Sterowniki Programowalne (SP) – Wykład #7

Język programowania PLC (*Instruction List - IL*)

WYDZIAŁ ELEKTROTECHNIKI I AUTOMATYKI – KATEDRA INŻYNIERII SYSTEMÓW STEROWANIA

Jarosław Tarnawski

październik 2016

Zawartość wykładu

- Norma IEC 61131 i rodzaje języków programowania
- Języki niskopoziomowe i wysokopoziomowe
- Czy w każdym PLC/PAC jest dostępny każdy język programowania?
- Akumulator i stos; sposób działania stosu
- Lista rozkazów IL
- Analogie LD i IL w styki/przekaźniki
- Operacje binarne z pośrednictwem stosu
- Realizacje niskopoziomowe funkcji wysokopoziomowych
- Zasoby IL dla PLC GE Fanuc 90-30
  - Sposoby wywołania przekaźników czasowych i liczników
  - Wywołanie bloku IL z programu głównego w LD
  - Deklarowanie zmiennych
  - Definiowanie pamięci na potrzeby Akumulatora i Stosu
- Regulator dwustanowy w IL; program sygnalizacji świetlnej w IL
- Wady i zalety języka IL
- Bibliografia

Norma IEC61131 - Programmable Logic Controllers składa się z 5 części:

W trzeciej części IEC61131-3 zdefiniowano języki programowania PLC. Dzięki definicji i ujednoliconemu sposobowi programowania użytkownicy mogą oczekiwać bardzo podobnego sposobu programowania PLC różnych producentów.
Określono model programowy, model komunikacyjny, typy i struktury danych.

Zawartość normy:

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)

Norma IEC61131 - Programmable Logic Controllers definiuje następujące języki programowania:

Języki graficzne:
1. Język drabinkowy (*Ladder Diagram* – LD)
2. Język bloków funkcyjnych (*Function Block Diagram* - FBD)

Języki tekstowe:
3. Lista instrukcji (*Instruction List* – IL)
4. Tekst strukturalny (*Structured Text* - ST)

5. Język grafów strukturalnych (*Sequential Function Chart* - SFC)

Niektórzy producenci dostarczają moduł programowania w języku C, ale nie jest on zdefiniowany w normie!

Instruction List – IL	Structured Text – ST

Język niskopoziomowy
Podobny do: Assembler
 	
Cechy:
Mnemoniki, Akumulator, Stos
 	
Typowe Rozkazy:
`LD wartosc`
`GT 10`
`JMPC skok1`
`LD adres1`
`ST wynik`
`skok1:`
`LD adres2`
`ST wynik`
`JMP etykieta`
`etykieta:`
	Język wysokopoziomowy
	Podobny do: Pascal, C, Basic
	
	Cechy:
	Operator podstawienia :=
	Koniec wiersza ;
	Kontrola typów danych 0.0 <> 0
	
	Typowe Rozkazy:
	
	`If..then..else..end_if;`
	`Case..of..end_case`
	`For..to..do..end_for;`
	`While..do..end_while;`
	`Repeat..until..end_repeat`

Języki programowania dostępne w PLC/PAC GE Fanuc

90-30	PACs	SoftPLC (Proficy ME)
LD IL C	LD FBD ST C	LD IL ST FDB SFC

LD – Ladder Diagram – język drabinkowy
FDB – Function Block Diagram – schematy bloków funkcyjnych
IL – Instruction List – lista instrukcji
ST – Structured Text – język strukturalny
SFC – Sequential Function Chart – graf sekwencji
C – język C

Język niskopoziomowy – przedostatni poziom przed językiem maszynowym

Rozkazy odpowiadają jednemu kodowi maszynowemu

Język symboliczny – ułatwiający programowanie, zapamiętanie np. polecenia LOAD jest łatwiejsze niż odpowiadającemu mu kodowi operacji maszynowej wraz z trybami adresowania i argumentami
Mnemoniki to nazwa krótkich rozkazów wykonywanych przez procesory
Procesory RISC (*Reduced Instruction Set Computers*)
CISC (*Complex Instruction Set Computers*)
w zależności od listy rozkazów

Jak czytamy w normie: Lista instrukcji składa się z instrukcji!

Każda instrukcja powinna zaczynać się w nowej linii i powinna zawierać operator z ewentualnym modyfikatorem i jeśli to konieczne dla konkretnej operacji jeden lub więcej operandów oddzielanych przecinkiem.

Instrukcja może być poprzedzona etykietą zakończoną znakiem :

Pomiędzy instrukcje mogą być wprowadzane puste linie.
Komentarze zawierają się wewnątrz znaków (* *)

etykieta	operator	operand	Komentarz
start:	LD	%I1	(*Naciśnięty przycisk*)
	ANDN	%M1	(*Brak blokady*)
	ST	%Q1	‘Załącz urządzenie

Akumulator (A) to rejestr w pamięci, który jest podstawą działania języka IL. Jest wykorzystywany do:

- wczytywania do niego wartości z komórek pamięci PLC,
- wykonywania operacji matematycznych/logicznych,
- przechowywania tymczasowych wyników,
- kopiowania stanu akumulatora do wybranych komórek pamięci.

Podstawowe operacje:
LD pamiec1 (*od LOAD*) - załaduj stan komórki pamiec1 do A
ST pamiec2 (*od STORE*) – zapisz stan A pod adres pamiec2

Może obsługiwać różne typy danych (BOOL, INT, REAL). W programowaniu obiektowym nazywa się to przeciążaniem. Dane dla jednej operacji muszą być zawsze tego samego typu.

Zapisanie komórki pamięci odbywa się z pośrednictwem akumulatora

Zawartość komórki pamięci %M1 zapisana jest do %Q2

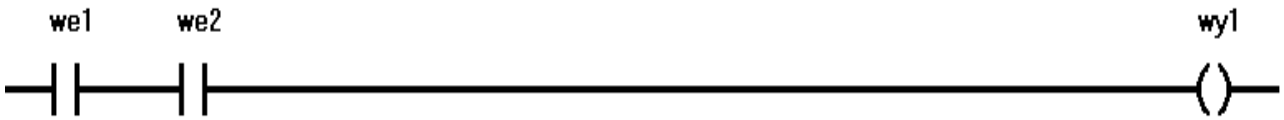
```
LD %M1
ST %Q2
```

Zawartość rejestru %R3 zapisana jest na wyjście analogowe %AQ1

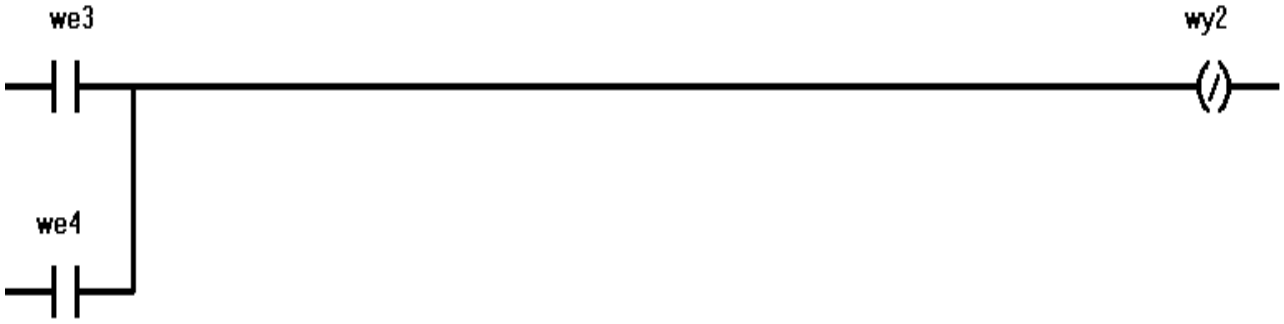
```
LD %R3
ST %AQ1
```

Operacje na bitach, analogie LD i IL

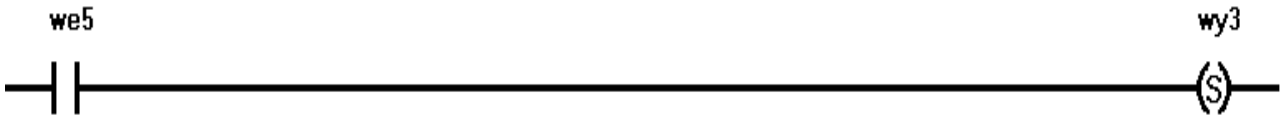
```
LD_BOOL we1
AND we2
ST_BOOL wy1
```



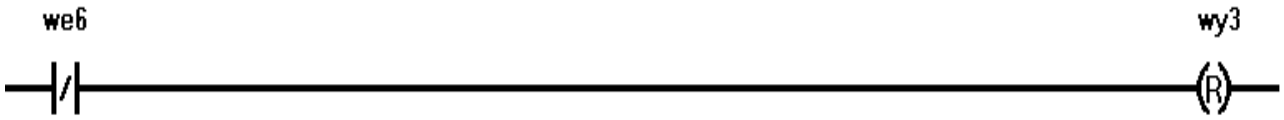

```
LD_BOOL we3
OR we4
STN_BOOL wy2
```




```
LD_BOOL we5
S wy3
```

