



Automazione industriale dispense del corso

16. Linguaggio a contatti (Ladder Diagram)

Luigi Piroddi
piroddi@elet.polimi.it

Introduzione

Il *linguaggio a contatti* (o *diagramma a scala*, dall'inglese *ladder diagram*, LD) è il più diffuso linguaggio di programmazione dei PLC.

E' uno standard di fatto del mercato americano.

Caratteristiche principali del LD:

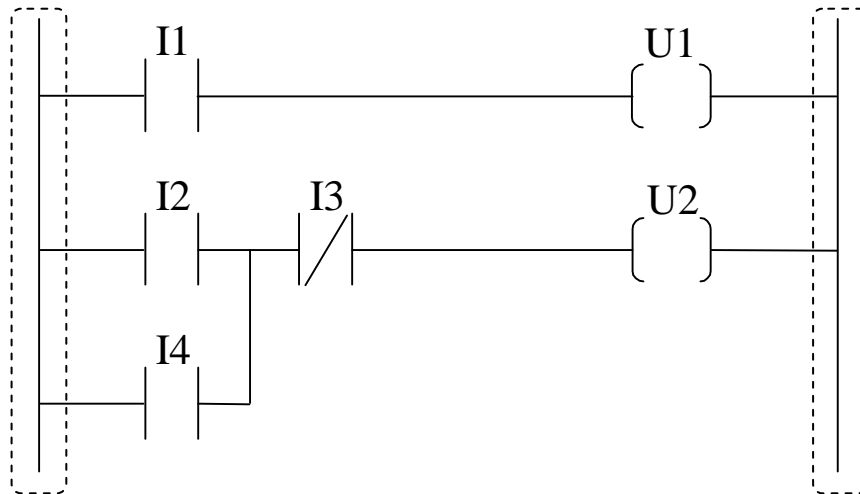
- ▶ è un linguaggio grafico
- ▶ si basa sulla trasposizione, in logica di programmazione, del funzionamento di una rete elettrica molto semplice, il cui obiettivo è di alimentare o non alimentare opportuni *utilizzatori elettrici (bobine)* tramite *interruttori* (chiamati anche *contatti* o *relé*)

La motivazione di questa trasposizione è di tipo storico:

- ▶ prima dei PLC si usavano batterie di relé elettromeccanici (v. quadri a relé)
- ▶ il linguaggio LD è nato per far accettare l'idea di “programmare” (e quindi l'uso del PLC) a chi era abituato a progettare sistemi di controllo logico con relé elettromeccanici

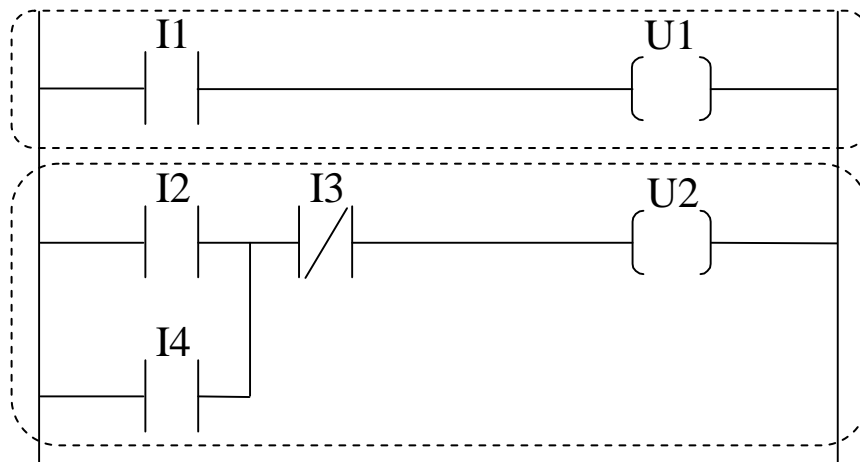
Elementi fondamentali

Gli elementi di base derivano dagli schemi di controllo a relé elettromeccanici:



- ▶ *montanti* della scala
linee verticali laterali che rappresentano l'alimentazione

- ▼ il montante di sinistra è il polo positivo collegato alla tensione V_{CC}
- ▼ il montante di destra è il polo negativo collegato a massa



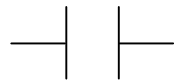
- ▶ *pioli* della scala (o *rung*)
linee orizzontali che contengono

- ▼ a sinistra *contatti* (elettrici) o *relé*
→ ingressi (o variabili interne)
- ▼ a destra *bobine*
→ uscite (o variabili interne)

Elementi di base: contatti

I *contatti* rappresentano i relé e possono essere associati agli ingressi digitali provenienti dal processo (o meglio al loro stato, rappresentato in particolari bit della memoria), oppure a condizioni interne al dispositivo.

contatto
normalmente
aperto



Se il bit associato vale 1, il contatto è chiuso e c'è continuità logica, altrimenti il contatto è aperto e non c'è continuità logica.

Può essere associato a

- bit di ingresso (Ix:y, bit y della word x)
- bit di uscita (Ux:y)
- bit associati a variabili interne (Wx:y)
- bit di stato di temporizzatori e contatori

Quasi tutti i sistemi consentono di dare ai bit anche dei nomi simbolici oltre ai nomi legati all'indirizzo fisico di memoria come quelli sopra, per migliorare la leggibilità dei programmi.

contatto
normalmente
chiuso


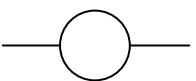
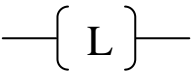





Il contatto è chiuso e assicura la continuità logica se la variabile booleana associata è falsa (bit a 0), altrimenti il contatto è aperto.

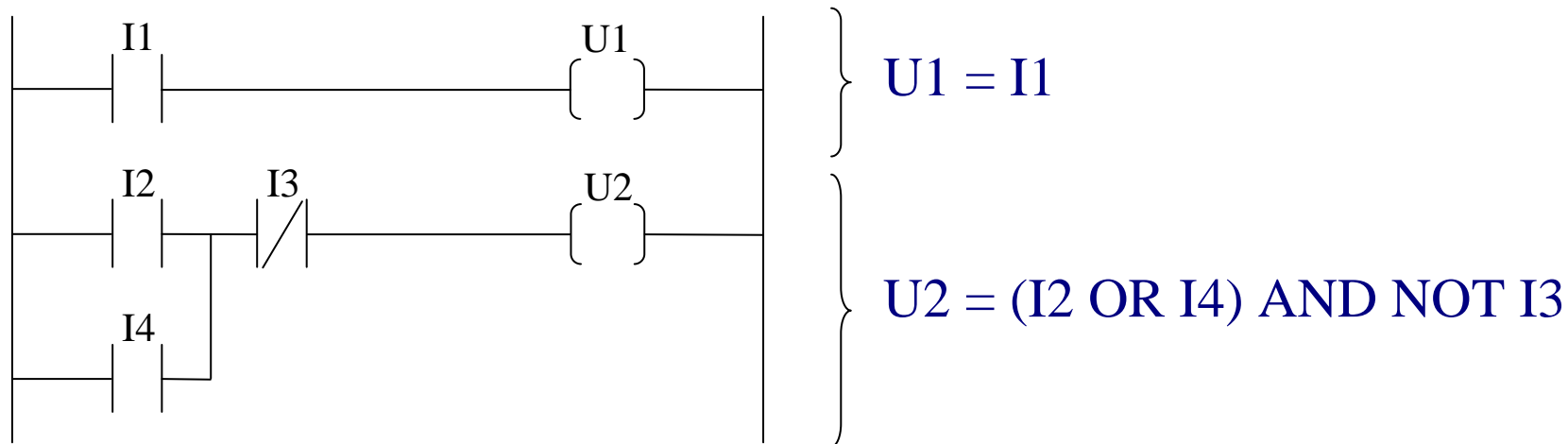
Un contatto associato ad un ingresso equivale ad un'istruzione di LOAD.

Elementi di base: bobine

Le bobine sono associate a bit di memoria, tramite i quali possono comandare uscite digitali oppure variare delle condizioni interne.

bobina	 oppure 	<p>La bobina rappresenta un generico utilizzatore elettrico, ad es. una resistenza, o una lampadina. Può essere associata a</p> <ul style="list-style-type: none"> ▪ bit di uscita (Ux:y) ▪ bit legati a variabili interne (Wx:y) <p>Va inserita sempre all'estremità destra del rung. La bobina si attiva quando passa corrente. Quindi, il bit associato sale al valore logico 1 (ON) se le condizioni logiche alla sua sinistra sono verificate, altrimenti vale 0 (OFF).</p>
bobina di tipo Latch (o Set)	 oppure 	<p>Quando si attiva, il bit associato va a 1 e mantiene tale valore finché non si attiva una bobina associata allo stesso bit.</p> <p>Il funzionamento è molto simile ad un flip-flop sollecitato con un impulso di durata finita sull'ingresso SET.</p>
bobina di tipo Unlatch (o Reset)	 oppure 	<p>Riporta allo stato logico 0 (OFF) un'uscita (v. ingresso RESET di un flip-flop).</p>

Esempio



Nel primo rung, se il contatto I1 è chiuso, la corrente può fluire dal polo positivo a quello negativo, attivando così la bobina U1.

