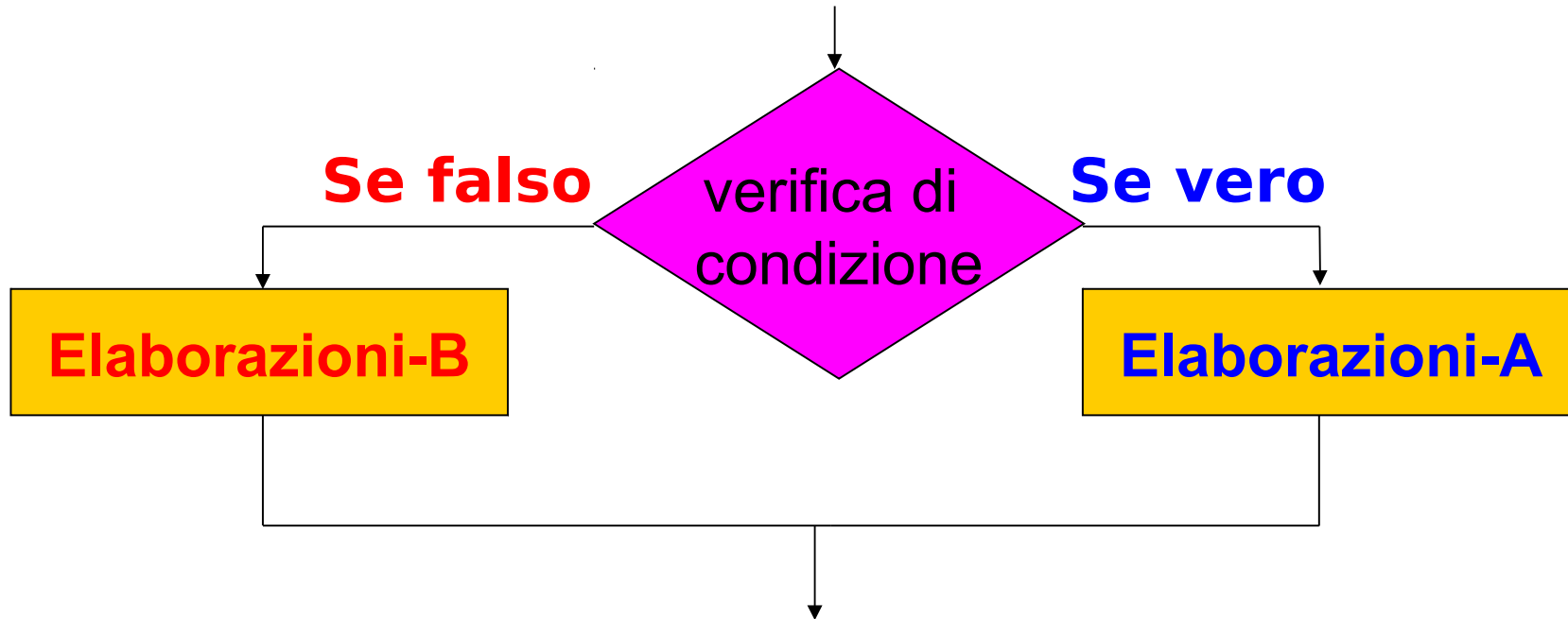


Programmazione al livello del CPU



Esecuzione condizionale

Eseguire blocco "A" o blocco "B".



La memoria RAM è lineare
(non ha una natura spaziale, come, ad esempio, il cervello)
Come implementare questa schema ?



Confronto

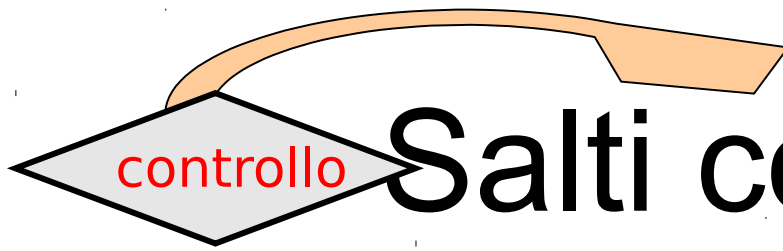
- **COMP $r1, r2$** confronta i valori-*integer* dei registri $r1$ e $r2$
Risultato: Se $Rr1 < Rr2$, memorizza -1 nel registro **RC**
Se $Rr1 = Rr2$, memorizza 0
Se $Rr1 > Rr2$, memorizza +1
- **COMP $r1, cost$** confronta il registro $r1$ con *la costante cost*
- **FCOMP $r1, r2$** confronta valori del tipo *float*

COMP Operazione registro1 registro2 bit non-utilizzati

[0010 0000] [iiii] [jjjj] [xxxxxxxx xxxxxxxx]

COMP Operazione registro costante

[1010 0000] [iiii] [xxxx xxxx xxxx xxxx xxxx]



