ML-PROG
Opis programu i języka

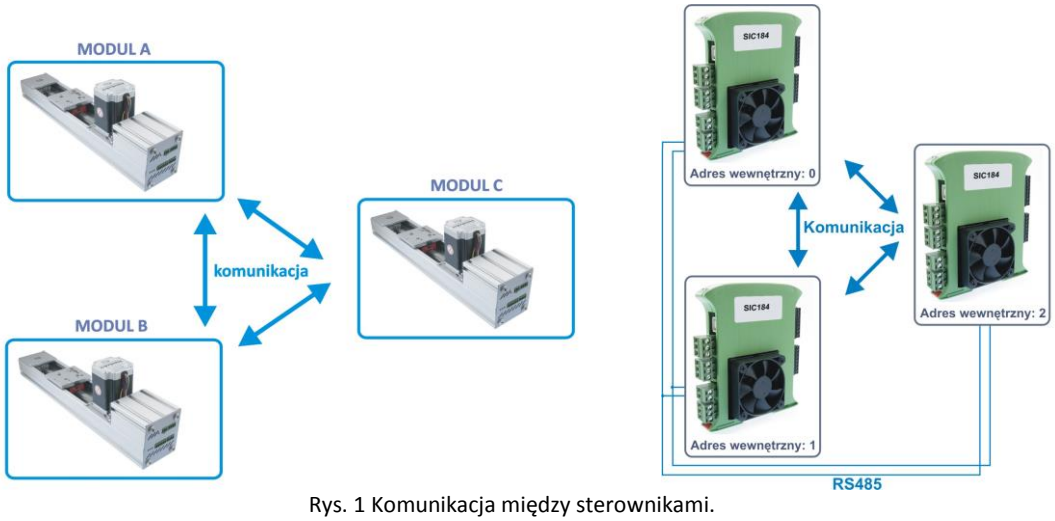
1. Wstęp

Oprogramowanie ML-PROG służy do tworzenia programów ruchu dla sterowników silników krokowych firmy WObit wyposażonych w programowalny generator trajektorii. Oprogramowanie współpracuje z:

- Modułami liniowymi **MLA-SI**
- Sterownikami **SIC184**

Programowanie w środowisku ML-PROG odbywa się w prostym języku tekstowym poprzez wprowadzanie odpowiednich komend. Przykładowo dla zadania ruchu 200mm wprowadza się komendę ***SetXdest(A,200)***, *gdzie* ***A*** oznacza adres sterownika, którego ma dotyczyć ta komenda. **Każda** komenda w programie zawiera adres modułu. Podczas programowania modułów konkretne komendy są przesyłane do odpowiedniego sterownika (jeśli podłączonych jest kilka sterowników na jednej magistrali RS485). Dzięki temu istnieje możliwość jednoczesnego programowania do 16-stu sterowników.

Synchronizacja między sterownikami odbywa się poprzez wysyłanie komunikatów przez dany moduł. Pozostałe moduły odbierają komunikaty i sprawdzają, czy któryś tyczy się niego. Jeśli tak, może wykonać zaprogramowaną wcześniej akcję.



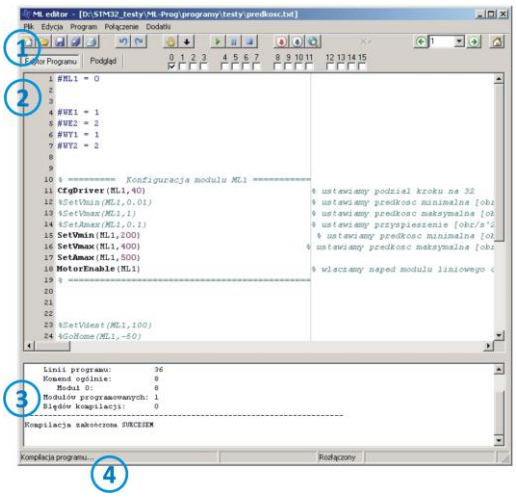
Rys. 1 Komunikacja między sterownikami.

Sterowniki mogą pracować także jako standardowe urządzenia **Modbus Slave** na magistrali RS485. Wówczas całkowita kontrola sterowników (zadawanie pozycji itp.) odbywa się z urządzenia nadrzędnego (master).

Możliwe jest także zaprogramowanie sterownika realizującego program ruchu, który reaguje na wartości zmiennych zapisywanych do rejestrów użytkownika (rejestry modbus).

2. Opis programu ML-PROG

Program ML-PROG pozawala na tworzenie programów ruchu dla 16-stu sterowników jednocześnie i ich równoległe programowanie (przy użyciu magistrali RS485 i konwertera USB-RS485). Program może być także przesłany do pojedynczego sterownika za pomocą aplikacji ***SIC184-KONFIGURATOR*** za pomocą złącza USB.



Rys. 2 Główne okno programu ML-PROG.

1 **Pasek narzędzi i skrótów**
2 **Okno główne**
3 **Okno komunikatów kompilacji i programowania**
4 **Pasek statusu kompilacji i komunikacji**

Pasek narzędzi i skrótów:

1 **Pasek narzędzi:**
Plik – operacje związane z plikiem programu
Edycja – edycja i opcje programu
Program – kompilacja programu, programowanie modułu, zatrzymywanie / uruchamianie programu
Połączenie – połączenie z konwerterem RS485<->USB, skanowanie podłączonych modułów, zmiana adresu modułu, podgląd transmisji
Dodatki – panel podpowiedzi komend

2 **Skróty związane z operacją na pliku i jego edycją**

3 **Skróty związane z kompilacją programu i programowaniem modułu**
- kompilowanie programu
- kompilowanie programu i przesyłanie do sterownika

4 **Skróty związane z uruchamianiem / zatrzymywaniem programu**
- uruchomienie programu w sterowniku
- wstrzymanie programu
- zatrzymanie programu

5 **Skróty związane z połączeniem, oraz skanowaniem podłączonych modułów**
- nawiązanie połączenia z konwerterem RS485<->USB
- zakończenie połączenia
- wyszukiwanie aktywnych modułów
- zmiana adresu modułu

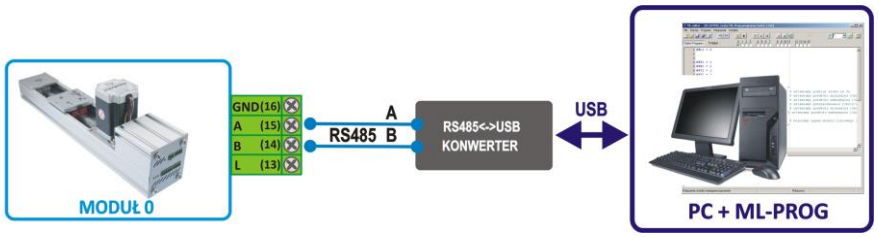
6 **Ręczne sterowanie modułem (dla wersji powyżej 1.16 dostępna tylko opcja bazowania)**

7 **Wybór okna edycji programu / podglądu programów dla poszczególnych modułów**

8 **Wybór adresu modułu (modułów) do programowania / sterowania**

2.1 Nawiązanie połączenia z modułem i zmiana adresu

Sterownik należy połączyć z komputerem PC za pomocą konwertera RS485<->USB wg Rys. 3. Sterownik posiada domyślny adres o wartości 0. Jeśli na linii ma być połączonych kilka modułów które chcemy programować/sterować z ML-ROG jednocześnie, należy najpierw zmienić adresy modułów (każdy podłączając osobno do komputera PC poprzez konwerter RS485<->USB i nadając im inne adresy).

