PIANO LAUREE SCIENTIFICHE

LABORATORIO PLS
LOGICA e INFORMATICA

Marco Lapegna
Dipartimento di Matematica e Applicazioni
Universita' degli Studi di Napoli Federico II

wpage.unina.it/lapegna

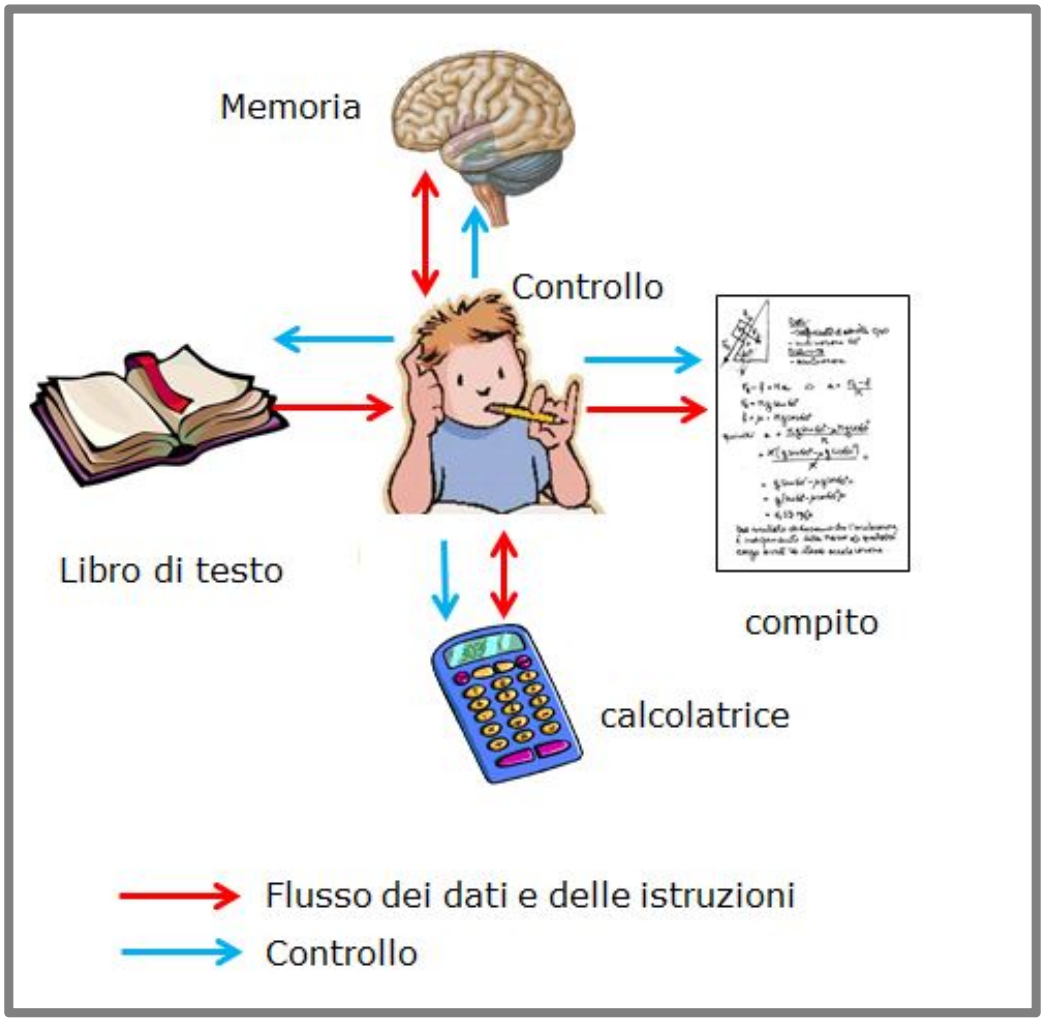
Uomo come esecutore di algoritmi

Un UOMO
e' un esempio di

esecutore di algoritmi

Esso e' infatti capace di

- Leggere da un libro e scrivere su un quaderno (interazione con l'esterno)
- Effettuare calcoli a mente o con un semplici strumenti (capacita' logico / aritmetica)
- Ricordare dati e istruzioni (capacita' di immagazzinare informazioni)
- Controllare tutte le attivita' in maniera coordinata (capacita' di controllo)



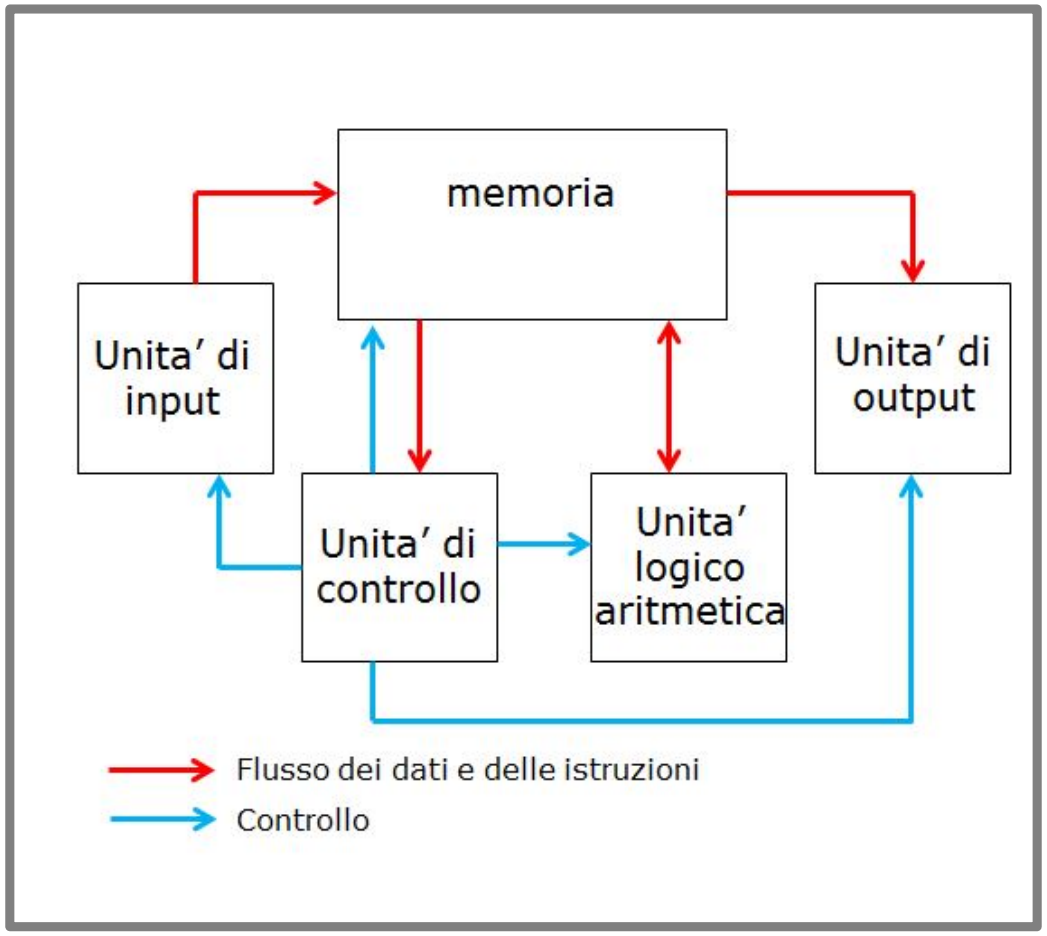
Schema uomo

La Macchina di Von Neumann

In maniera del tutto analoga funziona un calcolatore

Lo schema fu messo a punto da John von Neumann

Matematico ungherese del XX sec. emigrato in america nel 1933



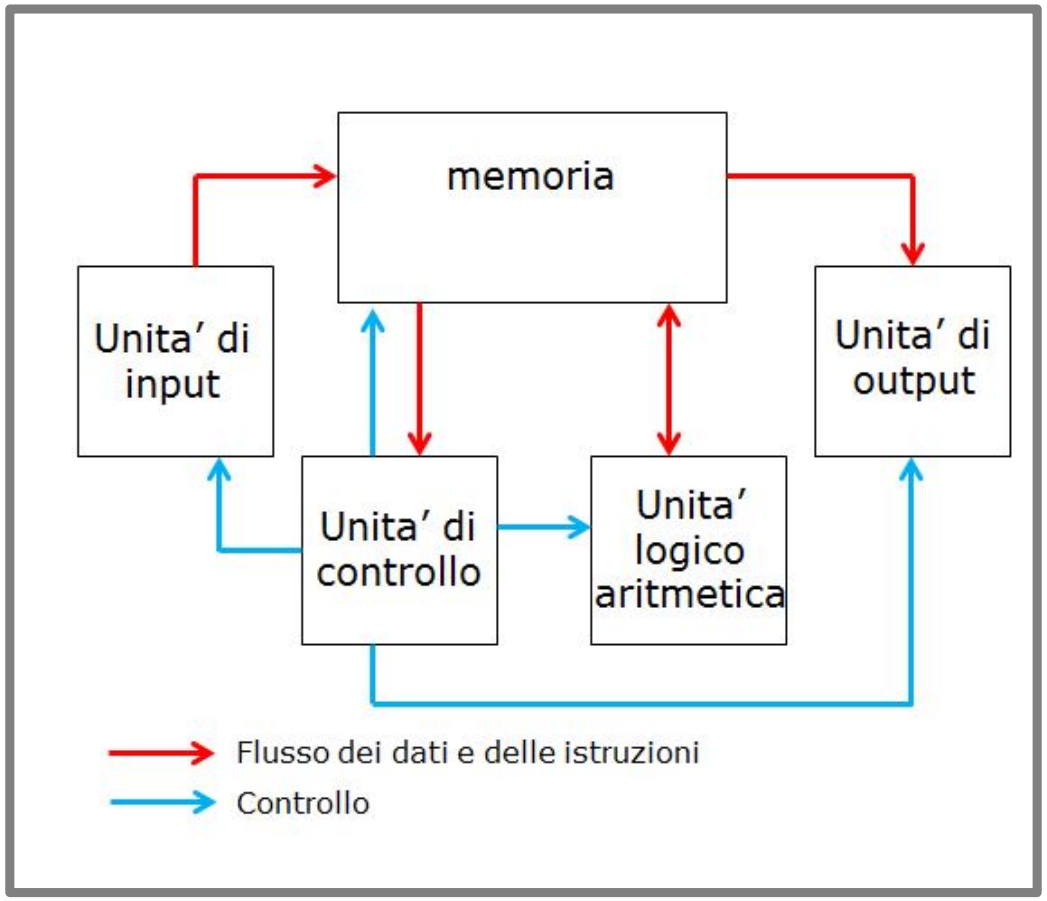
La macchina di Von Neumann

La Macchina di Von Neumann

I componenti fondamentali della macchina di Von Neumann sono:

- La Mamoria
- L'Unita' di Controllo
- L'Unita' logico aritmetica
- Le unita' di Input e Output

(CPU = unita' di controllo + unita' logico aritmetica)



La Macchina di Von Neumann

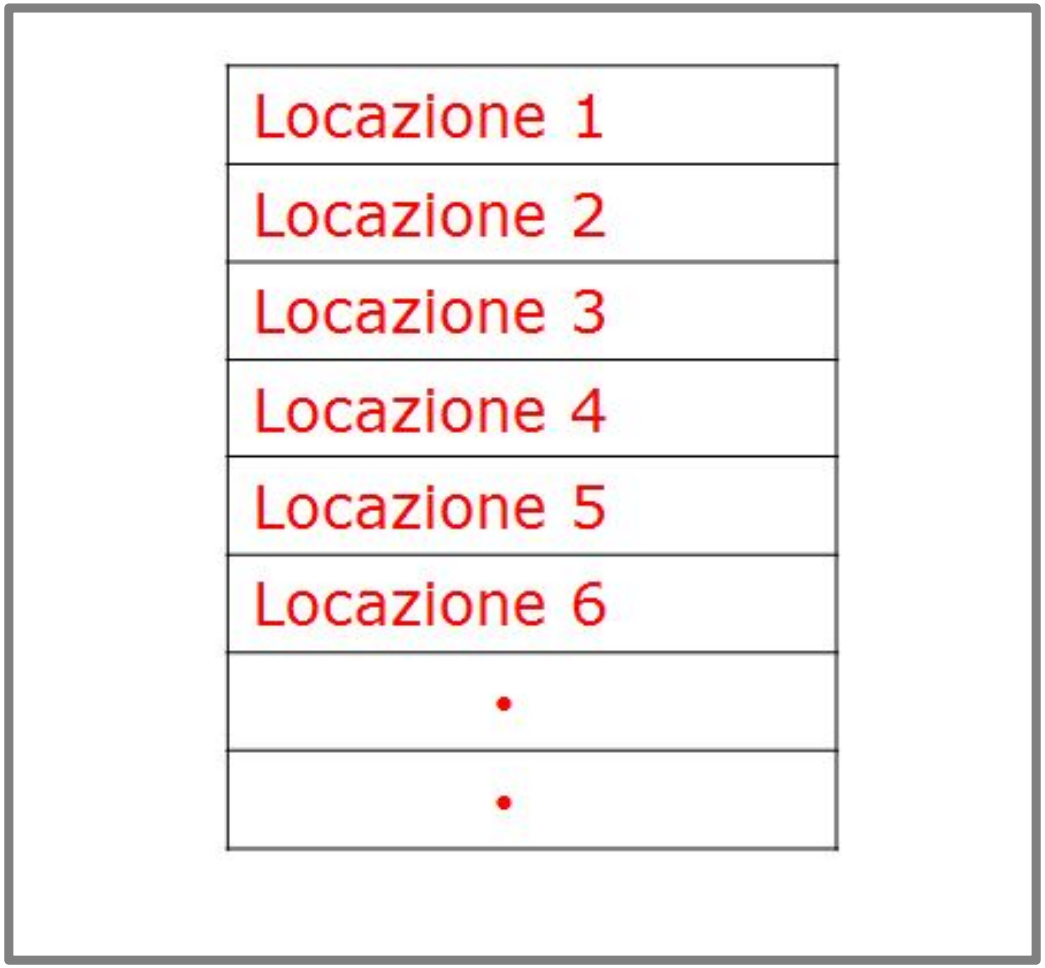
La memoria

La **memoria**
e' il dispositivo per

immagazzinare informazioni
(dati e istruzioni)

E' organizzata logicamente
come una lista di

LOCAZIONI DI MEMORIA
(o celle di memoria)



rappresentazione della memoria
suddivisa in locazioni

Le locazioni di memoria

Dal punto di vista tecnologico e' facile realizzare dispositivi capaci di memorizzare due stati

Per tale motivo ciascuna locazione e' composta da un insieme di componenti elementari, ciascuno dei quali capaci di rappresentare

0 oppure 1

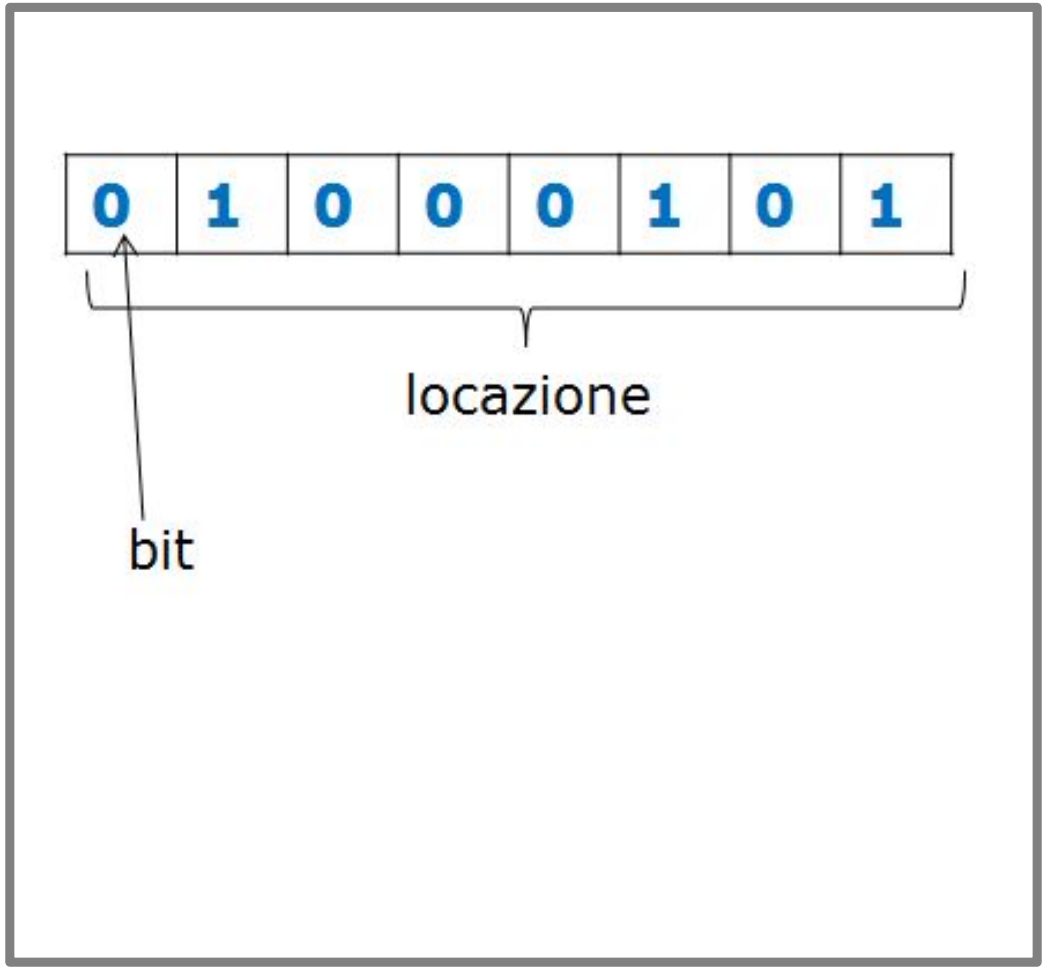
cioe'

una cifra binaria

anche detta

BIT

(contrazione di Binary digIT)



