

Instrukcja programowania EMM

wersja 5.1.2 – zmieniona i uzupełniona (26.02.2008 r.) - do użytku wewnętrznego

dr inż. Maciej Horczyczak
dr inż. Radosław Morek (redaktor)
mgr inż. Tomasz Staniszewski

Spis treści

1.	Bezpieczeństwo	2
2.	Programowanie	2
2.1	Pierwszy program – wiadomości podstawowe	2
2.2	Uruchamianie programu	3
2.2.1	EMM	3
2.2.2	Stanowisko studenckie	3
2.3	Podprogramy	4
3.	Instrukcje	5
3.1	Instrukcje obliczeniowe	5
3.1.1	Zmienne	5
3.1.2	Operatory i wyrażenia arytmetyczne	6
3.2	Instrukcje sterujące	7
3.2.1	Instrukcje skoków	7
3.2.2	Etykiety	8
3.2.3	Instrukcje wywołania i powrotu z procedury	9
3.2.4	Instrukcja pętli	9
3.3	Instrukcje wejścia/wyjścia konsolowego	11
3.3.1	Procedura pauza	12
3.4	Instrukcja freemem	12
3.5	Instrukcja spawn	12
4.	Stanowisko	12
4.1	System transportowy	13
4.1.1	Zderzaki	13
4.1.2	Podniesienie stacji	13
4.1.3	Kodowanie	14
4.1.4	Odczyt kodu	15
4.1.5	Przemieszczanie palet	16
4.2	Manipulator	17
4.2.1	Ruch	17
4.2.2	Manipulowanie części	18
4.2.3	Chwytaaki	19
4.2.4	Współrzędne na paletach	21
4.2.5	Współrzędne manipulatora	21
4.3	Odczyt stanów czujników stanowiska	25
4.4	Opis portów wyjściowych	27
5.	Uwagi	27
6.	FAQ	28

1. Bezpieczeństwo - BHP

Przed przystąpieniem do zajęć student musi bezwzględnie zapoznać się z instrukcją opisującą stanowisko EMM. Bez odpowiedniej wiedzy i znajomości zagadnienia nie zostanie dopuszczony do wykonywania ćwiczenia.

Proces włączania i uruchamiania stanowiska przeprowadza jedynie prowadzący! Stanowisko gotowe do zajęć ma uruchomiony komputer w którym pisze się i uruchamia programy. W przypadku niejasności, nie należy uruchamiać programu na stanowisku tylko sprawdzić informację w dokumentacji. W razie braku informacji w materiałach należy spytać się prowadzącego. Klawisz **[ESC]**^{*} na klawiaturze PC zatrzymuje wykonywanie programu.

Nie należy przekraczać ustawionych osłon modułu gdy znajduje się on w ruchu. Nie można wkładać rąk i innych części ciała oraz innych przedmiotów (nie będącymi elementami wyposażenia) w zasięg pracy robota i części ruchomych stanowiska. W razie wystąpienia niebezpieczeństwa zdrowia, kolizji lub nieprawidłowej pracy stanowiska należy nacisnąć dowolny awaryjny wyłącznik w postaci **czerwonego przycisku "grzyba"**, które są umieszczone w skrajnych częściach oraz pośrodku stanowiska. O zaistniałej sytuacji należy natychmiast powiadomić prowadzącego ćwiczenie.

Najbliższa pomoc medyczna to Przychodnia w budynku NT przy ulicy Narbutta 85, zaś apteczka pierwszej pomocy znajduje się na hali warsztatu

2. Programowanie

Programy sterujące pracą modułu montażowego pisać można w dowolnym edytorze tekstu ASCII. Zapis programu do wykonania na stanowisku musi odbywać się w postaci pliku tekstowego. Należy stosować nazwy plików tekstowych składające się do ośmiu znaków, bez polskich liter, zakończone rozszerzeniem *.prg

2.1 Pierwszy program – wiadomości podstawowe

Pierwszy program, nazwany jako witaj.prg to pierwsze kroki. W edytorze wpisano 3 poniższe linie:

```
# program wyświetlający napis Witaj
write "Witaj"
newl
stop
```

które tworzą pełny program. Objaśnienie, w kolejności linii (wierszy) programu:

- | | |
|----|---|
| 1: | # - znak ten oznacza, że dana linia jest jedynie komentarzem, który nie ma wpływu na działanie programu. Komentarz obejmuje całą jedną linię. Nie można dodawać komentarzy do linii zawierających polecenia. Jak widać powyżej nie stosuje się polskich liter w kodzie źródłowym programu. |
| 2: | instrukcja write wypisuje na ekran wartość wyrażenia lub stałą tekstową, patrz punkt 3.3 |
| 3: | przeniesienie wyświetlania do nowego wiersza |
| 4: | instrukcja stop powoduje natychmiastowe zakończenie wykonywania programu i jej wpisanie w ostatniej linii programu jest konieczne i niezbędne do prawidłowego działania programu !!!
W każdym programie powinna być przynajmniej jedna instrukcja stop . |

Program składa się z linii tekstu zawierających dyrektywy i polecenia języka. W jednej linii może znajdować się tylko jedno polecenie, dyrektywa, komentarz lub etykieta. Linia może mieć do 127 znaków długości. Separatory (znaki spacji i tabulacji) mogą występować wewnątrz linii, na jej początku i na końcu. W

* Klawisze klawiatury komputerowej są oznaczane poprzez ujęcie ich nazwy w nawiasy kwadratowe.

przykładach nie stosowano innych znaków (np. podkreślenia) zamiast spacji. Poszczególne elementy (stałe, identyfikatory, słowa kluczowe, operatory, etykiety) muszą być oddzielane separatorami. Nie istnieje mechanizm pozwalający na kontynuację jednej linii programu w następnej linii pliku - linia programu stanowi całość w ramach linii pliku. Wielkość liter jest znacząca; tzn. identyfikatory 'Aaa' i 'aaa' są różne.

Dobrze:	Źle:
write "Koniec"	write"Koniec"
let ld01 4	let ld01_4
while ld02 :etykieta_3	while ld02:etykieta 3

gdzie:

1:	instrukcja ma na celu wypisanie na ekranie słowa Koniec, instrukcja write i "koniec" są rozdzielone spacją	brak rozdzielenia jest powodem błędu i nie osiągnięcia celu
2:	instrukcja let przypisuje zmiennej ld01 wartość 4; w ten sposób wprowadza się w programie nową zmienną, nadając jej jednocześnie jakąś wartość,	instrukcja zamiast definiować zmienną i jej wartość co jest w przypadku tego języka programowania nierozdzielnie wprowadza zmienną ld01_4 bez wartości,
3:	dopóki wartość zmiennej ld02 jest różna od zera wykonywane są polecenia zawarte między tym wierszem a linią zawierającą wyrażenie :etykieta_3 (patrz 3.2.4),	między nazwą zmiennej ld02 i :etykieta brak spacji powoduje, że interpreter rozumie ld02:etykieta (bez 3) jako nazwę zmiennej; oddzielona cyfra 3 powoduje, że cała linia jest niezrozumiała dla interpretera (błąd programu)

Objaśnienia:

- **polecenie** – inaczej instrukcja,
- **dyrektywa** – zapis taki sam jako instrukcji, lecz na etapie kompilowania programu wykonywana jest podczas wczytywania programu,
- **komentarz** – wiersz lub wiersze ułatwiające programiście orientację w działaniach programu i celach poszczególnych jego części,
- **etykieta** – oznaczeniem miejsca w programie, np. do którego ma nastąpić przekierowanie.

2.2 Uruchamianie programu

2.2.1 EMM

Praca użytkownika EMM polega na obsłudze komputera PC z nakładką ekranową Norton Commander(R) pracującego w systemie DOS - zalecana znajomość instrukcji obsługi. Proces włączania stanowiska EMM **przeprowadza jedynie prowadzący!** Stanowisko gotowe do zajęć ma uruchomiony komputer w którym pisze się i uruchamia programy. Zapis plików na dysku tylko w przeznaczonym do tego miejscu **C:\student** lub w folderze wskazanym przez prowadzącego zajęcia.

W celu wykonania programu **witaj.prg** po akceptacji prowadzącego należy:

1. najechać kursorem na plik w nakładce ekranowej Norton Commander(R) i nacisnąć **[Enter]**.
2. zatwierdzić ponownym naciśnięciem klawisza **[Enter]** domyślną opcję **transpo** lub wybrać inną z podręcznego menu. Opcja **transpo** – kompilacja bez obsługi sterowania robotem i długotrwałej procedury jego bazowania. Opcja interpretera z bazowaniem - **robster**.

