PROGRAMMAZIONE

Diagrammi di flusso

Elaborazioni condizionali
e cicliche

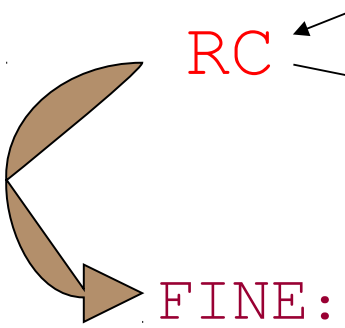
Programmazione: Primi passi

```
% Compito: X=X+Y
% - definire due variabili, X e Y, e darne valori iniziali
% - copiare X e Y in due registri, per fare aritmetica
% - aggiungere i registri
% - copiare il risultato dai registri alla variabile X
X: INT 10;     % X=10
Y: INT  2;     % Y=2

LOAD R0 X;     % R0=X
LOAD R1 Y;     % R1=Y
ADD R0 R1      % R0=R0+R1
STORE R0 X;    % X=R0
STOP;          % FINE
```

!! **Scrivere commenti** per poter facilmente spiegare e modificare il programma !!

Primi passi nel programmazione: Elaborazione condizionale

```
% Compito: Ordinare le due variabili X e Y in ordine crescente
% - definire le variabili X e Y e darne valori iniziali
% - confrontarne, caricandole prima nei registri
% - scambiarne i valori di X e Y se necessario
        X:      INT 10
        Y:      INT  2
        LOAD    R0, X
        LOAD    R1, Y
        COMP    R0, R1  % Confronta i valori
RC      BRLE    FINE    % Se ordinati, fine!
        STORE   R1, X   % Altrimenti, scambio.
        STORE   R0, Y
FINE:   STOP
```

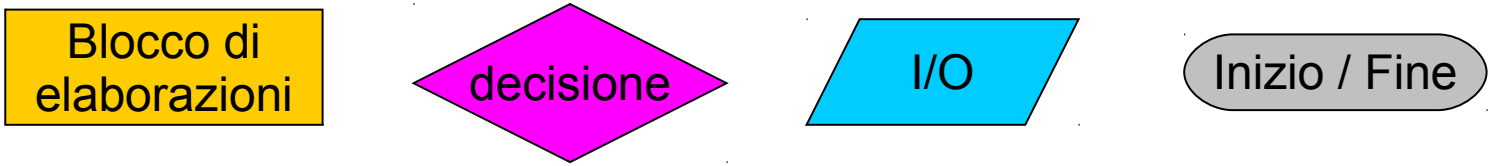


Capire la logica di un programma non lineare può essere difficile.
Quasi tutti i programmi sono cosi … La soluzione:

Diagrammi di Flusso

- Per capire la logica dei programmi con *cambio del flusso* è utile rappresentarla in modo visuale.

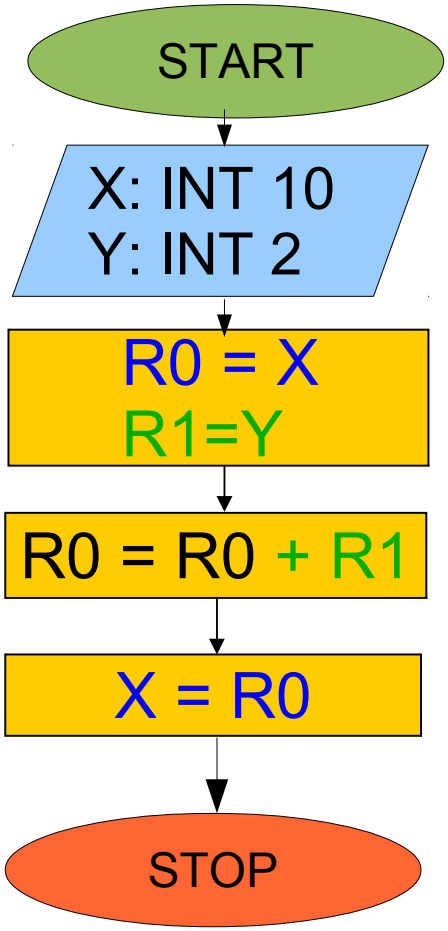
- La rappresentazione grafica più comune: **Diagrammi di Flusso**.
- Costruiti dai seguenti blocchi:



