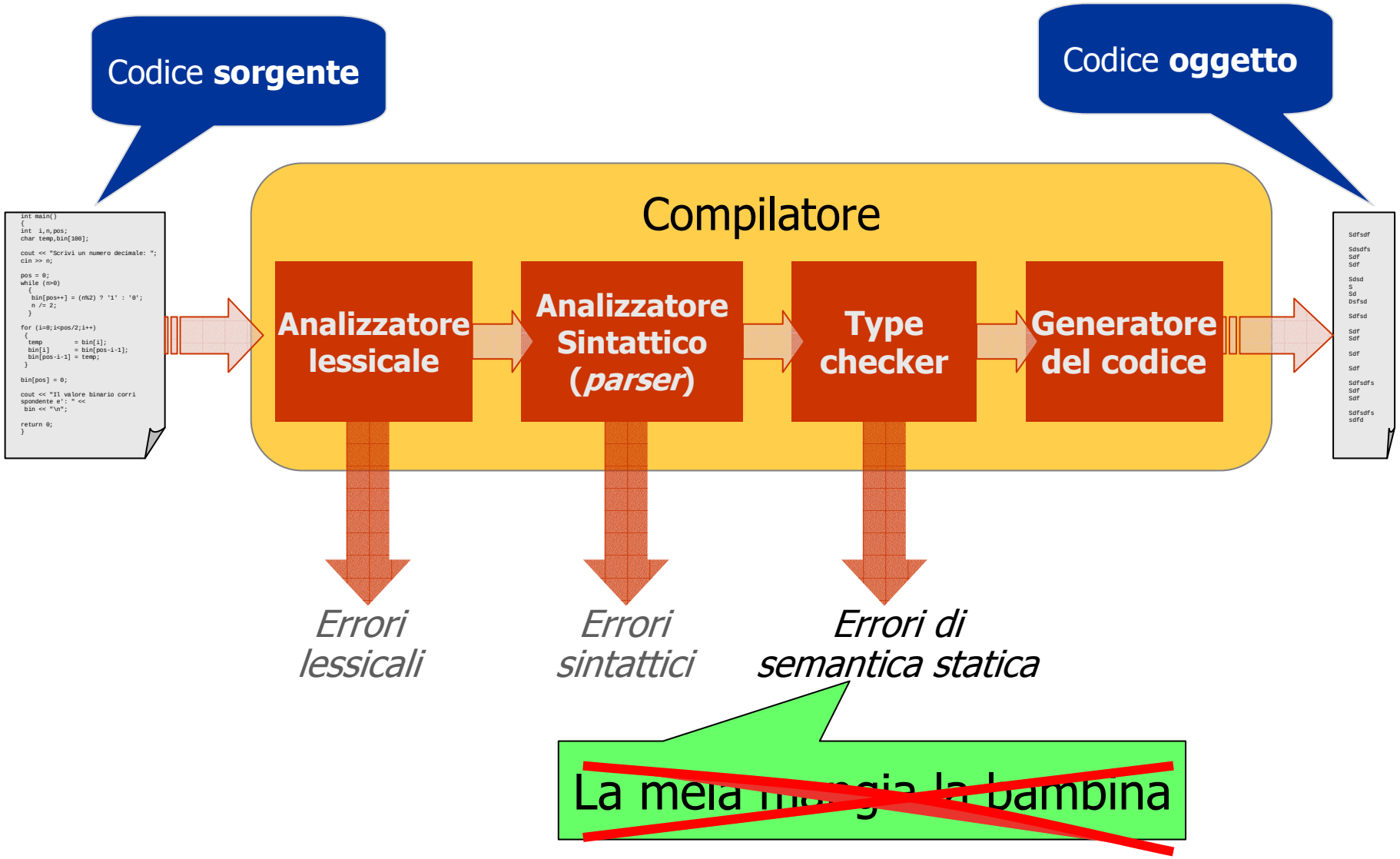
Il Compilatore



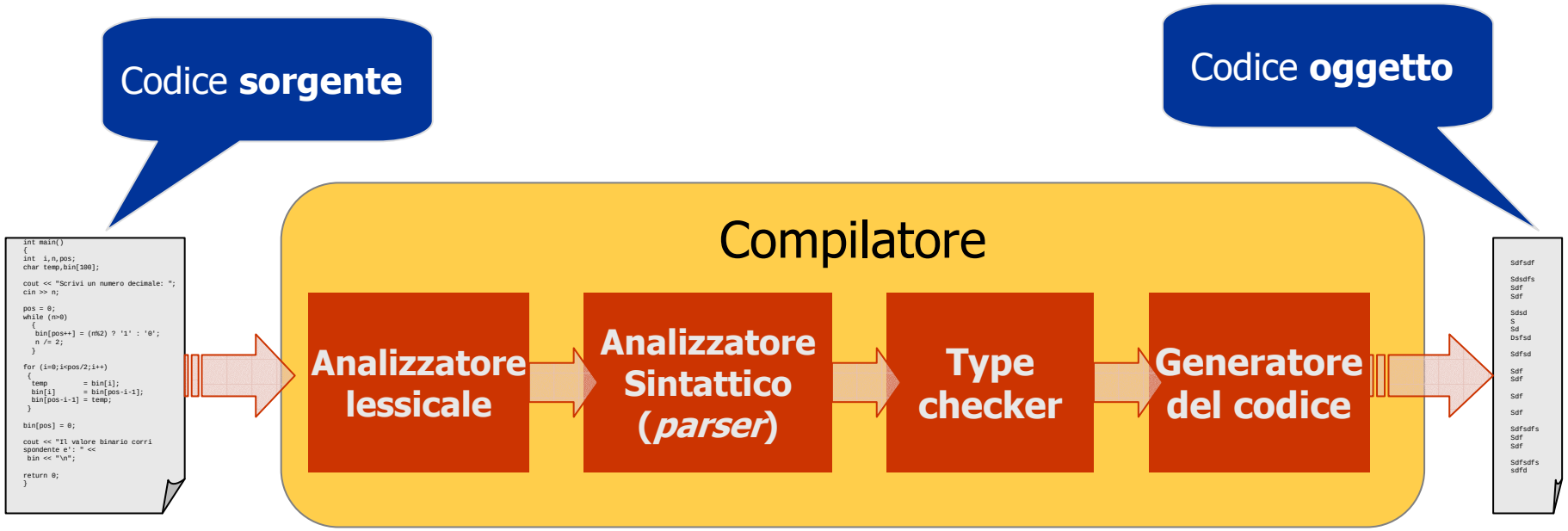
Il Compilatore



Il Compilatore



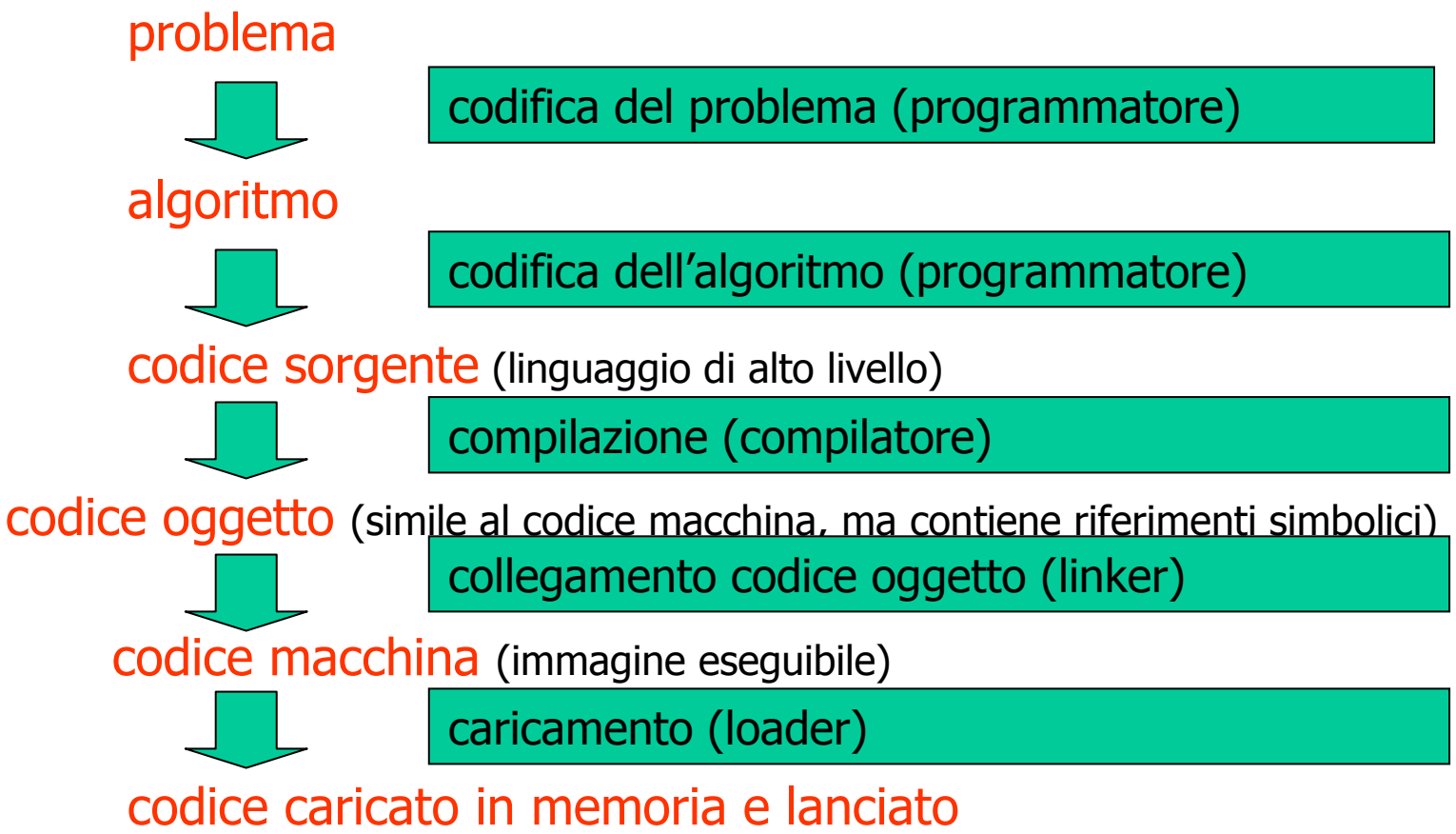
Il Compilatore



Il comportamento di un vero compilatore è molto più complesso di questa semplice schematizzazione: abbiamo infatti tralasciato per semplicità le fasi di preprocessamento, di gestione della tabella dei simboli, di assemblaggio, ottimizzazione e rilocazione del codice, di linking, ecc. ecc.

Dall'algoritmo all'esecuzione

Dalla formulazione di un problema all'esecuzione del codice che lo risolve si passa per diversi stadi:



Dall'algoritmo all'esecuzione

Codifica del programma: scrittura del programma con editor di testo (blocco note, Vi, emacs, etc.) definizione del codice sorgente

Compilazione: codice sorgente passato a compilatore che tradurrà in codice oggetto (in C compilazione preceduta da precompilazione).

Linking (collegamento): i programmi scritti in codice alto livello contengono riferimenti ad altre funzioni definite esternamente (librerire). Il linker collega il codice oggetto con il codice di queste funzioni e genera l'immagine eseguibile.

Caricamento: il programma deve essere caricato in memoria principale, questo è il compito del loader.

Esecuzione: il programma in memoria puo' essere eseguito dalla CPU

Ambienti di sviluppo **IDE** (Integrated Development Environment) permettono di gestire questo processo in tutte le sue fasi

Dall'algoritmo all'esecuzione: esempio

Problema: dati due interi positivi X ed Y eseguire il loro prodotto usando somma e sottrazione.

Algoritmo:

leggi *X* ed *Y*;
Somma $\leftarrow$ 0;
per Y volte ripeti
    Somma $\leftarrow$ Somma + X;
stampa la somma finale.

