

## 2. NORMA IEC 61131

Norma *IEC 61131* stanowi kontynuację i wykorzystuje szereg innych standardów. Odwołuje się ona do 10 innych norm (*IEC 50, IEC 559, IEC 617-12, IEC 617-13, IEC 848, ISO/AFNOR, ISO/IEC 646, ISO 8601, ISO 7185, ISO 7498*). W Europie została zatwierdzona jako norma *EN 61131*.

### Elementy składowe normy

#### **Część 1. Postanowienia ogólne** (ang. *General Information*)

Część ta zawiera ogólne definicje i typowe własności funkcjonalne, które odróżniają sterowniki programowalne PLC od innych systemów. Obejmuje ona standardowe własności sterowników PLC, jak np. cykliczne przetwarzanie programu aplikacyjnego korzystającego z przechowywanego w pamięci obrazu stanu wejść i wyjść sterownika lub przydział czasu pracy na komunikację z programatorem czy urządzeniami interfejsu operatora MMI.

#### **Część 2. Wymagania i badania dotyczące sprzętu** (ang. *Equipment Requirements and Tests*)

Zdefiniowane zostały tu elektryczne, mechaniczne i funkcjonalne wymagania dla urządzeń oraz odpowiednie testy jakości. Określono także warunki środowiskowe (temperatura, wilgotność powietrza itp.) oraz dokonano klasyfikacji sterowników i narzędzi programowania.

#### **Część 3. Języki programowania** (ang. *Programming Languages*)

Ujednolicono stosowane dotychczas języki programowania w zharmonizowany i zorientowany przyszłościowo system. Pojęcia podstawowe, zasady ogólne, model programowy i model komunikacyjny zostały opisane za pomocą formalnych definicji. Trzecia część normy specyfikuje syntaktykę i semantykę tekstowych i graficznych języków programowania, oraz elementy konfiguracji wspomagające instalację oprogramowania w sterownikach.

#### **Część 4. Wytyczne dla użytkownika** (ang. *User Guidelines*)

Część ta stanowi przewodnik dla użytkownika PLC, wspomagający go we wszystkich fazach projektowania systemu automatyki. Podane zostały praktyczne informacje, poczynając od analizy systemu i wyboru sprzętu, aż po zastosowania.

#### **Część 5. Wymiana informacji** (ang. *Messaging Service Specifications*)

Ta część normy dotyczy zasad komunikacji między sterownikami z różnych rodzin oraz z innymi urządzeniami. W połączeniu z normą ISO 9506 (*MMS*) specyfikującą zasady komunikacji w procesie produkcji określa ona funkcje adresowania urządzeń, wymiany danych, przetwarzania alarmów, sterowanie dostępem i administrowanie siecią.

W dalszym ciągu trwają prace nad udoskonaleniem i rozszerzeniem normy *IEC 61131*, ponieważ dotyczy ona nowoczesnej technologii. Stąd ciągły nacisk na wprowadzanie innowacji. Komitet IEC wydał dwa raporty, z których jeden zawiera propozycje rozszerzeń do normy (*Technical Report 2*), a drugi wskazówki dotyczące implementacji języków programowania dla sterowników programowalnych (*Technical Report 3*). Ponadto przedstawiona została korekta błędów dostrzeżonych po publikacji normy (ang. *Corrigendum*), a także dokument zawierający propozycje poprawek i udoskonaleń (ang. *Amendments*). W przygotowaniu jest również część dotycząca języka dla sterowania z wykorzystaniem zbiorów rozmytych (ang. *Fuzzy Control Language*).

W 1996r ukazały się polskie tłumaczenia pierwszej i drugiej części normy, oznaczone odpowiednio jako:

*PN-IEC 1131-1:1996. Sterowniki programowalne – Postanowienia ogólne*

oraz

*PN-IEC 1131-2:1996. Sterowniki programowalne – Wymagania i badania dotyczące sprzętu.*

W 1998 ukazała się trzecia część normy, oznaczona jako

*PN-EN 61131-3:1998. Sterowniki programowalne – Języki programowania*

a w 2002 r część piąta:

*PN-EN 61131-5:2002. Sterowniki programowalne Część5: Komunikacja.*

## Norma IEC 61131-3

Norma *IEC 61131-3* definiuje pojęcia podstawowe, zasady ogólne, model programowy i model komunikacyjny (wymiana danych między elementami oprogramowania) oraz podstawowe typy i struktury danych. Określono w niej dwie grupy języków programowania: języki tekstowe i graficzne.

W grupie *języków tekstowych* zdefiniowane zostały następujące języki:

- *Język IL* (ang. *Instruction List – Lista rozkazów*), będący odpowiednikiem języka typu *assembler*, którego zbiór instrukcji obejmuje operacje logiczne, arytmetyczne, operacje relacji, jak również funkcje przerzutników, czasomierzy, liczników itp.
- *Język ST* (ang. *Structured Text – Tekst strukturalny*), który jest odpowiednikiem języka algorytmicznego *wysokiego poziomu*, zawierającego struktury programowe i polecenia podobne do występujących w językach typu PASCAL lub C.

Do grupy *języków graficznych* opisanych w normie IEC 61131-3 należą:

- *Język LD* (ang. *Ladder Diagram – Schemat drabinkowy*), podobny do stykowych obwodów przekaźnikowych, w którym oprócz symboli styków, cewek i połączeń między nimi, dopuszcza się także użycie funkcji (np. arytmetycznych, logicznych, porównań, relacji) oraz bloków funkcjonalnych (np. przerzutniki, czasomierze, liczniki).
- *Język FBD* (ang. *Function Block Diagram – Funkcjonalny schemat blokowy*), będący odpowiednikiem schematu przepływu sygnału dla obwodów logicznych przedstawionych w formie połączonych bramek logicznych oraz funkcji i bloków funkcjonalnych, takich jak w języku LD.

Ponadto przedstawiono sposób tworzenia struktury wewnętrznej programu w postaci *schematu funkcji sekwencyjnej SFC* (ang. *Sequential Function Chart*), który pozwala na opisywanie zadań sterowania sekwencyjnego za pomocą grafów zawierających kroki (etapy) i warunki przejścia (tranzycji) między tymi krokami. W celu otrzymania odpowiedniej struktury programu można wykorzystać SFC, w którym definicje akcji dla poszczególnych kroków oraz warunki przejścia programuje się w jednym z czterech wymienionych wyżej języków.

IEC 61131-3 specyfikuje syntaktykę i semantykę wymienionych języków programowania. *Syntaktyka* (inaczej *składnia*) opisuje elementy języka i sposób ich użycia, natomiast *semantyka* ich znaczenie.

Definiowane są również elementy *konfiguracji*, które wspomagają instalację oprogramowania w sterownikach PLC, oraz możliwości *komunikacyjne* w celu ułatwienia połączenia sterowników z innymi elementami automatycznego systemu sterowania.

Wyróżniono następujące elementy języków programowania dla sterowników PLC:

- Typy danych (ang. *Data types*);
- Jednostki organizacyjne oprogramowania (ang. *Program organization units*);
- Elementy schematu funkcji sekwencyjnej (ang. *Sequential Function Chart*);
- Elementy konfiguracji (ang. *Configuration elements*).

*Typy danych* służą określeniu struktury danych w sterowniku, zarówno stałych jak i zmiennych, a w szczególności zakresu wartości jakie mogą przyjmować dane oraz obszaru pamięci potrzebnego do ich przechowywania.

*Jednostki organizacyjne oprogramowania* stanowią najmniejsze niezależne jednostki oprogramowania aplikacji użytkownika. Składają się na nie:

- Funkcje (ang. *Functions*),
- Bloki funkcjonalne (ang. *Function blocks*),
- Programy (ang. *Programs*).

*Elementy konfiguracji* wspomagają instalowanie i uruchamianie programów w systemach sterownikowych. Zalicza się do nich:

- Konfiguracje (ang. *Configurations*),
- Zasoby (ang. *Resources*),
- Zadania (ang. *Tasks*),
- Zmienne globalne (ang. *Global variables*),
- Ścieżki dostępu (ang. *Access paths*).

*Konfiguracja* jest elementem języka, który odpowiada *systemowi* sterowników programowalnych rozumianemu jako całość, obejmującą wszystkie pozostałe elementy oprogramowania. *Zasób* z kolei jest programowym odpowiednikiem sprzętu realizującego *funkcje przetwarzania sygnałów*, łącznie z funkcjami określonymi przez podłączone czujniki i elementy wykonawcze (ang. *sensor and actuator interface*) oraz urządzenia operatorskie *MMI* (ang. *Man-Machine Interface*).

## Struktura programu

*Jednostki organizacyjne oprogramowania, oznaczane dalej w skrócie POU (z ang. Program Organization Unit), takie jak programy, bloki funkcjonalne (w skrócie FB, ang. Function Block) czy funkcje stanowią najmniejsze, niezależne jednostki oprogramowania aplikacji użytkownika.*

POU mogą wywoływać się według przedstawionej wyżej kolejności, która odpowiada zmniejszającym się możliwościom funkcjonalnym, lub wywoływać się wzajemnie w ramach danego typu POU. Oznacza to, że z poziomu programu można wywoływać FB lub funkcję, z FB można wywoływać inny FB lub funkcję, a funkcja może wywoływać tylko inną funkcję. Niedopuszczalne jest wywoływanie rekurencyjne, tzn. POU nie może wywoływać siebie ani bezpośrednio, ani pośrednio, chociaż w językach stosowanych w komputerach PC takie wywołania są dopuszczalne. W normie nie występuje takie pojęcie jak podprogram (ang. *subroutine*) – rolę podprogramów występujących w innych językach pełnią tu bloki funkcjonalne i funkcje.

Podstawowa różnica między *funkcją* a *blokiem funkcjonalnym* polega na tym, że wywołanie funkcji z tymi samymi argumentami (parametrami wejściowymi) zawsze daje tę samą wartość na wyjściu (wartość funkcji), w przeciwieństwie do bloku funkcjonalnego, którego wywołanie z tymi samymi argumentami wejściowymi niekoniecznie musi prowadzić do tych samych wartości wyjściowych. FB posiada bowiem wewnętrzne zmienne zawierające pewną *informację o stanie* (można powiedzieć, że jest to „element dynamiczny”), podczas gdy funkcja nie zawiera wewnętrznej informacji o stanie (jest to więc „element statyczny”).

Wszystkie wartości zmiennych wyjściowych oraz konieczne wartości zmiennych wewnętrznych FB są przechowywane pomiędzy kolejnymi chwilami wykonania, stąd każdy występujący w programie FB posiada swoją własną *nazwę* oraz *strukturę danych* (w normie nazywane jest to *ukonkretnieniem* albo *egzemplarzem* – ang. *instance*). Każdy FB musi mieć zdefiniowanych tyle ukonkretnionych egzemplarzy, ile razy jest wywoływany w danym POU, co wynika stąd, że każdy egzemplarz musi mieć swoją własną strukturę danych do przechowywania stanu. Jeżeli np. w POU (programie lub FB) występuje konieczność użycia dwóch czasomierzy typu *TON*, to trzeba zadeklarować dwa egzemplarze (dwie nazwy) takiego czasomierza.

*Programy* stoją na szczycie hierarchii POU i mają możliwość dostępu do wejść i wyjść sterownika, zmiennych globalnych i ścieżek dostępu oraz udostępniania ich innym POU.

Każdy POU zawiera następujące elementy:

- typ i nazwa POU (w przypadku funkcji także typ danej wyjściowej);
- deklaracja zmiennych (wejściowych, wyjściowych i lokalnych);
- ciało POU (kod programu).

*Typ POU* określony jest przez odpowiednie słowo kluczowe: *PROGRAM*, *FUNCTION\_BLOCK* lub *FUNCTION*, po którym występuje nazwa własna POU. Dodatkowo w przypadku funkcji określa się także typ funkcji, np.:

*FUNCTION ALARM :BOOL;*

oznacza deklarację funkcji o nazwie *ALARM*, która na wyjściu daje wartość typu *BOOL*.

## Deklaracja zmiennych

Zmienne (ang. *variables*) wykorzystywane są do przechowywania i przetwarzania informacji. Umieszczone są one w pamięci danych sterownika, chociaż ich lokalizacja nie musi już być bezpośrednio wskazana przez użytkownika (w postaci podania konkretnego adresu w pamięci), w przeciwieństwie do stosowanych poprzednio metod deklaracji zmiennych w sterownikach. Wg normy deklarowane zmienne mogą być umieszczone w pamięci sterownika automatycznie przez system programujący, natomiast muszą mieć ustalony *typ danej* (ang. *data type*).

W normie IEC 61131-3 zdefiniowano pewną liczbę *elementarnych typów danych* (*BOOL*, *BYTE*, *INT* itd.), które różnią się zarówno wartościami, jakie mogą przyjmować, jak i liczbą zajmowanych bitów. Możliwe jest także definiowanie przez użytkownika swoich własnych typów danych, czyli *typów pochodnych*, w postaci np. tablic lub struktur.

Zmienne mogą być także *przyporządkowane* pewnym *adresom wejść* lub *wyjść* (zmienne reprezentowane bezpośrednio), oraz mogą być *podtrzymywane bateryjnie* w czasie zaniku napięcia zasilania.

Wszystkie używane w POU zmienne muszą być *zadeklarowane*, przy czym może to być zrobione poza POU (np. zmienne globalne), wewnątrz POU (zmienne lokalne) lub jako parametry wejściowe i wyjściowe. Na początku POU, po deklaracji jego nazwy, występuje część deklaracyjna dla zmiennych.

Przykład deklaracji zmiennych:

```
VAR                                (* początek deklaracji zmiennych w programie *)
  S1 AT %I1 : BOOL;                (* zmienna S1 typu BOOL, przypisana do wejścia %I1 *)
  S2 AT %I2 : BOOL;                (* zmienna S2 typu BOOL, przypisana do wejścia %I2 *)
  K3A      : BOOL;                (* zmienna K3A typu BOOL, bez podania adresu *)
  OPOZNIENIE : TON;              (* ukonkretnienie bloku czasomierza TON *)
END_VAR                             (* koniec deklaracji zmiennych *)
VAR RETAIN                             (* deklaracja zmiennych z podtrzymaniem bateryjnym *)
  K1 AT %Q1 : BOOL;                (* zmienna K1 typu BOOL, przypisana do wyjścia %Q1 *)
END_VAR                             (* koniec deklaracji zmiennych *)
```

W przedstawionym przykładzie zmiennej *K3A* nie przyporządkowano adresu w pamięci sterownika, a więc system programujący automatycznie dokona lokalizacji tej zmiennej w pamięci. Gdyby programista żądał przechowywania tej zmiennej w konkretnym bicie pamięci sterownika, to deklaracja tej zmiennej wyglądałaby w następujący sposób:

```
K3A AT %M1 : BOOL;
```

Deklaracja *OPOZNIENIE : TON*; informuje o tym, że w programie będzie używany blok funkcjonalny typu *TON* o konkretnej nazwie *OPOZNIENIE* i system programujący musi zarezerwować odpowiednią strukturę danych dla tego bloku. Struktura tych danych jest zdefiniowana przez deklarację standardowego bloku *TON*.