- Ogni passo elementare del algoritmo è rappresento con un blocco.
- Un blocco "**A**" potrebbe essere collegato tramite freccia con un altro blocco "**B**", indicando che "**B**" dovrebbe essere eseguito dopo "**A**".
- Un blocco decisionale è collegato con due o più blocchi.
- La sequenza dei passi che costruiscono l'algoritmo si codifica con la sequenza dei blocchi, collegati con frecce.

Esecuzione non-condizionale

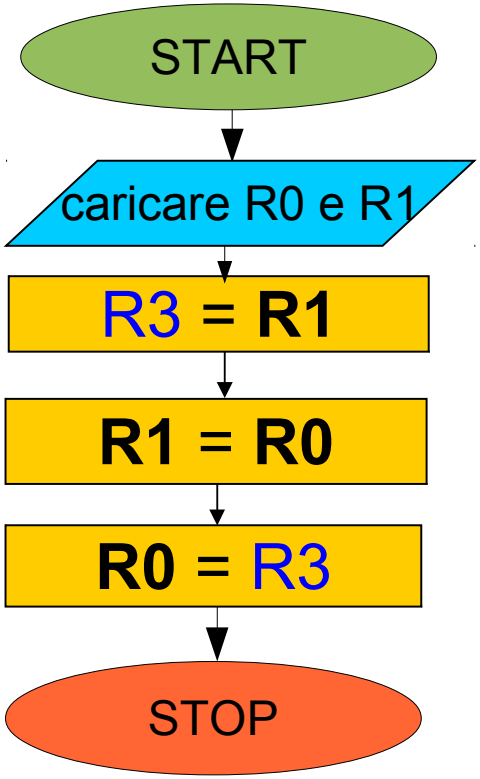
Esempio: Aggiungere Y al X

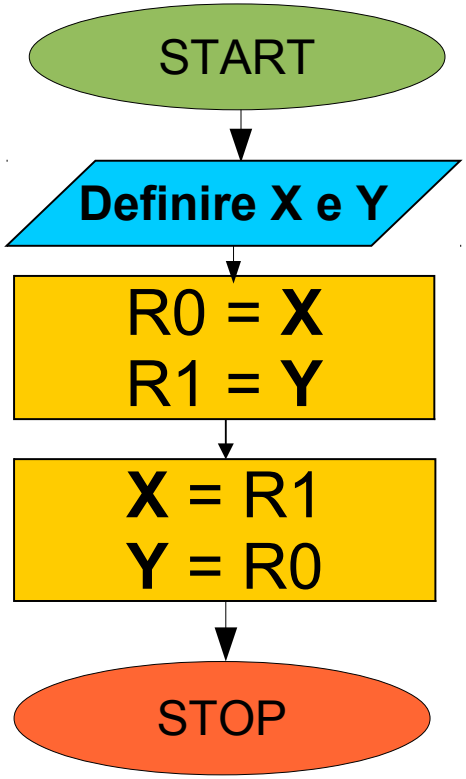



```
                    ; DATI
X:INT 10            ; X=10
Y:INT  2            ; Y=2
                    ; PROGRAMMA
LOAD   R0,X         ; X→R0
LOAD   R1,Y         ; Y→R1
ADD    R0,R1        ; R0=R0+R1
STORE  R0,X         ; R0→X
STOP                ; fermi !
```

Esecuzione non-condizionale

ESERCIZIO
Scambiare il contenuto di:
(a) due registri R0 e R1
(b) due variabili X e Y