In logica binaria, questo si traduce nell'espressione:

Se $I1 = 1$, allora $U1 = 1$, altrimenti $U1 = 0$,

dove

- ▶ alimentato = vero (1)
- ▶ non alimentato = falso (0)

Programmazione in LD

Per costruire un programma si dispongono uno dopo l'altro i pioli (le istruzioni).

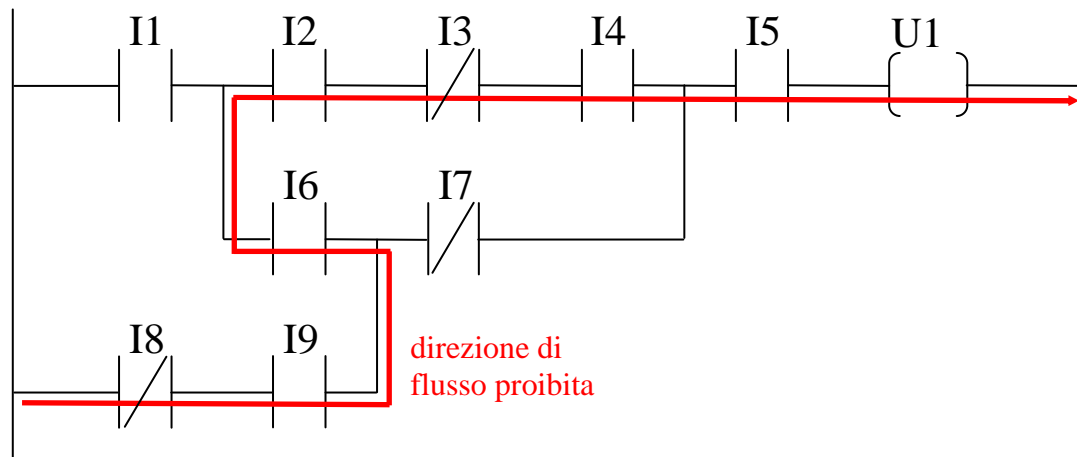
La corrispondenza di un programma LD (per definizione sequenziale) con una rete elettrica (per natura sede di fenomeni simultanei) non può essere esatta!

Occorre specificare:

- ❶ come vengono scanditi i pioli?
- ❷ quando vengono aggiornati ingressi e uscite?

Modalità di scansione dei pioli

I pioli vengono scanditi dall'alto verso il basso e da sinistra verso destra.
La “corrente” non può fluire da destra a sinistra come in una rete elettrica.



U1 si attiva se:

- ▶ $I1 = 1, I2 = 1, I3 = 0, I4 = 1, I5 = 1$
- ▶ $I1 = 1, I6 = 1, I7 = 0, I5 = 1$
- ▶ $I8 = 0, I9 = 1, I7 = 0, I5 = 1$

Se $I1 = 0$ e $I7 = 1$, la bobina non si attiva lungo il cammino I8-I9-I6-I2-I3-I4-I5 neanche se $I8 = 0, I9 = 1, I6 = 1, I2 = 1, I3 = 0, I4 = 1, I5 = 1$. Infatti, il contatto I6 sarebbe percorso in direzione contraria a quella consentita per convenzione.

Lo schema è equivalente all'istruzione logica seguente:

$$U1 = I5 \text{ AND } [(I4 \text{ AND NOT}(I3) \text{ AND } I2 \text{ AND } I1) \text{ OR } (\text{NOT}(I7) \text{ AND } ((I6 \text{ AND } I1) \text{ OR } (I9 \text{ AND NOT}(I8))))]$$

Sincronizzazione I/O

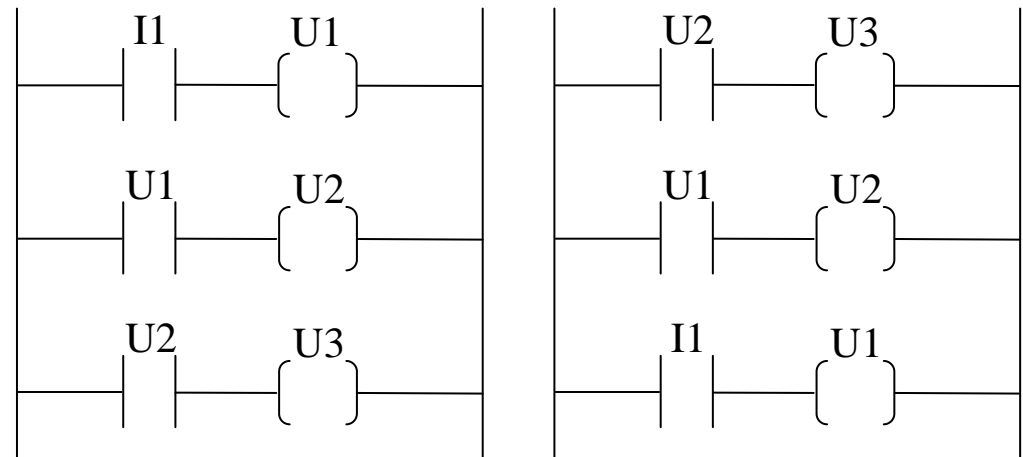
La sincronizzazione con gli ingressi e le uscite avviene secondo il ciclo di copia massiva:

- ▶ ogni piolo viene scandito in ogni ciclo di scansione (a meno di istruzioni di salto)
- ▶ le uscite associate alle bobine normali vengono scritte *ad ogni ciclo*, e il loro valore permane fino alla prossima esecuzione (ciclo successivo) della stessa istruzione
- ▶ il valore delle variabili lette in ingresso rimane costante per tutto il ciclo

I due programmi raffigurati a lato sono diversi.

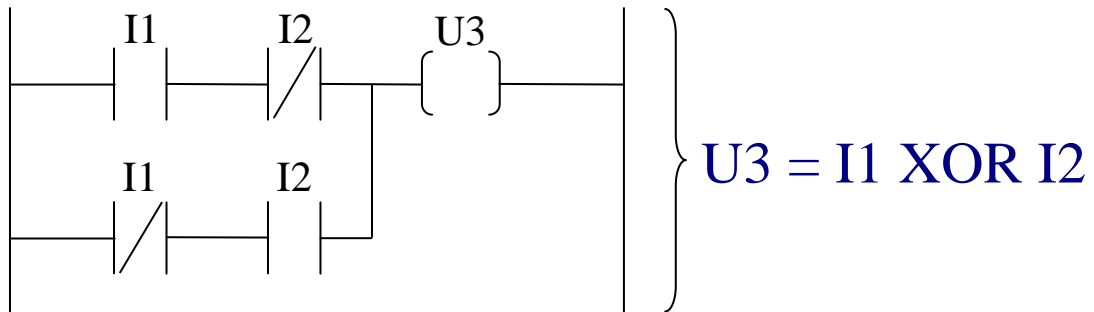
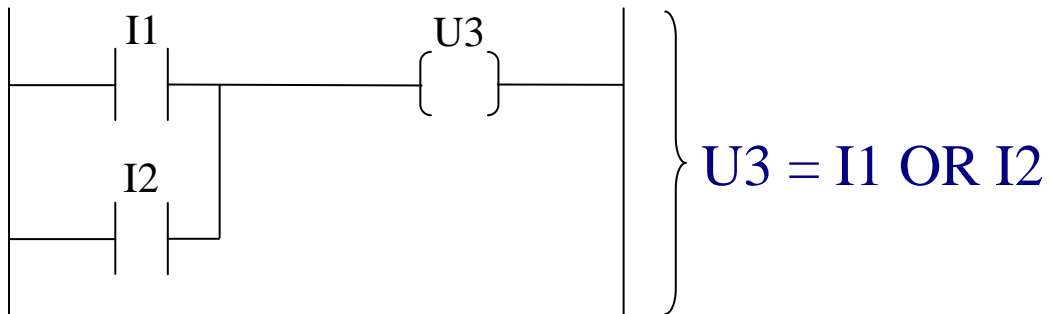
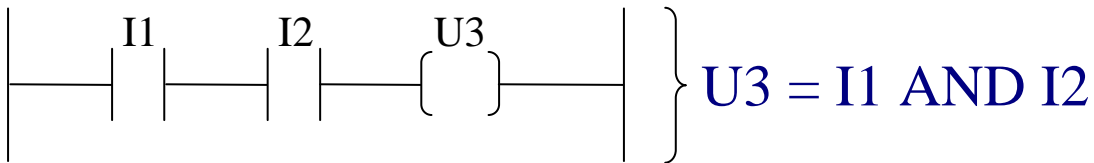
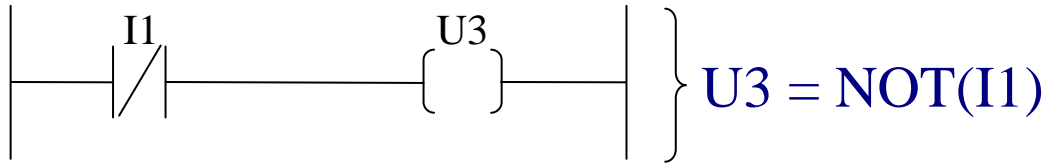
Invertendo l'ordine dei pioli, cambia il risultato dell'esecuzione delle istruzioni.

L'ordine dei rung è rilevante!



LD è un linguaggio che descrive il ciclo operativo del PLC: in un programma LD è scritta la sequenza delle cose che il PLC deve fare ad ogni ciclo.

