

Instrukcja Programowania Robota Kuka KR 125

Zestawienie poleceń

Wstęp

Instrukcja ta została przetłumaczona z oryginału jako dodatek do pracy magisterskiej przez Dariusz Wróblewskiego i Łukasza Nurka.

Zawiera ona zbiór komend oraz zasad programowania robota KUKA KR125. Przydatna jest ona szczególnie w czasie programowania w trybie off-line.

Mamy nadzieję, iż pomoże ona innym studentom w lepszym zrozumieniu zasad programowania robotów przemysłowych oraz będzie cenną pomocą naukową w czasie prowadzenia zajęć praktycznych na hali Wydziału Mechanicznego Politechniki Szczecińskiej i ćwiczeń z wykorzystaniem wyżej wymienionego robota.

ANIN - analog input;

Wczytywanie w cyklach wejścia analogowego

Argumenty:

Stan - może przyjmować wartości on/off cyklicznego czytania;

Wartość sygnału - czyli wynik operacji może być zmienną, elementem pola albo sygnałem;

Mnożnik - może być stałą, zmienną, sygnałem albo elementem pola;

Nazwa sygnału - opisuje symboliczną nazwę zdefiniowaną poleceniem SIGNAL i musi być to nazwa pewnego wejścia analogowego;

Offset - wartość offset może być stałą, zmienną albo wartością pola;

Opis:

Moduł analogowy udostępnia interfejs analogowy, który może być czytany poprzez predefiniowaną zmienną sygnałową \$ANIN. Wejście analogowe może być wczytywane przez dłuższy okres czasu instrukcją ANIN albo jednorazowo do zmiennej typu REAL poprzez znak „=”, podczas cyklicznego wczytywania komendą ANIN. Wartości analogowe są wczytywane w odstępach 14ms. W tym samym czasie dopuszczalne są dwie komendy ANIN-ON. Dwie komendy ANIN-ON można wpisać do tej samej wartości sygnału albo odczytywać z tego samego interfejsu analogowego. Dzięki opcjonalnej arytmetyce komendy ANIN można wyjście analogowe połączyć z dalszymi operatorami i udostępnić wartość sygnału. Jedna komenda ANIN-ON może czytać maksymalnie trzy interfejsy analogowe.

UWAGA:

Wszystkie wykorzystane zmienne albo pola muszą zostać zadeklarowane w liście zmiennych.

Wskazówka:

Moduł analogowy udostępnia 8 analogowych interfejsów wielkości 12 bitów (4,88mV ogniwa galwanicznego). Napięcie wejściowe może się wahać od -10V do 10V, ale nie może przekraczać 35V. Na wejściu od strony hardware'u może mieć przydzielony nr interfejsu poprzez parametr systemowy.

Przykład:

1.

Korekta toru(zmienna systemowa \$TECHIN[1]) powinna zostać zmodyfikowana według wejścia sensora \$ANIN[2]. Wejście sensora \$ANIN[2] jest połączone z symboliczną zmienną SIGNAL1. Wynik mnożenia zmiennej FAKTOR przez SIGNAL1 jest dodawane do zmiennej OFFSET i przypisywany cyklicznie zmiennej systemowej korekty toru \$TECHIN[1].

SIGNAL SIGNAL1 \$ANIN[2]

ANIN ON \$TECHIN[1] = FAKTOR * SIGNAL1 + OFFSET

2.

Za pomocą komendy ANIN OFF cykliczne odpytywanie SIGNAL1 kończy się.

ANIN OFF SIGNAL 1

ANOUT - analog output;

Sposób podawania wyjścia analogowego.

Argumenty:

Stan - może być on/off;

Nazwa sygnału - zdefiniowana wcześniej komenda SIGNAL;

Mnożnik - może być stałą zmienna sygnałem albo elementem pola;

Operand - może być zmienną, sygnałem albo polem;

Opis:

Podawanie wyjścia analogowego inaczej niż w przypadku wyjścia binarnego lub cyfrowego nie jest podawane w postaci wartości zmiennej, ale sterowane za pomocą komendy ANOUT. Za pomocą argumentu „Stan” cykliczne podawanie sygnału analogowego jest wyłączane albo włączane. Nazwa sygnału jest podawana za pomocą komendy SIGNAL.

