I linguaggi IL e ST

Lo standard IEC 61131-3 offre due strumenti di programmazione testuale: una lista di istruzioni in stile 'assembly' e un linguaggio strutturato di alto livello

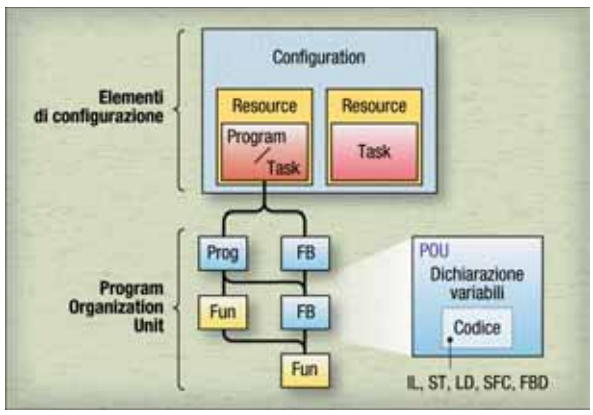
Fonte: www.visaliaelitepoolservice.com

Massimo Giussani

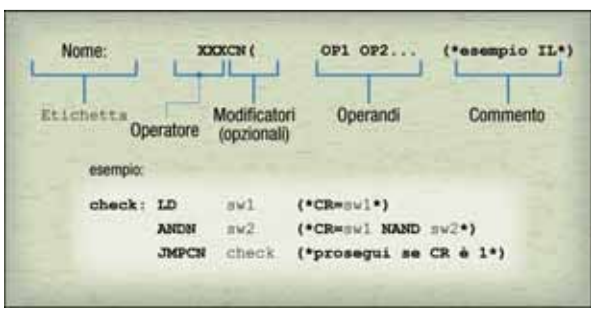
I controllori logici programmabili (PLC) presentano un'architettura hardware ottimizzata per compiti di controllo basati su operazioni prevalentemente booleane, retaggio di un passato di automazione imperniata sull'apertura e chiusura di contatti per mezzo di relè. Con il passaggio ai sistemi digitali, i progettisti di sistemi di automazione industriale hanno trovato un nuovo interlocutore, il microprocessore, il cui linguaggio macchina è rappresentato da sequenze di bit poco maneggevoli. I produttori di PLC hanno così prodotto negli anni una serie di linguaggi evoluti atti a semplificare la transizione verso la nuova tecnologia e a velocizzare la progettazione e configurazione dei sistemi di automazione e controllo. I linguaggi di più basso livello sono quelli di tipo 'assembly': ricorrono a codici mnemonici e a etichette per rappresentare in maniera più comprensibile al programmatore le sequenze di bit di 'operandi' e operatori da 'dare in pasto' alla CPU. Linguaggi di questo tipo hanno fatto la loro comparsa su PLC di piccole e medie dimensioni e sono stati utilizzati con profitto per la realizzazione di programmi non eccessivamente convoluti. Con il crescere della complessità dei problemi di automazione si è fatta sentire la necessità di linguaggi con un più alto livello di astrazione, dotati di una sintassi più ricca e, possibilmente, più vicina a quella del linguaggio naturale. Linguaggi sulla falsariga di C, Pascal e Basic hanno aperto la strada a un approccio strutturato alla programmazione dei PLC, grazie a costrutti iterativi e decisionali, che gestiscono il flusso delle istruzioni.

Lo standard IEC 61131-3

I PLC erano solitamente configurati e utilizzati da personale tecnico più avvezzo alla logica dei relè che alla programmazione di elaboratori elettronici, perciò quasi tutti i produttori di PLC introdussero uno strumento di configurazione in grado di tradurre in linguaggio macchina gli schemi di connessione di contatti e bobine, che contraddistinguevano i sistemi analogici. Nacquero così diverse implementazioni di quello che sarà standardizzato come il linguaggio grafico LD, il cui nome 'Ladder Diagram' è dovuto al fatto che le connessioni si dispongono come i pioli di una scaletta (in inglese 'ladder').
Altri linguaggi grafici, basati sull'interconnessione di blocchi funzione e la descrizione di macchine agli stati finiti, vennero introdotti negli anni dai diversi produttori, spesso in più varianti e dialetti tra loro incompatibili.
La normativa IEC 61131 ha rappresentato un primo importante passo nella direzione della standardizzazione e dell'apertura delle architetture dei sistemi di controllo industriale. In particolare, la terza parte di questa norma, relativa ai linguaggi di programmazione, offre una descrizione del modello software orientata alla modularità e al riutilizzo del codice. Si tratta di un modello stratificato di tipo gerarchico, in cui ogni strato contiene una parte dichiarativa, che specifica le variabili globali, locali o d'interfaccia utilizzate e le proprie competenze (in termini dei sottolivelli che racchiude).
Lo standard stabilisce la struttura di tre forme di POU

