SISTEMI PER L'INDUSTRIA E PLC

SEZIONE 2

Dalle logiche a Relais ai PLC

CONTROLLO DISCRETO E CONTROLLO CONTINUO

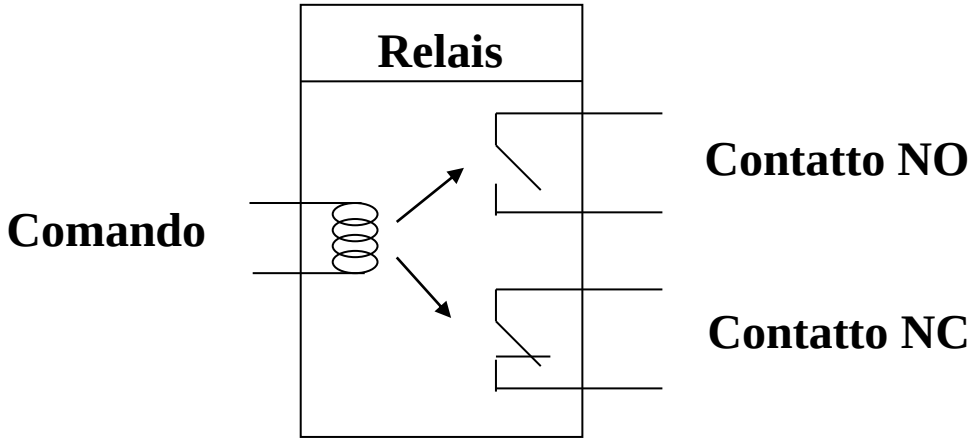
- **Controllo continuo o retroazionato (lento, preciso, adattabile)**
  - **Imposto un valore di riferimento REF(k)**
  - **Misuro il valore attuale VAL(k) e calcolo l'errore ERR(k) = REF(k)-VAL(k)**
  - **Modifico l'uscita OUT(k) in modo da minimizzare ERR(j) j>k**
  - **REF, VAL, ERR, OUT sono valori "analogici" (integer) campionati (es. PID)**
  - **REF, VAL, ERR sono valori "analogici", OUT è discreto (boolean) (es. ON-OFF)**

- **Controllo discreto o in anello aperto (veloce, economico, time-based)**
  - **Imposto un valore di riferimento REF (stabile per un ciclo di lavoro)**
  - **Attuo in uscita uno stimolo (OUT boolean) noto a priori per un certo tempo T**
  - **Verifico gli effetti nei processi di controllo delle tolleranze ed in caso modifico la conoscenza a priori sul ciclo (ad esempio adatto il tempo T)**

- **L'automazione di fabbrica tradizionalmente usa l'uomo come controllore retroazionato (vede, sente, tocca, odora e agisce di conseguenza) e le macchine in controllo discreto**
  - **Dal Fordismo si utilizzano le logiche a relais, dagli anni 90 soppiantate dai PLC**

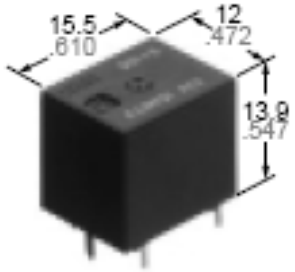
RELAIS (RELE')

- **Organi di comandi discreti**
- **I Relais sono:**
  - **Amplificatori di potenza (comando a 100mA, contatto a 10A)**
  - **Moltiplicatori di contatti**
  - **Utilizzati da personale non altamente qualificato**
  - **Isolatori naturali**
  - **Lenti (tempi: 10μs-10ms)**
  - **Negatori naturali**
  - **Utilizzati per semplici funzioni a logica booleana (logiche "a interruttori")**
  - **Ingombranti, dissipativi**

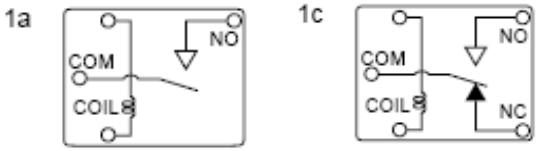


RELAIS

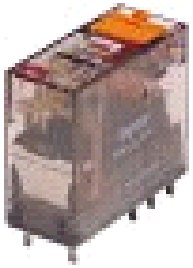
- Relè elettromeccanico Matsushita (~2€)
  - Low-cost, automotive
  - 5A, 14V
  - Ritardi ~ 10ms



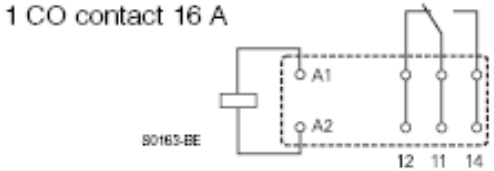
mm inch

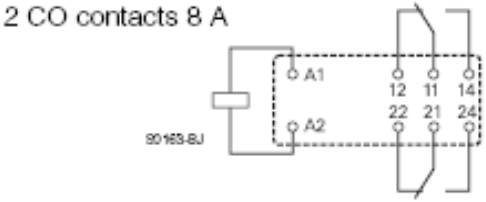


- Relè Elettromec. Tyco (~4€)
  - Low-cost, automation
  - 8-16A, 6..110V_{DC} 24..230V_{AC}
  - Ritardi ~ 10ms

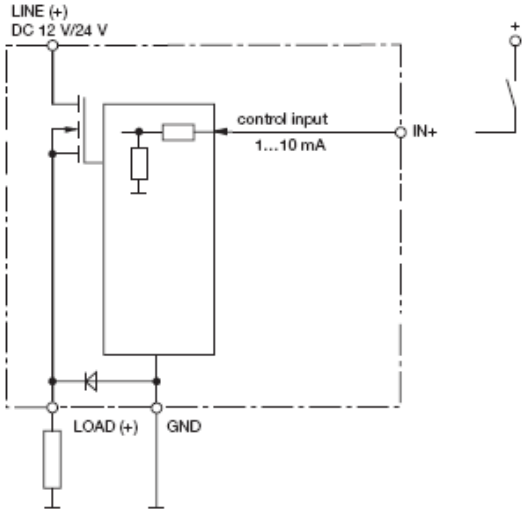


Nota: se prendo 11 e 12 ho un contatto NC, se prendo 11 e 14 ho un contatto NO (rif. 1 CO contact 16A)