Kwalifikator *RETAIN* umożliwia określenie tych zmiennych, których wartości powinny być przechowywane w pamięci w czasie zaniku napięcia zasilania, tu dotyczy to zmiennej *K1* przyporządkowanej do wyjścia *%Q1*.

## Kod jednostki oprogramowania

Zadanie do zaprogramowania:

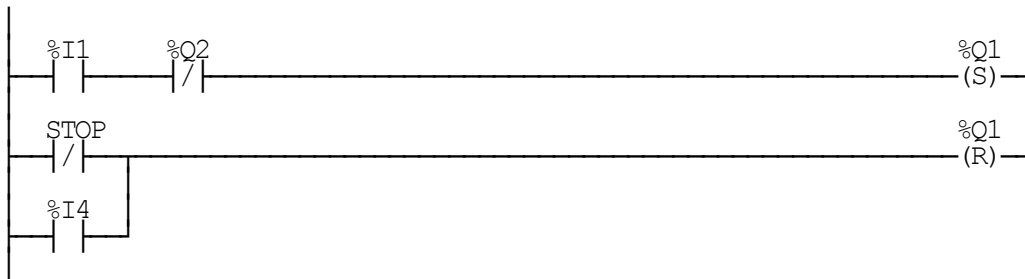
„Jeżeli zmienna *STOP* ma wartość 0 lub wejście *%I4* ma wartość 1, to wyłącz wyjście *%Q1*, w przeciwnym razie jeżeli wejście *%I1* ma wartość 1 i wyjście *%Q2* ma wartość 0, to załącz wyjście *%Q1*, w przeciwnym razie pozostaw *%Q1* bez zmian”.

lub inaczej:

$$\%Q1 := NOT (NOT STOP OR \%I4) AND (\%Q1 OR \%I1 AND NOT \%Q2)$$

Zadanie zrealizować przy pomocy przerzutnika RS.

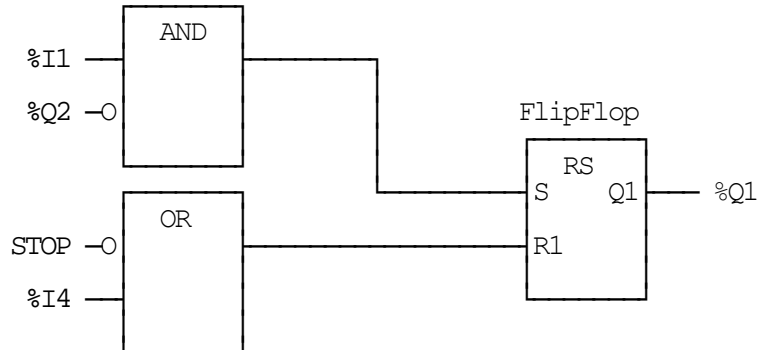
Przykład w języku LD:



Przykład w języku IL:

```
(* pierwszy szczebel drabinki *)
LD    %I1
ANDN  %Q2
S     %Q1
(* drugi szczebel drabinki *)
LDN   STOP
OR    %I4
R     %Q1
```

Przykład w języku FBD:



Przykład w języku ST:

```
(* wywołanie przerzutnika *)
FlipFlop( S:= %I1 AND NOT %Q2, R1:=NOT STOP OR %I4 );
(* ustawienie wyjścia *)
%Q1 := FlipFlop.Q1
```



## Elementy wspólne języków

*Ograniczniki* (ang. *delimiters*) są to znaki specjalne, takie jak + - \$ = := # ; ( ) \* oraz *spacja*, których znaczenie omówione będzie w dalszej części podręcznika. Użytkownik może wstawić spację wszędzie w tekstach programów, za wyjątkiem słów kluczowych, literałów oraz identyfikatorów.

*Słowa kluczowe* (ang. *keywords*) są identyfikatorami (nazwami) standardowymi, zdefiniowanymi w normie jako elementy danego języka. Słowa kluczowe nie mogą być używane jako identyfikatory wprowadzane przez użytkownika (nazwy własne). W zasadzie słowa kluczowe powinny być pisane z użyciem dużych liter. Do słów kluczowych należą:

- nazwy deklaracji, np. *PROGRAM*, *FUNCTION*, *VAR\_INPUT*, *END\_PROGRAM*;
- nazwy elementarnych typów danych, np. *BOOL*, *INT*, *REAL*, *TIME*;
- nazwy standardowych funkcji i bloków funkcjonalnych, np. *AND*, *ADD*, *MOVE*, *SHL*, *TON*, *CTU*;
- nazwy parametrów wejściowych i wyjściowych standardowych funkcji i bloków funkcjonalnych;
- operatory języka IL;
- operatory i instrukcje języka ST;
- elementy SFC;
- zmienne *EN* i *ENO* w językach graficznych.

*Literały* (ang. *literals*) służą przedstawianiu wartości danych (zmiennych lub stałych). Ich format zależy od typu danej, który jednocześnie określa zakres zmienności.

*Identyfikatory* (ang. *identifiers*) są ciągami znaków alfanumerycznych, których programista może użyć do nazwania własnych: zmiennych, jednostek POU itp. Nazwa taka musi zaczynać się od litery lub pojedynczego znaku podkreślenia \_, po których może występować dowolna liczba liter, cyfr lub znaków podkreślenia. Jednak dopuszczalna długość nazwy może zależeć od konkretnego systemu programującego. Norma IEC 61131-3 wymaga tylko, aby przynajmniej sześć pierwszych znaków nazwy było znaczących. Nie rozróżnia się między dużymi a małymi literami, np. nazwy takie jak *ALA*, *Ala* czy *ala* powinny być traktowane przez system programujący jednakowo.

Programista może także wstawić do tekstu programu swoje *komentarze* w dowolnym miejscu, w którym może używać spacji. Jedynie w języku IL komentarze powinny być wstawione na końcu. Komentarze ograniczone są przez kombinację znaków (\* oraz \*). Komentarze nie mają żadnego znaczenia składniowego ani semantycznego, służą jedynie do przedstawienia dodatkowej informacji użytkownikowi. Nie mogą być one zagnieżdżane, tzn. nie można wstawiać komentarza do komentarza.

## Literaly

Przedstawianie danych w postaci liczbowej:

Opis	Przykłady
Liczby całkowite	<i>-12 0 123_456 +986</i>
Liczby rzeczywiste	<i>-12.0 0.0 0.456 3.14159_26</i>
Liczby rzeczywiste z wykładnikami	<i>-1.34E-12 lub -1.34e-12 1.234E6 lub 1.234e6</i>
Liczby dwójkowe	<i>2#1111_1111 (255) 2#1110_0000 (240)</i>
Liczby ósemkowe	<i>8#377 (255) 8#340 (240)</i>
Liczby szesnastkowe	<i>16#FF lub 16#ff (255) 16#E0 lub 16#e0 (240)</i>
Boolowskie zero i jedynka	<i>0 1</i>
Boolowskie FAŁSZ i PRAWDA	<i>FALSE TRUE</i>

Przykłady przedstawiania danych w postaci czasów trwania:

Opis	Przykłady
Literaly bez podkreśleń z przedrostkiem krótkim	<i>T#14ms T#14.7s T#14.7m T#14.7h t#14.7d t#5d14h12m18s3.5ms</i>
Literaly bez podkreśleń z przedrostkiem długim	<i>TIME#14ms time#14.7s</i>
Literaly z podkreśleniem z przedrostkiem krótkim	<i>T#14ms T#14.7s T#14.7m T#14.7h t#25h_15m t#5d_14h_12m_18s_3.5ms</i>
Literaly z podkreśleniem z przedrostkiem długim	<i>TIME#25h_15m time#5d_14h_12m_18s_3.5ms</i>

Przedrostki używane przy przedstawianiu danych w postaci godziny dnia i daty:

<b>Lp</b>	<b>Opis</b>	<b>Przedrostek</b>
1	Data (ang. <i>date</i> )	<i>DATE# D#</i>
2	Godzina dnia (ang. <i>time of day</i> )	<i>TIME_OF_DAY# TOD#</i>
3	Data i godzina (ang. <i>date and time</i> )	<i>DATE_AND_TIME# DT#</i>

Przykłady przedstawiania danych w postaci daty i godziny dnia:

<b>Zapis z długim przedrostkiem</b>	<b>Zapis z krótkim przedrostkiem</b>
<i>DATE#1984-06-25</i> <i>date#1984-06-25</i>	<i>D#1984-06-25</i> <i>d#1984-06-25</i>
<i>TIME_OF_DAY#15:36:55.36</i> <i>time_of_day#15:36:55.36</i>	<i>TOD#15:36:55.36</i> <i>tod#15:36:55.36</i>
<i>DATE_AND_TIME#1984-06-25-15:36:55.36</i> <i>date_and_time#1984-06-25-15:36:55.36</i>	<i>DT#1984-06-25-15:36:55.36</i> <i>dt#1984-06-25-15:36:55.36</i>

## Elementarne typy danych

Lp.	Słowo kluczowe	Typ danych	Liczba bitów
1	<i>BOOL</i>	Boolowski	1
2	<i>SINT</i>	Liczba całkowita krótka	8
3	<i>INT</i>	Liczba całkowita	16
4	<i>DINT</i>	<i>Liczba całkowita podwójna</i>	32
5	<i>LINT</i>	Liczba całkowita długa	64
6	<i>USINT</i>	Liczba całkowita krótka bez znaku	8
7	<i>UINT</i>	Liczba całkowita bez znaku	16
8	<i>UDINT</i>	Liczba całkowita podwójna bez znaku	32
9	<i>ULINT</i>	Liczba całkowita długa bez znaku	64
10	<i>REAL</i>	Liczba rzeczywista	32
11	<i>LREAL</i>	Liczba rzeczywista długa	64
12	<i>TIME</i>	Czas trwania	
13	<i>DATE</i>	Data	
14	<i>TIME_OF_DAY</i> lub <i>TOD</i>	Godzina dnia	
15	<i>DATE_AND_TIME</i> lub <i>DT</i>	Data i czas	
16	<i>STRING</i>	Ciąg znaków o zmiennej długości	
17	<i>BYTE</i>	Bajt – ciąg 8 bitów	8
18	<i>WORD</i>	Słowo – ciąg 16 bitów	16
19	<i>DWORD</i>	Słowo podwójne – ciąg 32 bitów	32
20	<i>LWORD</i>	Słowo długie – ciąg 64 bitów	64

Przegląd uniwersalnych typów danych:

<i>ANY</i>					
<i>ANY_BIT</i>	<i>ANY_NUM</i>			<i>ANY_DATE</i>	
	<i>ANY_INT</i>		<i>ANY_REAL</i>		
<i>BOOL</i>	<i>INT</i>	<i>UINT</i>	<i>REAL</i>	<i>DATE</i>	<i>TIME</i>
<i>BYTE</i>	<i>SINT</i>	<i>USINT</i>	<i>LREAL</i>	<i>TIME_OF_DAY</i>	<i>STRING</i>
<i>WORD</i>	<i>DINT</i>	<i>UDINT</i>		<i>DATE_AND_TIME</i>	i typy
<i>DWORD</i>	<i>LINT</i>	<i>ULINT</i>			poходne
<i>LWORD</i>					

## Pochodne typy danych

Przykłady deklaracji typów pochodnych:

Lp.	Przykład
1	Alias, nowa nazwa dla typu elementarnego: <i>TYPE FLOATING : REAL ; END_TYPE</i>
2	Typ wyliczeniowy (ang. <i>Enumerated data type</i> ): <i>TYPE</i> <i>ANALOG_SIGNAL_TYPE : (SINGLE_ENDED, DIFFERENTIAL);</i> <i>END_TYPE</i>
3	Typ okrojony (ang. <i>Subrange data type</i> ) <i>TYPE</i> <i>ANALOG_DATA : INT (-4095..4095);</i> <i>END_TYPE</i>
4	Deklaracja tablicy danych (ang. <i>Array data type</i> ) <i>TYPE</i> <i>ANALOG_16_INPUT: ARRAY [1..16] OF ANALOG_DATA;</i> <i>ANALOG_ARRAY: ARRAY[1..4,1..16] OF ANALOG_DATA;</i> <i>END_TYPE</i>
5	Deklaracje typu strukturalnego (ang. <i>Structured data type</i> ) <i>TYPE</i> <i>ANALOG_CHANNEL_CONFIG :</i> <i>STRUCT</i> <i>RANGE : ANALOG_RANGE;</i> <i>MIN_SCALE : ANALOG_DATA;</i> <i>MAX_SCALE : ANALOG_DATA;</i> <i>END_STRUCT;</i> <i>ANALOG_16_INPUT_CONFIG :</i> <i>STRUCT</i> <i>SIGNAL_TYPE : ANALOG_SIGNAL_TYPE;</i> <i>FILTER_PARAMETER : SINT (0..99);</i> <i>CHANNEL : ARRAY [1..16] OF ANALOG_CHANNEL_CONFIG;</i> <i>END_STRUCT;</i> <i>END_TYPE</i>

## Deklaracja wartości początkowych dla typu danych

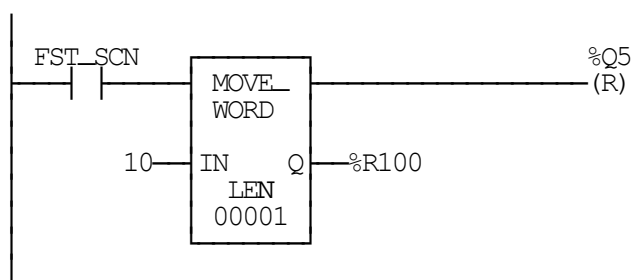
Domyślne wartości początkowe dla danych typu elementarnego:

Typy danych	Wartości początkowe
<i>BOOL, SINT, INT, DINT, LINT</i>	<i>0</i>
<i>USINT, UINT, UDINT, ULINT</i>	<i>0</i>
<i>BYTE, WORD, DWORD, LWORD</i>	<i>0</i>
<i>REAL, LREAL</i>	<i>0.0</i>
<i>TIME</i>	<i>T#0S</i>
<i>DATE</i>	<i>D#0001-01-01</i>
<i>TIME_OF_DAY</i>	<i>TOD#00:00:00</i>
<i>DATE_AND_TIME</i>	<i>DT#0001-01-01-00:00:00</i>
<i>STRING</i>	“ (pusty ciąg znaków)

