Nowy materiał do opanowania. Zasady programowania sterowników PLC w języku tekstowym listy instrukcji IL (*Instruction List*) oraz strukturalnym ST (STL) (*Structured Text*)

Ciąg dalszy prezentowania zasad programowania, tworzenia programów oraz stosowania języków tekstowych.

Cerdo: Inny język nie oznacza zmiany podejścia do programowania, ale wykorzystanie innych symboli (tak jak w różnych językach na świecie używane są inne słowa do wyrażenie tej samej myśli).

Grupę języków tekstowych zgodnie z Normą EN 61131-3 tworzą dwa języki:

- język strukturalny ST (*Structured Text*) – odpowiednik algorytmicznego języka wysokiego poziomu takiego jak FORTRAN, PASCAL, C++, PYTHON zawierającego struktury programowe takie, jak: IF...WHILE…, FOR... (Ten rodzaj programowania już Państwo znacie).
- język listy instrukcji IL (*Instruction List*) (odpowiednik STL) – odpowiednik języka typu asembler, którego zbiór instrukcji obejmuje operacje logiczne, arytmetyczne, operacje rotacji, jak również funkcje przerzutników, czasomierzy, liczników itp.;

Język strukturalny ST (*Structured Text*)

Język strukturalny wykorzystuje struktury algorytmicznych języków programowania wyższego poziomu, takich jak Pascal, Python czy C/C++. Podstawowymielementami są zdania logiczne zawierające warunki oraz polecenia związane z wyrażeniami, które składają się z operatorów i operandów. Najczęściej wykorzystywane struktury zdań języka ST przedstawiono poniżej. Dwa pierwsze są strukturami zdań umożliwiającymi dokonywanie wyboru.

IF *warunek* **THEN** *polecenie1* **ELSE** *polecenie2*

W czasie realizacji tego fragmentu programu sprawdzany jest *warunek* występujący po słowie kluczowym IF. Jeżeli jest on spełniony (wyrażenie logiczne z nim związane jest spełnione) następuje wykonanie *polecenia1* występującego po słowie THEN. W przeciwnym wypadku wykonywane jest *polecenie2*, które występuje po słowie ELSE. To samo realizowane jest w C++ poprzez instrukcję:

if (*warunek*) { *polecenie1* }**else**{ *polecenie*2} w C++

Operacja wielokrotnego wyboru:

CASE *selektor* **OF** *lista poleceń z etykietami* **ELSE** *inne polecenia*

Konstrukcja ta umożliwia wykonanie różnych poleceń w zależności od wartości, jaką przyjmuje *selektor*, który jest wyrażeniem typu INTEGER. *Lista poleceń* zawiera grupy poleceń, z których każda charakteryzuje się swoją unikalną *etykietą*. W czasie realizacji programu wykonana zostanie ta grupa poleceń, której etykieta odpowiada obliczonej wartości *selektora*. Gdy wyliczona wartość nie odpowiada żadnej *etykiecie*, będą *wykonane inne polecenia*, które zostały umieszczone po słowie ELSE. To samo realizuje w C++ instrukcja:

switch (*selektor*) {

case *etykieta 1*: *1 polecenie z lista poleceń; break*;

case *etykieta 2*: *2 polecenie z lista poleceń; break*;

….

case *etykieta n*: *n polecenie z lista poleceń; break*;

default: *inne polecenia; break;*

} w C++

Drugą grupę konstrukcji używanych w języku ST stanowią konstrukcje iteracyjne, umożliwiające powtarzanie wybranego fragmentu programu.

FOR *wartość początkowa* **TO** *wartość końcowa* **BY** *krok* **DO** *polecenia*

Struktura ta pozwala na iteracyjne wykonanie *polecenia* tyle razy, ile wynika to z odjęcia *wartości początkowej* od *wartości końcowej* z określonym *krokiem*. Służy ona do programowania zadań iteracyjnych w przypadku znajomości liczby potrzebnych iteracji. To samo realizuje w C++ instrukcja:

for (int i = *wartość początkowa;* i< *wartość końcowa;* i=i+ *krok*)

{ *polecenia* }

Inna instrukcja iteracyjna:

WHILE *warunek* **DO** *polecenia* **REPEAT** *polecenia* **UNTIL** *warunek*

Obydwie te konstrukcje służą do zaprogramowania cyklicznego wykonywania *poleceń*, gdy liczba iteracji nie jest z góry znana, a decyduje o niej *warunek* podany w zdaniu po słowie WHILE lub UNTIL. W języku C++ realizuje to instrukcja:

do { *polecenia* }

while (*warunek=true*)

a więc instrukcja **do-while**, inna niż „sama” **while**.

