Laboratorio di Automazione Industriale

Francesco Corrini [1067709] Giuseppe Lombardo [1065314]
Mauro Riva [1053644]

A. A. 2021/2022

Indice

Elenco delle figure

1 Esercizio LADDER

1.1 Descrizione dell'approccio risolutivo

Prima di iniziare qualsiasi operazione sul nastro è necessario che il PLC riceva in input dall'utente il numero di componenti da inserire nei box dei kit di assemblaggio. Ottenuto questo numero, alla pressione del pulsante *START* si procede ad alimentare per un istante il contenitore posto sopra il nastro in modo che lasci cadere i componenti. Si attiva poi il nastro per 1 secondo così che i componenti si posizionino esattamente al di sotto della fotocamera. Ai fini dell'esercizio si è supposto che la camera sia sempre accesa e che trasmetta dati in modo continuo al PLC, in questo modo sarà sufficiente memorizzare l'input della camera al momento dello stop del nastro per sapere il numero di componenti effettivamente presenti sul nastro. Se questo numero non coincide con quello richiesto dall'utente inizialmente allora è richiesto l'intervento di un addetto, altrimenti si incrementa il contatore contaUguali e si procede senza interruzioni. Il passo successivo consiste nell'attivare nuovamente il nastro per 1 secondo e terminare il ciclo trasferendo i componenti nei box. A questo punto viene incrementato il contatore dei cicli totali e se ha raggiunto i 100 cicli completati allora il sistema si ferma in attesa di manutenzione, altrimenti si procede al reset delle variabili locali per poi rimanere in attesa della pressione di *START*. Una volta effettuata la manutenzione, qualora prevista, si è deciso di resettare entrambi i contatori anche se non espressamente richiesto dall'esercizio: il modo più logico e utile di usare *contaUguali* è per effettuare una statistica di quanti cicli su 100 si sono dovuti bloccare a causa di un imprecisione del contenitore. Avere a disposizione questo dato può essere utile per massimizzare l'utilizzo del tempo macchina.

1.2 Risoluzione e spiegazione variabili locali

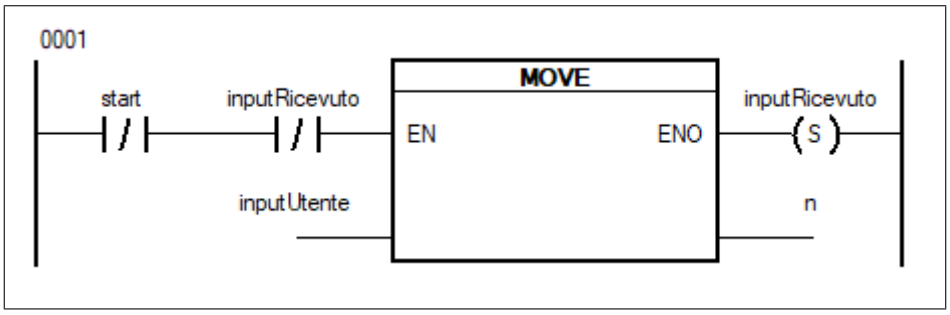

Figura 1: Acquisizione numero N dall'utente

La variabile *inputUtente* viene memorizzata nella variabile locale *n* e rappresenta il numero di componenti che l'utente vuole siano processati nel prossimo ciclo del macchinario. L'utilizzo della variabile *n* e del valore booleano *inputRicevuto* sono necessari per permettere all'utente di modificare il valore di *inputUtente* durante l'esecuzione di un ciclo senza alterare i dati relativi al ciclo in lavorazione.

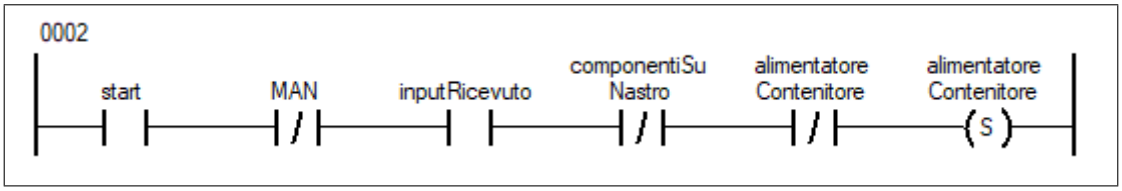

Figura 2: Alimentazione contenitore

alimentatoreContenitore rappresenta l'output che permette al contenitore di lasciare cadere i componenti sul nastro, mentre *componentiSuNastro* è un valore booleano che consente al PLC di sapere se ci sono dei componenti sul nastro.