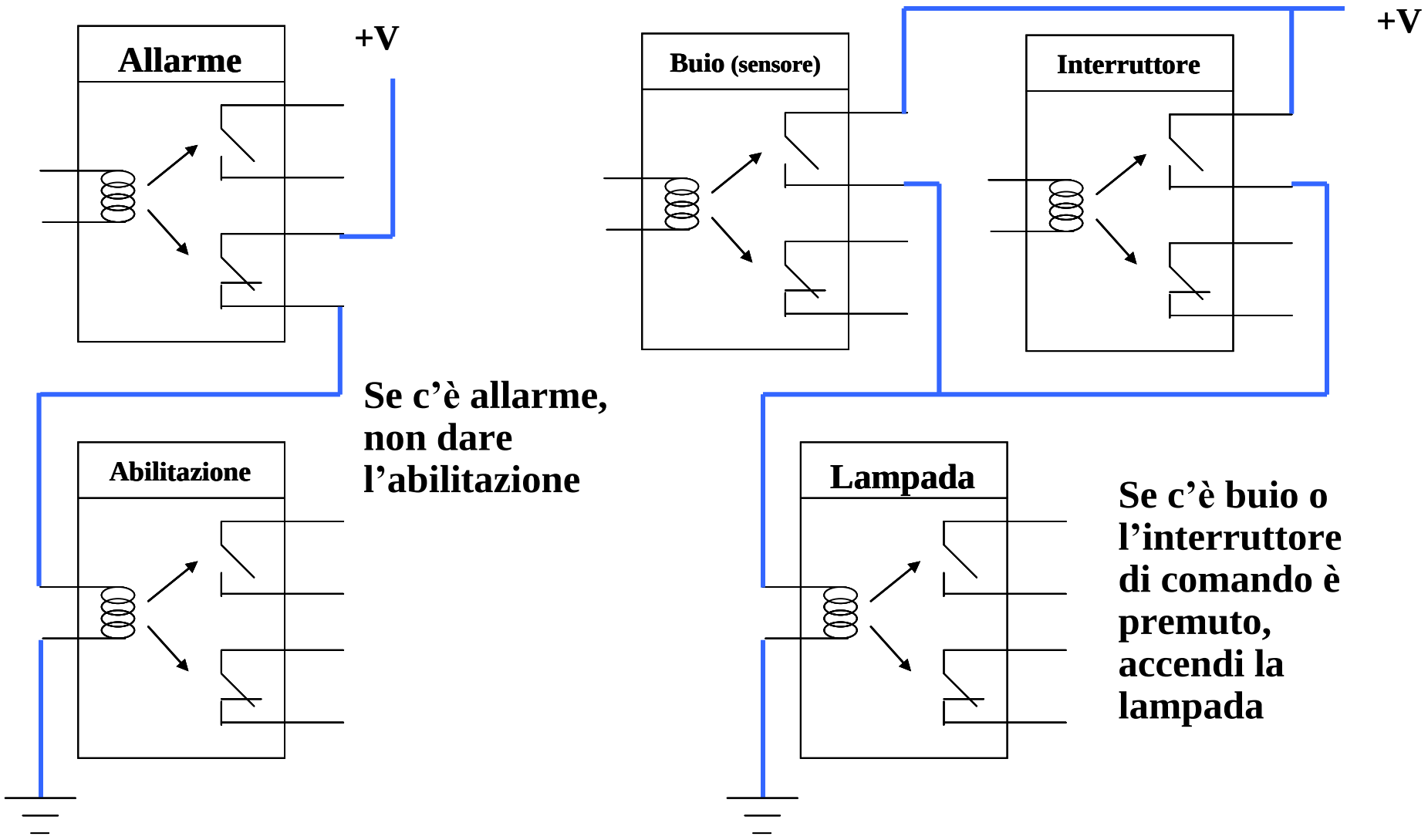




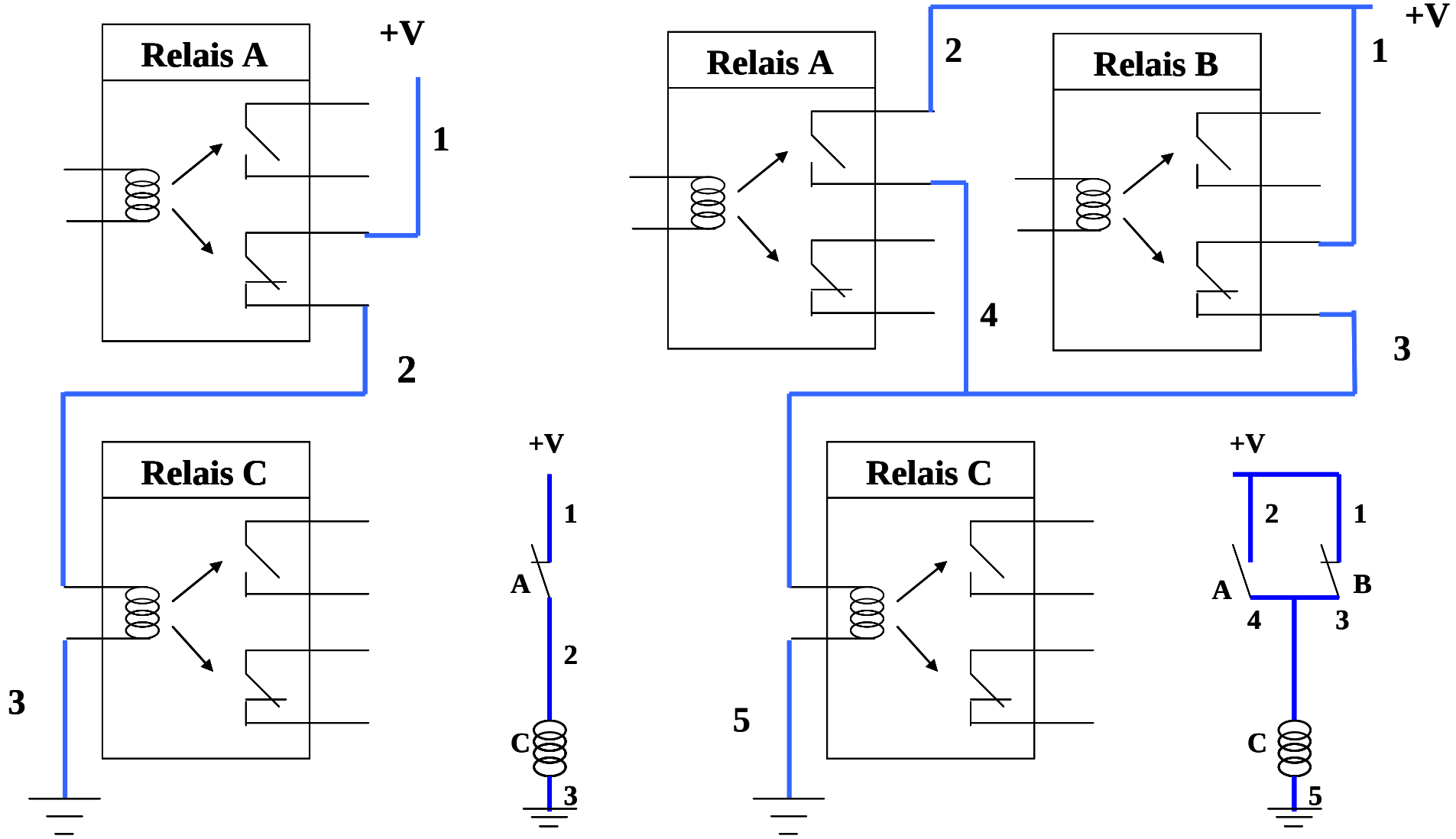
- Relè potenza ETA (~50€)
  - Hig -cost, autom.
  - 15A, max
  - sovracorrente
  - Ritardi<5ms



LOGICHE A RELE', programmare = connettere



LOGICHE A RELE': cablaggi e forme di rappresentazione

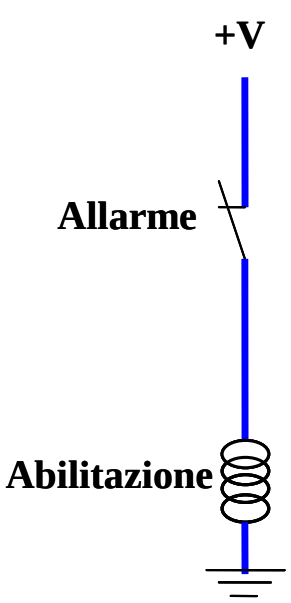


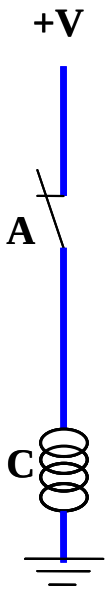
SCHEMA A RELE E EQUAZIONE LOGICA BOOLEANA

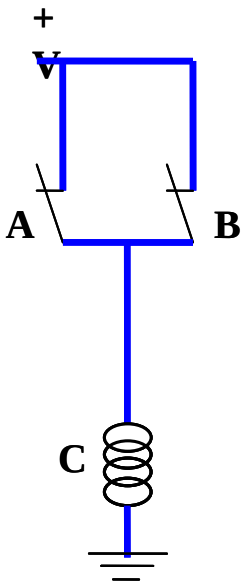
Gli schemi a relais sono un "linguaggio" dal semplice costrutto:

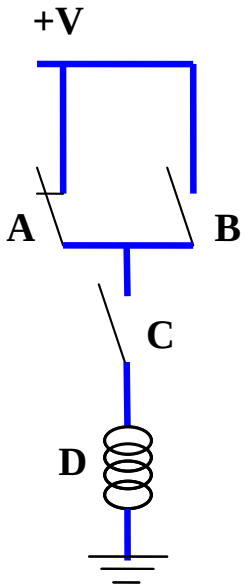
If "INPUT" then Out=1 else Out=0

(Out è una bobina, INPUT è una funzione logica a contatti (AND –serie-, OR-parallelo-, NOT-contatto NC-)









Se non c'è Allarme c'è Abilitazione
Abilitazione=NOT(Allarme)

C = ?

C = ?

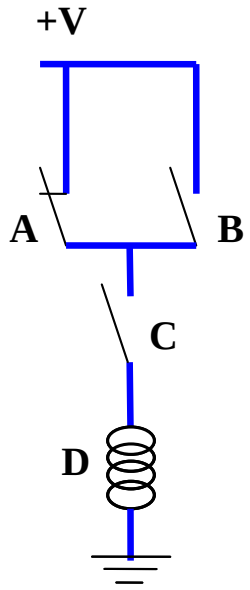
D = ?

LOGICHE A RELE': dallo schema a rele' all'equazione logica

- **Dato uno schema a rele' posso trovare l'equazione logica secondo due metodi**
  - **Approccio "bottom-up": parto dalla bobina e costruisco (basso verso alto, sinistra verso destra) l'equazione risolvendo i costrutti serie e parallelo**

    D = ( )&( )
    D = (C)&(( )+( ))
    D = (C)&((!A)+(B)) = C&(!A+B)

  - **Dato uno schema a rele' trovo l'equazione logica costruendo una tabella ingressi uscita e verificando se passa (1) o non passa (0) corrente nella bobina di uscita per ogni combinazione degli interruttori di ingresso ("0"=non attivo; "1"=attivo)**



D = ?

A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Nota: "0" per A significa non attivo, ossia interruttore chiuso (A è un NC) mentre "0" per B significa interruttore aperto (B è un NO)

D = !A&!B&C + !A&B&C + A&B&C se minimizzata diventa D = C&(!A+B)

LOGICHE A RELE': dall'equazione logica allo schema a rele'

- **Dato un problema, lo analizzo mediante tabella ingressi uscite cosi' da analizzare tutte le possibili configurazioni di ingresso e avere sempre un'uscita ben posta**

Esempio: Se c'è l'abilitazione (A) allora fai andare il motore (D), se ci sono anomalie (B) o allarmi (C) allora il motore (D) è fermo

A	B	C	D	
0	0	0	0	Nota: non è chiaro ma in logica di sicurezza è meglio avere il motore fermo
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	
1	0	1	0	Nota: non è chiaro ma in logica di sicurezza è meglio avere il motore fermo
1	1	0	1	Nota: non è chiaro ma "anomalie" è sempre meno forte di "allarme". Da chiedere chiarimenti!!!
1	1	1	0	Nota: non è chiaro ma in logica di sicurezza è meglio avere il motore fermo

D = A&!B&!C + A&B&!C (minimizzando D = A&!C)

- **Data un'equazione logica in forma SOP realizzo lo schema a rele' assegnando una bobina all'uscita e costruendo in parallelo tanti rami quanti sono i minterm dove ogni ramo e' costituito dalla serie dei contatti, in forma vera o negata, che formano il minterm ("OR"=parallelo; "AND"=serie)**

- **<u>Minimizzare l'equazione logica significa usare meno rele'</u>:**
  - **Riduzione dei costi componente, dei costi cablaggio e degli ingombri**

DAL PROBLEMA ALLO SCHEMA A RELE'

- **Problema: se c'è abilitazione (A=1) e non c'è luce (B=0) oppure se c'è buio (B=0) e c'è richiesta (C=1), allora la lampada (D) deve essere accesa**

- **Problema: in una stanza ci sono 3 sensori di luce: A, B, C ("1"=c'è luce). Accendi la lampada (D) se la maggioranza dei sensori dice che c'è buio**

- **Problema: in una stanza ci sono 2 sensori di luce: A e B ("1"=c'è luce). Accendi la lampada (D) se per entrambi sensori c'è buio, accendi il segnalatore (C) se i sensori sono in disaccordo (funzione XOR)**

- **Problema: se si è in modalità "JOG" (A=1) e non c'è allarme (B=0) e qualcuno tiene premuto il pulsante di JOG (C=1) allora fai andare il motore (D)**

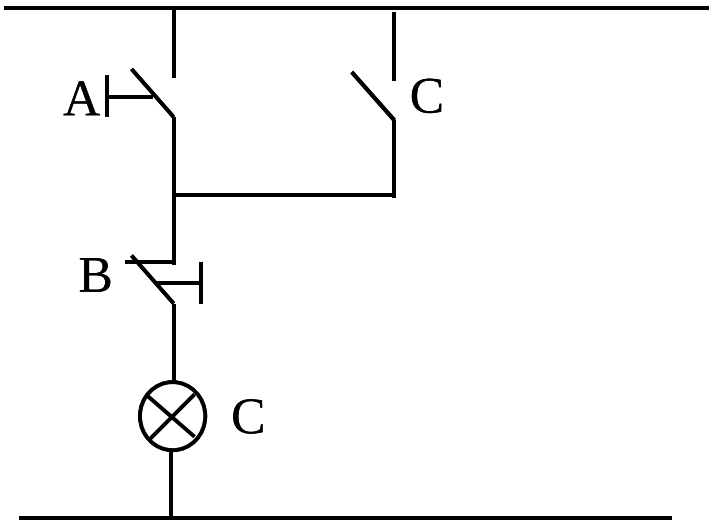
- **Problema: se si preme il pulsante Start (A=1) e non si preme il pulsante Stop (B=0) allora fai andare il motore (C) fino a quando non si preme il pulsante Stop (memoria)**

LA FUNZIONE DI MEMORIA REALIZZATA A RELE'

