PLC e SCADA, Sect.0

Alessandra Flammini
alessandra.flammini@unibs.it
Stefano Rinaldi
stefano.rinaldi@unibs.it
Ufficio 24 Dip. Ingegneria dell'Informazione
030-3715627 Martedì 16:30-18:30

Prerequisiti da corso Sistemi per l'Industria e PLC

(chi ha già frequentato il corso della triennale "Sistemi per l'Industria e PLC" usi queste slide solo come ripasso. Il materiale del corso è disponibile sul mio sito web http://alessandra-flammini.unibs.it/ -> SDPLC e SIPLC -> Sistemi per l'Industria e PLC)

Ambiente industriale

Automazione di fabbrica (oggetto del corso)

- **La fabbrica produce semilavorati o prodotti finiti partendo da materiali grezzi o semilavorati acquistati che vengono trasformati (cella di lavorazione basata su macchine utensili) e assemblati (cella di lavorazione basata su celle manuali/robotizzate di montaggio)**
- **Materiali grezzi (o riuso) -> semilavorato -> prodotto finito**
- **La fabbrica può comprendere più lavorazioni (aree) suddivise su più macchine (celle). Ad esempio produzione di lamiera da billette: area preriscaldo e sbozzatori, area laminazione (composta da più celle di laminazione), area taglio, area finitura, piegatura e confezionamento. Punti chiave: costi e tempi di produzione**
- **Tipici impianti ad automazione di fabbrica: automotive, plastica**

Controllo di processo (es. Oil&Gas, rete elettrica, rete idrica, carta)

- **L'impianto, in genere molto esteso e non sorvegliato da operatori, distribuisce/trasforma risorse (lavorazione continua, affidabilità)**

La cella di lavorazione

"una cella di lavorazione trasforma energia elettrica in energia meccanica e attua lavoro, tipicamente mediante un motore elettrico (o un sistema idraulico) sulla base di alcuni riferimenti (es. dimensioni) e dello scostamento della lavorazione dai riferimenti misurato mediante sensori"

Una cella di lavorazione prevede lavorazione e movimentazione

- **Lavorazione, fase "utile" ai fini della produzione (poca automazione)**
- **Movimentazione, fase "inutile", minimizzare tempi e costi (tanta automazione)**

Cella ={sensori, attuatori, rete elettrica, controllori, SCADA}

- **Rete elettrica: sistema trifase di correnti e tensioni in Media Tensione (10kV, 15kV, 20kV) o in Bassa Tensione (380V)**
- **Trasformatore di energia: motori (elettrici) e pompe (idraulici)**
- **Sensore: elemento che converte grandezze fisiche (dimensioni, posizioni, ecc.) in grandezze elettriche a bassa potenza permettendone la misura**
- **Attuatore: convertitore inverso al sensore. Oltre a lampade, forni, magneti, interruttori, ci sono gli azionamenti che adattano la rete elettrica alle esigenze del motore (che deve andare a una certa velocità e potenza per svolgere la lavorazione –potenza- o la movimentazione –velocità-)**

Controllori = PLC: tipi di PLC

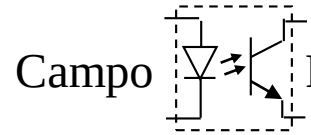
- **Compatti (in basso)**
  - **Monoblocco con integrato alimentatore, CPU e ingressi e uscite**
- **Espandibili**
  - **Il PLC è composta da un blocco Alimentatore, un Blocco CPU e vari blocchi dedicati a ingressi e uscite digitali e analogiche Monoblocco con integrato alimentatore, CPU e ingressi e uscite**
- **Modulari (in alto)**
  - **Come espandibili ma con possibilità di Multi-CPU**

- **Soft-PLC (non raffigurati)**
  - **Schede PC con sistema operativo in tempo reale e software di programmazione uguale a PLC e I/O remotato su periferia connessa con bus di campo o Real-time Ethernet (RTE)**
- **Periferia (riquadro in alto a destra)**
  - **Sistema modulare di ingressi e uscite con CPU per elaborazione locale connesso ad un controllore mediante bus di campo o RTE (sistema di comunicazione a bassa latenza e basso jitter)**

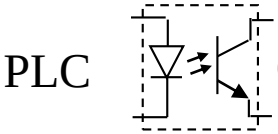
Hardware dei PLC espandibili o modulari

- **Alimentatore**
  - **E' il blocco più importante e contiene supercapacitor per evitare effetti dei buchi di rete e per dare tempo al PLC, in caso di mancanza rete, di attuare procedure di messa in sicurezza. Spesso l'alimentatore si fa carico del controllo del corretto funzionamento della CPU –watch dog- (l'alimentatore si aspetta un impulso periodico dalla CPU, altrimenti la resetta)**
- **CPU**
  - **gestisce il BUS di collegamento con i moduli di I/O e con i moduli funzionali**
  - **Ha le memorie non volatili (flash) per la memorizzazione del programma applicativo, dei dati di configurazione e delle variabili applicative impostate come "a ritenzione", che permettono una ripartenza del programma da dove il programma è stato interrotto**
  - **Ha le memoria volatili per le variabili e la copia in esecuzione del codice**
  - **Ha un software di base (del costruttore) e un software applicativo**
- **Moduli**
  - **Moduli di ingresso logico (isolatore galvanico), alto parallelismo (32, 64)**
  - **Moduli di uscita logica (isolatore galvanico -32,64- o relais -8,16-)**
  - **Moduli di I/O analogico (4, 8) basse prestazioni (10bit, kSa/s)**
  - **Moduli per sensori/attuatori specifici (encoder, termocoppie)**
  - **Moduli di comunicazione verso bus di campo**
  - **Moduli intelligenti e dotati di un proprio FW**

Isolamento in ingresso

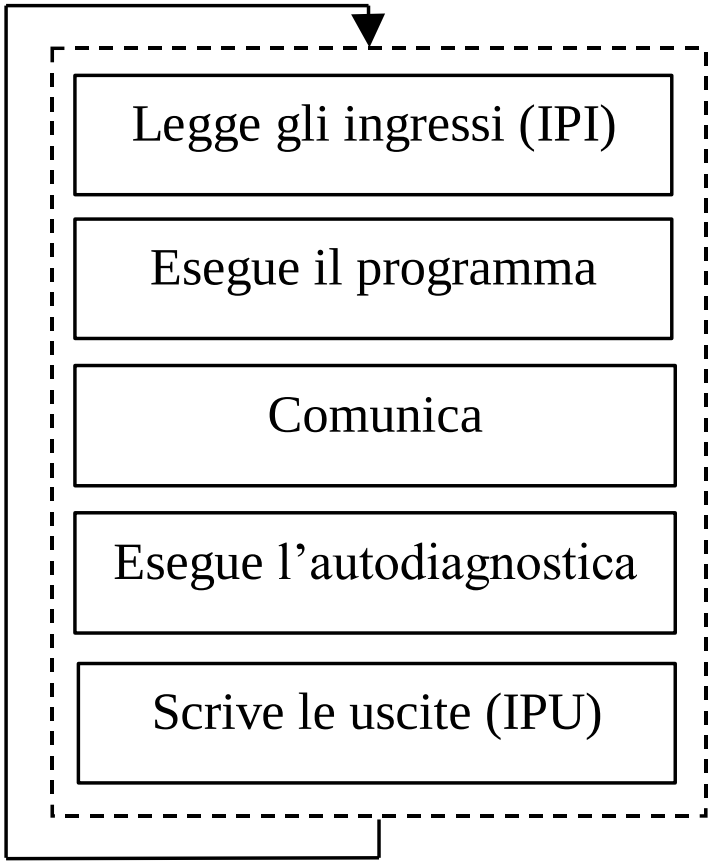


Isolamento in uscita



Software di base del PLC, il ciclo di scansione

- **Il PLC viene usato per effettuare ciclicamente una serie di istruzioni del tipo "se... allora..." e quindi il suo software di base è organizzato in un ciclo**
- **Il programma applicativo è costituito da una sequenza di istruzioni**
  - **Gli ingressi non cambiano durante il ciclo**
  - **L'ultima istruzione ha priorità sulle uscite**
  - **La sequenza dura < Tmax**
  - **No loop di attesa**
  - **Interrogazioni feed forward**
  - **Ciclo di scansione (asincrono, <1ms)**



- **Fase "Comunica"**
  - **Diagnostica via PC**
  - **Fase di background (<Tlimite)**