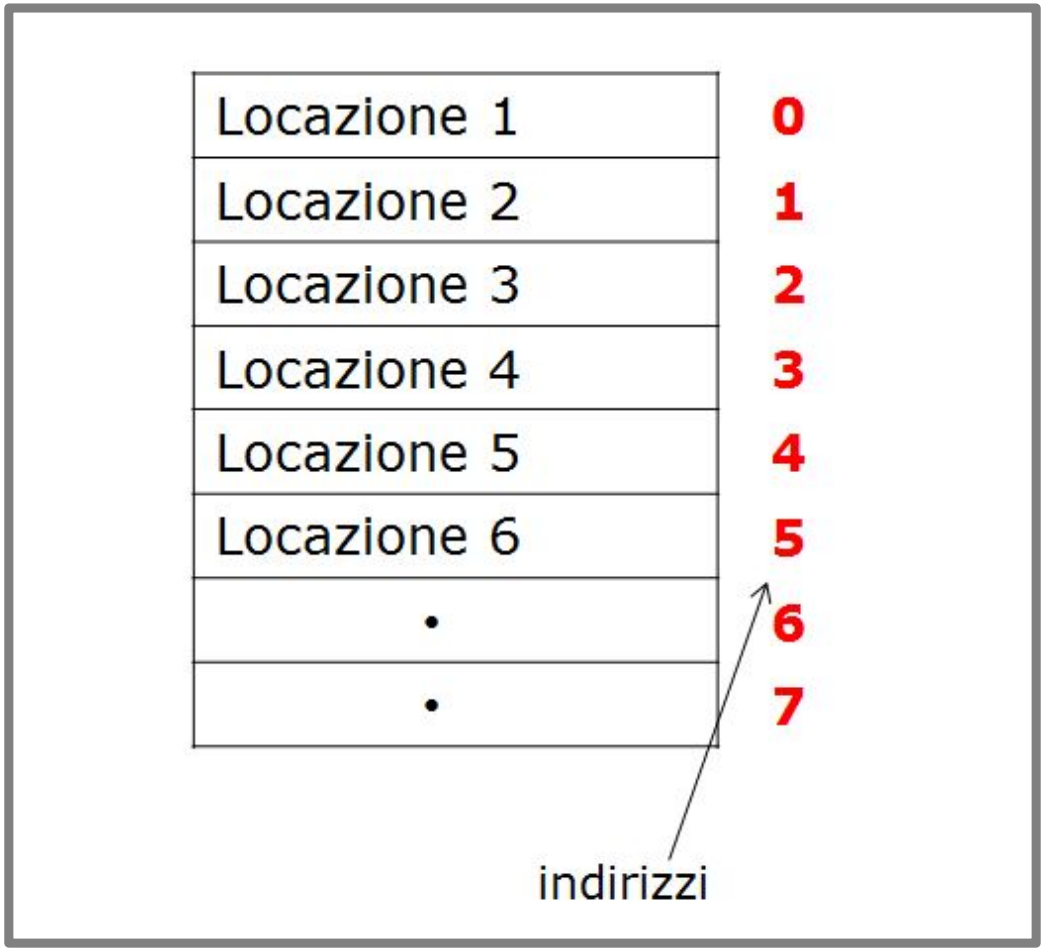
Un esempio di locazione di memoria a 8 bit

Gli indirizzi

Per poter accedere rapidamente ai dati presenti in memoria, le locazioni sono univocamente individuate da un

INDIRIZZO

Per tale motivo la locazione e' la minima quantita' di memoria a cui si puo' accedere



Gli indirizzi delle locazioni di memoria

Bit e byte

Nella prima fase dell'era informatica moderna (anni '50 del XX sec.)
ogni azienda strutturava le locazioni di memoria
con un numero di bit differente dalle altre

Conseguenza:
grandi problemi di portabilita' delle applicazioni da un calcolatore ad un altro

Ben presto sorse l'esigenza di
uniformare la dimensione delle locazione di memoria

Oggi la dimensione comune delle locazioni di memoria di tutti i calcolatori e'

8 bit cioe' 1 Byte

Le parole

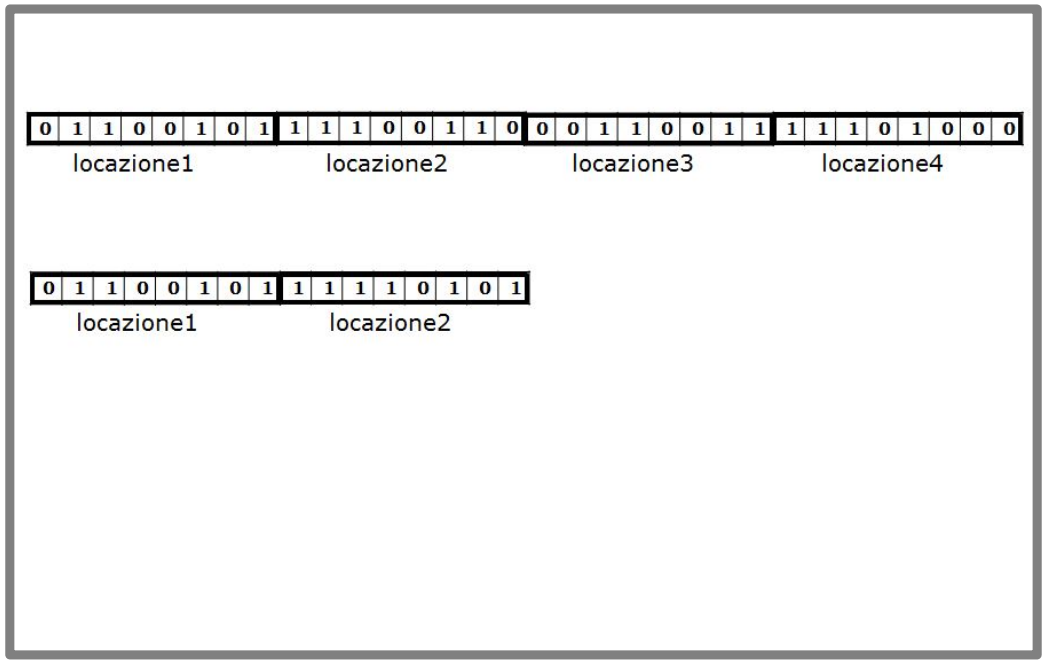
Con una sola locazione e' possibile memorizzare una quantita' di informazione molto piccola per le esigenze concrete

Ad esempio con 8 bit e' possibile memorizzare

- Un numero intero (con segno) minore di 128
- Una lettera dell'alfabeto o un carattere

Per tale motivo le locazioni sono raggruppate in

PAROLE o WORD



Alcuni esempi di parole a 4 e 2 locazioni

Tipicamente oggi i calcolatori hanno parole di

- 32 bit (cioe' 4 locazioni)
- 64 bit (cioe' 8 locazioni)

Operazioni sulla memoria

Sulle locazioni di memoria e' possibile effettuare due tipi di operazioni:

LETTURA

- Consiste nel prelevare un dato dalla memoria (ad esempio per utilizzarlo in una somma)
- L'operazione e' conservativa (il dato rimane nella locazione di memoria)

SCRITTURA

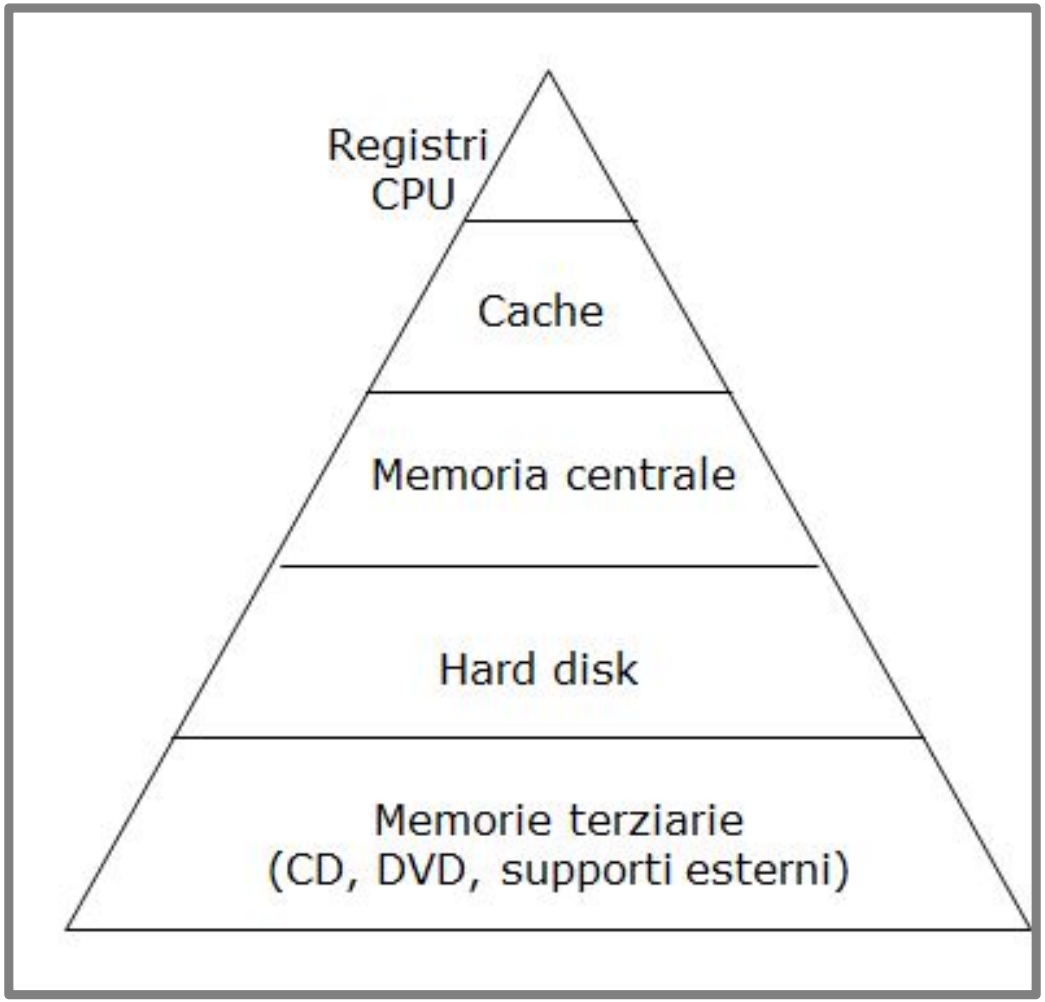
- Consiste nel mettere un dato in una locazione di memoria (ad esempio per memorizzare un dato di input)
- L'operazione e' distruttiva (un eventuale dato gia' presente si perde)

Osservazione

Da un punto di visto logico,
qualunque dispositivo capace
di memorizzare dati o istruzioni
e' una memoria

Dal punto di vista pratico
in un calcolatore esistono
diversi tipi di memoria,
con caratteristiche e funzioni diverse

Gerarchia di memoria



Gerarchia di memoria in un calcolatore

Gerarchia della memoria

Le principali caratteristiche che differenziano le varie memoria sono

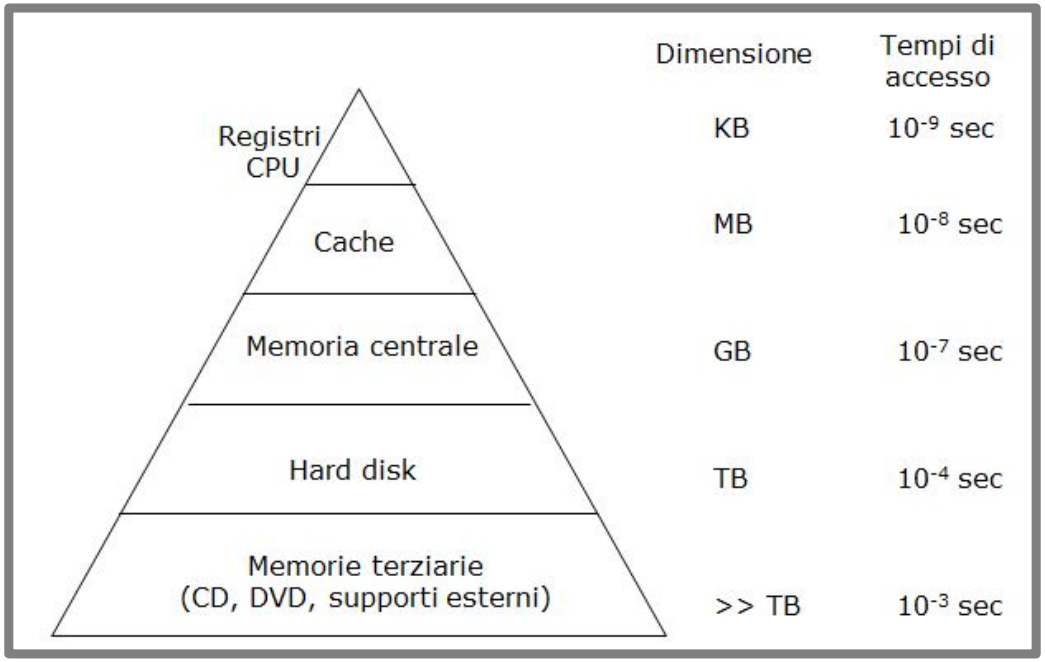
Il tempo di accesso e la dimensione

Memorie di alto livello (mem. Centrale e cache):

- Veloci (pochi ns) e piccole (MB - GB)
- Adatte a fornire dati alla CPU ad una adeguata velocita'
- Utilizzate solo dai programmi in esecuzione

Memorie di basso livello (hard disk e altri supporti):

- Lente (alcuni ms) e grandi (>> TB)
- Adatte a conservare grandi quantita' di dati e programmi in maniera permanente
- Utilizzate per conservare dati e programmi non utilizzati dalla CPU in quel momento



Dimensioni e tempi di accesso dei livelli di memoria

Il costo della memoria e' proporzionale a

N*V o equivalentemente **N / T**

Dove:

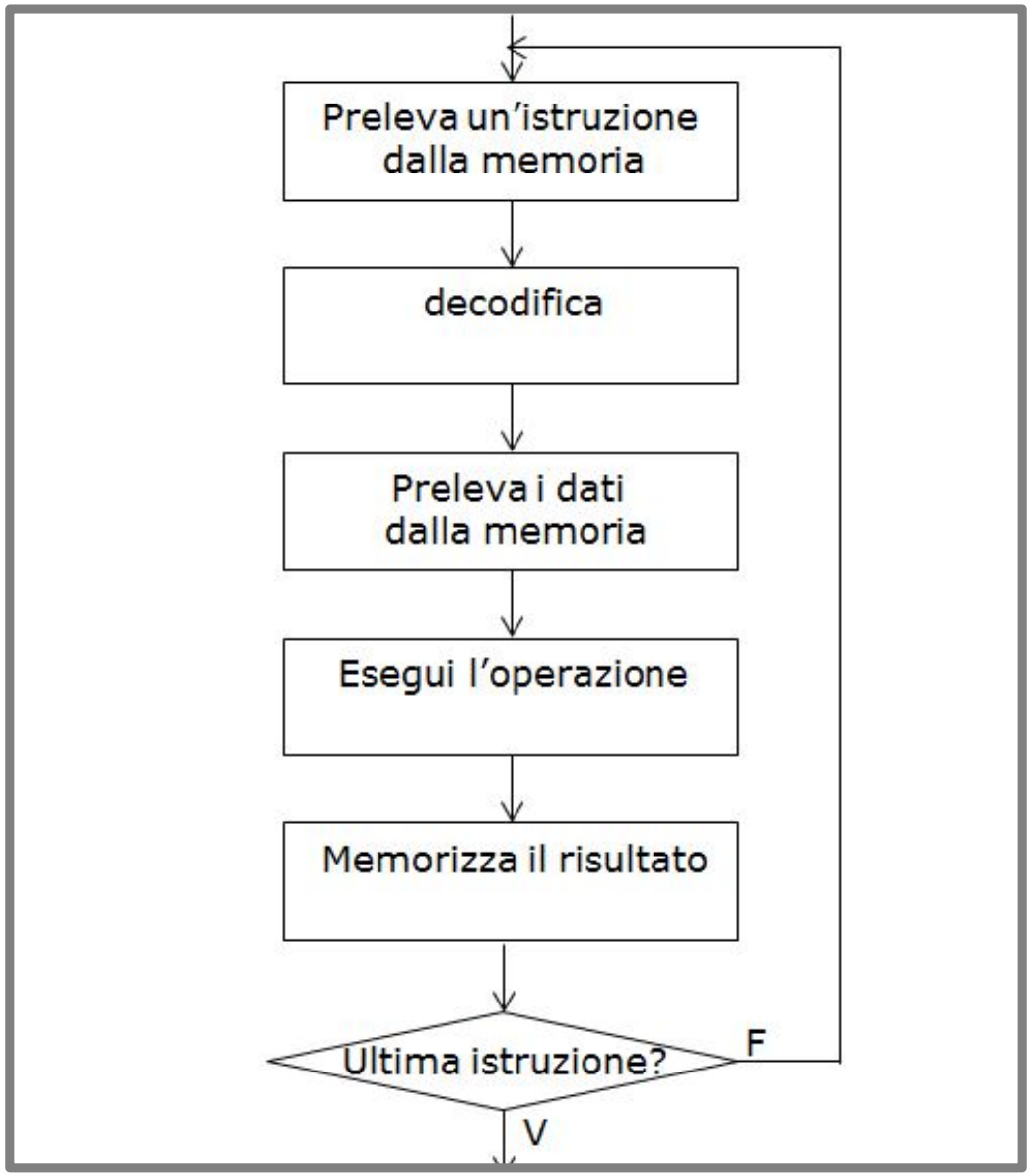
- N e' la dimensione in byte
- V e' la velocita'
- T e' il tempo di accesso

Un volantino pubblicitario
con indicazione dei livelli di memoria

Unita' di controllo

E' una componente della CPU ed e' l'unita' che coordina tutte le attivita' del calcolatore

- Preleva dalla memoria le istruzioni (fetch)
- Le interpreta (decode)
- Preleva i dati dalla memoria (load)
- Esegui l'istruzione (execute)
- Memorizza il risultato (store)
- Determina la successiva istruzione da eseguire