2.2.2 Stanowiska studenckie

W trakcie realizacji ćwiczenia zespoły lub indywidualnie studenci pracują na stanowiskach PC1 – PC4 połączonych przez sieć komputerową z EMM. Dostęp do komputera modułu poprzez odpowiedni katalog na zmapowanym dysku G: z np. Komputer PC1 powinien pracować w katalogu 1 itp. Użytkowanie innych katalogów niż przyznane jest zabronione. W przypadku pracy na stanowisku studenckim, a także w czasie

pracy na EMM bez nakładki, należy uruchomić program wpisując nazwę interpretera oraz nazwę programu jako parametr:

```
transpo.exe witaj.prg [enter]
```

W przypadku gdy program nie znajduje się w tym samym katalogu co interpreter należy nazwę programu poprzedzić ścieżką dostępu np.

```
transpo.exe c:\student\program\witaj.prg [enter]
```

W systemie Windows(R) ikonę pliku tekstowego *.prg można przesunąć nad ikonę programu transpo w oknie Eksploratora. Możliwe jest również uruchomienie interpretera poleceń a następnie podanie nazwy programu lub ścieżki i nazwy programu. Konieczne jest tak jak i w poprzednim przypadku podanie pełnej nazwy z rozszerzeniem programu w postaci *.prg

Realizację programu można w dowolnej chwili przerwać naciskając klawisz [ESC]. Wykonanie programu rozpoczyna się od wykonania treści pierwszej linii programu. O ile polecenia nie wskazują inaczej linie programu wykonywane są kolejno według kolejności ich występowania w pliku / plikach programu. Program zapisany jest w pliku / plikach ASCII.

2.3 Podprogramy

W przypadku korzystania z podprogramu używa się jedynej występującej w języku dyrektywy include. Składnia jej jest następująca:

```
include nazwa_pliku
```

gdzie nazwa_pliku jest poprawna nazwa pliku w systemie MS-DOS, np.

```
include plik.prg
include c:\prg\lib\uk_trans.prg
```

Dyrektywa powoduje dołączenie w miejscu jej wystąpienia do bieżącego pliku programu zawartości pliku podanego jako argument dyrektywy. Dyrektywy include mogą być zagnieżdżone i mogą występować w plikach wielokrotnie. Poniżej zamieszczono przykład programu wykorzystującego podprogram:

```
# program wykorzystujący podprogram witaj.prg
include witaj.prg
newl
write "Koniec programu glownego"
wait 12
stop
```

gdzie:

- | | |
|----|---|
| 1: | komentarz |
| 2: | wywołanie programu witaj.prg jako podprogramu, dyrektywa include powoduje, że podczas uruchamiania programu kod źródłowy podprogramu jest "wstawiany" wewnątrz kodu źródłowego programu, w miejscu, w którym jest wstawiona dyrektywa include ; dyrektywy niezależnie od lokalizacji w kodzie źródłowym są wykonywane na etapie uruchamiania programu, |
| 3: | efektem instrukcji newl jest przejście do nowej linii wyświetlanych na ekranie danych, |
| 4: | instrukcja wypisania na ekranie |
| 5: | czekaj określony czas, gdzie 1=0,25 sek. |
| 6: | zakończenie programu |

Uwagi:

Po uruchomieniu powyższego programu **nie pokaże** się na ekranie napis: **Koniec programu głównego**. Wynika to z faktu, że w programie **witaj.prg** występuje instrukcja **stop**. W podprogramach nie stosuje się instrukcji **stop** w znaczeniu końca programu, lecz może ona występować np. przy instrukcjach sterujących. Należy je jednak używać gdy wymaga tego algorytm działania programu. Instrukcja **stop** występuje najczęściej w programie głównym lub samodzielnych programach. Po poprawieniu programu **witaj.prg**, polegającym na usunięciu ostatniej linii zawierającej instrukcję **stop** i ponownym uruchomieniu całość programu zadziała. Brak instrukcji **stop** w programie głównym lub samodzielnych programach spowoduje wyświetlenie błędu po wykonaniu wszystkich instrukcji w postaci nazwy pliku z ostatnim ostatniej linijki programu a także numerem linii. Instrukcja **newl** (skrót od **NEWLine**) powoduje przejście do nowej linii ekranu. Brak instrukcji kasującej cały ekran konsoli i ustawiającej kursor w jego lewym górnym rogu. Instrukcja **wait** powoduje zawieszenie wykonywania programu (odczekanie) liczby ćwiartek sekund wskazanej przez wartość wyrażenia podaną po spacji. Składnia jest następująca:

wait wyrażenie

Jeśli liczba ta jest ujemna, instrukcja **wait** odczekuje 0 sekund. W naszym przypadku liczba 12/4 daje 3 sekundy oczekiwania.

3. Instrukcje

W języku występują cztery rodzaje instrukcji:

- instrukcje podstawienia wartości pod zmienną (występuje przy obliczeniach): **let**
- instrukcje sterujące: **goto**, **goifplus**, **goifzero**, **call**, **return**, **stop**, **while**, **break**, **continue**,
- instrukcje wejścia / wyjścia: **write**, **newl**, **read**, **wait**, **beep**.

3.1 Instrukcje obliczeniowe

Instrukcja obliczeniowa **let** ma następującą składnię:

let zmienna wyrażenie

gdzie:

zmienna jest nazwą zmiennej (definiowanej i ewentualnie - przy pierwszym wystąpieniu nazwy zmiennej - deklarowanej), natomiast **wyrażenie** jest dowolnym wyrażeniem arytmetycznym. Przykłady podano poniżej.

3.1.1 Zmienne

W programie możliwe jest używanie zmiennych. Są to wyłącznie globalne zmienne proste typu rzeczywistego. Identyfikatorem (nazwą) zmiennej jest dowolny poprawny identyfikator. Zmienne są deklarowane (i jednocześnie definiowane) przez ich pierwsze wystąpienie jako 1 argumentu polecenia **let**. Zmienne istnieją aż do wykonania polecenia **stop**. Odwołanie się do nieistniejącej (niezdefiniowanej) zmiennej stanowi błąd. Przykład definiowania (deklarowania) zmiennej:

```
# deklaracja zmiennej a (jednoczesne nadanie wartosci 5)
let a 5
```

gdzie:

1: komentarz

2: deklaracja zmiennej o nazwie **a**, o zadanej wartości równej 5

Dopuszcza się przeciążenie identyfikatora jako jednocześnie nazwy zmiennej i nazwy etykiety, np.:

`goifplus ident :ident`

3.1.2 Operatory i wyrażenia arytmetyczne

W języku zaimplementowano poniższe operatory arytmetyczne, stosując zapis wg tzw. Zasady odwrotnej notacji polskiej (ONP).

Operator	Liczba argumentów	Znaczenie:	Przykład zapis klasyczny:	Przykład wg ONP:
*	2	mnożenie	$a*b$	<code>a b *</code>
/	2	dzielenie	a/b	<code>a b /</code>
+	2	dodawanie	$a+1$	<code>a 1 +</code>
-	2	odejmowanie	$a-3$	<code>a 3 -</code>
log	1	logarytm naturalny	$\log 5$	<code>5 log</code>
power	2	potęgowanie x^y	2^3	<code>2 3 power</code>
sin	1	sinus	$\sin 0.7$	<code>0.7 sin</code>
arctg	1	arc tangens	$\arctg 0.7$	<code>0.7 arctg</code>

Wszystkie operatory przyjmują argumenty rzeczywiste i zwracają wartości rzeczywiste. Wszystkie kąty wyrażane są w radianach. Brak jest operatorów unarnych (jednoargumentowych, np. $-a$) i operatorów przetwarzających napisy.

Wyrażeniem arytmetycznym jest sekwencja identyfikatorów zmiennych liczbowych, operatorów i stałych liczbowych oddzielonych separatorami. W wyrażeniach nie stosuje się nawiasów oraz znaku równości.

Przykład 3.1

```
# deklaracja zmiennej wynik (jednoczesne nadanie wartosci 2+3)
let wynik 2 3 +
write "wynik = "
write wynik
```

gdzie:

1: komentarz

2: deklaracja zmiennej o nazwie **wynik**, której wartość wynosi $2 + 3$

3: instrukcja wypisania na ekranie słowa i znaku “wynik = ”

4: instrukcja wypisania na ekranie wartości zmiennej **wynik**, a ponieważ brak jest instrukcji newl wartość ta jest wypisana w tej samej linii co słowo i znak “wynik = ”

Przykład 3.2

```
# deklaracja zmiennej a (jednoczesne nadanie wartosci 2)
let a 2
write "potega "
write a
write " = "
write a 2 power
```

gdzie:

1:	komentarz
2:	deklaracja zmiennej o nazwie a , której wartość wynosi 2
3:	instrukcja wypisania na ekranie słowa i spacji “potega ”
4:	instrukcja wypisania na ekranie wartości zmiennej a , a ponieważ brak jest instrukcji newl wartość ta jest wyświetlona na ekranie w tym samym wierszu co słowo “potega ”
5:	instrukcja wypisania na ekranie spacji i znaków “ = “ i podobnie jak powyżej brak instrukcji newl powoduje wyświetlanie w tym samym wierszu
6:	obliczenie potęgi o wykładniku równym 2 zmiennej a i wypisanie wyniku obliczenia na ekranie w tym samym wierszu, co dotychczas

3.2 Instrukcje sterujące

```
goto
goifplus
goifzero
call
return
stop
while
break
continue
```

3.2.1 Instrukcje skoków

Instrukcja **goto** powoduje zmianę kolejności wykonywania poleceń programu - skok do linii oznaczonej podaną etykietą. Składnia instrukcji jest następująca:

```
goto :etykieta
```

przy czym etykieta **:etykieta** musi istnieć w programie np.:

```
:poczek
write "Witaj"
goto :poczek
```

Uwaga:

Powyższy fragment kodu źródłowego program będzie wyświetlał napis w sposób ciągle – jeden za drugim aż do końca linijki, a następnie nastąpi przejście do nowego wiersza itd. Aby zakończyć należy nacisnąć klawisz [Esc].

