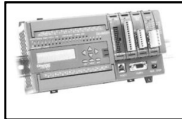


# PLC



## Controllori a Logica Programmabile (Programmable Logic Controller)



Ing. Elena Mainardi

PLC

## I nomi del PLC nel mondo

PLC = Programmable Logic Controller

API = Automate Programmable Industriel

SPS = Speicherprogrammierbare Steuerung

Le macchine automatiche esistono praticamente da dopo la rivoluzione industriale.

Nell'800 ovviamente erano macchine molto semplici, poi, con il progresso della tecnologia e la scoperta e l'utilizzo dell'elettricità, hanno cominciato a diventare sempre più complesse.

Fino agli anni '70, le grosse macchine automatiche che già esistevano nelle catene di montaggio e nei grandi stabilimenti, funzionavano essenzialmente a relè. I relè erano collegati a motori e/o a generici attuatori e comandati da ingressi come i sensori.

Es. se sensore di pressione dà un valore basso → fai scattare il relè che comanda il motore per l'apertura della valvola

In pratica le macchine automatiche erano macchine ANALOGICHE, piene di relè cablati con struttura fissa.

Quindi modificare il comportamento della macchina automatica comportava il dover rifare il cablaggio dei componenti di cui si voleva modificare il comportamento, cioè togliere i cavi precedentemente installati e rifare le connessioni → ECONOMICAMENTE ONEROSO, PERDITA DI TEMPO....

## Il problema del cablaggio!!!



Ing. Elena Mainardi

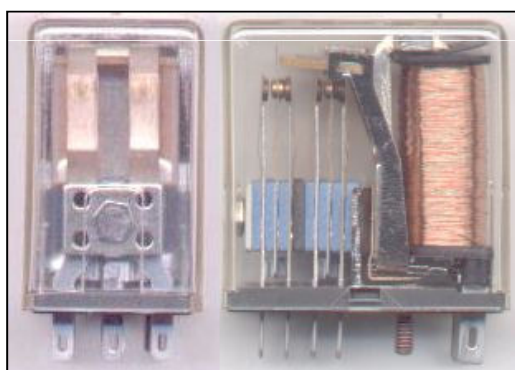
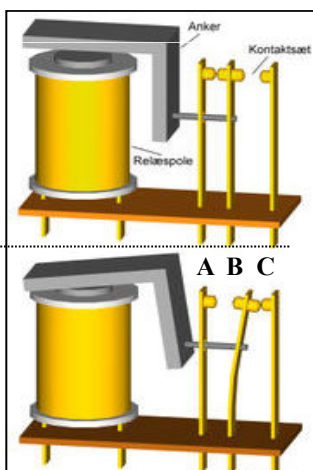
PLC

### Cos'è un relè?

Il **relè** è un dispositivo che utilizza le variazioni della corrente per influenzare le condizioni di un altro circuito. Se ne possono trovare elettronici, elettromagnetici, a induzione, a semiconduttore e termici. Il relè elettromagnetico è il più diffuso ed è costituito da un elettromagnete che, facendo passare un flusso di corrente in una bobina di filo, attrae una struttura di ferro, aprendo e chiudendo un contatto. In sostanza il relè è un interruttore che non viene azionato a mano ma da un elettromagnete.

Non passa corrente. La bobina non esercita nessuna forza di attrazione. Sono a contatto i terminali A e B

Passa corrente. La bobina diventa un elettromagnete e attira a sé l'ancorina di metallo, che apre il contatto tra A e B e lo chiude tra B e C



Ing. Elena Mainardi

PLC

Inconvenienti:

- **Debugging difficoltosi**; soprattutto nei sistemi di una certa complessità (riuscire ad individuare un relè rotto, o mal collegato, all'interno di un enorme circuito contenuto in un armadio di sei metri, può non essere un'operazione semplice da fare).
- **Manutenzioni onerose**; legata all'utilizzo di una gran quantità di strumenti elettromeccanici (i relè), sensibili, per loro stessa natura, all'usura ed all'invecchiamento.
- **Ingombro elevato** per quanto in origine i relè fossero considerati dispositivi di ridotte dimensioni, la necessità di controlli sempre più complessi nel campo dell'automazione industriale, ha portato gli impianti di maggiore dimensione a misurarsi con la necessità di realizzare circuiti fisicamente molto grandi.
- **Consumo elevato** l'alimentazione della bobina di eccitazione di un relè comporta consumi non indifferenti.
- **Ridotta affidabilità**: la logica presenta un alto numero di cablaggi interni e di relè ausiliari che ne diminuiscono di fatto l'affidabilità;
- **Il numero di contatti messi a disposizione da ciascun relè è limitato** a 2 o 3, per cui in molti casi risulta necessario installare più relè in parallelo;
- Nel caso siano necessarie modifiche o ampliamenti, si presentano **lunghi tempi di fermo impianto** sia per studiare che per realizzare le modifiche.

Tutti questi motivi spinsero la “**General Motors**”, che negli anni '60 era sicuramente uno dei più grossi utilizzatori d'automazione industriale al mondo (possedeva impianti di grandi dimensioni sparsi un po' per tutti gli Stati Uniti d'America), ad organizzare una gara per la progettazione di un dispositivo che potesse sostituire i quadri a relè nel controllo degli impianti.

- La “gara” fu vinta, agli inizi degli anni 70, dalla BEDFORD ASSOCIATES, che propose una macchina chiamata MODular Digital CONTroller (MODICON 084), il progenitore dei PLC.
- Tale nuova tecnologia si applicò inizialmente alle catene di montaggio automatizzate degli stabilimenti di produzione di automobili
- Il **MODICON** fu il primo plc ad essere commercializzato e prodotto in larga scala.
- Il motivo principale per il quale nacque l'esigenza del plc fu dunque la necessità di eliminare i costi elevati per rimpiazzare i sistemi di controllo complicatissimi basati su relè.
- Infatti le esigenze di innovazione delle ditte erano tali da richiedere continue variazioni dello schema relè, cioè continue modifiche alle macchine automatiche, con grossi rischi di errore ad ogni variazione e grosse spese.



Una delle persone che lavorò al progetto Modicon 084 fu **Dick Morley**, che è considerato il “padre” dei PLC.



Modicon 084 (1969)

Il marchio Modicon fu venduto nel 1977 alla Gould Electronics, poi venne acquistato dalla tedesca AEG e infine dalla Schneider Electric, l'attuale proprietaria.

## **Cos'è un PLC?**    Controllore a Logica Programmabile (Programmable Logic Controller)

Un PLC è un oggetto hardware componibile che, se opportunamente programmato, è in grado di gestire sistemi complessi, ricevendo in ingresso dati da sensori e/o comandi da operatori umani, elaborando dati e programmi al suo interno, scambiando dati attraverso reti di comunicazione, producendo uscite che vanno a comandare attuatori e/o dispositivi di segnalazione (allarmi, segnalazioni a pannello operatore...)

## **Dove viene usato un PLC?**

In ambito industriale, per il controllo di macchine automatiche, di impianti e di processi complessi

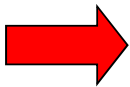
Il PLC nasce come elemento sostitutivo della logica cablata e dei quadri di controllo a relè .

## **In che settori industriali viene impiegato il PLC?**

- MACCHINE UTENSILI
- MACCHINE PER LO STAMPAGGIO
- MACCHINE PER IMBALLAGGIO
- MACCHINE PER IL CONFEZIONAMENTO
- ROBOT / MONTAGGIO
- REGOLAZIONE PROCESSI CONTINUI
- MACCHINE TESSILI
- SISTEMI DI MOVIMENTAZIONE/TRASPORTO
- .....

Attualmente i PLC vengono spesso usati anche per:

- Impianti di illuminazione di media/alta complessità (impianti di supermercati, stazioni, cinema...)
- Impianti domotici (di automazione della casa)

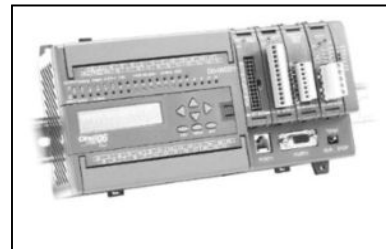


Dove si deve comandare un numero elevato di attuatori (anche solo dei semplici relè per accendere la luce), con possibilità di avere comandi differenziati e complessi (accendi solo le luci della fila di destra, solo le luci di sinistra, accendi le luci quando la luminosità esterna cala, accendi le luci se rilevi la presenza di qualcuno nella stanza) e di poter agevolmente variare il funzionamento del sistema (basta riprogrammare il PLC, non si deve rifare tutto l'impianto elettrico!)

I vantaggi del PLC rispetto alla logica cablata a relè

- il cablaggio di un quadro di automazione diventa elementare in quanto basta portare ciascun segnale individualmente sulla morsettiera del PLC;
- è “semplice” controllare eventuali anomalie o scoprire guasti;
- è possibile programmare centinaia di relè ausiliari, temporizzatori e contatori senza aumentare lo spazio occupato nel quadro.
- è possibile, tramite il software di programmazione, modificare il funzionamento dell'automatismo anche mentre questo è in funzione o con pause di pochi istanti
- è possibile adattare il funzionamento alle esigenze di produzione (ad es. per un cambio formato), sostituendo il programma;
- alta affidabilità del prodotto : i casi di guasto sono rari

## Com'è fatto un PLC?



Il PLC è un oggetto hardware componibile

- Ha bisogno di essere alimentato → modulo di alimentazione
- Ha bisogno di ricevere informazioni dai sensori → moduli di ingresso
- Deve poter comandare attuatori → moduli di uscita
- Deve eventualmente poter comunicare con altri PLC o PC → moduli di comunicazione
- Deve eseguire il programma, eseguire calcoli, supervisionare tutti i moduli... → modulo CPU
- Deve esserci una struttura meccanica che contiene tutti i moduli di cui si compone → rack (armadio, cestello)

+ eventualmente altri moduli con funzioni particolari, anche per gestire eventuali periferiche

Ing. Elena Mainardi

PLC

- 1) Modulo di alimentazione: fornisce alimentazione a tutti i moduli.

Tipicamente i PLC vengono alimentati a 24Vdc.

Esistono anche PLC collegabili direttamente alla tensione di rete, solitamente più ingombranti delle versioni a 24V



2) Moduli di ingresso: consentono al PLC di leggere lo stato dei sensori ad esso collegati.

### **Schede di ingresso digitali**

Sono utilizzate per il controllo di grandezze "digitali", cioè di tensioni a due valori (ad esempio 0V o 24V, oppure 0V 110V). Ogni scheda può gestire da 4 a 32, o 64 ingressi digitali differenti. I segnali dal campo vengono fatti arrivare con cavi elettrici fino alla morsettiera della scheda.

### **Schede di ingresso analogiche**

Permettono la rilevazione di grandezze elettriche il cui valore può variare entro un intervallo. Le grandezze in gioco sono in tensione o in corrente. Ad esempio sono disponibili schede di ingresso analogiche in corrente, con un intervallo variabile tra 4mA e 20mA. Molti produttori di PLC rendono disponibili schede con ingressi analogici per sonde di temperatura sia Pt100 che termocoppie, T,J,K ecc. Queste schede sono disponibili con varie risoluzioni (8-12-14-16 bit) e con 1 o più ingressi distinti disponibili in morsettiera o connettore.

Bisogna scegliere adeguatamente le schede ingresso/uscita rispetto ai dispositivi presenti nel sistema:

Ad esempio è inutile avere un PLC con convertitore A/D eccessivamente più preciso dello strumento di misura che genera il segnale analogico.

### 3) Moduli di uscita: consentono al PLC di comandare gli attuatori ad esso collegati

#### **Schede di uscita digitali**

Sono utilizzate per i comandi di attuatori digitali. Ad esempio un relè è un attuatore digitale, in quanto può avere soltanto due stati stabili: diseccitato, o eccitato. Altro esempio di attuatore è una valvola digitale a due stati: aperta, chiusa. Anche nel caso di schede di uscita digitali, si possono gestire da un minimo di 4 ad un massimo di 64 uscite digitali differenti

#### **Schede di uscita analogiche**

Permettono di controllare degli attuatori variabili. Possono essere in corrente o in tensione ed avere una determinata soluzione esprimibile in bit. Ad esempio è possibile comandare un motore elettrico tramite un inverter variandone la velocità, tramite la frequenza, da zero alla sua massima velocità. Oppure si può variare la velocità di un ventilatore, o la luminosità di una sorgente luminosa...

Le uscite possono essere costituite da transistor (per circuiti in corrente continua), triac (per circuiti in corrente alternata fino a 250V) ma normalmente si usano relè elettromeccanici con portata variabile tra 1 e 2A .

Per correnti superiori è necessario appoggiarsi a relè o contattori esterni, ma per azionare grossi contattori è necessario un relè intermedio.

Sul manuale del PLC si deve verificare l'esatta portata del contatto del relè che si deve confrontare con la corrente di spunto del contattore o della bobina (il carico non è mai collegato direttamente all'uscita).

## I moduli remoti

Quando l'impianto (o la macchina) è distribuito in una vasta area, può essere conveniente l'installazione di moduli remoti.

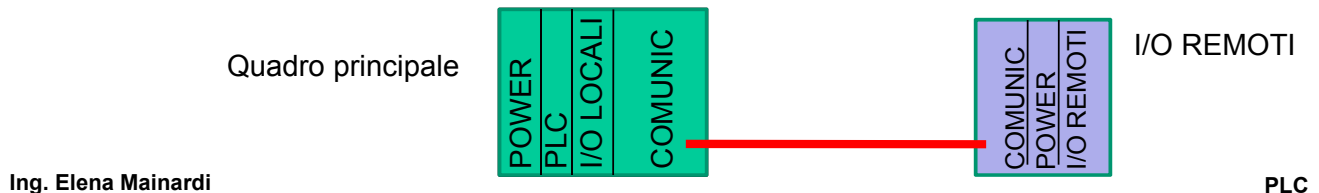
A seconda del costruttore esistono vari tipi di moduli : analogici/digitali, ingressi, uscite o moduli misti.

Normalmente questi moduli richiedono un modulo master o scanner che deve essere installato in uno slot libero del PLC

Il modulo scanner serve per dialogare con i moduli remoti, e solitamente ne può gestire fino ad alcune decine.

Con tali moduli, all'atto dell'installazione, si risparmia :

- spazio nel quadro del PLC
- la posa di un grande numero di cavi
- il cablaggio di numerosi fili con relativa complessità di collegamento, prova e collaudo
- maggior facilità nell'individuare i guasti



Ing. Elena Mainardi

PLC

## Moduli di conteggio veloce

Gli ingressi digitali del PLC non possono acquisire segnali che variano troppo velocemente nel tempo : generalmente un ingresso perchè venga letto dal programma deve permanere almeno 0.5 secondi.

Quando si ha bisogno di encoder o dispositivi di posizionamento similari che emettono impulsi ad alta velocità, tali dispositivi non si possono connettere direttamente agli ingressi del PLC, ma a dei moduli appositi, che poi comunicano con il PLC.

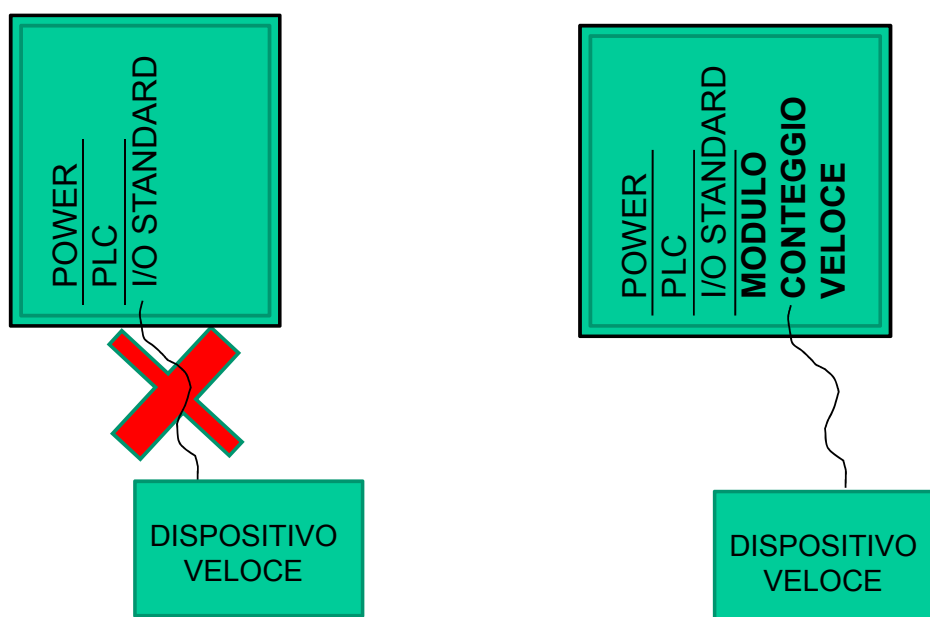
## Moduli di conteggio veloce

Vi sono alcuni modelli di PLC compatti provvisti di un ingresso per segnali digitali ad alta velocità (generalmente l'ingresso zero), già completo di contatore adatto per encoder controllabile dal software.

Nell'utilizzo di queste funzionalità è indispensabile confrontare la massima velocità degli impulsi forniti dall'encoder con la massima frequenza supportata dall'ingresso del PLC.

Nei casi in cui gli ingressi del PLC non siano adatti per acquisire segnali digitali ad alta velocità, cioè con frequenze nell'ordine del kHz, è necessario ricorrere a speciali moduli di ingressi digitali ad alta velocità con funzione di contaimpulsi

## Moduli di conteggio veloce



## Riassumendo

### Moduli I/O

I moduli Input sono convertitori di segnali elettrici che convertono i segnali provenienti dai sensori presenti sul campo in segnali aventi formato e livello tale da poter essere processati dalla CPU, al contrario i moduli Output convertono i segnali processati dalla CPU in segnali che possono essere compresi dagli attuatori.

Esempi di sensori e attuatori sono:

- Microinterruttori
- Sensori di prossimità
- Sensori di temperatura
- Elettrovalvole
- Relè

Tutti i moduli I/O sono otticamente isolati dal bus, assicurando sicurezza e immunità ai disturbi.

Tutti i moduli I/O sono configurabili da software

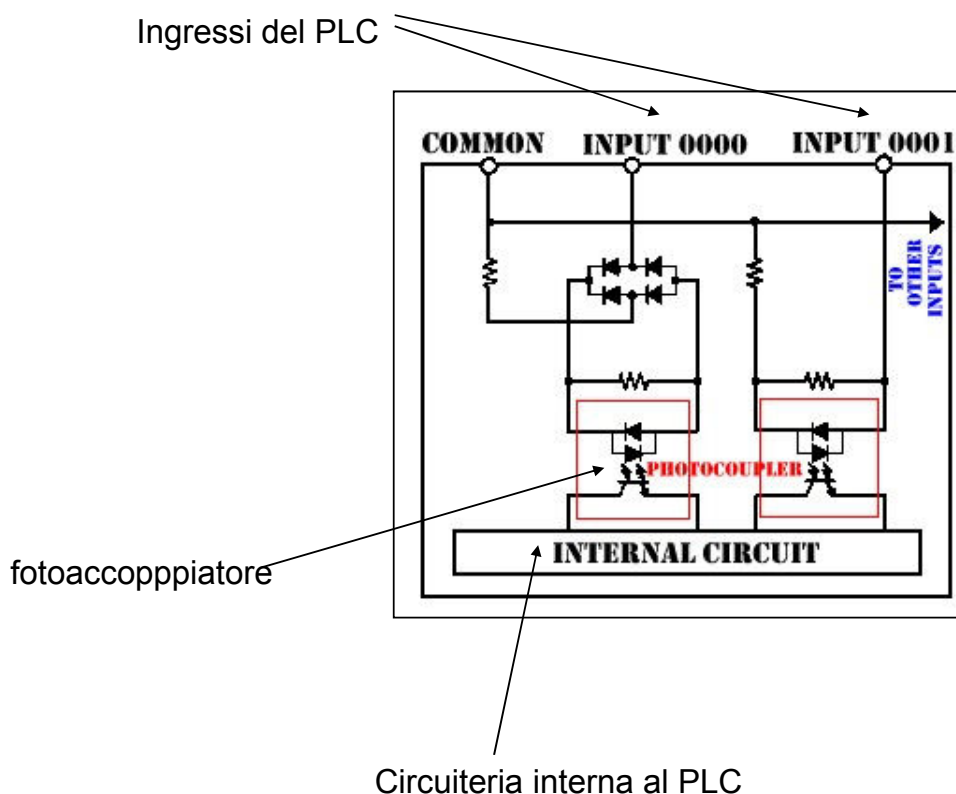
N.B. Normalmente i circuiti di ingresso ad un PLC sono costituiti da fotoaccoppiatori che dal lato "campo" gestiscono segnali a 24 Vcc, mentre dal lato interno a 5Vcc.

Per le uscite generalmente il microprocessore pilota delle bobine di relé o dei transistor "open collector".

Ing. Elena Mainardi

PLC

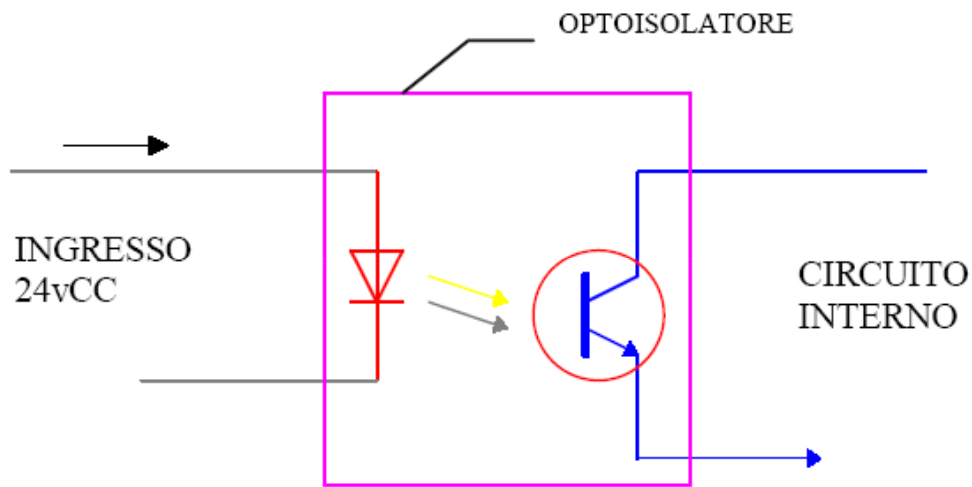
## Isolamento ottico



Ing. Elena Mainardi

PLC

## Isolamento ottico



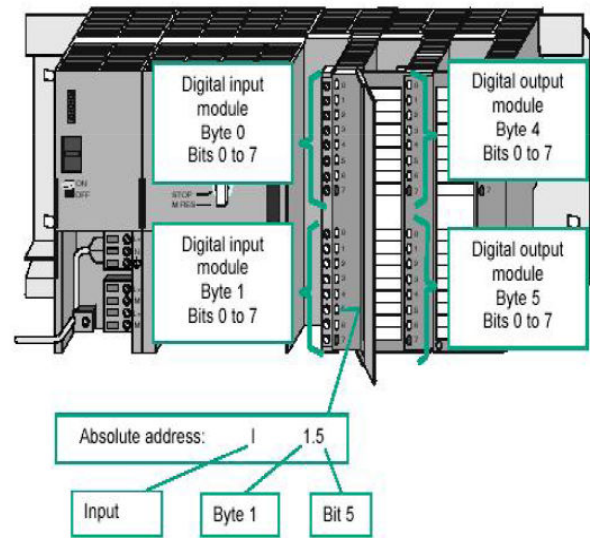
**4) Rack (o armadio o cestello):** è la struttura che racchiude e contiene tutti moduli che compongono il PLC , assicurandone la connessione meccanica e il collegamento elettrico

## Come indirizzo un I/O di un PLC? (cioè come identifico un ingresso o un'uscita fisica del PLC?)

→ Con un codice alfanumerico ←

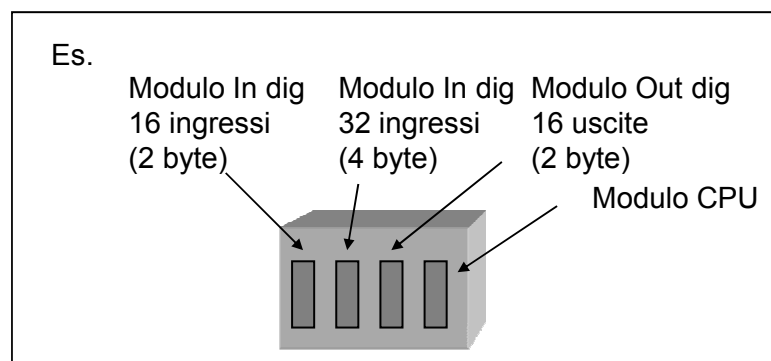
### Indirizzamento dell'I/O

Ogni in e out dei vari moduli digitali è un bit, associato per esempio ad un sensore ON/OFF o ad un relè. Quindi per identificare una singola uscita (O) o un singolo ingresso (I) basta specificare in che modulo mi trovo e a che numero di linea corrisponde il segnale di mio interesse.



Ing. Elena Mainardi

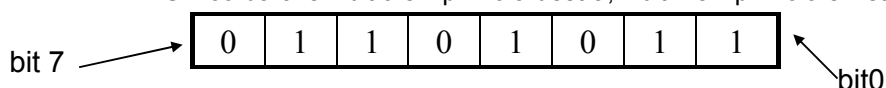
PLC



In fase di configurazione del sistema dirò:

- associo ai bytes del modulo d'ingresso più a sinistra i valori 0 e 1 (infatti il primo modulo ha 2 bytes). Quindi I1.3 identificherà il quarto bit del secondo byte del primo modulo d'ingresso
- associo ai bytes del secondo modulo d'ingresso i valori 2, 3, 4 e 5 (infatti ha 4 bytes). Quindi I4.6 identificherà il settimo bit del terzo byte del secondo modulo d'ingresso
- associo ai bytes del modulo d'uscita i valori 0 e 1. Quindi O0.0 identificherà il primo bit del primo byte del modulo d'uscita

N.B. Si ricorda che il bit 0 è il primo a destra, il bit 7 è il primo a sinistra



Ing. Elena Mainardi

PLC

5) **Modulo CPU** : è il modulo che governa tutto il PLC, su cui gira il programma. Può avere memoria interna e/o esterna

All'interno della CPU ci sono varie parti, tra cui:

- unità di gestione, ovvero informazioni di gestione del PLC stesso, impostate dal costruttore e trasparenti all'utente;
- archivio di temporizzatori e contatori funzionali all'operatività del PLC;
- memorie immagine del processo, cioè le informazioni in ingresso ed i comandi in uscita del processo;
- memoria utente, in cui vengono scritti i programmi che il PLC deve eseguire;
- interfaccia per il dispositivo di programmazione, che comunica con gli strumenti di programmazione;
- bus dati, comando, indirizzi per la veicolazione dei dati fra le varie parti e con l'esterno della CPU.

