MIĘDZYNARODOWA NORMA IEC 61131

2.1. Geneza powstania normy

Przez lata rozwoju sterowników programowalnych producenci wprowadzali różne metody programowania sterowników *PLC* (*Programmable Logic Controllers*). Wśród języków programowania największą popularność zyskał język schematów drabinkowych ze względu na jego podobieństwo do sposobu projektowania klasycznych systemów sterowania stykowo-przekaźnikowego. Często także do opisu algorytmu sterowania wykorzystuje się metodę grafów sekwencji. Jednak poszczególne rozwiązania proponowane przez różnych producentów sterowników i oprogramowania do nich różniły się między sobą, co dla projektantów i użytkowników takich systemów stanowiło dużą niedogodność i powodowało konieczność ciągłego dostosowywania się do innych wymagań. W związku z coraz powszechniejszym stosowaniem sterowników *PLC* powstała konieczność ich standaryzacji, a w szczególności metod oprogramowania.

Doświadczenia zebrane przez producentów i użytkowników zaowocowały wydaniem w 1993 roku przez Międzynarodową Komisję Elektroniki normy *IEC 1131 „Programmable Controllers”*. Obecnie zmieniono jej oznaczenie na *IEC 61131*. W ośmioczęściowej normie *IEC 61131 „Programmable Controllers”* zawarto pełny zakres postanowień normalizacyjnych dotyczących sterowników. Składa się ona z następujących części [4][8]:

1. Informacje ogólne (*General Information*),
2. Sprzęt i wymagania testowe (*Equipment and Test Requirements*),
3. Języki programowania (*Programing Languages*),
4. Wytyczne użytkownika (*User Guidelines*),
5. Wymiana informacji (*Messaging Service*),
6. Komunikacja sterowników programowalnych (*Programmable controller communications via fieldbus*),
7. Programowanie sterowania rozmytego (*Fuzzy control programming*),

8. Wytyczne dla aplikacji i implementacji programów sterowników programowalnych (*Guidelines for the application and implementation of languages for programmable controlers*).

W pierwszych dwóch częściach normy, zawarte są postanowienia ogólne, w tym definicje głównych elementów sterownika oraz wymagania i badania sterowników. Warto w tym miejscu przytoczyć definicję sterownika programowalnego oraz systemu sterownika programowalnego. Według pierwszej części normy *IEC 61131*, brzmią one następująco:

Definicja 2.1

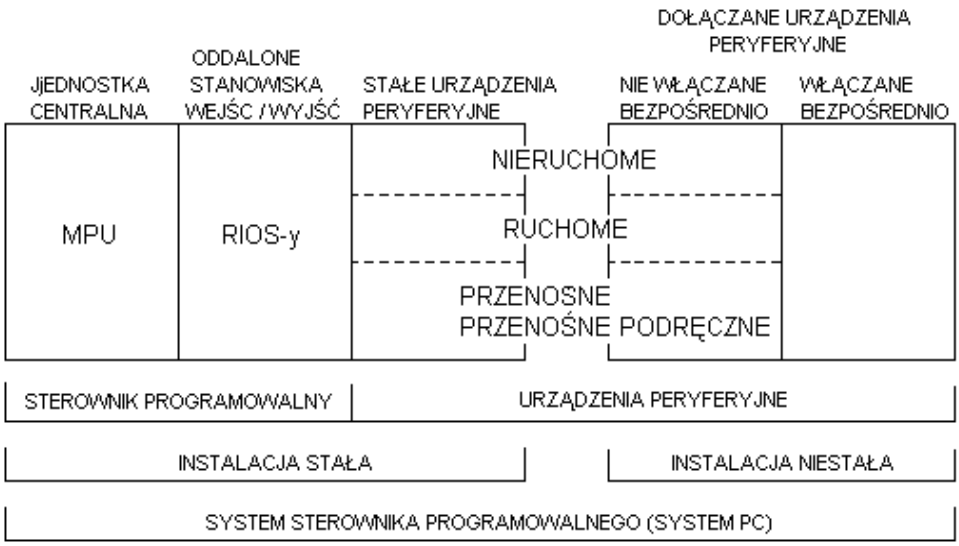
Sterownik programowalny (PC)

Cyfrowy system elektroniczny przeznaczony, do stosowania w środowisku przemysłowym, który posługuje się pamięcią programowalną, do wewnętrznego przechowywania zorientowanych na użytkownika instrukcji do implementowania specyficznych funkcji: logicznych, sekwencyjnych, taktujących, zliczających i arytmetycznych w celu sterowania przez cyfrowe lub analogowe wejścia i wyjścia, szeroką gamą maszyn i procesów. Zarówno PC, jak i związane z nim urządzenia peryferyjne są przeznaczone do łatwego połączenia w przemysłowy system sterowania i w prosty sposób spełniają funkcje przewidywane dla nich [1].

Definicja 2.2

System sterownika programowalnego (system PC)

Konfiguracja zestawiana przez użytkownika, składająca się ze sterownika programowalnego i związanych z nim urządzeń peryferyjnych, która jest niezbędna do przewidywanego zautomatyzowanego systemu. Składa się ona z jednostek wzajemnie połączonych, za pomocą kabli i łączówek w instalacji stałej, a za pomocą kabli lub innych środków w przypadku przenośnych i niestacjonarnych urządzeń peryferyjnych (Rys. 2.1) [1].



Rys.2.1. System sterownika programowalnego (system *PLC*)

Należy zwrócić uwagę na rozbieżność w oznaczeniu sterownika. W literaturze oznacza się go jako *PLC* natomiast w normie *PC*. W niniejszym opracowaniu korzysta się z oznaczenia bardziej popularnego czyli *PLC*.

Część trzecia normy, dotyczy języków programowania i stanowi jej najważniejszą część. Przede wszystkim dzięki niej ujednolicono koncepcję programowania *PLC* tak, aby w oparciu o wprowadzone zasady, użytkownik był w stanie programować bez większych trudności różne systemy *PLC* [4]. Czwarta część *Wytyczne dla użytkownika* jest poddana ankietyzacji powszechnej. Pozostałe części (5,6,7 i 8) są dopiero opracowywane [8].

2.2. Programowanie sterowników *PLC* według normy

Jak już wyżej wspomniano programowaniem sterowników *PLC* zajmuje się trzecia część normy: *IEC 61131-3*. Definiuje ona pojęcia podstawowe, zasady ogólne, model programowy i model komunikacyjny (wymiana danych między elementami oprogramowania) oraz podstawowe typy i struktury danych. W niniejszym rozdziale skupiono się przede wszystkim na sposobie programowania sterowników z wykorzystaniem grafu sekwencji *SFC*.

W normie określono dwie grupy języków programowania: języki graficzne i tekstowe. Zdefiniowano je następująco:

Definicja 2.3
Język graficzny
Język programowania oparty na reprezentacji graficznej [1].