Instrukcja **goifplus** powoduje skok do linii zawierającej podana etykietę, jeśli wartość podanej zmiennej jest dodatnia. Składnia instrukcji **goifplus** jest następująca:

goifplus *zmienna* :*etykieta*

przy czym etykieta :*etykieta* musi istnieć w programie np.:

```
let a 2
goifplus a :dodatnie
write "a niedodatnie – bez skoku"
stop
:dodatnie
write "a dodatnie – dobrze"
stop
```

Uwaga:

Tylko przy niedodatnich wartościach deklarowanych zmiennej **a** pojawi się informacja o nie wykonaniu skoku do etykiety i program się zakończy.

Instrukcja **goifzero** powoduje skok do linii zawierającej podana etykietę, jeśli wartość podanej zmiennej jest równa 0.0 Składnia instrukcji **goifzero** jest następująca:

goifzero *zmienna* :*etykieta*

przy czym etykieta :*etykieta* musi istnieć w programie np.:

```
# deklaracja zmiennej ld14 (jednoczesne nadanie wartosci 3, innej niż 0 i 1)
let ld14 3
# odczyt do zmiennej ld14 stanu portu wejsciowego nr 2 (wartosc 0 lub 1)
get ld14 2
goifzero ld14 :brak_chwybaka
goto :dalej
:brak_chwybaka
write "brak chwytaka na wieszaku"
stop
:dalej
write "chwytak na wieszaku"
```

Uwaga:

W przypadku braku chwytaka stan portu nr 2 (ld14) wynosi 0 i następuje skok do etykiety :brak_chwybaka ,
wyświetlenie komunikatu i zakończenie programu.

3.2.2 Etykiety

Etykiety są to identyfikatory poprzedzone znakiem “:” (dwukropek). Służą do oznaczania określonych miejsc w kodzie źródłowym programu. Etykieta nie może zawierać separatora. Etykieta może wystąpić w programie w dwóch postaciach:

- definiującej: jako określająca miejsce w programie np. :
:tu_skoczyc
- wskazującej: jako wskazanie na określone miejsce w programie np. :
goto :tu_skoczyc

Postać definiująca etykiety może wystąpić tylko jeden raz w całym programie. Patrz też informacje i przykłady w instrukcjach skoków (3.2.1).

3.2.3 Instrukcje wywołania i powrotu z procedury

Instrukcja **call** służy do skoku do wskazanej etykieta linii z zapamiętaniem miejsca powrotu. Instrukcja **return** powoduje powrót do zapamiętanego miejsca. Obie instrukcje implementują skok ze śladem pozwalający na tworzenie procedur (podprogramów). Procedury mogą być zagnieżdżone. W jednej procedurze może być wiele instrukcji **return**. Procedura wewnątrz kodu źródłowego programu lub podprogram w postaci wydzielonego i zapisanego w oddzielnym pliku kodu źródłowego służy do wielokrotnego wykorzystywania. Takie rozwiązania pozwalają na uporządkowanie określonych działań programów i nie powtarzanie wielokrotnie tego samego ciągu instrukcji w kodzie źródłowym. Jeżeli program daną czynność ma wykonywać wielokrotnie to należy dążyć do opracowania procedury lub podprogramu. Ma to także znaczenie w przypadku wprowadzania zmian, które nanosi się tylko w jednym miejscu, a nie w wielu. Składnia powyższych instrukcji jest następująca:

```
call :etykieta
...
return
```

Z wywołaniem procedury (tzn. użyciem polecenia **call**) nie wiążą się żadne mechanizmy przekazywania parametrów i tworzenia zmiennych lokalnych. Parametry można przekazywać przez tzw. zmienną globalną. Instrukcja **stop** powoduje natychmiastowe zakończenie wykonywania programu. Patrz punkt 2.1 i 2.3

3.2.4 Instrukcja pętli

Instrukcje **while**, **break** i **continue** pozwalają na organizację pętli ze sprawdzaniem warunku na początku. Składnia tych instrukcji jest następująca:

```
while zmienna :etykieta
...
#zakres pętli – między instrukcja while a :etykieta
...
:etykieta
break
continue
```

Wykonanie instrukcji **while** polega na:

- | | |
|----|---|
| | sprawdzeniu czy wartość <i>zmiennej</i> sterującej zmienna jest różna od 0.0, |
| 1: | <ul style="list-style-type: none">jeśli tak, wykonywany jest zakres pętli aż do napotkania etykiety :etykieta, a następnie wraca się ponownie do wiersza 1 kodu programu,jeśli nie, wykonanie pętli jest kończone i sterowanie przekazywane jest do linii następnej po linii zawierającej etykietę :etykieta |
| 2: | początek zakresu pętli |
| 3: | komentarz |
| 4: | koniec zakresu pętli |
| 5: | etykieta :etykieta |
| 6: | instrukcja break powoduje natychmiastowe przerwanie wykonywania zakresu pętli, |
| 7: | instrukcja continue polega na natychmiastowym porzuceniu wykonywania zakresu pętli i przejściu do kolejnego sprawdzenia. |

Pętle mogą być wzajemnie zagnieżdżone. Napotkane polecenia break i continue rozumiane są jako odnoszące się do aktualnie wykonywanej (najmniej zagnieżdżonej) pętli.

Przykład 3.3

```
# podprogram ruchu palety w lewo do stacji lewej
write "Ruch w lewo"
set 13
write "Zderzak lewej stacji aktywny"
set 4
# definicja zmiennej petli (najczesciej wartosc 1)
let stacja 1
# definicja zmiennej Id03 (nadanie wartosci 2, rozna od odczytanej 1 lub 0)
let Id03 2
# definicja petli (zmienna, etykieta)
while stacja :na_stacji
# odczyt portu 23 do zmiennej Id03
get Id03 23
# obliczenie zmiennej petli (stacja = stacja - Id03)
let stacja stacja Id03 -
# definicja etykiety wyjścia z petli
:na_stacji
clear 13
write " obroty stop"
newl
write "Paleta na stacji roboczej w gore"
set 0
wait 1
newl
write "Paleta podniesiona"
```

Przykład 3.4.a i b

z użyciem break	z użyciem continue
<pre># demo34a - polecenie break let n 5 write "n=" write n newl while n :koniec let n n 1 - break write "n=" write n newl :koniec write "n=" write n newl # koniec</pre>	<pre># demo34b - polecenie continue let n 5 write "n=" write n newl while n :koniec let n n 1 - continue write "n=" write n newl :koniec write "n=" write n newl # koniec</pre>
<pre>programik wypisuje: n=5 n=4</pre>	<pre>programik wypisuje: n=5 n=0</pre>

3.3 Instrukcje wejścia / wyjścia konsolowego

Instrukcjami wejścia (klawiatura) / wyjścia (ekran, głośnik systemowy) konsolowego są instrukcje o następującej składni:

```
write wyrażenie
write "stała tekstowa"
newl
wait wyrażenie
beep wyrażenie
read zmienna
file
```

- | | |
|----|---|
| 1: | instrukcja write wysyła na konsolę wartość wyrażenia lub stałą tekstową (tekst), |
| 2: | |
| 3: | instrukcja newl wysyła na konsolę znak przejścia do nowej linii, |
| 4: | instrukcja wait odczekuje podaną jako <i>wyrażenie</i> liczbę ćwiartek sekundy, |
| 5: | instrukcja beep generuje dźwięk trwający podaną jako <i>wyrażenie</i> liczbę ćwiartek sekundy, |
| 6: | instrukcja read wczytuje z klawiatury liczbę i jej wartość przypisuje zmiennej <i>zmienna</i> , która musi być wcześniej zadeklarowana; gdy odczyt nie odbył się poprawnie, następuje błąd i przerwanie pracy programu |
| 7: | instrukcja file |

Przykład 3.5

```
#program potega
:start
# definicja etykiety
newl
# przejście do nowej linii
write "Podaj liczbę podnoszona do potegi: "
let liczba 0
# definicja zmiennej z nadaniem wartosci 0
read liczba
#odczyt zmiennej liczba
let potega 0
# definicja zmiennej z nadaniem wartosci 0
let potega liczba 2 power
2005-01-12 Instrukcja programowania EMM 8
# obliczenie potegi (potega = liczba ^2)
write "Potega liczby "
write liczba
write " wynosi "
write potega
beep 1
wait 2
beep 1
# sygnał dzwiekowy podwojny
goto :start
# skok do etykiety
```

3.3.1 Procedura *pauza*

Procedura służy do okresowego zatrzymywania wykonania programu w czasie jego testowania. Wywołanie procedury skutkuje komunikatem na ekranie i oczekiwaniem na naciśnięcie klawisza [Enter]. Składnia tej procedury jest następująca:

call :pauza

lub

let pauza *parametr*

call :pauza

parametr

Parametrem (opcjonalnym) jest numer pauzy. W wypadku wielu wywołań procedury *parametr* pozwala zorientować się o które wywołanie chodzi.

