**DOCUMENTAZIONE PER L'UTENTE**

# *Manuale programmazione ISO*

SIDAC S.r.L
✉ **21017 Samarate (Varese) Via Acquedotto, 111**
**E_Mail info@sidaccnc.it**
☎ **Tel. 0331.22.30.19 r.a.**
☎ **Fax. 0331.22.30.78**

**Ver. 1.0 - Aggiornamento 15.03.00**
**Edizione 15.03.00**

# SOMMARIO

- Il manuale, in perfette condizioni, viene fornito al cliente assieme alla macchina.
- Conservare il manuale in prossimità della macchina ad immediata disposizione dell'utente, ed archiviarne le eventuali copie in un luogo idoneo al suo mantenimento in ottime condizioni.
- In caso di smarrimento o deterioramento, richiedere ulteriori copie del manuale direttamente a Sidac.
- Il presente manuale rispecchia lo stato della macchina all'atto della vendita. Sidac si riserva il diritto di aggiornarlo senza l'obbligo di adeguare le versioni precedenti.
- Si ricorda che ai sensi della normativa vigente, il manuale di istruzioni costituisce parte integrante della macchina; esso deve pertanto accompagnarla in ogni suo spostamento.

# 1.0 Fondamenti di programmazione

Il linguaggio di programmazione ISO è stato introdotto al fine di programmare i cicli di lavoro automatici, e/o semiautomatici, di macchine a controllo numerico.
L'attuale versione sviluppata permette di coprire applicazioni che coinvolgono un numero di assi controllati variabile da un minimo di 1 ad un massimo di 9.

Il programma ISO è costituito da tre unità fondamentali:
- Il *programma base*, o *corpo principale*;
- I *sottoprogrammi*;
- I *programmi di semiautomatico*.

L'unica di queste unità sicuramente necessaria è il *corpo principale* che rappresenta anche il punto di inizio esecuzione, le altre unità, descritte in seguito, possono essere presenti o meno a seconda delle necessità.
Per quanto riguarda la costruzione del programma, quale ad esempio l'inserimento del *programma principale* e l'inserimento o la rimozione di *sottoprogrammi* o *programmi di semiautomatico*, si rimanda al manuale relativo alla descrizione del funzionamento del CNC.

## 1.1 Corpo principale del programma

Il corpo principale del programma è costituito da un file al quale è possibile assegnare il nome desiderato ma che deve presentare obbligatoriamente l'estensione .MPF, ad esempio:

**Testa.mpf**

Il contenuto del file non deve inoltre presentare alcuna intestazione particolare ma può cominciare immediatamente con le istruzioni messe a disposizione dal linguaggio di programmazione.

## 1.2 Sottoprogrammi

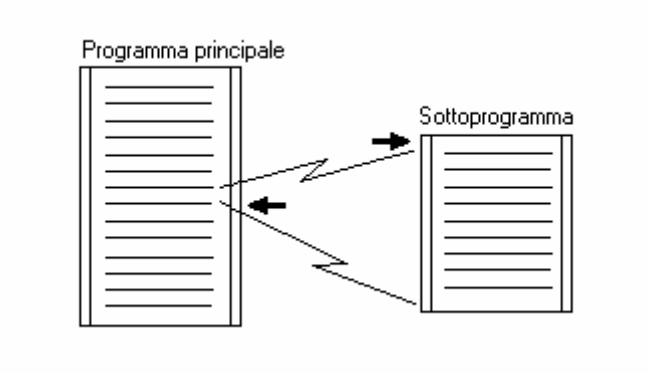
I sottoprogrammi sono dei file contenenti blocchi di istruzioni esattamente come il programma principale. Il loro utilizzo è semplicemente legato alla necessita di mantenere concettualmente separate parti di programma che si riferiscono ad una particolare lavorazione o che devono essere eseguite più volte durante il ciclo macchina, in questo modo è possibile programmare solo una volta una sequenza di lavorazione che invece deve essere ripetuta più volte.
A differenza del programma di principale i file relativi ai sottoprogrammi devono presentare l'estensione .SPF, ad esempio:

**Sottoprogramma1.spf**
**Sottoprogramma2.spf**
**…**

Ogni sottoprogramma deve presentare come prima riga una intestazione avente il seguente formato:

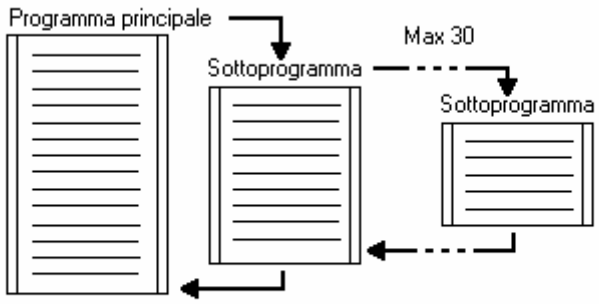
**PROC "Nome sottoprogramma" SAVE**

Dove "Nome sottoprogramma" indica il nome con cui verrà chiamato il sottoprogramma dal programma principale o da altri sottoprogrammi.
Esempio di esecuzione:




### 1.2.1 Concetto di annidamento

Un sottoprogramma può contenere a sua volte chiamate ad un o più sottoprogrammi che a loro volta chiamano altri sottoprogrammi ottenendo una sorta di annidamento. Il valore massimo di annidamento, o *profondità di annidamento*, è pari a 30, ciò significa che a partire dal programma principale possono partire fino a 30 richiami di sottoprogrammi annidati dopo di che è necessario ritornare da un sottoprogramma per poterne richiamare un altro.
Esempio di annidamento:




## 1.3 Programmi di semiautomatico

Con questo termine sono indicati dei programmi che possono essere richiamati dal CNC attraverso la pagina di semiautomatico.
I programmi di semiautomatico hanno la medesima struttura del programma principale e i relativi file devo avere estensione .MPF

Un esempio è il seguente.

**Semiautomatico1.mpf**
**Semiautomatico2.mpf**
**…**

Ognuno di questi programmi deve però presentare come prima riga una intestazione avente il seguente formato:

**SEMI “Nome programma semiautomatico”**

Dove “Nome programma semiautomatico” indica il nome con cui viene identificato il programma nel CNC.
Per informazioni sulla selezione e l’esecuzione dei semiautomatici riferirsi al manuale utente della macchina.

# 2.0 Il Linguaggio di programmazione

Prima di analizzare le istruzioni a disposizione del programmatore, per ottenere la reale movimentazione degli organi della macchina, verrà definito il concetto di *variabile* e verranno analizzati i *costrutti base* che il linguaggio ISO mette a disposizione per la realizzazione del programma.

## 2.1 Le variabili o parametri R

Il linguaggio mette a disposizione del programmatore 2880 variabili, identificate tramite la lettera *R*, che possono contenere i dati utilizzati durante l'esecuzione del programma.

Il valore contenuto in queste variabili può essere impostato direttamente a livello di CNC tramite l'utilizzo di opportune pagine di editor dati, ma può anche essere il risultato di una manipolazione delle stesse durante la stesura del programma.
A livello di programmazione infatti, non solo è possibile leggere il contenuto di queste variabili ma è anche possibile assegnargli un valore numerico o utilizzarle per sviluppare complessi calcoli matematici. Più avanti verranno descritti, a tale proposito, anche gli operatori matematici (e logici) che sono messi a disposizione dal linguaggio.

