# Corso di
# “Automazione Industriale”

Prof. Ferrarini

# Appunti sui PLC

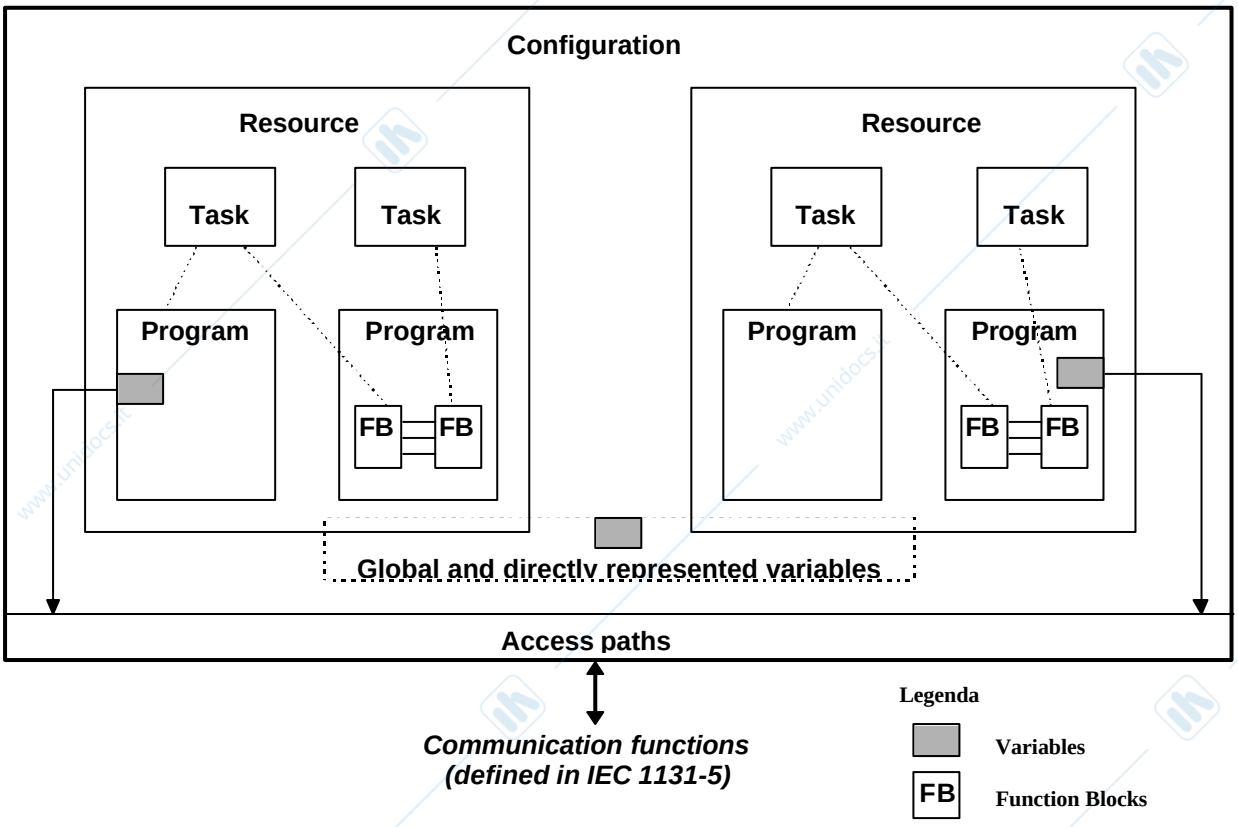
# 0. SOMMARIO

# 1. IEC 6-1131

## 1.1 Introduzione alla IEC 6-1131

La IEC 6-1131 è una normativa internazionale pubblicata per la prima volta nel 1993 sull'uso dei PLC e la parte della normativa che si occupa esclusivamente della loro programmazione è la terza.
Tra gli scopi principali di tale normativa vi sono il tentativo di ridurre il divario con le moderne tecniche informatiche, formalizzare in modo più astratto ed essenziale i problemi di automazione e controllo, e infine proporre anche uno standard nel campo del controllo logico, per favorire la riusabilità della conoscenza e una reale concorrenza tra i costruttori.

## 1.2 Il modello software proposto da IEC 6-1131 parte 3




Il termine Configuration che fa da intestazione del modello, corrisponde ad un Programmable Controller System, cioè generalmente ad un PLC, mentre i due Resource costituiscono il supporto di esecuzione dei programmi. Le resource sono inoltre autonome tra loro. I Program, invece, sono eseguiti sotto il controllo di zero o più task. Essi sono dei contenitori di costrutti eseguibili, scritti nei linguaggi previsti dallo standard.

## *1.3 Task e programmi*

I task specificano l'attivazione di parti di programmi, o interi programmi loro assegnati. Tale attivazione può essere periodica, oppure condizionata al verificarsi di un particolare evento; quindi i programmi costituiscono l'apice di una struttura che il progettista può comporre gerarchicamente utilizzando i Blocchi funzione (function block) e le Funzioni (function). Tali blocchi funzione e le funzioni costituiscono quindi gli elementi di riferimento per il deciso orientamento alla modularità, propugnato dallo standard.

### 1.3.1 Funzioni

Le funzioni sono porzioni di costrutti eseguibili che restituiscono un valore, dipendente dagli ingressi, senza avere variabili "interne" (di stato). Lo standard IEC 6-1131 permette la definizione e l'utilizzo di uno svariato numero di funzioni già definite in libreria, come ad esempio:

- Funzioni matematiche
- Funzioni trigonometriche
- Funzioni logiche

Permette inoltre all'utente di definire le proprie funzioni e riutilizzarle liberamente in un progetto.

### 1.3.2 Blocchi funzione

Lo standard permette inoltre la definizione di blocchi funzione (a tutti gli effetti vere e proprie routine di codice, che possono essere dotate di variabili interne, o di stato).Essi hanno la caratteristica di poter essere salvate in librerie e riutilizzate all'interno di vari progetti indipendentemente dal linguaggio usato.

### 1.3.3 I moduli software

I moduli, che nello standard sono detti POU (Program Organisation Unit), sono:

- programmi
- blocchi funzione
- funzione

Di essi, possono essere invocate anche diverse istanze, consentendo così un facile (ri)utilizzo di porzioni di progetti precedentemente sviluppati o acquistati sul mercato sotto forma di librerie.

### 1.3.4 Tipi di dati

Tra i tipi di dati che si possono utilizzare, vi sono variabili globali e locali (che vengono dichiarate in una POU). Lo standard consente una totale libertà nella scelta di nomi mnemonici, in modo da facilitare il riuso dei moduli, semplificando decisamente la stesure del programma da parte del programmatore.
Le variabili sono di due tipi: variabili generiche (ANY) e variabili specifiche (enumerated e subrange, struct e array) e vengono inizializzate entrambe con valori noti. Viene utilizzato un indirizzamento di tipo assoluto, ed inoltre esiste una proprietà chiamata “retain” per le variabili che devono essere persistenti.

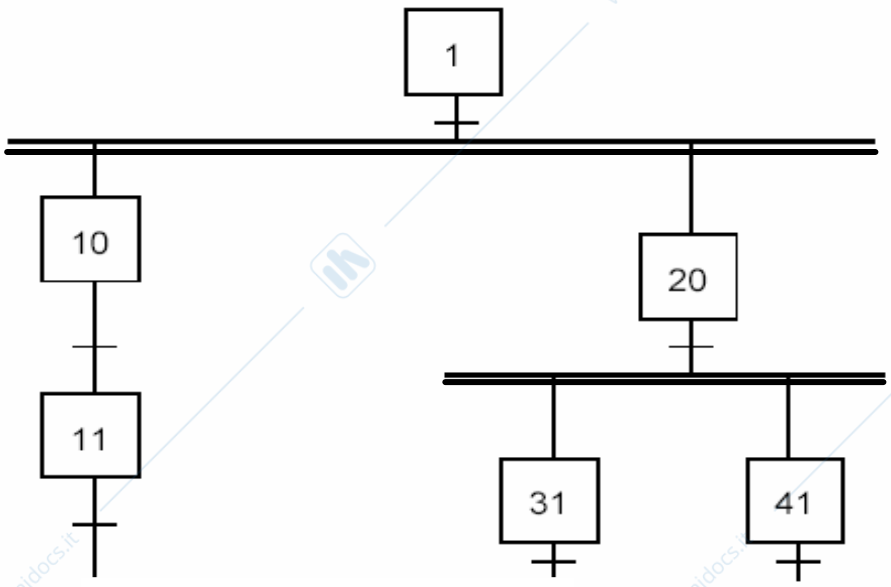
## *1.4 I linguaggi grafici e testuali*

Una POU è strutturata nel seguente modo:

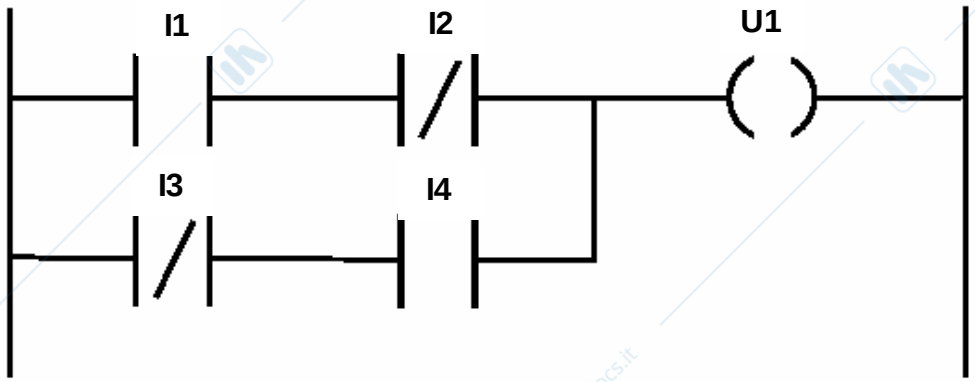
1. Sequential Function Charts (SFC)
2. Ladder Diagrams (LD)
3. Function Block Diagrams (FBD)
4. Instruction List (IL)
5. Structured Text (ST)

Dove i punti 1,2 e 3 indicano dei linguaggi di tipo “grafico”, mentre i restanti 4 e 5 indicano linguaggi di tipo “testuale”. È da notare inoltre, che il SFC, stranamente, è concepito come uno strumento di organizzazione interna di una POU, pur essendo a tutti gli effetti uno strumento formale ed eseguibile. Vediamo di seguito degli esempi scritti nei vari linguaggi.

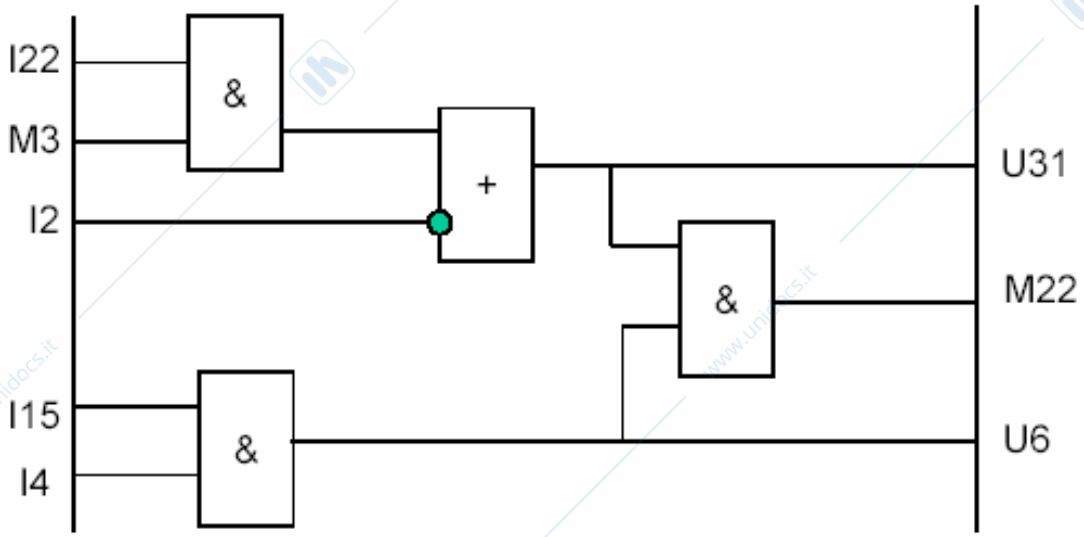
### 1.4.1 Sequential Function Chart



### 1.4.2 Ladder Diagram




### 1.4.3 Function Block Diagram




### 1.4.4 Istruction List

```
Start  LD    i25
       ADD   var
       MUL   10
       SUB   livello
       GT    25
       STO   m2
       LD    u39
```

### 1.4.5 Structure Text

```
if (livello<livello_max)
then
     valvola1=true
else
     allarme=true
     valvola1=false
end_if
aux=i25*10-4
```

# 2. SEQUENCIAL FUNCTIONAL CHART (GRAFCET)

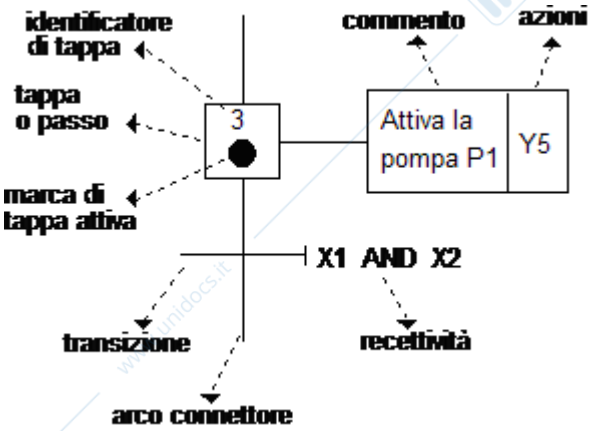
## *2.1 Introduzione*

L'SFC è stato sviluppato in Francia nei primi anni '70. L'intento dei promotori e degli sviluppatori di questo formalismo è stato non solo quello di risolvere un problema di controllo industriale preciso (cioè la specifica e la programmazione dei controllori logici programmabili), ma anche quello di definire e regolamentare uno standard industriale a cui i vari costruttori di dispositivi avrebbero potuto in un secondo tempo adeguarsi. Si osservi come in ambito accademico il primo obiettivo è quello cui tipicamente si mira, mentre in ambito industriale il secondo obiettivo è sentito come prioritario o comunque di importanza strategica. La non standardizzazione della programmazione dei PLC infatti lega i gestori di impianti ai produttori di PLC, magari anche per lunghi periodi al fine di non dovere investire tempo e denaro per istruire il proprio personale e convertire il codice preesistente.
Gli SFC sono diagrammi a passi, adatti per impostare i livelli di astrazione più elevati del progetto. Questi SFC ricordano i diagrammi a flusso di programma (flow chart) usati nella progettazione del software e servono ad identificare i passi concettuali principali in cui è possibile scomporre una logica di controllo di un impianto e a definire i criteri di passaggio da un passo ad un altro. Più in dettaglio, gli elementi sintattici fondamentali di uno SFC sono i passi (step), le azioni (action) associabili ai passi, e le transizioni (transition). Le azioni sono (richieste di) comandi inviati agli attuatori. I passi sono considerabili come "stati" della sequenza di controllo, i quali possono essere attivi o non attivi, e ad essi possono essere associati insiemi di azioni da svolgere contemporaneamente. Le transizioni rappresentano il passaggio tra passi. Tale passaggio è condizionato al verificarsi della condizione logica associata alla transizione stessa, la quale può essere specificata in uno dei quattro linguaggi sopra citati. Le azioni possono essere a loro volta specificate da uno schema SFC (azioni composte) oppure specificate direttamente, facendo riferimento alle variabili di stato e uscita. Nel primo caso, l'utente ha a disposizione un meccanismo di astrazione (attraverso la gerarchia di azioni), il quale è utile anche nella definizione di componenti di una libreria di azioni composte. Nel secondo caso le singole azioni, che devono poi essere codificate con uno o più dei quattro linguaggi di programmazione come di seguito specificato, possono essere dotate, nello schema SFC, di qualificatori riassunti nella tabella seguente.

| | | |
|---|---|---|
| **N** | Non-stored | L'azione termina quando il passo diventa inattivo |
| **S** | Set (Stored) | L'azione continua anche quando il passo diventa inattivo, e termina quando l'azione viene resettata |
| **R** | Reset | Termina un'azione attivata con i qualificatori S, SD, SL o DS |
| **L** | time Limited | L'azione comincia quando il passo diventa attivo e continua finché il passo diventa inattivo o trascorre un certo intervallo di tempo |
| **D** | time Delayed | Un timer viene settato quando il passo diventa attivo; se il passo è ancora attivo dopo l'azzeramento del timer, l'azione comincia e termina quando il passo si disattiva |
| **P** | Pulse | l'azione comincia quando il passo diventa attivo/disattivo e viene eseguita una sola volta |
| **SD** | Stored and time Delayed | L'azione comincia dopo un ritardo anche se il passo diventa inattivo e continua finché non resettata |
| **DS** | Delayed and Stored | Un timer viene settato quando il passo diventa attivo; se il passo è ancora attivo dopo l'azzeramento del timer, l'azione comincia e continua finché non resettata |
| **SL** | Stored and time Limited | L'azione comincia quando il passo diventa attivo e continua finché non viene resettata o non trascorre un certo intervallo di tempo |

Infine, si vuole fornire una piccola retrospettiva storica sul formalismo degli SFC. Questo infatti deriva in modo diretto dallo standard francese Grafcet, elaborato fin dagli anni '70. Il Grafcet è nato dallo sforzo congiunto di costruttori ed utilizzatori di PLC e di associazioni scientifiche e universitarie, prime fra tutte l'AFCET (Association française de cybernétique économique et technique). L'idea ispiratrice è stata quella di definire un modello ad eventi discreti atto alla specifica per sistemi di controllo logico per automazione industriale, con la caratteristica di essere nel contempo formale e semplice da interpretare ed utilizzare. Dopo qualche anno di indifferenza da parte del mercato, nel 1988 è stata proposta da IEC una sua prima normalizzazione (IEC 60848: Preparation of function charts for control systems), e quindi una sua utilizzazione più massiccia con la normativa IEC 1131-3. Si vuole ancora ricordare che il Grafcet è un modello di sistema ad eventi discreti, come le reti di Petri, da cui differisce per la sincronizzazione dell'evoluzione (scatto delle transizioni) e per il vincolo logico sulle marcature (marcature binarie), mentre il formalismo SFC viene proposto come strumento di programmazione, e quindi costituisce un formalismo meno astratto e quindi meno indipendente dalla macchine su cui verrà eseguito.

## 2.2 I simboli utilizzati nel Grafcet




I simboli utilizzati nel Grafcet sono riportati nella figura a lato.
La tappa rappresenta il generico passo di una sequenza di comando; in qualche senso, essa rappresenta lo "stato" logico di un programma. Una tappa può essere attiva o non attiva. Nel caso sia attiva, vengono eseguite, se del caso, le azioni ad essa associate. La transizione rappresenta, invece, il passaggio dell'attivazione delle tappe. È concettualmente simile alle transizioni di stato degli automi a stati finiti. La recettività è una condizione logica usata nel determinare quale tappa eventualmente attivare. In essa possono comparire nomi di variabili di ingresso, nomi di tappe (indicate con la lettera X seguita dal numero progressivo che identifica le tappe), variabili intermedie, valori di temporizzazioni.
Le azioni sono comandi verso il campo. Possono contenere semplici indicatori di quali variabili attivare, ma anche come attivarle. In particolare, in Grafcet si possono attivare azioni a livello e azioni impulsive. Le prime attivano un'uscita fintantoché la tappa corrispondente è marcata. Le seconde producono sull'uscita un andamento ad impulso di durata finita. Tale durata deve essere specificata accanto al nome della variabile di uscita ed è comunque indipendente dal fatto che la tappa associata sia o meno attiva. Più in dettaglio, sulle varie tipologie di azioni si veda la precedente Tabella 1 sui *qualificatori* di azioni. Le azioni di una tappa possono essere multiple e anche condizionate: in questo ultimo caso contengono una istruzione del tipo:

```
if t1>5, Y5
```

con l'ovvio significato di attivare l'uscita Y5 solo se la condizione $t_1>5$ è verificata.

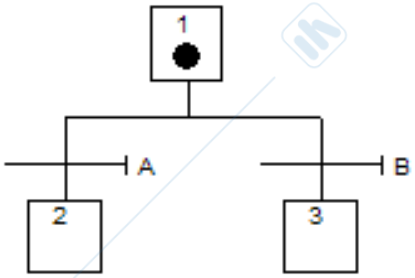
La sintassi del linguaggio Grafcet è espressa da regole, le più importanti delle quali sono qui riportate.

- Una tappa diventa attiva se la transizione a monte ha una recettività vera e se la tappa a monte di tale transizione è attiva (in questo caso si dice che la transizione è abilitata allo scatto).
- Nel caso in cui una transizione sia abilitata all'attivazione della tappa a valle la tappa a monte viene disabilitata. Se però in un certo istante la tappa a monte dovesse essere contemporaneamente attivata e disattivata, essa viene attivata.
- Tutte le transizioni che in un certo istante sono abilitate a scattare, scattano contemporaneamente.

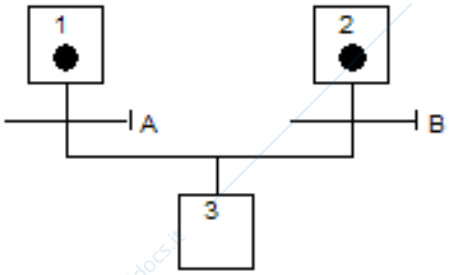
## 2.3 Gli operatori principali

Gli operatori principali sono 4, divisi in 2 gruppi: quelli di tipo OR (le due figure sottostanti),

OR - divergenza

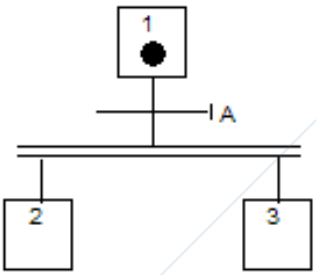



OR - convergenza

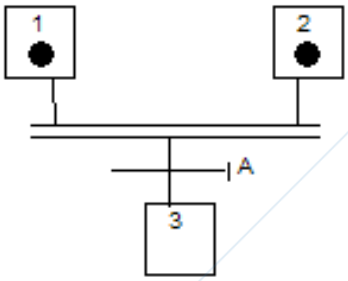



e quelli di tipo AND (figure seguenti).

AND - divergenza




AND - convergenza



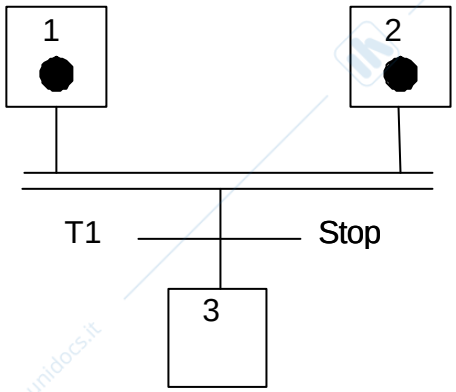
Si osservi che la struttura OR-divergenza, che ben sembra prestarsi ad esprimere situazioni di conflitto o di alternativa tra più soluzioni, può facilmente tramutarsi, ai fini del risultato, nell'operatore AND-divergenza. Ciò accade quando le recettività A e B dell'OR-divergenza non sono mutuamente esclusive. Ciò è palesemente fuorviante in quanto ci si aspetta che operatori diversi svolgano funzioni diverse. In realtà, tale situazione è ammessa da una parte per dare maggiore flessibilità al linguaggio (istruzioni diverse possono essere modellizzate in modi diversi, a seconda del contesto), dall'altra per potere tradurre in modo quasi diretto le espressioni del linguaggio naturale, poco strutturato per definizione, usate per la specifica del codice. In particolare, un uso dell'OR-divergenza è quello di realizzare costrutti chiamati combinazione di selezione e parallelo. In linguaggio naturale, essi sono espressi nel modo seguente:

> *Se si verifica la condizione A, allora esegui l'operazione O1; inoltre, se è vera anche la condizione B, allora esegui anche l'operazione O2.*

## 2.4 Regole di evoluzione

Le regole di evoluzione per l'SFC, si basano sulle seguenti condizioni in cui si possono trovare le transizioni:

- Una transizione si dice abilitata se tutti i passi a monte sono attivi
- Una transizione si dice superabile se è abilitata e la sua recettività assume il valore logico *vero* (a volte si dice che la transizione *può scattare*).

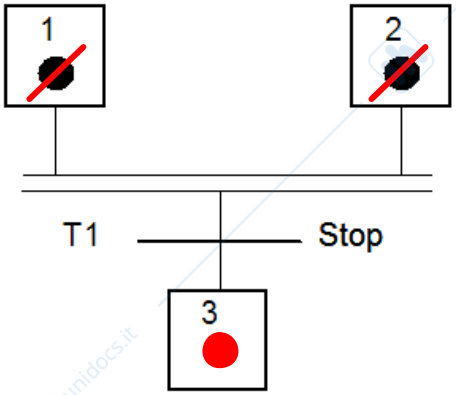



Per esempio, nella figura a lato, la transizione **T1** è *abilitata*, ed è *superabile* se la variabile **Stop** assume il valore logico "vero".

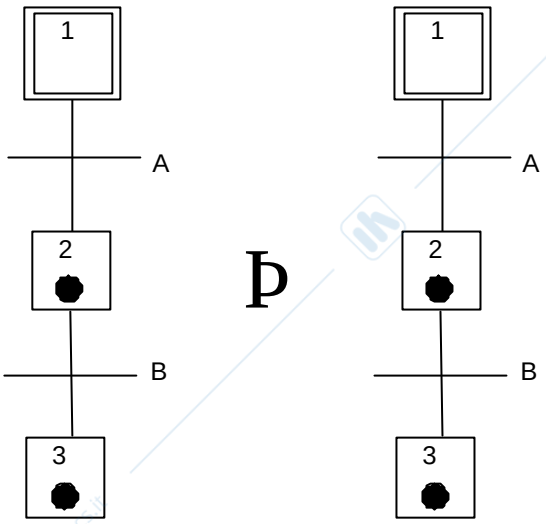
A questo punto possiamo introdurre le "regole" vere e proprie, esse sono:

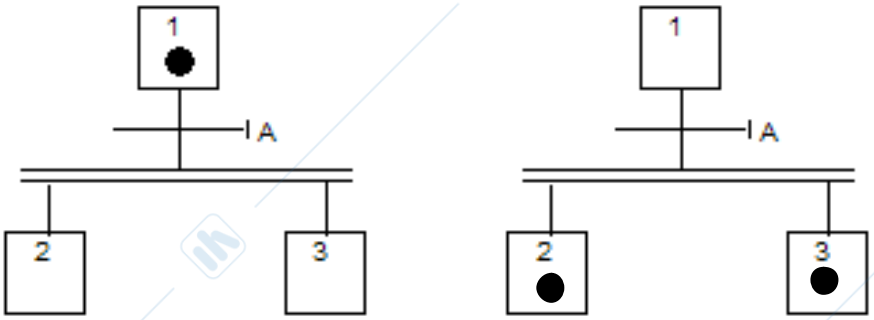
- Se una transizione è superabile, essa viene effettivamente superata: tutti i passi a monte della transizione vengono disattivati, mentre tutti quelli a valle vengono abilitate.
- Tutte le transizioni superabili in un certo istante, vengono superate contemporaneamente.

Se la situazione prima del superamento della transizione è quella illustrata nella figura a fianco, in nero, quella indicata in rosso, rappresenta la situazione dopo il superamento di **T1**.

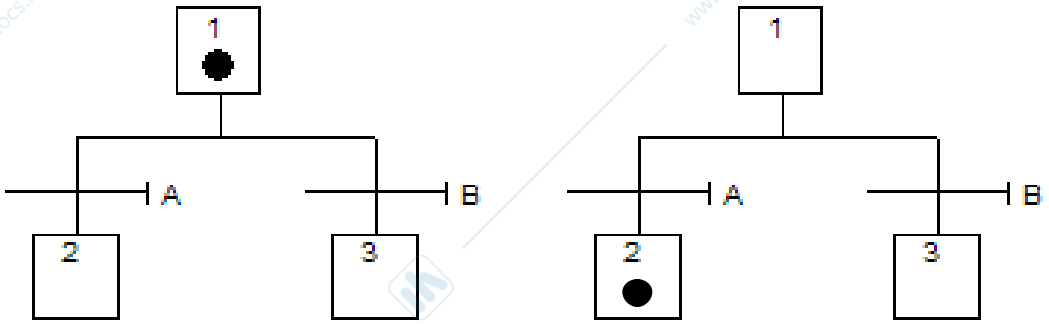


È da notare che nell'applicare le regole di evoluzione può accadere che un passo debba essere contemporaneamente disattivato e attivato. Convenzionalmente, tale passo rimane attivo ma, tuttavia, tale situazione è da usare con cautela. Chiariamo meglio il concetto con l'esempio a lato che mostra l'attivazione e disattivazione di un passo.
Le figure che seguono invece, mostrano l'evoluzione di alcune delle strutture classiche illustrate precedentemente.
In particolare, le prime due immagini rappresentano la struttura di AND-divergenza rispettivamente, prima e dopo il superamento della transizione.

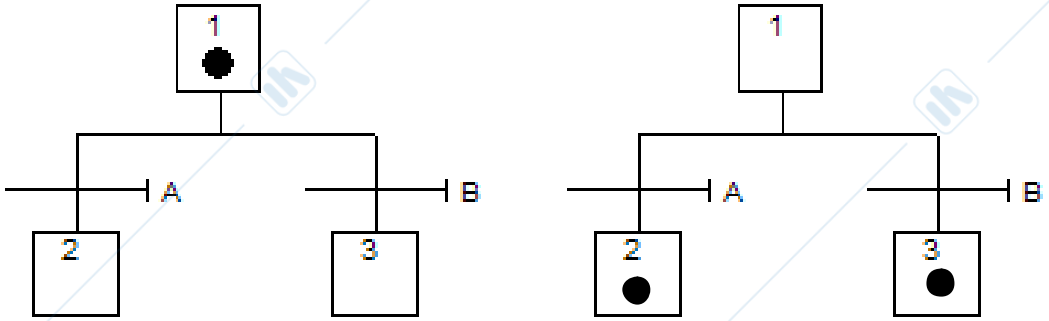



Le figure sottostanti, invece rappresentano la struttura di OR-divergenza in due differenti casi. Le prime due, il caso in cui **A**="vero" e **B**="falso", rispettivamente prima e dopo:



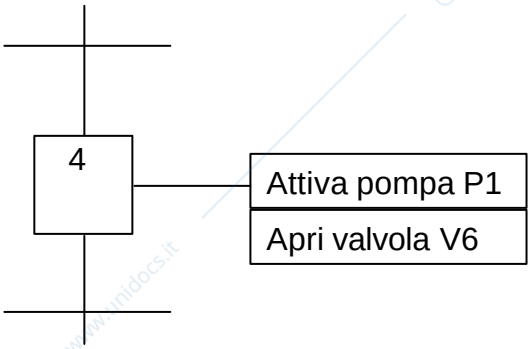
Mentre quelle che seguono rappresentano sempre la struttura di OR-divergenza, ma nel caso in cui sia **A** che **B** assumono il valore "vero".

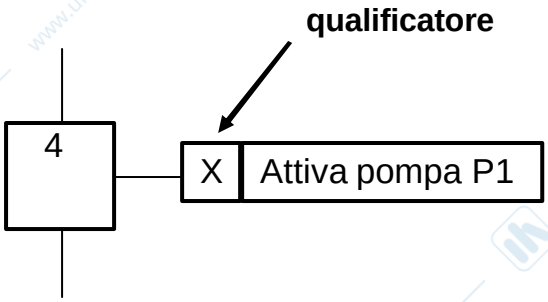



Le possibilità di confusione si possono evitare con la mutua esclusione delle condizioni, cioè assegnando una priorità alla transizione di sinistra piuttosto che a quella di destra.