Funzioni logiche base (statiche)



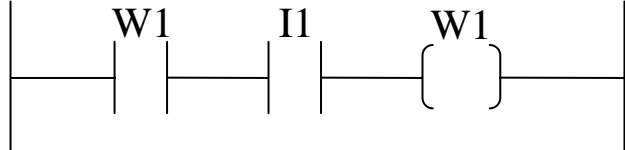
Elementi dinamici

Le istruzioni di base del LD sono “statiche”:

- ▶ assegnano il valore di un’uscita in base ad una pura combinazione logica delle variabili di ingresso associate ai contatti

E’ possibile però definire anche delle *variabili dinamiche* (variabili di stato):

- ▶ sono associate sia a contatti che a bobine

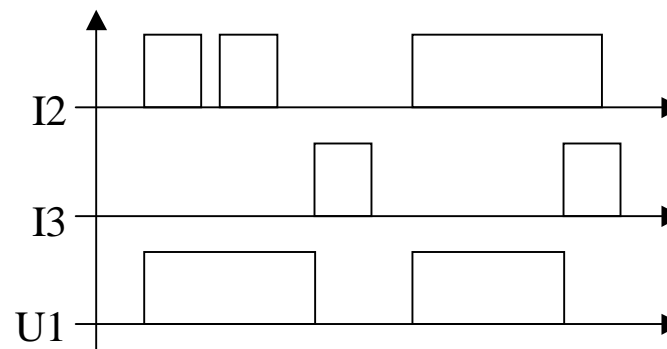
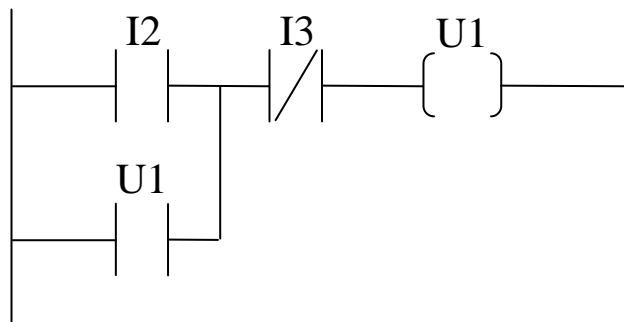
- ▶ esempio:  $W1(k) := I1(k) \text{ AND } W1(k-1)$

Esempi di elementi dinamici:

- ▶ circuito bistabile o flip-flop a reset vincente
- ▶ flip-flop a set vincente
- ▶ riconoscitore di fronte di salita/discesa
- ▶ flip-flop di tipo D

Circuito bistabile o flip-flop a reset vincente

Obiettivo: l'ingresso I2 (set) deve attivare l'uscita, e I3 (reset) disattivarla; qualora entrambi gli ingressi I2 e I3 siano attivi, l'uscita deve essere resettata a 0 (“vince” I3).

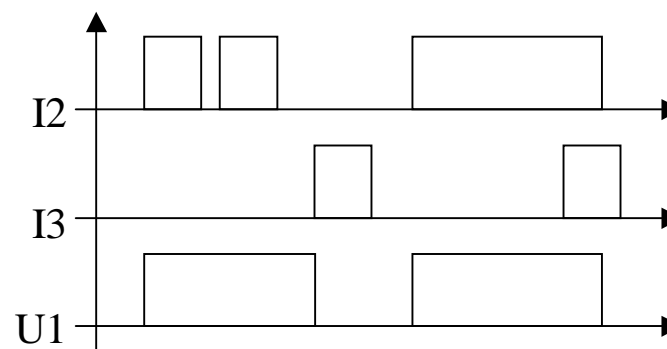
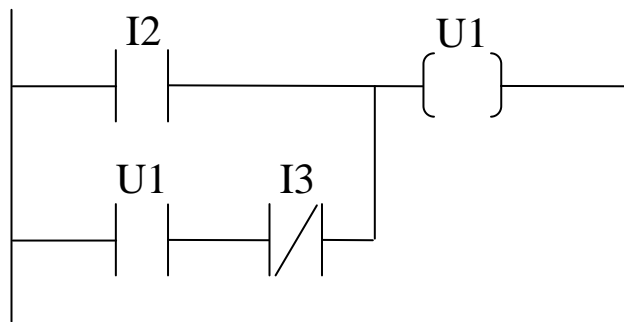


Funzionamento dinamico:

- ❶ Inizialmente tutte le variabili sono al valore 0.
- ❷ Quando sono verificate sia $I2 = 1$ che $I3 = 0$, U1 commuta a 1.
- ❸ U1 rimane a 1 finché I3 è nullo (U1 si auto-alimenta, grazie al contatto U1).
- ❹ Un impulso di durata finita su I3 pone U1 a 0.

Flip-flop a set vincente

Obiettivo: l'ingresso I2 (set) deve attivare l'uscita, e I3 (reset) disattivarla; qualora entrambi gli ingressi I2 e I3 siano attivi, l'uscita deve essere settata a 1 (“vince” I2).

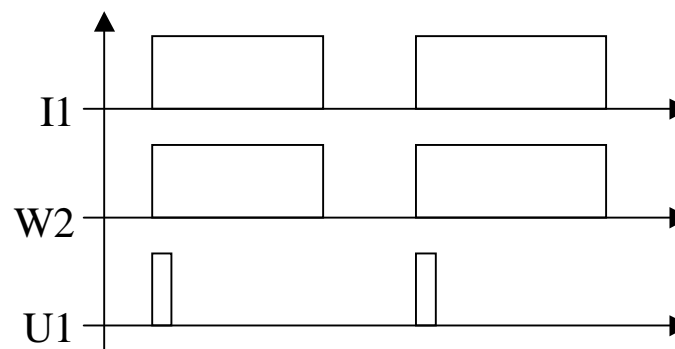
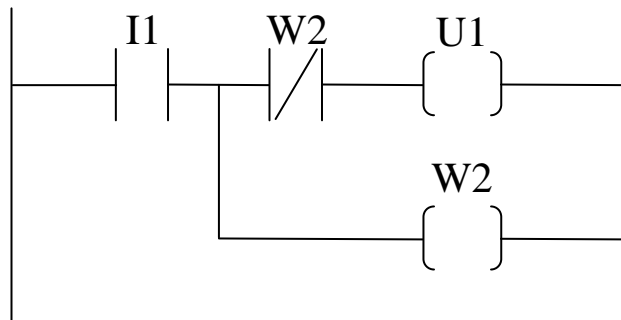


Funzionamento dinamico:

- ❶ Inizialmente tutte le variabili sono al valore 0.
- ❷ Il piolo assegna il valore 1 a U1 se $I2 = 1$.
- ❸ Finché $I3 = 0$, U1 rimane a 1 (si autoalimenta).
- ❹ U1 viene resettata se sia $I3 = 1$ che $I2 = 0$.

Riconoscitore di un fronte di salita

Obiettivo: generare un impulso della durata di un tempo di ciclo in corrispondenza di un fronte di salita su I1.



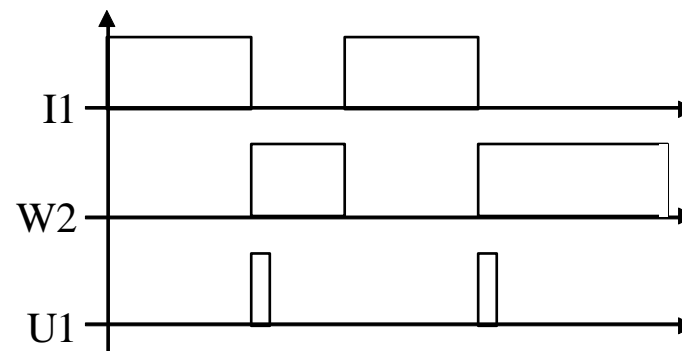
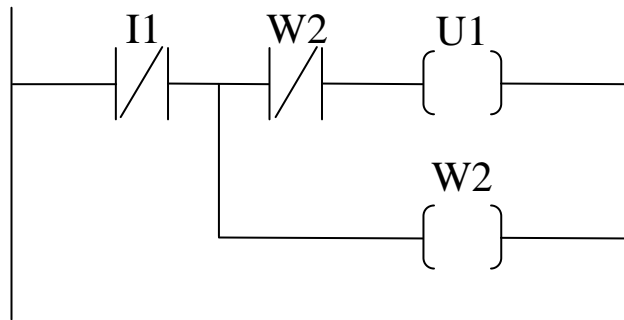
Funzionamento dinamico:

- ❶ Inizialmente tutte le variabili sono al valore 0.
- ❷ Quando si ha un fronte di salita su I1, viene alimentata la bobina U1. Nello stesso ciclo viene alimentata anche W2.
- ❸ Al ciclo successivo, essendo $W2 = 1$, viene interrotta l'alimentazione su U1 (indipendentemente dal valore di I1).

Usando la variabile ausiliaria W2 siamo riusciti a creare un impulso di durata pari ad un tempo di ciclo tutte le volte che si verifica un fronte di salita su I1.

Riconoscitore di un fronte di discesa

Obiettivo: generare un impulso della durata di un tempo di ciclo in corrispondenza di un fronte di discesa su I1.