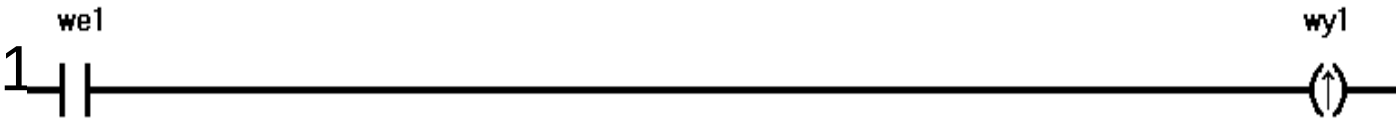



```
LDN_BOOL we6
R wy3
```



Operacje na bitach, analogie LD i IL

LD_BOOL we1
PT wy1

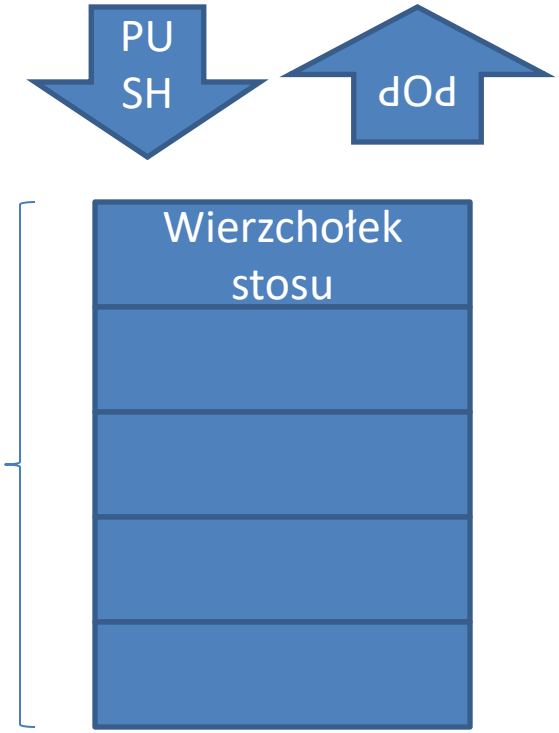


LD_BOOL we2
NT wy2

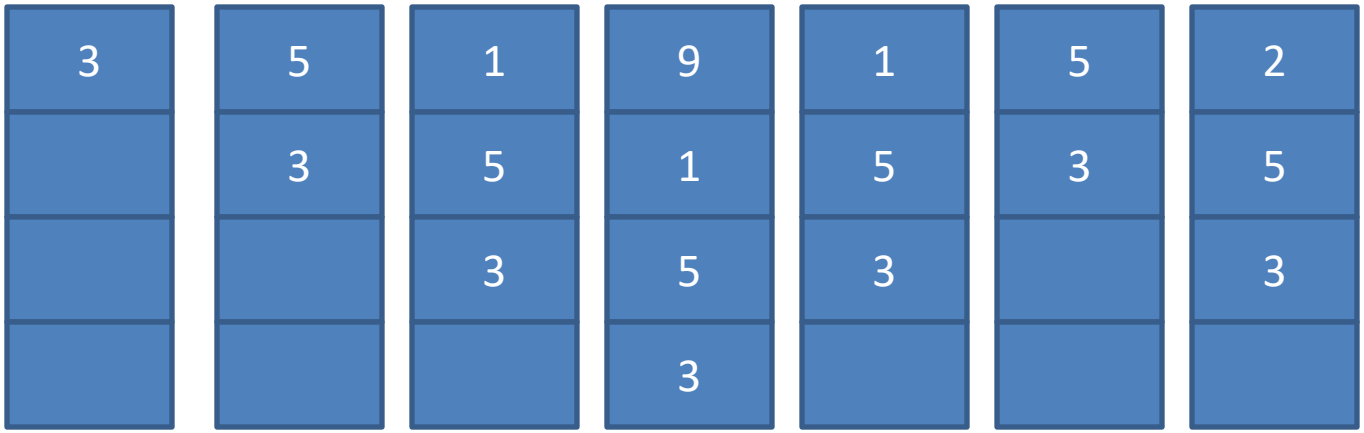
Stos

Liniowa struktura pamięci działająca wg zasady LIFO (ang. *Last In First Out*). Dostęp do stosu występuje wyłącznie przez wierzchołek stosu. Posiada dwie funkcje do obsługi:

Odłóż na stos	push(argument)
Pobierz ze stosu	wynik=pop()



krok1	krok2	krok3	krok4	krok5	krok6	krok7
Push(3)	Push(5)	Push(1)	Push(9)	Pop()	Pop()	Push(2)



Stan stosu po każdej operacji
(przy założeniu pustego stosu przed krokiem 1)

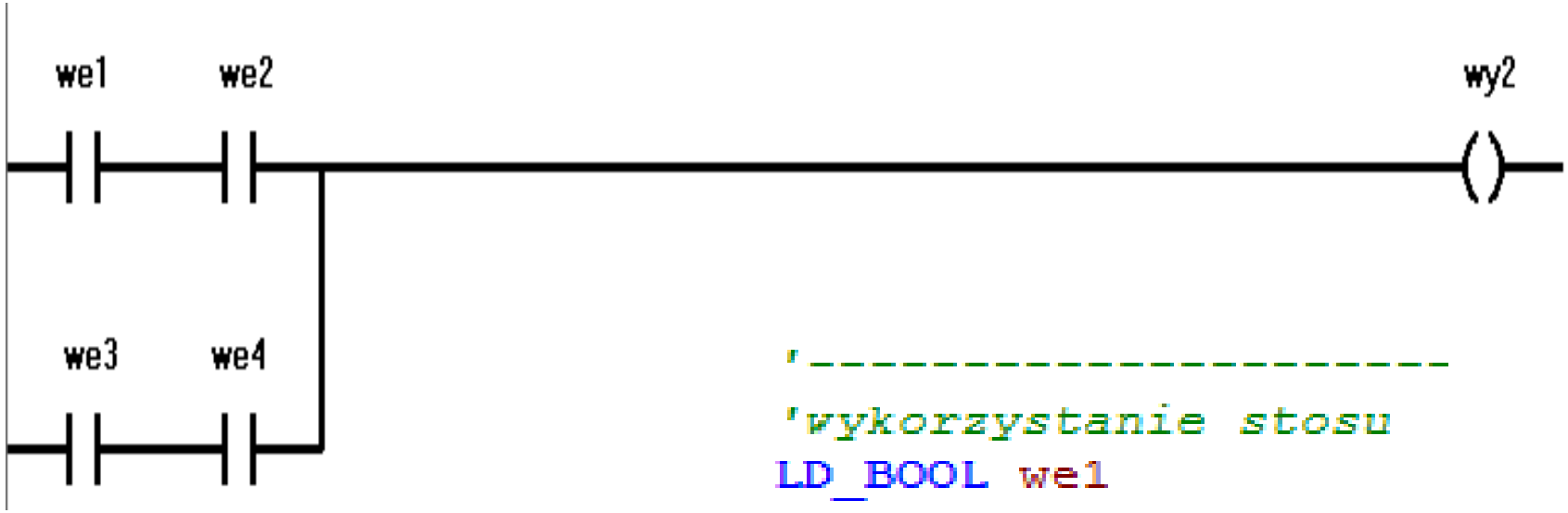
Operandy „(„ oraz „)” w języku IL

W logice IL występują tzw. operacje zagnieżdżone (ang. nested) do których obsługi wykorzystuje się stos. Operacje wywoływane w programie są z wykorzystaniem znaków nawiasów.

Nawias otwierający „(„ powoduje zapisanie zawartości Akumulatora na stosie natomiast nawias zamykający „)” powoduje wczytanie danej z wierzchołka stosu do Akumulatora.

Tak jak w przypadku akumulatora stos może być przeładowywany tj. przyjmować różne typy danych (ale dane muszą być tego samego typu dla jednej operacji)

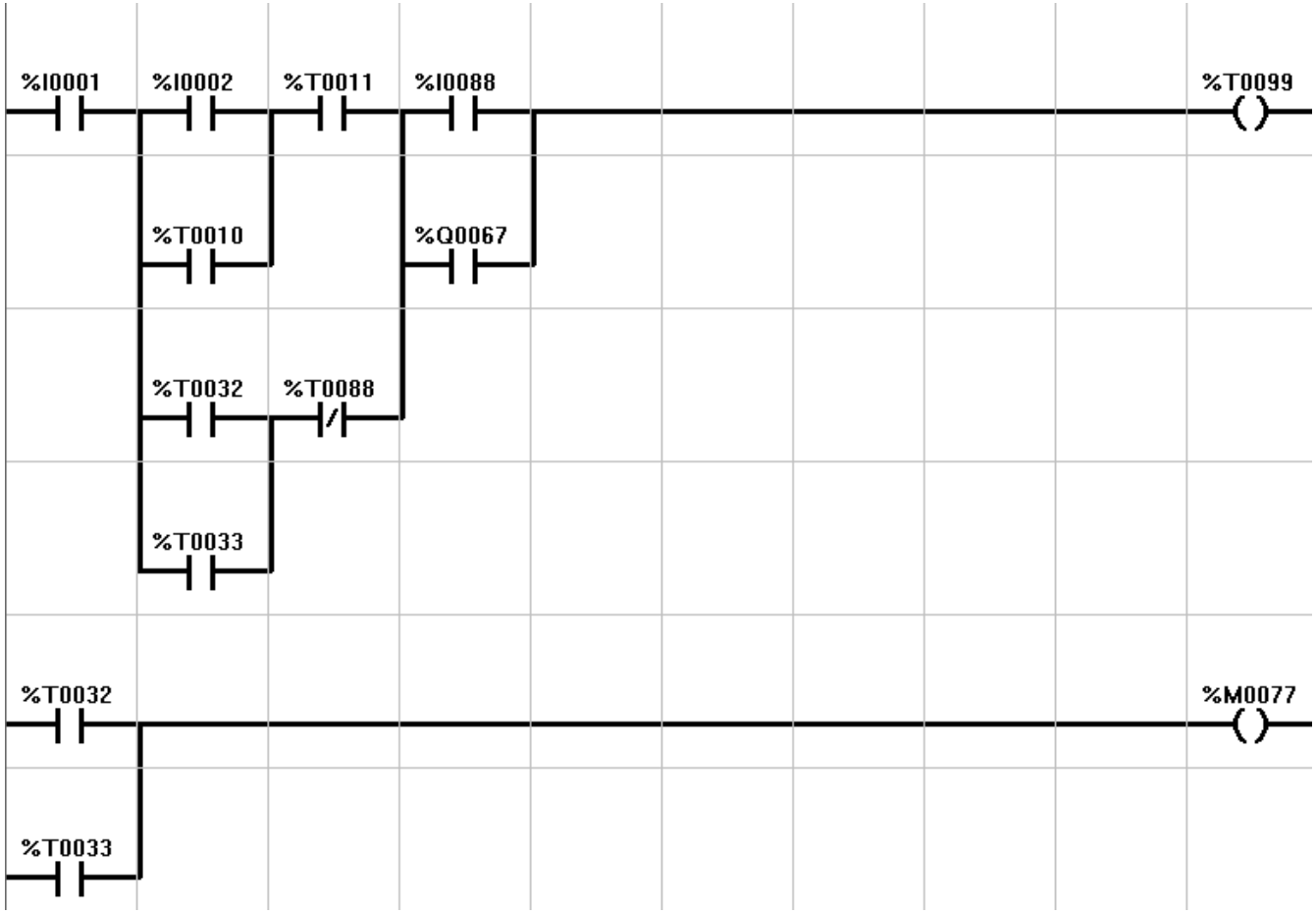
Operandy „(„ oraz „)” w języku IL




```
'----------------------
'wykorzystanie stosu
LD_BOOL we1
AND we2
OR(
   LD_BOOL we3
   AND we4
   )
ST_BOOL wy2
'----------------------
```