Parametr przekazuje się w zmiennej *pauza*.

Wartość *parametru* pozostaje niezmienna aż do kolejnej - jawnej - zmiany w programie

3.4 Instrukcje *freemem*

Instrukcja *freemem* powoduje wykasowanie wszystkich zmiennych istniejących w programie. Wartości zmiennych są tracone, a ich identyfikatory stają się nieznane (wszystkie zmienne przestają istnieć - sytuacja jest taka, jak przy uruchomieniu programu).

Składnia instrukcji *freemem* jest następująca:

freemem

Instrukcja przeznaczona jest do współpracy z instrukcją *spawn*.

3.5 Instrukcje *spawn*

Instrukcja *spawn* powoduje przerwanie wykonywania bieżącego programu i załadowanie programu podanego jako *argument*. Istniejące w programie zmienne zostają zachowane. Nazwę pliku podaje się w cudzysłowie. Przy wykonaniu instrukcji *spawn* stan systemu (robota, układu transportowego, portów, itd) pozostaje niezmienny.

Składnia instrukcji *spawn* jest następująca:

spawn "nazwa_pliku"

np.

spawn "c:\pliki\program2.prg"

4. Stanowisko

Do instrukcji sterujących stanowiska zalicza się instrukcje *set* i *clear*. Składnia ich jest następująca:

set wyrażenie

clear wyrażenie

gdzie:

wyrażenie *wyrażenie* wyznacza numer portu wyjściowego (od 0 do 14) patrz (4.4).

Instrukcje **set** i **clear** odpowiednio ustawiają stan wskazanego przez wyrażenie portu wyjściowego na "1" lub "0".

Przykład 4.1

```
# podprogram wystawia zderzak stacji kodowania i chowa go po 2 sekundach
set 2
# ustawienie "1" na porcie wyjściowym 2
wait 8
# czas oczekiwania
clear 2
# wyzerowanie portu wyjściowego 2
```

Uwaga:

Przy programowaniu należy uwzględnić czas potrzebny na przetworzenie sygnału sterującego w urządzeniu wykonawczym. Przeważnie okres zadziałania wynosi 0,25 – 1 sekundy w zależności od wielkości elementu wykonującego ruch.

4.1 System transportowy

Rozmieszczenie elementów systemu transportowego określone są w części 1 instrukcji opisującej budowę stanowiska.

4.1.1 Zderzaki

Zderzaki są jednokierunkowe – zatrzymują paletę na stacji tylko przy odpowiednim kierunku najazdu palety. Przy niewłaściwym kierunku ruchu paleta nie zatrzyma się na stacji tylko w jej pobliżu. Zderzaki po wystereowaniu stanowią przeszkodę w ruchu palet w obu kierunkach.

set 2	zderzak stacji kodującej w górze,
set 3	zderzak stacji prawej w górze,
set 4	zderzak stacji lewej w górze,
set 5	zderzak stacji końcowej (odkładczej, po lewej stronie) w górze,
set 12	zderzak stacji odczytu w dole (umożliwiony przejazd palety)

Zderzaki po wystereowaniu są aktywne (zatrzymują palety - wyjątek zderzak stacji odczytu). Zderzaki zatrzymują palety na stacjach ze swojej prawej strony, tj. unieruchomienie palety na odpowiedniej pozycji odbywa się przy ruchu jej w lewo) - wyjątek stanowi zderzak końcowy stanowiska odkładczego z lewej strony stanowiska.

4.1.2 Podniesienie stacji

set 0	stacja robocza lewa i prawa jednocześnie uniesione (pozycjonowanie)
set 13 set 14	uniesienie stacji załadowniczo-wyładowczej (po prawej stronie stanowiska), należy stosować obie komendy jednocześnie, przy czym wpisanie obu komend musi następować w kolejnych liniach

Przykład 4.2

```
# podprogram unoszacy palete na stacji zaladowczo-wyladowczej
set 13
set 14
write "Paleta na stacji zal-wyl uniesiona"
```

Do podniesienia stacji zał-wył wykorzystujemy właściwość systemu transportowego polegającą na braku jednoczesnego występowania “1” na obu wspomnianych wyjściach portu. Wynika to z uwagi na oczywistą niemożność równoczesnego ruchu w dwóch kierunkach. Kombinacja tych dwóch sygnałów sterujących “1” jest wykorzystywana do wysterowania podnośnika stacji zał-wył bez konieczności zwiększania ilości sygnałów (np. poprzez multiplikację) przy ograniczonych zasobach systemowych sterownika PLC.

4.1.3 Kodowanie

Do identyfikacji palet wykorzystywany jest kod ustawiany w kostce kodowej zamontowanej na palecie. Odpowiedni kod może być przypisany np. odpowiedniemu rozmieszczeniu elementów na palecie.

set 8	kodowanie palety – trzpień 3 wysunięty,
set 9	kodowanie palety – trzpień 4 wysunięty,
set 10	kodowanie palety – trzpień 1 wysunięty,
set 11	kodowanie palety – trzpień 2 wysunięty,
clear 8	kodowanie palety – trzpień 3 cofnięty,
clear 9	kodowanie palety – trzpień 4 cofnięty,
clear 10	kodowanie palety – trzpień 1 cofnięty,
clear 11	kodowanie palety – trzpień 2 cofnięty,

Kod palety może zostać nadany jedynie zatrzymanej palecie na stacji kodowania. Proces kodowania polega na pojedynczym wysunięciu trzpienia z pary i zwłoce czasowej przed komendą chowającą dany trzpień. Potem należy wysunąć kolejny trzpień z pary nadających określony kod. Kolejność wysuwania trzpieni dowolna.

kod palety 10	trzpień 1 i 3
kod palety 9	trzpień 2 i 3
kod palety 6	trzpień 1 i 4
kod palety 5	trzpień 2 i 4

Przykład 4.3

```
# podprogram ustawia kod 5
set 9
wait 1
clear 9
set 11
wait 1
# ciąg dalszy listingu na następnej stronie
```

```
clear 11
wait 1
write "ustawiony kod palety 5 "
```

Kodowanie palety odbywa się na stacji kodowania w czasie spoczynku palet. Zwłoka czasowa po komendzie wysuwającej trzpień umożliwia jego ruch. Brak opóźnienia nie pozwala przemieścić się trzpieniowi do pozycji kodującej. Identyczna sytuacja jest przy jego cofaniu. Przed przemieszczeniem palety ze stacji kodowania należy upewnić się czy w programie znajdują się komendy chowające trzpień i występuje zwłoka czasowa na ruch powrotny trzpieni. W przeciwnym wypadku trzpień działać będą jak zderzak. Z uwagi na mniejsze ich wymiary niż zderzak spowoduje to uszkodzenie głowicy kodującej.

Kodowanie palety można przeprowadzić także z zastosowaniem procedury **setkodpal**, która posiada następującą składnię:

```
let kodpal parametr
call :setkodpal
```

<i>parametr</i>	parametrem wywołania jest poprawny kod (tj. ustawialny na palecie), tzn.: 5, 6, 9, 10
-----------------	---

Uwagi:

- kody inne niż poprawne są ignorowane,
- procedura nie działa, gdy na stacji kodowania nie ma palety.

4.1.4 Odczyt kodu

Na podstawie sygnałów z stacji odczytu odpowiadających wysuniętym poszczególnym trzpieniom w kostce kodowej można zidentyfikować paletę. Paleta w czasie odczytu kodu palety musi znajdować się na stacji odczytu. Kod palety w postaci dziesiętnej otrzymywany jest przypisując odczytanym sygnałom następujące wagi:

```
kod_palety = trzpień_4 * 8 + trzpień_3 * 4 + trzpień_2 * 2 + trzpień_1 * 1
co odpowiada
kod_palety = ( ( trzpień_4 * 2 + trzpień_3 ) * 2 + trzpień_2 ) * 2 + trzpień_1 * 1
```

Przykład 4.4

```
# podprogram odczytuje aktualny kod palety
let kodpal 0
# łączniki drogowe LD poprzez podprogram ld_czyt.prg
include ld_czyt.prg
let kodpal ld10 2 * ld09 + 2 * ld08 + 2 * ld07 +
write "Kod palety = "
write kodpal
```

Odczyt kodu palety można zrealizować poprzez procedurę **getkodpal**, której składnia jest następująca:

```
call :getkodpal
```

Procedura nie wykorzystuje żadnych parametrów. W odpowiedzi na wywołanie procedura ta zwraca wartość, która jest przekazywana w zmiennej globalnej *kodpal*.