Definicja 2.4
Język tekstowy
System składający się z prawidłowo zdefiniowanych, zwykle skończonych, zestawów znaków; reguł łączenia znaków w celu tworzenia wyrazów lub innych wyrażeń oraz specyficznego przyporządkowania znaczeń do słów lub wyrażeń [1].

W grupie języków tekstowych zdefiniowane zostały następujące języki [4]:

- **Język listy instrukcji *IL*** (***Instruction List***), będący odpowiednikiem języka typu asembler, którego zbiór instrukcji obejmuje operacje logiczne, arytmetyczne, operacje relacji, jak również funkcje przerzutników, czasomierzy, liczników itp. A oto jego definicja:

Definicja 2.5
Tekstowy język programowania, używający rozkazów w celu przedstawienia programu użytkowego systemu PC [1].

- **Język strukturalny *ST*** (***Structured Text***), który jest odpowiednikiem języka algorytmicznego wysokiego poziomu, zawierającego struktury programowe takie jak:

  If ... then ... else ... end_if,
  Case ... of ... end_case,
  For ... to ... do ... end_for,
  While ... do ... end_while,
  Repeat ... end_reapeat.

Poniżej przedstawiono definicję języka strukturalnego.

Definicja 2.6

Tekstowy język programowania posługujący się przyporządkowaniami, sterowaniami podprogramami, instrukcjami wyboru i iteracji w celu przedstawienia programu użytkowego systemu PC [1].

Do języków graficznych opisanych w normie *IEC 61131-3* należą [4]:

- **Język schematów drabinkowych *LD*** (***Ladder Diagram***), podobny do stykowych obwodów przekaźnikowych, w których dopuszcza się użycie także funkcji: arytmetycznych, logicznych, porównań i relacji jak również bloków funkcyjnych: przerzutników, czasomierzy, liczników, regulatora *PID* czy bloków programowych. Definicja języka schematów drabinkowych *LD* według normy brzmi następująco:

Definicja 2.7

Język programowania używający schematów drabinkowych w celu przedstawienia programu użytkowego systemu PC [1].

- **Język schematów blokowych *FBD*** (***Function Block Diagram***), będący odpowiednikiem schematów przepływu sygnału dla obwodów logicznych przedstawionych w formie połączonych bramek logicznych oraz bloków funkcyjnych takich jak w języku *LD*. Poniżej podano jego definicję:

Definicja 2.8

Język programowania używający funkcjonalnych schematów blokowych w celu przedstawienia programu użytkowego systemu PC [1].

Ponadto norma *IEC 61131-3* określa sposób tworzenia struktury wewnętrznej programu w postaci **grafu sekwencji *SFC*** (***Sequential Function Chart***), który pozwala na opisywanie zadań sterowania sekwencyjnego za pomocą grafów zawierających etapy (kroki) i warunki przejścia (tranzycje) między etapami. Poniżej przedstawiono definicje z pierwszej części normy: sekwencyjnego systemu sterowania oraz graficznego odpowiednika programu sekwencyjnego – tablicy sekwencji funkcji (grafu sekwencji *SFC*).

Definicja 2.9

Sekwencyjny system sterowania

System sterowania, w którym poszczególne kroki są wykonywane w ustalonej uprzednio kolejności. Przejście od jednego kroku sekwencji do następnego jest uzależnione od spełnienia zdefiniowanych uprzednio warunków. System taki może być uzależniony czasowo, tzn. warunki przejścia są tylko funkcjami czasu, lub uzależniony od warunków zewnętrznych, tzn. warunki przejścia są funkcjami tylko sygnałów wejściowych lub kombinacją tych uzależnień [1].

Definicja 2.10

Tablica sekwencji funkcji (SFC)

Graficzne przedstawienie programu sekwencyjnego, składające się z połączonych kroków, działań i skierowanych połączeń z warunkami przejścia [1].

Grafy *SFC* mogą być wykorzystywane przy programowaniu sterownika w jednym ze zdefiniowanych w normie języków (wymienione powyżej) w celu otrzymania odpowiedniej struktury programu użytkownika [4].

W normie *IEC 61131-3* wyróżniono następujące elementy języków programowania dla sterowników *PLC* [1][4]:

1. Typy danych (*Data types*),
2. Elementy organizacyjne oprogramowania (*Program organization units*):
   - Funkcje (*Functions*),
   - Bloki funkcyjne (*Function blocks*),
   - Programy (*Programs*),
3. Elementy programowania za pomocą grafu sekwencji (*SFC*),
4. Elementy konfiguracji:
   - zmienne globalne (*Global variables*),
   - zasoby (*Resources*),
   - zadania (*Tasks*),
   - ścieżki dostępu (*Access paths*).

W kolejnym punkcie rozdziału przedstawiono elementy programowania za pomocą grafu sekwencji *SFC*.

2.3. Programowanie w oparciu o graf sekwencji *SFC*

Opis programowania algorytmu sterowania za pomocą grafu sekwencji umożliwia przejrzyste tworzenie struktury wewnętrznej programu. W czasach gdy systemy sterownikowe były stosunkowo małe, opis działania takich systemów w postaci typowej dla automatów sekwencyjnych był całkowicie wystarczający dla spełnienia wymagań dotyczących projektowania systemów sterowania logicznego. Stosowano głównie dwie metody opisu automatów sekwencyjnych [4]:

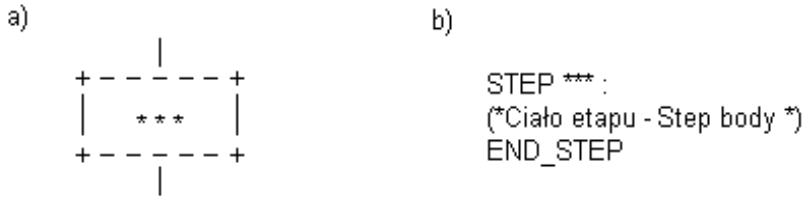
- Pierwsza z nich polegała na przedstawieniu sekwencyjnych układów asynchronicznych w postaci grafu przejść lub tablicy stanów. Opisy takie reprezentują automat w sposób niezależny od jego realizacji sprzętowej.
- Druga metoda modelowania układu sekwencyjnego wiązała się ściśle z jego fizyczną realizacją. Tutaj powszechnie stosuje się, aż do chwili obecnej, przedstawienie automatu w formie schematów drabinkowych bazujących na graficznej analogii do schematów układów przekaźnikowych.

Jednak już na początku lat siedemdziesiątych pojawiła się potrzeba modelowania niezwykle złożonych systemów sterowania sekwencyjnego dla projektowania programów sterownikowych. Stawianym wymaganiom sprostał opis, który otrzymał nazwę ***Grafcet*** (patrz Rozdział 3). W oparciu o zasady zaproponowane w metodzie *Grafcet* zdefiniowano w normie *IEC 61131-3* sposób opisu działania sterownika w postaci grafu sekwencji ***SFC*** (Sequential Function Chart). Formalizm ten jest podobny do opisu *Grafcet* i może być wykorzystywany przy programowaniu sterownika w jednym ze zdefiniowanych w normie języków w celu stworzenia odpowiedniej struktury wewnętrznej programu użytkownika [4].