Il modulo CPU permette l'implementazione di funzioni logiche, di temporizzazione, matematiche, di comunicazione etc

### MEMORIA DI SISTEMA

Contiene il sistema operativo (firmware) del PLC, costituito da:

- routine di autotest iniziale
- dati del setup
- librerie

### MEMORIA DI PROGRAMMA

Contiene la sequenza di istruzioni (programma utente) che verrà eseguita dalla CPU

•Esistono diverse possibilità:

- RAM (per sviluppo e collaudo)
- EPROM (per programma definitivo)
- EEPROM (sia per fase di sviluppo che per versione definitiva)

### MEMORIA DATI

Contiene le informazioni relative alle varie aree dati interne e di I/O

•Poichè, in funzione delle elaborazioni del programma, è necessario effettuare sulle aree dati veloci operazioni di lettura e di scrittura, è possibile utilizzare soltanto memorie di tipo RAM

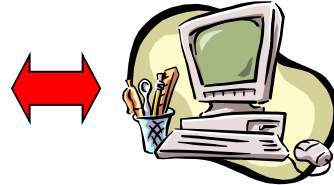
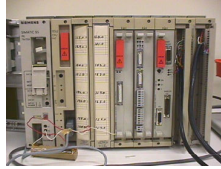


## 6) Moduli di comunicazione ( o moduli di rete)

Il PLC durante il suo funzionamento può comunicare con computer, con altri PLC oppure con altri dispositivi.

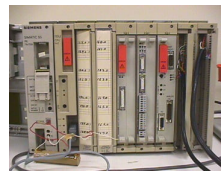
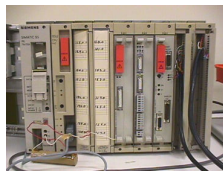
La comunicazione con computer e altri dispositivi avviene tramite tipi di connessione standard come:

- RS232
- RS422/RS485
- TCP/IP
- ...



La comunicazione con altri PLC o con sensori intelligenti avviene tramite protocolli standard, ad esempio:

- Profibus
- DeviceNet
- TCP/IP
- Modbus
- Modbus Plus
- Modbus TCP/IP
- Controlnet
- Powerlink
- CAN BUS



## L'interfaccia di rete

Con l'installazione di un modulo di rete, o interfaccia di rete, è possibile interconnettere assieme più plc, e fare in modo che questi si scambino informazioni.

Il modulo di rete normalmente non occupa uno slot nel rack, ma viene affiancato esternamente alla CPU

Il modulo di rete è spesso galvanicamente isolato (optoisolatore), in quanto deve garantire il perfetto funzionamento del plc anche con la presenza di sovratensioni anomale nella rete di interconnessione.

## Protocolli di comunicazione

Quando si connettono delle apparecchiature in rete è importante definire il protocollo di comunicazione.

Nelle applicazioni d'ufficio dei PC esistono vari tipi di reti e di protocolli; alcuni esempi sono la rete Ethernet e la RS 232.

In ambito industriale un protocollo molto usato è il RS485, che garantisce una buona immunità ai disturbi e può arrivare fino a 1200 metri.

Tale protocollo di comunicazione permette di interconnettere fino a 32 dispositivi ad una velocità di 19200 bps (bit per secondo).

Ogni dispositivo, che nella rete costituisce un nodo, viene poi identificato con un numero da 0 a 31 che deve essere assegnato in fase di configurazione.

Normalmente il nodo 0 è riservato al dispositivo di programmazione, il quale può essere connesso in qualunque punto della rete per programmare uno qualunque dei plc collegati.

Altri protocolli (bus di campo) molto usati sono PROFIBUS, CANOPEN, INTERBUS, CONTROLNET etc, che superano i 5-10 Mbps.

## Esempi di moduli di rete

- Moduli per rete Ethernet TCP/IP. Questi moduli rendono possibile la comunicazione tra controller e dispositivi su una rete Ethernet che usa il protocollo TCP/IP. Un modulo Ethernet può essere inserito in un sistema PLC preesistente e connesso ad una rete Ethernet con fibra ottica o cavo twistato.
- Moduli LonWorks. Questi moduli rendono possibile la comunicazione tra controller e dispositivi su una rete basata sulla tecnologia Echelons LonWorks
- Moduli Profibus DP. Questi moduli rendono possibile la comunicazione tra controller e dispositivi su una rete basata su Profibus DP
- Moduli ControlNet. Questi moduli rendono possibile la comunicazione tra controller e dispositivi su una rete basata su ControlNet

## 7) Moduli speciali

- Moduli di conteggio veloce
- Moduli di gestione interrupt veloci
- Modulo controllo assi
- Moduli PID
- Schede espansione memoria
- Moduli di backup
- Web Server TCP-IP
- Porte seriali
- ....

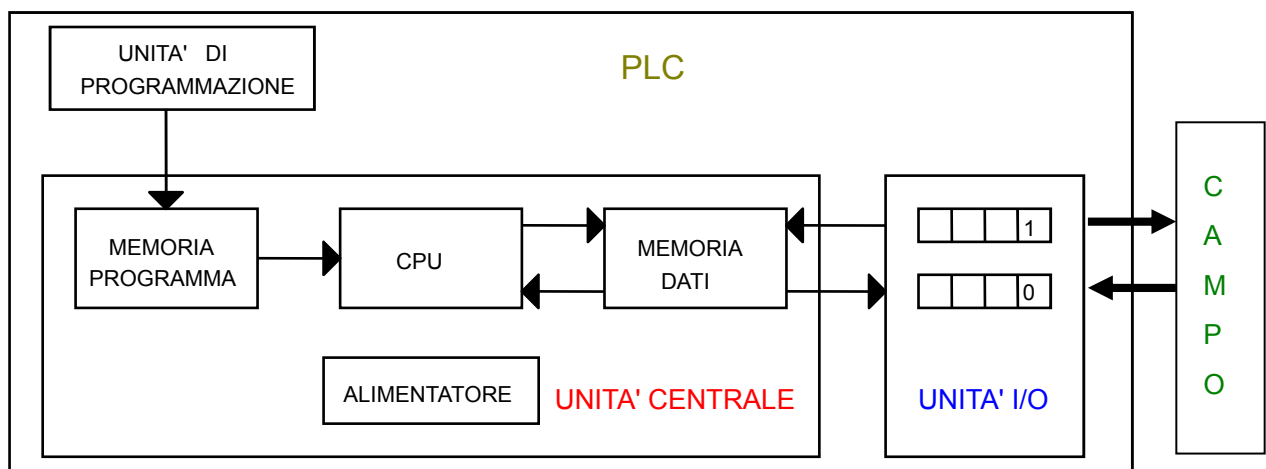
### Periferiche

Permettono il "colloquio" tra l'operatore ed il PLC

- Console di programmazione o PC
- Pannello operatore
- Sistema di supervisione o SCADA
- Interfaccia stampante
- Programmatore di EEPROM
- ....

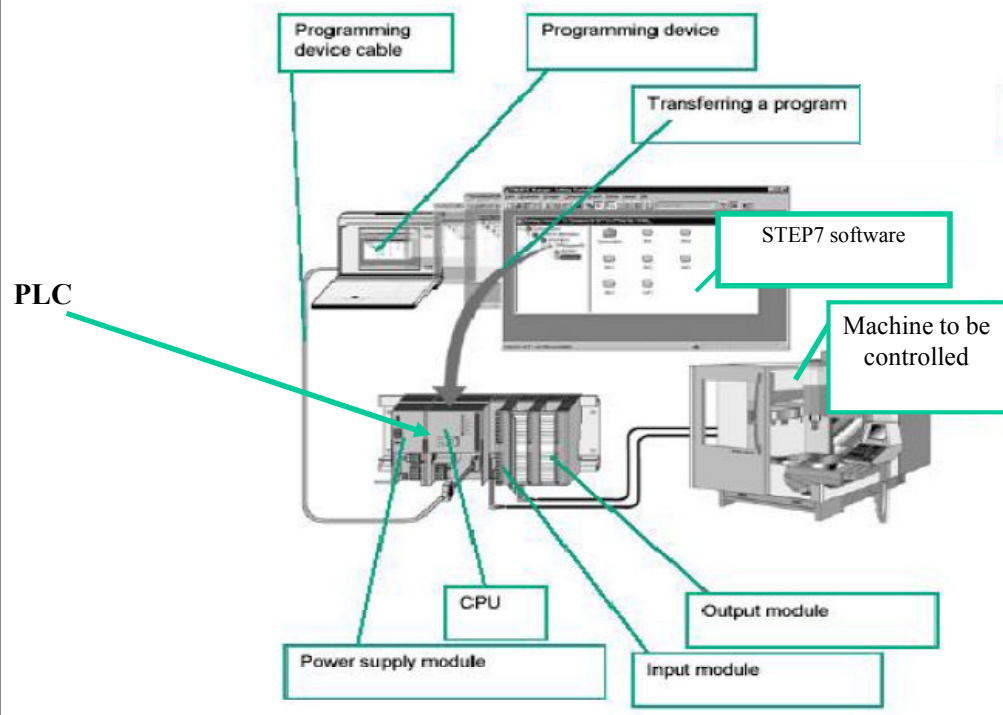
→ Occhio, possibile domanda d'esame: cos'è un PLC? Di che moduli si può comporre? ←

## HARDWARE DI UN PLC



...dove per campo s'intende lo spazio fisico in cui stanno i sensori e gli attuatori, o l'impianto che si deve controllare (infatti si parla spesso di reti di sensori di campo, o di bus di campo – detti fieldbus - etc)

## Architettura di un sistema a PLC



Ing. Elena Mainardi

PLC

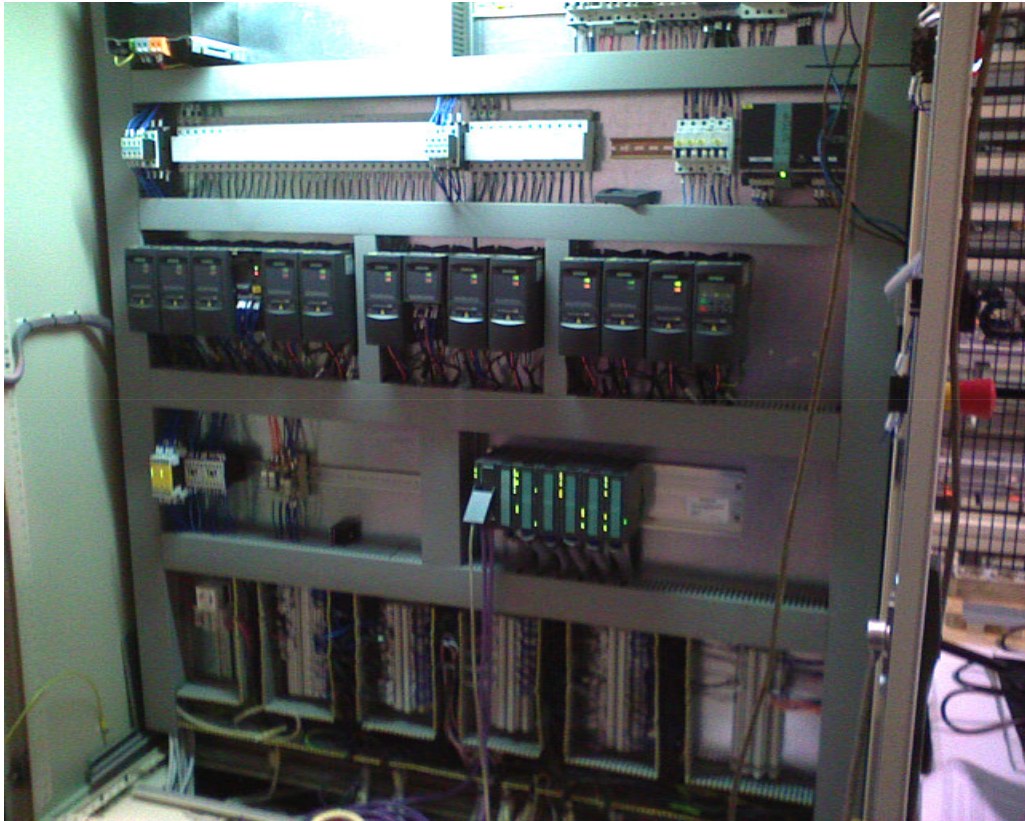
**L'armadio con il  
quadro elettrico**



Ing. Elena Mainardi

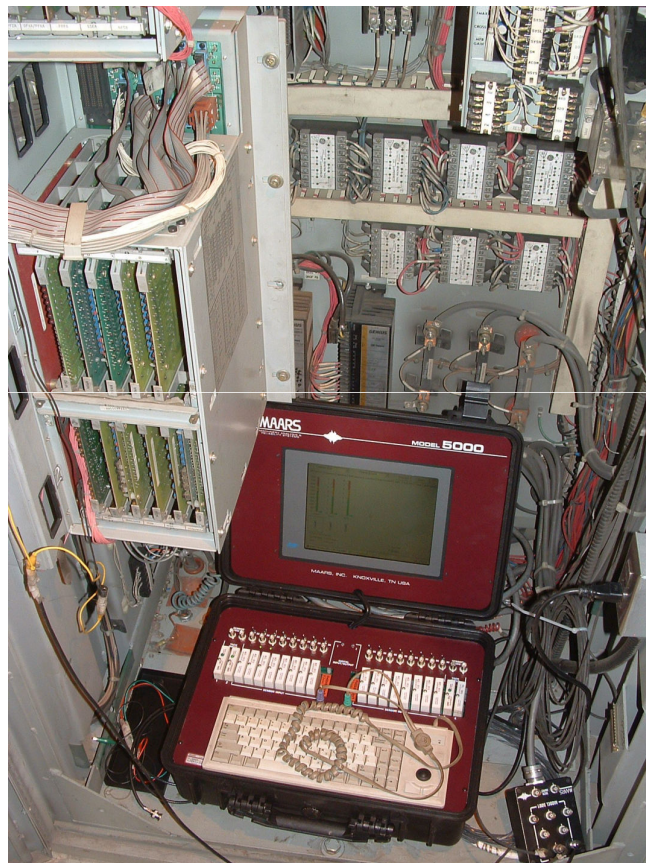
PLC





Ing. Elena Mainardi

PLC



Ing. Elena Mainardi

PLC



Ing. Elena Mainardi

PLC

## Chi produce PLC?

- |                   |                       |                     |
|-------------------|-----------------------|---------------------|
| ▪ Siemens         | ▪ Cutler Hammer       | ▪ Toyoda            |
| ▪ Omron           | ▪ Modcom              | ▪ SunX              |
| ▪ Rockwell        | ▪ Mitsubishi          | ▪ Vipa              |
| ▪ Beckhoff        | ▪ ABB                 | ▪ FF Automation     |
| ▪ B&R             | ▪ Eckelmann           | ▪ Horner            |
| ▪ Allen Bradley   | ▪ Cincinnati Milicron | ▪ Christ Elektronik |
| ▪ Telemecanique   | ▪ GE Fanuc            | ▪ Indotech          |
| ▪ Hitachi         | ▪ Honeywell           | ▪ Serra             |
| ▪ CGE             | ▪ Mitsubishi          | ▪ Amot              |
| ▪ Moeller         | ▪ Modicon             | ▪ Festo             |
| ▪ Lovato          | ▪ Reliance            | ▪ Keba              |
| ▪ National        | ▪ Square D            | ▪ Jetter            |
| ▪ Sprecher Schuch | ▪ Texas Instruments   | ▪ ...               |
|                   |                       | ▪ ...               |

Ing. Elena Mainardi

PLC



Un PLC è un oggetto hardware componibile che, se opportunamente programmato, è in grado di gestire sistemi complessi, ricevendo in ingresso dati da sensori e/o comandi da operatori umani, elaborando dati e programmi al suo interno, scambiando dati attraverso reti di comunicazione, producendo uscite che vanno a comandare attuatori e/o dispositivi di segnalazione (allarmi, segnalazioni a pannello operatore...)



Di fatto un PLC è una sorta di “piccolo” PC, ma con caratteristiche diverse

## Differenze PC - PLC

Caratteristica	PC	PLC
Spostamento di dati	> 500.000 kByte/sec	< 10 kByte/s
Dimensione dei programmi	> 10.000 kByte	< 10 kByte
Operazioni binarie tipiche	spostamento di 32 bit	operazioni su 1 bit singolo
Frequenza Microprocessore	> 1 GHz	< 100 MHz
Funzionamento tipico	8 ore al giorno	24 ore su 24
Immunità a disturbi elettrici	scarsa	elevata
Condizioni ambientali	interno climatizzato	da 0 a 55 °C
Programmazione	Compilata con linguaggi ad alto livello	diretta, praticamente in "linguaggio macchina"
Criticità temporali	No	Sì

Spostamento dati: quando si usa un PC il microprocessore al suo interno sposta centinaia di MegaByte al secondo attraverso i vari bus (basti pensare ai motori di grafica per i giochi, o alle applicazioni di video editing...), mentre nel PLC si spostano pochi Byte, se non addirittura bit singoli.

Criticità temporali: un PC è normalmente usato per programmi utente (web, posta elettronica, programmi di tipo office, grafica, giochi etc) dove non ci sono vincoli temporali stringenti. (se una mail la ricevo ora o tra 40 secondi non succede nulla)

Invece un PLC deve governare impianti complessi, dove magari più sistemi devono cooperare sincronizzandosi per trasportare-lavorare un pezzo, dove ci sono molte parti meccaniche in movimento etc → deve soddisfare a vincoli temporali assolutamente stringenti (se c'è una situazione di pericolo e l'allarme scatta ora o scatta tra 40 secondi, c'è una bella differenza!)

## Esistono anche dei mini o micro PLC per:

- Didattica
- Hobby
- Piccole applicazioni domestiche (apertura automatica porte, cancelli, riscaldamento...)
- Gestione acquari
- ...



Ing. Elena Mainardi

PLC

Questi mini PLC, spesso, hanno una modularità limitata o addirittura assente, ovvero hanno già a bordo la CPU e un numero limitato di ingressi e uscite analogiche e/o digitali (PLC compatti)

**Es. Modulo logico 26 I-O ST 24V Schneider Electric (linea ZELIO)**



Un altro esempio è la serie LOGO della Siemens

Ing. Elena Mainardi

PLC



### Modulo logico 26 I-O ST 24V Schneider Electric

#### Caratteristiche

- Impiego nel terziario/edilizia:
  - automazione dei sistemi d'accesso (cancelli, porte, barriere...),
  - automatismi dei sistemi di illuminazione,
  - automatismi dei compressori e dei sistemi di climatizzazione

#### • Programmazione: si può effettuare

- in modo diretto utilizzando i tasti funzione del modulo logico (linguaggio a contatti),
- su PC con il software Zelio Soft.

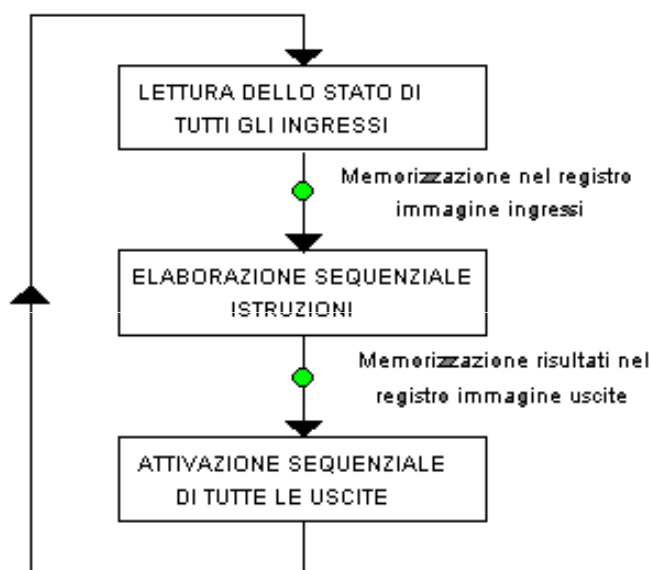
Su PC la programmazione può essere realizzata in linguaggio a contatti (LADDER), o in linguaggio blocchi funzione (FBD).

Numero d'I/O	26
Ingressi digitali	16
Di cui analogici 0..10V	6
Uscite a relè	0
Uscite a transistor	10
Orologio	si
Tensione alimentazione	24Vdc



Ing. Elena Mainardi

## Come funziona un PLC?

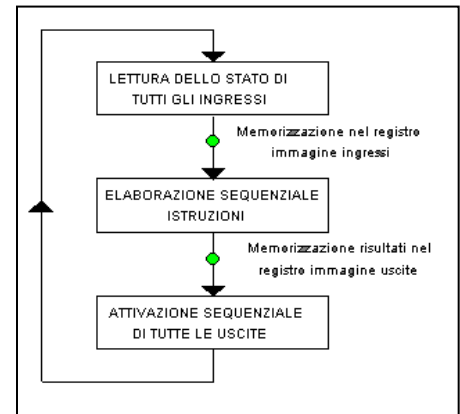


Una volta programmato, il PLC esegue ciclicamente, in continuazione, il programma → ELABORAZIONE CICLICA

## Come funziona un PLC?

### Funzionamento classico

- 1) La prima operazione che compie è la lettura di tutti gli ingressi (digitali, analogici, on board o su bus di campo, cioè provenienti da schede remote ovvero collegate al plc tramite una rete di comunicazione)
- 2) Dopo aver letto tutti gli ingressi, il loro stato viene memorizzato in una memoria definita REGISTRO IMMAGINE DEGLI INGRESSI.
- 3) A questo punto viene elaborato il programma. Le istruzioni di comando vengono elaborate in sequenza dalla cpu.
- 4) Al termine dell'elaborazione, il risultato viene memorizzato nel REGISTRO IMMAGINE DELLE USCITE.
- 5) Infine, il contenuto dell'immagine delle uscite viene scritto sulle uscite fisiche ovvero le uscite vengono attivate.



Ing. Elena Mainardi

PLC

## Il tempo di ciclo

Eseguire il programma utente dall'inizio alla fine vuol dire fare un ciclo (ciclo macchina)

ELABORAZIONE CICLICA: il tempo che il controllore impiega per una singola elaborazione del blocco di tutte le istruzioni del programma, è denominato tempo di ciclo (esecuzione del programma utente una volta sola). Il tempo di ciclo si può calcolare sommando il tempo necessario ad ogni istruzione

- Eventi molto rapidi possono essere "persi" tra un ciclo e il successivo
- Il tempo di ciclo di esecuzione del PLC costituisce un limite alla rapidità di risposta del PLC ad un allarme

Ing. Elena Mainardi

PLC

## Il tempo di ciclo

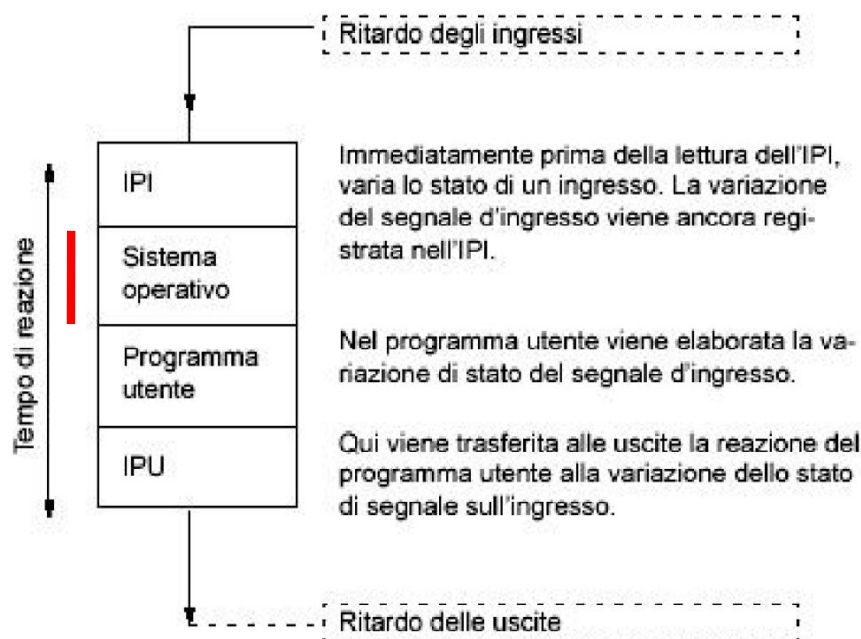
Il ciclo di elaborazione non ha durata costante

Ad esempio in ogni programma ci sono delle istruzioni condizionali (if..then..) tali per cui se si verifica una certa condizione si esegue un certo codice, altrimenti si esegue un altro codice → non in tutti i cicli eseguo sempre lo stesso codice, quindi lo stesso numero e tipo di istruzioni → da un'esecuzione all'altra del ciclo di programma può cambiare il tempo di ciclo

Il tempo di ciclo è costantemente controllato da un apposito sistema definito watchdog, che al superamento del tempo massimo preimpostato causa un allarme che pone il plc nello stato di STOP.

## Ciclo di esecuzione del PLC

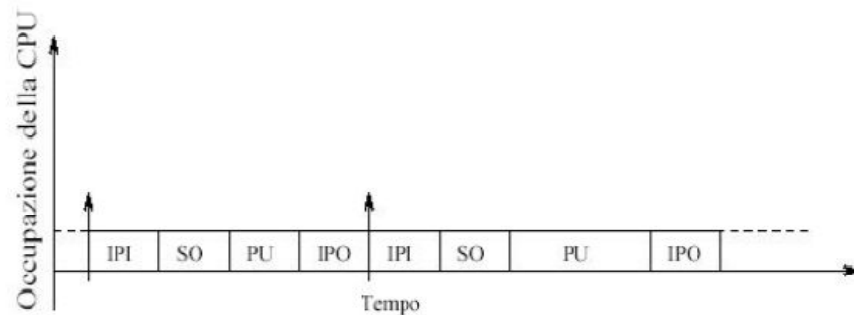
### Funzionamento classico



**IPI : Immagine del processo in ingresso**  
**IPU : Immagine del processo in uscita**

## Attività del sistema operativo

- Aggiornamento dell'immagine degli ingressi.
- Aggiornamento dell'immagine delle uscite.
- Esecuzione del programma utente.
- Gestione dei programmi utente attivati su interruzione:
  - Interruzione generata da un evento
  - Interruzione generata da un temporizzatore (interruzione ciclica)



**SO : Sistema operativo**

**PU : Programma utente**

**IPI : Immagine del proc di ingresso**

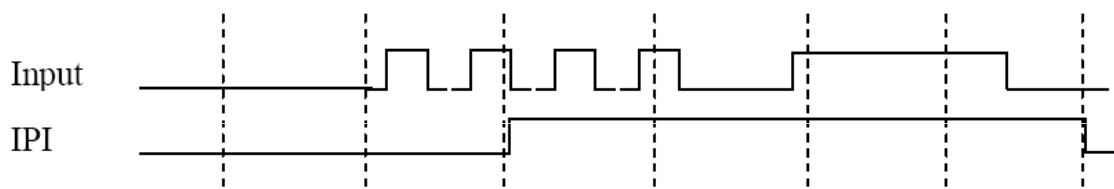
**IPO: Immagine del proc di uscita**

Ing. Elena Mainardi

PLC

## IMMAGINE DEGLI INGRESSI

Il ciclo di esecuzione del PLC basato su questo schema ha il vantaggio di "congelare" gli ingressi per tutta la durata dell'elaborazione del programma utente e quindi non si corre il rischio che alcune parti di programma eseguano elaborazioni su variabili di ingresso non omogenee.