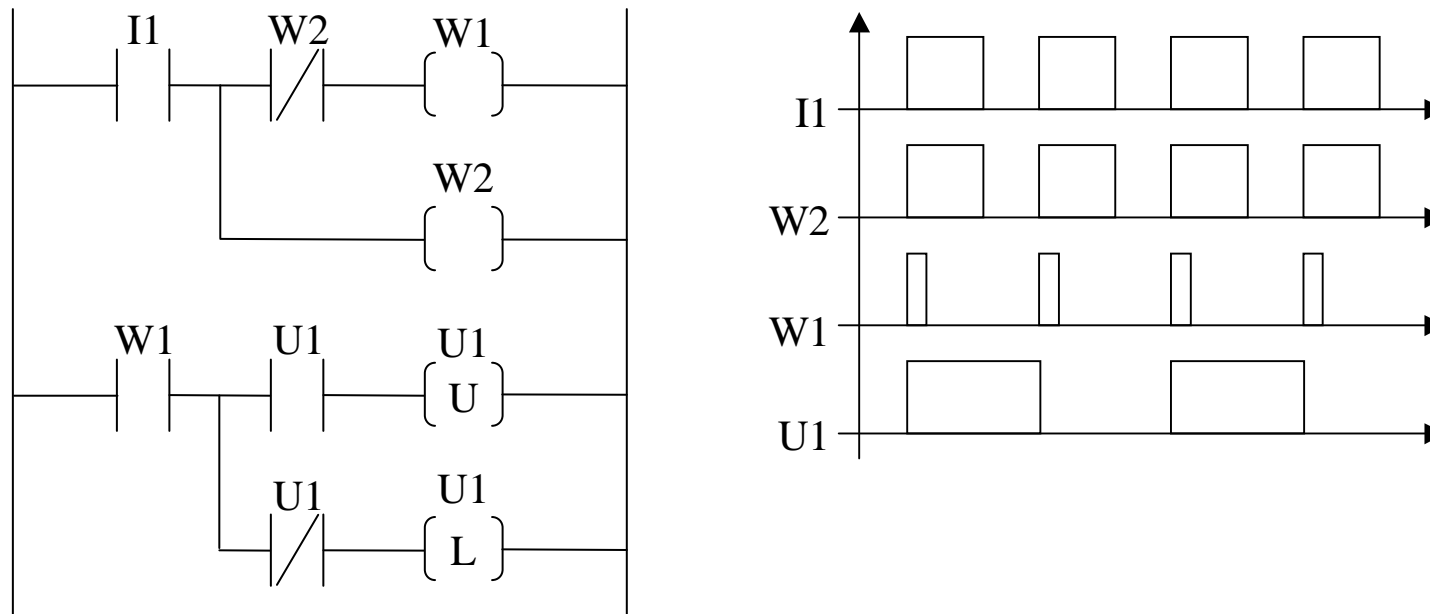


Funzionamento dinamico:

- ① Si supponga che I1 sia inizializzato a 1 (le altre variabili sono inizialmente a 0).
- ② Quando si ha un fronte di discesa su I1, viene alimentata la bobina U1. Nello stesso ciclo viene alimentata anche W2.
- ③ Al ciclo successivo, essendo $W2 = 1$, viene interrotta l'alimentazione su U1.
- ④ Quando I1 torna a 1, la bobina W2 non è più alimentata e si ripristinano le condizioni iniziali.

Flip-flop di tipo D

Obiettivo: far commutare l'uscita su ogni impulso in ingresso.

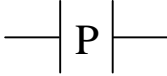

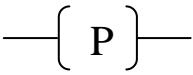
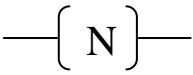


Si può sfruttare il riconoscitore di un fronte di salita appena visto (v. parte superiore del circuito) per settare la variabile interna **W1** a 1 in corrispondenza di un fronte di salita sull'ingresso **I1** (**W1** rimane a 1 per un tempo di ciclo).

Il secondo rung commuta **U1** quando **W1** = 1 (se **U1** vale 0 è attiva la bobina di tipo latch, mentre se **U1** vale 1 è attiva la bobina di tipo unlatch).

Riconoscitori di fronti: istruzioni complesse

Alcuni ambienti CAD di programmazione di PLC prevedono direttamente l'uso di istruzioni complesse di riconoscimento di fronti, per semplificare la programmazione:

contatto di tipo P (positive edge)		Il contatto si chiude sul fronte di salita della variabile booleana associata (ovvero se la variabile passa, tra due successive esecuzioni della stessa istruzione, da falso a vero).
contatto di tipo N (negative edge)		Il contatto si chiude sul fronte di discesa della variabile associata.
bobina di tipo P		La bobina assegna il valore 1 alla variabile booleana associata solo quando la sua alimentazione, tra due esecuzioni successive, passa da assente a presente.
bobina di tipo N		La bobina assegna il valore 1 alla variabile booleana associata solo quando la sua alimentazione passa da presente ad assente.

Istruzioni per il controllo del programma

Servono delle istruzioni particolari per modificare il normale ordine di esecuzione di un programma.

- ▶ salto a un'istruzione specifica
- ▶ Master Control Relay
- ▶ Zone Control Last State
- ▶ salto a sottoprogramma

Istruzione di salto

—[JMP]— —|LBL|—

L'istruzione di salto permette, se alimentata, di saltare a un rung dove è presente l'etichetta corrispondente.

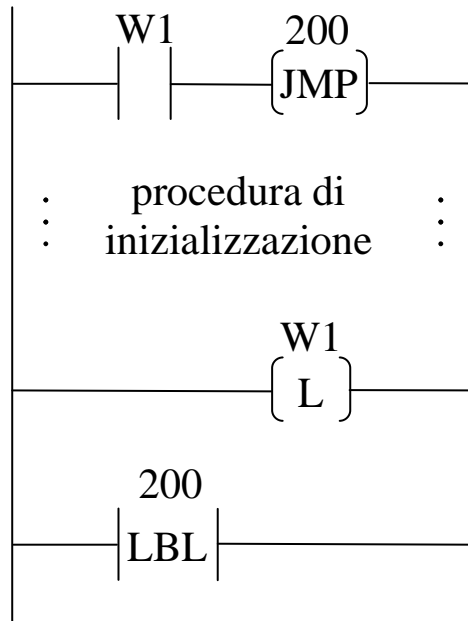
Grazie all'istruzione di salto è possibile implementare dei costrutti di programmazione del tipo IF-THEN-ELSE e cicli WHILE e FOR.

E' inoltre possibile implementare istruzioni di inizializzazione (v. esempio successivo).

Attenzione:

- ▶ se si saltano parti di programma contenenti temporizzatori o contatori si può pregiudicarne il corretto incremento del valore
- ▶ se si dimentica di inserire l'etichetta, l'esecuzione del programma si può bloccare
- ▶ saltare dentro una zona controllata da un Master Control Relay può determinare dei comportamenti non previsti

Esempio: inizializzazione



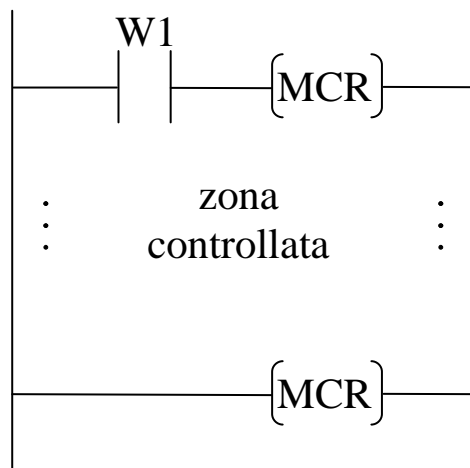
La procedura racchiusa tra il rung con il JMP e quello con LBL viene eseguita solo la prima volta.

Dal secondo ciclo di scansione in avanti W1 risulta vera e quindi l'esecuzione del programma salta al rung con LBL.

Master Control Relay

—[MCR]—

L’istruzione MCR permette di controllare, attraverso un solo insieme di condizioni, l’esecuzione di un’intera zona di programma.



L’istruzione MCR va posta come uscita *condizionata* di un rung all’inizio della zona da controllare e come uscita *incondizionata* alla fine.

Se il rung di abilitazione dell’MCR ha continuità logica, le istruzioni nella zona vengono eseguite, altrimenti non vengono eseguite e tutte le bobine non a ritenuta vengono azzerate.

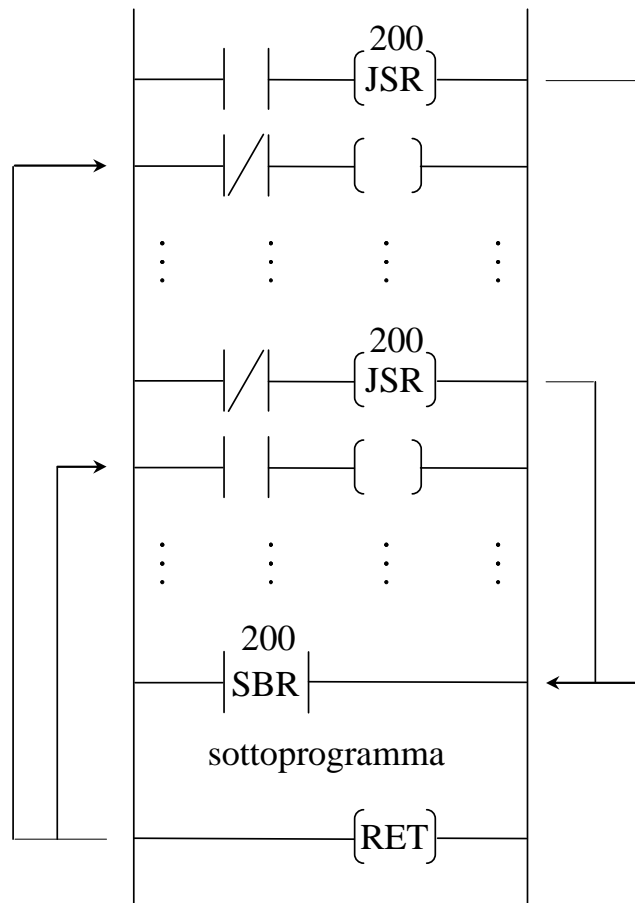
Zone Control Last State

—[ZCL]—

L'istruzione ZCL è simile all'MCR, ma se il rung di abilitazione non è alimentato, le uscite della zona controllata (le cui istruzioni non vengono eseguite) vengono mantenute al loro ultimo stato.