*NOTA*
I parametri R sono dei tipi DOUBLE (precisione doppia, virgola mobile) e quindi permettono di supportare valori compresi tra: -179769E308 e –4,94065E-324 per valori negativi e tra 4,9406E-324 e 1,79769E308 per numeri positivi.

### 2.1.1 Metodo di indirizzamento

E' possibile accedere al contenuto di un parametro o assegnargli un valore tramite due metodi di indirizzamento:

- Indirizzamento diretto;
- Indirizzamento indiretto.

**Indirizzamento diretto**
Si ottiene facendo seguire alla lettera R direttamente il numero di variabile al quale si vuole accedere in lettura o in scrittura.
Esempio di formato:

**R27**
Permette di fare riferimento alla variabile numero 27

**Indirizzamento indiretto**
Si ottiene utilizzando un altro parametro R e/o una costante come spiazzamento di indirizzo rispetto al numero di parametro R a cui si fa riferimento:

Esempio di formato:

**R[170 + R45]**
In questo caso si fa riferimento alla variabile numero 170 più il contenuto della variabile R45.
Se ad esempio la variabile R45 contiene il valore 32 il risultato è quello di accedere alla variabile numero 170+32 cioè 202.

Non è possibile utilizzare il parametro R0 che è utilizzato direttamente dal sistema.

## 2.2 Costrutti di base del linguaggio

Il linguaggio mette a disposizione del programmatore una serie di costrutti fondamentali che permettono di regolare il flusso logico di esecuzione del programma.

Le strutture, analizzate a seguire, sono:
- Esecuzione condizionata *if-endif*;
- Salti incondizionati *goto*;
- Salto condizionato *case-of*;
- Chiamate a sottoprogrammi.

### 2.2.1 Esecuzione condizionata if-endif

L'utilizzo del costrutto IF-ENDIF permette di selezionare, sulla base di qualche condizione logica, quale parte del programma deve essere eseguita oppure se una parte di programma deve o non deve essere eseguita.

La struttura più semplice presenta la seguente sintassi:

**…**
**IF "condizione logica"**
**…**
**…**
**ENDIF**
**…**

in questo caso la parte di programma racchiusa tra le due parole chiave IF e ENDIF viene eseguita solamente se è verificata la condizione che segue l'istruzione IF (per quanto riguarda la sintassi delle condizioni logiche fare riferimento al paragrafo che descrive gli operatori matematici e logici).

La struttura più complessa presenta invece la seguente sintassi:

**…**
**IF "condizione logica"**
**…**
**…**
**ELSE**
**…**
**…**
**ENDIF**
**…**

in questo caso, se è verificata la condizione logica che segue l'istruzione IF, viene eseguita la parte

di codice racchiusa tra le parole chiave IF e ELSE, altrimenti viene eseguita la parte di programma contenuta tra le parole chiave IF e ENDIF.

Il costrutto IF-ENDIF può anche essere annidato, con una profondità di annidamento praticamente infinita, ottenendo una struttura analoga alla seguente:

```
IF “condizione 1”
        …
        IF “condizione 2”
                …
        ELSE
                …
        ENDIF
        …
ELSE
        …
        IF “condizione 3”
                …
        ELSE
                …
        ENDIF
        …
ENDIF
```

*NOTA*
Qualora, a seguito della verifica di una condizione logica, l’istruzione da eseguire sia una sola, è possibile utilizzare la parola chiave THEN scrivendo tutto su una unica linea di comando:

**IF “condizione logica” THEN “istruzione”**

### 2.2.2 Salto incondizionato goto

Durante l’esecuzione del programma è possibile richiedere una salto incondizionato ad un’altra riga del programma utilizzando la parola chiave *GOTO.*
La riga del programma a cui si desidera saltare deve essere individuata tramite l’utilizzo di una etichetta a cui si fa riferimento nell’istruzione goto.
L’etichetta può essere una qualsiasi sequenza alfanumerica terminate con il simbolo “:” (due punti).
Il salto incondizionato può inoltre avvenire solamente all’interno del programma principale o del sottoprogramma nel quale è utilizzata l’istruzione; non è quindi possibile eseguire un salto da una riga di un sottoprogramma ad una riga appartenente ad un altro sottoprogramma (in altri termini salti possono avvenire solo all’interno dello stesso file).
Se la riga a cui si fa riferimento nell’istruzione goto è precedente, nel programma, all’istruzione stessa bisognerà utilizzare la forma *GOTOB*, se viceversa è successiva bisognerà utilizzare la forma *GOTOF*.

La sintassi è quindi la seguente:

**…**
**“etichetta:”**
**…**
**…**
**…**
**GOTOB etichetta**
**…**

oppure

**…**
**GOTOF etichetta**
**…**
**…**
**…**
**“etichetta:”**

**Esempio di programmazione:**
…
*Ripeti:*
…
…
*GOTOB Ripeti*
…

### 2.2.3 Salto condizionato case-of

Analogamente a quanto descritto per l’istruzione di salto incondizionato, è possibile definire dei salti a qualsiasi riga del programma, che però risultano essere vincolati dal soddisfacimento di opportune condizione logiche, tramite l’utilizzo del costrutto CASE-OF.

La sintassi utilizzata è la seguente:

**CASE “condizione logica” OF “val 1” GOTOF “etichetta 1”…“val n” GOTOF “etichetta n”**
**…**
**…**
**“etichetta 1:”**
**…**
**…**
**“etichetta 2”:**
**…**
**…**
**“etichetta n”:**
**…**

Il significato del costrutto sopra descritto è il seguente:

se è la condizione logica contenuta tra le parole chiave CASE e OF assume il valore “val 1” viene eseguito un salto (GOTOF) alla riga etichettata “etichetta 1”, analogamente se la condizione logica assume il valore “val n” viene eseguito un salto alla riga etichettata “etichetta n”.

Qualora la condizione logica non assuma alcun valore contenuto nell’istruzione il programma proseguirà dall’istruzione successiva a quella di CASE-OF.
Se una volta eseguito il copro di programma che segue l’etichetta a cui si è saltati, non si vuole eseguire le istruzioni che appartengono al corpo di programma che segue l’etichetta successiva, è necessario eseguire un salto incondizionato al punto desiderato.
Analogamente a quanto detto per i salti incondizionati anche questo costrutto permette di eseguire salti a righe contenute nello stesso programma principale, o sottoprogramma, a cui l’istruzione appartiene.

**Esempio di programmazione:**

…
…
CASE “condizione” OF valore_1 GOTOF etichetta_1 ….. valore_n GOTOF etichetta_n
…

etichetta_1:
…
…
GOTOF Fine

etichetta_2:
…
…
GOTOF Fine

…
etichetta_n:
…
…
GOTOF Fine:

Fine:
…
…

| |
|---|
| *NOTA*<br>I salti possono essere eseguiti solo in avanti GOTOF. |

### 2.2.4 Chiamate a sottoprogrammi

La chiamata ad un sottoprogramma è eseguita semplicemente utilizzando il nome con cui è stato intestato il sottoprogramma stesso (vedi paragrafo relativo alla definizione dei sottoprogrammi 1.2).