Przykłady deklaracji wartości początkowych:

Lp.	Przykład
1	<i>TYPE PI : REAL := 3.1415925 ; END_TYPE</i>
2	<i>TYPE ANALOG_RANGE :</i> <i>(BIPOLAR_10V, (* -10 do +10 VDC *)</i> <i>UNIPOLAR_1_10V, (* +1 do +10 VDC *)</i> <i>UNIPOLAR_0_10V, (* 0 do +10 VDC *)</i> <i>UNIPOLAR_1_5V, (* +1 do +10 VDC *)</i> <i>UNIPOLAR_0_5V, (* 0 do +10 VDC *)</i> <i>UNIPOLAR_4_20MA, (* +4 do +20 mADC *)</i> <i>UNIPOLAR_0_20MA, (* 0 do +20 mADC *)</i> <i>) := UNIPOLAR_1_5V ;</i> <i>END_TYPE</i>
3	<i>TYPE ANALOG_DATAZ : INT (-4095..4095) := 0 ; END_TYPE</i>
4	<i>TYPE</i> <i>ANALOG_16_INPUT: ARRAY[1..16] OF ANALOG_DATA := 8(-4095), 8(4095);</i> <i>END_TYPE</i>

Przykład ustawienia warunków początkowych w sterowniku GE Fanuc:



## Zmienne

Słowa kluczowe do deklaracji zmiennych:

Typ zmiennej	Opis
<i>VAR</i>	Deklaracja zmiennych <i>wewnętrznych</i> w POU.
<i>VAR_INPUT</i>	Deklaracja zmiennych dostarczanych do POU z <i>zewnątrz</i> , nie mogą być zmieniane w POU.
<i>VAR_OUTPUT</i>	Deklaracja zmiennych <i>wyprowadzanych</i> z POU na zewnątrz do innych POU.
<i>VAR_IN_OUT</i>	Deklaracja zmiennych <i>dostarczanych</i> do POU z zewnątrz i <i>wyprowadzanych</i> na zewnątrz, mogą być zmieniane.
<i>VAR_EXTERNAL</i>	Deklaracja użycia zmiennych zdefiniowanych w konfiguracji jako <i>VAR_GLOBAL</i> , mogą być zmieniane.
<i>VAR_GLOBAL</i>	Deklaracja zmiennych <i>globalnych</i> .
<i>VAR_ACCESS</i>	Deklaracja <i>ścieżek dostępu</i> .

Przykład deklaracji zmiennej:

(\*↓ typ zmiennych      ↓ atrybut – opcjonalnie \*)  
*VAR\_OUTPUT*            *RETAIN*

(\*↓ nazwa zmiennej    ↓ typ danej   ↓ wartość początkowa – opcjonalnie \*)  
*Moja\_zmienna*    : *DINT* := 100;

*END\_VAR* (\* koniec deklaracji bloku zmiennych \*)



## Zmienne proste

Przedrostki określające położenie i rozmiar zmiennej:

Lp.	Przedrostek	Opis
1	<i>I</i>	Położenie – wejście (ang. <i>Input</i> )
2	<i>Q</i>	Położenie – wyjście (ang. <i>Output</i> )
3	<i>M</i>	Położenie – pamięć (ang. <i>Memory</i> )
4	<i>X</i>	Rozmiar – pojedynczy bit
5	brak	Rozmiar – pojedynczy bit
6	<i>B</i>	Rozmiar – bajt (8 bitów)
7	<i>W</i>	Rozmiar – słowo (16 bitów)
8	<i>D</i>	Rozmiar – słowo podwójne (32 bity)
9	<i>L</i>	Rozmiar – słowo poczwórne (64 bity)

Jeśli nie zadeklarowano inaczej, to zmienne jednobitowe są typu *BOOL*

Przykłady deklaracji i użycia zmiennych prostych:

*VAR*

(\* zmienne reprezentowane bezpośrednio \*)

*AT %IW1 : INT;* (\* słowo wejściowe o adresie 1, zmienna typu *INT* \*)

*AT %QD8 : DINT;* (\* podwójne słowo wyjściowe rozpoczynające się od adresu 8, zmienna typu *DINT* \*)

(\* zmienne symboliczne ulokowane \*)

*XXX AT %QW3 : INT;* (\* słowo wyjściowe o adresie 3, zmienna typu *INT* \*)

*YYY AT %QX16 : BOOL;* (\* bit 16 na wyjściu \*)

(\* zmienne symboliczne z automatycznym przydziałem pamięci \*)

*A,B,C : INT;* (\* przydzielenie 3 kolejnych słów pamięci na 3 zmienne typu *INT* \*)

*DDD : DINT* (\* przydzielenie kolejnych 2 słów pamięci na zmienną typu *DINT* \*)

*RRR : REAL;* (\* przydzielenie kolejnych 2 słów pamięci na zmienną typu *REAL* \*)

*END\_VAR*

(\* ciało *POU* w języku *IL* \*)

...

*LD %IW1* (\* użycie bezpośredniej reprezentacji zmiennej \*)

*ST XXX* (\* użycie zmiennej symbolicznej *XXX* \*)

...

Deklaracja typu danych umożliwia systemowi programującemu sprawdzenie poprawności użycia zmiennej w POU już w trakcie kompilacji. Wprowadzenie np. do programu ciągu instrukcji:

```
LD %IWI (*ładuj do akumulatora zmienną typu INT *)  
ST DDD (*prześlij zawartość akumulatora do zmiennej typu DINT *)
```

dla deklaracji zmiennych jw. powinno spowodować w trakcie kompilacji zasygnalizowanie błędu wynikającego z mieszania zmiennych należących do różnych typów danych.

## Zmienne wieloelementowe

Zmiennymi wieloelementowymi są tablice i struktury.

Przykłady deklaracji zmiennych wieloelementowych:

```
VAR  
WEJSCIA AT %IWI : ARRAY[0..3] OF INT; (* tablica jednowymiarowa *)  
INPUT_TAB : ANALOG_ARRAY; (* tablica dwuwymiarowa *)  
MODULE_CONFIG : ANALOG_16_INPUT_CONFIG; (* struktura *)  
END_VAR
```

*Tablica* jest zbiorem elementów tego samego typu, do których dostęp uzyskuje się za pomocą jednego lub więcej *indeksów* umieszczonych w nawiasach kwadratowych i oddzielonych przecinkami.

Zmienna w postaci tablicy może być zadeklarowana przy użyciu słowa kluczowego *ARRAY* z podaniem zakresu zmian indeksów, albo jako zmienna należąca do typu danych, który został zdefiniowany jako tablica.

*Zmienną strukturalną* jest zmienna deklarowana jako struktura, której elementy należą do różnych typów danych. Kolejne elementy struktury reprezentowane są przez identyfikatory oddzielone kropkami. Zmienna o nazwie *MODULE\_CONFIG* jest zadeklarowana jako struktura. Przypisanie wartości *SINGLE\_ENDED* do elementu *SIGNAL\_TYPE* tej zmiennej można przedstawić w następujący sposób:

```
MODULE_CONFIG.SIGNAL_TYPE:=SINGLE_ENDED.
```

Jednym z elementów zmiennej strukturalnej *MODULE\_CONFIG* jest także tablica *CHANNEL*, która zawiera 16 elementów będących strukturami zadeklarowanymi jako typ *ANALOG\_CHANNEL\_CONFIG*, w którym z kolei występuje element *RANGE*. Przypisanie wartości *BIPOLAR\_10V* do elementu *RANGE* struktury, która jest piątym elementem tablicy *CHANNEL* w strukturze *MODULE\_CONFIG* można zapisać więc jako:

```
MODULE_CONFIG.CHANNEL[5].RANGE:=BIPOLAR_10V.
```

## Wartości początkowe zmiennych

Z chwilą „wystartowania” elementu konfiguracji, każda ze zmiennych skojarzonych z takim elementem lub przypisanym mu programem powinna zostać zainicjowana przez przypisanie jej miejsca w pamięci, przy czym jej wartość początkowa zależy od informacji podanej przez programistę w deklaracji zmiennej. Może więc to być wartość:

- jaką posiadała przed „zatrzymaniem” danego elementu konfiguracji;
- początkowa, zadeklarowana przez użytkownika;
- domyślna, zdefiniowana przez typ danych, do jakiego należy zmienna.

Za pomocą atrybutu *RETAIN* użytkownik może zadeklarować, że wartość zmiennej powinna być zachowana na czas zatrzymania zasobu, z którym jest skojarzona, pod warunkiem, że własność taką zapewnia zastosowany sprzęt. Zmienna, której wartość w czasie zatrzymania zasobu jest *zachowywana* w pamięci podtrzymywanej bateryjnie nosi nazwę *zmiennej podtrzymywanej* (ang. *retentive variable*).

Nadawanie wartości początkowych zmiennym *podtrzymywanym* podlega następującym regułom:

- W przypadku tzw. *ciepłego restartu* wartości początkowe tych zmiennych powinny być równe ich wartościom *zachowanym* w pamięci;
- W przypadku tzw. *zimnego restartu* wartości początkowe powinny być równe wartościom *zadeklarowanym* przez użytkownika, a w przypadku braku takiej deklaracji wartościom *domyślnym*.

*Ciepły restart* (ang. *warm restart*) występuje w przypadku powrotu napięcia zasilającego po jego zaniku. Natomiast *zimny restart* (ang. *cold restart*) związany jest z uruchomieniem sterownika (przejściem w tryb wykonywania) po załadowaniu programu do sterownika lub zatrzymaniu spowodowanym wystąpieniem błędu.

Zmiennym nie podtrzymywanym (ang. *non-retentive variables*) nadawane są wartości początkowe tak, jak przy zimnym restarcie, a zmiennym reprezentującym wejścia sterownika – wartości zależne od rozwiązania zastosowanego przez producenta sterownika.

## Atrybuty zmiennych

<i>Atrybut</i>	<i>Opis</i>
<i>RETAIN</i>	zmienna podtrzymywana
<i>CONSTANT</i>	stała (tzn. zmienna niemodyfikowana)
<i>R_EDGE</i>	zmienna reaguje tylko na zbocze narastające
<i>F_EDGE</i>	zmienna reaguje tylko na zbocze opadające
<i>READ_ONLY</i>	zmienna może być tylko czytana
<i>READ_WRITE</i>	zmienna może być czytana i zapisywana

Przykłady deklaracji atrybutów dla zmiennych:

```

VAR CONSTANT          (* deklaracja stałej *)
  PI : REAL := 3.1415925;
END_VAR
VAR_OUTPUT RETAIN     (* deklaracja zmiennej wyjściowej podtrzymywanej *)
  Q : WORD;
END_VAR
VAR_INPUT             (*deklaracja zmiennych wejściowych *)
  WEJ1 : BOOL R_EDGE; (* reagującej tylko na zbocze narastające *)
  WEJ2 : BOOL F_EDGE; (* reagującej tylko na zbocze opadające *)
END_VAR
VAR_ACCESS            (* deklaracja ścieżki dostępu *)
  CSX : PI.Z : REAL READ_ONLY (* dostęp do zmiennej tylko do odczytu *);
END_VAR

```

Atrybut *CONSTANT* służy do deklaracji „zmiennej”, której wartość nie zmienia się w trakcie wykonywania programu, należy więc ją traktować jako *stałą*. Nie zawsze w tekście programu należy wpisywać wartość stałą jako *literal*. Lepiej czasami jest zadeklarować stałą w bloku deklaracji zmiennych. Nie powoduje to zwiększenia zajętości pamięci (a czasami pozwala wręcz na zaoszczędzenie), natomiast nie prowadzi do niejednoznaczności.

Np. literal *1* może zarówno oznaczać daną typu *BOOL*, jak i liczbę całkowitą typu *INT*, *UINT*, *SINT* itp. Natomiast w przypadku użycia deklaracji:

```

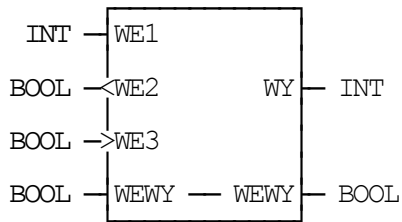
VAR CONSTANT
  Jedynka : INT := 1;
  Jeden   : DINT := 1;
END_VAR

```

stała *Jeden* różni się od stałej *Jedynka* typem danej, mimo że obie stałe mają tę samą wartość. A co za tym idzie, podlegają one działaniom czy funkcjom określonym dla innych typów danych, np. stałej *Jeden* nie można użyć jako parametru wejściowego w funkcji *MUL\_INT*.

## Deklaracja zmiennych w sposób graficzny

Przykłady deklaracji zmiennych wejściowych i wyjściowych w sposób graficzny i tekstowy:



```
VAR_INPUT
  WE1 : INT;
  WE2 : BOOL F_EDGE;
  WE3 : BOOL R_EDGE;
END_VAR
VAR_OUTPUT
  WY : INT;
END_VAR
VAR_IN_OUT
  WEWY : BOOL;
END_VAR
```

## Jednostki organizacyjne oprogramowania (POU)

Wyróżnione zostały trzy typy POU:

- funkcje,
- bloki funkcjonalne,
- programy.

Nazwa POU musi być unikalną nazwą w ramach jednego projektu, ponieważ po zadeklarowaniu POU jej nazwa i interfejs zewnętrzny są dostępne wszystkim innym POU w projekcie. Nie ma więc tu możliwości deklaracji lokalnych podprogramów, tak jak zwykle jest to możliwe w językach programowania komputerów PC.

POU mogą być dostarczane przez producentów (dotyczy to w szczególności FFB standardowych) lub programowane przez użytkownika (tzw. FFB pochodne).

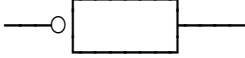
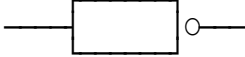
## Funkcje

Podstawową zasadą tworzenia funkcji jest, by instrukcje zawarte w ciele funkcji zastosowane do zmiennych wejściowych zawsze dostarczały jednoznaczny wynik w postaci wartości funkcji, bez względu na to jak często i w jakiej chwili funkcja jest wywoływana. W tym sensie funkcje mogą być traktowane jako rozszerzenie podstawowego zbioru operacji sterownika.

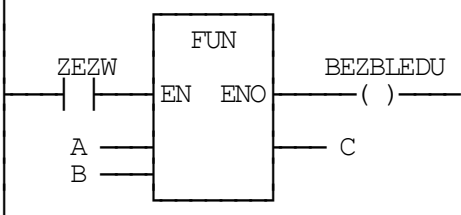
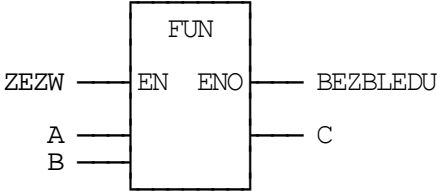
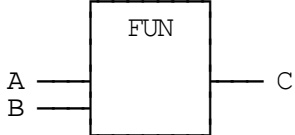
W językach tekstowych funkcje używane mogą być jako operatory (rozkazy) w języku IL lub jako operandy w wyrażeniach języka ST.

