Cenni sull'Architettura degli Elaboratori

Eugenio G. Omodeo

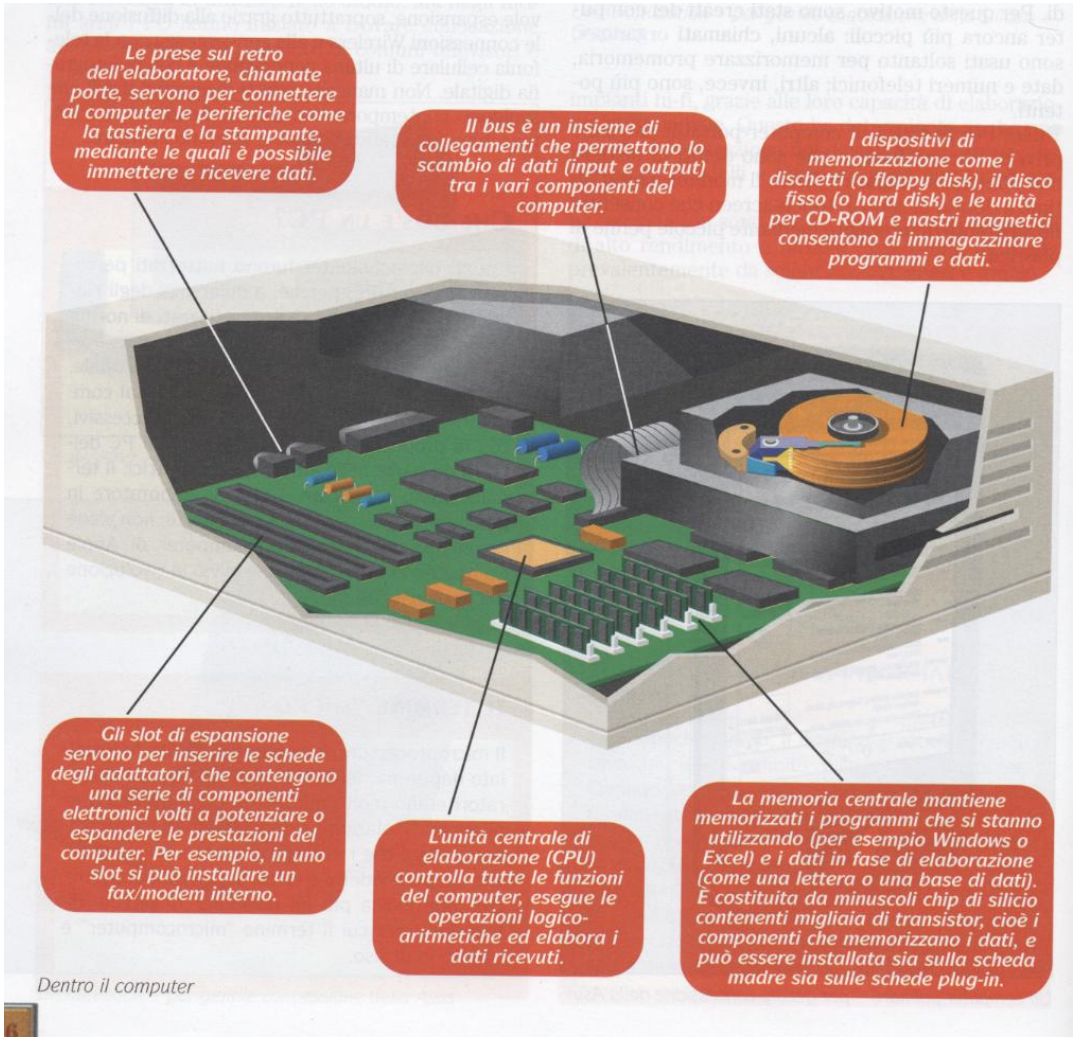
UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Dip. Matematica e Geoscienze — DMI

Trieste, 08/10/2015

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

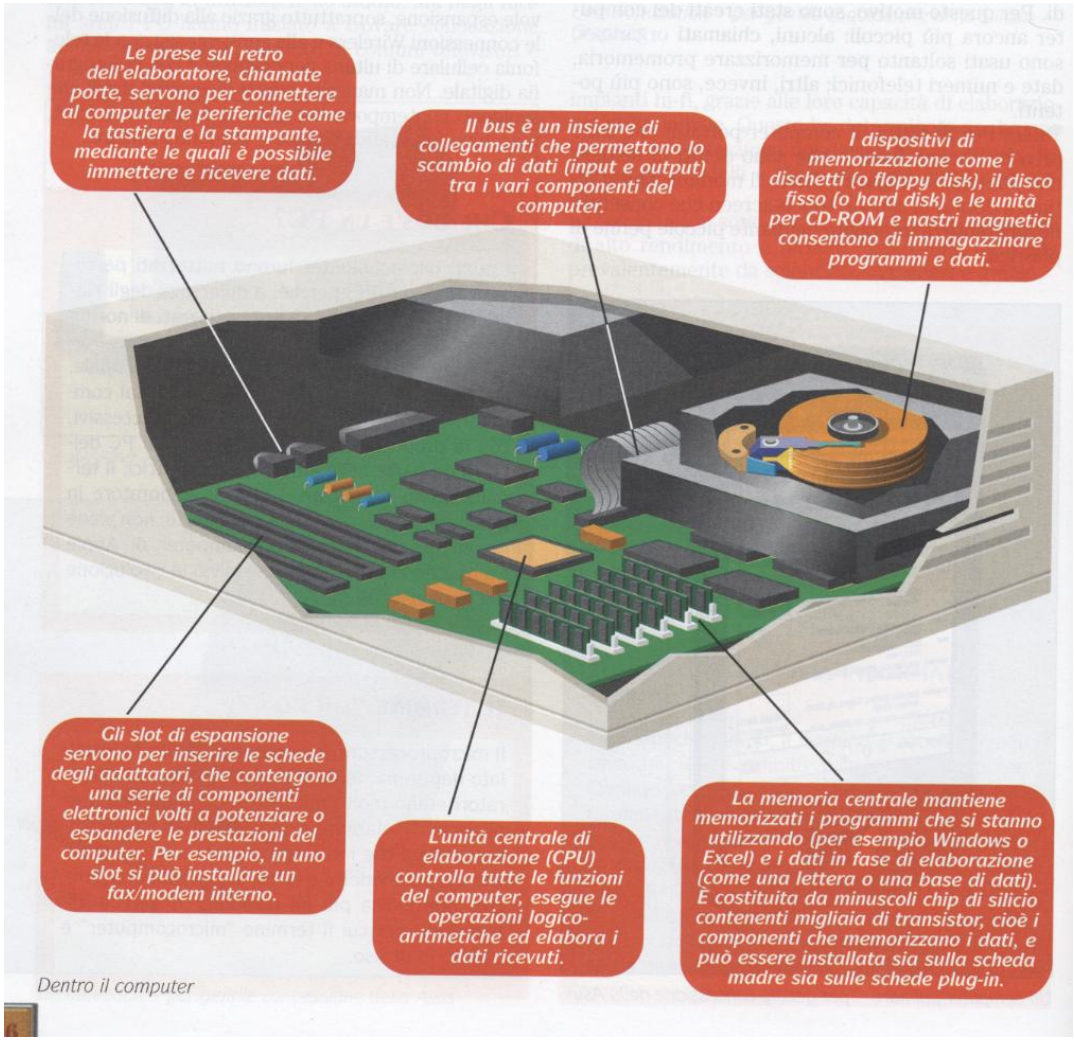
Interno di un calcolatore *desktop*



Dentro il computer

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Interno di un calcolatore *desktop*

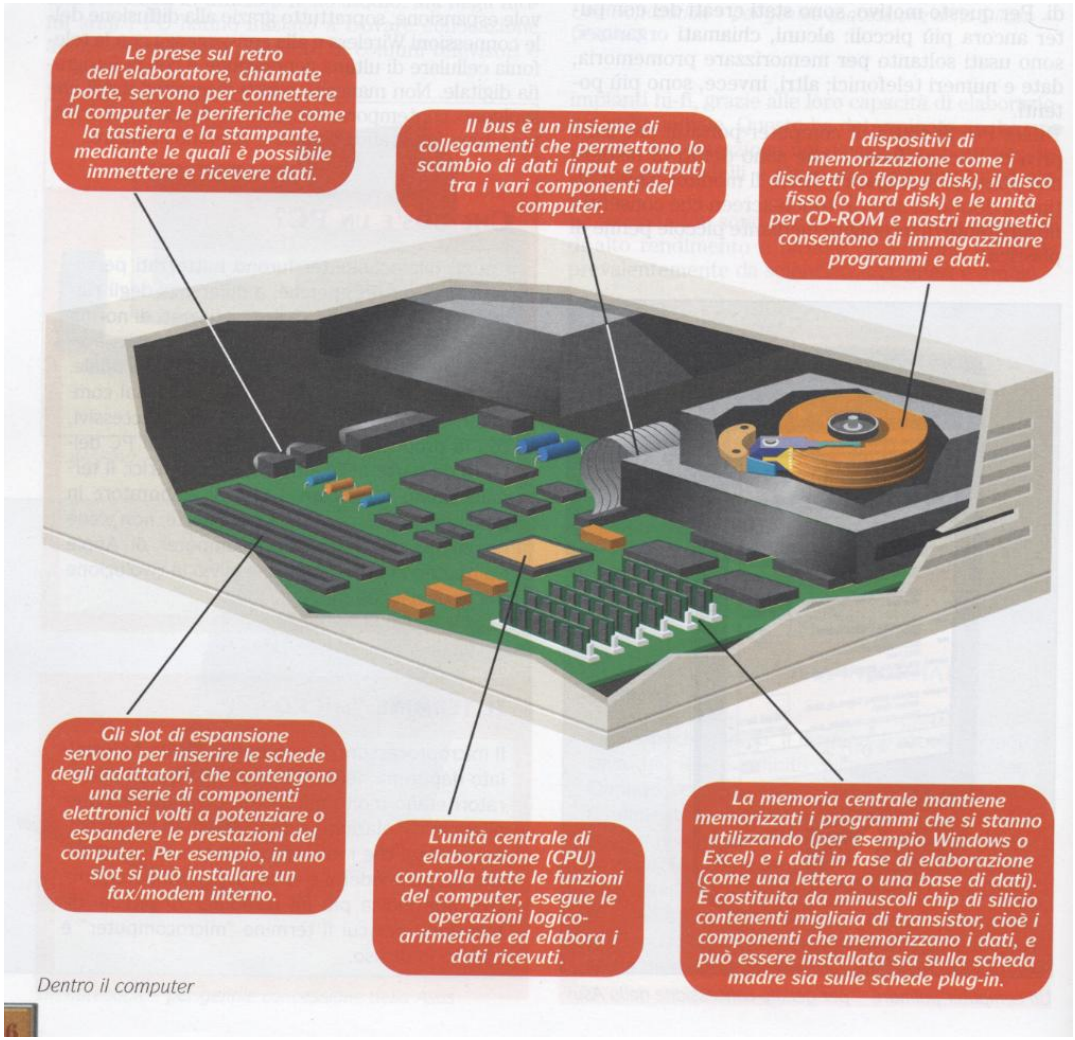


Dentro il computer

(In che differisce, questo, da un *laptop* ?)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Interno di un calcolatore *desktop*