Es. Se all'istante di "congelamento" degli ingressi il sensore A dà un'informazione numerica pari a 100 e durante il ciclo di esecuzione il valore del sensore cambia, il programma, per quel ciclo, vedrà sempre e comunque 100, e tutte le istruzioni che faranno riferimento al sensore A useranno come valore 100, fino alla fine di quel ciclo. Altrimenti potrebbe succedere, se il valore non fosse "congelato", che per esempio un'istruzione usi 100, e due istruzioni dopo, se viene ancora chiamato in causa il sensore A, l'istruzione usi il valore 101, se il sensore ha cambiato il proprio valore. I dati non sarebbero omogenei!!!

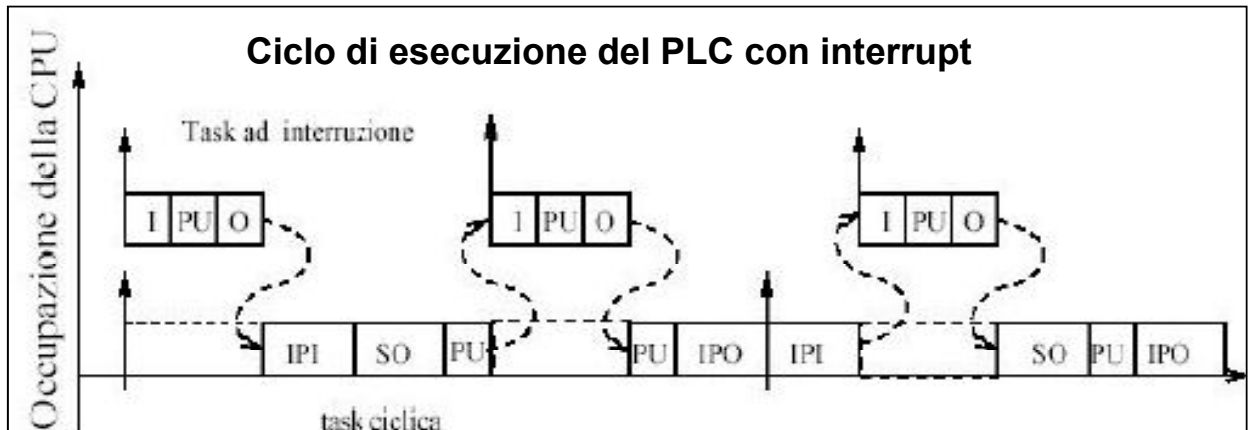
Ing. Elena Mainardi

PLC

## Variazioni sul ciclo di elaborazione del PLC

Esecuzione di task in parallelo al ciclo di elaborazione normale con priorità più elevate:

- Task eseguite periodicamente
- Task eseguite in risposta ad un segnale esterno (interruzione)



Ing. Elena Mainardi

PLC

## Come funziona un PLC?

Il “congelamento” degli ingressi e delle uscite è il modo più classico con il quale un generico PLC lavora.

Ma nella realtà non sempre è così.

Possiamo distinguere diversi tipi di ciclo.

PLC di marche differenti possono lavorare in modo differente, e questo comporta che il programmatore tratti ingressi e uscite in modo diverso.

## Funzionamento classico

### Ciclo sincrono in ingresso e sincrono in uscita

In questo caso il PLC acquisisce tutti gli ingressi all'inizio del ciclo e li memorizza in un'area di memoria che viene denominata immagine degli ingressi. Tale immagine rimane invariata per tutto il ciclo del PLC che, anche se gli ingressi dovessero cambiare, non li considera. Allo stesso modo, durante l'elaborazione del programma le uscite vengono calcolate e memorizzate in un'area denominata immagine delle uscite: solo alla fine dell'esecuzione del ciclo macchina corrente esse vengono effettivamente inviate sul campo.

Questo tipo di ciclo ha come vantaggio proprio il fatto di essere completamente sincrono e come tale di lasciare invariati ingressi e uscite durante un ciclo macchina. Questa caratteristica risulta essere particolarmente utile in fase di ricerca guasti e collaudo, in quanto essa permette in ogni istante di fornire una descrizione completa dei valori delle variabili coinvolte nel funzionamento del PLC.

Per contro questo tipo di ciclo, proprio per le sue caratteristiche, comporta ritardi nell'acquisizione degli ingressi e nel comando delle uscite più elevati di tutti gli altri che verranno analizzati.

## Altre modalità di funzionamento

### Ciclo asincrono in ingresso e sincrono in uscita

In questo ciclo le uscite sono trattate come nel caso precedente, mentre non esiste l'immagine degli ingressi. Pertanto, quando durante l'esecuzione del programma utente si fa riferimento a uno specifico ingresso, questo viene letto direttamente dal campo. Si noti che questa operazione non richiede più tempo di un accesso all'immagine degli ingressi vista nel caso precedente in quanto, dato che gli ingressi fisici del PLC sono mappati nella sua memoria, l'acquisizione di un ingresso consiste, anche in questo caso, in una operazione di accesso alla memoria.

Questo tipo di ciclo presenta, a parità di numero di istruzioni del programma utente, tempi di esecuzione più bassi rispetto al caso precedente in quanto manca la parte di elaborazione dell'immagine degli ingressi.

Quando si usa un PLC con ciclo asincrono in ingresso bisogna tenere conto che, proprio per le sue caratteristiche, durante un ciclo macchina una qualsiasi routine software se eseguita più volte sempre con gli stessi ingressi, può dare uscite diverse (ciò è ovvio poiché il valore degli ingressi dipende dagli istanti di acquisizione anche nello stesso ciclo macchina). Questo non è un inconveniente, tuttavia si tratta di un comportamento che va tenuto in debita considerazione, dato che esso potrebbe dare luogo a deduzioni errate sul funzionamento del PLC e quindi dell'impianto.

## Altre modalità di funzionamento

### Ciclo sincrono in ingresso e asincrono in uscita

In questo tipo di ciclo il PLC non elabora l'immagine delle uscite, le quali vengono inviate sul campo direttamente quando si incontrano le relative istruzioni di comando nel programma utente. Si tratta, anche in questo caso di un ciclo con bassi tempi di esecuzione, tuttavia è necessario prestare attenzione alla programmazione del PLC in quanto si potrebbero verificare inconvenienti quando la stessa uscita viene comandata più volte nello stesso ciclo macchina. Infatti, qualora i valori assegnati risultassero diversi, e l'intervallo di tempo tra un comando e il successivo fosse troppo breve, potrebbero verificarsi delle oscillazioni sui dispositivi comandati.

### Ciclo asincrono in ingresso e asincrono in uscita

E' il ciclo con i tempi di reazione più bassi in assoluto, esso incorpora tutti i pregi e i difetti dei cicli asincroni. Un esempio del funzionamento di questo ciclo è dato dalla sequenza:

*leggi ingresso sul campo  
usa il valore in una data funzione  
elabora valore di un'uscita  
invia uscita sul campo*

Ing. Elena Mainardi

PLC

## Altre modalità di funzionamento

- Modalità di lettura indipendenti dall'immagine degli ingressi possono riguardare anche gli ingressi "veloci" (che vengono letti molto frequentemente), ingressi legati alle routine di interrupt (che possono intervenire in qualsiasi momento del ciclo di scansione), e gli I/O analogici.
- Quando non si utilizzano le immagini di processo, la lettura di un ingresso viene effettuata nel momento stesso in cui viene invocata, e la modifica dello stato di una uscita si ripercuote immediatamente sul campo. L'organizzazione del programma è più libera e risulta possibile strutturare il ciclo con punti di attesa e feedback senza problemi, con una facile e naturale traduzione sia di strutture combinatorie sia di strutture sequenziali.
- Naturalmente a fronte di una più elevata efficienza e flessibilità di programmazione, si ha una più difficile gestione e, se non vengono adottate opportune tecniche informatiche, è più probabile ottenere sistemi poco robusti rispetto a errori di programmazione.

Ing. Elena Mainardi

PLC

## Altre modalità di funzionamento

Ad esempio con i PLC Saia-burgess si ha la possibilità di accedere direttamente ai segnali di I/O senza passare attraverso l'immagine del processo, con una notevole riduzione dei tempi di reazione

Anche i PLC Rockwell Allen – Bradley funzionano di default in modo asincrono.

La famiglia Logix5000 ha come caratteristica di leggere gli ingressi continuamente ed in tempo reale (del processore e non del processo), quindi durante le scansione può avvenire che cambi lo stato degli ingressi ed uscite. Gli I/O vengono rinfrescati indipendentemente dalla CPU, è possibile inconsistenza dei dati durante il ciclo di esecuzione del programma

E' da notare però che ingressi asincroni (rinfrescati in modo indipendente dall'esecuzione del ciclo di programma) possono essere resi sincroni tramite istruzione CPS (Copy Synchronous )

Per contro, alcuni PLC che gestiscono gli I/O in modo sincrono possono tuttavia forzare i valori delle uscite in modo che esse siano subito attuate, e non alla fine del ciclo (con istruzioni tipo IMMEDIATE OUT)

→ Occhio, possibile domanda d'esame: come funziona un PLC? Che operazioni cicliche esegue? Quali sono i possibili funzionamenti alternativi? ←

## Secondo quali tecniche si programma un PLC?

In generale il software di una macchina automatica descrive una regola di comportamento che mette in relazione i segnali dei sensori (ingresso) con i segnali per gli attuatori (uscite).

Es.

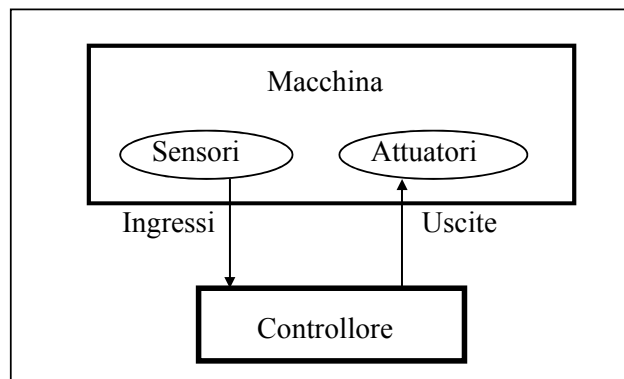
Se temperatura termocoppia > 40°C → allora aziona  
motore ventilatore

ingresso                      uscita



## Posso organizzare il mio software in due modi diversi

- O pianifico le azioni da eseguire (uscite) in base a condizioni rilevate sul processo o interne al controllore → struttura condition-driven
- O pianifico le azioni in base allo "stato" di funzionamento del processo → struttura state-driven



Ing. Elena Mainardi

PLC

## Approccio condition-driven

- Il programma di controllo comanda determinate azioni sul processo in base al presentarsi di determinate condizioni:
  - Eventi segnalati da appositi sensori a bordo macchina.
  - Eventi interni al controllore (timer, contatori...).

- La struttura del programma si può ricondurre ad una sequenza di istruzioni:

If <condizione> THEN <azione>

Ad esempio:

**IF** FOTOCELLULA **THEN** AVANZAMENTO MOTORE

**IF** TIMER.Q **THEN** STOP MOTORE

Ing. Elena Mainardi

PLC

## Approccio state-driven

- Il programma di controllo comanda determinate azioni sul processo in base allo stato di evoluzione del processo.
- Lo stato di evoluzione del processo dipende dalla sequenza degli ingressi che si sono presentati fino a quel momento.

### Definizione formale di stato:

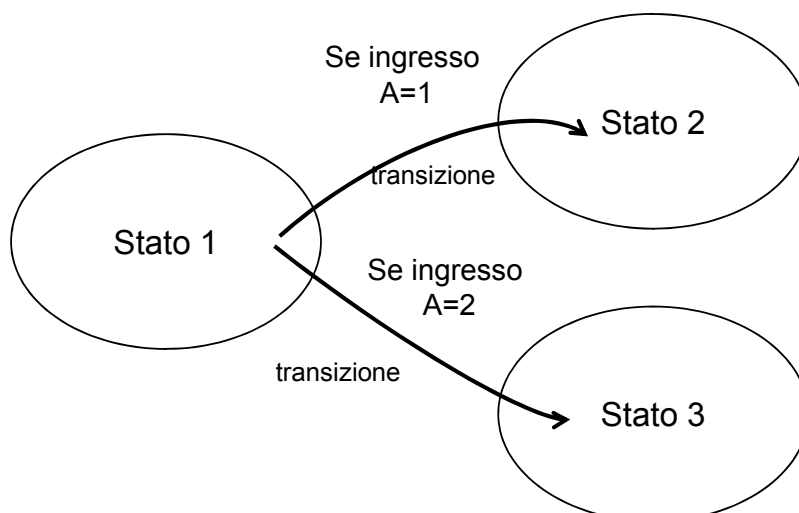
“Uno stato è una condizione di funzionamento del processo che persiste per un periodo di tempo significativo ed è distinguibile da ogni altra condizione di funzionamento”

Una condizione di funzionamento si dice distinguibile da ogni altra se differisce:

- Negli eventi che vengono accettati nello stato
- Nelle transizioni che si diramano dallo stato
- Nelle azioni che vengono eseguite

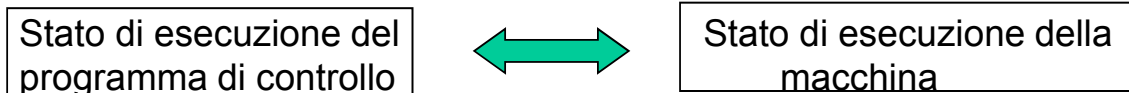
## Transizioni

Una transizione è una risposta ad un evento che causa un cambiamento nello stato del processo



## Modellare sequenze di processo mediante Macchine a stati finiti

- I sistemi automatici sono rappresentabili mediante passi sequenziali descrivibili mediante sistemi a stati finiti.
- I sistemi di controllo devono replicare queste proprietà in modo che:



- Lo stato del sistema di controllo deve sempre essere congruente con quello del processo controllato.

In generale un software industriale interagisce con un sistema dinamico modellabile mediante una sequenza di stati di lavorazione. Lo stato interno del software deve essere congruente con lo stato del processo controllato

**Esempio:** procedure di avviamento a caldo ed avviamento a freddo delle macchine automatiche.

- **Avviamento a freddo (cold restart).**

Procedura di partenza della macchina completa. Tale procedura è utilizzata quando lo stato della macchina al momento della ripresa non è congruente con lo stato in cui la macchina si è fermata. (perché per esempio mentre la macchina è ferma spostato a mano alcune componenti meccaniche, pulisco gli ingranaggi e via dicendo)

- **Avviamento a caldo (warm restart).**

Procedura di partenza della macchina in cui vengono saltate alcune procedure di avvio, in quanto lo stato raggiunto dalla macchina prima dello stop è stato mantenuto

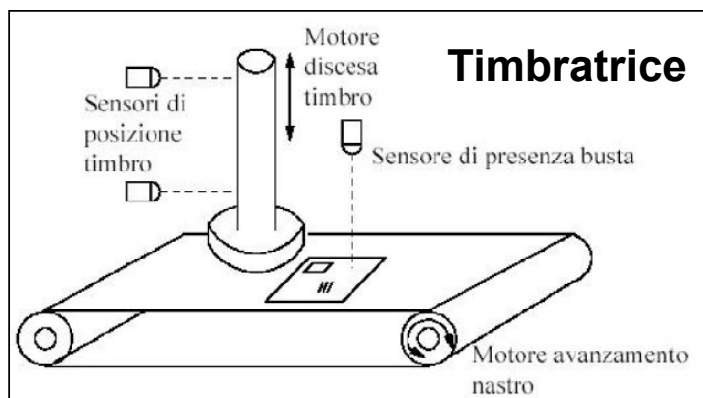
### Esempio di programmazione condition-driven e state-driven

La timbratrice deve ripetitivamente salire e scendere per effettuare timbri su buste che vengono poste sul supporto inferiore da un'altra macchina.

segnale di “busta presente” → avvia motore in direzione di discesa fino a che il sensore di presenza inferiore segnala che il pistone di timbratura è arrivato al fine–corsa.

→ il motore deve invertire la sua corsa e far risalire il pistone fino a che non viene attivato il sensore di fine corsa superiore

→ A questo punto la macchina timbratrice segnala che la busta può essere rimossa dalla sede di lavorazione.



Ing. Elena Mainardi

### Progetto del controllo Condition Driven

- Il progetto richiede di elencare i possibili eventi diagnosticabili dai sensori (segnali di ingresso)

ARRIVO\_BUSTA  
FINE\_CORSA\_BASSO  
FINE\_CORSA\_ALTO

- Per ciascun evento (o combinazione di eventi) si definisce un'azione da impostare sulla macchina:

MOTORE\_NASTRO  
MOTORE\_TIMBRO  
MOTORE\_TIMBRO\_DIREZIONE

## Progetto del controllo Condition Driven

```

IF ARRIVO_BUSTA THEN
    MOTORE_NASTRO := FALSE;
    MOTORE_TIMBRO := TRUE;
    MOTORE_TIMBRO_DIREZIONE := TRUE;
END_IF;

IF FINE_CORSA_BASSO THEN
    TIMBRATO := TRUE;
    MOTORE_TIMBRO_DIREZIONE := FALSE;
END_IF;

IF FINE_CORSA_ALTO AND TIMBRATO THEN
    TIMBRATO := FALSE;
    MOTORE_TIMBRO := FALSE;
    MOTORE_NASTRO := TRUE;
END_IF;

```

Ing. Elena Mainardi

PLC

## Progetto del controllo state-driven

- Analisi degli stati operativi del processo, che vengono identificati in base alla descrizione del funzionamento della macchina.
- Per definire uno stato operativo di funzionamento ci chiediamo:
  - Quali sono le azioni che la macchina compie in un determinato momento?
  - Tali azioni sono differenti da quelle che esegue in altri momenti del ciclo produttivo?

### Esempio

1. La timbratrice è in attesa dell'arrivo della busta. Non viene eseguita nessuna azione, se non l'azionamento del motore del nastro (l'azione eseguita è quindi l'attesa che arrivi la busta successiva).
2. Il timbro scende per timbrare la busta.
3. Il timbro sale per raggiungere la posizione di riposo (dopo aver timbrato)

Ing. Elena Mainardi

PLC

## Progetto del controllo state-driven

### Osservazione

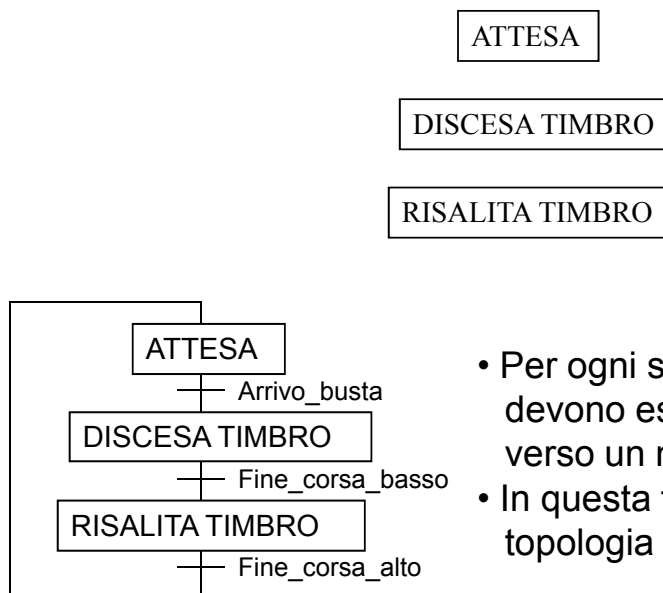
→ In questa fase (identificazione degli stati operativi) non specifichiamo l'implementazione delle operazioni che il processo deve eseguire in ciascuna fase, ma solo la sequenza logica delle operazioni.

→ Specifichiamo cosa deve fare la macchina e non come lo fa!

→ La separazione dell'implementazione dalla progettazione concettuale è un buon metodo per suddividere il problema a blocchi rendendone più facile la comprensione e, quindi la ricerca della soluzione.

## Progetto del controllo state-driven

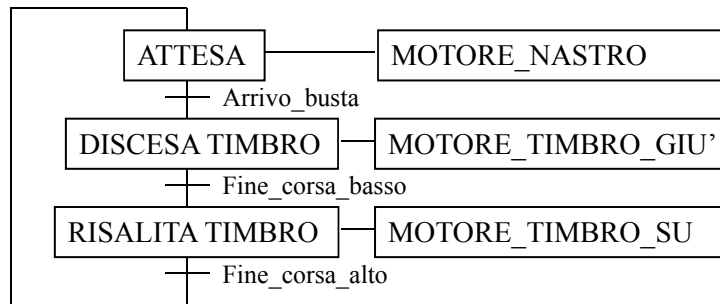
- Il progetto si imposta analizzando quali sono le fasi operative della macchina.
- Ad ogni fase operativa distinta associamo uno stato.



- Per ogni stato: quali sono gli eventi che devono essere verificati per evolvere verso un nuovo stato.
- In questa fase viene definita quindi la topologia della rete.

## Progetto del controllo state-driven

- Determiniamo le azioni che debbono essere eseguite ad ogni passo.



### Osservazioni

Una volta costruito il diagramma degli stati, è importante verificare che:

- Sia stato indicato lo stato iniziale, quello da cui parte la macchina a stati quando il software di controllo viene resettato.
- Non vi siano stati da cui non diparte alcun collegamento (stati non connessi)
- A parte i soli stati iniziali, non vi siano stati a cui non arrivi alcun collegamento (stati non raggiungibili).

## Progetto del controllo state-driven: costruzione incrementale del diagramma degli stati

La costruzione del diagramma degli stati può avvenire in modo incrementale:

1. Si individua lo stato (gli stati) iniziale.
2. Si individuano le transizioni a partire dallo stato iniziale.
3. Si procede in modo incrementale individuando stati e transizioni, ripercorrendo l'evoluzione sequenziale della macchina.

E' così che conviene risolvere gli esercizi!!!!



## Le azioni ripetute

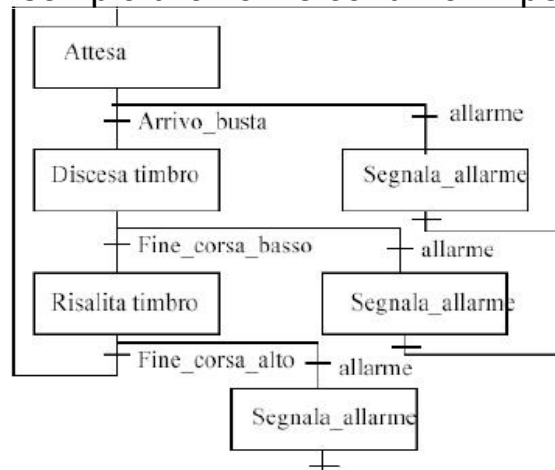
1. A volte vi sono azioni che vengono ripetute in molti (a volte tutti) gli stati dell'automa.
2. Un esempio tipico è costituito dagli allarmi o dalle eccezioni di funzionamento
3. In tal caso è possibile:

- Trattare le gestioni di questi eventi in un programma separato, scritto con approccio event-driven (quando le azioni o le condizioni sono le stesse in tutti gli stati, tipicamente per gli allarmi).

## Le azioni ripetute

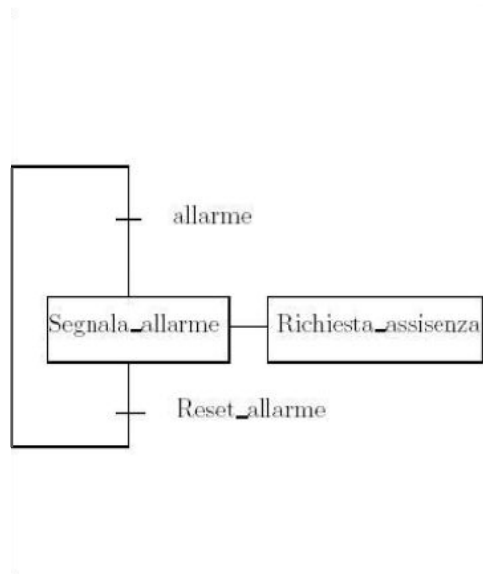
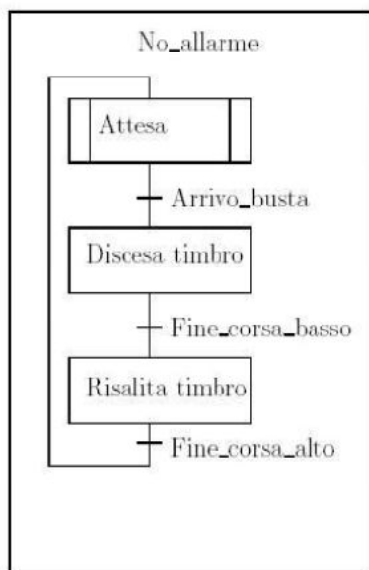
Se in una macchina a stati finiti esistono azioni che si ripetono su più stati, significa che tali stati non sono significativi per quella azione, e quindi essi debbono essere riuniti in un solo stato associato a quella azione. In altre parole, si è “erroneamente” (in senso logico e non formale) suddiviso su più stati una azione indivisa, che sarebbe più corretto assegnare ad uno stato ben preciso

Esempio di azioni e condizioni ripetute



## Le azioni ripetute

Sarebbe meglio...



## Programmazione condition-driven vs. state-driven

Vantaggi dell'approccio state-driven:

- Decomposizione della complessità della macchina.
- Semplicità nella analisi funzionale del processo.
- Facilita l'operazione di verifica del software.
- Risulta agevole valutare le possibili eccezioni di funzionamento del processo.

## Vantaggi dell'approccio state-driven

La programmazione state-driven consente di strutturare in maniera logica e ben organizzata una sequenza di controllo, isolando stati logici di funzionamento della macchina in cui solo un sottoinsieme di sensori ed attuatori sono di interesse ai fini della esecuzione della produzione.

Questo consente di avere:

- Decomposizione della complessità della macchina. In ogni stato logico di funzionamento solo un sottoinsieme di sensori ed attuatori sono di interesse, per cui possiamo semplificare notevolmente la logica di comando legandola allo stato in elaborazione al momento.
- Semplicità nella analisi funzionale del processo. La scomposizione in stati di funzionamento facilita l'operazione di analizzare le fasi operative del processo e quindi agevola la stesura delle specifiche del software.
- Facilita l'operazione di verifica del software. Risulta infatti agevole eseguire la verifica del software in quanto è sufficiente seguire i passi del programma e parallelamente gli stati di funzionamento della macchina.
- Risulta agevole valutare le possibili eccezioni di funzionamento del processo.
- Concentrandosi su una condizione operativa della macchina è più facile analizzare i possibili comportamenti dell'automatismo nel caso di condizioni anomale di funzionamento o addirittura nel caso di possibili guasti.