- **Fase "autodiagnosi"**
  - **Diagnostica locale (CPU)**
  - **Diagnostica dell'I/O**
  - **Il PLC deve poter tempestivamente identificare una situazione di guasto per poter mettere in sicurezza le uscite**

Software di base del PLC, immagini di processo

- **Immagini di processo degli ingressi (IPI):**
  - **variabili di memoria nelle quali viene memorizzato il valore degli ingressi fisici (segnali cablati ai morsetti di ingresso) all'inizio del ciclo**
  - **il programma applicativo (ciclo k) si svolge a ingressi "congelati"**
  - **è possibile accedere direttamente agli ingressi fisici senza modificare le IPI**

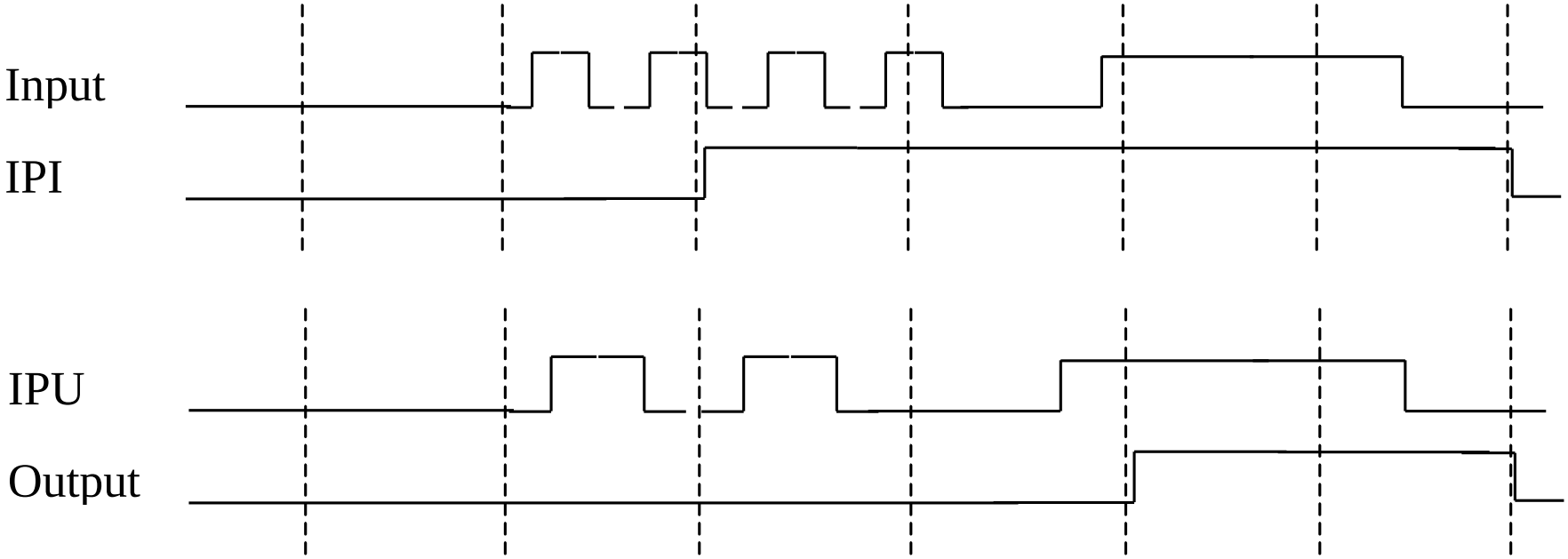
- **Immagini di processo delle uscite (IPU):**
  - **variabili di memoria nelle quali il programma applicativo scrive come se fossero le uscite logiche ma che vengono effettivamente scaricate sulle uscite fisiche alla fine del ciclo**
  - **l'ultima scrittura di un'uscita è la sola che ha effetto**
  - **sincronizzazione delle uscite fisiche**

- **Le immagini di processo non si applicano a I/O logico "veloce", I/O legato a interrupt, I/O analogico, I/O logico di alcuni PLC**

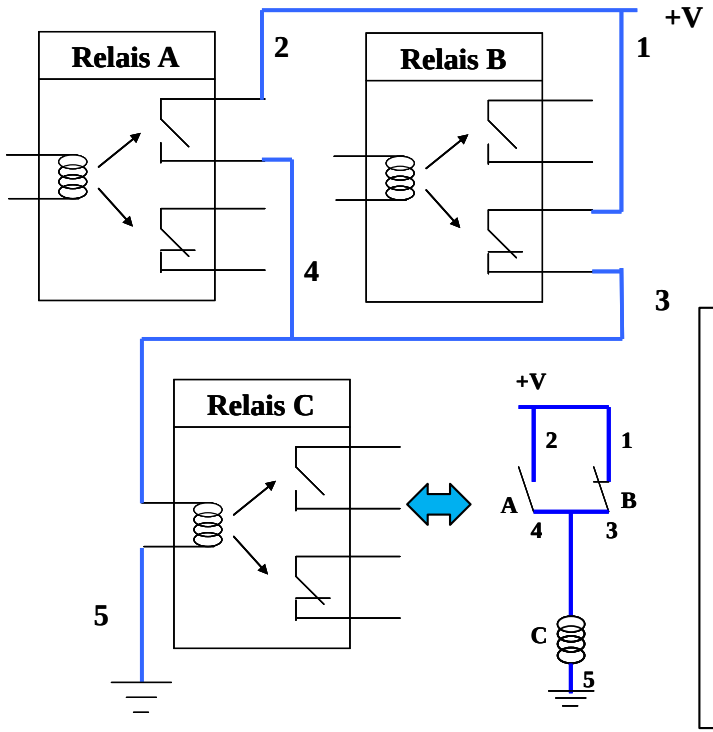
- **NOTA: come è possibile creare una semplice corrispondenza tra il morsetto al quale è cablato il segnale fisico e la corrispondente IPI/IPU?**
  - **IPI/IPU sono indirizzi di memoria assoluti e non rilocabili (indirizzamento geografico/posizionale, modulo per modulo, morsetto per morsetto)**

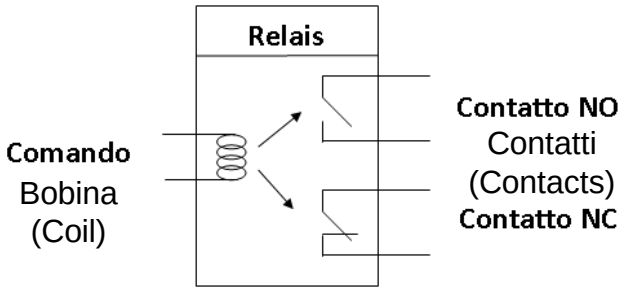
Software di base del PLC, immagini di processo

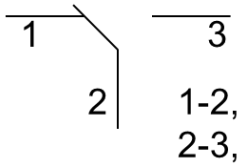
- **Immagini di processo:**
  - **Alcune variazioni degli ingressi possono andare perse (dipende da T_ciclo)**
  - **È consigliabile accedere alle uscite in un unico punto del programma**
  - **Trattandosi di un sistema campionato (IPI) è necessario un filtro HW anti-aliasing (previsto dall'HW del PLC e impostabile sulla base del tempo di ciclo massimo, che dipende dalla complessità del programma)**



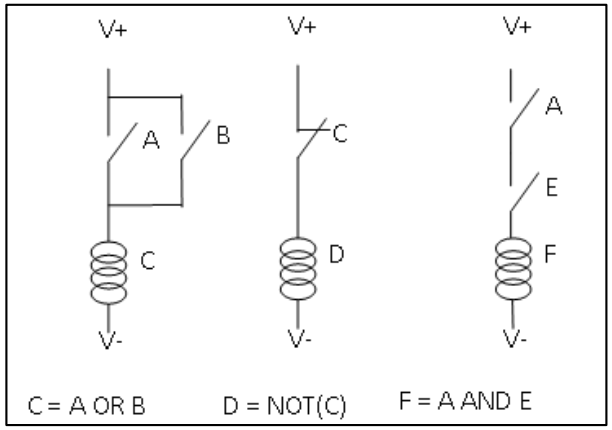
Il PLC sostituisce le logiche a relais (anni '80-'90)