<i>kodpal</i>	Możliwe wartości to: 5, 6, 9 lub 10
---------------	-------------------------------------

Wartość zmiennej *kodpal* może być dowolnie wykorzystywana, nie zaleca się jednak jej zmieniania.

Uwagi:

- procedura nie działa, gdy na stacji odczytu nie ma palety.

4.1.5 Przemieszczanie palet

set 13	ruch palety w prawo,
set 14	ruch palety w lewo,

Uwaga: Aby zatrzymać trwający ruch palet w prawo konieczna jest komenda

```
clear 13
```

Zmiana kierunku ruchu palet wymaga między zatrzymaniem ruchu w danym kierunku a uruchomieniem przeciwnego ruchu 1 sekundowej przerwy czasowej na zatrzymanie się silnika.

Przykład 4.5

```
# program zmiana kierunku ruchu palet
set 13
# ruch palet w prawo
wait 10
# czas oczekiwania (ewentualnie odczyt z czujnika)
clear 13
# zatrzymanie ruchu w prawo palet
wait 4
# opóźnienie wymagane przy zmianie kierunku!!!
set 14
# ruch palet w lewo
wait 10
# czas oczekiwania
clear 14
# zatrzymanie ruchu w prawo palet
```

Uwaga:

Gdy po komendzie nakazującej ruch w danym kierunku nie odwołamy ruchu palet a dodatkowo polecimy ruszać się transporterowi w przeciwnym kierunku, nastąpi awaryjne zatrzymanie ruchu i podniesienie się stacji załadowczo-wyładowczej niezależnie czy na stacji znajduje się paleta czy nie. Należy unikać tej sytuacji.

Ruch palet może być zrealizowany poprzez użycie procedury *ruchpal*, której składania jest następująca:

```
let pal2poz parametr
call :ruchpal
```


gdzie jako parametr stosuje się:

1	stacja załadowczo-wyładowcza,
2	stacja kodowania,
3	stacja odczytu,
4	stacja robocza prawa,
5	stacja robocza lewa,
6	lewa skrajna stacja końcowa,

Uwagi:

- przed uruchomieniem programu wykorzystującego bibliotekę paleta musi znajdować się na pozycji 1 (stacja załadowczo-wyładowcza),
- wywołanie procedury z innym parametrem niż 1, 2, 3, 4, 5 lub 6 powoduje zatrzymanie programu,
- wywołanie jałowe (np. z *parametrem* 3 gdy paleta stoi na stacji 3) są ignorowane.

Przykład 4.6

```
# przemieszczenie palety z pozycji dowolnej na pozycje 5 realizuje sekwencja
let pal2poz 5
call :ruchpal
```

4.2 Manipulator

Przestrzeń roboczą określa rysunek 1. Podano na nim również współrzędne chwytaków na wieszakach, pozycji bezpiecznej oraz pozycję kołków bazujących stacji roboczych.

Uwaga: **Przestrzeń robocza układu symulatora i systemu EMM różnią się nieznacznie.**

4.2.1 Ruch

Przemieszczenie kiści robota z chwytem lub bez niego może dokonywać się jedynie w jednej osi na raz. Do czasu zakończenia ruchu zatrzymane jest przetwarzanie kolejnych linii programu.

Instrukcja sterująca ruchem robota ma postać:

robotgo os pozycja

gdzie:

<i>os</i>	zmienna lub stała oznaczająca oś (1-3), 1-ruch w poprzek (lewo-prawo), 2-ruch w głąb (przód-tył), 3-ruch wzdłuż (górze-dół) - patrz punkt 4.2.4,
<i>pozycja</i>	wyrażenie określające pozycje (położenie) w danej osi w mm,

Przykład 4.7

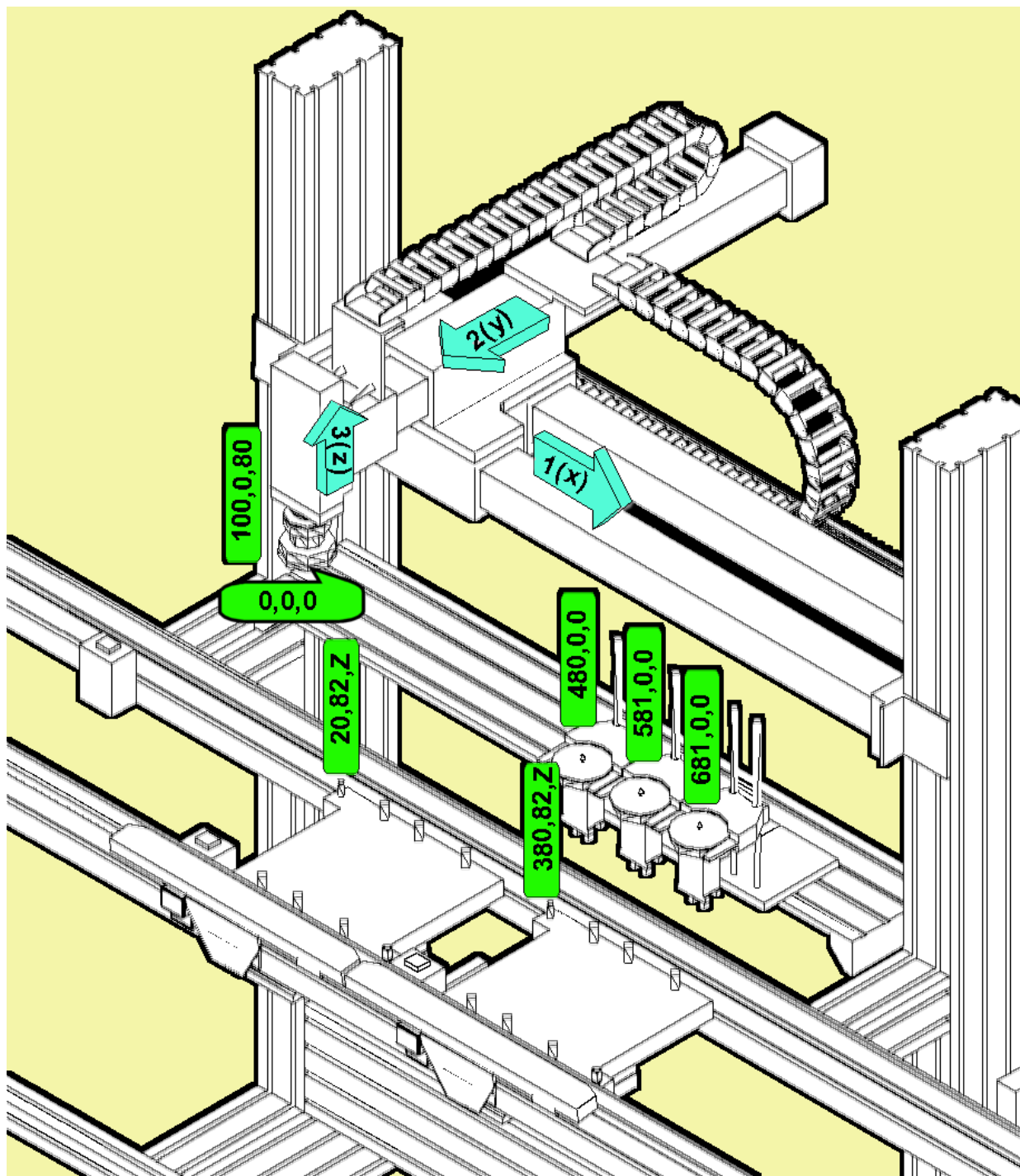
```
# ruch w trzeciej osi (gora-dol) na pozycje 80
robotgo 3 80
```

Zakresy ruchów w poszczególnych osiach wynoszą.(rys.1):

Oś 1(x) - wartości od 0 do 690

Oś 2(y) - wartości 0-380

Oś 3(z) - wartości 0-80

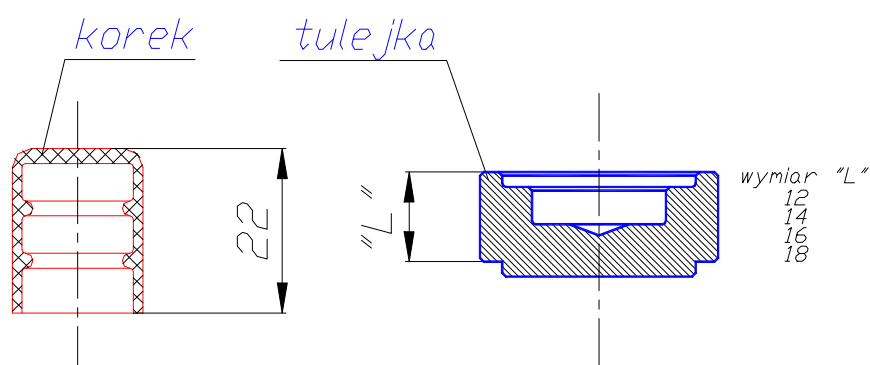


Rys.1

4.2.2 Manipulowane części

Robot może przemieszczać dwa rodzaje części przedstawione na rysunku 2.