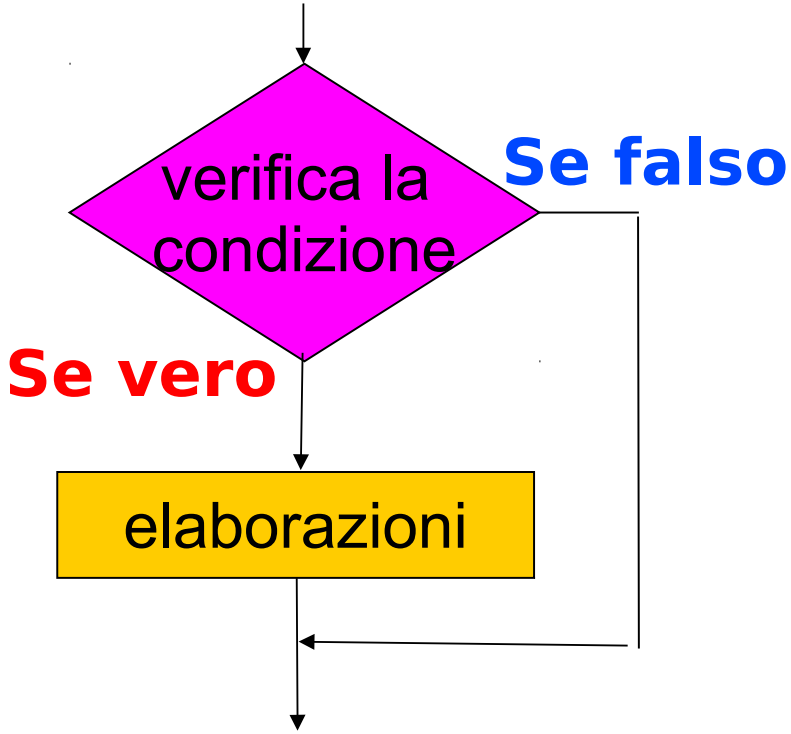


COMPITO

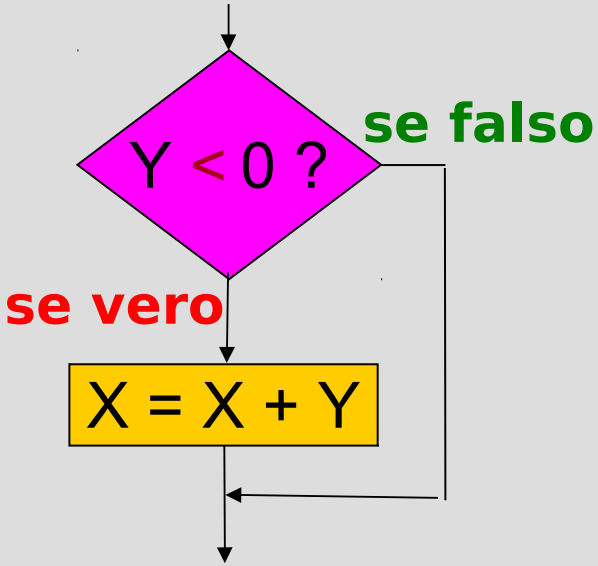
Scrivere due programmi
in Assembler
per il nostro CPU
che realizzano gli algoritmi

Esecuzione condizionale

Eseguire una serie di elaborazioni
solo se una condizione è vera



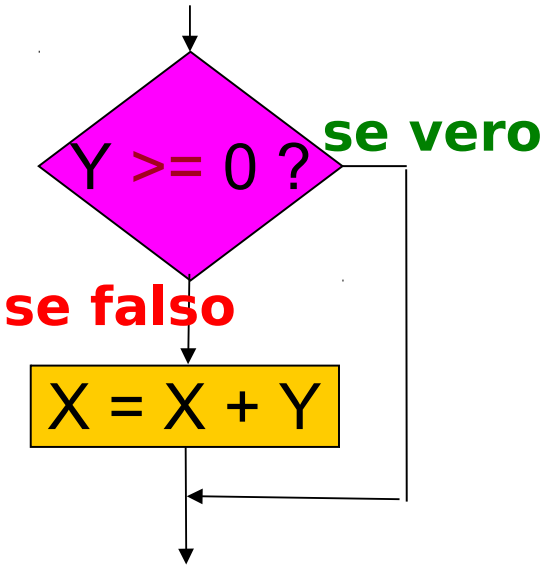
Esempio:
Aggiungere Y ad X
solo se Y<0



La logica “complimentare”: Saltare un blocco di elaborazioni

Per implementare l'esecuzione di un blocco di elaborazioni solo se “A” è vera, allora saltarlo se la condizione complimentare (not A) è vera.