Un possibile, semplice algoritmo alla base dell'unita' di controllo

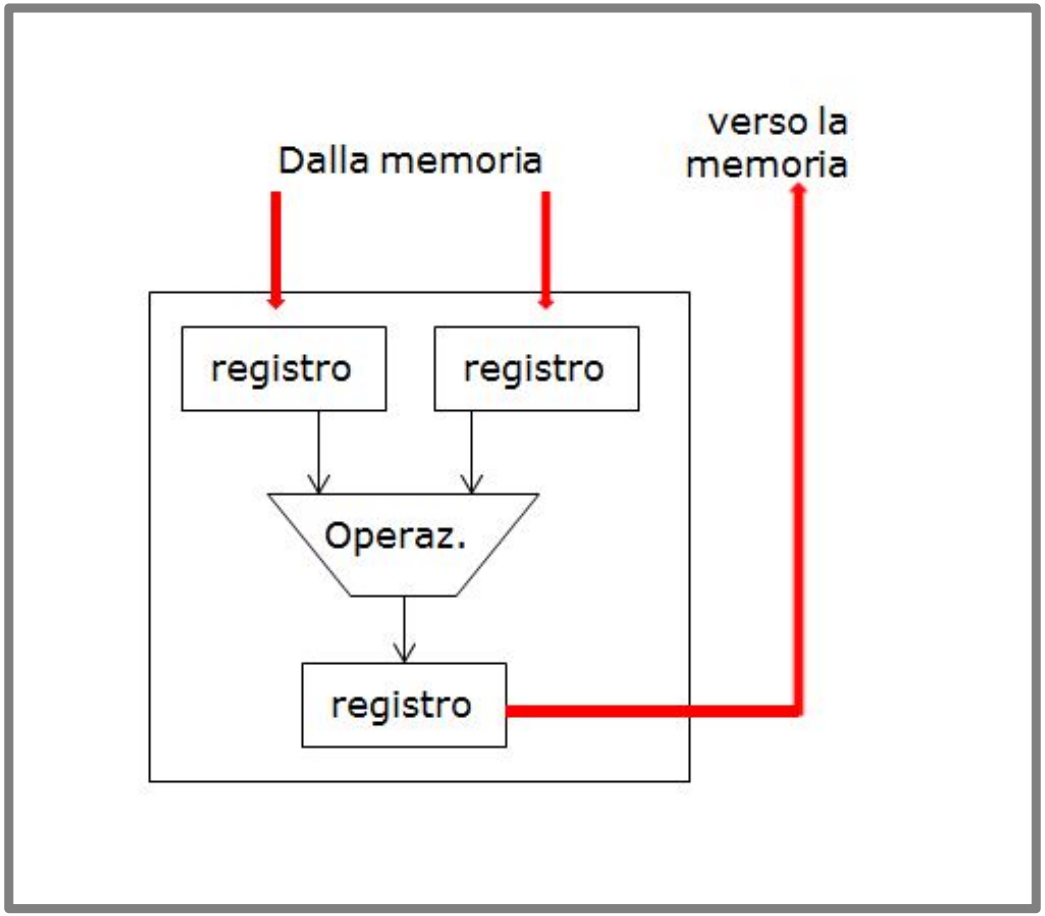
Unita’ logico aritmetica

E’ una componente della CPU ed e’ l’unita’ che esegue le

operazioni logiche ed aritmetiche,

eseguendo i comandi provenienti dall’unita’ di controllo

I dati provenienti dalla memoria sono depositati in appositi registri della CPU per una veloce elaborazione nel caso in cui servissero piu’ volte



Schema di unita’ logico aritmetica

La CPU

Unita' di Controllo
+
Unita' logico aritmetica
=
CPU

(Central Processing Unit)

Caratteristica principale della CPU:
Numero di azioni (istruzioni o operazioni) esguite nell'unita' di tempo

Unita' di misura
Hz (hertz) = 1/sec

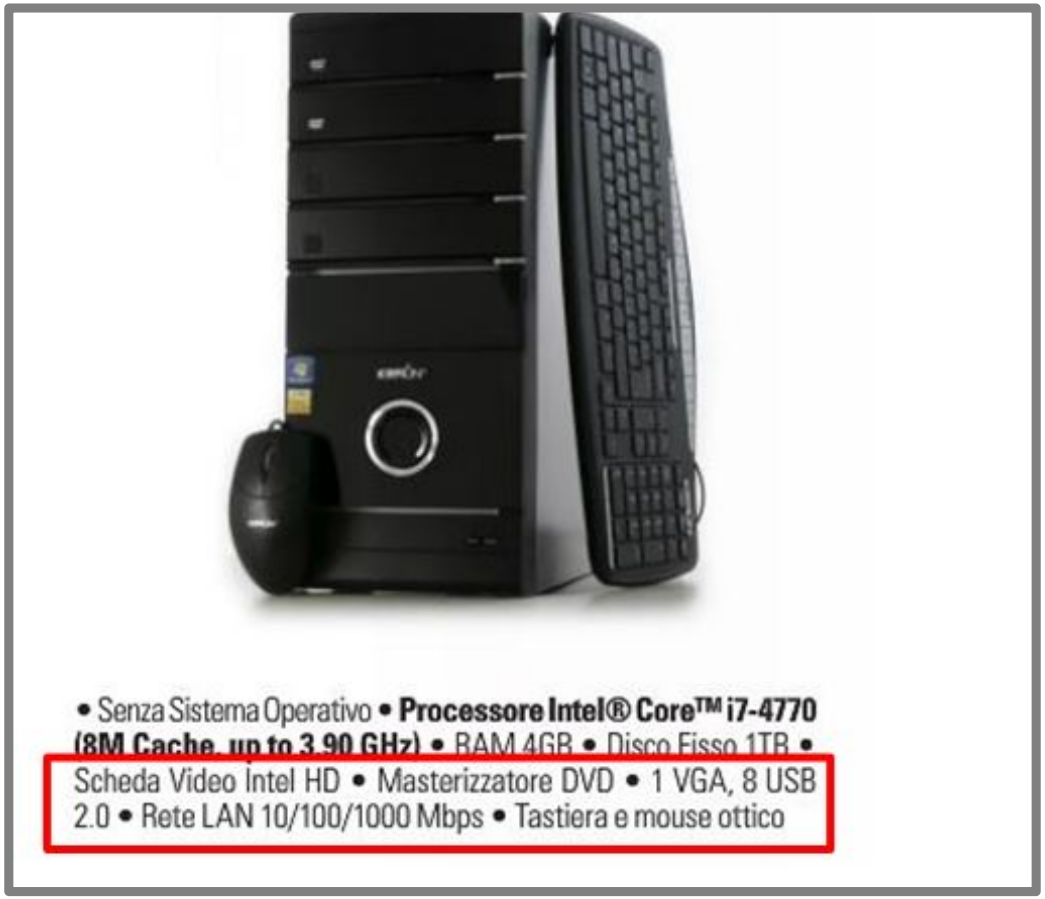
Es. 1 GHz = circa 10^9 istruzioni al sec.

Un volantino pubblicitario
con indicazione della velocita' operativa della CPU

Le unita'di I/O

Sono le interfacce del calcolatore con il mondo esterno

- Consente l'interazione con l'uomo (video, tastiera, stampante, mouse)
- Consente l'accesso alle reti
- Si preoccupa di convertire i dati in formato binario
- Trasferisce dati da e per la memoria



Un volantino pubblicitario
con indicazione delle unita' di I/O

Tipi di dati

Per tipo di dato si intende **una classificazione** che definisce

- i **possibili valori** che un certo dato puo' assumere
- Le **operazioni** che possono essere fatte su quel dato
- Il **significato** in memoria di quel dato

I **tipi di dati elementari** sono:

- Il tipo intero
- Il tipo alfanumerico
- Il tipo logico
- Il tipo reale

Il tipo Intero

Il **tipo intero** indica
il sottinsieme dei numeri relativi (interi positivi e negativi)
che possono essere rappresentati
nella memoria di un calcolatore

Poiche' le locazioni di memoria sono costituiti da bit,
i dati di tipo intero
sono rappresentati col sistema binario (base $b=2$)

Esempio

Nella **rappresentazione posizionale**,
le cifre di un numero sono i
coefficienti di una combinazione delle potenze della base

Ad esempio il numero intero *K=18* (in base *b=10*)
puo' essere rappresentato in base *b=2* come

K=10010

INFATTI

$$10010=1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 16+2 = 18$$

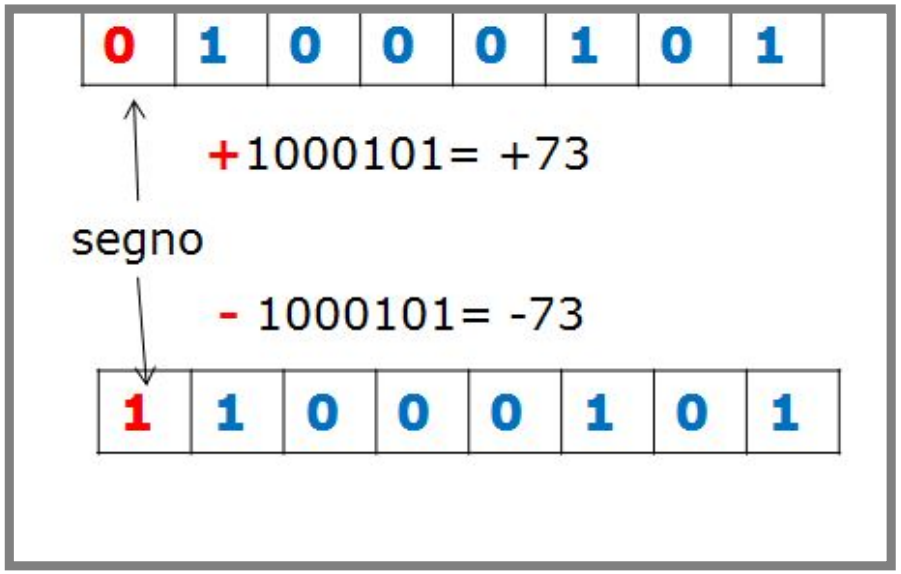
Memorizzare il tipo di dato intero

In una locazione di memoria si rappresentano le cifre binare della rappresentazione del numero intero con la convenzione che **il primo bit rappresenta il segno**

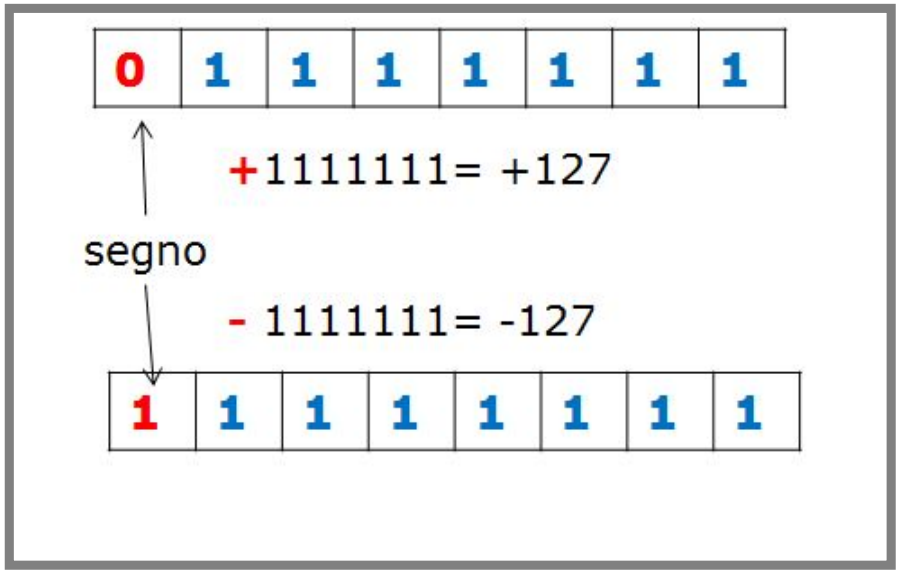
Ad esempio 1 = + e 0 = -

La limitatezza di una locazione di memoria implica

un Massimo e un Minimo numero intero rappresentabile



rappresentazione di due numeri interi di segno diverso



il massimo e il minimo intero rappresentabile in una locazione di 8 bit

Memorizzare il tipo intero

Con una sola locazione di memoria (8 bit) e' possibile rappresentare un insieme molto limitato di numeri interi (da -127 a 127)

Comunemente nei calcolatori si
usano parole da 4 locazioni (32 bit)

Quindi

$MAX = 2^{31} - 1$ $\qquad$ $-MAX = -(2^{31} - 1)$

Massimo intero rappresentabile $\qquad$ Minimo intero rappresentabile

In generale, se *L* e' la lunghezza della parola

$MAX = 2^{L-1} - 1$ $\qquad$ $-MAX = -(2^{L-1} - 1)$

Un numero intero K e' rappresentabile se

$-MAX \leq K \leq MAX$

Operazioni sui numeri interi

Sul tipo di dati intero sono definite le tradizionali operazioni aritmetiche:

- Addizione
- Sottrazione
- Moltiplicazione
- Divisione

OSSERVAZIONE

Poiche' nella rappresentazione del tipo intero non e' previsto il punto decimale, la divisione tra numeri interi produce sempre un numero intero per troncamento

Esempio: A=5, B=2 ➔ A/B = 2

Il tipo di dato logico

Il **tipo di dato Logico** indica l'insieme costituito
da due soli elementi:

{ vero ; falso }

Si utilizza quindi nella verifica
della **veridicita' di una proposizione**

Esempio:

P: Napoli e' in Campania — **P** e' una proposizione **vera**

Q: 3 e' maggiore di 5 — **Q** e' una proposizione **falsa**

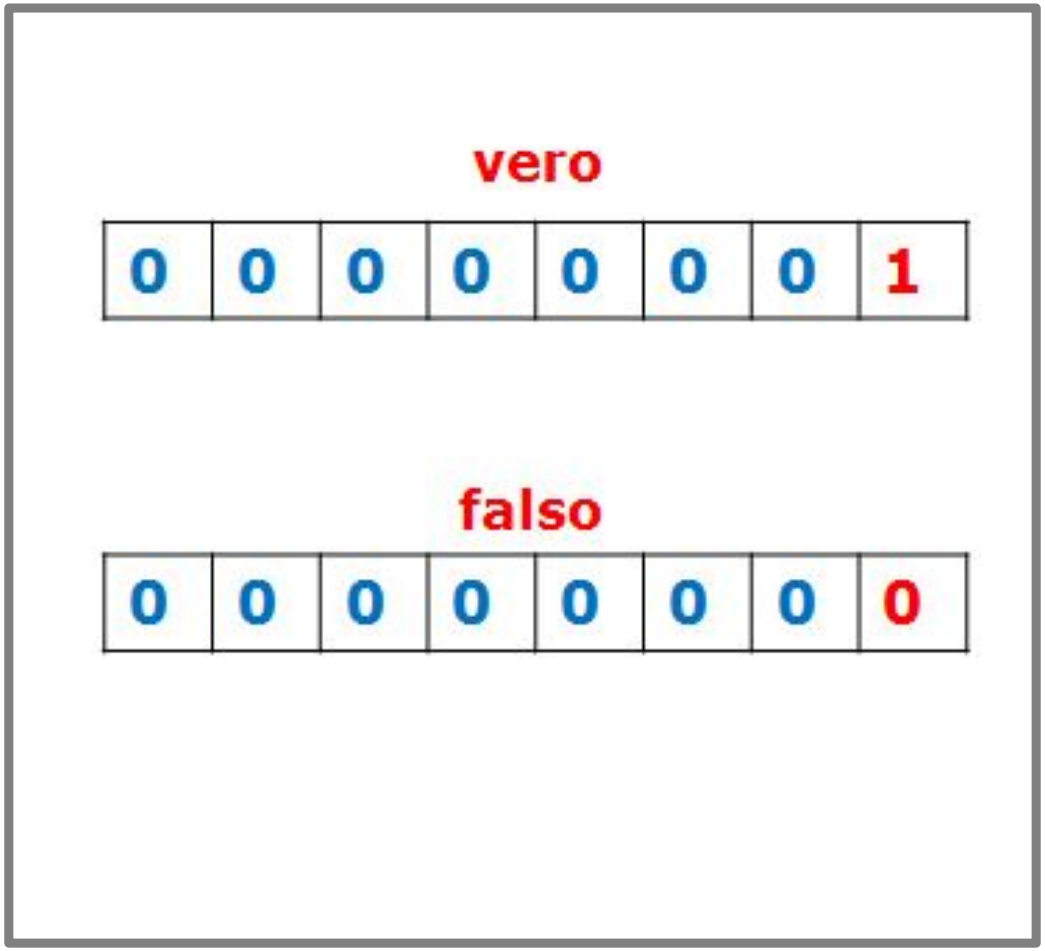
rappresentazione

Per rappresentare i due valori in memoria sarebbe sufficiente un solo bit

MA

Poiche' la minima quantita' di memoria indirizzabile e' la locazione di memoria

Il tipo di dato logico e' rappresentato utilizzando un'intera locazione



Rappresentazione in memoria del tipo di dato logico

Operazione di negazione (not)

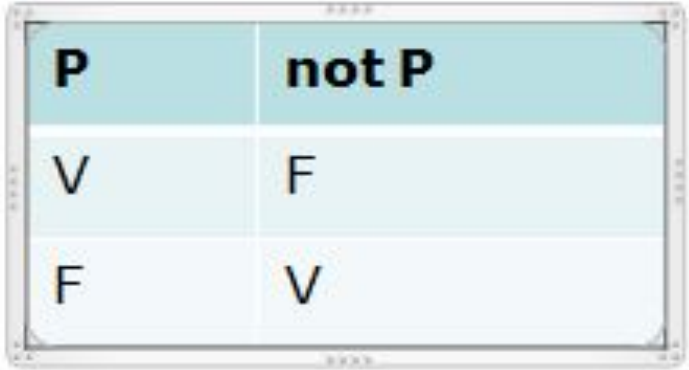
L'operazione di **negazione** e' un'operazione logica che agisce su un dato logico, cambiandone il valore

Esempio:

Siano due numeri A=3 e B=5

P: A e' minore di B
(vero)

not P: A **non** e' minore di B
(falso)