Inne operatory języka ST

Operacja	Symbol	Operacja	Symbol
nawiasy	(*wyrażenie*)	dodawanie	+
obliczanie wartości funkcji	nazwa(lista argumentów)	odejmowanie	-
potęgowanie	**	porównanie	<,>,<=,>=
wartość przeciwna	-	równość	=
uzupełnienie	not	nierówność	<>
mnożenie	*	iloczyn logiczny	&, and
dzielenie	/	suma logiczna modulo 2	xor
reszta z dzielenia	mod	suma logiczna	or

Jak widać język ST oraz jego wykorzystanie/stosowanie w programowaniu sterowników PLC jest bardzo podobne do poznanego już programowania proceduralnego np. mikrokontrolerów.

W dalszej części wprowadzone będą szczegóły dotyczące języka IL, podobny do asemblera, o którym wspomniano w materiałach z Diagnostyki z dnia 06.05.2020 (od strony 6)

Język listy instrukcji IL (*Instruction List*), STL (STatement List),

Cechą charakterystyczną języka IL jest sekwencja instrukcji podobnych do kodów programowania mikroprocesora w języku asembler. Każda instrukcja zaczyna się w nowej linii. Instrukcja zawiera nazwę operatora z ewentualnymi modyfikatorami oraz operand (jeden lub więcej, oddzielone przecinkami, w zależności od wymagań operatora). Instrukcja może być poprzedzona przez etykietę zakończoną dwukropkiem, natomiast ewentualny komentarz powinien być ostatnim elementem linii. Między instrukcjami można wprowadzać puste linie.

Listę standardowych operatorów i dozwolonych modyfikatorów przedstawiono w poniższej tabeli:

Standardowe operatory i modyfikatory IL (STL)

Operator	Modyfikator	Operand	Opis
LD	N, NOT	zmienna, stała	operand do wynik bieżący
ST	N	zmienna	wynik bieżący do operand
S		zmienna logiczna	ustaw wartość operandu
R		zmienna logiczna	wyzeruj wartość operandu
AND , A	N, (	zmienna logiczna	logiczne AND
&	N, (	zmienna logiczna	logiczne AND
OR , O	N, (	zmienna logiczna	logiczne OR
XOR	N, (	zmienna logiczna	logiczna suma z wyłączeniem
ADD	(	zmienna, stała	dodawanie
SUB	(	zmienna, stała	odejmowanie
MUL	(	zmienna, stała	mnożenie
DIV	(	zmienna, stała	dzielenie
GT	(	zmienna, stała	porównanie: >
GE	(	zmienna, stała	porównanie: >=
EQ	(	zmienna, stała	porównanie: =
LE	(	zmienna, stała	porównanie: <>
LT	(	zmienna, stała	porównanie: <=
NE	(	zmienna, stała	porównanie: <
)			ograniczenie modyfikatora
CAL	C, N	nazwa	wywołanie bloku funkcyjnego
JMP	C, N	etykieta	skok do etykiety
RET	C, N		powrót z wywołania

Najczęściej poszczególne operatory działają w ten sposób, że wartość wyrażenia jest obliczana jako wynik działania operatora na wartość bieżącą wyrażenia z uwzględnieniem wartości operandu. Operand to zmienna, liczba, parametr, który podlega działaniu reprezentowanego przez operator. np. instrukcję AND %IX1 należy interpretować jako (operandy - wynik, %IX1, operator – ***and***):

wynik = wynik ***and*** %IX1

Wyrażenie podstawowe może zostać uzupełnione o modyfikator w postaci nawiasów. Lewy nawias „(” oznacza wstrzymanie wykonania operacji aż do napotkania prawego nawiasu „)”. Następująca sekwencja rozkazów:

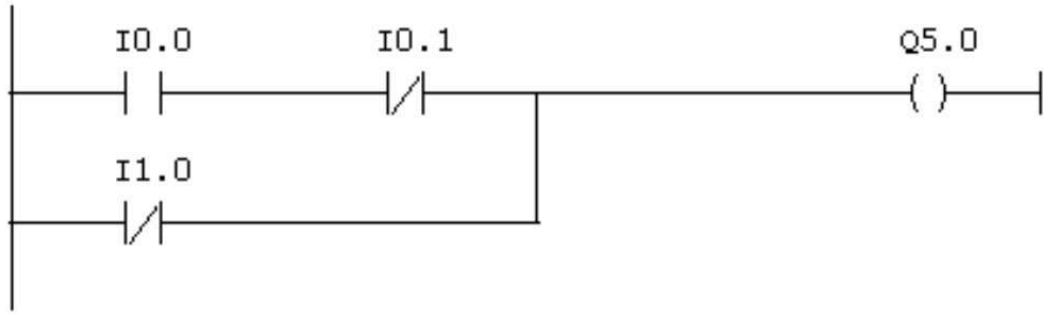
AND (%IX1 OR %IX2) zostanie zinterpretowana jako:

wynik = wynik ***and*** (%IX1 ***or*** %IX2).

Podsumowując tymczasowo dotychczasową wiedzę, porządkującą metod programowania PLC:

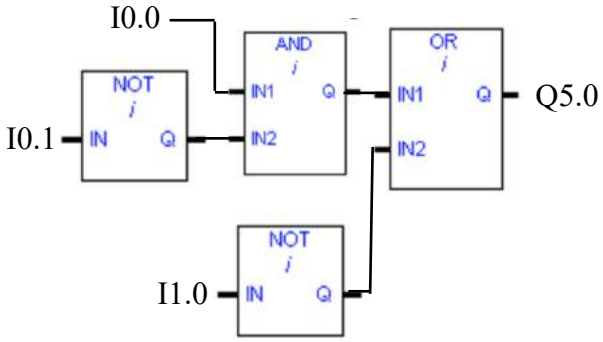
Pierwszy sposób, najbardziej intuicyjny dla osób mających doświadczenie w projektowaniu obwodów elektrycznych, to schemat drabinkowy (stykowy) – LAD (**LA**dder **D**iagram). Wykorzystuje się tutaj elementy, które nawiązują do obwodów elektrycznych.

Przykładowy program w LD(LAD):



Drugi poznany sposób to schemat funkcyjny – FBD (**F**unction **B**lock **D**iagram)**.** Ten sposób reprezentacji nawiązuje do projektowania układów elektronicznych. Programista ma więc do czynienia z bramkami AND, OR, XOR oraz bloczkami realizującymi określone funkcje.

Powyższy program w odmianie FBD:



Trzeci teraz omawiany sposób to STL (**ST**atement **L**ist), **IL** (**I**nstruction **L**ist), w swojej formie przypomina assembler (język instrukcji procesora). Zapis STL i IL będzie stosowany zamiennie. Ten sposób programowania będzie najbardziej intuicyjny dla osób, które mają doświadczenie w programowaniu w językach niskiego poziomu np. w programowaniu mikrokontrolerów.

Powyższy program (program w LD i FBD powyżej) może wyglądać, następująco (patrz: tabela *Standardowe operatory i modyfikatory IL (STL)*)::

```
A     I     0.0
AN    I     0.1
ON    I     1.0
=     Q     5.0
```

lub bardziej sugestywnie (patrz: tabela *Standardowe operatory i modyfikatory IL (STL)*):

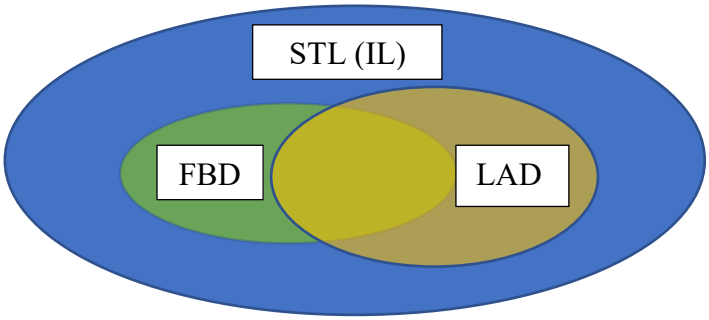
```
AND   I     0.0
ANDN  I     0.1
ORN   I     1.0
=     Q     5.0
```

Rodzi się pytanie: który język zapisu programu wybrać?