Supponendo che esista il sottoprogramma avente la seguente intestazione:

**PROC “sottoprogramma 1” SAVE**

La sintassi utilizzata per la chiamata sarà la seguente:

**…**
**“sottoprogramma 1”**
**…**

La parte di codice successiva a “sottoproramma 1” verrà eseguita terminata l’esecuzione del codice contenuto nel sottoprogramma chiamato.

**Esempio di programmazione:**

Se esiste il sottoprogramma:

PROC Alza_carrello SAVE

La chiamata è eseguita nel seguente modo:

…
…
Alza_carrello
…
…

| |
|---|
| *NOTA*<br>Il ritorno da un sottoprogramma chiamato a quello chiamante è eseguito utilizzando l’istruzione particolare M17 descritta nella sezione riguardante le funzioni M. |

## Operatori matematici, di confronto e logici

Appartengono a questa categoria tutti i comandi che permettono di eseguire operazioni di calcolo (matematici), di confronto e logiche sulle variabili R.

### 2.3 Operatori matematici

Gli operatori matematici messi a disposizione dal linguaggio di programmazione sono i seguenti:

- Somma algebrica: +,
- Sottrazione algebrica: -,
- Moltiplicazione: *,
- Divisione: /,
- Assegnamento: =.

*NOTA*
Se utilizzato in una operazione matematica il simbolo = (uguale) ha il significato di assegnamento; permette cioè di assegnare alla variabile che si trova alla sua sinistra il valore risultato del calcolo matematico che si sviluppa alla sua destra.

Gli operatori matematici non hanno nessun ordine di priorità quindi il calcolo matematico si svolge ordinatamente da sinistra verso destra.

**Esempio di calcolo matematico:**

R27 = R34 + R56 / R[45 + R1]

In questo caso viene assegnato alla variabile R27 il valore risultato della seguente sequenza di calcolo: somma algebrica del valore contenuto in R34 con quello contenuto in R56 il cui risultato è successivamente diviso per il valore contenuto nella variabile R45 spiazzata del valore contenuto in R1 (indirizzamento indiretto). Il risultato è stato quindi ottenuto eseguendo le operazioni da sinistra verso destra.

Se, viceversa, è necessario ottenere come risultato la somma del valore contenuto in R34 con il valore risultato della divisione di R56 con R[45 + R1], cioè eseguire prima la divisione e successivamente la somma algebrica, bisogna ricorrere all'utilizzo delle parentesi tonde ( ) nel seguente modo:

R27 = R34 + (R56 / R[45 + R1])

In altre parole i calcoli contenuti all'interno delle parentesi tonde vengono eseguiti prima delle operazioni esterne. L'utilizzo delle parentesi può essere annidato con profondità di annidamento potenzialmente infinita.

## 2.4 Operatori di confronto

Gli operatori di confronto messi a disposizione dal linguaggio di programmazione sono i seguenti:
- maggiore: >,
- minore: <,
- uguale: ==,
- maggiore o uguale: >=,
- minore o uguale: <=,
- diverso: <>.

Vengono utilizzati per eseguire un confronto matematico tra il valore risultato di un'espressione matematica che si trova alla destra dell'operatore con il valore risultato di un'espressione che si trova alla sinistra dello stesso.

**Esempio di operazione di confronto:**

(R25 + R18) < (R7 * R[45 + R3] / R32)

in questo caso verifico se il risultato della somma tra il contenuto di R25 sommato al contenuto di R18 è minore o meno del risultato del contenuto di R7 moltiplicato per il contenuto dir R[45 + R3] e diviso per il contenuto di R32.

Una operazione di confronto ritorna come risultato unicamente due valore:
- VERO qualora il confronto sia soddisfatto,
- FALSO qualora il confronto non sia soddisfatto.

L'utilizzo degli operatori di confronto, in eventuale combinazione con quelli logici descritti nel paragrafo successivo, permette di sviluppare più o meno complesse "condizioni logiche" utilizzate in alcuni costrutti descritti nei paragrafi precedenti, quali quelli di esecuzione condizionata (if-endif) o di salto condizionato (case-of).

## 2.5 Operatori logici

Gli operatori logici messi a disposizione dal linguaggio sono i seguenti:
- prodotto logico: AND,
- somma logica: OR,
- negazione logica: NOT.

Sono utilizzati per eseguire operazioni logiche sui risultati delle operazioni di confronto, quindi ammettono come operandi solo strutture che ritornano come risultato unicamente i valori VERO o FALSO.
In altre parole permettono di eseguire delle operazioni su espressioni che utilizzano gli operatori di confronto descritti nel paragrafo precedente.

**Esempio di calcolo logico:**

(R27 > (R56 * 2)) AND ((R23 / (R16 – R45)) == (R32*R46))

supponendo che l'espressione a sinistra dell'operatore AND sia verificata (valore VERO) e quella alla sua destra non sia verificata (valore FALSO) il calcolo si riduce a *Vero AND Falso* Il cui risultato è FALSO.

Come per i calcoli matematici descritti nel paragrafo 3.1, il calcolo logico procede ordinatamente da sinistra verso destra quindi anche gli operatori and, or e not non hanno nessun ordine di priorità. Anche in questo caso se si desidera eseguire un calcolo prima di un altro è necessario ricorrere all'utilizzo delle parentesi tonde ( ). In altre parole i calcoli logici racchiusi tra parentesi vengono eseguiti prima di quelli esterni.

*NOTA*
Per una migliore comprensione dei risultati dei calcoli logici riportiamo le tavole della verità degli operatori descritti.

**Tavola verità prodotto logico AND**

| ***AND*** | **Vero** | **Falso** |
|---|---|---|
| **Vero** | *Vero* | *Falso* |
| **Falso** | *Falso* | *Falso* |

**Tavola verità somma logica OR**

| ***OR*** | **Vero** | **Falso** |
|---|---|---|
| **Vero** | *Vero* | *Vero* |
| **Falso** | *Vero* | *Falso* |

**Tavola verità negazione logica NOT**

| ***NOT*** | |
|---|---|
| **Vero** | *Falso* |
| **Falso** | *Vero* |

## 3.0 Istruzioni operative del linguaggio

Nella sezione successiva viene descritto l'insieme delle istruzioni, messe a disposizione dal linguaggio, che permettono al programmatore di definire i percorsi degli assi della macchina, e i relativi comandi di percorso o funzioni preparatorie G.
Successivamente viene descritto l'utilizzo dei comandi M che permettono di eseguire e controllare le funzioni di PLC sviluppate per ogni particolare applicazione del CNC.

Le istruzioni operative sono le seguenti:

- funzioni preparatorie: G,
- funzioni di PLC: M,
- programmazione dei percorsi,
- definizione delle velocità di percorso,
- istruzioni generiche.

### 3.1 Funzioni preparatorie G

Le funzioni G permettono di definire le caratteristiche del percorso assi che si intende programmare e, salvo specifiche indicazioni, sono da considerarsi MODALI.

*NOTA*
Con il termine *modale* si intende che l'impostazione di percorso, eseguita tramite l'utilizzo di una funzione G, rimane attiva fino all'utilizzo di una nuova G che ne cambia lo stato.