2.3.1. Składnia grafu sekwencji *SFC*

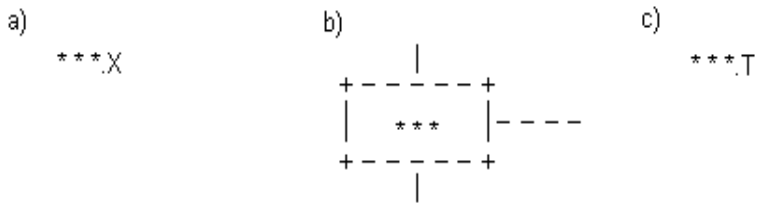
Element oprogramowania sterownika może być zorganizowany w postaci grafu sekwencji *SFC*, który składa się z wzajemnie sprzężonych **etapów** (*steps*) i **przejść** (*transitions*). Z każdym etapem jest skojarzony zbiór odpowiednich **działań** (*actions*) a każdemu przejściu między krokami towarzyszy **warunek przejścia** (*transition condition*). Ponieważ wymienione elementy *SFC* wymagają pamięci dla przechowania informacji o stanie systemu, jedynymi elementami oprogramowania, które mogą być z nich zbudowane są bloki funkcyjne i programy. Jeżeli jakakolwiek część takiego elementu jest programowana przy użyciu *SFC*, to w ten sam sposób powinien być programowany cały element. Jeżeli element oprogramowania nie jest zorganizowany w postaci grafu *SFC*, to powinien być traktowany jako pojedyncze działanie wykonywane pod kontrolą jednostki wywołującej[1][4].

Etap określa sytuację, w której zachowanie się elementu oprogramowania w odniesieniu do jego wejść i wyjść jest zdefiniowane przez skojarzony z etapem zestaw działań. Etap może być **aktywny** (*active*) lub **nieaktywny** (*inactive*). W każdej chwili stan elementu oprogramowania jest określony przez zestaw aktywnych etapów oraz wartości zmiennych wewnętrznych i wyjściowych tego elementu. Etap jest przedstawiany graficznie jako prostokąt zawierający nazwę etapu lub w postaci tekstowej za pomocą konstrukcji *STEP...END_STEP* (Rys 2.2.). Łączenie kolejnych etapów wykonuje się za pomocą linii pionowych (wejście u góry, wyjście na dole) lub odpowiednio, za pomocą zestawu deklaracji *TRANSITION...END_TRANSITION* [1][4].



Rys. 2.2. Etap:
a) postać graficzna (z kierunkiem łączenia). *** - nazwa etapu,
b) postać tekstowa, *** - nazwa etapu

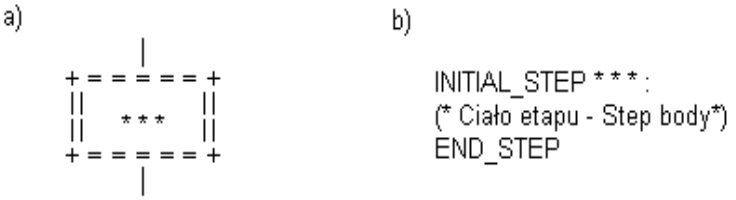
Aktywność etapu określa **wskaźnik etapu** (*step flag*), który jest reprezentowany przez zmienną logiczną ***.X, gdzie, *** jest nazwą etapu przyjmującą wartość *1* dla etapu aktywnego i *0* dla etapu nieaktywnego. Czas aktywności etapu (*step elapsed time*) jest określony przez zmienną ***.T typu *TIME*. Wartość tej zmiennej pozostaje bez zmian, gdy etap przechodzi w stan nieaktywny [1][4]. Powyższe zagadnienia przedstawione są na Rysunku 2.3.



Rys. 2.3. a) Wskaźnik etapu – postać ogólna, *** - nazwa etapu,
b) Wskaźnik etapu- bezpośredni dostęp do zmiennej logicznej ***.X (linia ---),
c) Czas aktywności etapu – postać ogólna, ***.T - zmienna typu TIME (czas)

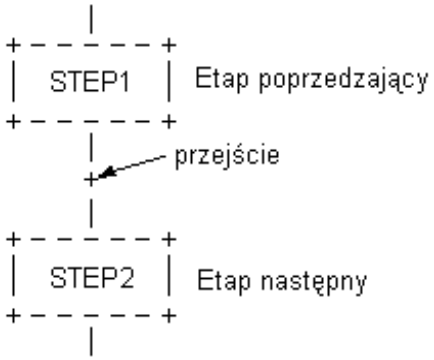
Zakres stosowania nazw etapów, ich wskaźników i czasów aktywności jest lokalny w ramach elementu oprogramowania, w którym się pojawiają. Ważne jest również to, że zmienne ***.X oraz ***.T nie mogą być modyfikowane przez program użytkownika, ich wartości mogą być tylko odczytywane. Domyślną wartością początkową dla czasu aktywności etapu jest *t#0s*. Domyślną wartością początkową dla wskaźnika etapu jest logiczne *0*, z wyjątkiem etapu początkowego (opisany poniżej), którego wskaźnik przyjmuje z chwilą inicjacji *1*.

Stan początkowy elementu oprogramowania jest określony przez wartości początkowe jego zmiennych wewnętrznych i wyjściowych oraz przez zbiór **etapów początkowych** (*initial steps*), tj. etapów aktywnych w chwili początkowej. Każda sieć *SFC* lub jej tekstowy odpowiednik powinny zawierać tylko jeden etap początkowy. Etap początkowy zaznacza się jako prostokąt rysowany podwójną linią, a w przypadku reprezentacji tekstowej jako konstrukcję *INITIAL_STEP...END_STEP* (Rys. 2.4) [1][4].



Rys. 2.4. Etap początkowy:
a) postać graficzna (łącznik górny nie jest wymagany jeśli nie ma poprzednika),
b) postać tekstowa

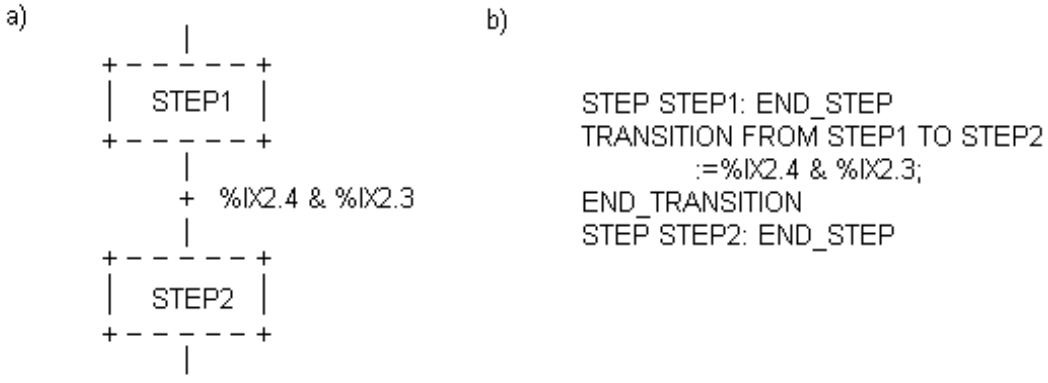
Przejście (*transition*) przedstawia warunki logiczne, których spełnienie jest wymagane do przeniesienia sterowania z jednego lub więcej etapów poprzedzających przejście do jednego lub więcej etapów występujących za przejściem. Przejście przedstawiane jest graficznie jako poziomy odcinek przecinający linią łączącą kroki (Rys. 2.5).



