UKŁADY SEKWENCYJNE

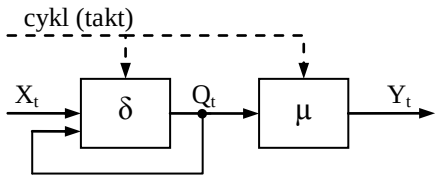
Wprowadzenie. Napełnianie i opróżnianie. Programowanie. Zbiornik z trzema zaworami. Układ Start–Stop. Podnośnik góra–dół.

WPROWADZENIE

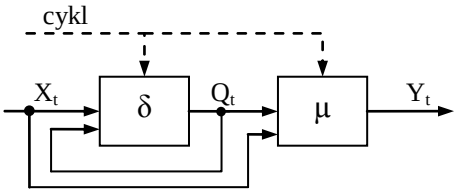
1. Automaty Moore'a i Mealy'ego

Układy sekwencyjne nazywa się zwykle automatami.

Automat Moore'a



Automat Mealy'ego



Funkcja przejścia: $\begin{cases} Q_{t+1} = \delta(Q_t, X_t) \\ Y_t = \mu(Q_t) \end{cases}$ $\begin{cases} Q_{t+1} = \delta(Q_t, X_t) \\ Y_t = \mu(Q_t, X_t) \end{cases}$

Funkcja wyjścia:

W automacie Moore'a wyjście zależy wyłącznie od stanu wewnętrznego Q (głównie takie automaty będą rozpatrywane). Stan wewnętrzny jest reprezentowany przez pamięć. Zmienna *stan* występuje w typowych realizacjach programowych układów sekwencyjnych.

2. Automaty synchroniczne i asynchroniczne

Automat synchroniczny – zmiany stanu następują w ściśle określonych momentach czasu (w automatach asynchronicznych tak nie jest).

Realizacje programowe układów sekwencyjnych, zwłaszcza złożonych, są zwykle automatami synchronicznymi. Automaty asynchroniczne są prostsze i dopuszczają zmiany cyklu (tzn. czasu wykonania programu).

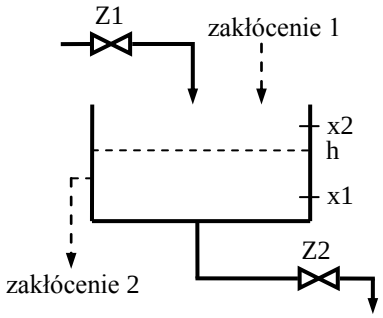
3. Rozpoznanie układu kombinacyjnego i sekwencyjnego

Jeżeli na podstawie obserwacji aktualnych wejść nie da się wprost stwierdzić, co należy uczynić, to mamy do czynienia z układem sekwencyjnym, tzn. z pamięcią (nie można podjąć decyzji znając tylko aktualne wejścia; trzeba rozważyć, co zdarzyło się wcześniej).

NAPEŁNIANIE I OPRÓŻNIANIE

1. Problem

Chodzi o cykliczne, naprzemienne napełnianie i opróżnianie zbiornika za pomocą zaworów Z1, Z2.



h, x1, x2 – poziomy: aktualny, dolny, górny
Z1, Z2 – zawory: wlewowy, spustowy

Specyfikacja (algorytm):

- h < x1 → otwórz Z1, zamknij Z2
- h > x2 → zamknij Z1, otwórz Z2

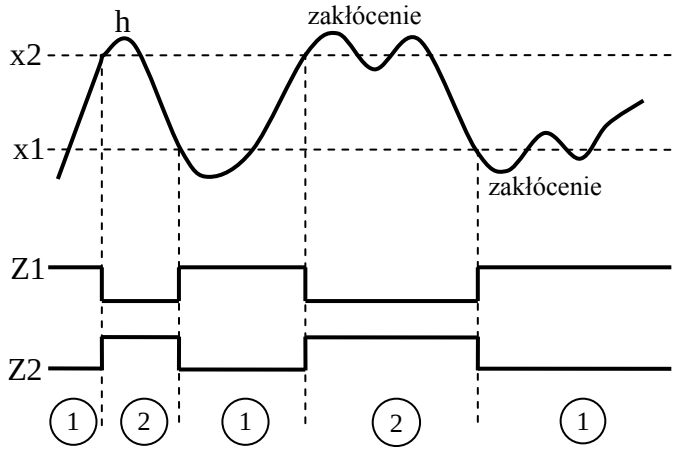
Pytanie. Co uczynić przy poziomie pośrednim?
Ponieważ odpowiedź *nie jest* natychmiastowa, bo potrzebne jest rozważenie stanu poprzedniego, więc mamy do czynienia z układem sekwencyjnym.

2. Projektowanie systematyczne

1. Przebiegi czasowe
2. Definiowane stanów automatu
3. Graf stanów
4. Program: C, ST, LD

3. Przebiegi czasowe

Kreśli się je w uzgodnieniu z formułującym zadanie.



4. Definiowane stanów

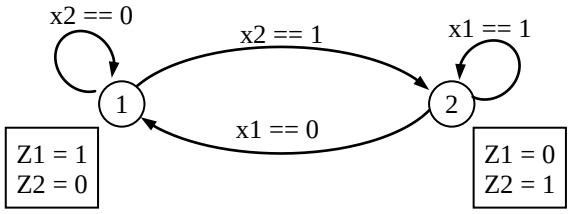
Należy zwracać uwagę na te momenty, gdy automat ma zareagować, tzn. zmienić wyjście (sterowanie). Pomiędzy tymi momentami mamy do czynienia ze stanami stabilnymi, które należy odpowiednio nazwać. Są one podstawą do dalszej analizy.

(1) – napełnianie, (2) – opróżnianie

Stany stabilne zaznacza się na przebiegu (zob. rysunek).

5. Graf stanów

Konstruowanie: 1) stany – małe kółka z nazwami
2) przejścia – zorientowane strzałki z opisem warunku przejścia z tablicy
3) wyjścia – prostokąty z wartościami odpowiednio do stanu
4) pętle pozostawania w stanie – opis warunku (opcjonalnie).



Uwaga. Opisy grafu odpowiadają instrukcjom języka C.