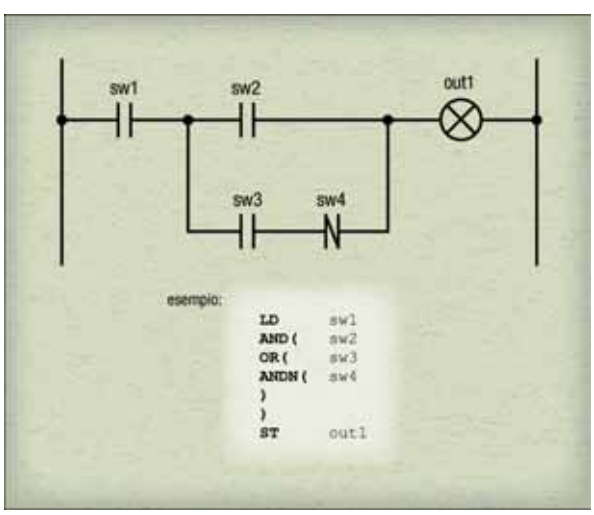


Struttura dei programmi di un PLC secondo la norma IEC 61131-3



Struttura di una riga di istruzioni IL ed esempio di uso dei modificatori

(Program Organization Unit): programmi ('program'), blocchi funzionali ('function block') e funzioni ('function'). Alla parte dichiarativa le POU fanno seguire le istruzioni che implementano la loro funzione in uno dei cinque linguaggi standardizzati ammessi dalla norma: Instruction List (IL), Structured Text (ST), Ladder Diagram (LD), Sequential Function Chart (SFC) e Function Block Diagram (FBD).



Esempio di codice IL e relativo schema LD

Instruction List

È il linguaggio testuale di più basso livello definito dallo standard IEC 61131-3: tutti gli altri linguaggi possono essere convertiti in codice IL, pur non essendo sempre vero il contrario. Per la sua vicinanza alla macchina hardware, questo linguaggio è particolarmente indicato per scrivere codice altamente ottimizzato. Realizzare un'applicazione in IL è tuttavia poco agevole, per via del fatto che è necessario preoccuparsi di ogni dettaglio implementativo, di fatto sobbarcandosi anche gli oneri che sono solitamente di pertinenza del compilatore di un linguaggio di alto livello. La maggior parte dei programmatori ricorre a IL per ottimizzare le porzioni di codice da eseguire con la massima efficienza, oppure quando non vi sono alternative, ad esempio perché si sta utilizzando un dispositivo palmare per la programmazione del PLC.
Un programma IL è costituito da una sequenza di istruzioni disposte su righe separate. Ogni istruzione è composta da un operatore, rappresentato da un codice mnemonico, ad esempio LD per LoaD (caricare in memoria), eventualmente specializzato tramite uno o più modificatori, e da uno o più operandi. Il codice può essere preceduto da un'etichetta di testo, che identifica la riga di cui fa parte e permette di gestire in maniera agevole le istruzioni di salto e le chiamate a subroutine. È inoltre possibile inserire dei commenti a fine riga, racchiudendoli tra parentesi tonde con asterischi (* *).
La sintassi tipica prevede un operatore e un operando, con il risultato CR (Current Result) implicitamente assegnato a un registro molto particolare della CPU: l'accumulatore. Dato che quest'ultimo conserva il risultato dell'ultima operazione effettuata, può essere utilizzato come secondo operatore in un'operazione binaria, prima di ricevere il risultato dell'operazione stessa.
Ad esempio la sequenza di istruzioni:

```
start:  LD    in1
        AND   in2    (*CR = in1 AND in2*)
        ST    out1
```

carica nell'accumulatore il contenuto della variabile in1 (definita nella parte dichiarativa della POU), effettua un'operazione di AND logico con il contenuto della variabile in2, memorizzando il risultato corrente nell'accumulatore, poi trasferisce tale risultato nella variabile out2.
Deve essere premura del programmatore fare in modo che tutti gli operandi, quindi anche il contenuto dell'accumulatore quando esso sia un operando implicito, siano compatibili tra loro per tipo.