Salto a sottoprogramma

—[JSR]— —| SBR |— —[RET]—



L’istruzione JSR permette di saltare ad un sottoprogramma, delimitato da un rung con l’istruzione di inizio sottoprogramma (SBR, contatto) e un rung costituito dall’uscita incondizionata RET.

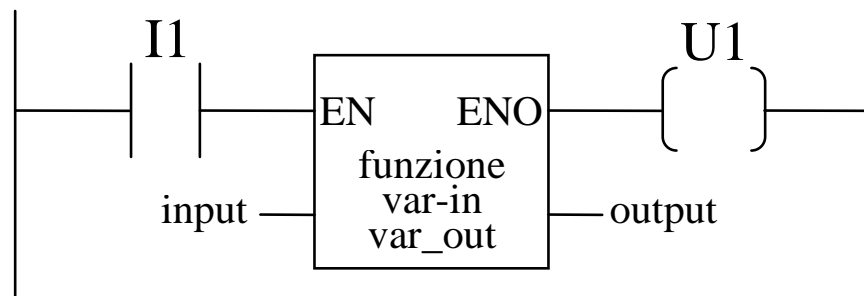
Quando il rung con JSR ha continuità elettrica, l’esecuzione del programma prosegue dal rung contenente SBR (con la stessa etichetta di JSR).

L’esecuzione del rung con RET è poi seguita da quella del rung immediatamente successivo a quello con JSR.

La porzione di codice tra SBR e RET non viene eseguita durante la normale esecuzione, ma solo in seguito ad un salto a sottoprogramma.

Istruzioni con function block

Al fine di rendere più flessibile il linguaggio LD, la normativa permette di estendere l'insieme di istruzioni di base usando funzioni e function block, realizzando istruzioni complesse come quelle disponibili nei linguaggi di programmazione ad alto livello.



- ▶ la funzione viene eseguita quando l'ingresso di abilitazione EN è percorso da corrente
- ▶ nell'esecuzione può fare uso di variabili di ingresso booleane (input) e non (var-in)
- ▶ in uscita fornisce un segnale booleano (ENO) che indica l'avvenuta esecuzione, ed eventuali altri segnali booleani (output) e non (var-out) che rappresentano il risultato
- ▶ a volte la bobina di uscita è omessa nella codifica e si fa riferimento direttamente alla variabile ENO

Esempi di istruzioni complesse realizzate con function block:

- ▶ istruzioni di temporizzazione
- ▶ istruzioni di conteggio
- ▶ istruzioni di trasferimento di memoria
- ▶ operazioni aritmetico/logiche
- ▶ istruzioni di comparazione
- ▶ registro a scorrimento a destra
- ▶ regolatore PID
- ▶ istruzioni di comunicazione via rete

Istruzioni di temporizzazione e conteggio

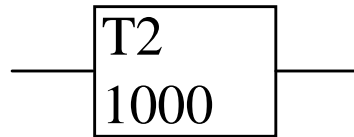
Temporizzatori e istruzioni correlate:

- ▶ temporizzatore
- ▶ temporizzatore a ritenuta
- ▶ ritardo di spegnimento
- ▶ oscillatore ad onda quadra

Istruzioni di conteggio:

- ▶ contatore ad incremento

Temporizzatore



Parametri:

- ▶ indirizzo (Tx)
- ▶ intervallo di conteggio (p.es.: centesimi di secondo, valore massimo 360000, pari ad un'ora)

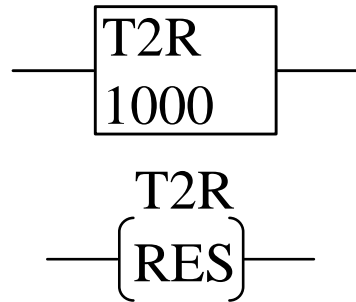
Se il piolo che contiene il blocco Tx consente il flusso di corrente, allora il temporizzatore conta il trascorrere del tempo fino a raggiungere il valore preimpostato.

A questo punto la variabile Tx diventa vera e lo rimane fino al reset del temporizzatore, che si ha quando cessa la continuità elettrica.

Lo stato del temporizzatore è accessibile all'indirizzo Tx (falso durante il conteggio, vero alla fine del conteggio (fino al reset)).

Il tempo contato dal temporizzatore fino all'istante corrente è accessibile all'indirizzo Tx.acc.

Temporizzatore a ritenuta



Parametri:

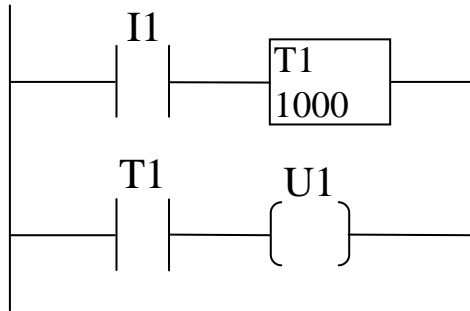
- ▶ indirizzo (TxR)
- ▶ intervallo di conteggio (p.es.: centesimi di secondo)

Funziona come il temporizzatore standard, ma il valore corrente del conteggio viene conservato se cessa la continuità elettrica e il conteggio riprende appena passa corrente.

Per fermare e riavviare il temporizzatore occorre utilizzare il comando apposito di reset.

NB. Le istruzioni come i temporizzatori e i contatori sono implementate in maniera diversa a seconda del costruttore.

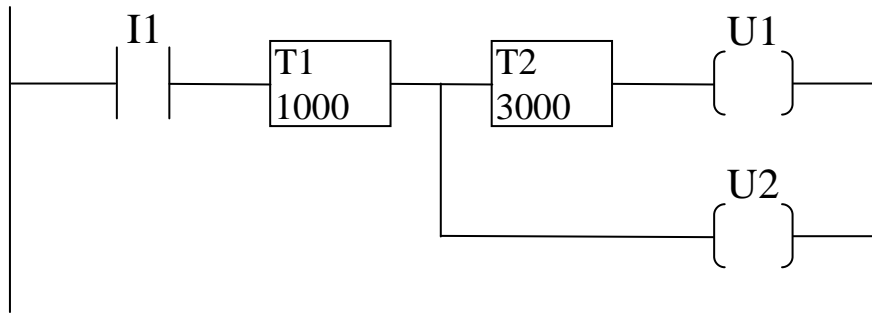
Esempio



Quando la variabile di ingresso I1 sale a 1 il temporizzatore incomincia a contare. Quando il temporizzatore raggiunge il valore preimpostato pari a 1000, T1 = 1 e si attiva l'uscita U1.

Esempio

I temporizzatori possono essere collegati in cascata.



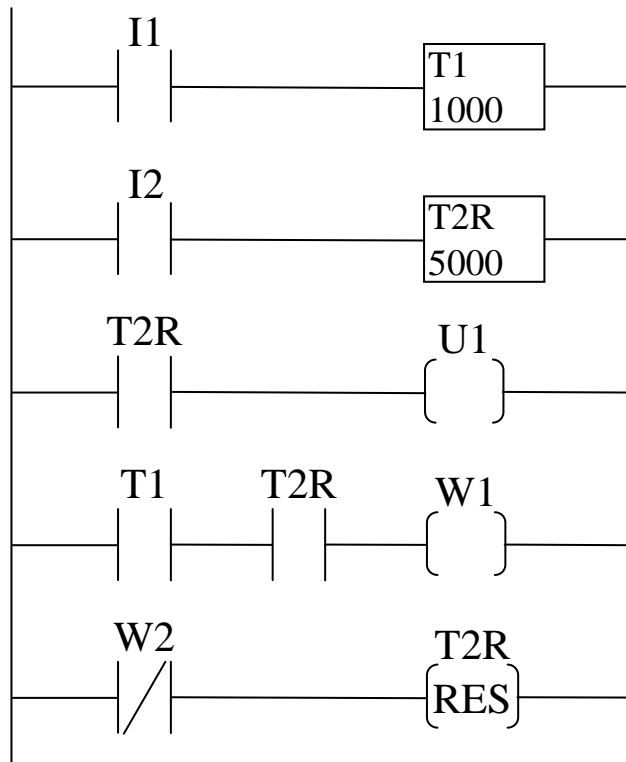
Quando la variabile di ingresso I1 sale a 1 il temporizzatore T1 incomincia a contare.