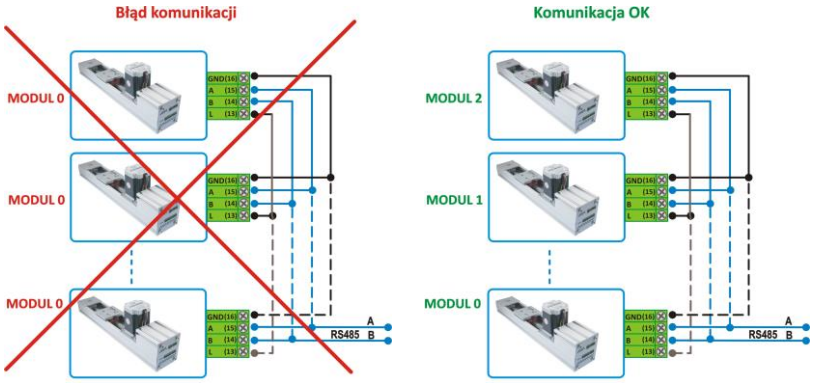


Rys. 3 Połączenie sterownika z komputerem PC

By zmienić adres modułu należy:

- Połączyć program ML-PROG z konwerterem RS485<->USB (***Połączenie*** -> ***Połącz***)
- Zmienić adres modułu domyślnego 0 na inny w zakresie 1..15 (***Połączenie*** -> ***Nadaj adres*** , Rys. 4)

Rys. 4 Okno zmiany adresu modułu.



Rys. 5 Błędne i poprawne adresowanie modułów.

UWAGA!
Każdy moduł posiada domyślny adres o wartości 0. W przypadku programowania kilku modułów jednocześnie, przed ich pierwszym podłączeniem należy zmienić wartość adresu dla każdego modułu z osobna, tak by były one różne.

2.2 Wyszukiwanie podłączonych (aktywnych) modułów

Poprawnie zaadresowane moduły (każdy posiadający inny adres) mogą pracować na jednej magistrali, oraz mogą być wspólnie programowane. By sprawdzić ilość podłączonych modułów i ich adresy należy przejść do ***Połączenie*** - > ***Skanuj moduły*** . Dioda LINK powinna mrugać, sygnalizując aktywną komunikację. Skanowanie modułów trwa około 7 sekund. Po zakończonym procesie skanowania na pasku statusu pokazywana jest informacja o modułach (**Rys. 6**)

Rys. 6 Pasek statusu z widocznymi aktywnymi modułami.

Przykłady odczytywania adresów:
Adres 0 ……. Adres 15
| |
1100000000000000**1** - aktywne moduły o adresach 0,1 i 15)

3. Sterowanie napędem z poziomu języka ML-PROG

Sterowanie napędem z poziomu języka programu ML-PROG polega na zadawaniu odpowiednich komend. Podstawowe komendy sterujące ruchem pozwalają zadać prędkość, pozycję względną (przesunięcie lub ilość obrotów silnika względem aktualnej pozycji), pozycje bezwzględną (zadanie położenia lub ilości obrotów silnika względem punktu bazowania) oraz przyspieszenie, z jakim realizowany jest ruch przy zadawania położenia lub prędkości.

Komendy interpretowane przez ML-PROG umożliwiają ponadto wysterowanie wyjść tranzystorowych, odczytywanie stanów wejść cyfrowych, a także realizację takich funkcji jak oczekiwanie, pętle i skoki. Możliwe jest również wysyłanie komunikatów pomiędzy sterownikami w protokole wewnętrznym lub sterowanie innymi sterownikami lub urządzeniami po magistrali MODBUS-RTU.

3.1 Bazowanie napędu

Po każdym uruchomieniu napędu zalecane jest wykonanie operacj bazowania, by ustalić pozycję początkową (zerową). Wykonuje się to za pomocą funkcji ***GoHome****(adres_modułu, prędkość)*. Na rysunku **Rys. 7** pokazano zakres ruchu modułu, który oznaczony jest literą L. Funkcja bazowania powoduje ruch w stronę napędu, w którym to miejscu znajduje się wbudowany czujnik krańcowy. Dla sterowników SIC184 bazowanie powoduje ruch zawsze w tą samą stronę, aż do osiągnięcia sygnału z wejścia czujnika krańcowego 1 (Home 1).

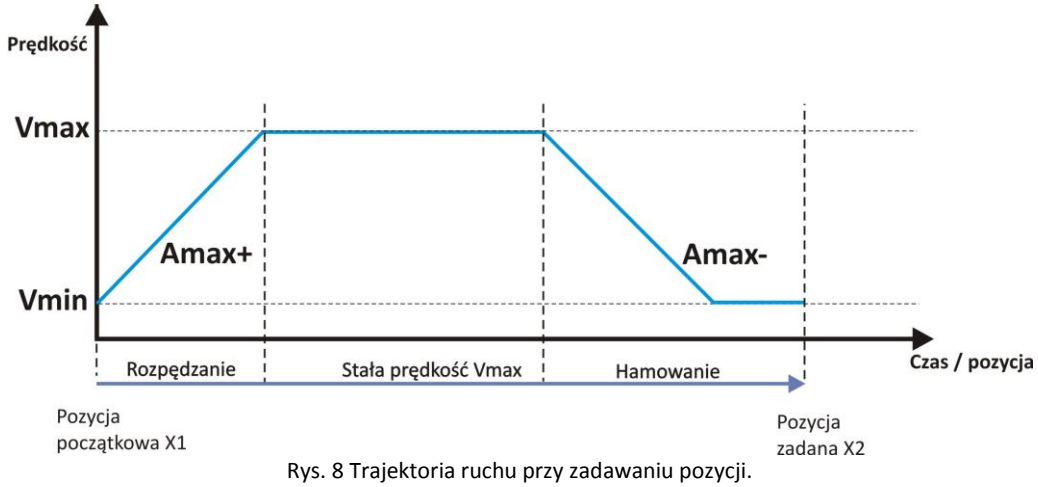
Rys. 7 Punkt bazowania modułu.