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTROTECHNIKI I AUTOMATYKI

Operandy „(„ oraz „)” w języku IL




```
LD_BOOL V_I00001
AND(
     LD_BOOL V_I00002
     OR V_T00010
     AND V_T00011
     OR(
        LD_BOOL V_T00032
        OR V_T00033
        ST_BOOL V_M00077
        ANDN V_T00088
       )
     AND V_T00088
     OR V_Q00067
     )
ST_BOOL V_T00099
```

#	Program	Akumulator	Stos
1	LD_BOOL V_I1	V_I1	
2	AND (	V_I1	V_I1
3	LD_BOOL V_I2	V_I2	V_I1
4	OR V_T10	V_I2 OR V_T10	V_I1
5	AND V_11	V_I2 OR V_T10 AND V_T11	V_I1
6	OR (		V_I2 OR V_T10 AND V_T11 V_I1
7	LD_BOOL V_T32	V_T32	V_I2 OR V_T10 AND V_T11 V_I1
8	OR V_T33	V_T32 OR V_T33	V_I2 OR V_T10 AND V_T11 V_I1
9	ST_BOOL V_M77	V_T32 OR V_T33	V_I2 OR V_T10 AND V_T11 V_I1
10	ANDN V_T88	V_T32 OR V_T33 ANDN V_T88	V_I2 OR V_T10 AND V_T11 V_I1
11	)	(V_I2 OR V_T10 AND V_T11) OR (V_T32 OR V_T33 ANDN V_T88)	V_I1
12	AND V_T88	(V_I2 OR V_T10 AND V_T11) OR (V_T32 OR V_T33 ANDN V_T88) AND V_T88	V_I1
13	OR V_Q67	(V_I2 OR V_T10 AND V_T11) OR (V_T32 OR V_T33 ANDN V_T88) AND V_T88 OR V_Q67	V_I1
14	)	V_I1 AND (V_I2 OR V_T10 AND V_T11) OR (V_T32 OR V_T33 ANDN V_T88) AND V_T88 OR V_Q67	
15	ST_BOOL V_T99		

Jak zrealizować w IL funkcje IF … THEN … ?

Skok jeśli akumulator jest równy 1 (TRUE)

JMPC etykieta

Skok jeśli akumulator jest równy 0 (FALSE)

JMPCN etykieta

Jeśli warunek nie jest spełniony (dla skoków warunkowych JMPC i JMPCN) program jest kontynuowany od następnej po skoku instrukcji

Skok bezwarunkowy:

JMP etykieta

Etykiety

```
LD_BOOL we8
JMPC tutaj
```

odpowiednik
IF (we8 == 1) THEN GOTO tutaj

```
LD_INT rejestr4
GE rejestr5
ST_BOOL wy7

tutaj:
LD_INT rejestr6
MUL rejestr6
ST_INT rejestr7
```

Etykiety wyróżnia w programie nazwa etykiety oraz znak dwukropka
W tym przypadku etykietą jest słowo tutaj:

Etykieta jest wykorzystana jako argument funkcji JMPC (skoku warunkowego w przypadku stanu wysokiego w akumulatorze)

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTROTECHNIKI I AUTOMATYKI

Instrukcje IL w normie IEC 61131

LP	OPERATOR	MODYFIKATOR	SKŁADNIA
1	**LD**	**N**	Załaduj operand do Akumulatora (Aku)
2	**ST**	**N**	Zapisz wartość Aku pod adres operandu
3	**S** **R**		Ustaw (wpisz 1) pod adres operandu i podtrzymaj jeśli Aku jest równy 1 Skasuj (wpisz 0) pod adres operandu i podtrzymaj jeśli Aku jest równy 1
4	**AND**	**N, (**	Logiczne AND
5	**&**	**N, (**	Logiczne AND
6	**OR**	**N, (**	Logiczne OR
7 7a	**XOR** **NOT**	**N, (**	Logiczne exlusive OR Negacja
8	**ADD**	**(**	Dodawanie
9	**SUB**	**(**	Odejmowanie
10	**MUL**	**(**	Mnożenie
11 11a	**DIV** **MOD**	**(** **(**	Dzielenie Dzielenie – modulo
12	**GT**	**(**	Porównanie >
13	**GE**	**(**	Porównanie >=
14	**EQ**	**(**	Porównanie =
15	**NE**	**(**	Porównanie <>, !=
16	**LE**	**(**	Porównanie <=
17	**LT**	**(**	Porównanie <
18	**JMP**	**C, N**	Skok pod etykietę
19	**CAL**	**C, N**	Wywołanie bloku funkcyjnego
20	**RET**	**C, N**	Powrót z funkcji, bloku funkcyjnego lub programu
21	**)**		Wykonaj wcześniej rozpoczętą operację

Instrukcje IL w GE Fanuc

Operacje na bitach i słowach bitowych
AND, OR, XOR, NOT
Operacje matematyczne
ADD, SUB, MUL, DIV, MOD
Relacje matematyczne
EQ, NE, GE, GT, LT, LE, RANGE
Przekaźniki czasowe i liczniki
ONDTR, TMR, UPCTR, DNCTR
Operacje na tablicach
Operacje sterujące programem
Bloki regulatorów (np. PID(adress, SP,PV,man,up,dn) a CV w A)

Do dyspozycji są właściwie niemal wszystkie operacje z LD.

Zasoby systemu operacyjnego (zmienne systemowe, bloki funkcyjne) są dostępne niezależnie od języka!

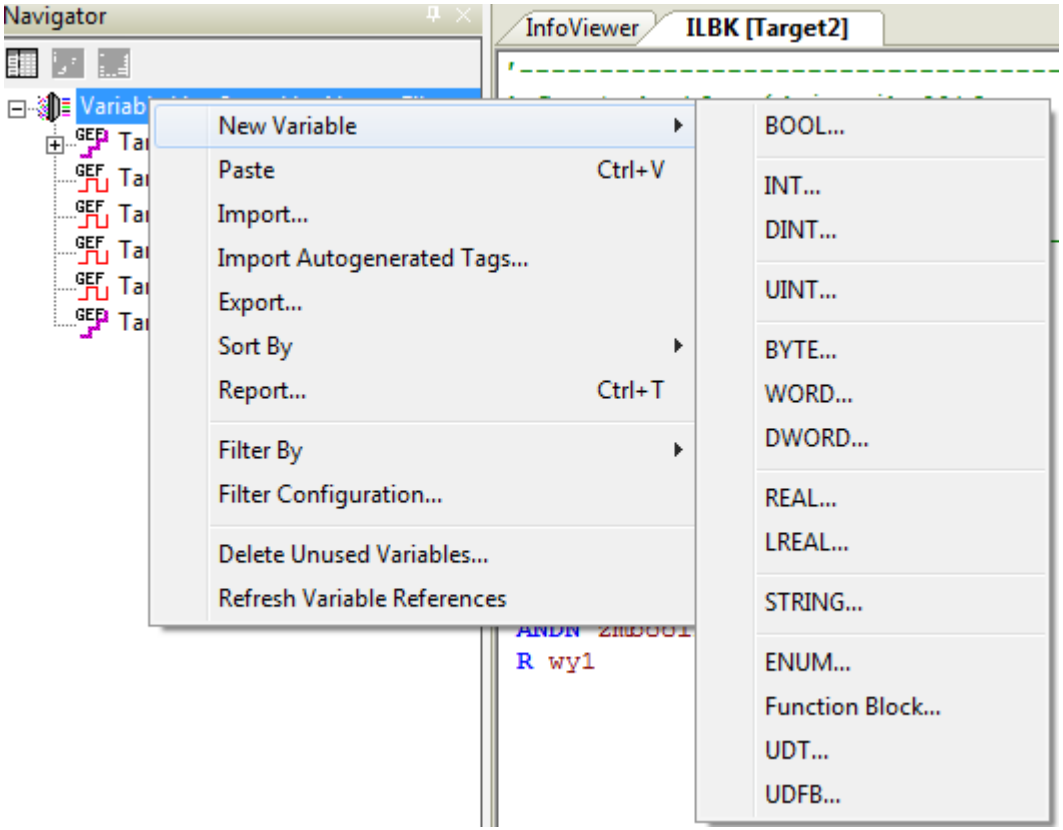
Norma wprowadza sekcję VAR … END_VAR; do definiowania zmiennych