Wynik uzyskiwany za pomocą obliczenia wartości wyjścia analogowego jest obliczany cyklicznie. Nie może być jednak zbyt wysoki, jego obliczenie musi się mieścić w ramach jednego cyklu. Wynik obliczenia musi się zawierać w zakresie od -1 do 1 albo od 0 do 1 w zależności od konfiguracji hardware'u. Jeśli wynik przekracza tę granicę to jest ustalany na wartość brzegową, a dodatkowo pojawia się wskazówka że sygnał osiągnął wartość graniczną. Ten stan utrzymuje się do czasu aż sygnał znowu wróci do swoich granic (od -1 do 1 lub od 0 do 1).

Za pomocą opcjonalnego słowa kluczowego DELAY można zaprogramować zarówno dodatnie jak i ujemne opóźnienie, aby sygnał ustawić trochę wcześniej lub później, a wartość opóźnienia jest mierzona w sekundach czasu rzeczywistego.

Wskazówka:

Sterowanie robota udostępnia 16 wyjść analogowych (\$ANOUT[1]...\$ANOUT[16]). Niewykorzystane wyjścia mogą służyć jako markery.

Przykład:

1.

Symbolicznej nazwie ANALOGI zostaje przypisane wyjście analogowe \$ANOUT[5]. Włączając obliczanie wartości analogowej uaktywnia się następujące obliczenia iloczyn zmiennej FAKTOR i zmiennej systemowej \$TECHVAL[1], zwiększa się o wartość OFFSET1. Jest to liczone cyklicznie i przypisywane do wyjścia analogowego \$ANOUT[5]. Komenda ANOUT OFF ANALOGI ustalanie wyjścia analogowego jest zakończone.

```
SIGNAL ANALOGI $ANOUT[5]  
ANOUT ON ANALOGI = FAKTOR * $TECHVAL[1] + OFFSET1
```

```
ANOUT OFF ANALOG 1
```

2.

Wyjście analogowe \$ANOUT[1] jest przypisane sterowaniu dozownika z taśmą klejącą o symbolicznej nazwie KLEBER (wyjście analogowe jest funkcją). Wartość sterująca jest obliczana przez podzielenie prędkości toru na dwa (zmienna systemowa \$VEL_ACT) i przez dodanie stałej 0,2. Sygnał wyjściowy obliczany cyklicznie jest opóźniany przez opcjonalny parametr DELAY o 0,05s. Komendą ANOUT OFF KLEBER komenda ta kończy ustawianie wyjścia analogowego.

```
SIGNAL KLEBER $ANOUT[1]  
ANOUT ON KLEBER = 0.5*$VEL_ACT + 0.2 DELAY = 0.05
```

```
ANOUT OFF KLEBER
```

BRAKE

Zatrzymanie ruchu robota.

Argumenty:

F - dodanie opcjonalnego parametru F powoduje zatrzymanie w dużo szybszym czasie. Jeśli dla wybranego typu silnika został zaprojektowany tor maszyny za pomocą zmiennej \$EMSTOP_PATH to robot będzie zatrzymany w optymalnym czasie na tym torze. W przeciwnym wypadku osie będą zatrzymywane synchronicznie w czasie a tor zostanie opuszczony (robot zejdzie ze swojego toru).

Przykład:

```
DEF STOPUP()
BRAKEF KLEBER +
FALSE
WAITFOR$IN[10]
LIN $POS_RET
KLEBER = TRUE
END
```

CIRC

Ruch po okręgu

Argumenty:

Punkt pomocniczy - jest to geometryczne wyrażenie dowolnego punktu pomocniczego na torze po okręgu. Jedyne dopuszczalne są współrzędne kartezjańskie. Punkt pomocniczy może być także wyuczony, aby to uczynić należy podać pojedynczy znak „!”. Jako punkt pomocniczy oraz pozycję docelową. System odniesienia w kartezjańskim układzie dla pozycji pomocniczej jest definiowany poprzez zmienną systemową \$BASE. Kąt kierunkowy i kąt statusowy oznaczony jako S i T. Wewnątrz punktu pomocniczego są ignorowane.

Jeśli jako punkt pomocniczy wstawi się niezdefiniowane struktury to wartości tego punktu będą wyjęte z aktualnej pozycji;

Pozycja docelowa - jest to geometryczne wyrażenie opisujące pozycję docelową. Tylko współrzędne kartezjańskie są dozwolone do użytku. Pozycja docelowa może także być uczoną, do tego celu używamy znaku „!”. Punkt odniesienia jest definiowany zmienną systemową \$BASE.