#### 3.1.1 Comando di rapido G00

Il comando G00 imposta la velocità di rapido e disattiva automaticamente la modalità G01.
Tutti i movimenti, programmati dopo questa istruzione, vengono eseguiti alla massima velocità consentita per ogni singolo asse.
In questa condizione non sono chiaramente considerate le eventuali velocità impostate con le apposite istruzioni e non viene eseguito nessun movimento interpolato; il movimento degli assi, comandato nella medesima istruzione, è completamente indipendente (gli assi partiranno assieme ma arriveranno in posizione unicamente in funzione della quota richiesta e della loro massima velocità).

*NOTA*
Si definisce INTERPOLATO un movimento in cui gli assi partono assieme e arrivano sulla posizione programmata nel medesimo istante. Il massimo numero di assi che possono essere mossi in interpolazione è pari a 3.

### 3.1.2 Comando di interpolazione G01

Il comando G01 imposta la velocità di lavoro programmata e disattiva automaticamente la modalità G00.
Tutti i movimenti programmati dopo questa istruzione vengono eseguiti alla velocità di lavoro impostata (riferirsi al paragrafo relativo alla definizione delle velocità).
Qualora il numero di assi programmati nella medesima istruzione non è superiore a 3, e non sono utilizzati comandi di impostazione velocità dedicati ad ogni singolo asse, ma unicamente il comando di impostazione velocità di lavoro, il movimento degli assi risulta essere *interpolato* (gli assi partono assieme e arrivano in posizione nello stesso istante).
Qualora nella medesima istruzione viene programmato il movimento di un numero di assi maggiore a 3, o viene definita una velocità dedicata per qualche asse, il movimento degli assi viene viceversa eseguito alle velocità di lavoro programmate per ogni singolo asse e in modo completamente indipendente (gli assi partono assieme ma arriveranno in posizione in funzione unicamente della quota richiesta e della velocità programmata).

### 3.1.3 Interpolazione circolare oraria G02

Il comando G02 imposta la modalità di interpolazione circolare *oraria* tra coppie di assi e disattiva automaticamente le funzioni G00 e G01.
La coppia di assi programmata nell'istruzione successiva esegue un movimento di rotazione oraria attorno ad un centro impostato, e fino al raggiungimento delle quote impostate, alla velocità di lavoro definita nell'istruzione.

*NOTA*
In una *interpolazione circolare* una coppia di assi esegue una rotazione (oraria o antioraria) nel piano 2D descrivendo un arco di circonferenza attorno ad un punto impostato come centro di rotazione.

### 3.1.4 Interpolazione circolare antioraria G03

Analogo al comando G02 descritto nel paragrafo precedente ma con impostazione della rotazione in senso *antiorario*.

### 3.1.5 Programmazione assoluta G90

Il comando G90 imposta la modalità di programmazione assoluta e disattiva automaticamente la modalità G91.
Tutti i movimenti degli assi programmati dopo questa istruzione sono da intendersi riferiti a quote assolute rispetto al punto di azzeramento della macchina. In altri termini le quote programmate nel comando di movimento sono quelle effettivamente raggiunte dagli assi rispetto al punto di azzeramento iniziale.

### 3.1.6 Programmazione incrementale G91

Il comando G91 imposta la modalità di programmazione relativa e disattiva automaticamente la modalità G90.
Tutti i movimenti degli assi programmati dopo questa istruzione sono da intendersi relativi rispetto alla loro posizione attuale. In altri termini il valore di quota programmato è l'incremento algebrico rispetto alla quota attuale dell'asse.

### 3.1.7 Arresto preciso G60

Il comando G60 imposta la modalità di movimentazione con arresto preciso e disattiva automaticamente la modalità G64.
Tutti i movimenti degli assi programmati dopo questa istruzione eseguiranno movimenti con arresto preciso sulla quota impostata.

*NOTA*
Nella modalità di *arresto preciso* ogni movimento programmato prevede la partenza dell'asse con una rampa di accelerazione e un arresto dell'asse sulla quota impostata con rampa di decelerazione.

### 3.1.8 Funzionamento continuo G64

Il comando G64 imposta la modalità di movimentazione in continuo degli assi e disattiva automaticamente la modalità G60.
Tutti i movimenti programmati dopo questa istruzione vengono eseguiti senza arresto degli assi nei punti di discontinuità, ovvero tra la programmazione di un movimento e quella del movimento successivo. I punti di discontinuità sono raccordati in modo da ottenere un profilo di movimento, che si avvicina alla traiettoria reale programmata, in funzione della parametrizzazione degli assi.

# 4.0 Funzioni G particolari

Sono funzionalità operative anziché preparatorie come quelle descritte precedentemente.

## 4.1 Tempo di attesa G04

La funzione G04 permette di impostare un tempo di ritardo.
La sintassi di questa istruzione è la seguente:

**G04 F“tempo di attesa”**

dove “tempo di attesa” è una costante, o un parametro R, che esprime in *secondi* il tempo di attesa che si desidera programmare.

**Esempio di programmazione**

G04 F1.25

viene eseguito un tempo di attesa della durata di 1.25 secondi.

## 5.0 Programmazione del percorso

In questa sezione viene descritto come programmare il movimento di uno o più assi del sistema utilizzando le istruzioni di percorso.

Ogni asse del sistema può essere programmato utilizzando una delle seguenti modalità:
- programmazione movimento in modalità BLOCCANTE,
- programmazione movimento in modalità NON BLOCCANTE.

Nella modalità BLOCCANTE non viene eseguita l'istruzione successiva del programma ISO fino a quando l'asse, o gli assi, programmati nell'istruzione corrente non hanno terminato il loro movimento.
Nella modalità NON BLOCCANTE, viceversa, viene continuata l'esecuzione del programma subito dopo aver attivato il movimento dell'asse, o degli assi, programmati con questa modalità; in altri termini non è atteso il termine del movimento degli assi programmati in questa modalità per procedere nell'esecuzione del programma ISO.

*NOTA*
In una medesima istruzione (linea di programma) è possibile programmare il movimento di uno o più assi. Se non è richiesto un movimento interpolato, G01, è possibile programmare il percorso di alcuni assi in modalità bloccante e di altri in modalità non bloccante. In questo modo il sistema manda in esecuzione l'istruzione ISO successiva non appena risultano terminati i movimenti programmati come bloccanti.

### 5.1 Programmazione bloccante del percorso

La programmazione bloccante del percorso di un asse avviene tramite l'istruzione Pos[].
La sintassi utilizzata è la seguente:

**Pos["indice asse"] = "posizione programmata"**

dove:
- "indice asse" rappresenta l'identificatore dell'asse che si intende muovere,
- "posizione programmata" indica la quota programmata espressa in *millimetri* [mm.]

Se il CNC controlla 3 assi identificati dai caratteri ascii: X, Y e Z, "indice asse" è uno di questi caratteri in funzione dell'asse che si vuole muovere.
La "posizione programmata" è invece definita tramite una costante numerica, una variabile R o un calcolo matematico complesso.