Salti condizionali

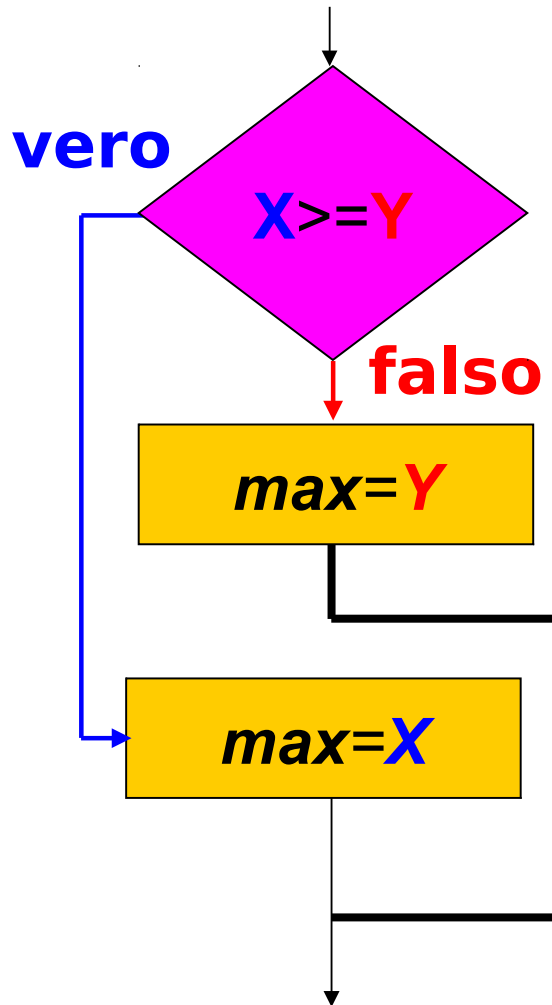
- **BRLT *addr*** se $RC = -1$, spostare il PC all'*addr*
- **BRLE *addr*** se $RC \leq 0$, spostare il PC all'*addr*
- **BREQ *addr*** se $RC = 0$, spostare il PC all'*addr*
- **BRNE *addr*** se $RC \neq 0$, spostare il PC all'*addr*
- **BRGT *addr*** se $RC = 1$, spostare il PC all'*addr*
- **BRGE *addr*** se $RC \geq 0$, spostare il PC all'*addr*

BRLT Operazione bit non-utilizzati nuovo indirizzo per il PC

[0100 0001] [xxxx] [aaaa aaaa aaaa aaaa aaaa]

Esempio: scegliere uno dei due numeri

Compito: assegnare alla variabile Z il massimo dei due numeri X e Y



```
X: INT 5;
```

```
Y: INT -3;
```

```
Z: INT;
```

```
LOAD R0 X;
```

```
LOAD R1 Y;
```

```
COMP R0 R1;
```

```
BRGE Big0; % Se  $R0 > R1$ 
```

```
STORE R1 Z; %  $R0 < R1 \Rightarrow Z = R1$ 
```

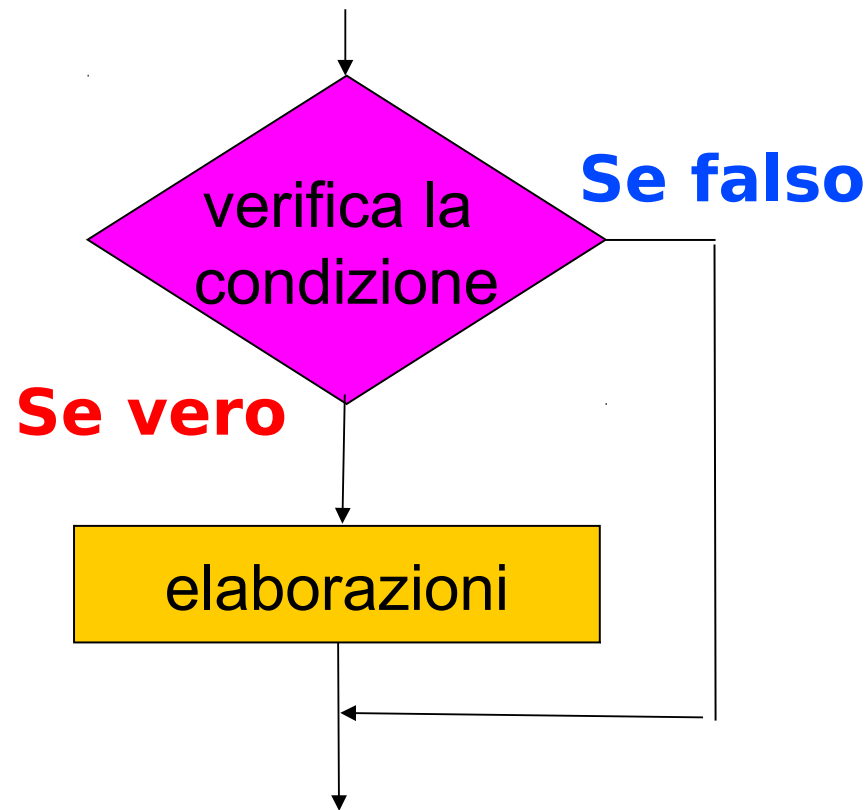
```
BRANCH FINE;
```

```
Big0: STORE R0 Z; %  $R0 > R1 \Rightarrow Z = R0$ 
```

```
FINE: STOP;
```

Esecuzione condizionale (un blocco)