P	not P
V	F
F	V

Tavola della verita' dell'operazione di negazione

Operazione di congiunzione (and)

L'operazione di **congiunzione** e' un'operazione logica che agisce su due dati logici, ed e' vera solo se entrambi i dati sono veri

Esempi:

Siano A=3 X=4 B=5

P: X maggiore di A(vero)

Q: X minore di B (vero)

P and Q:

X maggiore di A **e** X minore di B (cioe' X compreso tra A e B) = vero

P	Q	P and Q
V	V	V
V	F	F
F	V	F
F	F	F

Tavola della verita' dell'operazione di congiunzioine

Operazione di disgiunzione (or)

L'operazione di **disgiunzione** e' un'operazione logica che agisce su due dati logici, ed e' vera se almeno uno dei dati e' vero

Esempi:

Siano A=3 X=6 B=5

P: X minore di A (falso)

Q: X maggiore di B (vero)

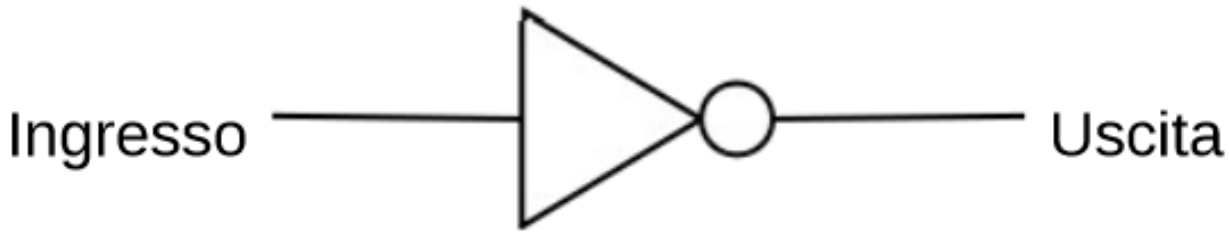
P OR Q:

X minore di A **o** X maggiore di B (cioe' X esterno a [A , B]) = vero

P	Q	P or Q
V	V	V
V	F	V
F	V	V
F	F	F

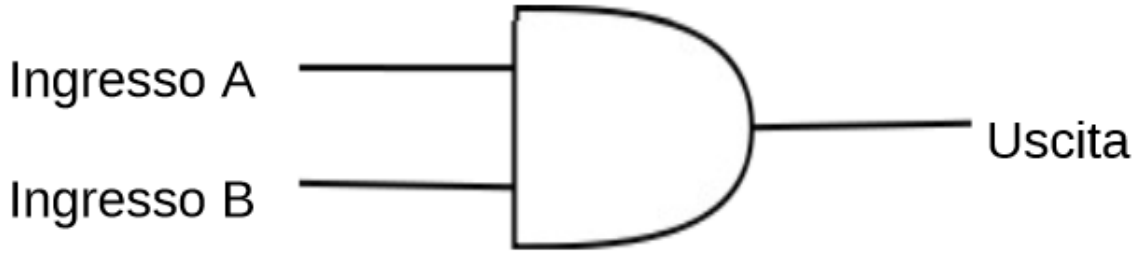
Tavola della verita' dell'operazione di disgiunzione

NOT



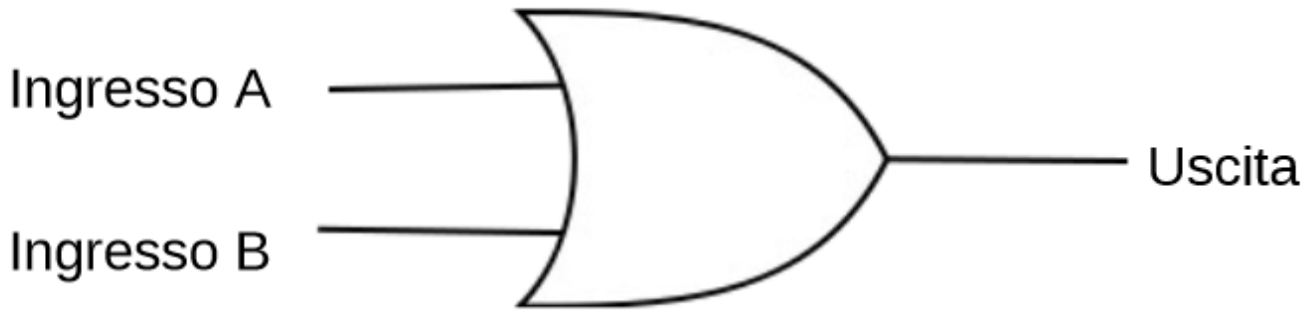
Ingresso	Uscita
0	1
1	0

AND



Ingresso A	Ingresso B	Uscita
0	0	0
0	1	0
1	0	0
1	1	1

OR



Ingresso A	Ingresso B	Uscita
0	0	0
0	1	1
1	0	1
1	1	1

OPERATORI UNIVERSALI

Un insieme finito di operatori si dice composto da Operatori Universali se qualsiasi funzione booleana puo' essere scomposta in termini di questi operatori

Thm. AND, OR e NOT sono un insieme di Operatori Universali

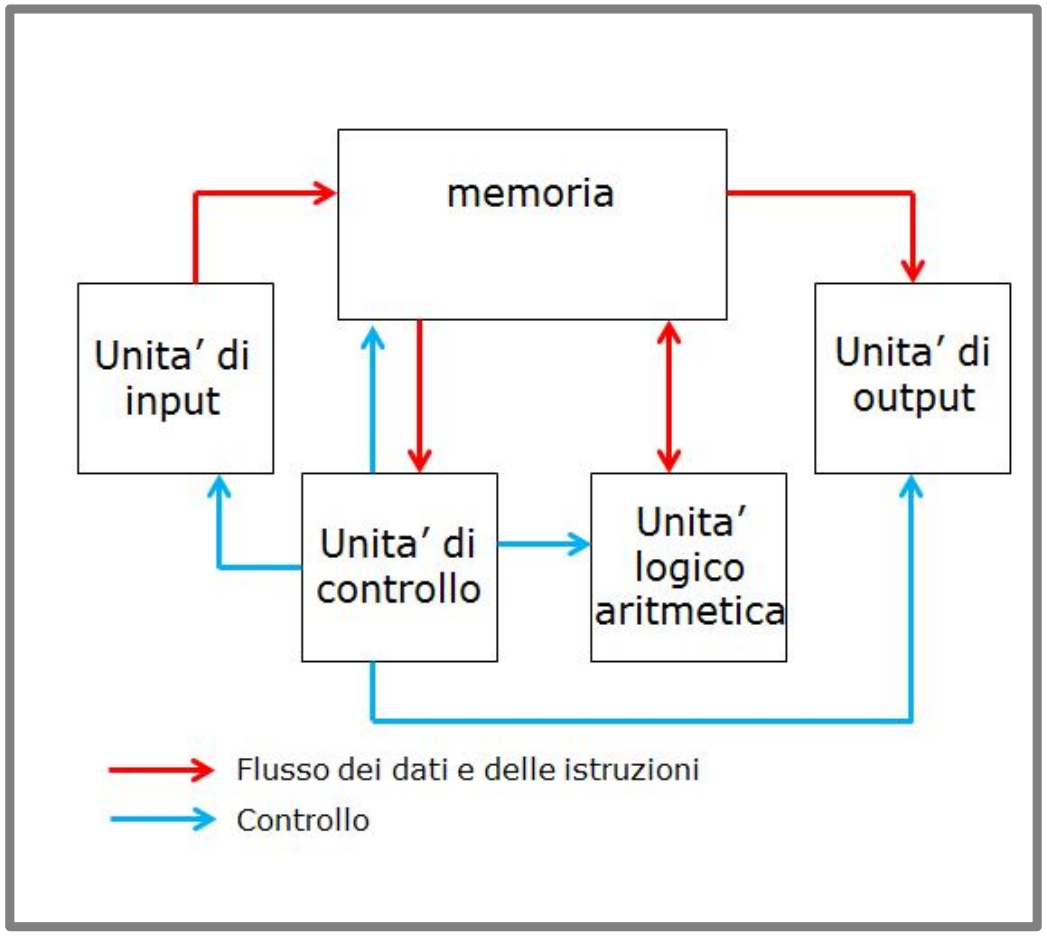
Thm. Un insieme di operatori e' universale se e solo se AND, OR e NOT possono essere definiti in termini di questi operatori

La Macchina di Von Neumann

In maniera del tutto analoga funziona un calcolatore

Lo schema fu messo a punto da John von Neumann

Matematico ungherese del XX sec. emigrato in america nel 1933



La macchina di Von Neumann

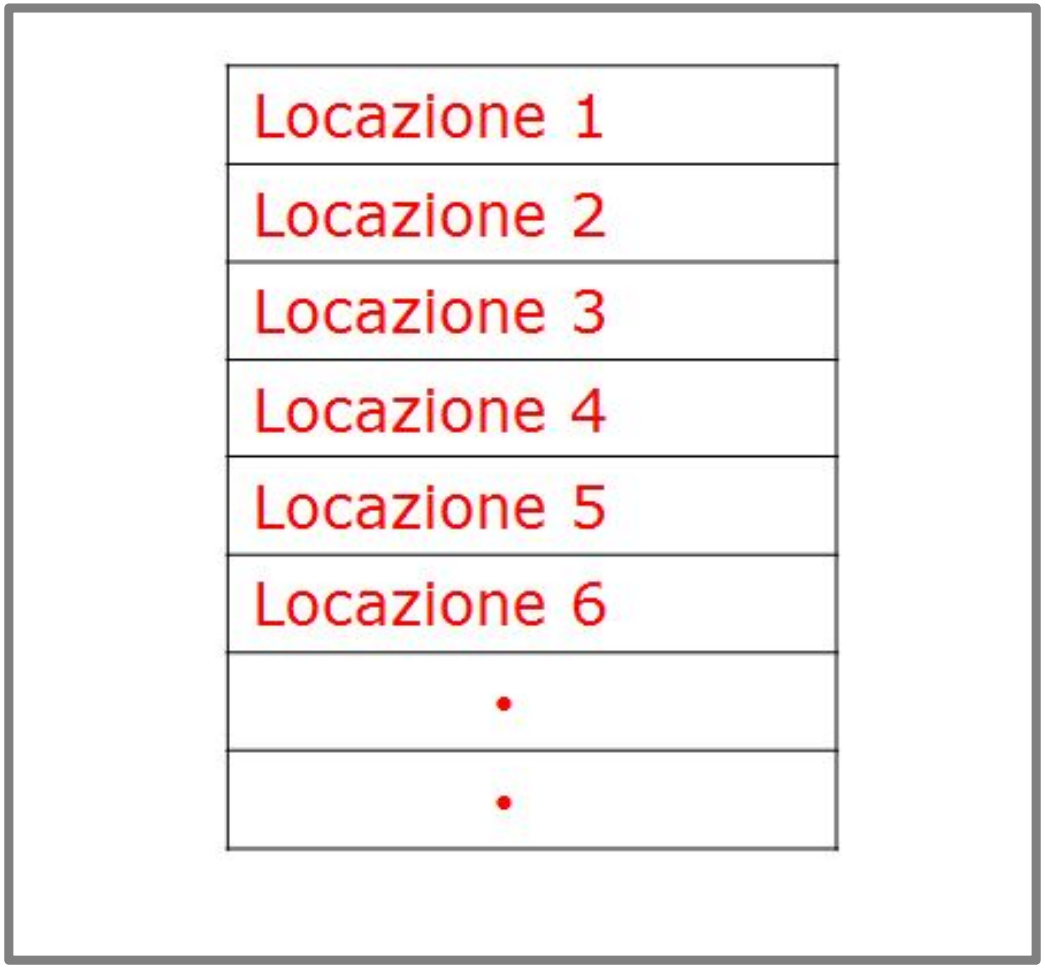
La memoria

La **memoria**
e' il dispositivo per

immagazzinare informazioni
(dati e istruzioni)

E' organizzata logicamente
come una lista di

LOCAZIONI DI MEMORIA
(o celle di memoria)



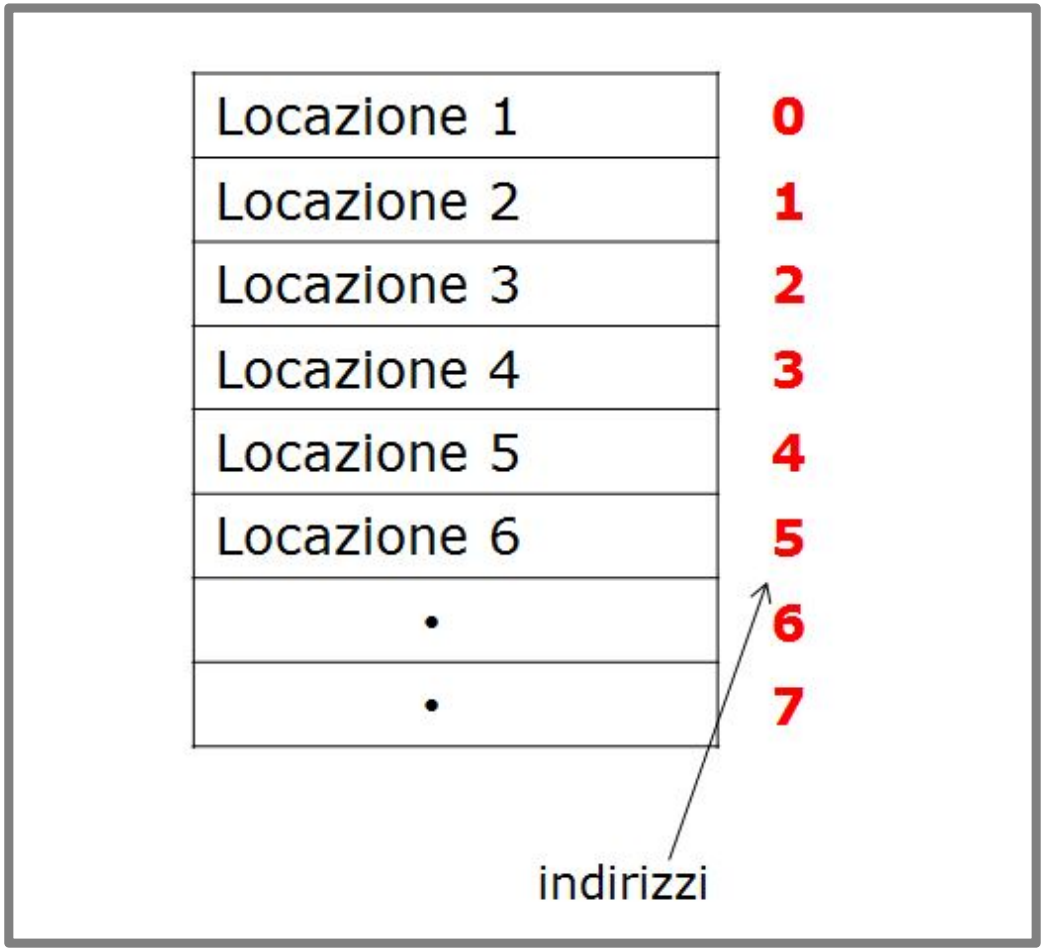
rappresentazione della memoria
suddivisa in locazioni

Gli indirizzi

Per poter accedere rapidamente ai dati presenti in memoria, le locazioni sono univocamente individuate da un

INDIRIZZO

Per tale motivo la locazione e' la minima quantita' di memoria a cui si puo' accedere



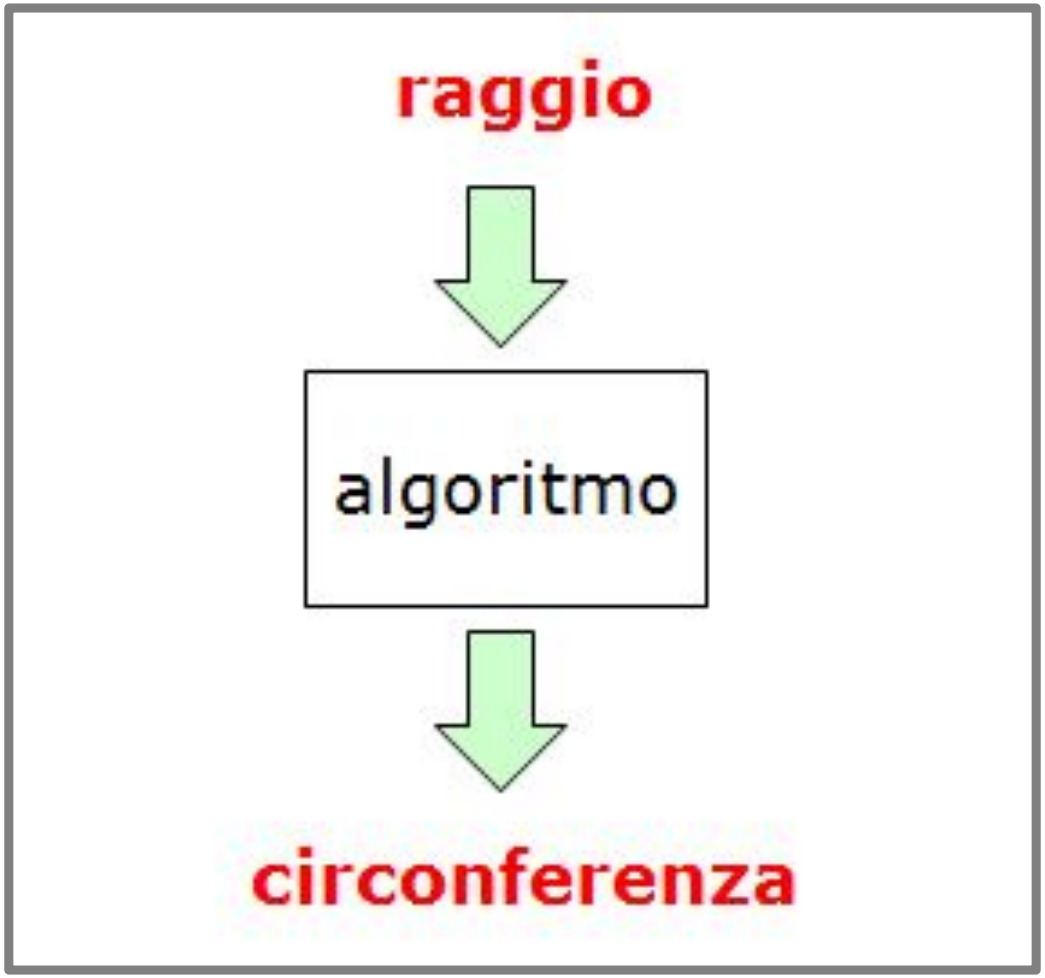
Gli indirizzi delle locazioni di memoria