UWAGA!
Dla wersji modułu MLA-SI bez wbudowanego enkodera zaleca się wykonywanie bazowania także podczas pracy modułu (np. podczas dojeżdżania w pobliże pozycji bazowania), gdyż w przypadku przypadkowego zatrzymania wózka podczas jego ruchu tracona jest informacja o rzeczywistej pozycji względem punktu początkowego (bazowania).

3.2 Parametry ruchu napędu – prędkość i przyśpieszenie, podział kroku, moc

Przy zadawaniu pozycji należy ustawić parametry ruchu (trajektorii) takie jak prędkość minimalna, maksymalna oraz przyspieszenie. W przeciwnym wypadku sterownik przyjmie parametry domyślne.

- **Prędkość minimalna** (***SetVmin***) – określa minimalną prędkość, z jaką wykonywany będzie ruch. Dotyczy trybów zadawania pozycji (***SetXmove***, ***SetXdest),*** *a także prędkości* ***SetVdest***;
- **Prędkość maksymalna** (***SetVmax***) – określa maksymalna prędkość, jaką może osiągnąć napęd przy wykonywaniu ruchu. Dotyczy tylko trybów zadawania pozycji (***SetXmove***, ***SetXdest***);
- **Przyspieszenie** (***SetAmax***) – określa przyśpieszenie / opóźnienie, z jakim wykonywany będzie ruch. Dotyczy trybów zadawania pozycji (***SetXmove***, ***SetXdest***) oraz prędkości (***SetVdest***).



Rys. 8 Trajektoria ruchu przy zadawaniu pozycji.

Ponadto możliwe jest skonfigurowanie podział kroku sterownika oraz prąd silnika. Realizowane jest to za pomocą następujących komend:

- ***CfgDriver****(adres_modułu, podział_kroku*) – konfiguracja podziału kroku sterownika, możliwe wartości to: 2,8,16,32,64,10,20,40;
- **CfgPower***(adres_modułu,moc_sterownika)* – konfiguracja prądu (mocy) sterownika, możliwe wartości 10…100 (%).

Domyślne parametry napędu :

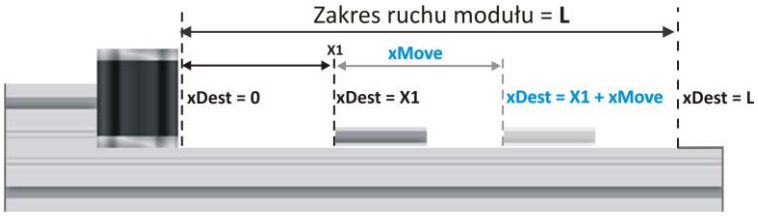
	MLA-SI	**SIC184**
Prędkość minimalna (Vmin)	**10** [mm/s]	**0.05** [obr/s]
Prędkość maksymalna (Vmax)	**50** [mm/s]	**1** [obr/s]
Przyspieszenie (Amax)	**50** [mm/s^2]	**2** [obr/s^2]
Podział kroku (Drive)	**16** [1/16 kroku]	**16** [1/16 kroku]
Moc	**70%**	**70%**

3.3 Zadawanie względnego / bezwzględnego położenia

Na rysunku **Rys. 9** pokazano ideę zadawania bezwzględnej oraz względnej pozycji.
Zadanie pozycji bezwzględnej (***Xdest***) oznacza zawsze ruch względem pozycji bazowej (zerowej). Do zadania pozycji bezwzględnej służy komenda ***SetXdest****(adres_modułu, pozycja_bezwzględna)*, gdzie pozycja bezwzględna dla zbazowanego napędu może być wartością z zakresu 0…X, gdzie X – maksymalna droga jaką może osiągnąć napęd.

Reset pozycji bezwzględnej następuje w przypadkach:
- włączenia zasilania sterownika,
- zatrzymania wykonywania programu programu (podczas pracy bez enkodera),
- wykonania funkcji bazowania ***GoHome****(),*
- *wykrycia sygnału z wejścia Home1,*
- wykonania funkcji ***ResetX()***

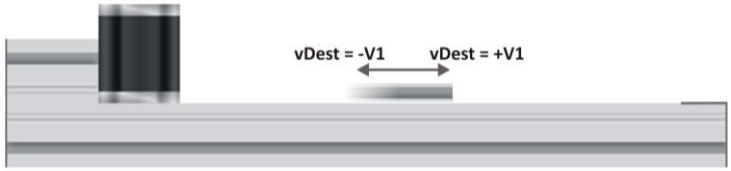
Zadanie pozycji względnej (xMove) oznacza wykonanie ruchu przez napęd o zadaną wartość względem pozycji aktualnej. Do zadania pozycji względnej służy komenda ***SetXmove****(adres_modułu, pozycja_względna).*



Rys. 9 Zadawanie pozycji bezwzględnej (xDest), oraz względnej (xMove) na przykładzie modułu MLA-SI.

3.4 Zadawanie prędkości

Prędkość zadawana jest w jednostkach [**mm/s**] dla modułów MLA-SI lub w [obr/s] dla sterowników SIC184. Do zadawania prędkości służy funkcja ***SetVdest****(adres_modułu, prędkość)*, gdzie prędkość ze znakiem „-" oznacza ruch napędu w przeciwną stronę. Ruch z zadaną prędkością trwa dopóki nie zadana zostanie inna komenda związana z ruchem (zadanie pozycji, hamowanie, bazowanie itp.).



Rys. 10 Zadanie prędkości z kierunkiem ruchu na przykładzie modułu MLA-SI.

3.5 Opis języka i idea programowania w ML-PROG

Programowanie w środowisku ML-PROG polega na zapisaniu komend sterujących w postaci tekstowej. Wszystkie komendy wymagają podania adresu sterownika (liczba z zakresu 0-15), dla którego ma być dana komenda przeznaczona, np.: ***SetXdest****(adres,pozycja)*.