Jeżeli pozostawiono, NAM jako programistom, wybór (nie jest narzucony przez standard zakładu, ani przez wymagania zdefiniowane przez klienta) można wybrać ten, w którym autor programu czuje się najswobodniej.

Istnieją jednak pewne ograniczenia. Dużą część programów można zapisać w LAD lub FBD, ale istnieją pewne operacje (np. adresowanie pośrednie), w których programista nie ma wyboru i musi skorzystać z zapisu STL.

Dostępność funkcji w poszczególnych językach można przedstawić na podstawie poniższego diagramu obszarowego:



Obszary LAD i FBD w przeważającej części pokrywają się. Jednak wszystko to, co można zrealizować w językach LAD i FBD, można zrealizować w STL i znacznie więcej. Można wymienić wady i zalety STL (IL):

Zalety języka STL (IL)

+ większe możliwości niż LAD czy FBD;
+ większa swoboda programowania niż w językach graficznych;
+ szybsze pisanie kodu (po osiągnięciu pewnej wprawy);
+ możliwość umieszczania komentarza do każdej linii kodu;

Wady STL (IL)

-trudniejsza, mniej intuicyjna analiza kodu programu;
-wymagane większe doświadczenie programisty, a przede wszystkim od osoby pracującej z programem;

Przejdźmy do instrukcji do tworzenia instrukcji:

W linii kodu w STL można wyróżnić następujące elementy:

etykieta: rozkaz parametr // komentarz do operacji

Linia kodu będzie najczęściej zawierać **rozkaz (operację).** Może to być rozkaz bezparametrowy, ale częściej jednak wykorzystywane są rozkazy występują z **parametrami,** np. znane z powyższego, pierwszego programu:

etykieta: rozkaz parametr // komentarz do operacji

```
(tu brak etykiety:)    A          I 1.2       //sprawdź czy I1.2 = 'true'
```

jeżeli dana linia kodu ma być oznaczona **etykietą,** to jej deklaracja pojawi się z lewej strony, np.:

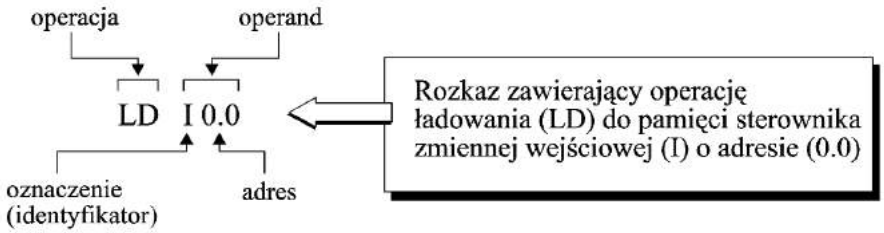
```
etyk: A I 1.2
```

Każda linia kodu może kończyć się **komentarzem,** dodanie komentarza pozwala na zwiększenie czytelności programu. Komentarz rozpoczyna się przy pomocy dwóch ukośnych kresek, np.:

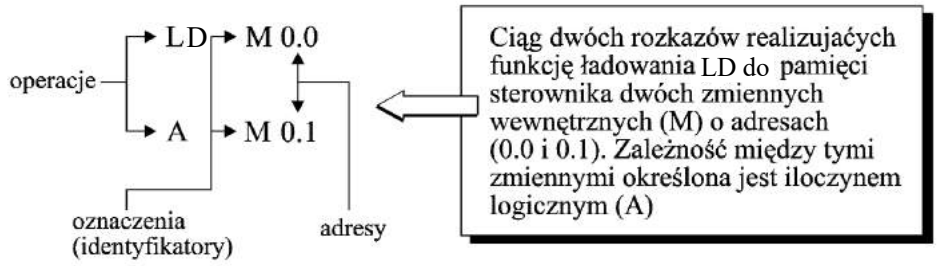
```
etyk: A I 1.2 // sprawdź wejście START
```