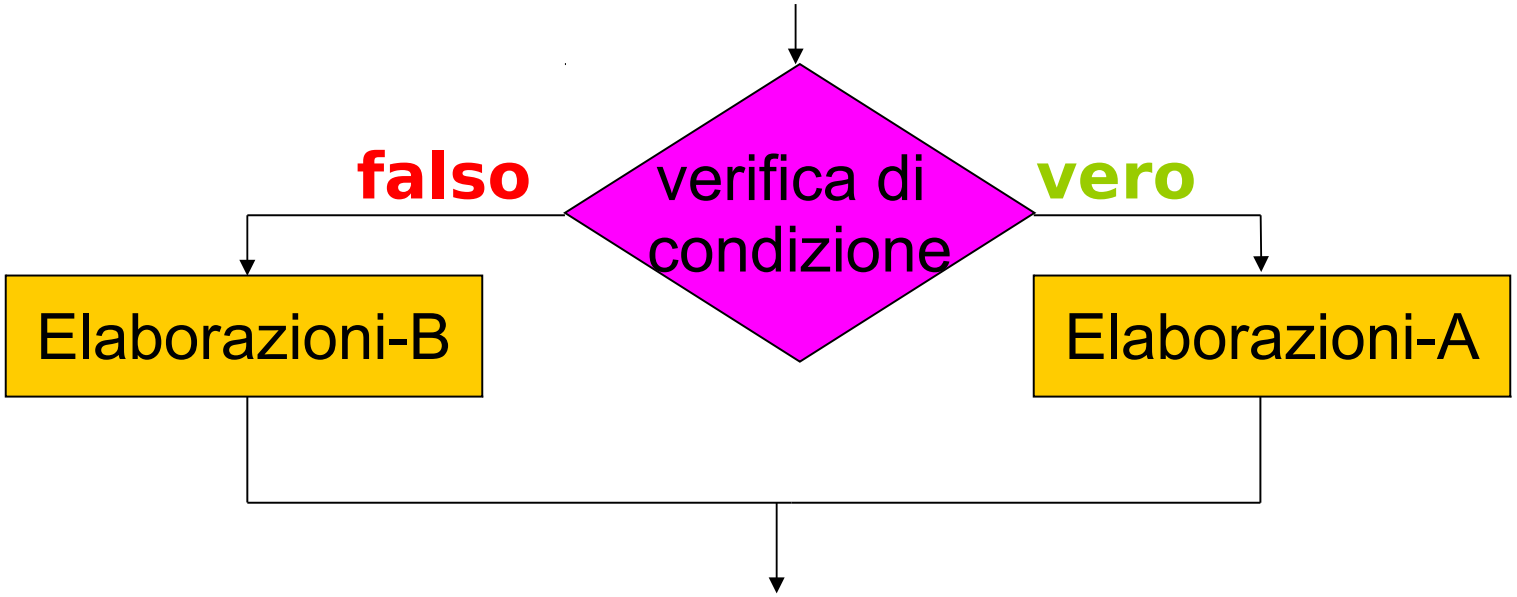
Esempio:
Aggiungere Y al X
solo se Y<0




```
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
STOP
FINE:
```

Esecuzione condizionale (2)

Eseguire blocco “**A**” o blocco “**B**”.