## *2.5 Azioni*

Servono per esplicitare i "comandi" da emettere e sono associate ai passi; un passo può avere più azioni, eseguite in parallelo.
Inoltre le azioni hanno attributi (*qualificatori*) per descrivere in modo non ambiguo l'andamento temporale delle corrispondenti variabili di uscita e possono essere semplici o aggregate.



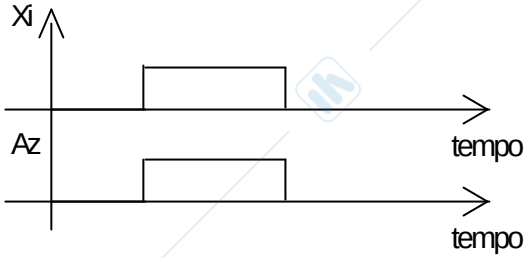



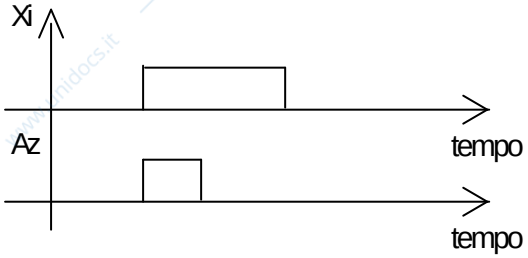

Nella pagina seguente, vediamo più in dettaglio i vari tipi di qualificatore.

### 2.5.1 Qualificatori di azioni

I qualificatori sono i seguenti:

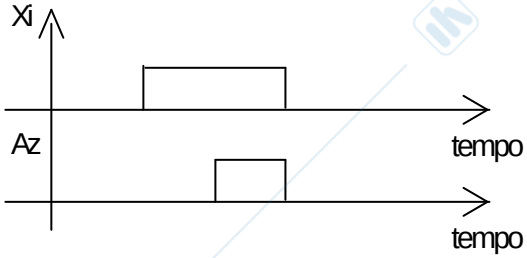
- **N** (“Non-stored”), in cui l’azione (indicata con **Az** nelle seguenti figure) termina quando il passo (indicato con **Xi** nelle seguenti figure ) diventa inattivo. Viene anche detta “continua”.




- **L** (“time-Limited”, limitata nel tempo), in cui l'azione comincia quando il passo diventa attivo e continua finché il passo diventa inattivo o trascorre un certo intervallo di tempo.

- **D** (“Delayed”, ritardata nel tempo) in cui un timer viene inizializzato e fatto decrementare non appena il passo diventa attivo; se il passo è ancora attivo dopo l'azzeramento del timer, l'azione comincia e termina quando il passo si disattiva.




- **P** (“Pulse”, impulsiva), in cui l'azione comincia quando il passo diventa attivo (o disattivo, a seconda delle opzioni) e viene eseguita una sola volta, come ad esempio, un calcolo aritmetico, la partenza di un temporizzatore, l’aggiornamento di un contatore.

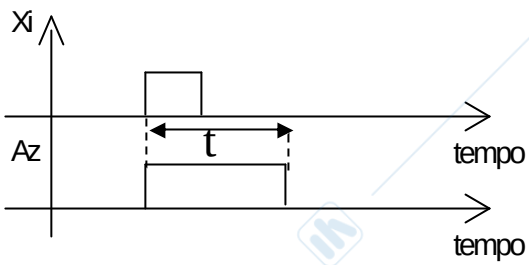
- **S** (“Stored” o “Set”, memorizzata), in cui l'azione continua anche quando il passo diventa inattivo, e termina quando l'azione viene resettata.

- **R** (“Reset”, cancellata), che fa terminare un’azione “memorizzata” (precedentemente attivata con i qualificatori **S**, **SD**, **SL** o **DS**). È da notare, inoltre, che più passi potrebbero avere la stessa azione con il qualificatore “**S**”.

- **SD** (“Stored and time-Delay”), in cui l'azione comincia dopo un ritardo anche se il passo diventa inattivo e continua finché non viene resettata.

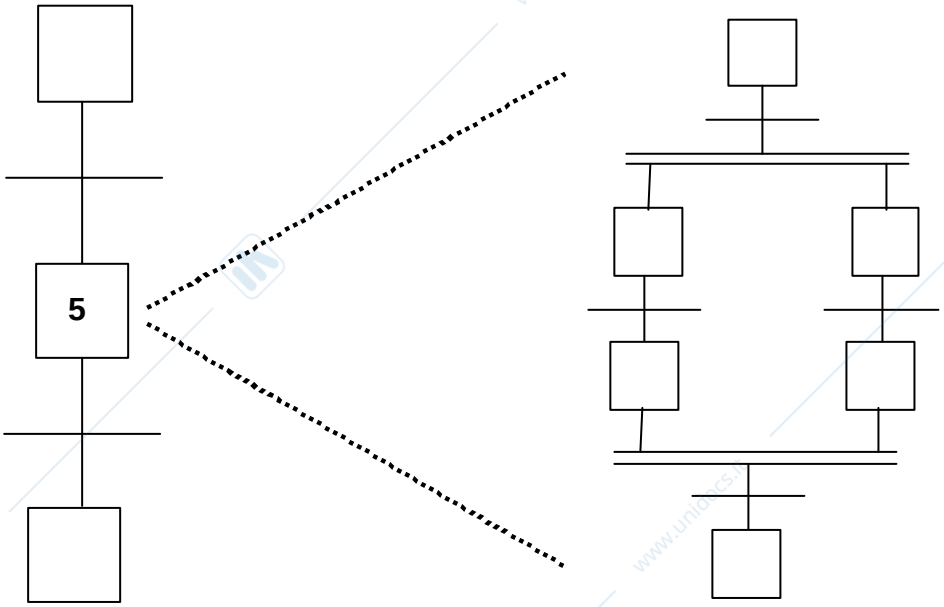
- **DS** (Delay and “Stored”), dove un timer viene settato quando il passo diventa attivo e se il passo è ancora attivo dopo l'azzeramento del timer, l'azione comincia e continua finché non viene resettata.

- **SL** (“Stored and time-Limited”), in cui l'azione comincia quando il passo diventa attivo e continua finché non viene resettata o non trascorre un certo intervallo di tempo.




### 2.5.2 Azioni aggregate

Le regole di evoluzione delle azioni aggregate sono uguali a quelle dei sottoprogrammi, cioè quando si attiva il passo associato ad un’azione aggregata, essa comincia ad evolvere e, quindi, termina quando tale passo si disattiva. Esse, inoltre permettono una gerarchia di attivazione.




Sorgono però alcune problematiche di implementazione, per esempio:

- Cosa succede alle azioni elementari eventualmente presenti nell’espansione del passo 5 quando il passo si disattiva?
- Cosa succede, invece, alle azioni elementari eventualmente presenti nell’espansione del passo 5 se la transizione a valle del passo stesso può scattare quando questo si attiva?
- Cosa fare delle variabili “locali”?
- Cosa fare delle azioni di tipo **S**, **DS**, **SD**, **SL**?

La risoluzione di tali “dettagli” non è, purtroppo standardizzata.

## *2.6 Condizioni logiche delle azioni*

Le condizioni associate alle transizioni sono espressioni logiche molto generali, e possono includere, variabili di ingresso, variabili temporali, variabili di stato e variabili "condivise".

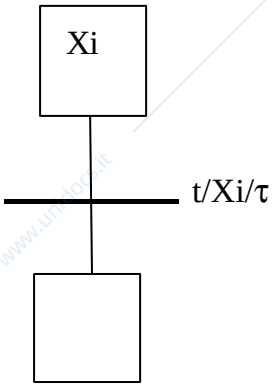
### 2.6.1 Variabili di ingresso

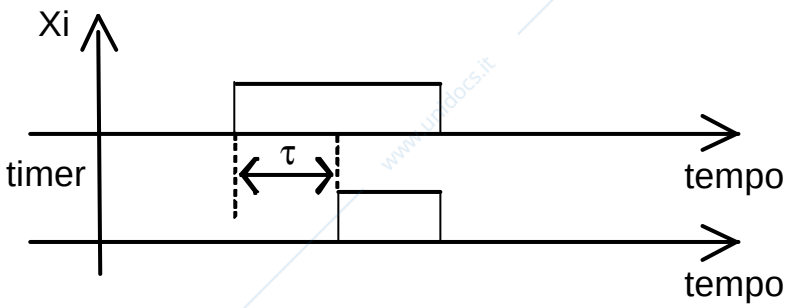
Possono essere di due tipi: booleane, ad esempio "presenza_pezzo" (indica con il valore "0" o "1" se il prezzo esiste oppure no), o reali, come potrebbe essere la variabile "temperatura". Un esempio di condizione logica potrebbe essere la seguente:

```
(temperatura ≤ 25°) AND NOT presenza_pezzo
```

### 2.6.2 Variabili temporali

Il tempo è modellizzato con dei temporizzatori (timer), associati ai passi, la cui uscita ha valore iniziale "0" e rimane a "0" anche quando il passo associato si attiva. Passa, invece, al valore "1" quando è trascorso un intervallo di tempo dato a partire dall'ultima attivazione del passo, e ritorna a "0" quando il passo si disattiva.



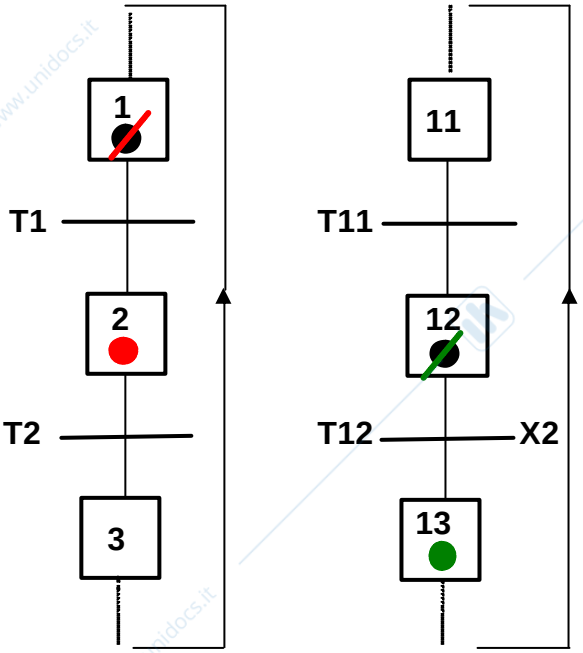
### 2.6.3 Variabili di stato

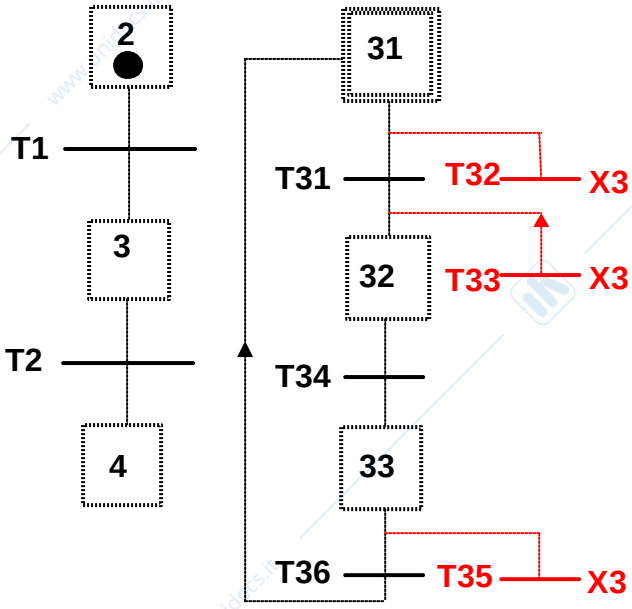
Ogni passo ha associato una variabile logica che ne rappresenta lo stato di attivazione (**Xi**) e che può comparire nella condizione associata a qualche transizione; è dunque possibile influenzare l'evoluzione di uno schema Grafcet tramite lo stato corrente di una altro Grafcet, ma tale caratteristica va usata con accortezza, poiché l'evoluzione di uno schema può facilmente bloccarsi per la presenza di troppi vincoli (topologici e logici).
Noi vedremo, a titolo di esempio:

- sincronizzazione implicita
- forzatura
- blocco

**Sincronizzazione implicita:**

Il superamento della transizione **T12** è condizionato all'attivazione del passo 2 (come in precedenza, l'evoluzione della rete è rappresentata mediante l'uso del colore rosso, per il primo scatto, e con il colore verde per il secondo).
A questo punto, però, nascono dei problemi. Il superamento di **T12**, infatti, dipende dall'attivazione del passo 3, ma quest'ultima si ha quando **T2** viene superata, e ciò dipende da **X13** che, sfortunatamente, si attiva solose **T12** viene superata. Si ha così un conseguente blocco dello schema.
Per risolvere questo inconveniente si può procedere nel seguente modo:



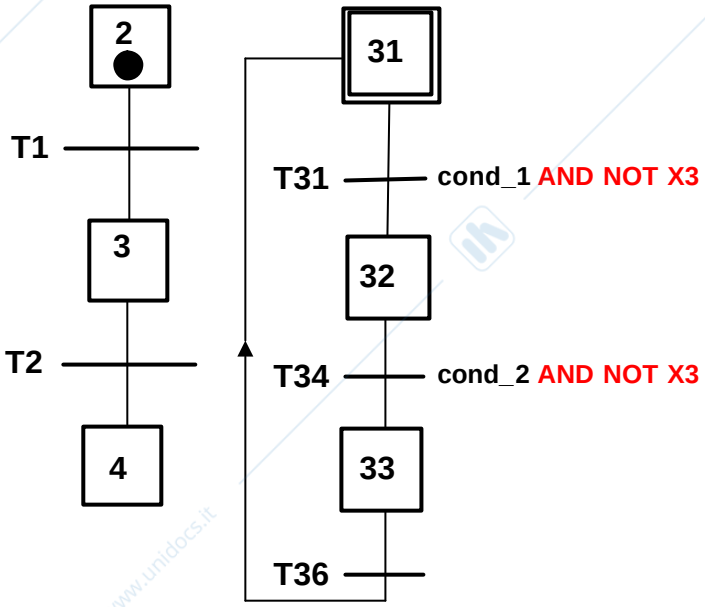




**Forzatura:**

Il passo corrente (cioè lo stato) della porzione di schema sulla destra viene forzato al passo 32, quando nello schema di sinistra si attiva il passo 3.

**Blocco:**

La porzione di schema sulla destra si "congela", quando nello schema di sinistra si attiva il passo 3.




### 2.6.4 Variabili condivise

È possibile definire una "azione fittizia" su una variabile definita in una "memoria condivisa" (variabile "interna"). Tale variabile può quindi essere usata nella condizione logica associata a qualunque transizione, come ad esempio, in un passo, è possibile incrementare un contatore e altrove è possibile leggerne il valore corrente.

## *2.7 Conclusioni*

Il Grafcet è un formalismo molto ricco: contiene strutture modellistiche che servono a rappresentare quanto serve nella maggioranza dei problemi di automazione industriale (controllo logico). Anzi, abbiamo anche più scelte per rappresentare un unico comportamento (ad esempio, sincronizzazione). L'evoluzione di uno schema Grafcet è data dalla topologia dello schema (cioè la disposizione logica di passi e transizioni, o meglio l'informazione che dice quale nodo è collegato a quale altro) e dalle variabili di ingresso, temporali e condivise. È difficile,inoltre, predire l'evoluzione delle variabili di uscita per eventuali azioni con qualificatori **S**, **SL**, **SD**, **DS** (presenza di ulteriori elementi di memoria di tipo "flip-flop"): non basta infatti "guardare" lo schema. Grafcet quindi è potente dal punto di vista modellistico (parallelismo: più passi possono essere contemporaneamente attivi), molto più degli automi a stati finiti, è ricco di costrutti, espressioni e variabili ed inoltre "facile" da simulare. Purtroppo però è difficile da analizzare.

# 3. LADDER DIAGRAM

## 3.1 Introduzione

Il più diffuso linguaggio di programmazione per i controllori a logica programmabile, o PLC, è il "Ladder Diagram", che letteralmente significa "diagramma a scala"; un nome giustificato dalla disposizione grafica dei suoi simboli che ricorda proprio una scala.
In italiano, però, viene usato maggiormente il termine "linguaggio a contatti" o "diagramma a relè". Questa denominazione invece, trae origine dal fatto che gli elementi di base del linguaggio ricordano contatti elettrici. Nel seguito, per semplicità, certi di non recare alcuna confusione, si utilizzerà sempre il termine "LD" per indicare il ladder diagram anche nelle sue diverse traduzioni.

## 3.2 Caratteristiche principali

Il ladder diagram è un linguaggio grafico che si basa sulla trasposizione, in logica di programmazione, del funzionamento di una rete elettrica molto semplice, in cui l'obiettivo è alimentare opportuni utilizzatori elettrici (bobine) tramite interruttori (chiamati anche contatti o relè). Il motivo che giustifica tale trasposizione è storico: le logiche di controllo prima del PLC, infatti, venivano realizzate direttamente con circuiti elettrici, quindi apparse subito naturale sviluppare un linguaggio che fosse il più vicino possibile a quel tipo di tecnologia e che potesse, quindi, essere compreso ed utilizzato anche dai tecnici dell'epoca, i quali possedevano una scarsa, o nulla conoscenza informatica.
Le prime istruzioni disponibili nel linguaggio a contatti furono quindi proprio quelle che rappresentavano il contatto normalmente aperto o normalmente chiuso di un relè, e in seguito sono state poi stabilite una serie di convenzioni riguardanti la disposizione degli elementi grafici e i rispettivi nomi per estenderne le potenzialità e l'espressività. Di seguito ne vediamo alcuni.

## 3.3 Convenzioni

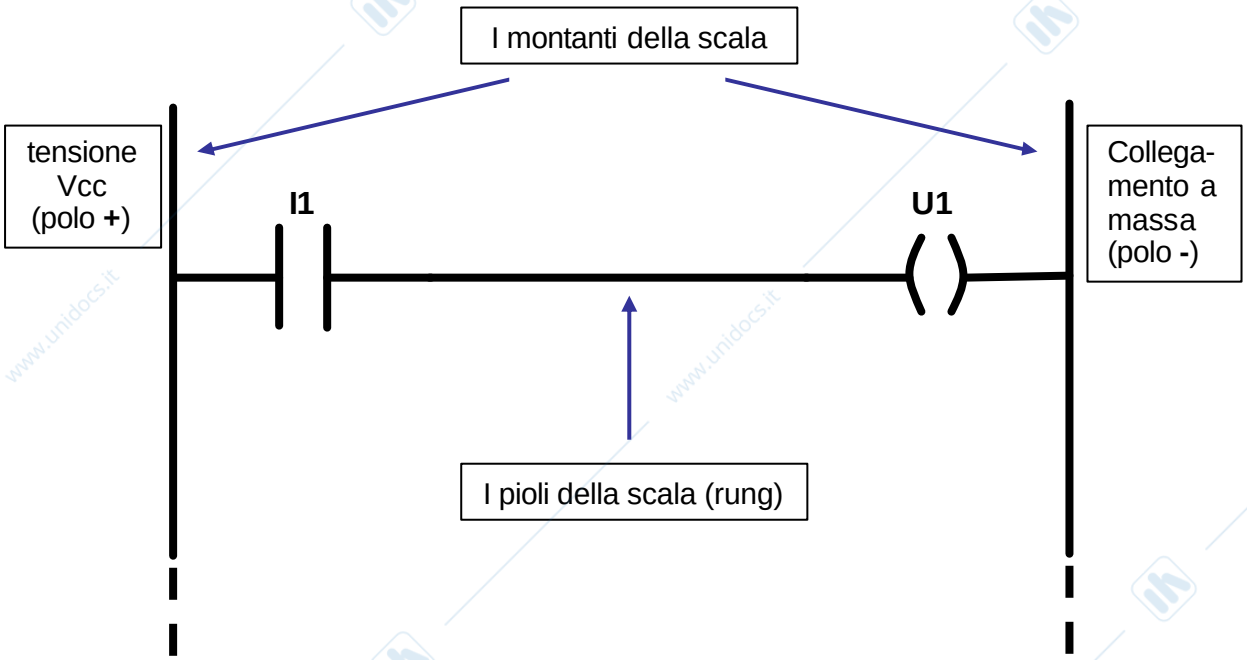
### 3.3.1 Area Dati e convenzioni tipiche

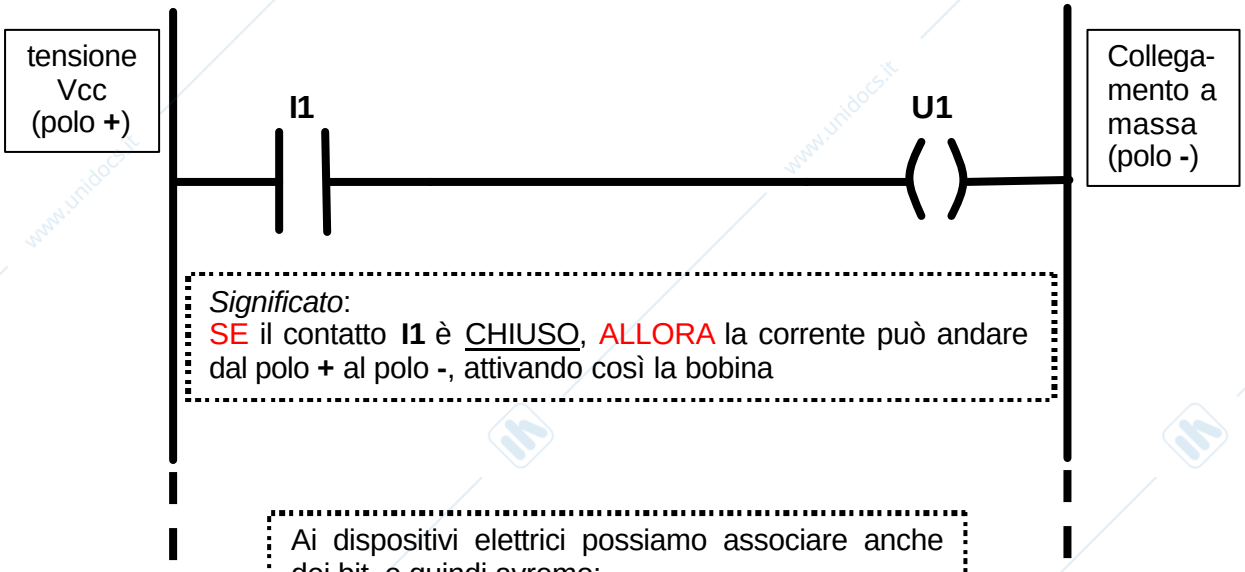
Premettendo che le convenzioni cambiano da costruttore a costruttore, noi useremo solo i nomi simbolici più semplici.

| | |
|---|---|
| Area degli Ingressi | **Ix:y** ( Il bit y della word x) |
| Area delle Uscite | **Ux:y** |
| Area dei Temporizzatori (timer) | **T1…Tn** |
| Area dei Contatori | **C1…Cn** |
| Area PID, e Area Utente | **P1…PN e W1…Wn** |

### 3.3.2 Elementi di Base del ladder diagram

Di seguito sono riportati gli elementi principali che compongono un programma in ladder diagram.




*Significato*:
SE il contatto **I1** è CHIUSO, ALLORA la corrente può andare dal polo **+** al polo **-**, attivando così la bobina

Ai dispositivi elettrici possiamo associare anche dei bit, e quindi avremo:
SE **I1** è CHIUSO, ALLORA **U1** si attiva cioè

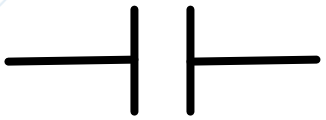
SE **I1** =1 , ALLORA **U1** = 1, ALTRIMENTI **U1** = 0

### 3.3.3 Elementi di base per gli ingressi

Gli elementi di base del LD sono le bobine e i contatti. Il contatto può essere associato ad un bit di ingresso (**Ix:y**), uscita (**Ux:y**), interno (**Wx:y**), e anche al bit di stato di contatori e temporizzatori. Le bobine, invece, possono rappresentare un'uscita fisica (**Ux:y**) o un marker interno (**Wx:y**). Convenzionalmente, gli ingressi si mettono sulla sinistra del grafo, mentre le uscite sulla destra.

**Contatto normalmente aperto:**

- Se il bit associato vale 1, il contatto si chiude
- Se il bit associato vale 0, il contatto si apre

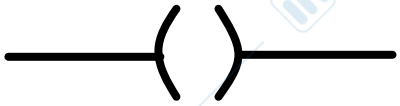


**Contatto normalmente chiuso:**

- Se il bit associato vale 1, il contatto si apre
- Se il bit associato vale 0, il contatto si chiude

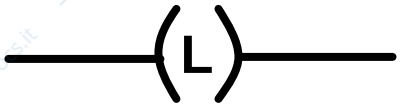
**Bobina:**

Si attiva quando passa corrente. Quindi, il bit ad essa associata sale al valore logico 1 (ON) se le condizioni logiche alla sua sinistra sono verificate e viene quindi stabilita la continuità elettrica.



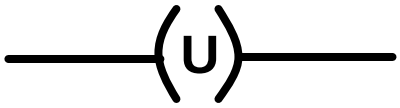
**Latch bobina:**

Mantiene lo stato logico 1 (ON) anche quando le condizioni di attivazione vengono a mancare (simile al *SET* di un *flip-flop*).

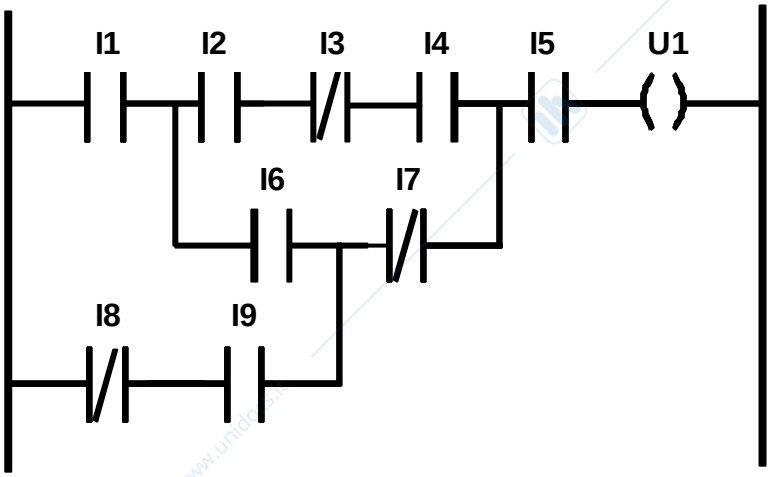



**Unlatch bobina:**

Riporta allo stato logico 0 (OFF) un'uscita (simile al RE*SET* di un *flip-flop*).



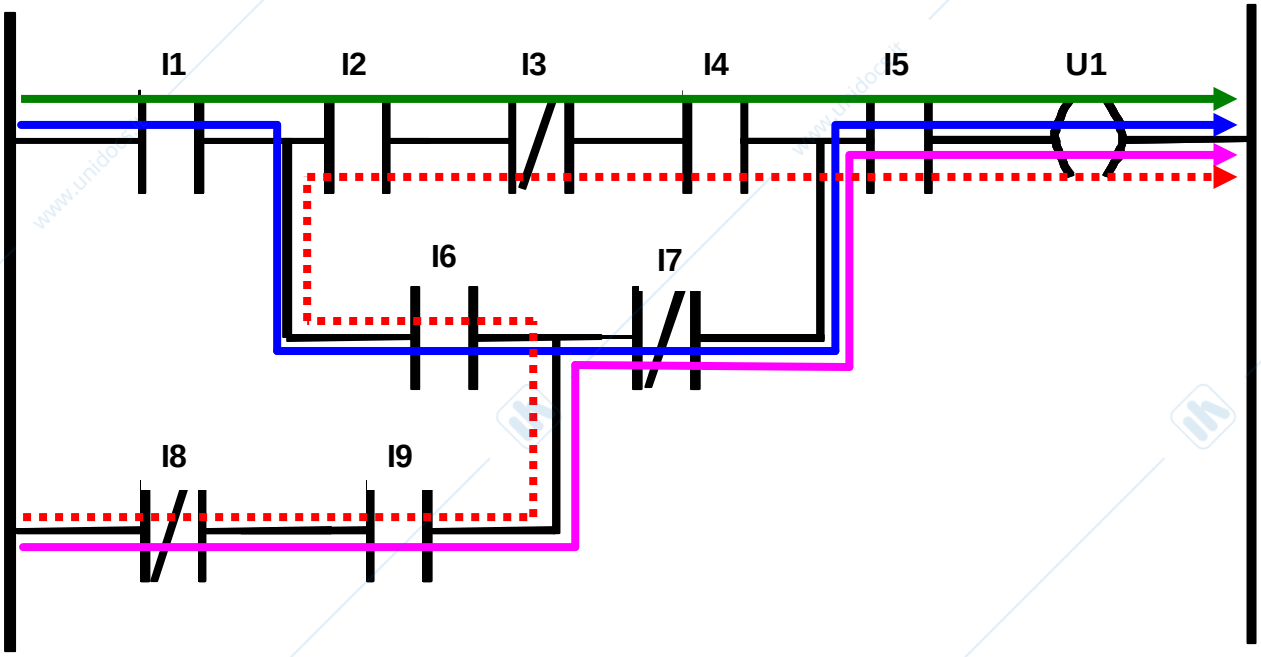
## 3.4 Costruzione di un programma




Se disponiamo i pioli (cioè le istruzioni) uno dopo l'altro, siamo in grado di costruire un programma.
È necessario però porre particolare attenzione al fatto che il LD è un linguaggio, non una rete elettrica, quindi occorre specificare come vengono interpretati i pioli. Si renderà quindi necessario definire come devono essere scanditi i pioli e quando verranno letti gli ingressi e aggiornate le uscite.
A questo scopo, imponiamo che i pioli vengano scanditi dall'alto verso il basso, e da sinistra verso destra. Quindi, nel singolo piolo, il flusso di energia nei vari dispositivi andrà solo da sinistra a destra, senza possibilità che quest'ultimo si inverta.
Nella figura seguente vediamo tre flussi di energia consentiti, e uno non consentito. Essi sono:

- I1 I2 I3 I4 I5 U1 *Consentito*
- I1 I6 I7 I5 U1 *Consentito*
- I8 I9 I7 I5 U1 *Consentito*

- I8 I9 I6 I2 I3 I4 I5 U1 *Non Consentito*

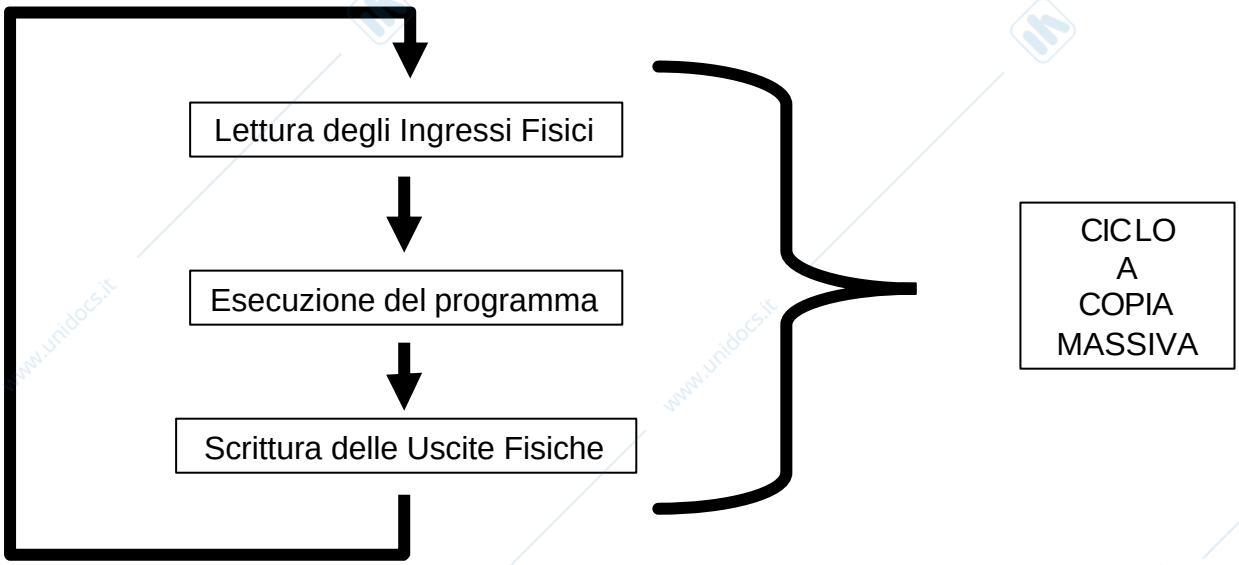



Quindi concludendo possiamo scrivere:

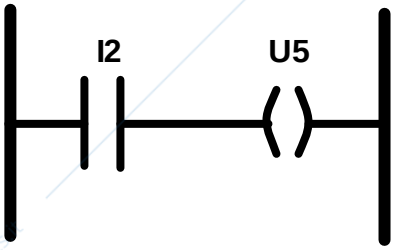
**U1**= **I5** AND [(**I4** AND NOT(**I3**) AND **I2** AND **I1**) OR (NOT(**I7**) AND (**I6** AND **I1** OR (NOT(**I8**) AND **I9**)))]

## 3.5 Sincronizzazione con I/O

Il seguente schema illustra la sincronizzazione tra l'ingresso e l'uscita.