PROGRAMOWANIE

1. Język C

- **Instrukcja *switch***

  Jest to podstawowa instrukcja do programowania sekwencji. Kod programu jest bezpośrednim odzwierciedleniem grafu automatu. Warunki zapisane przy pętlach nie występują w programie (chodzi o pozostawanie w stanie).

```
char x1,x2,Z1,Z2;
char stan=1;
...
switch(stan)
   {
    case 1: Z1=1; Z2=0;
            if(x2==1) stan=2;
            break;
    case 2: Z1=0; Z2=1;
            if(x1==0) stan=1;
   }
```

- **if … else**

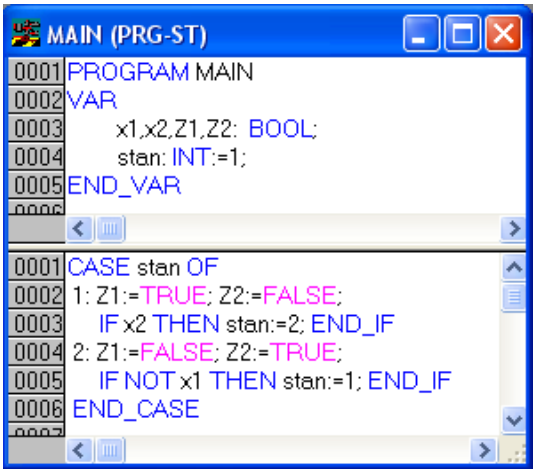
```
if(stan==1)
     { Z1=1; Z2=0; if(x2) stan=2; }
else
if(stan==2)
     { Z1=0; Z2=1; if(!x1) stan=1; }
```

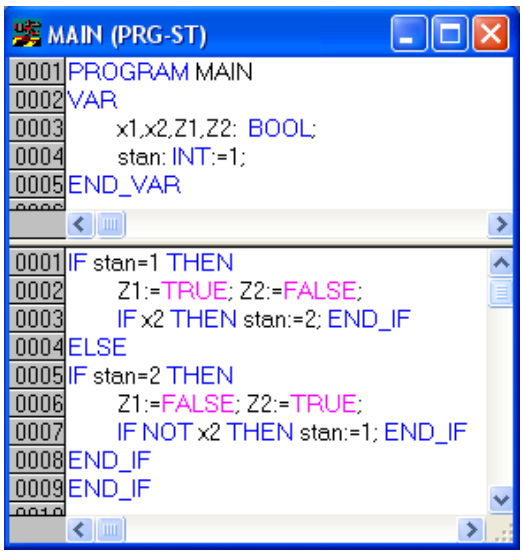
Uwaga. else jest potrzebne, aby w danym cyklu wykonać tylko jedną instrukcję *if*.

Rozwiązanie *if ... else* jest stosowane w prostych językach skryptowych, w których brak odpowiednika instrukcji *switch*.

2. Język ST

Odpowiednikiem instrukcji *switch* jest CASE … OF





3. Niepoprawne pomiary

Wymaganie technologiczne (przykładowe). W przypadku niepoprawnych pomiarów obydwa zawory należy zamknąć.

x2 \ x1	0	1
0	1	1
1	0	1

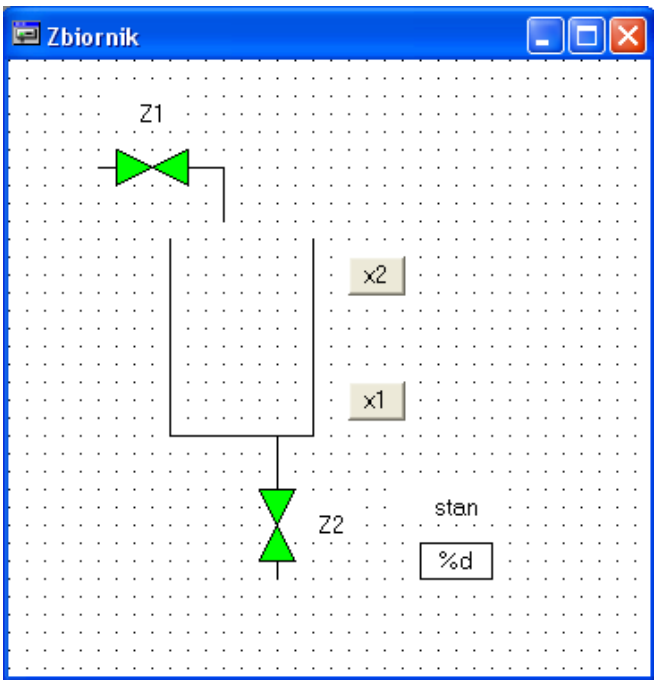
niepoprawność (zero) $\rightarrow x2 \cdot \overline{x1}$

```
MAIN (PRG-ST)
0001 PROGRAM MAIN
0002 VAR
0003     x1,x2,Z1,Z2: BOOL;   stan: INT:=1;
0004 END_VAR

0001 IF x2 AND NOT x1 THEN
0002     Z1:=FALSE; Z2:=FALSE;
0003 ELSE
0004     CASE stan OF
0005     1: Z1:=TRUE; Z2:=FALSE;
0006        IF x2 THEN stan:=2; END_IF
0007     2: Z1:=FALSE; Z2:=TRUE;
0008        IF NOT x1 THEN stan:=1; END_IF
0009     END_CASE
0010 END_IF
```

4. Prosta wizualizacja

- Ikona *Visualization* (na dole) > eksplorator *Visualizations*, menu *Add Object* > *Name* … Zbiornik
- Docelowy obraz



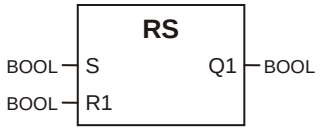
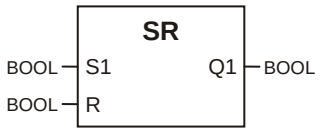
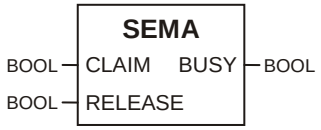
- Elementy
  Zawory: *Polygon*, *Variables – Change color*, *MAIN.Z1/Z2*
  Przyciski: *Text*, *Input – Toggle variable*, *MAIN.x1/x2*, *Colors* – zielony/czerwony
  Wyświetlacz stanu: *Rectangle*, *Text – %d* (format), *Variables – Text display*, *MAIN.stan*
  Napisy: *Rectangle*, *Text* – Z1, Z2, stan, *Colors – No frame color*.
- Porównać wizualizację układów:
  – nie uwzględniającego możliwych niepoprawnych pomiarów
  – zamykającego zawory przy niepoprawnych pomiarach.