W językach graficznych funkcje reprezentowane są w postaci prostokątów o wymiarach zależnych od liczby wejść.

Graficzne przedstawianie negacji sygnałów boolowskich:

Lp.	Opis	Symbol
1	Negacja wejścia	
2	Negacja wyjścia	

Przykłady użycia wejścia *EN* i wyjścia *ENO*:

Lp.	Opis	Przykład
1	Język <i>LD</i> – użycie <i>EN</i> i <i>ENO</i> jest obowiązkowe.	
2	Język <i>FBD</i> – użycie <i>EN</i> i <i>ENO</i> jest opcjonalne.	
3	Użycie funkcji w języku <i>FBD</i> bez pary <i>EN/ENO</i> .	

Para *EN /ENO* nie występuje w językach tekstowych, stąd pewna trudność w bezpośrednim tłumaczeniu programu z języka graficznego na tekstowy.

### **Deklaracja funkcji pochodnej (ang. *derived function*)**

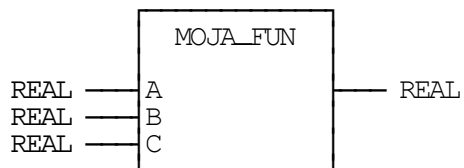
Przykład deklaracji funkcji w języku ST:

```
FUNCTION MOJA_FUN : REAL (* Nazwa funkcji i jej typ *)  
  VAR INPUT (* Parametry wejściowe *)  
    A, B : REAL;  
    C : REAL:=1.0; (* Nadanie wartości początkowej *)  
  END_VAR  
  MOJA_FUN:=A*B/C; (* Obliczenie wartości wyjściowej funkcji *)  
END_FUNCTION
```

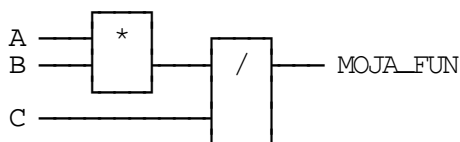
Przykład deklaracji funkcji w języku FBD:

FUNCTION

(\* Nazwa funkcji, typ wyjścia funkcji i parametry wejściowe \*)



(\* Ciało funkcji \*)



END\_FUNCTION

## Bloki funkcjonalne

*Blok funkcjonalny (FB, ang. function block)* jest jednostką organizacyjną oprogramowania, która z chwilą wykonania może dostarczać na wyjściu jedną lub wiele wartości, w przeciwieństwie do funkcji, która ma tylko jedno wyjście.

Wywołanie FB z tymi samymi parametrami wejściowymi niekoniecznie musi prowadzić do tych samych wartości wyjściowych, ponieważ blok posiada *strukturę danych* zawierającą informację o *stanie bloku* (jest więc „elementem dynamicznym”). Tak więc wszystkie wartości zmiennych wyjściowych, konieczne wartości zmiennych wewnętrznych oraz, w zależności od implementacji, wartości parametrów wejściowych lub odwołania do nich są przechowywane pomiędzy kolejnymi chwilami wykonania FB.

W programie FB mogą być wykorzystywane wielokrotnie, ale dla każdego wywołania musi być utworzona odpowiednia struktura danych, w której przechowywana będzie informacja o stanie wywoływanego FB. Tworzenie takiej struktury danych nazywa się tworzeniem egzemplarza FB lub ukonkretnianiem (ang. *instantiation*). Każdy utworzony egzemplarz (ang. *instance*) musi posiadać swój identyfikator (swoją nazwę).

Deklaracja zmiennej i deklaracja egzemplarza FB:

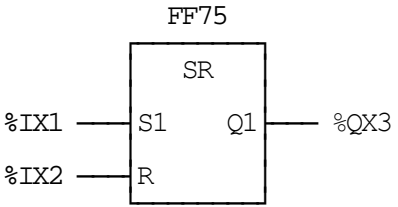
```
VAR
(* ↓ Nazwa                ↓ Typ danej lub FB *)
Ala      :      BOOL;    (* deklaracja zmiennej *)
Licznik1 :      CTU;    (* deklaracja egzemplarza FB *)
Licznik2 :      CTU;    (* deklaracja innego egzemplarza FB *)
END_VAR
```

Przykład struktury danych dla licznika CTU:

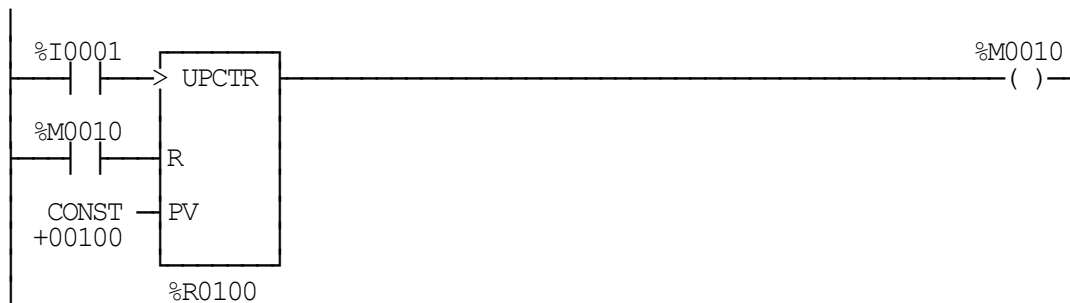
```
TYPE CTU:
STRUCT
(* wejścia *)
CU :      BOOL R_EDGE;    (* licz w górę *)
R  :      BOOL;          (* zeruj licznik *)
PV :      INT;           (* wartość zadana *)
(* wyjścia *)
Q  :      BOOL;          (* wyjście załączone *)
CV :      INT;           (* wartość bieżąca *)
END_STRUCT;
END_TYPE
```



Przykład tworzenia egzemplarza i wywołania bloku funkcjonalnego:

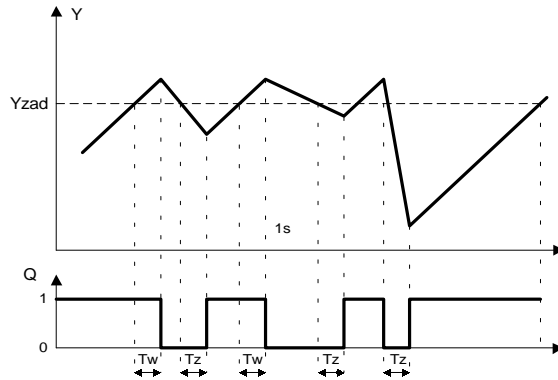
Graficznie (w języku FBD)	Tekstowo (w języku ST)
	<pre> (* Deklaracja *) VAR FF75: SR; END_VAR  (* Wywołanie *) FF75(S1:=%IX1, R:=%IX2);  (* Przypisanie wyjścia *) %QX3:= FF75.Q1; </pre>

Przykład użycia bloku licznika UPCTR w sterowniku GE Fanuc:



## Deklaracja bloku funkcjonalnego pochodnego (ang. *derived function block*)

Przebiegi czasowe dla przekaźnika dwupołożeniowego z histerezą czasową



Przykład deklaracji bloku funkcjonalnego w języku ST realizującego zadanie przekaźnika:

### *FUNCTION\_BLOCK PRZEKAZNIK*

(\* Łącze zewnętrzne \*)

*VAR\_INPUT*

*We* : *BOOL*;

*Y* : *INT*;

*Yzad* : *INT*;

*Tz* : *TIME* := *t#100ms*;

*Tw* : *TIME* := *t#100ms*;

*END\_VAR*

*VAR\_OUTPUT*

*Q* : *BOOL*;

*END\_VAR*

*VAR*

*Timer\_ON* : *TON*;

*Timer\_OFF* : *TON*;

*FlipFlop* : *SR*;

*END\_VAR*

(\* Ciało Bloku Funkcjonalnego \*)

*IF We THEN*

*IF (Y < Yzad) THEN*

*Timer\_ON(IN:=We, PT:=Tz);*

*END\_IF*

*IF (Y > Yzad) THEN*

*Timer\_OFF(IN:=We, PT:=Tw);*

*END\_IF*

*FlipFlop(S:=Timer\_ON.Q, R1:=Timer\_OFF.Q);*

*Q:=FlipFlop.Q1;*

*END\_IF*

*END\_FUNCTION\_BLOCK*

(\* Wejścia \*)

(\* Wejście uruchamiające FB, domyślnie = 0 \*)

(\* Wartość zadana \*)

(\* Opóźnienie załączenia, domyślnie = 100ms \*)

(\* Opóźnienie wyłączenia, domyślnie = 100ms \*)

(\* Wyjście \*)

(\* Domyślnie = 0 \*)

(\* Zmienne wewnętrzne, lokalne \*)

(\* nazwy egzemplarzy bloków funkcjonalnych \*)

(\* Uruchomienie przekaźnika \*)

(\* Y poniżej Yzad \*)

(\* ustaw czas, po którym ma być załączony \*)

(\* Y powyżej Yzad \*)

(\* ustaw czas, po którym ma być wyłączony \*)

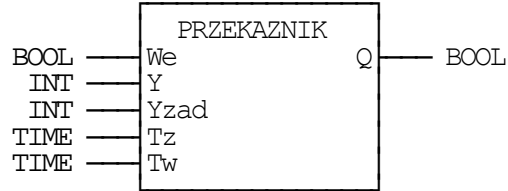
(\* wywołanie przekaźnika \*)

(\* nadanie wyjścia \*)

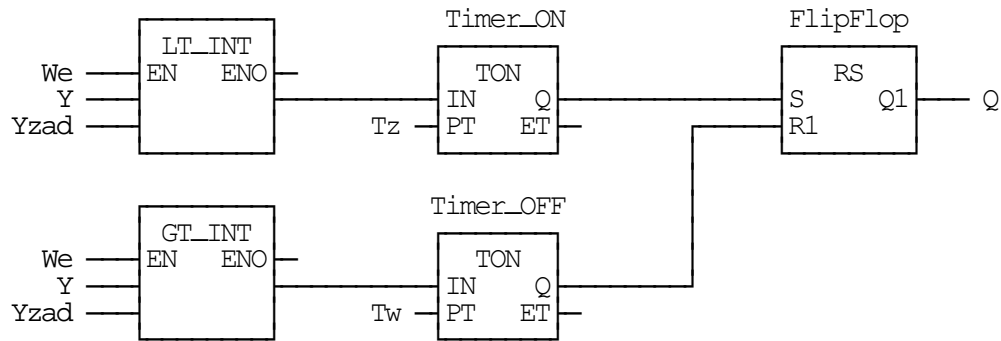
## Przykład deklaracji bloku funkcjonalnego w języku FBD:

FUNCTION\_BLOCK

(\* Łącze zewnętrzne \*)



(\*\* Ciało Bloku Funkcjonalnego \*\*)



END\_FUNCTION\_BLOCK

## Wywoływanie FFB

Wzajemne wywoływanie POU podlega następującym zasadom:

- Programy mogą wywoływać funkcje lub bloki funkcjonalne (FB), ale nie odwrotnie;
- FB mogą wywoływać inne FB;
- FB mogą wywoływać funkcje, ale nie odwrotnie;
- Funkcje mogą wywoływać tylko inne funkcje.

Niedozwolone jest wywoływanie rekursywne, tzn. POU nie może wywoływać siebie samego, ani bezpośrednio, ani w sposób pośredni, chociaż wywoływanie rekursywne jest dopuszczone w innych językach programowania, np. dla komputerów PC. Gdyby dopuszczone było wywołanie rekursywne, to system programujący sterownik nie byłby w stanie obliczyć maksymalnej pamięci potrzebnej w chwili wykonania programu rekursywnego. Wywoływanie rekursywne można łatwo zastąpić przez tworzenie odpowiednich iteracji, np. przez pętlę.

Przykład rekursywnego wywołania funkcji w języku ST:

```
FUNCTION FF1 :BOOL;  
  VAR_INPUT  
    X : INT;  
  END_VAR  
  
  IF FF1(X) THEN ..... (* Zabronione rekursywne wywołanie definiowanej funkcji *)  
  
  END_IF  
  
END_FUNCTION
```

Przykład rekursywnego wywołania egzemplarza FB w języku ST:

```
FUNCTION_BLOCK FB1  
  VAR_INPUT  
    We1 : INT;  
  END_VAR  
  VAR  
    MojFB : FB1; (*Zabronione rekursywne tworzenie egzemplarza definiowanego FB *)  
    XX : INT;  
  END_VAR  
  
  FB1(XX);      (*Zabronione rekursywne wywołanie definiowanego FB *)  
  
END_FUNCTION_BLOCK
```

Przykład rekursywnego wywołania funkcji w sposób pośredni:

*(\* Deklaracja funkcji FF1 \*)*

*FUNCTION FF1 : BOOL;*

*VAR\_INPUT*

*X : INT;*

*END\_VAR*

*VAR*

*Y : REAL;*

*END\_VAR*

*...*

*Y:=FF2(0.0);*            *(\*Zabronione wywołanie funkcji FF2 powodujące rekurencję \*)*

*...*

*END\_FUNCTION*

*(\* Deklaracja funkcji FF2 \*)*

*FUNCTION FF2 : REAL;*

*VAR\_INPUT*

*X : REAL;*

*END\_VAR*

*...*

*IF FF1(5) THEN .....*    *(\*Zabronione wywołanie funkcji FF1 powodujące rekurencję \*)*

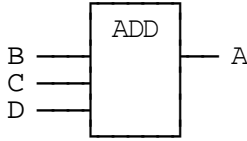
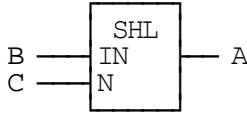
*...*

*END\_IF*

*END\_FUNCTION*

Wywołanie funkcji lub FB powoduje przekazanie parametrów wejściowych do zmiennych wejściowych wywoływanego POU. Zmienne wejściowe zadeklarowane w definicji POU są nazywane *parametrami formalnymi*, natomiast wprowadzane do nich dane wejściowe nazywa się *parametrami aktualnymi*, by podkreślić, że zawierają one aktualne wartości wejść.

Przykłady użycia nazw parametrów formalnych:

Przykład	Opis
	Graficzne przedstawienie użycia funkcji <i>ADD</i> w języku FBD – brak nazw parametrów formalnych.
$A := ADD(B, C, D);$	Użycie funkcji <i>ADD</i> w języku ST bez nazw parametrów formalnych.
	Graficzne przedstawienie wywołania funkcji <i>SHL</i> . Występują nazwy parametrów formalnych.
$A := SHL(IN:=B, N:=C);$	Użycie funkcji <i>SHL</i> w języku ST z nazwami parametrów formalnych.