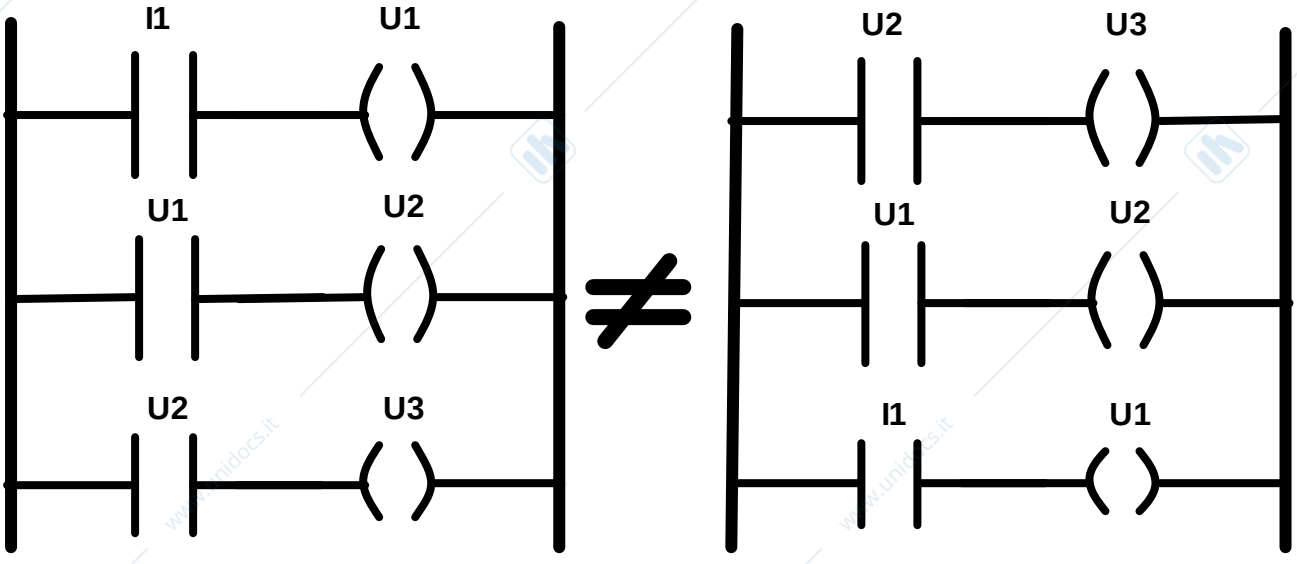



È necessario far notare che ogni piolo viene scandito in ogni ciclo (a meno delle istruzioni di salto): pertanto le uscite associate alle bobine normali (senza latch o unlatch) vengono scritte ad ogni ciclo.



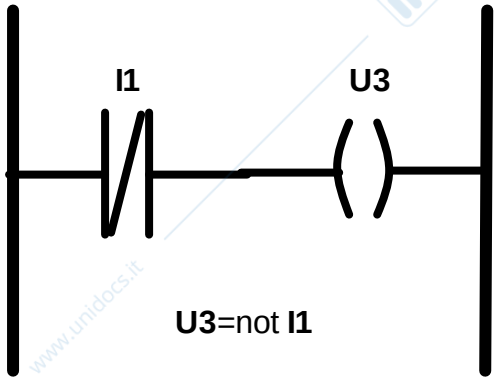

Quindi con il LD della figura a fianco *ad ogni ciclo viene scritto 0 o 1* in **U5**. Inoltre, il suo valore *permane fino alla prossima esecuzione (al ciclo successivo) della stessa istruzione.*

Inoltre, il valore delle variabili lette in ingresso rimane costante per tutto il ciclo del programma quindi i seguenti due programmi sono diversi, in quanto quello di sinistra assicurerà la contemporanea attivazione delle tre uscite alla fine del primo ciclo di esecuzione in cui il contatto **I1** viene rilevato come ON. Il programma a destra, invece, attiverà la sola uscita **U1** alla fine di quel ciclo, mentre le altre uscite verranno attivate nei due cicli di esecuzione successivi.

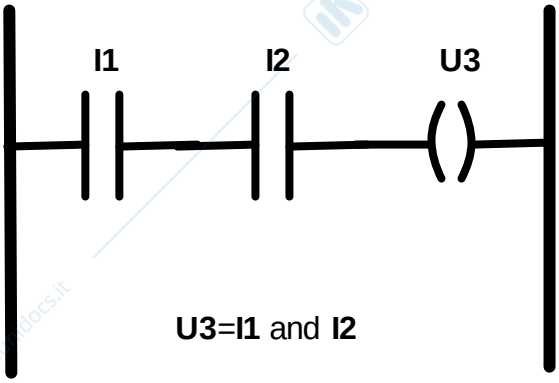


## *3.6 Esempi di programmi in LD*

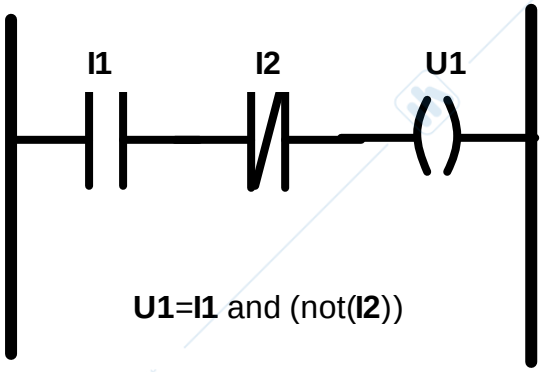
Programmazione di una **NOT**



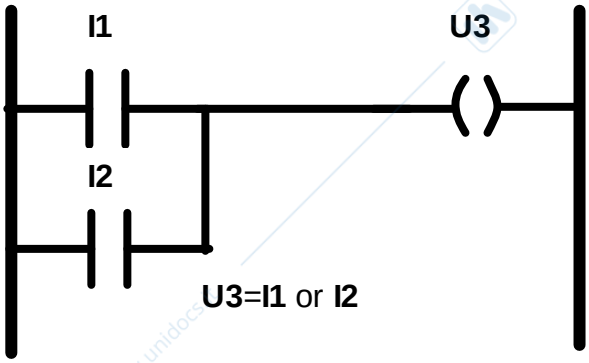

Programmazione di una **AND**



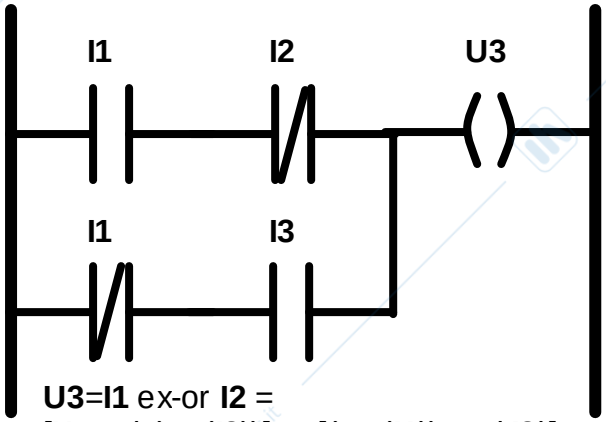

Esempio di **Rung**




Programmazione di una **OR**




Programmazione di una **EX-OR**




| I1 | I2 | U3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

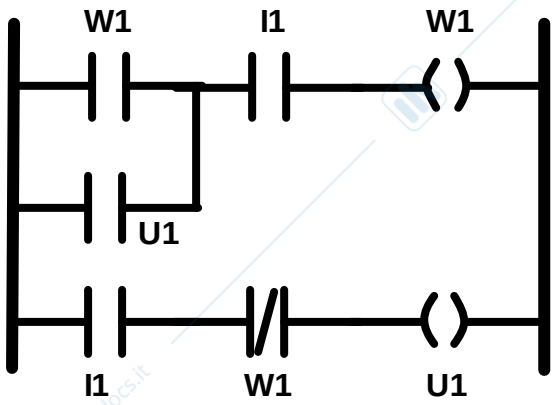
# 4. ELEMENTI DINAMICI IN LD

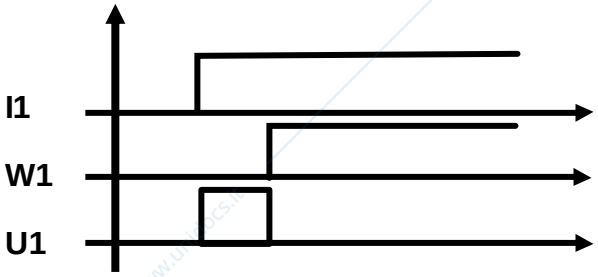
Le istruzioni di base del LD sono “logiche”, infatti assegnano un’uscita in base ad una pura combinazione logica delle variabili associate ai contatti; tuttavia, è possibile in LD associare ad un contatto anche una variabile di uscita e tale possibilità, unita all’esecuzione sequenziale e ciclica di uno schema LD, rende possibile la creazione di elementi dinamici, o con memoria, e l’identificazione di variazioni di una variabile.
Gli esempi seguenti trascurano il tempo di ciclo.

## *4.1 Elementi di memoria*

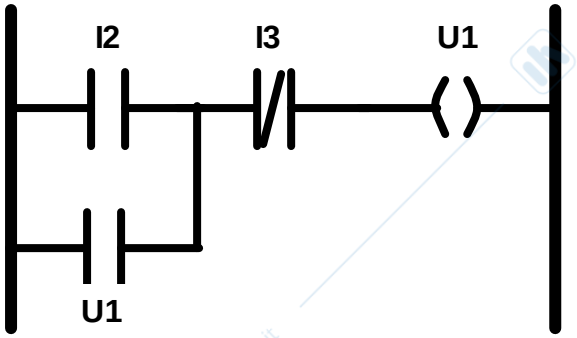
### 4.1.1 Monostabile

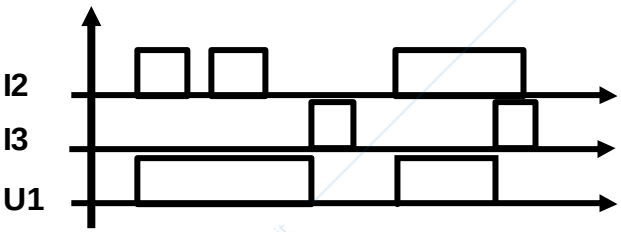
L’elemento monostabile genera un impulso di durata un tempo di ciclo dopo un fronte di salita su **I1**.



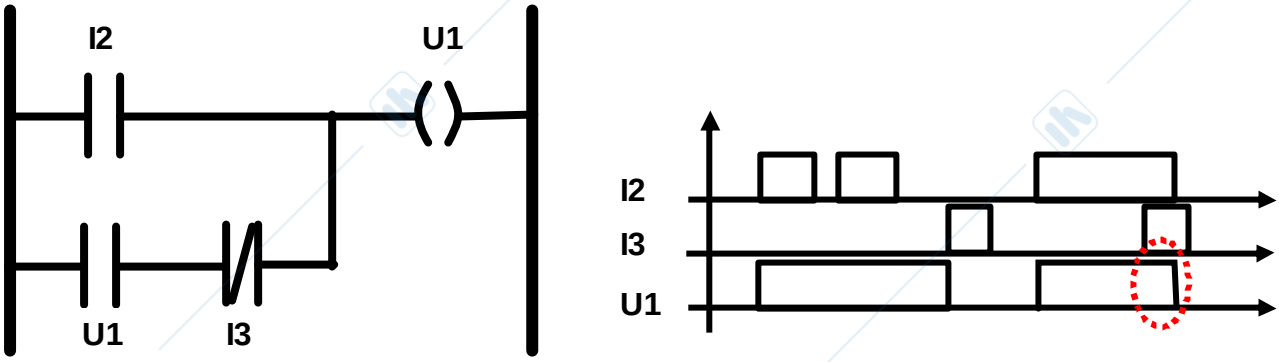




### 4.1.2 Flip-flop a reset vincente

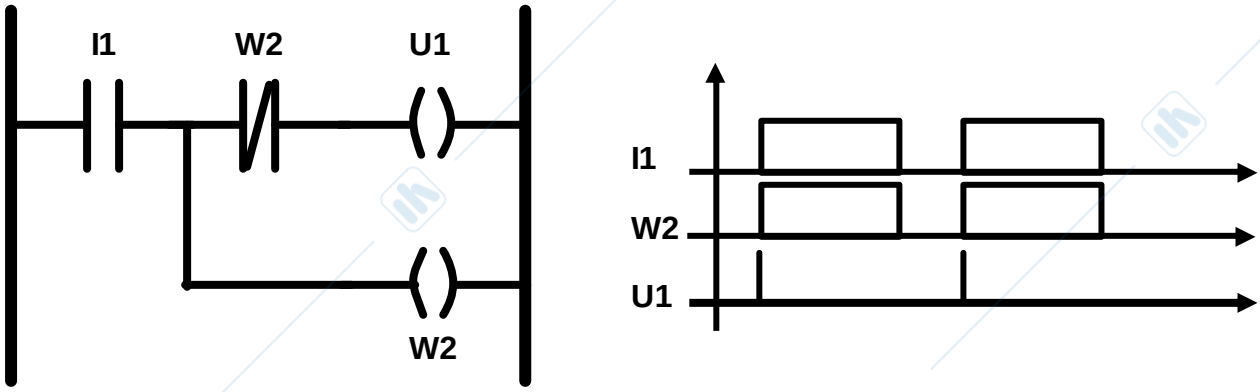


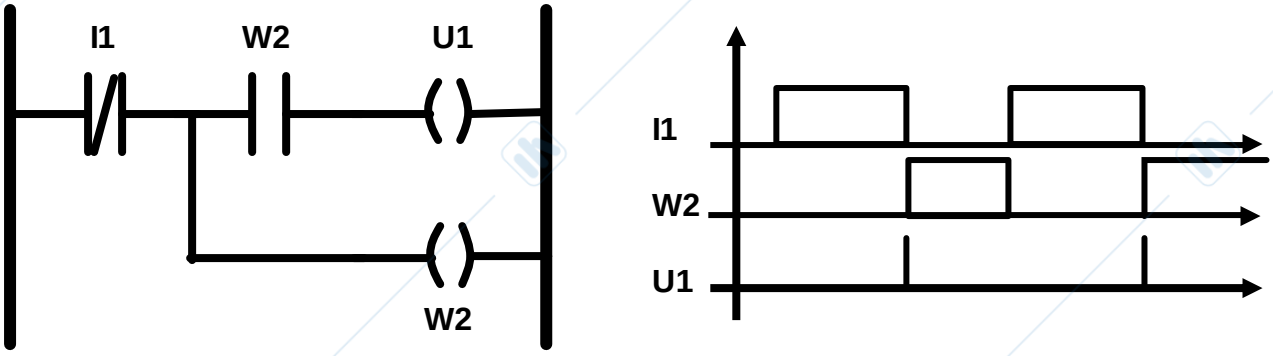
### 4.1.3 Flip-flop a set vincente




### 4.1.4 Riconoscimento di fronte di salita

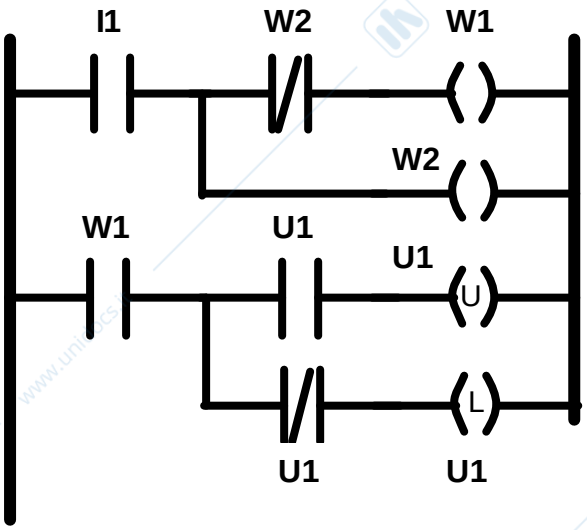


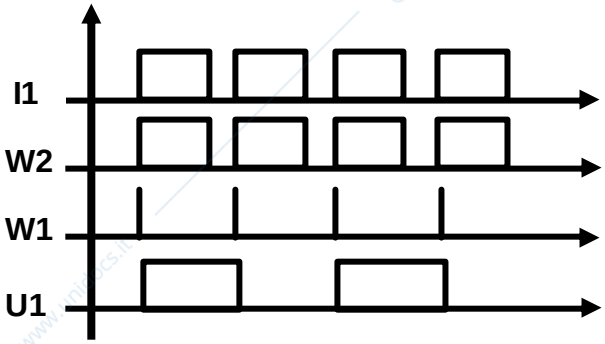

### 4.1.5 Riconoscimento di fronte di discesa



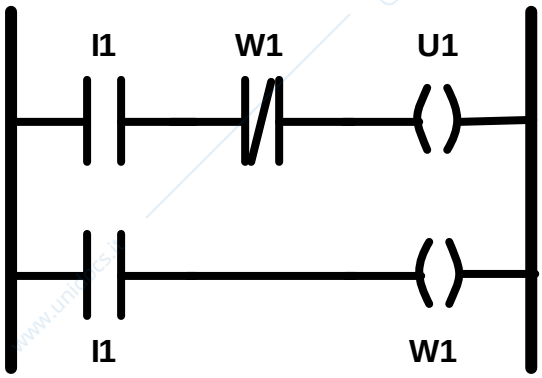
### 4.1.6 Flip-flop di tipo D

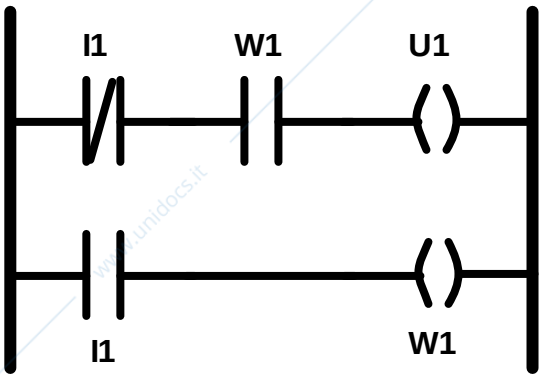
Nel flip flop di tipo D un impulso su **I1** fa commutare l'uscita.

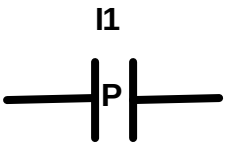


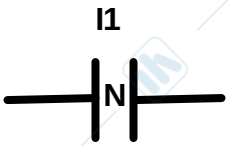

### 4.1.7 Varianti nel riconoscimento di fronti



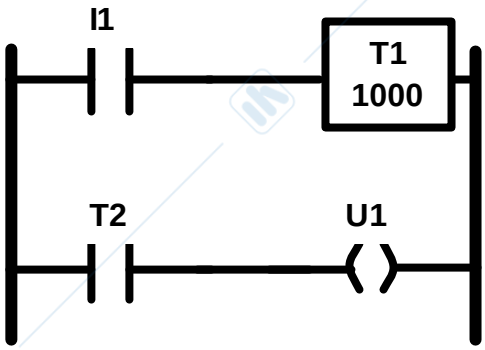



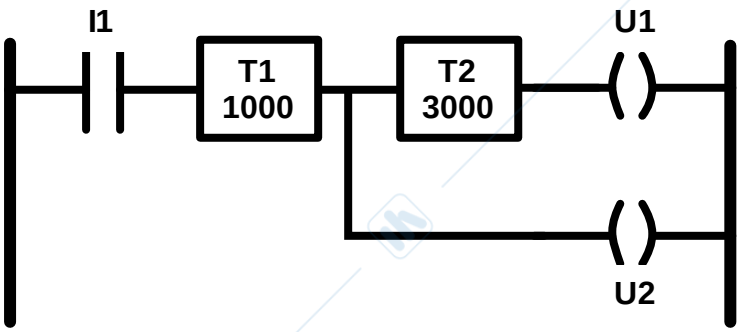

Alcuni ambienti CAD di programmazione di PLC in LD prevedono la presenza di istruzioni avanzate. Ad esempio, un'attributo al contatto identifica che tale contatto si chiude sul fronte di salita (**P**, dall'inglese *Positive edge*) o sul fronte di discesa (**N**, dall'inglese *Negative edge*) della variabile associata.

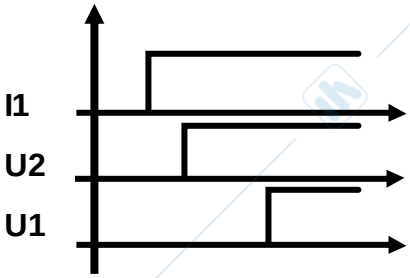


# 5. ISTRUZIONI DI TEMPORIZZAZIONE

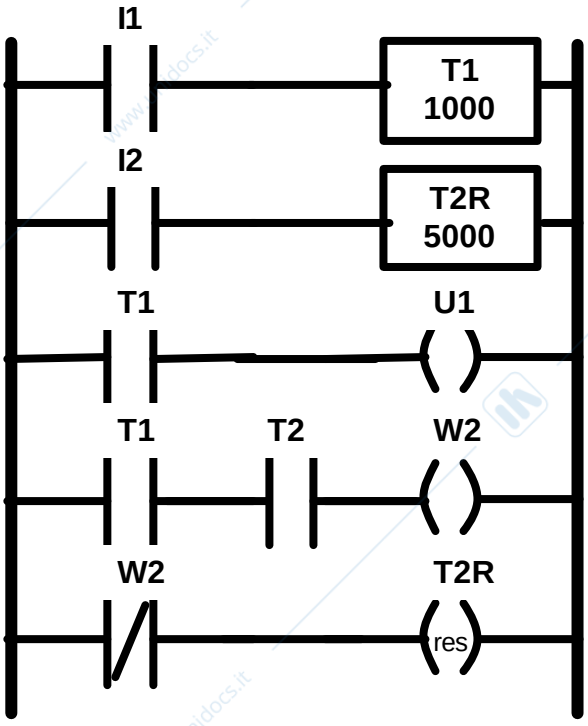



Il temporizzatore (il cui simbolo è "Tx") viene inserito all'interno di un piolo e se quest'ultimo consente il fluire della corrente, il temporizzatore conta il trascorrere del tempo fino ad un valore preimpostato. Una volta raggiunto tale valore "Tx" assume il valore "vero". In Tx.acc è possibile leggere il tempo trascorso. Se il flusso di corrente si interrompe prima del completamento del tempo preimpostato, il temporizzatore si disattiva.
Esistono anche altri tipi di temporizzatori: il temporizzatore a ritenuta, per esempio, (simbolo TxR) continua a contare anche quando il flusso di corrente si interrompe. Il reset temporizzatore (**RES**), invece, interrompe il conteggio e si re-inizializza al valore "0".

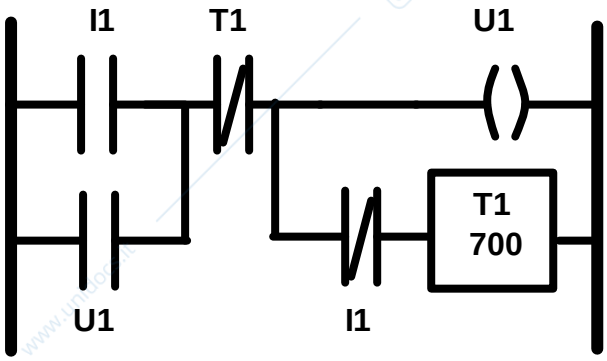


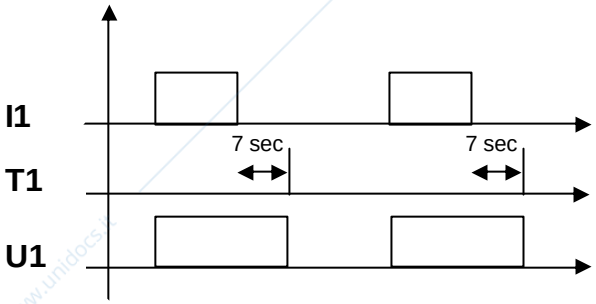




Nella figura a lato, sono rappresentati, dall'alto verso il basso, un temporizzatore **T1** impostato 10 secondi alimentato tramite il contatto **I1**, un temporizzatore a ritenuta **T2**, impostato a 50 secondi alimentato tramite **I2**, l'uscita **U1** alimentata dalla fine del conteggio di T1, il marker **W2** reso vero dalla fine del conteggio di entrambi i temporizzatori e infine il reset del temporizzatore **T2**.

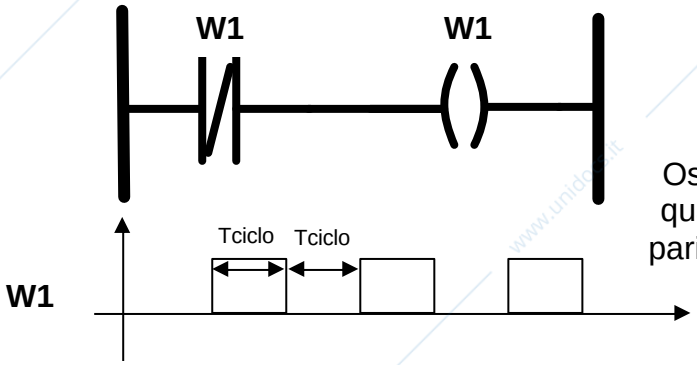


Se si volesse abilitare un'uscita, quando è presente un certo ingresso, e tale abilitazione dovesse durare per 7 secondi dopo la sua fine, otterremmo quello che si chiama "ritardo di spegnimento", realizzato nel modo seguente. Si noti che il temporizzatore **T1** viene alimentato quando l'ingresso **I1** passa allo stato OFF. Dopo aver contato il tempo impostato, esso disalimenta l'uscita **U1** che era stata attivata dall'ingresso e poi memorizzata.

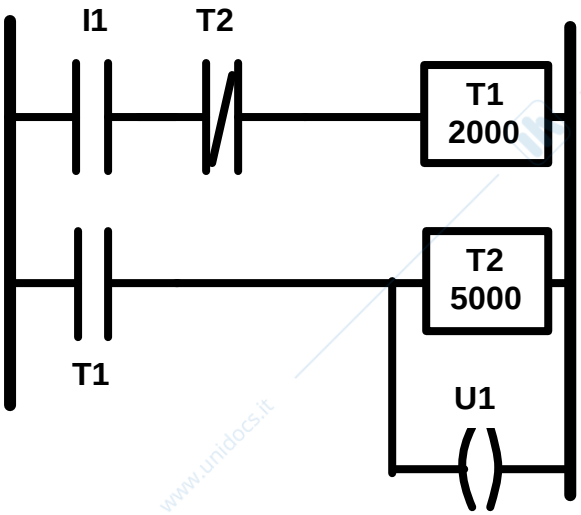


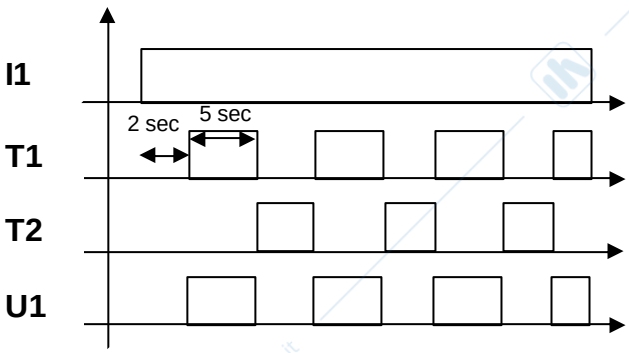




Quando è presente un certo ingresso si vuole azionare un'uscita con un ciclo di on/off di 5 e 2 secondi (oscillatore ad onda quadra), rispettivamente. Nella figura seguente è mostrato un esempio di oscillatore realizzato senza l'ausilio di temporizzatori, mentre invece in quella successiva, il temporizzatore **T1**, alimentato tramite l'ingresso **I1** e lo stato del temporizzatore **T2**, conta il tempo impostato e alla fine alimenta **T2** e l'uscita **U1**. La fine del conteggio di **T2** resetta **T1** e quindi disalimenta l'uscita e resetta anche lo stesso **T2**, facendo ripartire il ciclo che durerà fino a che l'ingresso, **I1** sarà ON.



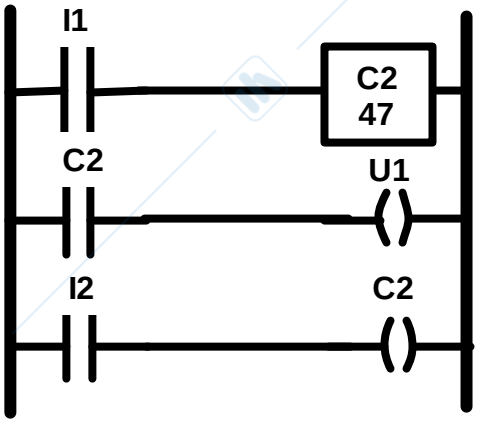

Oscillatore ad onda quadra con periodo pari a 2 tempi di ciclo

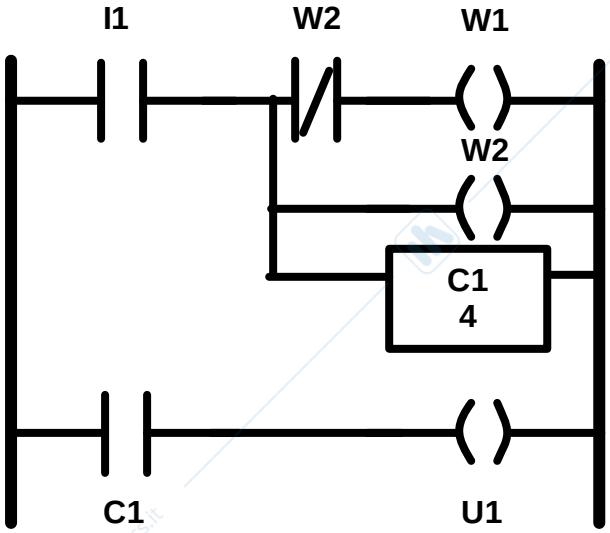


# 6. ISTRUZIONI DI CONTEGGIO

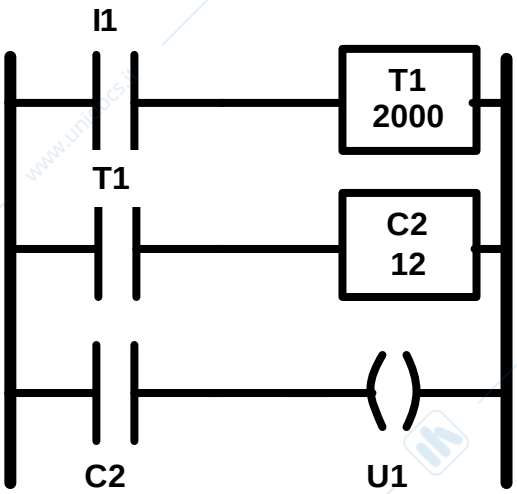
Nel contatore ad incremento, se il piolo di attivazione subisce una transizione da "falso" a "vero" (0 → 1), allora il contatore Cx si incrementa di un'unità. Cx.acc conterrà il valore attuale del contatore, mentre Cx assumerà il valore "vero" nel momento in cui il contatore raggiungerà il valore preimpostato.
Anche per i contatori, il Reset, riporta a zero il contatore Cx → **RES**.
Se si volesse abilitare un'uscita alla quarta occorrenza di un certo ingresso e tenerla alta fino alla successiva occorrenza che è anche la prima di un nuovo ciclo di conteggio, si potrebbe utilizzare la configurazione riportata nella figura sottostante.