- Il problema dello Start (A) Stop (B) di un motore C

  C_{new}=!A&!B&C_{old}+A&!B&!C_{old}+A&!B&C_{old}=
  = …. = (A + C)&!B

- **Autoritenuta Reset-prevalente**



A	B	C"prima"	C"poi"	
0	0	0	0	Se A e B sono falsi allora C non cambia
0	0	1	1	
0	1	0	0	Se B è vero allora C è falso
0	1	1	0	
1	0	0	1	Se A è vero e B è falso allora C è vero
1	0	1	1	
1	1	0	0	Se B è vero allora C è falso
1	1	1	0	

Con i Relè posso fare qualunque funzione logica combinatoria e con memoria

I RELE' GESTISCONO SENSORI E ATTUATORI BINARI

- **Gli ingressi di una logica a relè sono "contatti-equivalenti"**
  - **Un ingresso (es. sensore) deve avere come uscita un contatto pulito NA o NC**

- **Sensore = traduce una grandezza fisica in una grandezza elettrica**
  - **Un sensore di temperatura con uscita 0-5V nel range 0-100°T può essere utilizzato in una logica a relè?**
  - **Un pulsante con uscita 0V quando non è premuto e 5V quando è premuto può essere utilizzato in una logica a relè?**

- **Le uscite di una logica a relè sono "bobine-equivalente"**
  - **Un'uscita (es. un attuatore) deve avere come ingresso una bobina**

- **Attuatore = oggetto che traduce una grandezza elettrica in una fisica**
  - **Un forno che richiede in ingresso un segnale di riferimento di temperatura tra 1V (T=100°C) e 2V (T=200°C) può essere gestito da una logica a relè?**
  - **Un ventilatore che si pilota con un segnale tra terra (OFF) e 24V(ON) può essere gestito da una logica a relè?**

LOGICHE DI COMANDO E SEGNALAZIONE (RELE')

- **Gestione logiche di allarme:**
  - **Segnalazione (lampade, sirene,…) sulla base dell'intervento di alcuni allarmi**
  - **Allarmi a soglia**
    - **Valore sotto soglia di attenzione: lampada spenta**
    - **Valore sopra soglia di attenzione ma sotto soglia di allarme: lampada che lampeggia lenta**
    - **Valore sopra soglia di allarme: lampada che lampeggia veloce**
    - **Valore sopra soglia di allarme continuativamente da oltre un tempo T: lampada accesa fissa (condizione memorizzata)**

- **<u>Tipologie di allarme con memorizzazione</u>:**
  - **Allarme con memorizzazione (il più critico): serve un reset generale**
  - **Allarme con memorizzazione, riconoscimento (o ripristino) e rientro: c'è un comando di ripristino gestito dall'operatore che sblocca l'allarme se questo non è più attivo (se allarme e ripristino entrambi on vince l'allarme)**
  - **Allarme con memorizzazione e riconoscimento (o ripristino): c'è un comando di ripristino gestito dall'operatore che sblocca l'allarme (se allarme e ripristino entrambi on vince il ripristino)**

LOGICHE DI COMANDO E SEGNALAZIONE (RELE')

- **Altre funzioni correlate agli allarmi:**
  - **Riconoscimento del primo allarme**
    - **Normalmente ci sono vari allarmi, anche più di 100, che si succedono a seguito di un problema e le azioni da intraprendere sono diverse a seconda del diverso ordine temporale delle segnalazioni**
    - **Serve un timestamp (riferiemento temporale univoco) che permetta di ordinare la successione degli eventi**
    - **Valore sotto soglia di attenzione: lampada spenta**
  - **Rilevazione degli allarmi più prioritari**
    - **Gli allarmi a maggiore priorità richiedono azioni più tempestive**
  - **Gestione anomalie**
    - **Le anomalie sono allarmi non memorizzati che possono comportare azioni immediate correttive o azioni differite nel tempo (es. manutenzione)**

LOGICHE DI COMANDO E SEGNALAZIONE (RELE')

- **Gestione logiche di comando:**
  - **Abilitazione comandi da locale in manuale**
  - **Abilitazione comandi da remoto in manuale (es. pulpito)**
  - **Abilitazione comandi in automatico (es. da computer)**
- **Gestione logiche di movimentazione**
  - **Arresto di motori a seguito di allarmi o azioni di pulsanti o sensori di presenza (finecorsa, cellule fotoelettriche)**
  - **Avvio di motori in presenza di abilitazione e in assenza di cause di arresto**
  - **….**
- **Gestione motori. Un motore può essere azionato:**
  - **Localmente, a bassa velocità, per prova (cablaggi, blocchi,…)**
  - **Localmente, a velocità nominale, con comandi di marcia e arresto**
  - **Da Pulpito, a velocità nominale, con comandi di marcia e arresto**
  - **In automatico da remoto, a velocità nominale, con comandi di marcia e arresto**
  - **…**
  - **C'è un problema di coesistenza di modi (priorità, interblocchi,…)**

LOGICA CABLATA (RELE') E PROGRAMMABILE (PLC)

PLC (PROGRAMMABLE LOGIC CONTROLLER) E RELAIS

- **I PLC sono nati con la funzione di sostituire le operazioni logiche che venivano effettuate collegando in vario modo i relais. Il primo linguaggio di programmazione dei PLC (ladder diagram) ricalca gli schemi elettrici a relais (linguaggio a contatti)**

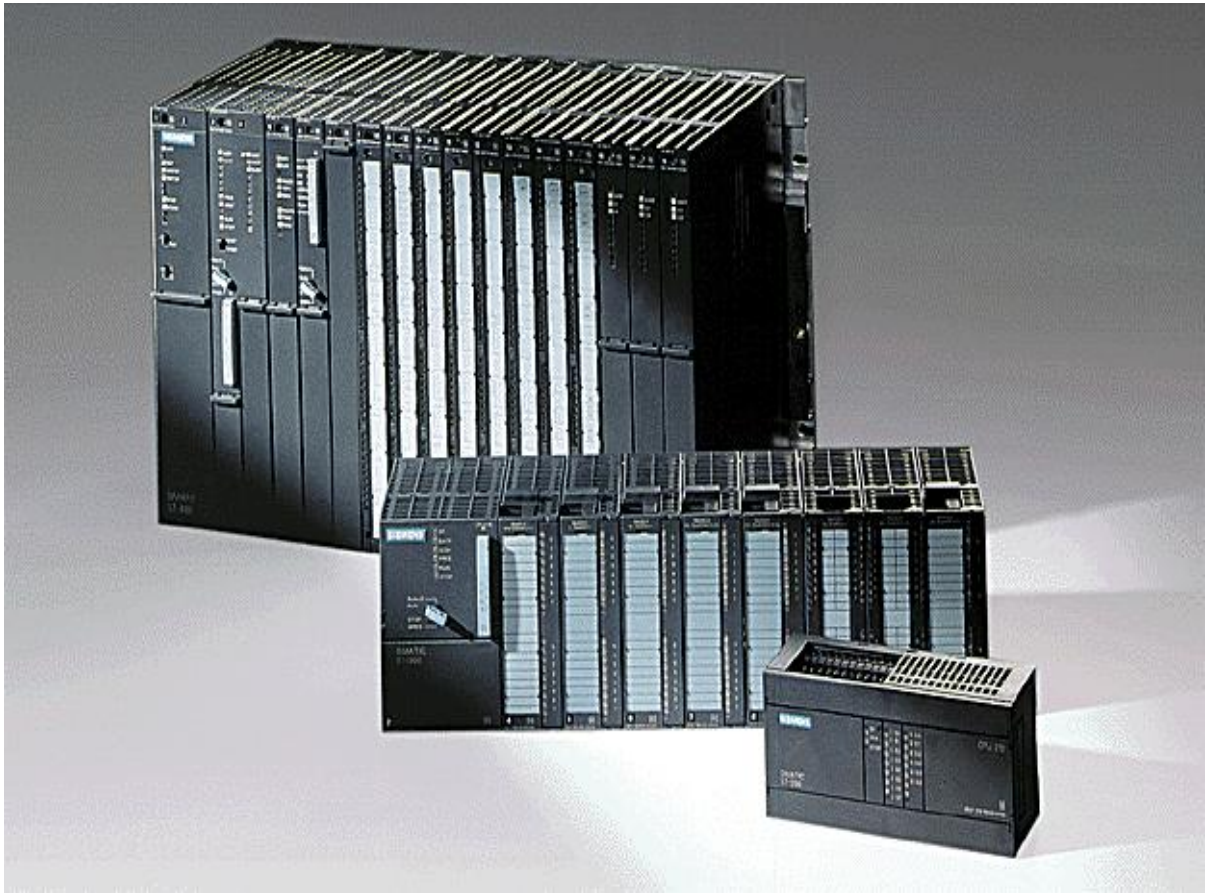
- **Problemi dei primi PLC (anni 70-80)**
  - **Costo, Scarsa affidabilità**
  - **Diffidenza degli addetti ai lavori (manutentori,…) verso un sistema "black box"**

- **Rispetto alle logiche a relais:**
  - **Riduzione dei cablaggi, degli ingombri e della potenza dissipata**
  - **Elevata versatilità verso "upgrade" (SW), aumento della velocità di elaborazione**
  - **Consente nuovi campi d'impiego (regolazione e controllo, calcolo multivariabile,…)**

- **Rispetto ai PC**
  - **Elabora segnali oltre che informazioni, orientato alle operazioni logiche, SW di base affidabile e in tempo reale, non soffre di obsolescenza digitale**

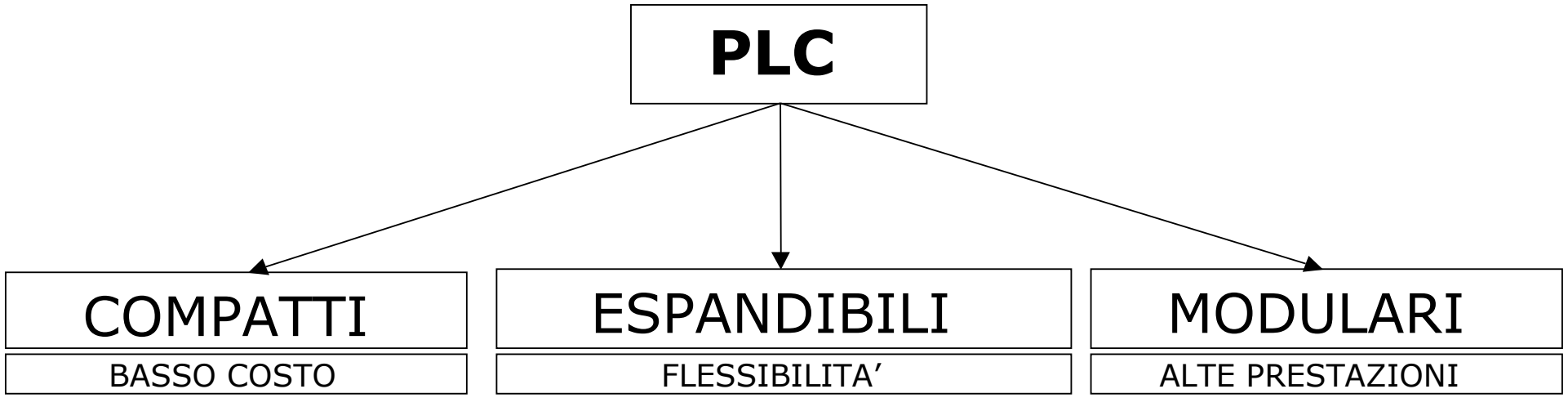
PLC: CARATTERISTICHE GENERALI

- **Gestiscono sensori e attuatori (abilitazioni, allarmi, movimentazioni, sequenze,..)**
- **HW modulare**
  - **Espandibile**
  - **Diversi moduli**
    - **CPU**
    - **ingressi logici**
    - **uscite logiche**
    - **ingressi analogici**
    - **ingressi dedicati**
    - **....**
    - **moduli funzionali**
- **SW "semplice"**
  - **schemi a contatti**
  - **struttura ciclica**
  - **autodiagnostica**