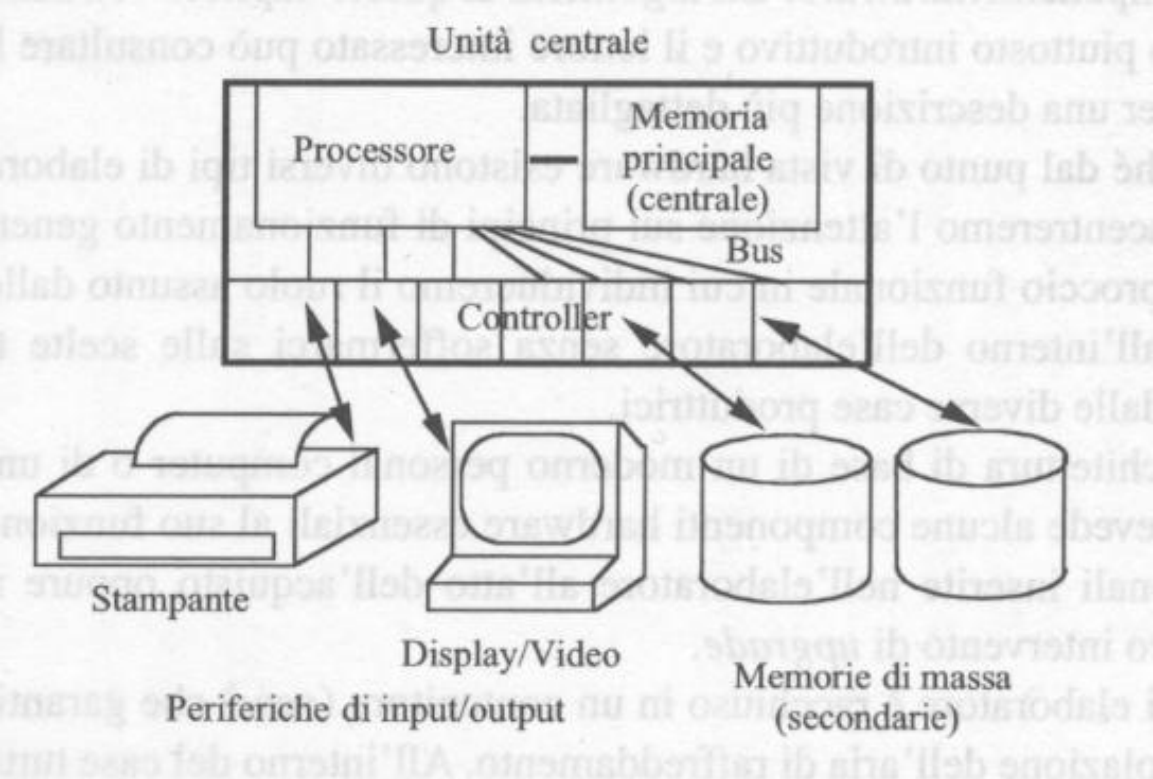


Dentro il computer

(Il disco è un HD o un SSD ?)

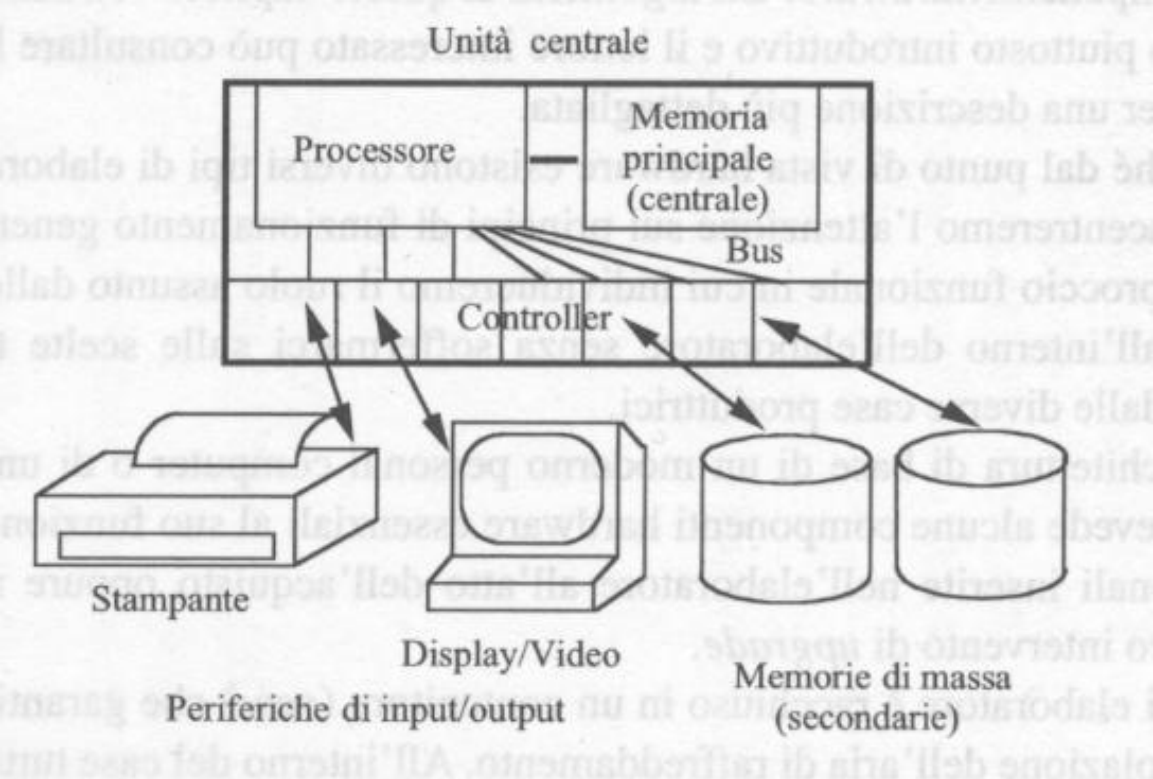
UNIVERSITÀ DEGLI STUDI DI TRIESTE

Altro ritratto di un *computer*



UNIVERSITÀ DEGLI STUDI DI TRIESTE

Altro ritratto di un *computer*



(Come individuare ciò che davvero caratterizza un computer ?)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Una definizione rigorosa

Calcolatore $=_{\text{Def}}$ dispositivo elettronico veloce che accetta in ingresso informazione digitalizzata, la elabora in base a una lista (detta *programma*) di istruzioni memorizzate al suo interno e fornisce in uscita l'informazione risultante.

V. C. Hamacher, Z. G. Vrasenic, S. G. Zaky (2001)

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Una definizione rigorosa ???

Calcolatore $=_{\text{Def}}$ dispositivo elettronico veloce che accetta in ingresso informazione digitalizzata, la elabora in base a una lista (detta *programma*) di istruzioni memorizzate al suo interno e fornisce in uscita l'informazione risultante.

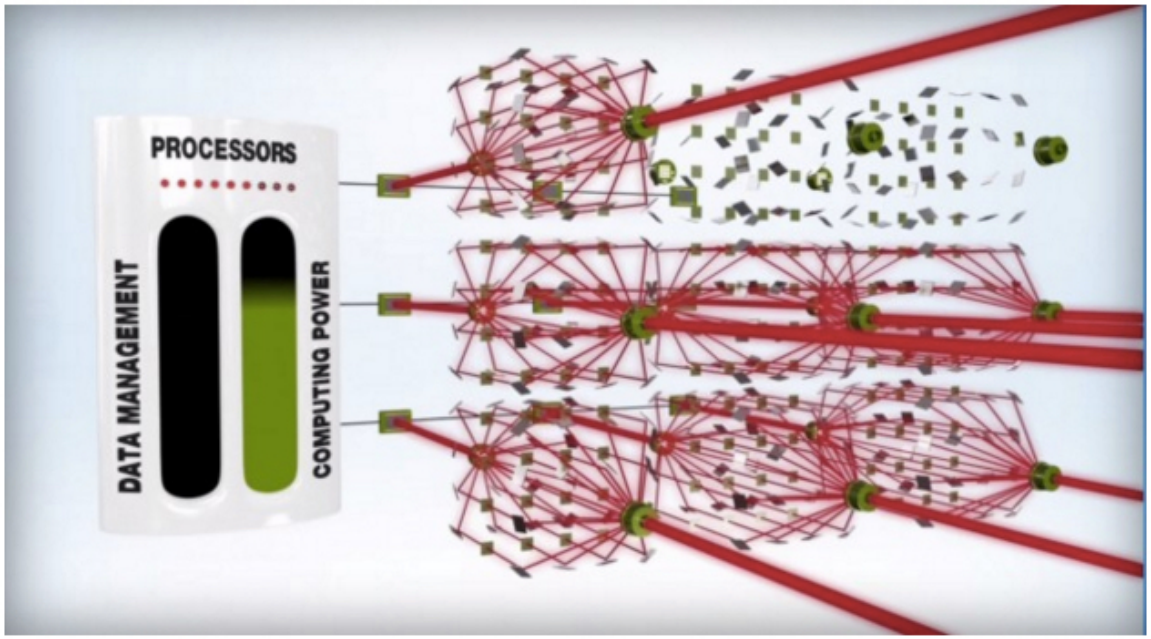
V. C. Hamacher, Z. G. Vrasenic, S. G. Zaky (2001)

Dubbio: Quanto è cruciale l'elettronica ?

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

By 2020, you could have an exascale speed-of-light optical computer on your desk

By Sebastian Anthony on August 8, 2014 at 1:29 pm | 62 Comments



UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Breve digressione

WIKIPEDIA
The Free Encyclopedia

Optical or **photonic computing** uses photons produced by lasers or diodes for computation. For decades, photons have promised to allow a higher bandwidth than the electrons used in conventional computers.

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Flashback: Già nel XIX secolo...

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Flashback: Già nel XIX secolo...

A LETTER

TO

SIR HUMPHRY DAVY, BART.

PRESIDENT OF THE ROYAL SOCIETY, ETC. ETC.

ON THE APPLICATION OF MACHINERY

TO THE PURPOSE OF

Calculating and Printing Mathematical Tables,

FROM

CHARLES BABBAGE, ESQ. M.A.

F.R.S. LOND. AND EDIN., MEMBER OF THE CAMBRIDGE PHILOSOPHICAL SOCIETY,
SECRETARY OF THE ASTRONOMICAL SOCIETY OF LONDON, AND
CORRESPONDENT OF THE PHILOMATHIC SOCIETY OF PARIS.

LONDON:

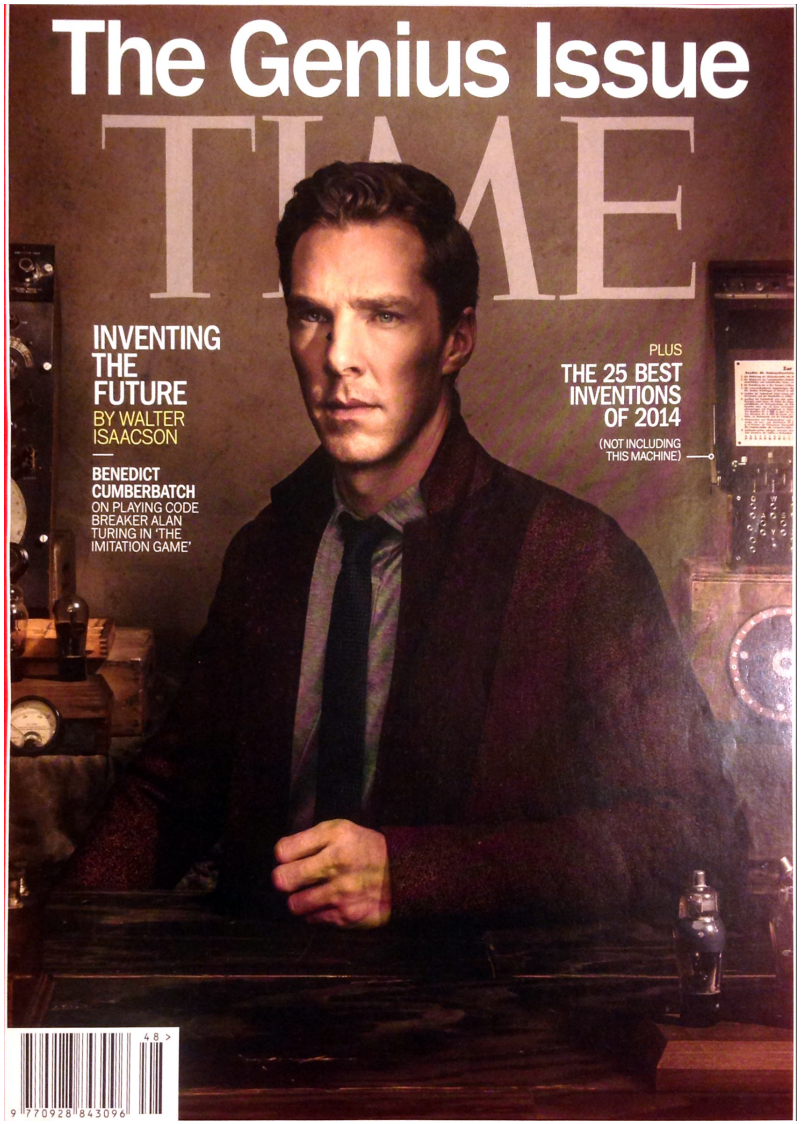
PRINTED BY R. AND A. TAYLOR, SHOE-LANE:

PUBLISHED BY J. BOOTH, DUKE-STREET, PORTLAND-PLACE; AND BALDWIN,
CRADOCK, AND JOY, PATERNOSTER-ROW.