Rys. 2.5. Reprezentacja graficzna przejścia

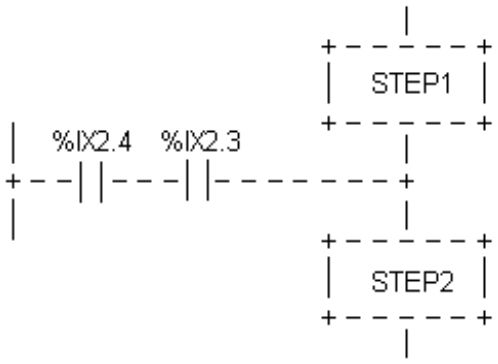
Z każdym przejściem jest skojarzony **warunek przejścia** (*transition condition*), który jest wynikiem rozwiązania pojedynczego wyrażenia logicznego. Warunek przejścia zawsze prawdziwy oznacza się symbolem *1* lub słowem kluczowym *TRUE*. A oto niektóre sposoby kojarzenia warunków przejścia z wykorzystaniem różnych języków opisanych w normie *IEC 61131-3* [1][4]:

- Warunek przejścia zapisany w języku *ST* (Rys. 2.6)



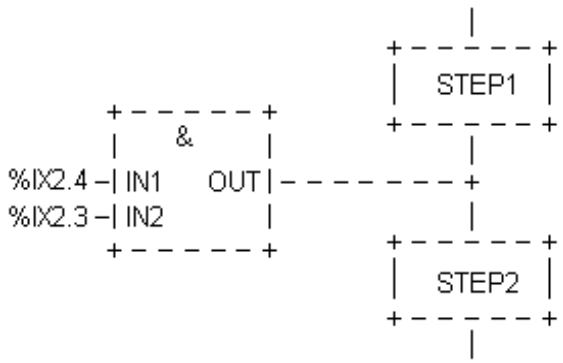
Rys. 2.6. Warunek przejścia zapisany w języku *ST*:
a) postać graficzna,
b) postać tekstowa

- Warunek przejścia zapisany w języku *LD* (Rys. 2.7)



Rys. 2.7. Warunek przejścia zapisany w języku *LD*

- Warunek przejścia zapisany w języku *FBD* (Rys.2.8)

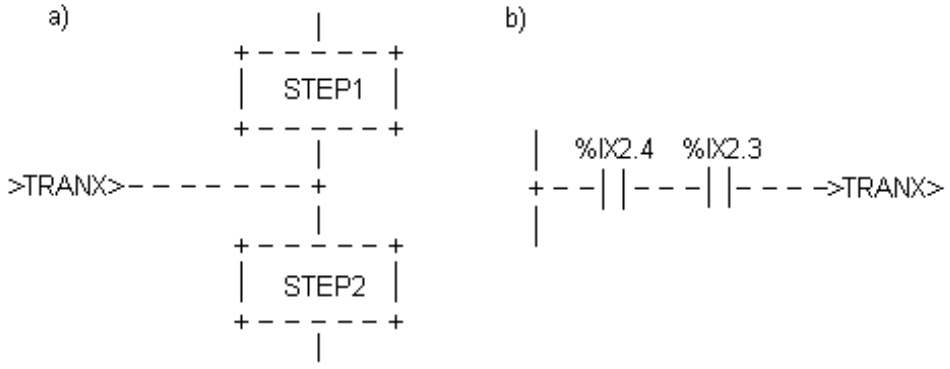


Rys. 2.8. Warunek przejścia zapisany w języku *FBD*

- Warunek przejścia zapisany w języku *IL*

```
STEP STEP1: END_STEP
TRANSITION FROM STEP1 TO STEP2
    LD      %IX2.4
    AND     %IX2.3
END_TRANSITION
STEP STEP2: END_STEP
```

- Warunek przejścia z użyciem łącznika do przejścia (Rys. 2.9).



Rys. 2.9. Warunek przejścia z użyciem łącznika:
a) łącznik do przejścia,
b) warunek przejścia zapisany w języku *LD*

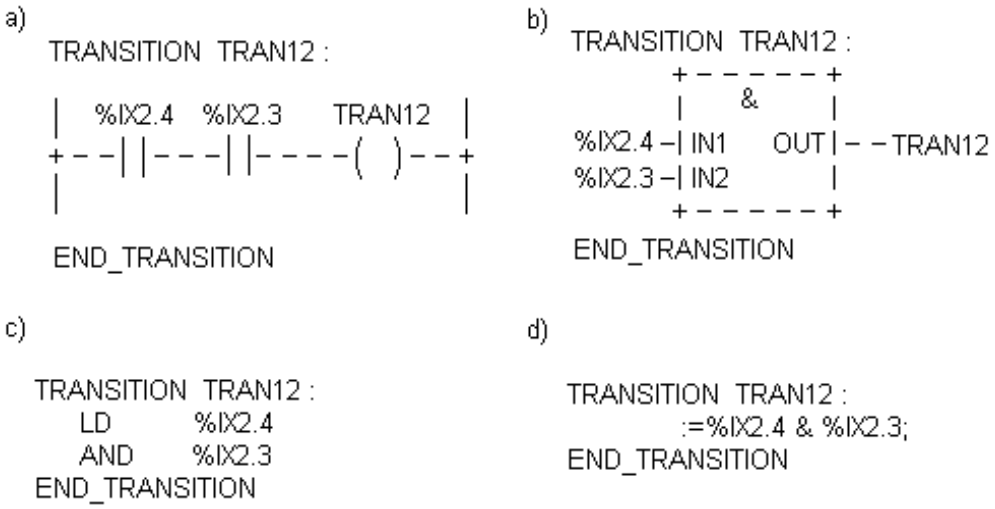
Warunki przejścia można opisywać oddzielnie, tzn. nie bezpośrednio przy samych przejściach lecz przy użyciu łączników do przejścia TRANX. Przykład użycia łącznika pokazano na powyższym rysunku.

- Warunek przejścia z użyciem nazwy przejścia (Rys. 2.10 i Rys 2.11)

Jest to sposób podobny do przedstawionego powyżej (z użyciem łączników do przejścia). Same nazwy przejść mogą posłużyć do kojarzenia warunków przejścia z samym przejściem.