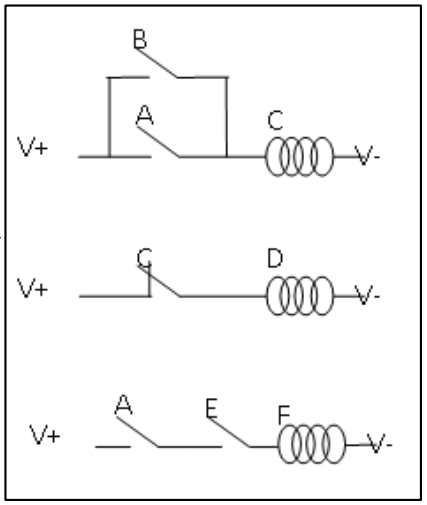






I Contatti possono essere cablati come NO o NC





- **Relais: facendo scorrere corrente nella bobina i contatti NO –normally open- si chiudono e quelli NC –normally closed- si aprono**
- **Collegando opportunamente i contatti NO e NC (parallelo = OR, serie = AND) si ottengono funzioni logiche booleane. Mediante percorsi di retroazione si possono ottenere le memorie (es. OUT =!Res&(Set+Out))**
- **Ruotando a sinistra di 90° uno schema a relais si ottiene una sequenza di istruzioni (ladder diagram)**

Il linguaggio logico booleano: Ladder Diagram

- **Logiche a relais**
  - **Out = 1 se e solo se nella bobina passa corrente, ossia se c'è almeno un percorso di contatti chiusi che porta l'alimentazione fino alla bobina (flusso di corrente dall'alto in basso)**
- **Ladder Diagram**
  - **Se (logiche contatti) = 1 allora Out =1 altrimenti Out =0 (flusso di "corrente" da sinistra a destra)** ***If Input then Out=1 else Out=0***
  - **Contatto NO = interrogazione vera di variabile booleana -| |-**
  - **Contatto NC = interrogazione negata di variabile booleana -|/|-**
  - **Esistono altri operatori di ingresso booleani:**
    - **NOT (nega l'espressione logica a sinistra del NOT) -|NOT|-**
    - **Rilevatore di fronte di salita (instanzia una cella di memoria –OldVar- che memorizza il valore vecchio della variabile Var e l'interrogazione "P" si attiva se Var&!Oldvar) -|P|-**
    - **Rilevatore di fronte di discesa (instanzia una cella di memoria –OldVar- che memorizza il valore vecchio della variabile Var e l'interrogazione "N"si attiva se !Var&Oldvar) -|N|-**
  - **Esistono altri operatori di uscita booleani che, al contrario di bobina -( )-, rispondono al costrutto** ***If Input then Operation (else no operation and go on)***
    - **Set (imposta a 1 la variabile) –(S)-**
    - **Reset (imposta a 0 la variabile) –(R)-**

Il linguaggio logico booleano: Ladder Diagram

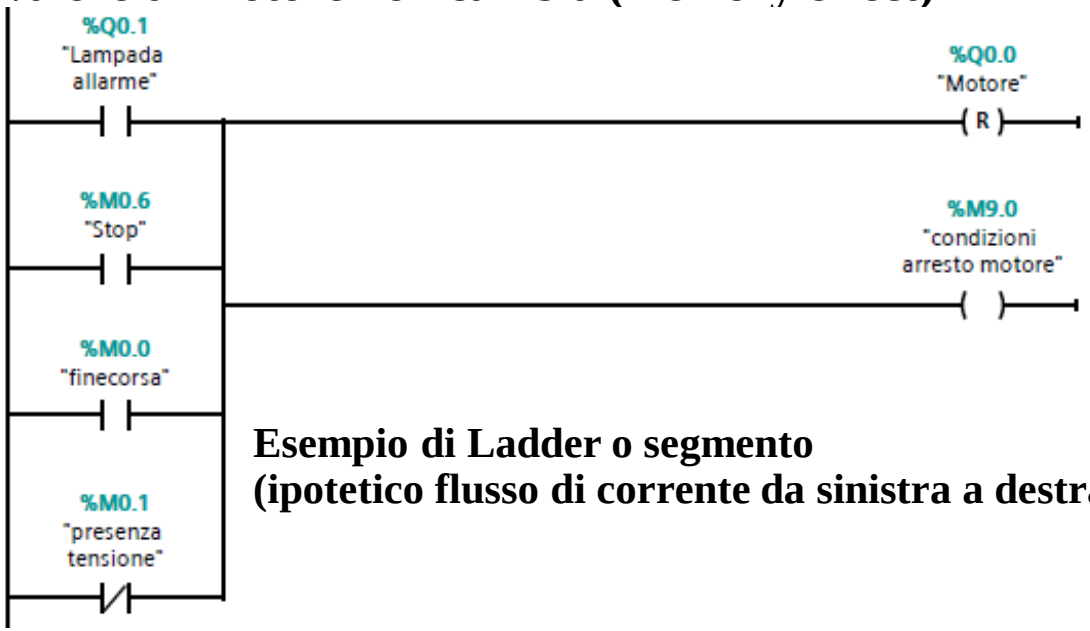
- **L'operatore di rilevazione di fronte di salita/discesa**
  - **La rilevazione del fronte positivo, invece dell'interrogazione a 1, è una condizione più restrittiva (ad es. permette di distinguere se un pulsante si è incastrato o è stato regolarmente pigiato, ossia se è passato dallo stato "inattivo" allo stato "attivo")**
  - **S1: *If* (fronte positivo(ripristino_p)).AND.(Allarme=0) *then* Lampada allarme:=0**
  - **Nota: subito dopo l'esecuzione della rilevazione del fronte, il valore di ripristino_var_p viene aggiornato con il valore di ripristino_p e quindi il segmento S2 non avrebbe mai effetto. E' perciò opportuno realizzare delle variabili di appoggio che si attivano solo nel ciclo in cui c'è il fronte e utilizzare tali variabili in più punti del programma (vedi S3, dove start_p può essere utilizzato più volte)**

Esempi di Ladder o segmenti (ipotetico flusso di corrente da sinistra a destra)

Il linguaggio logico booleano: Ladder Diagram

- **Logiche booleane basate su bobina di assegnazione**
  - ***If Input then Out=1 else Out=0***
  - ***If* (Lampada allarme=1).OR.(Stop=1).OR.(finecorsa=1).OR.(presenza tensione=0) *then* condizioni arresto motore:=1 *else* condizioni arresto motore:=0**
  - **Nota: condizioni arresto motore viene comunque sempre modificato e quindi non può essere utilizzato in uscita in nessuna altra parte del programma**

- **Logiche booleane basate su bobine di Set e Reset**
  - ***If Input then Operation (else no operation and go on)***
  - ***If* (Lampada allarme=1).OR.(Stop=1).OR.(finecorsa=1).OR.(presenza tensione=0) *then* Motore:=0 altrimenti il valore di Motore non cambia (memory effect)**
  - **Nota: le logiche Set Reset possono non agire (non avere alcun effetto) e quindi coesistono con altre istruzioni di uscita in logica Set Reset. Ad esempio, A Set C (ladder1) e B set C (ladder2) equivale a A.OR.B set C**



Esempio di Ladder o segmento (ipotetico flusso di corrente da sinistra a destra)