Funkcje lub FB mogą być wywoływane nawet wtedy, gdy lista parametrów wejściowych jest niekompletna lub przedstawiona w innej kolejności niż występuje w deklaracji POU. W takim przypadku nazwy parametrów formalnych muszą pojawić się w sposób jawny, aby system programujący mógł prawidłowo przypisać odpowiednie parametry aktualne do formalnych.

Jeżeli lista parametrów jest niekompletna, to pominiętym wejściowym parametrom formalnym nadawane są wartości początkowe – zdefiniowane przez użytkownika lub domyślne. Takie rozwiązanie zapewnia, że zmienne wejściowe zawsze mają nadaną wartość.

Przykład deklaracji FB o trzech parametrach wejściowych:

```

FUNCTION_BLOCK Fblok    (* Deklaracja FB *)
  VAR_INPUT
    Par1 : BOOL;
    Par2 : TIME;
    Par3 : INT;
  END_VAR
  (* ciało FB *)
END_FUNCTION_BLOCK

```

Przykłady w języku IL wywołania FB z różnymi listami wejść:

*VAR*

*FB1, FB2, FB3 : Fblok; (\* utworzenie 3 egzemplarzy bloku Fblok \*)*

*AT %I1 : BOOL;*

*AT %IW1 : INT;*

*END\_VAR*

*(\* Wywołanie z pełną listą wejść \*)*

*CAL FB1(Par1 := %I1, Par2 := t#10s, Par3:= %IW1)*

*(\* Wywołanie z listą wejść w innej kolejności \*)*

*CAL FB2(Par3 := %IW1, Par1 := %I1, Par2:= t#10s)*

*(\* Wywołanie z niepełną listą wejść \*)*

*CAL FB1(Par2 := t#10s, Par1:= %I1)*

## Funkcje standardowe

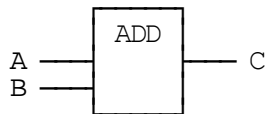
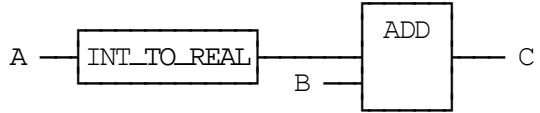
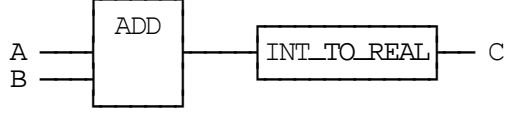
W normie wyróżniono siedem grup funkcji standardowych:

1. Funkcje konwersji typów (ang. *Type conversion functions*);
2. Funkcje liczbowe (ang. *Numerical functions*);
3. Funkcje na ciągach bitów (ang. *Bit string functions*);
4. Funkcje wyboru i porównania (ang. *Selection and comparison functions*);
5. Funkcja na ciągach znaków (ang. *Character string functions*);
6. Funkcje na typach danych związanych z czasem (ang. *Functions of time data types*);
7. Funkcje na wyliczeniowych typach danych (ang. *Functions of enumerated data types*).

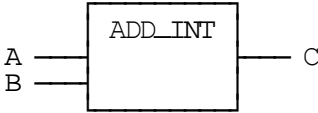
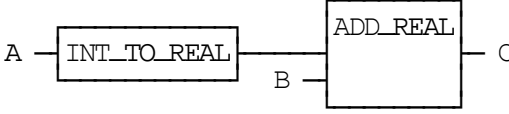
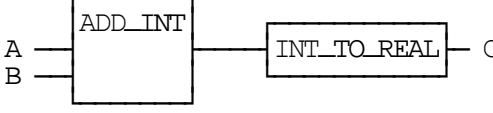
Przykład funkcji przeciążonej i funkcji o nadanych typach.

Lp.	Opis	Przykład
1	Funkcja przeciążona (ang. <i>overloaded function</i> ) – tu dla danych liczbowych typu uniwersalnego <i>ANY_NUM</i>	
2	Funkcja o nadanych typach (ang. <i>typed function</i> ) – tu dla parametrów typu <i>INT</i> .	

Przykłady konwersji typów danych dla funkcji przeciążonych.

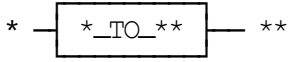

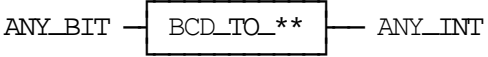

Lp.	Deklaracja zmiennych	Przykłady użycia w językach FBD i ST
1	<pre>VAR   A : INT ;   B : INT ;   C : INT ; END_VAR</pre>	 <pre>C := A + B; (* ST *)</pre>
2	<pre>VAR   A : INT ;   B : REAL ;   C : REAL ; END_VAR</pre>	 <pre>C := INT_TO_REAL(A) + B; (* ST *)</pre>
3	<pre>VAR   A : INT ;   B : INT ;   C : REAL ; END_VAR</pre>	 <pre>C := INT_TO_REAL(A + B); (* ST *)</pre>

Przykłady konwersji typów dla funkcji o nadanych typach.

Lp.	Deklaracja zmiennych	Przykłady użycia w językach FBD i ST
1	<pre>VAR   A : INT ;   B : INT ;   C : INT ; END_VAR</pre>	 <pre>C := ADD_INT(A, B); (* ST *)</pre>
2	<pre>VAR   A : INT ;   B : REAL ;   C : REAL ; END_VAR</pre>	 <pre>C := ADD_REAL(INT_TO_REAL(A), B); (* ST *)</pre>
3	<pre>VAR   A : INT ;   B : INT ;   C : REAL ; END_VAR</pre>	 <pre>C := INT_TO_REAL(ADD_INT(A, B)); (* ST *)</pre>



Funkcje konwersji typów:

Lp.	Forma graficzna	Przykład użycia w języku ST
1	* —  **	<i>A:=INT_TO_REAL(B);</i>
2	ANY_REAL —  ANY_INT	<i>A:=TRUNC(B);</i>
3	ANY_BIT —  ANY_INT	<i>A:=BCD_TO_INT(C);</i>
4	ANY_INT —  ANY_BIT	<i>C:=INT_TO_BCD(A);</i>

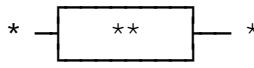
Dla przedstawionych w tabeli przykładów w języku ST założono, że użyte w nich zmienne zostały zadeklarowane jako:

```
VAR
  A :INT;
  B :REAL;
  C :WORD;
END_VAR
```

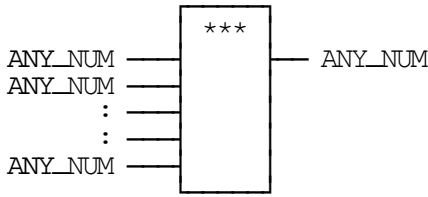
W języku IL pierwszy przykład można by zapisać w następujący sposób (pozostałe wyglądałyby podobnie):

```
LD B
INT_TO_REAL
ST A
```

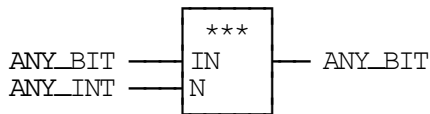
Standardowe funkcje liczbowe jednej zmiennej:

Forma graficzna		Przykłady użycia w językach ST i IL	
 <p>* – oznacza typ wejścia/wyjścia (I/O) ** – oznacza nazwę funkcji</p>		<p><math>A := SIN(B);</math> (* język ST *)</p> <p><math>LD B</math> (* język IL *) <math>SIN</math> <math>ST A</math></p>	
Lp.	Nazwa funkcji	Typ I/O	Opis
Funkcje podstawowe			
1	<i>ABS</i>	<i>ANY_NUM</i>	Wartość bezwzględna
2	<i>SQRT</i>	<i>ANY_REAL</i>	Pierwiastek kwadratowy
Funkcje logarytmiczne			
3	<i>LN</i>	<i>ANY_REAL</i>	Logarytm naturalny
4	<i>LOG</i>	<i>ANY_REAL</i>	Logarytm dziesiętny
5	<i>EXP</i>	<i>ANY_REAL</i>	Funkcja wykładnicza o podstawie $e$
Funkcje trygonometryczne			
6	<i>SIN</i>	<i>ANY_REAL</i>	Sinus kąta w radianach
7	<i>COS</i>	<i>ANY_REAL</i>	Cosinus kąta w radianach
8	<i>TAN</i>	<i>ANY_REAL</i>	Tangens kąta w radianach
9	<i>ASIN</i>	<i>ANY_REAL</i>	Arcus sinus
10	<i>ACOS</i>	<i>ANY_REAL</i>	Arcus cosinus
11	<i>ATAN</i>	<i>ANY_REAL</i>	Arcus tangens

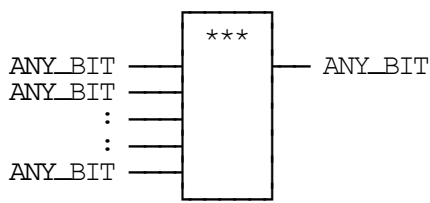
Standardowe funkcje arytmetyczne:

Forma graficzna			Przykład użycia w językach ST i IL	
 <p>*** – oznacza nazwę funkcji lub symbol</p>			<p><math>A := ADD(B, C, D);</math> (* język ST *)  lub  <math>A := B + C + D;</math></p> <p><i>LD B</i> (* język IL *)  <i>ADD C</i>  <i>ADD D</i>  <i>ST A</i></p>	
Lp.	Nazwa	Symbol	Opis	
Funkcje arytmetyczne rozszerzalne (o zmiennej liczbie wejść)				
12	<i>ADD</i>	+	Dodawanie	$OUT := IN_1 + IN_2 + \dots + IN_n$
13	<i>MUL</i>	*	Mnożenie	$OUT := IN_1 * IN_2 * \dots * IN_n$
Funkcje arytmetyczne o stałej liczbie wejść				
14	<i>SUB</i>	-	Odejmowanie	$OUT := IN_1 - IN_2$
15	<i>DIV</i>	/	Dzielenie	$OUT := IN_1 / IN_2$
16	<i>MOD</i>		Reszta z dzielenia	$OUT := IN_1 \text{ modulo } IN_2$
17	<i>EXPT</i>	**	Potęgowanie	$OUT := IN_1^{IN_2}$
18	<i>MOVE</i>	:=	Przepisanie	$OUT := IN$

### Standardowe funkcje przesuwania bitów

Forma graficzna		Przykład użycia w językach ST i IL
 <p style="text-align: center;">*** – oznacza nazwę funkcji</p>		<p><math>A := SHL(IN := B, N := 5);</math> (* język ST *)</p> <p><i>LD B</i> (* język IL *)  <i>SHL 5</i>  <i>ST A</i></p>
Lp.	Nazwa	Opis
1	<i>SHL</i>	Przesuń bity w argumencie <i>IN</i> o <i>N</i> pozycji w lewo wprowadzając 0 na pozycje bitów z prawej strony
2	<i>SHR</i>	Przesuń bity w argumencie <i>IN</i> o <i>N</i> pozycji w prawo wprowadzając 0 na pozycje bitów z lewej strony
3	<i>ROR</i>	Przesuń cyklicznie w prawo <i>N</i> bitów w argumencie <i>IN</i> (rotacja – bity z prawej strony przechodzą na lewą)
4	<i>ROL</i>	Przesuń cyklicznie w lewo <i>N</i> bitów w argumencie <i>IN</i> (rotacja – bity z lewej strony przechodzą na prawą)

### Standardowe funkcje Boolowskie

Forma graficzna		Przykład użycia w językach ST i IL	
 <p style="text-align: center;">*** – oznacza nazwę funkcji lub symbol</p>		<p><math>A := AND(B, C, D);</math> (* język ST *)  lub  <math>A := B \&amp; C \&amp; D;</math></p> <p><i>LD B</i> (* język IL *)  <i>AND C</i>  <i>AND D</i>  <i>ST A</i></p>	
Lp.	Nazwa	Symbol	Opis
5	<i>AND</i>	$\&$	Mnożenie Boolowskie $OUT := IN_1 \& IN_2 \& \dots \& IN_n$
6	<i>OR</i>	$\geq 1$	Suma Boolowska $OUT := IN_1 OR IN_2 OR \dots OR IN_n$
7	<i>XOR</i>	$= 2k + 1$	Suma wykluczająca $OUT := IN_1 XOR IN_2 XOR \dots XOR IN_n$
8	<i>NOT</i>		Negacja $OUT := NOT IN_1$

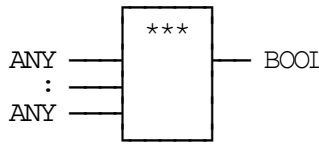
Standardowe funkcje wyboru

Lp.	Forma graficzna	Opis i przykłady w języku ST
1		<p>Wybór wartości: <math>OUT := IN0</math> jeśli <math>G=0</math>  <math>OUT := IN1</math> jeśli <math>G=1</math></p> <p>Przykład:  <math>A := SEL(G := 0, IN0 := X, IN1 := 255);</math>  daje w wyniku <math>A := X</math>.</p>
2a		<p>Wybór wartości maksymalnej:  <math>OUT := MAX\{IN_1, IN_2, \dots, IN_n\}</math></p> <p>Przykład:  <math>A := MAX(B, C, D);</math></p>
2b		<p>Wybór wartości minimalnej:  <math>OUT := MIN\{IN_1, IN_2, \dots, IN_n\}</math></p> <p>Przykład:  <math>A := MIN(B, C, D);</math></p>
3		<p>Ogranicznik:  <math>OUT := MIN\{MAX\{IN, MN\}, MX\}</math></p> <p>Przykład:  <math>A := LIMIT(IN := B, MN := 0, MX := 255);</math>  daje w wyniku <math>A := 0</math> gdy <math>B &lt; 0</math>, <math>A := 255</math> gdy <math>B &gt; 255</math>,  poza tym <math>A := B</math>.</p>
4		<p>Multiplexer – wybiera jedno z <math>N</math> wejść w zależności od wartości wejścia <math>K</math>.</p> <p>Przykład:  <math>A := MUX(K := 0, IN0 := B, IN1 := C, IN2 := D);</math>  daje w wyniku <math>A := B</math>.</p>

Przedstawiony w tabeli przykład dla multipleksera w języku ST w przypadku języka IL wyglądałby następująco:

LD 0  
MUX B, C, D  
ST A

## Standardowe funkcje porównania

Forma graficzna			Przykład użycia w języku ST
 <p style="text-align: center;">*** – oznacza nazwę funkcji lub symbol</p>			$A := GT(B, C, D);$ lub $A := (B > C) \& (C > D);$
Lp	Nazwa	Symbol	Opis
5	<i>GT</i>	>	=1 gdy kolejne wejścia tworzą sekwencję malejącą $OUT := (IN_1 > IN_2) \& (IN_2 > IN_3) \& \dots \& (IN_{n-1} > IN_n)$
6	<i>GE</i>	>=	=1 gdy kolejne wejścia tworzą sekwencję nie rosnącą $OUT := (IN_1 >= IN_2) \& (IN_2 >= IN_3) \& \dots \& (IN_{n-1} >= IN_n)$
7	<i>EQ</i>	=	=1 gdy wszystkie wejścia są sobie równe $OUT := (IN_1 = IN_2) \& (IN_2 = IN_3) \& \dots \& (IN_{n-1} = IN_n)$
8	<i>LE</i>	<=	=1 gdy kolejne wejścia tworzą sekwencję nie malejącą $OUT := (IN_1 <= IN_2) \& (IN_2 <= IN_3) \& \dots \& (IN_{n-1} <= IN_n)$
9	<i>LT</i>	<	=1 gdy kolejne wejścia tworzą sekwencję rosnącą $OUT := (IN_1 < IN_2) \& (IN_2 < IN_3) \& \dots \& (IN_{n-1} < IN_n)$
10	<i>NE</i>	<>	=1 gdy kolejne wejścia różnią się między sobą $OUT := (IN_1 <> IN_2) \& (IN_2 <> IN_3) \& \dots \& (IN_{n-1} <> IN_n)$

W przedstawionym przykładzie porównania w języku ST zakłada się, że zmienna *A* jest typu *BOOL*, a pozostałe zmienne są dowolnego typu (ale wszystkie tego samego) należącego do typu uniwersalnego *ANY*.