Rys. 2.10. Użycie nazwy przejścia *TRAN12*

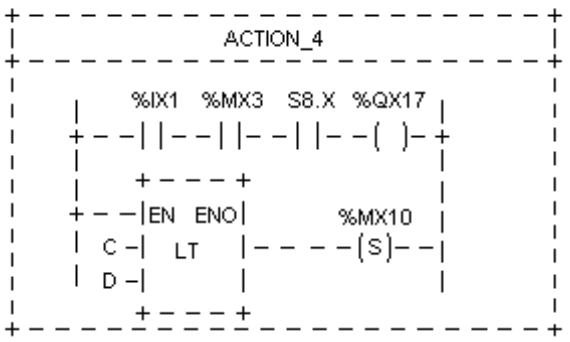
Wtedy to warunki przejścia mogą przyjąć następującą postać (Rys. 2.11):



Rys. 2.11. Warunek przejścia z użyciem nazwy przejścia, zapisany w języku:
a) *LD*,
b) *FBD*,
c) *IL*,
d) *ST*

Z każdym etapem *SFC* skojarzone jest pewne **działanie** (*actions*). Działanie może być wyrażone przez zmienną logiczną, zestaw instrukcji w języku *IL*, poleceń w języku *ST*, szczebli języka *LD*, obwodów *FBD* a także jako graf *SFC* (Rys. 2.12). Etap, który zawiera zero działań realizuje funkcję *WAIT*, tzn. oczekiwane jest spełnienie warunków przejścia do następnego etapu. Ponadto działaniem może być także każda zmienna logiczna deklarowana w blokach *VAR* (deklaracja zmiennych wewnętrznych w elemencie) lub *VAR_OUTPUT* (deklaracja zmiennych wyprowadzonych na zewnątrz do innych jednostek oprogramowania) albo ich graficzny odpowiednik. Deklaracja działania jest lokalna w zakresie elementu oprogramowania, który ją zawiera [1][4].

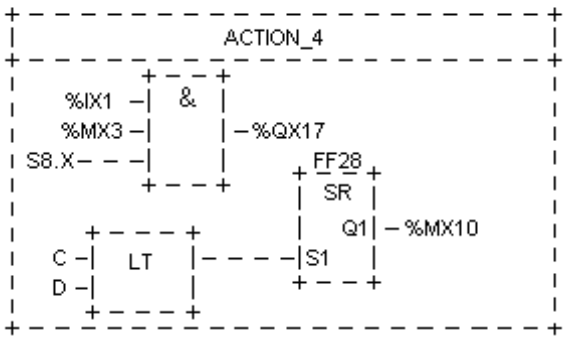
a)



b)

```
ACTION  ACTION_4 :
   %QX17 = %IX1 & %MX3 & S8.X ;
   FF28 (S1 := (C < D) ) ;
   %MX10 := FF28.Q ;
END_ACTION
```

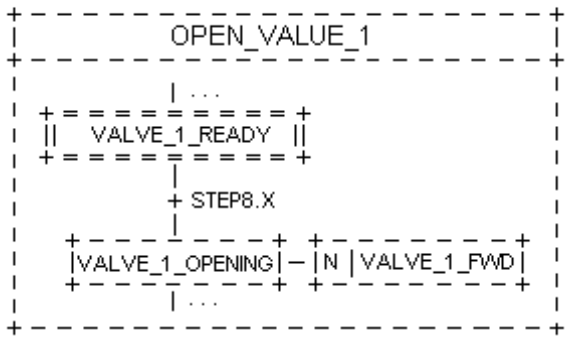
c)



d)

```
ACTION      ACTION_4 :
LD          S8.X
AND         %IX1
AND         %MX3
ST          %QX17
LD          C
LT          D
S1          FF28
LD          FF28.Q
ST          %MX10
END_ACTION
```

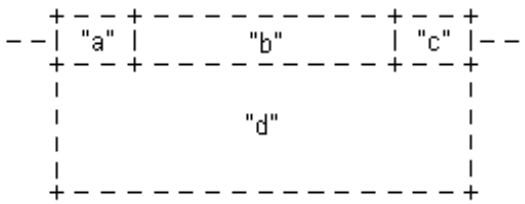
e)



Rys. 2.12. Przykłady deklaracji działań:
a) graficzna w języku *LD*,
b) tekstowa w języku *ST*,
c) graficzna w języku *FBD*,
d) tekstowa w języku *IL*,
e) z użyciem elementów *SFC*

W prezentowanych przykładach wskaźnik etapu *S8.X* został użyty w celu uzyskania wymaganego działania polegającego na tym, że z chwilą gdy etap *S8* przestaje być aktywny, wyjście *%QX17:=0*, a zmienna *%MX10* zachowuje swoją poprzednią wartość.

W sterownikach programowalnych, w których możliwe jest stosowanie formalizmu *SFC* (patrz Rozdział 5) istnieją również odpowiednie mechanizmy do kojarzenia etapów z działaniami (*step/action association*). Najczęściej kojarzy się etap z działaniem za pomocą tzw. **bloku działania** (*action block*), który w sposób graficzny reprezentuje przetwarzanie logicznej zmiennej wejściowej, oraz **kwalifikatora działania** (*action qualifier*) w celu określenia warunków zezwolenia na wykonanie zadeklarowanego zestawu instrukcji tworzących działanie. Sposób graficznego przedstawienia bloku działania pokazano na Rysunku 2.13.



Rys. 2.13. Graficzna reprezentacja bloku działania

Na powyższym rysunku poszczególne pola oznaczone literami mają następujące znaczenie [1][4]:

- „*a*” kwalifikator działania (patrz Tabela 2-1),
- „*b*” nazwa działania,
- „*c*” zmienna logiczna zwrotna,
- „*d*” działanie definiowane w językach *IL*, *ST*, *LD* lub *FBD*.

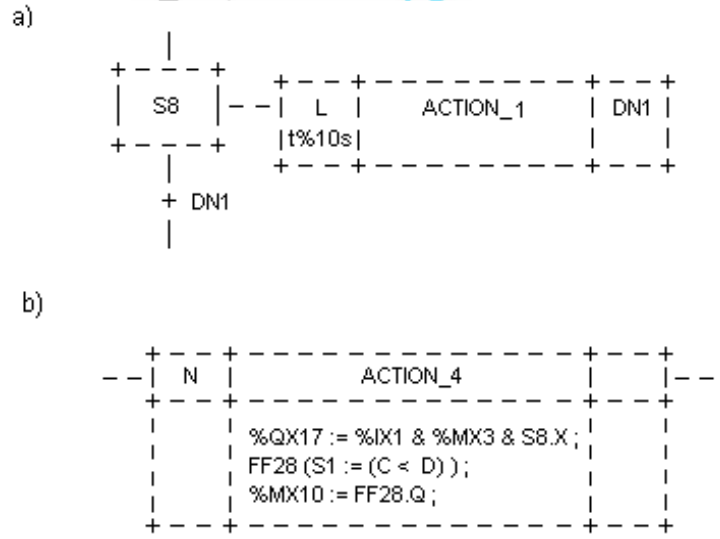
Jak już wspomniano, zarządzanie wykonywaniem działań może być realizowane za pomocą skojarzonych z nimi kwalifikatorów działania. Kwalifikatory działania specyfikują, czy określone działanie ma być uruchomione i z jakimi ewentualnymi ograniczeniami czasowymi. Tablica 2-1 zawiera zestawienie zdefiniowanych w normie *IEC 61131-3* kwalifikatorów działania. Kwalifikatorom *L*, *D*, *SD*, *DS.*, i *SL* towarzyszy zmienna typu *TIME* określająca czas trwania (patrz Rys. 2.14a, 2.15).