Il linguaggio Ladder Diagram, tipi di dati

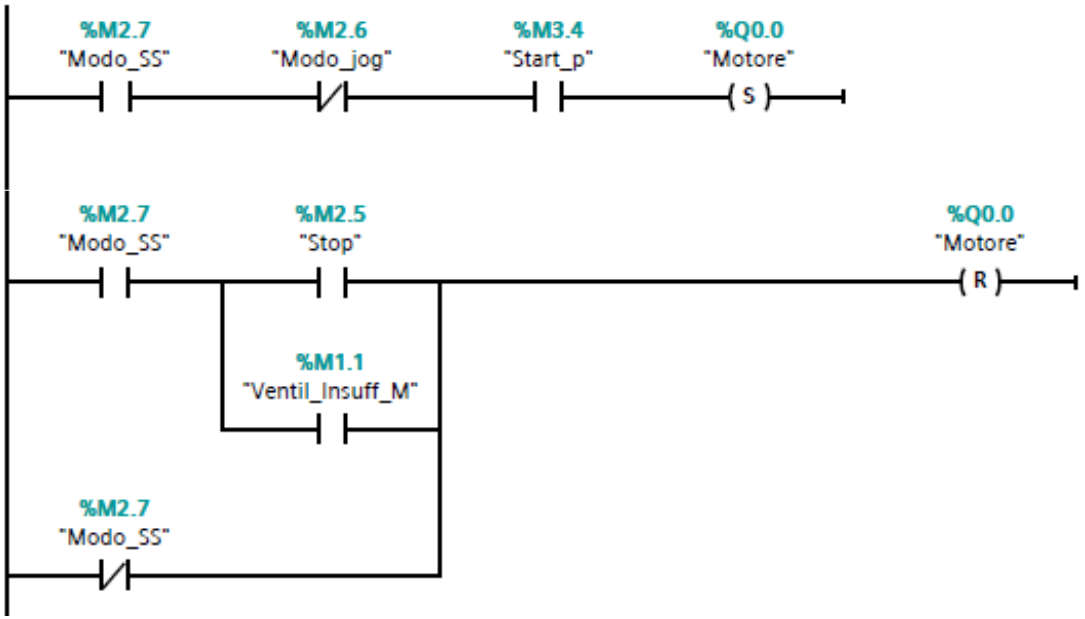
- **Numeri binari**
  - **BIT (variabile booleana), stringhe di bit (BYTE (8 bit), WORD (16 bit), DWORD (32 bit), LWORD (64 bit, non tutti i PLC)**
- **Numeri interi**
  - **SINT (intero con segno a 8 bit, -128…+127)**
  - **USINT (intero senza segno a 8 bit, 0…+255)**
  - **INT e UINT (intero, con e senza segno, a 16 bit)**
  - **DINT e UDINT (intero, con e senza segno, a 32 bit)**
  - **LINT e ULINT (intero, con e senza segno, a 64 bit, non tutti i PLC)**
- **Numeri in virgola mobile**
  - **REAL (floating point IEEE754 a 32 bit)**
  - **LREAL (floating point IEEE754 a 64 bit, non tutti i PLC)**
- **Dati di tipo Time**
  - **32 bit, esprime il numero intero di millisecondi nel formato con segno da**
  - **T#-24d20h31m23s648ms a T#+24d20h31m23s647ms (Es. T#1s630ms)**
  - **Conversione implicita con DWORD, DINT e UDINT**
- **Altri tipi di dati**
  - **ARRAY, STRUCT, non supportati da tutti i PLC**

Linguaggio Ladder Diagram, operatori e memoria

- **Operatori di ingresso**
  - **Operatori di confronto (==, <>, >, <, >=, <=)**
- **Operatori di uscita**
  - **Tutti gli operatori di uscita, tranne la bobina di assegnazione, operano in logica Set Reset, ossia *If Input then Operation (else go on –memory effect-)***
  - **Operatori per bit e stringhe di bit –bit_field- (Set, Reset)**
  - **Operatori logici a parola (AND, OR, NAND, NOR, XOR,…)**
  - **Operatori di trasferimento (Move, Move_Blk)**
  - **Operatori di conversione di tipo (Ceil –da Floating Point a Intero superiore- Floor –da FP a intero inferiore, Convert)**
  - **Operatori aritmetici (ADD, SUB, MUL, DIV, MOD,…)**
- **Organizzazione della memoria (PLC S7-1200)**
  - **La memoria è organizzata a byte indirizzati in modo assoluto. Ad esempio:**
    - **M6.4 è il bit 4 del byte all'indirizzo 6**
    - **MB6 è il byte all'indirizzo 6. Nota: se modifico M6.4 cambio anche MB6**
    - **MD6 è la double word che occupa gli indirizzi 6,7,8,9**
  - **Tutti gli ingressi I e le uscite Q sono mappati in memoria (Immagini di processo di ingressi e uscita e organizzati a byte)**
  - **La tabella delle Variabili permette di dare dei simbolici agli indirizzi fisici**
  - **Esistono anche strutture dati rilocabili (Data Blocks)**

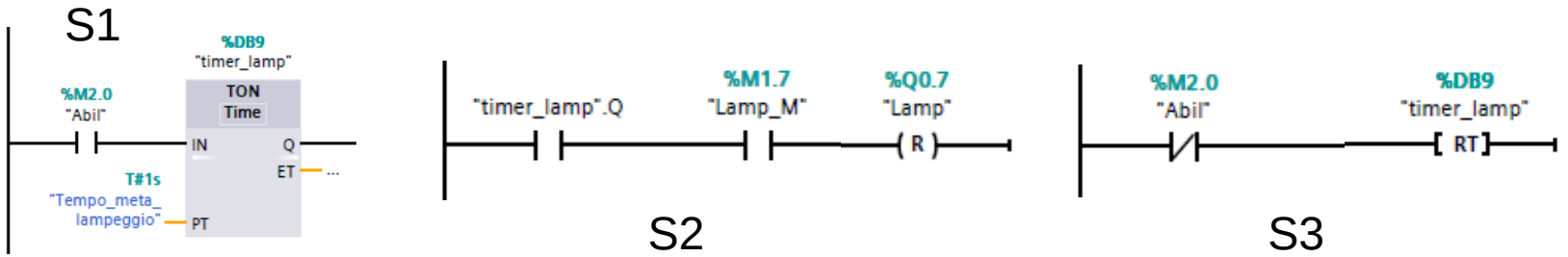
Il linguaggio Ladder Diagram, ordine dei segmenti

- **Un programma in Ladder è la sequenza ordinata di più segmenti**
- **L'ultimo segmento ha la priorità sui precedenti**
  - **Segmento 1: *If* (Modo_SS=1).AND.(Modo_jog=0).AND.(Start_p=1) *then* Motore:=1**
  - **Segmento 2: *If* ((Modo_SS=1).AND.((Stop=1).OR.(Ventil_insuff_M=1)).OR. (Modo_SS=0) *then* Motore:=0**
  - **Quindi se (Modo_SS=1).AND.(Modo_jog=0).AND.(Start_p=1).AND.(Stop=1) Segmento1 setta Motore(IPU) e Segmento2 lo resetta, ma l'uscita fisica Motore è fissa a zero, perché è stata settata (e poi resettata) solo la IPU**