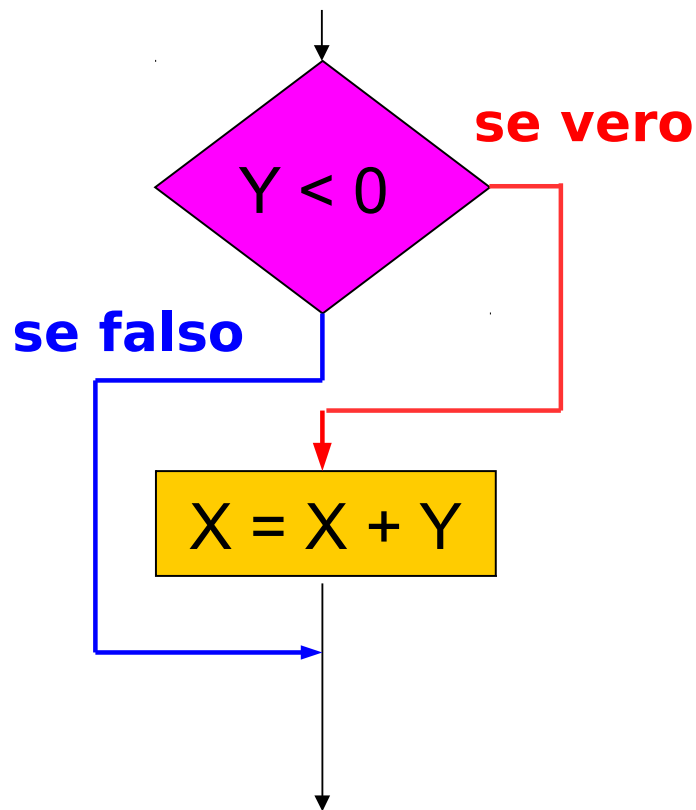
Eseguire un blocco di elaborazioni solo se una condizione è vera



Elaborazione condizionale

Per implementare l'esecuzione condizionale di un blocco di elaborazioni, andare all'inizio di questo blocco se la condizione è vera. Altrimenti saltare il blocco.

Esempio: aggiungere Y al X solo se $Y < 0$



```
X: INT 5;
```

```
Y: INT -3;
```

```
LOAD R0 X;
```

```
LOAD R1 Y;
```

```
COMP R1 0;
```

% $Y \geq 0$?

```
BRLT Fare;
```

% YES? Fare la somma

```
BRANCH Fine;
```

% NO? Non fare la somma

```
Fare: ADD R0 R1;
```

% $Acc = Acc + X$

```
STORE R0 X;
```

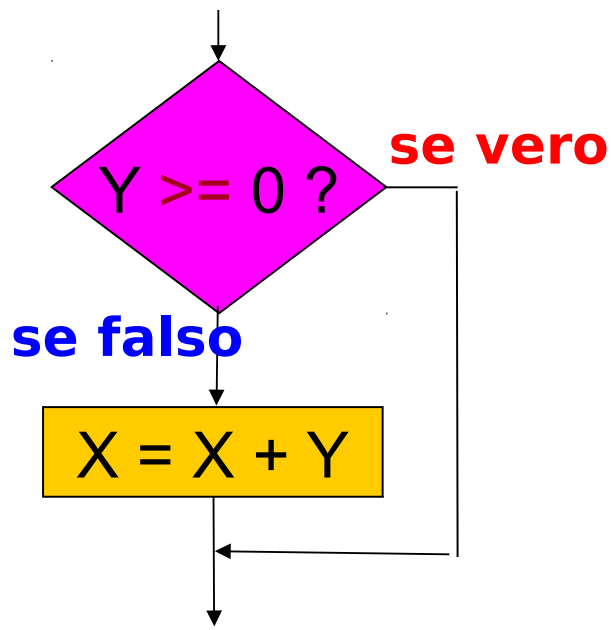
% $Y + X \rightarrow X$

```
Fine: STOP;
```

La logica “complimentare”

Possiamo evitare il salto non-condizionale se **saltiamo il blocco da eseguire nel caso che la condizione complimentare (not A) è vera.**

Esempio:
Aggiungere Y al X
solo se $Y < 0$



```
X: INT 5;
Y: INT -3;

LOAD R0 X;
LOAD R1 Y;
COMP R1 0;           % Y >= 0 ?
BRGE Fine;          % YES? → FINE
ADD R0 R1;          % Acc=Acc+X
STORE R0 X;         % Y+X → X
Fine: STOP;
```

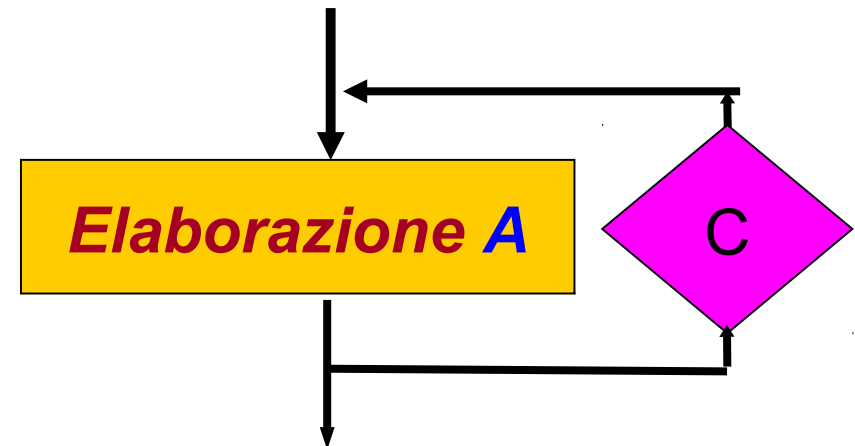

Elaborazioni iterative

Spesso serve ad eseguire più volte (iterare) un blocco di elaborazioni "A".



