# Introduzione all'Informatica

In questa lezione:

- Cosa è l'informatica
- Problemi, algoritmi e programmi
- Cosa è un calcolatore
- Linguaggi di programmazione
- La macchina URM
- Algoritmi e programmi per la URM





# Cosa non è l'informatica

Lo studio del calcolatore.

- Il calcolatore è solo uno strumento.
- Ci sono molti oggetti di uso comune al giorno d'oggi in grado di elaborare informazioni: tablets, smart-phones, smart-tv, robot, lavatrici...
- Ci sono settori dell'informatica che riguardano aspetti non legati a nessun tipo di calcolatore, come l'informatica teorica.
- quindi ...

Edsger W. Dijkstra:

L'informatica non riguarda i calcolatori più di quanto l'astronomia non riguardi i telescopi.

## Cosa non è l'informatica

Lo studio di un linguaggio di programmazione.

- Un linguaggio di programmazione è solo uno strumento per descrivere soluzioni (algoritmiche) a problemi.
- Esistono tanti linguaggi di programmazione e addirittura classi di linguaggi: logici (es. Prolog), imperativi (es. Pascal), ad oggetti (es. Java, C++), funzionali (es. Lisp)...
- Non si possono non considerare altri aspetti come la adeguatezza e complessità delle soluzioni trovate.
- Quindi un linguaggio di programmazione è solo un mezzo per raggiungere uno scopo, non un fine.





## Cosa non è l'informatica

Lo studio delle applicazioni.

- Gli utenti finali *utilizzano* le applicazioni (es. word processors, fogli elettronici, email ...).
- Gli informatici si occupano della *specifica*, della *progettazione*, della *realizzazione* e della *validazione* delle applicazioni e dei sistemi su cui girano.
- L'abilità nell'uso di una applicazione è un aspetto limitato.
- Quindi apprendere l'uso di una applicazione non è parte dell'ingegneria informatica più di quanto saper guidare non è parte dell'ingegneria automobilistica.

# Cosa è l'informatica (1)

Un approccio tradizionale parte dalla definizione di algoritmo:

- *una procedura per risolvere un problema in un numero finito di passi*

e definisce l'informatica come:

Lo studio degli algoritmi, che comprende:

- le loro proprietà formali e matematiche
- le loro realizzazioni hardware
- le loro realizzazioni software
- le loro applicazioni





# Cosa è l'informatica (2)

Una proposta più recente (della Association of Computing Machinery, ACM) enfatizza l'oggetto del procedimento di calcolo:

- l'*informazione*

e definisce l'informatica come:

Lo studio sistematico degli algoritmi che descrivono e trasformano l'informazione: la loro teoria, analisi, progetto, efficienza, realizzazione, e applicazione

# Cosa è l'informatica (3)

Una proposta più generale (che include la precedente) parte dal concetto di informazione definisce l'informatica come:

La scienza della rappresentazione e dell'elaborazione dell'informazione.





# Informatica

Il termine **Informatica**, introdotto da Karl Steinbuch (Informatik) nel 1957 e poi usato in Francia negli anni '60 (Informatique), deriva dalla contrazione di **informazione automatica**.

**Informatica**

Trattamento automatico dell'informazione.

Trattamento: rappresentazione, conservazione, elaborazione

automatico: deleghiamo uno strumento

informazione: relazione tra dati

## Cos'è un problema

"Qual è il prodotto di 12 per 13?" è una domanda.
È un caso particolare del problema:
"Calcolare il prodotto dei numeri $x$ e $y$"

Un problema è una classe di domande omogenee a cui dare una risposta mediante una procedura ben definita.

Ogni singola domanda si chiama *istanza* del problema.

Ogni istanza è caratterizzata da:

- un insieme di dati di partenza (es. $\{12, 13\}$), l'input
- un risultato cercato (es. 156), l'output

Vogliamo dare una "risposta" che vada bene per ogni istanza del problema. Cioè vogliamo dare una soluzione al problema. Come fare?





## Algoritmo

La soluzione di un problema è definita da una procedura che, a partire da qualsiasi insieme di dati di partenza, produce un risultato. In particolare la soluzione è un algoritmo, di cui diamo una definizione più dettagliata.

Un algoritmo è una sequenza finita di passi elementari che dai dati di partenza produce un risultato per ogni istanza di un problema.

Cosa è un "passo elementare"?
Ricordate l'algoritmo per trovare il prodotto di due numeri?

```
 12 ×
 13 =
 ---
 36
12 -
---
156
```

In questo caso i passi elementari, che si assume si sappiano compiere, sono il calcolo della somma e del prodotto di cifre (tabellina di Pitagora!).

La parola "algoritmo" deriva dal nome del matematico Muhammad ibn Musa al-Khwarizmi (c. 780–850).

# Esempio di algoritmo

Problema: Qual è il costo netto di un prodotto pagato $c$ con iva inclusa al $t$ per cento?

Algoritmo:

| |
|---|
| dividi $c$ per $1 + t/100$ |
| mostra il risultato |

L'algoritmo è corretto?

Quanti passi svolge?

Se l'iva è al 20%, e ho pagato 120 euro, qual è il costo netto?
e se ho pagato 100?





# Un problema più complesso

Problema: fare due squadre equilibrate assumendo che ogni giocatore abbia un valore associato.

Come risolvereste il problema conoscendo il valore dei giocatori?

Gli informatici studiano questi tipi di problemi.