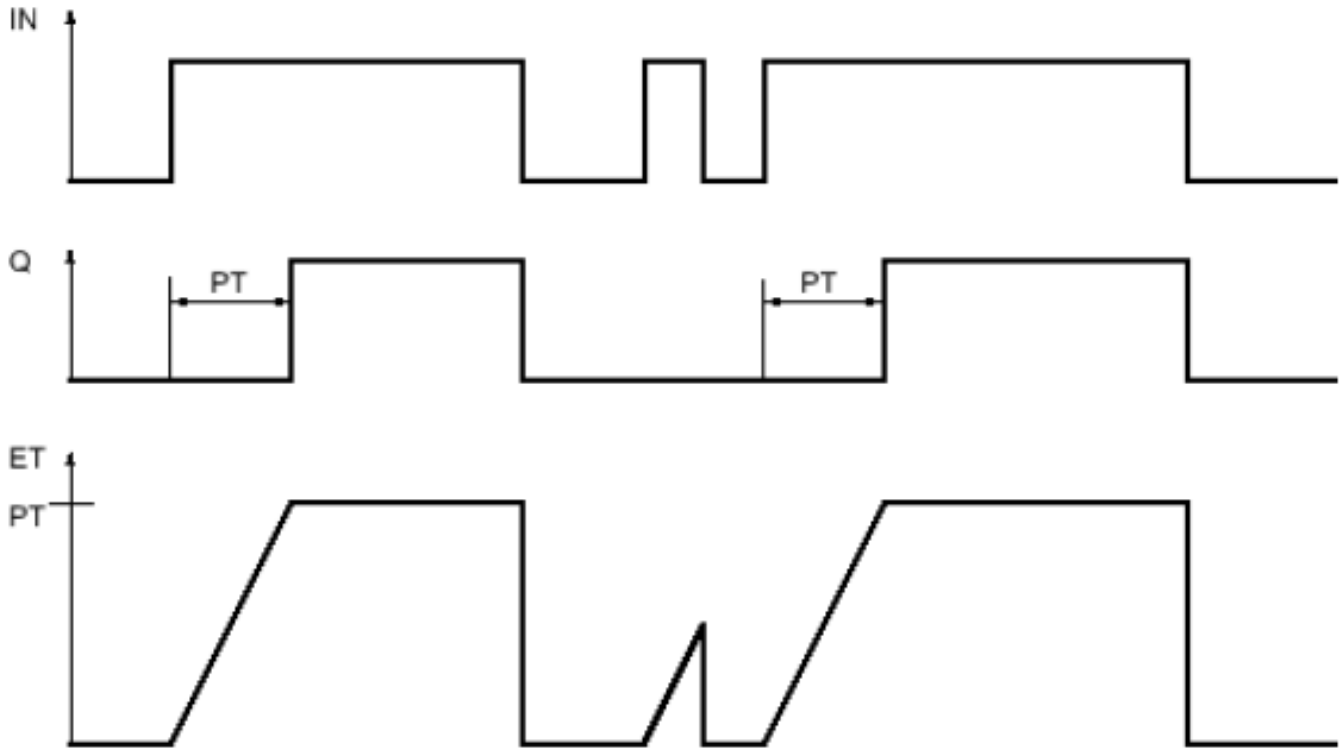
Il linguaggio Ladder Diagram: timer

- **Un temporizzatore o timer è un elemento che, se abilitato, inizia a contare il tempo che passa in unità ms e permette di realizzare sequenze temporizzate, filtri e gestire segnali di ingresso e uscita**
- **Un timer è costituito da un blocco dati (DB):**
  - **Ingresso IN –ingresso di abilitazione- (dato di tipo bit)**
  - **Ingresso PT –costante di tempo- (dato di tipo Time)**
  - **Ingresso R –reset- (dato di tipo bit)**
  - **Uscita ET –valore corrente o tempo accumulato- (dato di tipo Time)**
  - **Uscita Q –uscita, raggiungimento costante di tempo- (dato di tipo bit)**
- **Un timer può essere implementato dal costruttore in modi diversi:**
  - **Il valore ET si aggiorna quando va in esecuzione il segmento che dichiara il timer (es. S1) –utilizzato dal PLC S7-1200-**
  - **Il valore ET si aggiorna in corrispondenza dell'aggiornamento delle IPI e/o delle IPU**
  - **Il valore ET si aggiorna ad interrupt**
  - **Nel complesso, conviene che i timer siano prima instanziati (es. S1), usati (es. S2) e infine resettati (es. S3)**



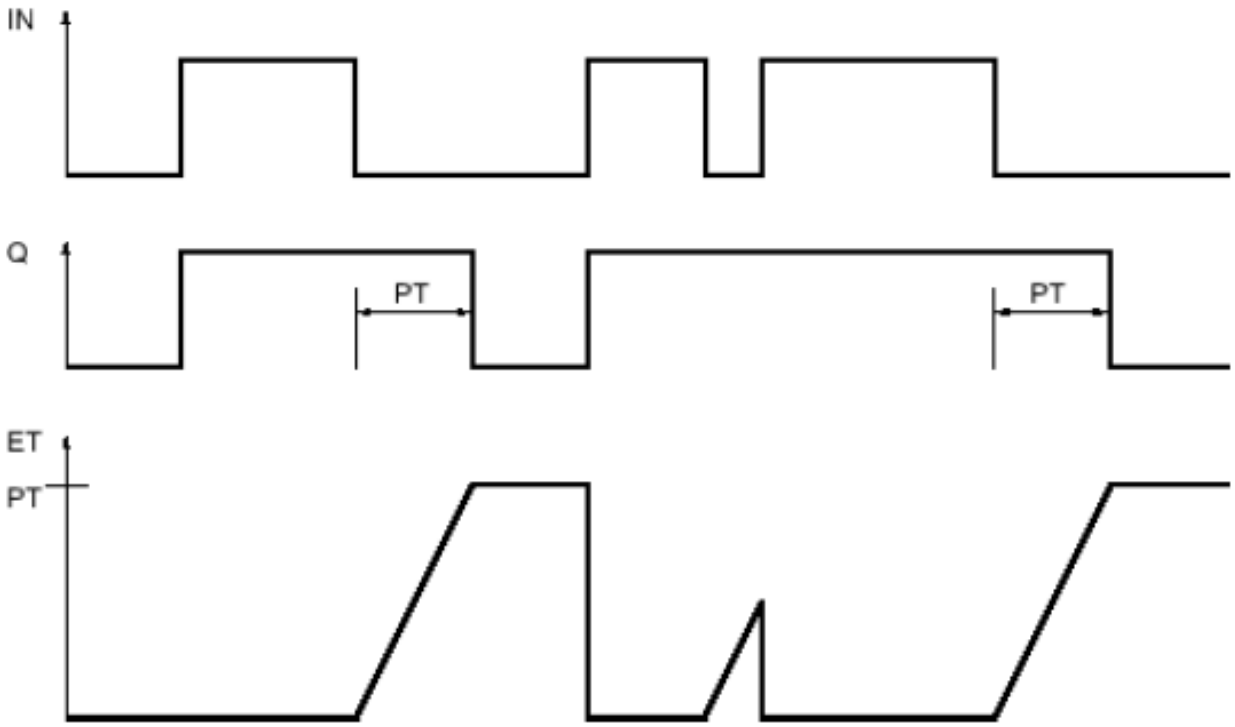
Il linguaggio Ladder Diagram: timer TON

- **Un timer TON (ritardo all'inserzione) ritarda il fronte di salita**
- **Se l'informazione presente su IN ha un fronte di salita, si attende PT quindi l'uscita Q va a 1 fino a quando IN non ha una commutazione a zero. La commutazione a zero di IN durante PT azzera il Timer. E' il timer più utilizzato insieme a TONR**



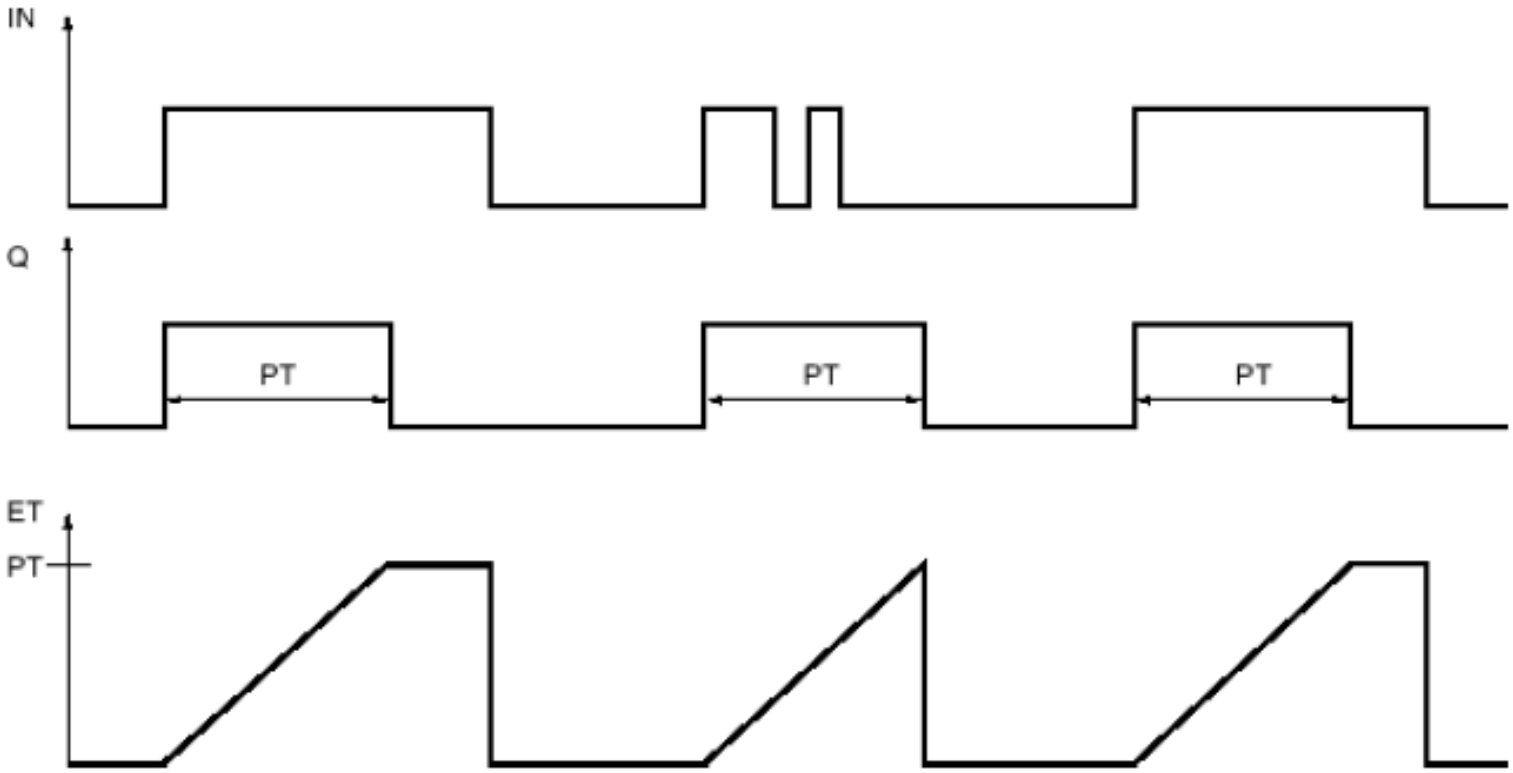
Il linguaggio Ladder Diagram: timer TOF

- **Un timer TOF (ritardo alla disinserzione) ritarda il fronte di discesa**
- **Se l'informazione presente su IN ha un fronte di salita l'uscita Q è posta a 1; al fronte di discesa di IN, l'uscita rimane a 1 e vi rimane per un tempo PT quindi viene posta a 0. Un nuovo fronte di salita di IN fa ripartire il meccanismo (retrigger)**