I temporizzatori e i contatori, possono essere usati in cascata tra loro per aumentare gli intervalli di conteggio, o sfruttarne le caratteristiche congiunte.

Contatori e temporizzatori



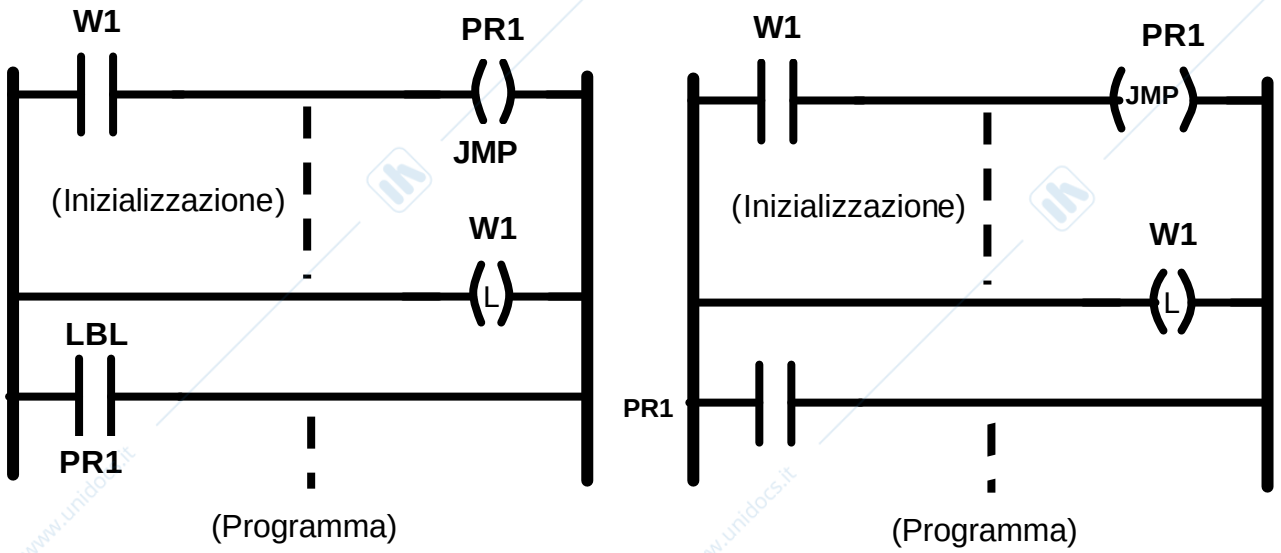
# 7. CONTROLLO DEL PROGRAMMA

I "salti" sono le istruzioni che permettono di modificare il normale ordine di esecuzione di un programma. L'istruzione di "salto" permette, se alimentata, di saltare a un rung dove è presente l'istruzione di etichetta corrispondente. Il simbolo è **–(JMP)–**, associato al numero di un'etichetta; l'etichetta corrispondente associata alo stesso numero, deve essere la prima istruzione a sinistra del suo rung e ha il simbolo **–|LBL|–**.
Se è necessario saltare ad un sottoprogramma, allora si utilizza l'etichetta **–(JSR)–** e il rispettivo inizio del sottoprogramma viene indicato con **–|SBR|–**. Esso poi termina con un ritorno dal sottoprogramma mediante l'etichetta **–(RET)–** che è codificata come un'uscita incondizionata la quale fa ritornare ad eseguire il programma principale dal rung successivo a quello dell'istruzione di salto. Le porzioni di programma delimitate dalla coppia –|SBR|– e –(RET)– non vengono eseguite durante la normale esecuzione del programma, ma solo in seguito ad un salto a sottoprogramma.
L'istruzione "Master Control Relay" permette, invece, di controllare attraverso un solo insieme di condizioni, l'esecuzione di una zona intera di un programma. Viene utilizzato il simbolo **–(MCR)–** e deve essere usata come uscita di un rung all'inizio della zona da controllare e come uscita incondizionata alla fine. Se il rung di abilitazione dell'MCR ha continuità logica, le istruzioni nella zona vengono eseguite, altrimenti i pioli della zona delimitata non vengono eseguiti e le bobine di uscita vengono resettate.
È necessario tenere presente, però, che le istruzioni di salto devono essere utilizzate con cautela; saltare parti di programma in cui vi siano contatori e/o temporizzatori ne può pregiudicare il corretto incremento del valore; se si dimentica, infatti, di inserire la relativa etichetta, si può bloccare l'esecuzione del programma. Saltare invece all'interno di una zona controllata da un Master Control Relay, poi, può indurre comportamenti non previsti.
Di seguito vengono riportati alcuni esempi dei concetti appena esposti.

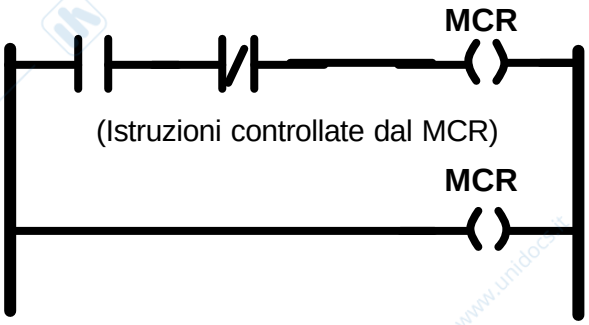
## *7.1 Esempi*

### 7.1.1 Esempi di utilizzo del salto

Nella figure seguenti, è riportato un esempio di utilizzo del salto per realizzare una procedura di inizializzazione che deve essere eseguita solo nel primo ciclo di esecuzione del programma. Se il marker **W1** risulta vero, e lo sarà dal secondo ciclo di esecuzione in poi, il salto viene eseguito.




### 7.1.2 Esempi di utilizzo del Master Control Relay

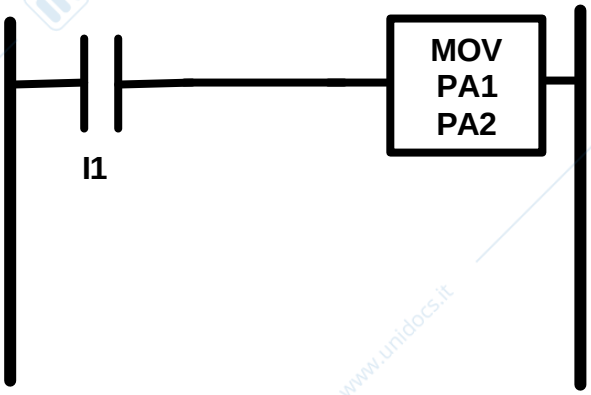



## *7.2 Funzioni e blocchi funzione*

Il LD è infine dotato di notevoli altre estensioni, rispetto ad una pura rete elettrica. Tali estensioni si ispirano alle istruzioni di un comune linguaggio di programmazione di alto livello, come il C, il Pascal, ecc. e quindi possono essere viste come “istruzioni complesse” che sono inglobate graficamente in LD in blocchi che vanno collegati con un piolo.

### 7.2.1 Istruzione MOV

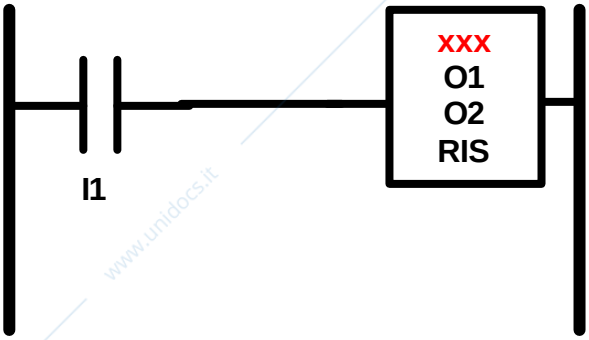
Viene utilizzata per trasferire il contenuto di una word **PA1** in un'altra word **PA2**. è un'istruzione di uscita,e come tale va posta all'estrema destra del rung.




### 7.2.2 Operazioni Aritmetico/Logiche

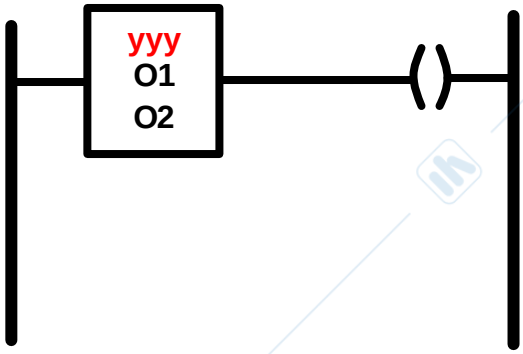
Per eseguire delle operazioni aritmetico/logiche a due operandi **O1** e **O2** si usano blocchi simili, la cui rappresentazione varia solo per il nome dell'operatore inserito a seconda delle necessità. Le operazioni possibili sono le seguenti (tra parentesi il significato):

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




Mentre il risultato viene inserito nell'indirizzo **RIS**. Anche queste istruzioni, come la precedente devono essere programmate come uscite di un rung.

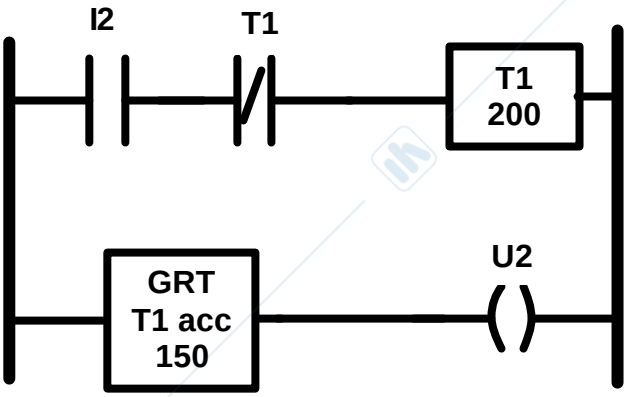
Nella categoria delle operazioni aritmetico/logiche, ci sono anche le istruzioni di comparazione che fanno parte delle condizioni di attivazione dei pioli. Il simbolo corrispondente è rappresentato nell'immagine a fianco dove al posto di "yyy", va inserito uno dei seguenti valori:




- **EQU** (uguale)
- **NEQ** (non uguale)
- **GEQ** (maggiore o uguale)
- **LEQ** (minore o uguale)
- **GRT** (maggiore)
- **LES** (minore)

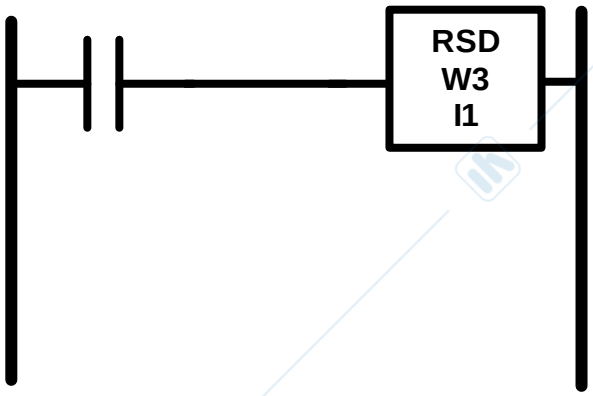
### 7.2.3 Esempio di oscillatore

La figura a lato, presenta una diversa versione dell'oscillatore, già presentato nel capitolo 4, la quale utilizza un solo temporizzatore usando il valore da esso accumulato e una istruzione di comparazione.




### 7.2.4 Registro a scorrimento a destra

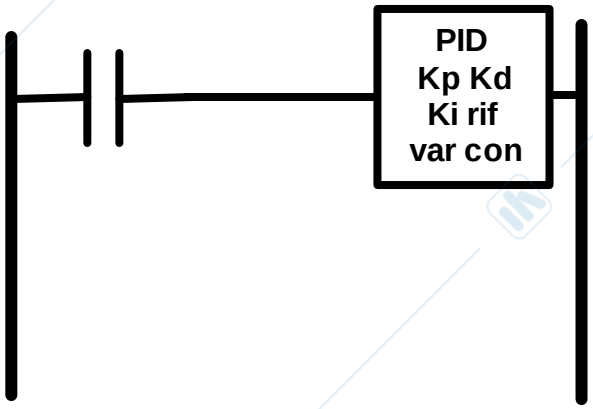
I registri a scorrimento sono particolari istruzioni che si rivelano molto utili nel tener conto del movimento di parti su linee di lavorazione o dello stato di avanzamento di una sequenza di operazioni. L'istruzione "Registro di scorrimento a destra" **RSD** è illustrata nella figura a lato. Nel caso in cui il piolo sia attivato la word W viene spostata (*shift*) a destra di un bit, mentre a sinistra entra il bit indicato come secondo operando. È di particolare importanza la possibilità di utilizzare uno qualsiasi dei bit della word su cui è definito l'RSD, come marker.




### 7.2.5 Istruzione PID

L'istruzione **PID** è una potente istruzione che realizza una funzione di Regolatore Proporzionale Integrale Derivativo, partendo da un ingresso analogico e producendo un'uscita analogica. Si indica con i simbolo della figura a destra dove sono indicati i seguenti parametri.

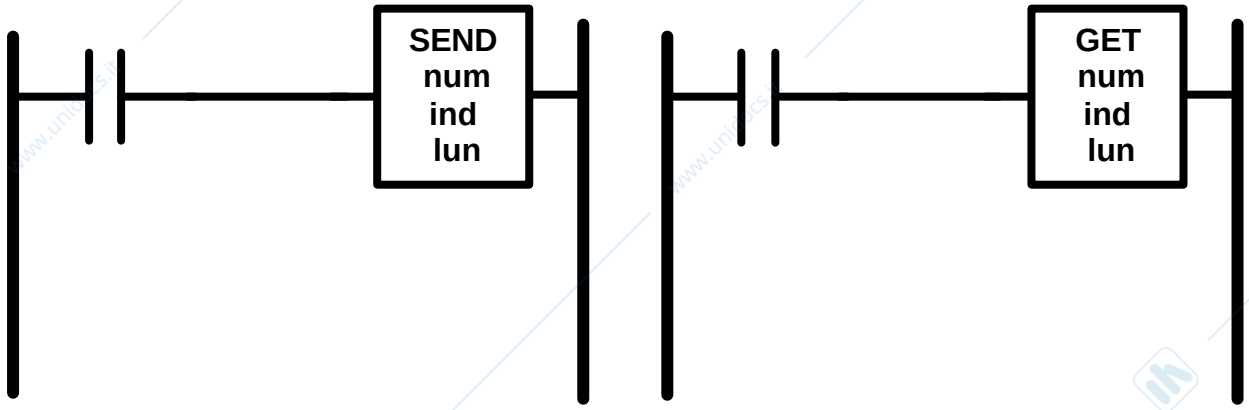
- **Kp** (guadagno proporzionale)
- **Kd** (guadagno derivativo)
- **Ki** (guadagno integrale)
- **rif** (word del riferimento)
- **var** (variabile da controllare)
- **con** (valore del controllo)



### 7.2.6 Istruzioni di comunicazione via rete

Esistono due istruzioni per la comunicazione attraverso la rete, **Send** e **Get**. La prima viene utilizzata per inviare un blocco di word ad un altro PLC connesso in rete, mentre la seconda per ricevere un blocco di word da un altro PLC.
Entrambe le istruzioni necessitano di alcuni parametri:

- **num** (identificativo del PLC con cui si intende comunicare)
- **ind** (indirizzo di partenza del blocco da spedire)
- **lun** (lunghezza del blocco)



# 8. SFC o LD?

## *8.1 SFC*

### 8.1.1 Vantaggi

Il SFC facilita la progettazione del controllo rendendola più agevole in quanto definisce costrutti più astratti e vicini al ragionamento umano, rispetto ad altri linguaggi, quali il passo, la transizione, il parallelismo, la sincronizzazione, ecc. Permette inoltre, una progettazione funzionale delle sole specifiche, cioè il comportamento desiderato del processo. Infine in un progetto sviluppato in SFC è migliore la manutenzione di funzioni già sviluppate.

### 8.1.2 Svantaggi

Nonostante sia stato sviluppato più di 25 anni fa, e nonostante sia presente in normative industriali (IEC) da più di 10 anni, non è supportato da tutti i PLC. Inoltre, soffre di una certa ambiguità per quanto riguarda la rappresentazione del costrutto di parallelismo in taluni casi (OR-divergenza non mutuamente esclusiva, AND-divergenza, azioni parallele in una tappa) e nelle condizioni, che possono comparire strutturalmente nello schema (attraverso le tappe), oppure attraverso le recettività o infine attraverso le azioni condizionate. L'assenza di "conflitti" strutturali e di controlli di chiusura dei rami aperti provoca una difficoltà di analisi formale;

## *8.2 LD*

### 8.2.1 Vantaggi

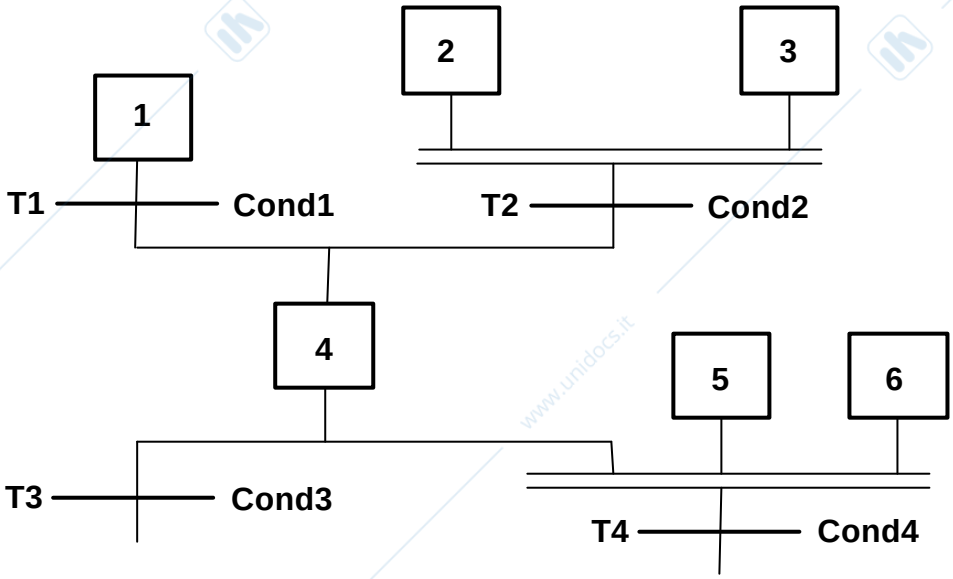
È supportato da tutti i PLC, anche se tale implementazione può variare da sistema a sistema, generando una sorta di "dialetti".

### 8.2.2 Svantaggi

Essendo un linguaggio di livello molto basso è difficile da utilizzare.

## 8.3 Equazioni booleane equivalenti

Osserviamo l'immagine seguente e consideriamo il generico passo (4) di un SFC.




L'equazione di attivazione dello stato è la seguente:

```
X4 = X1 • Cond1 + X2 • X3 • Cond2
```

Mentre l'equazione di disattivazione dello stato è:

```
X4 = X4 • NOT (Cond3 + X5 • X6 • Cond4)
```

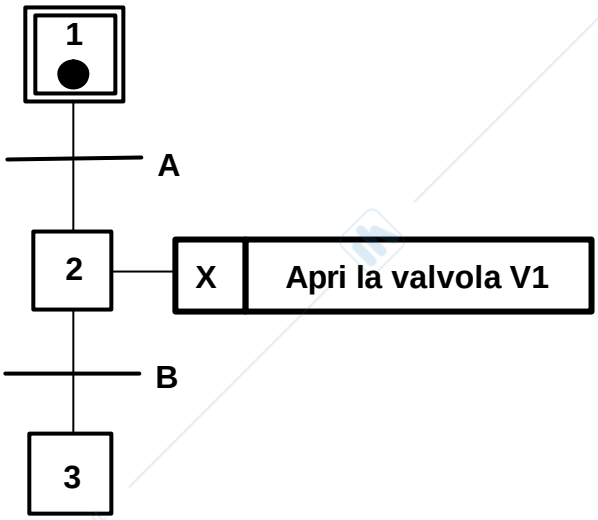
Infine l'equazione di uscita (dipendono dal qualificatore):

```
N:   Azione4 = X4
       ...
```

È facile notare che tali equazioni sono poco leggibili e "mantenibili" e mancano comunque le regole di interpretazione.

## 8.4 Interpretazione di un SFC

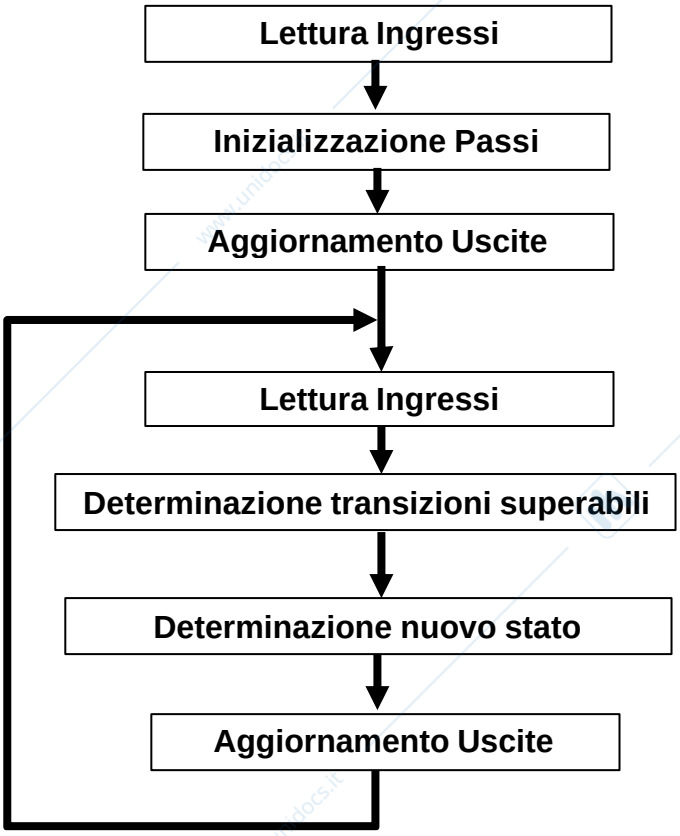
È possibile esprimere in modo formale l'interpretazione dell'evoluzione dinamica di uno schema SFC su una macchina reale come un PLC. Così facendo, si ottiene il non banale vantaggio di chiarire ed esplicitare possibili casi dubbi sull'evoluzione di uno schema.
La rappresentazione formale delle regole di evoluzione (o interpretazione) di uno schema SFC si chiama algoritmo di evoluzione (o di interpretazione), di cui in letteratura esistono molte varianti possibili, proprio perché uno schema può essere interpretato in modi leggermente diversi.




Ad esempio, nell'immagine a lato, se le condizioni **A** e **B** fossero contemporaneamente verificate, si attiverebbe il passo 2 e "immediatamente" dopo il passo 3. Purtroppo, però, con i dispositivi reali non è possibile ottenere delle attivazioni immediate, quindi l'uso di tale avverbio risulta improprio se applicato ai casi reali e quindi di volta in volta è necessario aver presente i gradi di tolleranza ai ritardi per progettare correttamente un dispositivo che risponda in maniera efficiente alle specifiche.
In riferimento all'esempio in figura, le considerazioni possibili sono molteplici; in primo luogo possiamo chiederci, cosa succederebbe alla variabile di uscita **Apri la valvola V1** se le condizioni **A** e **B** fossero contemporaneamente verificate, oppure se il qualificatore (**X** in figura) fosse invece "N", oppure ancora se quest'ultimo fosse "S".

## *8.5 Algoritmi di Evoluzione*

### 8.5.1 Algoritmi di Evoluzione <u>senza</u> ricerca di stabilità



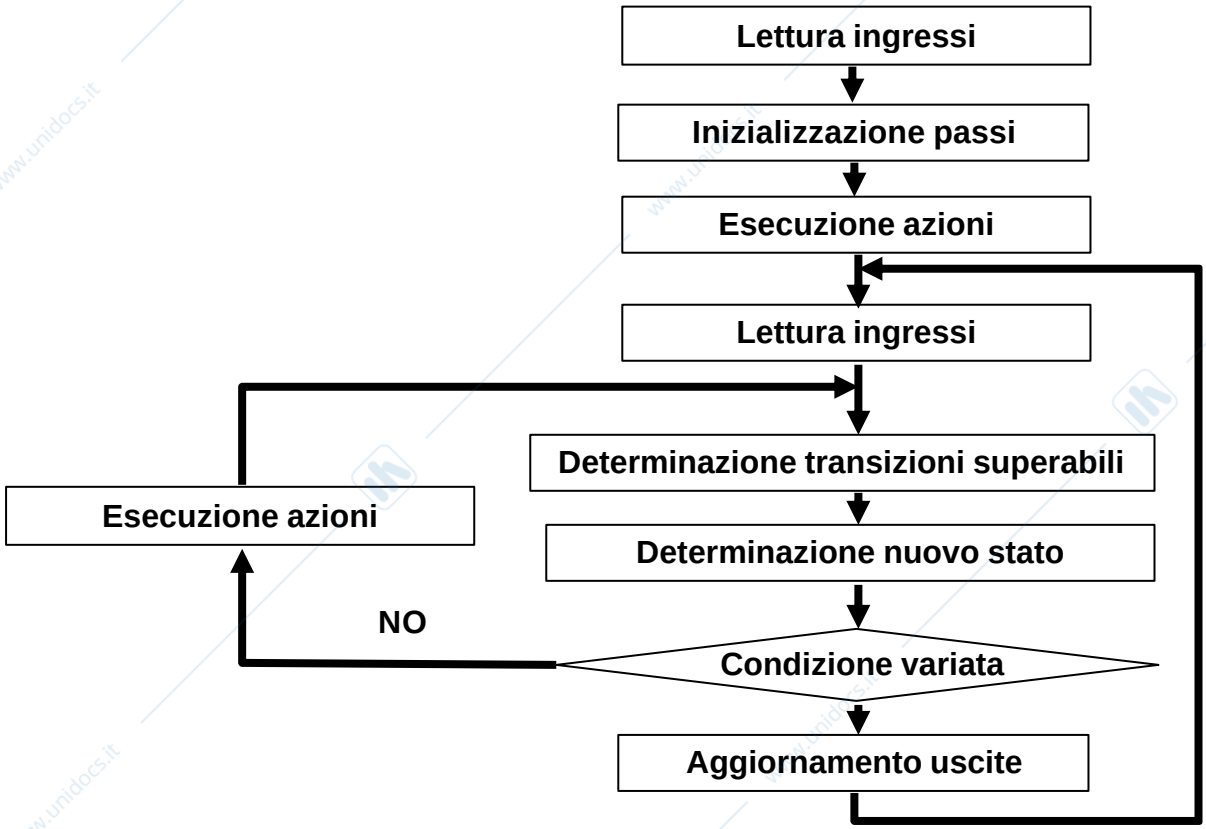

In questo caso, si realizza una rappresentazione in forma algoritmica delle regole di evoluzione tali che in presenza di sequenze di transizioni con condizione logica vera, queste vengano superate in cicli diversi, e non nello stesso ciclo. Quindi tutte le uscite associate ai passi intermedi verranno assegnate (anche se il gettone di attivazione rimarrà nel passo un tempo piccolo a piacere).
Si eseguono quattro passi ad ogni ciclo:

- Lettura Ingressi
- Transizioni superabili
- Determinazione nuovo stato
- Aggiornamento uscite

Tale algoritmo è detto "senza ricerca di stabilità", in quanto nel caso di più transizioni successive con condizione associata vera, eseguirebbe le azioni associate a tutte le fasi intermedie.

### 8.5.2 Algoritmi di Evoluzione con ricerca di stabilità

Un algoritmo che si avvicina di più al comportamento dettato dalle regole di evoluzione è quello "con ricerca di stabilità" presentato nella figura sottostante.
In tale algoritmo le uscite vengono aggiornate solo se la condizione dell'SFC ha raggiunto la stabilità, cioè quando essa, sotto l'azione degli stessi ingressi, non varia più. È da notare, comunque, che nelle fasi "instabili" deve essere garantita l'esecuzione delle azioni impulsive che sono quelle così definite dal programmatore o quelle che devono essere così considerate.