**Esempio di programmazione**

…
G01 Pos[Z] = (R27 + (R[32 + R1] / R16) –7.8)  Pos[X] = 7.25
…

In questo esempio è richiesto un movimento interpolato degli assi Z e X. L'asse Z deve raggiungere la quota risultato del calcolo matematico alla destra del simbolo di uguale e l'asse X la quota 7.25 mm.
L'istruzione ISO successiva viene eseguita solo quando sia X che Z hanno raggiunto la quota programmata.

Nel seguito verrà descritto come programmare anche la velocità del movimento.

*NOTA*
La forma *"indice asse"* = è equivalente all'espressione *Pos["indice asse"]* = quindi le seguenti scritture hanno il medesimo significato:

**Pos[X] = …** ⇔ **X = …**

## 5.2 Programmazione non bloccante del percorso

La programmazione non bloccante del percorso di un asse avviene tramite l'istruzione Posa[].
La sintassi utilizzata è la seguente:

**Posa["indice asse"] = "posizione programmata"**

dove:
- "indice asse" rappresenta l'identificatore dell'asse che si intende muovere,
- "posizione programmata" indica la quota programmata espressa in *millimetri* [mm.]

Valgono le medesime considerazioni fatte nel paragrafo precedente relativamente all'istruzione Pos[].

**Esempi di programmazione**

Esempio 1:
…
G01 Posa[X] = R27 / R13
…
In questo esempio è richiesto il posizionamento dell'asse X alla quota risultato del calcolo matematico R27 / R13. Dato che l'unico movimento programmato è di tipo non bloccante viene subito mandata in esecuzione l'istruzione ISO successiva.

Esempio 2:
…
G00 Pos[X] = 127.7  Pos[Z] = R[26 + R15] * R10  Posa[Y] = R5 + R4
…

In questo esempio è richiesto il posizionamento dell'asse X alla quota 127.7 mm., il posizonamento dell'asse Z alla quota risultato del calcolo R[26 + R15] * R10 e il posizinamento dell'asse Y alla quota risultato del calcolo R5 + R4. I movimenti di X e Z sono inoltre programmati come bloccanti mentre quello di Y come non bloccante; l'istruzione ISO successiva viene quindi eseguita quando sono terminati i movimenti di X e Z indipendentemente dallo stato del movimento di Y.
La velocità di movimento dei tre assi è quella di rapido (G00) e quindi il movimento risultante non è interpolato. Non ha del resto significato eseguire un movimento non bloccante in una interpolazione poiché gli assi programmati arriverebbero in quota contemporaneamente. Per questa ragione l'esecuzione di un'istruzione di movimento mista (bloccante/non bloccante) ha senso solo se gli assi sono programmati ha velocità diverse (e quindi senza interpolazione) o alla loro velocità di rapido (G00).

Esempio 3:

…
G01 Pos[X] = 12.4  Pos[Y] = R67 / R12  Posa[Z] = R45  Pos[V] = R[56 + R4] * R7
…

In questo esempio solo il movimento di Z è programmato come non bloccante. Anche se è stata impostata la modalità G01 (interpolazione) il fatto di aver programmato in una medesima istruzione il movimento di più di tre assi ne provoca la disattivazione; in questo modo gli assi eseguiranno un movimento indipendente funzione unicamente della velocità di lavoro impostata e della quota finale da raggiungere. Il passaggio all'istruzione ISO successiva avviene, di conseguenza, non appena sono terminati i movimenti programmati per X, Y e V.

Esempio 4:

…
G01 Posa[X] = 15.0  Posa[Y] = R45
…

In questo esempio sia il movimento di X che di Y è programmato come non bloccante ed è inoltre richiesta una interpolazione lineare. In questo caso l'istruzione ISO successiva viene mandata subito in esecuzione senza attendere il termine del posizionamento di X e Y.

***NOTA IMPORTANTE***
**Qualora si programmino movimenti non bloccanti è comunque importante essere sicuri che il posizionamento così richiesto sia terminato prima che venga mandata in esecuzione un'istruzione che programmi un altro posizionamento per il medesimo asse. Se questa condizione non viene rispettata il CNC segnalerà una condizione di emergenza arrestando l'esecuzione del programma.**

## 6.0 Programmazione delle velocità

In questa sezione viene descritto come programmare la velocità movimento di uno o più assi del sistema qualora non sia attiva la modalità di rapido G00.

La velocità di ogni asse del sistema può essere programmata utilizzando una delle seguenti modalità:
- velocità di lavoro comune F,
- velocità di lavoro dedicata ad un singolo asse Fa[].

L'impostazione di velocità viene ignorata qualora si attiva la modalità di rapido G00, mente è sempre necessaria nelle istruzioni di percorso quando la G00 non è attiva.

### 6.1 Velocità di lavoro comune F

L'istruzione F deve essere utilizzata sempre assieme alle istruzioni di percorso, Pos[] e Posa[], e permette di definire la velocità di lavoro con cui gli assi eseguiranno il percorso programmato.
La sintassi utilizzata è la seguente:

**"istruzione di percorso" F = "valore velocità"**

dove:
- "istruzione di percorso" indica la programmazione di percorso di uno o più assi come descritto nei paragrafi precedenti,
- "valore velocità" esprime la velocità da impostare in *mm./sec*.

La velocità così impostata è comune per tutti gli assi programmati nell'istruzione.

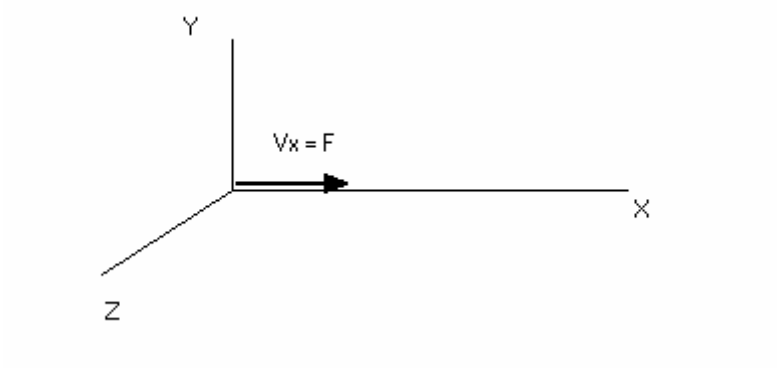
**Esempi di programmazione**

…
G01 Pos[X] = R34 Pos[Z] = R54 + R7 F = (R34 / R[45 + R5]) * R32
…
In questo esempio la velocità F è il risultato dell'espressione matematica (R34 / R[45 + R5]) * R32.

*NOTA*
Se il movimento programmato non è una interpolazione la velocità impostata è la reale velocità con cui ogni singolo asse esegue il posizionamento, viceversa il valore impostato si riferisce alla traiettoria descritta dagli assi che si muovono in interpolazione; quindi la reale velocità di ogni singolo asse è la componente, della velocità F impostata, lungo le direzioni di movimento dell'asse.