Due tipi di elaborazioni iterative (CICLI):

(i) **gestito da una condizione "C"** che dipende da un evento esterno o dal blocco di elaborazioni "A". **Assicurare la possibilità di uscire dal ciclo !!**

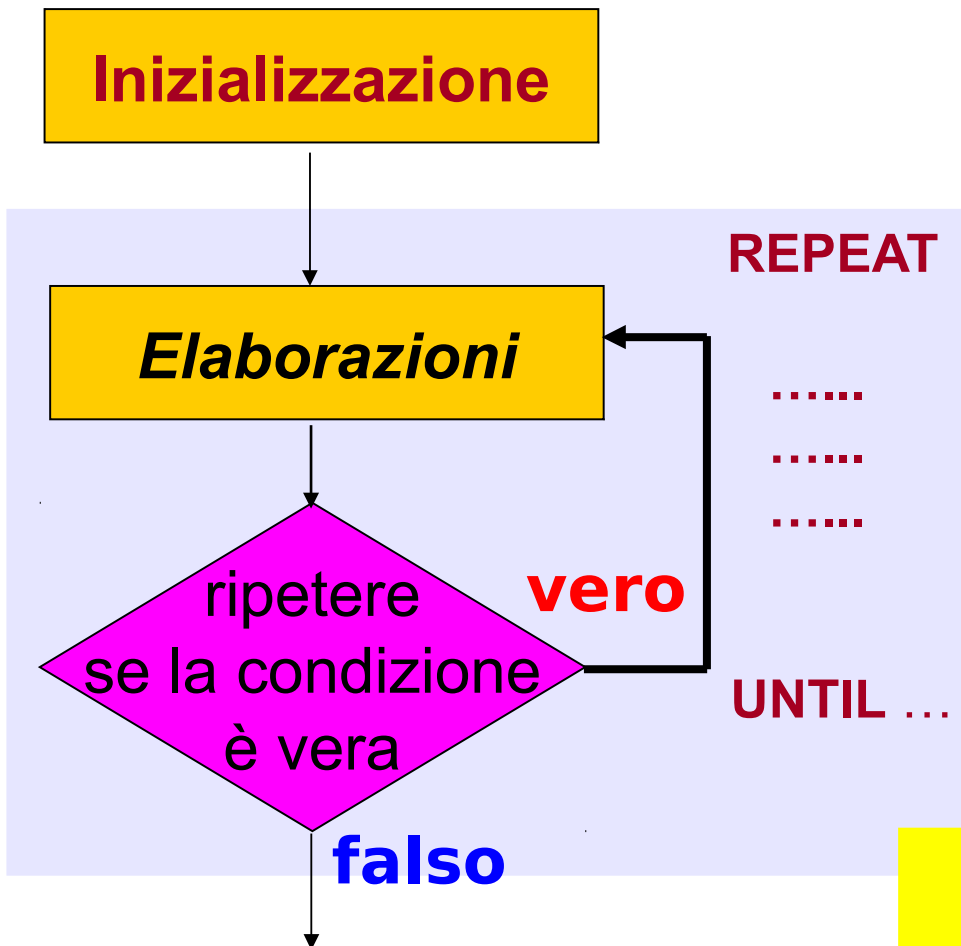


(ii) **eseguire il blocco di elaborazioni "A" esattamente N volte** (gestito da un contatore).



CICLI (1) Ripetere un'elaborazione finché la condizione **C** è **VERA**

Esempio: Sottrarre un numero k
da un altro N **finché** $N \geq 0$



```
N: INT 15
```

```
K: INT 4
```

```
LOAD R0 N
```

```
LOAD R1 K
```

```
Fare: SUB R0 R1 % N=N-K
```

```
COMP R0 0 % Se R0 >= 0 ..
```

```
BRGE Fare % .. ripetere
```

```
STORE R0 N
```