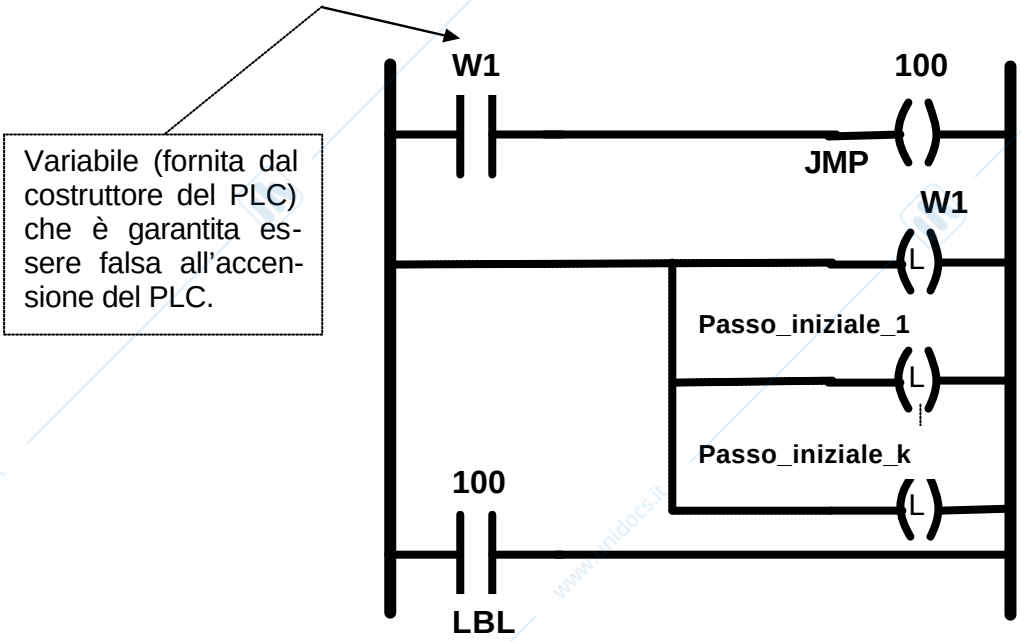
# 9. TRADUZIONE IN LADDER DIAGRAM

L'utilizzo del Sequential Functional Chart permette la progettazione degli algoritmi di controllo in una maniera più agevole rispetto, per esempio, al linguaggio a contatti. È infatti naturale individuare le fasi e le transizioni che costituiscono il comportamento desiderato del processo da automatizzare e quindi l'algoritmo di controllo. Il problema è che non tutti i controllori a logica programmabile prevedono l'SFC come linguaggio di programmazione, e anche quelli che lo prevedono propongono in realtà dei linguaggi simili che impongono notevoli limitazioni allo sviluppo del programma.
Affrontiamo quindi ora il problema di determinare un'implementazione in Ladder Diagram di uno schema SFC secondo l'algoritmo di evoluzione senza la ricerca di stabilità, in modo che esso sia effettivamente implementabile su una qualsiasi macchina per l'eleborazione di informazioni, come, per esempio un PLC. La scelta del Ladder Diagram per questo scopo, è giustificata dalla sua diffusione, ma i concetti qui esposti hanno anche una valenza più generale e nulla vieta di realizzarli mediante l'uso di altri linguaggi. Quello che conta veramente è che tale procedura di traduzione sia il più dettagliata possibile, in modo che si possa ottenere un programma eseguibile a partire da un SFC senza ulteriori sforzi progettuali.
La traduzione ottenuta dovrebbe, infine, presentare delle relazioni biunivoche tra le istruzioni che la compongono e le fasi e le transizioni dell'SFC di partenza, così da conservare le proprietà di autodocumentazione di quest'ultimo. Quello che si vorrebbe ottenere è che, se risultasse necessario modificare l'algoritmo, tale modifica dovrebbe essere operata sull'SFC progettato, dove è più semplice agire, ma dovrebbe anche poter essere direttamente convertita nelle modifiche da effettuare alla traduzione ottenuta.
L'algoritmo si può sintetizzare nel modo seguente: ad ogni passo si associa un bit di memoria che rappresenta lo stato del passo, mentre ad ogni transizione si associa un altro bit di memoria che rappresenta la condizione che una transizione sia superabile in un certo stato. Si hanno poi quattro sezioni che sono:

- Sezione di "inizializzazione"
- Sezione di "esecuzione delle azioni"
- Sezione di "valutazione delle transizioni"
- Sezione di "aggiornamento dello stato"

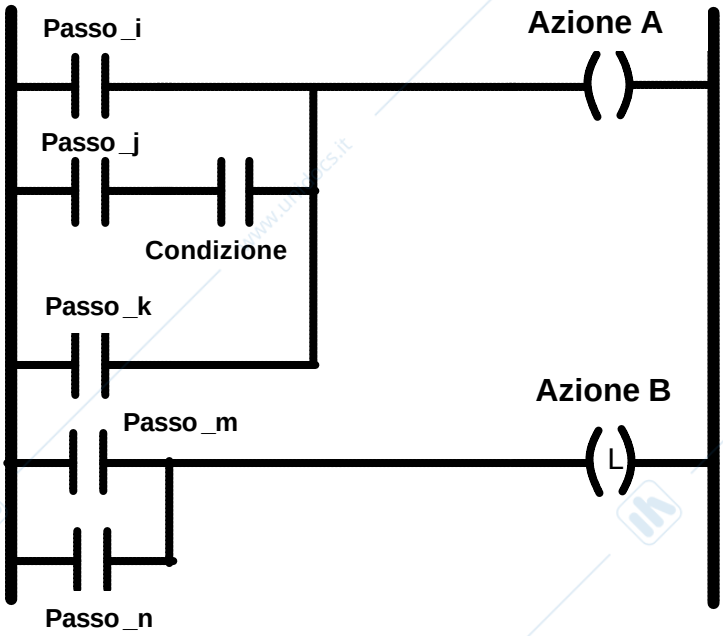
## *9.1 Inizializzazione*

Viene eseguita un'unica volta all'inizio del programma ed ha il compito di inizializzare al valore "1" gli stati dei passi iniziali previste dall'SFC. Ciò può essere realizzato utilizzando delle bobine a ritenzione associate ai marker delle fasi iniziali. Nella figura della pagina seguente è rappresentata in maniera schematica una possibile realizzazione di tale sezione utilizzando l'istruzione di salto associata al marker **W1** per ottenere che le operazioni di inizializzazione vengano eseguite una sola volta.



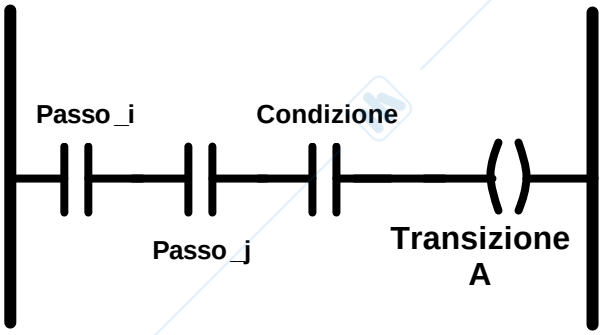

## 9.2 *Esecuzione delle azioni*

Tale sezione è dedicata alla esecuzione delle azioni associate alle fasi attive, o meglio, all'aggiornamento dell'area di memoria riservata alle uscite che, alla fine del ciclo di scansione, viene effettivamente riportata sulle uscite fisiche.
Per ogni azione di tipo N si prevede un piolo di abilitazione su cui si trovano in OR tutti i passi che prevedono quella azione; eventualmente, se un passo contiene un'azione condizionata da una condizione **C1**, lo stato di quel passo va in AND con **C1**. Per le azioni memorizzate si usano bobine di tipo Latch o Unlatch, mentre, per le azioni impulsive si dovrà codificare in LD l'opportuno rivelatore di fronte.



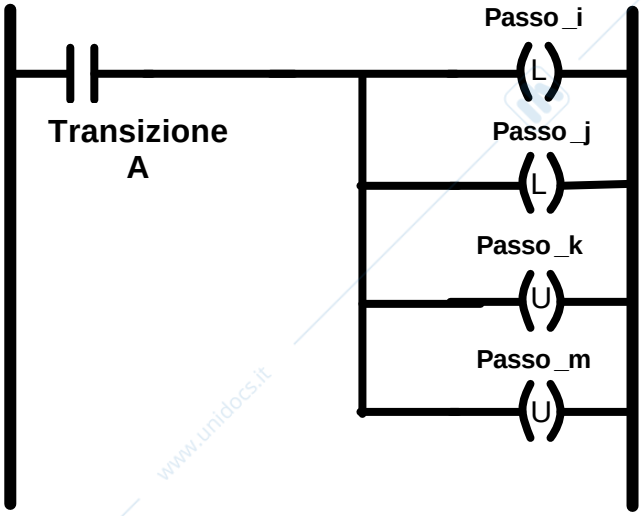
## 9.3 Valutazione delle transizioni

A questo punto è necessario valutare se una transizione è superabile o meno, ed aggiornare di conseguenza, lo stato del relativo marker. Ricordiamo che una transizione è superabile se le fasi a monte sono tutte attive e se la condizione associata risulta vera.
Il bit associato ad ogni transizione è attivato se la transizione è superabile ed inoltre, si ha un piolo per ogni transizione.



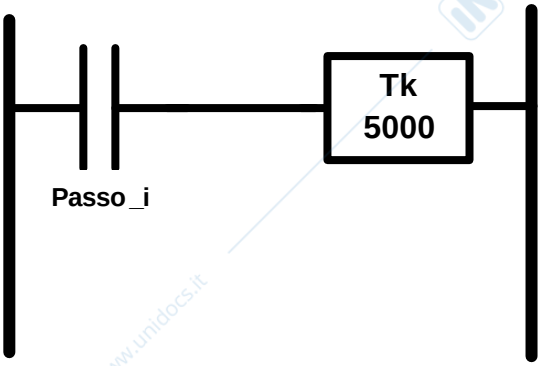

## 9.4 Aggiornamento dello stato

Il compito di quest'ultima fase, consiste nell'attivare, in corrispondenza delle transizioni valutate come superabili, le fasi a valle, e disattivare quelle a monte. Sarà composta quindi da un rung per ogni transizione in cui, se il marker di transizione ha valore "1", si attivano le fasi a valle e si disattivano le fasi a monte usando delle bobine a ritenuta (latch) associate ai marker delle fasi.



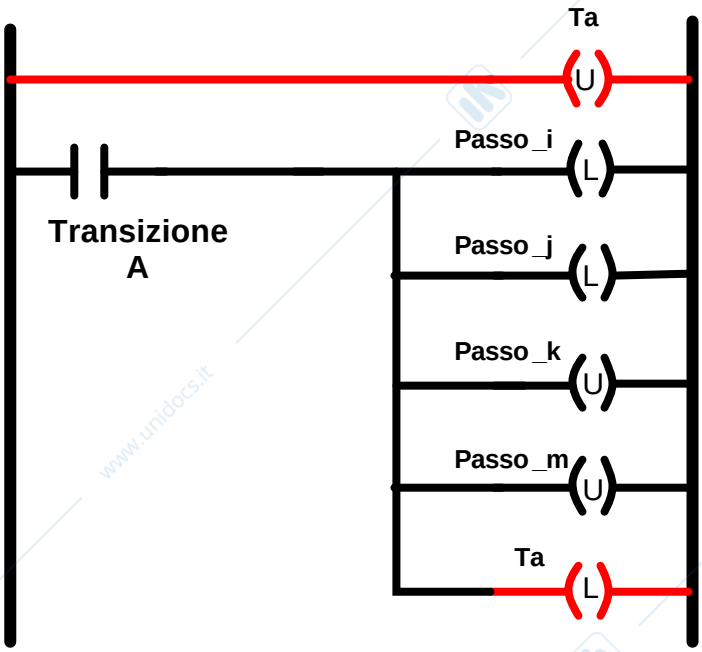

## 9.5 Traduzione delle variabili temporali

Le variabili temporali devono essere realizzate con dei temporizzatori senza ritenzione attivati dai bit di stato dei passi. Questi temporizzatori possono essere messi in una sezione a parte.



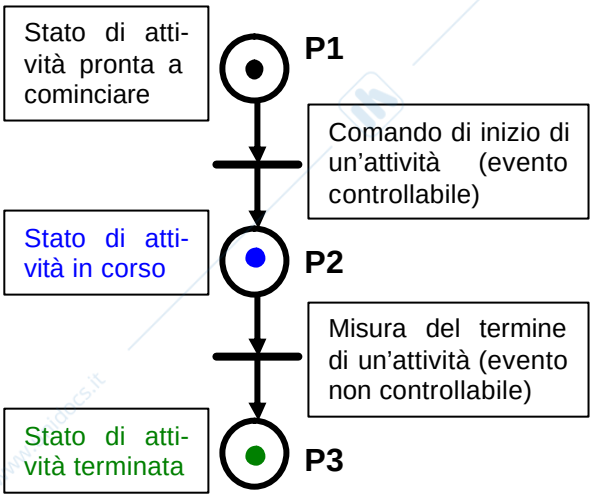
## 9.6 Algoritmo con ricerca di stabilità: Aggiornamento condizione

Si associa al bit “Ta” il significato di “almeno una transizione è stata abilitata durante la valutazione delle condizioni”. Questo bit può poi essere attivato durante la sezione di aggiornamento della condizione.



# 10. RETI DI PETRI NON AUTONOME

Le RP possono essere estese, per poter rappresentare controllori, con ingressi e uscite. Se le RP sono estese associando: segnali di comando ai posti, e condizioni logiche alle transizioni, allora i metodi di traduzione sono simili al caso del SFC, ma se invece le RP sono estese in modo che rappresentino un modello ad eventi in senso stretto, *sia gli ingressi sia le uscite saranno eventi* ed entrambi saranno associati alle *transizioni.*




Ad esempio, il metodo di modellizzazione delle attività a due eventi, illustrato a lato, si presta molto bene ad essere utilizzato con tale modello. I diversi colori indicano l'evoluzione della rete. Che inizia con un gettone nel posto **P1** che successivamente passerà al posto **P2** ed infine al posto **P3**. Le caselle di testo che descrivono il significato dei posti utilizzano gli stessi colori dei gettoni a cui si riferiscono; quando il gettone si trova nel posto **P2**, ad esempio, la relativa casella di testo indica che in quello stato l'"attività è in corso".

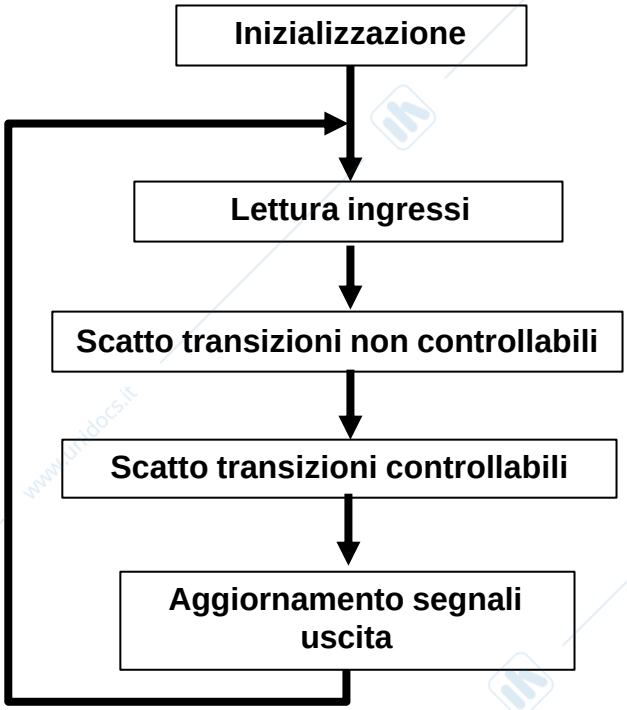
## *10.1 Interpretazione di una RP*

Rispetto al SFC, le RP sono più astratte e quindi è meno immediata la traduzione in linguaggi di basso livello come il LD. In particolare, le regole di evoluzione delle RP sono non deterministiche, cioè se più transizioni sono abilitate, ne scatta una a caso. Nascono però alcuni problemi, e tra questi per esempio vi è quello suggerito da questa domanda: "Quando scatta una transizione abilitata?" Il tempo, infatti, non è modellizzato nelle nostre RP, però un controllore deve reagire entro limiti di tempo prefissati. Inoltre, se abbiamo *n* transizioni abilitate, non ha nemmeno senso che scatti sempre la medesima transizione.
Naturalmente, la traduzione deve essere compatibile con una delle possibili evoluzioni della rete ed evitare di "tralasciare" lo scatto di qualche transizione, quindi per arrivare ad un algoritmo di evoluzione di una RP, dobbiamo ricordare che stiamo usando la RP come controllore, di conseguenza essa deve accettare le misure dall'impianto controllato (cioè quando l'impianto "genera" un evento, il controllore deve essere pronto ad accettarlo), e anche emettere i comandi "prima possibile" (se dal modello risulta che un comando può essere emesso, non ci sono motivi, infatti, per dilazionarlo).

## 10.2 Algoritmo di Evoluzione

L'idea di base consiste nel definire un controllore che prima accetta informazioni dall'impianto, e poi, sulla base del nuovo stato, prende decisioni "aggiornate".
L'algoritmo di evoluzione è fondamentalmente diviso in due porzioni: la prima è relativa all'accettazione degli eventi non controllabili (misure), mentre la seconda è relativa alla produzione di eventi controllabili (comandi).
Si eseguono quattro passi ad ogni ciclo:

- Lettura Ingressi
- Determinazione e scatto delle transizioni non controllabili
- Determinazione e scatto delle transizioni controllabili
- Aggiornamento segnali di uscita

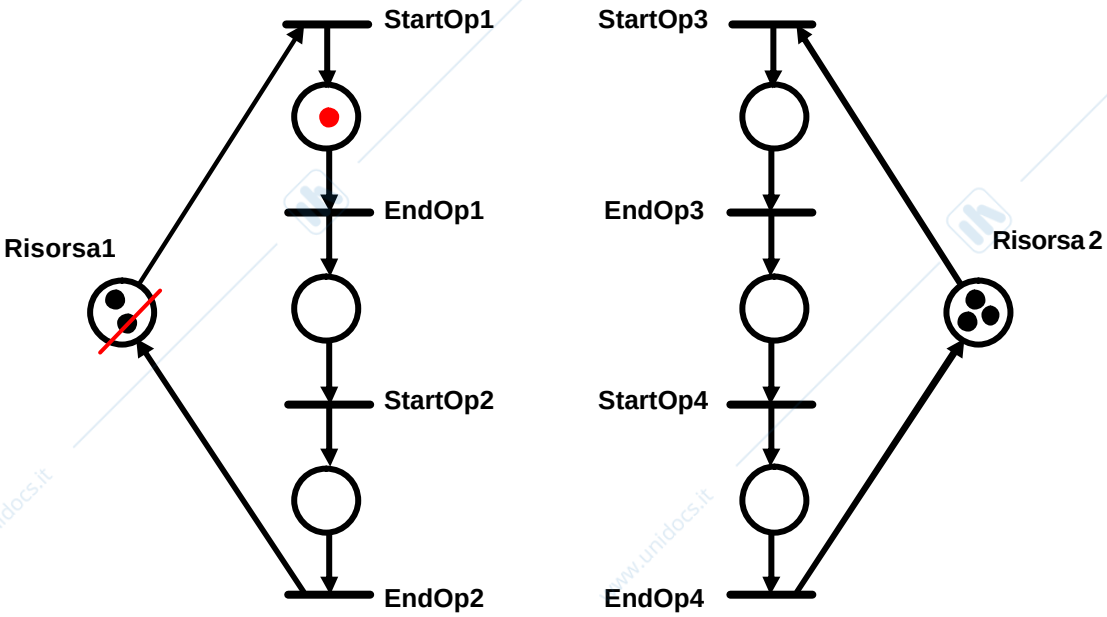


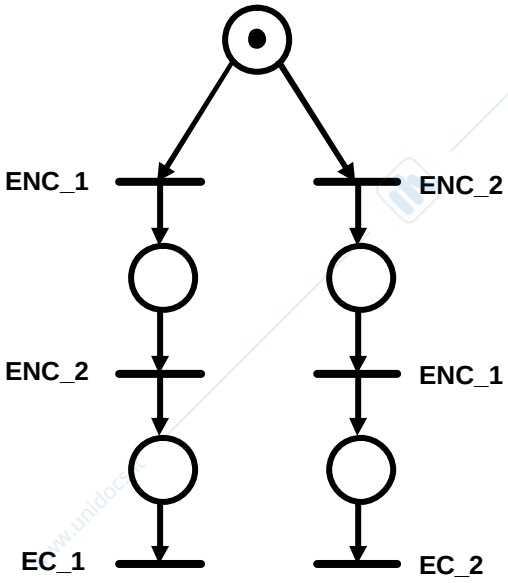

### 10.2.1 Problemi

A questo punto però, nascono alcuni problemi. Se abbiamo più transizioni abilitate nelle singole porzioni non controllabili e controllabili, qual è l'ordine da scegliere per il loro scatto? Una soluzione potrebbe essere quella di scegliere in base a dei criteri che ottimizzano qualche cifra di merito; ciò è in effetti possibile, ma è poco conveniente per fare del controllo logico. Faremo quindi l'ipotesi che non esista un ordine "migliore" di un altro.

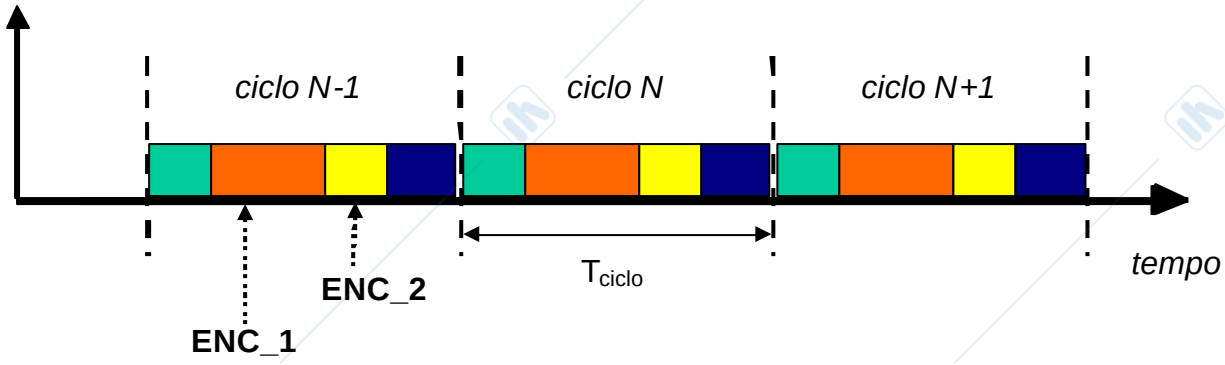
### 10.2.2 Ordine di scatto

Per ipotesi, lo stato raggiunto da un RP è indipendente dall'ordine con cui essa esegue una sequenza di eventi non controllabili.
Osserviamo l'immagine della pagina seguente dove in rosso è stato rappresentato lo spostamento dei gettoni nell'evoluzione della prima rete.




La rete a fianco, invece, è un esempio di rete che non soddisfa l'ipotesi fatta poco sopra.
**ENC_1** e **ENC_2** rappresentano gli eventi non controllabili, mentre **EC_1** ed **EC_2**, rappresentano quelli controllabili.
**ENC_1** e **ENC_2** possono sicuramente essere letti come "contemporanei", data la natura del PLC (segnali campionati). Il progettista deve quindi chiedersi quale azione vuole che il controllore compia qualora due eventi il cui ordine di accadimento è importante vengono rilevati "contemporaneamente".

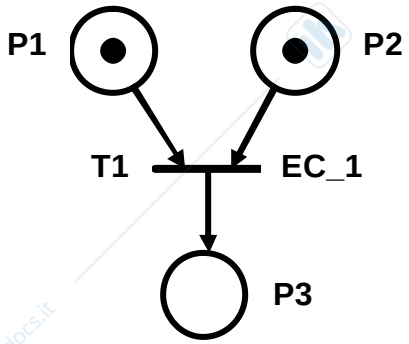


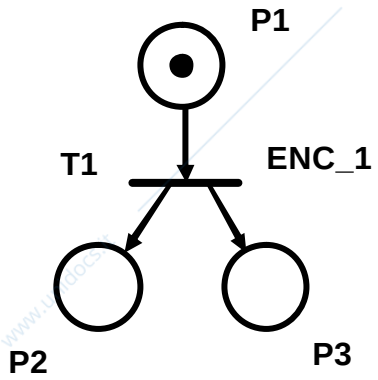
# 11. TRADUZIONE IN LADDER DIAGRAM

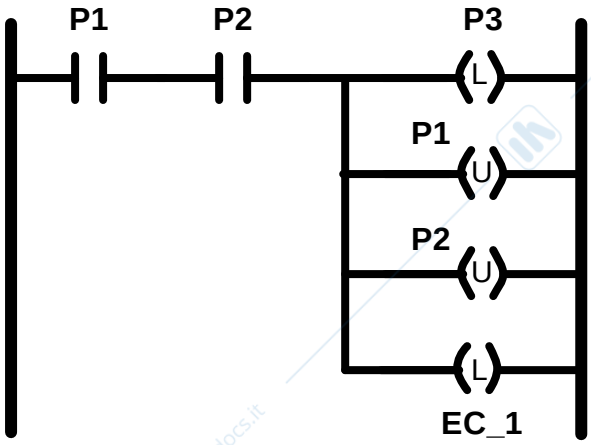
Affrontiamo ora il problema di determinare un'implementazione corretta in Ladder Diagram di una rete di Petri in cui sia gli ingressi sia le uscite siano eventi associati alle transizioni.
Come è già stato anticipato precedentemente per la traduzione dell'SFC, i concetti qui esposti possono essere realizzati anche in altri linguaggi, la scelta del Ladder Diagram è giustificata principalmente dalla sua diffusione.
Per semplicità, supporremo che la rete sia binaria, e procederemo nel seguente modo:

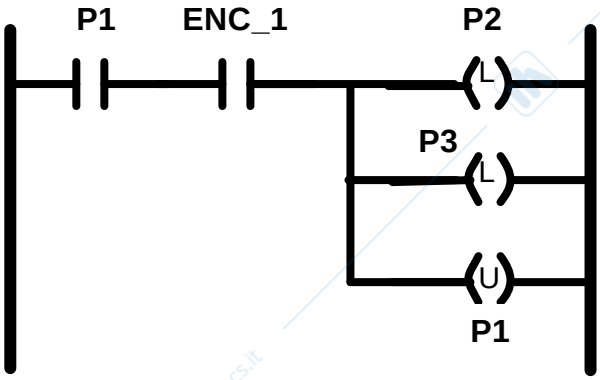
- Ad ogni posto assoceremo un bit di memoria che ne rappresenta lo stato
- Ad ogni transizione, invece, assoceremo un'istruzione (cioè un piolo) che ne rappresenta lo scatto
- Gli eventi controllabili saranno associati a bobine (impulsi pari a un tempo di ciclo)
- Gli eventi non controllabili saranno associati a contatti (impulsi pari a un tempo di ciclo)

## *11.1 Traduzione di una transizione*



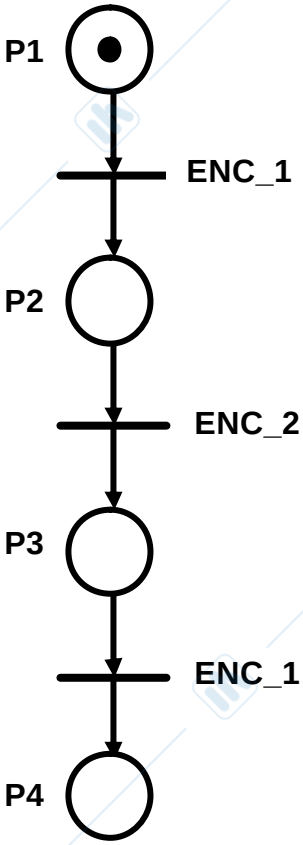



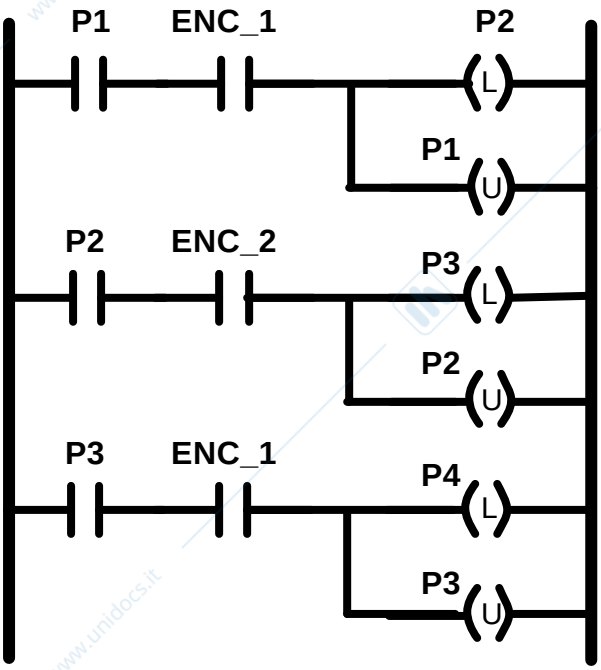



## 11.2 Effetto "valanga"

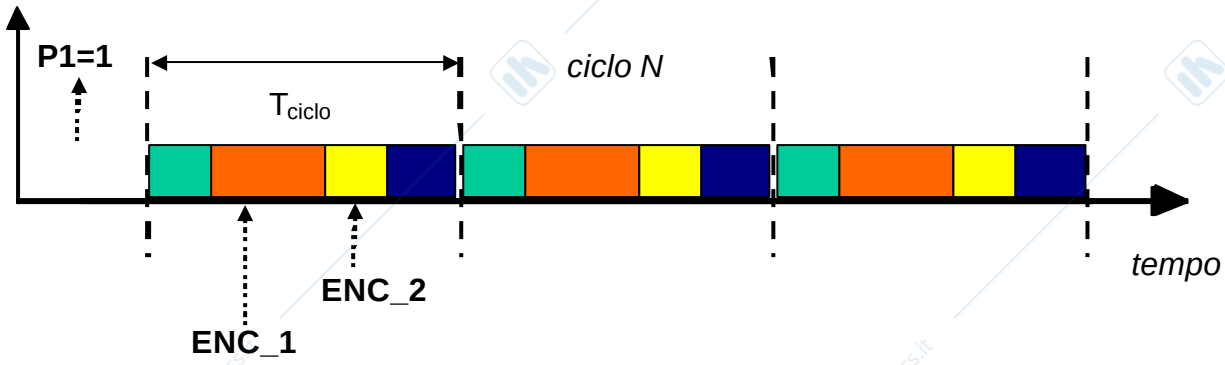
Supponiamo di avere una RP contenente una sequenza di transizioni non controllabili tra cui alcune associate allo stesso evento **ENC_1**. L'effetto "valanga" è l'effetto per cui, a causa di una non corretta implementazione del concetto di evento, al verificarsi dell'evento **ENC_1** il programma LD che implementa tale RP evolve come se fossero accaduti più eventi **ENC_1** contemporaneamente.
Osservando la situazione rappresentata nell'immagine a destra, notiamo che per marcare **P4** da **P1** devono accadere gli eventi **ENC_1**, **ENC_2** e poi ancora **ENC_1** (esattamente in questo ordine). Pertanto, da **P1**=1, se si verificano **ENC_1** e **ENC_2**, raggiungiamo **P3**=1.



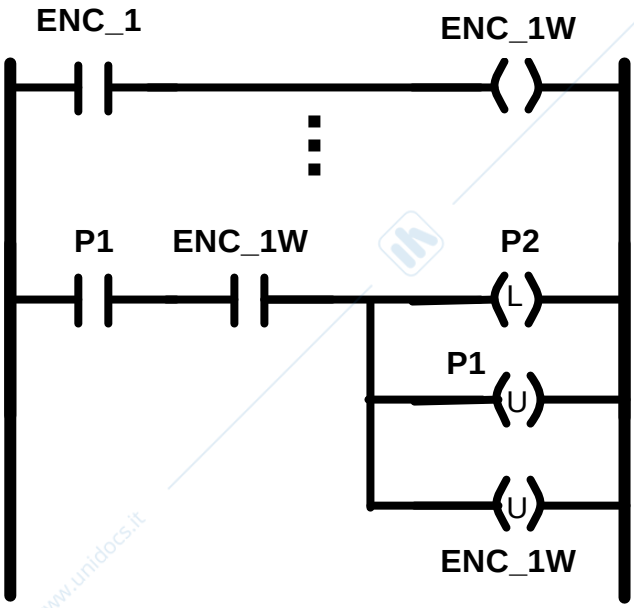




A sinistra è rappresentato lo schema in Ladder Diagram della rete di Petri.
Mentre, nel disegno sottostante, vediamo l'evoluzione nel tempo.
Considerando il primo piolo, abbiamo che, inizialmente, **P1**=1 e **ENC_1**=1. Successivamente, **P1** diventa 0 e **P2**=1. Passando a considerare il secondo piolo, la situazione di partenza è quella generata dal primo piolo, e quindi **P2**=1 e **ENC_2**=1. In seguito, **P2** assumerà il valore 0, mentre **P3**=1. L'ultimo piolo, avrà quindi **P3**=1 e **ENC_1**=1, mentre al termine **P3** passerà al valore 0 e **P4**=1.

Quindi, al termine del ciclo N, abbiamo "consumato" **ENC_1** due volte.



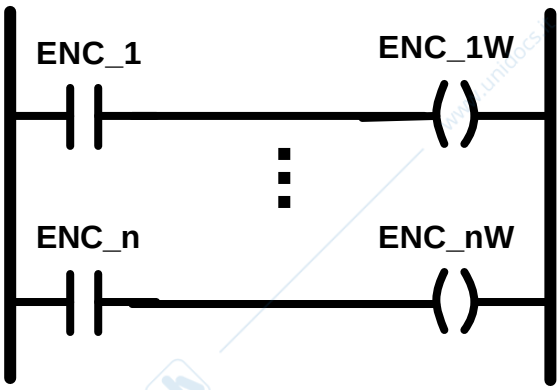
Il rimedio di cambiare l'ordine dei pioli è in se corretto, in questo caso, ma non è detto che sia sempre possibile, in generale, trovare un ordine che risolva tutti i problemi di ordinamento. Come rimedio generale, è possibile, invece, copiare l'ingresso associato ad un probabile effetto valanga in una variabile temporanea, da usare in luogo dell'ingresso vero, così da poterla "resettare" subito dopo averla consumata (così che non possa essere consumata in altri pioli).