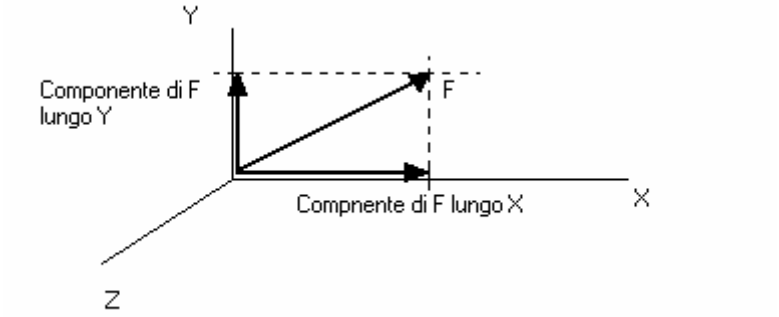
**Movimento non interpolato:**

G01 Pos[X] = 200.0  F = 150.0




**Movimento interpolato:**

G01 Pos[X] = 200.0  Pos[Y] = 150.5 F = 150.0



## 6.2 Velocità di lavoro dedicata Fa

L'istruzione Fa[] deve essere utilizzata sempre assieme alle istruzioni di percorso, Pos[] e Posa[], e permette di definire la velocità di lavoro dedicata per un particolare asse.
La sintassi utilizzata è la seguente:

**"istruzione di percorso" Fa["indice asse"] = "valore velocità"**

dove:

- "istruzione di percorso" indica la programmazione di percorso di un asse come descritto nei paragrafi precedenti,
- "indice asse" indica l'asse a cui la velocità si riferisce,
- "valore velocità" esprime la velocità da impostare in *mm./sec*.

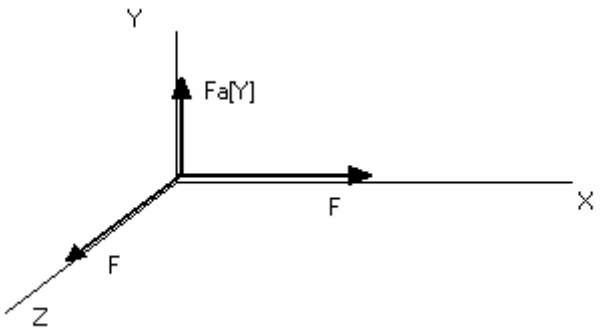
La velocità così impostata è dedicata solo per l'asse, programmato nell'istruzione, con identificatore "indice asse".

**Esempi di programmazione**

…
G01 Pos[X] = R34 Pos[Y] = 200.5 Fa[Y] = R13 / 2.0 Pos[Z] = R54 + R7 F = R[45 + R5]) * R32
…
In questo esempio F imposta la velocità di lavoro comune per gli assi X e Z, programmati nell'istruzione, mentre Fa[Y] imposta la velocità di lavoro dedicata per il movimento del solo asse Y.
Anche se è stata impostata la modalità G01 (interpolazione lineare), il fatto di aver programmato per uno dei tre assi una velocità di lavoro dedicata comporta l'automatica disattivazione dell'interpolazione. In altri termini i tre assi partiranno contemporaneamente ma X e Z raggiungeranno la posizione finale in funzione della quota programmata e della velocità di lavoro F impostata mentre Y raggiungerà la posizione finale in funzione della quota e della velocità di lavoro Fa[Y].

*NOTA*
Esempio di impostazione delle velocità

G01 Pos[X] = 200.6 Pos[Y] = R13 Fa[Y] = R32 Pos[Z] = R4 F = R2 + R5



# 7.0 Istruzioni generiche

Vengono ora descritte ulteriori istruzioni, di tipo generale, che il linguaggio mette a disposizione del programmatore.

## 7.1 Lettura della velocità massima

L'istruzione $MAX_VELO[] permette di recuperare l'informazione relativa alla velocità massima di sistema ammessa per un particolare asse.
La sintassi utilizzata è la seguente:

**$MAX_VELO[“indice asse”]**

dove:
- “indice asse” è l'identificatore dell'asse interessato.

Il valore recuperato, espresso in *mm./sec.* deve essere assegnato ad una variabile prima di essere utilizzato un qualche operazione di calcolo.

**Esempio di programmazione**

…
R22 = $MAX_VELO[Y]
…
In questo esempio la massima velocità di sistema ammessa per l'asse Y viene assegnata alla variabile di indice 22. In altri termini se la velocità massima consentita per l'asse Y è pari a 1220.0 mm./sec, dopo l'esecuzione dell'istruzione il contenuto di R22 sarà pari a 1220.0.

*NOTA*
La lettura della velocità massima può essere utilizzata, ad esempio, per eseguire successive impostazioni di velocità come valore percentuale del suo valore massimo:

F = R22 * R1 / 100

con R1 variabile compresa tra 0 ÷ 100

## 7.2 Lettura delle quote di estrema corsa

Le istruzioni $POS_LIMIT_PLUS[] e $POS_LIMIT_MINUS[] permettono di recuperare l'informazione relativa alla massima corsa di ogni singolo asse rispetto al punto di azzeramento della macchina.
La sintassi utilizzata è la seguente:

**$POS_LIMIT_PLUS[“indice asse”]**
oppure
**$POS_LIMIT_MINUS[“indice asse”]**

dove:
- “indice asse” è l'identificatore dell'asse interessato.

Il valore recuperato, espresso in *mm.*, deve essere assegnato ad una variabile prima di essere utilizzato un qualche operazione di calcolo.

Il valore ritornato da $POS_LIMIT_PLUS[] indica la massima corsa positiva dell'asse rispetto al punto di azzeramento della macchina, il valore ritornato da $POS_LIMIT_MINUS[] indica viceversa la massima corsa negativa.

**Esempio di programmazione**

…
R10 = $MAX_LIMIT_PLUS[X]
…
In questo esempio la massima corsa positiva ammessa per l'asse X viene assegnata alla variabile di indice 10. In altri termini se, rispetto al punto di zero, l'asse X può raggiungere la quota massima pari a 4500.5 mm., dopo l'esecuzione di questa istruzione il contenuto di R10 è pari a 4500.5.

…
R10 = $MAX_LIMIT_MINUS[X]
…
In questo esempio la massima corsa negativa ammessa per l'asse X viene assegnata alla variabile di indice R10.

## 7.3 Lettura della posizione

E' possibile leggere la posizione ideale oppure la posizione finale di ogni singolo asse.

### 7.3.1 Lettura della posizione ideale

L'istruzione $aa_iw[] permette di recuperare l'informazione relativa alla quota ideale di uno specifico asse.
La sintassi utilizzata è la seguente:

**$aa_iw[“indice asse”]**

dove:
- “indice asse” è l'identificatore dell'asse interessato.

Il valore ritornato da questa istruzione indica la quota ideale, espressa in mm. e riferita al punto di azzeramento della macchina, al momento dell'esecuzione dell'istruzione.
**Per questa ragione, se l'asse specificato è in movimento, il valore ritornato non corrisponde alla posizione di fine movimento ma al valore di posizione dell'asse nell'istante di esecuzione dell'istruzione.**
Il valore ritornato può essere assegnato ad una variabile oppure utilizzato direttamente in qualche operazione di confronto.

**Esempio di programmazione**

```
…
R10 = $aa_iw[Z]
…
```

In questo esempio la posizione ideale dell'asse Z viene assegnata alla variabile di indice 10.

```
…
IF ($aa_iw[Z] > R27)
        …
        …
ENDIF
…
```

In questo esempio se la posizione dell'asse Z supera il valore contenuto in R27, viene eseguito il corpo di programma contenuto nella sezione IF-ENDIF.