3.5.1 Definicje nazw

Każdą wartość liczbową możemy zastąpić zdefiniowaną wcześniej nazwą poprzedzoną znakiem „**#**".
Przykładowo, by pod wartość 100 przypisać nazwę „***Pozycja_1***" zapisujemy:

```
#Pozycja_1 100
```

Od tego momenty w dowolnym miejscu programu zamiast wartości 100 można użyć nazwy „***Pozycja_1***" np.:

```
SetXdest(ML1, Pozycja_1)
```

3.5.2 Komentarze w programie

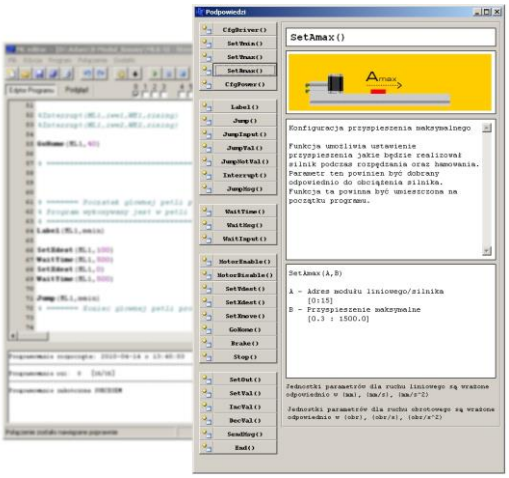
By dodać komentarz do programu należy poprzedzić go znakiem „%", np.:

```
% To jest komentarz
```

3.5.3 Panel komend i podpowiedzi

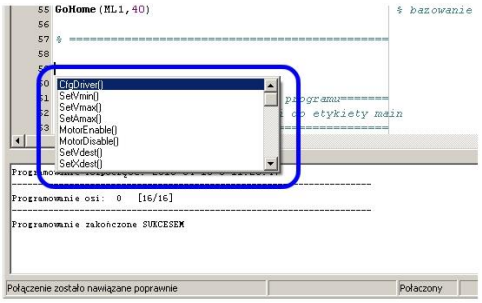
Panel komend pozwala na szybkie dodanie funkcji wraz z parametrami, a także opisuje działanie funkcji. Dostęp do panelu funkcji możliwy jest poprzez przycisk z paska skrótów (***Dodatki -> Panel***

podpowiedzi komend). Wciśnięcie przycisku  w otwartym panelu podpowiedzi pozwala na dodanie do programu danej komendy w miejscu kursora.



Rys. 11 Panel podpowiedzi komend.

Wciśnięcie klawiszy CTRL+SPACJA na klawiaturze w oknie edycji programu powoduje pojawienie się okna z listą dostępnych funkcji i możliwość ich szybkiego dodania do programu.



Rys. 12 Lista funkcji.

3.5.4 Wprowadzanie parametrów ruchu – dostęp do rejestrów użytkownika

Parametry liczbowe wprowadza się jako liczby całkowite lub z przecinkiem np.:

```
SetXdest(0, 120)        % zadanie pozycji całkowitej 120 (obr lub mm)
SetXdest(0, 2.105)      % zadanie pozycji niecałkowitej 2.105 (obr lub mm)
```

3.5.5 Dostęp do rejestrów użytkownika (rejestry modbus)

Sterownik posiada 50 rejestrów dowolnego przeznaczenia (**adresy 50-98**), dostępnych z zewnątrz przez RS485 MODBUS-RTU. Użytkownik może przechowywać w tych rejestrach dowolne wartości, np. parametry ruchu silnika (prędkości, przyśpieszenia, zadane pozycje) lub wartości sterujące pracą programu (ilość cykli, powtórzeń ruchu, zezwolenia na pracę itp.) . Zapisane przez użytkownika rejestry mogą być odczytywane przez program wykonywany w sterowniku, dzięki temu możliwe jest wpływanie na pracę programu w sterowniku.
Rejestry użytkownika mogą być dodatkowo zapamiętanie w pamięci nieulotnej sterownika – wówczas przesłane do nich wartości mogą być dostępne po ponownym włączeniu urządzenia.

W celu odwołania się do rejestru użytkownika w ML-PROG należy poprzedzić go znakiem „**$**” np.:

Ustawianie parametrów ruchu z rejestrów Modbus

Programista ma możliwość odwoływania się do rejestrów Modbus użytkownika z poziomu języka ML-PROG. Podczas tworzenia komend ruchu w programie ML-PROG używając komend takich jak: *SetVmin, SetVmax, SetAmax, SetVdest, SetXdest, SetXmove, GoHome* jako parametr można podać adres rejestru np.:

`SetVdest(0, $50)` - zadanie prędkości przechowywanej w rejestrze 50 (pierwszy rejestr użytkownika)

`SetXmove(0, $98)` - zadanie prędkości przechowywanej w rejestrze 98 (ostatni rejestr użytkownika)

UWAGA

Dla funkcji parametrów ruchu (*SetVmax, SetAmax, SetVdest, SetXdest, SetXmove, GoHome*) adresy rejestrów muszą być podawane zawsze jako parzyste *(np. 50, 62),* gdyż parametry ruchu przechowywane są zawsze w **dwóch sąsiednich rejestrach** jako wartość typu **REAL**. Np. podając adres rejestru 50, program odwoła się do rejestrów 50-51 przechowujących liczbę 4-bajtową.

Ustawianie zmiennych z rejestrów Modbus

Wartości z rejestrów modbus mogą być także wykorzystane do sterowania pracą programu (np. ustalenie ilości powtórzeń pętli, realizacja skoku w zależności od wartości, zwłoka czasowa). Przykładowo:

`WaitTime(0, $50)`- zwłoka czasowa (UWAGA, wartość musi być zapisana do rejestru typu DWORD)

`JumpVal(0,abcd,$60,20)` – skok do etykiety „abcd" gdy w rejestrze 60 będzie wartość 20.

`JumpVal(0,abcd,mem_0,$75)` – skok do etykiety „`abcd`" gdy wartość w pamięci mem_0 będzie równa wartości spod rejestru 75.

`SetVal(0,$60,0)` – zapisanie do rejestru o adresie 60 wartości 20.

`IncVal(0,$60)` – zwiększenie wartości rejestru o 1

`DecVal(0,$60)` – zmniejszenie wartości rejestru o 1

Funkcje ML-PROG	Opis	Typ rejestru Modbus	Przykład
SetVmin, SetVmax, SetAmax, SetVdest, SetXdest, SetXmove, GoHome	Zapis parametrów ruchu	REAL	*SetVdest(0,* ***$50****)*
WaitTime	Zwłoka czasowa	DWORD	*WaitTime(0,* ***$50****)*
JumpVal	Skok, gdy wartość rejestru równa	WORD	*JumpVal*(0,abcd,**$50**,20)
JumpNotVal	Skok, gdy wartość rejestru różna	WORD	*JumpNotVal*(0,abcd,**$50**,20)
SetVal	Ustawienie wartości rejestru	WORD	*SetVal(0,* **$50**)
IncVal	Zwiększenie wartości rejestru	WORD	*IncVal(0,* **$50**)
DecVal	Zmniejszenie wartości rejestru	WORD	*DecVal(0,* **$50**)