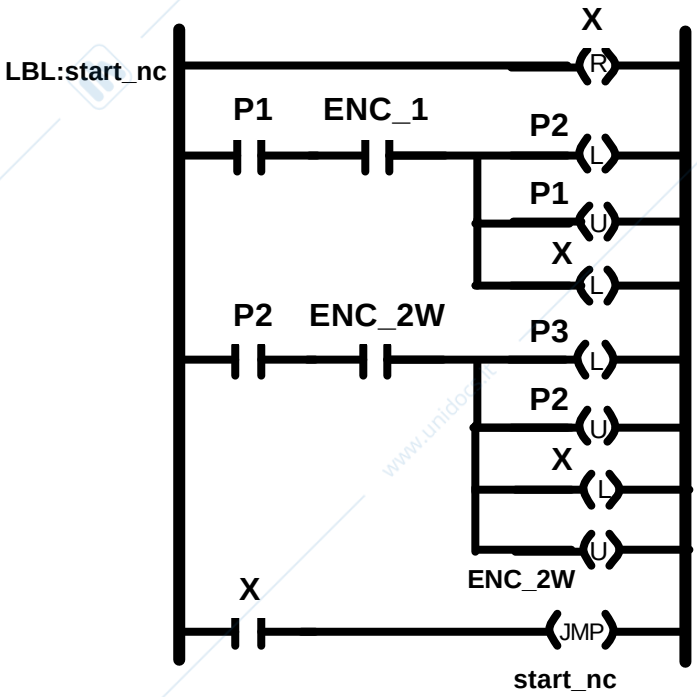

## *11.3 Algoritmo di implementazione*

In base a quanto precedentemente spiegato, l'algoritmo di implementazione delle RP è costituito dalle seguenti porzioni consecutive di LD.

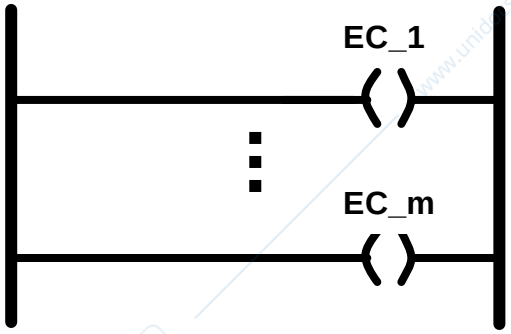
### 11.3.1 Creazione delle copie degli eventi soggetti all'effetto valanga



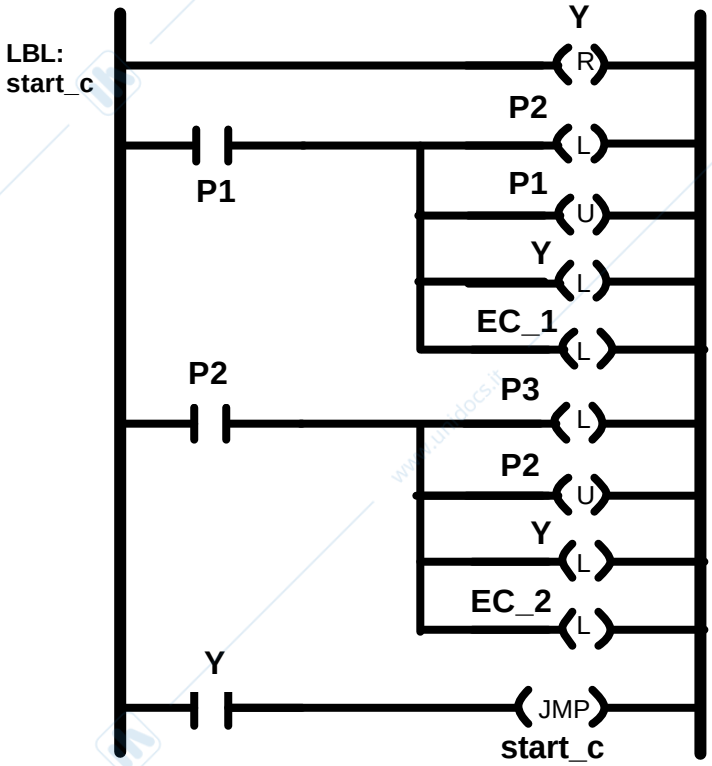
### 11.3.2 Accettazione degli eventi non controllabili




### 11.3.3 Reset delle variabili di comando



### 11.3.4 Generazione degli eventi controllabili




### 11.3.5 Alcune note sull'algoritmo di implementazione

In questo modo abbiamo generato impulsi di durata pari a un tempo di ciclo per gli eventi controllabili; occorrerà quindi aggiungere un'ultima porzione di LD per trasformare tali impulsi in segnali adatti a pilotare gli attuatori. Gli eventi non controllabili saranno poi generati dalle istruzioni LD che rilevano fronti di salita o discesa sui segnali di ingresso.
È da tenere presente, che l'effetto valanga non si presenta se tutti gli eventi non controllabili sono distinti, e nemmeno se tutti gli eventi non controllabili sono seguiti da eventi controllabili. Se la rete non è binaria, inoltre, occorre implementare la marcatura di un posto con dei contatori, e le condizioni di scatto con dei comparatori (marcatura ≥ peso arco). Infine le transizioni temporizzate si implementano con dei semplici timer.

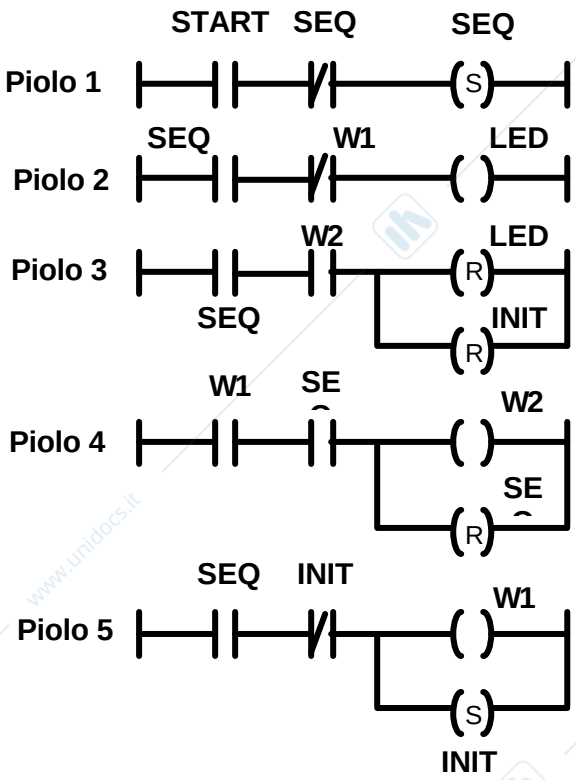
# 12. APPENDICE

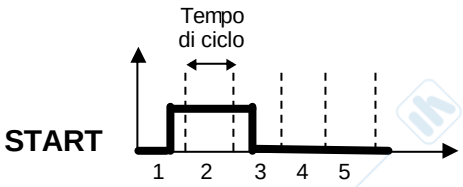
Infine riportiamo alcuni esercizi con soluzione che possono aiutare a comprendere meglio i concetti appena esposti.

## *12.1 Esercizi di Ladder Diagram*

### 12.1.1 Esercizio 1

Si consideri il seguente programma in Ladder Diagram in cui **START** è una variabile di ingresso, **LED** di uscita, e le altre etichette corrispondono a variabili interne.




| | Valore al termine del 1° ciclo | Valore al termine del 2° ciclo | Valore al termine del 3° ciclo | Valore al termine del 4° ciclo | Valore al termine del 5° ciclo |
|---|---|---|---|---|---|
| **SEQ** | 0 | | | | |
| **INIT** | 0 | | | | |
| **W1** | 0 | | | | |
| **W2** | 0 | | | | |
| **LED** | 0 | | | | |

*A) Aggiornare la tabella seguente inserendo i valori che tutte le variabili assumono al termine dei vari cicli di esecuzione del PLC.*

*B) Discutere in modo chiaro e sintetico gli effetti dell'inversione dell'ordine del 4° e 5° piolo.*

**Soluzione:**

A)

| | Valore al termine del 1° ciclo | Valore al termine del 2° ciclo | Valore al termine del 3° ciclo | Valore al termine del 4° ciclo | Valore al termine del 5° ciclo |
|---|---|---|---|---|---|
| **SEQ** | 0 | 1 | 0 | 0 | 0 |
| **INIT** | 0 | 1 | 1 | 1 | 1 |
| **W1** | 0 | 1 | 0 | 0 | 0 |
| **W2** | 0 | 0 | 1 | 0 | 0 |
| **LED** | 0 | 1 | 0 | 0 | 0 |

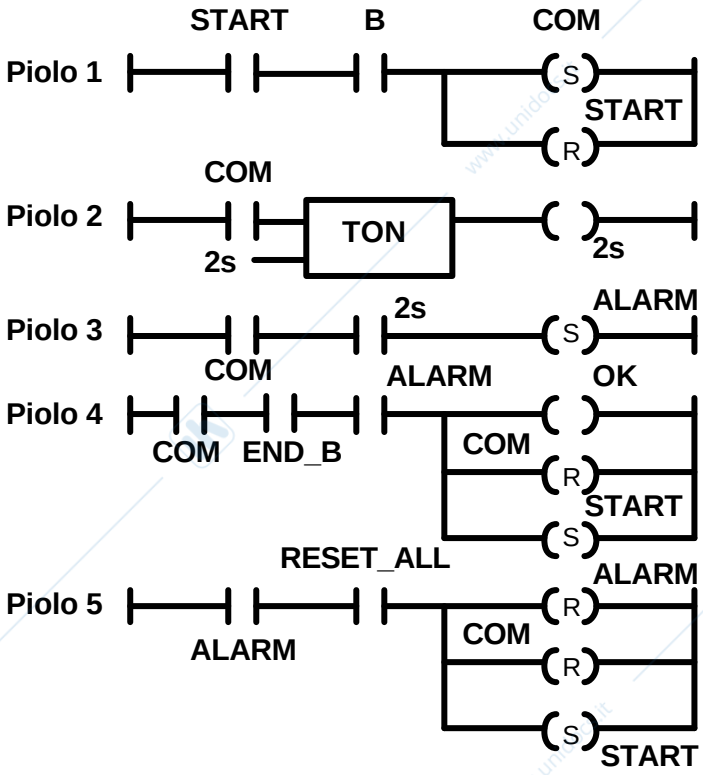
B) Si ha un effetto cascata sulle variabili W1 e W2 che al termine del 2° ciclo saranno entrambe a 1 (vere).

## 12.1.2 Esercizio 2

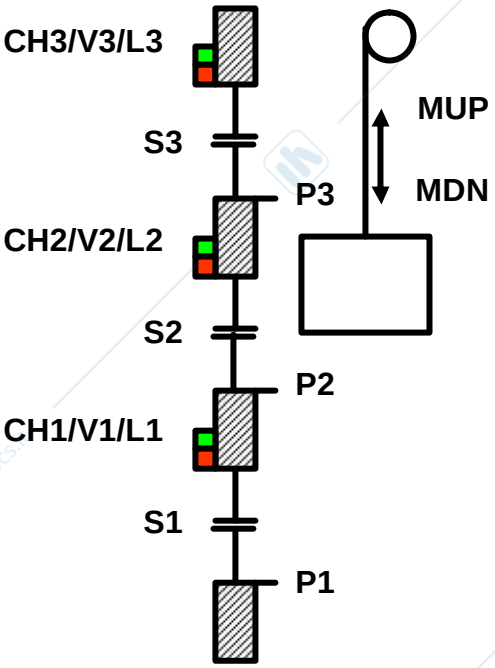
Scrivere un programma LD che soddisfi la specifica di comportamento di seguito descritta:

*Il sistema di controllo si pone in attesa del verificarsi della condizione **B**, alla cui occorrenza emette il comando **COM**. Se entro 2 secondi dall'occorrenza di **B** il segnale **END_B** non assume il valore logico TRUE, emette il comando **ALARM**, altrimenti il comando **OK**. In caso di allarme attivo, si attende il segnale **RESET_ALL** e ci si riporta nuovamente in attesa di **B**; nel caso contrario (emissione del comando **OK**) ci si riporta immediatamente in attesa di **B**.*

**Soluzione:**



### 12.1.3 Esercizio 3




Nel montacarichi in figura quando viene premuto il pulsante **CHx**, esso si illumina e il montacarichi si porta al piano x. Chiaramente il movimento è permesso solo se tutte le porte sono chiuse, e inoltre, in caso di attivazione del motore per più di 10 secondi, il sistema deve andare in blocco e segnalare un allarme.
Gli ingressi (misure) sono:

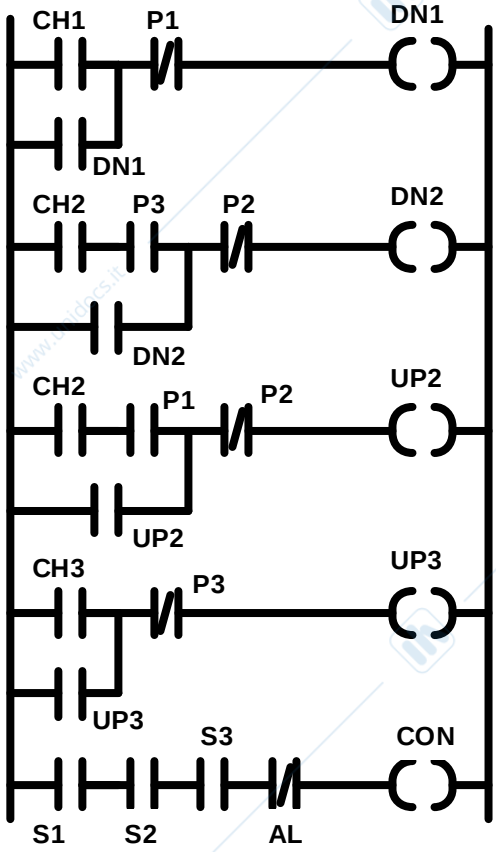
- **CH1** (chiamata piano 1)
- **CH2** (chiamata piano 2)
- **CH3** (chiamata piano 3)
- **S1** (porta 1 chiusa)
- **S2** (porta 2 chiusa )
- **S3** (porta 3 chiusa)
- **P1** (presenza piano 1)
- **P2** (presenza piano 2)
- **P3** (presenza piano 3)

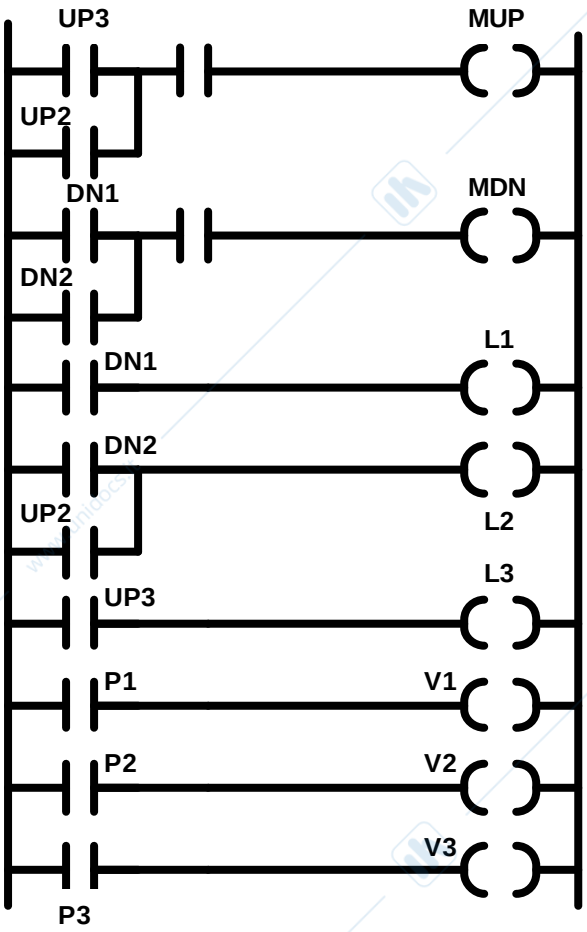
Mentre le uscite (comandi):

- **MUP** (motore su) (comando continuo)
- **MDN** (motore giù) (comando continuo)
- **L1** (luce chiamata piano 1)
- **L2** (luce chiamata piano 2)
- **L3** (luce chiamata piano 3)
- **V1** (luce presenza piano 1)
- **V2** (luce presenza piano 2)
- **V3** (luce presenza piano 3)
- **AL** (allarme blocco)

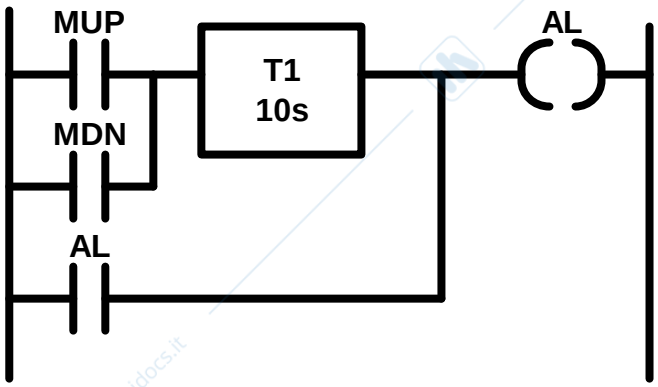
Il funzionamento è illustrato dai punti seguenti, che si riferiscono anche all'immagine della pagina seguente:

- **UP2** (salita al piano 2)
- **UP3** (salita al piano 3)
- **DN2** (discesa al piano 2)
- **DN1** (discesa al piano 1)
- **CON** (consenso al movimento)
- **AL** (allarme blocco)






Il seguente blocco, inoltre, viene utilizzato per gestire gli allarmi di blocco del montacarichi e per emettere i giusti comandi.




### 12.1.4 Esercizio 4

Il seguente esempio si riferisce ad una navetta per il trasporto persone.

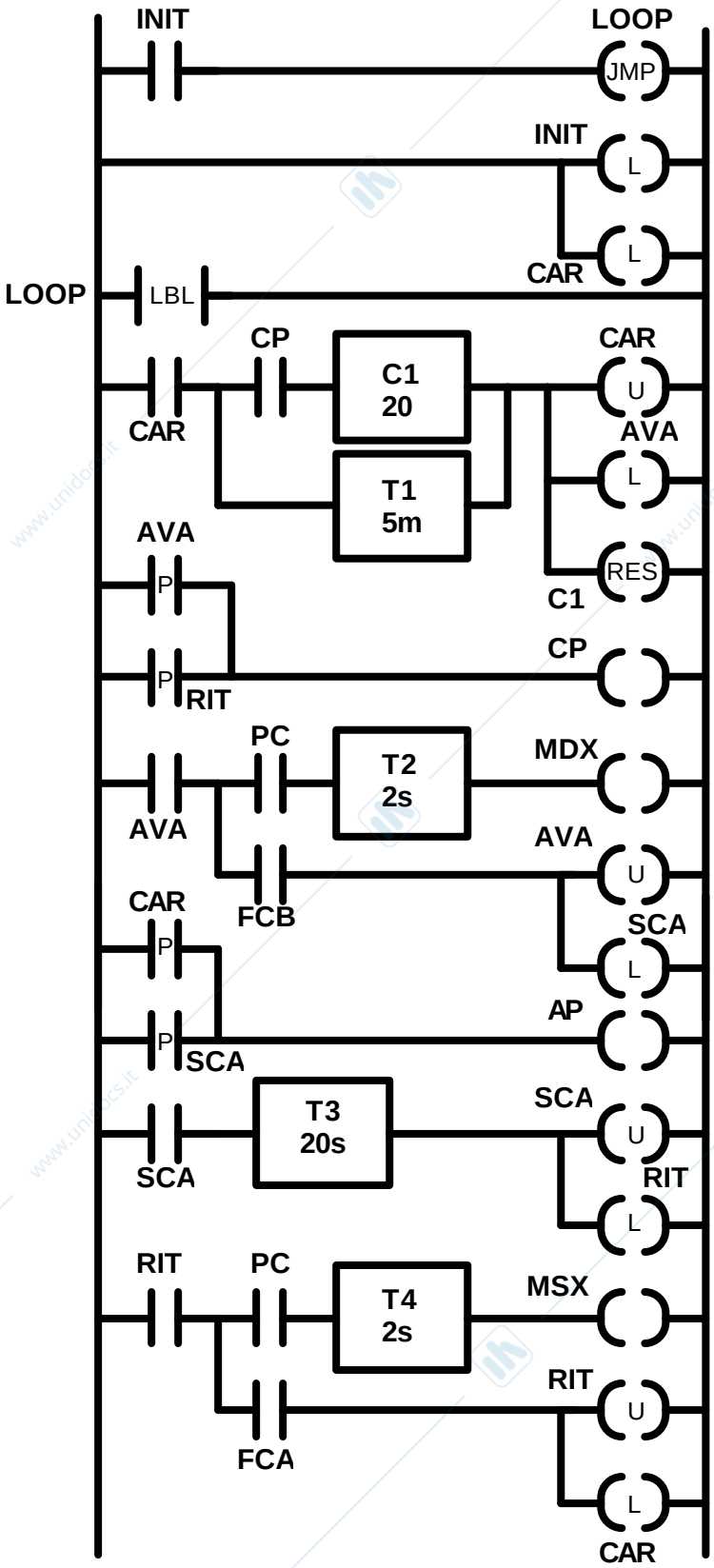

Gli ingressi e le uscite sono i seguenti:

Ingressi (misure)

- **CP** (rotazione tornello contapersone)
- **FCA** (fine corsa A)
- **FCB** (fine corsa B)
- **PC** (porte chiuse)

Uscite (comandi)

- **MDX** (motore verso destra) (com. continuo)
- **MSX** (motore verso sinistra) (com. continuo)
- **AP** (apri porte) (comando impulsivo)
- **CP** (chiudi porte) (comando impulsivo)




Possiamo individuare 4 fasi, segnalate dallo stato di variabili logiche:

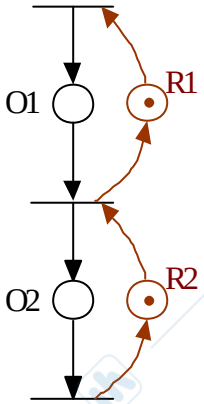
- **CAR** (carico persone)
- **AVA** (avanzam. navetta)
- **SCA** (scarico persone)
- **RIT** (ritorno navetta)

Le variabili logiche corrispondono allo stato di bobine latch/unlatch

## 12.2 Esercizi di SFC

### 12.2.1 Esercizio 1

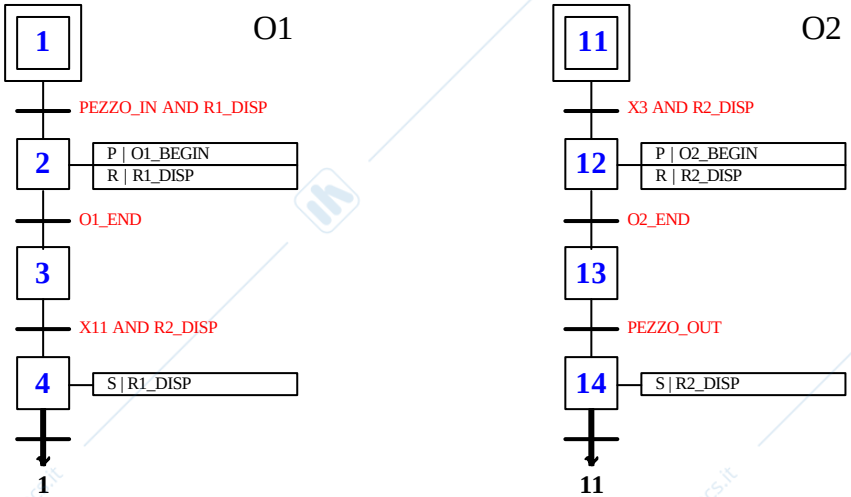
Si consideri la rete di Petri riportata in figura, che rappresenta il modello di una sequenza di operazioni **O1** e **O2** che usano rispettivamente la risorsa **R1** e la risorsa **R2**. Si noti che l'operazione **O1** non può essere attivata finchè non arriva un pezzo da lavorare (tale condizione è rilevata dal segnale **PEZZO_IN**). Analogamente, al termine dell'operazione **O2**, la risorsa **R2** rimane ancora occupata finché il pezzo lavorato non viene asportato manualmente (tale condizione è rilevata dal segnale **PEZZO_OUT**). I segnali di misura e di attuazione sono riportati nella tabella a lato.




| COMANDI | MISURE |
|---|---|
| O1_BEGIN<br>O2_BEGIN | O1_END<br>O2_END<br>PEZZO_IN<br>PEZZO_OUT |

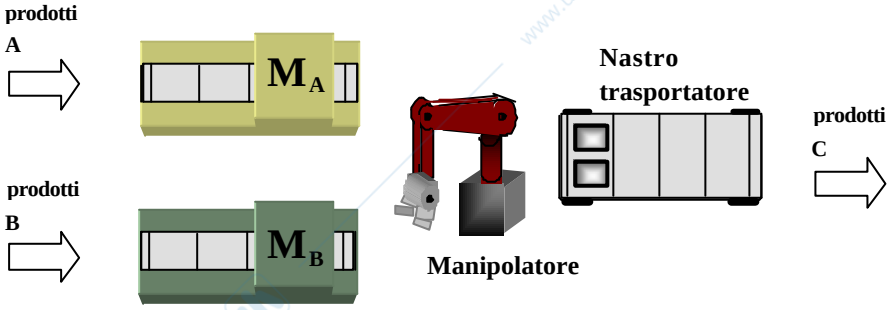
*Modellizzare il processo in SFC, in modo che il comportamento del modello sia identico a quello della rete di Petri considerata. Rappresentare esplicitamente lo stato delle risorse **R1** e **R2**, utilizzando le variabili logiche interne **R1_DISP** e **R2_DISP**.*

**Soluzione:**



## 12.2.2 Esercizio 2

Si consideri l'impianto automatizzato illustrato schematicamente nella figura sottostante. Il funzionamento è il seguente: le due macchine $\mathbf{M_A}$ e $\mathbf{M_B}$ lavorano pezzi grezzi diversi, caricandosi automaticamente da un buffer di ingresso che non è mai vuoto. Le macchine possono lavorare un solo pezzo grezzo alla volta. Un robot manipolatore provvede a trasferire i prodotti lavorati da tali macchine un verso un nastro trasportatore dotato di due posti, uno per il prodotto A e uno per il prodotto B, dove viene effettuata anche l'operazione di assemblaggio. Non è possibile prevedere a priori l'ordine di terminazione della lavorazione delle macchine $\mathbf{M_A}$ e $\mathbf{M_B}$. Quando entrambi i pezzi sono pronti sul nastro nei relativi posti, il robot procede all'assemblaggio, a seguito del quale occorre far avanzare il nastro di una quantità prefissata, lasciando spazio per successivi assemblaggi. I prodotti C assemblati vengono scaricati automaticamente alla fine del nastro.



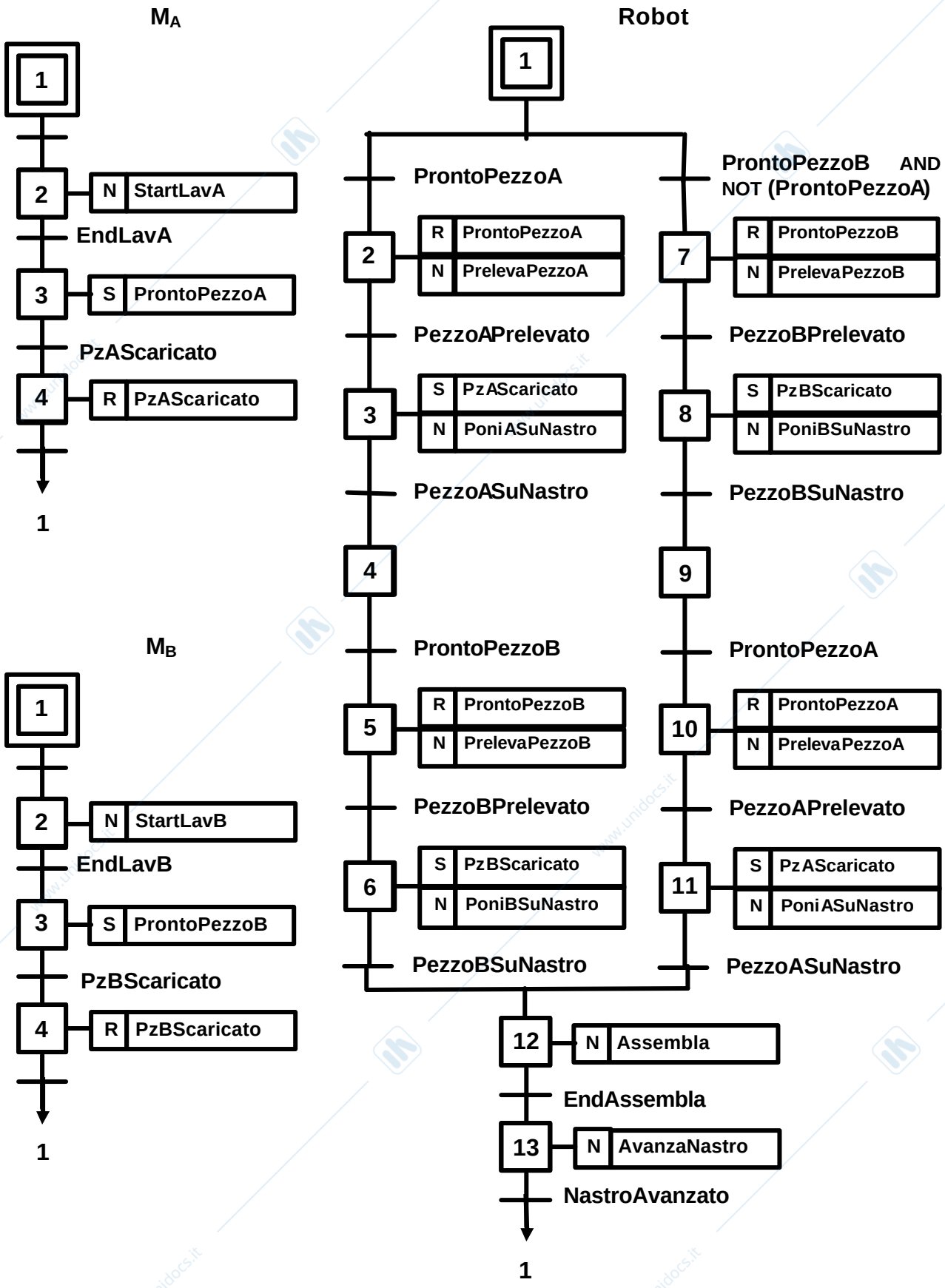

*Identificare con chiarezza le operazioni del robot e del nastro in modo sufficientemente semplificato (ad esempio, per la lavorazione della macchina si possono individuare il comando **StartLavA** e la misura **EndLavA**). Quindi progettare, utilizzando il formalismo SFC, un programma di automazione per il processo produttivo descritto, sfruttando, ove possibile, il parallelismo tra le operazioni.*

**Soluzione:**

Nella pagina seguente viene riportato lo schema SFC equivalente.