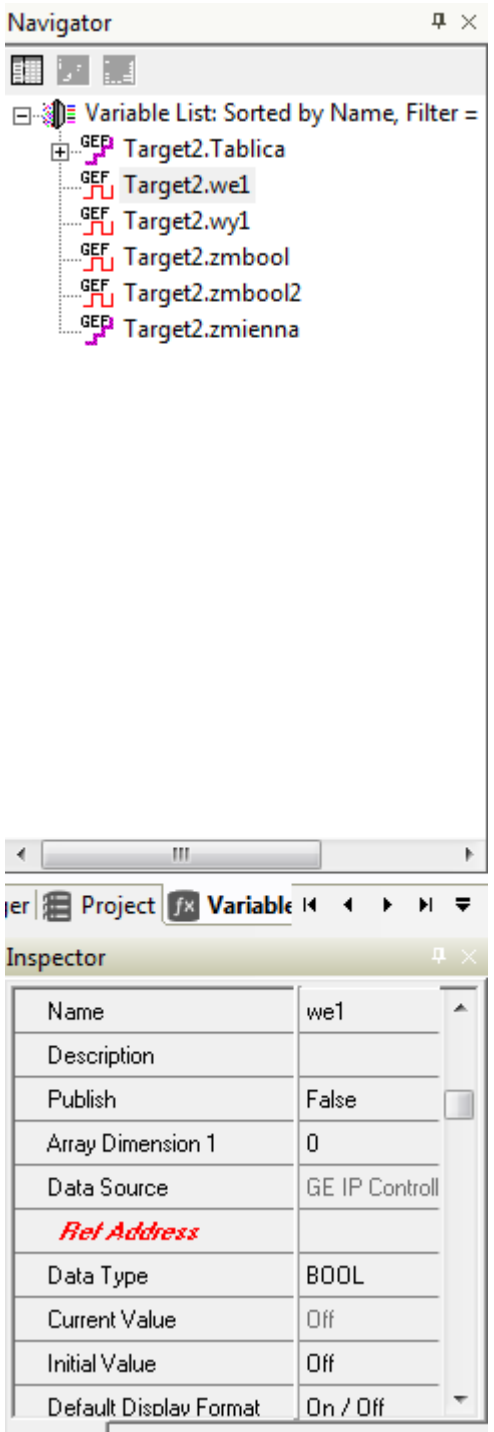
```
VAR
    A, B : INT;
END_VAR
```

Ale producenci PLC dostarczają swoje mechanizmy definiowania zmiennych i związywanie ich z fizycznymi adresami PLC

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTROTECHNIKI I AUTOMATYKI

Deklarowanie zmiennych





Instrukcje IL w GE Fanuc

IL Instructions

How do I... Related Topics

IL logic instructions are grouped according to the type of operation performed. The instruction groups are:

- Basic IL Instructions
- IL Advanced Math
- IL Basic Math Instructions
- IL Bit Instructions
- IL Communication Instructions
- IL Control Instructions
- IL Conversion Instructions
- IL Counters
- IL Data Move Instructions
- IL Data Table Instructions
- IL Relational Instructions
- IL Timers
- IL VersaMax Micro Motion

Basic Instructions

Accumulator | Coil | Boolean | Math | Relational | Program Flow

The basic IL instruction set supports operations on BOOL and 16-bit (INT, WORD, UINT) variables. These instructions generally operate on a new value for the same accumulator. The following instructions (sorted by functional group) are supported:

Accumulator Instructions (Basic IL)

Mnemonic	*Description*
LD_BOOL	Loads a BOOL value into the boolean accumulator. **Tip:** You can load #ALW_ON or #ALW_OFF.
LDN_BOOL	Loads the inverse of a BOOL value into the boolean accumulator.
LD_INT	Loads a single-precision integer (INT) value into the integer accumulator.
LD_ENO	Loads the enable output state from the previously executed instruction into the boolean accumulator.
ST_BOOL	Stores the content of the boolean accumulator to a BOOL variable.
STN_BOOL	Stores the inverse of the boolean accumulator to a BOOL variable.
ST_INT	Stores the content of the integer accumulator to a single-precision integer (INT) variable.
ST_DINT	Stores the content of the integer accumulator to a double-precision integer (DINT) variable.
ST_REAL	Stores the content of the integer accumulator to a single-precision floating point (REAL) variable.
ST_WORD	Stores the content of the integer accumulator to a WORD variable.

Boolean Instructions (Basic IL)

Mnemonic	*Description*
NOT	Inverts the state of the boolean accumulator.
AND	Performs a logical AND between the boolean accumulator and the BOOL operand *in*.
AND()	Performs a logical AND between the boolean accumulator and a boolean expression.
ANDN	Performs a logical AND between the boolean accumulator and the inverse of the BOOL operand *in*.
ANDN()	Performs a logical AND between the boolean accumulator and the inverse of a boolean expression.
OR	Performs a logical OR between the boolean accumulator and the BOOL operand *in*.
OR()	Performs a logical OR between the boolean accumulator and a boolean expression.
ORN	Performs a logical OR between the boolean accumulator and the inverse of the BOOL operand *in*.
ORN()	Performs a logical OR between the boolean accumulator and the inverse of a boolean expression.
XOR	Performs a logical XOR between the boolean accumulator and the BOOL operand *in*.
XOR()	Performs a logical XOR between the boolean accumulator and a boolean expression.
XORN	Performs a logical XOR between the boolean accumulator and the inverse of a BOOL operand *in*.
XORN()	Performs a logical XORN between the boolean accumulator and the inverse of a boolean expression.

Operacje matematyczne

Math Instructions (Basic IL)

Mnemonic	*Description*
ADD	Adds the content of the integer accumulator to the single-precision integer (INT) operand *in*.
SUB	Subtracts the INT operand *in* from the content of the integer accumulator.
MUL	Multiplies the content of the integer accumulator with the INT operand *in*.
DIV	Divides the content of the integer accumulator by the INT operand.
MOD	Divides the content of the integer accumulator by the INT operand *in* and stores the remainder in the accumulator.

Advanced Math

How do I... Related Topics

Use these IL instructions to perform trigonometric, exponential and logarithmic operations on data.

- Logarithmic
- Exponential
- Inverse Trig
- Square Root
- Trig

```
'operacje matematyczne
ADD_REAL(argument1, argument2)
ST_REAL wynik


MUL_REAL(wynik, argument3)
ST_REAL wynik02
```

Sprawdzanie poprawności wykonania operacji bloku funkcyjnego

Using ENO

When an instruction is executed, its ENO output indicates successful completion (ON, 1) or failure (OFF, 0). To check the state of the ENO output, use the LD_ENO operator, which loads the ENO state into the Boolean accumulator. You can branch to alternate flows of IL logic by testing the accumulator's value and jumping according to that value. The following example illustrates the use of ENO.

```
(*  Calculate the following expression:

        V_R00005 = (V_R00001 + 100) * V_R00077

    Set CALCOK variable to 1 if successful *)

        ADD_DINT(V_R00001, 100)
        ST_DINT V_R00003
        LD_ENO                  'Load the ENO output of ADD_DINT
        JMPCN error
        MUL_DINT(V_R00003, V_R00077)
        ST_DINT V_R00005
        LD_ENO                  'Load the ENO output of MUL_DINT
        JMPCN error
        LD_BOOL ALW_ON
        ST_BOOL CALCOK          'CALCOK = 1
        RET
error:
        LD_BOOL ALW_OFF
        ST_BOOL CALCOK          'CALCOK = 0
        RET
```

Timers

How do I... Related Topics

Use these IL built-in function blocks to time intervals from instruction list logic.

- Off Delay Timer
- On Delay Stopwatch Timer
- On Delay Timer

'przykładowe wykorzystanie timera odliczenie 1sekundy
LD_BOOL we1
TMR_TENTHS(rejestr, 10)
ST_BOOL wy1

On Delay Stopwatch Timer

ONDTR_HUNDS(*address, r, pv*)

The on-delay stop watch timer built-in function block increments every hundredth of a second while the boolean accumulator is True, and holds its value when the accumulator is False.

address: The first of three consecutive WORD registers. Word 1 - Current Value '*cv*', Word 2 - Preset Value '*pv*', Word 3 - Control Word.

r: BOOL variable; resets '*cv*' to zero when True.

pv: INT variable or constant (0 - 32,767); preset value in hundredths of a second.

ONDTR_TENTHS(*address, r, pv*)

The on-delay stop watch timer built-in function block increments every tenth of a second while the boolean accumulator is True, and holds its value when the accumulator is False.

address: The first of three consecutive WORD registers. Word 1 - Current Value '*cv*', Word 2 - Preset Value '*pv*', Word 3 - Control Word.

r: BOOL variable; resets '*cv*' to zero when True.

pv: INT variable or constant (0 - 32,767); preset value in tenths of a second.

ONDTR_THOUS(*address, r, pv*)

The on-delay stop watch timer built-in function block increments every thousandth of a second while the boolean accumulator is True, and holds its value when the accumulator is False.

address: The first of three consecutive WORD registers. Word 1 - Current Value '*cv*', Word 2 - Preset Value '*pv*', Word 3 - Control Word.

r: BOOL variable; resets '*cv*' to zero when True.

pv: INT variable or constant (0 - 32,767); preset value in thousandths of a second.

Counters

How do I... Related Topics

Use these IL built-in function blocks to count events occurring in a process under control.

- Up Counter
- Down Counter

'przykładowe wykorzystanie licznika
UPCTR
LD_BOOL we2
UPCTR(rejestr2, we3, 3)
ST_BOOL wy2

Up Counter

UPCTR(*address, r, pv*)

Each time the UPCTR built-in function block executes, it checks the current value of the Boolean accumulator. If that value has transitioned to ON, then UPCTR increments the current value 'CV'; otherwise, CV remains the same. When the 'PV' value is reached, the enable output turns ON and stays on until 'r' input becomes True to reset 'CV' to zero.

address: WORD variable; the beginning address of a three-word WORD array. Word 1 - Current Value 'CV', Word 2 - Preset Value 'PV', Word 3 - Control Word.

r: BOOL variable; resets the counter when True.

pv: INT variable or constant (0 - 32,767); preset value.

Down Counter

DNCTR(*address, r, pv*)

Each time the DNCTR built-in function block executes, it checks the current value of the Boolean accumulator. If that value has transitioned to ON, then DNCTR decrements the current value 'CV'; otherwise, CV remains the same. When the 'CV' value decrements to zero or less, the enable output turns ON and stays on until the 'r' input becomes True to reset 'CV' to the 'PV'.

address: WORD variable; the beginning address of a three-word WORD array. Word 1 - Current Value 'CV', Word 2 - Preset Value 'PV', Word 3 - Control Word.

r: BOOL variable; resets the counter when True.

pv: INT variable or constant (0 - 32,767); preset value.