Część typu korek (1 sztuka) przenoszona jest za pomocą chwytaka zewnętrznego (wyposażonego w zewnętrzne nakładki szczęk). Korek może być umieszczony bezpośrednio na palcu lub za pośrednictwem tulejek. Do dyspozycji jest siedem części typu tuleja o wymiarach "L" równych: 12 (2 sztuki), 14 (1 sztuka), 16 (2 sztuki) i 20 (2 sztuki). Umieszczane są one w gniazdach wykonanych na paletach. Patrz punkt 4.2.3 Budowa tulejek umożliwia umieszczenie jednej na drugiej. Uzyskana dzięki temu kombinacja wysokości rozszerza możliwości manipulacji częściami. Do przemieszczania pojedynczych tulejek nie obsadzonych korkiem wykorzystywany jest chwytak wewnętrzny.

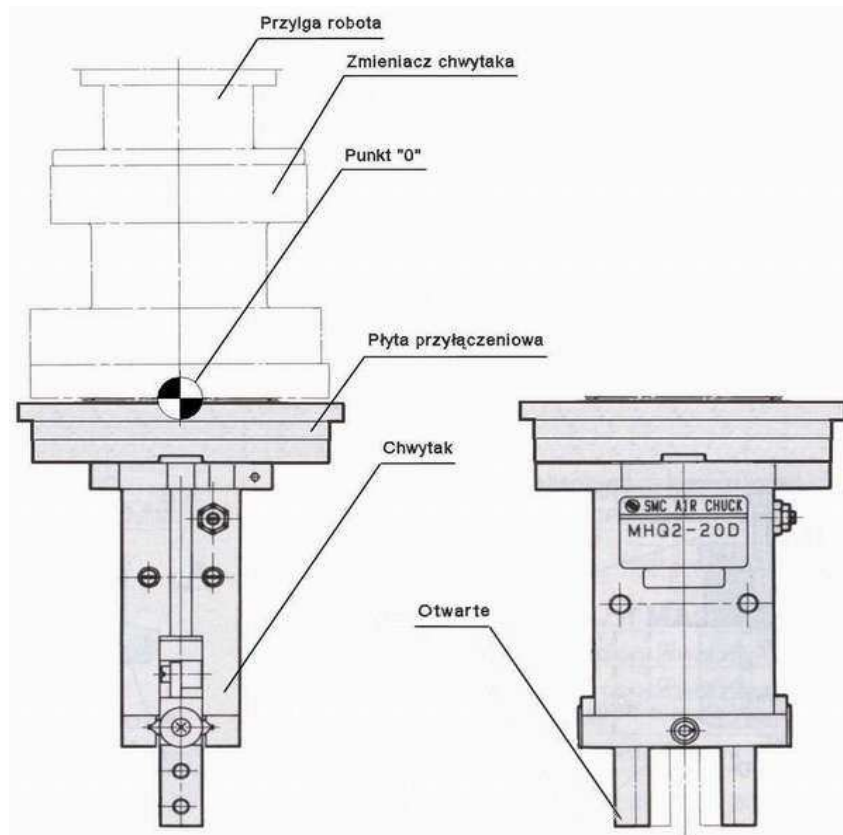


Rys. 2

4.2.3 Chwytaaki

set 7	chwytak zamocowany w kiści (przez zmieniacz chwytaków) robota,
set 6	otwórz szczęk chwytaka zamocowanego w kiści robota,

W celu zamknięcia szczęk chwytaka uchwyconego w kiści (rysunek 3) należy odwołać komendę otwierającą poprzez instrukcję **clear**. Odmocowanie chwytaka może odbyć się jedynie po dojeździe bezpieczną trajektorią przez robota do wolnego wieszaka. Po odmocowaniu konieczny jest ruch w osi 3 na współrzędną co najmniej 20 przed ruchem w pozostałych osiach lub zakończeniem programu.



Rys. 3

Pozycje chwytaków na wieszakach. Liczby określają kolejne współrzędne bezwzględne w poszczególnych osiach (oś 1, oś 2 i oś 3) w układzie EMM:

Wieszak lewy:	Wieszak środkowy:	Wieszak prawy:
480, 0, 0	581, 0, 0	681, 0, 0

Pobranie chwytaka pustą(!!!) kłasią manipulatora następuje od góry - trzecia oś o wartości minimum 20, następnie wymagane jest osiągnięcie dwóch pozostałych współrzędnych i dopiero po ich uzyskaniu obniżenie pustej kłasi w dół, do podanych wyżej wartości współrzędnych odpowiednich wieszaków. Po uchwyceniu chwytaka w kłasi niezbędny jest ruch do przodu z uchwytu wieszaka w osi 2 na pozycję 15, zapewniającej wysunięcie się ze sprężyn mocujących chwytak na wieszaku i umożliwiając ruch jedynie w pionie. Ruch w poprzek (oś 1) możliwa jest dopiero po minięciu (przyrost wartości współrzędnych w osi 2 z wartości 15 do 80) ramion wieszaka – niezależnie od współrzędnej w osi 3.

Procedura **chwytak** pozwala, przy zachowaniu określonych warunków na pominięcie ustalania dokładnego przebiegu ścieżki manipulatora w celu pobrania z magazynu żadanego chwytaka. Automatycznie następuje przejazd chwytaka z dowolnej pozycji na pozycję określoną jako bezpieczną. W kolejnych krokach następuje pobranie chwytaka i jego wyprowadzenie z magazynu. Procedura posiada następującą składnię:

let chwytak *parametr*

call :chwytak

gdzie jako parametr stosuje się:

0	odłóż chwytak, jeżeli w kłasi manipulatora nie znajduje się żaden chwytak to nic się nie stanie, jeżeli już jakiś chwytak jest założony, to zostanie odłożony na swój wieszak w magazynie,
1	pobierz chwytak nr 1 (tzn. ten w wieszaku nr 1 - skrajnym lewym) jeżeli w kłasi manipulatora jest już chwytak nr 1, to nic się nie dzieje; jeżeli jest już jakiś inny chwytak, to zostanie on odłożony na swoje miejsce i pobrany zostanie chwytak nr 1
2	analogicznie jak dla parametru 1 (tylko wieszak środkowy)
3	analogicznie jak dla parametru 1 (tylko wieszak prawy)

Uwaga:

- przed uruchomieniem programu wykorzystującego bibliotekę wszystkie trzy chwytaki muszą być w wieszakach,
- użycie innego parametru niż 0, 1, 2 lub 3 zatrzymuje program,
- po pobraniu chwytaka robot staje na pozycji:
x=450/550/650 (zależnie od chwytaka)
y=55
z=80
- procedurę należy wywoływać w bezpiecznym położeniu robota nad paleta, zalecane są następujące współrzędne:
x=450/550/650 (zależnie od chwytaka)
y=55
z=80
- za doprowadzenie w sposób bezpieczny do tego położenia odpowiada autor programu wykorzystującego bibliotekę,
- błąd uchwycenia chwytaka przez kłasi robota powoduje przerwanie programu.

Przykład 4.8.a i bprzykład z użyciem tzw. **języka niższego rzędu**

```
# demo 48a
# pobranie chwytaka nr 3
# kisc pusta; robot na pozycji 500,30,45
robotgo 3 80
robotgo 2 0
robotgo 1 650
robotgo 3 0
set 7
wait 2
get ch_pom 10
goifplus ch_pom :jest_ok
write "Bład uchwycenia chwytaka"
newl
stop
:jest_ok
robotgo 2 20
robotgo 3 80
robotgo 2 55
stop
# robot na pozycji 650,55,80
# koniec
```

przykład z użyciem tzw. **języka wyższego rzędu**

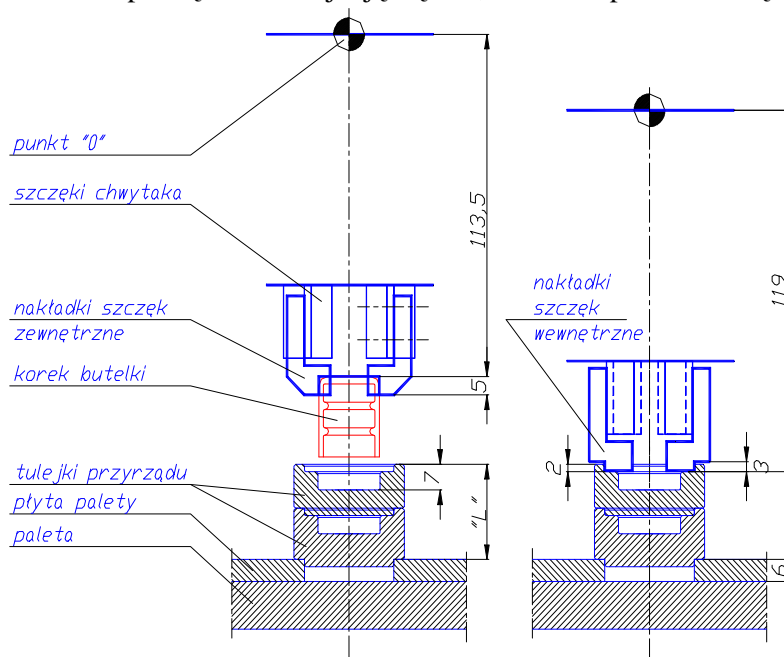
```
# demo 48b
# pobranie chwytaka nr 3
# kisc pusta; robot na pozycji 500,30,45
include robolib.prg
let chwytak 3
call :chwytak
stop
# robot na pozycji 650,55,80
# koniec
```