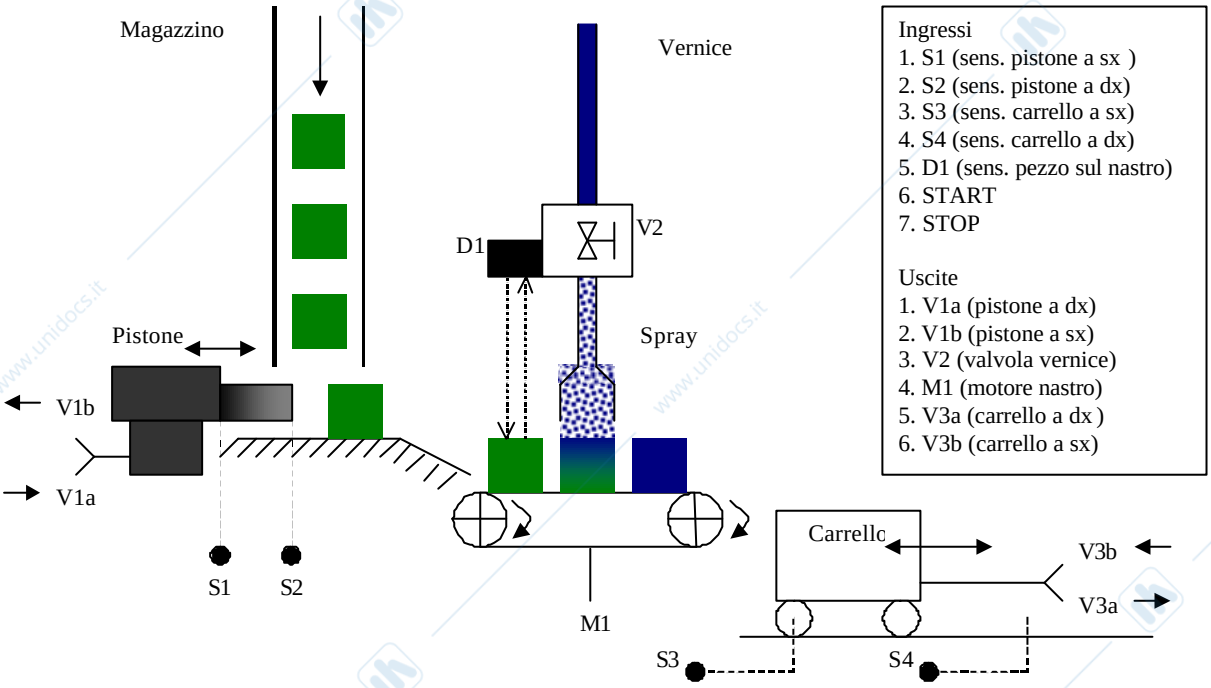
Per una maggiore comprensione si tengano presente le seguenti note riguardanti le operazioni del robot:

- **PrelevaPezzoA** (comando); **PezzoAPrelevato** (misura);
- **PrelevaPezzoB** (comando); **PezzoBPrelevato** (misura);
- **PoniAsuNastro** (comando); **PezzoAsuNastro** (misura);
- **PoniBsuNastro** (comando); **PezzoBsuNastro** (misura);
- **Assembla** (comando); **EndAssembla** (misura);
- **AvanzaNastro** (comando); **NastroAvanzato** (misura);



## 12.2.3 Esercizio 3

Si consideri il sistema di verniciatura rappresentato nella figura sottostante.

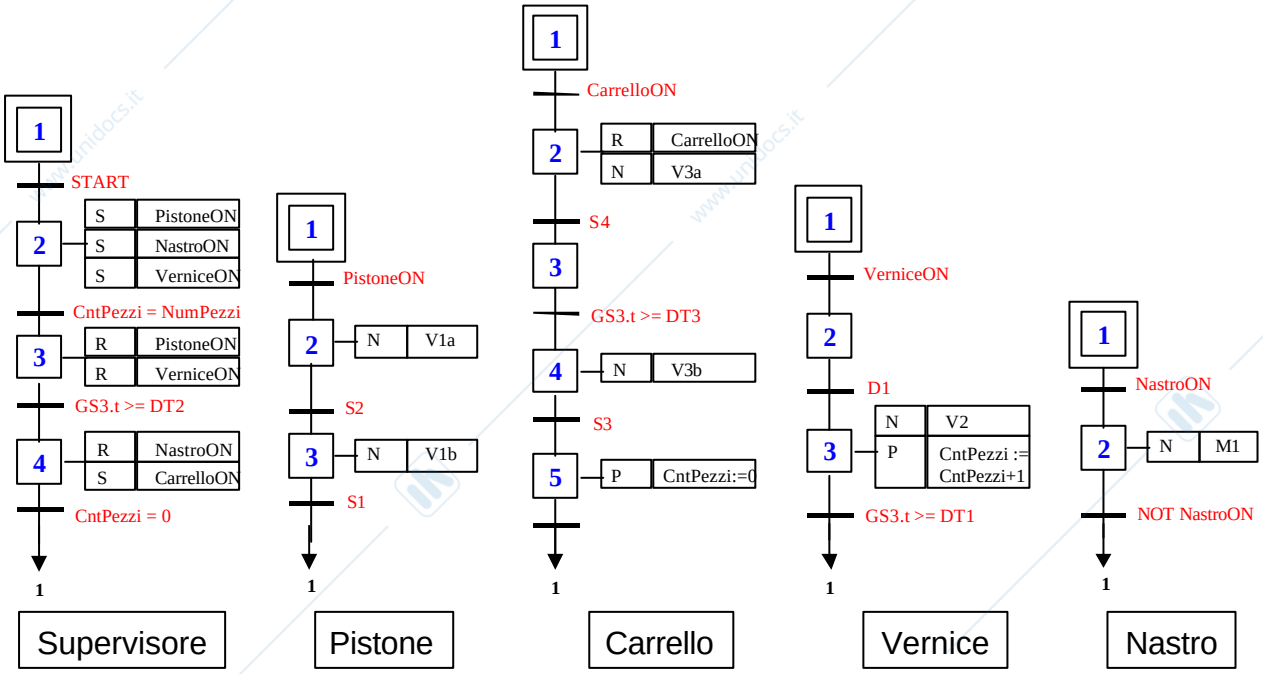



Nel suo stato iniziale, il pistone è fermo al finecorsa sinistro e la valvola della vernice è chiusa; il nastro trasportatore è spento e il carrello è fermo anch'esso al finecorsa sinistro.
Il sistema deve rispettare i seguenti vincoli: i pezzi, provenienti dal magazzino, sono spinti dal pistone (**V1a**, **V1b**) sullo scivolo che termina sul nastro trasportatore (in movimento) azionato da un motore (**M1**); ogni pezzo depositato sul nastro trasportatore viene rilevato tramite un sensore (**D1**); ogni volta che viene rilevata la presenza di un pezzo sul nastro viene aperta la valvola che aziona il circuito per la vernice spray per un intervallo di tempo **DT1**. Dopo essere stato verniciato, ogni pezzo cadrà nel carrello.
Quando 3 pezzi sono stati verniciati il pistone si ferma e dopo un intervallo di tempo **DT2** anche il nastro trasportatore si fermerà. Il carrello con i 3 pezzi verniciati si sposta fino al finecorsa destro, dove viene svuotato automaticamente. Infine, trascorso un intervallo di tempo prestabilito **DT3** dal raggiungimento del finecorsa destro, il carrello torna nella posizione iniziale e la sequenza di lavorazione può ricominciare.

*Si progetti uno schema SFC che controlla l'impianto di verniciatura in modo che le specifiche sopra scritte siano rispettate. Si ponga particolare attenzione a garantire la massima flessibilità di uso dei vari componenti del sistema.*

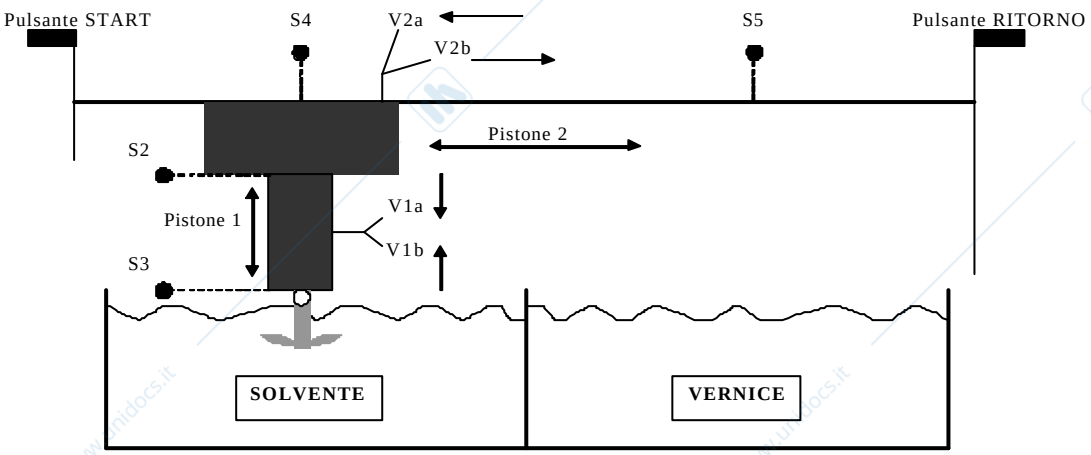
**Soluzione:**

Assumiamo che il tempo di scivolo di un pezzo verso il sistema di spray sia molto minore del tempo di ritorno del pistone, in modo che il conteggio di un pezzo avvenga prima che un nuovo pezzo possa essere spinto sullo scivolo. Assumiamo inoltre che tutti i segnali siano di tipo continuo, ovvero non ci preoccupiamo di generare per un segnale di uscita un impulso di durata finita.
Una soluzione possibile è quella mostrata in figura, dove si sono rappresentati diversi moduli SFC, ognuno dedicato ad un componente del sistema, più un "supervisore" che coordina i vari moduli.




## 12.2.4 Esercizio 4

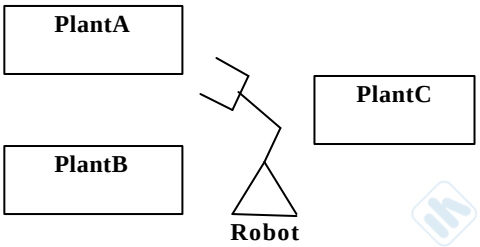
Si consideri il processo industriale per la pulizia e verniciatura di componenti metalliche, rappresentato nella figura seguente.



Inizialmente, il pistone 1 è in posizione ALTO (**S2**) e il pistone 2 è in posizione SINISTRA (**S4**). La sequenza di lavorazione è la seguente: il componente da lavorare viene posizionato su di un uncino posto all'estremità del pistone 1 (operazione manuale) e il componente è sopra il serbatoio dove è contenuto il solvente. Dopo la pressione del pulsante **START**, il pistone 1 passa in posizione BASSO (**S3**), quindi il componente viene lasciato nel solvente per 3 secondi. Successivamente, il pistone 1 passa in posizione ALTO (**S2**), poi il pistone 2 passa in posizione DESTRA (**S5**), posizionando il componente sopra il serbatoio dove è contenuta la vernice, quindi il pistone 1 passa in posizione BASSO (**S3**) in modo tale da immergere il componente nella vernice. Il componente viene lasciato nella vernice per 5 secondi, dopodichè il pistone 1 ritorna in posizione ALTO (**S2**). A questo punto, i pistoni rimangono fermi fino a che un operatore (dopo aver provveduto a rimuovere il pezzo lavorato dall'uncino) preme il pulsante **RITORNO**, che consente al pistone 2 di passare in posizione SINISTRA (**S4**): il sistema è tornato alla posizione iniziale.
I segnali di misura dall'impianto e quelli di attuazione, sono i seguenti:

- Ingressi per il PLC (provenienti dal campo)
  - **START**: segnale di inizio (tasto premuto da un operatore)
  - **RITORNO**: segnale di ritorno (tasto premuto da un operatore)
  - **S2**: sensore pistone 1 in alto
  - **S3**: sensore pistone 1 in basso
  - **S4**: sensore pistone 2 a sinistra
  - **S5**: sensore pistone 2 a destra
- Uscite del PLC (comandi per l'impianto, tutti di tipo continuo)
  - **V1a**: sposta il pistone 1 in basso
  - **V1b**: sposta il pistone 1 in alto
  - **V2a**: sposta il pistone 2 a sinistra
  - **V2b**: sposta il pistone 2 a destra

A) *Realizzare il programma di automazione per l'impianto utilizzando il linguaggio SFC. E' possibile utilizzare tutti gli ingressi e le uscite sopra definiti. In caso si desideri utilizzare variabili ausiliarie, è necessario definirle in maniera univoca e comprensibile.*




Si supponga poi che l'impianto (nel seguito denominato PlantA) appena descritto, sia affiancato da un altro impianto identico (PlantB) e da un impianto leggermente diverso (PlantC). PlantA e PlantB sono affiancati, mentre tra questi e PlantC è interposto un robot scaricatore, come mostrato in figura. *PlantC* è costruttivamente identico a PlantA e PlantB: la differenza consiste solo nel fatto che la sua prima vasca contiene della vernice protettiva per il pezzo e che la sua seconda vasca contiene uno speciale polimero liquido necessario per fissare la vernice protettiva al pezzo lavorato.
La presenza del Robot scaricatore fa sì che i segnali di Ritorno per PlantA e PlantB e il segnale di Start per PlantC non siano più generati da un operatore tramite la pressione di un pulsante, ma siano generati dal Robot stesso alla fine di opportune operazioni. Il Robot, infatti, grazie ad alcuni sensori, riconosce automaticamente quando un pezzo verniciato è disponibile alla fine di PlantA o di PlantB, lo prende, manda il segnale di ritorno opportuno, lo carica per PlantC e si rimette in attesa che un nuovo pezzo verniciato sia disponibile.

*B) Si chiede di progettare, utilizzando il formalismo del linguaggio SFC, un programma di automazione per il processo produttivo descritto, sfruttando, ove possibile, il parallelismo tra le operazioni. Per semplicità, si considerino già implementate con sequenze SFC le lavorazioni dei sistemi PlantA, PlantB, PlantC e Robot; fare quindi riferimento a queste sequenze utilizzando dei macro-passi che le identificano o in alternativa utilizzando opportunamente variabili interne per attivarle e disattivarle.*

**Suggerimento**: Per ogni plant si definiscono due macro passi, uno per la lavorazione (Sequence) e uno per il riposizionamento dei pistoni nelle condizioni iniziali (Init) e due segnali per attivare la lavorazione (Start) e per rendere l'impianto disponibile per una nuova lavorazione (Ritorno) come mostrato in tabella:

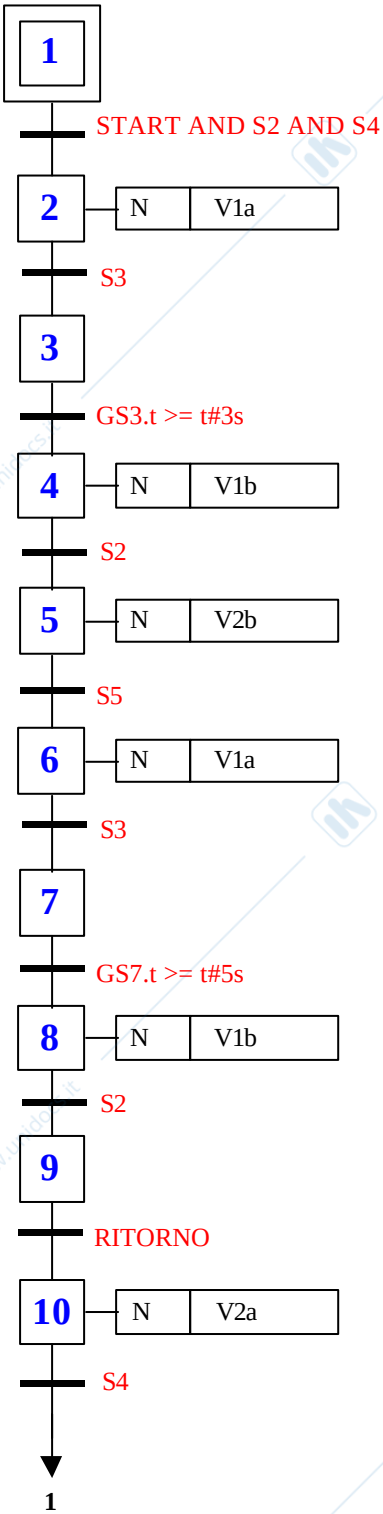
| | **PlantA** | **PlantB** | **PlantC** |
|---|---|---|---|
| macro passo | InitA | InitB | InitC |
| macro passo | SequenceA | SequenceB | SequenceC |
| segnale di ingresso | StartA (pulsante premuto dall'operatore) | StartB (pulsante premuto dall'operatore) | StartC (segnale attivato dal robot) |
| segnale di ingresso | RitornoA (segnale attivato dal robot) | RitornoB (segnale attivato dal robot) | RitornoC (pulsante premuto dall'operatore) |

Si suppone inoltre che il Robot possa utilizzare i seguenti segnali:

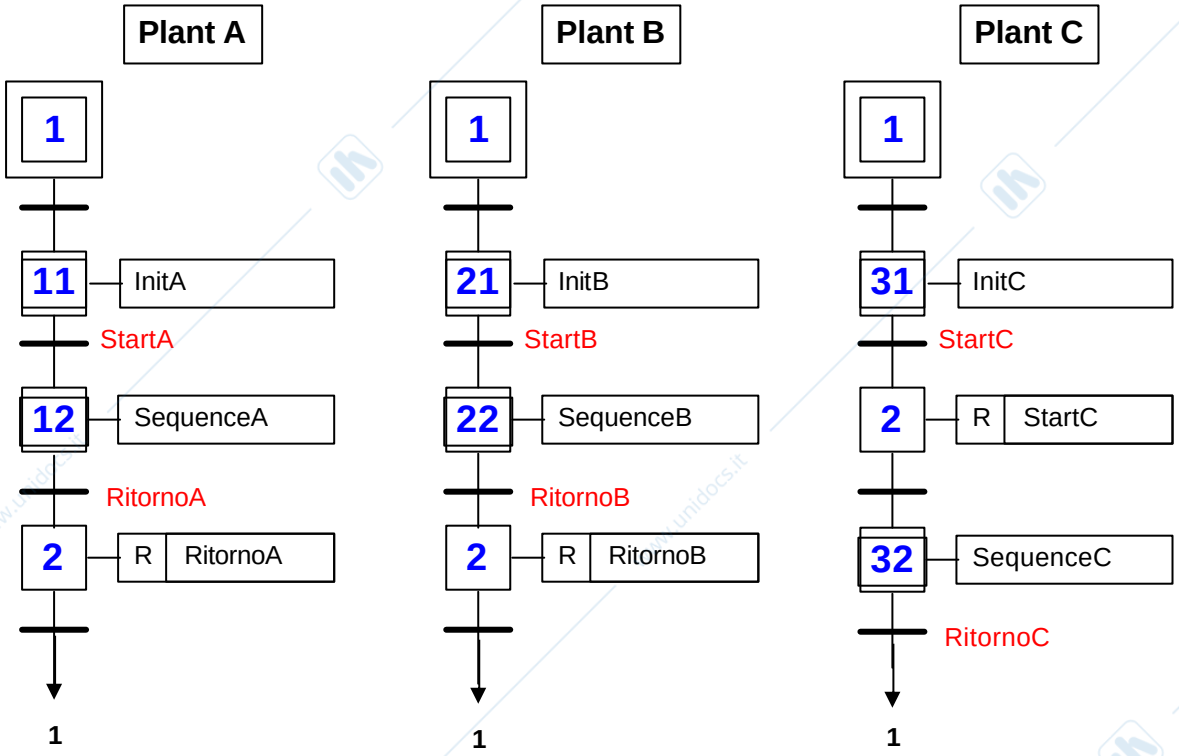
| **Ingressi** | **Uscite** |
|---|---|
| ProntoPezzoA (True quando un pezzo è pronto alla fine del PlantA) | ScaricaA (azione di tipo continuo) |
| ProntoPezzoB (True quando un pezzo è pronto alla fine del PlantB) | ScaricaB (azione di tipo continuo) |
| ScaricatoA (True quando un pezzo è stato scaricato da PlantA) | CaricaC (azione di tipo continuo) |
| ScaricatoB (True quando un pezzo è stato scaricato da PlantB) | RitornoA (azione di tipo continuo) |
| DisponibilePlantC (True quando PlantC è disponibile per effettuare una lavorazione) | RitornoB (azione di tipo continuo) |
| CaricatoC (True quando un pezzo è stato caricato per PlantC) | StartC (azione di tipo continuo) |

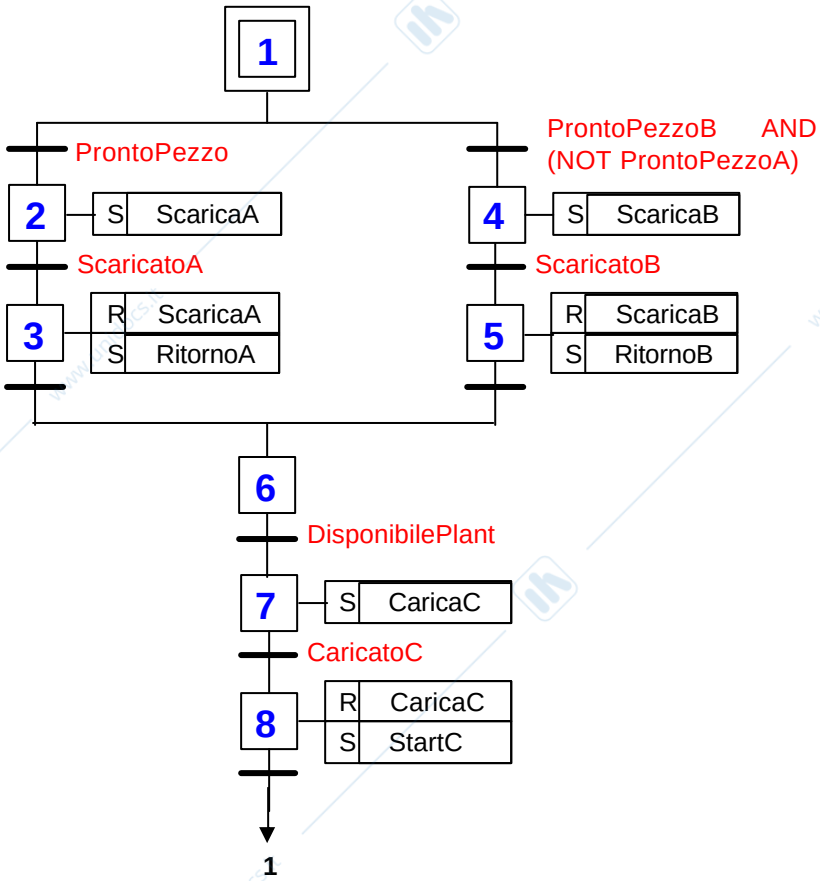
**Soluzione:**

A)



B)



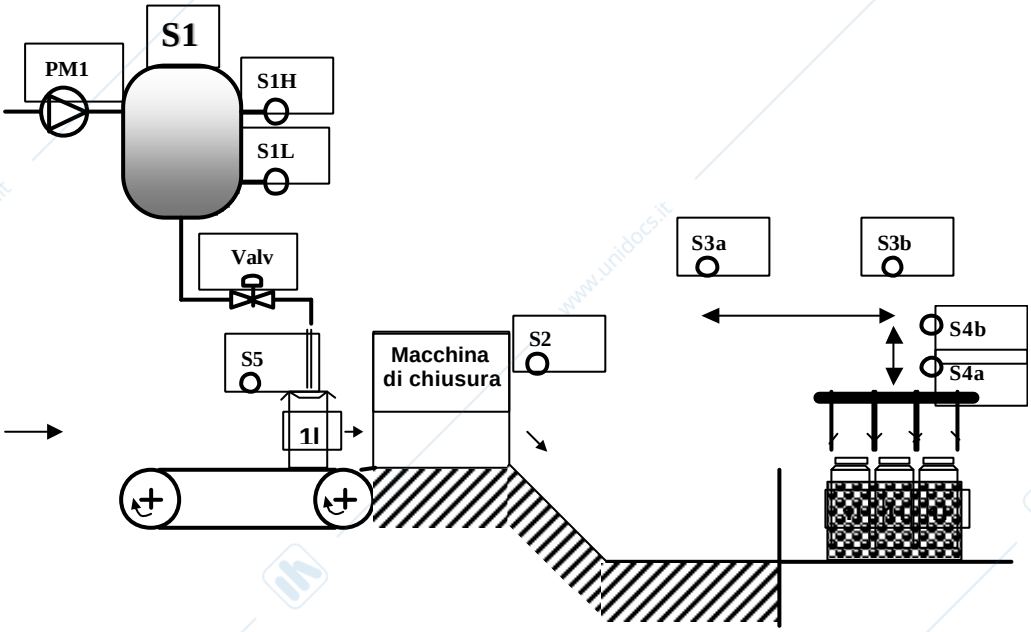

Le operazioni scarica/carica del robot funzionano nel modo seguente:

Il robot è fermo in posizione di riposo con/senza pezzo. Si muove verso il pezzo da caricare/scaricare, carica/Scarica il pezzo, torna in posizione di riposo senza/con il pezzo.

I segnali **CaricatoA**, **CaricatoB**, **ScaricatoA**, ecc… diventano “true” quando il robot è tornato alla posizione di riposo dopo aver effettuato l’operazione.

## 12.2.5 Esercizio 5

In figura è rappresentata una porzione di un impianto per la preparazione e l'imballaggio di cartoni di latte da 1 litro. Il sistema è composto da un serbatoio **S1** di capacità molto maggiore di un litro, un nastro trasportatore, una macchina per la chiusura ermetica del cartone ed una navetta per il trasporto dei cartoni nella cassa.




Gli elementi di controllo presenti sull'impianto sono indicati in tabella.

| Tipo | Nome | Descrizione |
|---|---|---|
| Attuatore\continuo | **PM1** | Pompa per il trasporto del latte da un serbatoio INFINITO al serbatoio **S1** |
| Attuatore\continuo | **Valv** | Valvola per il passaggio del latte dal serbatoio **S1** al cartone (normalmente chiusa) |
| Sensore | **S1H** | Indica il livello alto del serbatoio **S1** (se attivo = serbatoio pieno) |
| Sensore | **S1L** | Indica il livello basso del serbatoio **S1** (se non attivo = serbatoio vuoto) |
| Attuatore\continuo | **K1** | Motore di avanzamento nastro |
| Sensore | **S5** | Indica la presenza di un cartone sul nastro nella posizione corretta per il riempimento. Quando è alto è possibile riempire il cartone. |
| Sensore | **S2** | Indica il passaggio di un cartone chiuso all'uscita della macchina di chiusura |
| Sensore | **S3a** | Finecorsa sinistro della navetta |
| Sensore | **S3b** | Finecorsa destro della navetta |
| Sensore | **S4a** | Finecorsa basso della navetta |
| Sensore | **S4b** | Finecorsa alto della navetta |
| Attuatore\continuo | **Ndx** | Porta la navetta a destra |
| Attuatore\continuo | **Nsx** | Porta la navetta a sinistra |
| Attuatore\continuo | **Nh** | Porta la navetta in alto |
| Attuatore\continuo | **Nl** | Porta la navetta in basso |

All'accensione, il latte proveniente dal resto dell'impianto deve essere portato all'interno del serbatoio **S1**. Quando il serbatoio è pieno e un cartone vuoto è presente sul nastro nella posizione corretta (segnalata da **S5**), la valvola viene aperta per un tempo t assegnato a priori. Si assuma che in ingresso al nastro ci sia un numero infinito di cartoni vuoti. Quando il cartone è stato riempito il nastro viene riattivato in modo da portare un nuovo cartone vuoto nella posizione di riempimento; questo movimento permette anche di portare il cartone pieno nella macchina di chiusura per essere chiuso e datato. Le operazioni svolte dalla macchina di chiusura sono tutte controllate da un controllore dedicato e non sono dunque rilevanti nel nostro sistema. Quando la macchina di chiusura ha concluso le operazioni il cartone esce dalla macchina (attivando il sensore **S2**) e scivola su un piano inclinato. Quando sul piano sono presenti 3 cartoni la navetta, dotata di apposita attrezzatura, li prende tutti e li porta nella cassa che è dunque pronta per essere immessa sul mercato. Considerando come posizione iniziale quella rappresentata in figura (navetta in alto a destra), la navetta deve dunque spostarsi verso sinistra, scendere per agganciare i cartoni, salire, spostarsi verso destra, scendere per rilasciare i cartoni nella cassa e risalire riportandosi così nella posizione iniziale. Si assuma che lo scarico della cassa piena e il carico di una vuota siano svolti correttamente da un operatore.