PLC: DISPONIBILITA' DEL MERCATO

- **Opera**
  - **A livello di cella (i più complessi, tra i quali i controllori distribuiti e i Soft-PLC)**
  - **A livello di comando (i più semplici)**



	COMPATTI	ESPANDIBILI	MODULARI
Struttura HW	Monoblocco	Modulare	Modulare
Linguaggi SW	Logici (ladder)	Logici (ladder)	Evoluti (IEC1131-3)

- **https://www.siemens.com/global/en/home/products/automation/systems/industrial/plc.html**

PLC “MODULARI”

- **Strutture HW modulari per applicazioni multi-task, multi-CPU**

- **Il PLC modulare è un’architettura multi-CPU per applicazioni molto critiche (potenza centralizzata)**
  - **Molto costoso**
  - **Le architetture centralizzate sono spesso sostituite dalla architetture distribuite (tanti PLC più semplici che cooperano), tranne che per applicazioni dove:**
    - **Il ritardo introdotto dalla comunicazione è incompatibile con la latenza tra l’evento (ingresso) e la reazione (uscita)**
    - **Il PLC espandibile non ha sufficiente risorse (potenza/velocità di calcolo, memoria,..)**

- **Elevata velocità (ridotti tempi di ciclo <1ms)**

- **Limitati a pochi casi applicativi e sostituiti da architetture distribuite costituite da tanti PLC “snelli” e un “Soft-PLC” (vedi dopo)**

PLC ESPANDIBILI

- **Strutture HW modulari per applicazioni a singola CPU**

- **Architetture decentrate:**
  - **Possibili per applicazioni non critiche in termini di tempo (la comunicazione rallenta)**
  - **Salvaguardia degli investimenti:**
    - **Modifico CPU mantenendo moduli di I/O**
    - **Modifico/aggiungo moduli di I/O mantenendo CPU**
  - **Più "gestibili" grazie anche alla standardizzazione dei ricambi**
  - **Più attuali grazie alla potenza crescente delle CPU**

- **Linguaggi semplici**

- **Strutture di dati "globali"**

PLC COMPATTI

- **Strutture HW monoblocco con un limitato numero di opzioni**

- **Architetture fortemente decentrate:**
  - **Usati anche come periferia (concentratore/gestore di I/O localizzati)**
  - **Pochi dati da scambiare, semplice connettività**
  - **Molto semplici da:**
    - **testare (in fase di avviamento o ricerca guasti)**
    - **manutenere (impatto/costo delle modifiche)**
    - **aggiornare (minore cifra da ammortare)**
  - **Più attuali grazie alla potenza crescente delle CPU**

- **Linguaggi molto semplici**

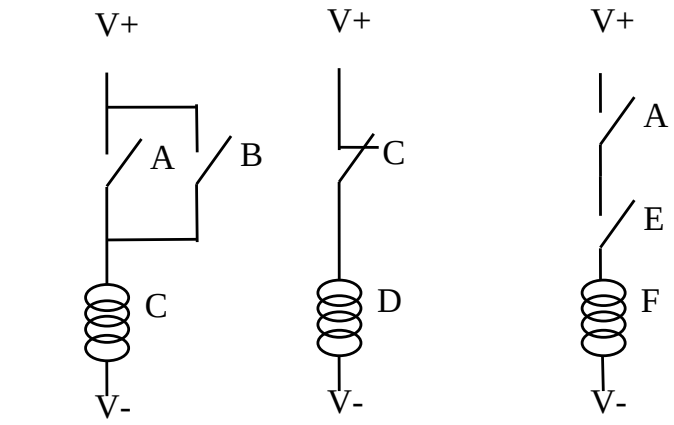
- **Costi molto contenuti (elevata concorrenza)**

- **Ampio campo di applicabilità (industriale, civile,...)**

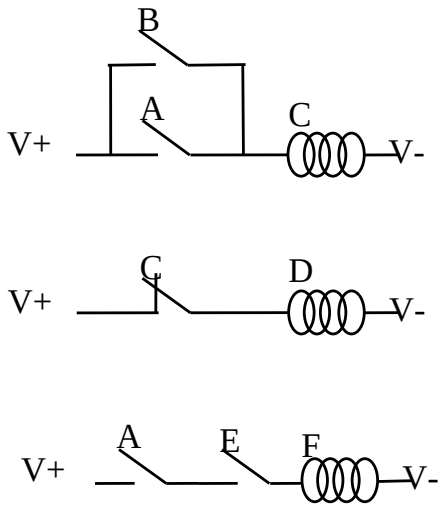
Programmazione: Configurazione, Tabella Simboli e Blocco Codice

- **Il PLC spesso è un sistema modulare, quindi devo dirgli di quali moduli è composto**
  - **Configurazione**
  - **Attenzione! Quando collego il sensore Sx all'ingresso j del modulo k devo poterlo individuare e richiamare nel programma, quindi:**
    - **Ingressi e uscite sono mappati in memoria (array di byte con indirizzamento assoluto), quindi ad ogni modulo, a seconda della sua posizione rispetto alla CPU, è associato un range di indirizzi, all'interno del quale recuperare il bit j**
- **Il PLC ha ingressi e uscite che vengono collegate a sensori e attuatori secondo quanto descritto negli schemi funzionali**
  - **Lista delle attribuzioni o Tabella di I/O**
  - **Il PLC è una scatola nera e vedo tutti i segnali che entrano e che escono**
  - **Naturalmente ci sono anche celle di memoria che possono essere dichiarate nella Tabella di I/O o in blocchi dati**
- **Le funzionalità e le relazioni tra ingressi e uscite sono descritte nel programma**
  - **Blocco di codice**

DAGLI SCHEMI A RELAIS AD UN “LINGUAGGIO”



C = A OR B D = NOT(C) F = A AND E



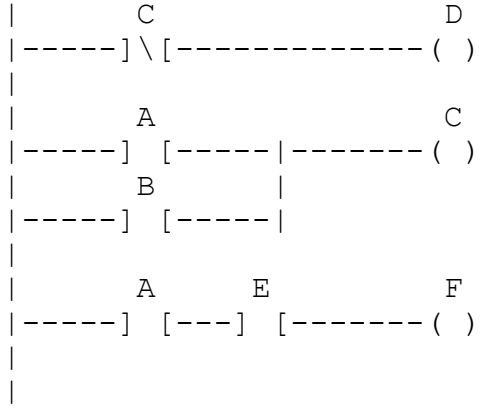

```
|       A                 C
|-----] [-----|-------( )
|       B       |
|-----] [-----|
|
|       C                 D
|-----]\[-------------( )
|
|       A     E           F
|-----] [---] [-------( )
|
|
                Programma 1
```

- **Routando a sinistra di 90° uno schema a relais si ottiene qualcosa che assomiglia ad una sequenza di istruzioni (programma)**
- **Nello schema a relais tutti i segmenti sono eseguiti contemporaneamente**
- **Nei programmi le istruzioni vengono eseguite in sequenza**
  - **I programmi 1 e 2 sono uguali?**

```
|       C                 D
|-----]\[-------------( )
|
|       A                 C
|-----] [-----|-------( )
|       B       |
|-----] [-----|
|
|       A     E           F
|-----] [---] [-------( )
|
|
                Programma 2
```

IL “LINGUAGGIO” LOGICO BOOLEANO

- **Questo nuovo “linguaggio”, derivato dagli schemi a relais per essere comprensibile agli operatori (anni 80), ne includeva le operazioni (*if input then Out=1 else Out=0*)**
  - **NOT (si prende il contatto normalmente chiuso, come nei relais)**
  - **AND (si prende la serie dei contatti, come nei relais)**
  - **OR (si prende il parallelo dei contati, come nei relais)**
  - **SET o RESET (si usa una memoria che viene vista come una “bobina speciale”, è più semplice gestire la memoria rispetto alle autoritenute) *if input then Out=1***
  - **Si possono creare altre funzioni (es. rilevatore di fronte) che con i relais erano molto più difficili da realizzare**
  - **Attenzione! Nel PLC non ci sono contatti e bobine, ma solo memorie**



NOT

OR

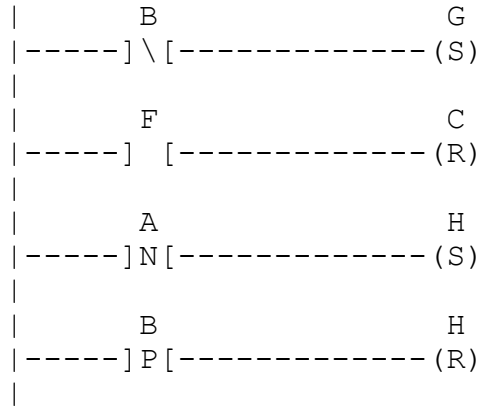
AND

SET

RESET

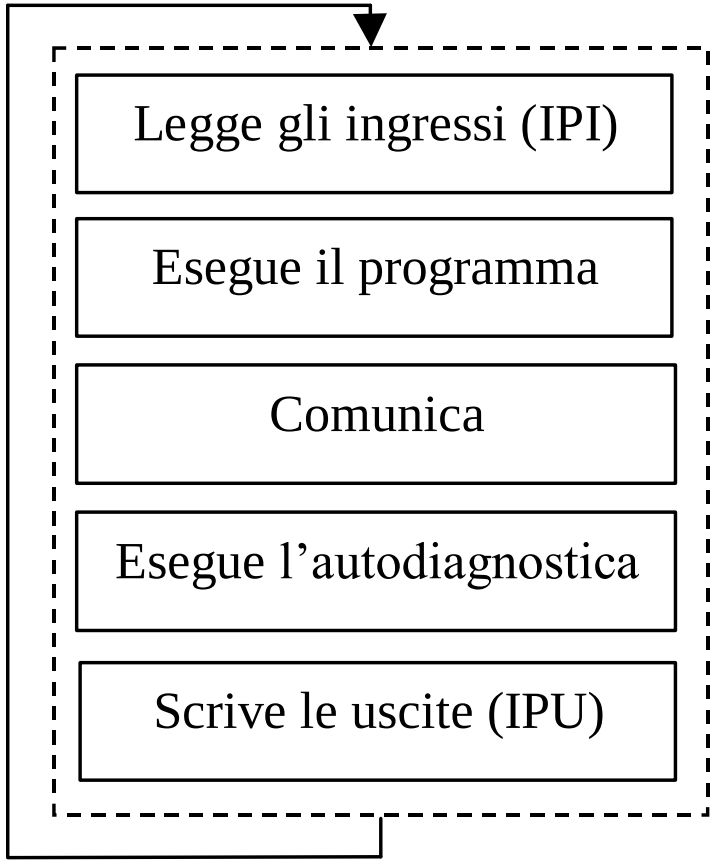
Rilevatore di fronte negativo

Rilevatore di fronte positivo



CICLO DI FUNZIONAMENTO (CICLO DI SCANSIONE)