Esempio: calcolo della circonferenza del cerchio

Si vuole calcolare la circonferenza del cerchio per molteplici valori del raggio:

- Raggio = 1
  circonferenza = $2 \pi 1$
- Raggio = 2.5
  circonferenza = $2 \pi 2.5$
- Raggio = 5
  circonferenza = $2 \pi 5$

Alcuni dati sono **costanti** (2 e π)
Altri dati sono **variabili** (raggio e circonferenza)



Input e output dell'algoritmo

Le variabili

Nell'algoritmo precedente

- I valori di 2 e π
- sono costanti e definiti per ogni esecuzione

- I valori di *C* e *R*

non sono definiti e rappresentano quantita' che variano da esecuzione a esecuzione

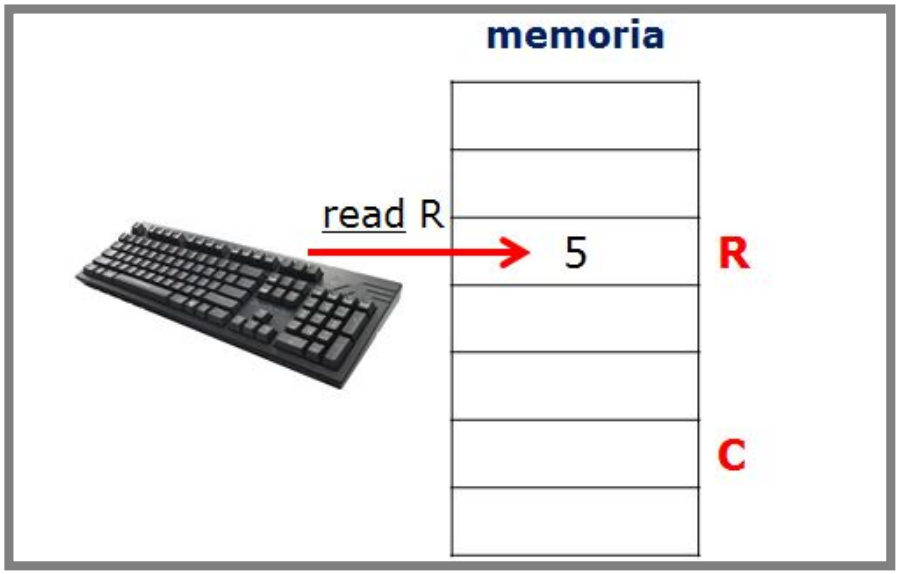
C e *R* indicano generici numeri reali

variabili di tipo reale

Istruzioni read e print

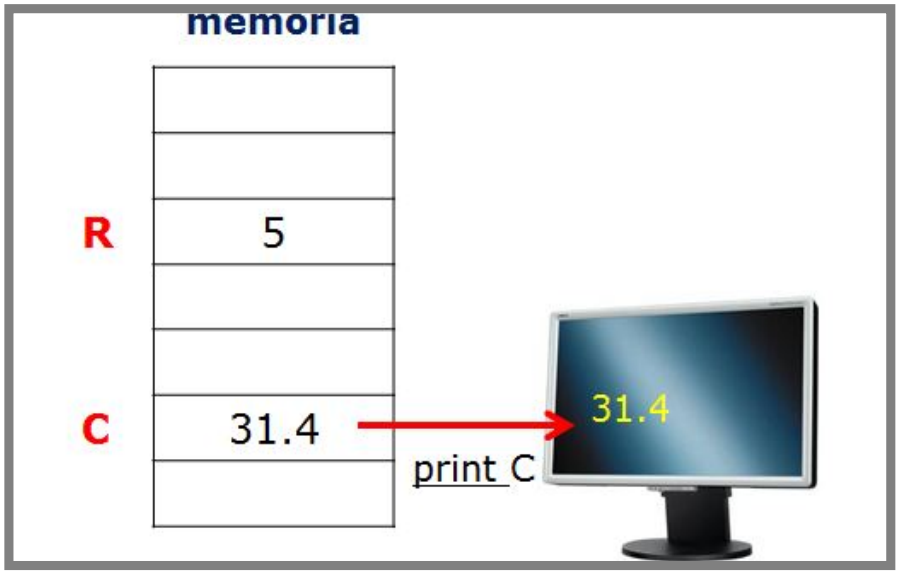
Dal punto di vista pratico, una **variabile** e' il **nome dato ad una locazione di memoria**

L'istruzione "read R" **legge** un valore dall'unita' di input e lo **assegna alla variabile R**



Esecuzione istruzione read

L'istruzione "print C" **visualizza** sull'unita' di output il **contenuto della variabile C**



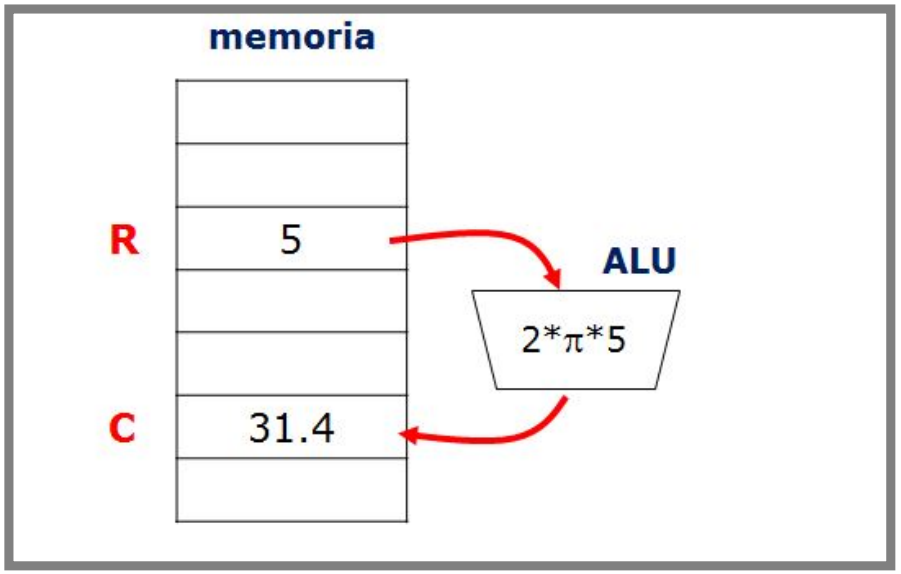
Esecuzione istruzione print

Assegnazione

L'istruzione

$C = 2 * \pi * R$

- **Preleva** dalla memoria il contenuto della variabile R
- Lo **moltiplica** per $2\ \pi$
- **Memorizza** il risultato nella variabile C



Esecuzione di una istruzione di assegnazione

Osservazioni

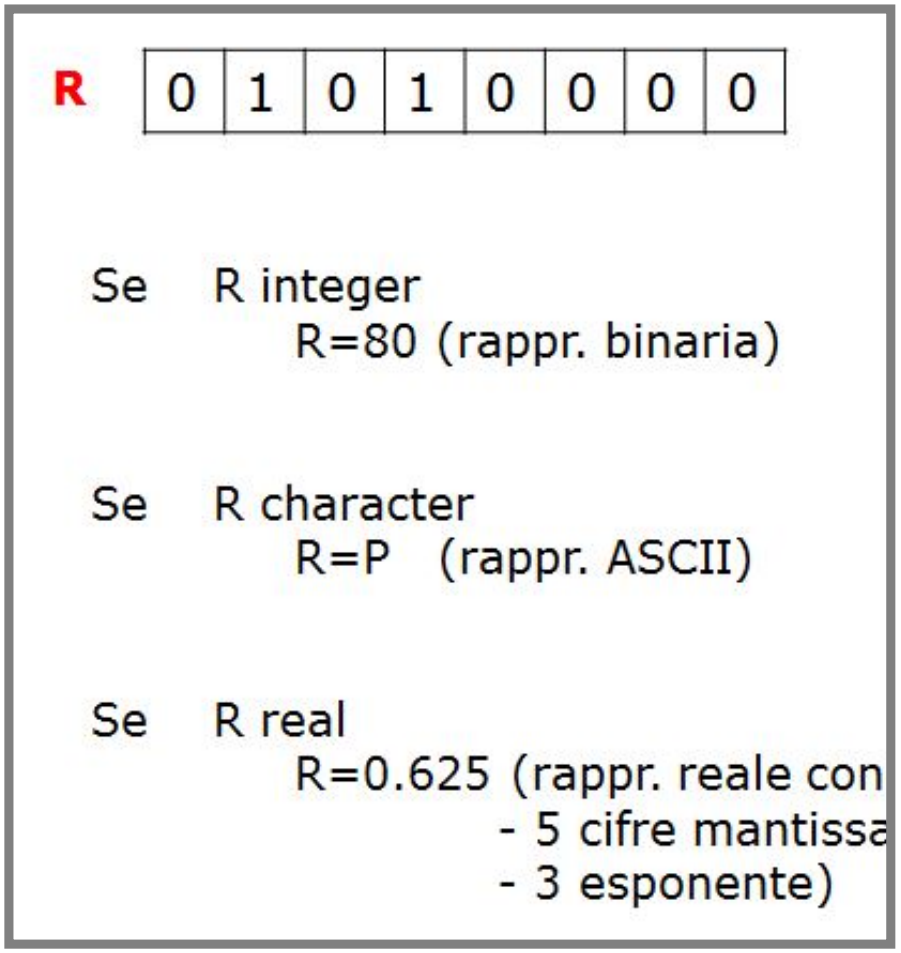
- Il segno "=" e' una operazione di **assegnazione**
- La variabile R deve essere **definita** (cioe' avere un valore)

Dichiarazione delle variabili

Dal punto di vista pratico, una **variabile** e' il **nome dato ad una locazione di memoria**

Poiche' la squenza di bit presente nella locazione di memoria ha significato diverso a secondo del tipo di dato contenuto

In pascal-like e' necessario dichiarare il tipo delle variabili



Diverso valore di R a secondo della dichiarazione

Esempi:

```
var R , C  : real        (R e C variabili reale)
var N      : integer     (N variabile intera)
var nome   : character   (nome variabile carattere)
var L      : logical     (L variabile logica)
```

Calcolo della circonferenza (seconda versione)

La dichiarazione delle variabili consente al calcolatore di interpretare correttamente il contenuto delle variabili

La dichiarazione delle variabili va fatta prima dell'utilizzo delle variabili stesse

Si ottiene quindi il seguente algoritmo

```
begin circonferenza

var   R, C  : real

read  R

C = 2*π * R

print  C

end circonferenza
```

Algoritmo per il calcolo della circonferenza

Le costanti

Nell'istruzione " R = P" come distinguere

- Assegna a R il contenuto della variabile P
- Assegna a R il carattere P

Analogamente, nell'istruzione " C = 6", come distinguere

- Assegna a C il numero intero 6
- Assegna a C il carattere 6

Necessita' di definire un modo per usare i diversi tipi di costanti
(intero, reale, carattere, logico)

Le costanti

- Le costanti di **tipo carattere** sono racchiuse tra una **coppia di apici**
  Esempio: ‘casa’ , ‘Napoli’ , ‘Maria’

- Le costanti di **tipo logico** sono racchiuse tra una **coppia di punti**
  Esempio: .true. , .false.

- Le costanti di **tipo intero** sono **sequenze di cifre** eventualmente con segno
  Esempio: +80 , -45 , 981

- Le costanti di **tipo reale** sono **sequenze di cifre con il punto decimale**
  Esempio: 45.23 , 0.0032 , -23.12,

  Per il tipo reale e’ possibile utilizzare anche la notazione esponenziale
  Esempio: 12.3e5 , 0.e0 , 0.1e-6

Calcolo della circonferenza (terza versione)

Nell'algoritmo per il calcolo della circonferenza il valore del simbolo (costante) π e' ambiguo

Specificando il valore come una costante di tipo reale si ottiene la versione finale dell'algoritmo

```
begin circonferenza

var   R, C  : real

read  R

C = 2 * 3.1415 * R

print  C

end circonferenza
```

Algoritmo per il calcolo della circonferenza

Calcolo della circonferenza (quarta versione)

Definizione = Assegnazione di un valore ad una variabile

Puo' avvenire in due modi

- Tramite lettura (istruzione read)
- Tramite assegnazione (operatore =)

Utilizzando una ulteriore variabile (PG), l'algoritmo per il calcolo della circonferenza puo' essere riscritto come riportato

```
begin circonferenza

var   R, C, PG  : real

read  R

PG = 3.141592

C = 2 * PG * R

print  C

end circonferenza
```

Algoritmo per il calcolo della circonferenza

Alcune regole di base

Tutte le variabili devono essere **definite prima di essere utilizzate**

```
begin circonferenza

var   R, C , PG : real

read  R

C = 2 * PG * R

print  C

end circonferenza
```

ERRORE !! : La variabile PG non e' definita

Il valore dell'espressione deve essere dello **stesso tipo di quello della variabile a cui e' assegnato**

```
begin circonferenza

var   R, C : real

var   PG : integer

read  R

PG = 3.141592

C = 2 * PG * R

print  C

end circonferenza
```

ERRORE !! : La variabile PG non e' usata in modo corretto

Esempio: calcolo area del trapezio

Osservazione:

Nella definizione di una variabile**, prima** si valuta l'espressione al secondo membro, e **dopo** si assegna il valore ottenuto alla variabile al primo membro

Una variabile puo' essere
utilizzata piu' volte ridefinendo
il suo valore nel corso dell'algoritmo

Esempio:

Algoritmo per il calcolo
dell'area di un trapezio

Dati di input: Bmag, Bmin, H
Dati di output: Area

```
begin trapezio
var   Bmag, Bmin  : real
var   H,  Area : real
read  Bmag,  Bmin
read  H
Area = Bmag + Bmin
Area = Area * H
Area = Area /2
print  Area
end trapezio
```

Algoritmo per l'area di un trapezio

Esecuzione dell'algoritmo (1)

Calcolo dell'area di un trapezio con
Bmag = 10 , Bmin= 8 , H = 3
(Area = 27)

Fig.1: esecuzione di read Bmag, Bmin

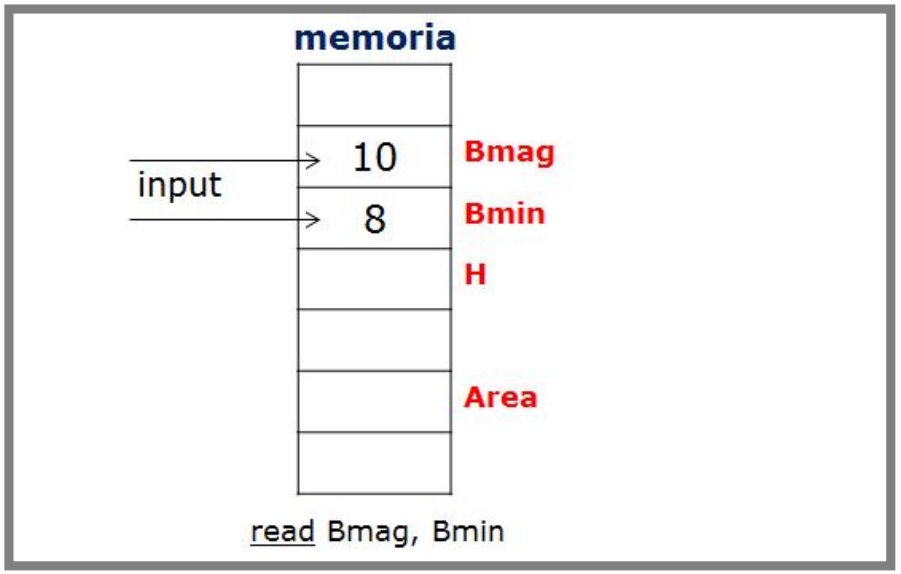


Fig.1: esecuzione di "read Bmag, Bmin"

Fig.2: esecuzione di read H

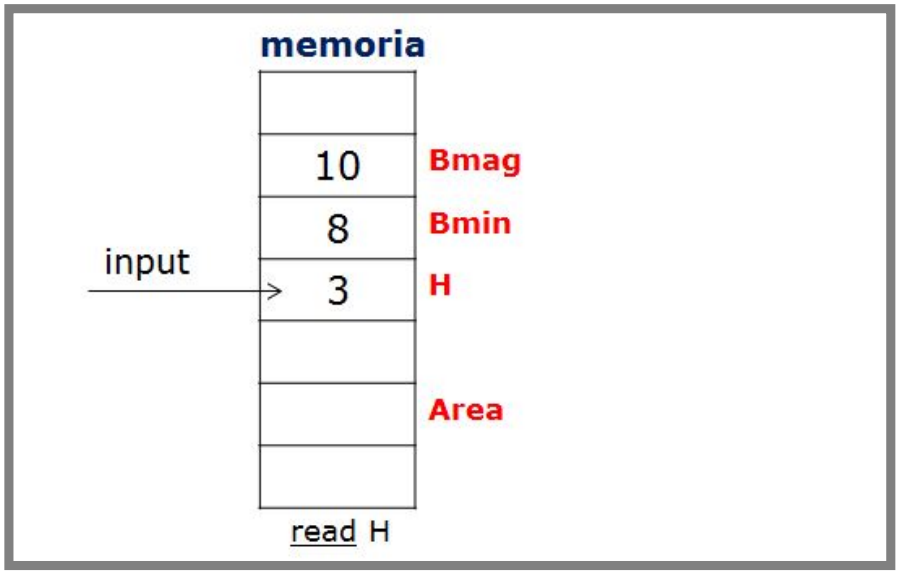


Fig.2: esecuzione di "read H"

Esecuzione dell'algoritmo (2)