Tabela 2-1. Kwalifikatory działania [1][4]

Kwalifikator	Opis
Brak kwalifikatora	Nie pamiętany (kwalifikator zerowy)
N	Nie pamiętany
R	Nadrzędne kasowanie (*overriding Reset*)
S	Załączenie (*Set*) – pamiętany
L	Ograniczenie czasowe (*time Limited*)
D	Opóźnienie czasowe (*time Delayed*)
P	Pulsacja (*Pulse*)
SD	Załączenie i opóźnienie czasowe
DS.	Opóźnienie czasowe i załączenie
SL	Załączenie i ograniczenie czasowe

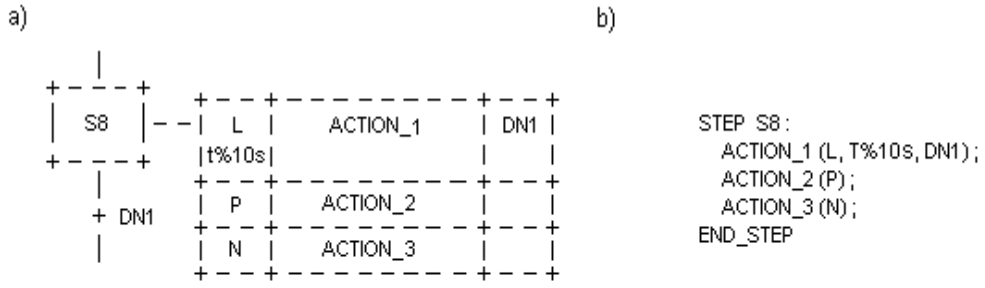
Pole „*a*” może być pominięte jeżeli kwalifikatorem działania jest *N*. Użycie zmiennej logicznej zwrotnej, oznaczonej jako pole „*c*” na Rysunku 2.13 jest opcjonalne. Wyznaczone działanie może wykorzystywać tę zmienną do sygnalizacji kompletności wykonania, przekroczenia czasu, wystąpienia błędów itp.

Przykłady kojarzenia etapów z działaniami przedstawiono na Rysunku 2.14 i 2.15.



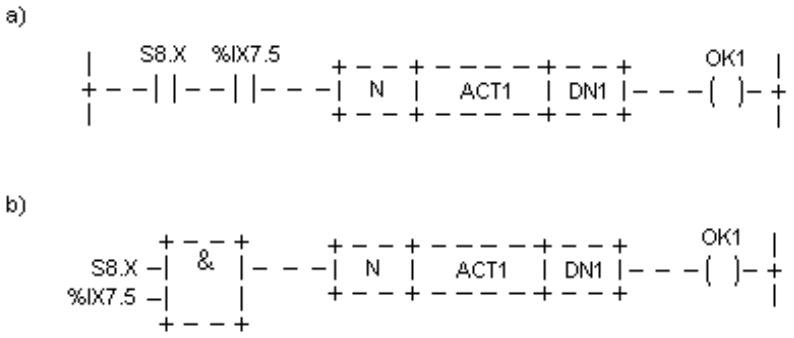
Rys. 2.14. Kojarzenie etapu z działaniami:
a) blok działania,
b) wykorzystanie pola „d” w bloku działania

W przypadku kaskadowego łączenia bloków działania, jak pokazano to na Rysunku 2.15, możliwe jest wykorzystanie wielu zmiennych zwrotnych, chociaż bloki te mają tylko jedną wspólną zmienną wejściową oddziaływującą równocześnie na wszystkie połączone bloki.



Rys. 2.15. Kaskadowe łączenie bloków działania:
a) postać graficzna,
b) ciało etapu zapisane tekstowo

Bloki działania oprócz kojarzenia z etapami grafu *SFC* mają bezpośrednio zastosowanie w językach programowania opisanych w normie. Na Rysunku 2.16 przedstawiono przykładowe ich użycie w schemacie drabinkowym *LD* oraz w schemacie bloków funkcyjnych *FBD*.



Rys. 2.16. Zastosowanie bloku działania:
a) w schemacie drabinkowym *LD*,
b) w schemacie bloków funkcyjnych *FBD*

2.3.2. Reguły modelowania za pomocą grafu *SFC*

Warunki początkowe grafu *SFC* są określone przez etap początkowy (*initial step*), który jest w stanie aktywnym z chwilą inicjacji programu użytkownika lub bloku funkcyjnego zawierającego graf *SFC*.

Zmiana stanu procesu polega na przechodzeniu między kolejnymi, bezpośrednio połączonymi etapami i zależy od warunku przejścia między nimi.

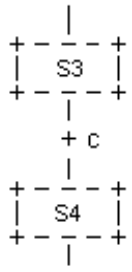
Przejście (tranzycja) jest dostępne tylko wtedy, gdy wszystkie bezpośrednio poprzedzające etapy są aktywne. Kasowanie przejścia występuje wtedy, gdy jest ono dostępne i jest spełniony warunek przejścia (wyrażenie logiczne określające warunek przejścia ma wartość *1*). Kasowanie przejścia powoduje dezaktywację wszystkich bezpośrednio poprzedzających je etapów i aktywację wszystkich etapów występujących bezpośrednio po nim [1][4].

Przy konstrukcji grafu *SFC* trzeba pamiętać o następujących wymaganiach:

- dwa etapy nie mogą być bezpośrednio połączone – muszą być zawsze rozdzielone przejściem;
- dwa przejścia nie mogą być bezpośrednio połączone – muszą być zawsze rozdzielone etapem.

Zasady modelowania za pomocą sekwencji etapów i przejść w grafie *SFC*, przedstawione w normie *IEC 61131-3* są następujące [1][4]:

- **Sekwencja pojedyncza** (*single sequence*) – etapy i przejścia są powtarzane po sobie (Rys. 2.17).

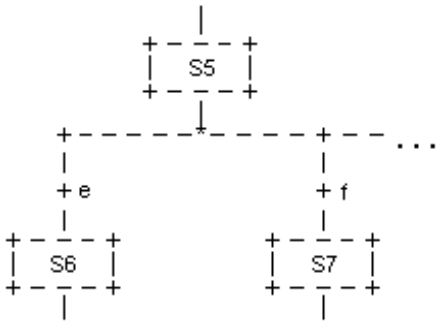


Rys. 2.17. Sekwencja pojedyncza

Przejście od realizacji etapu *S3* do *S4* jest możliwe tylko wtedy, gdy etap *S3* jest aktywny i spełniony jest warunek *c*.