Ten sam przykład można zapisać w języku IL jako:

```
LD B
GT C    (* Wynik porównania (C > B) staje się wynikiem bieżącym CR typu BOOL *)
AND(    (* Wynik bieżący CR jest zapamiętywany *)
LD C
GT D    (* Wynik porównania (D > C) staje się wynikiem bieżącym CR typu BOOL *)
)       (* Koniec zagnieżdżenia – wykonanie AND na CR bieżącym i CR poprzednim *)
ST A    (* Zapamiętanie wyniku bieżącego w zmiennej A *)
```

Standardowe funkcje na ciągach znaków

Lp.	Forma graficzna	Opis i przykłady w języku ST
1		<p>Obliczanie długości ciągu. Np.  <math>A:=LEN('ASTRING');</math>                      daje w wyniku <math>A:=7</math>.</p>
2		<p><math>L</math> znaków z lewej strony ciągu <math>IN</math>. Np.  <math>A:=LEFT(IN:='ASTR', L:=3)</math>                      daje w wyniku <math>A:='AST'</math>.</p>
3		<p><math>L</math> znaków z prawej strony ciągu <math>IN</math>. Np.  <math>A:=RIGHT(IN:='ASTR', L:=3);</math>                      daje w wyniku <math>A:='STR'</math>.</p>
4		<p><math>L</math> znaków z ciągu <math>IN</math> począwszy od znaku <math>P</math>-tego. Np.  <math>A:=MID(IN:='ASTR', L:=2, P:=2);</math>                      daje w wyniku <math>A:='ST'</math>.</p>
5		<p>Łączenie ciągów. Np.  <math>A:=CONCAT('AB', 'CD', 'E');</math>                      daje w wyniku <math>A:='ABCDE'</math>.</p>
6		<p>Wstawienie ciągu <math>IN2</math> do ciągu <math>IN1</math> po <math>P</math>-tym znaku. Np.  <math>A:=INSERT(IN1:='ABC', IN2:='XY', P:=2);</math>                      daje w wyniku <math>A:='ABXYC'</math>.</p>
7		<p>Kasowanie <math>L</math> znaków w ciągu <math>IN</math>, począwszy od <math>P</math>-tego znaku. Np.  <math>A:=DELETE(IN:='ABXYC', L:=2, P:=3);</math>                      daje w wyniku <math>A:='ABC'</math>.</p>
8		<p>Zastąpienie <math>L</math> znaków w ciągu <math>IN1</math> przez ciąg <math>IN2</math> począwszy od <math>P</math>-tego znaku. Np.  <math>A:=REPLACE(IN1:='ABCD', IN2:='X', L:=2, P:=2);</math>                      daje w wyniku <math>A:='AXD'</math>.</p>
9		<p>Znalezienie miejsca pierwszego pojawienia się ciągu <math>IN2</math> w ciągu <math>IN1</math>. Np.  <math>A:=FIND(IN1:='ABCBC', IN2:='BC');</math>                      daje w wyniku <math>A:=2</math>.</p>

Standardowe funkcje na typach danych związanych z czasem

<b>Funkcje numeryczne i konkatencji</b>					
<b>Lp.</b>	<b>Nazwa</b>	<b>Symbol</b>	<b>IN1</b>	<b>IN2</b>	<b>OUT</b>
1	<i>ADD</i>	+	<i>TIME</i>	<i>TIME</i>	<i>TIME</i>
2			<i>TIME_OF_DAY</i>	<i>TIME</i>	<i>TIME_OF_DAY</i>
3			<i>DATE_AND_TIME</i>	<i>TIME</i>	<i>DATE_AND_TIME</i>
4	<i>SUB</i>	-	<i>TIME</i>	<i>TIME</i>	<i>TIME</i>
5			<i>DATE</i>	<i>DATE</i>	<i>TIME</i>
6			<i>TIME_OF_DAY</i>	<i>TIME</i>	<i>TIME_OF_DAY</i>
7			<i>TIME_OF_DAY</i>	<i>TIME_OF_DAY</i>	<i>TIME</i>
8			<i>DATE_AND_TIME</i>	<i>TIME</i>	<i>DATE_AND_TIME</i>
9			<i>DATE_AND_TIME</i>	<i>DATE_AND_TIME</i>	<i>TIME</i>
10	<i>MUL</i>	*	<i>TIME</i>	<i>ANY_NUM</i>	<i>TIME</i>
11	<i>DIV</i>	/	<i>TIME</i>	<i>ANY_NUM</i>	<i>TIME</i>
12	<i>CONCAT</i>		<i>DATE</i>	<i>TIME_OF_DAY</i>	<i>DATE_AND_TIME</i>
<b>Funkcje konwersji typów</b>					
13	<i>DATE_AND_TIME_TO_TIME_OF_DAY</i>				
14	<i>DATE_AND_TIME_TO_DATE</i>				



## **Standardowe bloki funkcjonalne**

Wśród FB standardowych wyróżnia się następujące grupy:

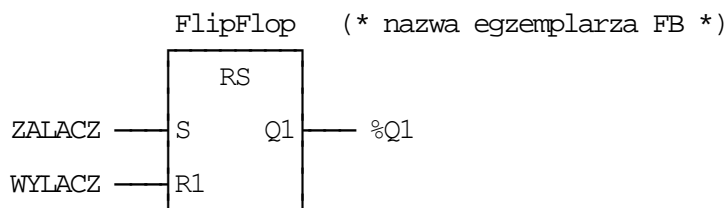
1. Elementy dwustanowe (ang. *Bistable elements*);
2. Elementy detekcji zbocza (ang. *Edge detection elements*);
3. Liczniki (ang. *Counters*);
4. Czasomierze (ang. *Timers*).

## Standardowe dwustanowe bloki funkcjonalne

Lp	Forma graficzna	Opis
1		Przerzutnik SR <i>S1</i> – wejście ustawiające (dominujące) <i>R</i> – wejście zerujące
2		Przerzutnik RS <i>S</i> – wejście ustawiające <i>R1</i> – wejście zerujące (dominujące)
3		Semafor <i>CLAIM</i> – wejście ustawiające semafor <i>RELEASE</i> – wejście zwalniające semafor

Przykłady użycia bloku RS w językach FBD, ST i IL:

(\* język FBD \*)



(\* języki tekstowe \*)

VAR

FlipFlop : RS;

(\* deklaracja egzemplarza FB \*)

END\_VAR

(\* język ST \*)

FlipFlop(S:=ZALACZ, R1:=WYLACZ);

(\* wywołanie \*)

%Q1:=FlipFlop.Q1;

(\* nadanie wartości wyjściu \*)

(\* język IL \*)

CAL FlipFlop(S:=ZALACZ, R1:=WYLACZ); (\* wywołanie \*)

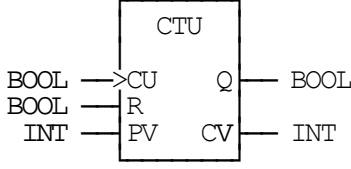
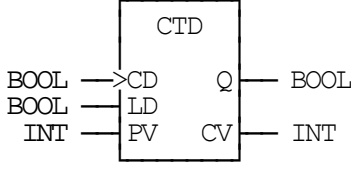
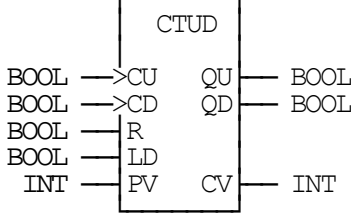
LD FlipFlop.Q1

ST %Q1

(\* nadanie wartości wyjściu \*)

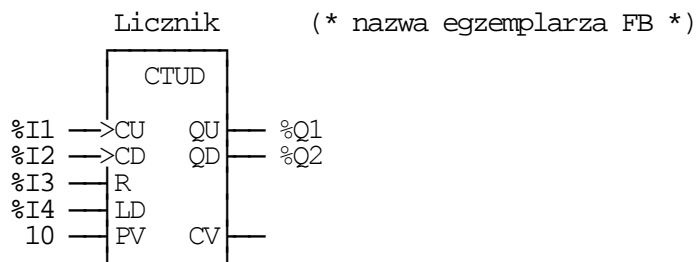


## Standardowe liczniki

Lp	Forma graficzna	Opis
1		<p>Licznik dodający</p> <p><i>CU</i> – wejście, którego zmiany z 0 na 1 są zliczane</p> <p><i>R</i> – wejście zerujące licznik</p> <p><i>PV</i> – wartość zadana</p> <p><i>Q</i> – wyjście załączane gdy <i>CV</i> osiągnie wartość <i>PV</i></p> <p><i>CV</i> – liczba zliczonych impulsów</p>
2		<p>Licznik odejmujący</p> <p><i>CD</i> – wejście, którego zmiany z 0 na 1 są zliczane</p> <p><i>LD</i> – wejście ustawiające <i>CV</i> na wartość <i>PV</i></p> <p><i>PV</i> – wartość zadana</p> <p><i>Q</i> – wyjście załączane gdy <i>CV</i> osiągnie wartość 0</p> <p><i>CV</i> – liczba zliczonych impulsów</p>
3		<p>Licznik dodająco-odejmujący</p> <p><i>CU</i> – wejście, jego zmiany z 0 na 1 są zliczane w górę</p> <p><i>CD</i> – wejście, jego zmiany z 0 na 1 są zliczane w dół</p> <p><i>R</i> – wejście zerujące licznik</p> <p><i>LD</i> – wejście ustawiające <i>CV</i> na wartość <i>PV</i></p> <p><i>PV</i> – wartość zadana</p> <p><i>QU</i> – wyjście załączane gdy <i>CV</i> osiągnie wartość <i>PV</i></p> <p><i>QD</i> – wyjście załączane gdy <i>CV</i> osiągnie wartość 0</p> <p><i>CV</i> – liczba zliczonych impulsów</p>

## Przykład użycia bloku CTUD w językach FBD, ST i IL

(\* język FBD \*)



(\* języki tekstowe \*)

VAR

Licznik : CTUD;

(\* deklaracja egzemplarza FB \*)

END\_VAR

(\* język ST \*)

Licznik(CU:=%I1, CD:=%I2, R:=%I3, LD:=%I4); (\* wywołanie \*)

%Q1:= Licznik.QU; (\* nadanie wartości wyjściu %Q1 \*)

%Q2:= Licznik.QD; (\* nadanie wartości wyjściu %Q2 \*)

(\* język IL \*)

CAL Licznik(CU:=%I1, CD:=%I2, R:=%I3, LD:=%I4); (\* wywołanie \*)

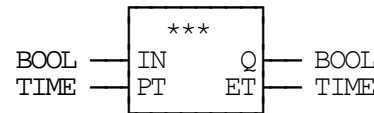
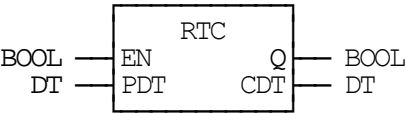
LD Licznik.QU

ST %Q1 (\* nadanie wartości wyjściu %Q1 \*)

LD Licznik.QD

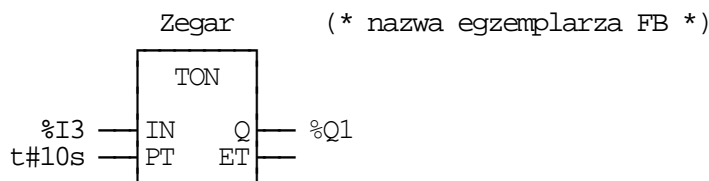
ST %Q2 (\* nadanie wartości wyjściu %Q2 \*)

## Standardowe czasomierze

№p	Forma graficzna	Opis
1 2 3	 <p>*** oznacza: <i>TP, TON, TOF</i></p>	<i>TP</i> – generator impulsu <i>TON</i> – opóźnione załączenie <i>TOF</i> – opóźnione wyłączenie <i>IN</i> – wejście uruchamiające czasomierz <i>PT</i> – wartość zadana czasu <i>ET</i> – czas mierzony
4		Zegar czasu rzeczywistego <i>PDT</i> – ustalone data i czas (ładowane przy zboczu narastającym na wejściu <i>EN</i> ) <i>CDT</i> – bieżące data i czas (gdy <i>EN = 1</i> ) <i>Q</i> – kopia <i>EN</i>

Przykład użycia czasomierza TON w języku FBD, ST i IL:

(\* język FBD \*)



(\* języki tekstowe \*)

VAR

Zegar : TON;

(\* deklaracja egzemplarza FB \*)

END\_VAR

(\* język ST \*)

Zegar(IN:=%I3, PT:=t#10s);

(\* wywołanie \*)

%Q1:= Zegar.Q;

(\* nadanie wartości wyjściu %Q1 \*)

(\* język IL \*)

CAL Zegar(IN:=%I3, PT:=t#10s)