3.5.6 Dostęp do wartości z wejścia analogowego

Wartość z wejścia analogowego 0-10V dostępna jest w rejestrze użytkownika $100 lub pod nazwą „ain" np.:

`SetVdest(0,ain)` - zadanie prędkości z wejścia analogowego

`JumpVal(0,abcd,ain,40)` – skok do etykiety abcd, gdy wartość wejścia analogowego będzie równa „40"

3.6 Opis komend (funkcji) języka ML-PROG

3.6.1 Komendy konfiguracji sterownika i parametrów ruchu

Podział kroku ***CfgDriver(A, X)***	**Pozwala na ustawienie podziału kroku dla silnika krokowego. Dostępne wartości to: 2, 5, 8, 10, 16, 20, 32, 40, 64. Zalecane podziały to 8, 10, 16, 20 i 32. Większy podział zmniejsza moment silnika, ale zapewnia płynniejszy ruch.**
Prędkość min/maks. ***SetVmax(A, X)***	**Pozwala ustawić maksymalną prędkość napędu przy osiąganiu pozycji zadanej.** **Jednostki: MLA-SI: [mm/s], SIC184: ustawiane w sterowniku** **Parametry: X – wartość liczbowa (0..9999.9), adres rejestru użytkownika ($50…$98), wartość wejścia analogowego (ain)**
Przyspieszenie ***SetAmax(A, X)***	**Pozwala ustawić przyspieszenie oraz zwalnianie napędu.** **Jednostki: MLA-SI: [mm/s^2], SIC184: ustawiane w sterowniku** **Parametry: X – wartość liczbowa (0...999999), adres rejestru użytkownika ($50…$98), wartość wejścia analogowego (ain)**

3.6.2 Komendy ruchu

Napęd On/Off ***MotorEnable(A)*** ***MotorDisable(A)***	**Włączenie / wyłączenie końcówki mocy sterownika.**
Zadanie prędkości ***SetVdest(A, X)***	**Pozwala zadać prędkość. Dokładny opis w rozdziale 3.4.** **Jednostki: MLA-SI: [mm/s], SIC184: ustawiane w sterowniku** **Parametry: X – wartość liczbowa (-9999….9999), adres rejestru użytkownika ($50…$98), wartość wejścia analogowego (ain)**
Zadanie pozycji ***SetXdest(A, X)*** ***SetXmove(A, X)***	**Pozwala zadać pozycję. Kolejna komenda zostanie wykonana dopiero po osiągnięciu przez napęd zadanej pozycji.** **Jednostki: MLA-SI: [mm], SIC184: ustawiane w sterowniku** **Parametry: X – wartość liczbowa (-999999….999999), adres rejestru użytkownika ($50…$98), wartość wejścia analogowego (ain)**
Bazowanie ***GoHome(A, X)***	**Funkcja bazująca napędu. Funkcja ta powinna być zawsze wykonywana po włączeniu zasilania. Przed rozpoczęciem zadawania komend ruchu. Kolejna komenda zostanie wykonana dopiero po zakończeniu bazowania. Dokładny opis w rozdziale 3.1.** **Jednostki: MLA-SI: [mm/s], SIC184: ustawiane w sterowniku** **Parametry: X – wartość liczbowa (0…9999), adres rejestru użytkownika ($50…$98)**
Zerowanie pozycji ***ResetX(A)***	**Funkcja zerująca aktualną pozycję.**

Hamowanie
Brake(A)
	Funkcja pozwala na płynne wyhamowanie silnika w przypadku, gdy znajdował się w ruchu np.:
	…
	`SetVdest(0, 100) % zadanie prędkości 100 mm/s`
	`WaitTime(0,5000) % opóźnienie 5 sekund`
	`Brake(A)  % płynne zatrzymanie napędu`
	…

Zatrzymanie ***Stop(A)***	**Funkcja nagłego zatrzymania modułu**

3.6.3 Skoki warunkowe i bezwarunkowe

Etykieta ***Label(A, ABCD)***	**Ustawia etykietę o dowolnej nazwie składającej się z 4 liter, do której może być wykonany skok.**

Skok do etykiety
Jump(A, ABCD)
	Wykonuje skok do etykiety o podanej nazwie, np.:
	…
	`Label(0,skok) %ustawienie etykiety o nazwie „skok"`
	…
	`Jump(0,skok) %skok do etykiety o nazwie „skok"`
	…

Skok do etykiety, jeśli wejście
JumpInput(A, ABCD, NrWe, STAN)
	Wykonuje skok do etykiety, gdy na wejściu o numerze NrWe występuje określony stan (on - wejście aktywne, off – wejście nieaktywne) np.:
	…
	`Label(0,skok) %etykieta „skok"`
	…
	…
	`JumpInput (0,skok,1,on ) %skok do etykiety o nazwie „skok", gdy na wejście jeden będzie aktywne.`
	…

Skok do etykiety, jeśli wartość
JumpVal(A, ABCD, ZM, VAL)
	Wykonuje skok do etykiety ABCD, gdy zmienna ZM będzie równa wartości VAL.
	Parametry:
	ZM – numer pamięci (mem_0…mem_9), wartość analogowa (ain) adres rejestru użytkownika ($50…$98)
	VAL – wartość stała (0….999999), adres rejestru użytkownika ($50…$98)
	Przykład:
	…
	`IncVal(0,mem_1) % zwiększenie wartości zmiennej mem_1 o 1.`
	`JumpVal(0,skok, mem_1,10) %skok do etykiety o nazwie „skok", gdy wartość zmiennej mem_1 będzie równa 10.`
	…
	`JumpVal(0,skok, mem_1,$40) %skok do etykiety o nazwie „skok", gdy wartość zmiennej mem_1 będzie równa wartości spod adresu $40.`
	…
	`Label(0,skok) %etykieta „skok"`
	…

Skok do etykiety, jeśli nie wartość
JumpNoVal(A, ABCD, ZM, VAL)
	Wykonuje skok do etykiety, gdy zmienna ZM będzie różna od wartości VAL/
	Parametry:
	ZM – numer pamięci (mem_0…mem_9), wartość analogowa (ain) adres rejestru użytkownika ($50…$98)
	VAL – wartość stała (0….999999), adres rejestru użytkownika ($50…$98)
	Przykład:
	…
	`IncVal(0,mem_1) % zwiększenie wartości zmiennej mem_1 o 1.`
	`JumpNotVal(0,skok, mem_1,10) %skok do etykiety o nazwie „skok", gdy wartość zmiennej mem_1 będzie różna od 10.`
	…
	…
	`Label(0,skok) %etykieta „skok"`
	…