Figura 3: Caduta componenti sul nastro

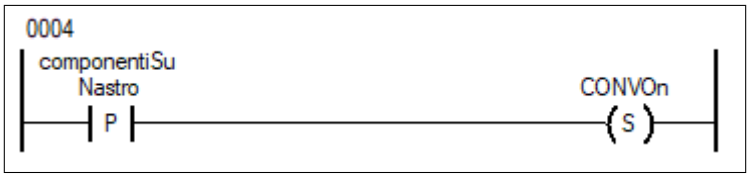

Figura 4: Prima attivazione del nastro

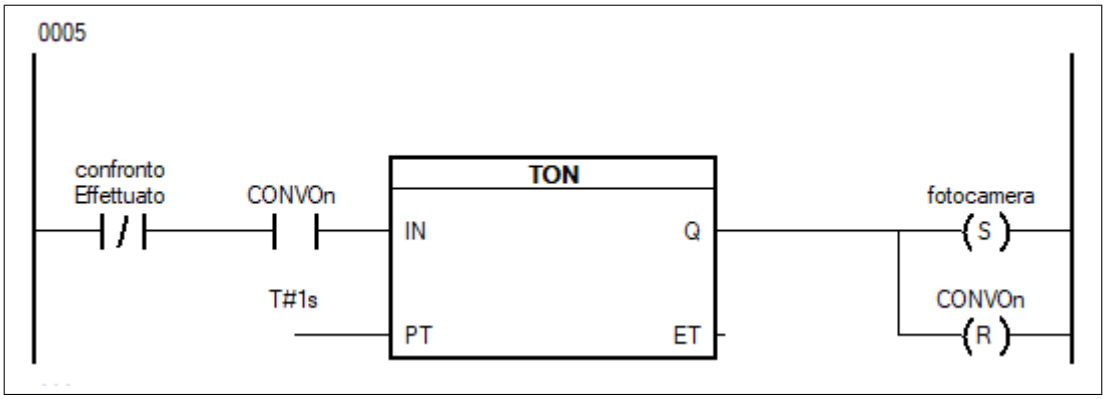

Figura 5: Primo blocco del nastro

fotocamera serve per segnalare al PLC che deve prendere il valore ricevuto in input dalla camera e memorizzarlo in locale, mentre *confrontoEffettuato* è necessario per non far ripetere questa riga quando il nastro riprende a muoversi.

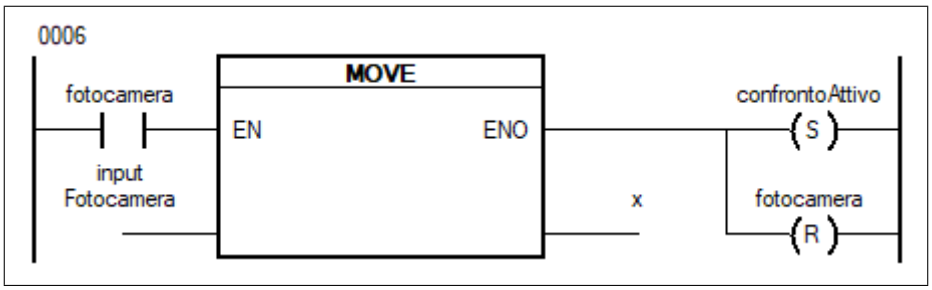

Figura 6: Acquisizione input fotocamera

x è la variabile locale che memorizza quanti componenti sono effettivamente presenti sul nastro, dato che per ipotesi si è supposto che *inputFotocamera* cambi continuamente. *confrontoAttivo* comunica al PLC che è possibile effettuare il confronto tra *n* e *x*.

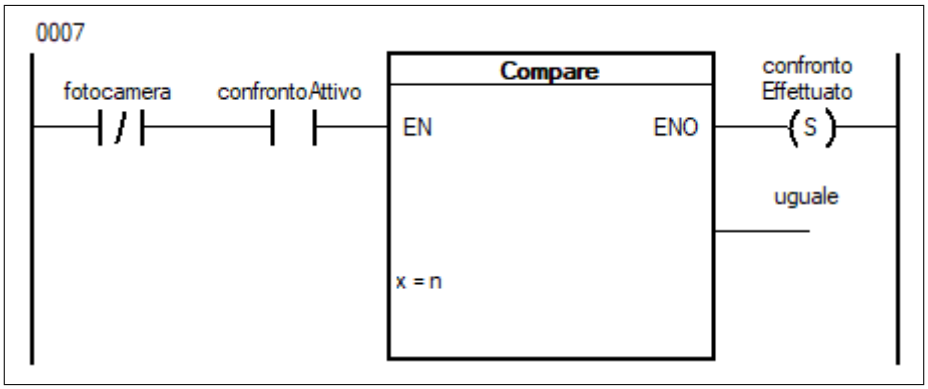

Figura 7: Confronto tra *x* e *n*

uguale è una variabile booleana vera quando x e n sono uguali, altrimenti è falsa.