- **Il PLC viene usato per effettuare ciclicamente una serie di istruzioni del tipo "se... allora..."**
- **Ciclo di scansione**
  - **T_ciclo_min ~ ms**
- **Esecuzione sequenziale del programma**
- **Architettura ciclica**
  - **no tempi di attesa**
  - **macchine a stati?**
- **Fase "comunica"**
  - **diagnostica via PC**
  - **Fase di background a T controllato**
- **Fase "autodiagnosi"**
  - **diagnostica locale (CPU)**
  - **diagnostica dei moduli di I/O**



ESECUZIONE PROGRAMMA UTENTE

- **La CPU del PLC è basata su un microcontrollore:**
  - **il microcontrollore deve eseguire il programma (non modificabile) del ciclo di scansione**
  - **task diagnostici ad interrupt (Es. watch dog)**
  - **tali programmi sono scritti, compilati, memorizzati dal costruttore**

- **Programma utente compilato:**
  - **soluzione più veloce**
  - **si perde la corrispondenza tra istruzione utente e istruzione eseguita**
  - **ogni modifica del programma utente implica una ricompilazione totale**
    ⇨ poco affidabile

- **Programma utente interpretato:**
  - **il programma utente genera un codice intermedio che viene interpretato**
  - **il programma non modificabile scritto dal costruttore include l'interprete**
  - **il programma utente è una serie di "pseudoistruzioni" allocata in un'area di memoria riservata**

IMMAGINI DI PROCESSO

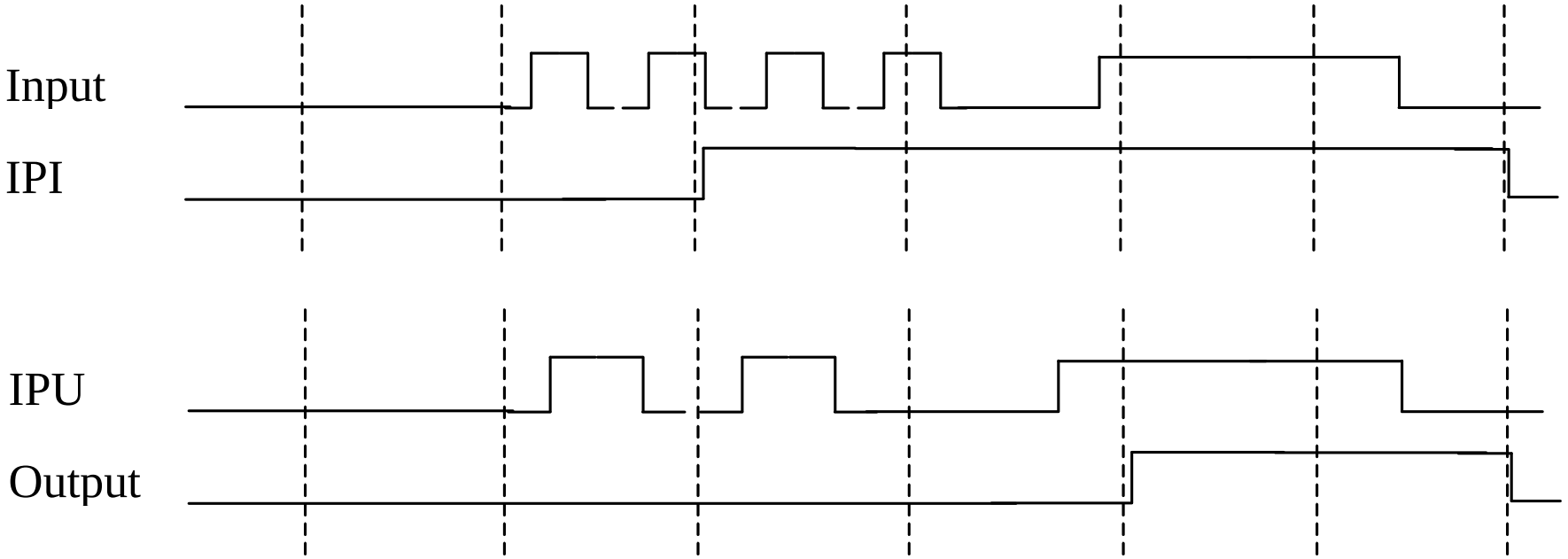
- **Immagini di processo degli ingressi (IPI):**
  - **variabili nelle quali viene memorizzato il valore degli ingressi logici all'inizio del ciclo**
  - **il programma applicativo (ciclo k) si svolge a ingressi congelati**
  - **è possibile accedere direttamente agli ingressi fisici senza modificare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

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

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

IMMAGINI DI PROCESSO

- **Immagini di processo:**
  - **Alcune variazioni degli ingressi possono andare perse (dipende da T_ciclo)**
  - **È consigliabile accedere alle uscite in un unico punto del programma**

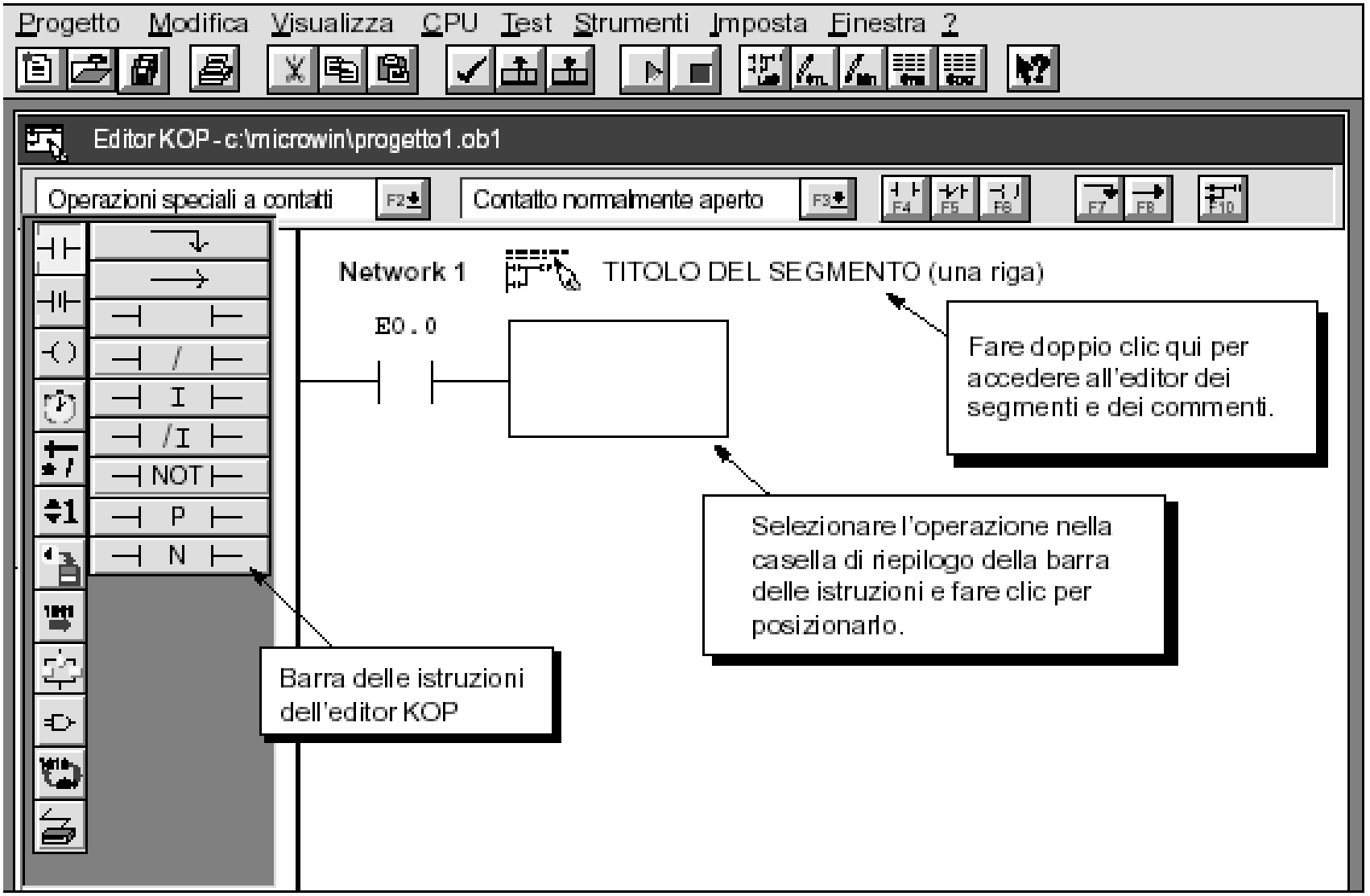


SISTEMI DI SVILUPPO PER PLC

- **Terminali di programmazione**
  - **Economici, dedicati**
  - **Soluzione valida solo per PLC compatti di fascia bassa**

- **Ambiente di sviluppo su PC**
  - **Stesura del codice**
  - **Compilazione**
  - **Download:**
    - **Blocco codice (OB1)**
    - **Blocco dati (DB1)**
    - **Configurazione CPU (CFG)**
      (password, parametri modo PROG, aree di backup, filtri d'ingresso, $T_{background}$)
    - **…**
  - **Inizializzazione variabili**
  - **Esecuzione controllata:**
    - **Numero di cicli di scansione**
    - **Possibilità di tenere monitorati o forzare variabili**

SISTEMI DI SVILUPPO PER PLC: esempio (STEP7)



ORGANIZZAZIONE DEL PROGETTO (STEP7)