Relational Instructions

How do I... Related Topics

Use these IL instructions to compare the values of two operands without altering them or generating a sum or difference.

Greater or Equal

GE_DINT(*in1, in2*)

If in1 ≥ in2, boolean accumulator ← 1

If in1 < in2, boolean accumulator ← 0

Compare two double precision integers. If *in1* is greater than or equal to *in2*, set the boolean accumulator. If *in1* is less than *in2*, clear the boolean accumulator.

in1: DINT variable or constant

in2: DINT variable or constant

GE_INT(*in1, in2*)

If in1 ≥ in2, boolean accumulator ← 1

If in1 < in2, boolean accumulator ← 0

Compare two single precision integers. If *in1* is greater than or equal to *in2*, set the boolean accumulator. If *in1* is less than *in2*, clear the boolean accumulator.

in1: INT variable or constant

in2: INT variable or constant

GE_REAL(*in1, in2*)

If in1 ≥ in2, boolean accumulator ← 1

If in1 < in2, boolean accumulator ← 0

Compare two REAL (floating point) values. If *in1* is greater than or equal to *in2*, set the boolean accumulator. If *in1* is less than *in2*, clear the boolean accumulator.

in1: REAL variable or constant

in2: REAL variable or constant

Regulator PID

PID

PID_IND(*address, sp, pv, man, up, dn*)

The PID_IND built-in function block governs the operation of an independent algorithm Proportional/Integral/Derivative control loop. Saves the Control Variable output to the integer accumulator. Must be followed by a ST_WORD instruction to store the accumulator's contents to a WORD variable.

address: WORD variable; the location of the first word of PID_IND built-in function block information, which consists of 40 consecutive registers of %R, %P, or %L memory.

sp: INT variable or constant; the control loop Set Point.

pv: INT variable; the control loop Process Variable.

man: BOOL variable; when true, PID_IND is in MANUAL mode.

up: BOOL variable; when PID_IND is in MANUAL mode and ***up*** is true, the Control Variable output to the accumulator is adjusted up,

dn: BOOL variable; when PID_IND is in MANUAL mode and ***dn*** is true, the Control Variable output to the accumulator is adjusted down.

PID_ISA(*address, sp, pv, man, up, dn*)

The PID_ISA built-in function block governs the operation of an ISA algorithm Proportional/Integral/Derivative control loop. Saves the Control Variable output to the integer accumulator. Must be followed by a ST_WORD instruction to store the accumulator's contents to a WORD variable.

address: WORD variable; the location of the first word of PID_ISA built-in function block information, which consists of 40 consecutive registers of %R, %P, or %L memory.

sp: INT variable or constant; the control loop Set Point.

pv: INT variable; the control loop Process Variable.

man: BOOL variable; when true, PID_ISA is in MANUAL mode.

up: BOOL variable; when PID_ISA is in MANUAL mode and ***up*** is true, the Control Variable output to the accumulator is adjusted up.

dn: BOOL variable; when PID_ISA is in MANUAL mode and ***dn*** is true, the Control Variable output to the accumulator is adjusted down.

Conversion Instructions

How do I... Related Topics

Use these IL instructions to convert data types, number formats and units in instruction list logic.

Convert REAL to DINT, INT, WORD

REAL_TO_DINT(*in*)

accumulator ← converted value

Convert the REAL *in* operand to a double-precision integer (DINT).

in: REAL variable; the value to be converted.

REAL_TO_INT(*in*)

accumulator ← converted value

Convert the REAL *in* operand to a single-precision integer.

in: REAL variable; the value to be converted.

REAL_TO_WORD(*in*)

accumulator ← converted value

Convert the REAL *in* operand to a WORD.

in: REAL variable; the value to be converted.

Convert INT to BCD4, REAL

INT_TO_BCD4(*in*)

accumulator ← converted value

Convert the single precision integer *in* operand to binary coded decimal format.

in: INT variable; the value to be converted.

INT_TO_REAL(*in*)

accumulator ← converted value

Convert the single precision integer (INT) *in* operand to a REAL value.

in: INT variable; the value to be converted.

Data Move Instructions

How do I... Related Topics

Use these IL instructions to transfer data between various memory locations from instruction list logic.

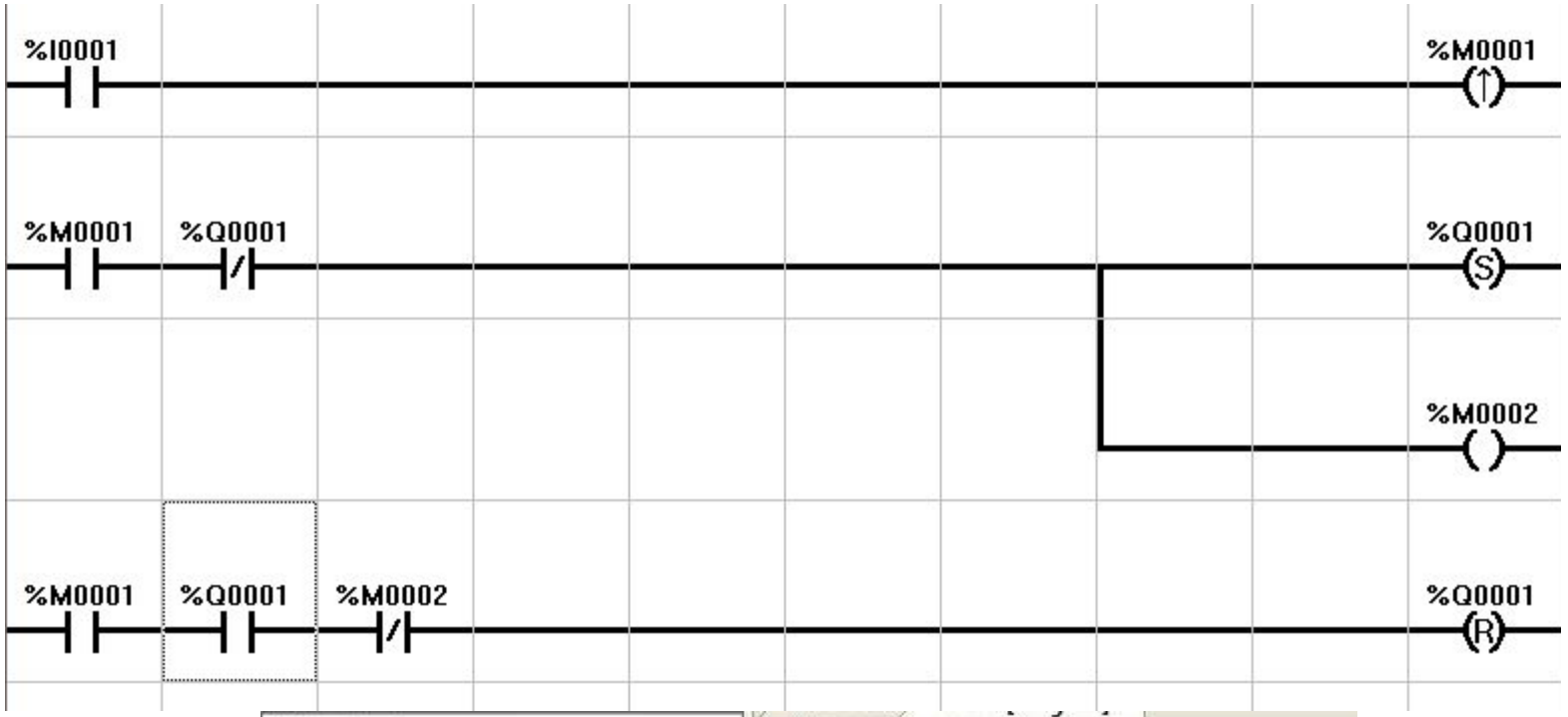
- Block Clear
- Block Move
- Communication Request
- Move Data
- Shift Register

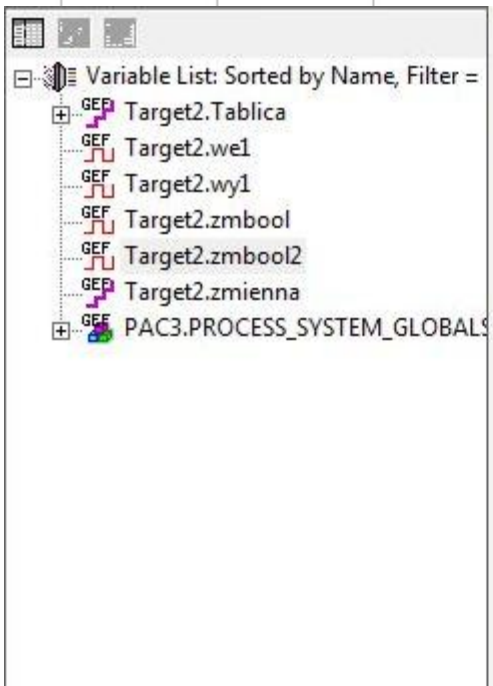
Bit Instructions

How do I... Related Topics

Use these IL instructions to perform bit-wise logical operations on binary strings (WORD/DWORD data).