Kąt obrotu - arytmetyczne wyrażenie, które w połączeniu ze słowem kluczowym CA umożliwia wydłużenie albo skrócenie łuku, jednostką miary jest stopień. Dla kąta obrotu nie ma ograniczenia, można w szczególności zaprogramować kąt obrotu jako większy niż 360° . Jeśli kąt obrotu jest dodatni to tor tego obrotu przyjmie kierunek taki, aby przejść przez pozycję startową, punkt pomocniczy aż do pozycji docelowej. Jeżeli jest negatywny to ruch będzie się odbywał w przeciwnym kierunku. Jeśli podaje się kąt obrotu to zaprogramowana pozycja docelowa z reguły nie jest pozycją docelową tylko zostanie ona ustalona na podstawie podanego kąta obrotu.

Opis:

Za pomocą instrukcji CIRC sterownik oblicza ruch po okręgu od aktualnej pozycji poprzez punkt pomocniczy do pozycji końcowej, która jest ustalana poprzez pozycję docelową, albo poprzez kąt obrotu. Ruch odbywa się przez punkty wyznaczone metodą interpolacji. Podobnie jak przy ruchu PTP trzeba zaprogramować prędkość i przyspieszenia oraz zmienne systemowe \$TOOL i \$BASE. Prędkości i przyspieszenia odnoszą się nie do ciągu motoru w każdej osi, ale do TCP. Należy zdefiniować następujące zmienne systemowe \$VEL (prędkość), \$ACC (przyspieszenie).

Sterowanie:

Sterowanie może być względem aktualnego systemu bazowego zdefiniowanego przez zmienną \$BASE albo względem układu współrzędnych maszyny (kierowanego po torze trójnoga). Ustalenie, która z tych opcji będzie ustawiona odbywa się przez zmienną systemową \$CIRC_TYPE. Dodatkowo można wybrać czy orientacja będzie stałą czy zmienną. Jeśli wybierze się zmienną orientację z uwzględnioną będzie zmianą orientacji w trakcie drogi od pozycji startowej do zaprogramowanego miejsca docelowego. Zmiana ta będzie przeprowadzana w stosunku do bazy albo do toru. Jeśli wybierze się stałą orientację to przy sterowaniu zorientowanym na bazę, orientacja w każdym punkcie będzie identyczna z orientacją w punkcie startowym. Natomiast przy wyborze sterowania zorientowanego na tor, orientacja relatywna będzie stała względem trójnoga. Ewentualnie dodana orientacja w pozycji docelowej przy wybraniu stałej orientacji zostanie zignorowana. Zmienna systemowa dotycząca ustalenia orientacji ma nazwę \$ORI_TYPE.

Status kąta:

Podczas ruchów po okręgu status kąta w punkcie końcowym jest zawsze identyczny ze statusem kąta w punkcie początkowym. Podane S i T wewnątrz pozycji docelowej typu POS i E6POS zostaną zignorowane.

UWAGA:

Aby zagwarantować zawsze jednakowe poruszanie się należy jednoznacznie i jednorazowo zdefiniować układ osi. Dlatego pierwsza komenda ruchu programu powinna być zawsze komenda PTP z podaniem S i T.

Wyglądanie (zmienianie toru):

Dokładne wyznaczanie punktów pomocniczych jest niepotrzebne i jest strata czasu. Dlatego w pewnym zdefiniowanym odstępnie od pozycji docelowej można zacząć wyglądać.

Przykład:

1.

Ruch po okręgu z uczeniem punktu pomocniczego i współrzędnych docelowych.

CIRC!

2.

Ruch po okręgu z wyuczonym punktem pomocniczym i punktem docelowym. Punkt końcowy ruchu będzie ustalony przez kąt -35° .

CIRC!,CA-35

3.

Ruch po okręgu z zaprogramowanym punktem pomocniczym i punktem docelowym za pomocą współrzędnych kartezjańskich.

CIRC {X 5,Y 0, Z 9.2},{X 12.3,Y 0,Z -5.3,A 9.2,B -5,C 20}

4.

Ruch po okręgu z zaprogramowanym punktem pomocniczym i punktem końcowym, wygładzanie jest aktywne.

CIRC {Z 500,X 123.6},PUNKT1,CA +260 CORI

5.

Stale zorientowane względem toru ruchu.

```
$ORI_TYPE=#CONST
$CIRC_TYPE=#PATH      CIRC
H_PUNKT,{X 34,Y 11,Z 0}
```

6.