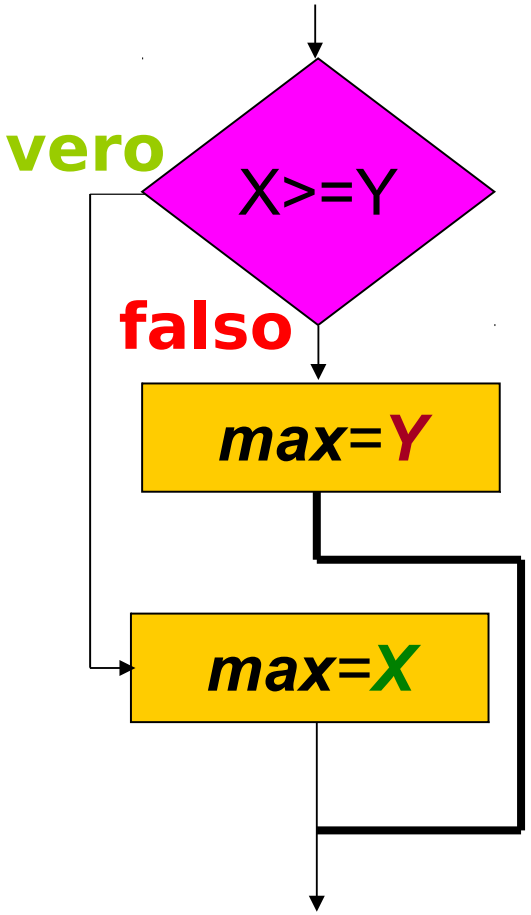
La memoria RAM è lineare

(non ha una natura spaziale, come, ad esempio, il cervello)

Come implementare questa schema ?

Esempio: scegliere uno dei due numeri

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



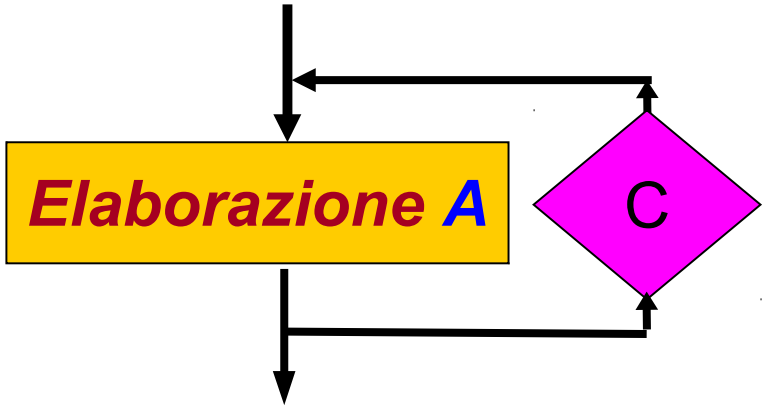
Non - eseguire questo codice utilizzando salto **non**-condizionale

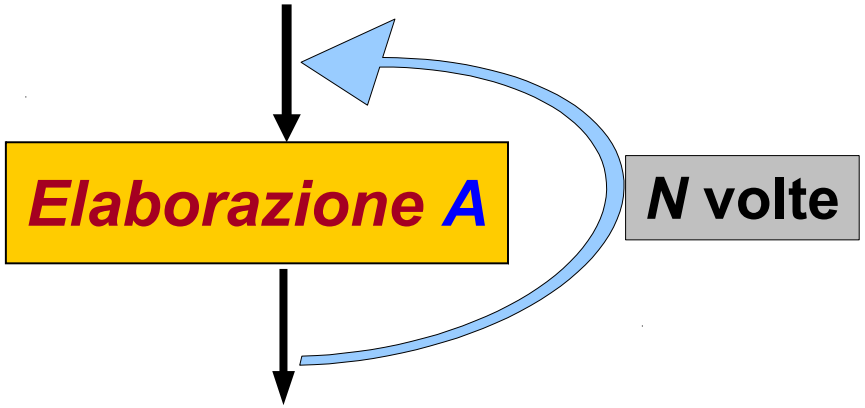
```
X: INT 5
Y: INT -3
Z: INT


LOAD R0 X
LOAD R1 Y
COMP R0 R1
BRGE   Big0    % Se R0>R1
STORE R1 Z     % R1>R0=>Z=R1
BRANCH FINE
STORE R0 Z     % R0>R1=>Z=R0
Big0:
STOP
FINE:
```

Elaborazioni iterativi

Spesso serve ad eseguire un blocco di elaborazioni “A” più volte.