- Bit Position
- Bit Sequencer
- Bit Set, Bit Clear
- Bit Test
- Logical AND
- Logical NOT
- Logical OR
- Logical XOR
- Masked Compare
- Rotate Bits Right, Rotate Bits Left
- Shift Bits Right, Shift Bits Left






```
'--------------------------------------
' Created: 16 październik 2016
'
' Description:
'
'--------------------------------------

LD_BOOL we1
PT zmbool

LDN_BOOL wy1
AND zmbool
ST_BOOL zmbool2
S wy1

LD_BOOL zmbool
AND wy1
ANDN zmbool2
R wy1
```

```
'---------------------------------------- wykorzystanie timerów i liczników

LD_BOOL we1
TMR_TENTHS(R1, 10)
ST_BOOL wy4


LD_BOOL we1
ONDTR_TENTHS(R4, bool_reset, 10)
ST_BOOL wy4

LD_BOOL #T_SEC
UPCTR(R7,  bool_reset2, 10)
ST_BOOL bool_reset2
```

Ostatni program zawiera konstrukcję licznika modulo 10 z funkcją autoresetowania

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTROTECHNIKI I AUTOMATYKI

```
'-------  program sygnalizacja świetlna

LD_BOOL wajchaSTART
PT zmpomoc
LD_BOOL zmpomoc
S stan1

LD_BOOL stan1
TMR_TENTHS(sw_timer1, 10)
R stan1
S stan2

LD_BOOL stan2
TMR_TENTHS(sw_timer2, 10)
R stan2
S stan3

LD_BOOL stan3
TMR_TENTHS(sw_timer3, 100)
R stan3
S stan4

LD_BOOL stan4
TMR_TENTHS(sw_timer4, 10)
R stan4
S stan5

LD_BOOL stan5
TMR_TENTHS(sw_timer5, 10)
R stan5
S stan6

LD_BOOL stan6
TMR_TENTHS(sw_timer6, 50)
R stan6
S stan7

LD_BOOL stan7
TMR_TENTHS(sw_timer7, 30)
R stan7
S stan1
```

```
LDN_BOOL wajchaSTART
R stan1
R stan2
R stan3
R stan4
R stan5
R stan6
R stan7

LD_BOOL stan1
OR stan2
OR stan5
OR stan6
OR stan7
ST_BOOL swiatlo_CZ_S

LD_BOOL stan2
OR stan4
ST_BOOL swiatlo_P_S

LD_BOOL stan3
ST_BOOL swiatlo_Z_S

LD_BOOL stan1
OR stan2
OR stan3
OR stan4
OR stan5
ST_BOOL swiatlo_CZ_P

LD_BOOL #T_SEC
AND stan7
OR stan6
ST_BOOL swiatlo_Z_P
```

Przykładowe programy w języku IL

```
'-------------------- petla: suma =0;  for (i=0; i<10; i++) {suma = suma + i;}
                        LD_INT 0
                        ST_INT suma
                        LD_INT 0
                        ST_INT i
                    poczatek_for:
                            LD_INT i
                            LT 10
                            JMPCN koniec_for
                            LD_INT suma
                            ADD i
                            ST_INT suma
                            LD_INT i
                            ADD 1
                            ST_INT i
                            JMP poczatek_for
                    koniec_for:
```

Przykładowe programy w języku IL

```
'-------------------------- funkcja IF we1 THEN wynik = 21 ELSE wynik = 12 endif
```

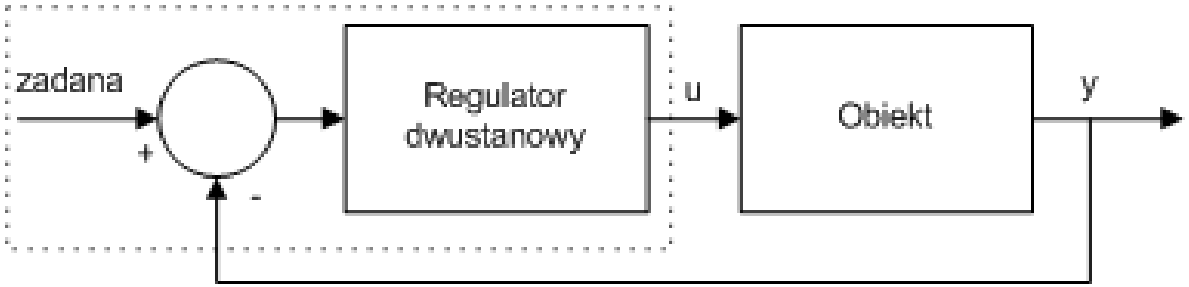
```
  LD_BOOL we1
  JMPCN skok_else
  LD_INT 12
  ST_INT wynik
  JMP koniec_if
skok_else:
  LD_INT 21
  ST_INT wynik
koniec_if:
```

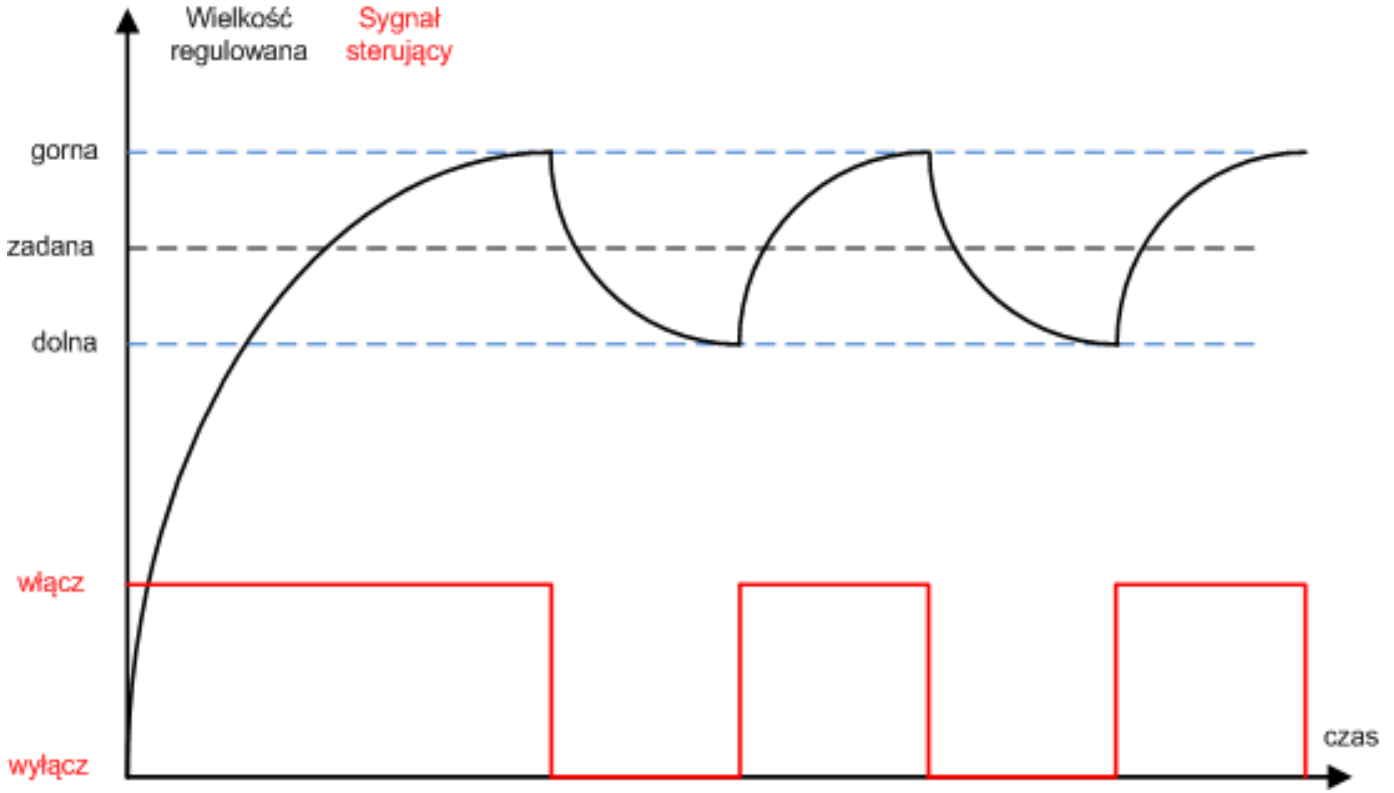
Przykładowe programy w języku IL

```
'---------------------------------- loczyn=1; x=1;  pętla while(x<5) {iloczyn=iloczyn*x; x++}

                                    LD_INT 1
                                    ST_INT x
                                    LD_INT 1
                                    ST_INT iloczyn
                                   poczatek_while:
                                    LD_INT x
                                    LT 5
                                    JMPCN koniec_while
                                    LD_INT iloczyn
                                   MUL x
                                    ST_INT iloczyn
                                    LD_INT x
                                    ADD 1
                                    ST_INT x
                                    JMP poczatek_while
                                  koniec_while:
```

Regulator dwustanowy w IL

```
        MUL_REAL(zadana, 1.1)
        ST_REAL gorna
        MUL_REAL(zadana, 0.9)
        ST_REAL dolna
        GT_REAL(y, gorna)
        JMPC skasuj
        LT_REAL(y, dolna)
        JMPC ustaw
        JMP koniec

ustaw:
        MOVE_REAL(100.0, 1, u)
        JMP koniec
skasuj:
        MOVE_REAL(0.0, 1, u)
koniec:
```