1822.

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Flashback: . . . e poi, nel XX secolo. . .



UNIVERSITÀ DEGLI STUDI DI TRIESTE

Un articolo che segna una svolta epocale

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING

[Received 28 May, 1936.—Read 12 November, 1936.]

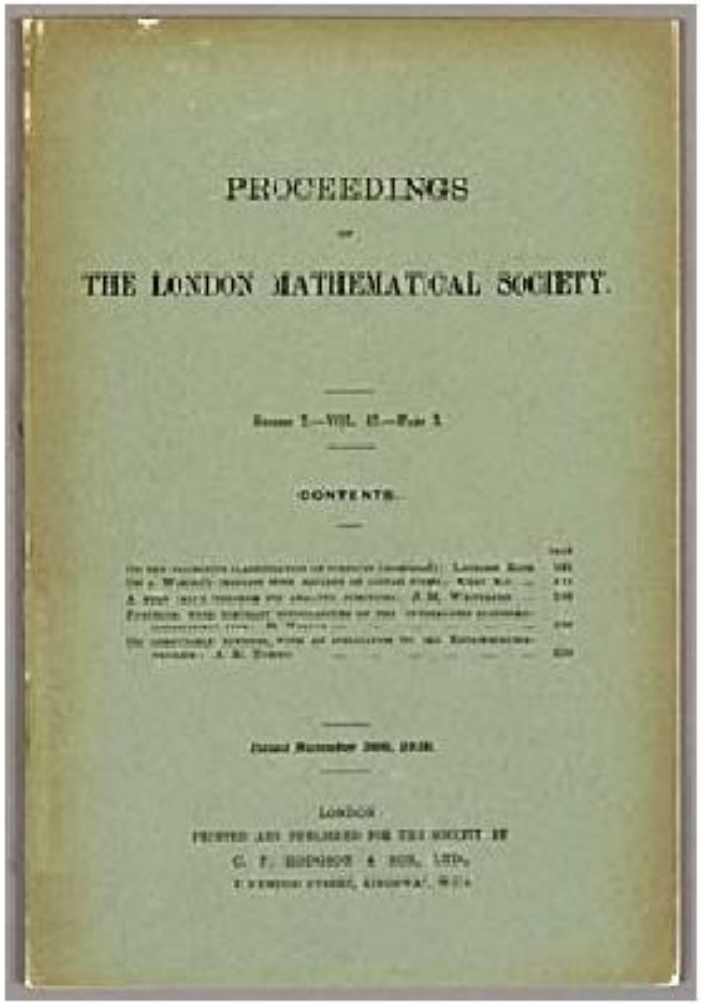
1. Computing machines.
2. Definitions.

Automatic machines.
Computing machines.
Circle and circle-free numbers.
Computable sequences and numbers.

3. Examples of computing machines.
4. Abbreviated tables

Further examples.

5. Enumeration of computable sequences.
6. The universal computing machine.
7. Detailed description of the universal machine.
8. Application of the diagonal process.
9. The extent of the computable numbers.
10. Examples of large classes of numbers which are computable.
11. Application to the Entscheidungsproblem.



PROCEEDINGS

OF

THE LONDON MATHEMATICAL SOCIETY.

CONTENTS.

Esplicitazione del concetto di *calcolabilità*

Tesi di Church–Turing

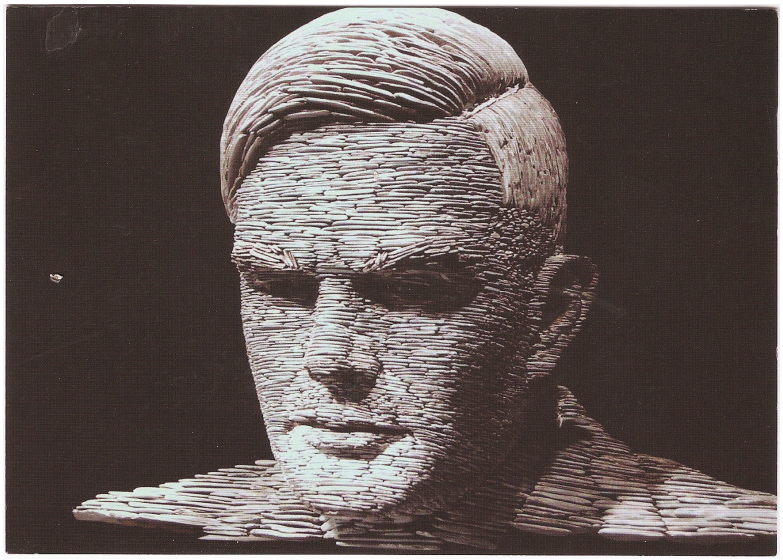
Varie nozioni matematiche di *computabilità*, autorevolmente proposte e ciascuna equipollente alle altre, catturano il concetto intuitivo di *calcolabilità*.

Esplicitazione del concetto di *calcolabilità*

Tesi di Church–Turing

Varie nozioni matematiche di *computabilità*, autorevolmente proposte e ciascuna equipollente alle altre, catturano il concetto intuitivo di *calcolabilità*.

Alonzo Church (1903–1995)



Alan Mathison Turing (1912–1954)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Esplicitazione del concetto di *universalità*

"It is possible to invent a single machine which can be used to compute any computable sequence. If this machine I is supplied with a tape on the beginning of which is written the $S.D$ of some computing machine M, then I will compute the same sequence as M. In this section I explain in outline the behavior of the machine. The next section is devoted to giving the complete table for I."

Alan Mathison Turing, 1936

UNIVERSITÀ DEGLI STUDI DI TRIESTE

The imitation game

"It is possible to invent a single machine which can be used to compute any computable sequence. If this machine I is supplied with a tape on the beginning of which is written the $S.D$ of some computing machine M, then I will compute the same sequence as M. In this section I explain in outline the behavior of the machine. The next section is devoted to giving the complete table for I."

Alan Mathison Turing, 1936

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Ora qualche conclusione circa il calcolatore

1. la *programmabilità* è caratteristica essenziale del calcolatore
   (= '*general purpose computer*')

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Ora qualche conclusione circa il calcolatore

1. la *programmabilità* è caratteristica essenziale del calcolatore (= '*general purpose computer*')
2. per quanto cruciale nella sua realizzazione, l'*elettronica* non dovrebbe far parte della definizione

Ora qualche conclusione circa il calcolatore

1. la *programmabilità* è caratteristica essenziale del calcolatore (= '*general purpose computer*')
2. per quanto cruciale nella sua realizzazione, l'*elettronica* non dovrebbe far parte della definizione
3. per vari aspetti metodologici, l'informatica (o '*computer science*') è in debito con la logica, in particolare con l'*Entscheidungsproblem*

- L'aspetto *hardware* del calcolatore è rappresentato dai circuiti elettronici ed elettromeccanici che lo compongono;

V. C. Hamacher, Z. G. Vrasenic, S. G. Zaky (2001)

1

- L'aspetto *hardware* del calcolatore è rappresentato dai circuiti elettronici ed elettromeccanici che lo compongono;
- l'*architettura* del calcolatore, invece, è definita come la combinazione delle funzionalità operative delle singole unità hardware che costituiscono il sistema di calcolo, il flusso di informazioni tra queste unità e il relativo controllo.

V. C. Hamacher, Z. G. Vrasenic, S. G. Zaky (2001)

[1]

- L'aspetto *hardware* del calcolatore è rappresentato dai circuiti elettronici ed elettromeccanici che lo compongono;
- l'*architettura* del calcolatore, invece, è definita come la combinazione delle funzionalità operative delle singole unità hardware che costituiscono il sistema di calcolo, il flusso di informazioni tra queste unità e il relativo controllo.

V. C. Hamacher, Z. G. Vrasenic, S. G. Zaky (2001)

E il software[1] ?

[1] Ossia i programmi...

HW / SW

"Al tempo dei primi calcolatori la distinzione fra hardware e software era chiarissima. Nel tempo si è piuttosto confusa ···

Hardware e software sono logicamente equivalenti.

si prevede che la funzione cambi."

A. S. Tenenbaum (2000)

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

HW / SW

"Al tempo dei primi calcolatori la distinzione fra hardware e software era chiarissima. Nel tempo si è piuttosto confusa ···

Hardware e software sono logicamente equivalenti.

Qualsiasi operazione venga effettuata dal software può essere direttamente inglobata nell'hardware:

L'hardware è software pietrificato.

si prevede che la funzione cambi."

A. S. Tenenbaum (2000)

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

HW / SW

"Al tempo dei primi calcolatori la distinzione fra hardware e software era chiarissima. Nel tempo si è piuttosto confusa ···

Hardware e software sono logicamente equivalenti.

Qualsiasi operazione venga effettuata dal software può essere direttamente inglobata nell'hardware:

L'hardware è software pietrificato.

Naturalmente è vero anche il contrario: qualsiasi istruzione eseguita dall'hardware può venir simulata dal software.

si prevede che la funzione cambi."

A. S. Tenenbaum (2000)

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

HW / SW

"Al tempo dei primi calcolatori la distinzione fra hardware e software era chiarissima. Nel tempo si è piuttosto confusa ···

Hardware e software sono logicamente equivalenti.

Qualsiasi operazione venga effettuata dal software può essere direttamente inglobata nell'hardware:

L'hardware è software pietrificato.

Naturalmente è vero anche il contrario: qualsiasi istruzione eseguita dall'hardware può venir simulata dal software. La decisione di inglobare certe funzioni nell'hardware e altre nel software si basa su fattori come costo, velocità, affidabilità, e frequenza con cui si prevede che la funzione cambi."