Przykład innej operacji(patrz: tabela *Standardowe operatory i modyfikatory IL (STL)):*

```
LD I 0.0  //Rozkaz zawierający instrukcję ładowania zmiennej I0.0
```




```
LD M 0.0  //Ciąg instrukcji-1 pierwsza instrukcja
A  M 0.1  //Ciąg instrukcji-2 pierwsza instukcja
```



Pierwszy program z zastosowaniem języka STL (IL)- wykorzystano tu podstawową funkcję AND

Układ wyzwalający prasę - idea układu sterującego jest prosta:

Aby uchronić operatora prasy przed skaleczeniem rąk zastosowano specjalny układ wyzwalający. Wyzwolenie prasy może nastąpić tylko i wyłącznie w momencie równoczesnego naciśnięcia dwóch przycisków S1 i S2 rozmieszczonych tak, aby nie było możliwe naciśnięcie obydwu przycisków jedną ręką.

Idea układu sterującego może być reprezentowana przez następujący schemat elektryczny, funkcjonalny:

Przyporządkowanie adresów wejścia/wyjścia sterownika do poszczególnych elementów:

Element	Funkcja	Adres
S1	Przycisk lewej ręki	I 0.3
S2	Przycisk prawej ręki	I 0.4
K1	Stycznik wyzwalający prasę	Q 4.3

Czyli program ma działać następująco:

1. Należy sprawdzić czy wejście I0.3 sterownika (przycisk S1) jest w stanie **wysokim** (przycisk S1 jest wciśnięty) **oraz** czy wejście I 0.4 także jest w stanie **wysokim** (przycisk S2 jest także w tym czasie wciśnięty).
2. Jeżeli te dwa warunki są jednocześnie spełnione należy wysterować (stan wysoki) wyjście Q 4.3 (podać prąd na cewkę stycznika K1).

Program w STL wygląda następująco:

```
A     I 0.3    // Jeżeli wejście I 0.3 jest w stanie wysokim
A     I 0.4    // i wejście I 0.4 jest w stanie wysokim
=     Q 4.3    // wtedy wysteruj wyjście Q 4.3
```

Jak to działa dokładnie?

Operacja A realizuje iloczyn logiczny pomiędzy aktualnym stanem RLO oraz aktualnym stanem parametru (np. I0.4). Wynik tej operacji jest zapisywany do RLO i dostępny dla następnej operacji. Dla pierwszej operacji ma miejsce bezpośrednie przepisanie wyniku sprawdzenia do RLO.

*(**RLO-R**esult of **L**ogic **O**perations. Pojęcie Wiąże się z pojęciem stosu i rejestrów, patrz materiały z Diagnostyki z 06.05.2020 oraz 29.04.2020 oraz załącznik na końcu materiałów).*

W powyższym przykładzie operacja:

```
A     I  0.3      // Jeżeli wejście I 0.3 jest w stanie wysokim
```

przepisuje stan wejścia I0.3 bezpośrednio do RLO (ponieważ jest to pierwsza operacja).

W następnej operacji dokonywana jest operacja logiczna AND (iloczyn, połączenie szeregowe) pomiędzy RLO oraz stanem wejścia I 0.4 - wynik tej operacji jest zapisywany do RLO:

```
A   I 0.4     // i wejście I 0.4 jest w stanie wysokim
```

W ostatniej operacji bieżący stan RLO przepisany jest na wyjście Q 4.3:

```
=  Q 4.3 //wtedy wysteruj(aktywuj, podaj prąd)wyjście Q 4.3
```

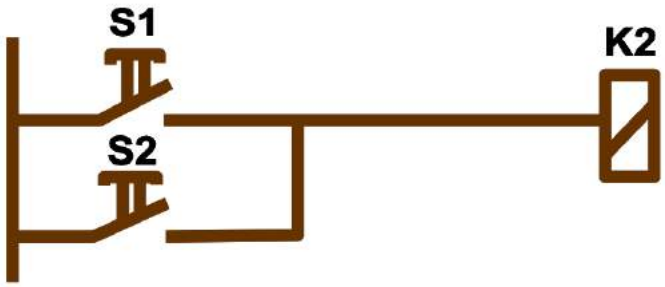
Drugi program z zastosowaniem języka STL (IL)- wykorzystano tu podstawową funkcję OR