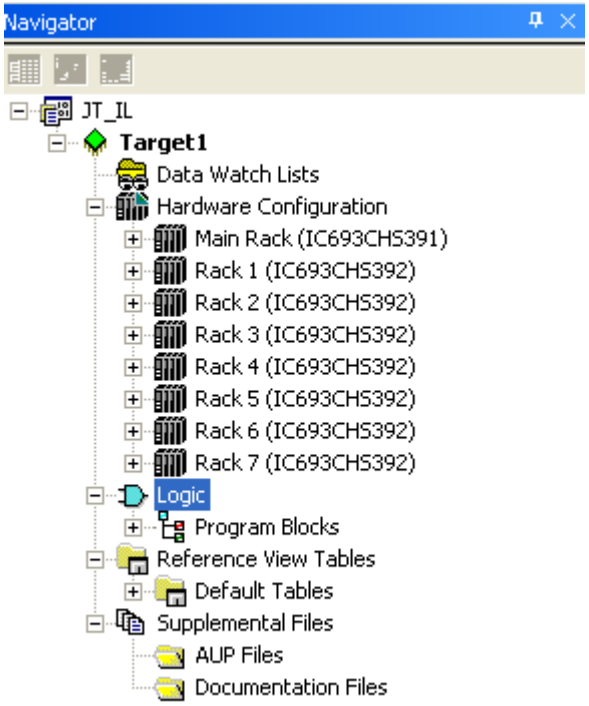


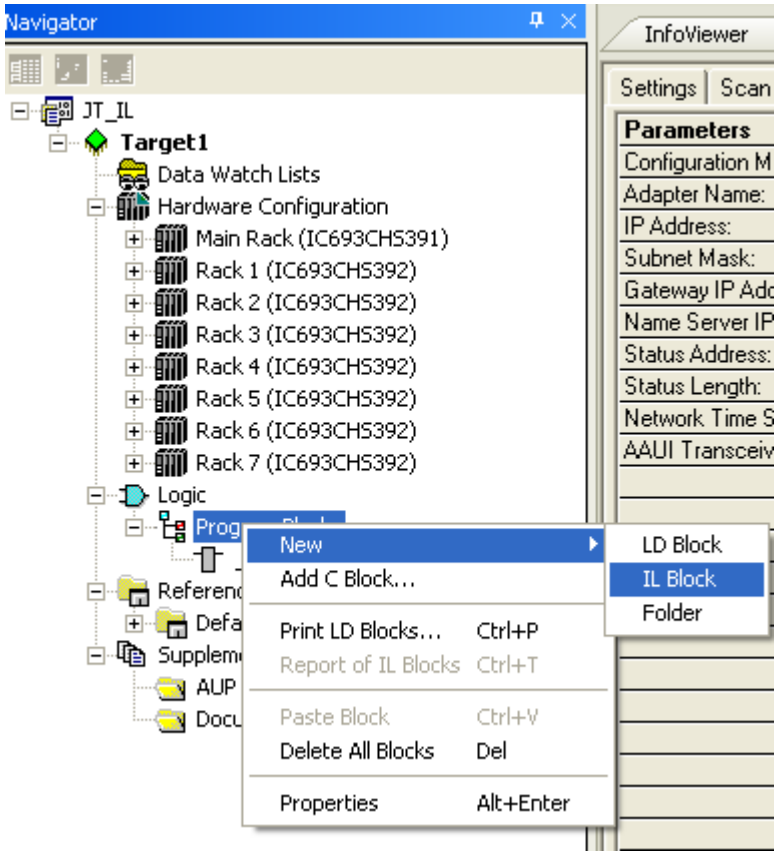


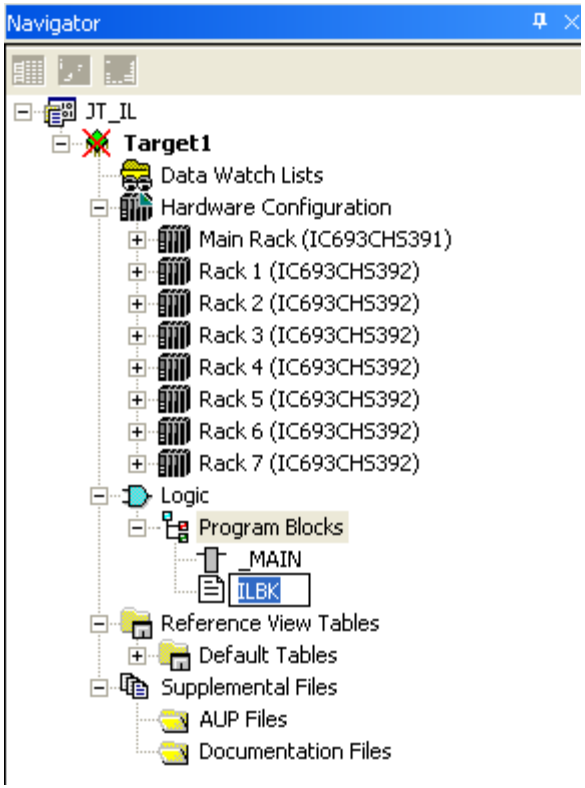
Komentarze w programie i podświetlanie składni

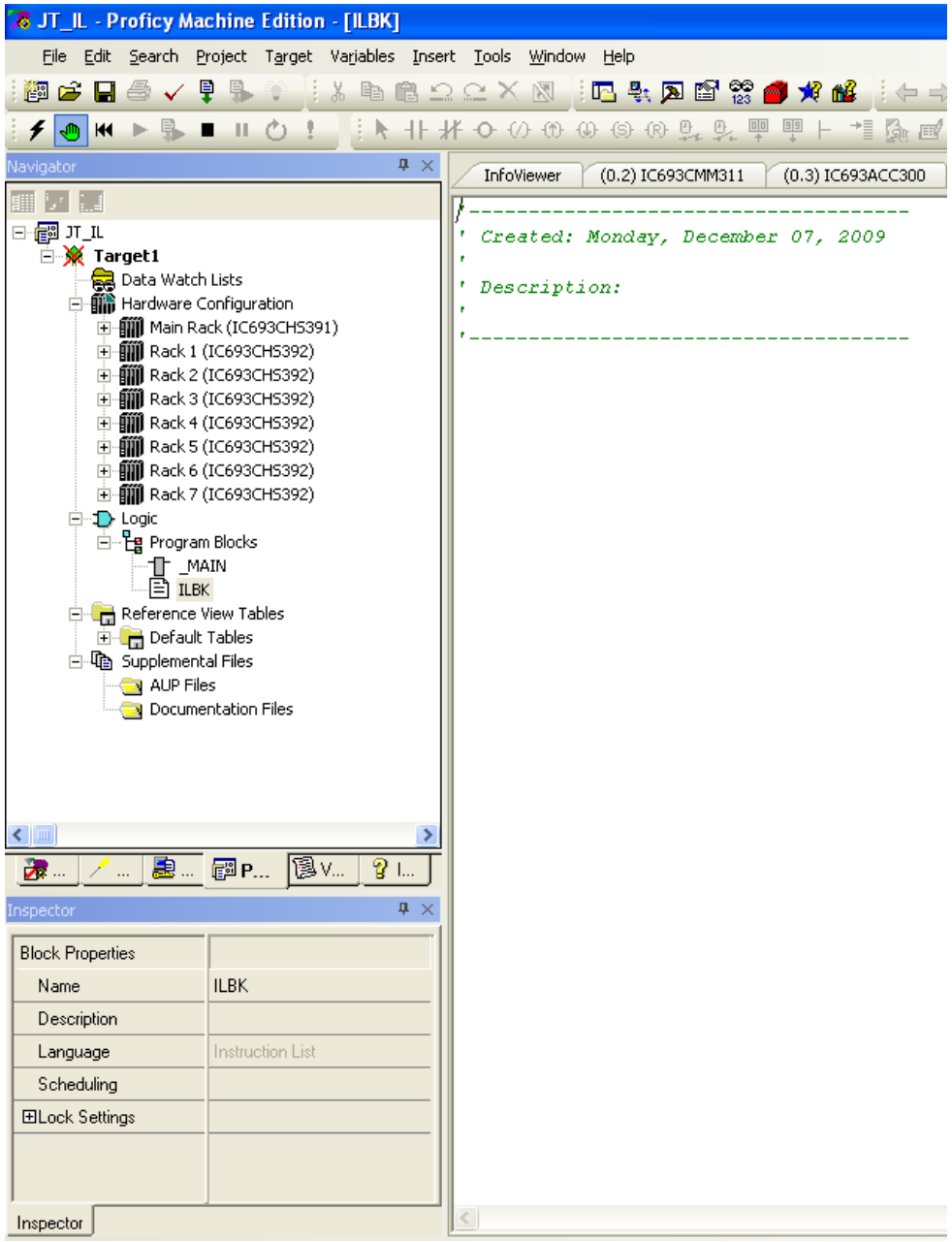
- Komentarzem w programie jest wszystko co znajduje się za znakiem apostrofu. Edytor posiada możliwość kolorowania składni (odróżniania słów kluczowych –mnemoników i