Quando T1.acc arriva a 1000, T1 diventa vero (e lo rimane finché I1 = 1 e assicura la continuità elettrica).

Contestualmente, si attiva l'uscita U2 e il temporizzatore T2 inizia a contare.

Dopo altre 3000 unità di tempo, si attiva anche U1.

Esempio



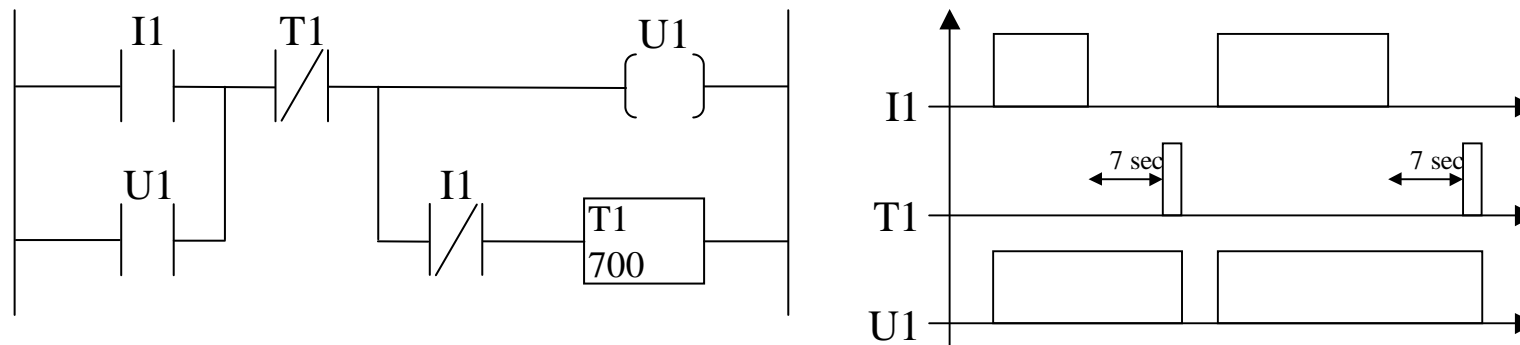
U1 memorizza l'avvenuto conteggio di T2R.

W1 si attiva quando entrambi i contatori hanno completato il conteggio.

Infine, il temporizzatore T2R si resetta quando W2 va a 0.

Ritardo di spegnimento

Obiettivo: l'uscita deve replicare l'ingresso, prolungandone la durata a 7 secondi.



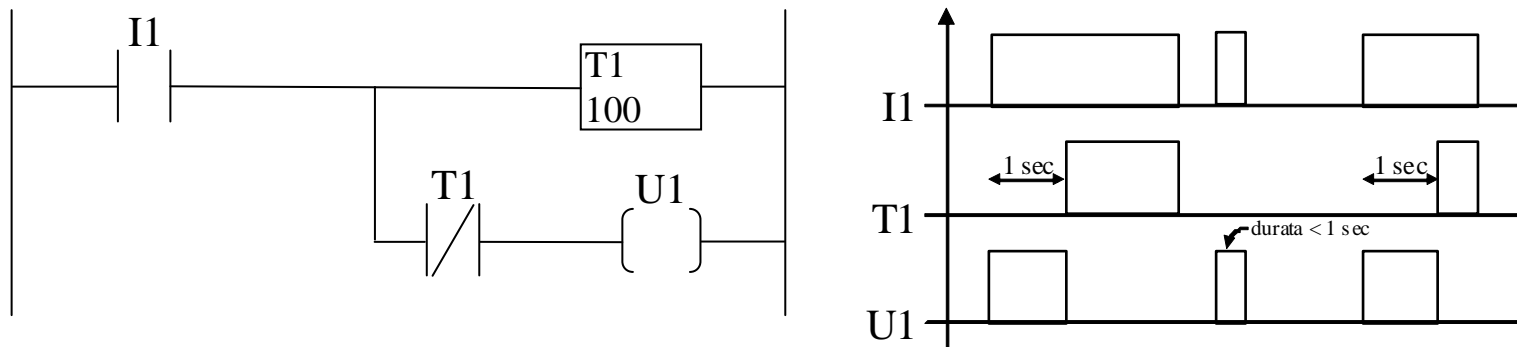
L'uscita **U1** ha lo stesso fronte di salita dell'ingresso **I1**, mentre il suo fronte di discesa viene ritardato rispetto all'ingresso di una quantità preimpostata.

Il contatto **U1** serve ad autoalimentare l'uscita **U1** finché il conteggio non sia terminato (**U1** si riazzera quando **T1** = 1).

Il temporizzatore inizia a contare quando **I1** va a 0.

Impulso all'accensione

Obiettivo: ogni volta che l'ingresso si attiva, l'uscita emette un impulso, al più di durata pari ad 1 secondo.



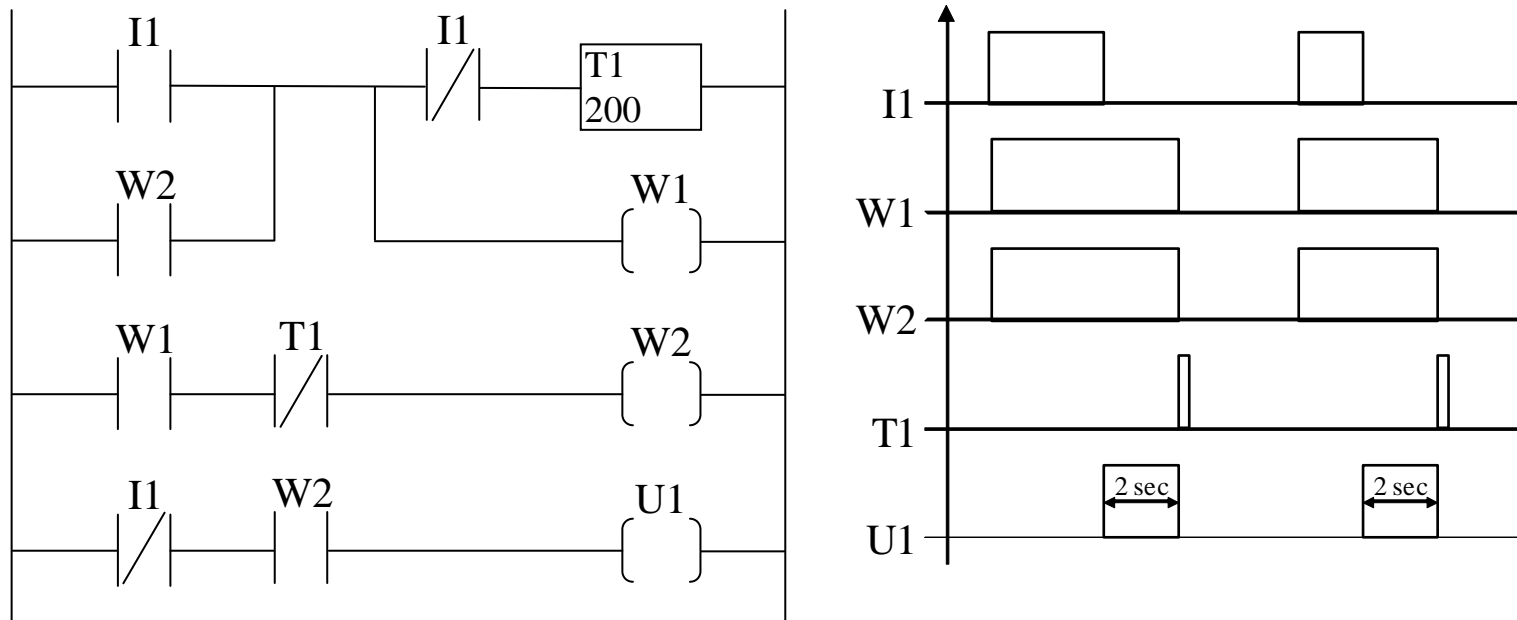
L'uscita U1 ha lo stesso fronte di salita dell'ingresso I1 (inizialmente $T1 = 0$).

Si riazzera se il temporizzatore raggiunge la sua soglia preimpostata (1 sec) e quindi $T1 = 1$, oppure prima se I1 si disattiva.

Il temporizzatore si resetta sul fronte di discesa di I1.

Impulso allo spegnimento

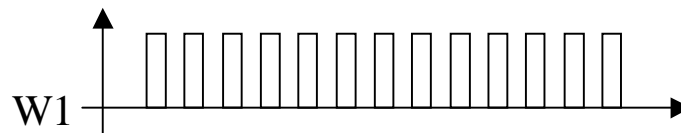
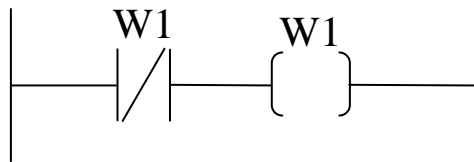
Obiettivo: ogni volta che l'ingresso si disattiva, l'uscita emette un impulso, di durata pari a 2 secondi.



Il temporizzatore viene alimentato quando I1 si disattiva (W1 e W2 servono a mantenere la continuità logica su T1). U1 è attiva durante il conteggio. Quando termina il conteggio, T1 commuta a 1, spegnendo W2 e U1 (contestualmente il temporizzatore si resetta). Il ciclo ricomincia quando I1 si riattiva.