Układ sterowanie wentylatorem – zasada jest również prosta:

W hali zainstalowane są dwa generatory spalinowe. Praca generatorów jest nadzorowana przez czujniki. Jeżeli generator pracuje czujnik zwraca stan wysoki. Praca pierwszego generatora sygnalizowana jest czujnikiem, włączającym styk S1, zaś drugiego przez czujnik, włączający styk S2. W przypadku pracy któregokolwiek z generatorów powinien pracować wentylator uruchamiany przekaźnikiem K2 zapewniającym prawidłowe przewietrzenie hali.

Zatem wentylator powinien pracować, gdy pracuje pierwszy lub drugi generator (ewentualnie obydwa). Logika jaka powinna być użyta do zrealizowania tego zadania może być przedstawiona za pomocą poniższego schematu elektrycznego.

Idea układu sterującego może być reprezentowana przez następujący schemat elektryczny, funkcjonalny:



Przyporządkowanie adresów wejścia/wyjścia do poszczególnych elementów:

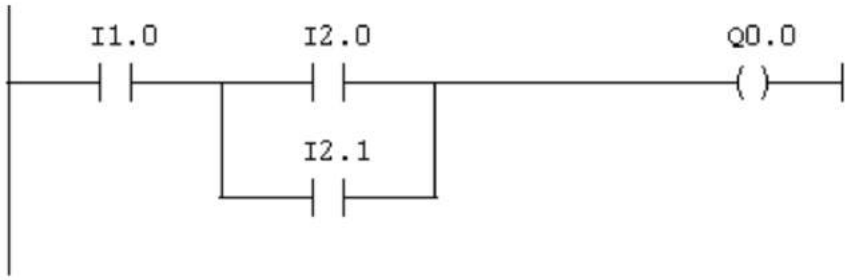
Element	Funkcja	Adres
S1	Generator 1	I 0.1
S2	Generator 2	I 0.2
K2	Wentylator	Q 8.3

Funkcja logiczna, która zrealizuje taką logikę to OR, czyli połączenie równoległe. Program w języku STL opisujący tę logikę ma następującą postać:

```
O     I 0.1      // Jeżeli wejście I 0.1 jest w stanie wysokim
O     I 0.2      // lub wejście I 0.2 jest w stanie wysokim
=     Q 8.3      // wtedy wysteruj wyjście Q 8.3
```

Operacje grupowania

W programach pisanych do tej pory nie było konieczności użycia operacji grupowania, czyli nawiasów. Jeżeli zaistniałaby konieczność zapisania następującej logiki w STL:



można by zapisać ją w następujący sposób:

```
A     I 1.0
A(
O     I 2.0
O     I 2.1
)
=     Q 0.0
```

Czyli w nawiasie został wypracowany wynik sumy dwóch sygnałów przemnożony logicznie przez sygnał I 1.0.

Uwaga:

W tym konkretnym przykładzie – odwracając kolejność składników iloczynu – można by zrezygnować z użycia nawiasów:

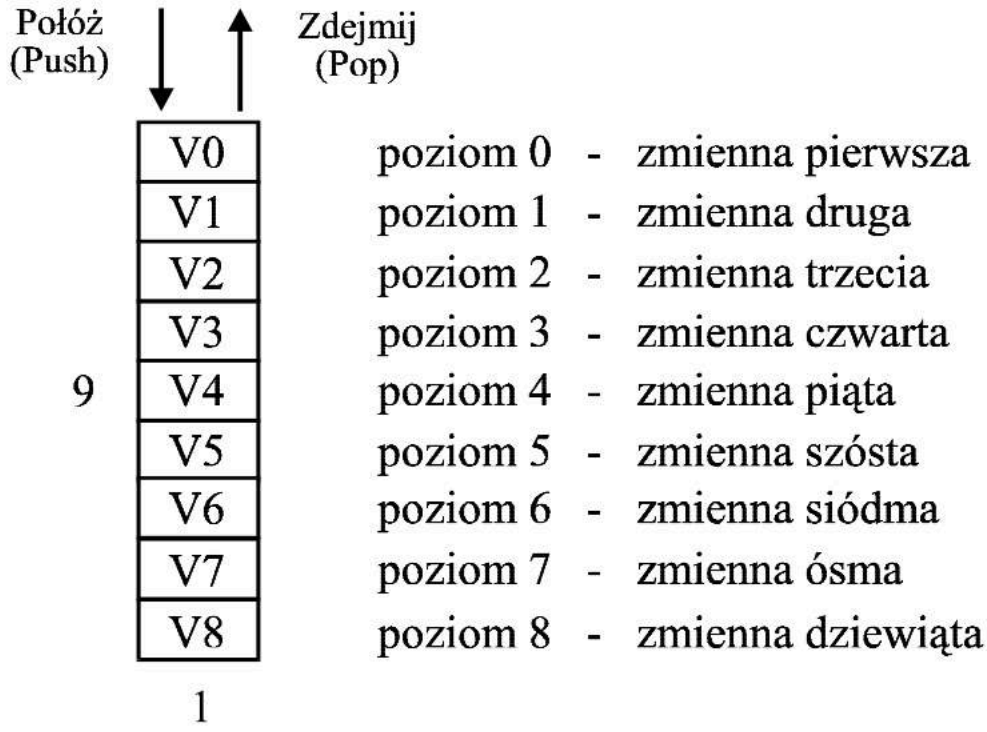
```
O     I 2.0
O     I 2.1
A     I 1.0
=     Q 0.0
```

ale nie zawsze będzie to możliwe.

Załącznik-uzupełnienie-przypomnienie:

Sterownik mikroprocesorowy wykonując dany program analizuje każdy pojedynczy rozkaz osobno krok po kroku w kolejności ich występowania począwszy od rozkazu pierwszego, a skończywszy na ostatnim (z góry na dół). Po realizacji ostatniego rozkazu mikroprocesor rozpoczyna proces analizy od początku. Proces ten nazywa się cyklicznym wykonywaniem programu, a czas potrzebny na przeanalizowanie wszystkich rozkazów danego programu nazywa się czasem jednego cyklu.

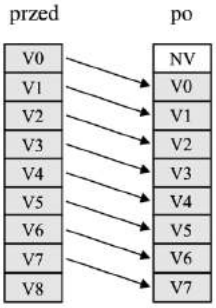
W celu dokładniejszego zrozumienia idei pisania programów w języku STL oraz umiejętności poprawnej analizy zapisu takiego programu ważna jest świadomość tego, w jaki sposób jednostka CPU wykorzystuje tzw. STOS. Stos stanowi specyficzny sposób organizacji przechowywania danych w pamięci sterownika. Reprezentację graficzną stosu przedstawia poniższy rysunek (Stos o wymiarze 9 x 1 bitów):



Stos przypomina strukturę wykorzystującą 9 bitów ułożonych jeden na drugim. Podstawową zasadą wykorzystania stosu jest zdejmowanie (Pop) ze stosu danych w odwrotnej kolejności do ich układania (Push). Praktycznie każda operacja logiczna i blok funkcyjny w języku STL (IL) są bezpośrednio związane są z wykorzystaniem STOS-u. Przykłady przedstawione na następnej stronie omawiają sposób wykorzystania STOS-u:

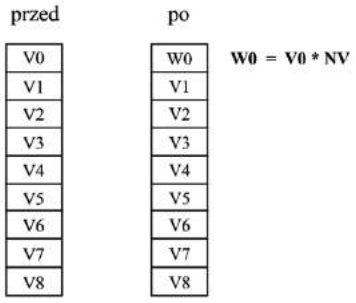
LD

Ładuj nową wartość (NV). Operacja ta powoduje położenie na stos wartości (NV) przy jednoczesnym przesunięciu wszystkich wartości początkowych stosu o jedną pozycję



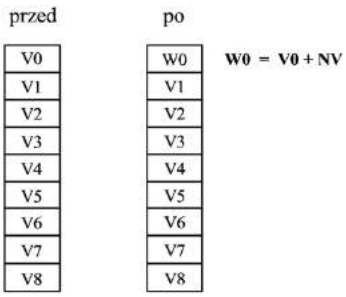
A

Iloczyn logiczny nowej wartości (NV) z najwyższą wartością początkową stosu (V0). Wynik operacji umieszczoany jest na stosie.