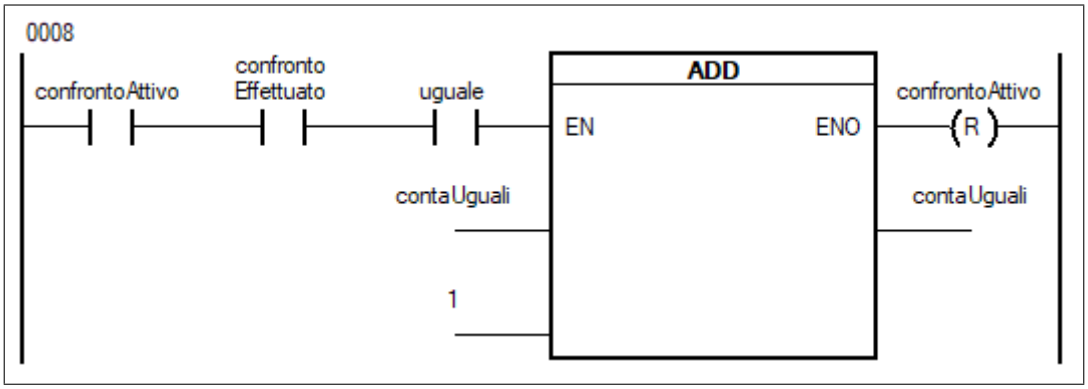

Figura 8: Incremento contatore se numero componenti uguale

contaUguali è il contatore che memorizza quante volte su 100 cicli il numero di componenti presenti sul nastro è uguale a quello richiesto.

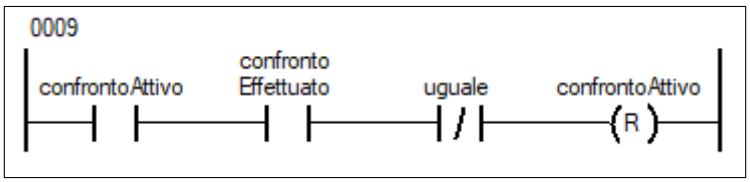

Figura 9: Reset *confrontoAttivo* quando *contaUguali* non deve essere incrementato

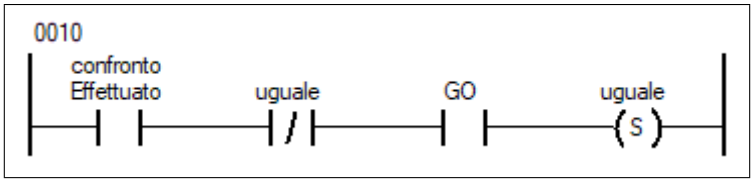

Figura 10: Attesa dell'intervento di un operatore per sistemare i componenti sul nastro