Skok do etykiety, jeśli impuls na wejściu ***Interrupt(A, ABCD, NrWe, STAN)***	**Konfiguracją wejścia do wykonania skok do etykiety, gdy na wejściu o numerze NrWe pojawi się sygnał o określonym zboczu (falling - opadające, rising – rosnące, falling_rising – opadające lub rosnące). Funkcja Interrupt powinna znajdować się na początku (przy inicjalizacji) programu, np.:** ***Interrupt(0, int1, 1, rising) % inicjalizacja wejścia 1 – skok do etykiety „int1", gdy na wejściu 1 pojawi się impuls rosnący*** … … ***Label(0,int1) %etykieta „int1",do której wykonany zostanie skok*** …
Skok do etykiety, jeśli wiadomość ***JumpMsg(A, ABCD, MESG)***	**Wykonuje skok do etykiety, gdy pojawi się wiadomość o określonej nazwie np.:** … ***JumpMsg(0,skok, mes1)*** **%skok do etykiety o nazwie „skok", gdy pojawi się wiadomość „mes1"** … … ***Label(0,skok) %etykieta „skok" do której wykonany zostanie skok*** …

3.6.4 Opóźnienia i oczekiwania na zdarzenie

Czekaj X ms ***WaitTime(A, X)***	**Funkcja wprowadzająca określona zwłokę czasową w ms między wykonywanymi komendami, np.:** … ***SetOut(0,1,on) % włączenie wyjścia 1*** ***WaitTime(0, 2000) % czekaj 2 sekundy*** ***SetOut(0,1,off) % wyłączenie wyjścia 1*** …
Czekaj na wiadomość ***WaitMsg(A, MESG)***	**Funkcja czekająca na wiadomość o określonej nazwie, np.:** … ***WaitMsg(0,mes1) % oczekiwanie na wiadomość „mes1"*** ***SetOut(0,1,on) % wyjście 1 zostanie załączone, gdy przyjdzie wiadomość „mes1"*** …
Czekaj na stan wejścia ***WaitInput(A, NrWe, STAN)***	**Funkcja czeka na określony stan wejścia, np.:** … ***WaitInput(0,1,on) % oczekiwanie na stan wysoki wejścia 1*** ***SetXdest(0,100) % zadanie pozycji*** …

3.6.5 Komendy sterujące przez MODBUS-RTU

Ustaw wyjście (funkcja 5 MODBUS) *WriteMCoil(A, B, C, STAN)*	**Funkcja umożliwia ustawienie (STAN = set) lub wyzerowanie (STAN = reset) bitu w rejestrze C w urządzeniu o adresie Modbus B. W momencie wysyłania komendy sterownik staje się urządzeniem nadrzędnym na magistrali.** … `WriteMCoil(0,1,1,on) % ustawienie wyjścia w rejestrze 1 urządzenia o adresie Modbus 1` …
Czekaj na wiadomość (funkcja 6 MOODBUS) *WriteM1Reg(A, B, C, VAL)*	**Funkcja umożliwia zapisanie wartości (VAL) typu INT w rejestrze C w urządzeniu o adresie Modbus B. W momencie wysyłania komendy sterownik staje się urządzeniem nadrzędnym na magistrali.** … `WriteM1Reg(0,1,1,1234) % zapisanie liczby 1234 do rejestru 1 urządzenia o adresie Modbus 1` …

Czekaj na stan wejścia (funkcja 16 MOODBUS) *WriteM2Reg(A, B, C, VAL)*	**Funkcja umożliwia zapisanie wartości (VAL) typu DIN lub REAL do dwóch rejestrów C i C+1 w urządzeniu o adresie Modbus B.**
	UWAGA: Liczba wysyłana jest jako REAL, gdy zawiera przecinek (np. 5.0), w przeciwnym wypadku wysyłana jest jako liczba całkowita DINT. W momencie wysyłania komendy sterownik staje się urządzeniem nadrzędnym na magistrali
	…
	`WriteM2Reg(0,1,1,12.345) % zapisanie liczby typu REAL 12.345 do rejestrów 1 i 2 urządzenia o adresie Modbus 1`
	…
	…
	`WriteM2Reg(0,1,1,150000) % zapisanie liczby typu DINT 150000 do rejestrów 1 i 2 urządzenia o adresie Modbus 1`
	…

3.6.6 Pozostałe funkcje

SetOut(A, NrWy, STAN)	**Funkcja włącza lub wyłącza określone wyjście sterownika, np.:**
	…
	`SetOut(0,1,on)  % włączenie wyjścia 1`
	`SetOut(0,2,off)  % wyłączenie wyjścia 2`
	…

SetVal(A, ZM, VAL)	**Ustawia określoną wartość zmiennej (mem_1..Mem_9). Zakres wartości 0-999999. Np.:**
	…
	`SetVal(0,mem_1,100)  % ustawienie zmiennej mem_1 na 100`
	`SetVal(0,mem_2,2000)  % ustawienie zmiennej mem_2 na 2000`
	…

IncVal(A, ZM)
DecVal(A, ZM)
	Zwiększenie lub zmniejszenie zmiennej o 1
	Parametry:
	ZM – numer pamięci (mem_0…mem_9), adres rejestru użytkownika ($50…$98)
	Przykład:
	…
	`IncVal(0,mem_1)  % zwiększenie pamięci mem_1 o 1`
	`DecVal(0,$40)  % zmniejszenie zmiennej spod adresu 40 o 1`
	…

SendMsg(A, MESG)	**Funkcja pozwalająca wysłać wiadomość do innych modułów na magistrali RS485, np.:**
	…
	`SendMsg(0,mes1)  % wysłanie wiadomości o nazwie „mes1"`
	`SendMsg(0,mes2)  % wysłanie wiadomości o nazwie „mes2"`
	…

IntEnable(A, NrWe)	**Funkcja pozwala włączyć przerwanie na danym wejściu** … `IntEnable(0,1)  % włączenie przerwania na wejściu 1` …
IntDisable(A, NrWe)	**Funkcja pozwala wyłączyć przerwanie na danym wejściu** … `IntDisable(0,1)  % wyłączenie przerwania na wejściu 1` …

Return(A)	**Funkcja powrotu z przerwania. Powoduje powrót do miejsca programu, w którym nastąpiło przerwanie.**
	…
	`Label(0,int1)  % etykieta przerwania 1`
	…
	… `% program wykonywany w przerwaniu`
	…
	`Return(0)  % powrót do programu przerwanego przez przerwanie`