4.2.4 Współrzędne na paletach

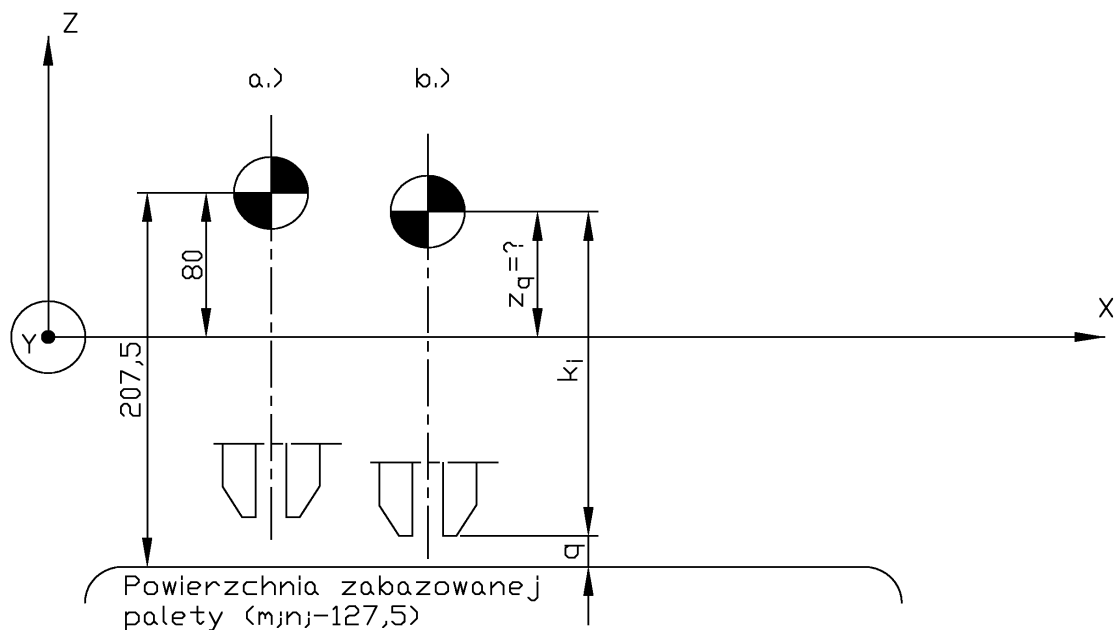
Rozmieszczenie gniazd w paletach zabazowanych na stacjach roboczych względem kołków bazowych przedstawia rysunek 5 (str.22).

4.2.5 Współrzędne manipulatora

Współrzędne osi 1 i 2 robota zgodne z punktem 4.2 i 4.2.2 Do uchwycenia manipulowanych części należy zająć robotem z zamocowanym chwytakiem zewnętrznym lub wewnętrznym (wyposażonym w odpowiednie nakładki szczęk) określoną pozycję nad paletą w osi 3, patrz rysunek 4. Punkt "0" robota wystawianego w osi 3 na współrzędna 80 znajdują się 207,5 mm nad powierzchnią zabazowanej palety.



Rys.4



Rys. 5 Schemat poglądowy przedstawiający punkt programowy, chwytak, paletę
 a.) dane wg instrukcji i układu rzeczywistego,
 b.) szkic do wyznaczenia współrzędnej z_q dla zadanej odległości q chwytaka od powierzchni zabazowanej palety

Rysunek 5 pozwala na wyznaczenie współrzędnej w osi Z dla żądanej odległości szczęk uchwytu od powierzchni zabazowanej palety. Na rysunku 5 poszczególne oznaczenia mają następujące znaczenia:

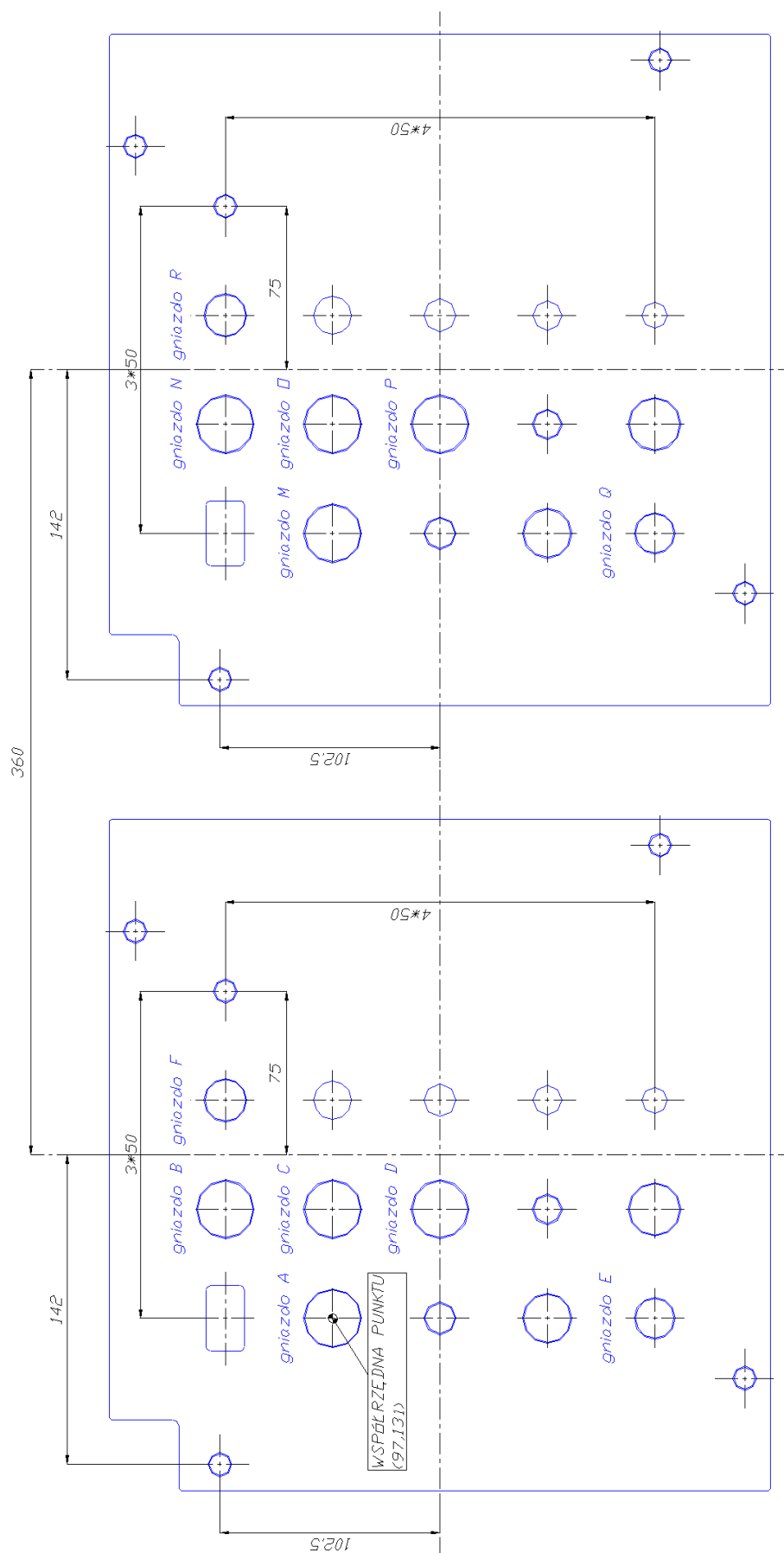
- X, Y, Z – układ współrzędnych,
- m,n – wartości współrzędnych odpowiednio w osiach x i y,
- q – zadana odległości q chwytaka od powierzchni zabazowanej palety,
- k_i lub k_1 – znana wysokość chwytaka
- z_q – poszukiwana współrzędna w osi z

$$z_q = 80 - [207,5 - (q + k_i)]$$

$$z_q = 80 - 207,5 + (q + k_i)$$

$$z_q = q + k_1 - 127,5$$

Na rysunku 6 przedstawiono wymiary i wzajemne zależności geometryczne palet na stanowiskach montażowych.