PRZERZUTNIK RS

1. Bloki dwustanowe normy PN–EN 61131–3

Oznaczenia wejść/wyjść na poniższych schematach i w opisach są następujące:
Q – wyjście typu BOOL,
R – wejście zerowania logicznego (reset),
S – wejście ustawiające (set),

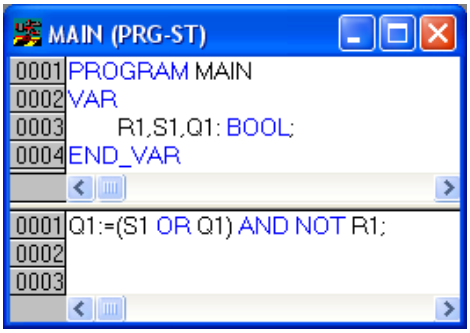
- Wartości początkowe wszystkich wejść są zerowe.

Bloki dwustanowe	
	**RS** – przerzutnik typu RS (*RS flip-flop*) Q1 = NOT R1 AND ($Q1_{n-1}$ OR S)
	**SR** – przerzutnik typu SR (*SR flip-flop*) Q1 =S1 OR (NOT R AND $Q1_{n-1}$)
	**SEMA** – semafor (*semaphore*) BUSY = TRUE dla CLAIM=TRUE BUSY = FALSE dla RELEASE=TRUE i CLAIM=FALSE

SEMA – głównie stosowany do definiowania dostępu do zasobów system operacyjnego

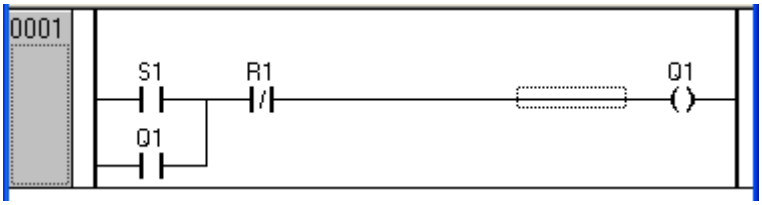
2. Realizacja przerzutnika RS według wzoru z normy

- ST



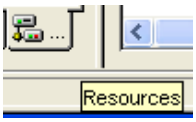
RS – układ podtrzymujący (zapamiętujący) przełącznik załącz/wyłącz

- LD

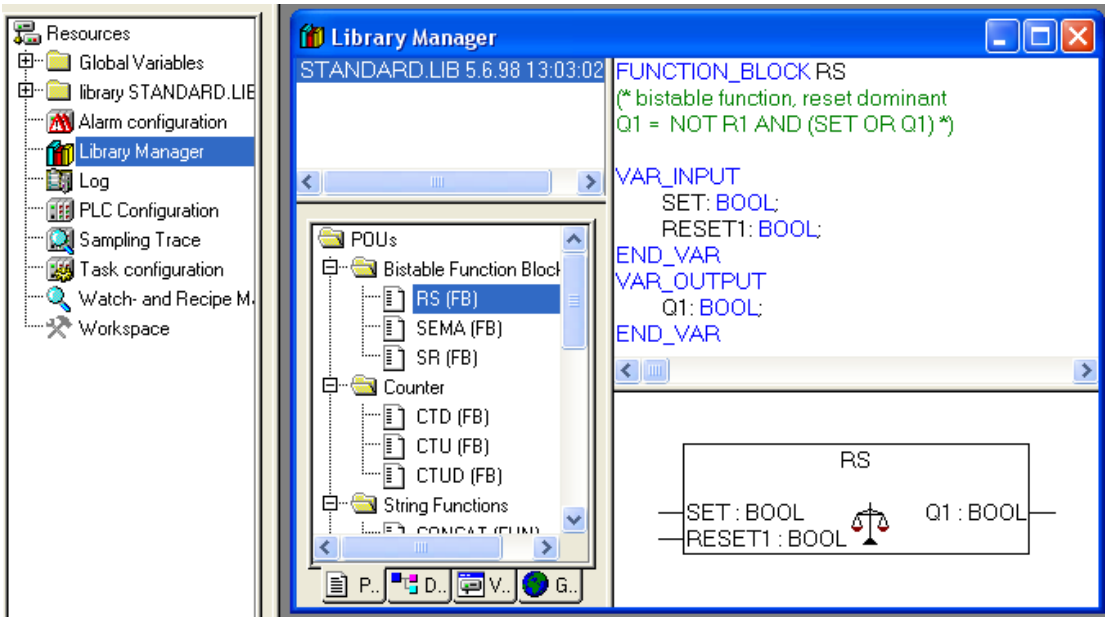


3. **Biblioteczny blok RS** (TwinCAT)

- Zasoby projektu – *Resources* (prawa dolna ikona pod eksploratorem)



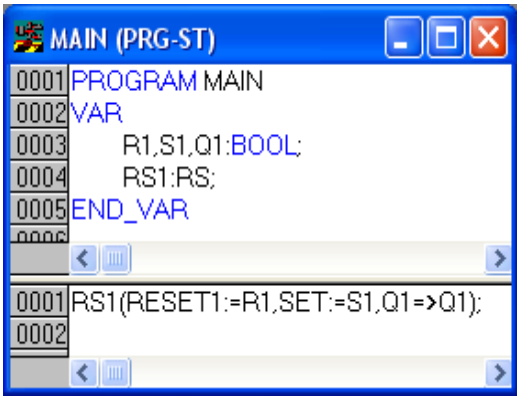
- *Library Manger* > *RS*(*FB*)