Ing. Elena Mainardi

PLC

## Vantaggi dell'approccio condition-driven:

- Migliore gestione di sistemi ad eventi non sequenziali. Nel caso di processi non sequenziali, l'approccio secondo la gestione diretta di eventi è più semplice. L'esempio più importante consiste nella gestione degli allarmi sul processo che hanno una caratteristica prettamente non sequenziale ma combinatoria.
- Gestisce intrinsecamente eventuali azioni manuali. Nel caso in cui l'operatore possa essere chiamato ad eseguire operazioni manuali (inceppamento di materiale nel processo etc . . . ) può accadere che lo stato della macchina non sia congruente con quello della macchina a stati del controllore, a causa proprio di manipolazioni del processo avvenute senza informare il sistema di controllo. In questo caso l'approccio ad eventi è più robusto dell'approccio a stati.

→ Occhio, possibile domanda d'esame: differenze, vantaggi e svantaggi delle programmazioni state driven e condition driven ←

Ing. Elena Mainardi

PLC

## Normative per i PLC

### Cos'è una norma?

→ Una norma è un insieme di specifiche che servono a standardizzare un certo ambito.

Es. Norma edile: le case per civile abitazione devono avere muri spessi non meno di tot centimetri, devono avere tot luminosità per tot metri quadri di spazio, devono avere un certo isolamento termico per il risparmio energetico....

### Perché una norma?

→ Anche nell'ambito dell'automazione industriale si è sentita l'esigenza di normare, per avere una certa standardizzazione e quindi una certa portabilità del software (cioè avere la possibilità di far girare lo stesso programma su PLC di ditte diverse)

## Chi emette le norme?

Le normative nel mondo sono emesse da due enti normatori :



- l' **ISO**, che emette norme di orientamento generale in quasi tutti i campi (inizialmente si occupava prevalentemente del settore meccanico)
- l'**IEC**, che emette norme nel settore Elettrico ed Elettronico

In Europa questa struttura si riflette sui due comitati tecnici "equivalenti" :



- il **CEN**, che lavora al pari con l'ISO,
- il **CENELEC**, che lavora al pari con l'IEC.

In Italia la situazione è praticamente la stessa con questi due enti :



- l' **UNI**, che lavora al pari con l'ISO, ed il CEN
- il **CEI**, che lavora al pari con l'IEC ed il CENELEC per le norme "elettriche"

## Lo Standard IEC 1131

- La International Electrotechnical Commission IEC istituisce nel 1988 diverse Task Force per elaborare uno standard per i controllori programmabili
- Nel 1992 viene emesso il documento IEC 1131, diviso in cinque parti:
  - Parte 1: Definizioni generali
  - Parte 2: Hardware
  - Parte 3: Linguaggi di programmazione
  - Parte 4: Linee guida per l'utente
  - Parte 5: Comunicazioni
- Nel 1993 le prime tre parti diventano Standard Internazionali (aggiungendo un 6 a 1131)

Descrizione	IEC	Cenelec	CEI
Controllori Programmabili - Informazioni Generali	1131-1	EN 61131-1	65-23
Controllori Programmabili - Linguaggi di Programmazione	1131-3	EN 61131-3	65-40

Esiste poi un'altra norma IEC, la 61499, più recente della 61131 e che ne apporta alcune modifiche

Ing. Elena Mainardi

PLC

## Norme di tipo software non esistono solo per i PLC

Anche per il linguaggio C esiste una norma: l'ANSI C (ISO C89), definita nel 1990 da parte dell'American National Standards Institute .

### Esempi di compilatori o ambienti di sviluppo integrati standard ANSI C o ANSI C++

- SC/SCpp
- Cilk ANSI C Based Compiler
- CCC386 C Compiler
- LCC – A Retargetable Compiler for ANSI C
- Leonardo IDE
- DJGPP
- Intel C++ Compiler for Linux
- ANYC C Compiler
- MrC/MrCpp
- Visual C/C++

### Esempi di compilatori o ambienti di sviluppo integrati non strettamente ANSI C o C++

- Cyclone C
- CINT C and C++ Interpreter

Ing. Elena Mainardi

PLC

### I linguaggi per PLC che seguono la norma 61131, certificati dall'associazione PLCOpen

Linguaggio



Ditta produttrici del PLC che usa quel linguaggio



Product name	Base Level	CL & RL	Company	Country
Codesys	IL & ST		<a href="#">3S Smart Software Solutions</a>	Germany
Concept	IL, ST, FBD	CL & RL for ST	<a href="#">Schneider Automation</a>	Germany
ISaGRAF	IL		<a href="#">ICS TripleX</a>	France
MELSEC MEDOC plus	IL		<a href="#">Mitsubishi Electric Europe</a>	Germany
MULTIPROG wt	IL & ST		<a href="#">KW-Software</a>	Germany
NAiS Control FPDWIN Pro	IL	CL & RL for ST	<a href="#">Panasonic Electric Works</a>	Germany
openDK	IL		<a href="#">infoteam Software</a>	Germany
openPCS	IL & ST		<a href="#">infoteam Software</a>	Germany
PUMA	IL & ST		<a href="#">KEBA</a>	Austria
RSLogix 5000 V.13		RL for ST	<a href="#">Rockwell Automation</a>	USA
S7-SCL	ST	RL for ST	<a href="#">Siemens</a>	Germany
Simotion Eng. Sytem Scout-standard	ST		<a href="#">Siemens</a>	Germany
SELECONTROL CAP 1131	IL		<a href="#">Selectron</a>	Switzerland
Sisteam Servicer (IEC 1131-3)	ST		<a href="#">TEAM</a>	Spain
S7-GGRAPH	SFC		<a href="#">Siemens</a>	Germany
Melsoft GX IEC Developer	IL, ST		<a href="#">Mitsubishi</a>	Germany

+altri linguaggi che seguono la norma ma non sono certificati (ad. Es. PLC di B&R, Beckhoff...)

Ing. Elena Mainardi

PLC

### I linguaggi per PLC che seguono la norma 61131, certificati dall'associazione PLCOpen

Product name	Base Level	CL & RL	Company	Country
Codesys	IL & ST		<a href="#">3S Smart Software Solutions</a>	Germany
Concept	IL, ST, FBD	CL & RL for ST	<a href="#">Schneider Automation</a>	Germany
ISaGRAF	IL		<a href="#">ICS TripleX</a>	France
MELSEC MEDOC plus	IL		<a href="#">Mitsubishi Electric Europe</a>	Germany
MULTIPROG wt	IL & ST		<a href="#">KW-Software</a>	Germany
NAiS Control FPDWIN Pro	IL	CL & RL for ST	<a href="#">Panasonic Electric Works</a>	Germany
openDK	IL		<a href="#">infoteam Software</a>	Germany
openPCS	IL & ST		<a href="#">infoteam Software</a>	Germany
PUMA	IL & ST		<a href="#">KEBA</a>	Austria
RSLogix 5000 V.13		RL for ST	<a href="#">Rockwell Automation</a>	USA
S7-SCL	ST	RL for ST	<a href="#">Siemens</a>	Germany
Simotion Eng. Sytem Scout-standard	ST		<a href="#">Siemens</a>	Germany
SELECONTROL CAP 1131	IL		<a href="#">Selectron</a>	Switzerland
Sisteam Servicer (IEC 1131-3)	ST		<a href="#">TEAM</a>	Spain
S7-GGRAPH	SFC		<a href="#">Siemens</a>	Germany
Melsoft GX IEC Developer	IL, ST		<a href="#">Mitsubishi</a>	Germany

Il linguaggio base dei PLC Siemens, STEP7 (abbreviato S7), non segue la norma (ma ci si avvicina). Invece alcune sue variazioni, tipo S7-SCL, la seguono

Ing. Elena Mainardi

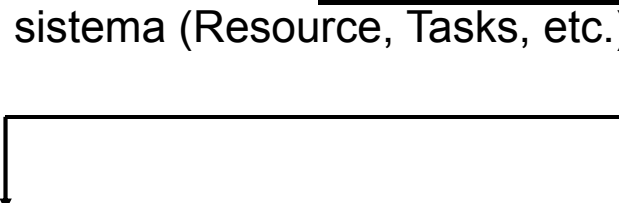
PLC

## Norma IEC 61131

- La parte terza dello Standard, IEC 1131-3, definisce:
  - Elementi comuni per la configurazione delle applicazioni, la dichiarazione di variabili, la strutturazione dei programmi.
  - Quattro linguaggi di programmazione, due di tipo grafico e due di tipo testuale
  - Funzioni e blocchi funzionali standard, per definire un set minimo di operazioni di conteggio, temporizzazione, elaborazioni matematiche, conversioni di tipo ecc...

## Gli elementi comuni della norma

- Tipi di dato e Dichiarazioni delle variabili
- Elementi di organizzazione dell'applicativo (Program Organization Units, POU), suddivisi in:
  - Functions
  - Function Blocks
  - Programs
- Schema formale per la descrizione di operazioni sequenziali,
- **Sequential Function Chart**. (SFC)
- Configurazione di sistema (Resource, Tasks, etc.)



N.B. È in pratica il 5° linguaggio definito dalla norma, anche se, invece che essere definito nella sezione "Linguaggi", è definito nella sezione "Elementi Comuni"



## In soldoni

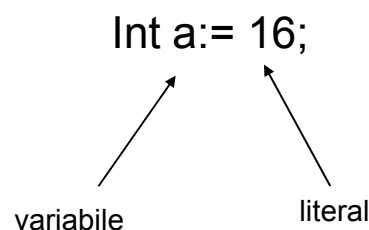
La norma 1131 ci dice come è strutturato un software che debba controllare un PLC, definendo quali sono i tipi di dato che si possono usare, come dichiarare le variabili, che istruzioni poter utilizzare, in che modo definire e dichiarare le funzioni, quali parti di un programma possono richiamarne altre, come fanno le varie sottoparti di un progetto software a scambiarsi dati (ad esempio usando variabili globali, o variabili passate per riferimento...).

## Literal constants e Tipi di dato

- Una costante letterale (literal constant) definisce la rappresentazione di costanti per un particolare tipo di dato.

- I tipi di dato semplice sono:

- Integer
- Bit string
- Real
- Time
- Character string



## Tipo Integer

IEC Data type	Description	bits	Range
SINT	Short integer	8	-128 to +127
INT	Integer	16	-32768 to +32767
DINT	Double integer	32	$-2^{31}$ to $+2^{31} - 1$
LINT	Long integer	64	$-2^{63}$ to $+2^{63} - 1$
USINT	Unsigned short integer	8	0 to 255
UINT	Unsigned integer	16	0 to +65535
UDINT	Unsigned double integer	32	0 to $+2^{32} - 1$
ULINT	Unsigned long integer	64	0 to $+2^{64} - 1$

## Integer literals

Literal constant	Examples
Decimal values (base 10)	-123, 0, 23_123
Binary values (base 2)	2#1111_1111 (255 decimal)
Octal values (base 8)	8#377 (255 decimal)
Hexadecimal values (base 16)	16#FF (255 decimal)

Ing. Elena Mainardi

PLC

## Tipo Real

IEC Data type	Description	bits	Range
REAL	Real number (IEEE floating point format)	32	$\pm 10^{\pm 38}$
LREAL	Long Real number	64	$\pm 10^{\pm 308}$

I valori definiti come REAL hanno una precisione di 1 parte su 223

I valori definiti come LREAL hanno una precisione di 1 parte su 252

## Real literals

Literal constant	Examples
Real constant	10.123, +12_123.21, -1.6eE-10, 0.327e+14

NOTA: La notazione esponenziale di un numero reale si indica con la potenza di 10 con cui il numero viene moltiplicato.

La notazione prevede l'utilizzo indifferente della lettera maiuscola 'E' e minuscola 'e'.

Ing. Elena Mainardi

PLC

## Tipo Date and Time

IEC Data type	Description	bits	Usage
DATE	Calendar date	Implementation dependent	Storing calendar date
TIME_OF_DAY or TOD	Time of the day	Implementation dependent	Storing time of the day
DATE_AND_TIME or DT	Date and time of the day	Implementation dependent	Storing the date and the time of the day

Ing. Elena Mainardi

PLC

## Time literals

Literal constant	Examples	Meaning
Short form	T#12d3h12m4s23ms	12 (d)ays, 3(h)ours 12 (m)inutes, 4 (s)econds and 23 (m)illi(s)econds
Long form	TIME#12d3h12m4s23ms	"

Gli identificativi TIME e T nelle forme estesa e compatta possono essere scritti in minuscolo (**time** e **t**).

## Date literals

Literal constant	Examples	Meaning
Long form	DATE#2002-04-21	21th April 2002
Short form	D#2002-04-21	"
Long form	TIME_OF_DAY#05:00:00.56	0.56 seconds past 5 o'clock
Short form	TOD#05:00:00.56	"
Long form	DATE_AND_TIME#2002 04 12-05:00:00.56	12th April 2002, 0.56 seconds past 5 o'clock
Short form	DT#2002-04-12-05:00:00.56	"

Ing. Elena Mainardi

PLC

## Tipo STRING

IEC Data type	Description	bits	Usage
STRING	Character strings	Implementation dependent	Storing textual information

## Tipo Bit string

IEC Data type	Description	bits	Usage
BOOL	Bit string of 1 bit. Has two states, FALSE (=0) and TRUE (=1)	1	Store logical states
BYTE	Bit string of 8 bits.	8	Binary information
WORD	Bit string of 16 bits.	16	”
DWORD	Bit string of 32 bits.	32	”
LWORD	Bit string of 64 bits.	64	”

## Tipo di dato generico

- La norma dà anche la possibilità di usare un tipo di dato generico.
- I tipi di dato generico ANY sono utilizzati per definire variabili in FUNCTION e FUNCTION BLOCK che supportano l'overloading delle variabili. L'overloading assomiglia al “casting” che vale per il linguaggio c.

In sostanza, per esempio, io posso scrivere una funzione dichiarando che il tipo di dato su cui opererà è ad esempio ANY\_MAGNITUDE (cioè un numero generico, senza dire se è intero, reale, time etc), e poi invocare la funzione passando per parametro una volta un intero, una volta un reale etc. La funzione si adatterà al tipo di dato che io mando quando la invoco.

## Tipo di dato generico

### Esempio

```
(* Definizione di una funzione *)

FUNCTION max : ANY_MAGNITUDE
  VAR_INPUT
    A,B : ANY_MAGNITUDE;
  END_VAR

  IF A >= B THEN  max := A;
  ELSE max := B;
  END_IF
END_FUNCTION
```

```
(* Main programme *)
VAR
  t : TIME;
  speed : REAL;
END_VAR

VAR CONSTANT
  MAX_TIME : TIME := T#2s;
  MAX_SPEED : REAL := 12.34;
END_VAR

...
IF max(t,MAX_TIME)=MAX_TIME THEN ....
...
IF max(speed,MAX_SPEED)=MAX_SPEED THEN ....
..
```

## Tipo di dato derivati

Oltre ai tipi di dato semplice (come intero, reale, stringa, time ...) esistono anche i tipi di dato derivati, esattamente come in linguaggio c

- La tipizzazione dei dati permette di eliminare incongruenze ed errori già in fase di compilazione e semplifica il compito del programmatore.
- l'utilizzo di tipi di dato derivato contribuisce alla leggibilità del software (miglioramento della qualità del software).

I tipo di dato derivato possono essere definiti con la sintassi :

```
TYPE NomeTipo:
    Descrizione ...
END TYPE
```

Il nuovo tipo può essere:

- Un tipo standard, che viene semplicemente ridefinito per comodità
- Un subrange di un tipo standard: la descrizione del tipo è TipoStandard (ValoreIniz..ValoreFin).
- Un elenco di valori costanti che la variabile può assumere, tipo enumerativo.
- Un vettore di dimensione finita, definito con ARRAY [Intervallo di Interi] OF Tipo.
- Una struttura, cioè un insieme di variabili di tipo non omogeneo, definita con STRUCT Var1:Tipo1 .. VarN:TipoN END STRUCT.

### Tipo di dato derivati

(\* 1 - Definizioni di variabili per memorizzare  
il valore di pressione di pompe \*)

Esempio: posso definire una  
variabile pump\_1 come  
REAL

```
VAR
    pump_1, pump_2, pump_3: REAL;
END_VAR;
```

(\* 2 - Definisco un tipo di variabile per evidenziare  
la responsabilita' della variabile - nel  
caso in esame la memorizzazione di un valore  
di pressione - il software sara' piu' chiaro \*)

oppure definire un tipo  
PRESSURE che di fatto è un  
alias di REAL e poi definire  
pump\_1 di tipo PRESSURE

```
TYPE
    PRESSURE : REAL;
END_TYPE
```

```
VAR
    pump_1, pump_2, pump_3: PRESSURE;
END_VAR;
```

```

TYPE
  MACHINE_STATUS : (STOP, RUN, FAULTY, STANDBY);
END_TYPE
...

```

```

VAR
  STATUS : MACHINE_STATUS;
END_VAR

```

```

IF STATUS = STOP THEN .....
END_IF

```

Esempio: enumerated

```

TYPE
  DEVICE_MODE : (INITIALISING, RUNNING, STANDBY, FAULTY);
  PUMP_MODE : (RUNNING, OFF, FAULTY);
END_TYPE

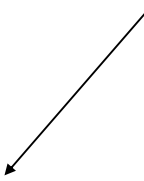
```

```

VAR
  AX100 : DEVICE_MODE;
  XV23 : PUMP_MODE;
END_VAR

```

Ambiguità perché ho definito  
il valore FAULTY sia nel  
tipo device\_mode che nel  
tipo pump\_mode



```

(* Se esistono ambiguita', occorre specificare il tipo *)
IF AX100 = DEVICE_MODE#FAULTY THEN
  XV23 = PUMP_MODE#OFF;

```

Ing. f

PLC

### Esempio: Subrange

```

TYPE
  MOTOR_VOLTS : INT(-6..+12);
END_TYPE

```

### Esempio: Array

```

VAR COSTANT
  MAX_CAM_POINT : INT := 100;
  NUMBER_OF_MOTOR : INT := 10;
END_VAR

```

```

TYPE
  SLAVE_POS : REAL;
  CAM_POINTS : ARRAY [1..MAX_CAM_POINT] OF SLAVE_POS;
  MOTION_MATRIX : ARRAY [1..MAX_CAM_POINT, 1..NUMBER_OF_MOTOR] OF SLAVE_POS;
END_TYPE

```

NOTA: il numero massimo della dimensione dei vettori dipende dalla implementazione e non è definito dalla norma.

## **Esempio: Structure**

```
TYPE PRESSURE_SENSOR:
  STRUCT
    INPUTS : PRESSURE;
    STATUS : DEVICE_MODE;
    CALIBRATION : DATE;
    CALIBRATION_PARAMETERS : ARRAY[0..10] OF REAL;
    HIGH_LIMIT : REAL(-20..+20);
  END_STRUCT;
END_TYPE
```

## **Dichiarazione delle variabili**

- Sintassi generica :  
<NOME> : <TIPO> ;  
<NOME> : <TIPO> := <VALORE INIZIALE> ;
- La norma sancisce che ogni variabile deve avere un valore iniziale.
- Se questo non viene definito nel programma, gli viene assegnato il valore di default nullo (D#0001-01-01 per le date).



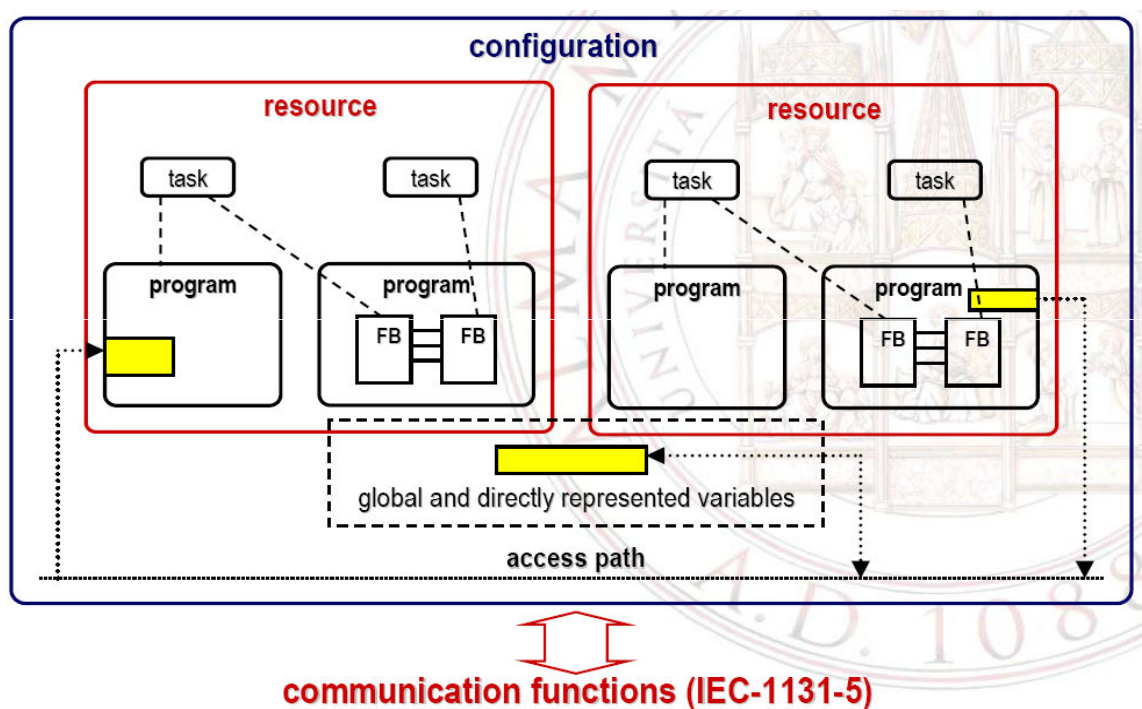
## Norme IEC 1131-3 Organizzazione del software

L'IEC ha elaborato un modello software di tipo stratificato e gerarchico che tenga in conto tutte le interazioni tra programma e ambiente operativo

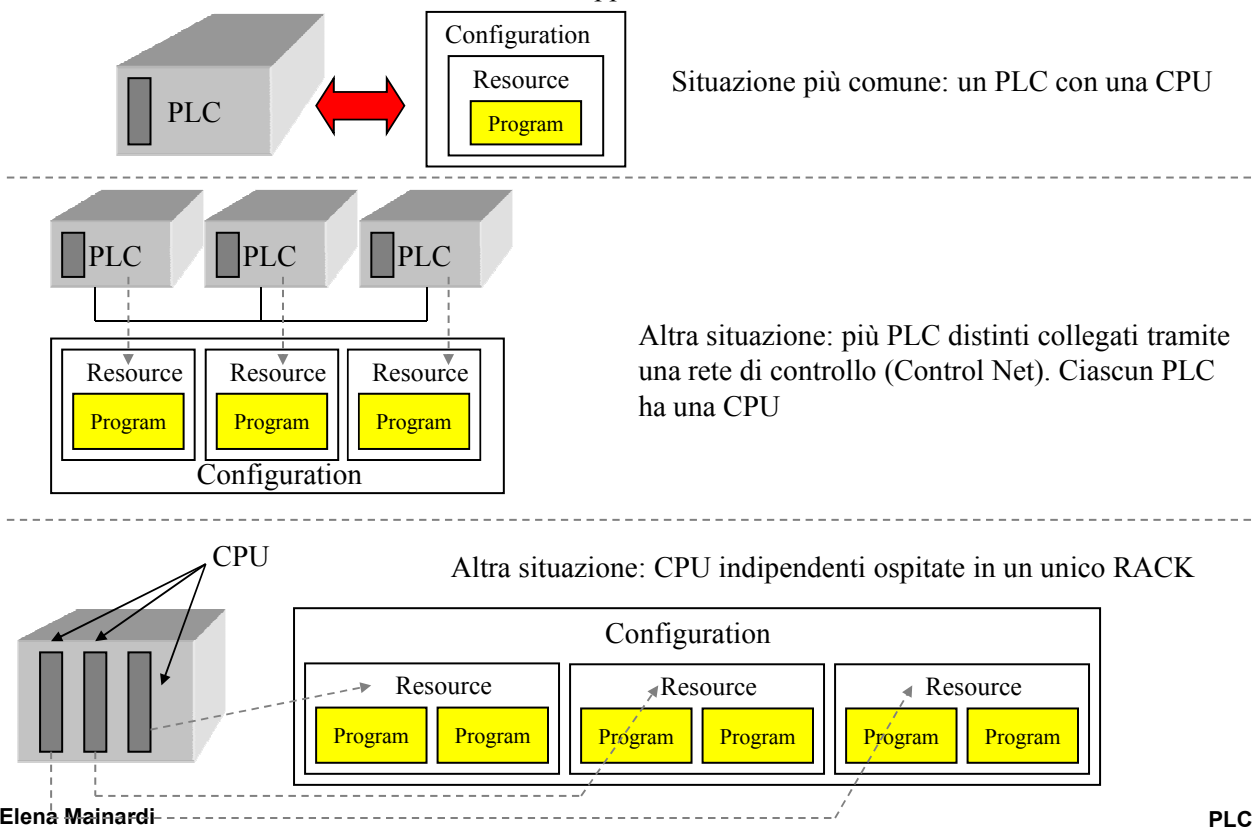
### Componenti del modello:

- configuration;
  - resource;
  - program; —————→
  - task; —————→
  - local and global variables;
  - directly represented variables;
  - function blocks; —————→
  - functions; —————→
  - access path
- Queste tre sono chiamate **POU** (program organization units)

## Il modello software di un applicativo per PLC



Configurazione dell'applicazione: è in pratica l'insieme di tutte le risorse hardware che userò nella mia applicazione



## configuration:

è l'elemento che contiene tutti gli altri elementi e corrisponde all'intero PLC o all'insieme di PLC, nel caso ce ne siano più di uno collegati tra loro. In un sistema di controllo è possibile avere anche più configuration in comunicazione tra loro, ognuna delle quali gestisce uno o più PLC.

Ogni configuration contiene una o più resource, ognuna delle quali contiene uno o più program, a loro volta eseguiti sotto il controllo di zero o più task. Un program può contenere zero o più function block e la chiamata a zero o più function.

**resource**: è l'elemento che corrisponde ad una unità funzionale di elaborazione dei segnali, connessa a dispositivi di interfaccia uomo-macchina e con funzioni di interazione con sensori ed attuatori. In pratica una resource è un modulo CPU!!

Caratteristiche di una resource:

- accede alle global ed alle directly represented variables;
- è l'interfaccia tra I/O fisici ed il programma

**program**: è l'insieme logico di tutti gli elementi di programmazione ed i costrutti necessari per l'elaborazione dei segnali da parte di un PLC. In pratica è l'insieme del codice (e delle sue variabili) che esegue le funzioni della mia applicazione. E' istanziato a livello di resource. Nell'ambito della stessa **Configuration** è possibile utilizzare più istanze dello stesso program in differenti **Resource**

Caratteristiche di un program:

- contiene costrutti realmente eseguibili ;
- può essere scritto in diversi linguaggi;
- la sua esecuzione avviene sotto il controllo di uno o più task

Il program è una POU molto simile al function block, però è un "contenitore più ampio". Nei PLC tradizionali esiste un solo programma e una sola risorsa (CPU) per eseguirlo, ma abbiamo visto che la norma prevede anche situazioni più complesse.