[SP12] **PARTITION**

INSTANCE: Finite set $A$ and a size $s(a) \in Z^+$ for each $a \in A$.
QUESTION: Is there a subset $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)$?

*Reference:* [Karp, 1972]. Transformation from 3DM (see Section 3.1.5).
*Comment:* Remains NP-complete even if we require that $|A'| = |A|/2$, or if the elements in $A$ are ordered as $a_1, a_2, \ldots, a_{2n}$ and we require that $A'$ contain exactly one of $a_{2i-1}, a_{2i}$ for $1 \leqslant i \leqslant n$. However, all these problems can be solved in pseudo-polynomial time by dynamic programming (see Section 4.2).

da "Computers and intractability", Garey M.R., Johnson D.S., 1979.

Il problema è difficile da risolvere (NP-complete): ogni algoritmo dovrà sostanzialmente provare tutte le combinazioni di squadre e scegliere la più equilibrata.

# Esempio di una procedura che non è un algoritmo

Problema: Sia dato un numero naturale $n$. Se è pari dividerlo per 2, altrimenti moltiplicarlo per 3 e aggiungere 1. Ripetendo la procedura su ogni nuovo numero trovato, si raggiunge 1?

Per esempio, la sequenza di numeri generati a partire da $n = 325$ è:

$325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1$

Algoritmo (?):

ripeti se $n$ è maggiore di 1:
    mostra $n$
    se $n$ è pari, sia $n$ uguale a $n/2$
    altrimenti sia $n$ uguale a $3 \cdot n + 1$

Perché la procedura nel riquadro non è propriamente un algoritmo?
Nel 1937 Lothar Collatz congetturò che la procedura porti ad 1, per ogni valore iniziale di $n$. Ad oggi la congettura non è dimostrata, quindi non possiamo asserire che la procedura termini in "numero finito di passi".
Paul Erdős: "La matematica non è ancora pronta per problemi di questo tipo"





# Cos'è un programma?

Un programma è una procedura scritta utilizzando un linguaggio formale in modo tale che possa essere eseguita da un calcolatore.

Un programma per un calcolatore è una collezione di istruzioni che permettono di risolvere istanze di un problema.

In questa definizione, per "istruzione" si intende un passo elementare eseguibile da un calcolatore.
Esempio di programma scritto in Python:

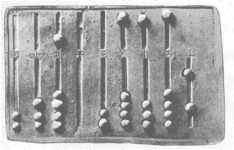
```
n = int(input("Inserisci un numero naturale: "))

while n > 1:
    print(n)
    if n % 2 == 0:
        n = n // 2
    else:
        n = 3*n + 1

print(n)
```

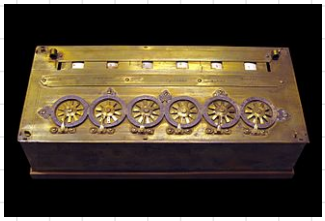
# Un po' di storia

Da sempre l'uomo ha avuto necessità di strumenti che lo supportassero nel calcolo.

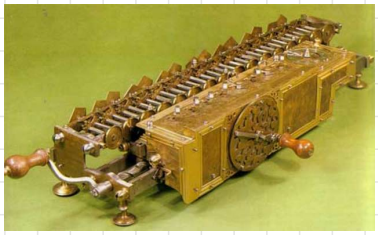


Abaco inventato 4000 anni fa. I romani facevano i conti muovendo sassolini (*calculi*, da cui calcolo)

Orologio calcolatore di Wilhelm Schickard (1621)



Pascalina di Baise Pascal (1642)



Calculating wheel di Gottfried Wilhelm Leibniz (1671)


Calcolatrice tascabile Curta di Curt Herzstark (1948), usa ancora le idee del progetto di Leibniz

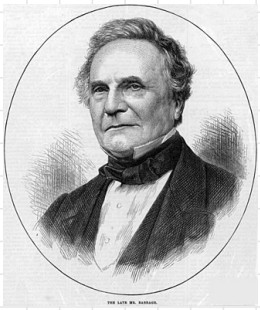




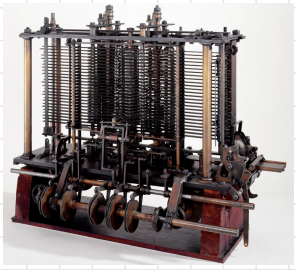
# Un po' di storia

Gli esempi visti sono calcolatrici, non veri computer. Con il termine calcolatrice si indicavano anche le addette ai calcoli negli uffici.


# Babbage e il primo computer meccanico



La Macchina Analitica è stato il primo prototipo di un computer meccanico sviluppato per eseguire compiti generici. Il progetto fu sviluppato dall'inglese Charles Babbage (1791–1871), che cercò anche di realizzarlo praticamente.

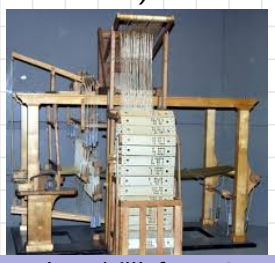


Parte della macchina analitica, museo della scienza, Londra.