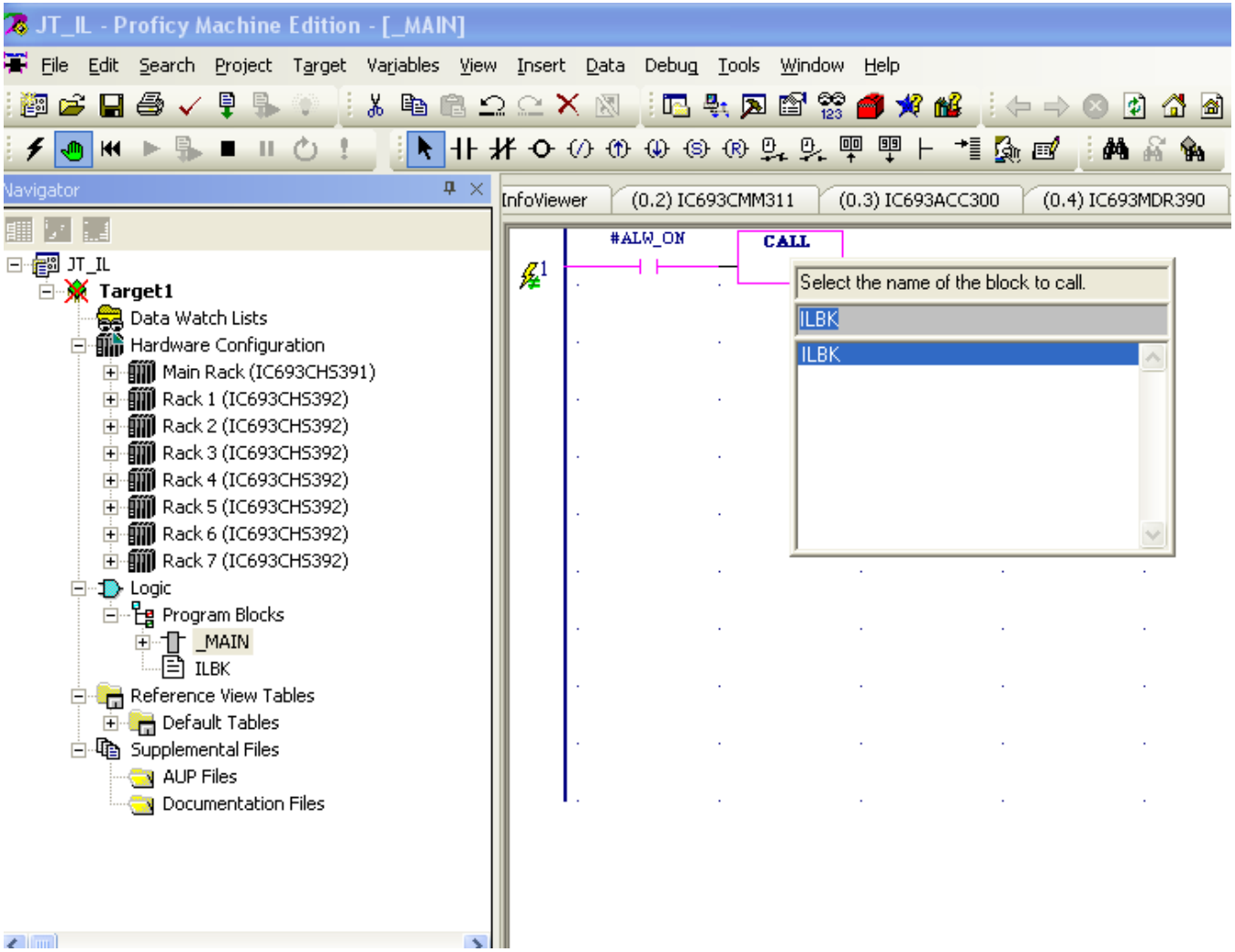
```
(*
The following is one way to solve ((A + B) x C) + (A - (D x C) + B).
This example shows how the accumulator and the stack accumulator are used in IL logic.
*)
    (*  Line number *)
LD C       '1  Load C into accumulator.
MUL(B      '2  Push C onto accumulator stack. Load B into accumulator.
 ADD A     '3  ADD A to B; load into accumulator.
 )         '4  Pop value of accumulator stack; multiply this value by the value in the
           '    accumulator; load into accumulator.
ADD(A      '5  Push accumulator value onto accumulator stack. Load A into accumulator.
 SUB(B     '6  Push accumulator value onto accumulator stack. Load B into accumulator.
  ADD(D    '7  Push accumulator value onto accumulator stack. Load D into accumulator.
   MUL C   '8  Multiply C by D; load into accumulator.
   )       '9  Pop value of accumulator stack; add this value to the value in the
           '    accumulator; load into accumulator.
  )        '10 Pop value of accumulator stack; subtract this value from the value in the
           '    accumulator; load into accumulator.
 )         '11 Pop value of accumulator stack; add this value to the value in the
           '    accumulator; load into accumulator.
ST myDINT  '12 Load value of accumulator into DINT variable myDINT.
```


Navigator
JT_IL
Target1
Data Watch Lists
Hardware Configuration
Main Rack (IC693CHS391)
Rack 1 (IC693CHS392)
Rack 2 (IC693CHS392)
Rack 3 (IC693CHS392)
Rack 4 (IC693CHS392)
Rack 5 (IC693CHS392)
Rack 6 (IC693CHS392)
Rack 7 (IC693CHS392)
Logic
Program Blocks
Reference View Tables
Default Tables
Supplemental Files
AUP Files
Documentation Files

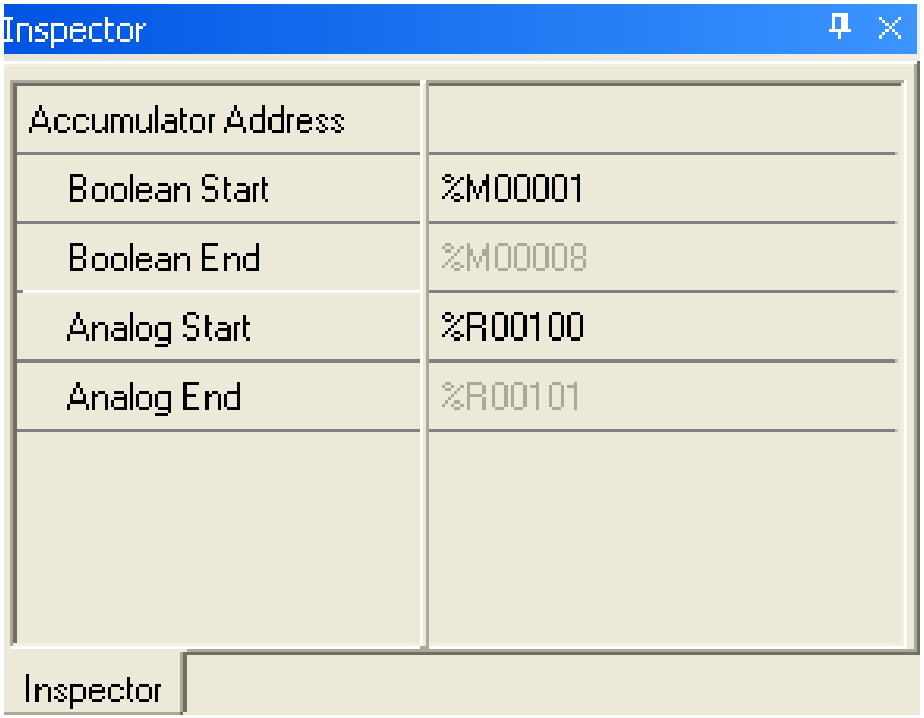

Navigator
JT_IL
Target1
Data Watch Lists
Hardware Configuration
Main Rack (IC693CHS391)
Rack 1 (IC693CHS392)
Rack 2 (IC693CHS392)
Rack 3 (IC693CHS392)
Rack 4 (IC693CHS392)
Rack 5 (IC693CHS392)
Rack 6 (IC693CHS392)
Rack 7 (IC693CHS392)
Logic
New
Add C Block...
Print LD Blocks... Ctrl+P
Report of IL Blocks Ctrl+T
Paste Block Ctrl+V
Delete All Blocks Del
Properties Alt+Enter
LD Block
IL Block
Folder
InfoViewer
Settings
Scan
Parameters
Configuration M
Adapter Name:
IP Address:
Subnet Mask:
Gateway IP Add
Name Server IP
Status Address:
Status Length:
Network Time S
AAUI Transceiv


Navigator
JT_IL
Target1
Data Watch Lists
Hardware Configuration
Main Rack (IC693CHS391)
Rack 1 (IC693CHS392)
Rack 2 (IC693CHS392)
Rack 3 (IC693CHS392)
Rack 4 (IC693CHS392)
Rack 5 (IC693CHS392)
Rack 6 (IC693CHS392)
Rack 7 (IC693CHS392)
Logic
Program Blocks
_MAIN
ILBK
Reference View Tables
Default Tables
Supplemental Files
AUP Files
Documentation Files


JT_IL - Proficy Machine Edition - [ILBK]
File Edit Search Project Target Variables Insert Tools Window Help
Navigator
JT_IL
Target1
Data Watch Lists
Hardware Configuration
Main Rack (IC693CHS391)
Rack 1 (IC693CHS392)
Rack 2 (IC693CHS392)
Rack 3 (IC693CHS392)
Rack 4 (IC693CHS392)
Rack 5 (IC693CHS392)
Rack 6 (IC693CHS392)
Rack 7 (IC693CHS392)
Logic
Program Blocks
_MAIN
ILBK
Reference View Tables
Default Tables
Supplemental Files
AUP Files
Documentation Files
InfoViewer
(0.2) IC693CMM311
(0.3) IC693ACC300
' Created: Monday, December 07, 2009
' Description:
Inspector
Block Properties
Name
ILBK
Description
Language
Instruction List
Scheduling
Lock Settings
Inspector


JT_IL - Proficy Machine Edition - [_MAIN]
File Edit Search Project Target Variables View Insert Data Debug Tools Window Help
Navigator
JT_IL
Target1
Data Watch Lists
Hardware Configuration
Main Rack (IC693CHS391)
Rack 1 (IC693CHS392)
Rack 2 (IC693CHS392)
Rack 3 (IC693CHS392)
Rack 4 (IC693CHS392)
Rack 5 (IC693CHS392)
Rack 6 (IC693CHS392)
Rack 7 (IC693CHS392)
Logic
Program Blocks
_MAIN
ILBK
Reference View Tables
Default Tables
Supplemental Files
AUP Files
Documentation Files
InfoViewer
(0.2) IC693CMM311
(0.3) IC693ACC300
(0.4) IC693MDR390
#ALW_ON
CALL
Select the name of the block to call.
ILBK
ILBK

Definiowanie adresów akumulatora i stosu



Zalety języka IL

- Duże zbliżenie do języka maszynowego
- Znajoma forma programu dla programujących w assemblerze
- Łatwa implementacja kompilatora
- Zwartość programu możliwość zmieszczenia na jednej stronie ekranu całej złożonej funkcji (niemożliwe do osiągnięcia w językach graficznych)
- Kontrola typów (wynikająca z konieczności korzystania z akumulatora)

Wady języka IL

- Znacznie mniejsza czytelność programu (w stosunku do języka drabinkowego)
- Korzystanie z pośrednictwa akumulatora i stosu akumulatora nie jest intuicyjne
- Dostosowanie użytkownika do maszyny, a nie maszyny do użytkownika
- Nauka tego rodzaju programowania jest trudniejsza niż języka drabinkowego

Bibliografia

- INTERNATIONAL STANDARD IEC 61131-3, Second edition 2003-01
- Legierski T., Kasprzyk J., Wyrwał J., Hajda J.: „Programowanie sterowników PLC”, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 1998
- Dokumentacja GE Fanuc,

POLITECHNIKA
GDAŃSKA

HISTORIA MĄDROŚCIĄ
PRZYSZŁOŚĆ WYZWANIEM