End(A)	**Zakończenie działania programu.**

4. Przykładowe programy

4.1 Sterowanie dwoma modułami liniowymi MLA-SI

```
% Definicje nazw modułów
#ML1 = 0
#ML2 = 1
% Definicje nazw numerów wejść i wyjść sterownika
#WE1 = 1
#WE2 = 2
#WY1 = 1
#WY2 = 2
% Zdefiniowanie podziału kroku dla sterowników
#STEPS = 32

% Zdefiniowanie nazw prędkości minimalnych, maksymalnych, przyśpieszeń (jednostki mm/s, mm/s^2)
#VMIN = 10
#VMAX = 200
#ACC  = 100

% =======  Konfiguracja modułu ML1 =====
CfgDriver(ML1,STEPS)                    % ustawiamy podzial kroku
SetVmin(ML1,VMIN)                       % ustawiamy predkosc minimalna [mm/s]
SetVmax(ML1,VMAX)                       % ustawiamy predkosc maksymalna [mm/s]
SetAmax(ML1,ACC)                        % ustawiamy przyspieszenie [mm/s^2]
MotorEnable(ML1)                        % wlaczamy naped modulu liniowego o adresie 0

% =========  Konfiguracja modułu  ML2 ===========
CfgDriver(ML2,STEPS)                    % ustawiamy podzial kroku
SetVmin(ML2,VMIN)                       % ustawiamy predkosc minimalna [mm/s]
SetVmax(ML2,VMAX)                       % ustawiamy predkosc maksymalna [mm/s]
SetAmax(ML2,ACC)                        % ustawiamy przyspieszenie [mm/s^2]
MotorEnable(ML2)                        % wlaczamy naped modulu liniowego o adresie 0

% =======Bazowanie modułów =============
GoHome(ML1,1)
GoHome(ML2,1)

% ===== Początek głównej pętli programu ML1 ====
% Program wykonywany jest w pętli do etykiety main
% ====================================
Label(ML1,main)                         % Ustawienie etykiety o nazwie "main"

SetXdest(ML1,120)                       % Zadanie pozycji dla modulu 1  - 300 mm
SetOut(ML1,on)                          % Włączenie wyjścia WY1
SendMsg(ML1,ml2a)                       % Wysłanie wiadomości o nazwie "ml2a"
WaitMsg(ML1,ml1a)                       % Czekanie na wiadomość o nazwie ml1a (wysyła ją moduł ML2)

SetXdest(ML1,0)                         % Zadanie pozycji dla modułu 1 – 0
SetOut(ML1,off)                         % Wyłączenie wyjścia WY1
SendMsg(ML1,ml2b)                       % Wysłanie wiadomości o nazwie "ml2b"
WaitMsg(ML1,ml1b)                       % Czekanie na wiadomość o nazwie ml1b (wysyła ją moduł ML2)


Jump(ML1,main)                          % Skok do etykiety o nazwie "main"
% ===== Koniec głównej pętli programu ML1 ===

% ===== Początek głównej pętli programu ML2 ====
% Program wykonywany jest w pętli do etykiety main
% ====================================
Label(ML2,main)                         % Ustawienie etykiety o nazwie "main"

WaitMsg(ML2,ml2a)                       % Czekanie na wiadomość o nazwie ml2a (wysyła ją moduł ML1)
SetXdest(ML2,120)                       % Zadanie pozycji dla modułu 1  - 300 mm
SetOut(ML2,on)                          % Włączenie wyjścia WY1
SendMsg(ML2,ml1a)                       % Wysłanie wiadomości o nazwie "ml1a"

WaitMsg(ML2,ml2b)                       % Czekanie na wiadomości o nazwie ml2b(wysyła ją moduł ML1)
SetXdest(ML2,0)                         % Zadanie pozycji dla modułu 1 – 0
SetOut(ML2,off)                         % Wyłączenie wyjścia WY1
SendMsg(ML2,ml1b)                       % Wysłanie wiadomości o nazwie "ml1b"

Jump(ML2,main)                          % Skok do etykiety o nazwie "main"
% ===== Koniec głównej pętli programu ML2==

% ======= Przerwanie od We1 modulu ML1 ======
Label(ML1,iwe1)                         % Ustawienie etykiety o nazwie "iwe1"
% …
Jump(ML1,main)                          % Skok do początku programu
% ======= Przerwanie od We1 modulu ML2 ======
Label(ML2,iwe1)                         % Ustawienie etykiety o nazwie "iwe2"
% …
Jump(ML2,main)                          % Skok do początku programu
% ====================================
```

4.2 Sterowanie silnikiem za pomocą sterownika SIC184

```
% Definicje nazw sterowników
#STER1 = 0

% Definicje nazw numerów wejść i wyjść sterownika
#WE1 = 1
#WE2 = 2
#WY1 = 1
#WY2 = 2

% Zdefiniowanie podziału kroku dla sterownika
#STEPS = 32

% Zdefiniowanie nazw prędkości minimalnych, maksymalnych, przyśpieszeń
#VMIN = 0.05                          % 0.05 obr/sek.
#VMAX = 3                             % 3 obr/sek.
#ACC  = 6                             % 6 obr/sek^2.

% =======  Konfiguracja napędu STER1 =====
CfgDriver(STER1,STEPS)                % ustawiamy podzial kroku
SetVmin(STER1,VMIN)                   % ustawiamy predkosc minimalna [obr/s]
SetVmax(STER1,VMAX)                   % ustawiamy predkosc maksymalna [obr /s]
SetAmax(STER1,ACC)                    % ustawiamy przyspieszenie [obr /s^2]
MotorEnable(STER1)                    % włączenie końcówki mocy sterownika (napędu)


% =======Bazowanie napędu =============
GoHome(STER1,1)                       % prędkość bazowania 1 [obr/s]


% ===== Początek głównej pętli programu====
% Program wykonywany jest w pętli do etykiety main
% ========================================
Label(STER1,main)                     % Ustawienie etykiety o nazwie "main"

SetXdest(STER1,50)                    % Zadanie pozycji - 50 [obr]
SetOut(STER1,on)                      % Włączenie wyjścia WY1
WaitTime(STER1,5000)                  % Czekanie 5 sek.
SetXdest(STER1,0)                     % Zadanie pozycji dla modułu 1 - 0
SetOut(STER1,off)                     % Wyłączenie wyjścia WY1
WaitTime(STER1,3000)                  % Czekanie 3 sek.


Jump(STER1,main)                      % Skok do etykiety o nazwie "main"
% ===== Koniec głównej pętli programu   ===


% ======= Przerwanie od We1 ======
Label(STER1,iwe1)                     % Ustawienie etykiety o nazwie "iwe1"
% …
Jump(STER1,main)                      % Skok do początku programu
```

Dokładny opis stosowania poszczególnych funkcji znaleźć można w samouczku składającego się z gotowych programów (dostępny w katalogu *programu ML-PROG*).