La macchina analitica, azionata da un motore a vapore, avrebbe dovuto comporsi di quattro parti:

- la memoria (store)
- un'unità di calcolo (mill)
- la sezione di ingresso (lettore di schede perforate)
- la sezione di uscita (stampa dei risultati)

Ispirato da telaio programmabile di Joseph-Marie Jacquard che usava schede perforate

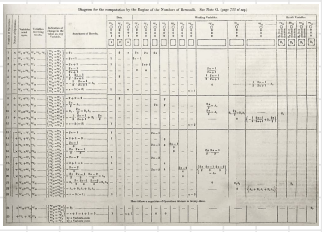






# Ada Lovelace: la prima programmatrice

Ada Byron contessa di Lovelace (1815 – 1852), ideò un algoritmo per la macchina analitica per generare i numeri di Bernoulli, considerato come il primo algoritmo per essere eseguito da una macchina



L'algoritmo è stato pubblicato come nota in un testo sulla macchina analitica che le fu chiesto di tradurre (1842).

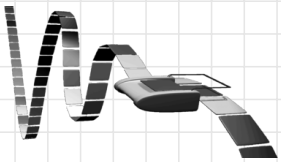
# Fondamenti logico-matematici

Nel 1928 David Hilbert, chiese di esibire una procedura, eseguibile del tutto meccanicamente, in grado di stabilire, per ogni formula espressa nel linguaggio della logica del primo ordine, se tale formula è o meno deducibile all'interno del sistema formale della logica del primo ordine.

Per rispondere a questa domanda Alonzo Church sviluppò il $\lambda$-calcolo su cui basò il concetto di funzione effettivamente calcolabile e Alan Turing introdusse il modello di calcolo oggi noto come Macchina di Turing.
Nel 1936 Church e Turing, indipendentemente, dettero una risposta negativa al problema posto da Hilbert.

In particolare, Turing studiò il problema della fermata che chiede se sia sempre possibile, dato un algoritmo e un input finito, stabilire se l'algoritmo in questione termini o continui la sua esecuzione all'infinito.
Turing dimostrò che non può esistere un algoritmo generale che possa risolvere il problema per tutti i possibili input e algoritmi. Il problema è indecidibile.

Alain Turing (1912 – 1954) e una rappresentazione artistica della Macchina di Turing



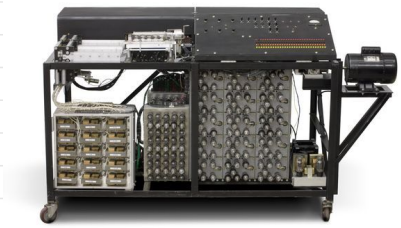




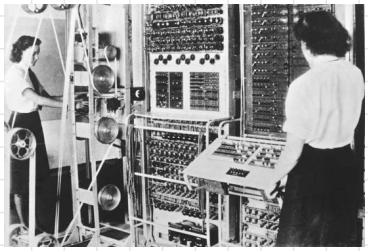
# Primi calcolatori elettronici

Non è chiaro chi ha costruito il primo calcolatore elettronico. Alcune importati tappe sono le seguenti.

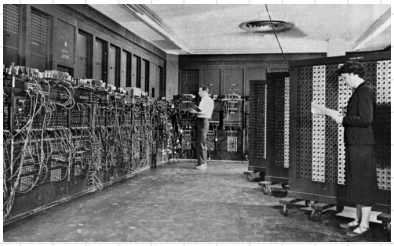
Prima generazione:
1937 - Calcolatore elettronico di J. V. Atanasoff e C. Berry: l'Atanasoff-Berry Computer (ABC).
1943 - Colossus: calcolatore elettronico per l'esercito statunitense
1946 - Electronic Numerical Integrator and Computer (ENIAC): primo calcolatore general–purpose, di J. P. Eckert e J. Mauchly (peso 30 tonnellate)



ABC (replica)



Colossus



ENIAC

# Primi calcolatori elettronici

Non è chiaro chi ha costruito il primo calcolatore elettronico. Alcune importati tappe sono le seguenti.

Seconda generazione (transistors invece di valvole termoioniche):
1951- Universal Automatic Computer (UNIVAC I): primo calcolatore commerciale
1953- la International Business Machine (IBM) introduce i calcolatori delle serie 650 and 700.
Vengono sviluppati linguaggi di programmazione, memorie secondarie come i nastri e i dischi, sistemi operativi, dispositivi di input (tastiere) e di output (stampanti)


Univac I


IBM650





# Primi calcolatori elettronici

Non è chiaro chi ha costruito il primo calcolatore elettronico. Alcune importati tappe sono le seguenti.
Terza generazione (invenzione del circuito integrato):
1963 - presente - I calcolatori si riducono di dimensioni. Molti calcoltori vengono introdotti nel mercato anche per uso domestico.
1981 - l'IBM introduce il personal computer (PC) per la casa e l'ufficio. Sistema operativo MS-DOS.
1984 - l'Apple produce il Macintosh interfaccia grafica a icone.

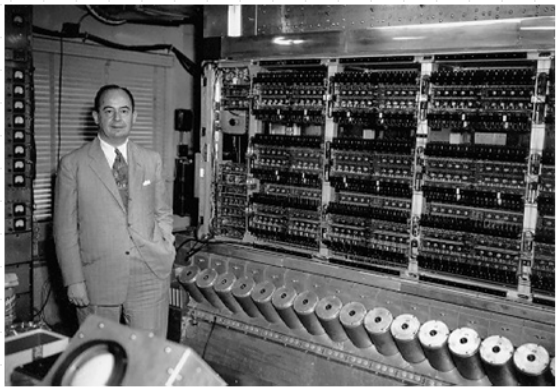

PC IBM


Olivetti M20 (1982)

Macintosh

# Come funziona un calcolatore elettronico

L'architettura dei moderni calcolatori è basata sul modello di Jhon von Neumann (1903-1957).