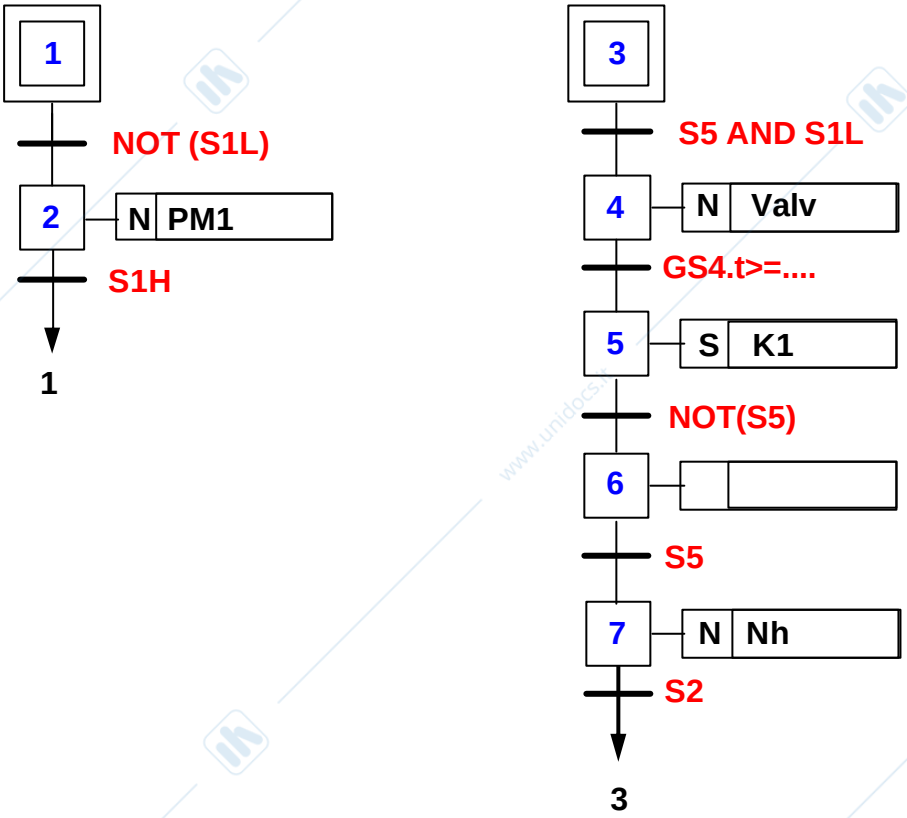
*Scrivere un programma SFC che implementi un controllo che soddisfi la specifica di comportamento descritta in modo tale da sfruttare al meglio le risorse dell'impianto e riducendo così i tempi di produzione.*

**Soluzione:**

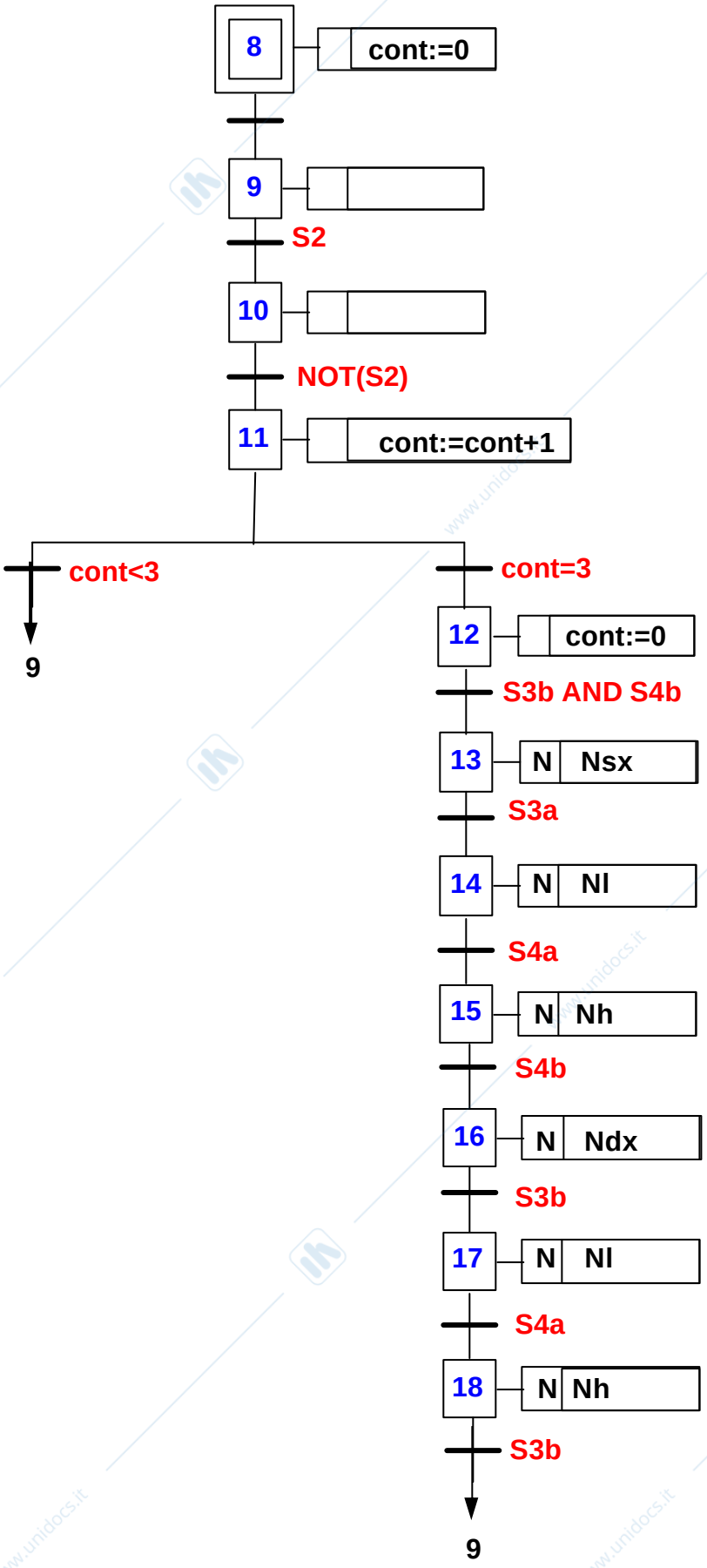
Il testo lascia aperte diverse possibilità di risoluzione, corrispondenti all'introduzione di ipotesi da parte dello studente, cui poi la soluzione stesa si deve necessariamente attenere.
Supponiamo ad esempio di introdurre le seguenti ipotesi:

- La pompa deve essere attivata ogniqualvolta il serbatoio contiene meno di 1 litro e deve restare attiva fino al riempimento del serbatoio.
- "Quando il serbatoio è pieno" significa quando in esso è presente una quantità di latte superiore a **S1L**.
- La portata della pompa è tale da riempire rapidamente il serbatoio.
- Alla pressione di **ST** un cartone vuoto sia già presente nella corretta posizione.
- Il riempimento di un nuovo cartone può avvenire solo dopo che la macchina ha concluso le sue operazioni, il che richiede un tempo superiore rispetto a quello necessario per ottenere un cartone vuoto nella posizione corretta.
- La navetta completa lo spostamento prima che un nuovo cartone esca dalla macchina.

La soluzione che si ottiene è la seguente:



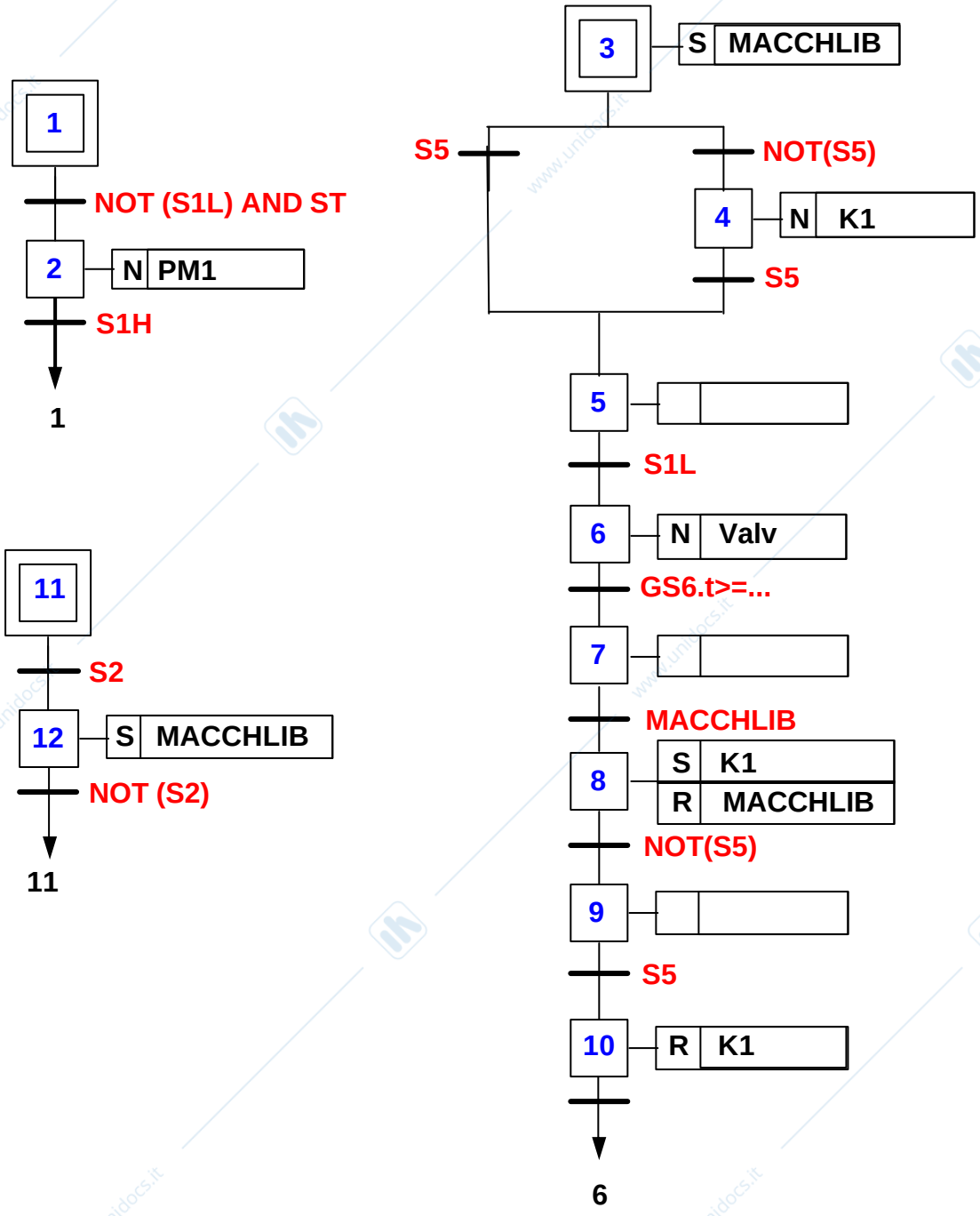

Lo schema completo è riportato invece nella pagina seguente.

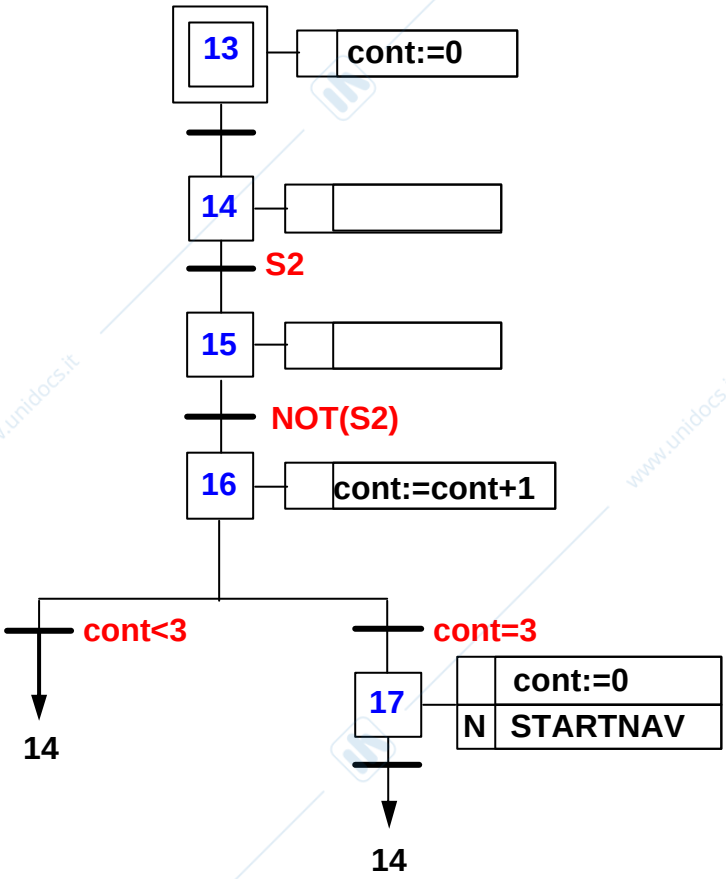


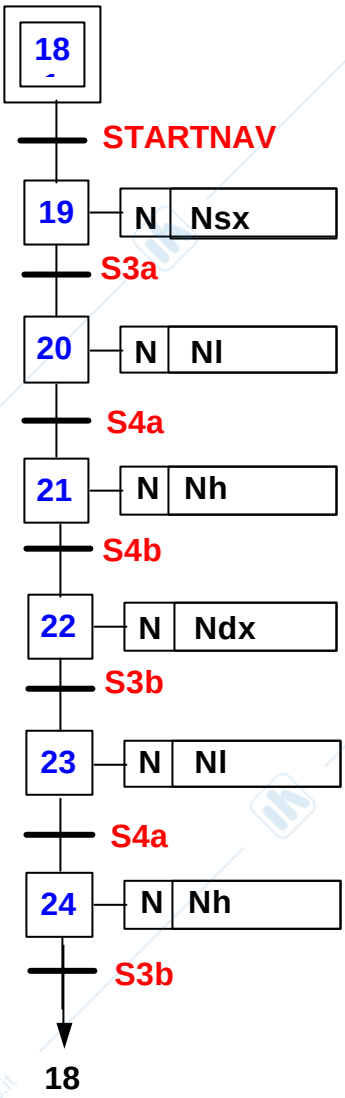
Interpretando invece letteralmente la frase “Quando il serbatoio è pieno” si avrebbe la misura “**S5** AND **S1H**” in corrispondenza della transizione a valle del passo 3.
Introducendo, invece delle sei ipotesi precedenti, solo le prime tre:

- La pompa deve essere attivata ogniqualvolta il serbatoio contiene meno di 1 litro e deve restare attiva fino al riempimento del serbatoio.
- “Quando il serbatoio è pieno” significa quando in esso è presente una quantità di latte superiore a **S1L**.
- La portata della pompa è tale da riempire rapidamente il serbatoio.

Si ottiene la seguente soluzione che sfrutta meglio le risorse:

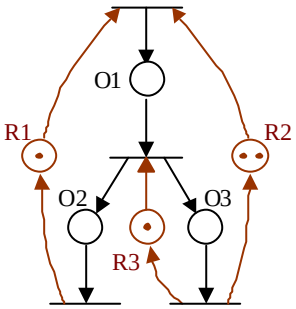




## 12.2.6 Esercizio 6:

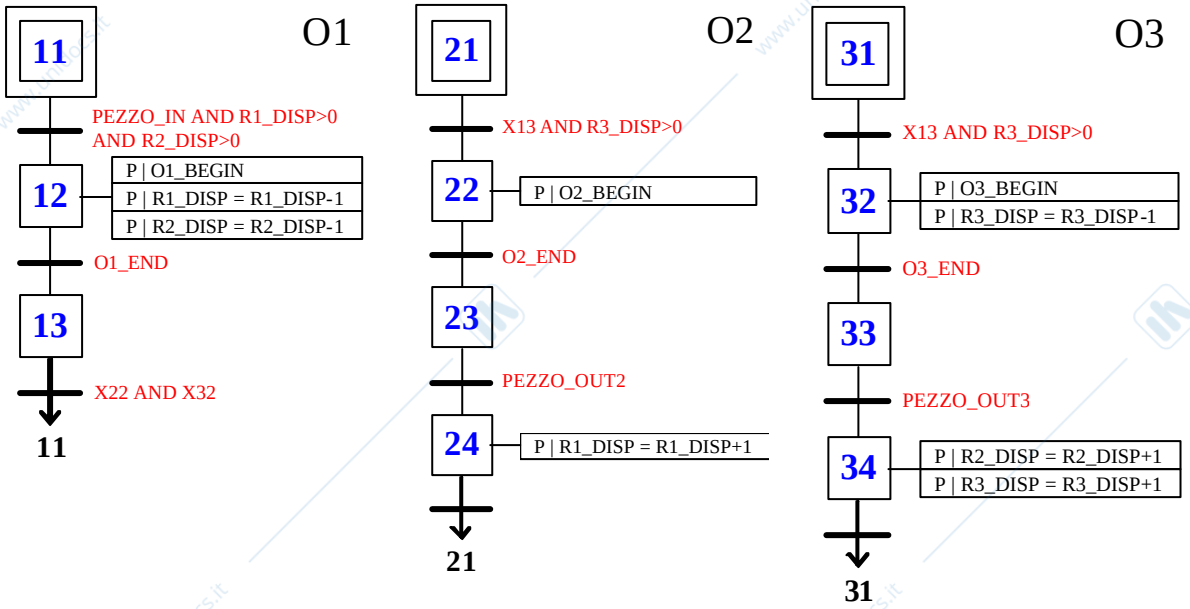
Si consideri la rete di Petri riportata in figura, che rappresenta il modello di un processo con 3 operazioni **O1**, **O2** e **O3**, che utilizza le risorse **R1**, **R2** e **R3**. Si noti che l'operazione **O1** non può essere attivata finché non arriva un pezzo da lavorare (tale condizione è rilevata dal segnale **PEZZO_IN**). Analogamente, al termine delle operazioni **O2** e **O3**, le risorse utilizzate rimangono ancora occupate finché i pezzi lavorati non vengono asportati manualmente (tale condizione è rilevata dai segnali **PEZZO_OUT2** e **PEZZO_OUT3**, rispettivamente). I segnali di misura e di attuazione sono riportati nella tabella a lato.




| COMANDI | MISURE |
|---|---|
| O1_BEGIN | O1_END |
| O2_BEGIN | O2_END |
| O3_BEGIN | O3_END |
| | PEZZO_IN |
| | PEZZO_OUT2 |
| | PEZZO_OUT3 |

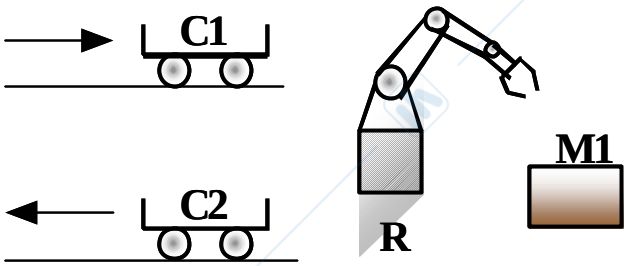
*Modellizzare il processo in SFC, in modo che il comportamento del modello sia identico a quello della rete di Petri considerata. Rappresentare esplicitamente lo stato delle risorse R1, R2 e R3, utilizzando le variabili interne R1_DISP, R2_DISP e R3_DISP.*

**Soluzione:**



## *12.3 Esercizi di Automi e Grafcet*

Il sistema di figura rappresenta una cella robotizzata costituita da due carrelli per carico (**C1**)/scarico (**C2**) di pezzi grezzi/lavorati, da un manipolatore e da una macchina per la lavorazione del pezzo (**M1**). **R** deve prelevare un pezzo grezzo da **C1**, caricarlo su **M1** e, al termine della lavorazione, scaricare il pezzo finito su **C2**.




*A) Specificare il comportamento desiderato di un supervisore che coordina i sistemi di controllo di R e M1 attraverso il formalismo degli automi di Mealy. Per semplicità, si consideri:*

- *un pezzo grezzo sempre disponibile in C1*
- *C2 sempre pronto ad accettare un pezzo finito*

**Soluzione:**

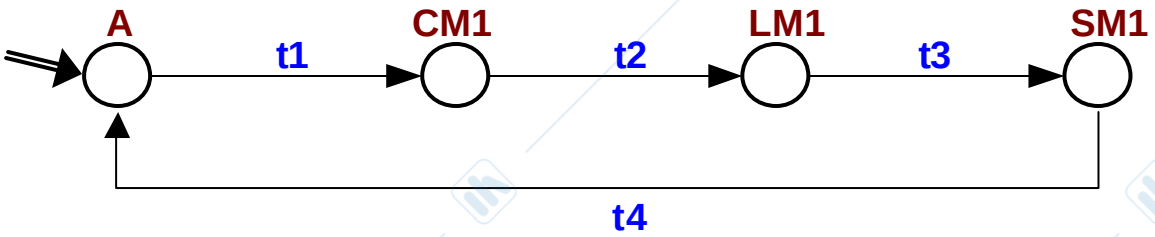
Stati di R e M1:

**R:** {**A**ttesa, **C**arico, **S**carico}
**M1:** {**V**uota, **L**avorazione, **P**iena}
Totale Stati possibili: $3 \times 3 = 9$

Stati desiderati

| R | M1 | | |
|---|---|---|---|
| A | V | A | *Attesa* |
| C | V | CM1 | *Carico M1* |
| A | L | LM1 | *Lavorazione M1* |
| S | P | SM1 | *Scarico M1* |

L'automa di Mealy è il seguente, in cui gli ingressi e le uscite sono associati alle transizioni (segnali scambiati tra supervisore e controllori locali).




Il significato delle transizioni è il seguente:

- **t1:** - /carica M1
- **t2:** fine carico M1/ lavora M1
- **t3:** fine lavorazione M1/Scarica M1
- **t4:** fine scarico M1/ -

*B) Si modifichi l'automa di cui al punto 1 in modo tale da considerare esplicitamente la disponibilità dei carrelli.*

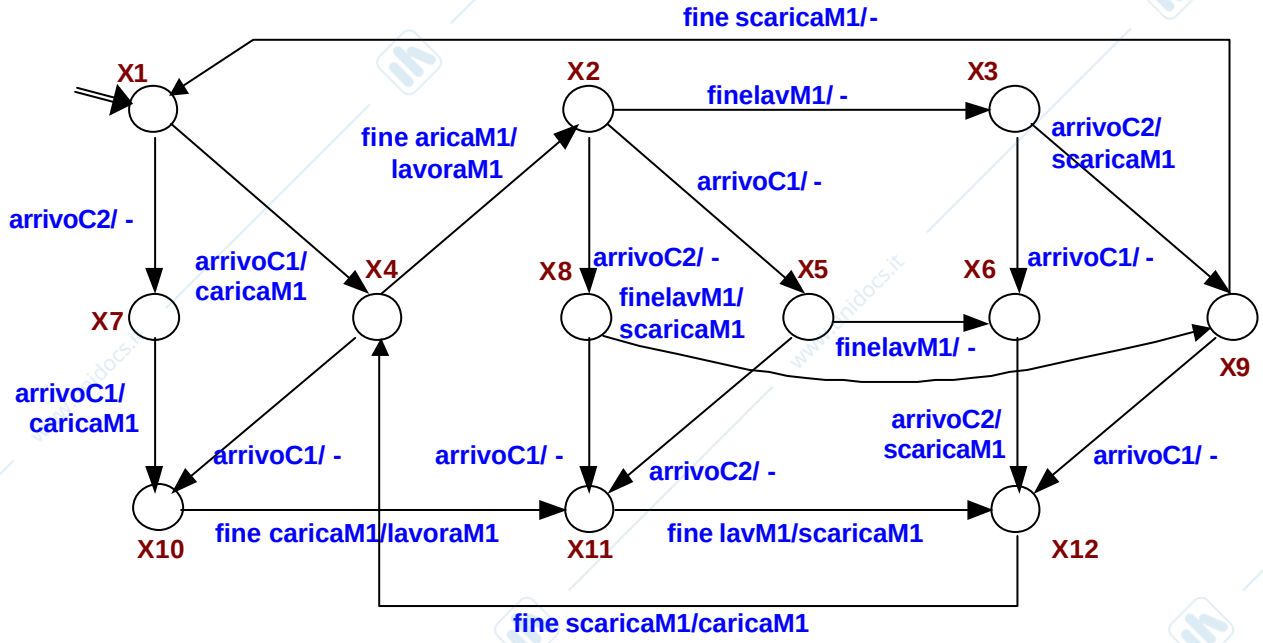
**Soluzione:**

Stati di R, M1, C1 e C2:

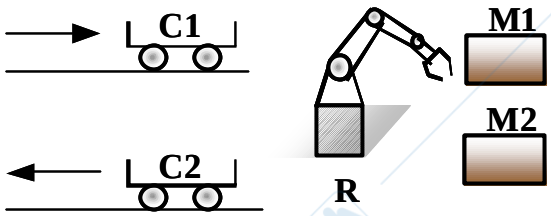
**R:** {**A**ttesa, **C**arico, **S**carico}
**M1:** {**V**uota, **L**avorazione, **P**iena}
**C1**,**C2**: {**P**ronto, **N**on pronto}
Totale Stati possibili: $3 \times 3 \times 2 \times 2 = 36$

Stati desiderati

| | R | M1 | C1 | C2 |
|---|---|---|---|---|
| **X1** | A | V | N | N |
| **X2** | A | L | N | N |
| **X3** | A | P | N | N |
| **X4** | C | V | P | N |
| **X5** | A | L | P | N |
| **X6** | A | P | P | N |
| **X7** | A | V | N | P |
| **X8** | A | L | N | P |
| **X9** | S | P | N | P |
| **X10** | C | V | P | P |
| **X11** | A | L | P | P |
| **X12** | S | P | P | P |



*C) Si consideri ora di aggiungere una seconda macchina M2 al sistema come da figura.*




*La specifica di funzionamento di questo nuovo sistema prevede che il pezzo venga lavorato in sequenza prima da M1 e poi da M2 e che il prodotto finito venga poi scaricato da M2 a C2. Ancora una volta, per semplicità, si consideri:*

- *un pezzo grezzo sempre disponibile in C1*
- *C2 sempre pronto ad accettare un pezzo finito*

*Fornire una specifica utilizzando sempre il formalismo degli automi di Mealy per questo nuovo sistema.*

**Soluzione:**

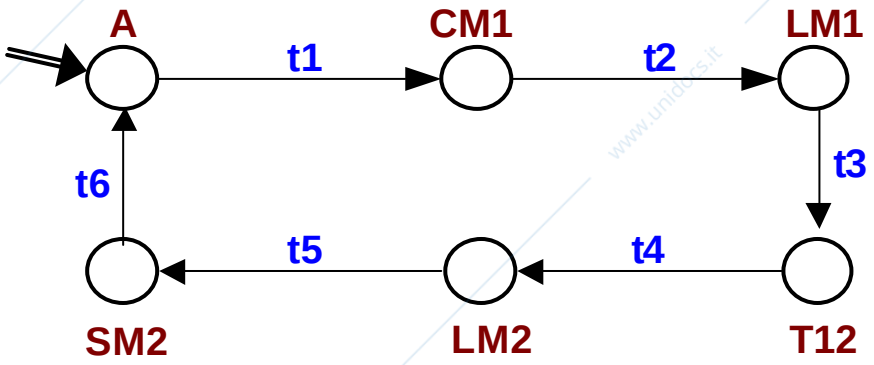
Stati di R e M1:

**R:** {**A**ttesa, **C**arico, **S**carico, **T**rasporto M1-M2}
**M1:** {**V**uota, **L**avorazione, **P**iena}
**M2:** {**V**uota, **L**avorazione, **P**iena}

Stati desiderati

| R | M1 | M2 | | |
|---|---|---|---|---|
| A | V | V | A | *Attesa* |
| C | V | V | CM1 | *Carico M1* |
| A | L | V | LM1 | *Lavorazione M1* |
| T | P | V | T12 | *Trasporto da T1 a T2* |
| A | V | L | LM2 | *Lavorazione M2* |
| S | V | p | SM2 | *Scarico M2* |

Ed ecco il corrispondente automa:




Con il relativo significato delle transizioni:

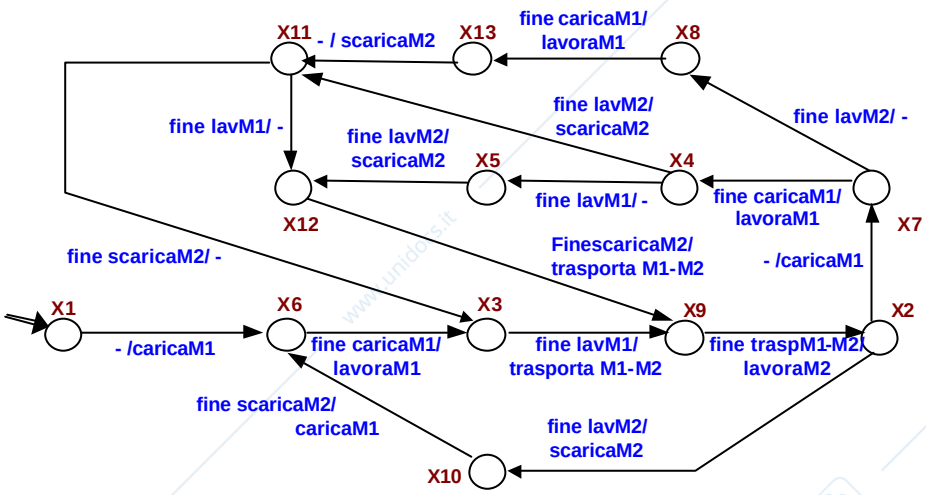
- **t1**: - /carica M1
- **t2**: fine carico M1/ lavora M1
- **t3**: fine lavorazione M1/trasporta M1-M2
- **t4**: fine trasporto M1-M2/ lavora M2
- **t5**: fine lavorazione M2/scarica M2
- **t6**: fine scarico M2/ -

*D) Il sistema di cui al punto 3 può eseguire più lavorazioni in parallelo. Modificare l'automa realizzato al punto 3 in modo tale da ottimizzare l'uso delle risorse.*

**Soluzione:**

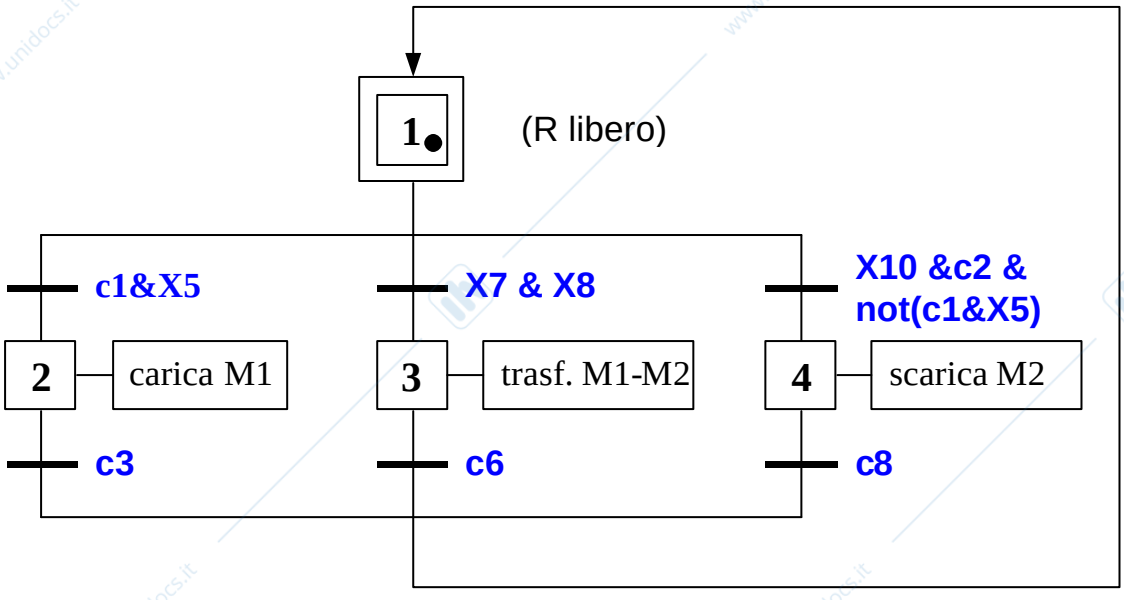
Stati desiderati:

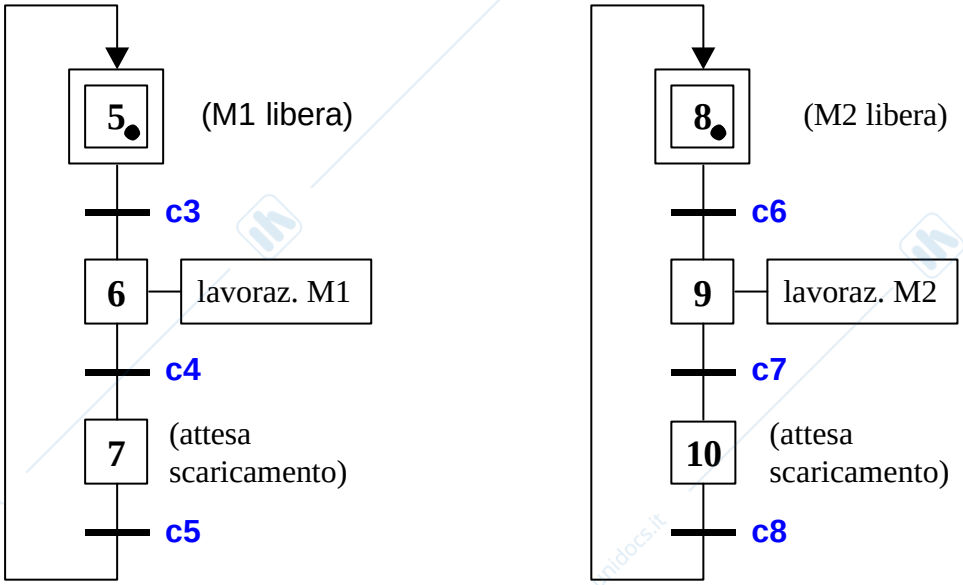
| | R | M1 | C1 |
|---|---|---|---|
| **X1** | A | V | V |
| **X2** | A | V | L |
| **X3** | A | L | V |
| **X4** | A | L | L |
| **X5** | A | P | L |
| **X6** | C | V | V |
| **X7** | C | V | L |
| **X8** | C | V | P |
| **X9** | T | P | V |
| **X10** | S | V | P |
| **X11** | S | L | P |
| **X12** | S | P | P |
| **X13** | A | L | P |




*E) Scrivere un programma SFC modulare che utilizzi al meglio le risorse del sistema di cui al punto 3. Si considerino esplicitamente le disponibilità delle risorse C1 e C2.*

**Soluzione:**






E le transizioni hanno il seguente significato:

- **c1**: C1 presente
- **c2**: C2 presente
- **c3**: fine caricamento M1
- **c4**: fine lavorazione M1
- **c5**: fine scaricamento M1
- **c6**: fine caricamento M2
- **c7**: fine lavorazione M2
- **c8**: fine scaricamento M2