Oscillatore ad onda quadra con periodo pari a 2 tempi di ciclo

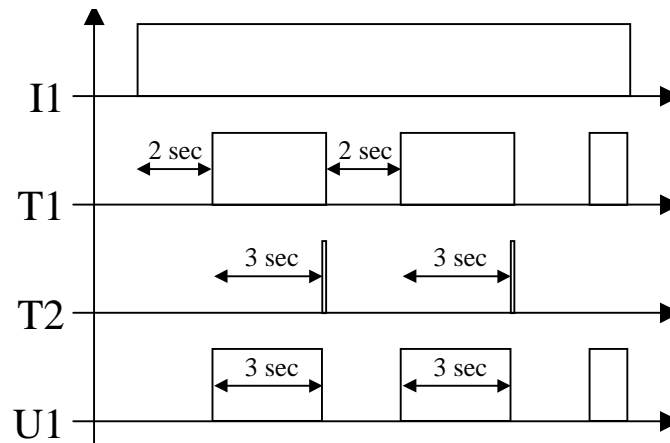
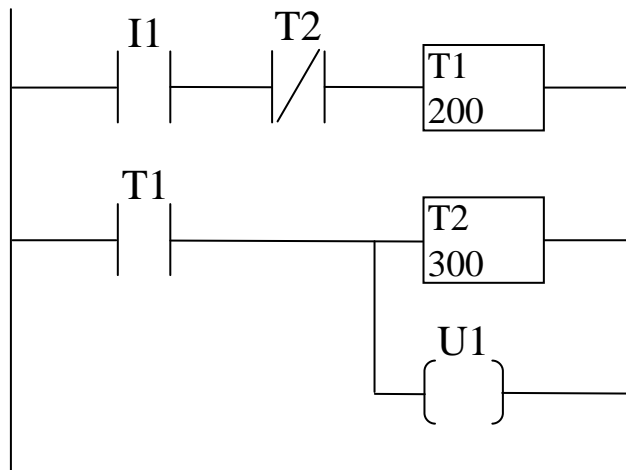
Obiettivo: commutare l'uscita ad ogni tempo di ciclo.



Per settare il periodo e il duty cycle dell'onda quadra occorre usare un insieme più complesso di istruzioni.

Oscillatore ad onda quadra con duty cycle e periodo settabili

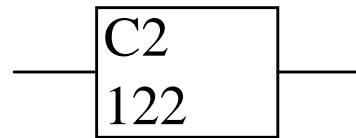
Obiettivo: generare un'onda quadra con periodo 5 sec, con 3 sec a 1 e 2 sec a 0.



Funzionamento dinamico:

- ❶ Inizialmente, $T1 = 0$ e $T2 = 0$.
- ❷ Quando I1 commuta a 1, T1 inizia a contare.
- ❸ Dopo 2 sec $T1 = 1$. T2 inizia a contare. Durante il conteggio, $T2 = 0$ e T1 non resetta.
- ❹ Dopo 3 sec il conteggio di T2 termina e $T2 = 1$.
- ❺ Il ciclo dopo T1 viene resettato a 0. Nello stesso ciclo anche T2 viene resettato a 0.
- ❻ Il ciclo dopo ancora T1 ricomincia a contare.

Contatore ad incremento



Parametri:

- ▶ indirizzo (Cx)
- ▶ valore da raggiungere nel conteggio



Se il piolo che contiene il contatore subisce una transizione da 0 (falso) a 1 (vero), il contatore si incrementa di un'unità.

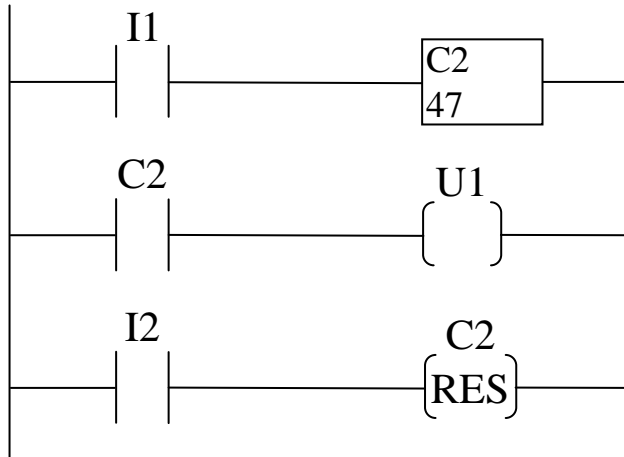
L'indirizzo Cx indica lo stato del contatore:

- ▶ falso durante il conteggio
- ▶ vero quando il conteggio raggiunge il valore preimpostato
- ▶ rimane vero fino al reset

Il valore parziale raggiunto dal contatore è accessibile all'indirizzo Cx.acc.

Per riavviare il contatore occorre utilizzare il comando apposito di reset.

Esempio

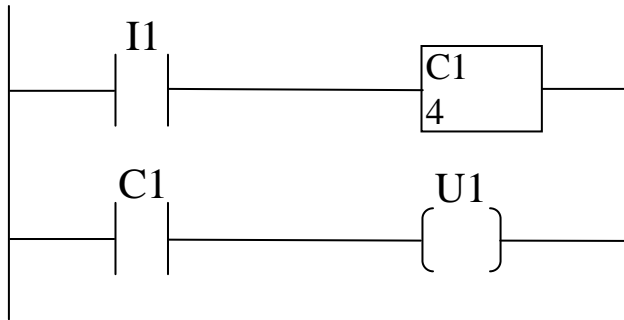


Quando I1 commuta da 0 a 1 47 volte $C2 = 1$ e si attiva l'uscita U1.

Il contatore viene riазzerato se $I2 = 1$.

Esempio: conteggio di eventi

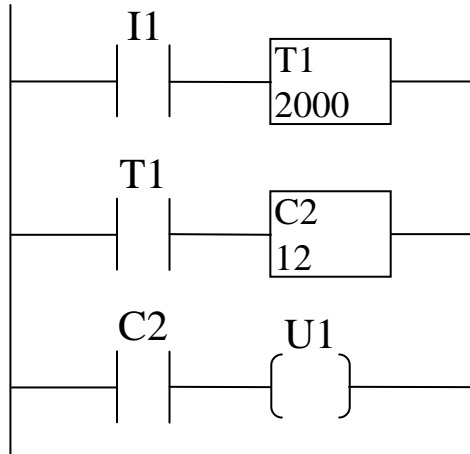
Obiettivo: contare 4 attivazioni dell'ingresso.



Il contatore si incrementa di 1 ad ogni fronte di salita di I1.

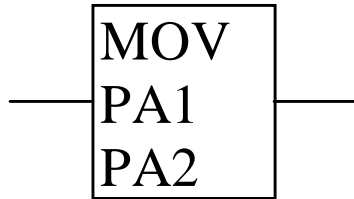
Al 4° fronte di salita di I1, $C1 = 1$ e si attiva l'uscita U1.

Esempio: uso contemporaneo di temporizzatori e contatori



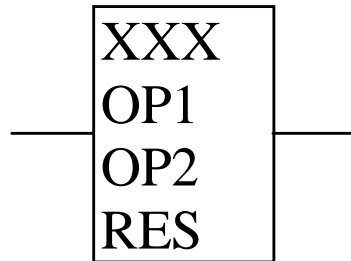
La variabile di uscita U1 vale 1 dopo che il temporizzatore T1 ha contato 2000 unità di tempo per 12 volte.

Trasferimento di memoria



Se il piolo che contiene come uscita l’istruzione MOV ha continuità elettrica, il contenuto della word di memoria all’indirizzo PA1 viene trasferito all’indirizzo PA2.

Operazioni aritmetico/logiche



Se il piolo che contiene come uscita l’istruzione XXX ha continuità elettrica, l’operazione associata ad XXX viene eseguita sui contenuti delle word di memoria agli indirizzi OP1 e OP2, e il risultato viene posto all’indirizzo RES.

XXX è il codice mnemonico associato all’operazione:

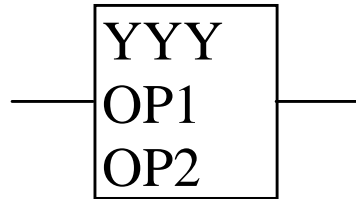
- ▶ ADD (addizione)
- ▶ MUL (moltiplicazione)
- ▶ SUB (sottrazione)
- ▶ DIV (divisione)
- ▶ AND (moltiplicazione binaria bit a bit)
- ▶ OR (addizione binaria bit a bit)

Alcuni PLC consentono di trattare solo numeri interi, altri anche i reali.

Il tipo di una variabile si decide quando la si dichiara, e le successive istruzioni che la coinvolgono si comportano di conseguenza (ad esempio eseguendo una divisione intera).

Su questo aspetto però ci sono differenze tra i prodotti ed è bene prestarvi attenzione, dichiarare sempre esplicitamente i tipi e non mescolarli.