Jhon Von Neumann e il calcolatore MANIAC, 1952, Los Alamos Scientific Laboratory.

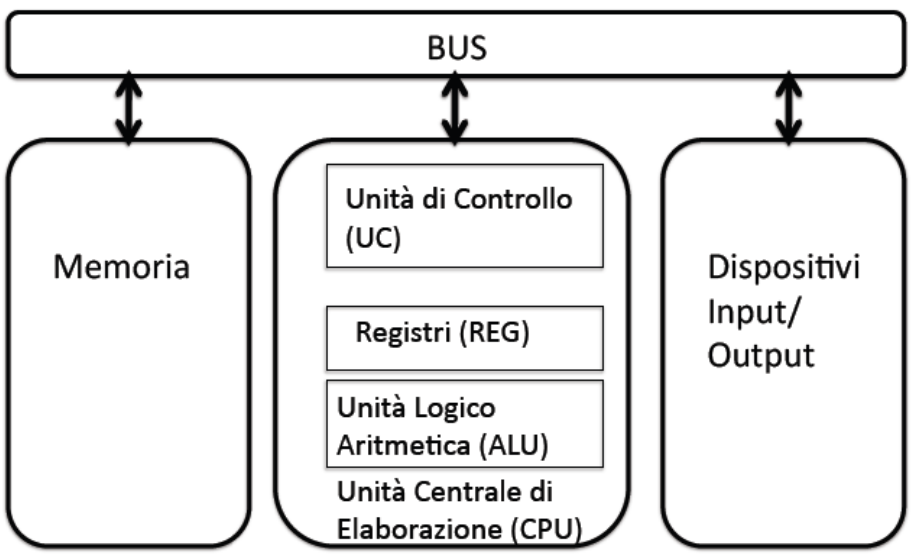
Von Neumann capì che occorreva che il programma non fosse rigidamente predisposto nell'hardware (tramite interruttori e cavi), e neppure letto sequenzialmente da lente schede perforate, ma risiedesse in una memoria scrivibile ad accesso veloce, assieme ai dati da elaborare e alle costanti numeriche.





# Come funziona un calcolatore elettronico

L'architettura dei moderni calcolatori è basata sul modello di Jhon von Neumann (1903-1957).



# Come funziona un calcolatore elettronico

## Componenti principali

I componenti principali di un calcolatore secondo l'architettura di von Neumann sono:

- la Memoria, dove vengono memorizzati programmi e dati (da elaborare e i risultati
- l'Unità di Elaborazione Centrale, detta CPU (Central Processing Unit) ha il compito di elaborare i dati secondo le istruzioni del programma
- le unità periferiche, per immettere dati nel calcolatore e per ottenerli in uscita dal calcolatore
- il Bus, che connette tra loro i componenti permettendo il transito dei dati





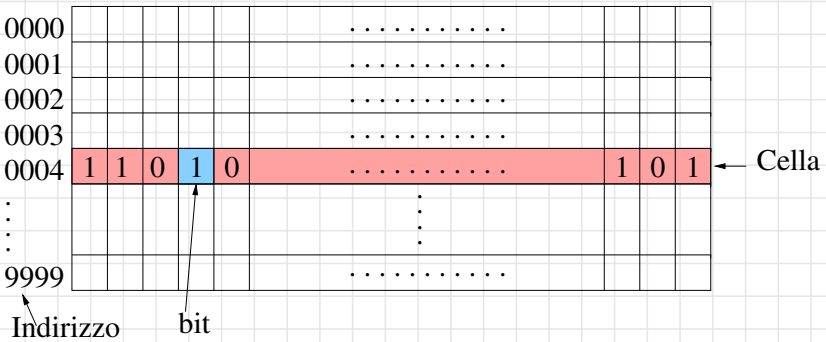
# Come funziona un calcolatore elettronico

## La memoria

La memoria è costituita da una serie di celle contenenti una parola (word). Ciascuna parola è composta da una sequenza di bit (binary digit).

Il bit è l'unità minima di memoria e può essere in uno di due stati, indicati come **0** e **1**.

Gli attuali calcolatori hanno celle con parole di 32 o 64 bit, organizzate in byte. Un byte è una sequenza di 8 bit. Quindi le celle di 32 bit sono composte da 4 byte, mentre quelle di 64 bit da 8 byte.



# Come funziona un calcolatore elettronico

## La memoria

Cosa posso memorizzare in un byte?
Se ho a disposizione 1 bit posso distinguere tra due stati, con 2 bit tra 4 stati, con 3 bit tra 8 stati. In generale con $n$ bit posso distinguere $2^n$ stati.
Esempio: con 3 bit posso distinguere 8 stati e associarli a numeri tra 0 e 7 o ai giorni della settimana.

| bit | num |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

| bit | giorno |
|---|---|
| 000 | – |
| 001 | lunedì |
| 010 | martedì |
| 011 | mercoledì |
| 100 | giovedì |
| 101 | venerdì |
| 110 | sabato |
| 111 | domenica |

Con un byte possiamo codificare i numeri tra 0 e $255 = 2^8 - 1$.
Oppure, con un byte possiamo codificare una qualsiasi lettera, maiuscola o minuscola, cifra o carattere di punteggiatura.