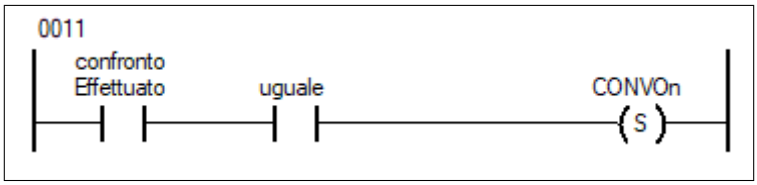

Figura 11: Seconda attivazione del nastro

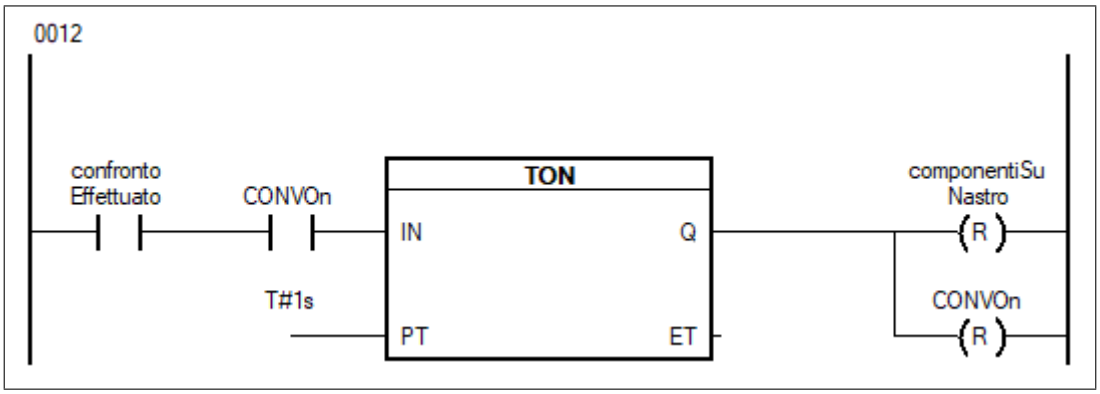

Figura 12: Secondo blocco del nastro

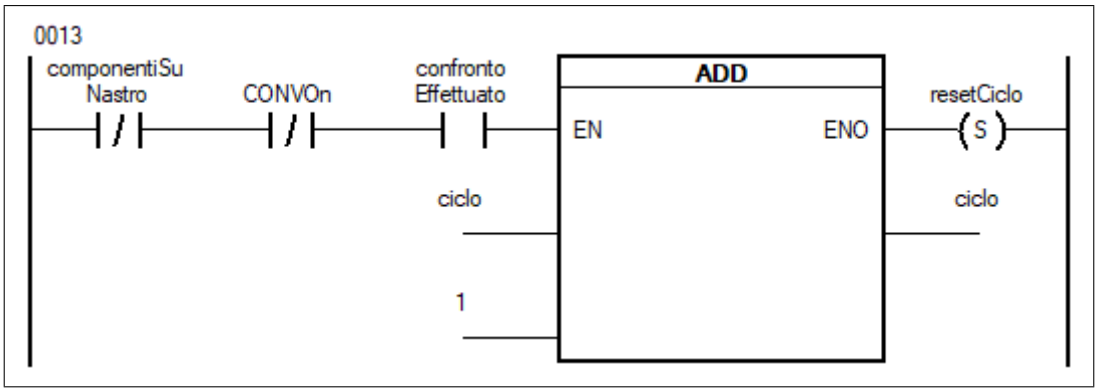

Figura 13: Incremento contatore cicli totali

resetCiclo segnala il termine di un ciclo di lavoro e la necessità di resettare le variabili locali ausiliarie. *ciclo* è il contatore dei cicli completati.

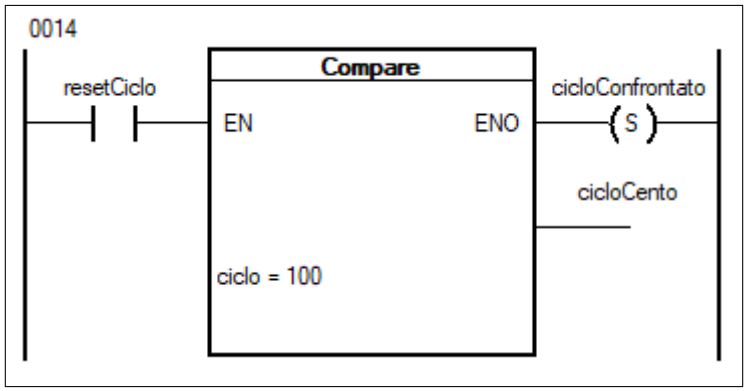

Figura 14: Confronto tra *ciclo e 100*

cicloConfrontato comunica che è stato effettuata l'uguaglianza tra *ciclo* e il numero cento. Il risultato di questa uguaglianza è memorizzato nella variabile *cicloCento*, il cui scopo è di segnalare se è necessaria o meno la manutenzione.