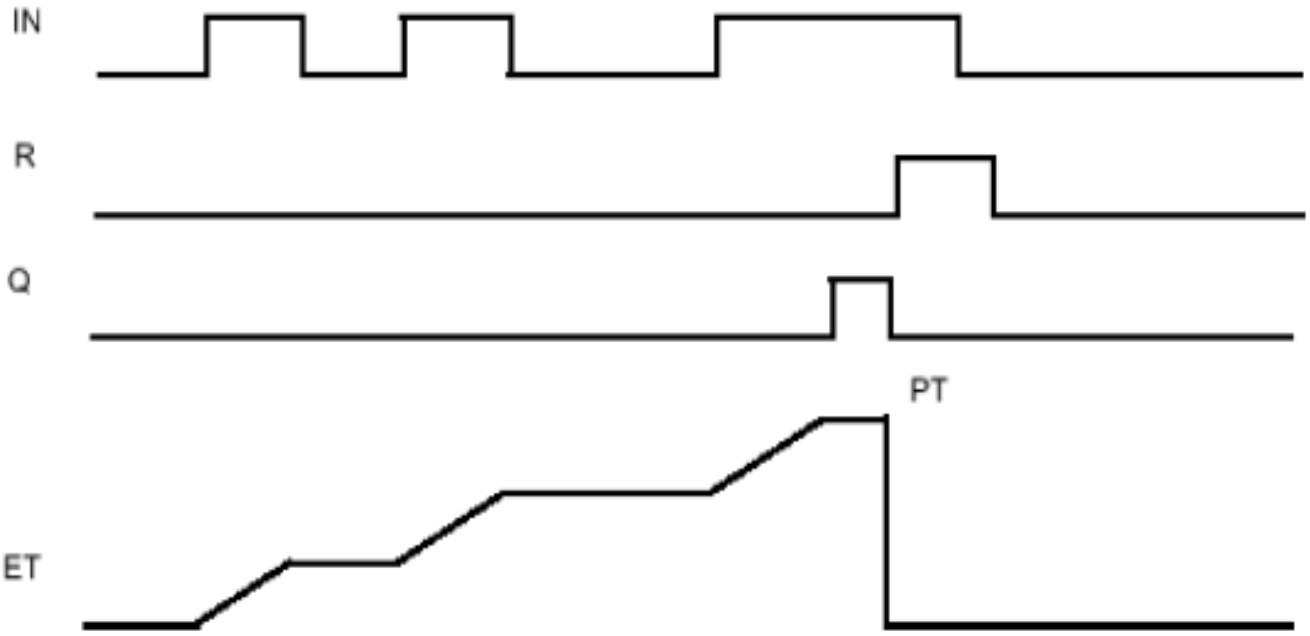
Il linguaggio Ladder Diagram: timer TP

- **Un timer TP (avvio impulso) avvia un impulso sul fronte di salita**
- **Se l'informazione presente su IN ha un fronte di salita, l'uscita Q viene impostata per una durata programmata PT indipendentemente da come evolve IN**



Il linguaggio Ladder Diagram: timer TONR

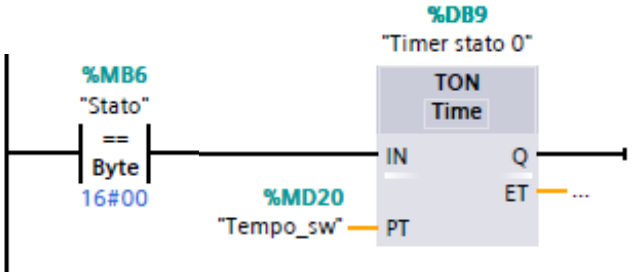
- **Un timer TONR (TON con memoria) è un accumulatore temporale**
- **Se l'informazione presente su IN ha un fronte di salita, ET accumula i valori temporali in corrispondenza di IN=1, mantenendo il valore precedente se IN=0; l'uscita Q viene posta a 1 allo scadere di PT. Il segnale di reset R (booleano) resetta ET e Q**
- **I timer TONR devono SEMPRE essere resettati (nei TON è sufficiente avere In=0)**



Il linguaggio Ladder Diagram: timer

- **I timer possono essere usati in modo booleano (uscita Q) o considerando ET (Time, assimilabile a DWORD o DINT o UDINT), che può essere letta e scritta, come accade per PT**
- **E' possibile avere timer con costante di tempo variabile (Es. MD20, di tipo Time)**

"Tempo_sw"	%MD20	Time

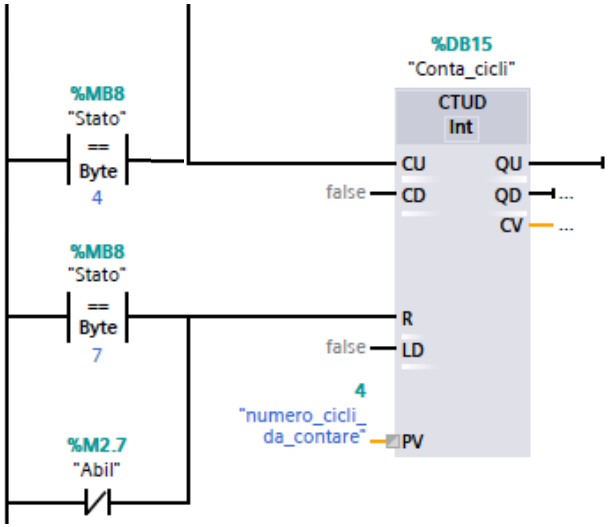


Tipo Timer	Il conteggio inizia	Il conteggio si ferma	Il conteggio si azzera (*)	Out è a 1 se timer attivo e	Il timer si disattiva
TON	Fronte salita In		Fronte discesa In	Se ET >= PT	Fronte discesa In
TOF	Fronte discesa In		Fronte salita In	Se ET < PT	Solo al reset
TP	Fronte salita In		Fronte discesa In & ET>=PT	Se ET < PT	Fronte discesa In & ET>=PT
TONR	Fronte salita In	Fronte discesa In	Solo al reset	Se ET >= PT	Solo al reset

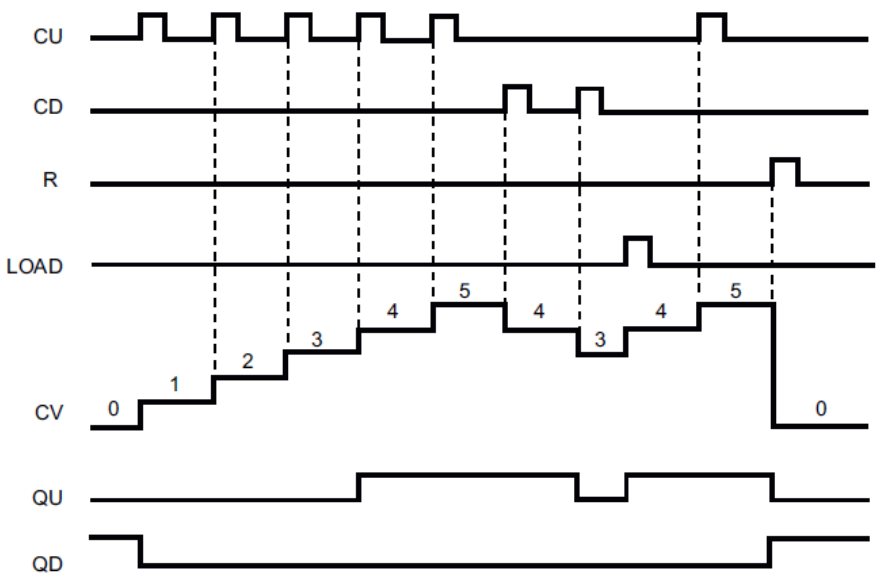
Il linguaggio Ladder Diagram: counter

- **Un contatore o counter è un elemento che conta eventi (inc., dec. o up&down) evento = commutazione da falso a vero di un Bool**
- **Un counter (Es. CTUD up&down) è costituito da un blocco dati (DB):**
  - **Ingresso CU –conteggio UP ossia in incremento- (dato di tipo bit)**
  - **Ingresso CD –conteggio DOWN ossia in decremento- (dato di tipo bit)**
  - **Ingresso R –reset- (dato di tipo bit)**
  - **Ingresso LOAD –Carica il valore PV in CV, attivo solo in CTD e CTUD- (bit)**
  - **Ingresso PV –Valore o soglia di conteggio- (tipo Sint, Int, Dint, Usint, Uint o Duint)**
  - **Uscita CV –valore corrente di conteggio- (tipo Sint, Int, Dint, Usint, Uint o Duint)**
  - **Uscita QU –uscita, raggiungimento soglia superiore (CV>=PV)- (dato di tipo bit)**
  - **Uscita QU –uscita, raggiungimento soglia inferiore (CV<=0)- (dato di tipo bit)**

Il Counter deve esempre essere resettato!