A. S. Tenenbaum (2000)

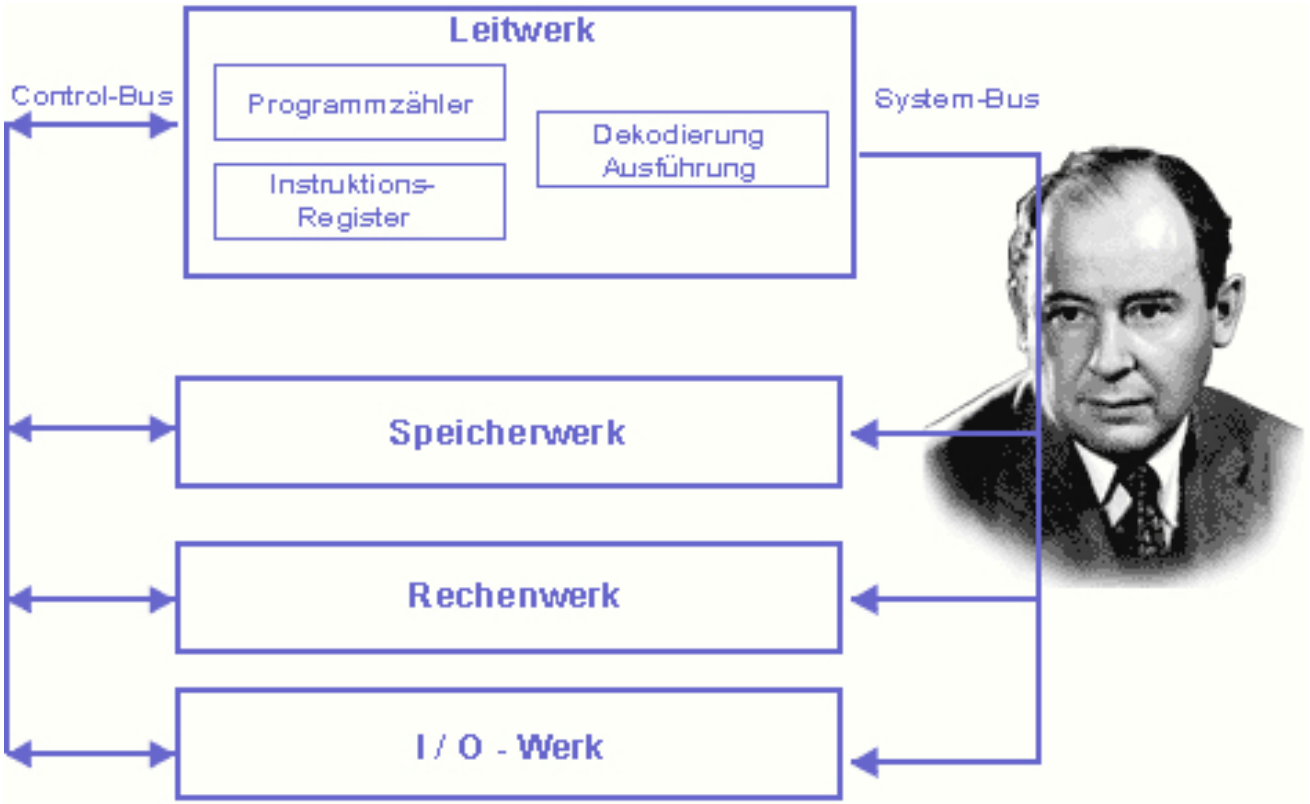
UNIVERSITÀ
DEGLI STUDI DI TRIESTE

A proposito di *architettura* del *calcolatore*

Considerando quanto è sfumata la distinzione fra HW e SW, piuttosto che di calcolatore (o '*computer*') qui sarebbe proprio parlare di sistema di elaborazione.

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Un'intramontabile organizzaz. dell'hardware

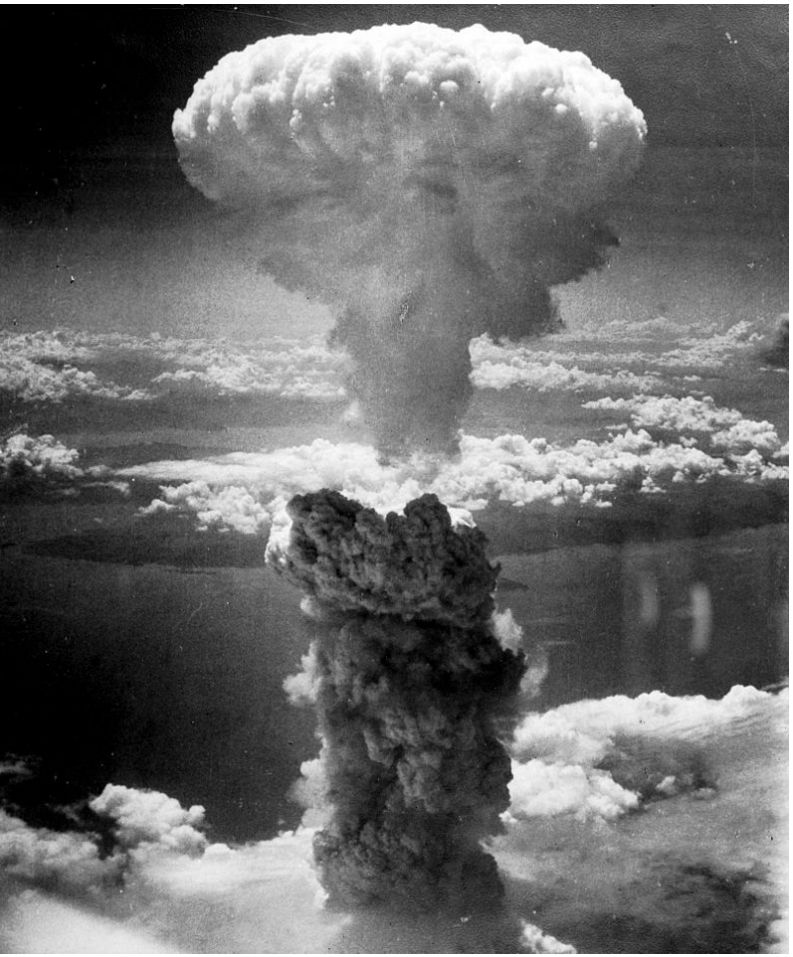


John Louis von Neumann, nato János Lajos Neumann (Budapest, 1903 – Washington, 1957), poliedrico "matematico" ungherese naturalizzato statunitense.

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

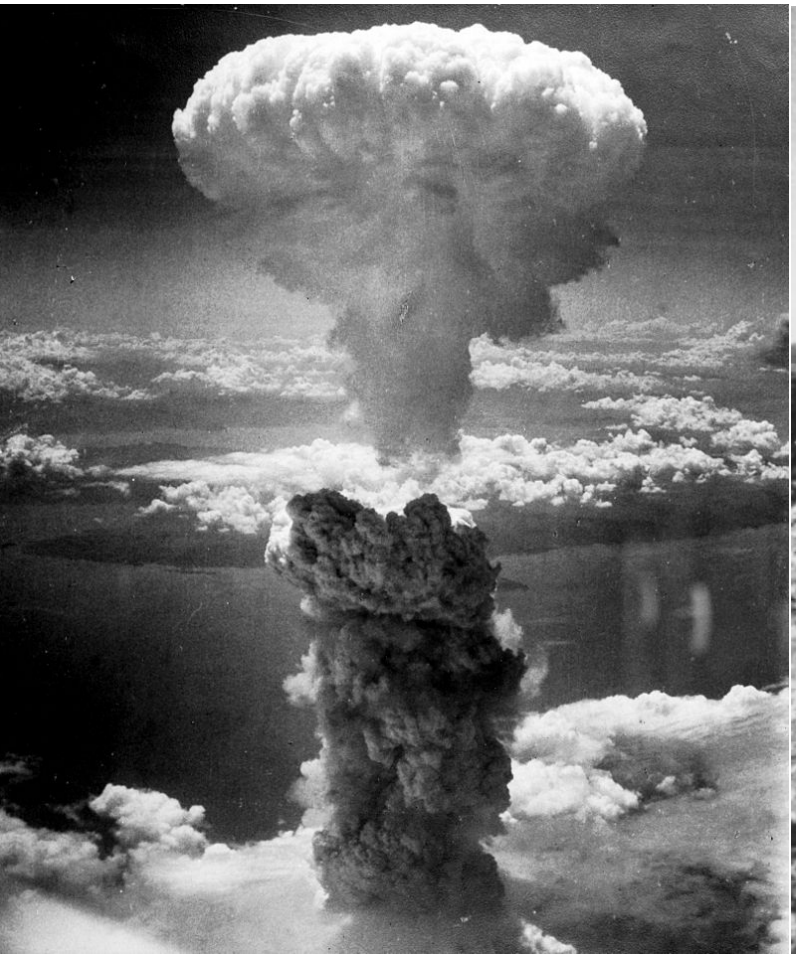
UNIVERSITÀ
DEGLI STUDI DI TRIESTE

(Il lato oscuro: 6 e 9 agosto 1945)



UNIVERSITÀ
DEGLI STUDI DI TRIESTE

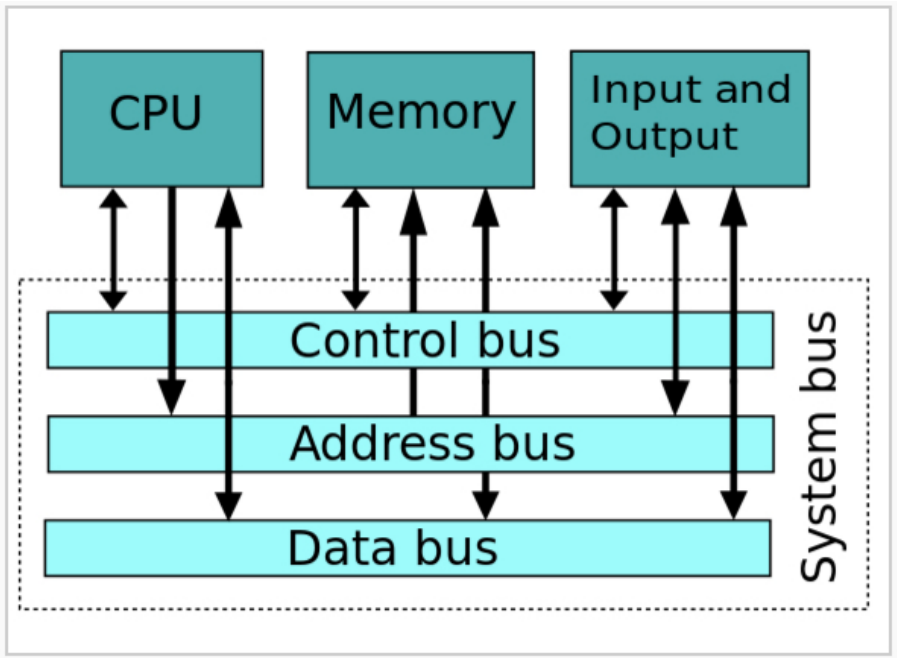
(Il lato oscuro: 6 e 9 agosto 1945)



Ubi solitudinem faciunt, pacem appellant (Tacito)

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

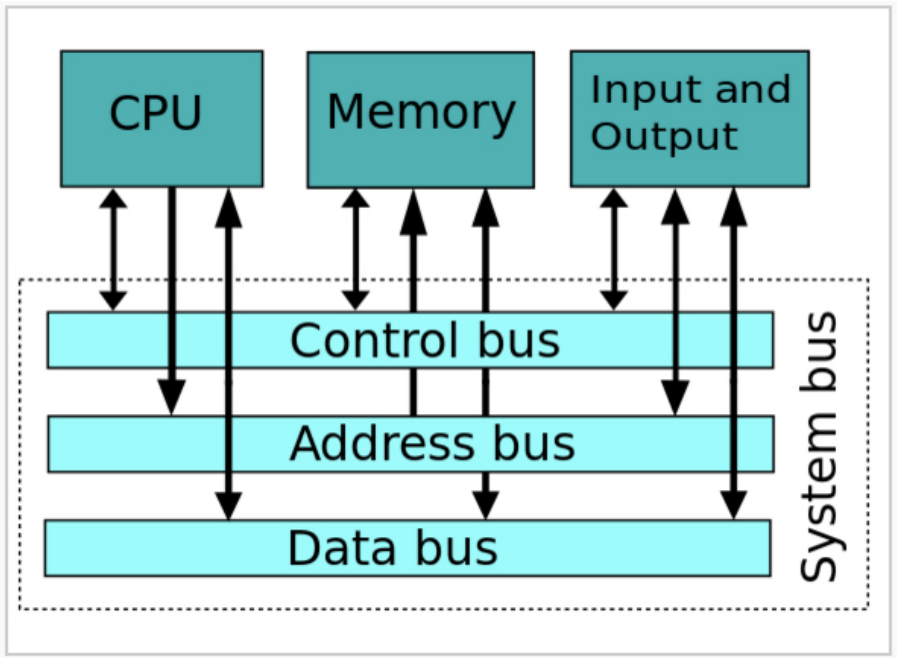
Schema dell'architettura di von Neumann



In senso orario:

1. Microprocessore

Schema dell'architettura di von Neumann

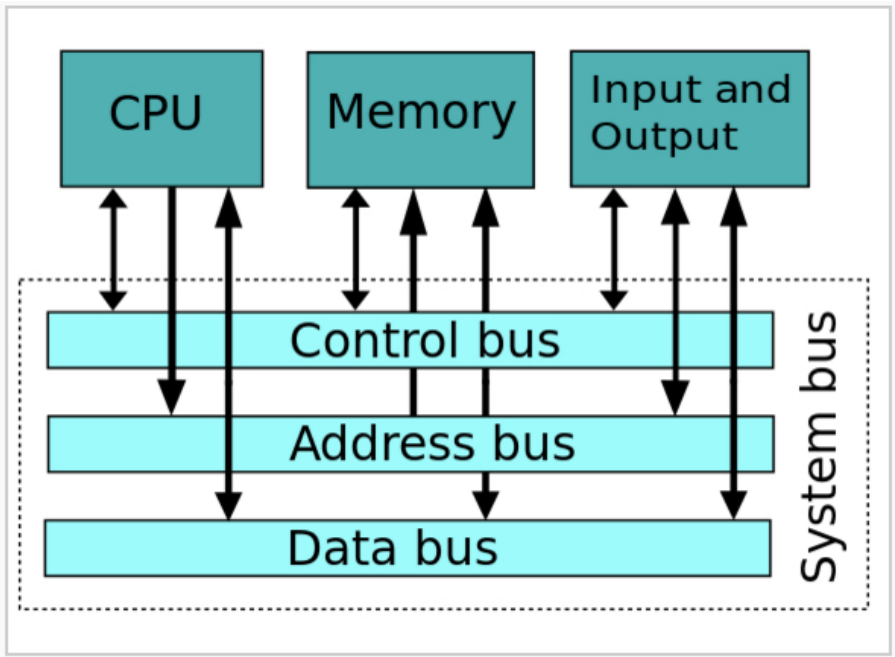


In senso orario:

1. Microprocessore
2. Memoria centrale

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Schema dell'architettura di von Neumann

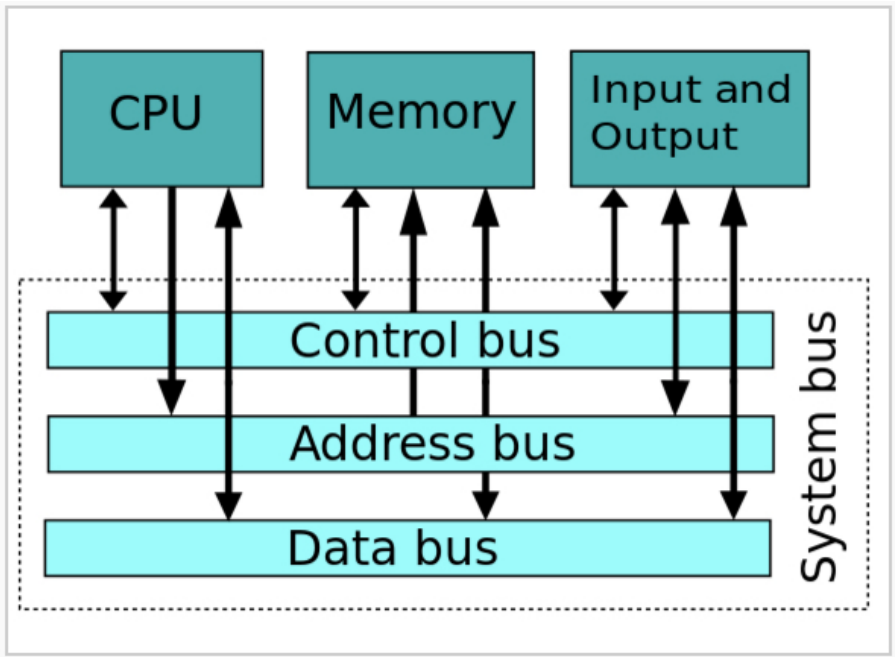


In senso orario:

1. Microprocessore
2. Memoria centrale
3. Interfacce con le periferiche

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Schema dell'architettura di von Neumann

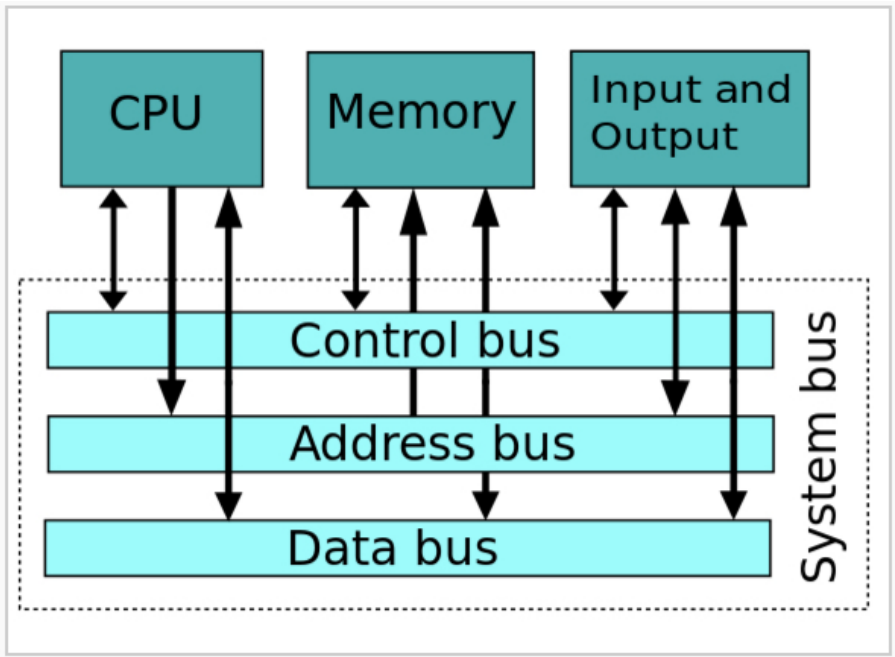


In senso orario:

1. Microprocessore
2. Memoria centrale
3. Interfacce con le periferiche
4. Bus di sistema (tripartito)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Schema dell'architettura di von Neumann



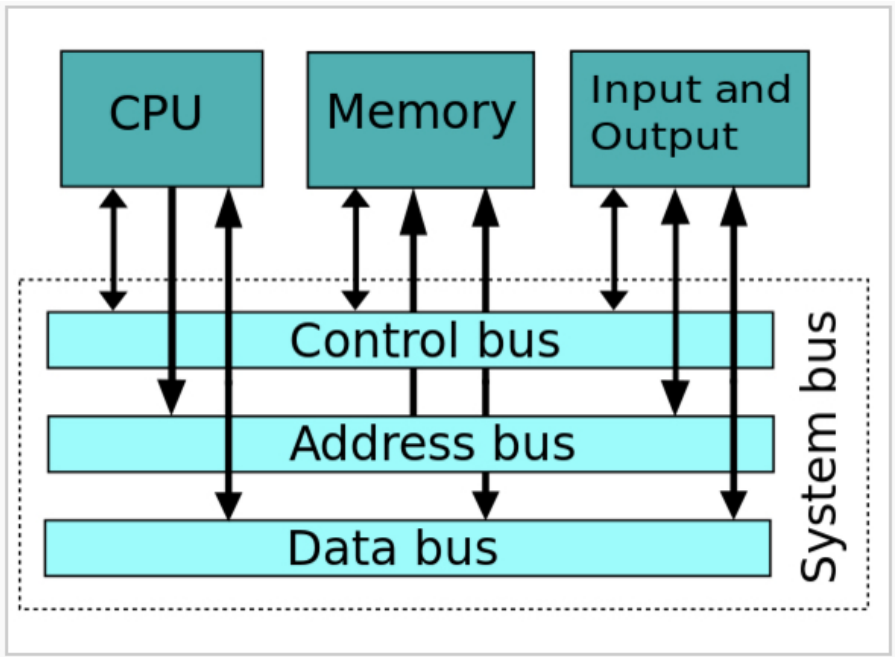
EDVAC

In senso orario:

1. Microprocessore
2. Memoria centrale
3. Interfacce con le periferiche
4. Bus di sistema (tripartito)

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Schema dell'architettura di von Neumann



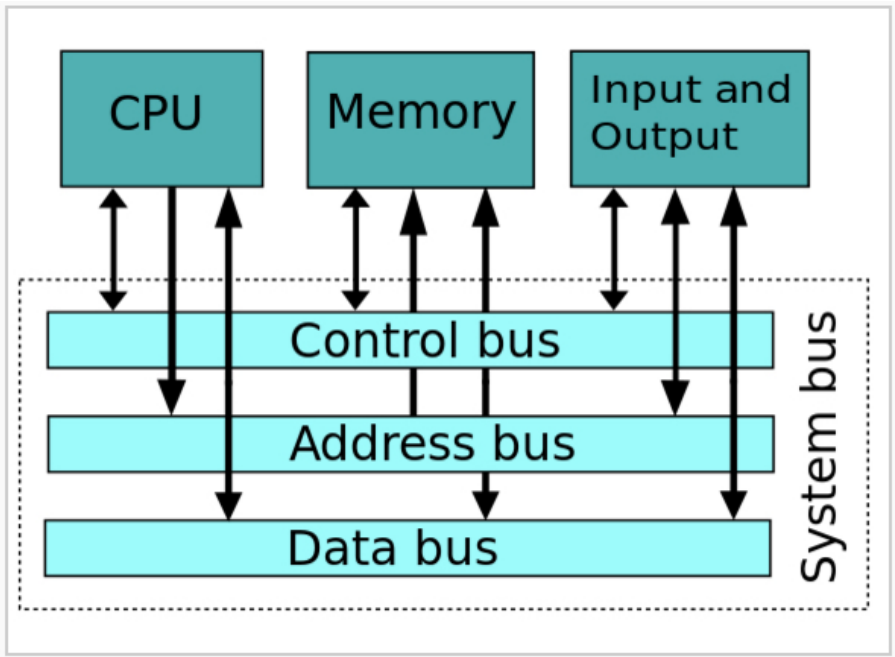
In senso orario:

1. Microprocessore
2. Memoria centrale
3. Interfacce con le periferiche
4. Bus di sistema (tripartito)

EDVAC IAS Machine

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Schema dell'architettura di von Neumann

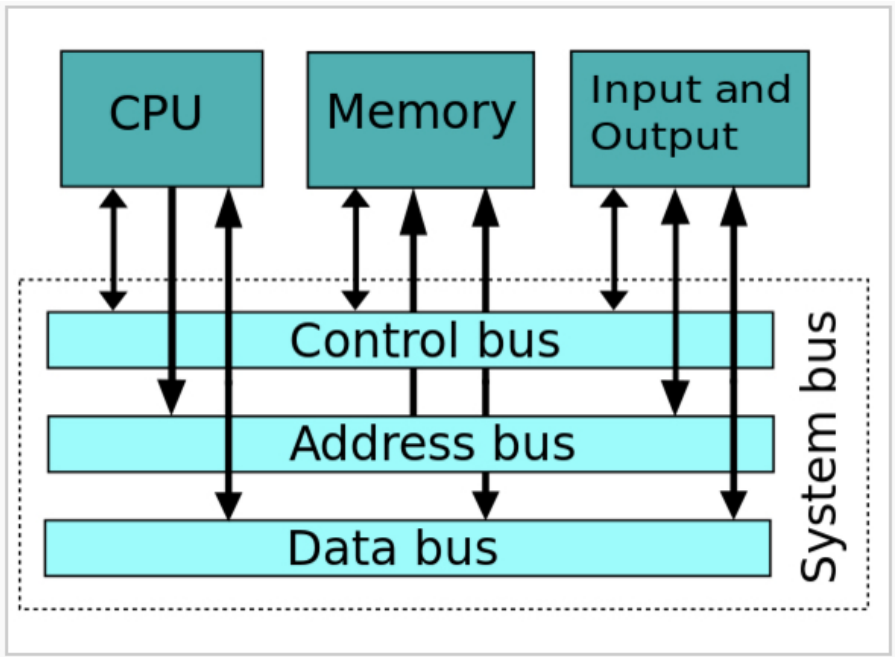


In senso orario:

1. Microprocessore
2. Memoria centrale
3. Interfacce con le periferiche
4. Bus di sistema (tripartito)

`EDVAC` `IAS Machine` `ACE` (?)