Wejścia/wyjścia: SET, RESET1, Q1

4. **Blok RS w programie**

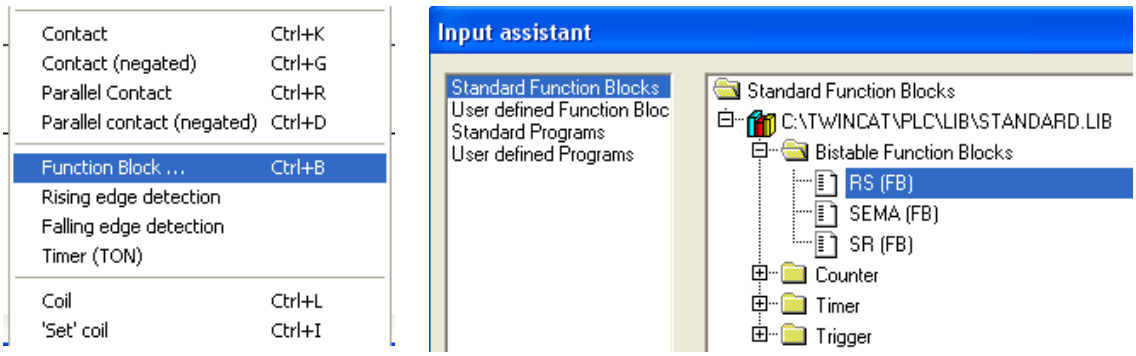
- ST

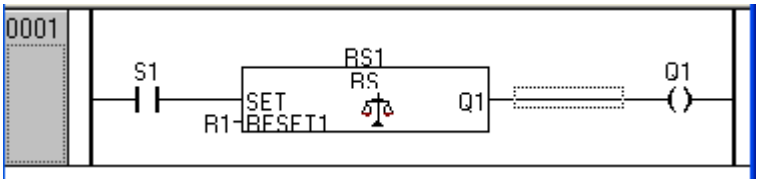


Deklaracje –	zmienne wejściowe i wyjściowe
–	instancja bloku (rezerwacja pamięci)

- LD

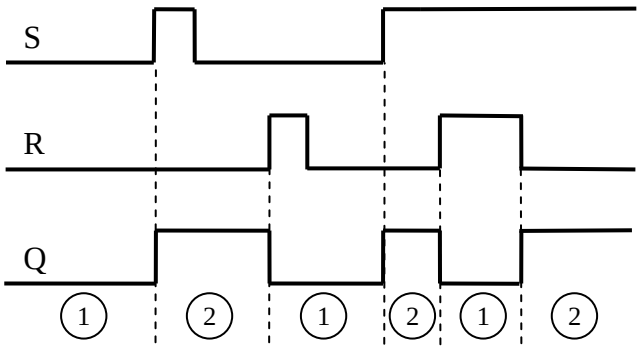
  Menu *Function Block…* lub ikona (górny pasek) > Okno *Input assistant* > *RS*(*FB*)





5. Program RS jako automat ST

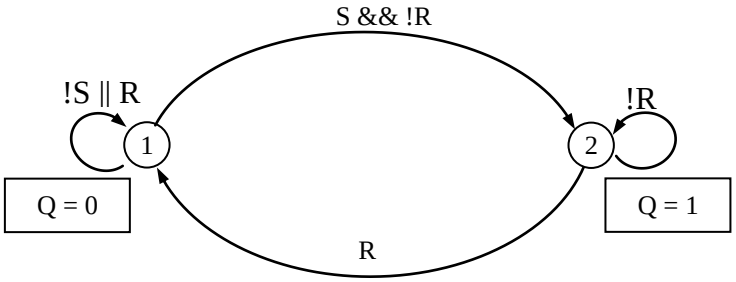
- Przebiegi czasowe



- Stany

(1) – wyzerowany (2) – ustawiony

- Automat



Oznaczenia jak w języku C.

- ST

```
MAIN (PRG-ST)
0001 PROGRAM MAIN
0002 VAR
0003     R1,S1,Q1:BOOL;
0004     stan: INT:=1;
0005 END_VAR

0001 CASE stan OF
0002 1: Q1:=FALSE;
0003   IF S1 AND NOT R1 THEN stan:=2; END_IF
0004 2: Q1:=TRUE;
0005   IF R1 THEN stan:=1; END_IF
0006 END_CASE
```

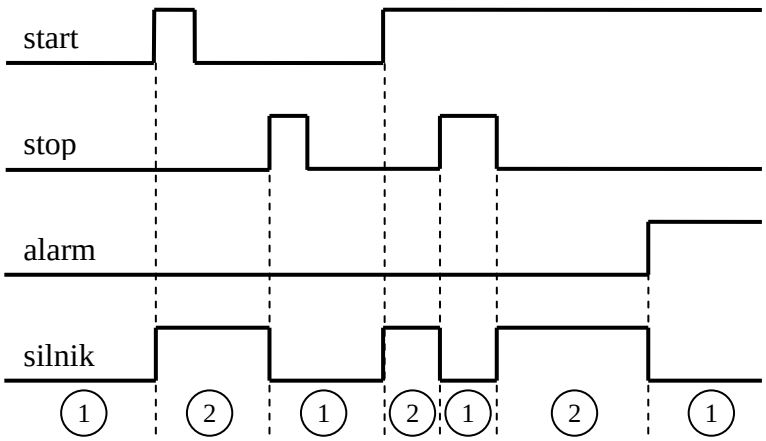
UKŁAD START–STOP

1. Problem

Silnik jest załączany przyciskiem *start*, a wyłączany przyciskiem *stop*. Raz włączony *silnik* pracuje, aż do przyciśnięcia *stop* (pomimo zwolnienia *start*). *Stop* ma priorytet nad *start*, tzn. jednoczesne naciśnięcie obydwu przycisków nie uruchamia *silnika*. Dostępny jest ponadto sygnał *alarm* działający tak jak *stop*, tzn. gdy jest on ustawiony, to *silnika* nie można uruchomić.

Wyjaśnienie. Zwykle *alarm* pochodzi z termicznego zabezpieczenia silnika.

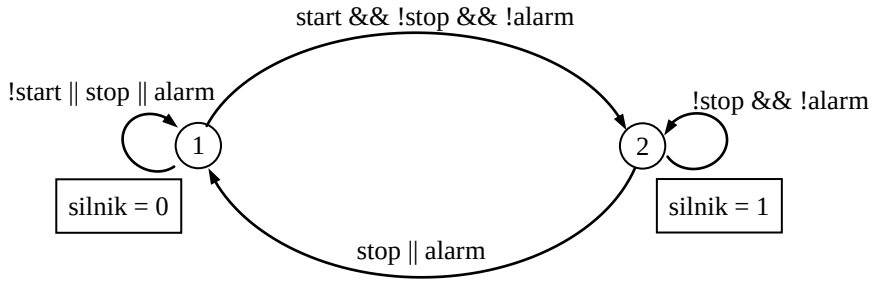
2. Przebiegi czasowe



3. Stany

(1) – zatrzymanie (2) – praca

4. Automat



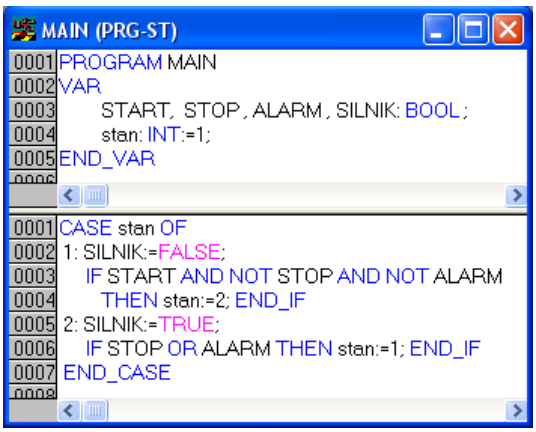
Oznaczenia jak w języku C.