Istruzioni di comparazione



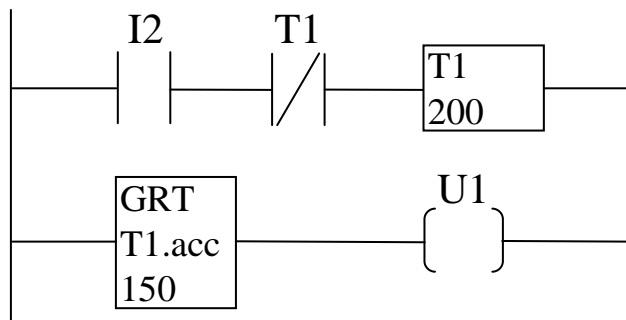
Il contatto associato all'istruzione YYY assicura continuità elettrica se è verificata la condizione associata all'operatore YYY, applicata ai contenuti delle word di memoria agli indirizzi OP1 e OP2.

YYY è il codice mnemonico associato all'istruzione di comparazione:

- ▶ EQU (=)
- ▶ NEQ (≠)
- ▶ GEQ (\geq)
- ▶ LEQ (\leq)
- ▶ GRT ($>$)
- ▶ LES ($<$)

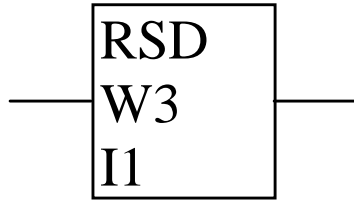
Esempio: oscillatore

Obiettivo: generare un'uscita ad onda quadra di periodo 2 sec e con un duty cycle del 25%.



Si usa il temporizzatore T1 per iniziare il conteggio e si setta la variabile U1 a 1 quando il valore raggiunto dal conteggio (T1.acc) è superiore a 150.

Registro a scorrimento a destra

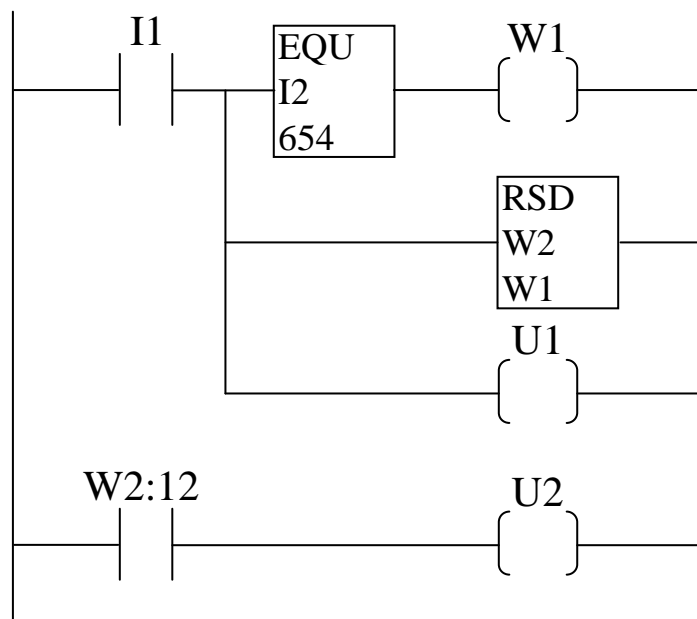


Se il piolo che contiene come uscita l’istruzione RSD ha continuità elettrica, il contenuto della word di memoria all’indirizzo W3 viene fatto scorrere verso destra di un bit (shift) e in prima posizione viene inserito il bit I1.

L’istruzione RSD serve per modellizzare il movimento di parti su linee di lavorazione, o per descrivere lo stato di avanzamento di una sequenza di operazioni.

Esempio: nastro trasportatore dotato di lettore di codice a barre

Obiettivo: un nastro trasportatore viene alimentato con pezzi diversi; un lettore di codice a barre rileva il codice del pezzo; se il codice rilevato è 654, il pezzo deve essere instradato su un altro nastro mediante un deviatore, che si trova a 4 passi di spostamento elementare dal lettore di codice a barre.



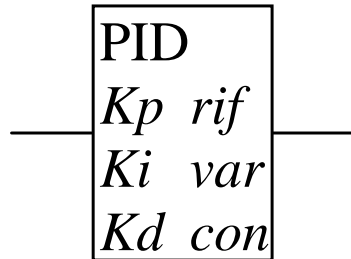
Ipotesi:

- ▶ il nastro si muove di un passo quando $U1 = 1$
- ▶ I1 segnala il completamento dello spostamento
- ▶ il lettore trasferisce nella word I2 il codice rilevato
- ▶ U2 attiva il deviatore
- ▶ W2 rappresenta il contenuto del nastro

Il programma riportato in figura segue lo scorrere del nastro con il registro a scorrimento inserendo

il valore 1 quando viene riconosciuto il codice giusto e smista il pezzo, quando esso raggiunge la posizione giusta (corrispondente al bit 12 di W2).

Regolatore PID

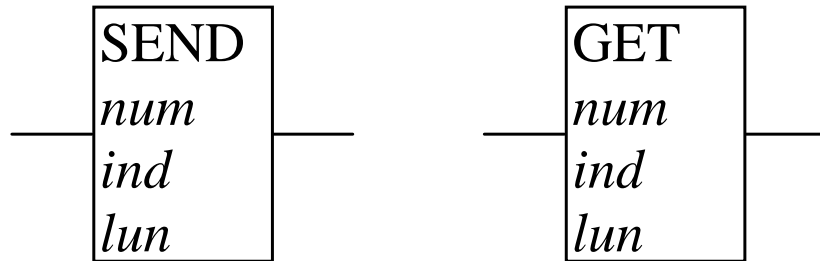


L'istruzione realizza una funzione di controllore PID (a tempo discreto).

Parametri:

- ▶ *Kp*, *Ki* e *Kd* sono i guadagni del controllore
- ▶ *rif* è l'indirizzo della word in area utente dove viene scritto il riferimento
- ▶ *var* è l'indirizzo della word in area ingressi dove si trova l'immagine della variabile da controllare
- ▶ *con* è l'indirizzo della word in area uscite dove andrà scritto il valore calcolato del controllo

Istruzioni di comunicazione via rete



Le istruzioni SEND e GET consentono di inviare e ricevere dati tra PLC.

L'istruzione SEND invia un blocco di word ad un altro PLC connesso in rete.

Simmetricamente, l'istruzione GET riceve un blocco di word da un altro PLC connesso in rete.

Parametri:

- ▶ *num* è l'identificativo del PLC
- ▶ *ind* è l'indirizzo di partenza del blocco da spedire
- ▶ *lun* è la lunghezza del blocco

La sintassi di tali istruzioni è una delle parti meno standardizzate di LD, anche perché fa uso della rete ed è proprio nei relativi protocolli che si concentrano molti degli aspetti proprietari dei diversi prodotti.

Non tutte le primitive di comunicazione implicano la sincronizzazione dei PLC coinvolti: si può mettersi in attesa che un PLC spedisca (e allora la sincronizzazione c'è), oppure semplicemente leggere dalla sua memoria (e allora si trova quel che al momento vi è scritto, senz'alcuna sincronizzazione).

Nei moderni sistemi di sviluppo per PLC il modello di comunicazione di gran lunga più diffuso è quello *a memoria condivisa* (*shared memory*):

- ▶ lo spazio d'indirizzamento di tutti i PLC in rete è unico, come se più CPU condividessero la stessa memoria
- ▶ questo consente, nell'ipotesi che il meccanismo fisico con cui lo si implementa sia perfetto, di trattare i problemi di comunicazione come accesso a memoria condivisa
- ▶ la dichiarazione delle variabili avviene a livello di sistema e ognuna di esse, nel suo indirizzo, ha anche l'indirizzo di rete del PLC cui logicamente appartiene

Dialetti del LD

Il linguaggio LD è stato creato assai prima dell'introduzione della normativa IEC 61131-3.

Come facilmente s'immagina, quindi, di LD esistono vari “dialetti”, che differiscono per la morfologia e per qualche elemento sintattico ma non per la semantica.

D'altra parte, essendo l'introduzione di uno standard il risultato di un negoziato, ed essendo l'oggetto normato qualcosa su cui molte aziende già fondavano un grande business, la normativa IEC 61131-3 fu concepita in modo da fissare solo i principi e non i dettagli.

Per gli elementi per i quali esistono differenze tra i vari dialetti di LD (p.es. i contatori), si adotta nel corso una notazione convenzionale che non è quella di nessun produttore specifico, ma che si trasforma in modo assolutamente ovvio in una qualsiasi di tali notazioni.

Uso del LD

Il LD è un linguaggio completo e disponibile su tutti i PLC, ma poiché si tratta di un linguaggio a basso livello, il codice che si produce è poco leggibile e, di conseguenza, difficile da verificare, documentare e mantenere.

In generale, non è quindi uno strumento adatto in fase di progettazione, per la quale si prestano di più linguaggi ad alto livello come le reti di Petri o il Sequential Functional Chart (SFC), linguaggio grafico derivato dalle reti di Petri. Inoltre, LD è un linguaggio procedurale il cui modello del codice ricalca il ciclo a copia massiva e consiste di fatto nel descrivere le operazioni da fare *ad ogni ciclo operativo del PLC*. Quindi:

- ▶ è innaturale esprimere in LD delle sequenze che invece hanno a che fare con il *ciclo operativo dell'impianto*
- ▶ risulta invece naturale realizzare quelle parti del sistema di controllo che si esprimono come *vincoli* (che vanno verificati ad ogni ciclo del PLC per dare o meno gli opportuni consensi alle operazioni che ne dipendono)

Useremo quindi LD essenzialmente per la supervisione e SFC per il controllo.