5. Dodatek - spis komend interpretowanych przez ML-PROG v1.20

Komenda	Opis	Parametry
Związane z konfiguracją sterownika i parametrami ruchu		
CfgDriver*(A, X)*	Ustawia podział kroku sterownika	X: [2,8,16,32,64,10,20,40]
SetVmin*(A, X)*	Ustawia prędkość minimalną	Zależne od typu sterownika
SetVmax*(A, X)*	Ustawia prędkość maksymalną	Zależne od typu sterownika
SetAmax*(A, X)*	Ustawia przyśpieszenie	Zależne od typu sterownika
CfgPower*(A ,X)*	Ustawia moc sterownika (10..100%)	X: [10..100]
Związane z zadawaniem ruchu modułu		
MotorEnable*(A)*	Włączenie napędu modułu	brak
MotorDisable*(A)*	Wyłączenie napędu modułu	brak
SetVdest*(A, X)*	Zadanie prędkości docelowej	Zależne od typu sterownika
SetXdest*(A, X)*	Zadanie pozycji docelowej (bezwzględnej)	Zależne od typu sterownika
SetXmove*(A, X)*	Zadanie przesunięcia (pozycji względnej)	Zależne od typu sterownika
GoHome*(A, X)*	Funkcja bazowania modułu	Zależne od typu sterownika
ResetX*(A)*	Reset pozycji	brak
Brake*(A)*	Funkcja płynnego hamowania modułu	brak
Stop*(A)*	Funkcja nagłego zatrzymania modułu	brak
Skoki warunkowe i bezwarunkowe		
Label*(A, ABCD)*	Utworzenie etykiety, do której może być wykonany skok	**ABCD** - nazwa etykiety (maks. 4 znaki)
Jump*(A, ABCD)*	Skok bezwarunkowy do etykiety o nazwie **ABCD**	**ABCD** - nazwa etykiety (maks. 4 znaki)
JumpInput*(A, ABCD, NrWe, STAN)*	Skok do etykiety **ABCD**, gdy wejście o numerze **NrWe** przyjmie stan **STAN**	**ABCD** – nazwa etykiety (maks. 4 znaki) **NrWe** – 1 lub 2 **STAN** – On lub Off
JumpVal*(A, ABCD, ZM, VAL)*	Skok do etykiety **ABCD**, gdy zmienna w pamięci **ZM** będzie równa wartości zadanej **VAL**.	**ABCD** - nazwa etykiety (maks. 4 znaki) **ZM** - zmienna w pamięci [mem_1... mem_9] **VAL** - wartość do porównania [0 : 999999]
JumpNoVal*(A, ABCD, ZM, VAL)*	Skok do etykiety **ABCD**, gdy zmienna w pamięci **ZM** będzie różna od wartości zadanej **VAL**.	**ABCD** - nazwa etykiety (maks. 4 znaki) **ZM** - zmienna w pamięci [mem_1... mem_9] **VAL** - wartość do porównania [0 : 999999]
Interrupt*(A, ABCD, NrWe, STAN)*	Skok do etykiety **ABCD** w przypadku, gdy na wejściu o numerze **NrWe** pojawi się stan **STAN**. Skok ten jest wykonywany niezależnie od realizowanego programu. Funkcja ta powinna być umieszczona na początku programu.	**ABCD** – nazwa etykiety (maks. 4 znaki) **NrWe** – 1 lub 2 **STAN** – tryb zbocza: **falling** - opadające, **rising** – rosnące, lub **falling_rising** – rosnące lub opadające
JumpMsg*(A, ABCD, MESG)*	Skok do etykiety **ABCD** w przypadku, gdy został wcześniej wysłany komunikat **MESG** przez inny moduł.	**ABCD** - nazwa etykiety (maks. 4 znaki) **MESG -** - nazwa wiadomości (maks. 4 znaki)
Opóźnienia		
WaitTime*(A, X)*	Funkcja oczekiwania w [ms]	**X** – wartość opóźnienia w [ms]
WaitMsg*(A, MESG)*	Oczekiwanie na wiadomość ABCD od innego modułu	**MESG -** - nazwa wiadomości (maks. 4 znaki)
WaitInput*(A, NrWe, STAN)*	Oczekiwanie na określony stan **STAN** wejścia o numerze **NrWe**	**NrWe** – 1 lub 2 **STAN** – On lub Off
Sterowanie przez MODBUS-RTU		
WriteMCoil*(A, B, C, STAN)*	Ustawienie wyjścia (funkcja 5 Modbus)	B – adres modbus urządzenia slave C – numer rejestru STAN – stan wyjścia (**set** lub **reset**)
WriteM1Reg*(A, B, C, VAL)*	Zapisanie 1 rejestru (liczby INT) (funkcja 6 Modbus)	B – adres modbus urządzenia slave C – numer rejestru VAL – wartość typu **INT** do zapisania
WriteM2Reg*(A, B, C, VAL)*	Zapisanie 2 rejestrów (liczby DINT lub REAL) (funkcja 16 Modbus)	B – adres modbus urządzenia slave C – numer rejestru VAL – wartość typu D**INT** lub REAL do zapisania
Pozostałe		
SetOut(A, NrWy, STAN)	Ustawienie wyjścia OC o numerze **NrWy** w stan **STAN**	**NrWy** – 1 lub 2 **STAN** – On lub Off
SetVal(A, ZM, VAL)	Przypisanie do zmiennej **ZM** wartości **VAL**	**ZM** - zmienna w pamięci [mem_1... mem_9] **VAL** – nowa wartość zmiennej [0 : 999999]
IncVal(A, ZM)	Zwiększenie wartości zmiennej **ZM** o 1	**ZM** - zmienna w pamięci [mem_1... mem_9]
DecVal(A, ZM)	Zmniejszenie wartości zmiennej **ZM** o 1	**ZM** - zmienna w pamięci [mem_1... mem_9]
SendMsg(A, MESG)	Wysłanie wiadomości MESG na magistrali RS485 do innych modułów.	**MESG -** - nazwa wiadomości (maks. 4 znaki)
IntEnable(A, NrWe)	Włączenie przerwania na wejściu **NrWe**.	**NrWy** – 1 lub 2
IntDisable(A, NrWe)	Wyłączenie przerwania na wejściu **NrWe**.	**NrWy** – 1 lub 2
Return(A)	Powrót z przerwania do programu	brak
End(A)	Funkcja umożliwiająca zakończenie programu	brak

,gdzie A – adres sterownika (modułu): [0…15].

PPH WObit E.J.K. SC
62-045 Pniewy, Dęborzyce 16
tel.(061) 22 27 410, fax.(061) 22 27 439
e-mail: wobit@wobit.com.pl www.wobit.com.pl