5. Programy

- **C**

```
char start, stop, alarm, silnik;
char stan=1;
   ...
switch(stan)
   {
    case 1: silnik=0;
            if(start&&!stop&&!alarm) stan=2;
            break;
    case 2: silnik=1;
            if(stop||alarm) stan=1;
   }
```

- **ST**



6. Wzór bezpośredni

- Wzór zastępuje automat

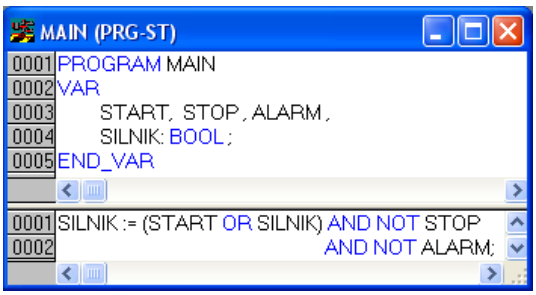
$$silnik_i = (start_i + silnik_{i-1}) \cdot \overline{stop_i} \cdot \overline{alarm_i}$$

gdzie $silnik_{i-1}$ jest wartością zmiennej z poprzedniego cyklu obliczeniowego (i-1). Pozostałe oznaczenia dotyczą cyklu aktualnego (i).

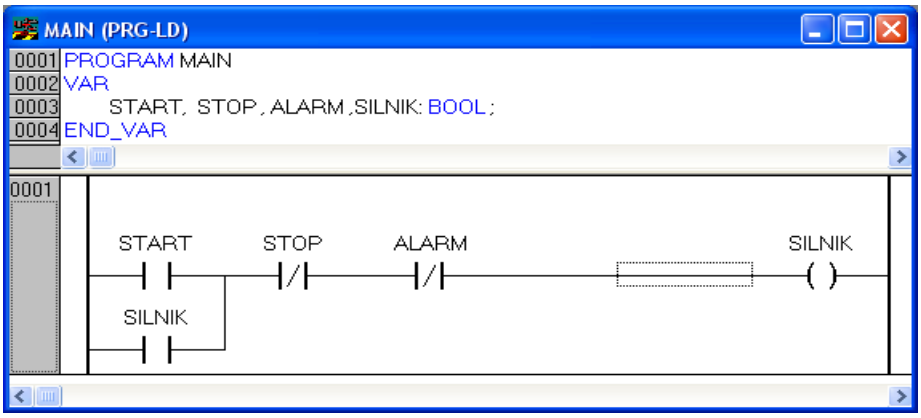
- **C**

```
silnik = (silnik||start)&&!stop&&!alarm;
```

- **ST**

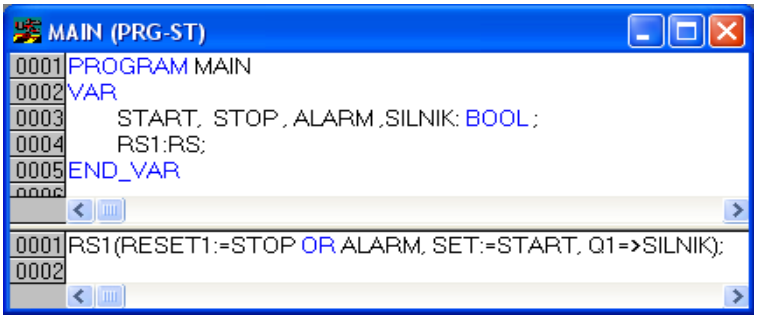


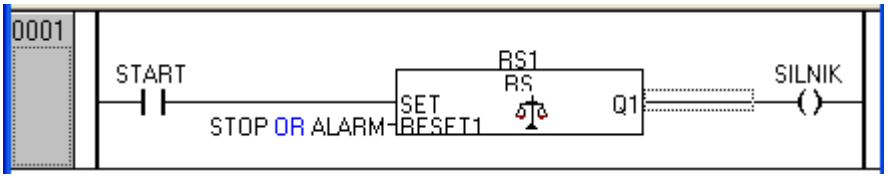
- **LD**



7. Start–Stop z przerzutnikiem RS

- Wzór opisujący układ Start–Stop (powyżej) jest prostym rozszerzeniem wzoru dla przerzutnika RS.
- ST i LD





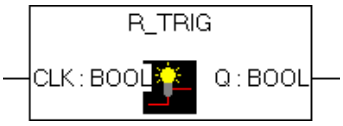
AUTOMATY W JĘZYKU LD

1. Detektory zbocza – PN–EN 61131–3

Detektory zbocza	
R_TRIG BOOL – CLK Q – BOOL	**R_TRIG** – detektor zbocza narastającego (*rising edge detector*) Q = ↑↓ dla CLK↑
F_TRIG BOOL – CLK Q – BOOL	**F_TRIG** – detektor zbocza opadającego (*falling edge detector*) Q = ↑↓ dla CLK↓

CLK↑ – narastające zbocze wejścia CLK; CLK↓ – zbocze opadające.

- R_TRIG w TwinCAT (*Library Manager*)



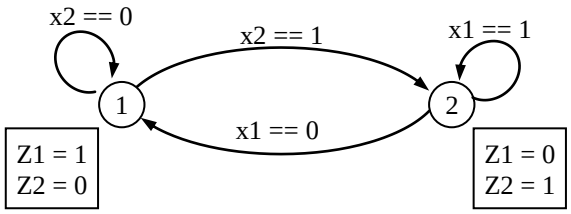
Bloki R_TRIG, F_TRIG służą do generacji pojedynczych impulsów, np. przejść między stanami automatu.