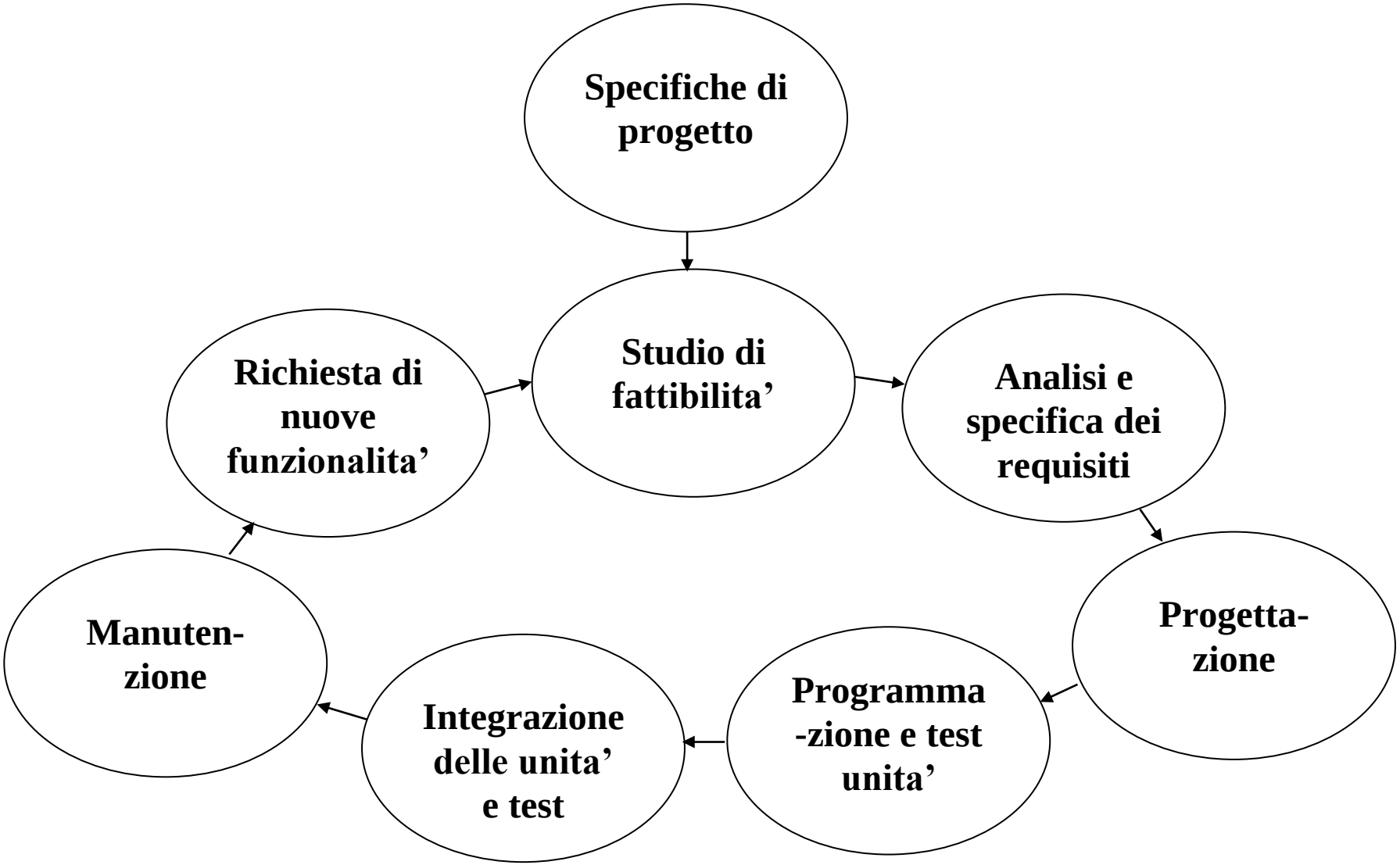
- **Organizzazione del progetto in blocchi (tipo directory)**
- **Blocchi organizzativi (OB)**
  - **OB1 = blocco principale (il programma nel ciclo di scansione, interrogato ad ogni ciclo)**
  - **Altri Blocchi con funzionalità particolari (Es. blocco che va in esecuzione ogni T, blocco che va in esecuzione al Reset, ecc.)**
- **Funzioni definite dall'utente (FC)**
  - **Contengono le routine di programma (blocchi privi di memoria propria, ma con possibili parametri di ingresso e uscita)**
- **Funzioni di sistema (SFC)**
- **Blocchi funzionali (FB)**
  - **Contengono le routine di programma (blocchi con memoria DB –Data Block- di istanza)**
- **Blocchi funzionali di sistema (SFB)**
- **Blocchi di tipo "dato" (DB)**
  - **Possono essere locali (di un solo blocco) o globali (accessibili da tutti i blocchi)**
  - **Possono essere "di istanza", ossia "instanziati" ogni volta che si chiama un FB**

PLC: MODALITA' DI FUNZIONAMENTO

- **Modalità STOP (/PROG):**
  - **non esegue il programma applicativo di sua iniziativa**
  - **esegue funzioni del sistema operativo di dialogo con il sistema di programmazione (terminale o PC)**
  - **esegue funzioni di:**
    - **diagnostica**
    - **configurazione**
    - **programmazione del programma utente (download / upload)**

- **Modalità RUN:**
  - **esegue il programma applicativo sotto il controllo del sistema operativo residente (ciclo di funzionamento)**
  - **effettua autodiagnostica**
  - **se connesso al sistema di programmazione può:**
    - **visualizzare all'operatore lo stato di variabili**
    - **eseguire il programma applicativo solo per un certo numero di cicli**
    - **effettuare leggere modifiche al programma**

SOFTWARE PER PLC: CICLO DI VITA



LINGUAGGI TRADIZIONALI

- **Ladder diagram (schemi a contatti, KOP per Siemens)**
  - **Orientato ai manutentori**
  - **Richiama gli schemi funzionali a relais**

- **Instruction List (lista di istruzioni, AWL per Siemens)**
  - **Orientato al personale informatico**
  - **Pseudoassembler**

- **Function Block Diagrams (Schemi funzionali, FUP per Siemens)**
  - **Orientato al personale elettronico**
  - **Segue lo standard ANSI/IEEE Std.91**

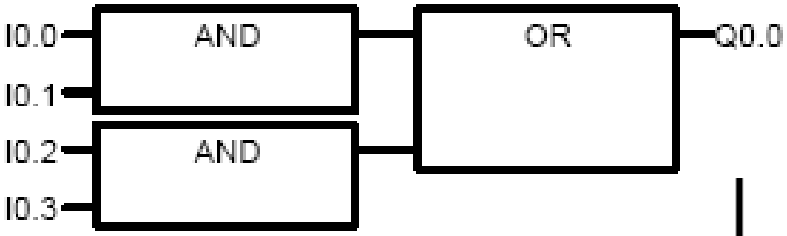
LINGUAGGI AWL, FUP, KOP

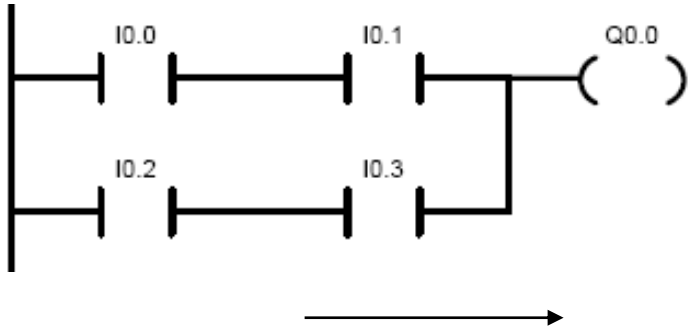
- **(Q0.0) = (I0.0)&(I0.1) + (I0.2)&(I0.3)** **nota: Q=uscita I=ingresso**

AWL	FUP	KOP
Lista di istruzioni	Schemi funzionali IEEE Std.91	Ladder Diagram

- **Dei tre linguaggi base, l'Instruction List (AWL) è praticamente sparito, in quanto sostituito da un nuovo linguaggio testuale (Structured Text)**

```
LD    I0.0
A     I0.1
LD    I0.2
A     I0.3
OLD
=     Q0.0
```





Ipotetico flusso

LINGUAGGIO LADDER

- **Dati (I dati possono essere condivisi da FB scritti con linguaggi diversi)**
  - **Il Ladder supporta tutti i tipi di dati semplici (bit, byte, word, integer, real,…)**
  - **Supporto dei dati relativi al tempo (Date&Time, Time)**
  - **Array difficilmente supportati (solo indirizzamento immediato o diretto)**

- **Istruzioni**
  - **Il codice sono sequenze di istruzioni con lo stesso costrutto (tranne la bobina)**
    if "condition" then "operation" (altrimenti no operation).
  - **Le condizioni sono test su variabili (booleane) o confronti tra variabili (numeriche)**
  - **Gli operatori comprendono varie tipologie (aritmetici, logici, trasferimento, abilitazione e configurazione oggetti quali timer, counter richiamo di blocchi di programma quali FC o FB, operatori di controllo di flusso del programma)**

- **Particolarità**
  - **Ordine delle istruzioni, data dependency, ecc. dipende dal ciclo di scansione e dal meccanismo delle immagini di processo e da meccanismi implementativi**

LINGUAGGIO LADDER, operatori e operatore “bobina”

- **Un programma scritto in ladder è una sequenza di istruzioni del tipo**
  If “condition” then “operation” (else no operation)
  - **“operation” comprende operatori logici (set, reset,..), aritmetici (add, mul,..) e di traferimento boolean (bobina) o numerico (MOVE)**
  - **Se un “operation” booleano è uguale in due segmenti le rispettive conditions sono in OR**
    - **If VAR1 then set(VAR5)**
    - **If VAR2 then set(VAR5)**

    Equivale a
    - **If VAR1 or VAR2 then set(VAR5)**
  - **Possono coesistere diverse operazions booleane sulla stessa uscita ma sarà solo l’ultima ad avere effetto (nel caso numerico tutte le operations hanno effetto)**

- **L’operatore bobina –( )- fa eccezione perché risponde al costrutto**
  If “condition” then *Bobina*=1 else *Bobina*=0
  - **La bobina non può coesistere con nessun altro operatore perché scrive sempre sull’Object cancellando qualsiasi effetto di memoria (manca “else no operation”)**

LINGUAGGIO LADDER, gli operatori di interrogazione booleana

- **Le "condition" sono normalmente reti logiche (AND = serie, OR = parallelo) di interrogazioni**

A
---| |--- Interrogazione vera, A_k=1? Se si passa corrente da sx a dx

A
---|\|--- Interrogazione negata, A_k=0? Se si passa corrente da sx a dx

A
---|P|--- Rilevatore di fronte positivo (A_k=1)&(A_{k-1}=0)?
Mem

A
---|N|--- Rilevatore di fronte negativo (A_k=0)&(A_{k-1}=1)?
Mem

- **Mem serve per memorizzare il valore vecchio di A = A_{k-1} e viene aggiornato con il valore di A_k alla fine dell'esecuzione di quel ladder, quindi è valido solo per una interrogazione**

LINGUAGGIO LADDER, gli operatori di uscita booleani

- **Le "operation" su variabili booleane sono normalmente assegnamenti, Set e Reset**

-( )- Operatore di Assegnamento (o "bobina")

Il valore precedente della variabile viene sovrascritto con il valore attuale (Se "condition" è vero la variabile booleana viene posta a 1 altrimenti viene posta a 0). Se una variabile booleana viene gestita mediante operatore di assegnamento, la variabile potrà solo essere interrogata (l'operatore di assegnamento non può coesistere con altri operatori di uscita).