- **Realizacja wyboru sekwencji** (*divergence of sequence selection*)

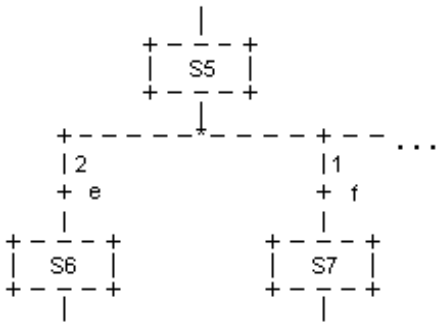
a) Wybór między różnymi sekwencjami jest reprezentowany przez tyle przejść umieszczonych pod linią poziomą ile jest możliwych sekwencji. Gwiazdka oznacza pierwszeństwo od lewej strony do prawej (Rys. 2.18).



Rys. 2.18. Realizacja wyboru sekwencji – pierwszeństwo wyboru od lewej do prawej

Przejście z etapu *S5* do *S6* jest realizowane tylko wtedy, gdy *S5* jest aktywny i jest spełniony warunek *e*, a przejście z *S5* do *S7* wtedy, gdy *S5* jest aktywny i jest spełniony warunek *f* a nie jest spełniony warunek *e*.

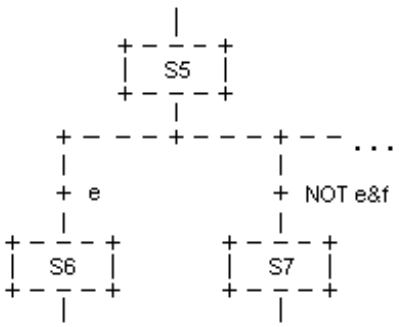
b) Gwiazdka i numerowane gałęzie wskazują na zadeklarowane pierwszeństwo wyboru przejścia. Gałęzie o niższym numerze mają wyższy priorytet (Rys. 2.19).



Rys. 2.19. Realizacja wyboru sekwencji – pierwszeństwo wyboru przejścia mają gałęzie o niższym numerze

Przejście od etapu *S5* do *S7* następuje wtedy, gdy *S5* jest aktywny i jest spełniony warunek *f*, a przejście z *S5* do *S6* wtedy, gdy *S5* jest aktywny i jest spełniony warunek *e* a nie jest spełniony warunek *f*.

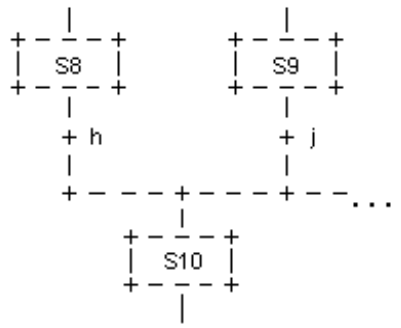
c) Połączenie gałęzi oznacza, że kolejne warunki przejścia muszą się wzajemnie wykluczać (Rys. 2.20).



Rys. 2.20. Realizacja wyboru sekwencji – wzajemnie wykluczające się przejścia

Przejście z etapu *S5* do *S6* następują wtedy, gdy *S5* jest aktywny i jest spełniony warunek *e*, a przejście z *S5* do *S7* wówczas, gdy *S5* jest aktywny i nie jest spełniony warunek *e* a jest spełniony warunek *f*.

- **Zakończenie wyboru sekwencji** (*convergence of sequence selection*) – wszystkie sekwencje kończą się symbolami przejścia umieszczonymi nad linią poziomą (Rys. 2.21).



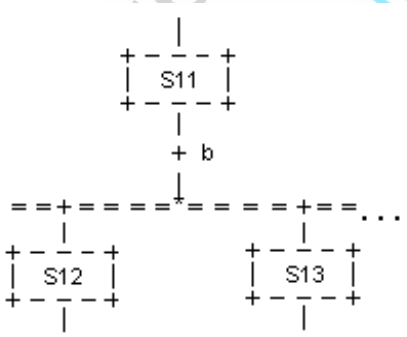
Rys. 2.21. Zakończenie wyboru sekwencji

Przejście do etapu *S10* jest możliwe wtedy, gdy etap *S8* jest aktywny i jest spełniony warunek *h* lub gdy *S9* jest aktywny i jest spełniony warunek *j*.

- **Sekwencje współbieżne** (*simultaneous sequences*)

Graf *SFC* umożliwia modelowanie procesów współbieżnych za pomocą tzw. sekwencji współbieżnych realizowanych w ten sposób, że kasowanie warunku przejścia prowadzi do jednoczesnej aktywacji kilku różnych etapów występujących bezpośrednio za nim. Sekwencje takich etapów rysowane są równolegle. Po równoczesnej aktywacji sekwencji współbieżnych dalsze wykonywanie każdej z nich przebiega niezależnie od pozostałych. W celu podkreślenia szczególnej natury takiego przejścia rozchodzenie się (*divergence*) i schodzenie (*convergence*) sekwencji współbieżnych jest zaznaczane przez podwójną linię poziomą.

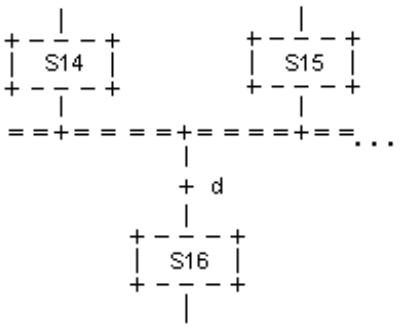
a) Początek sekwencji współbieżnych – jeden wspólny symbol przejścia umieszczony bezpośrednio nad podwójną linią poziomą (Rys. 2.22).



Rys. 2.22. Początek sekwencji współbieżnych

Przejście z etapu *S11* do etapów *S12*, *S13*, ... następuje tylko wtedy, gdy *S11* jest aktywny i jest spełniony wspólny warunek *b*. Po jednoczesnej aktywacji *S12*, *S13*, ... realizacja każdej sekwencji jest niezależna.

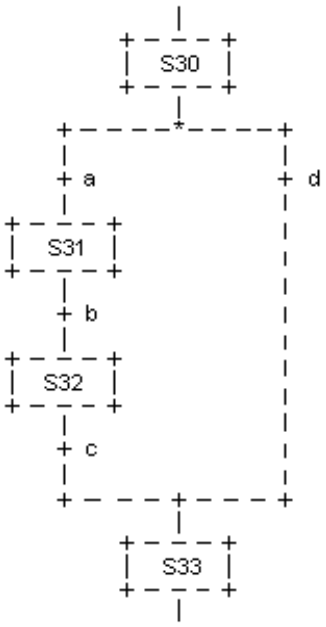
b) Zakończenie sekwencji współbieżnych – jeden wspólny symbol przejścia umieszczony bezpośrednio pod podwójną linią poziomą (Rys. 2.23).