Wygładzanie okrężno liniowe z punktu 2 do punktu 3. Wygładzanie zaczyna się 30mm od punktu2.

```
$APO.CDIS=30
$APO.CPTP=20
PTP PUNKT1
CIRC H_PUNKT,PUNKT2,CA WINKEL CDIS
PTP PUNKT3
```

CONFIRM

Służy do potwierdzenia odbioru komunikatów.

Argumenty:

Numer zarządzający - jako parametr podaje się numer będący liczbą całkowitą albo wyrażenie arytmetyczne, które obliczane jest do liczby, którą identyfikuje komunikat. Podanie 0 jako parametru powoduje konfirmację wszystkich możliwych potwierdzeń komunikatów.

Przykład:

1.

Potwierdzenie przyjęcia komunikatu numer 27.

```
CONFIRM 27
```

2.

Potwierdzenie przyjęcia komunikatu o numerze MNUMMER.

```
CONFIRM MNUMMER
```

3.

Potwierdzenie przyjęcia komunikatu o numerze MNUMMER+5.

```
CONFIRM MNUMMER+5
```

4.

Potwierdzenie wszystkich oczekujących komunikatów.

```
CONFIRM 0
```

5.

Po wysłaniu komunikatu sygnału stopu zostaje wysłany komunikat potwierdzenia. Przed dalszym opracowaniem komunikatu stopu, komunikat potwierdzenia musi zostać potwierdzony. Następujący program rozpoznaje i potwierdza przyjęcie komunikat automatycznie o ile tryb pracy silnika (nie ręczny) jest odpowiedni, i rzeczywiście stan stopu został ustawiony.

```
DEF AUTO_QUIT()  
INTM
```



```

DECL STOPMESS MLD ; predefiniowany typ struktury dla komunikatów
                                stopu
IF $STOPMESS AND $EXT THEN ; sprawdzenie komunikatu stopu i
                                trybu pracy
    M=MBX_REC($STOPMB_ID,MLD) ; odczytanie aktualnego stanu
                                w MLD
    IF M==0 THEN ; sprawdzenie czy można potwierdzić komunikat IF
        ((MLD.GR0==2) AND (MLD.STATE==1)) THEN
            CONFIRM MLD.CONFNO ; potwierdzenie danego
                                komunikatu ENDIF
    ENDIF ENDIF END

```

CONTINUE

Służy uniemożliwieniu zatrzymania przebiegu programu.

Opis:

Zmienną systemową \$ADVANCE można ustawić ile komend ruchu powinno być wykonanych. Jednak komendy, które oddziałują na otoczenie (komendy wejścia/wyjścia), ich przebieg obliczenia zostaje zatrzymany. Aby do tego nie dopuścić należy umieścić komendę CONTINUE przed operacją wejścia/wyjścia.

UWAGA:

Komenda CONTINUE jest ważna tylko dla bezpośrednio następującej po niej linii kodu.

Wskazówka:

Komendy, które powodują zatrzymanie programu, to:

ANOUT

HALT WAIT

PULSE

\$ANOUT

\$AXIS_ACT \$AXIS_BACK \$AXIS_FOR \$AXIS_RET

\$POS_ACT \$POS_BACK \$POS_FOR \$POS_RET

SIN \$OUT

\$OV_PRO

Dostęp do zmiennych importowanych oraz jeśli VL_STOP_BASE=TRUE (w /R1/\$CONFIG>DAT) także:

\$BASE \$TOOL \$EXBASE

Przypisanie do tych zmiennych również powoduje zatrzymanie programu.

Przykład:

1.

Uniemożliwienie zatrzymania programu podczas przypisania do zmiennej \$OUT:

```
CONTINUE
$OUT[1]=TRUE
CONTINUE
$OUT[2]=FALSE
```

DECL

Służy do definiowania zmiennych i pól.

Składnia służąca do definiowania zmiennych lub pól bez ustawiania ich wartości początkowych

```
<DECL>Datatyp Variable1,..., VariableN,  
      Feldname1 [Groessel,... ,Groesse3 ],  
  
      FeldnameN[Groessel,... ,Groesse3 ]
```

Składnia do ustawiania zmiennych z wartościami początkowymi (tylko w listach danych)

```
<DECL>Datentyp Variable = Wert
```

Składnia do pól, którym ustawia się początkową wartość (tylko w listach danych)

```
<DECL>Datentyp Feldname[Groessel,... ,Groesse3 ]  
  
      Feldname[Feldindex1,...,Feldindex3] = Wert  
  
      Feldname[Feldindex1,...,Feldindex3] = Wert
```

Argumenty:

Datentyp - podaje się typ danych definiowanej zmiennej. Mogą to być zmienne następujących typów: INT, REAL, CHAR, BOOL, FRAME, POS, E6POS, AXIS, E6AXIS oraz samodzielnie definiowane typy danych STRUĆ jak i ENUM.