Esiste anche l'operatore –(\)- che assegna il valore negato (poco usato)

-(S)- Operatore di SET

Se "condition" è vera allora la variabile booleana viene posta a 1 altrimenti mantiene il valore precedente. L'operatore di SET può coesistere con altri operatori di uscita (due o più operatori di SET sulla stessa variabile agiscono in OR logico)

-(R)- Operatore di RESET

Se "condition" è vera allora la variabile booleana viene posta a 0 altrimenti mantiene il valore precedente. L'operatore di RESET può coesistere con altri operatori di uscita

STRUTTURA DI UN PROGRAMMA PER PLC

- **Lista delle attribuzioni (Immagini di I/O, dati,..)**
  - **Nome, simbolo, commento**

- **Il programma può essere implementato in modo lineare (in un'unica sequenza) o strutturato (programmi che richiamano altri sottoprogrammi)**

- programma principale (OB1)

consta di una sequenza organizzata di istruzioni nel linguaggio selezionato. Può contenere "sottoprogrammi" in forma FC (con possibili parametri di I/O) e in forma FB (con possibili parametri di I/O e aree di memoria personalizzate)

- programmi associati ad eventi

Possono essere associate a ingressi o a eventi (eccezioni) quali la partenza "a freddo" o "a caldo" (es. OB100, associato al reset), la perdita del programma applicativo (batterie scariche), o a interrupt periodici…

USO DEI DATI

- **I dati (Immagini di I/O, variabili,..) sono contenuti nella lista delle attribuzioni**
  - **dare nomi simbolici appropriati**
  - **utilizzare lo spazio "commento"**
- **Allocare dati correlati contigui tra loro in un unico "blocco"**
  - **le sessioni di comunicazione trasferiscono tali "blocchi"**
  - **sovradimensionare i blocchi per tenere conto di eventuali modifiche**
- **Utilizzare tutti i "tools" del sistema di sviluppo per una buona:**
  - **significatività del dato**
  - **reperibilità del dato (in quali parti del programma viene utilizzato)**
  - **correlazione (a quali altri dati è correlato)**
- **I tipi di dati supportati sono semplici e "hardware oriented"**
  - **Bit per la gestione di segnali digitali (segnale a due livelli)**
  - **Word per la gestione di segnali analogici (es. 2^{16} livelli tra 0 e 10V)**
  - **Double word per vaibili di tipo "Time", e così via (strutture dati semplici)**

PLC: STRATEGIE DI PROGRAMMAZIONE

- **Ripartire il problema in più sottoproblemi**
- **Ogni segmento deve poter essere commentabile in modo chiaro e compiuto (strutturare il programma in sottoprogrammi e suddividere i segmenti complessi in più segmenti semplici facendo uso di merker come variabili intermedie)**
- **Seguire le logiche in sicurezza, ridondando gli interblocchi su più segmenti**
  - **Esempio: selettore RICETTE 1,2,3 (R1,R2,R3)**
    - **Senza interblocchi: Se R1 allora…**
    - **Con interblocchi: Se R1&!R2&!R3 allora…**
- **Considerare condizioni di guasto tipico (Es. strappo cavi)**
- **Raggruppare le condizioni relative ad un certo stato di una o più uscite secondo la logica del "minimo impatto delle modifiche" (se si aggiungesse una condizione, si dovrebbe modificare il programma nel minimo numero di punti)**
- **Uscite e merker devono essere assegnati una sola volta all'interno di ogni ciclo di scansione**

PLC: il concetto di interblocco

- **L'interblocco unilaterale: Il semplice problema del Set Reset di un motore**

```
   Start   Motore
|---| |-------(S)
    Stop    Motore
|---| |-------(R)
```

L'ordine dei segmenti Influisce sulle funzionalità

```
   Start  Stop    Motore
|-----| |----|\|-----(S)
     Stop          Motore
|-----| |------------(R)
```

L'ordine dei segmenti non influisce -> più affidabile!

- **L'interblocco totale: il caso del selettore Locale Remoto**

```
    Locale   StartL   Motore
|---| |-----| |-----(S)
   Remoto   StartR   Motore
|---| |-----| |-----(R)
```

L'ordine dei segmenti Influisce sulle funzionalità

```
   Locale  Remoto  StartL  Motore
|-----| |----|\|----| |-----(S)
   Locale  Remoto StartR  Motore
|-----|\|----| |----| |-----(R)
```

L'ordine dei segmenti non influisce -> più affidabile!

PLC: il concetto di variabile intermedia (merker)

- **Merker = equivalente del relè ausiliario nei circuiti elettromeccanici (Più semplicemente… una variabile di memoria gestita nella tabella dei simboli)**

```
   Start   Motore
|---| |-------(S)
   Stop    Motore
|---| |-----|--(R)
   Allarme1 |
|---| |-----|
   Allarme2 |
|---| |-----|
   Allarme3 |
|---| |-----|
```

```
   Allarme1
|---| |-----|---(Allarmi)
   Allarme2 |
|---| |-----|
   Allarme3 |
|---| |-----|
   Start   Motore
|---| |-------(S)
   Stop    Motore
|---| |-----|--(R)
   Allarmi  |
|---| |-----|
```

- **Il merker Allarmi (es. M2.3) aumenta la leggibilità**

PLC: il concetto di variabile intermedia (merker)

- **I merker sono organizzati a byte, come gli ingressi e le uscite booleani**

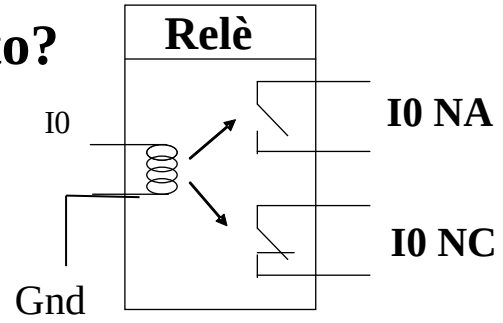
  I merker sono organizzati a byte su indirizzi fissi
  (MB5 = quinto byte dell'area di memoria M)

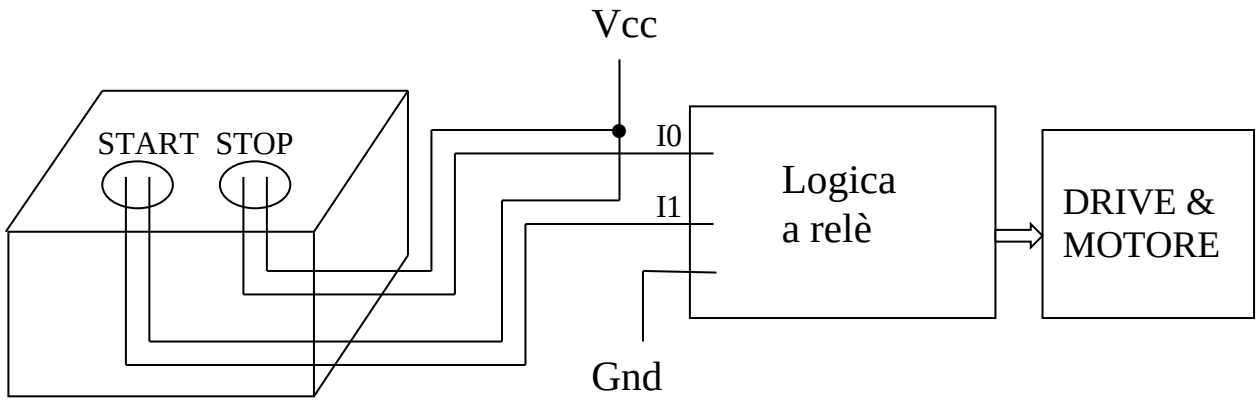
  Per accedere a Word, si usa MW ma si occupa l'indirizzo e l'indirizzo successivo
  Normalmente le word si fanno partire da indirizzi pari (anche lasciando "buchi")
  (MW4 = MB4+MB5)
  Attenzione:
  Var_12 = MB4
  Var_37 = MW4
  Il sistema mi permette questa dichiarazione, ma se modifico Var_12, automaticamente si modifica anche Va_37 e viceversa)

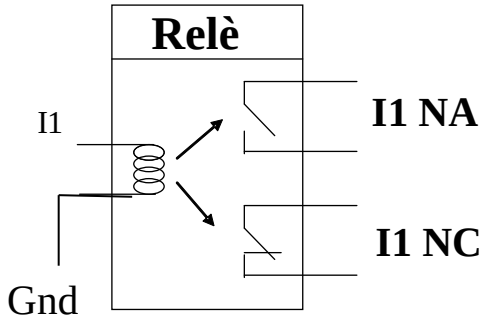
  Per accedere a Long, si usa MD (stessa logica vista per le word)
  Per accedere ai bit, si usa M13.0 (bit 0 –LSB- di MB13)

Contatti NA e NC

- **Considerare condizioni di guasto tipico (Es. strappo cavi, sensore non alimentato)**
- **Cosa succede se strappo i cavi? Come si comporta il sistema?**
  - **Contatti NA per i segnali abilitanti (Es. START, Ripristino_Allarme,…)**
  - **Contatti NC per i segnali inibenti (Stop, Allarme,…)**
- **E se mi cambiano un NA con un NC devo ricablare tutto?**
- **Si usano relè di appoggio così da realizzare logiche che vedono sempre ingressi NA (E' il principio dei PLC)**