**task**: è l'elemento che si occupa di controllare e mettere in esecuzione un program o una sua parte.

Può essere:

- periodico (cioè è mandato in esecuzione ogni tot istanti di tempo);
- attivato da evento (viene mandato in esecuzione se succede un particolare evento)

N.B. Un program è solo un insieme di righe di codice (se è scritto in ST, oppure un insieme di oggetti grafici se scritto in un linguaggio grafico) e di dichiarazioni di variabili, che non interagisce con l'hardware se non viene mandato in esecuzione da un task.

Finchè un task non manda in esecuzione il program, il program non fa niente, non ha alcun effetto. E' il task che rende esecutivo il codice di un program.

Eccezioni alla norma:

Non tutti i PLC funzionano allo stesso modo

- Ad esempio l'ambiente di sviluppo dei PLC B&R non ammette task ad evento, ma solo ciclici.
- Sarà cura del programmatore associare alle task più veloci i compiti più critici
- In Siemens non tutti gli ingressi possono essere associati a task ad evento: bisogna che siano di un modulo speciale

Nel caso più semplice di una macchina automatica che debba essere controllata da un singolo PLC con un singolo modulo CPU, quindi, dovendo scrivere il software avremo una configuration (tutto il nostro progetto) che contiene una resource (la gestione del modulo CPU) all'interno della quale girerà il nostro program, che verrà eseguito da un task.

### **Di cosa si compone un program?**

Di dichiarazioni, istruzioni e ovviamente chiamate a “subroutines” che nel caso dei PLC si chiamano function e function block

→ Occhio, possibili domande d'esame: definire i concetti di configuration, resource e task. Che tipi di task esistono? ←

### **POU: functions**

Le functions sono moduli di programma (cioè insieme di variabili e codice) riutilizzabili (cioè possono essere richiamate più e più volte) il cui valore delle variabili interne è solo temporaneo

→ non hanno memoria

Le variabili interne alla function, ogni volta che la sua esecuzione è terminata, vengono resettate.

Ne consegue che ogni volta che si richiama una function passandole gli stessi parametri di ingresso, essa produrrà sempre la stessa uscita.

La function è definita anche come un POU che, quando viene eseguita, restituisce un unico elemento (che può essere multidimensionale, es. vettori, strutture).

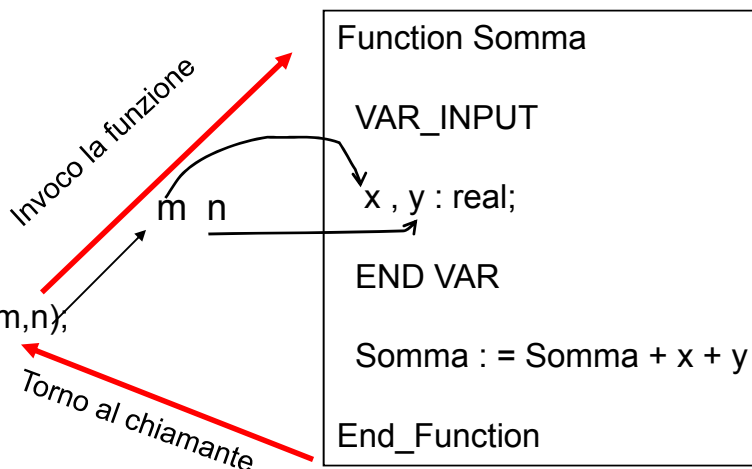
L'invocazione della funzione può essere utilizzata come operando in un'espressione in linguaggio testuale.

Ogni volta che si richiama una function passandole gli stessi parametri di ingresso, essa produrrà sempre la stessa uscita

Es.

### Programma

```
-----
-----
-----
Add = Somma (m,n),
-----
-----
-----
```



Se passo m = 3 e n = 5, ottengo sempre 8, anche se chiamo la funzione 100 volte!!!

## POU: functions

### Esempio

```
RES2 := MUL (SIN(X), COS(Y));
```

In questa riga di codice sono state chiamate le funzioni MUL (che esegue una moltiplicazione), SIN (seno) e COS(coseno). SIN e COS sono usate come operandi della funzione MUL. Entrambe SIN e COS restituiscono un unico valore. Questi due valori vengono moltiplicati dalla funzione MUL, che a sua volta restituisce un unico valore, che verrà immagazzinato in RES2.

MUL, SIN e COS sono funzioni standard, già predefinite e utilizzabili dall'utente.

Ovviamente il programmatore può scriversi le **sue** funzioni.

La definizione di una funzione creata dall'utente richiede:

- Una lista di variabili d'ingresso ed interne (temporanee)
- La descrizione dell'algoritmo implementato dalla funzione. Tale algoritmo può essere implementato in un qualunque linguaggio della norma ad esclusione dell'SFC

### Testuale

```
FUNCTION SIMPLE_FUN : REAL
  VAR_INPUT
    A,B : REAL ;      (* External interface specification *)
    C : REAL := 1.0;
  END_VAR

  SIMPLE_FUN := A*B/C; (* Function body specification *)

END_FUNCTION
```

### Grafica

```
FUNCTION
+-----+
| SIMPLE_FUN |
REAL----|A      |----REAL
REAL----|B      |(* External interface specification *)
REAL----|C      |
+-----+

+-----+ (* Function body specification *)
A---| * | +---+
B---|   |---| / |---SIMPLE_FUN
+-----+ |   |
C-----|   |
+-----+

END_FUNCTION
```

Ing. Elena Mainardi

PLC

### Un piccolo dettaglio

```
FUNCTION SIMPLE_FUN : REAL
  VAR_INPUT
    A,B : REAL ;      (* External interface specification *)
    C : REAL := 1.0;
  END_VAR

  SIMPLE_FUN := A*B/C; (* Function body specification *)

END_FUNCTION
```

Le variabili specificate col costrutto VAR\_INPUT, sono variabili che la POU prende dal chiamante e NON PUO' MODIFICARE.

Però in questo esempio la variabile C, pur essendo dichiarata come input, viene inizializzata al valore 1.

Come mai?

→ Perché la norma dice che una POU deve poter essere invocata anche senza passarle tutti i parametri di ingresso. Se non viene passato qualche parametro, il suo valore viene inizializzato a quello che compare nella dichiarazione all'interno della POU.

In altre parole, nell'esempio posso invocare la funzione in più modi, ad esempio:

1. simple\_fun(8,4,2), nel qual caso A = 8, B = 4, C = 2, e il risultato restituito sarà  $8*4/2=16$
2. Simple\_fun(8,4), nel qual caso A = 8, B = 4 e C = 1 (in quanto non è stato specificato), e il risultato restituito sarà  $8*4/1=32$

Definizione testuale della funzione PROVA

```

FUNCTION PROVA : REAL
  VAR_INPUT
    A,B : REAL ;    (* External interface specification *)
    C : REAL := 1.0;
  END_VAR
  VAR
    COUNTP1 : INT ;
  END_VAR

  COUNTP1 := ADD(A,1); (*Function body specification *)
  PROVA := A*B/C;
END_FUNCTION

```

Invocazione di funzioni in linguaggio testuale

```

VAR
  X,Y,Z,RES1,RES2 : REAL;
  EN1,V : BOOL;
END_VAR
RES1 := DIV(IN1 := COS(X), IN2 := SIN(Y), ENO => EN1);
RES2 := MUL (SIN(X), COS(Y));
Z := ADD(EN := EN1, IN1 := RES1, IN2 := RES2, ENO => V);

```

Ing. Elena Mainardi

PLC

Riprendiamo l'esempio di prima

```

VAR
  X,Y,Z,RES1,RES2 : REAL;
  EN1,V : BOOL;
END_VAR

RES1 := DIV(IN1 := COS(X), IN2 := SIN(Y), ENO => EN1);
RES2 := MUL (SIN(X), COS(Y));
Z := ADD(EN := EN1, IN1 := RES1, IN2 := RES2, ENO => V);

```

IN1 e IN2 sono gli identificativi degli ingressi.

Nel primo esempio vuol dire che la funzione div farà la divisione tra cos(x) e sin(y).

Ogni funzione o function block, poi, ha per definizione della norma due parametri di enable: EN (enable di ingresso) e ENO (enable di uscita).

Gli enable si usano opzionalmente, e servono più che altro nei linguaggi di programmazione grafica, per una questione di ordine di esecuzione dei blocchi.

In questo caso, essendo il linguaggio testuale, l'uso degli enable è superfluo.

Infatti in questo caso vorrebbe dire: quando la funzione div ha finito, fa in modo che la fine della sua esecuzione (ENO che va alto) porti ad un valore logico vero la variabile booleana EN1.

Quando EN1 è true, la funzione ADD è abilitata a partire.

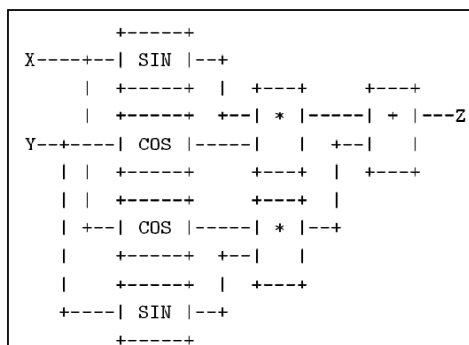
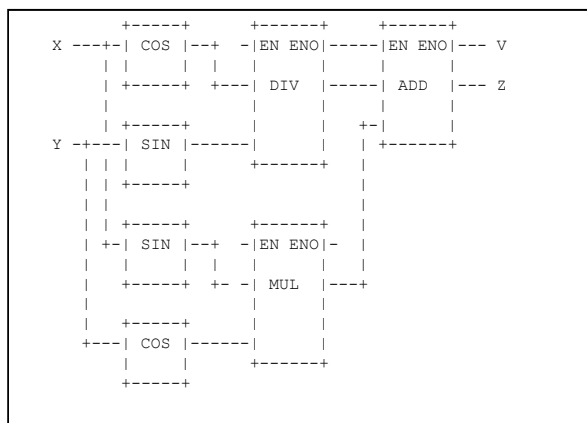
L'uso degli enable in questo caso è superfluo perché tanto in un linguaggio testuale ogni istruzione viene eseguita solo al termine della precedente, quindi è ovvio che la funzione ADD venga eseguita dopo che la div ha finito.

Ing. Elena Mainardi

PLC



### Invocazione di funzioni in linguaggio grafico

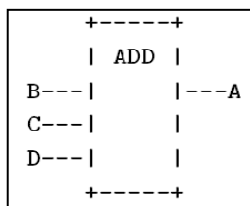


$$Z = \sin X \cos Y + \sin Y \cos X$$

Ing. Elena Mainardi

PLC

### Rappresentazione grafica dell'invocazione della funzione ADD

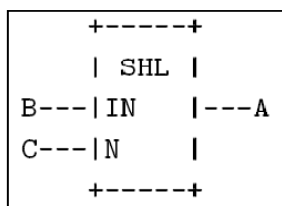


### Rappresentazione testuale dell'invocazione della funzione ADD

A:= ADD (B,C,D);

### Rappresentazione grafica dell'invocazione della funzione SHL

(Shift left, è una funzione che shifta a sinistra di n posizioni i bit di cui si compone il dato)



### Rappresentazione testuale dell'invocazione della funzione SHL

A:= SHL (IN := B, N := C);

Ing. Elena Mainardi

PLC

### Elenco di funzioni standard definite dalla norma IEC 61131-3

- Function per la conversione tra tipi
- Es: INT TO REAL, BCD TO INT, ecc...
  
- Function aritmetiche: ADD, SUB, MUL, DIV, COS,..
  
- Function combinatorie e di scorrimento: per operazioni logiche, AND, OR, XOR, NOT, e manipolazione di stringhe di bit, SHL, SHR, ROR,..
  
- Function di selezione e comparazione, MIN, MAX,.. e GT, GE, EQ ..
  
- Function per stringhe di caratteri, LEFT, CONCAT, INSERT ...

### POU: function blocks

I function blocks (FB) sono moduli di programma riutilizzabili formati da:

- dati: variabili utilizzate all'interno del FB (parametri I/O, variabili interne)
- algoritmo: insieme di istruzioni che costituiscono il corpo del FB

L'algoritmo elabora un nuovo set di parametri in uscita e di variabili interne a partire dal valore corrente di ingressi e variabili interne

Le variabili utilizzate dal function block sono statiche, ovvero quando il FB viene richiamato, esse hanno il valore che avevano alla fine dell'esecuzione della precedente chiamata al FB. Ne consegue che chiamate successive al FB passando sempre gli stessi parametri di ingresso possono dare variabili di uscita diverse.

Es.

Programma

```

-----
-----
-----
Add = Somma (m,n);
-----
Add = Somma (m,n);
-----
-----

```

Invoco il function block

m n

Torno al chiamante

```

Function_block Somma
VAR_INPUT
x , y : real;
END_VAR

VAR_OUTPUT
Somma : real;
END_VAR

VAR
somma_temp:int;
END_VAR

somma_temp:=somma_temp+x+y;
Somma := somma_temp;
End_Function_Block

```

La prima volta, se passo  $m = 3$  e  $n = 5$  ottengo  $Add = 8$ .  
 La seconda volta, se ripasso  $m = 3$  ed  $n = 5$ , essendo la variabile `somma_temp` rimasta al valore che aveva prima, cioè a 8, ottengo  $Add = 8 + 3 + 5 = 16$

## Un piccolo dettaglio

Poiché un programmatore può aver bisogno di variabili NON statiche anche dentro ad un function block, la norma mette a disposizione un altro costrutto per la dichiarazione delle variabili di un function block, che le rende non statiche.

```

VAR_TEMP
a: real;
c: time;
END_VAR

```

Il function block è anche definito come un POU che, quando viene eseguita, restituisce uno o più elementi in uscita (a differenza della function, che ne può restituire uno solo)

Un function block è istanziato a livello di program o di un altro function block, a differenza del program, che è istanziato a livello di resource.

Ciascuna istanza di un Programma o di un Function Block condivide lo stesso codice, ma ha la sua area dati privata in memoria

Nell'ambito dello stesso **Program** è possibile utilizzare più istanze dello stesso Function Block

La definizione di un blocco funzione (FB) richiede:

- Una lista di variabili d'ingresso ed interne (temporanee e statiche)
- La descrizione dell'algoritmo implementato dalla funzione. Tale algoritmo può essere implementato in un qualunque linguaggio della norma, anche l'SFC

#### Esempio: definizione di un FB di nome COUNTER

```

TYPE
    ModeType : (RESET, COUNT, HOLD);
END_TYPE

FUNCTION_BLOCK COUNTER

    (* Define external interface *)
    VAR_INPUT
        MODE : ModeType := RESET; (* Valore di default *)
    END_VAR
    VAR_OUTPUT
        OUT : INT := 0; (* Valore iniziale *)
    END_VAR

    (* Define private data *)
    VAR
        CountStatus : INT := 0;
    END_VAR

    (* Define private algorithm *)
    IF MODE = RESET THEN
        CountStatus = 0;
    ELSEIF MODE = COUNT THEN
        CountStatus := CountStatus + 1;
    END_IF

    OUT := CountStatus;
END_FUNCTION_BLOCK

```

**Pou:**

- functions
- function blocks
- programs

### Cos'è un'ISTANZA? Cosa vuol dire che un FB è istanziato a livello di program o di un altro FB?

Per prendere confidenza col concetto di istanziazione, forniamo un esempio analogo nel caso della programmazione in linguaggio c.

Pensiamo al caso di definizione di un tipo di dato in un generico programma in c

```
Es. TYPEDEF struct
    {primo_campo:integer;
      secondo_campo:real;
      terzo_campo:char;
    } mio_record ;
```

In questo caso ho definito un mio tipo di dato. Ogni volta che, nel main o nelle altre funzioni del mio codice, vorrò usare una variabile di tipo "mio\_record", dovrò scrivere: "nome\_variabile : mio\_record;"  
Quindi il tipo di dato "mio\_record" è definito una sola volta al di fuori di tutte le funzioni, e viene "istanziato" ogni volta che dichiaro una variabile di tipo "mio\_record" all'interno di una qualche funzione.

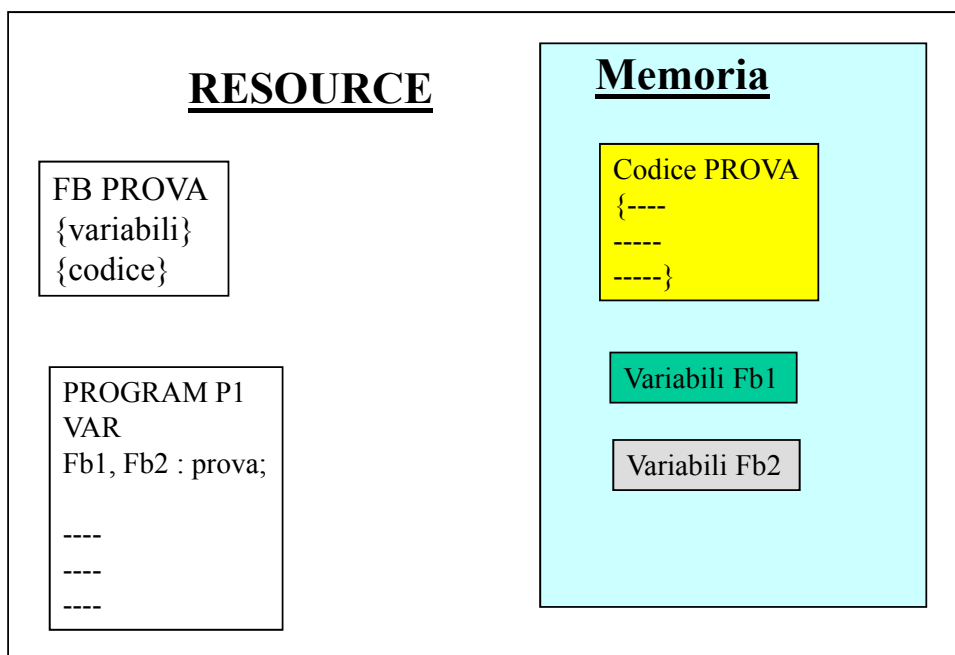
### Cos'è un'ISTANZA? Cosa vuol dire che un FB è istanziato a livello di program o di un altro FB?

Nel caso del FB per PLC, invece che di variabili si parla di una POU, cioè di un insieme di variabili e codice.

L'FB io lo posso definire dentro ad un program o dentro ad un altro FB, ma poi ogni volta che voglio avere effettivamente la possibilità di richiamare questo FB, lo devo istanziare. Pensando all'esempio della pagina precedente, dopo aver definito il FB di tipo "counter", potrò per esempio scrivere:

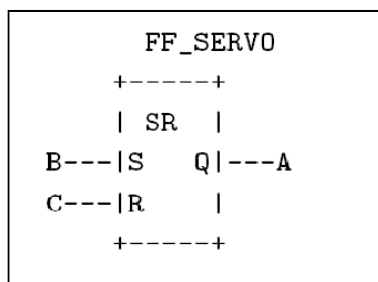
```
VAR
    counter1:counter;
END_VAR
```

A questo punto, se in qualche riga di codice invoco counter1, manderò in esecuzione un codice di tipo "counter".



In questo esempio ho definito un function block PROVA e all'interno di un program l'ho istanziato due volte. Vorrà dire che nella memoria avrò una zona col codice del function block e due zone per i dati delle due istanze del function block

Dichiarazione grafica del FB "FF\_SERVO" di tipo SR (Set Reset)



Dichiarazione testuale di un FB di nome FF\_SERVO di tipo SR

```
VAR (* Declaration *)
  FF_SERVO: SR
END_VAR
FF_SERVO(S:=B, R:=C); (* Invocation *)
A:= FF_SERVO.Q; (* Assign output *)
```

Come si può notare, siccome un function block non restituisce un unico valore, come invece fa la funzione, per usarlo devo sempre scrivere due istruzioni, una in cui lo invoco FF\_SERVO(...); e una in cui assegno la sua uscita ad una mia variabile A:=FF\_SERVO.Q;

Graphical (FBD language)	Textual (ST language)
<pre>           FF75           +-----+              SR        %IX1--- S1  Q1 ---     %QX3     %IX2--- R                  +-----+ </pre>	<pre> VAR FF75: SR; END_VAR      (* Declaration *)  FF75(S1:=%IX1, R:=%IX2); (* Invocation *) %QX3 := FF75.Q1 ;          (* Assign Output *) </pre>
<pre>           MyTon           +-----+     +-----+    TON    a--  NE  ---O EN  ENO -- b--       r-- IN   Q O- out     +-----+ -- PT   ET --           +-----+ </pre>	<pre> VAR a,b,r,out : BOOL; MyTon : TON; END_VAR  MyTon(EN := NOT (a &lt;&gt; b),       IN := r,       NOT Q =&gt; out); </pre>

Ing. Elena Mainardi

PLC

(\* Textual declaration in ST language \*)

```

FUNCTION_BLOCK DEBOUNCE

(** External Interface **)
VAR_INPUT
    IN : BOOL ;                (* Default = 0 *)
    DB_TIME : TIME := t#10ms ; (* Default = t#10ms *)
END_VAR
VAR_OUTPUT
    OUT : BOOL ;                (* Default = 0 *)
    ET_OFF : TIME ;            (* Default = t#0s *)
END_VAR
VAR
    DB_ON : TON ;               (** Internal Variables **)
    DB_OFF : TON ;              (** and FB Instances **)
    DB_FF : SR ;
END_VAR

(** Function Block Body **)

DB_ON(IN := IN, PT := DB_TIME) ;
DB_OFF(IN := NOT IN, PT:=DB_TIME) ;
DB_FF(S1 :=DB_ON.Q, R := DB_OFF.Q) ;
OUT := DB_FF.Q ;
ET_OFF := DB_OFF.ET ;

END_FUNCTION_BLOCK

```

(Graphical declaration in FBD language \*)

```

FUNCTION_BLOCK

(** External Interface **)
          +-----+
          |  DEBOUNCE  |
    BOOL---|IN          OUT|---BOOL
    TIME---|DB_TIME  ET_OFF|---TIME
          +-----+

(** Function Block Body **)

          DB_ON          DB_FF
          +-----+      +-----+
          |  TON  |      |  SR  |
IN-----+-----|IN  Q|-----|S1 Q|---OUT
          | +---|PT ET| +---|R   |
          | | +-----+ | +-----+
          | |          | |
          | |  DB_OFF  |
          | | +-----+ |
          | |  TON  | |
          +---|---O|IN  Q|---+
    DB_TIME---+---|PT ET|-----ET_OFF
          +-----+

END_FUNCTION_BLOCK

```

Ing. Elena Mainardi

PLC

**Esempi di FUNCTION BLOCKS standard definiti dalla norma 61131-3****operazioni di temporizzazione**

```

+-----+
|  ***  |
BOOL---|IN   Q|---BOOL
TIME---|PT   ET|---TIME
+-----+

```

- \*\*\* = TON: Timer On Delay, temporizzatore con ritardo all'inserzione.
- \*\*\* = TOF: Timer Off Delay, temporizzatore con ritardo alla disinserzione.
- \*\*\* = TP: Timer Pulse, temporizzatore ad impulso.

- Elementi bistabili, cioè analoghi a flip-flop Set/Reset:

```

+-----+
|  SR  |
BOOL---|S1 Q1|---BOOL
BOOL---|R   |
+-----+

```

- Rilevatori di fronti, di salita o di discesa:

```

+-----+
| R_TRIG |
BOOL---|CLK   Q|---BOOL
+-----+

```

Ing. Elena Mainardi

PLC

**Esempi di FUNCTION BLOCKS standard definiti dalla norma 61131-3****operazioni di conteggio**

```

+-----+
|  CTU  |
BOOL--->CU   Q|---BOOL
BOOL---|R    |
INT---|PV CV|---INT
+-----+

```

Up counter, contatore in avanti. Con un fronte di salita su R, azzerava il termine del conteggio (CV), poi conta i fronti di salita su CU. Quando il valore di conteggio raggiunge PV, Q va alto.

```

+-----+
|  CTD  |
BOOL--->CU   Q|---BOOL
BOOL---|LD    |
INT---|PV CV|---INT
+-----+

```

Down counter, contatore all'indietro. Con un fronte di salita su LD, inizializza il contatore(CV) al valore di PV, e lo decrementa ad ogni fronte di salita di CU. Quando il valore di conteggio si annulla, Q va alto.

Ing. Elena Mainardi

PLC



## **POU: program**

- I programmi (program) sono “contenitori” di alto livello gerarchico del codice dell'applicazione.
- Il processo di definizione ed istanziiazione è analogo a quello dei function block.
- Caratteristiche del program (vs. il function block):
  - Il PROGRAM può contenere variabili ad indirizzamento diretto (AT).
  - I programmi possono contenere definizioni di variabili globali.
  - I programmi possono contenere access variable (VAR ACCESS), per la comunicazione con dispositivi remoti.
  - I programmi non possono contenere istanze di altri programmi. Istanze di programmi sono dichiarati solo all'interno di RESOURCE.

In sostanza un program è la forma più estesa di POU, e viene dichiarata a livello di risorsa.

Concettualmente il program può essere considerato analogo al function block.

### **Esempio: definizione di un program a livello di resource**

```
PROGRAM fermenter
    (* variable declaration *)
    VAR_INPUT (* Program inputs *)
        Reagent_Code : INT;
        Sterilise : BOOL;
        Ferment_Period : TIME;
    END_VAR
    VAR_OUTPUT (* Program output *)
        Yeld : REAL;
        status : WORD;
    END_VAR
    VAR (* Internal variables and function blocks *)
        pH Loop, Temp Loop : PID;
        (* Body not shown, but can be ST, FBD, LD, SFC or IL *)
    END_PROGRAM
```

### **Esempio: dichiarazione del program sopra definito**

```
PROGRAM Line1 : fermenter ( Reagent_Code := A1,
                            Sterilise := A2,
                            Ferment_Period := FTIME,
                            Yeld => AJ_43, Status => KX56 )
```

N.B. “Line1” sarà un program di tipo “fermenter”. Al suo parametro di ingresso Reagent\_Code associa la variabile A1, a Sterilise associa A2, a Ferment\_Period FTIME. La sua uscita Yeld andrà nella mia variabile AJ\_43, mentre Status andrà alla variabile KX56

Ricapitolando, delle tre POU il Program è quella più generale e con meno restrizioni, la Function è quella più restrittiva e più dedicata ad una singola operazione specifica

Tipo di variabile	PROG	FB	FUN
VAR	1	1	1
VAR_INPUT	1	1	1
VAR_OUTPUT	1	1	0
VAR_IN_OUT	1	1	0
VAR_EXTERNAL	1	1	0
VAR_GLOBAL	1	0	0
VAR_ACCESS	1	0	0

Le funzioni non possono avere variabili IN\_OUT né OUTPUT.

La norma, inizialmente, diceva diversamente, ma dipende dall'implementazione che ne fa ogni ambiente di sviluppo per PLC.