Fig.3: esecuzione di Area = Bmag +Bmin

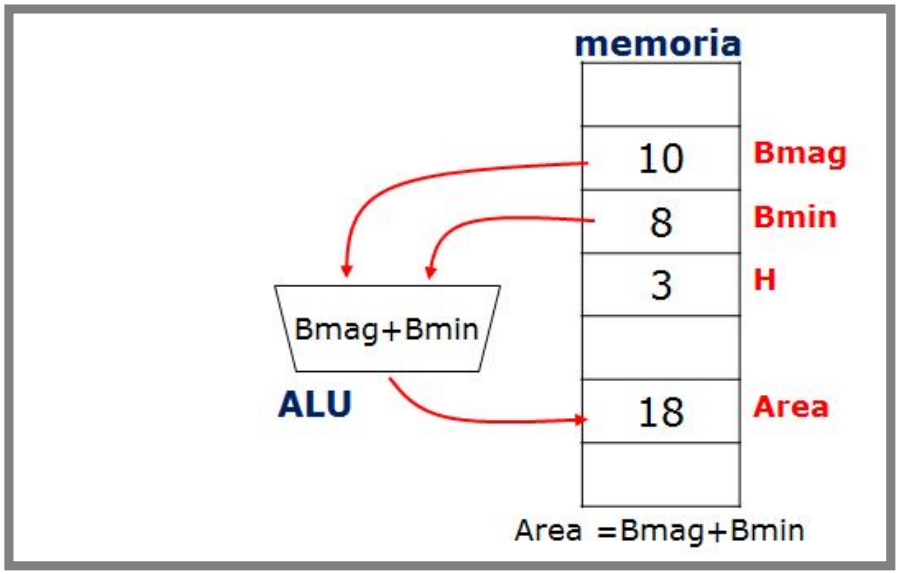


Fig.3: esecuzione di "Area = Bmag+Bmin"

Fig.4: esecuzione di Area = Area * H

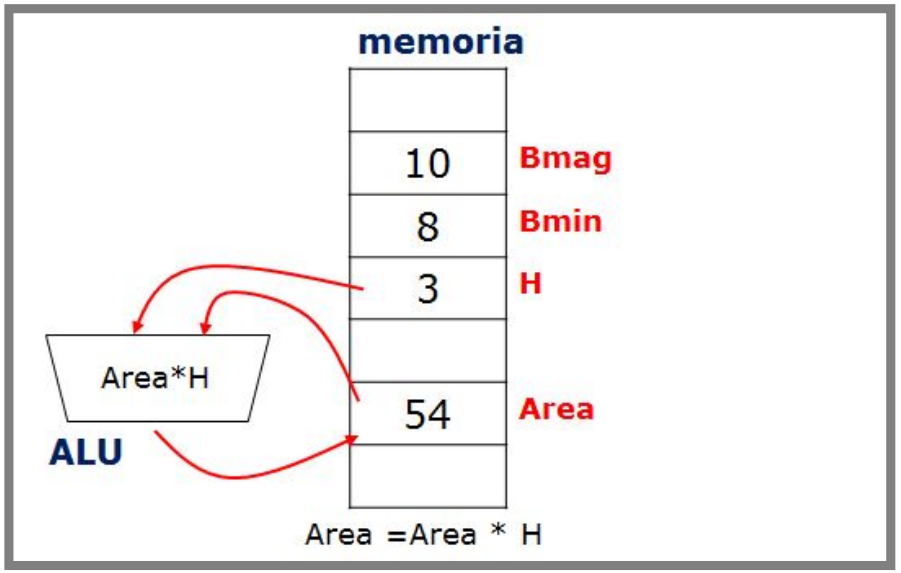


Fig.4: esecuzione di "Area = Area * H"

Esecuzione dell'algoritmo (3)

Fig.5: esecuzione di Area = Area / 2

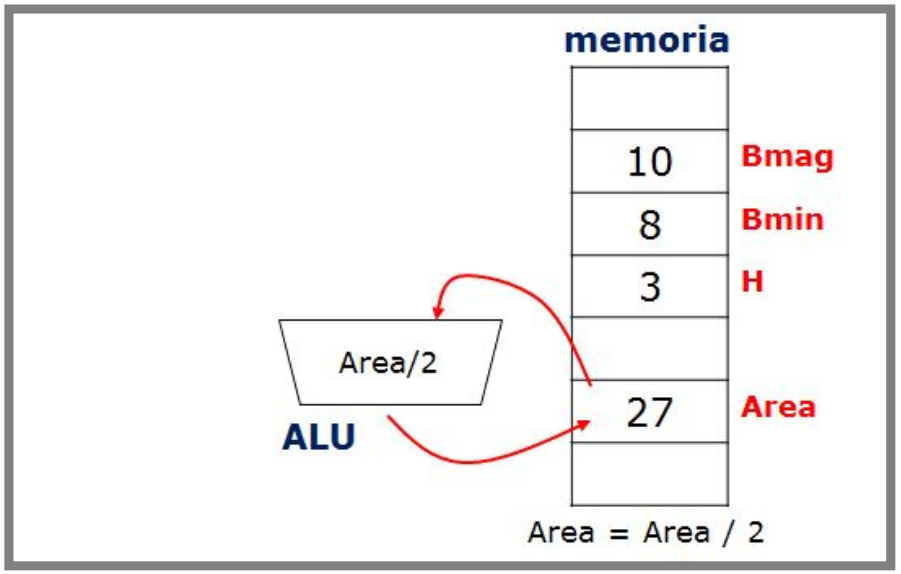


Fig.5: esecuzione di "Area = Area / 2"

Fig.6: esecuzione di print Area

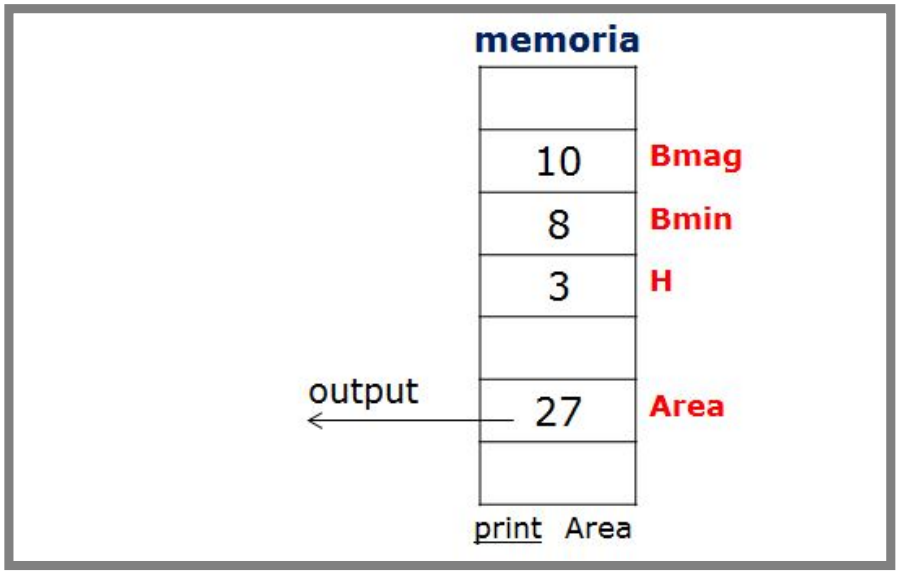


Fig.6: esecuzione di "print Area"

Un esempio

Scambio di due variabili

- Dati di input: A e B (ad es. 5 e 3)
- Dati di output: le stesse variabili A e B con i valori invertiti (3 e 5)

La coppia di istruzioni

A = B
B = A

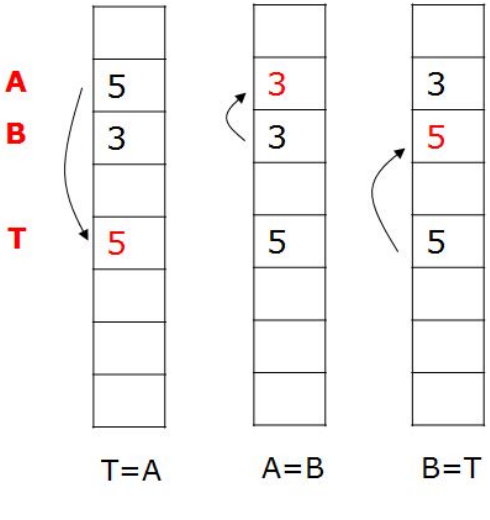
Non funziona !!

(si perde subito il valore di A)

necessaria una terza variabile (ad es. T) per salvare il valore di A ed assegnarlo successivamente a B

```
begin scambio
var  A,  B,  T: integer
read A, B
T = A
A = B
B = T
print A, B
end scambio
```

Algoritmo per lo scambio di due variabili



Evoluzione delle variabili A, B e T in memoria

Struttura di selezione

Spesso in un algoritmo occore eseguire operazioni diverse a secondo del valore (vero o falso) di una condizione logica

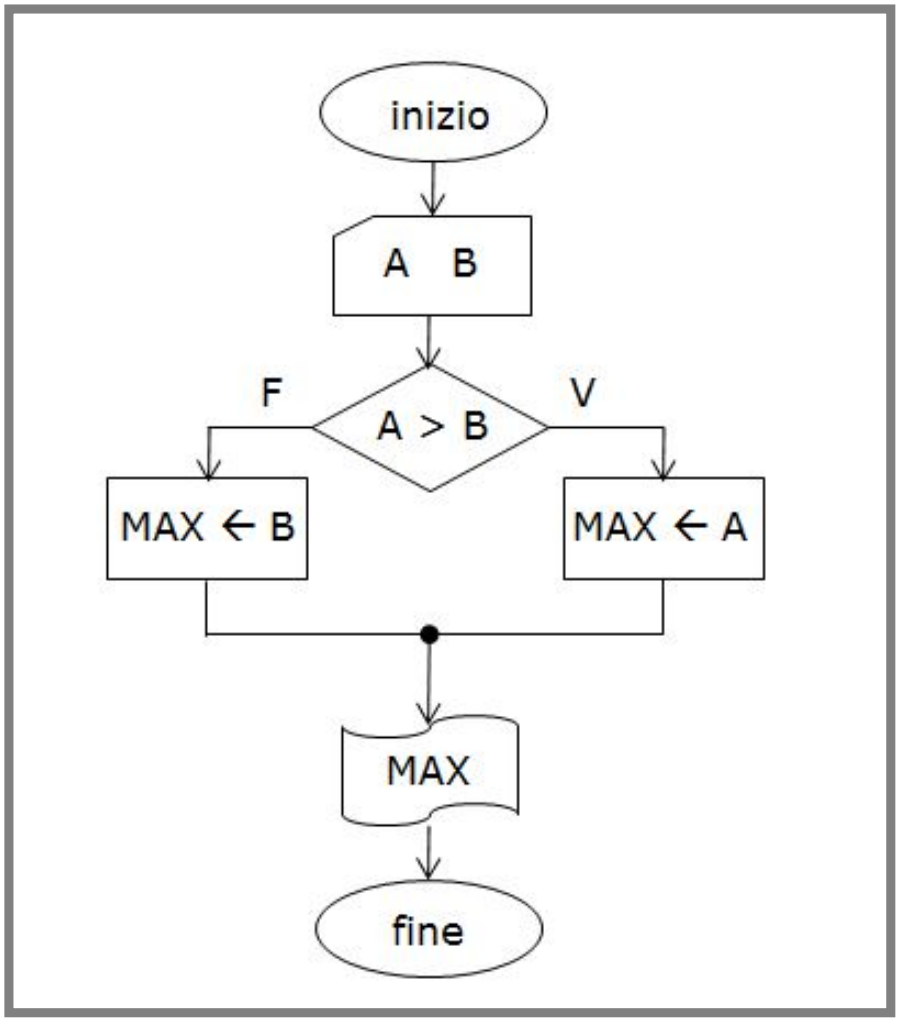
Esempio:

Il massimo tra due numeri

Dati di **input**: i due numeri (A e B)

Dati di **output**: il massimo (MAX)

Idea: si confrontano i due numeri (A e B). Se il primo e' maggiore del secondo il massimo (MAX) e' A, altrimenti e' B

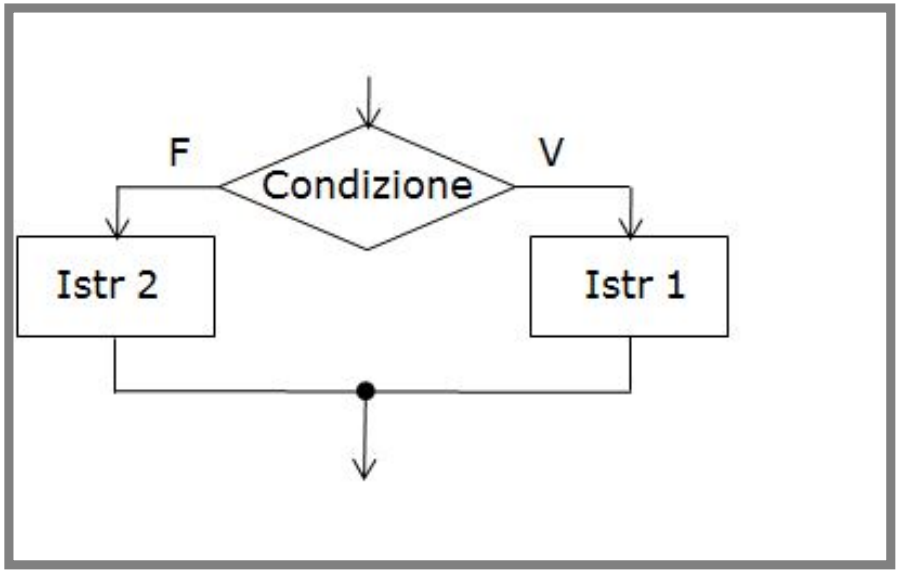


Flow chart del massimo di due numeri

Struttura di selezione

In Pascal like la struttura di selezione e' realizzata con la struttura

" if-then-else-endif "



Flow chart della struttura di selezione

In generale

- Viene valutata **l'espressione logica (if)**
- Se e' vera viene eseguito il blocco di istruzioni **prima di "else" (Istr 1)**
- se e' falsa viene eseguito il blocco di istruzioni **dopo di "else" (Istr 2)**

- L'istruzione **"endif"** segnala la fine della struttura

```
....
if (condizione) then
    Istr 1
else
    Istr 2
endif
....
```

Struttura di selezione in pascal-like

Algoritmo del massimo in pascal like

L'algoritmo del massimo
in pascal like e' allora

- Se (A > B) si assegna il valore di A alla variabile MAX
- Altrimenti si assegna il valore di B alla variabile MAX

- Dopo l'istruzione **"endif"** l'esecuzione dell'algoritmo riprende regolarmente

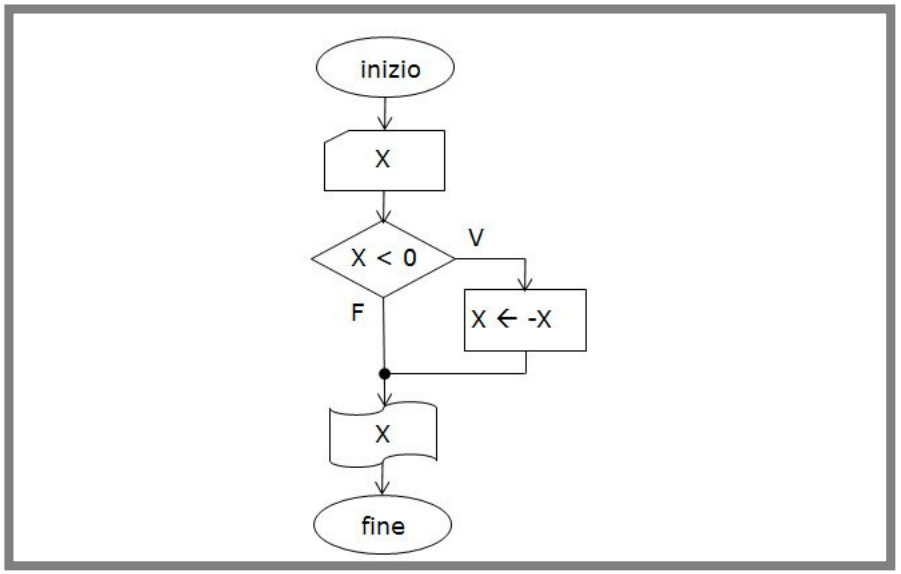
```
begin  massimo
var  A, B, MAX:  real
read   A, B
if  (A > B) then
     MAX = A
else
     MAX = B
endif
print   MAX
end  massimo
```

Algoritmo del massimo in pascal like

Struttura di selezione semplificata

in alcuni casi e' necessario eseguire istruzioni solo se la condizione e' vera, ma nessuna istruzioni se la condizione e' falsa

In questo caso manca il comando "else" e le relative istruzioni



il valore assoluto in flow chart

Esempio:

Valore assoluto di un numero

Dati di **input**: il numero X

Dati di **output**: lo stesso numero X, eventualmente cambiato di segno **se e' negativo**

```
begin valass
var X: real
read X
if (X < 0) then
    X = -X
endif
print X
end valass
```

il valore assoluto in pascal like

operatori relazionali e logici

Nella valutazione dell'espressione logica e' possibile verificare la veridicita' di **una o piu' condizioni basate su operatori relazionali**

> (maggiore) < (minore) == (uguale) /= (diverso)

combinate tra loro mediante **operatori logici**

AND OR NOT

Esempio:

(A > B) AND (X == 0)

Osservazioni

- Il risultato dell'espressione logica deve essere sempre **vero** o **falso**
- gli operatori relazionali vengono valutati prima degli operatori logici