Ci sono due principali tipi di elaborazioni iterative (CICLI):



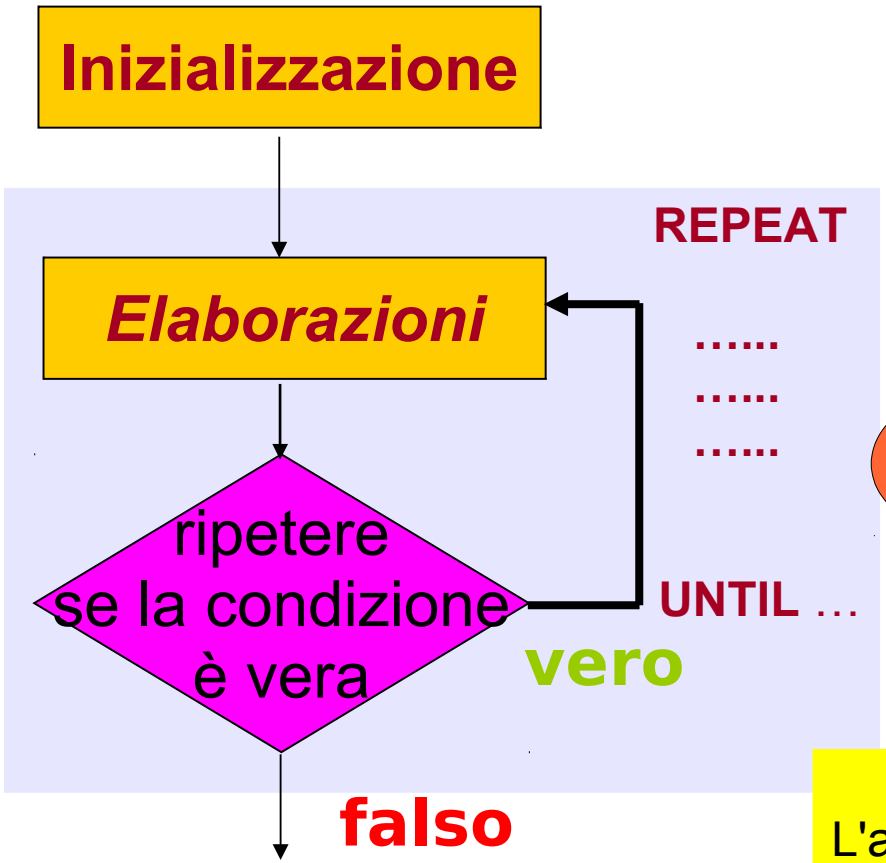


(i) gestite da una condizione “C”, che dipende da un evento esterno o dagli elaborazioni “A” stessi

(ii) eseguire il blocco di elaborazioni “A” esattamente **N** volte (gestite da un contatore).