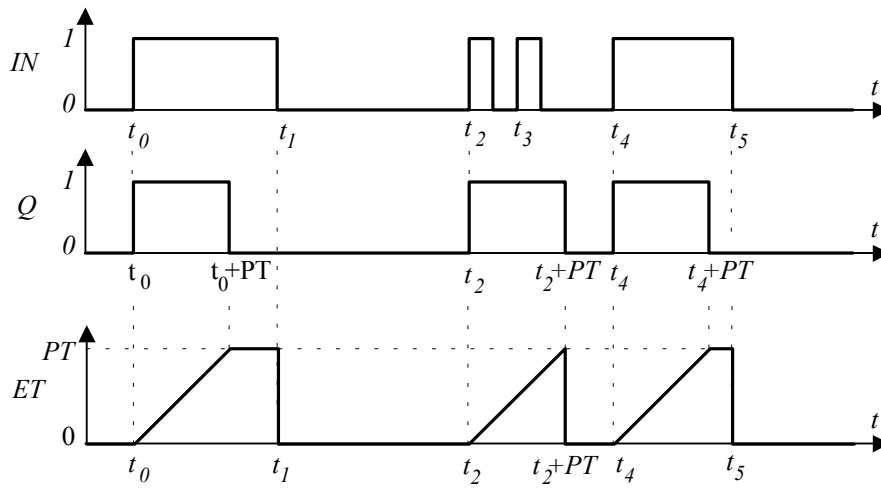
(\* wywołanie \*)

LD Zegar.Q

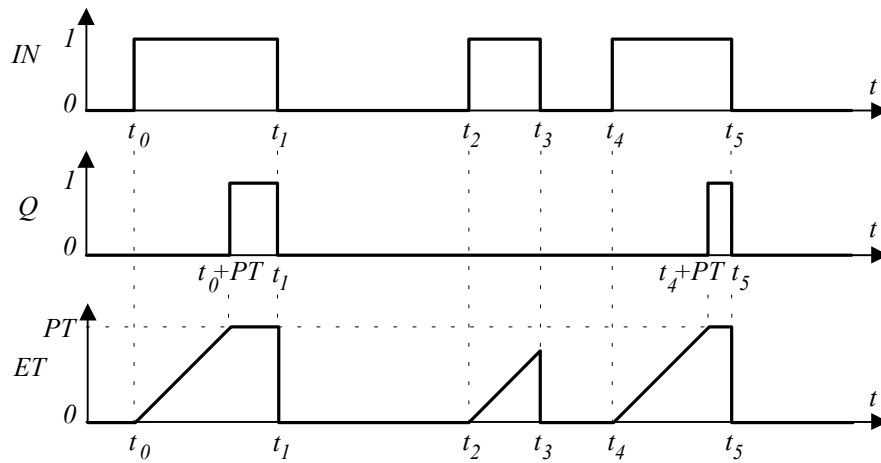
ST %Q1

(\* nadanie wartości wyjściu %Q1 \*)

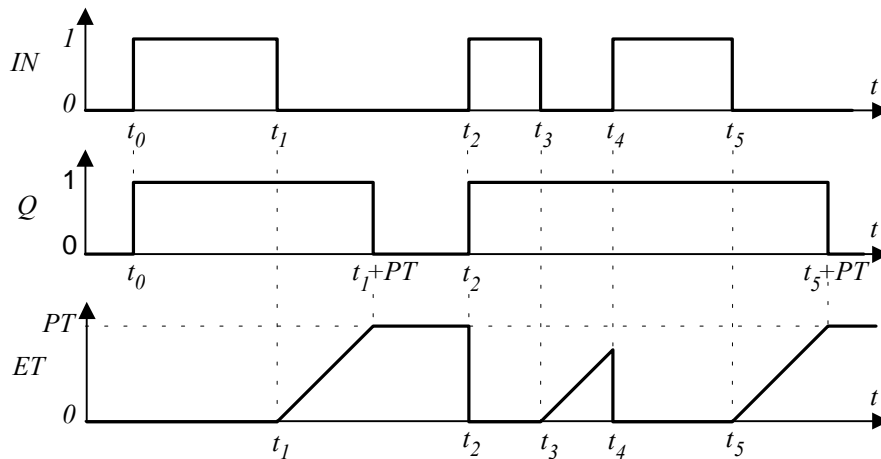
Wykresy czasowe sygnałów dla generatora impulsu TP:



Wykresy czasowe sygnałów dla czasomierza załączającego TON:



Wykresy czasowe sygnałów dla czasomierza wyłączającego TOF:



## Język LD (Schemat drabinkowy)

Symbole styków:

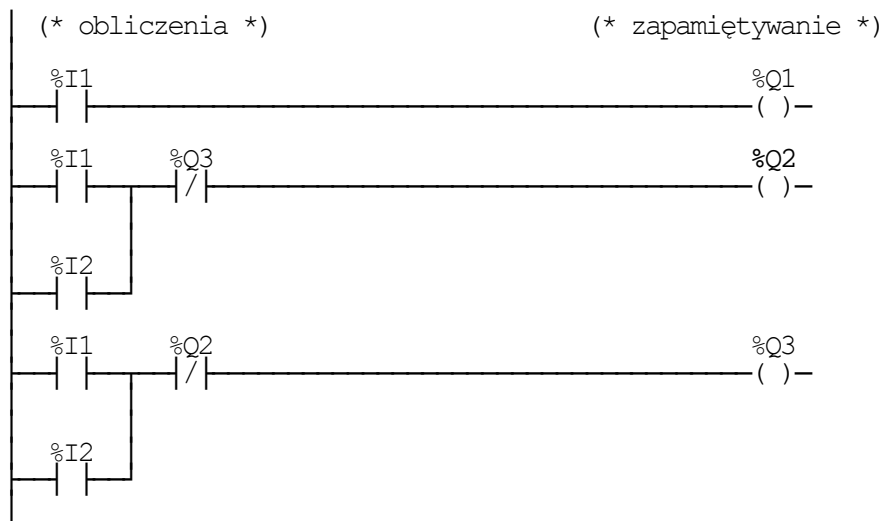
Styki	Symbol	Opis
Styki statyczne	$\begin{array}{c} *** \\ \text{---}   \text{---} \end{array}$	<p><i>Styk zwierny (normalnie otwarty, ang. normally open contact)</i></p> <p>Stan połączenia z lewej strony styku jest przenoszony na prawą stronę, jeżeli skojarzona zmienna boolowska ma wartość <i>1</i>. W przeciwnym razie prawe połączenie jest w stanie <i>OFF</i>.</p>
	$\begin{array}{c} *** \\ \text{---}   /   \text{---} \end{array}$	<p><i>Styk rozwierny (normalnie zamknięty, ang. normally closed contact)</i></p> <p>Stan połączenia z lewej strony styku jest przenoszony na prawą stronę, jeżeli skojarzona zmienna boolowska ma wartość <i>0</i>. W przeciwnym razie prawe połączenie jest w stanie <i>OFF</i>.</p>
Styki impulsowe (wrażliwe na zbocze)	$\begin{array}{c} *** \\ \text{---}   \text{P}   \text{---} \end{array}$	<p><i>Styk wrażliwy na zbocze narastające (ang. Positive transition-sensing contact)</i></p> <p>Połączenie z prawej strony styku jest w stanie <i>ON</i> w czasie jednego wykonania, jeśli połączenie z lewej strony jest w stanie <i>ON</i>, a skojarzona zmienna boolowska zmieniła wartość z <i>0</i> na <i>1</i>. Poza tym stan połączenia z prawej strony jest <i>OFF</i>.</p>
	$\begin{array}{c} *** \\ \text{---}   \text{N}   \text{---} \end{array}$	<p><i>Styk wrażliwy na zbocze opadające (ang. Negative transition-sensing contact)</i></p> <p>Połączenie z prawej strony styku jest w stanie <i>ON</i> w czasie jednego wykonania, jeśli połączenie z lewej strony jest w stanie <i>ON</i>, a skojarzona zmienna boolowska zmieniła wartość z <i>1</i> na <i>0</i>. Poza tym stan połączenia z prawej strony jest <i>OFF</i>.</p>



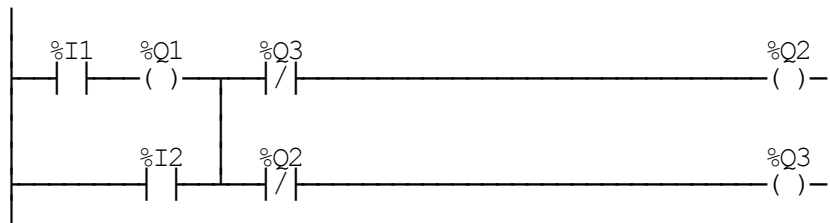
Symbole cewek:

Cewki	Symbol	Opis
Cewki zwykłe	*** — ( ) —	<i>Cewka (ang. coil)</i> Stan połączenia z lewej strony cewki jest przenoszony na prawą stronę i zapamiętywany w skojarzonej zmiennej boolowskiej.
	*** — (/) —	<i>Cewka negująca (ang. negated coil)</i> Stan połączenia z lewej strony cewki jest przenoszony na prawą stronę, a jego odwrotność jest zapamiętywana w skojarzonej zmiennej boolowskiej.
Cewki zatrzas kiwane	*** — (S) —	<i>Cewka ustawiająca (ang. Set coil, Latch coil)</i> Skojarzona zmienna przyjmuje wartość 1, jeżeli połączenie z lewej strony jest w stanie ON. Wartość ta pozostanie niezmienną, aż do chwili wyzerowania przez cewkę kasującą ( R ).
	*** — (R) —	<i>Cewka kasująca (ang. Reset coil, Unlatch coil)</i> Skojarzona zmienna przyjmuje wartość 0, jeżeli połączenie z lewej strony jest w stanie ON. Wartość ta pozostanie niezmienną, aż do chwili ustawienia przez cewkę ustawiającą ( S ).
Cewki podtrzymywane, cewki z pamięcią	*** — (M) —	<i>Cewka z zapamiętaniem stanu (ang. Retentive coil, Memory coil)</i>
	*** — (SM) —	<i>Cewka ustawiająca z zapamiętaniem stanu (ang. Set retentive coil)</i>
	*** — (RM) —	<i>Cewka kasująca z zapamiętaniem stanu (ang. Reset retentive coil)</i>
Cewki impulsowe (wrażliwe na zbocze)	*** — (P) —	<i>Cewka wrażliwa na zbocze narastające (ang. Positive transition-sensing coil)</i> Skojarzona zmienna przyjmuje wartość 1 na czas jednego wykonania, jeśli połączenie z lewej strony zmieniło stan z OFF na ON. Stan połączenia z lewej strony jest zawsze przenoszony na prawą.
	*** — (N) —	<i>Cewka wrażliwa na zbocze opadające (ang. Negative transition-sensing coil)</i> Skojarzona zmienna przyjmuje wartość 1 na czas jednego wykonania, jeśli połączenie z lewej strony zmieniło stan z ON na OFF. Stan połączenia z lewej strony jest zawsze przenoszony na prawą.

Przykład przejrzystej struktury obwodów w języku LD:



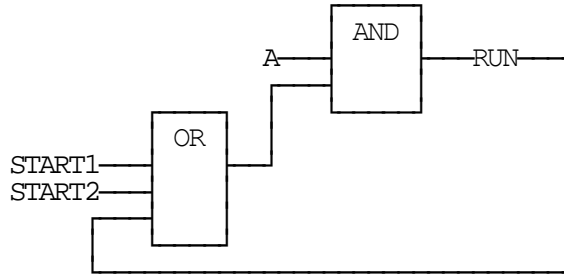
Przykład nieprzejrzystej struktury obwodów w języku LD:



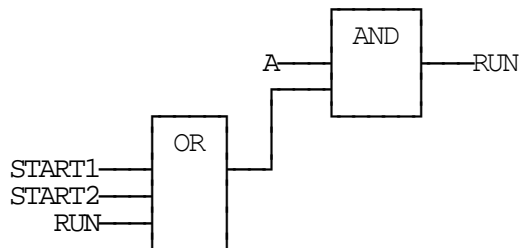
## Język FBD (Funkcjonalny schemat blokowy)

Przykłady sprzężenia zwrotnego - pętla jawna i pętla ukryta:

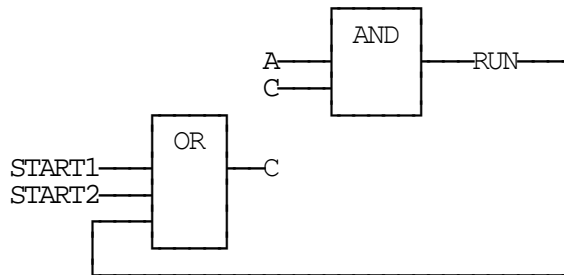
(\* przykład 1 - pętla jawna \*)



(\* przykład 2 - pętla ukryta \*)



(\* przykład 3 - pętla ukryta \*)



## Język IL (Lista rozkazów)

Przykład sekwencji rozkazów:

Etykieta	Operator	Operand	Komentarz
START :	LD	%IX1	(* WCIŚNIJ PRZYCISK *)
	ANDN	%MX5	(* NIE WSTRZYMANE *)
	ST	%QX2	(* ZAŁĄCZ *)

Zasada działania uniwersalnego akumulatora:

VAR

Operand1, Operand2, Wynik : INT :=0;

END\_VAR

Et1: LD Operand1 (\* CR ← Operand1, tu wartość 0 \*)

ADD 10 (\* CR ← CR + 10, tu CR = 10 \*)

ST Wynik (\* Wynik ← CR, CR bez zmian \*)

GT 0 (\* czy CR > 0? – tak, więc CR := TRUE \*)

JMPC Et2 (\* wykonaj skok do etykiety Et2, gdy CR = TRUE, CR bez zmian \*)

ADD Operand2 (\* do CR dodaj Operand2 – Błąd !!! – niezgodność typów \*)

Et2:

## Operatory języka IL

Lp.	Operator	Modyfikatory	Operand	Opis
1	<i>LD</i>	<i>N</i>	*	<i>CR</i> przyjmuje wartość operandu ( <i>Load</i> )
2	<i>ST</i>	<i>N</i>	*	Przesłanie <i>CR</i> do operandu ( <i>Store</i> )
3	<i>S</i> <i>R</i>		<i>BOOL</i> <i>BOOL</i>	Jeśli <i>CR</i> = 1, to operand ustaw na 1 ( <i>Set</i> ) Jeśli <i>CR</i> = 1, to zeruj operand ( <i>Reset</i> )
4	<i>AND</i>	<i>N</i> , (, <i>N</i> (	<i>BOOL</i>	Boolowskie <i>AND</i> operandu i <i>CR</i>
5	<i>OR</i>	<i>N</i> , (, <i>N</i> (	<i>BOOL</i>	Boolowskie <i>OR</i> operandu i <i>CR</i>
6	<i>XOR</i>	<i>N</i> ,(, <i>N</i> (	<i>BOOL</i>	Boolowskie ( <i>eXclusive OR</i> ) operandu i <i>CR</i>
7	<i>ADD</i>	(	*	Dodawanie ( <i>ADDition</i> ) operandu i <i>CR</i>
8	<i>SUB</i>	(	*	Odejmowanie ( <i>SUBtraction</i> ) operandu i <i>CR</i>
9	<i>MUL</i>	(	*	Mnożenie ( <i>MULTiplication</i> ) operandu i <i>CR</i>
10	<i>DIV</i>	(	*	Dzielenie ( <i>DIVision</i> ) <i>CR</i> przez operand
11	<i>GT</i>	(	*	Porównanie: <i>CR</i> > operand ( <i>Greater Than</i> )
12	<i>GE</i>	(	*	Porównanie: <i>CR</i> >= operand ( <i>Greater than or Equal</i> )
13	<i>EQ</i>	(	*	Porównanie: <i>CR</i> = operand ( <i>Equal</i> )
14	<i>NE</i>	(	*	Porównanie: <i>CR</i> <> operand ( <i>Not Equal</i> )
15	<i>LE</i>	(	*	Porównanie: <i>CR</i> <= operand ( <i>Less than or Equal</i> )
16	<i>LT</i>	(	*	Porównanie: <i>CR</i> < operand ( <i>Less Than</i> )
17	<i>JMP</i>	<i>C</i> , <i>CN</i>	<i>LABEL</i>	Skok do etykiety ( <i>JuMP to label</i> )
18	<i>CAL</i>	<i>C</i> , <i>CN</i>	<i>NAME</i>	Wywołanie ( <i>CALl</i> ) FB o nazwie <i>NAME</i>
19	<i>RET</i>	<i>C</i> , <i>CN</i>		Powrót ( <i>RETurn</i> ) z wywołanej funkcji lub FB
20	)			Operator ograniczający dla modyfikatora (

Przykład zagnieżdżenia wyrażeń w nawiasach

```
LD X1      (* CR ← X1 *)
MUL( X2    (* CR ← X2 *)
  SUB( X3   (* CR ← X3 *)
    ADD X4  (* CR ← X3 + X4 *)
  )        (* CR ← X2 - (X3 + X4) *)
)        (* CR ← X1 * (X2 - (X3 + X4)) *)
ST Y      (* CR bez zmian *)
```

Przykład wywołania funkcji w języku IL:

```
FUNCTION MojaFun : INT; (* Deklaracja funkcji *)
  VAR_INPUT
    We1, We2, We3 : INT; (* Parametry wejściowe *)
  END_VAR
(* ciało funkcji *)
  LD We1
  ADD We2
  ADD We3
  ST MojaFun (* Wartość funkcji na wyjściu *)
END_FUNCTION

(* fragment POU wywołującego MojaFun *)
VAR
  Par1, Par2, Par3, Wynik : INT; (* Deklaracja zmiennych *)
END_VAR

LD Par1 (* Wprowadzenie pierwszego parametru wejściowego *)
MojaFun Par2, Par3 (* Wywołanie funkcji z pozostałymi parametrami wejściowymi *)
ST Wynik (* Zapamiętanie wartości MojaFun w zmiennej Wynik *)
```

Wywoływanie bloków funkcjonalnych w języku IL:

Lp.	Opis i przykład
1	<i>CAL</i> z listą wejść: <i>CAL C10(CU:=%IX10, PV:=15)</i>
2	Przesyłanie wejść za pomocą operatorów <i>LD</i> i <i>ST</i> : <i>LD 15</i> <i>ST C10.PV</i> <i>LD %IX10</i> <i>ST C10.CU</i> <i>CAL C10</i>
3	Użycie operatorów wejściowych <i>LD 15</i> <i>PV C10</i> <i>LD %IX10</i> <i>CU C10</i> <i>CAL C10</i>

W przykładach przyjęto, że *C10* jest zadeklarowanym egzemplarzem standardowego licznika *CTU*, tzn. w jednostce wywołującej użyta została deklaracja

*VAR C10: CTU; END\_VAR.*

Operatory wejściowe w języku IL dla standardowych bloków funkcjonalnych:

Lp.	Operatory	Bloki funkcjonalne
1	<i>SI, R</i>	<i>SR</i>
2	<i>S, RI</i>	<i>RS</i>
3	<i>CLK</i>	<i>R_TRIG</i>
4	<i>CLK</i>	<i>F_TRIG</i>
5	<i>CU, R, PV</i>	<i>CTU</i>
6	<i>CD, LD, PV</i>	<i>CTD</i>
7	<i>CU, CD, R, LD, PV</i>	<i>CTUD</i>
8	<i>IN, PT</i>	<i>TP</i>
9	<i>IN, PT</i>	<i>TON</i>
10	<i>IN, PT</i>	<i>TOF</i>

Pozostałe elementy języka IL:

- *TYPE ... END\_TYPE;*
- *VAR ... END\_VAR;*
- *VAR\_INPUT ... END\_VAR;*
- *VAR\_OUTPUT ... END\_VAR;*
- *VAR\_IN\_OUT ... END\_VAR;*
- *VAR\_EXTERNAL ... END\_VAR;*
- *FUNCTION ... END\_FUNCTION;*
- *FUNCTION\_BLOCK ... END\_FUNCTION\_BLOCK;*
- *PROGRAM ... END\_PROGRAM;*
- *STEP ... END\_STEP;*
- *TRANSITION ... END\_TRANSITION;*
- *ACTION ... END\_ACTION*



## Język ST (Tekst strukturalny)

Operatory języka ST:

Lp.	Symbol	Opis	Przykład
1	(wyrażenie)	Wyrażenie w nawiasach	$(X+Y)*(X-Y)$
2	<i>Funkcja</i> (lista parametrów)	Obliczanie wartości funkcji	$LN(A), MAX(X, Y),$
3	**	Potęgowanie	$X**Y$
4	-	Negacja arytmetyczna (Wartość przeciwna)	$-10, -X$
5	NOT	Negacja boolowska (dopełnienie)	$NOT (X > Y)$
6	*	Mnożenie	$X * Y$
7	/	Dzielenie	$X / Y$
8	MOD	Reszta z dzielenia ( <i>MOD</i> ulo)	$13 MOD 10$ (wynik: 3)
9	+	Dodawanie	$X + Y$
10	-	Odejmowanie	$X - Y$
11	<, >, <=, >=	Porównywanie	$T\#1h > T\#30m$ (wynik: <i>TRUE</i> )
12	=	Równość	$T\#1d = T\#24h$ (wynik: <i>TRUE</i> )
13	<>	Nierówność	
14	AND lub &	Iloczyn boolowski	$(X > Y) AND (X < Z)$
15	XOR	Suma boolowska modulo 2 ( <i>eXclusive OR</i> )	$TRUE XOR FALSE$
16	OR	Suma boolowska	$TRUE OR FALSE$

Instrukcje języka ST:

Lp.	Instrukcja	Przykłady
1	Przypisanie	$A:=B$ ; $CV:=CV+1$ ; $Y:=\text{SIN}(X)$ ; $D:=\text{INT\_TO\_REAL}(C)$
2	Wywołanie FB Użycie wyjścia FB	$\text{Moj\_TMR}(\text{IN}:=\%IX5, \text{PT}:=T\#300\text{ms})$ ; $A:=\text{Moj\_TMR.Q}$ ;
3	<i>RETURN</i>	<i>RETURN</i> ;
4	<i>IF</i>	$D:=B*B-4*A*C$ ; (* oblicz wyróżnik *) $\text{IF } D < 0.0 \text{ THEN } \text{NROOTS}:=0$ ; (* brak pierwiastków *) $\text{ELSIF } D = 0.0 \text{ THEN}$ (* jeden pierwiastek *) $\text{NROOTS}:=1$ ; $X1 := -B / (2.0 * A)$ ; $\text{ELSE}$ (* dwa pierwiastki *) $\text{NROOTS}:=2$ ; $X1 := (-B + \text{SQRT}(D)) / (2.0 * A)$ ; $X2 := (-B - \text{SQRT}(D)) / (2.0 * A)$ ; $\text{END\_IF}$ ;
5	<i>CASE</i>	$\text{ERROR}:=0$ ; (* zmienna boolowska *) $\text{XW}:=\text{BCD\_TO\_INT}(Y)$ ; (* wyznacz wartość wybieraka *) $\text{CASE } \text{XW} \text{ OF}$ (* instrukcja wyboru *) 1,4: $\text{DISPLAY}:=\text{TEKST1}$ ; 2: $\text{DISPLAY}:=\text{TEKST2}$ ; (* blok instrukcji ) $Y:=\text{SIN}(Z)$ ; (* kolejne instrukcje, gdy $\text{XW} = 2$ *) 3,5..10: $\text{DISPLAY}:=\text{STATUS}(\text{XW}-3)$ ; $\text{ELSE } \text{DISPLAY}:= ''$ ; (* $\text{XW}$ poza zakresem 1..10 *) $\text{ERROR}:=1$ ; $\text{END\_CASE}$ ; (* koniec instrukcji wyboru *)
6	<i>FOR</i>	$J:=101$ ; $\text{FOR } I:=1 \text{ TO } 100 \text{ BY } 2 \text{ DO}$ $\text{IF } \text{WORDS}(I) = \text{'KEY'}$ THEN $J:=I$ ; $\text{EXIT}$ ; $\text{END\_IF}$ ; $\text{END\_FOR}$ ;
7	<i>WHILE</i>	$J:=1$ ; $\text{WHILE } J \leq 100 \text{ AND } \text{WORDS}(J) \neq \text{'KEY'}$ DO $J:=J+2$ ; $\text{END\_WHILE}$ ;
8	<i>REPEAT</i>	$J:=-1$ ; $\text{REPEAT}$ $J:=J+2$ ; $\text{UNTIL } J = 101 \text{ OR } \text{WORDS}(J) = \text{'KEY'}$ $\text{END\_REPEAT}$ ;
9	<i>EXIT</i>	$\text{EXIT}$ ;

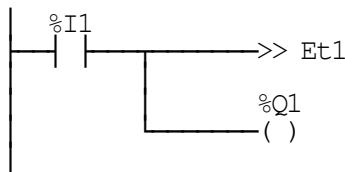
## Kompatybilność języków

Każdy z języków programowania posiada pewne cechy, które powodują, że do zaprogramowania niektórych zagadnień nadaje się lepiej, niż inne języki. Język LD jest najodpowiedniejszy do programowania operacji logicznych (algebra Boole'a). Języki tekstowe są wygodniejsze w procedurach zarządzania pamięcią lub przy programowaniu obliczeń iteracyjnych. Programy w językach graficznych są stosunkowo łatwe do analizy, czego nie można powiedzieć o języku IL, który za to jest najbardziej elastyczny, itd.

Niektóre pakiety programowania umożliwiają wykonanie takiego przetłumaczenia, aczkolwiek norma nie stawia takich wymagań. Podstawowy problem stanowi tu niepełna kompatybilność poszczególnych języków programowania. Szczególna trudność pojawia się przy przechodzeniu z języków graficznych na tekstowe i odwrotnie. Wynika to z różnego sposobu przetwarzania w obu grupach języków. Te pierwsze są przede wszystkim językami proceduralnymi, tzn. instrukcje są wykonywane w nich jedne po drugich, podczas gdy podstawą w językach graficznych jest przepływ sygnału („prądu” w języku LD), który może być realizowany równolegle.

Przykład dla porównania własności języka LD i IL:

a) język LD



b) język IL

```
LD %I1
JMPC Et1
ST %Q1
```

Stąd, aby ustrzec się tego typu niejednoznaczności, w niektórych systemach programowania wprowadza się pewne ograniczenia w językach graficznych, takie jak np.:

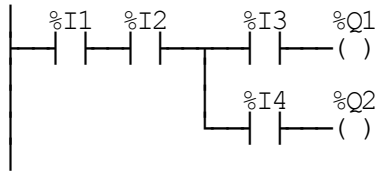
- nie dopuszcza się do wprowadzania operacji logicznych w obwodzie po operacjach przypisania (np. w języku LD po wprowadzeniu cewki nie można już dalej rozwijać obwodu);
- albo w ogóle nie dopuszcza się do stosowania instrukcji sterujących (typu skok), albo jeśli się dopuszcza, to w trakcie wykonywania obwodu z góry na dół napotkanie takiej instrukcji powoduje zakończenie wykonywania danego obwodu.

Właściwie to stosowanie instrukcji skoku w językach graficznych stoi w pewnej sprzeczności z zasadą równoległego przetwarzania sygnału w obwodzie. Ponadto w językach tych w ogóle nie występują instrukcje dla obliczeń iteracyjnych (typu *REPEAT*, *WHILE*, *FOR*), stąd tłumaczenie tego typu konstrukcji z języka ST na języki graficzne wymaga tworzenia skomplikowanych obwodów, w których w sposób jawny operuje się wskaźnikiem pętli i sprawdza warunki jej zakończenia za pomocą funkcji porównania (np. *EQ*).

Wykorzystanie własności równoległego przetwarzania w językach graficznych poprzez łączenie jednego wyjścia z większą liczbą wejść wymaga przy tłumaczeniu na języki tekstowe wprowadzania dodatkowych zmiennych w celu uzyskania kompatybilności programów.

Przykład pokazujący konieczność użycia zmiennej pomocniczej w języku IL:

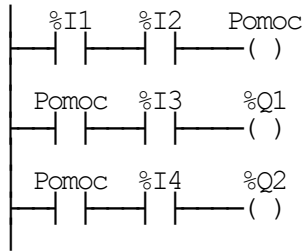
a) język LD



b) język IL

```
LD %I1
AND %I2
ST Pomoc
AND %I3
ST %Q1
LD Pomoc
AND %I4
ST %Q2
```

c) wersja równoważna w języku LD



Kolejny problem, to występowanie przy wywołaniu funkcji w językach graficznych pary *EN/ENO*, dla której nie ma odpowiednika w języku tekstowym. W tym przypadku trzeba by uzupełnić odpowiednie wywołanie funkcji przez warunek typu *IF ... THEN*.

Stosunkowo najłatwiej jest wykonać tłumaczenie w obrębie języków tekstowych, aczkolwiek także tutaj można natrafić na pewne trudności wynikające ze specyfiki języka. W języku ST nie występują instrukcje skoku, które w IL umożliwiają realizację pętli, stąd przejście z tekstu napisanego w IL na ST może nastroczać pewne problemy. Z kolei w IL nie ma instrukcji iteracyjnych, więc trzeba je organizować niejako „na piechotę”, poprzez wprowadzanie wskaźnika dla iteracji i wykorzystanie instrukcji porównania lub skoków.