Rys. 6

W celu uchwycenia przez chwytak najniższej tulei spoczywającej bezpośrednio w gnieździe zabazowanej palety należy opuścić się z zamkniętymi szczękami wewnętrznymi nad odpowiednim gniazdem na współrzędną w osi 3 wynoszącą 2.5 (kropka). Umieszczenie korka w tej tulei wymaga opuszczenia chwytaka z elementem w zamkniętych szczękach na wysokość 14 w osi 3 (nad tym gniazdem).

Przykład 4.9

```
# podprogram pobranie chwytaka prawego
:chap_prawy
write "Pobranie chwytaka prawego z pozycji gotow do pozycji gotow"
newl
# przejście do nowej linii
write "Osiaganie pozycji - W gore"
robotgo 3 80
write ", w bok"
robotgo 1 100
write ", do tylu"
robotgo 2 0
write " - pozycja bezpieczna"
newl
write "W prawo po chwytak"
robotgo 1 650
write ", w dol"
robotgo 3 0
write " na dole."
newl
write "Pobierz chwytak..."
set 7
write "uchwycony."
newl
write "Do przodu"
robotgo 2 15
write ", wyjazd ze sprezyn zakonczony."
newl
write "W gore"
robotgo 3 80
write ", do przodu"
robotgo 2 80
write ", w lewo"
robotgo 1 100
write ", do tylu."
robotgo 2 0
write " Robot w pozycji bezpiecznej"
return
```

Współrzędne pozycji bezpiecznej robota wynoszą: **100, 0, 80**. W tej pozycji należy ustawiać robota w czasie przerw w jego wykorzystywaniu. Pozycja bezpieczna jest także zalecana jako początek i koniec podprogramów z wykorzystaniem robota. Przed zakończeniem programu należy odłożyć pobrany chwytak na wolny wieszak.

Uwaga: **System symulatora ma inną przestrzeń roboczą.**

4.3 Odczyt stanów czujników stanowiska

Instrukcja get odczytuje stan portu wejściowego wskazanego przez wyrażenie i zapisuje ją w zmiennej *zmienna*. Składnia jej jest następująca:

get zmienna wyrażenie

gdzie:

<i>zmienna</i>	przechowuje stan portu
<i>wyrażenie</i>	wyznacza numer portu wejściowego (0-23)

Przykład 4.10

```
# definicja zmiennej Id03 (nadanie wartosci 2, rozna od odczytanej 1 lub 0)
let Id03 2
# odczyt portu 23 do zmiennej Id03
get Id03 23
```

Spis czujników w zależności od numeru portu:

0	paleta na stacji załadowczo-wyładowczej
1	wieszak prawy zajęty,
2	wieszak środkowy zajęty,
3	wieszak lewy zajęty,
4	chwytak wewnętrzny,
5	czujnik prawy chwytaka daje sygnał – otwarty,
8	chwytak trzeci,
9	czujnik lewy chwytaka daje sygnał – zamknięty,
10	chwytak zewnętrzny
12	paleta przy zderzaku końcowym z lewej (stanowisko odkładcze),
13	przycisk start stacji załadowczo-wyładowczej,
14	paleta na stacji kodowania,
15	paleta na stacji odczytu,
16	trzcień kostki kodowej 1 z 4 lewy / górny (działa na przemian z 2),
17	trzcień kostki kodowej 2 z 4 lewy / dolny (działa na przemian z 1)
18	trzcień kostki kodowej 3 z 4 prawy / górny (działa na przemian z 4)
19	trzcień kostki kodowej 4 z 4 prawy / dolny (działa na przemian z 3)
20	wjazd na stację Prawa
21	paleta na stacji Prawej w dole
22	wjazd na stację Lewą
23	paleta na stacji Lewej w dole

Zmienna zmienna musi być wcześniej zadeklarowana przez wystąpieniem w instrukcji `get`. Patrz (3.1)

Dla ułatwienia i ujednolicenia zmiennych odczyt może następować poprzez podprogram `ld_czyt.prg` przypisujący do zmiennych `ldNN` stany portów wejściowych "0" lub "1" gdzie *NN* to numer zmiennych:

ld01	paleta na "lewy koniec",
ld02	wjazd na "stacja Lewa" zmiana stanu "1" na "0",
ld03	paleta na "stacja Lewa",
ld04	wjazd na "stacja Prawa" zmiana stanu "1" na "0",
ld05	paleta na "stacja Prawa",
ld06	paleta na "stacja odczytu",
ld07	kod 1 z 4 lewy / górny (działa na przemian z 2),
ld08	kod 2 z 4 lewy / dolny (działa na przemian z 1),
ld09	kod 3 z 4 prawy / górny (działa na przemian z 4),
ld10	kod 4 z 4 prawy / dolny (działa na przemian z 3),
ld11	paleta na "stacja kodowania",
ld12	paleta na "stacja załadowcza",
ld13	"wieszak prawy" zajęty,
ld14	"wieszak środkowy" zajęty,
ld15	"wieszak lewy" zajęty,
ld16	chwytak sygnał "zamknięty",
ld17	chwytak sygnał "otwarty",
ld18	przycisk "start" "stacja załadowcza" naciśnięty,
ld19	kod chwytaka wewnętrznego,
ld20	kod chwytaka bez nakładek szczęk,
ld21	kod chwytaka zewnętrznego

Podprogram `ld_czyt.prg` przypisujący stany wszystkich portów wejściowych do zmiennych wymaga aby przed jego pierwszym użyciem wywołać podprogram `ld_def.prg` definiujący zmienne. W przypadku ruchu palet aby wychwycić występowanie krótkich czasowo sygnałów z jedynie z łączników drogowych dotyczącego systemu transportowego zaleca się korzystać z podprogram `ld_trans.prg`. Dzięki zmniejszeniu ilości sprawdzanych portów podprogram wykonuje się szybciej. W krańcowym przypadku, tj. użycia wewnątrz pętli dużej ilości instrukcji, zaleca się użycia pojedynczej komendy `get` wykonywanej dużo szybciej w stosunku do wywoływania odpowiedniego podprogramu czytającego stan wielu portów wejściowych.

4.4 Opis portów wyjściowych

Instrukcje sterujące względem numer portu wyjściowego:

0	stacja robocza Lewa i Prawa uniesione (palety pozycjonowane),
1	wolne,
2	zderzak stacji kodującej w górze,
3	zderzak stacji prawej w górze,
4	zderzak stacji lewej w górze,
5	zderzak stacji odkładczej w górze (aktywny z lewej strony),
6	otwórz szczęki chwytaka zamocowanego w kiści robota,
7	chwytak zamocowany w kiści (przez zmieniacz chwytaków) robota,
8	kodowanie palety - trzpień 3 wysunięty,
9	kodowanie palety - trzpień 4 wysunięty,
10	kodowanie palety - trzpień 1 wysunięty,
11	kodowanie palety - trzpień 2 wysunięty,
12	zderzak stacji odczytu w dole (paleta nie zatrzymywana),
13	ruch palet w prawo,
14	ruch palet w lewo (z 13 uniesienie stacji załadowczo-wyładowczej)

5. Uwagi

Napisany program przed uruchomieniem na stanowisku montażowym należy sprawdzić w symulatorze. Sprawdzenie odbywa się ręcznie z uwagi na brak bezpośredniej możliwości jednoczesnego uruchomienia kompilacji i symulatora.

Instrukcja i oprogramowania symulatora dostępnego pod adresem <http://www.cim.pw.edu.pl/monma I/> odnoszą się do stanowiska montażowego przed modyfikacją jego usytuowania w budynku ST. W czasie zmiany lokalizacji we wrześniu 2003 nastąpiła reorganizacja, głównie we współrzędnych robota. Z tego powodu w korzystaniu z tej instrukcji podczas pracy z symulatorem do czasu zmodernizowania jego oprogramowania należy uwzględniać poprawki dostępne w instrukcji symulatora.

Wykorzystanie następujących procedur:

chwytak
ruchpal
setkodpal
getkodpal
pauza

wymaga zastosowania biblioteki w swoim programie. Biblioteka zawarta jest w pliku `robolib.prg`, który należy dołączyć poleceniem:

```
include robolib.prg
```

ewentualnie z podaniem ścieżki dostępu, np.

```
include c:\programy\robolib.prg
```

Dyrektywa `include` powinna wystąpić na początku pliku.

6. FAQ Najczęściej zadawane pytania

Po zakończeniu programu paleta przemieszcza się poza stację docelową	Należy uwzględnić bezwładność układu transportowego przy jego wyłączaniu.
Program działa ale kompilator zgłasza błąd.	W programie musi wystąpić instrukcja stop .
Podniesiony zderzak opuszcza się po zakończeniu programu.	Koniec pracy kompilatora wymusza powrót stanowiska EMM do stanu początkowego.

Instrukcja została przygotowana na podstawie:

- "Język sterowania montażowego gniazda obróbkowego - gramatyka języka" wersja 1.12 autorstwa dr inż. Macieja HORCZYCZAKA ITM-PW
- "Biblioteka ROBOLIB" wersja 0.92 autorstwa dr inż. Macieja HORCZYCZAKA ITM-PW