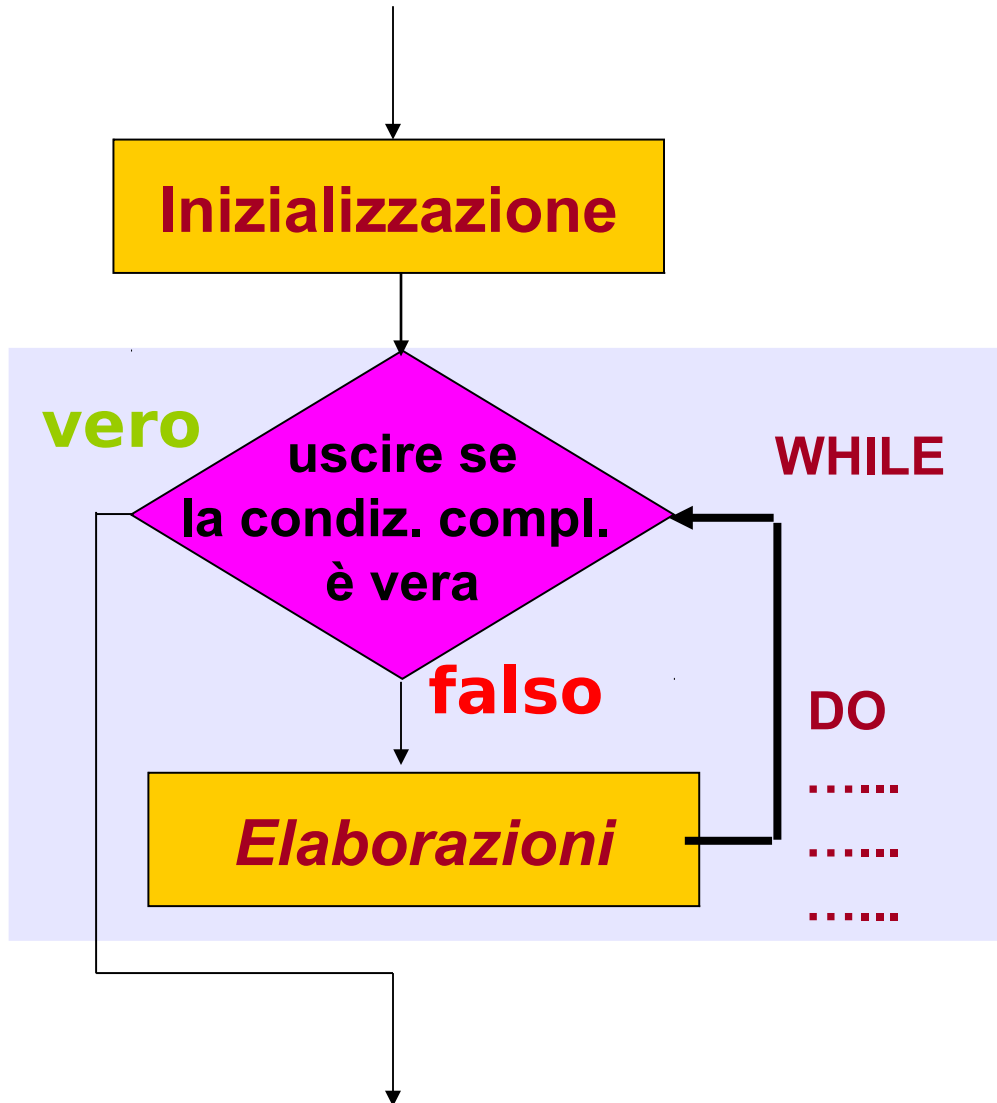
```
STOP
```

Se $N < 0$?

L'algorithmo non è corretto per tutti i dati possibili

CICLI(2) Ripetere un'elaborazione solo se e finché la condizione **C** è **VERA**

Esempio: Sottrarre un numero k da un altro N **solo e finché** $N \geq k$



Test:

```
N: INT 15
K: INT 6

LOAD R0 N
LOAD R1 K
COMP R0 R1      % Se R0<R1 ..
BRLT Post      % .. Uscire
SUB R0 R1       % N=N-K
BRANCH Test    % .. ripetere
Post: STORE R0 N
STOP
```

CICLI (3) Contatore

Definizione: un contatore è una **variabile** numerica che memorizza il numero di volte per i quali un certo evento si verifica.

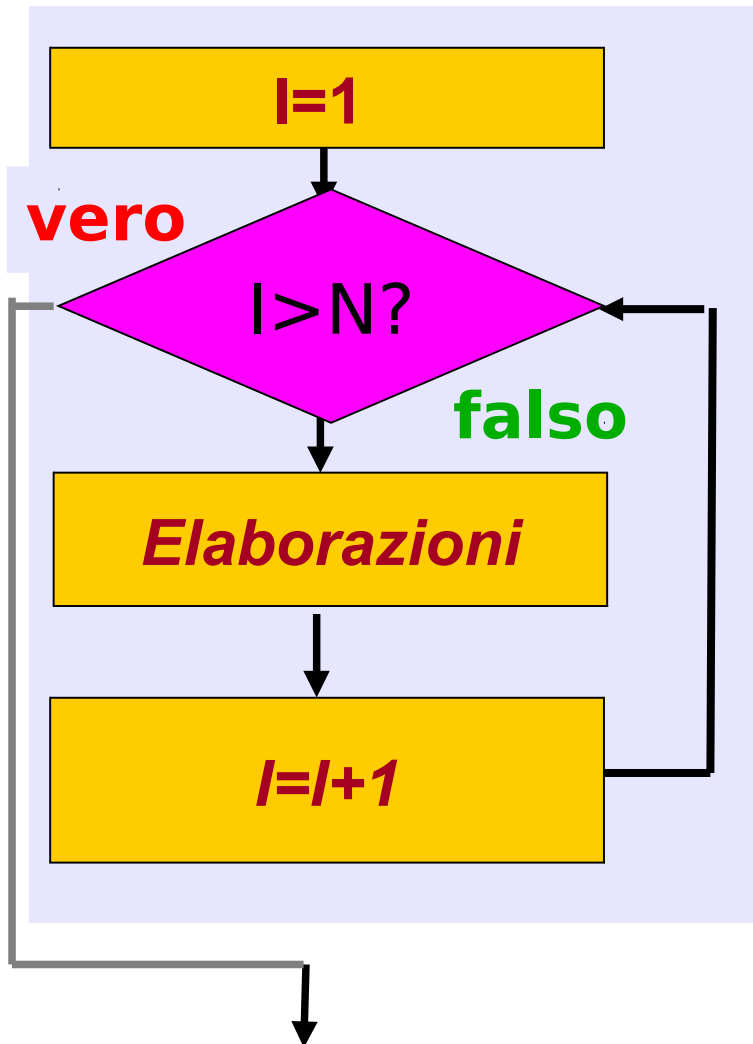
- Nei cicli viene utilizzato per controllare che un blocco di elaborazioni “**A**” viene eseguito esattamente per un certo numero di volte **N**.