I costrutti del linguaggio IL

Il linguaggio IL descritto dallo standard IEC prevede diversi tipi di operatori, a partire da quelli deputati al trasferimento dei dati da e verso l'accumulatore (ST e LD, che stanno per STore e LoaD) e per forzare dei valori logici nell'operando (S e R, per Set e Reset). Sono ovviamente presenti operatori booleani, come AND, OR, XOR, e matematici, come ADD, SUB, Mult e DIV, che effettuano le operazioni tra il risultato corrente memorizzato nell'accumula-

tore e l'operando che li segue. Il modificatore N permette di applicare una negazione logica all'operando che lo segue.
Ad esempio:

```
LDN     in1     (*CR=NOT(in1)*)
AND     in2     (*CR=NOT(in1) AND in2*)
R       out1    (*out1=FALSE se CR=TRUE*)
```

Per i confronti sono disponibili gli operatori LT (Less Than), LE (Less Equal), EQ (EQual), NE (Not Equal), GE, GT, che confrontano il valore dell'accumulatore con l'operando attribuendo a CR l'esito booleano.
Sono presenti un operatore di salto (JMP) e di chiamata (CAL), che accettano come operando l'etichetta che specifica la linea di codice alla quale passare. Il ritorno da una subroutine avviene quando si incontra una istruzione RET. Questi operatori accettano il modificatore C per subordinare l'esecuzione dell'istruzione al contenuto booleano dell'accumulatore (o al suo complementare, se si aggiunge anche il modificatore N).
Ad esempio:

```
CAL     routine1(*esegui routine1*)
CALCN   routine2(*esegui routine2 se CR=FALSE*)
JMPC    fine    (*salta a fine se CR=TRUE*)
```

Nel linguaggio IL le parentesi tonde permettono di comporre più istruzioni; la parentesi aperta è sostanzialmente un modificatore applicabile al codice mnemonico dell'operando. Gli operandi delle istruzioni ai livelli di annidamento più bassi vengono memorizzati in uno stack, in modo da liberare l'accumulatore per le istruzioni più interne, fino a quando non si raggiunge il massimo livello di annidamento e si può cominciare a ripercorrere la sequenza a ritroso, prelevando i valori precedentemente salvati.

Structured Text

È questo l'altro linguaggio di programmazione in formato di solo testo standardizzato dalla norma IEC. Come lascia presagire il nome, si tratta di un linguaggio di alto livello, che porta la programmazione strutturata nel mondo dei PLC.
Il linguaggio ST è dotato di costrutti decisionali e iterativi che permettono di gestire il flusso delle istruzioni. Un primo evidente vantaggio di questo approccio è rappresentato dalla maggiore comprensibilità del codice, la cui sintassi è meno criptica di quella che caratterizza IL.
Un programma in linguaggio ST è composto da istruzioni separate da punto e virgola (;). Una stessa riga di programma può ospitare più istruzioni, oppure una singola istruzione può essere scomposta su più righe per maggiore chiarezza. I caratteri di fine riga (EOL) e di ritorno a capo (LF) sono considerati alla stregua di spazi, proprio per rendere possibile l'indentazione del codice. I commenti, rappresentati come in IL da testo racchiuso da parentesi con asterischi (* *), possono essere posizionati in qualsiasi punto all'interno delle istruzioni. Tutto questo contribuisce a migliorare la leggibilità dei programmi.
In ST è poi possibile assegnare a delle variabili definite dall'utente dei valori ricavati dalla valutazione di espressioni. L'assegnamento è rappresentato da :=

variabile := espressione

Le istruzioni ST sono sostanzialmente liste di espressioni;