2. Zasady programowania automatów

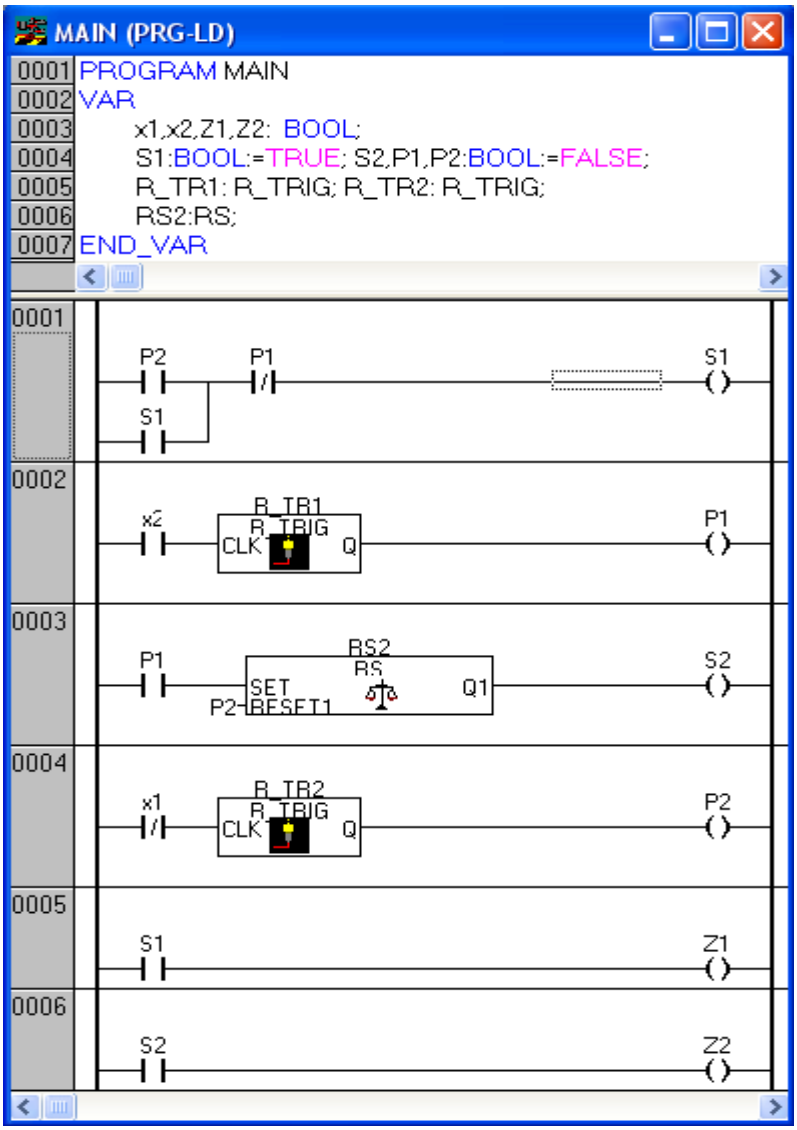
- Stany automatu są reprezentowane przez zmienne S1, S2, … (BOOL).
- Przejścia między stanami reprezentują zmienne P1, P2, … (BOOL).
- Szczebel stanu S jest układem z podtrzymaniem (zapamiętaniem) lub przerzutnikiem RS ustawianym przez przejście dochodzące (lub przejścia), a zerowanym przez przejście wychodzące.
- Szczebel przejścia P zawiera warunek przejścia (z grafu automatu), stan S, z którego przejście wychodzi oraz blok R_TRIG generujący impuls.
- Wyjście sterujące jest równe logicznej sumie stanów S (połączenie równoległe), w których wyjście to ma być ustawione (na TRUE).

3. Napełnianie i opróżnianie

- Automat



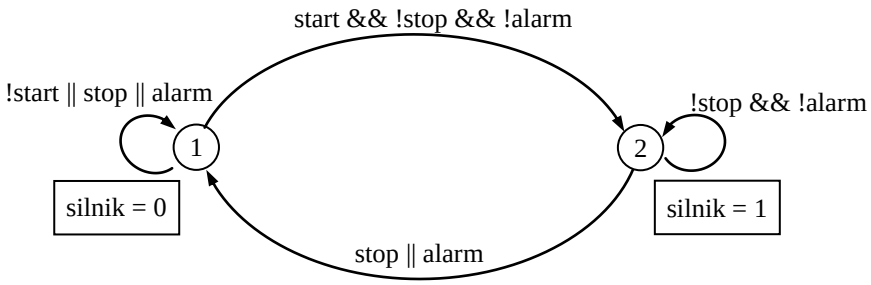
- LD



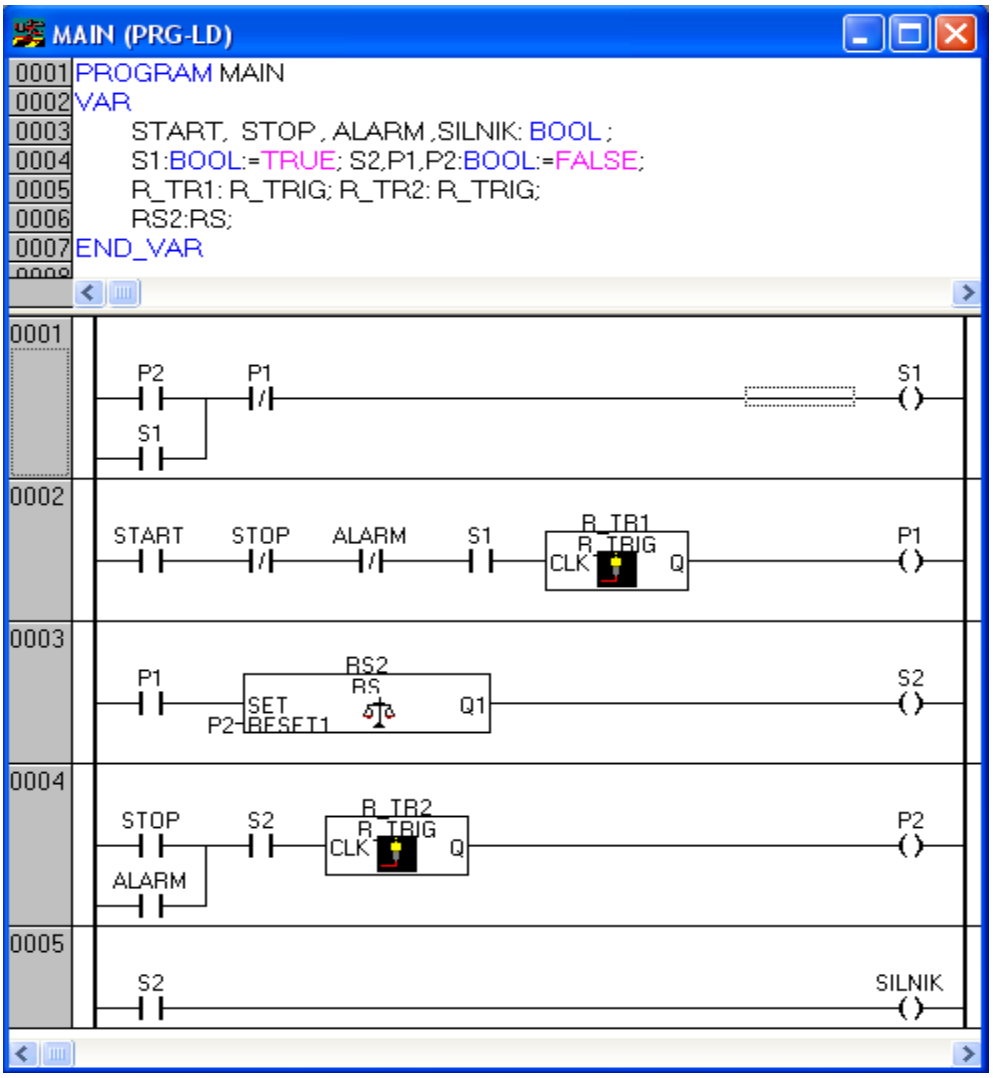
Szczeble S1, S2 zrealizowano alternatywnie.