Funzionamento:

- contatore progressivo
 - prima di entrare nel ciclo, inizializzare il contatore con 1
 - eseguire l'elaborazione se il contatore ha **valore minore o uguale a N**
 - ad ogni ripetizione incrementare il contatore con 1
- contatore regressivo:
 - inizializzare il contatore con N
 - eseguire l'elaborazione se il contatore ha **valore maggiore di 0**
 - ad ogni ripetizione decrementare il contatore con 1

CICLI (3a) Contatore progressivo

FOR I = 1 TO N



```
N: INT 4 % Numero elabor.
```

```
LOAD R1 N
```

```
LOAD R0 1 % contatore
```

```
COMP R0 R1 % Se R0<0 ..
```

```
BRGT Usc % .. Uscire
```

```
%% ELABORAZIONI %%
```

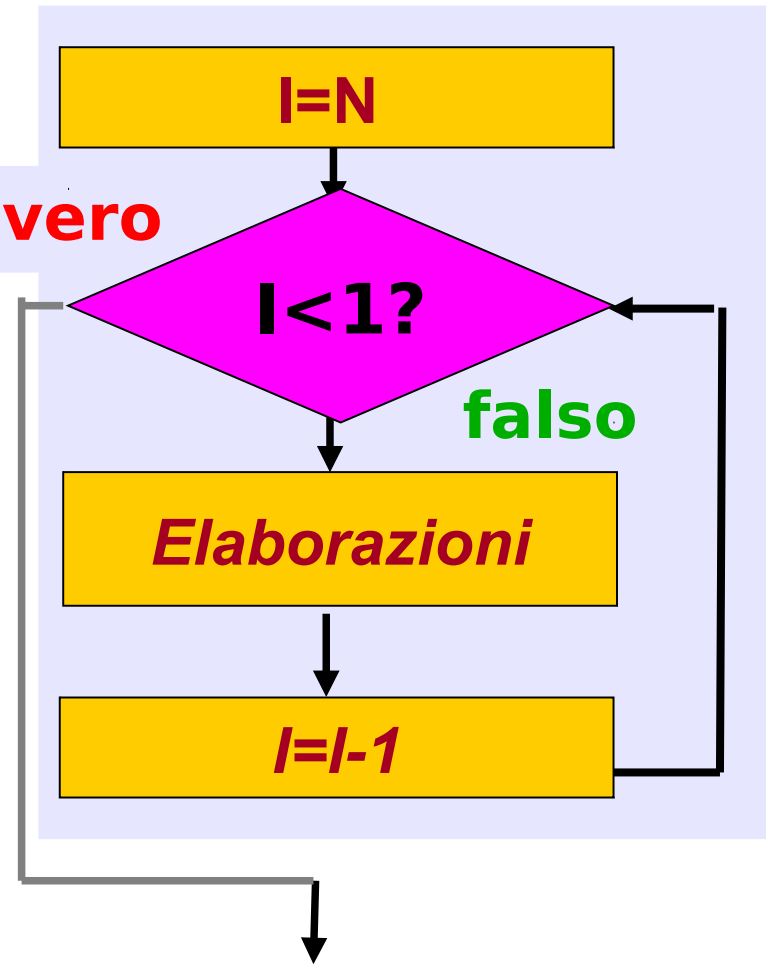
```
ADD R0 1 % contat.+1
```

```
BRANCH Test % → testa
```

```
Usc: STOP
```

CICLI (3b) Contatore regressivo

FOR I = N TO 1



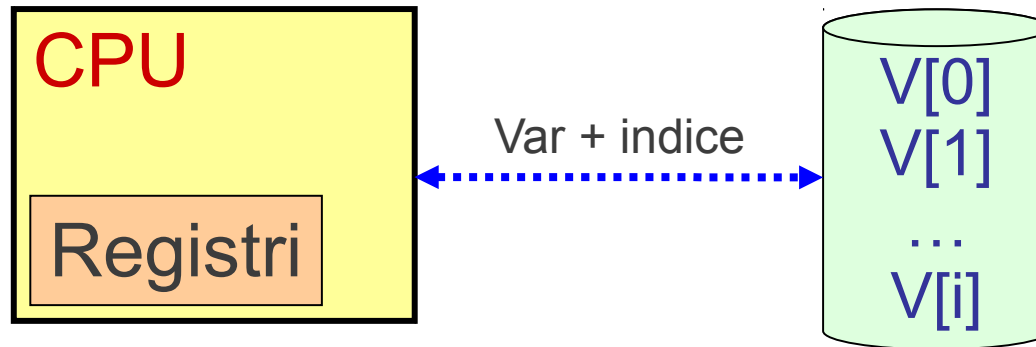
```
N: INT 4 % Numero elabor.

LOAD R0 N % contatore
testa: COMP R0 1 % Se R0<0 ..
      BRLT Usc % .. Uscire
      %% ELABORAZIONI %%
      SUB R0 1 % contat.-1
      BRANCH Test % → testa
Usc: STOP
```