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

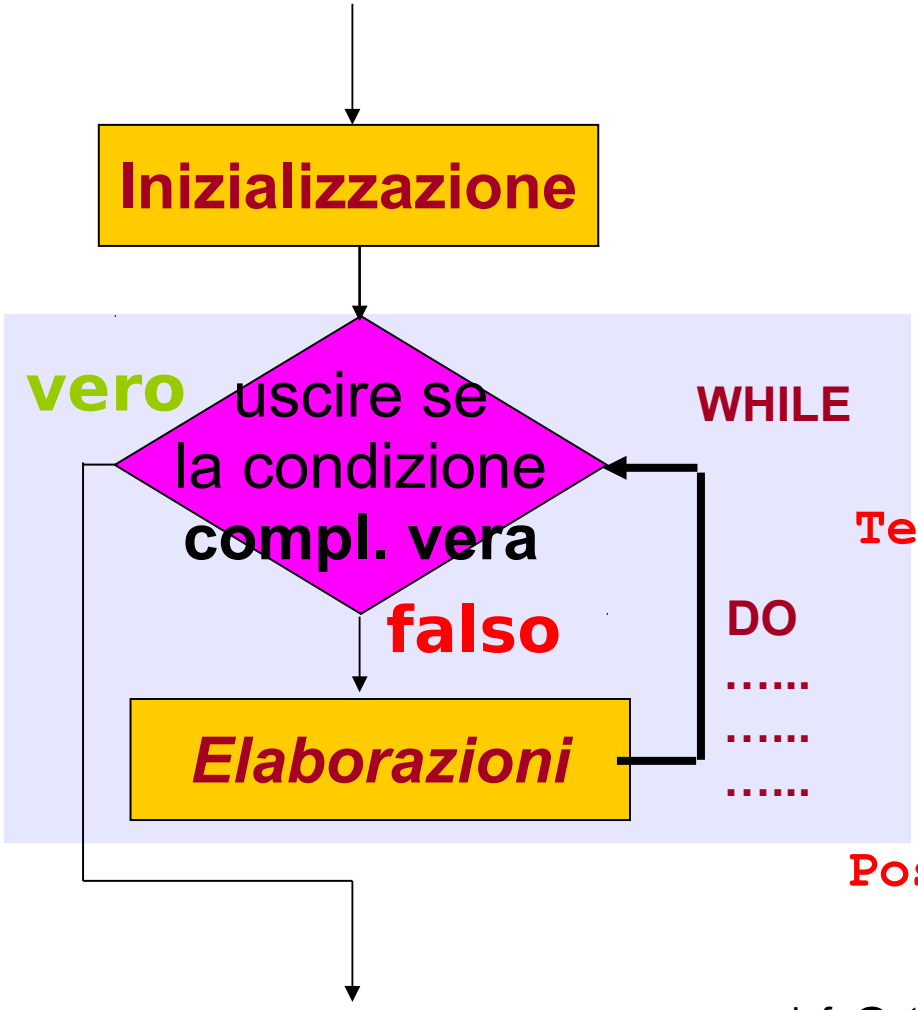
Esempio: Sottrarre un numero ***k*** da un altro ***N*** **finché** ***N***>=0




```
N: INT 15
K: INT 4

LOAD R0 N
LOAD R1 K
Pos:SUB  R0 R1   % N=N-K
COMP R0 0    % Se R0>=0 ..
BRGE  Pos    % .. ripetere
STORE R0 N
STOP
```