### 7.3.2 Lettura della posizione finale

L'istruzione $aa_fw[] permette di recuperare l'informazione relativa alla quota finale di uno specifico asse.
La sintassi utilizzata è la seguente:

**$aa_fw[“indice asse”]**

dove:

- “indice asse” è l’identificatore dell’asse interessato.

Il valore ritornato da questa istruzione indica la quota finale, espressa in mm. e riferita al punto di azzeramento della macchina, al momento dell’esecuzione dell’istruzione.
**Per questa ragione, se l’asse specificato è in movimento, il valore ritornato corrisponde comunque alla posizione di fine movimento, mentre corrisponde alla posizione attuale se l’asse è a riposo.**
Il valore ritornato può essere assegnato ad una variabile oppure utilizzato direttamente in qualche operazione di confronto.

**Esempio di programmazione**

```
…
R10 = $aa_fw[Z]
…
```

In questo esempio la posizione finale dell’asse Z viene assegnata alla variabile di indice 10.

```
…
IF ($aa_iw[Z] > R27)
        …
        …
ENDIF
…
```

In questo esempio se la posizione dell’asse Z supera il valore contenuto in R27, viene eseguito il corpo di programma contenuto nella sezione IF-ENDIF.

## 7.4 Lettura di un ingresso digitale

L'istruzione $aa_in[] permette di recuperare l'informazione relativa allo stato logico di un ingresso digitale (PLC).

La sintassi utilizzata è la seguente:

**$aa_in["indice ingresso"]**

dove:
- "indice ingresso" identifica il punto di acquisizione dell'ingresso digitale.

Il valore ritornato da questa istruzione è 0 o 1, a seconda che l'ingresso digitale sia rispettivamente acceso o spento, e può essere assegnato ad una variabile oppure utilizzato direttamente in una istruzione di confronto.

**Esempio di programmazione**

```
…
R10 = $aa_in[27]
…
```

In questo caso viene assegnato, alla variabile di indice 10, il valore 0 se il punto di acquisizione 27 è spento, o il valore 1 se è acceso.

```
…
IF ($aa_in[27] == 1)
        …
        …
ENDIF
…
```

In questo esempio se il valore fisico dell'ingresso numero 27 è pari a 1 viene eseguito il copro di programma contenuto nella sezione IF_ENDIF.

*NOTA*
Il punto di acquisizione identifica il numero di ingresso digitale del PLC. Dato che ogni scheda di ingresso/uscita è composta da 2 porte con 8 punti di acquisizione (0 ÷ 7), riferirsi all'ingresso numero 27 significa riferirsi al punto di ingresso fisico numero 4 della porta 2 della seconda scheda.

## 7.5 Scrittura di una uscita digitale

L'istruzione $aa_out[] permette di impostare fisicamente una uscita digitale del sistema (PLC).
La sintassi utilizzata è la seguente:

**$aa_out["indice uscita"]**

dove:

- “indice uscita” identifica il punto di impostazione dell’uscita digitale.

Il valore impostato con questa istruzione permette di forzare a 1 o a 0 logico lo stato di una uscita digitale.

**Esempio di programmazione**

…
$aa_out[32] = 0
…
In questo esempio viene forzato a zero logico lo stato dell’uscita digitale identificata dal punto numero32.

…
$aa_out[32] = 1
…
In questo esempio viene forzato a uno logico lo stato dell’uscita digitale identificata dal punto numero32.

*NOTA*
Dato che le porte di ingresso/uscita sono configurabili punto per punto via software, un’uscita digitale può anche essere riletta come se si trattasse un punto di acquisizione (ingresso).
Sono quindi ammesse le stesse operazioni descritte nel paragrafo 8.4 per l’istruzione $aa_in[], utilizzando l’istruzione $aa_out[].
Esempio:

R10 = $aa_out[32]
In questo caso viene riletto il valore attuale dell’uscita numero 32 e assegnato alla variabile di indice 10.

## 7.6 Lettura/scrittura di un marker

Il PLC mette a disposizione 200 flag di sistema, che possono assumere lo stato logico 0 o 1, denominati *marker*. Questi flag possono essere utilizzati dal programmatore come punti di ingresso/uscita non fisici ma puramente logici.
L'istruzione $aa_marker[] permette di leggere, o forzare, lo stato di ogni singolo flag.

La sintassi utilizzata è la seguente:

**$aa_marker[“indice flag”]**

dove:
- “indice flag” identifica un particolare flag di sistema e deve avere un valore compreso tra 0 e 200.

Questa istruzione può essere utilizzata esattamente come le istruzioni $aa_in[] e $aa_out[].

**Esempi di programmazione**

```
…
$aa_marker[18] = 1
…
```

In questo caso viene assegnato il valore 1 al flag di sistema numero 18.

```
…
R10 = $aa_marker[18]
…
```

In questo caso viene assegnato il valore del flag di sistema numero 18 alla variabile di indice 10.

```
…
IF ($aa_marker[18] == 1)
        …
        …
ENDIF
…
```

In questo caso se il valore del flag di sistema numero 18 è pari a 1 viene eseguito il corpo del programma contenuto nella sezione IF-ENDIF.

| |
|---|
| *NOTA*<br>Gli unici valori ammessi per i flag di sistema sono 1 e 0. |

## 7.7 Programmazione degli anticipi

Abbiamo visto che quando si programma il percorso di un asse in modalità *bloccante*, l'istruzione successiva viene eseguito solo al termine del posizionamento dell'asse, mentre quando si utilizza la modalità *non bloccante* l'istruzione successiva viene eseguita subito dopo la partenza dell'asse.
A volte può essere necessario eseguire un gruppo di istruzioni quando il posizionamento di un particolare asse ha raggiunto una quota desiderata anticipando, di fatto, il proseguimento del programma rispetto alla modalità di programmazione bloccante.
L'istruzione WHEN-CANCEL permette di ottenere la modalità di funzionamento sopra descritta.

La sintassi utilizzata è la seguente:

**WHEN "condizione logica" DO**
**…**
**…**
**CANCEL**
**"istruzione di percorso"**

dove:

- "condizione logica" indica la condizione che deve essere verificata e deve basarsi sulla lettura della quota ideale dell'asse sul quale si vuole eseguire l'anticipo,
- "istruzione di percorso" è una istruzione di programmazione di uno o più assi che deve necessariamente contenere la programmazione *bloccante* dell'asse testato nella "condizione logica".

In altre parole il corpo di programma contenuto nella sezione WHEN-CANCEL viene eseguito quando la quota ideale dell'asse, sul quale si vuole eseguire l'anticipo, programmato in modalità bloccante nell'istruzione di percorso, soddisfa la condizione di confronto che compare tra le parole chiave WHEN e DO.
Se al termine del movimento programmato la condizione non è risultata soddisfatta, ad esempio perché la quota programmata è minore della quota di confronto, l'istruzione WHEN viene abortirta.