Strutture dati: Vettori

- Per poter elaborare più valori, occorre organizzarne in una **struttura dati**. L'efficienza degli algoritmi dipende dalla struttura utilizzata.
- La struttura più semplice è il **vettore**, nel quale **N** valori sono memorizzati in **N** successive celle (*struttura lineare*).
- Un vettore **X** (o **X[]**) è localizzato nella RAM partendo da un certo indirizzo (del vettore **X**).
- Ogni elemento del vettore **X** viene riferito tramite un **indice i** che varia da **0** a **N-1** (elemento **i**: **X[i]**)
- L'indice i di ogni elemento **X_i** determina il suo indirizzo relativo rispetto l'indirizzo del vettore **X**, (*i * byte-per-codificare-un-valore-X_i*)
- L'indirizzo assoluto (nella RAM) di ogni elemento **X_i** è uguale all'*indirizzo del vettore X* più *l'indirizzo relativo di X_i*.

Riferimento a dati vettoriali



- **LOAD R , Var , Ri**
LOAD R , Var , i carica nel registro-dati R il contenuto del elemento $[Ri]/i$ della variabile Var (l'indice = contenuto del registro Ri)
- **STORE R , Var , Ri**
STORE R , Var , i copia il contenuto del registro dati R nel elemento $[Ri]/i$ della variabile Var

In *microcalc*, ogni variabile potrebbe essere un vettore (massimo 20 elementi):
V: INT 10 15 20 25 30; - definisce i valori iniziali di un vettore V (i=0,1,2,3,4)
LOAD R1 V 4; - carica in R1 il 5° elemento di V
STORE R1 V 8; - imposta il valore del 9° elemento di V

ACCUMULATORE

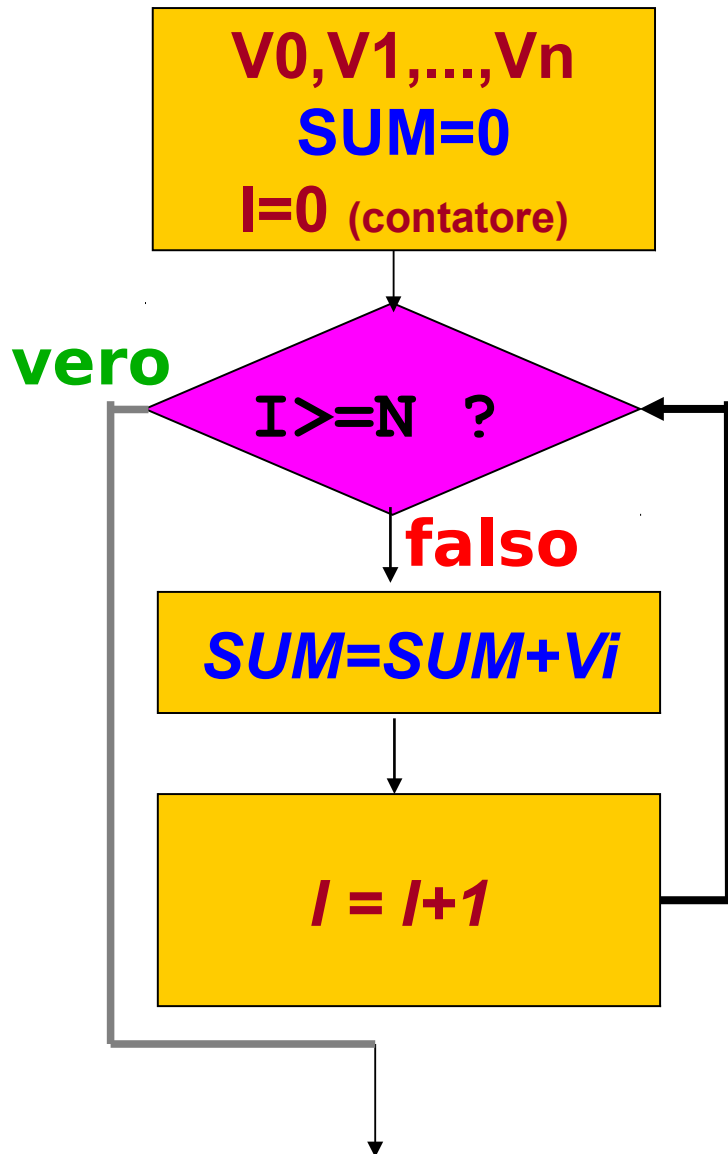
- Se abbiamo istruzioni per sommare (moltiplicare, ecc) due numeri scalari, come possiamo applicare la stessa operazione su più numeri ?
- Soluzione: definire una variabile *accumulatore A* ed applicare la stessa operazione tra A e ogni valore in questione.
- **Algoritmo**: dati i n valori ($X_1, X_2, \dots X_n$)
 - 1) **inizializzare** l'accumulatore Y
 - 2) **applicare l'operazione tra l'accumulatore e ogni valore X_i** , memorizzando il risultato nell'accumulatore.
- **Inizializzazione**:

Metodo A: “**azzerare**” per accumulare poi tutti gli elementi X_i

 - per *sommare / moltiplicare*, inizializzare con 0 / 1
 - per *max / min*, inizializzare con un numero piccolo / grande.