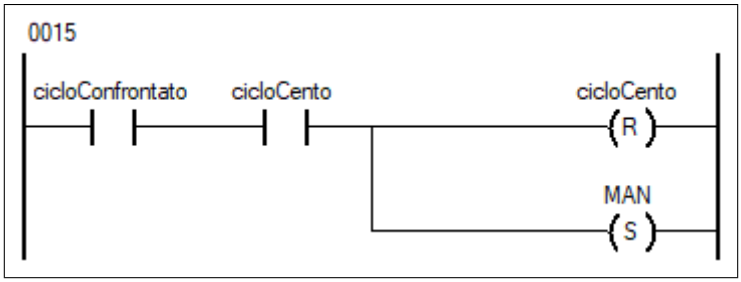

Figura 15: Attivazione richiesta manutenzione

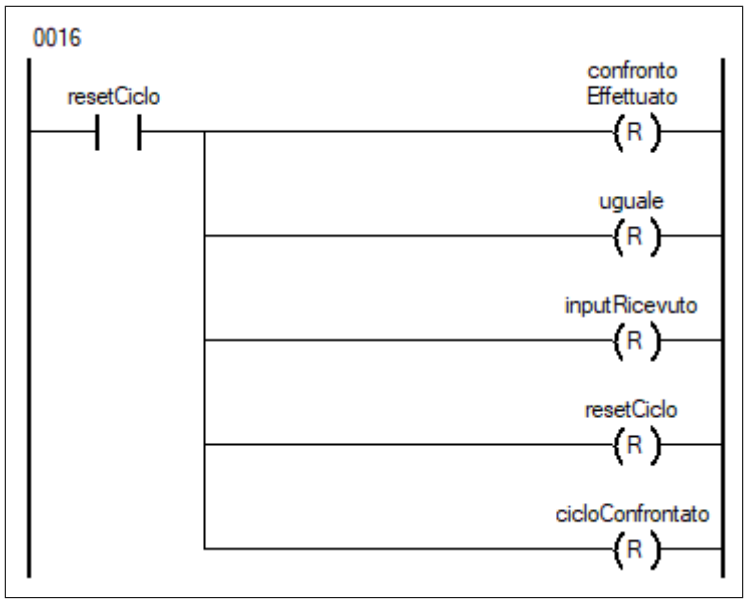

Figura 16: Reset variabili locali ausiliarie

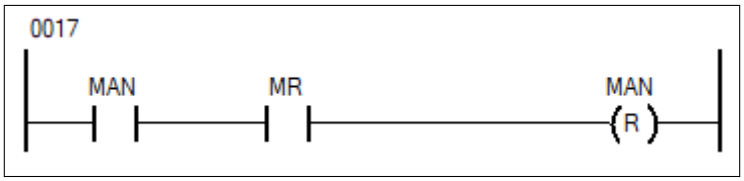

Figura 17: Manutenzione effettuata

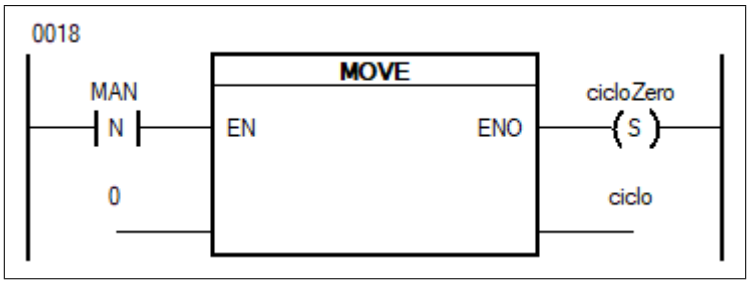

Figura 18: Reset *ciclo* al completamento della manutenzione

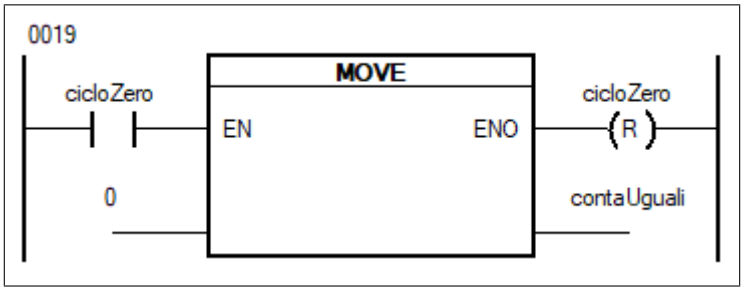

Figura 19: Reset *contaUguali* al completamento della manutenzione

2 Esercizio SFC

2.1 Spiegazione risoluzione

Il programma deve gestire un nastro trasportatore che trasporta un bancale sul quale vengono caricate e scaricate delle bottiglie. Si suppone come stato iniziale del programma *WAIT*: ci si trova in questo stato quando il nastro trasportatore è fermo (*LM* e *RM* settati a FALSE) ed il bancale si trova sul lato sinistro (il sensore *LS* segna TRUE). Se sul bancale sono presenti delle bottiglie un operatore deve rimuoverle. Un sensore *ES* segna TRUE quando il bancale è vuoto ed una variabile *cycles* conta quanti cicli di carica/scarica sono avvenute. Si esce da *WAIT* quando l'operatore preme un tasto *START*, *ES* segna TRUE e *cycles* è minore di 1000. Se quest'ultima fosse maggiore, si andrebbe nello stato *ALARM*: in questo stato l'allarme *MAI* è settato a TRUE e si esce dallo stato solo quando, dopo aver effettuato delle operazioni di manutenzione, un operaio preme il pulsante *MAIR* e si riporta il programma in *WAIT*. Uscendo dallo stato viene resettato il contatore con una exit action scritta in testo strutturato.