Il linguaggio permette di programmare fino a 3 anticipi nel seguente modo:

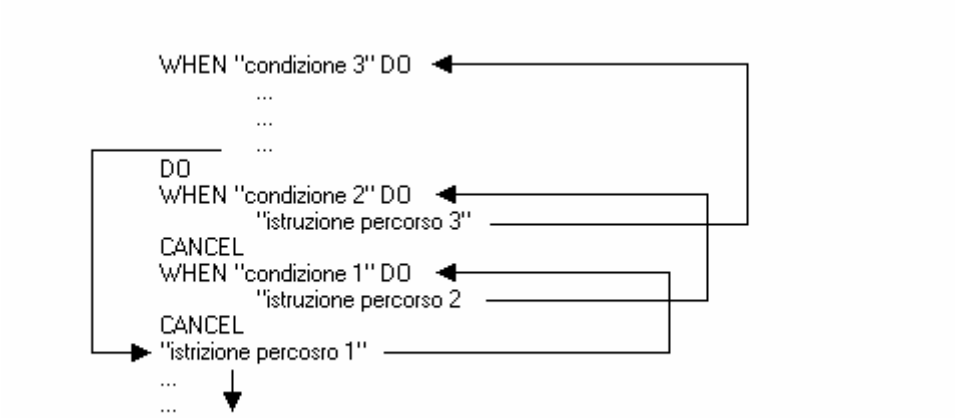
**WHEN "condizione logica 3" DO**
**…**
**…**
**…**
**CANCEL**

**WHEN "condizione logica 2" DO**
**"istruzione di percorso 3"**
**CANCEL**

**WHEN "condizione logica 1" DO**
**"istruzione di percorso 2"**
**CANCEL**
**"istruzione di percorso 1"**

In questo caso l'istruzione di percorso 1 attiverà il primo blocco quando è soddisfatta la condizione logica 1, quindi l'esecuzione dell'istruzione di percorso 2 attiverà il secondo blocco quando viene soddisfatta la condizione logica 2, ed infine l'istruzione di percorso 2 attiverà il terzo blocco quando viene soddisfatta la condizione logica 3.
Mentre i corpi del primo e del secondo anticipo possono contenere unicamente l'istruzione di percorso che attiva il blocco precedente, l'ultimo anticipo può contenere un pacchetto di istruzioni qualsiasi.
Il programma continuerà con l'istruzione successiva alla "istruzione di percorso 1" quando sono esaurite tutte le operazioni e i movimenti programmati nei vari blocchi WHEN-CANCEL e i relativi movimenti di attivazione.

*NOTA*
Diagramma di funzionamento:




**Esempio di programmazione**

…
WHEN $aa_iw[X] > (R27 / R12) DO
…
…
CANCEL
G01 Pos[X] = R7 + R5  Pos[Z] = R32  F= 120.0
…
In questo esempio viene programmato il movimento dell'asse X, anche se assieme a quello dell'asse Z, alla quota risultato del calcolo R7 + R5; quando la posizione ideale dell'asse X diventa maggiore al risultato del calcolo R27 / R12, viene eseguito il corpo di programma contenuto nella sezione WHEN-CANCEL.
Alla fine dell'esecuzione delle istruzioni programmate nella sezione WHEN-CANCEL e dei movimenti programmati nell'istruzione di percorso il programma esegue l'istruzione successiva.
Se la quota di confronto, (R27 / R12), è maggiore di quella a cui viene programmato l'asse, (R7 + R5), il blocco WHEN viene automaticamente annullato appena termina il movimento dell'asse X.

```
…
WHEN $aa_iw[Y] < R25 DO
        …
        …
CANCEL
WHEN $aa_iw[X] > (R27 / R12) DO
        G01 Pos[Y] = R67 F = R18
CANCEL
G01 Pos[X] = R7 + R5  Pos[Z] = R32  F= 120.0
…
```

In questo esempio l'esecuzione dell'anticipo WHEN, associato al movimento dell'asse X e funzionante come descritto nell'esempio precedente, programma un movimento dell'asse Y innescando il secondo anticipo WHEN, associato all'asse Y.
Quando il secondo anticipo è soddisfatto, ossia la quota raggiunta da Y diventa minore del contenuto della variabile R25, viene eseguito il corpo contenuto nella realtiva sezione WHN-CANCEL.

*NOTA*
Se l'istruzione da eseguire nella sezione WHEN è una sola è possibile usare anche la seguente sintassi:

**WHEN "condizione logica"**
**"istruzione"** ⇔ **WHEN "condizione logica" DO "istruzione"**
**CANCEL**

# 8.0 Funzioni PLC: M

Le funzioni M permettono di richiamare e mandare in esecuzione complesse sequenze di PLC.
Come per le istruzioni di percorso anche le funzioni M possono essere richiamate secondo due modalità:

- esecuzione di una sequenza di PLC in modalità BLOCCANTE,
- esecuzione di una sequenze di PLC in modalità NON BLOCCANTE.

Nella modalità BLOCCANTE non viene eseguita la successiva istruzione ISO fino a quando la sequenza di PLC, attivata dalla M chiamata, non è terminata.
Nella modalità NON BLOCCANTE, viceversa, viene continuata l'esecuzione del programma subito dopo avere attivato la sequenza di PLC associata alla M chiamata.

## 8.1 Programmazione di una M bloccante

La sintassi utilizzata per eseguire una sequenza di PLC in modalità *bloccante* è la seguente:

**M"indice sequenza"**

dove:

- "indice sequenza" è un valore numerico ed indica l'identificatore di una particola sequenza PLC.

**Esempio di programmazione**

…
M23
…
In questo esempio viene attivata, e quindi richiesta l'esecuzione, della sequenza di PLC identificata con il numero 23.
Solo quando la sequenza in questione risulta terminata viene eseguita l'istruzione ISO successiva.

## 8.2 Programmazione di una M non bloccante

La sintassi utilizzata per eseguire una sequenza di PLC in modalità *non bloccante* è la seguente:

**M=NB("indice sequenza")**

dove:

- "indice sequenza" è un valore numerico ed indica l'identificatore di una particola sequenza PLC.

**Esempio di programmazione**

…
M=NB(34)
…
In questo esempio viene attivata, e quindi richiesta l'esecuzione, della sequenza di PLC identificata con il numero 34.
Dato che ne è stata richiesta l'esecuzione in modalità non bloccante viene subito mandata in esecuzione l'istruzione ISO successiva.

| |
|---|
| *NOTA*<br>Le sequenze di PLC sono normalmente sviluppate in maniera dedicata per ogni singola applicazione del CNC. |

## 8.3 Funzioni M particolari

Esistono del funzioni M, richiamabili solo in modalità bloccante, alle quali non è associata una funzionalità di PLC ma che hanno dei significati particolari sotto descritti.

- **Funzione M30:** deve essere utilizzata come identificatore di fine programma. In altre parole quando il programma ISO, durante la sua esecuzione, incontra l'istruzione M30 termina.

- **Funzione M17:** deve essere utilizzata come identificatore di fine sottoprogramma. In altre parole quando il programma ISO incontra l'istruzione M17 ritorna da un sottoprogramma chiamato al programma chiamante. Da ciò si evince che l'istruzione M17 deve sempre essere presente in qualsiasi sottoprogramma per indicarne la fine.

- **Funzione M0:** ad essa è associata una particolare sequenza di PLC che provoca l'arresto dell'esecuzione del programma ISO in corso fino alla nuova pressione del pulsante di start.