Struttura di iterazione

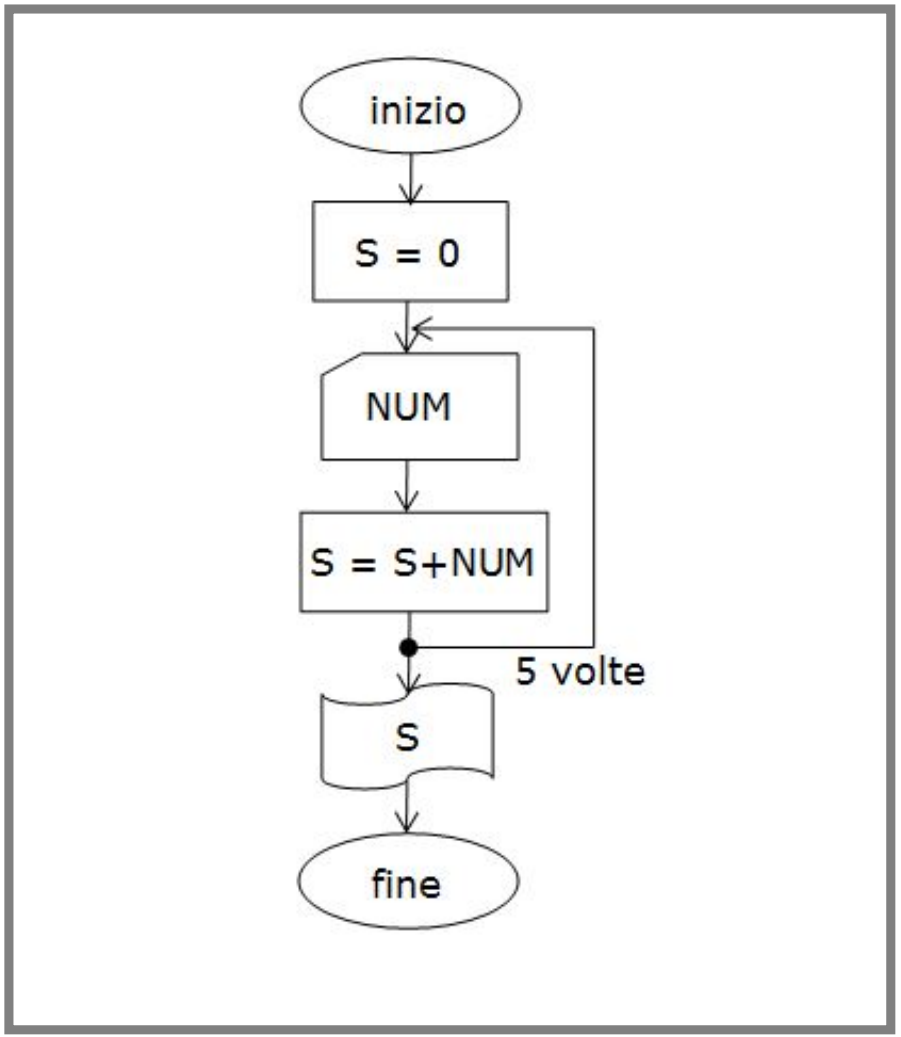
Spesso in un algoritmo e' necessario ripetere piu' volte lo stesso blocco di istruzioni

Esempio:
Somma di 5 numeri

Dati di **input**: i 5 numeri

Dati di **output**: la somma

Idea: leggere un numero alla volta (NUM) e sommarlo ad una variabile inizialmente posta uguale a zero (S)

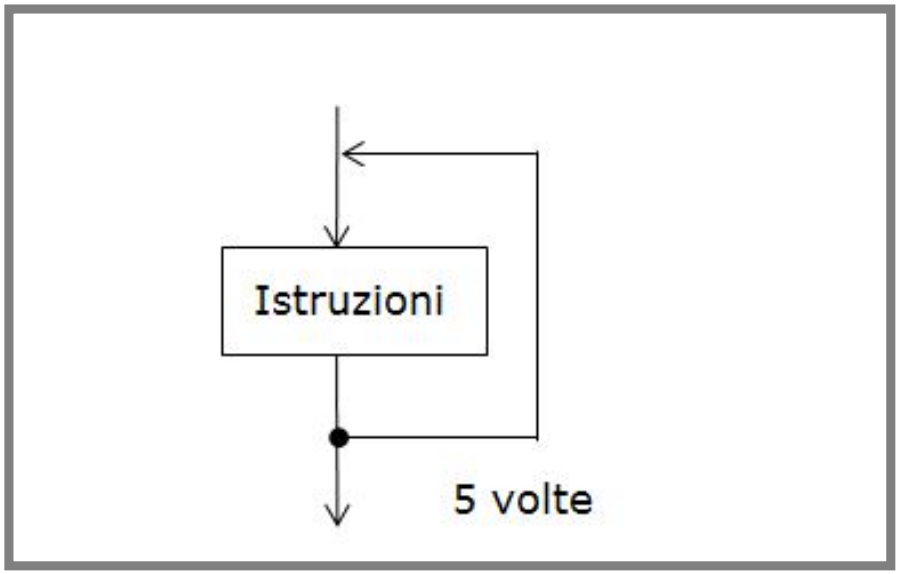


Flow chart dell'algoritmo per la somma di 5 numeri

Struttura di iterazione in pascal like

In Pascal like la struttura di iterazione e' realizzata con la struttura

" for - endfor "



Flow chart della struttura di iterazione "for-endfor"

In generale

- Vengono eseguite le istruzioni tra **"for"** e **"endfor"** per tutti i valori dell'indice **I** compresi tra il primo e l'ultimo valore

- L'indice I e' una variabile a tutti gli effetti e va dichiarata

```
....
for  I = 1 to 5
     Istruzioni
endfor
....
```

La struttura di iterazione "for-endfor" in pascal like

Algoritmo della somma in pascal like

L'algoritmo della somma
in pascal like e' allora

- La coppia di istruzioni
  read NUM
  S = S + NUM
  Viene ripetuta 5 volte con valori dell'indice
  I = 1, 2, 3, 4, 5

- Dopo l'istruzione **"endfor"** l'esecuzione dell'algoritmo riprende regolarmente

```
begin somma

var S, NUM: real

var I : integer

S = 0

for  I = 1 to 5

    read NUM

    S = S + NUM

endfor

print  S

end somma
```

Algoritmo della somma in pascal like

Esempio di esecuzione

Dati di input: 5; 6; 1; 8; -1

Primo passo (I=1)
NUM = 5 S = S+NUM (S = 5)

Primo passo (I=2)
NUM = 6 S = S+NUM (S = 11)

Primo passo (I=3)
NUM = 1 S = S+NUM (S = 12)

Primo passo (I=4)
NUM = 8 S = S+NUM (S = 20)

Primo passo (I=5)
NUM = -1 S = S+NUM (S = 19)

Risultato: 19 (ultimo valore di S)

I	S			NUM			
1	5			5			Primo passo
2	11			6			Secondo passo
3	12			1			Terzo passo
4	20			8			Quarto passo
5	19			-1			Quinto passo

Evoluzione dei dati in memoria nel corso dell'algoritmo della somma

In generale

La struttura di iterazione "for – endfor" esegue il blocco di istruzioni per tutti i valori dell'indice I compresi tra le variabili intere Start e End

Start: primo valore della variabile I

End: ultimo valore della variabile I

E' possibile specificare un valore opzionale anche negativo (Passo) che rappresenta l'incremento della variabile I (se manca, Passo=1)

```
begin algoritmo
var I, Start, End, Passo: integer
...
for I = Start to End step Passo
    istruzioni
endfor
...
end algoritmo
```

Sintassi generale della
struttura di iterazione "for-endfor"_

Esempio: Start=11, End=5, Step=-2

for I = Start to End step Passo

produce la sequenza di valori per l'indice I:

11, 9, 7, 5

Algoritmo della somma (II versione)

E’ possibile avere una versione piu’ generale dell’algoritmo

Somma di N numeri

In questo caso prima della struttura “for-endfor” si definisce la variabile N, che si usa come valore finale dell’indice I

si ottiene un algoritmo valido per sommare qualunque insieme di numeri

```
begin  somma
var S,  NUM:  real
var I , N : integer
read N
S = 0
for   I = 1 to N
    read  NUM
    S = S + NUM
endfor
print   S
end  somma
```

Algoritmo della somma di N numeri in pascal like

Strutture innestate

Le strutture di controllo
possono essere innestate l'una nell'altra

- La prima struttura che si apre deve essere l'ultima che si chiude
- In caso di piu' strutture "for-endfor" e' necessario utilizzare indici diversi
- Spostare verso destra le istruzioni interne ad una struttura aiuta la leggibilita' dell'algoritmo

```
begin algoritmo
var I, J, K : integer
...
for  I = 1 to N
    for J = 1 to M
        for K = 1 to L
            if (condizione ) then
                istruzioni
            endif
        endfor
    endfor
endfor
...
end algoritmo
```

Esempio di strutture di controllo innestate

Esempio: il massimo di N numeri (positivi)

Dati di input:

N = 5 (la quantita' di numeri da esaminare)

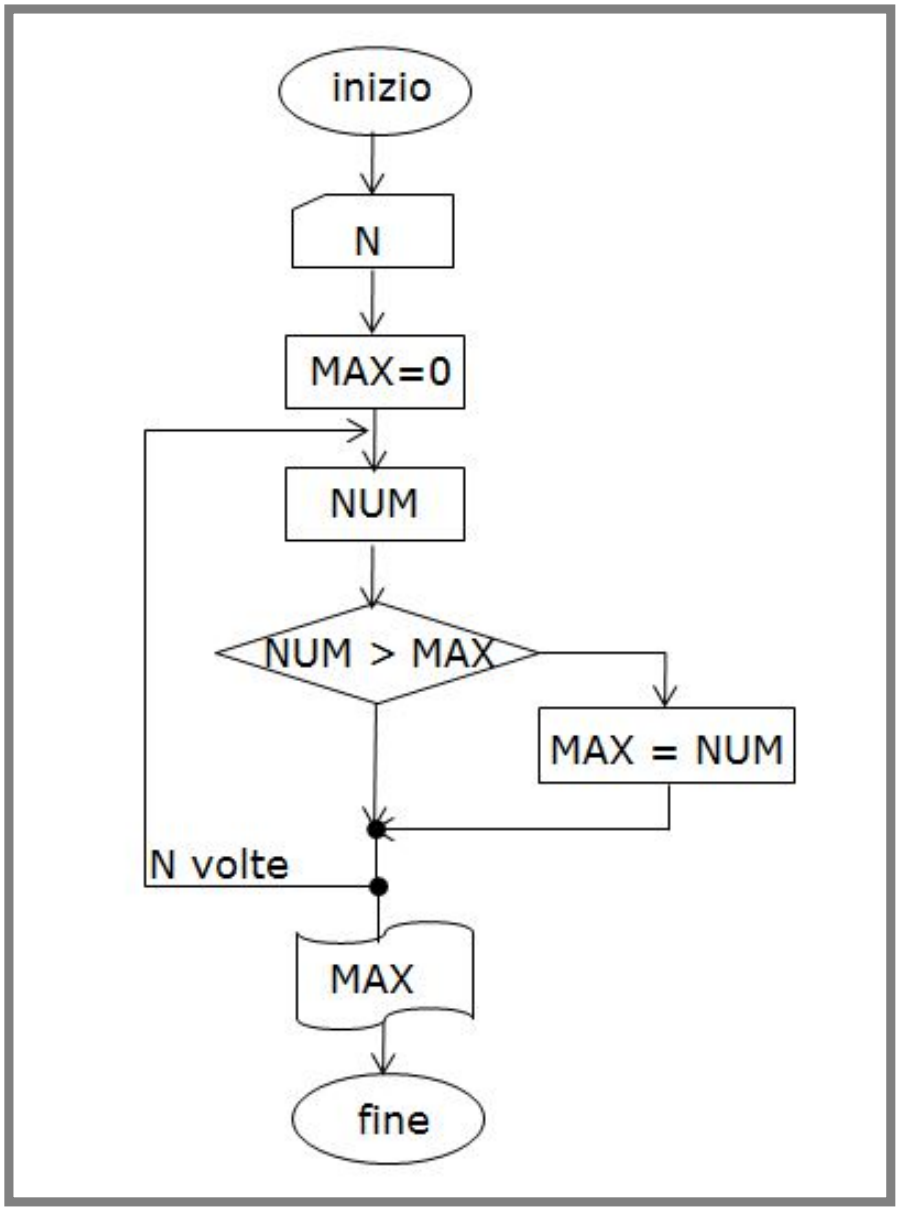
NUM = 5; 6; 1; 8; 2

Risultato: Il massimo (MAX=8)

Idea: leggere un numero alla volta (NUM) e confrontarlo con la variabile che contiene il massimo dei numeri esaminati (MAX). Se NUM e' maggiore di MAX si cambia il valore di MAX

2 strutture innestate"

- "for - endfor" per leggere i numeri uno alla volta
- "if-endif" per confrontare NUM e MAX



Flow-chart per la ricerca del massimo

Algoritmo del massimo in pascal-like

Dati di input: 5; 6; 1; 8; 2

Primo passo (I=1)
NUM = 5 NUM > MAX ? (si) → MAX=5

Primo passo (I=2)
NUM = 6 NUM > MAX ? (si) → MAX=6

Primo passo (I=3)
NUM = 1 NUM > MAX ? (no)

Primo passo (I=4)
NUM = 8 NUM > MAX ? (si) → MAX=8

Primo passo (I=5)
NUM = 2 NUM > MAX ? (no)

Risultato: 8 (ultimo valore di MAX)

```
begin massimo
var MAX, NUM: real
var I , N : integer
read N
MAX = 0
for I = 1 to N
    read NUM
    if (NUM> MAX) then
        MAX = NUM
    endif
endfor
print MAX
end massimo
```

Algoritmo della ricerca del massimo in pascal-like

Esempio

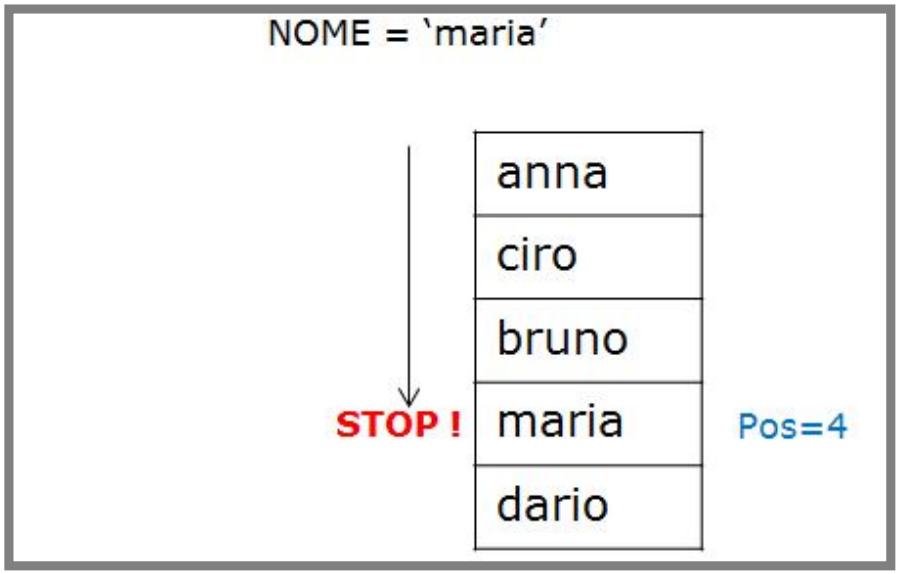
Dato un array di nomi (un elenco) di lunghezza N determinare la posizione nell'array di un nome da cercare

Dati di input:

- N (lunghezza dell'elenco)
- Elenco(1),..,Elenco(N) (l'elenco)
- NOME (nome da cercare)

Dati di output

- Pos (la posizione di NOME in Elenco)



Ricerca di 'maria' nell'array Elenco

IDEA

Esaminare uno alla volta gli elementi di Elenco, e confrontarli con il NOME

RICERCA SEQUENZIALE

Problema: quando terminare?