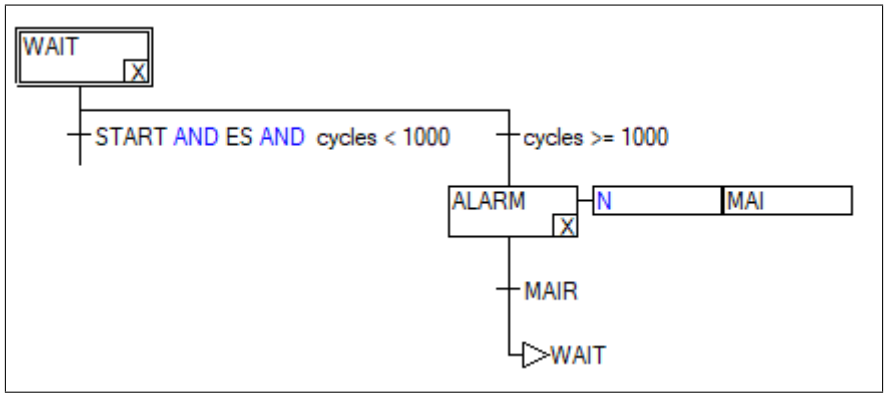

Figura 20: Ramo di attivazione dell'allarme, disattivazione da parte dell'operatore e reset del contatore

Quando invece viene attivata la transizione *START*, *ES* e *cycles* < 1000 si esegue la exit action (in testo strutturato) che incrementa *cycles* di uno e si passa nello stato *RIGHTM*: in questo stato *RM* (motore che muove il nastro verso destra) è settato a TRUE. Si esce dallo stato quando il sensore *RS* (che segnala il margine destro del nastro) segna TRUE e si entra nello stato *LOAD*. Qui un macchinario carica delle bottiglie sul bancale e si ferma quando la fotocellula *CELL* segnala TRUE, che fa scattare la transizione di uscita verso lo stato *SECURITY*. Infatti, per motivi di sicurezza, dopo che le bottiglie sono state caricate il programma deve attendere cinque secondi prima di riprendere. Trascorso quel tempo si esce da *SECURITY* e si entra in *LEFTM*, nel quale è attivo *LM* (motore sinistro del nastro). Si rimane in questo stato fino a che *LS* (il quale segnala che il bancale ha

raggiunto il margine sinistro) non diventa TRUE ed il programma si riporta in *WAIT* dove ricomincia il ciclo.

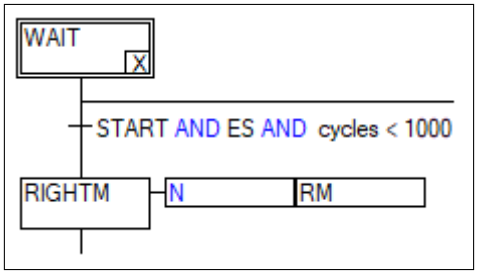

Figura 21: Uscita dallo stato di attesa ed inizio di un normale ciclo

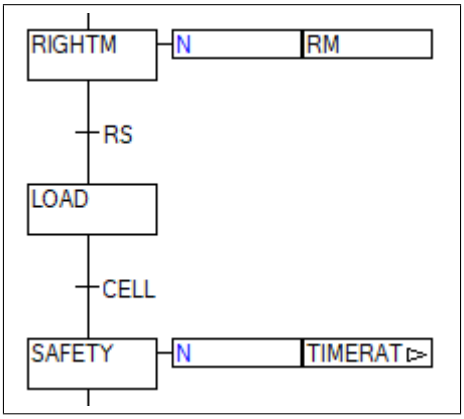

Figura 22: Arrivo del bancale sulla destra, caricamento delle bottiglie ed attesa di sicurezza