# Come funziona un calcolatore elettronico

## La memoria

La quantità di memoria si misura in byte e multipli del byte:

| Simbolo | Nome | Valore effettivo | | Valore pratico |
|---|---|---|---|---|
| B | Byte | $2^0$ | 1 | 8 bit |
| KB | KiloByte | $2^{10}$ | 1.024 | ~ 1.000 |
| MB | MegaByte | $2^{20}$ | 1.048.576 | ~ 1.000.000 |
| GB | GigaByte | $2^{30}$ | 1.073.741.824 | ~ 1.000.000.000 |
| TB | TeraByte | $2^{40}$ | 1.099.511.627.776 | ~ 1.000 Miliardi |
| PB | PetaByte | $2^{50}$ | 1.125.899.906.842.624 | ~ Un Milione di Miliardi |

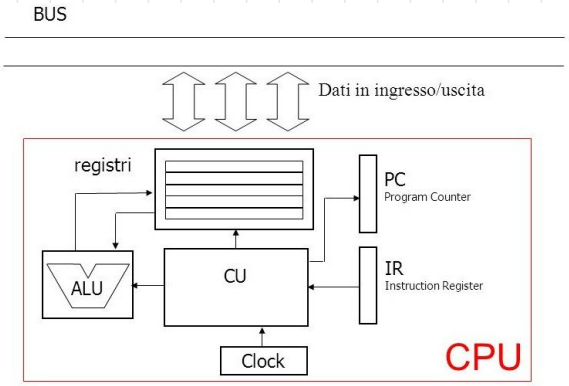
Se in un calcolatore abbiamo una memoria principale (RAM) di 8 GB, allora vuol dire che sono presenti più di 8 miliardi di byte (64 miliardi di bit).

# Come funziona un calcolatore elettronico

## L'Unità di Elaborazione Centrale (CPU)

La CPU è il dispositivo principale del calcolatore ed è composta da:

- l'Unità di controllo, anche CU (Control Unit), controlla tutte le operazioni del calcolatore, preleva le istruzioni dalla memoria e i dati necessari all'esecuzione, invia i segnali di esecuzione dell'istruzione alle altre unità.
- l'Unità logico-aritmetica, detta ALU (Arithmetic & Logic Unit) si occupa di eseguire le istruzioni aritmetiche e logiche.
- i registri, memoria locale della CPU dove memorizzare gli operandi e i risultati delle operazioni.








# Come funziona un calcolatore elettronico

## Cliclo Fetch-decode-execute

La CPU esegue continuamente un ciclo di tre fasi: fetch, decode, execute.

fetch: prelievo dalla memoria dell'istruzione da eseguire (l'indirizzo è nel registro Program Counter (PC)). L'istruzione viene messa nell'Istruction Register (IR). Il PC viene incrementato automaticamente per puntare alla prossima istruzione. Unità coinvolte: memoria e unità di controllo.

decode: decodifica dell'struzione presente nell'IR e prelievo degli eventuali operandi che vengono messi nei registri. Unità coinvolte: memoria e unità di controllo.

execute: l'unità di calcolo esegue l'istruzione presente nell'IR. Eventuali risultati nei registri vengono trasferiti in memoria dall'unità di controllo. Se l'istruzione prevede un salto, il PC viene sovrascritto.

# Come funziona un calcolatore elettronico

## Periferiche e Bus

Le unità periferiche sono dispositivi necessari per la comunicazione da e per il calcolatore. Si distinguono in:

- Unità di input: per immettere le informazioni nel calcolatore (programmi e dati)
- Unità di output: riceve le informazioni dalla memoria per renderle disponibili all'utente

Esempi di unità di input: tastiera, mouse, touchpad, microfono
Esempi di unità di ouput: schermo, altoparlanti, stampanti
Esempi di unità di input/output: memorie secondarie, touch-screen, dispositivi di collegamento alle reti

L'ultimo componente è il

- Bus: canale di comunicazione che permette ai dati di transitare tra le componenti di un calcolatore





# Il linguaggio macchina

Il linguaggio macchina è un linguaggio direttamente comprensibile ed eseguibile dall'elaboratore, senza nessuna traduzione.

Caratteristiche:

- istruzioni ed operandi presenti in memoria
- tutto è espresso in forma binaria

Conseguenze:

- difficile da scrivere
- difficile da interpretare per esseri umani

# Esempio giocattolo di istruzione in LM

Una istruzione in linguaggio macchina può essere a lunghezza fissa (per esempio di una cella) o variabile da 1 a più byte.
Vediamo un esempio di istruzione a lunghezza fissa.

| byte 1 | byte 2 | byte 3 | byte 4 |
|---|---|---|---|
| 10010101 | 00110100 | 00110101 | 00110110 |

Assummendo che il codice operativo del byte 1 (10010101) si riferisca all'istruzione "addiziona" e che i codici nei byte 2,3 e 4 agli indirizzi di memoria dove reperire gli operandi e mettere il risultato, l'interpretazione dell'istruzione è:

Addiziona il contenuto delle celle 52 e 53 e metti il risultato nella cella 54.





# Il linguaggio assemblativo

Il linguaggio assemblativo (assembly) è un linguaggio di basso livello (comunque legato alla macchina) che consente al programmatore di ignorare il formato binario del linguaggio macchina. Ogni codice operativo del linguaggio macchina viene sostituito da una sequenza di caratteri che lo rappresenta in forma mnemonica.