Raffinamento:

leggi *X* ed *Y*;
Somma $\leftarrow$ 0;
Cont. $\leftarrow$ 0;
fino a quando Cont. $\neq$ Y ripeti
    Somma $\leftarrow$ Somma + X;
    Cont. $\leftarrow$ Cont. + 1;
stampa la somma finale.

Dall'algoritmo all'esecuzione: esempio

Codice sorgente in C:

```
#include<stdio.h>
int main() {
   int x,y;
   int i = 0;
   int sum = 0;

   printf("introduci due numeri\n");
   scanf("%d%d",&x,&y);
   while (i != y) {
      sum = sum + x;
      i = i + 1;
   }
   printf("il prodotto di %d e %d è %d",x,y,sum);
   return 0;
}
```

Dall'algoritmo all'esecuzione: esempio

La compilazione genera codice macchina, vediamo il codice assembler corrispondente

Codice assember:

	READ X READ Y	scanf scanf	Legge valori e mem in celle X, Y
	LOAD R1 ZERO STORE R1 I	i = 0	Carica 0 in R1 Mem. R1 in cella I
	LOAD R1 ZERO STORE R1 SUM	sum = 0	Carica 0 in R1 Mem. R1 in cella SUM
INIZ	LOAD R1 I LOAD R2 Y SUB R1 R2 JUMPZ FINE	if (i==y) jump to FINE	Carica I in R1 Carica Y in R2 R1 = R1 – R2 se R1 è 0 vai a FINE
	LOAD R1 SUM LOAD R2 X SUM R1 R2 STORE R1 SUM	sum = sum + x	Carica SUM in R1 Carica X in R2 R1 = R1 + R2 mem. R1 in SUM
	LOAD R1 I LOAD R2 UNO SUM R1 R2 STORE R1 I	i = i + 1	Carica I in R1 Carica 1 in R2 R1 = R1 + R2 mem. R1 in SUM
	JUMP INIZ		Salta ad INIZ
FINE	WRITE SUM STOP	printf	Stampa SUM

Dall'algoritmo all'esecuzione: esempio

Nota:

1. Ad ogni istruzione C (alto livello) corrispondono più istruzioni assembler
2. I salti interrompono la sequenzialità del programma (non strutturato)

Il compilatore genera il linguaggio macchina (non assembler).

Vediamo di seguito un esempio di un possibile linguaggio macchina

Dall'algoritmo all'esecuzione: esempio

Esempio di linguaggio macchina:

Istr. Assembler	Codice Operativo
LOAD R1 ind	0000
LOAD R2 ind	0001
STORE R1 ind	0010
STORE R2 ind	0011
SUB R1 R2	0100
SUM R1 R2	0101
JUMP ind	0110
JUMPZ ind	0111
READ ind	1000
WRITE ind	1001
STOP	1010

Segue la codifica del prog. di esempio nel linguaggio macchina introdotto

Indirizzi	Cod.operativo	Operando	Istr. assembler
00000	1000	10101	READ X
00001	1000	10110	READ Y
00010	0000	10111	LOAD R1 ZERO
00011	0010	11001	STORE R1 I
00100	0000	10111	LOAD R1 ZERO
00101	0010	11000	STORE R1 SUM
00110	0000	11001	INIZ LOAD R1 I
00111	0001	10110	LOAD R2 Y
01000	0101	-----	SUB R1 R2
01001	0111	10011	JUMPZ FINE
01010	0000	11000	LOAD R1 SUM
01011	0001	10101	LOAD R2 X
01100	0100	-----	SUM R1 R2
01101	0010	11000	STORE R1 SUM
01110	0000	11001	LOAD R1 I
01111	0001	11010	LOAD R2 UNO
10000	0100	-----	SUM R1 R2
10001	0010	11001	STORE R1 I
10010	0110	00110	JUMP INIZ
10011	1001	11000	FINE WRITE SUM
10100	1011	-----	STOP
10101			X
10110			Y
10111	0000	00000	ZERO
11000			SUM
11001			I
11010	0000	00001	UNO