S1



Il linguaggio Ladder Diagram: osservazioni

- **La maggior parte dei programmi per PLC è scritto in linguaggio Ladder e consta di:**
  - **Tabella dei simboli, dove ingressi, uscita e memoria o "Merker" (IPI I, IPU Q, memoria M) sono elencati secondo il costrutto: Nome_simbolico, tipo di dato, indirizzo assoluto, commento**
  - **Programma principale (OB1), che consta di una sequenza organizzata di istruzioni nel linguaggio selezionato. Può contenere "sottoprogrammi" in forma FC (con possibili parametri di I/O) e in forma FB (con possibili parametri di I/O e aree di memoria personalizzate)**
  - **Programmi associati ad eventi: possono essere associati a ingressi o a eventi (eccezioni) quali la partenza "a freddo" o "a caldo" (es. OB100, associato al reset), la perdita del programma applicativo (batterie scariche), o a interrupt periodici…**
  - **Blocchi dati, gestiti da sistema (come i Timer e i counter), gestiti dal programma principale o relativi a "sottoprogrammi" di tipo FB**
- **Lo standard IEC61131-3 prevede altri linguaggi di programmazione**

Il linguaggio Ladder Diagram: uso dei dati

- **I dati (Immagini di I/O, variabili,..) sono allocati secondo l'indirizzo fisico (non deallocati come nella programmazione su PC)**
  - **La "fisicità" dell'indirizzo è un esigenza**
  - **Si pensi ad un PLC con 2 moduli identici di Ingressi Logici (32 Cad.): il cablatore utilizza il terzo contatto del secondo modulo (lo distingue perché lo vede) e lo collega ad un finecorsa; il programmatore sa che c'è un finecorsa ma come recupera l'informazione? I moduli hanno indirizzamento geografico (posizionale) fisso e analogamente è per le variabili**
    - **Il "sistema operativo" del PLC può non avere un MMU (o non potente e standard come gli OS)**
  - **Uno stesso indirizzo fisico può essere accessibile da due variabili logiche distinte (Attenzione!)**

- **Allocare dati correlati contigui tra loro in un unico "blocco"**
- **Le sessioni di comunicazione trasferiscono tali "blocchi"**
- **Sovradimensionare i blocchi per tenere conto di eventuali modifiche**

- **I dati (IPI, IPU, merker) sono nella Tabella dei simboli o lista delle attribuzioni (dare nomi simbolici per una buona significatività e reperibilità del dato)**

Il linguaggio Ladder Diagram: strategie

- **Ripartire il problema in più sottoproblemi (strutturare il programma in sottoprogrammi e suddividere i segmenti complessi in più segmenti semplici facendo uso di variabili intermedie), affrontando prima le funzionalità indipendenti e poi quelle dipendenti**

- **Seguire le logiche in sicurezza, ridondando gli interblocchi**
  - **Esempio di interblocco bilaterale: selettore RICETTE 1,2,3 (R1,R2,R3)**
    Senza interblocchi: Se R1 allora…
    Con interblocchi: Se R1&!R2&!R3 allora…
  - **Esempio di interblocco unilaterale: avviamento Start-Stop di un motore**
    Senza interblocchi: Se Start allora Set(Motore), se Stop allora Reset..
    Con interblocchi: Se Start&!Stop allora…, se Stop allora…

- **Considerare condizioni di guasto tipico (Es. strappo cavi)**

- **Strutturare il programma secondo la logica del "minimo impatto delle modifiche" (se si aggiungesse una funzionalità si dovrebbe modificare il programma nel minimo numero di punti)**

- **Controllare la data dependency e minimizzare i casi da testare**

Il linguaggio Ladder Diagram: strategie

Considerare condizioni di guasto tipico
(Es. strappo cavi, sensore non alimentato: cosa succede se strappo i cavi? come si comporta il sistema?)

- **I segnali abilitanti (Es. START, Ripristino_Allarme,…) sono cablati utilizzando contatti NA (segnale attivo <-> passa corrente)**
- **I segnali inibenti (Stop, Allarme,…) sono cablati NC (segnale attivo <-> non passa corrente**
- **Se si strappano i cavi o se i sensori non sono alimentati allora non passa corrente e, in sicurezza, non si attivano i segnali abilitanti e si attivano i segnali inibenti, che portano in sicurezza l'impianto**
- **Il programma, con segnali che si attivano a 1 in logica vera e segnali che si attivano a 0 in logica negata, diventa illeggibile**
- **Si usano variabili di appoggio che negano o no il segnale di ingresso a seconda che sia cablato NC o NO**

Il linguaggio Ladder Diagram: strategie

1. **Si considerano tutti i segnali di ingresso e uscita con le loro polarità e si configura l'hardware correttamente (moduli di I/O, filtri, uscite in condizione di sicurezza,…)**
2. **Si predispongono delle variabili di appoggio per gli ingressi, in modo da poterli trattare tutti in logica vera (XOR dei byte di ingresso con maschere, poiché A.XOR.0=A e A.XOR.1=!A)**
3. **Si predispongono dei filtri TON per i segnali di ingresso, in modo da filtrare l'attivazione dei segnali**
4. **Si predispongono variabili che si attivano sui fronti dei segnali di ingresso, così da poterle utilizzare in più punti del programma (tipicamente nel caso di pulsanti associati ad azioni abilitanti)**
5. **Si predispongono variabili dove memorizzare lo stato delle uscite a inizio ciclo, in modo che le uscite si possono interrogare in più punti del programma ricevendo sempre la stessa risposta**
6. **Si implementano le funzionalità, prima quelle indipendenti poi quelle dipendenti**

Il linguaggio Ladder Diagram: strategie

1. Si considerano tutti i segnali di ingresso e uscita con le loro polarità e si configura l'hardware correttamente (moduli di I/O, filtri, uscite in condizione di sicurezza,…)
2. Si predispongono delle variabili di appoggio per gli ingressi, in modo da poterli trattare tutti in logica vera (XOR dei byte di ingresso con maschere, poiché A.XOR.0=A e A.XOR.1=!A)
3. Si predispongono dei filtri TON per i segnali di ingresso, in modo da filtrare l'attivazione dei segnali
4. Si predispongono variabili che si attivano sui fronti dei segnali di ingresso, così da poterle utilizzare in più punti del programma (tipicamente nel caso di pulsanti associati ad azioni abilitanti)
5. Si predispongono variabili dove memorizzare lo stato delle uscite a inizio ciclo, in modo che le uscite si possono interrogare in più punti del programma ricevendo sempre la stessa risposta
6. Si implementano le funzionalità, prima quelle indipendenti poi quelle dipendenti

Applicazioni industriali in Ladder: allarmi

- **Tipicamente si distingue tra:**
  - **Eventi (modificano il comportamento del controllore, ad esempio cambiando tipo di lavorazione)**
  - **Anomalie (eventi non memorizzati che modificano il comportamento del controllore per mettere in sicurezza l'impianto)**
  - **Allarmi (eventi memorizzati che modificano il comportamento del controllore per mettere in sicurezza l'impianto, anche dopo che il segnale di allarme si è disattivato)**

- **Gli allarmi possono essere allarmi booleani o a soglia e sono segnalati mediante lampade o sirene**
  - **Valore sotto soglia di attenzione: lampada spenta**
  - **Valore sopra soglia di attenzione ma sotto soglia di allarme: lampada che lampeggia lenta**
  - **Valore sopra soglia di allarme: lampada che lampeggia veloce**
  - **Valore sopra soglia di allarme continuativamente da più di un tempo T: lampada accesa fissa. Tale condizione viene memorizzata**