Per esempio, il codice operativo per l'addizione potrebbe essere trascritto come ADD e l'istruzione precedente diventerebbe:

```
ADD   $52,  $53,  $54
```

Quindi un programma scritto in assembly è facilmente leggibile dai programmatori. Per poter essere eseguito su un calcolatore deve essere tradotto in linguaggio macchina.

Questo è il compito di un programma detto assemblatore (assembler).

# I linguaggi di alto livello

Anche il linguaggio assemblativo è di difficile uso perché legato alla macchina più che alle esigenze del programmatore.

Sono quindi stati introdotti i linguaggi di alto livello (C, Fortran, C++, Java, Python, ...).

Per essere eseguiti hanno comunque la necessità di essere tradotti in linguaggio macchina.





# I traduttori

Esistono due tipi di traduttori:

i compilatori: accettano in ingresso l'intero programma e producono in uscita la rappresentazione dell'intero programma in un altro linguaggio.
Possono esserci catene di compilatori: ad esempio i primi compilatori del C++ traducevano il codice C++ in un programma in C, che veniva compilato in assembly, a sua volta tradotto in linguaggio macchina dall'assembler.

gli interpreti: traducono ed eseguono direttamente ciascuna istruzione del programma sorgente, una dopo l'altra.

Il Python è un linguaggio (essenzialmente) interpretato.

# Esempio di programma in C++

Un semplice programma in C++, HelloWorld:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, World!\n";
    return 0;
}
```





# Codice in Assembly

Parte del codice in assembly del programma HelloWorld:

```
        ...

        movl    %edi, -4(%rbp)
        movl    %esi, -8(%rbp)
        cmpl    $1, -4(%rbp)
        jne     .L5
        cmpl    $65535, -8(%rbp)
        jne     .L5
        movl    $_ZStL8__ioinit, %edi
        call    _ZNSt8ios_base4InitC1Ev
        movl    $__dso_handle, %edx
        movl    $_ZStL8__ioinit, %esi
        movl    $_ZNSt8ios_base4InitD1Ev, %edi
        call    __cxa_atexit

        ...
```

# Codice in linguaggio macchina

Parte del codice in linguaggio macchina del programma HelloWorld:

```
00000880: bfc3 bfc2 ba48 1060 00c2 be71 1160 00c2  .....H.`...q.`..
00000890: bf50 0640 00c3 a875 c3be c3bf c3bf c290  .P.@...u........
000008a0: c389 c383 5548 c289 c3a5 c2be c3bf c3bf  ....UH..........
000008b0: 0000 c2bf 0100 0000 c3a8 c2af c3bf c3bf  ................
000008c0: c3bf 5dc3 8366 2e0f 1fc2 8400 0000 0000  ..]..f..........
000008d0: 0f1f 0041 5741 5641 c289 c3bf 4155 4154  ...AWAVA....AUAT
000008e0: 4cc2 8d25 c3b6 0520 0055 48c2 8d2d c3be  L..%... .UH..-..
000008f0: 0520 0053 49c2 89c3 b649 c289 c395 4c29  . .SI....I....L)
00000900: c3a5 48c2 83c3 ac08 48c3 81c3 bd03 c3a8  ..H.....H.......
00000910: c387 c3bd c3bf c3bf 48c2 85c3 ad74 2031  ........H....t 1
00000920: c39b 0f1f c284 0000 0000 004c c289 c3aa  ...........L....
00000930: 4cc2 89c3 b644 c289 c3bf 41c3 bf14 c39c  L....D....A.....
00000940: 48c2 83c3 8301 4839 c3ab 75c3 aa48 c283  H.....H9..u..H..
00000950: c384 085b 5d41 5c41 5d41 5e41 5fc3 83c2  ...[]A\A]A^A_...
00000960: 9066 2e0f 1fc2 8400 0000 0000 c3b3 c383  .f..............
00000970: 0000 48c2 83c3 ac08 48c2 83c3 8408 c383  ..H.....H.......
00000980: 0000 0001 0002 0048 656c 6c6f 2c20 576f  .......Hello, Wo
00000990: 726c 6421 0a00 0001 1b03 3b40 0000 0007  rld!......;@....
000009a0: 0000 00c2 8cc3 bdc3 bfc3 bfc2 8c00 0000  ................
000009b0: c3bc c3bd c3bf c3bf 5c00 0000 c3b2 c3be  ........\.......
000009c0: c3bf c3bf c2b4 0000 000c c3bf c3bf c3bf  ................
000009d0: c394 0000 004a c3bf c3bf c3bf c3b4 0000  .....J..........
000009e0: 006c c3bf c3bf c3bf 1401 0000 c39c c3bf  .l..............
000009f0: c3bf c3bf 5c01 0000 1400 0000 0000 0000  ....\...........
```

← Hello World!

indirizzi – contenuto celle di 64 bit – interpretazione a caratteri





# La macchina URM

Per poter definire senza ambiguità concetti di base come istruzione, programma, memoria ed altri, introduciamo un modello formale di calcolatore.

La nostra idealizzazione matematica di un calcolatore è la *Macchina a Registri Illimitati* o macchina URM da *Unlimited Register Machine*, modello introdotto da Shepherdson e Sturgis nel 1963.

Questa idealizzazione da un lato è vicina all'architettura di un calcolatore moderno, dall'altro è equivalente alla *macchina di Turing*.

# La macchina URM

## La memoria

La macchina URM ha un infinito numero di registri di memoria indicati con $R_0, R_1, R_2, \ldots$, ognuno dei quali ad ogni istante di tempo contiene un numero naturale.
Denotiamo il numero contenuto nel registro $R_n$ con $r_n$. I registri possono essere rappresentati come:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| $r_0$ | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ | $\ldots$ |

Il contenuto dei registri può essere alterato dalla macchina URM in risposta a certe *istruzioni* che sa riconoscere.
L'idealizzazione consiste nel fatto che

- i registri sono in numero illimitato, mentre in un calcolatore reale il numero di celle di memoria è finito;
- ogni registro contiene un numero naturale, comunque grande, mentre una cella di memoria ha un numero di bit finito.





# La macchina URM

## Le istruzioni

Il linguaggio della macchina URM è composto da solo quattro tipi di istruzioni. Queste istruzioni corrispondono a operazioni veramente molto semplici utilizzate per eseguire dei calcoli con i numeri naturali.

Una sequenza finita di istruzioni $I_1, I_2, \ldots, I_s$ costituisce un *programma*.

Le istruzioni sono:

azzeramento: permette di azzerare il contenuto di un registro

successore: incrementa di 1 il contenuto di un registro

trasferimento: copia il contenuto di un registro in un altro registro

salto: permette di saltare avanti o indietro nella sequenza di istruzioni

# La macchina URM

## Azzeramento

Per ogni $n = 0, 1, 2, \ldots$ esiste una istruzione di azzeramento $Z(n)$. Il comportamento della macchina URM a seguito dell'istruzione $Z(n)$ è porre il contenuto di $R_n$ a 0, lasciando invariato il contenuto degli altri registri.

*Esempio:* Supponiamo che la macchina sia nella seguente configurazione:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| 9 | 6 | 5 | 23 | 7 | 0 | 0 | $\ldots$ |

e che obbedisca all'istruzione di azzeramento $Z(2)$. Allora la configurazione risultante sarà:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| 9 | 6 | 0 | 23 | 7 | 0 | 0 | $\ldots$ |





# La macchina URM

## Successore

Per ogni $n = 0, 1, 2, \ldots$ esiste una istruzione di successore $S(n)$. Il comportamento della macchina URM a seguito dell'istruzione $S(n)$ è di incrementare di 1 il contenuto di $R_n$, lasciando invariato il contenuto degli altri registri.

*Esempio:* Supponiamo che la macchina sia nella seguente configurazione:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 5 | 3 | 17 | 0 | 4 | $\ldots$ |

e che obbedisca all'istruzione di successore $S(2)$. Allora la configurazione risultante sarà:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| 8 | 16 | 6 | 3 | 17 | 0 | 4 | $\ldots$ |

# La macchina URM

## Trasferimento

Per ogni $m = 0, 1, 2, \ldots$ e $n = 0, 1, 2, \ldots$ esiste una istruzione di trasferimento $T(m, n)$. Il comportamento della macchina URM a seguito dell'istruzione $T(m, n)$ è di rimpiazzare il contenuto del registro $R_n$ con il numero $r_m$ contenuto in $R_m$ (cioè trasferire $r_m$ in $R_n$), lasciando invariato il contenuto degli altri registri compreso $R_m$.

*Esempio:* Supponiamo che la macchina sia nella seguente configurazione:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| 81 | 1 | 50 | 0 | 7 | 10 | 4 | $\ldots$ |

e che obbedisca all'istruzione di successore $T(4, 0)$. Allora la configurazione risultante sarà:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| 7 | 1 | 50 | 0 | 7 | 10 | 4 | $\ldots$ |