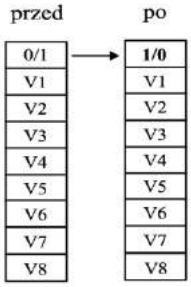


O

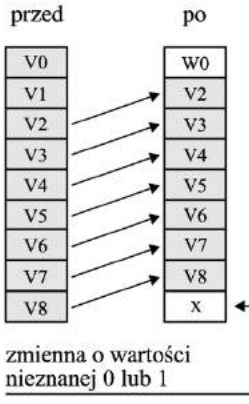
Suma logiczna nowej wartości (NV) z najwyższą wartością początkową stosu (V0). Wynik operacji umieszczany jest na stosie.



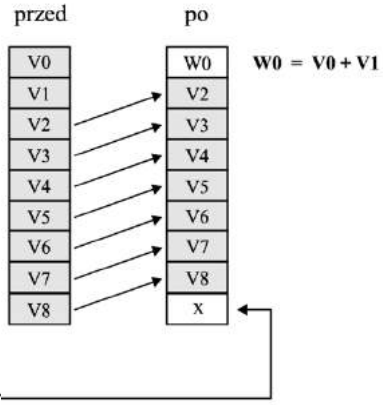
logiczne NIE



Iloczyn logiczny dwóch pierwszych wartości początkowych stosu (V0 i V1)

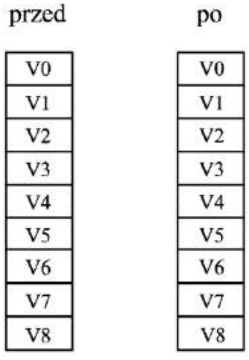


Suma logiczna dwóch pierwszych wartości początkowych stosu (V0 i V1).



=

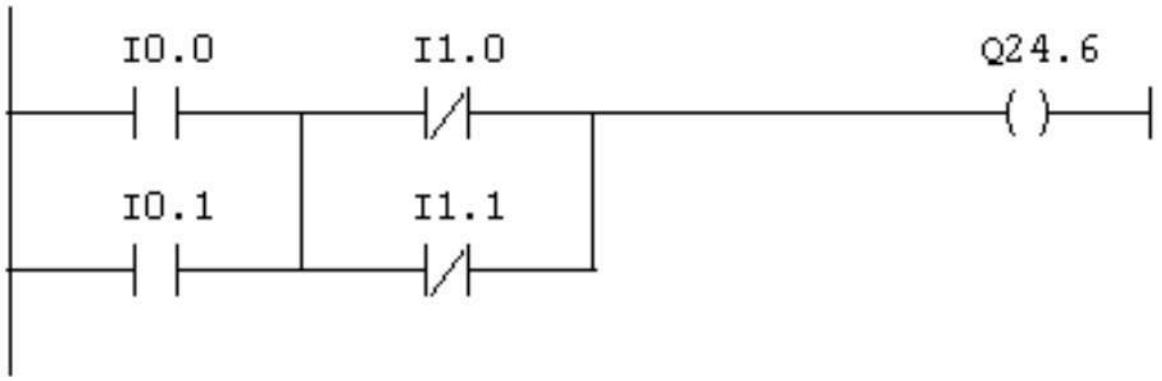
Funkcja przyporządkowania kopiuje pierwszą wartość początkową stosu (V0) pod wskazany adres. Zawartość stosu nie ulega zmianie



-----Koniec-----

Zadanie do wykonania i odesłania pocztą zwrotną.

Proszę zapiać w jeżyku STL (IL), który jest realizowany w sposób jak przedstawia to schemat zapisany w języku LD (LAD):



Należy używać symboli z tabeli *Standardowe operatory i modyfikatory IL (STL):*
A, O, AN, ON, =.

……

Zalecana literatura z przedmiotu Programowanie urządzeń mechatronicznych, do tej i poprzednich lekcji:

„GE Fanuc Automation, Sterowniki programowalne, Opis funkcji.” Kraków, wrzesień 1999.

„Programowanie sterowników PLC”, Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 1998, Autorzy: T. Legierski, J. Kasprzyk, J. Hajda, J. Wyrwał.