Schema dell'architettura di von Neumann



In senso orario:

1. Microprocessore
2. Memoria centrale
3. Interfacce con le periferiche
4. Bus di sistema (tripartito)

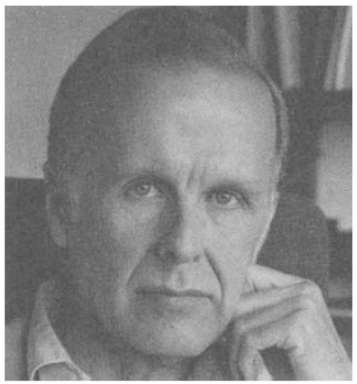
EDVAC IAS Machine ~~ACE~~ (?)

CISC vs RISC

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



Author's address: 91 Saint Germain Ave., San Francisco, CA 94114.

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

613

Communications of the ACM

August 1978
Volume 21
Number 8

INFORMATION PROCESSING 83, R.E.A. Mason (ed.)
Elsevier Science Publishers B.V. (North-Holland)

THE ARCHITECTURES IN THE FIFTH GENERATION COMPUTERS

Tohru MOTO-OKA
The University of Tokyo, Japan

Kazuhiro FUCHI
Institute for New Generation Computer Technology, Japan

Invited Paper

The fifth generation computers are being developed predominantly for use with knowledge information processing systems expected to come into wide spread utilization in the 1990s. Inference machines and relational algebra machines are typical of the core processors which constitute the fifth generation computer systems. A new logic programming language is being designed for use with the fifth generation kernel language to act as the interface between hardware and software in these machines. Several examples of proposed architectures are described. VLSI-oriented architecture and parallel processing control mechanisms are the main research topics.

1. INTRODUCTION

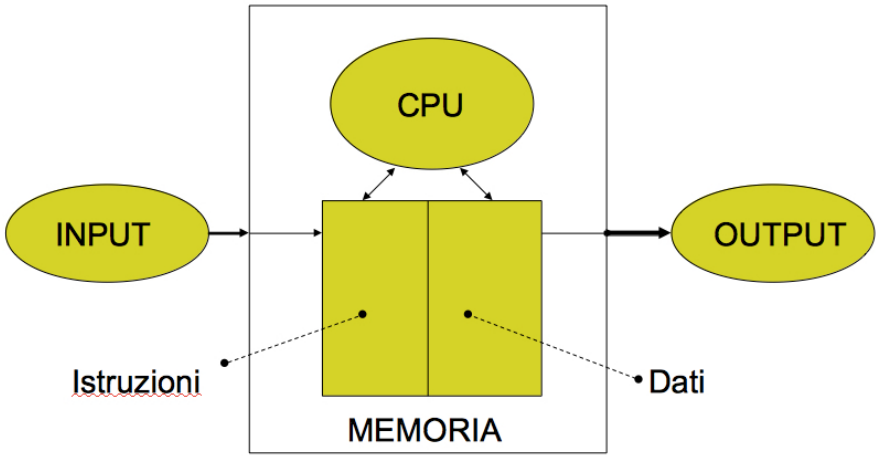
In the context of the Fifth Generation Computer Project, which is a project promoted on a national scale in Japan, a fifth generation computer is defined to be a computer for use in knowledge information processing with the widest range of applicability among the computers which will be produced in the 1990's[1][2].

Those types of computers expected to be in wide use in the 1990's include main frame computers whose chief aim will be to perform conventional business computations, as well as supercomputers for scientific calculations and computers which will serve as the components of systems such as intelligent robots. In this paper, however, we will adhere to the definition given

able; it is also necessary to develop technology for creating VLSI-oriented architectures which also enable high speed processing through the incorporation of parallel processing. To this end, we need to get away from the sequential control mechanism employed in conventional von Neumann-type computers, and instead employ control mechanisms which can incorporate parallel processing in a natural way -- e.g., data flow machines and reduction machines.

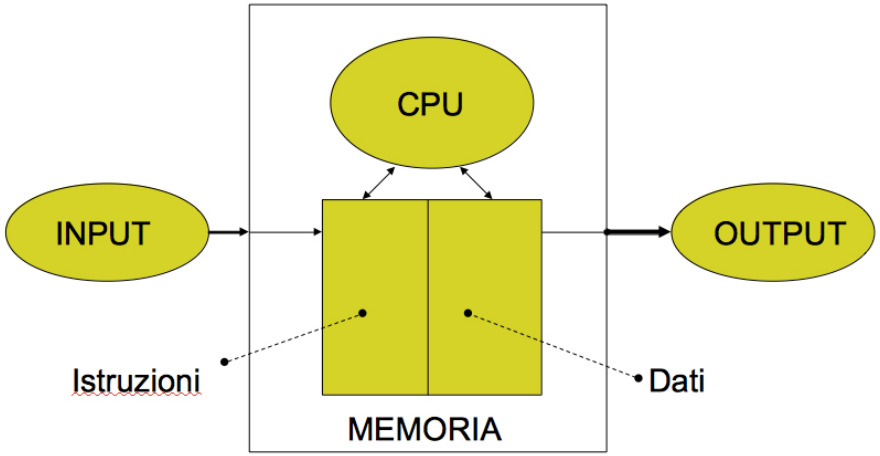
v) It should be possible to perform knowledge information processing at the users' sites as well as at processing centers where a large amount of knowledge has been accumulated. A knowledge information processing machine of the type referred to above should, therefore, be realized not only in the form of an ultra-large

Computer a programma memorizzato



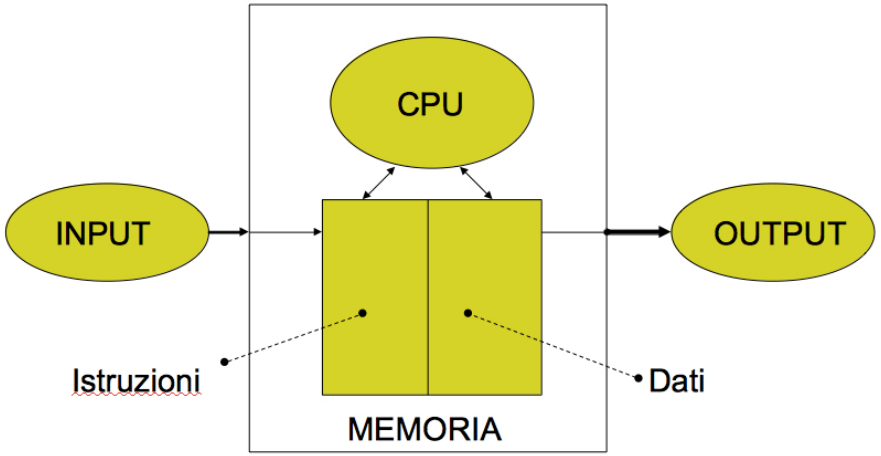
L'espressione "*stored-program computer*" ("computer a programma memorizzato") viene utilizata in riferimento alla memoria centrale: fu von Neumann a introdurla in *First draft of a report on the EDVAC*, datato 30 giugno 1945, con il significato particolare che gli attribuiamo oggi.

Computer a programma memorizzato



Pregio: Rapidità di accesso alle istruzioni.

Computer a programma memorizzato

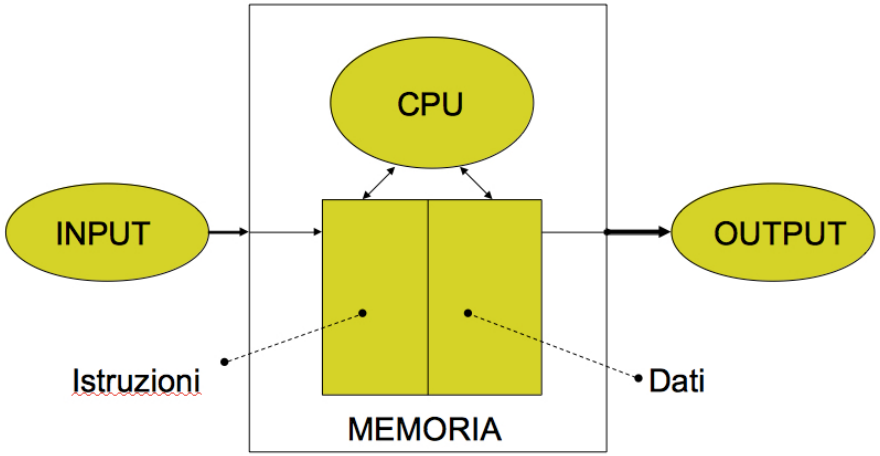


Pregio: Rapidità di accesso alle istruzioni.

Vulnerabilità: Possibilità che un programma, alterando le proprie istruzioni, pregiudichi il suo stesso funzionamento.

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Computer a programma memorizzato

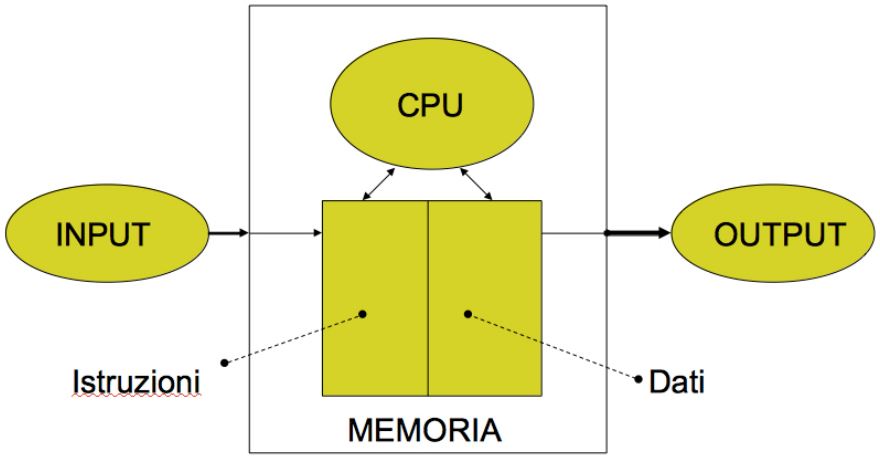


Pregio: Rapidità di accesso alle istruzioni.

Vulnerabilità: Possibilità che un programma, alterando le proprie istruzioni, pregiudichi il suo stesso funzionamento.

Pregio: Possibilità che un programma modifichi le proprie istruzioni, com'è giusto avvenga in un processo di apprendimento.