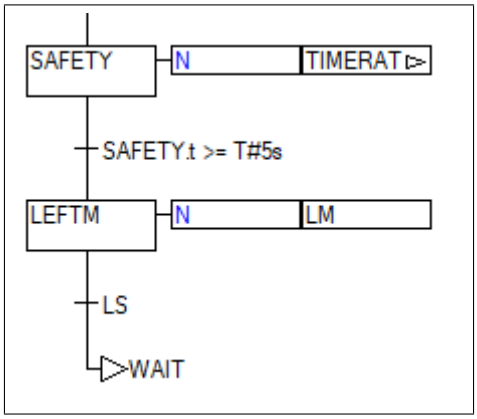

Figura 23: Fine dell'attesa e ritorno in attesa sul lato sinistro

L'unica variabile introdotta, oltre a *cycles* per contare i cicli, è *TIMERATTIVO*, una azione "fittizia" eseguita nello stato *SAFETY* nel quale in realtà non deve essere eseguito niente, ma bisogna solo attendere cinque secondi. Le altre variabili sono tutte collegati agli input ed output del sistema.

3 Esercizio ST

3.1 Descrizione dell'approccio risolutivo

Per la risoluzione di questo esercizio in testo strutturato si è deciso di procedere con un approccio simile al LADDER. All'avvio il treno si trova nella stazione 1, ha le porte aperte e non ci sono allarmi attivi. Dopo aver aspettato 30 secondi le porte provano a chiudersi: se c'è un ostacolo verranno effettuati altri 2 tentativi di chiusura distanziati tra di loro di 10 secondi. In caso di insuccesso anche nell'ultimo tentativo scatta un allarme e il treno resta fermo fino al reset dell'allarme da parte del capostazione. Alla chiusura delle porte, se non ci sono allarmi attivi e se non serve effettuare manutenzione, il treno inizia a muoversi verso la prossima stazione secondo il valore della variabile *Direzione* (ottenuta dal function block *Movimento*), che può essere TRUE se il treno deve muoversi in avanti o FALSE se il movimento è a ritroso. All'ingresso del treno in una delle 3 stazioni, segnalato da uno dei sensori di STOP, il motore viene fermato e si aspettano 5 secondi prima di aprire le porte. Ai fini dell'esercizio si è esclusa a priori la possibilità che i sensori di STOP possano essere attivati da oggetti che non siano il treno (cosa che potrebbe succedere se il sensore è una semplice fotocellula) e quindi non sono stati implementati controlli tra la fermata verso la quale il treno sta viaggiando e il numero del sensore che si è attivato. Ogni volta che il treno si trova nella prima stazione il contantore *Ciclo*, inizializzato a 0, viene incrementato di uno. La manutenzione viene richiesta quando questo contatore è maggiore di 100, in quanto al termine del centesimo ciclo avrà valore 101. A questo punto il treno resta in attesa fino alla pressione del pulsante *MAIR* che resetterà l'allarme di manutenzione *MAI*.

3.2 Spiegazione variabili locali

- *Ciclo*: contatore dei giri completati dal treno, è inizializzato a 0 e resettato quando è maggiore di 100;
- *TimerAperturaPorte*: valore booleano usato per indicare quando si è in attesa dei 5 secondi prima dell'apertura delle porte in seguito alla fermata del treno in una stazione;
- *TON_0*: timer di 5 secondi per l'apertura delle porte;
- *AperturaPorte*: variabile booleana che rappresenta lo stato delle porte: TRUE quando sono aperte, FALSE quando sono chiuse;
- *Ostacolo*: booleano che segnala quando la chiusura delle porte è stata bloccata da un ostacolo;
- *TON_1*: timer di 30 secondi per la chiusura delle porte;
- *TON_2* e *TON_3*: timer di 10 secondi per la chiusura delle porte in caso di ostacolo;
- *AvvioMovimento*: variabile che indica l'avvenuta chiusura delle porte;
- *Fermata*: rappresenta il numero dell'ultima stazione visitata dal treno;
- *Direzione*: valore booleano impiegato per indicare in quale direzione si deve muovere il treno: TRUE in avanti, FALSE all'indietro;
- *Movimento_0*: oggetto del function block *Movimento*;

3.3 PROGRAM _INIT

```
1 PROGRAM _INIT
2   Ciclo := 0;
3   Fermata := 1;
4   Direzione := TRUE;
5   AperturaPorte := TRUE;
6 END_PROGRAM
```

Codice 1: Inizializzazione variabili

All'avvio avremo *Ciclo* inizializzato a 0 in modo che durante il primo giro abbia valore 1, *Fermata* a 1 poiché il treno si trova nella prima stazione, *Direzione* a TRUE perché deve muoversi in avanti e *AperturaPorte* a TRUE per permettere ai passeggeri di salire prima di partire verso la seconda fermata.