4. Układ Start–Stop

- Automat



- LD

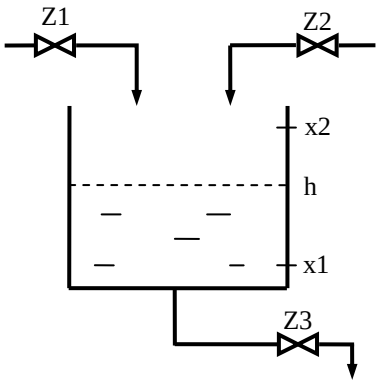


ZBIORNIK Z TRZEMA ZAWORAMI

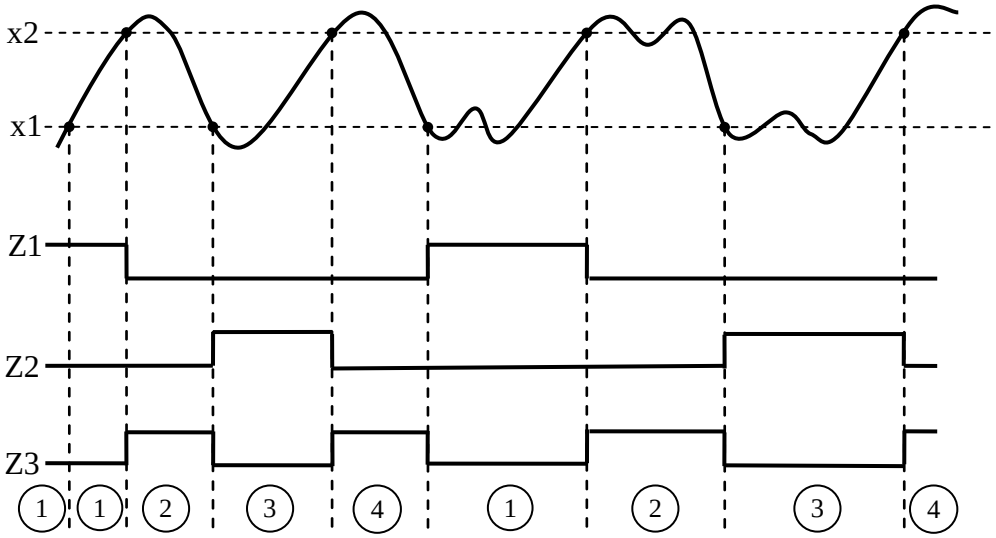
1. Problem

Zbiornik pokazany na rysunku jest napełniany na zmianę zaworami dopływowymi Z1, Z2, a po napełnieniu opróżniany zaworem Z3. Sekwencja pracy wygląda następująco:

- otwarcie Z1, aż poziom osiągnie x2
- zamknięcie Z1, otwarcie Z3, aż poziom spadnie do x1
- zamknięcie Z3, otwarcie Z2, aż poziom osiągnie x2
- zamknięcie Z2, otwarcie Z3, aż poziom spadnie do x1
- zamknięcie Z3, otwarcie Z1 itd.



2. Przebiegi czasowe



3. Stany

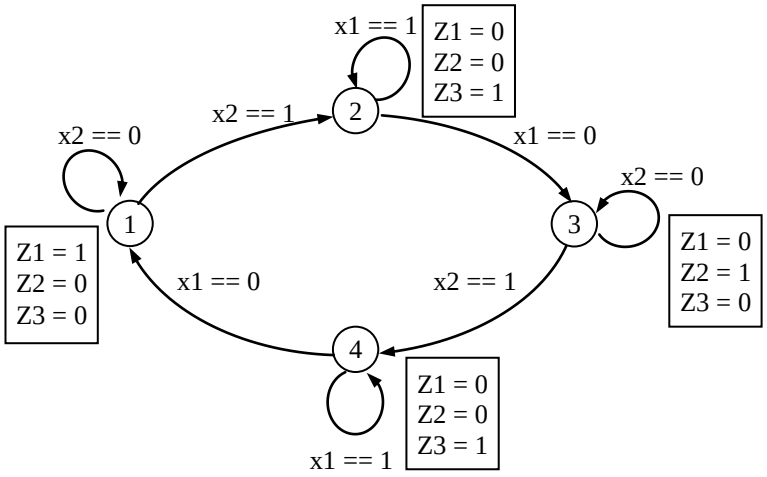
(1) – napełnianie z zaworu Z1

(2) – opróżnianie po Z1

(3) – napełnianie z zaworu Z2

(4) – opróżnianie po Z2

4. Automat



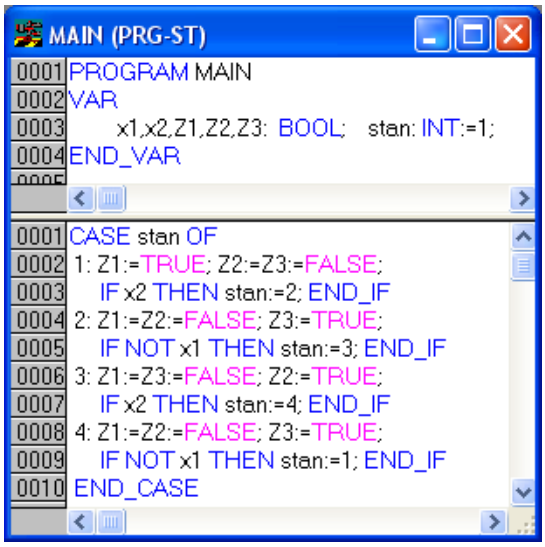
Oznaczenia jak w języku C.