Se N < 0 ?
L'algoritmo 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***>=k

```
N: INT 15
K: INT 6

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

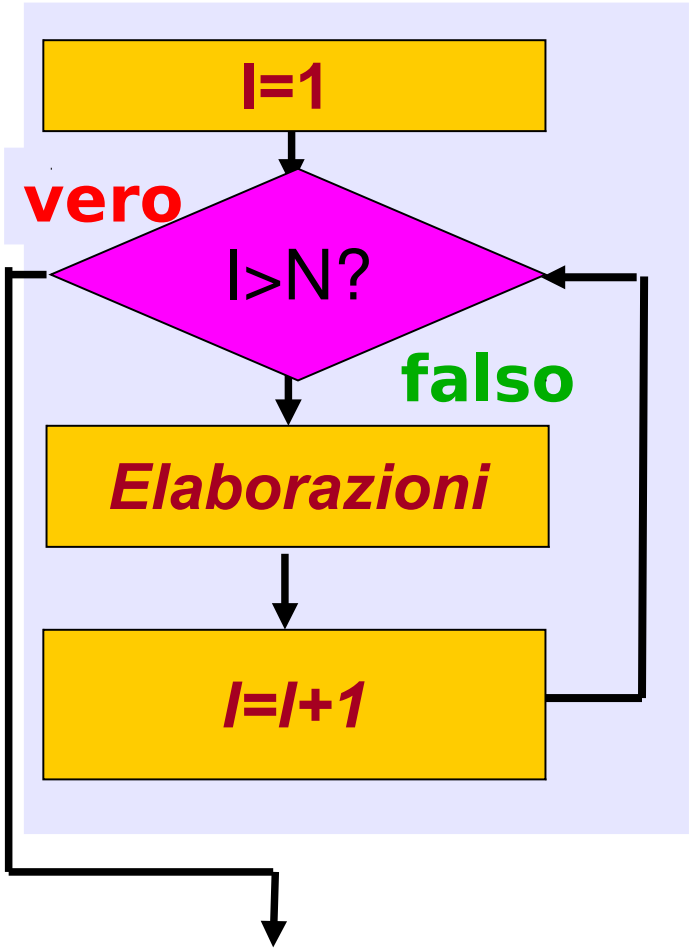
- viene utilizzato per controllare che una certa elaborazione "**A**" viene eseguita esattamente per un certo numero di volte **N.**

Funzionamento:

- contatore progressivo
  - prima di entrare nel ciclo, inizializzare il contatore con 1
  - entrare nel ciclo 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
  - entrare nel ciclo se il contatore ha **valore maggiore di 0**
  - ad ogni ripetizione, decrementare il contatore con 1

CICLI (3a) Contatore progressivo

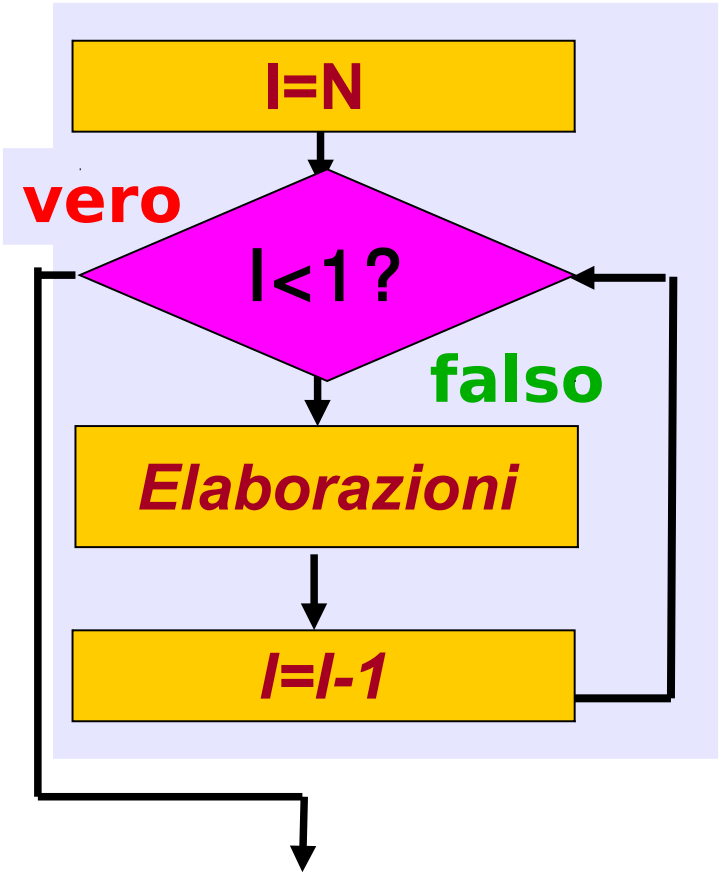
FOR I = 1 TO N




```
N: INT 4 % Numero elabor.


      LOAD R1 N
      LOAD R0 1     % contatore
Test: 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
   COMP R0 1    % Se R0<0 ..
testa:BRLT Usc  % .. Uscire
   %% ELABORAZIONI %%
   SUB  R0 1    % contat.-1
   BRANCH Test  % → testa
Usc: STOP
```