# La macchina URM

## Salto

Un algoritmo può avere passi in cui scegliere tra sequenze alternative di azioni oppure, in altre situazioni, passi da ripetere più volte. La macchina URM è in grado di far fronte a queste esigenze con l'istruzione di salto.
Esempio: con il salto può ottenere il seguente comportamento:

> *Se $r_6 = r_2$ vai alla decima istruzione del programma, altrimenti prosegui con l'istruzione seguente.*

L'istruzione che ci fornisce questo comportamento è $J(6, 2, 10)$.
In generale, per ogni $m = 0, 1, 2, \ldots$, $n = 0, 1, 2, \ldots$ e $q = 1, 2, 3, \ldots$ esiste una istruzione di salto $J(m, n, q)$ tale che:
se $r_m = r_n$, la macchina URM salta alla $q$-esima istruzione;
se $r_m \neq r_n$, la macchina URM procede con l'istruzione che segue.

Se il salto non è possibile perché il programma ha meno di $q$ istruzioni, allora la macchina URM si ferma.

# Algoritmi per la macchina URM

Un algoritmo per la macchina URM deve tener conto che per "passo elementare" si intende una delle quattro istruzioni appena viste.
Supponiamo di voler calcolare $f(x) = x + 2$ assumendo che inizialmente il valore della $x$ sia in $R_0$, e che il valore finale $x + 2$ sia nello stesso registro.
Un possibile algoritmo potrebbe essere:
– *Incrementa di 2 il contenuto del registro* $R_0$
ma l'incremento di 2 non è un passo elementare. Quindi un algoritmo più adatto alla URM potrebbe essere:
– *Incrementa di 1 il contenuto del registro* $R_0$
– *Incrementa di 1 il contenuto del registro* $R_0$
Ma una volta provato che un registro può essere incrementato di un valore costante allora possiamo accettare anche il primo algoritmo. In generale, se abbiamo provato che una funzione può essere calcolata possiamo accettare il calcolo di quella funzione come "passo" per la macchina URM.

Un domanda naturale: quali funzioni può calcolare la macchina URM?





# Programmi per la macchina URM

Per far eseguire un calcolo alla URM dobbiamo fornirle un *programma* $P$, che consiste in una sequenza finita di istruzioni $I_1, I_2, \ldots, I_s$, ed una *configurazione iniziale* – cioè una sequenza $a_0, a_1, a_2, \ldots$ di numeri naturali nei registri $R_0, R_1, R_2, \ldots$.
Esempio: Per calcolare la funzione $f(x) = x + 2$ un possibile programma è il seguente:

$$\begin{array}{ll} I_1 & S(0) \\ I_2 & S(0) \end{array}$$

La URM inizia il calcolo eseguendo l'istruzione $I_1$, poi $I_2$, $I_3$ e così via, a meno che non incontri un'istruzione di salto: se l'istruzione $I_k$ coincide con l'istruzione $J(m, n, q)$ e $r_m = r_n$ allora la URM passerà ad eseguire l'istruzione $I_q$, altrimenti proseguirà con l'istruzione $I_{k+1}$.

Se l'URM deve eseguire l'istruzione $I_t$ e questa non è definita, cioè $t > s$, allora l'URM si ferma e la sequenza $r_0, r_1, r_2, \ldots$ dei contenuti dei registri è detta *configurazione finale*.

# Un programma per la macchina URM

## Scrittura

Vogliamo calcolare la funzione $f(x, y) = x + y$. La configurazione iniziale è quindi $x, y, 0, 0, \ldots$. Il nostro programma aggiungerà 1 a $r_0$ $y$ volte, usando $R_2$ come contatore. In generale alla $k$-esima volta la configurazione sarà:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ | $\ldots$ |
|---|---|---|---|---|---|---|---|
| $x + k$ | $y$ | $k$ | 0 | 0 | 0 | 0 | $\ldots$ |

Il calcolo terminerà quando $k = y$, lasciando $x + y$ in $R_0$. Un possibile programma è il seguente:

$I_1$ $J(2, 1, 5)$
$I_2$ $S(0)$
$I_3$ $S(2)$
$I_4$ $J(0, 0, 1)$





# Un programma per la macchina URM

## Esecuzione

Supponiamo di voler calcolare $f(5, 2) = 7$. La macchina URM parte da:

| $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $\ldots$ |
|---|---|---|---|---|---|
| 5 | 2 | 0 | 0 | 0 | $\ldots$ |

Vediamo la sequenza di stati dopo ogni istruzione:

| Istr. | $R_0$ | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $\ldots$ |
|---|---|---|---|---|---|---|
| $I_1$ | 5 | 2 | 0 | 0 | 0 | $\ldots$ |
| $I_2$ | 6 | 2 | 0 | 0 | 0 | $\ldots$ |
| $I_3$ | 6 | 2 | 1 | 0 | 0 | $\ldots$ |
| $I_4$ | 6 | 2 | 1 | 0 | 0 | $\ldots$ |
| $I_1$ | 6 | 2 | 1 | 0 | 0 | $\ldots$ |
| $I_2$ | 7 | 2 | 1 | 0 | 0 | $\ldots$ |
| $I_3$ | 7 | 2 | 2 | 0 | 0 | $\ldots$ |
| $I_4$ | 7 | 2 | 2 | 0 | 0 | $\ldots$ |
| $I_1$ | 7 | 2 | 2 | 0 | 0 | $\ldots$ |

Programma:

$I_1$ $J(2, 1, 5)$
$I_2$ $S(0)$
$I_3$ $S(2)$
$I_4$ $J(0, 0, 1)$

Al termine del calcolo l'URM è in uno stato finale con $R_0$ uguale a 7.

# Funzioni calcolabili dalla macchina URM

Nell'esempio precedente abbiamo dimostrato che la funzione $f(x, y) = x + y$ è calcolabile.

Si possono realizzare programmi per calcolare la differenza, il prodotto e il quoziente tra due numeri interi, nonché molte altre funzioni anche con domini differenti da quello considerato.

In questa sede non indagheremo oltre sulle capacità della macchina URM, ma è possibile dimostrare che l'insieme delle funzioni URM–calcolabili è lo stesso delle funzioni calcolabili con la macchina di Turing.





# Esercizi

- Verificate quanti byte di memoria ha il vostro telefonino e confrontateli con quelli presenti nella memoria del calcolatore M20 dell'Olivetti. Quante volte è più grande la memoria del telefonino?
- Quanti bit sono necessari per codificare ogni giorno di un anno?
- Si stima che gli atomi dell'universo conosciuto siano $10^{80}$. Di quanti bit avremmo bisogno per codificare ciascuno di essi?
- Scrivere un programma URM per raddoppiare il valore nel primo registro.
- Scrivere un programma URM per calcolare la differenza tra il valore nel primo registro e quello nel secondo (assumendo che il primo valore sia più grande del secondo). Al termine, il risultato deve essere posto nel primo registro.
- Scrivere un programma URM che restuisca 1 se il contenuto del primo registro è maggiore di quello nel secondo registro e che restituisca 0 altrimenti. Il risultato deve essere scritto nel primo registro.