5. Programy

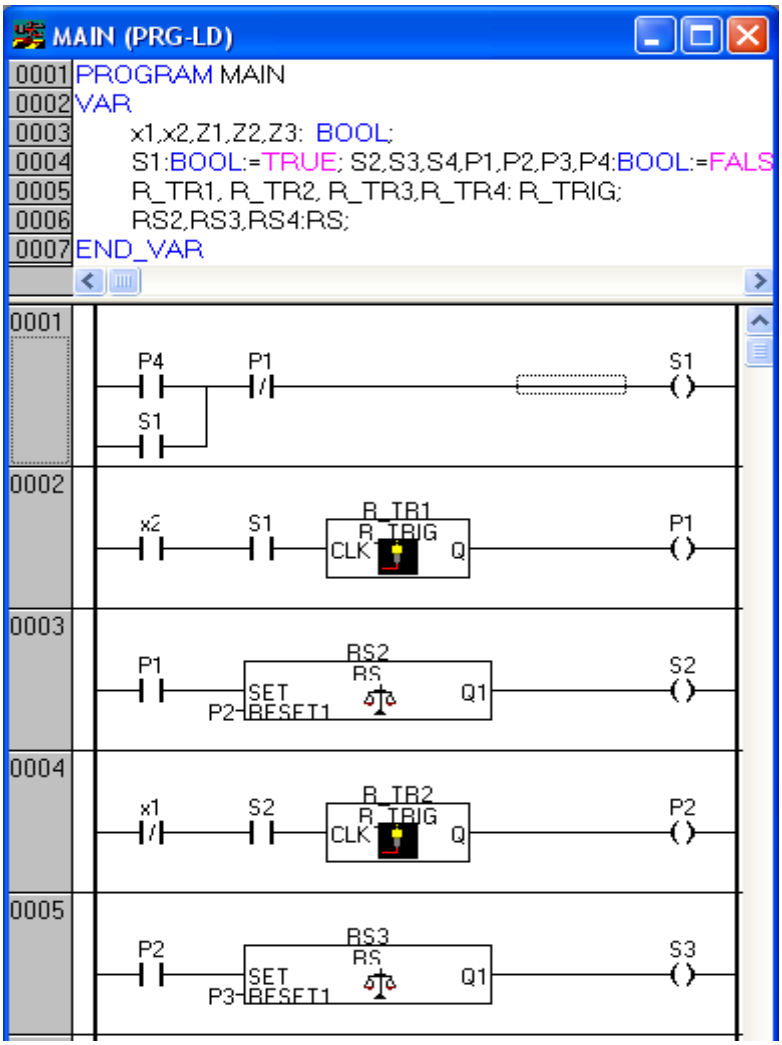
- **C**

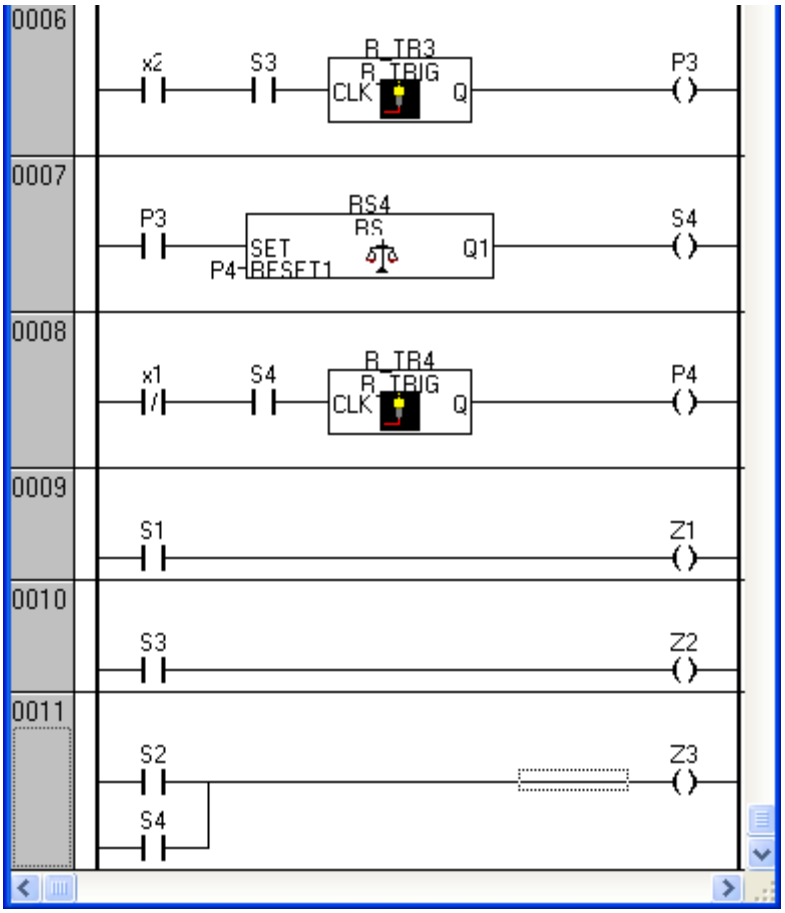
```
switch(stan)
    {
     case 1:  Z1=1;  Z2=Z3=0;
              if(x2)   stan=2;
              break;
     case 2:  Z1=Z2=0;  Z3=1;
              if(!x1)  stan=3;
              break;
     case 3:  Z1=Z3=0;  Z2=1;
              if(x2)  stan=4;
              break;
     case 4:  Z1=Z2=0;  Z3=1;
              if(!x1)  stan=1;
              break;
    }
```

- **ST**



- LD




0006
x2
S3
R_TR3
R_TRIG
CLK
Q
P3
0007
P3
RS4
RS
SET
P4
RESET1
Q1
S4
0008
x1
S4
R_TR4
R_TRIG
CLK
Q
P4
0009
S1
Z1
0010
S3
Z2
0011
S2
S4
Z3