Tabella 1: Principali costrutti del linguaggio IL

Operatore	Mod.	Effetto
Assegnamento		
LD	N	Carica operando nell'accumulatore
ST	N	Assegna l'accumulatore nell'operando
S, R		Quando l'accumulatore ha valore booleano vero carica nell'operando il valore booleano vero (Set) o falso (Reset)
Matematici		
AND, OR, XOR	N, (	Assegna all'accumulatore il risultato delle operazioni booleane tra accumulatore e operando
NOT		Assegna all'accumulatore il complemento a 1 del valore booleano dell'operatore
ADD, SUB, MUL, DIV, MOD	(	Assegna all'accumulatore il risultato delle operazioni matematiche tra accumulatore e operando
Confronto		
LT, LE, NE, EQ, GE, GT	(	Assegna all'accumulatore il risultato booleano del confronto tra accumulatore e operando
Gestione flusso		
JMP	C, N	Salta alla riga indicata dall'operando
CALL	C, N	Chiama la subroutine indicata dall'operando
RET	C, N	Forza il ritorno da una POU
(		Termina una sequenza iniziata da un'istruzione modificata con (

Tabella 2: Principali costrutti del linguaggio ST

Istruzione	Effetto
Assegnamento	
var:=espr	Assegna il risultato della valutazione dell'espressione espr alla variabile var
Matematici	
AND (&), OR, XOR	Restituisce il valore dell'operazione logica specificata con notazione infissa
NOT (!)	Restituisce il valore booleano negato dell'operando
+, -, *, /, **, MOD	Restituisce il valore dell'operazione specificata con notazione infissa
(…)	Altera le priorità predefinite nella valutazione di un'espressione
Confronto	
<=, <, <>, =, >=, >	Restituisce il valore booleano del confronto tra gli operandi
Gestione flusso decisionali	
IF…THEN…ELSE	Subordina la valutazione di un'espressione al risultato di un test booleano
CASE…OF…	Sceglie l'espressione da eseguire in base al valore assunto da una variabile selettore
Iterativi	
FOR…TO…BY…DO	Ripete il ciclo un numero predefinito di volte
WHILE…DO…	Ripete il corpo ciclo se e fintanto che il test posto all'inizio è soddisfatto
REPEAT…UNTIL…	Esegue e ripete il corpo del ciclo fintanto che il test posto alla fine è soddisfatto
EXIT	Forza l'uscita da un ciclo prima che si verifichi la condizione di terminazione naturale

un'espressione è composta da operatori dotati di livelli di priorità ben definiti, che agiscono sugli operandi per produrre un valore. Le variabili possono essere ingressi, uscite, valori statici o valori assegnati dinamicamente.
Il linguaggio ST è particolarmente indicato per implementare, con poche righe di codice, elaborazioni con complesse operazioni di calcolo o test condizionali ad alternative multiple.
Operazioni che se implementate nel linguaggio di basso livello IL, comporterebbero una quantità tale di operazioni di salto da rendere scarsamente leggibile e modificabile il programma.

I costrutti del linguaggio ST

Tra gli operatori, al fianco di quelli aritmetici e booleani, hanno particolare importanza le parentesi, che permettono di modificare la priorità di esecuzione delle operazioni, e gli operatori di confronto (<, <=, =, >=, >), che sono usati nei test dei costrutti iterativi e decisionali.
I costrutti condizionali sono di due tipi: un selettore binario nella forma IF... THEN... ELSE… che prevede l'annidamento di ELSEIF, come qui esemplificato:

```
IF test1 THEN espr1;
   ELSEIF test2 THEN espr2;
   ELSE espr3;
END IF;
```

e un selettore multiplo nella forma CASE… OF…

```
CASE var OF
   val1: espr1;
   val2: espr2;
      ELSE  espr3;
END CASE;
```

che offre una forma più compatta, ma meno flessibile, in quanto non si presta alla verifica di test booleani complessi. I costrutti iterativi prevedono un ciclo incondizionato nella forma FOR... DO...,

```
FOR cnt:=ci TO cf BY passo
      DO espr;
END FOR;
```

e due cicli condizionali con verifica per uscita dal ciclo all'inizio o alla fine dello stesso, rappresentati rispettivamente dai costrutti WHILE... DO...

```
WHILE test
     DO espr;
END WHILE;
```

e REPEAT... UNTIL...

```
REPEAT espr
      UNTIL test;
END REPEAT;
```

L'uscita da un ciclo può inoltre essere imposta ricorrendo all'istruzione EXIT. Particolare cautela deve essere esercitata nel definire le condizioni di uscita dal ciclo, non solo per evitare una banale situazione di ciclo infinito, ma anche per evitare che il programma passi troppo tempo nel ciclo stesso, oppure che non restituisca il controllo qualora si renda necessario per il verificarsi di situazioni di emergenza. ■