Ad esempio Codesys in effetti non consente ad una funzione di avere VAR IN\_OUT né VAR\_OUTPUT

→ Occhio, possibile domanda d'esame: quali sono le POU? Saperne elencare le differenze←

## Tasks

- Un Task è un elemento per il controllo di esecuzione di un POU
- L'esecuzione può avvenire su base periodica o in corrispondenza del fronte di salita di una variabile booleana.

Non è detto che ad un programma debba essere obbligatoriamente associato un task, ma un programma senza task associato ha la più bassa priorità.

## Dichiarazione del Task

- SINGLE: l'esecuzione viene agganciata al fronte di salita di una variabile booleana.
- INTERVAL: l'esecuzione è periodica con intervallo specificato. Se l'intervallo specificato è zero, il task viene ripetuto ciclicamente senza essere agganciata ad un periodo preciso.
- PRIORITY: specifica la priorità del task. 0 è la priorità più elevata.

## Esempi

- Task periodica con periodo  $t = 30ms$ , priorità massima:  
`TASK Fast_Interlocks(INTERVAL :=t#30ms, PRIORITY :=0);`
- Task associata ad un evento, fonte di salita di LogFlag:  
`TASK log_task(SINGLE :=LogFlag, PRIORITY :=3);`

## Assegnazione di un POU ad una TASK

Siccome il task è l'elemento che manda in esecuzione un codice, secondo determinate modalità (o quando si verifica un evento, o ogni tot istanti di tempo...) ci dovrà essere un modo di dire quale POU associare a quale task.

- Definisco due tasks:  
`TASK SLOW_1(INTERVAL := t#20ms, PRIORITY := 2) ;`  
`TASK FAST_1(INTERVAL := t#10ms, PRIORITY := 1) ;`
- Associo il programma P1 di tipo F alla task SLOW\_1:  
`PROGRAM P1 WITH SLOW_1 : F(x1 := %IX1.1) ;`
- Associo il programma P2 di tipo G alla task FAST\_1, assegnando contestualmente variabili di ingresso ed uscita:  
`PROGRAM P2 WITH FAST_1 : G(OUT1 => w,`  
`IN1 := %IX1.1, IN2 := %IX1.2)`

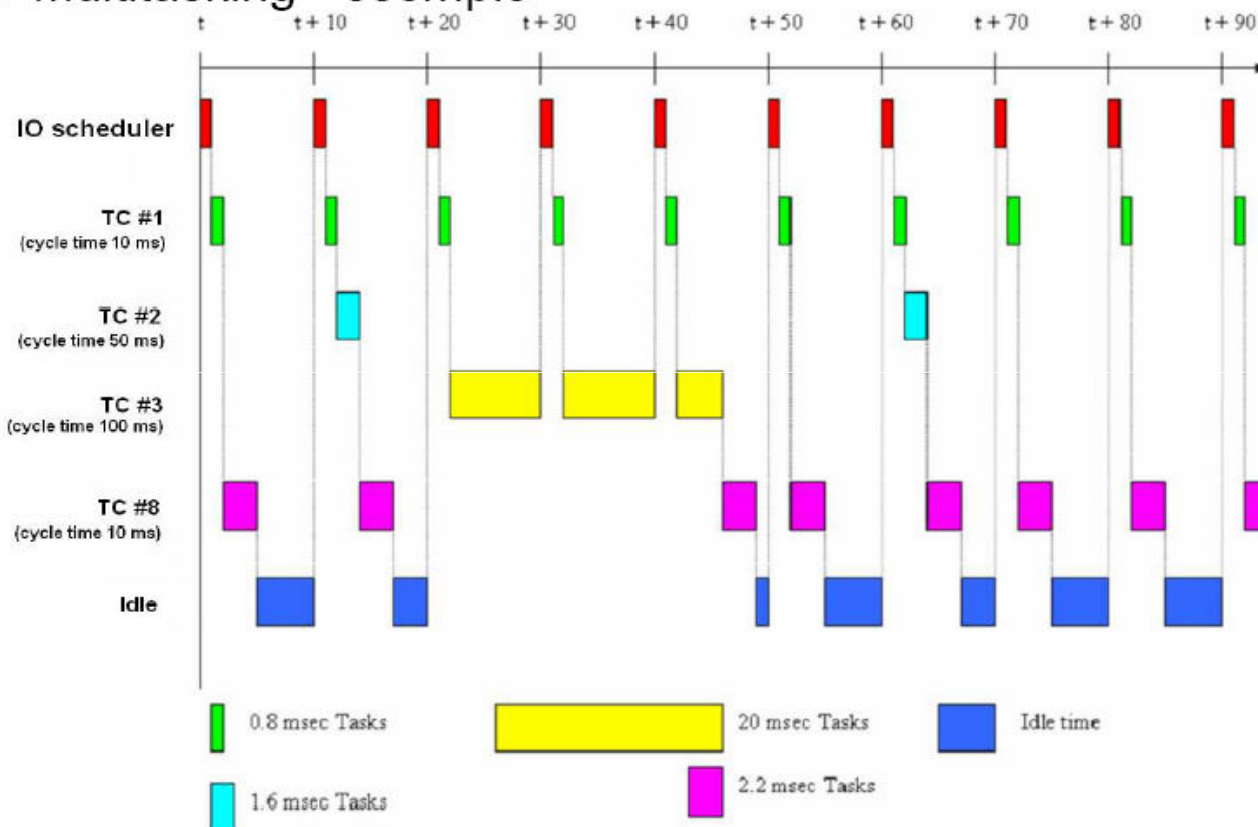
## SCHEDULING

Quando ci sono molti task dichiarati, ad ogni task è generalmente assegnato un intervallo diverso ed una priorità. L'istante di tempo in cui un particolare task viene seguito dipende dal tipo di scheduling, dalle priorità e dagli intervalli degli altri task e da quanto tempo ognuno degli altri task sia pronto ed in attesa di essere eseguito.

Un PLC può fornire due metodi di scheduling: non preemptive e preemptive.

- **Non-preemptive scheduling**: non consente l'interruzione di una task da task più prioritarie. E' semplice da implementare, ma non consente una esecuzione veramente Real Time.
- **Preemptive scheduling**: consente l'interruzione di una task da una più prioritaria. Consente di eseguire il controllo effettivamente in modalità Real Time.

### ■ Multitasking - esempio



## Un esempio completo

```

CONFIGURATION CELL_1 (* Nome della configurazione *)

    VAR_GLOBAL (* Variabili definite a livello globale
                della configurazione, visibili da
                tutte le funzioni e blocchi funzionali
                definiti nelle risorse (i.e. CPUs)
                appartenenti alla configurazione *)
        w: UINT;
    END_VAR

(* Definizione di una Resource (i.e. CPU) *)
RESOURCE STATION_1 ON PROCESSOR_TYPE_1
    VAR_GLOBAL (* Variabili definite a livello CPU *)
        z1: BYTE;
    END_VAR

    TASK SLOW_1 (INTERVAL := t#20ms, PRIORITY := 2) ;
    TASK FAST_1 (INTERVAL := t#10ms, PRIORITY := 1) ;

    PROGRAM P1 WITH SLOW_1 : F(x1 := %IX1.1) ;
    PROGRAM P2 : G(OUT1 => w, FB1 WITH SLOW_1,
                  FB2 WITH FAST_1) ;
END_RESOURCE

```

Ing. Elena Mainardi

PLC

```

RESOURCE STATION_2 ON PROCESSOR_TYPE_2 (* CPU *)
    VAR_GLOBAL
        z2      : BOOL ;
        AT %QW5 : INT  ;
    END_VAR

    TASK PER_2 (INTERVAL := t#50ms, PRIORITY := 2) ;
    TASK INT_2 (SINGLE := z2,          PRIORITY := 1) ;

    PROGRAM P1 WITH PER_2 :
        F(x1 := z2, x2 := w) ;
    PROGRAM P4 WITH INT_2 :
        H(HOUT1 => %QW5, FB1 WITH PER_2);
END_RESOURCE

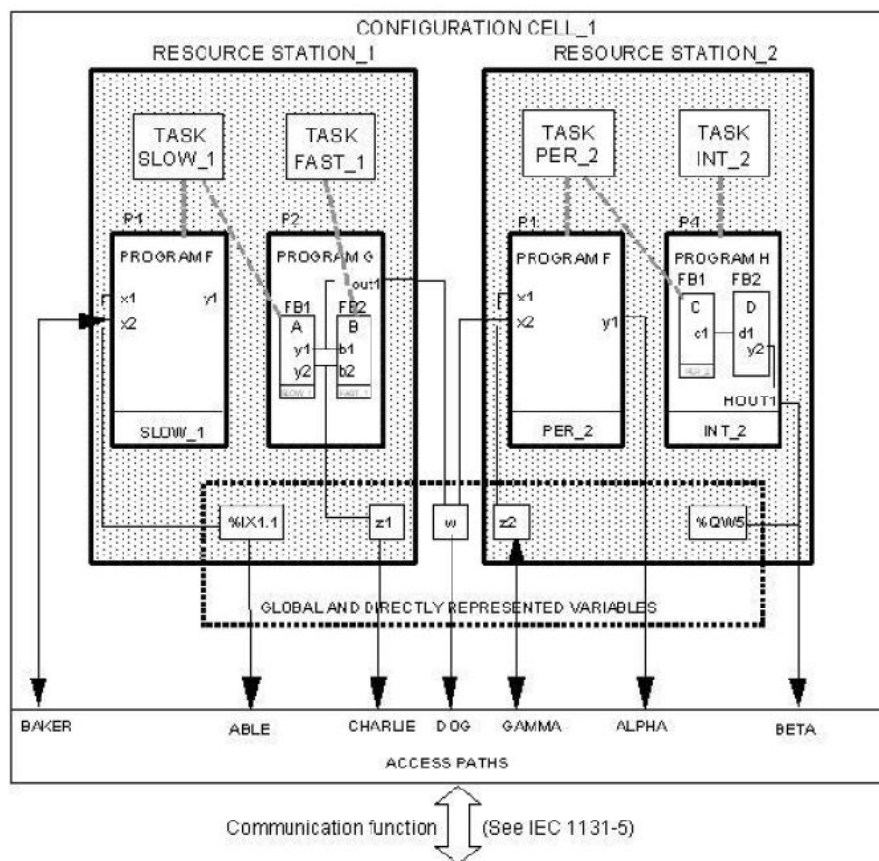
VAR ACCESS
    (* Variabili definite nella RESOURCE STATION_1 e STATION_2
       e rese disponibili in altre CONFIGURATION *)
    ABLE : STATION_1.%IX1.1 : BOOL READ_ONLY ;
    BAKER : STATION_1.P1.x2 : UINT READ_WRITE ;
    CHARLIE : STATION_1.z1 : BYTE ;
    DOG : w : UINT READ_ONLY ;
    ALPHA : STATION_2.P1.y1 : BYTE READ_ONLY ;
    BETA : STATION_2.P4.HOUT1 : INT READ_ONLY ;
    GAMMA : STATION_2.z2 : BOOL READ_WRITE ;
END_VAR

END_CONFIGURATION

```

Ing. Elena Mainardi

PLC



Ing. Elena Mainardi

PLC

## Tipologia di variabili e loro visibilità

Oltre ad esserci svariati tipi di dato, ci sono anche modi diversi di vedere una variabile, esattamente come in c.

Infatti una variabile può avere differenti visibilità, ed essere di tipo locale, o globale, o, nel caso dei linguaggi per PLC, una variabile di comunicazione tra più parti del programma etc etc

**Che tipo di variabili esistono, in base alla loro visibilità e alle loro caratteristiche?**

**Access path:** sono percorsi di comunicazione tra le diverse configurations ai quali è possibile accedere attraverso un set di variabili dichiarate usando il costrutto VAR\_ACCESS. Sono una interfaccia tra la configuration in cui sono dichiarate ed eventuali configurations remote

**Variabili globali:** possono essere dichiarate a livello di Configuration, Resource o Programs.

- Le variabili globali definite all'interno di Configuration, Resource o Programs, sono visibili in tutti i moduli che essi contengono, purché definite EXTERNAL:

```
VAR_GLOBAL
  <NOME> : <TIPO> ;
  <NOME> : <TIPO> := <VALORE INIZIALE> ;
END_VAR
```

Le variabili globali debbono essere di tipo EXTERNAL per poter essere utilizzate nei moduli contenuti \*\*.

```
VAR_EXTERNAL
  <NOME> : <TIPO> ;
  <NOME> : <TIPO> ;
END_VAR
```

**NOTA:** La visibilità delle variabili è controllabile dal programmatore, evita errori, permette dichiarazioni di variabili locali con gli stessi nomi.

\*\*Quindi, se definisco una variabile come global in resource e poi la voglio usare in un programma istanziato nella resource, nel programma la devo dichiarare come external.

## Variabili ad indirizzamento diretto

- In quasi tutti i PLC la memoria è indirizzabile con accesso diretto tramite indirizzo.
- La memoria del PLC è considerata suddivisa in tre regioni principali:
  - (I) input
  - (Q) output
  - (M) registri di memoria interna

Le variabili ad indirizzamento diretto sono quindi indicate mediante una codifica a tre simboli:

%		X (bit)	Indirizzo fisico
	I	B (byte)	
	Q	W (word)	
	M	D (Double word)	
		L (Long word)	

Ing. Elena Mainardi

PLC

Il simbolo X (bit) è il valore di default.

- L'indirizzo fisico della variabile è costituito da uno o più campi separati da un punto. La norma non specifica il significato di questi campi.
- Esempi:
  - %I100 (\* Bit 100 di ingresso \*)
  - %IX100 (\* Bit 100 di ingresso \*)
  - %IW122 (\* Word 122 di ingresso \*)
  - %IW10.1.21 (\* Potrebbe essere rack 10, modulo 1, canale 21 \*)
  - %QL100 (\* Long word 100 di output \*)
  - %MW132 (\* Word 132 di memoria interna \*)

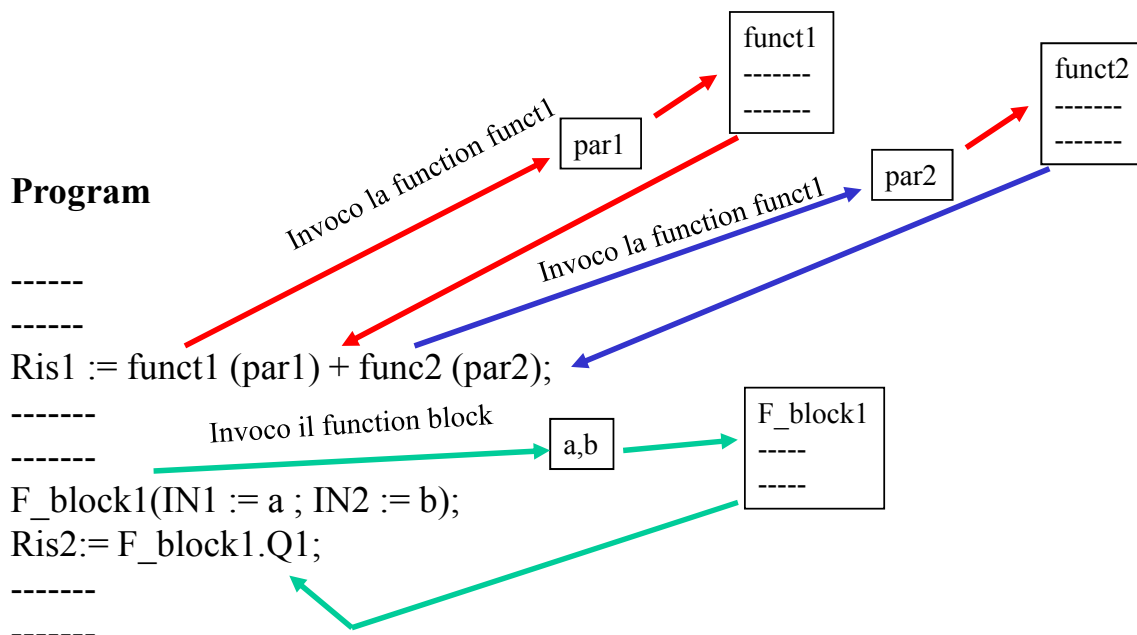
Ing. Elena Mainardi

PLC



## Variabili per il passaggio di parametri

- Parametri formali: rappresentano **l'interfaccia** di scambio dati tra una POU e il resto dell'applicazione.



## Variabili per il passaggio di parametri

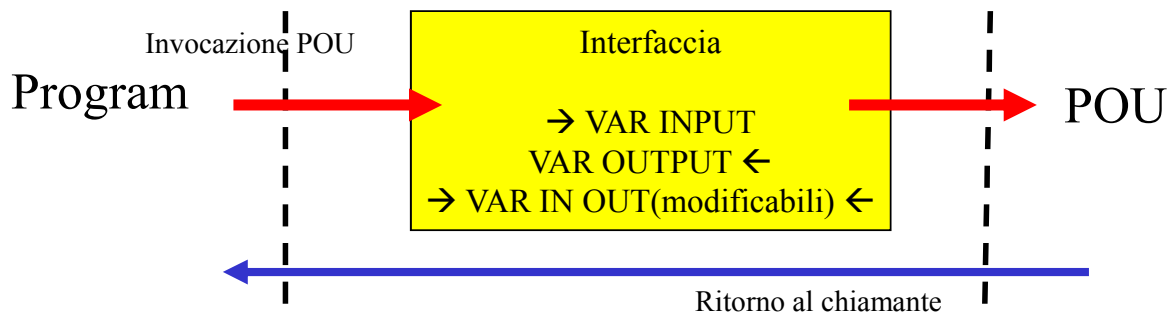
Sintassi di dichiarazione:

– VAR INPUT .. END VAR: parametri di ingresso, il loro valore viene ricevuto dall'esterno all'atto della chiamata della POU e non può essere modificato.  
(=> passaggio per valore).

– VAR OUTPUT .. END VAR: parametri di uscita, il loro valore viene elaborato all'interno della POU e fornito all'esterno. Chi chiama la POU deve copiarlo in una propria variabile.

– VAR IN OUT .. END VAR: parametri di ingresso/uscita, il loro valore viene ricevuto dall'esterno della POU, ma questa lo può modificare.  
(=> passaggio per riferimento).

## Variabili per il passaggio di parametri



## Variabili locali

- Sono dichiarate a livello di POU, sono accessibili solo nelle istruzioni della POU che le dichiara, non a quelle che essa contiene.

- Sintassi:

```
VAR
  AVERAGE_SPEED : REAL;
  Inhibit : REAL;
END_VAR
```

- Le variabili sono statiche (il valore permane tra una chiamata e l'altra di una POU) se definite in PROGRAMS e FUNCTION BLOCKS.
- Le variabili sono volatili (il valore non permane tra una chiamata e l'altra di una POU) se definite in FUNCTIONS.

## Attributi per le variabili

Gli Attributi per le variabili sono opzioni da aggiungere dopo la parola chiave che identifica il tipo di variabile.

Questi attributi possono essere:

- **RETAIN**: indica che il valore delle variabili dichiarate in tale sezione deve essere ricordato anche in caso di caduta di alimentazione del PLC (Memoria “tamponata” da batteria.)
- **CONSTANT**: il loro valore non può essere modificato nel programma. Nella dichiarazione occorre quindi anche inizializzarle al valore desiderato.
- **AT**: seguito da un indirizzo fisico della memoria di sistema, indica che il nome della variabile rappresenta un alias per l’indirizzamento diretto.

N.B.: le variabili non indirizzate direttamente, se non hanno l’attributo AT, vengono allocate nella memoria di lavoro del PLC.

→ Occhio, possibile domanda d’esame: saper spiegare visibilità e attributi delle variabili←

## Esempi

```
VAR RETAIN (* Mantiene le variabili nell'avviamento a caldo *)
  Pz_counter : LINT;
END_VAR
```

```
VAR CONSTANT (* Definisce costanti iniziali *)
  StartUp_speed : REAL := 12.3;
END_VAR
```

```
VAR (* Definisce l'indirizzo fisico di sensori ed attuatori *)
  TEMPERATURE_SENSOR AT %IW10 : WORD;
  DIG_OUTPUT AT %QX120: ARRAY[0..15] OF BOOL; (* definisce gli
    indirizzi delle uscite digitali a partire
    dall'indirizzo 120 *)
END_VAR
```

## Linguaggi di programmazione di un PLC (definiti dallo Standard IEC 61131-3)

I linguaggi della Norma sono quattro, due dei quali con sintassi testuale e due con sintassi grafica.

### Sintassi testuale

- Instruction List , IL
- Structured Text, ST

### Sintassi grafica

- Ladder Diagram, LD
- Function Block Diagram, FBD

Infine esiste anche **SFC** (sequential function chart), che di fatto è il quinto linguaggio

## Linguaggio Ladder Diagram

- Linguaggio grafico definito secondo lo stile del metodo di programmazione di PLC maggiormente diffuso nel mondo: la logica ladder o linguaggio a Contatti
- Formalismo di successo per **motivi storici**: significato istruzione ladder , funzionamento quadri elettromeccanici a bobine e relè.
- Rappresentazione grafica di flusso virtuale di corrente tra due barre di potenziale:
  - Passa corrente → valore logico TRUE
  - Non passa corrente → valore logico FALSE

↓

**Per motivi storici** in quanto i PLC sono l'evoluzione delle macchine a relè e bobine. Il ladder sfrutta proprio un simbolismo grafico basato su come, una volta, venivano progettate queste macchine. E' stato definito in modo tale che i vecchi progettisti di macchine automatiche riuscissero a programmare i PLC, basandosi su come avrebbero impostato lo schema elettrico della macchina stessa

## Linguaggio Ladder Diagram: contatti

Simbolo	Descrizione
*** -   -	<i>Contatto normalmente aperto</i> La corrente scorre solamente se il valore della variabile booleana indicata da *** è vero.
*** - / -	<i>Contatto normalmente chiuso</i> La corrente scorre solamente se il valore della variabile booleana indicata da *** è falso.
*** - P -	<i>Contatto rilevatore di fronti di salita</i> La corrente scorre solamente se il valore della variabile booleana indicata da *** è passato da un valore falso nella scansione precedente del programma ad un valore vero in quella attuale.
*** - N -	<i>Contatto rilevatore di fronte di discesa</i> La corrente scorre solamente se il valore della variabile booleana indicata da *** è passato da un valore vero nella scansione precedente del programma ad un valore falso in quella attuale.

Ing. Elena Mainardi

PLC

## Linguaggio Ladder Diagram: bobine I

Simbolo	Descrizione
*** -( )-	<i>Bobina</i> La variabile booleana *** assume valore vero (per la sola durata della scansione corrente del programma), se l'espressione booleana a sinistra della bobina ha valore vero
*** -(/)-	<i>Bobina negata</i> La variabile booleana *** assume valore falso (per la sola durata della scansione corrente del programma), se l'espressione booleana a sinistra della bobina ha valore vero.
*** -(S)-	<i>Bobina con ritenuta</i> La variabile booleana *** assume valore vero (e lo mantiene fino ad un esplicito reset), se l'espressione booleana a sinistra della bobina ha valore vero.
*** -(R)-	<i>Bobina con ritenuta negata</i> La variabile booleana *** assume valore falso, se l'espressione booleana a sinistra della bobina ha valore vero

È come dire  
Set Reset.  
Altrove le si  
può trovare  
indicate come  
latch unlatch

## Linguaggio Ladder Diagram: bobine II

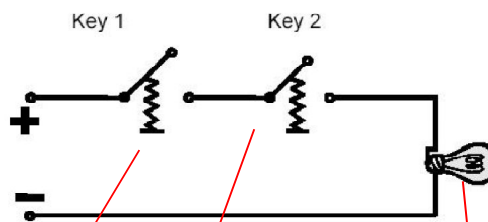
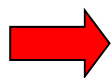
Simbolo	Descrizione
*** -(P)-	<i>Bobina per fronti di salita</i> Se l'espressione booleana a sinistra della bobina passa da un valore falso nella precedente scansione del programma ad un valore vero in quella attuale la variabile booleana *** assume valore vero (solo per la durata della scansione corrente).
*** -(N)-	<i>Bobina per fronti di discesa</i> Se l'espressione booleana a sinistra della bobina passa da un valore vero nella precedente scansione del programma ad un valore falso in quella attuale la variabile booleana *** assume valore vero (solo per la durata della scansione corrente).

Ing. Elena Mainardi

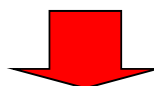
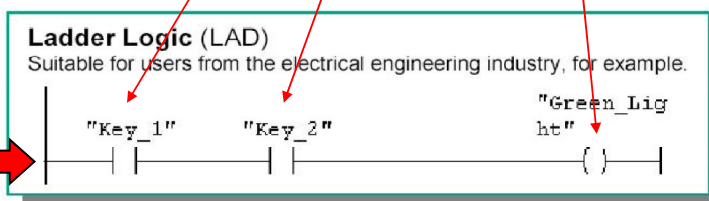
PLC

### Programmazione in Ladder Diagram

Questo circuito elettrico a relè (key1 e key2) e bobina (l'attuatore che fa accendere la lampadina)...