3.4 FUNCTION_BLOCK *Movimento*

```
1 FUNCTION_BLOCK Movimento
2   CASE F OF
3     1: Stazione := 2; DMoto := TRUE;
4     2: IF(Dir) THEN Stazione := 3; DMoto := TRUE; ELSE Stazione := 1;
          DMoto := FALSE; END_IF
5     3: Stazione := 2; DMoto := FALSE;
6   END_CASE
7 END_FUNCTION_BLOCK
```

Codice 2: Function block

A questa funzione vengono passati 2 parametri: *F* che rappresenta l'ultima stazione visitata dal treno e *Dir* che indica in quale direzione sta viaggiando. Quando il treno si trova nella stazione 1 o nella stazione 3 la direzione del movimento è obbligata, altrimenti è necessario restituire il valore di *Dir* al programma chiamante. Il function block restituisce la variabile intera *Stazione* e la variabile booleana *DMoto*.

3.5 PROGRAM _CYCLIC

```
PROGRAM _CYCLIC

  IF(RESET_ALARM) THEN
    ALARM := FALSE;
  END_IF

  IF(MAIR) THEN
    MAI := FALSE;
  END_IF

  IF(Ciclo > 100) THEN
    Ciclo := 0;
    MAI := TRUE;
  END_IF

  IF(MOT_FW OR MOT_BW) THEN
    IF(EDGEPOS(STOP_1) OR EDGEPOS(STOP_2) OR EDGEPOS(STOP_3)) THEN
      MOT_FW := FALSE;
      MOT_BW := FALSE;
      TimerAperturaPorte := TRUE;
    END_IF
  END_IF

  IF(TimerAperturaPorte) THEN
    TON_0(IN := TimerAperturaPorte, PT := T#5s);
    IF(EDGEPOS(TON_0.Q)) THEN
      TimerAperturaPorte := FALSE;
      AperturaPorte := TRUE;
      TON_0(IN := FALSE, PT := T#5s);
    END_IF
  END_IF

  IF(AperturaPorte AND NOT Ostacolo AND NOT ALARM AND NOT MAI) THEN
    TON_1(IN := AperturaPorte, PT := T#30s);
    IF(EDGEPOS(TON_1.Q) AND NOT OBS) THEN
      AperturaPorte := FALSE;
      TON_1(IN := FALSE, PT := T#30s);
    ELSIF(EDGEPOS(TON_1.Q) AND OBS) THEN
      Ostacolo := TRUE;
      TON_1(IN := FALSE, PT := T#30s);
    END_IF
  END_IF

```

```
44    IF(Ostacolo) THEN
45      TON_2(IN := Ostacolo, PT := T#10s);
46      IF(TON_2.Q AND NOT OBS) THEN
47        AperturaPorte := FALSE;
48        Ostacolo := FALSE;
49        TON_2(IN := FALSE, PT := T#10s);
50      ELSIF(TON_2.Q AND OBS) THEN
51        TON_3(IN := Ostacolo, PT := T#10s);
52        IF(TON_3.Q AND NOT OBS) THEN
53          AperturaPorte := FALSE;
54          Ostacolo := FALSE;
55          TON_3(IN := FALSE, PT := T#10s);
56          TON_2(IN := FALSE, PT := T#10s);
57        ELSIF(TON_3.Q AND OBS) THEN
58          Ostacolo := FALSE;
59          ALARM := TRUE;
60          TON_3(IN := FALSE, PT := T#10s);
61          TON_2(IN := FALSE, PT := T#10s);
62        END_IF
63      END_IF
64    END_IF
65
66    IF(EDGENEG(AperturaPorte)) THEN
67      AvvioMovimento := TRUE;
68    END_IF
69
70    IF(AvvioMovimento AND NOT MAI AND NOT ALARM) THEN
71      IF(Fermata = 1) THEN
72        Ciclo := Ciclo + 1;
73      END_IF
74      Movimento_0(F := Fermata, Dir := Direzione);
75      Direzione := Movimento_0.DMoto;
76      Fermata := Movimento_0.Stazione;
77      IF(Direzione) THEN
78        MOT_FW := TRUE;
79        MOT_BW := FALSE;
80      ELSE
81        MOT_BW := TRUE;
82        MOT_FW := FALSE;
83      END_IF
84      AvvioMovimento := FALSE;
85    END_IF
86  END_PROGRAM
```

Codice 3: Programma ciclico