Applicazioni industriali in Ladder: allarmi

- **Tipologie di allarme con memorizzazione:**
  - **Allarme con memorizzazione: serve un reset generale dell'impianto, si tratta degli allarmi più gravi**
  - **Allarme con memorizzazione e riconoscimento: c'è un comando di Ripristino che sblocca l'allarme (in caso si attivi Ripristino quando Allarme è ancora attivo vince il Ripristino). Si applica agli allarmi meno gravi, dove si permette all'operatore di riprendere le normali funzionalità**
  - **Allarme con memorizzazione, riconoscimento e rientro: c'è un comando di Ripristino che sblocca l'allarme (in caso si attivi Ripristino quando Allarme è ancora attivo vince l'allarme)**

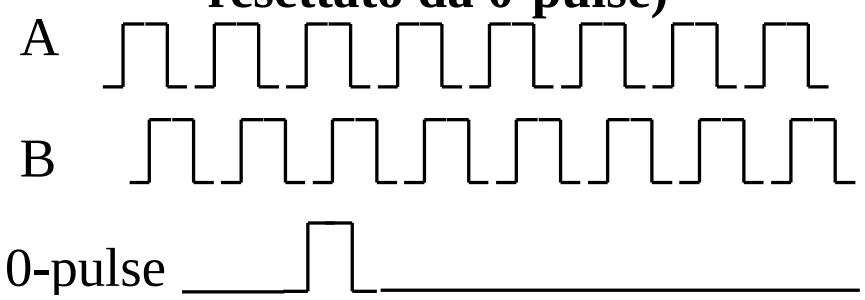
- **Funzioni correlate alla gestione di allarmi**
  - **Riconoscimento del primo allarme intervenuto (normalmente le azioni per mettere in sicurezza l'impianto provocano ulteriori allarmi ed anomalie per cui è complicato individuare la prima causa d'allarme)**
  - **Priorità e riconoscimento degli allarmi**

Applicazioni industriali in Ladder: motori

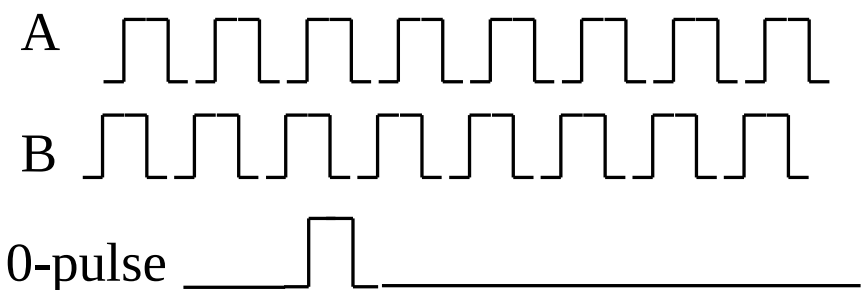
- **Normalmente si distingue tra:**
  - **Motori non controllati: il PLC controlla direttamente l'interruttore che porta l'alimentazione di rete al motore. Il motore gira a "50Hz", ossia 50 giri/s, ossia 3000rpm (round per minute). Si tratta di motori ausiliari di piccola taglia, come ad es. la ventilazione di motori grossi**
  - **Motori con soft-start: il PLC controlla il motore mediante un segnale booleano di uscita, ma per evitare che il motore dissipi potenza in calore all'avviamento (coppia di spunto necessaria per superare le inerzie), il PLC in realtà comanda un Soft-Start che regola progressivamente la potenza data al motore**
  - **Motori controllati (mediante azionamento o drive): il PLC dispone di moduli specifici che permettono di interfacciarsi ad un azionamento (drive) che, modulando opportunamente le tensioni di rete, in termini di frequenza, fase e modulo, permette di regolare la coppia, la velocità e la posizione angolare del motore. Spesso il PLC dispone di un modulo di interfaccia encoder (sensore accoppiato alla parte mobile –rotore- e fissa –statore- del motore che rileva velocità e posizione)**

Applicazioni industriali in Ladder: motori

- **L'encoder è un sensore per la rilevazione di velocità angolare, verso di rotazione e posizione angolare di un motore**
  - **L'encoder genera N di impulsi per giro (tipicamente 1.000-10.000)**
  - **Il segnale A è quindi un segnale che può variare da 0 a 1 MHz (N=20000, 50 giri/s). La frequenza del segnale A è troppo elevata rispetto al ciclo di scansione del PLC e servono dei moduli specifici con ingressi "veloci". I moduli "encoder" sono basati su FPGA che implementano contatori in λ/4 (segnale a frequenza maggiore x4 ottenuto generando un impulso in corrispondenza dei fronti di A e B)**
  - **Il segnale B è sfasato dal segnale A di +90° o -90° a seconda del verso di rotazione. Il segnale 0-pulse permette di rilevare la posizione P (P=360°·N_A/N, dove N_A è il valore del contatore degli impulsi di A resettato da 0-pulse)**



AVANTI



INDIETRO

Applicazioni industriali in Ladder: motori

- **Le logiche di comando dei motori tengono conto del fatto che un motore possa essere gestito (es. avviato –Start- o fermato –Stop, prevalente su Start-) in contesti diversi (I modi riportati sono selezionati mediante un selettore posto vicino al motore):**
  - **Modo Jog: modo di test nel quale si verifica che il motore si muova correttamente, che sia ben fissato ai supporti, ecc. Vicino al motore c'è un pulsante di JOG e il motore va (in genere a velocità ridotta) fintanto che il pulsante è premuto e si arresta quando il pulsante viene rilasciato**
  - **Modo Locale: modo di test nel quale si verifica il corretto funzionamento del motore a piena velocità. Vicino al motore ci sono due pulsanti di Start e Stop che rispettivamente avviano e fermano il motore**
  - **Modo Remoto: modo operativo da operatore su pulpito di comando o in sala controllo**
  - **Modo Automatico: i segnali Start e Stop arrivano da un sistema automatico e non da un operatore**
- **E' fondamentale la mutua esclusione o priorità tra i modi**
  - **Il PLC spesso implementa funzioni di diagnostica del selettore**

Applicazioni industriali in Ladder: sequenze

- **La maggior parte dei programmi per PLC può essere suddivisa in sequenze automatiche, dove il programma passa da uno stato all'altro sulla base di diverse transizioni, che dipendono da segnali di ingresso, eventi, e/o dal trascorrere del tempo**

- **Spesso i controlli non sono retroazionati (leggi livello, se livello < soglia allora riempi altrimenti ferma e passa oltre) ma temporizzati (riempi per Ti, dove Ti viene aggiustato sulla base dei controlli di qualità e dei processi batch): controlli più veloci, economici ed affidabili (sensorless)**

- **Tali sequenze automatizzate hanno un segnale di abilitazione (Abil). Se Abil->0 la sequenza automatica si arresta e:**
  - **la sequenza prosegue in manuale (per poi magari riprendere in automatico)**
  - **oppure la mancanza di Abil viene considerata assimilabile ad un guasto, la sequenza viene interrotta, l'operatore ripristina le condizioni iniziali e si riparte da zero**

Applicazioni industriali in Ladder: sequenze

- **Le sequenze realizzate in Ladder sono semplici:**
  - **C'è un solo stato attivo alla volta**
  - **Prima si esaminano le transizioni condizionate (quelle da uno specifico stato verso un altro specifico stato) poi quelle incondizionate (da qualsiasi stato verso uno specifico stato) impostando una variabile "Stato_futuro" (in questo modo l'ultima transizione è prioritaria)**
  - **Si effettua l'eventuale passaggio di stato memorizzando lo stato di provenienza (Stato_old<= Stato; Stato<= Stato_futuro)**
  - **Si eseguono i passi (sequenza di azioni) per ogni stato o combinazione di stato**
  - ***If* Stato=i *then* … (cose da fare mentre si è nello stato i)**
  - ***If* (Stato=i).AND.(Stato_old<>i) *then* … (cose da fare solo una volta all'ingresso nello stato i da qualsiasi stato)**
  - ***If* (Stato<>i).AND.(Stato_old=i) *then* … (cose da fare solo una volta all'uscita dallo stato i verso qualsiasi stato**
  - ***If* (Stato=i).AND.(Stato_old=j) *then* … (cose da fare solo una volta a seguito della transizione da j a i)**
- **NOTA: normalmente le transizioni sono più dei passi e lasciarle prive di azioni semplifica la fase di test&debug**