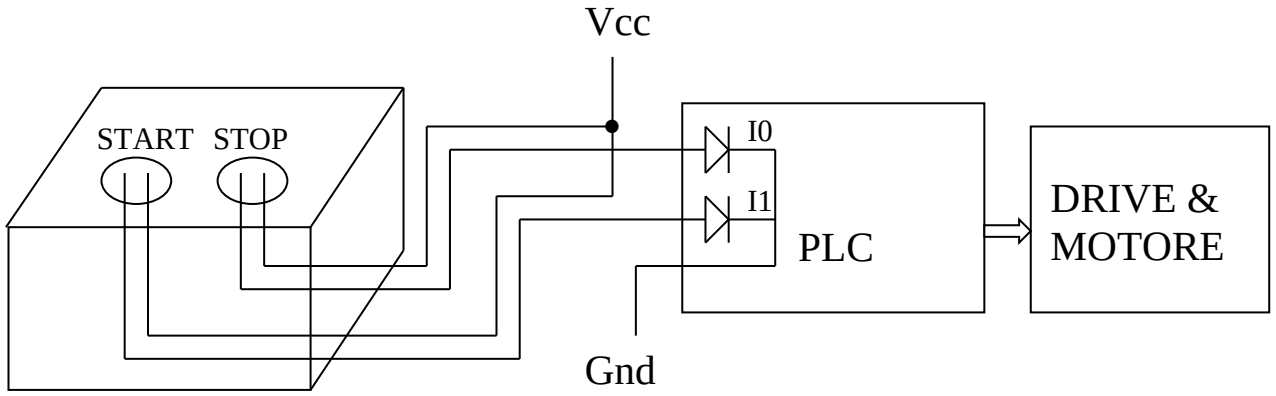






PLC: strategie di programmazione

- **Considerare guasti tipici, sostituzione di componenti, modifiche**
  - **Contatti NA per i segnali abilitanti (Es. START, Ripristino)**
  - **Contatti NC per i segnali inibenti (Es. STOP, Allarme)**
  - **Si usavano relè di appoggio per gestire sempre logiche in NA**
  - **L'ingresso del PLC è come un relè di appoggio**
  - **Se l'ingresso mi arriva NA l'interrogazione "è attivo?" corrisponde a --| |--**
  - **Se l'ingresso mi arriva NC l'interrogazione "è attivo?" corrisponde a --|\|--**



PLC: strategie di programmazione

- **Il programma deve essere leggibile**
- **Vale la logica del "minimo impatto delle modifiche"**
  - **Un programma è fatto meglio di un altro se una modifica sulle funzionalità implica un numero minore di modifiche al programma**

```
   Start  Stop  Motore
|---| |----|\|---(S)
   Stop         Motore
|---| |----------(R)
```

Ottimo se gli ingressi sono NA

```
   Start  Stop  Motore
|---| |----| |---(S)
   Stop         Motore
|---|\|----------(R)
```

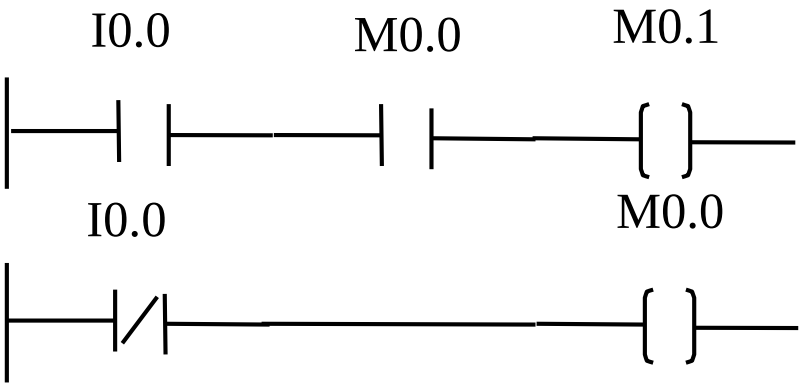
Ingresso Stop NC
"Se c'è Start e Stop avvia Motore??"

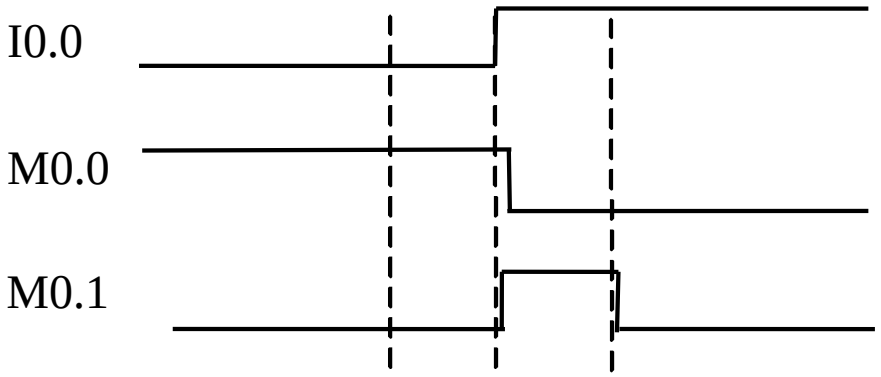
```
   Start  Start_M
|---| |-----( )
   Stop   Stop_M
|---|\|-----( )

   Start_M Stop_M  Motore
|---| |-----|\|-----(S)
   Stop_M          Motore
|---| |-------------(R)
```

Leggibile e con minimo impatto delle modifiche

LADDER: ORDINE DEI SEGMENTI

- **Si deve porre grande attenzione nell'ordine dei segmenti**
- **Se si prova a invertire la posizione dei due ladder, sul segnale M0.1 non si ha più la rilevazione del fronte di salita di I0.0**





Infatti, perché si abbia un impulso su M0.1, M0.0 deve essere ritardato rispetto a I0.0
NOTA: nei PLC esiste l'operatore "rilevatore di fronte", al quale è associata una cella di memoria (Nell'esempio M0.0)

GESTIONE DI MOTORI: STRATEGIE

- **MODO JOG**

  Quando il motore viene cablato (o in caso di guasto) deve esserci localmente una modalità di prova

  - **Pulsante JOG -> il motore va (a velocità limitata) solo mentre il pulsante è premuto**

- **START-STOP**

  Esiste una modalità START-STOP (il pulsante avvia o arresta) su quadro (locale) e su pulpito di comando (remoto)

  - **Può esistere una modalità START-STOP da remoto**

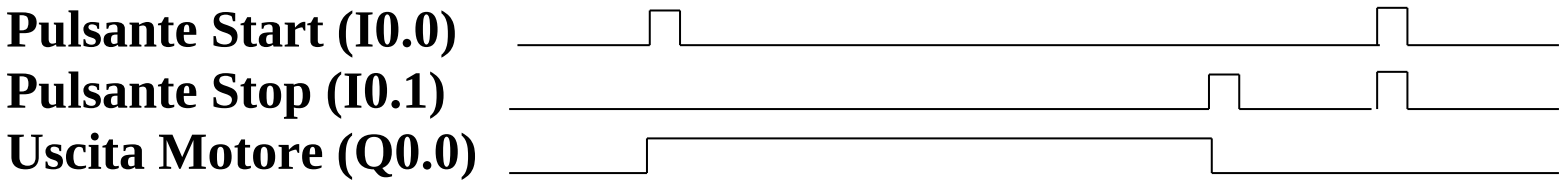
- **AUTOMATICO**

  Ad impianto avviato, il motore è normalmente gestito in automatico

- **….**

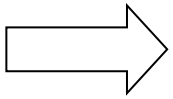
- **Il programma deve "separare i contesti" -> uso di variabili "motore virtuale"**

MOTORI: logiche di Start/Stop



- Deriva dalla logica a relè (autoritenuta Reset-prevalente)
- E se il pulsante di Start si incastra?

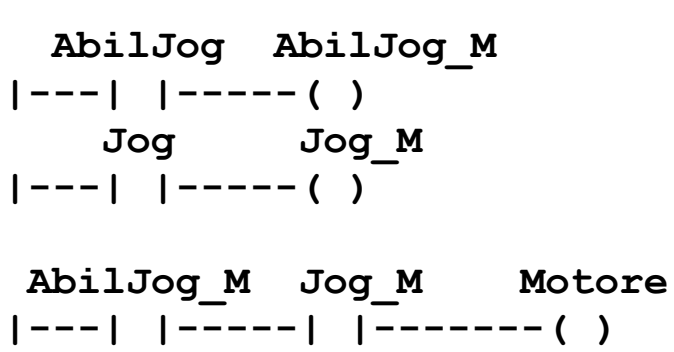
```
   Start  Start_M
|---| |-----( )
   Stop   Stop_M
|---|\|-----( )

   Start_M Stop_M  Motore
|---| |-----|\|-----(S)
   Stop_M          Motore
|---| |-------------(R)
```



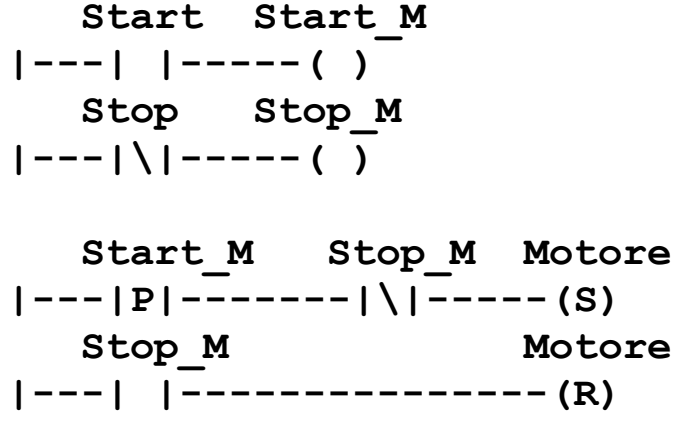
```
   Start  Start_M
|---| |-----( )
   Stop   Stop_M
|---|\|-----( )

   Start_M      Stop_M  Motore
|---| |-----|p|---|\|-----(S)
   Stop_M          Motore
|---| |-------------(R)
```

GESTIONE DI MOTORI: STRATEGIE

- **Il motore può essere gestito in modi diversi**

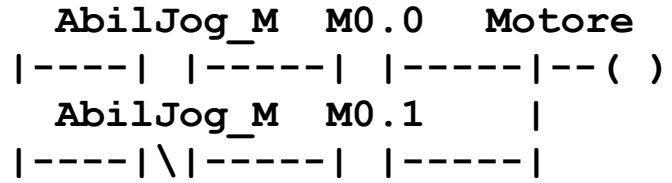


M0.0



M0.1

- **Presi singolarmente i due programmi funzionano, messi insieme no**
- **<u>Istruzioni di Set-Reset e di Assegnazione non possono coesistere</u>**
- **E' meglio creare delle memorie di appoggio (motori virtuali M0.0 e M0.1)**



GESTIONE DI MOTORI: STRATEGIE

- **Progetto del programma = Segmentazione del programma in più parti (stazioni)**
  (Stazione = sottoparte omogenea e ben identificabile dell'impianto)
- **In ogni stazione c'è tipicamente un motore principale, ma possono esserci dei motori ausiliari (ventilatori, condizionatori,…)**
- **Il motore principale può essere gestito in più modi (locale, pulpito, remoto, automatico,…)**
- **Per ogni stazione vengono definite:**
  - **le procedure di inizializzazione**
  - **il programma, ripartendo tra sequenze "in manuale" e "in automatico"**
    (l'insieme dei comandi manuali è più vasto di quelli automatici)
  - **Cronologicamente si affrontano:**
    - **le sequenze "in manuale" (da locale, da pulpito,…in logica di sicurezza)**
    - **le sequenze "in automatico"**
      (ridondanza e timeout sui sensori che determinano la transizione)
    - **strutture dati per la tracciabilità e strutture dati condivise con SCADA**