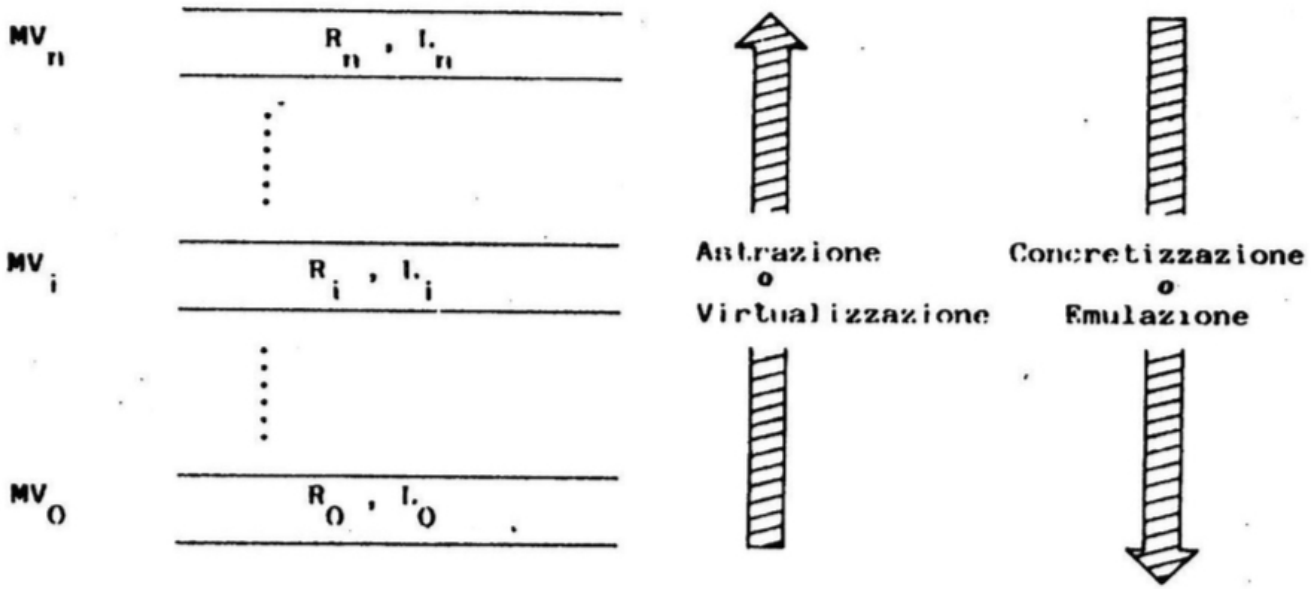
Computer a programma memorizzato



Dopo il *Colossus Mark I* del 1943, il *Colossus Mark II* del 1944 e l'ENIAC del 1946, viene realizzato nel 1948 a Manchester, UK, lo *Small-Scale Experimental Machine*, primo computer elettronico a programma memorizzato della storia. A partire dal 1948 il computer a programma memorizzato si diffonde, diventando in breve tempo la norma per il computer programmabile.

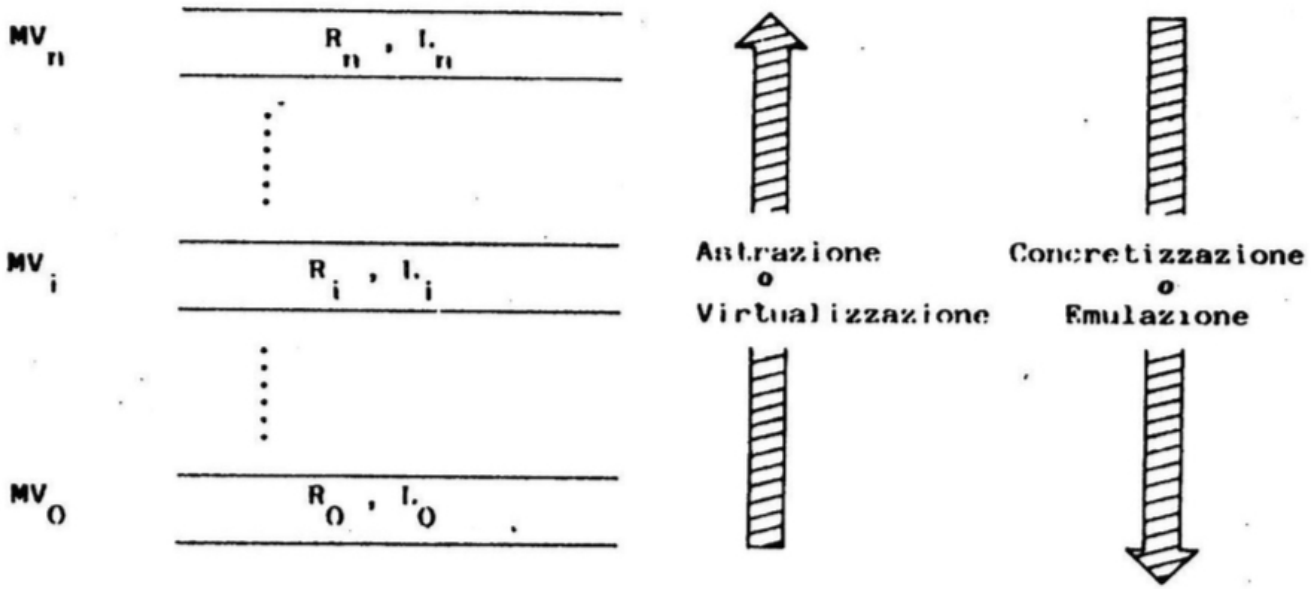
Dalle componenti a un'architettura

Le funzionalità di un sistema di elaborazione nel suo complesso possono essere viste come ripartite, secondo un certo numero di livelli, o **macchine virtuali** $\{MV_0, MV_1, \ldots, MV_n\}$, come schematizzato in figura:



Dalle componenti a un'architettura

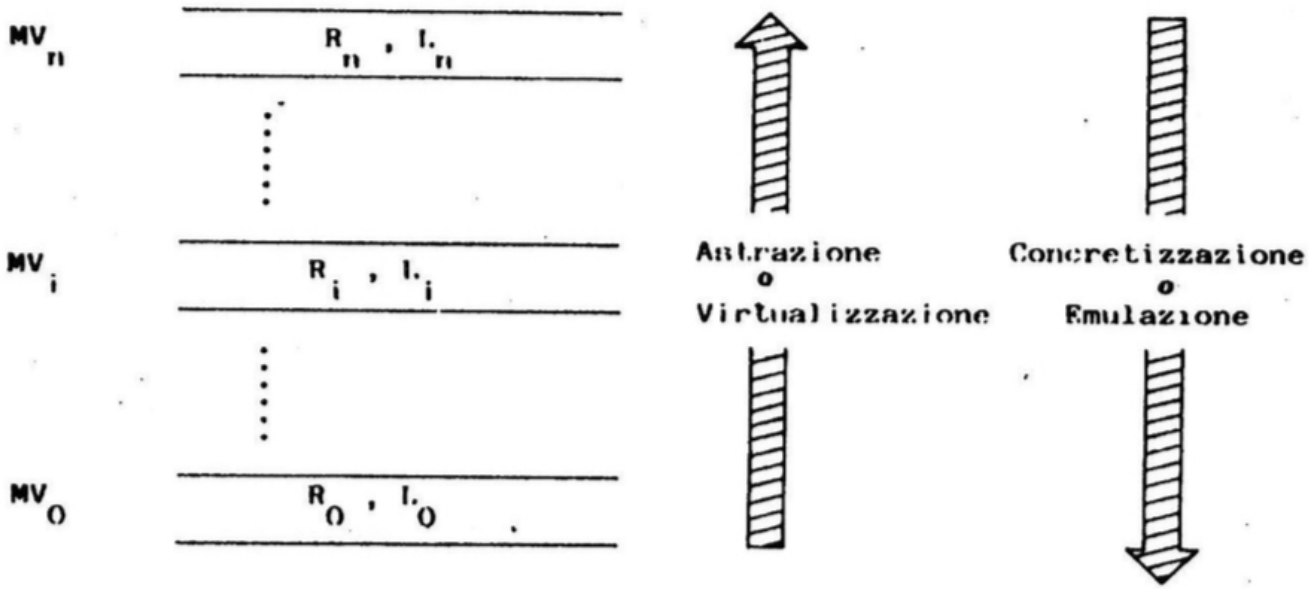
Le funzionalità di un sistema di elaborazione nel suo complesso possono essere viste come ripartite, secondo un certo numero di livelli, o **macchine virtuali** $\{MV_0, MV_1, \ldots, MV_n\}$, come schematizzato in figura:



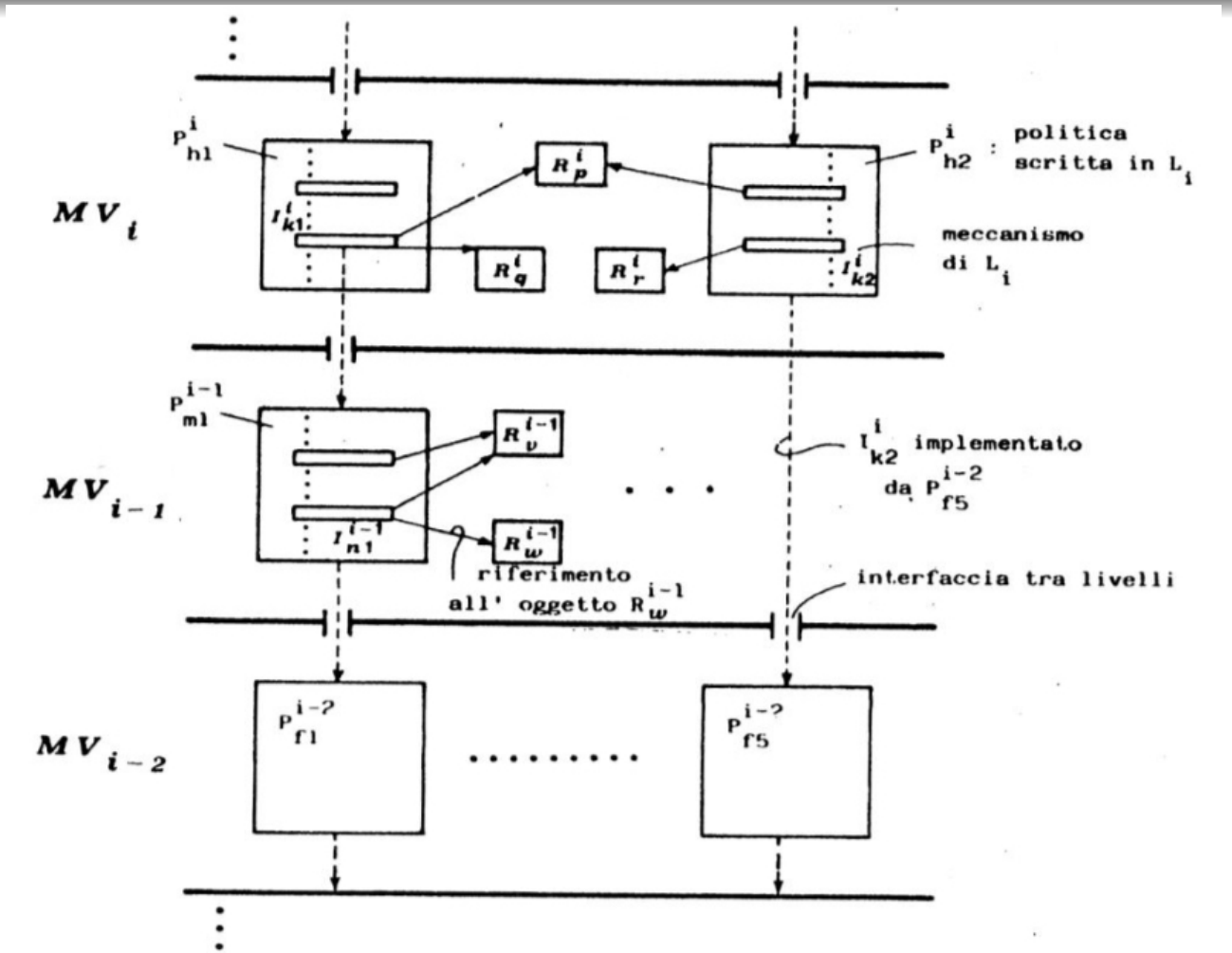
Il livello MV_0 è tipicamente quello dei componenti fisici (elettronici) a partire dai quali viene costruita una **astrazione**, o **virtualizzazione**, sempre maggiore delle funzionalità di interesse per il sistema e degli oggetti su cui esso è destinato ad operare. Il livello MV_n è quello che possiamo chiamare delle "applicazioni" in quanto fornisce la visione, di volta in volta ritenuta più opportuna degli oggetti e degli e degli strumenti mediante i quali il sistema viene utilizzato dall'esterno. Questo livello necessita normalmente, per potere essere effettivamente implementato, di un procedimento di **concretizzazione**, o **emulazione** ottenuto mediante uno o più livelli intermedi $\{MV_j \mid j = 1, \ldots, n\text{-}1\}$.