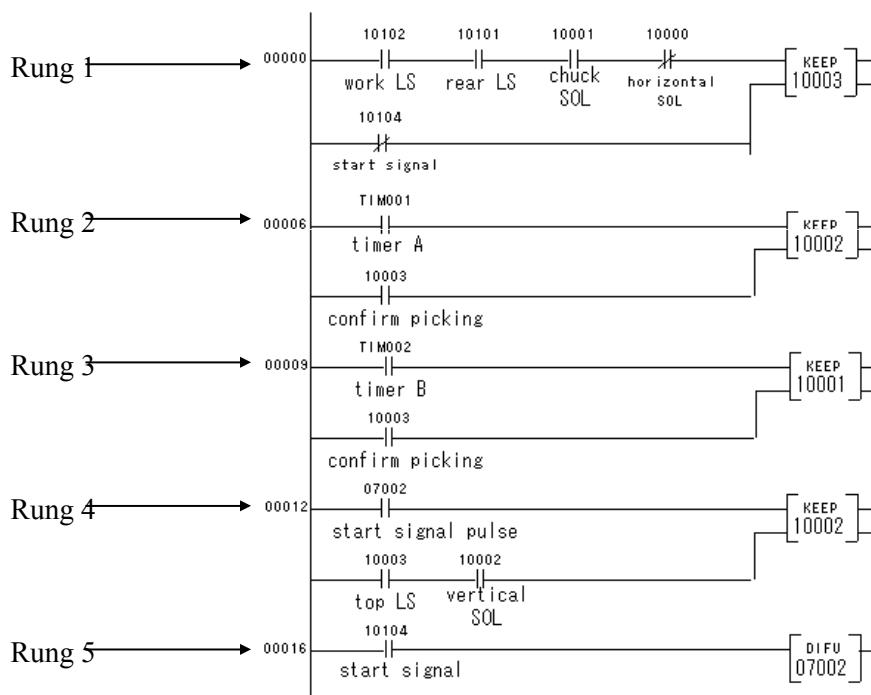


...corrisponde a questa "riga di codice grafico" in ladder



Se key1 è premuto e key2 è premuto allora attiva la bobina Green\_light

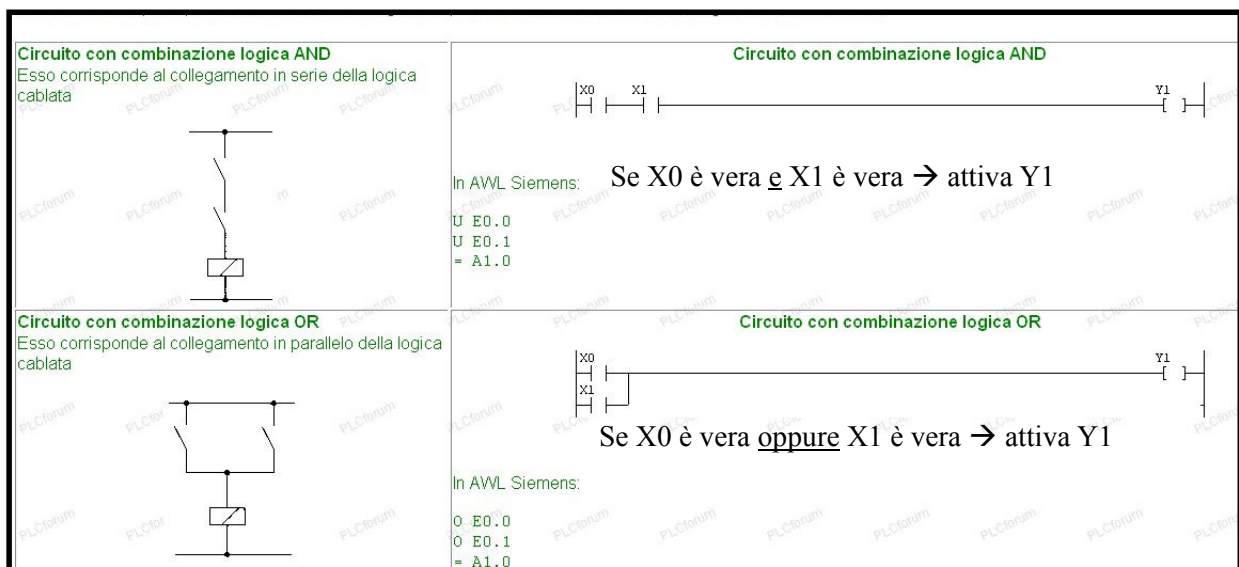
Ladder significa letteralmente "scala a pioli", dato che esteticamente lo schema ricorda appunto una scala; nel mondo anglosassone ogni ramo orizzontale viene chiamato *rung*, ossia piolo.



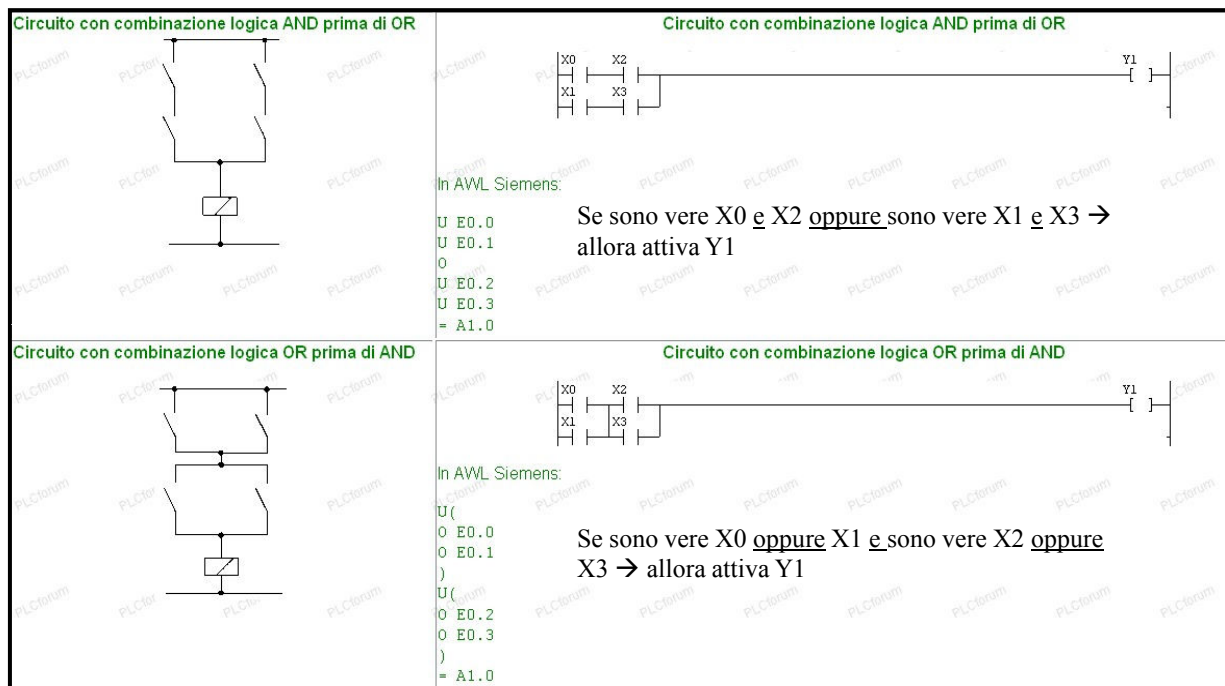
- Un programma scritto in linguaggio Ladder viene eseguito valutando un rung alla volta
- L'ordine di valutazione dei rung è quello che procede dal primo rung in alto verso l'ultimo rung in basso
- Ogni rung è valutato da sinistra a destra
- Quando l'ultimo rung viene valutato, si inizia nuovamente a valutare il primo rung (dopo aver aggiornato le uscite e letto gli ingressi)

Per eseguire una AND tra due ingressi basta metterli consecutivamente sulla stessa riga

Per eseguire una OR tra due ingressi bisogna metterli in parallelo

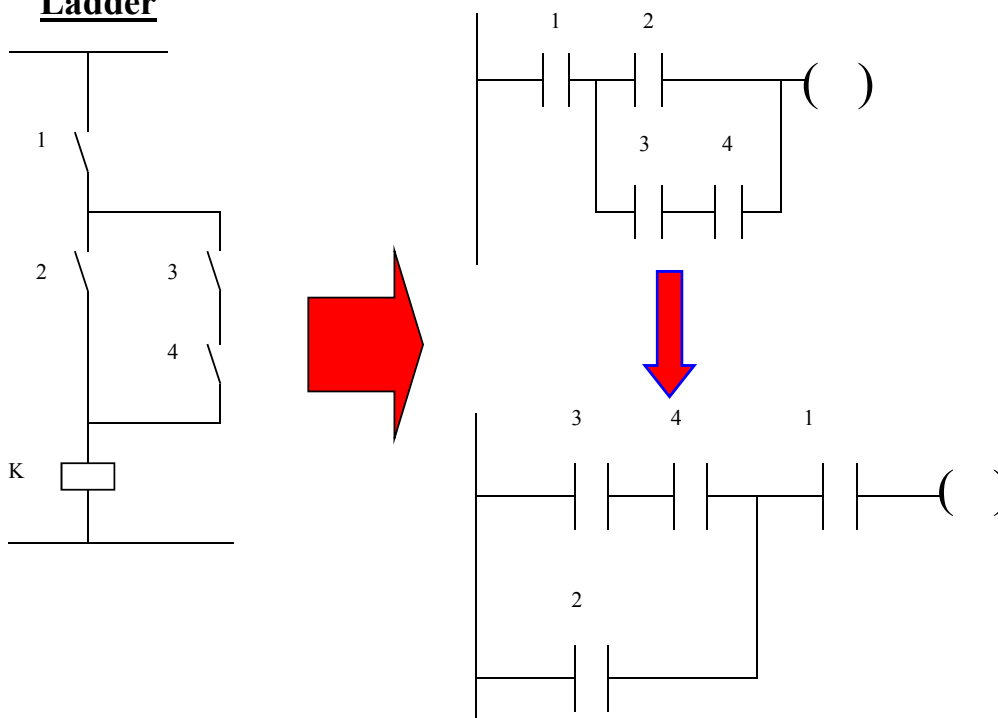


## Reti logiche complesse



Ing. Elena Mainardi

PLC

Ladder

Ing. Elena Mainardi

PLC



## Ladder

- Non esiste limite al numero dei contatti da inserire nelle linee di combinazione (memoria permettendo)
- Tutti gli output possono essere utilizzati come contatti di input (via software)
- Nessun contatto può essere disposto a destra dell'output

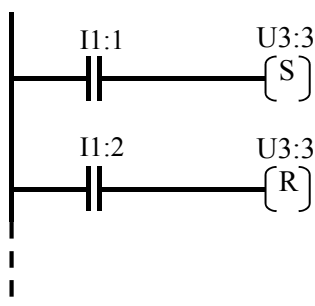
## Esempi di network Ladder Diagram

```

| Automatic                      Emergency Drive1_OK Motor1_On |
+----| |-----+---|/|-----| |----- ( )----+|
|                               |                               |
| Manual Nastro1_Man_Cmd |                               |
+----| |-----| |-----+                               |
|                               |                               |

```

Se (Automatic è vero oppure sono veri Manual e Nastro1\_man\_cmd) e Emergency è falso e Drive1\_ok è vero → allora attiva la bobina Motor1\_on

LadderBobine con ritenuta

Rendendo attivo l'ingresso I1:1 del PLC si eccita l'uscita costituita dalla bobina con ritenuta U3:3.

A questo punto la bobina rimane attiva (eventualmente anche in altre scansioni cicliche del programma) finchè non diventa attivo l'ingresso I1:2, che la "resetta", cioè la disattiva

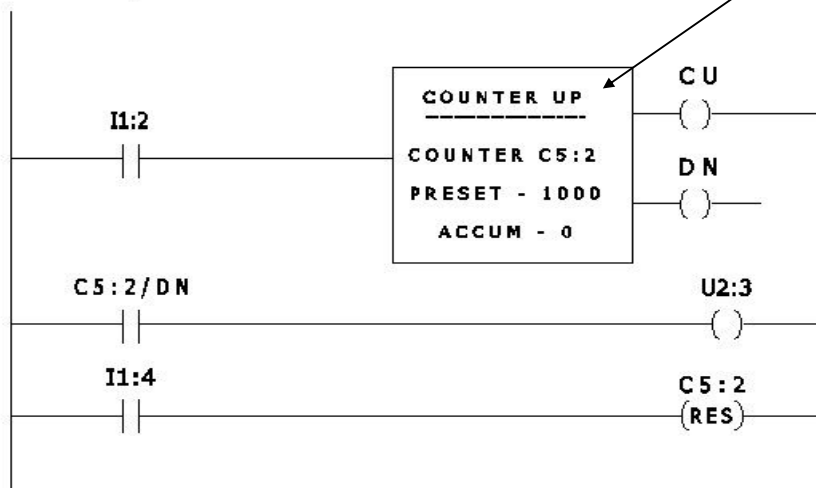
Ing. Elena Mainardi

PLC

Nel ladder, oltre che semplici funzioni di AND e OR tra variabili, posso richiamare **funzioni e function block**. Gli schemi diventano ovviamente più complessi

**Istruzioni di conteggio**

- CTU counter up contatore ad incremento



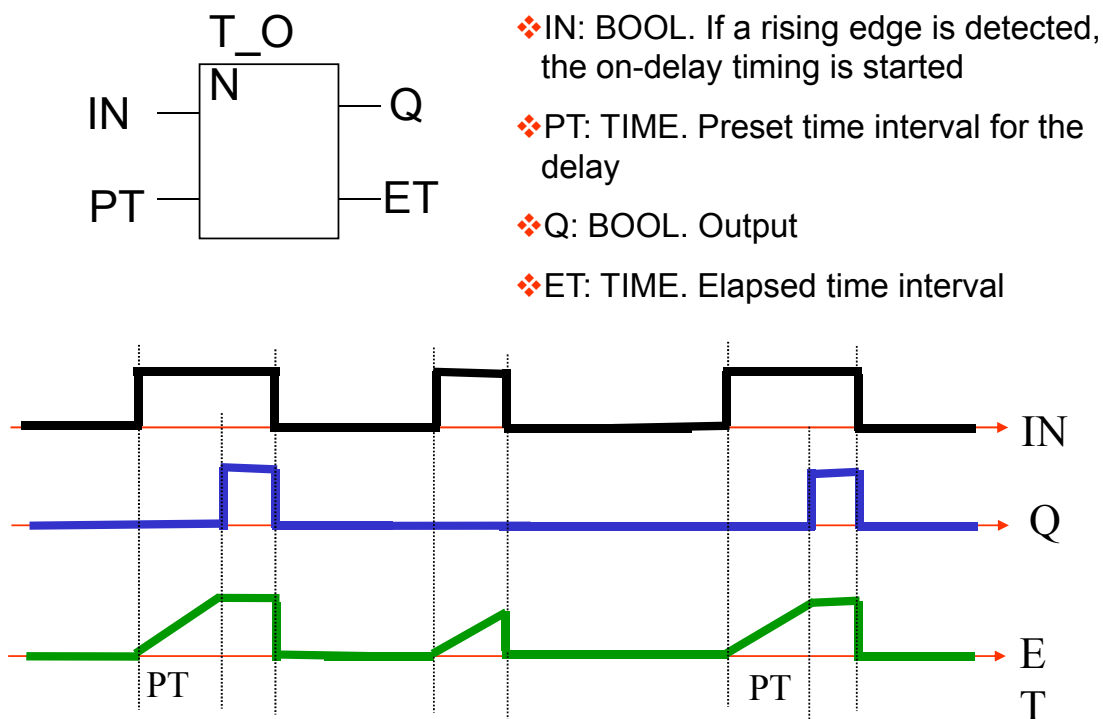
Blocco  
funzionale di  
conteggio in  
avanti, chiamato  
C5:2

Ad ogni impulso sull'ingresso I1:2 del PLC viene conteggiata un unità su ACCUM, resettabile a 0 tramite l'ingresso I1:4. Quando si raggiunge il livello impostato su PRESET (nell'esempio 1000) si attiva DN che aziona C5:2/DN e fa eccitare l'uscita U2:3; anche in quest'ultimo caso il RESET, azionato sempre dall'ingresso I1:4, riporta a 0 il conteggio e disabilita l'uscita DN.

Ing. Elena Mainardi

PLC

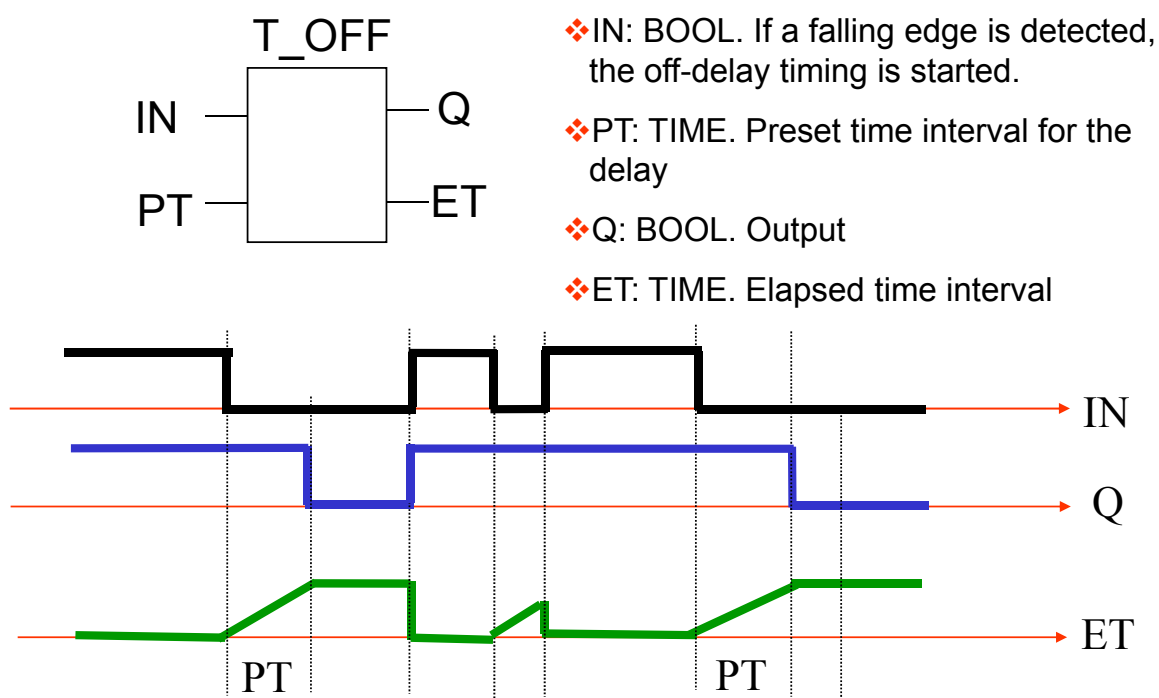
## Function Block di Uso Comune: Timer T\_ON



Ing. Elena Mainardi

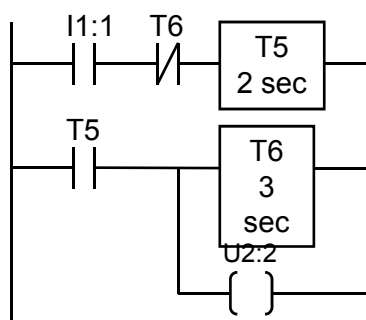
PLC

## Function Block di Uso Comune: Timer T\_OFF



Ing. Elena Mainardi

PLC

**Esempio: realizzazione di un oscillatore ad onda quadra tramite timers Ton**

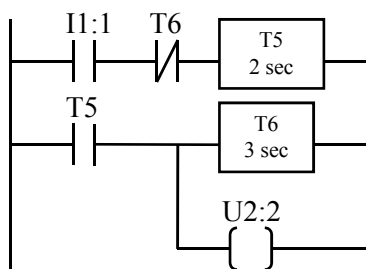
Blocco funzionale temporizzatore T5

Blocco funzionale temporizzatore T6

Nell'esempio si realizza un oscillatore caratterizzato da un'uscita con livello logico alto per 3 secondi e basso per 2 secondi (quindi il periodo è  $T = 3 + 2 = 5$  s e il duty cycle è  $Ton/T = 3/5 = 60\%$ )

Ing. Elena Mainardi

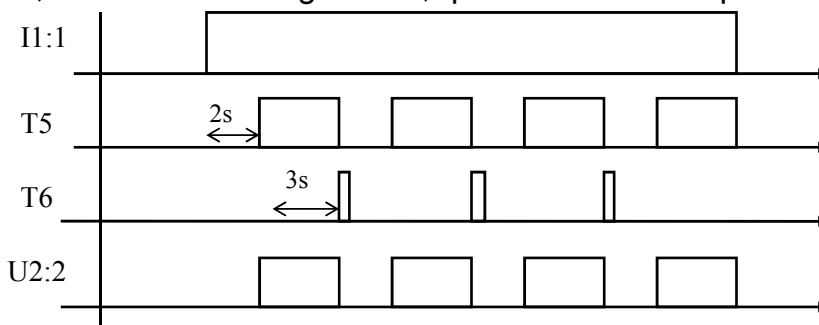
PLC



All'inizio è tutto a valore logico false (0) → né i timers né l'uscita U2:2 sono attivi. Poi ad un certo punto l'ingresso I1:1 si attiva → siccome T6 è "spento" e il suo valore, sul primo rung, è in logica negata (|/|) → si attiva il timer T5, che conta per 2 secondi, durante i quali la sua uscita rimane bassa.

Al termine dei 2 secondi, l'uscita di T5 va a valore logico alto, quindi attiva, sul secondo rung, il timer T6, che conta per 3 secondi, durante i quali la sua uscita rimane bassa.

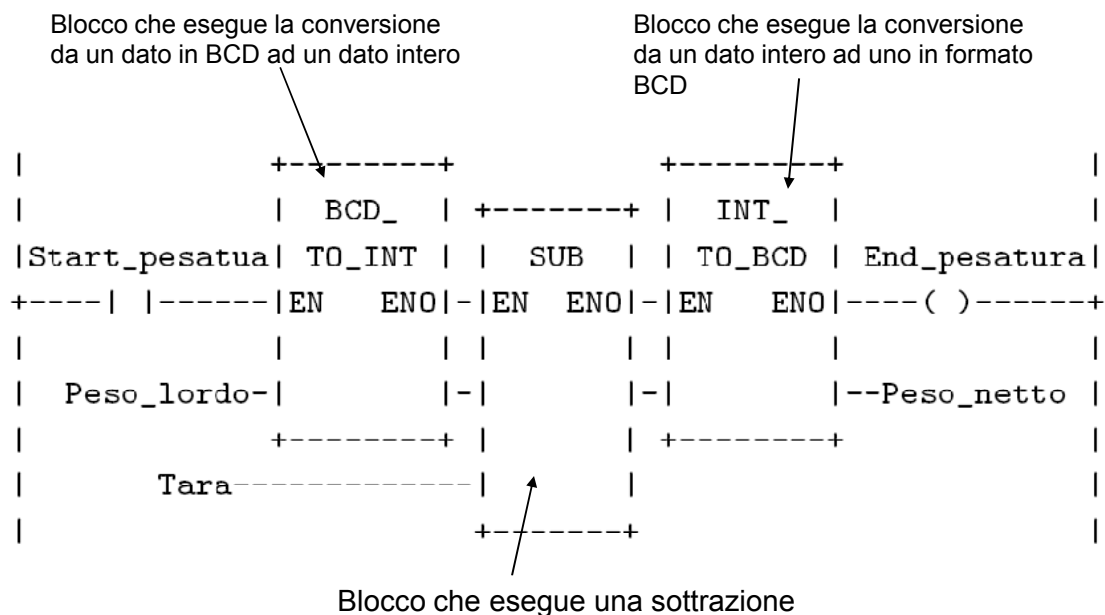
Al termine dei 3 secondi, T6 va a valore logico alto, quindi disattiva sul primo rung T5, e così via.



Ing. Elena Mainardi

PLC

## Esempio ladder con chiamata a function block



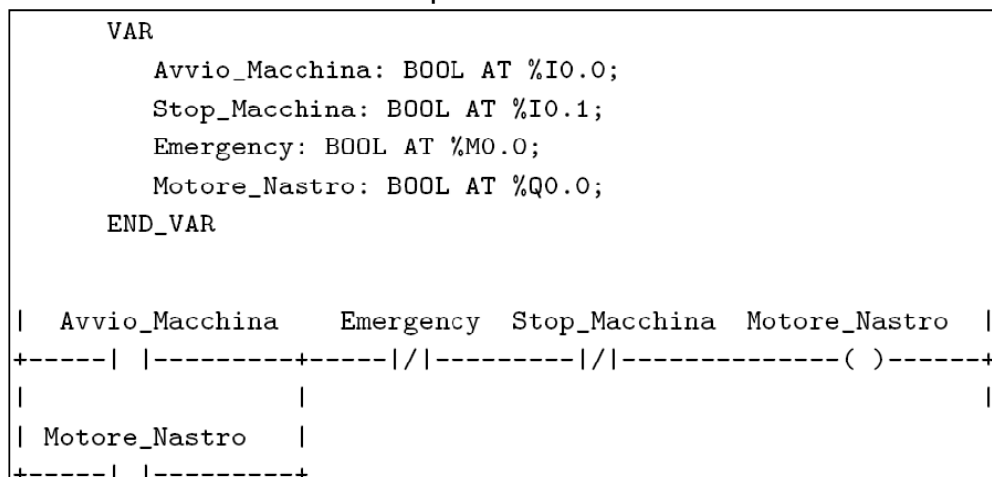
Se Start\_pesatura è vera allora converti Peso\_lordo in intero, sottraigli Tara, riconverti il risultato in un BCD, metti il risultato in Peso\_netto e attiva 'uscita End\_pesatura

Ing. Elena Mainardi

PLC

## Esempio ladder: controllo dell'avviamento di dispositivi di attuazione

- Molte delle operazioni tipiche del controllo di Macchine Automatiche sono riconducibili alla abilitazione di una singola uscita booleana.
- Un esempio di tale necessità è rappresentato dalla movimentazione di nastri trasportatori, normalmente dotati di un controllo attivo/disattivo separato.
- Per tali motori è quindi sufficiente la chiusura del contatto di alimentazione tramite un segnale logico in uscita dal PLC.
- Tale segnale sarà disabilitato in funzione dell'arresto della macchina, sia esso dovuto al termine "normale" della produzione o ad una condizione di emergenza.



Ing. Elena Mainardi

PLC

```

VAR
    Avvio_Macchina: BOOL AT %IO.0;
    Stop_Macchina: BOOL AT %IO.1;
    Emergency: BOOL AT %MO.0;
    Motore_Nastro: BOOL AT %Q0.0;
END_VAR

| Avvio_Macchina      Emergency  Stop_Macchina  Motore_Nastro  |
+-----| |-----+-----|/|-----|/|----- ( )-----+
|
| Motore_Nastro      |
+-----| |-----+

```

N.B. Si noti che l'uscita Motore\_Nastro è messa anche come ingresso.

Ora, di fatto Motore\_Nastro è un'uscita, ma posso utilizzare il suo stato anche come ingresso. Quindi quando Motore\_Nastro come uscita va a 1, anche la corrispettiva Motore\_Nastro variabile andrà a 1.

Questo serve per la cosiddetta **autoritenuta**: se Avvio\_Macchina è un pulsante e non un interruttore, non posso pretendere che un operaio stia a premere il pulsante per delle ore affinché il tutto proceda. Invece così l'operaio preme una volta il pulsante (poi lo rilascia), in quell'istante il rung si attiva e, se Emergency=0 e Stop\_Macchina=0 → Motore\_Nastro (uscita) =1.

Da questo punto in poi, al successivo ciclo, se anche Avvio\_Macchina=0 (l'operaio ha lasciato il pulsante), la variabile Motore\_Nastro vale 1 (come la corrispettiva uscita che è stata attivata), quindi se Emergency e Stop\_Macchina sono false l'uscita è sempre attiva.

Ing. Elena Mainardi

PLC

```

VAR
    Avvio_Macchina: BOOL AT %IO.0;
    Stop_Macchina: BOOL AT %IO.1;
    Emergency: BOOL AT %MO.0;
    Motore_Nastro: BOOL AT %Q0.0;
END_VAR

| Avvio_Macchina      Emergency  Stop_Macchina  Motore_Nastro  |
+-----| |-----+-----|/|-----|/|----- ( )-----+
|
| Motore_Nastro      |
+-----| |-----+

```

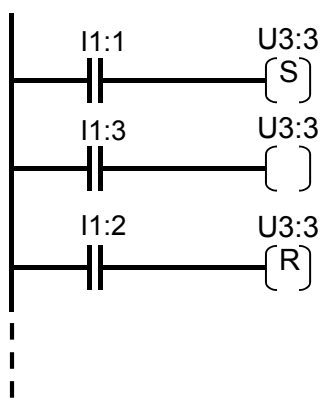
Ricordando che la scansione del programma da parte del PLC avviene in modo ciclico, e che il valore delle variabili di memoria e di uscita viene mantenuto tra due cicli di elaborazione successivi, la valutazione del segmento permette l'inserimento del motore del nastro **quando il segnale Avvio Macchina è vero anche per un solo ciclo di elaborazione**, mentre per tutti i cicli successivi verrà mantenuto grazie al passaggio di "corrente virtuale" nel contatto condizionato dal valore dell'uscita Motore Nastro (autoritenuta). Tale bobina sarà eccitata fintanto che non si ha un valore vero né dall'ingresso Stop Macchina, né dalla condizione di allarme Emergency, variabile di memoria interna del PLC modificata in altri segmenti di programma in funzione di eventuali condizioni di malfunzionamento generale della macchina.