Ci si deve arrestare quando:

- Si trova il NOME nell'Elenco

  oppure

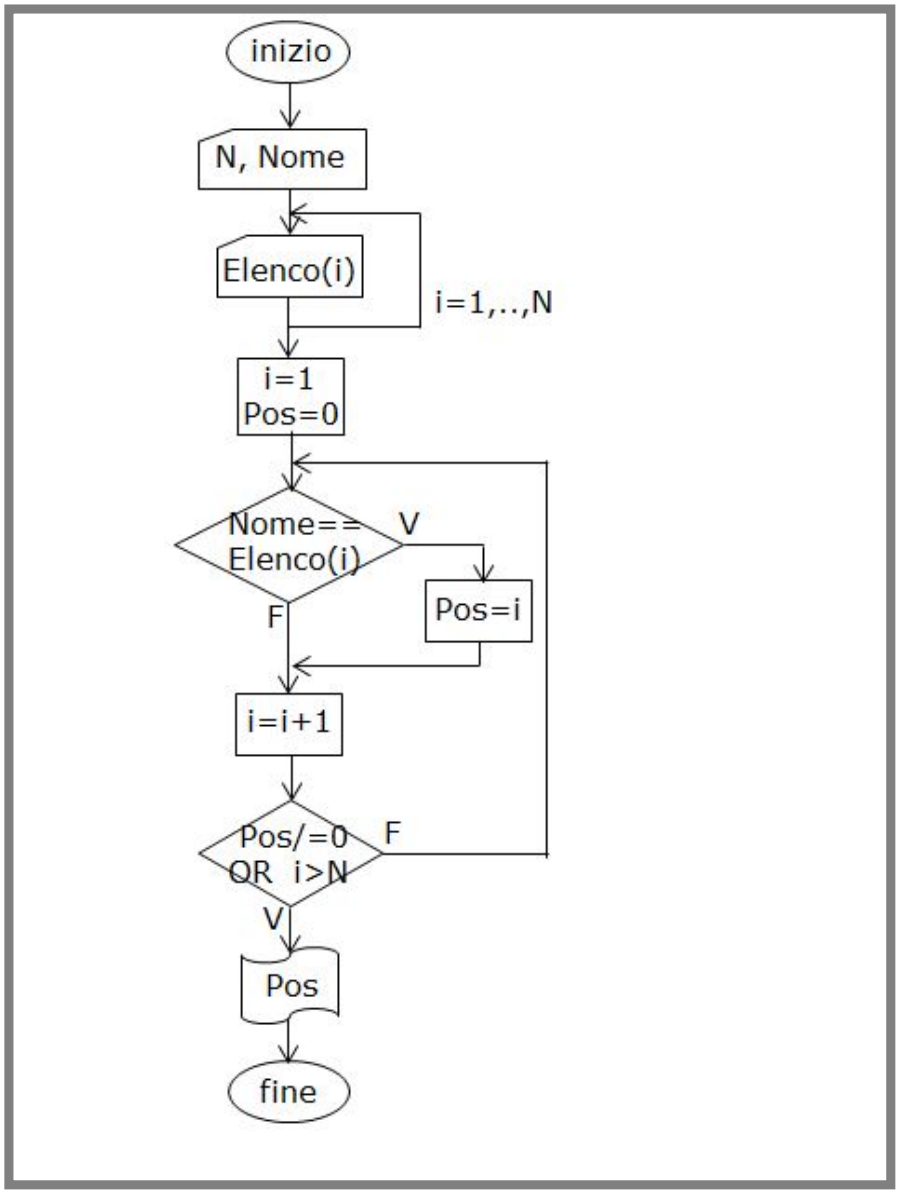
- Si scorre tutto l'Elenco ma NOME non e' presente (si assume Pos=0)

Esempio1: N = 5; NOME = 'maria';
Elenco=('anna', 'ciro', 'bruno', 'maria', 'dario')

Pos = 4 (i=4 iterazioni)

Esempio2: N = 5; NOME = giacomo;
Elenco=('anna', 'ciro', 'bruno', 'maria', 'dario')

Pos = 0 (i=5 iterazioni)

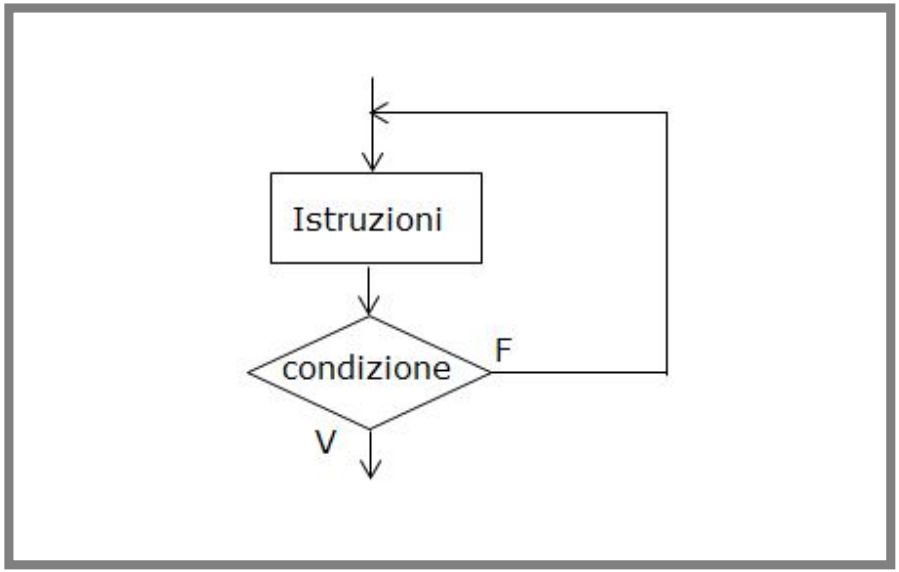


Flow chart della ricerca sequenziale

La struttura "repeat-until" in pascal like

In pascal-like una tale struttura di iterazione e' realizzata con la struttura

"repeat-until "



Flow chart della struttura "repeat-until"

In generale

- Vengono eseguite ripetutamente le **Istruzioni** comprese tra "**repeat**" e "**until**"
- Ci si arresta quando la **condizione posta alla fine della struttura risulta vera**

```
repeat
    Istruzioni
until (condizione)
```

La struttura "repeat-until" in pascal-like

Ricerca sequenziale in pascal-like

Osservazione:

In generale la condizione di arresto puo' essere la combinazione di piu' operazioni logiche legate dagli operatori OR, AND e NOT

L'algoritmo per la ricerca sequenziale in pascal like e' allora

```
begin  ricbin
var   i, N, Pos : integer
var  Elenco(10), NOME : array of character
read  N
Pos = 0
i = 1
repeat
     if ( Elenco(i) == NOME ) then
          Pos = i
     endif
     i = i+1
until ( Pos /= 0  OR   i > N)
print  Pos
end  ricbin
```

Ricerca squenziale in pascal-like

"for-endfor" e "repeat-until"

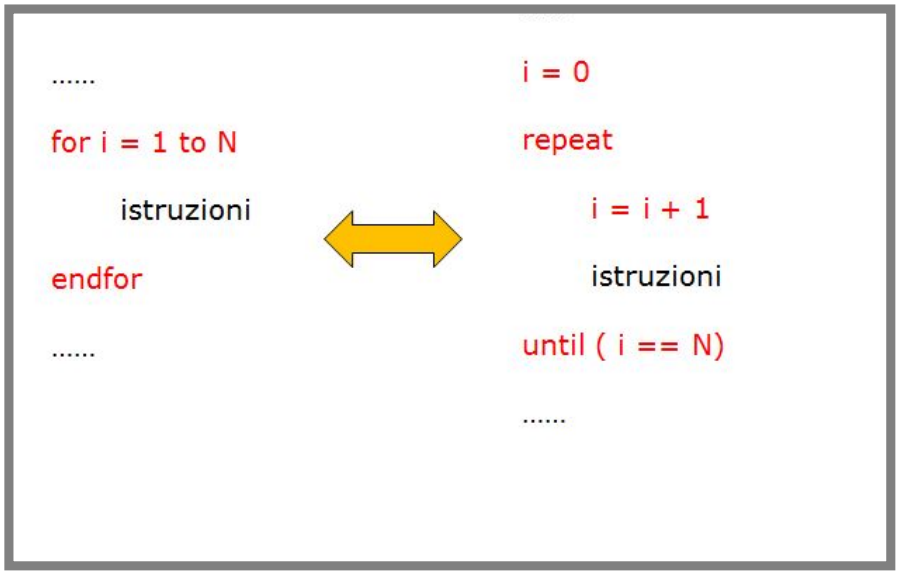
Osservazione:

In generale la condizione di arresto puo' essere la combinazione di piu' operazioni logiche legate dagli operatori OR, AND e NOT

La struttura "repeat-until" e' piu' flessibile e generale della struttura "for-endfor"

Una struttura "repeat-until" con una condizione di arresto solo sul numero di passi e' equivalente ad una struttura "for-endfor"

L'indice i va gestito esplicitamente



Equivalenza tra "for-endfor" e "repeat-until"

esempio

La struttura repeat-until e' utile per far si che un dato di input sia corretto

Esempio:

Leggere un array di lunghezza N, assicurandosi prima che N > 0

Idea :

Leggere ripetutamente N fino a che N>0

```
begin  leggi
var   A(100) array of float
var   i, N :integer
repeat
     read N
until ( N > 0)
for i = 1 to N
     read A(i)
endfor
end leggi
```

Lettura con controllo di un dato di input

Esempio: calcolo del M.C.D.

M.C.D. tra A e B (non entrambi nulli)

Metodo delle divisioni successive
(o di Euclide)

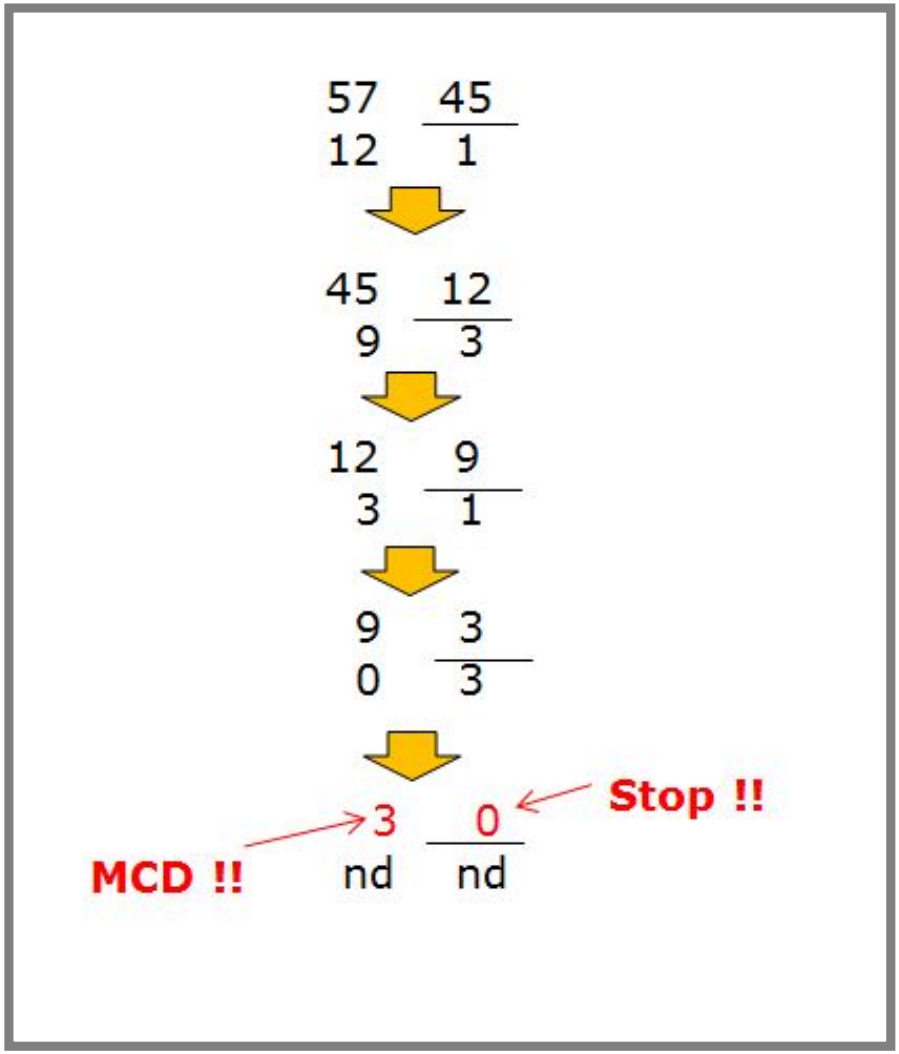
Si DIVIDE il MAGGIORE di essi per il MINORE;

Se il RESTO della divisione è ZERO, il M.C.D. è il numero MINORE (ovvero il divisore);

Se il RESTO della divisione è DIVERSO da ZERO si DIVIDE il numero MINORE per tale RESTO.

Si continua così fino ad ottenere una divisione per ZERO. Il M.C.D. cercato è il DIVIDENDO dell'ultima divisione.

Esempio: M.C.D.(57,45) = 3



Il metodo delle divisioni successive per il MCD

Un possibile algoritmo

E' necessario effettuare
la divisione (intera) tra A e B

Calcolare
- Il quoziente Q
- Il resto R

Rinominare A e B per la successiva divisione

Ripetere finche' B==0

PROBLEMA

Cosa accade se il dato di input B=0?

```
begin  MCD
var   A, B, Q, R , MCD : integer
read  A,  B
repeat
    Q = A/B
    R = A – B*Q
    A = B
    B = R
until ( B == 0)
MCD = A
print   A
end  MCD
```

Un possibile algoritmo in pascal like per il MCD tra A e B

Problema

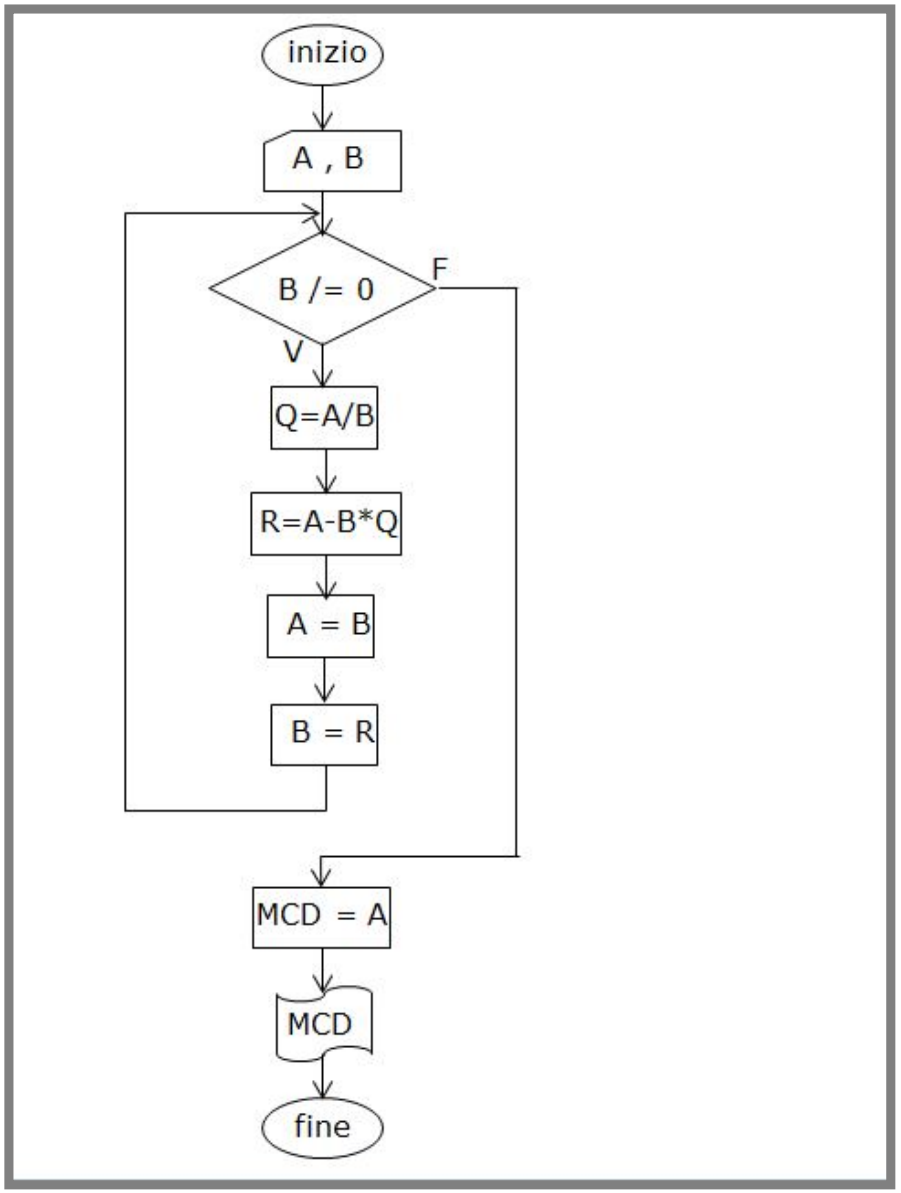
Se il minore dei due numeri (cioe' B) e' uguale a zero, non e' possibile eseguire l'algoritmo

Ma il MCD con B=0 e' definito !

Es. M.C.D. (5,0) = 5 (cioe' A)

Soluzione:

Eseguire il test su B **PRIMA** della divisione



Flow chart per il MCD (seconda versione)

La struttura iterativa “while-endwhile”

La struttura di iterazione

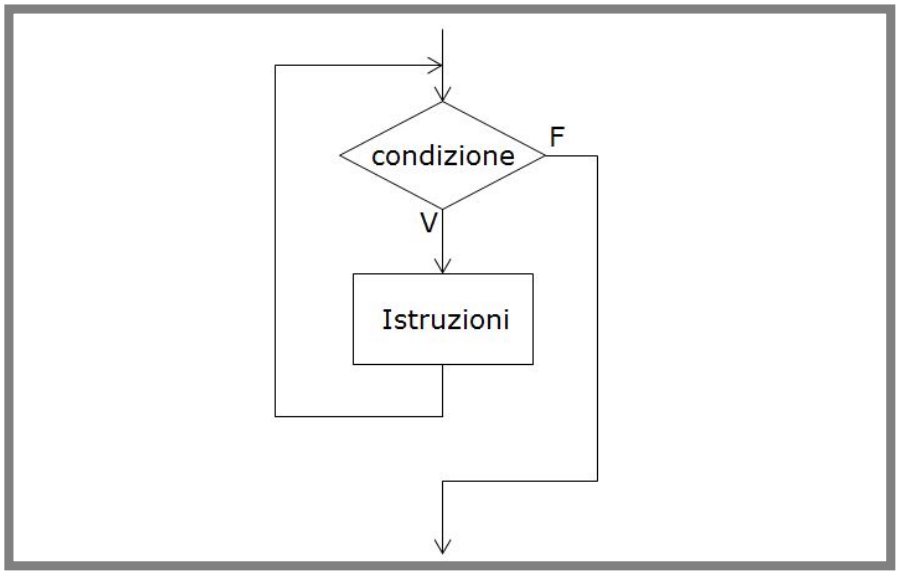
while-endwhile

Esegue ripetutamente le istruzioni comprese tra “while” e “endwhile”, ma a differenza del repeat-until

PRIMA valuta la condizione

DOPO esegue le istruzioni

Le istruzioni vengono eseguite se la condizione risulta VERA



Flow chart della struttura while-endwhile

```
while (condizione)
    Istruzioni
endwhile
```

La struttura while-endwhile in pascal like

Algoritmo del MCD in pascal like

L'algoritmo del MCD puo' allora essere riscritto utilizzando la struttura while-endwhile

In questo modo l'algoritmo puo' essere utilizzato anche quando il dato di input B=0

```
begin  MCD2
var   A, B, Q, R , MCD : integer
read  A,  B
while (B /= 0)
     Q = A/B
     R = A – B*Q
     A = B
     B = R
endwhile
MCD = A
print   A
end  MCD
```

Algoritmo del MCD in pascal like (seconda versione, con struttura while-endwhile)

Esecuzione dell'algoritmo

Esempio:

A = 57 B = 45

Dopo 4 iterazioni del ciclo while il MCD si trova nella variabile A

	A	B		Q	R		memoria	
Prima del ciclo while	57	45						
Al termine del 1° passo	45	12		1	12			
Al termine del 2° passo	12	9		3	9			
Al termine del 3° passo	9	3		3	3			
Al termine del 4° passo	3	0		3	0			

Evoluzione dei dati in memoria nel corso dell'algoritmo

Esempio:

A = 57 B = 0

Il ciclo while non viene eseguito e il MCD si trova nella variabile A