Metodo B: inizializzare l'accumulatore con il **primo valore X_1** , per accumulare poi con tutti i valori tranne il primo.

SOMMA VETTORIALE



```
V: INT 3 10 2 -1 -4; Vettore V
N: INT 5; lunghezza di V
S: INT; Accumulatore - Somma
```

```
LOAD R0 0; Indice i=0
LOAD R1 N; Numero elementi in V
LOAD R2 0; Acc=0
```

```
Testa: COMP R0 R1; i-N
BRGE Post; se  $i \geq N$ , fine ciclo
LOAD R3 V R0;  $V[i]$ 
ADD R2 R3;  $Acc = Acc + V[i]$ 
ADD R0 1;  $i = i + 1$ ;
BRANCH Testa;
```

```
Post: STORE R2 S;  $S = Acc$ 
STOP;
```

SOMMA VETTORIALE (2)

V_0, V_1, \dots, V_n
 $SUM = V_0$
 $I = 1$ (contatore)

$I \geq N ?$

vero

falso

$SUM = SUM + V_i$

$I = I + 1$

V : INT 3 10 2 -1 -4; Vettore V
 N : INT 5; lunghezza di V
 S : INT; Accumulatore - Somma

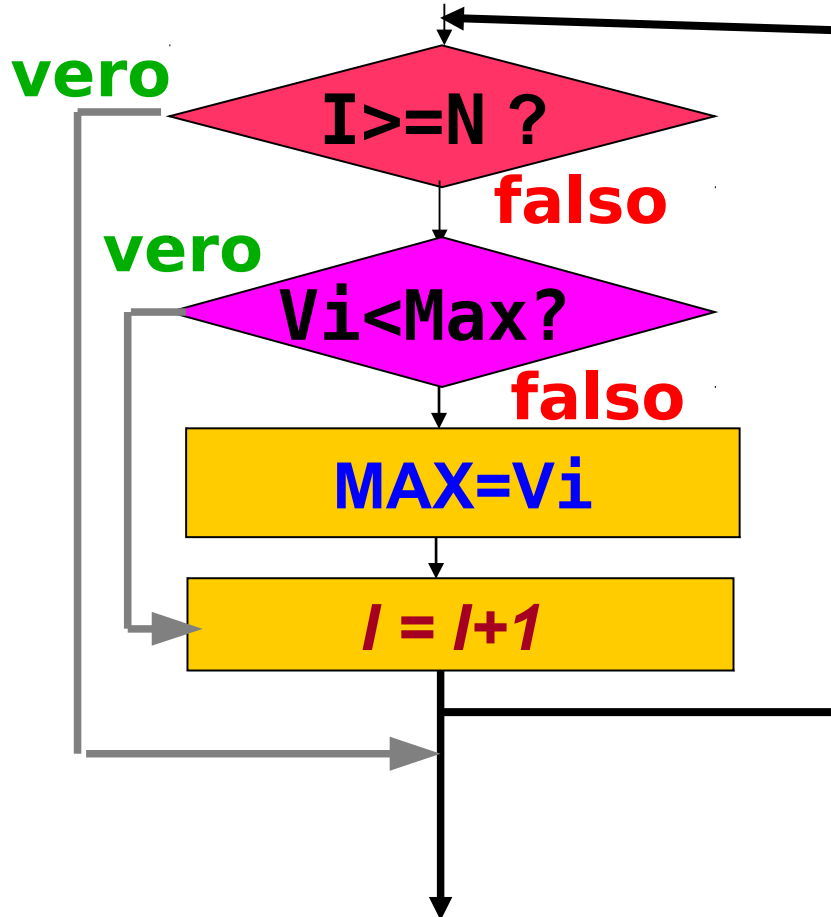
```
LOAD R0 1; Indice i=0  
LOAD R1 N;  
LOAD R2 V 0; Acc=V[0]
```

```
Testa: COMP R0 R1; i-N  
BRGE Post; se  $i \geq N$ , fine ciclo  
LOAD R3 V R0;  $V[i]$   
ADD R2 R3; Acc=Acc+ $V[i]$   
ADD R0 1;  $i=i+1$ ;  
BRANCH Testa;
```

```
Post: STORE R2 S;  
STOP;
```

VALORE MASSIMO

V_0, V_1, \dots, V_n
 $MAX = X_0$
 $I = 1$ (contatore)



V: INT 3 10 2 -1 -4; **Vettore V**
N: INT 5; lunghezza di V
M: INT; **Accumulatore** - Max

LOAD R0 1; Indice i=0

LOAD R1 N;

LOAD R2 V 0; Max=V[0]

Testa: COMP R0 R1; i-N

BRGE Post; se i>=N, fine ciclo

LOAD R3 V R0; R2=V[i]

COMP R3 R2; Xi<Max?

BRLT Next; Yes; next

LOAD R2 R3; No: Max=Xi

Next: ADD R0 1; i=i+1

BRANCH **Testa;**

Post: STORE R2 S;

STOP;