Naturalmente, questo semplice segmento può essere modificato in maniera molto intuitiva per considerare altre condizioni necessarie all'avviamento o sufficienti per lo spegnimento, rispettivamente inserite nel ramo a monte della diramazione di autoritenuta o a valle di essa.

Ing. Elena Mainardi

PLC

### Possibili fonti di errore o di ambiguità in ladder



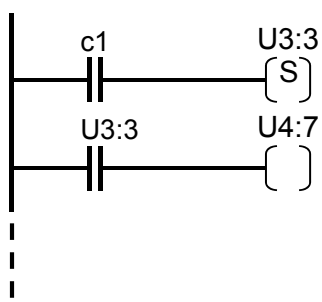
Cosa succede se uso, in un programma ladder, la stessa bobina (nel nostro caso U3:3) sia come bobina a ritenuta che come bobina impulsiva?

#### La cosa non è sintatticamente scorretta, ma è di per sé un errore logico.

Supponiamo che in un certo ciclo di programma gli ingressi siano:

I1:1 true                      I1:3 false                      I1:2 false

A questo punto, eseguendo il primo rung, la bobina U3:3 si setterebbe (cioè andrebbe a 1). Però al secondo rung, essendo I1:3 false, la bobina U3:3 va direttamente a false, pur non essendo attiva la condizione I1:2 che è legata al reset della bobina. Questo è un errore logico!



La variabile associata ad una bobina può anche essere presa come variabile

associata alla condizione di un contatto.

Nel caso di esempio, se c1=true, la variabile associata a U3:3 diventa subito vera (anche se fisicamente la bobina si attiverà solo alla fine del ciclo), quindi anche U4:7 diventerà vera, essendo la variabile U3:3 anche la condizione di attivazione o meno di U4:7. Anche U4:7, fisicamente, sarà attivata solo alla fine del ciclo

Ing. Elena Mainardi

PLC

Esempio ladder: controllo di un motore elettrico con modalità simmetrica di sicurezza  
“interblocco”

VAR

...

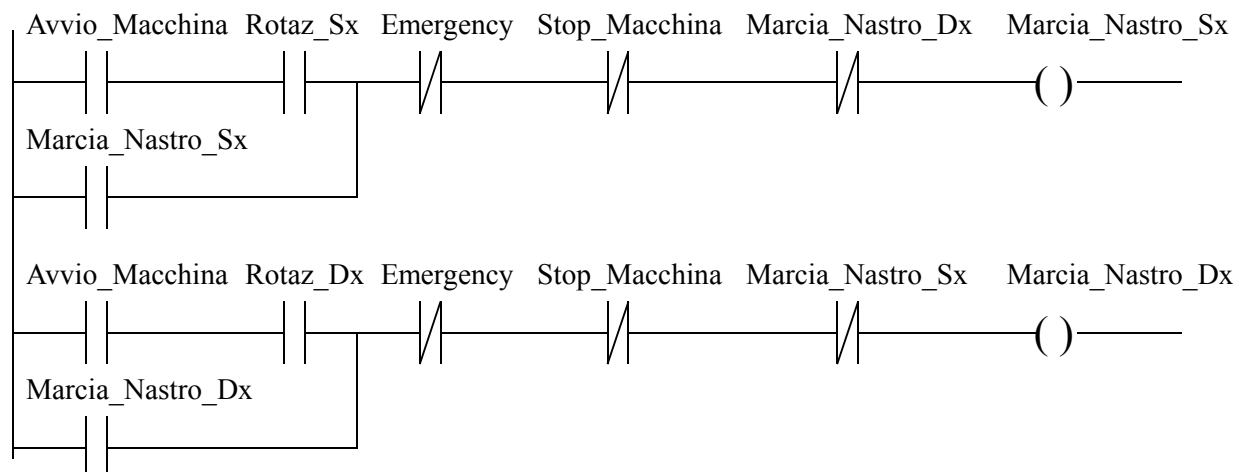
Rotaz\_Sx:BOOL AT %M0.1

Rotaz\_Dx:BOOL AT %M0.2

Marcia\_Nastro\_Sx:BOOL AT %Q0.1

Marcia\_Nastro\_Dx:BOOL AT %Q0.2

END\_VAR



Ing. Elena Mainardi

PLC

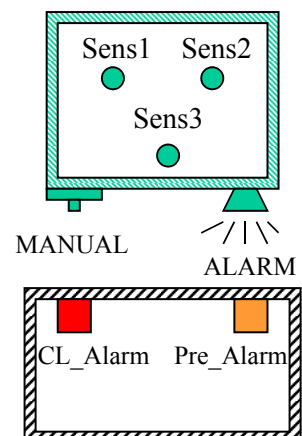
### Alimentazione di un motore in due sensi di marcia (orario e antiorario) con interblocco

**Soluzione:** controllare l'abilitazione dei comandi di rotazione oraria o anti-oraria con segmenti analoghi al precedente, ma nei quali sia inserito un "interblocco" che impedisca un'attivazione contemporanea dei due segnali (condizione che provocherebbe un cortocircuito sui terminali del motore).

Come si può notare, l'interblocco è rappresentato dal valore dell'uscita di comando della rotazione simmetrica a quella attivata dalla bobina in esame. In tal modo, è effettivamente impossibile abilitare contemporaneamente entrambe le uscite, anche se per un errore del programma vi fosse un ciclo di esecuzione nel quale le variabili booleane Rotaz Dx e Rotaz Sx fossero contemporaneamente vere.

### Esempio ladder: controllo diagnostico e rilevazione di allarmi

- Nell'esempio seguente, si considera il caso in cui il PLC debba monitorare i segnali provenienti da sensori che rilevano la presenza di un incendio in una zona a rischio.
- Nella zona da controllare sono presenti tre sensori identici, posizionati in punti opportuni, e si desidera che nel caso in cui uno solo di questi sensori fornisca un segnale logico vero, venga avviata una segnalazione di "pre-allarme" con l'accensione di un segnale luminoso. Tale segnalazione può essere spenta se il sensore non fornisce più un valore vero.
- Se invece, più sensori sono attivi contemporaneamente, allora verrà generata una condizione di allarme e di arresto immediato della parte operativa, condizione che potrà essere resettata solamente attraverso un intervento dell'operatore umano.

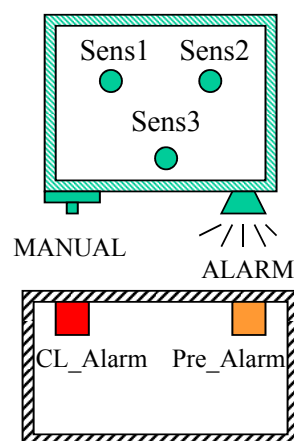




VAR

```
Sens1:BOOL AT %IO.0;
Sens2:BOOL AT %IO.1;
Sens3:BOOL AT %IO.2;
MAN1:BOOL AT %IO.3; (* E' possibile anche una segnalazione
                    manuale di incendio *)
CL_Alarm:BOOL AT %IO.4; (* Reset manuale della condizione
                        di allarme *)

Alarm:BOOL AT %Q0.0;
Pre_alarm:BOOL AT %Q0.1;
END_VAR
```



Ing. Elena Mainardi

PLC

```
(* Verifica delle condizioni di pre-allarme *)

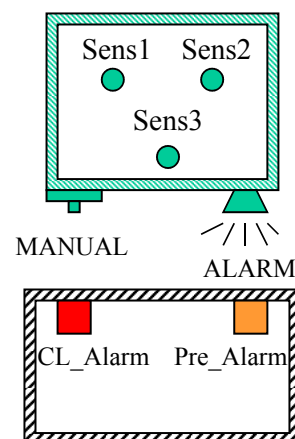
|      Sens1                      Pre_alarm
+-----| |-----+------( )-----|
|
|      Sens2                      |
+-----| |-----+
|
|      Sens3                      |
+-----| |-----+

(* Verifica delle condizioni di allarme generale *)

|      Sens1  Sens2                      Alarm
+-----| |-----| |------(S)-----|
|
|      Sens1  Sens3                      |
+-----| |-----| |-----+
|
|      Sens2  Sens3                      |
+-----| |-----| |-----+
|
|      MAN1                      |
+-----| |-----+

(* Reset manuale della segnalazione di allarme *)

|      CL_Alarm                      Alarm
+-----|P|------(R)-----|
|
```



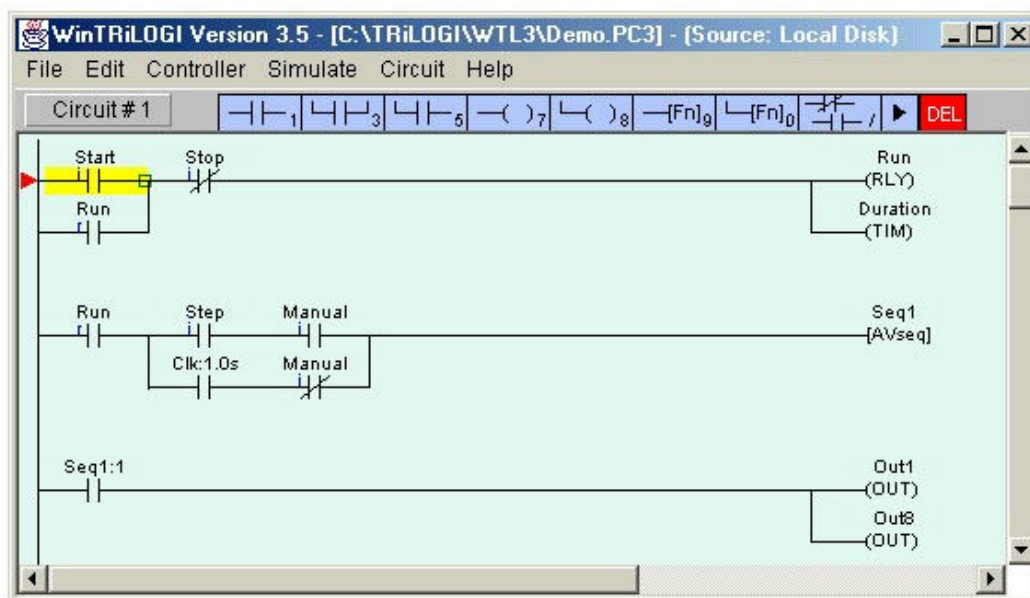
N.B.  
Si ricorda che | P |  
è un rilevatore di  
fronti di salita

Ing. Elena Mainardi

PLC

## Come scrivo, fisicamente, un programma in ladder?

Tramite degli editor grafici con già preimpostati i blocchi per le variabili di ingresso, gli interruttori, le bobine, i function blocks etc

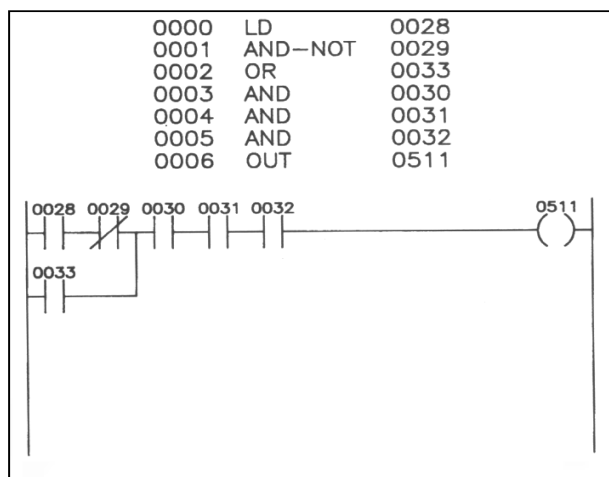


Ing. Elena Mainardi

PLC

## Linguaggio Instruction list (IL)

E' un linguaggio testuale di basso livello con una struttura simile all'assembler. IL è un linguaggio adatto alla soluzione di semplici problemi e alla produzione di codice ottimizzato, ma non supporta la programmazione strutturata. Il linguaggio IL può essere interpretato direttamente da molti PLC conformi alla norma IEC1131-3. E' proprio per questo motivo che tale linguaggio è talvolta considerato *il linguaggio PLC* nel quale si possono tradurre tutti i linguaggi conformi alla norma IEC1131-3.



Esempio di programma scritto in IL e sua corrispondenza con ladder

Ing. Elena Mainardi

PLC

## Linguaggio Instruction list (IL)

Definizione della funzione di nome funz\_IL, scritta in Instruction List, che ritorna un intero

```

FUNCTION funz_IL : INT
VAR_INPUT                                (*nessuna variabile passata dal chiamante*)
END_VAR
VAR
    Var1: INT;
END_VAR

    LD      7                            (*corpo della funzione*)
    ADD     2,4,7
    ST      Var1
    ST      funz_IL                      (*valore di ritorno*)
  
```

Invocazione da un program:

ris\_funz\_IL: INT; (\*dichiarazione della variabile di ritorno\*)

ris\_funz\_IL:= funz\_IL(); (\*chiamata alla funzione\*)

## Linguaggio Instruction list (IL): possibili istruzioni

### Operator Meaning

LD	Make current result equal to the operand
ST	Save current result at the position of the operand
S	Then put the Boolean operand exactly at TRUE if the current result is TRUE
R	Then put the Boolean operand exactly at FALSE if the current result is TRUE
AND	Bitwise AND
OR	Bitwise OR
XOR	Bitwise exclusive OR
ADD	Addition
SUB	Subtraction
MUL	Multiplication
DIV	Division
GT	>
GE	>=
EQ	=
NE	<>
LE	<=
LT	<
JMP	Jump to the label
CAL	Call program or function block or
RET	Leave POU and return to caller.

## Linguaggio Instruction list (IL)

### Un esempio di programma

```
LD      2
MUL     2
ADD     3
ST      Erg
```

(\*la variabile Erg, dichiarata come INT, varrà 7, cioè  $2 * 2 + 3$ \*)

## Linguaggio Instruction list (IL)

### Un esempio di programma

```
LD      TRUE      (*load TRUE in the accumulator*)
ANDN    BOOL1      (*execute AND with the negated value of the BOOL1 variable*)
JMPC    mark       (*if the result was TRUE, then jump to the label "mark"*)

LDN     BOOL2      (*save the negated value of *)
ST      ERG        (*BOOL2 in ERG*)
label:
LD      BOOL2      (*save the value of *)
ST      ERG        (*BOOL2 in ERG*)
```

## Linguaggio Structured Text

- Linguaggio di alto livello, pascal-like, efficace per realizzare complesse elaborazioni matematiche o gestire strutture dati particolari.
- Prevede tutti i costrutti tipici dei linguaggi strutturati:
  - Assegnazione e valutazione di espressioni
  - Selezione di alternative
  - Iterazione
- Assegnazione: Variabile := Espressione.

Variable e risultato dell'espressione devono essere compatibili per tipo.  
Es.: RES := A+B-C\*ABS(D); (\* A,B,C,D, RES variabili numeriche \*)

## Linguaggio Structured Text

Chiamate di Function Blocks si effettuano semplicemente con istruzioni tipo:

```
FB_nome_istanza(Par1 := A, Par2:=B ...);
```

- I parametri di uscita di un FB sono accessibili direttamente con:  
FB nome istanza.Out1.
- Chiamate di Functions devono essere effettuate in una espressione, in modo che il risultato della funzione venga usato esplicitamente:  
RES := Fun\_Def(Par1 := A, Par2:=B ...);

## Linguaggio Structured Text

### Costrutti di selezione

- Selezione fra due alternative:

```
IF Expr_Booleana_1
  THEN Istruzioni_1;
ELSIF Expr_Booleana_2
  THEN Istruzioni_2;
ELSE
  Istruzioni_3;
END_IF
```

- Selezione multipla:

```
CASE Expr_Interi OF
  Val_Interi_1 : Istruzioni_1;
  Val_Interi_2 : Istruzioni_2;
  ...
  ELSE Istruzioni_N
END_CASE
```

## Linguaggio Structured Text

- Costrutti di iterazione:

- REPEAT Istruzioni UNTIL Expr Bool END REPEAT;
- WHILE Expr Bool DO Istruzioni END WHILE;
- FOR Var Inter := Val Iniz TO Val Finale BY Passo DO Istruzioni  
END FOR;
- Istruzione EXIT forza l'uscita da tutti i costrutti precedenti.

N.B.: L'iterazione si intende nello stesso ciclo di esecuzione del controllore. Pertanto occorre attenzione nella programmazione per evitare loop che prolunghino il tempo di esecuzione fino alla scadenza del watchdog timer impostato.

**Operatori**

Simbolo	Operazione	Priorità
( ) fun(args) - NOT	Valutazione di funzione Negazione (matematica) Negazione (booleana)	MAGGIORE
**	Elevamento a potenza	
* / MOD	Moltiplicazione Divisione Modulo	

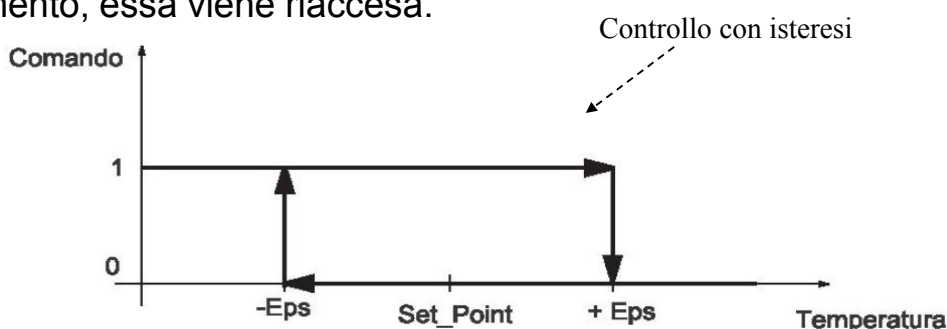
Simbolo	Operazione	Priorità
+	Somma	
-	Sottrazione	
<, <=, >, >=	Test comparativi	
=	Uguaglianza (test)	
<>	Disuguaglianza	
& o AND	AND booleano	
XOR	OR booleano esclusivo	
OR	OR booleano	MINORE

Ing. Elena Mainardi

PLC

**Esempio ST: Elaborazione di numeri reali**

- Si supponga di dover controllare la temperatura di un forno, avendo a disposizione un sensore analogico ed una serpentina comandata tramite un interruttore “acceso/spento”.
- Il tipo di controllo normalmente utilizzato in questi casi è il cosiddetto controllo on/off con isteresi.
- Il valore numerico reale fornito dal sensore viene confrontato con il valore di temperatura che si desidera mantenere.
  - Se la serpentina è accesa e questi differiscono per una quantità maggiore di un certo valore, la serpentina viene spenta.
  - Viceversa, se la serpentina è spenta e la temperatura scende oltre al di sotto del riferimento, essa viene riaccesa.



Ing. Elena Mainardi

FUNCTION\_BLOCK Isteresi

In ST

VAR\_INPUT

Temperatura:REAL;

Set\_Point:REAL;

Eps:REAL;

END\_VAR

VAR\_OUTPUT

Comando:BOOL;

END\_VAR

(\* Algoritmo interno del Function Block \*)

IF Comando THEN

IF Temperatura &gt; (Set\_Point + Eps) THEN Comando := FALSE;

END\_IF;

ELSIF

Temperatura &lt; (Set\_Point - Eps) THEN Comando := TRUE;

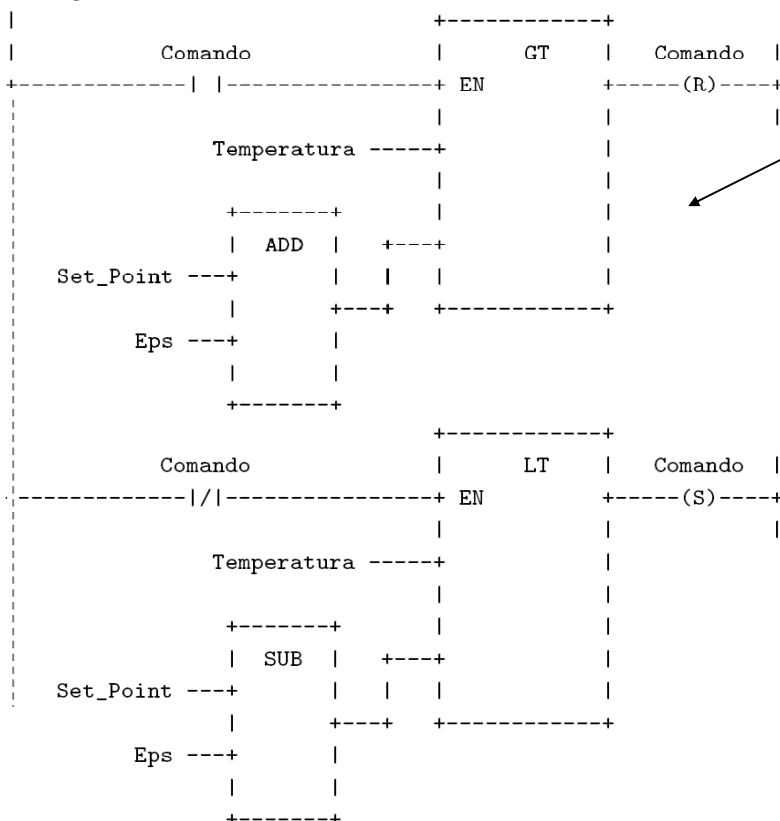
END\_IF;

END\_FUNCTION\_BLOCK

Ing. Elena Mainardi

PLC

(\* Algoritmo interno del Function Block \*)



Volendo risolvere  
lo stesso  
problema in  
ladder

N.B.  
GT= greater than  
LT= lower than  
Sono blocchi comparativi

Ing. Elena Mainardi

PLC



## Esempio ST: Esecuzione di una media aritmetica in ST

```

VAR
    Temperature:ARRAY[1..32] OF REAL;
    Max, Min, Media: REAL;
END_VAR

VAR_TEMP
    Somma:REAL;
    Count:INT;
END_VAR

Max := 0;
Min := 0;
Somma := 0;

FOR Count := 1 TO 32 DO

    Somma := Temperature[Count] + Somma;

    IF Temperature[Count] > Max THEN Max := Temperature[Count];
    END_IF;

    IF Temperature[Count] < Min THEN Min := Temperature[Count];
    END_IF;

END_FOR

Media := Somma / 32;

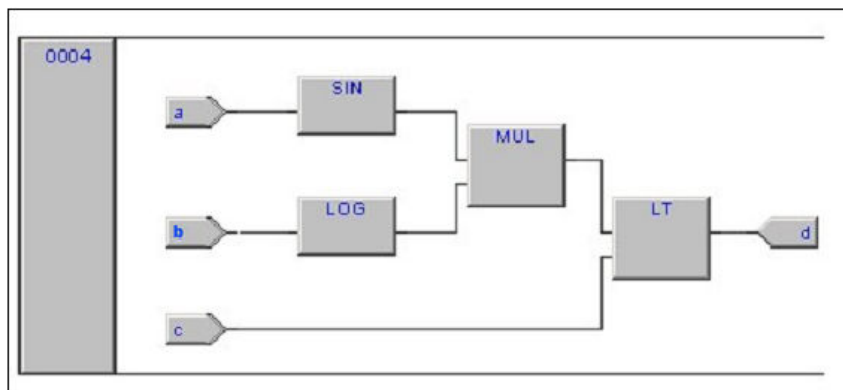
```

Ing. Elena Mainardi

PLC

## Linguaggio Function Block Diagram (FBD)

E' un linguaggio grafico in cui i controllori sono modellati come flussi di dati e di segnali attraverso elementi di processo (*function blocks*). FBD trasforma la programmazione testuale (ST) nella connessione di blocchi predefiniti, migliorando così la modularità e il riutilizzo di codice.

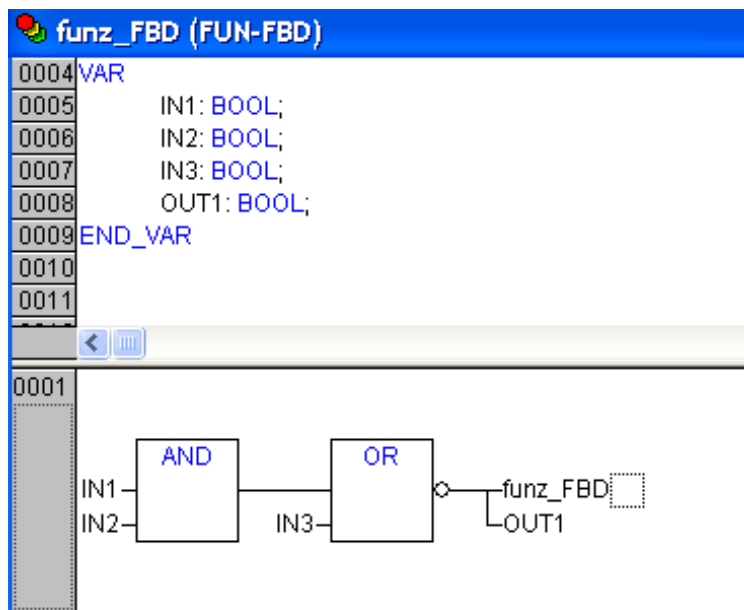


Esempio: le variabili di ingresso a e b sono usate per calcolare il valore  $\text{SIN}(a) * \text{Log}(b)$ . Il risultato di questo prodotto è confrontato con il valore della variabile di ingresso c. Se il risultato della moltiplicazione è minore di c, la variabile di output d è fissata a ON, altrimenti a OFF.

Ing. Elena Mainardi

PLC

## Linguaggio Function Block Diagram (FBD)



Esempio: funzione di nome funz\_FBD, scritta in function block diagram, che ha come valore di ritorno IN1 AND IN2 OR IN3

Con riferimento alla programmazione state driven o condition driven, si può dire che:

- Il ladder è sicuramente un linguaggio condition driven
- L'ST può essere entrambi, dipende da come imposto il mio programma

Vedremo invece che l'SFC è decisamente state driven

→ Occhio, possibile domanda d'esame: quali sono i linguaggi messi a disposizione dalla norma? Saper elencare le caratteristiche principali di ognuno di essi ←

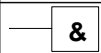
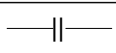
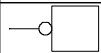
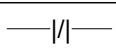
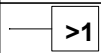
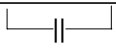
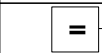
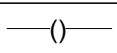
Linguaggi di programmazione uguali o simili possono essere chiamati con nomi differenti a seconda della marca del PLC.

Il caso Siemens:

FUP  $\leftrightarrow$  FBD

KOP  $\leftrightarrow$  ladder

AWL  $\leftrightarrow$  IL

FUP	KOP	AWL
		U
		N
		O
		=

UND

NICHT

ODER

ZUWEISUNG



AND

NEGAZIONE

OR

ASSEGNAZIONE