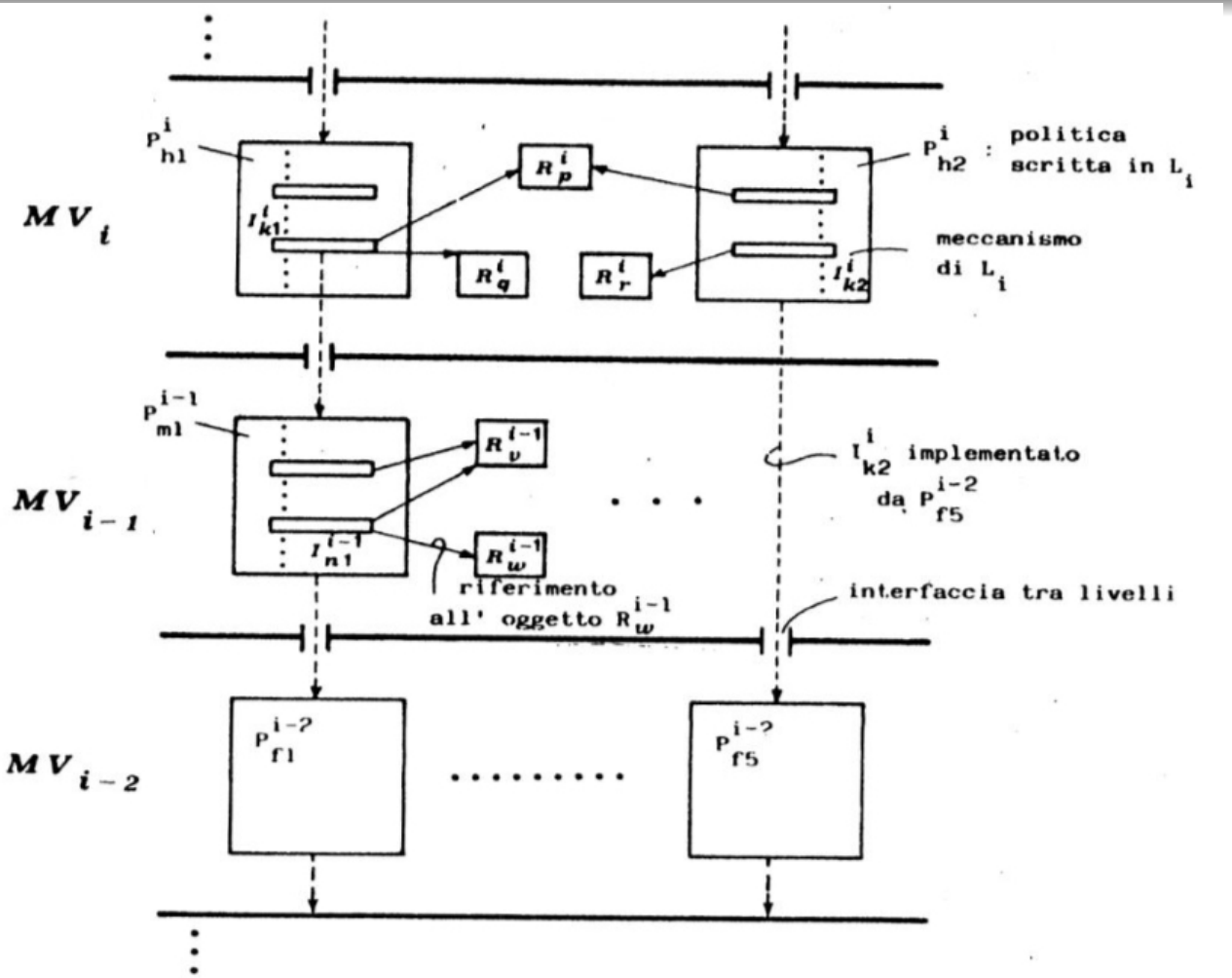
Dalle componenti a un'architettura

Le funzionalità di un sistema di elaborazione nel suo complesso possono essere viste come ripartite, secondo un certo numero di livelli, o **macchine virtuali** $\{MV_0, MV_1, \ldots, MV_n\}$, come schematizzato in figura:

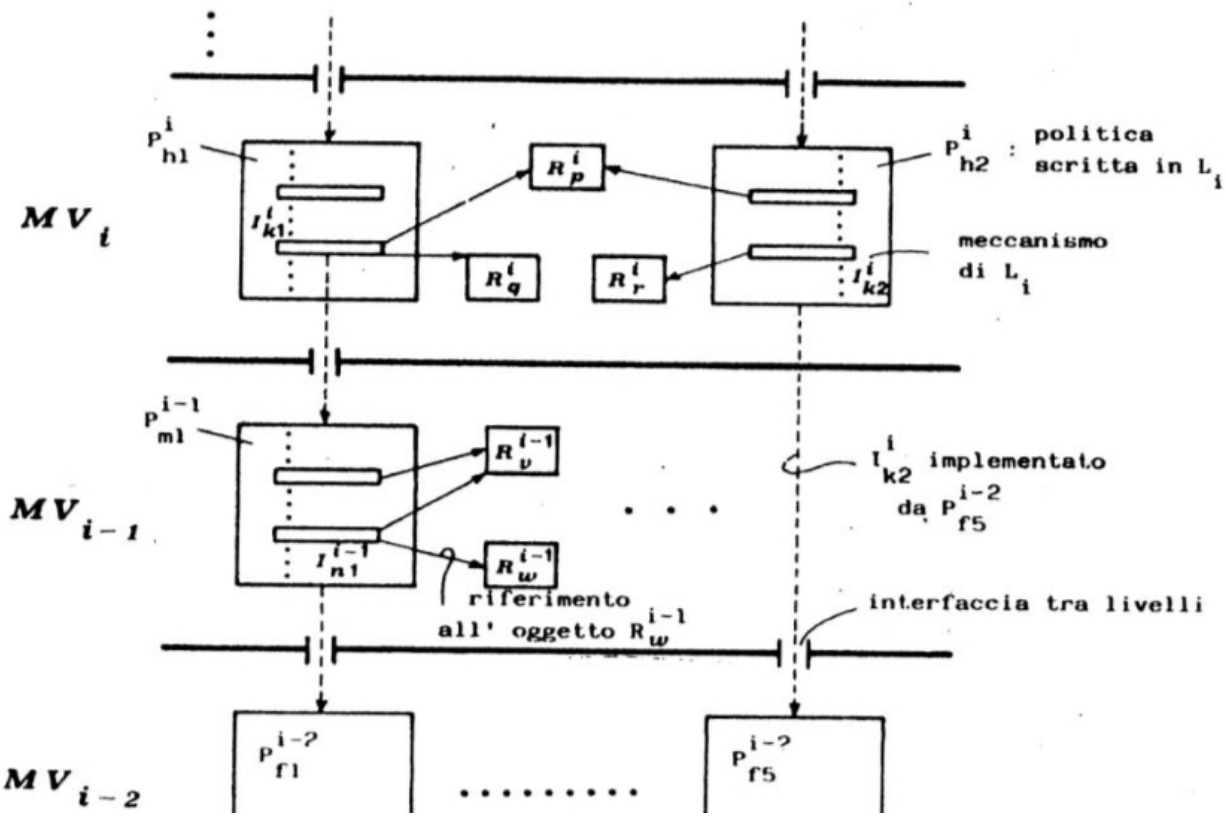


Il livello MV_0 è tipicamente quello dei componenti fisici (elettronici) a partire dai quali viene costruita una **astrazione**, o **virtualizzazione**, sempre maggiore delle funzionalità di interesse per il sistema e degli oggetti su cui esso è destinato ad operare. Il livello MV_n è quello che possiamo chiamare delle "applicazioni" in quanto fornisce la visione, di volta in volta ritenuta più opportuna degli oggetti e degli e degli strumenti mediante i quali il sistema viene utilizzato dall'esterno. Questo livello necessita normalmente, per potere essere effettivamente implementato, di un procedimento di **concretizzazione**, o **emulazione** ottenuto mediante uno o più livelli intermedi $\{MV_j \mid j = 1, \ldots, n-1\}$.


MV_i
P^i_{h1}
I^i_{k1}
R^i_p
R^i_q
R^i_r
P^i_{h2} : politica scritta in L_i
I^i_{k2}
meccanismo di L_i
MV_{i-1}
P^{i-1}_{m1}
R^{i-1}_v
I^{i-1}_{n1}
R^{i-1}_w
riferimento all' oggetto R^{i-1}_w
I^i_{k2} implementato da P^{i-2}_{f5}
interfaccia tra livelli
MV_{i-2}
P^{i-2}_{f1}
P^{i-2}_{f5}



I livelli $\{ MV_0, \ldots, MV_n\}$ sono ordinati secondo una **relazione gerarchica**. Ciò significa che ogni istruzione primitiva I_k^i, *o meccanismo* del linguaggio L_i $(i > 0)$ è implementata da un programma P_k, o **politica**, scritto nel linguaggio L_j, come mostrato in figura:



dove:

a) $0 \leq j < i$: anche se il caso più frequente è quello in cui $j = i - 1$, talvolta possono esistere varie ragioni, legate alle prestazioni del sistema e/o al costo dell'implementazione, per le quali conviene che sia $j < i - 1$;

b) se $I_{k1}^{\ i}$ e $I_{k2}^{\ i}$ sono meccanismi distinti di L_i, essi possono essere implementati da politiche scritte rispettivamente con L_{j1} e L_{j2}, dove $j1 \neq j2$, anche il caso più frequente è quello in cui $j1 = j2$: valgono considerazioni analoghe al caso precedente.

Esempio: Un programma Java '*multi-threaded*'...

Posso scrivere un programma Java (si tratta di un testo) che

Esempio: Un programma Java '*multi-threaded*'...

Posso scrivere un programma Java (si tratta di un testo) che

- una volta tradotto in codice JVM
  (ossia codice per la '*Java Virtual Machine*')...

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Esempio: Un programma Java '*multi-threaded*'...

Posso scrivere un programma Java (si tratta di un testo) che

- una volta tradotto in codice JVM
  (ossia codice per la '*Java Virtual Machine*')...
- ...può essere mandato in esecuzione su qualsiasi(?) Sistema di Elaborazione...

Esempio: Un programma Java '*multi-threaded*'...

Posso scrivere un programma Java (si tratta di un testo) che

- una volta tradotto in codice JVM
  (ossia codice per la '*Java Virtual Machine*')...
- ...può essere mandato in esecuzione su qualsiasi(?) Sistema di Elaborazione...
- ...creando l'illusione che vi siano piú micro processori ...

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Esempio: Un programma Java '*multi-threaded*'...

Posso scrivere un programma Java (si tratta di un testo) che

- una volta tradotto in codice JVM
  (ossia codice per la '*Java Virtual Machine*')...
- ...può essere mandato in esecuzione su qualsiasi(?) Sistema di Elaborazione...
- ...creando l'illusione che vi siano piú micro processori ...
- ...e che la memoria abbia una capienza maggiore della sua effettiva capacità fisica

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Esempio: Un programma Java '*multi-threaded*'...

Posso scrivere un programma Java (si tratta di un testo) che

- una volta tradotto in codice JVM
  (ossia codice per la '*Java Virtual Machine*')...
- ...può essere mandato in esecuzione su qualsiasi(?) Sistema di Elaborazione...
- ...creando l'illusione che vi siano piú micro processori ...
- ...e che la memoria abbia una capienza maggiore della sua effettiva capacità fisica

  (CPU virtuali, Memoria virtuale...)

UNIVERSITÀ
DEGLI STUDI DI TRIESTE