Rys. 2.23. Zakończenie sekwencji współbieżnych

Przejście z etapów *S14*, *S15*, ... do etapu *S16* następuje tylko wtedy, gdy wszystkie etapy dołączone do linii podwójnej są aktywne i jest spełniony wspólny warunek *d*.

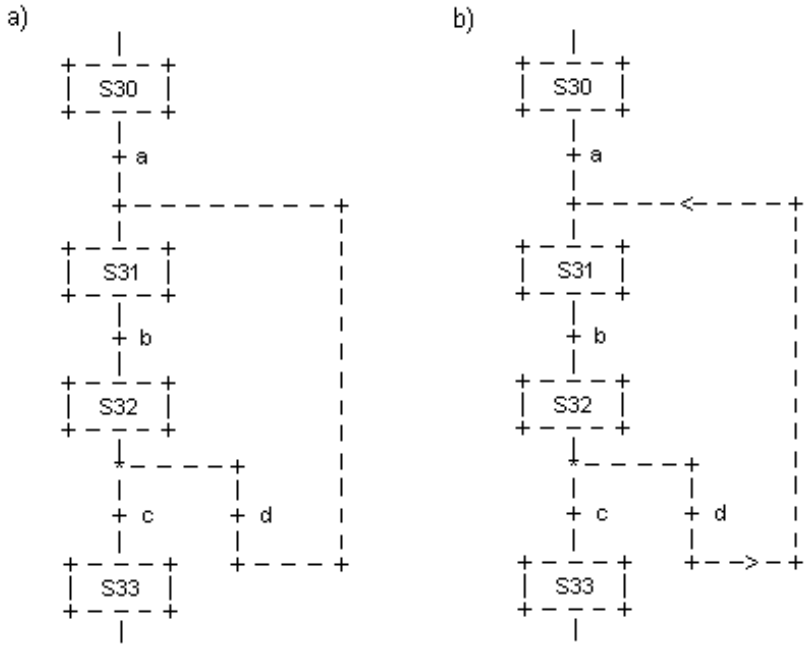
- **Ominięcie sekwencji** (*sequence skip*) – jest to szczególny przypadek wyboru sekwencji, gdy któraś z gałęzi nie zawiera żadnych etapów (Rys. 2.24).



Rys. 2.24. Ominięcie sekwencji

Przejście z etapu *S30* do *S33* wystąpi wtedy, gdy *S30* jest aktywny i nie jest spełniony warunek *a*, natomiast jest spełniony warunek *d*. Pozostałe dwa warianty realizacji wyboru sekwencji (Rys. 2.19 i 2.20) mają również zastosowanie w ominięciu sekwencji. Sposób ich realizacji jest taki sam.

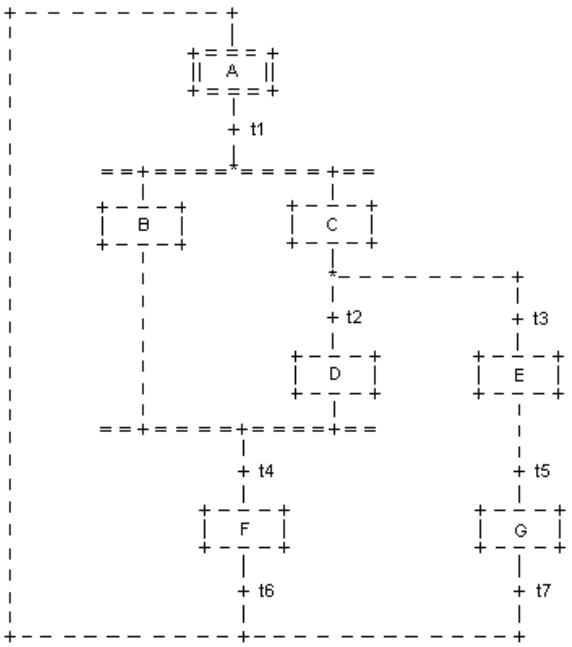
- **Cykliczna realizacja sekwencji** (*sequence loop*) – jest to szczególny przypadek wyboru sekwencji, gdy któraś z gałęzi wraca do etapu poprzedniego.



Rys. 2.25. Cykliczna realizacja sekwencji:
a) pętla bez oznaczenia kierunku,
b) pętla z oznaczeniem kierunku

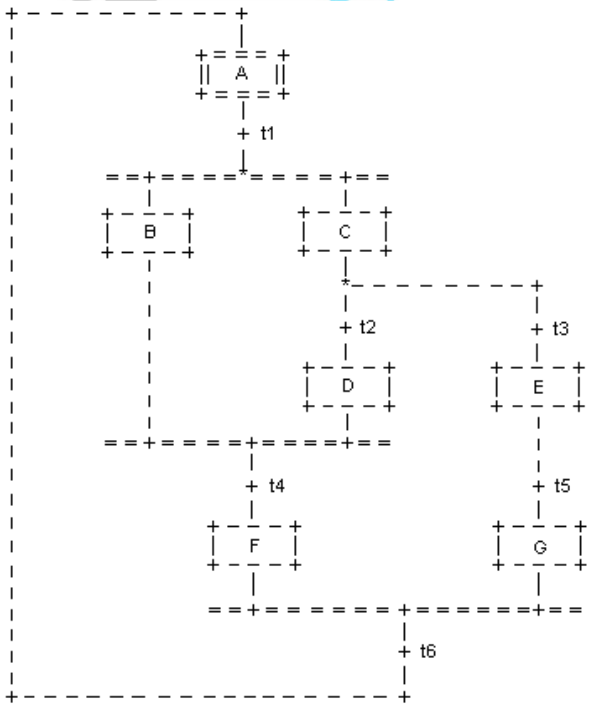
Przejście z etapu *S32* do *S31* wystąpi wtedy, gdy jest spełniony warunek *d* a nie jest spełniony warunek *c*. W celu zwiększenia przejrzystości grafu dopuszcza się użycie znaków „<” i „>” umieszczonych między znakami „-„ dla określenia kierunku przepływu sygnału sterującego, jak pokazano na Rysunku 2.25b. Podobnie jak w ominięciu sekwencji tak i w cyklicznej realizacji sekwencji, warianty z Rysunków 2.19 i 2.20 mają również tutaj zastosowanie.

Przy korzystaniu z przedstawionych zasad należy pamiętać, by do sekwencji współbieżnych nie wprowadzać wyboru sekwencji o ścieżkach kończących się poza wspólną linią kończącą sekwencję współbieżną. Rysunek 2.26 pokazuje przykład takiego błędu, który może prowadzić do niekontrolowanej realizacji sekwencji w grafie.



Rys. 2.26. Przykład grafu *SFC* o nieokreślonym działaniu

Rysunek 2.27 przedstawia z kolei przykład błędu prowadzącego do tego, że realizacja grafu może nigdy nie zostać zakończona.



Rys. 2.27. Przykład grafu „nierozwiązywalnego