Variable - nazwa zmiennej

Feldnam - nazwa pola (tablicy)

Wielkość - podaje się w nawiasach kwadratowych (tablica), jej max. wymiar to 3 elementy.

Wartość - przy ustawianiu wartości początkowej

Index tablicy - indeks w tablicy

Opis:

W programie wszystkie zmienne, których chce się używać powinny zostać zadeklarowane za pomocą nazwy i typu danych. Dostępne są typy proste, złożone i zdefiniowane przez użytkownika. Deklaracja zaczyna się słowem

kluczowym DECL następnie po niej występuje typ danych i lista zmiennych lub pól, które powinny być danego typu przy deklarowaniu zmiennych i pól jednego z predefiniowanych typów. Słowo kluczowe DECL może zostać pominięte. Predefiniowane typy danych to INT, REAL, CHAR, i BOOL oraz typy POS, E6POS, FRAME, AXIS, E6AXIS etc. deklaracja zmiennych typu POS może zostać całkowicie pominięta, typ ten jest typem standardowym i domyślnym. Zmienne można definiować i jednocześnie ustawiać ich wartość początkową. Deklaracja razem z ustawieniem wartości początkowej nie jest dozwolona w pewnej części programów i funkcji deklaracyjnej (na początku programu). Wartość początkową zmiennych typów podstawowych podaje się jako zwykłą stałą. Wartość zmiennych będących typu strukturalnego jest agregatem. Wartość podaje się po znaku „,=”. Pola, czyli tablice muszą być definiowane i muszą mieć ustalone wartości pojedynczo dla każdego elementu nie można podać listy elementów. Wyjątkiem są tablice znaków, wtedy do takiej tablicy można przypisać cały tekst.

Przykład:

1.

Deklaracja bez ustawienia wartości początkowej.

```
DECL POS PI  
INTA1,A2  
REAL GES[7],BES[7],b  
DECL SPARTYP S_PAR[3]
```

2.

Deklaracja pól z ustawieniem wartości początkowych (tylko w listach danych).

```
INT A[7] ; Tablica 7 wartości całkowitych  
A[1]=27 ; Pierwszemu elementowi przypisano liczbę 27  
A[2]=313  
A[6]=11
```

```
CHARTEXT1[80]  
TEXT1[]="Meldetext"
```

```
CHAR TEXT2[2,80] TEXT2[1,  
]="ErsterMeldetext" TEXT2[2,  
]="ZweiterMeldetext"
```


3.

Deklaracja zmiennej z ustawienia wartości początkowej.

FRAME F1={X 123.4, Y -56.7, Z 89.56}

DEF

Uzgadnianie programów i podprogramów.

Argumenty:

Nazwa programu - przy globalnych funkcjach max. 8 znaków przy funkcjach lokalnych max. 12 znaków.

Lista parametrów - typy danych wszystkich parametrów zewnętrznego podprogramu w zdefiniowanej kolejności. Przy parametrach typu pole rozmiary pola podawane są następująco [], [,], [, ,]

Deklaracje - w deklaracjach wszystkie komendy są niedozwolone. Granica pomiędzy deklaracjami a poleceniami jest definiowana poprzez pierwsze polecenie. W deklaracjach trzeba uzgodnić (deklaracje formalnego parametru lokalnego podprogramu, deklaracje typów danych, deklaracje lokalnych zmiennych bez wstępnej definicji, deklaracja nazw kanałów i sygnałów).

Polecenia - wyjście z podprogramu poleceniem RETURN. Bez polecenia RETURN ostatnim wykonanym poleceniem jest END.

Opis:

DEF jest używana do deklaracji globalnego podprogramu lub lokalnego. Przy wykonaniu programu rozróżnia się 2 rodzaje przekazywanych parametrów. Przekazanie z parametrem wejścia i wyjścia.

Przykład:

1.

Definicja programu bez formalnych parametrów.

```
DEF PROG()
```

```
END
```

2.

Definicja programu z formalnymi parametrami STROM, SPANNUNG

automatycznie ustawianymi jako OUT.

```
DEF SCHWEISS (STROM,SPANNUNG)
```

```
END
```

3.

Definicja programu z formalnymi parametrami STROM, SPANNUNG automatycznie ustawianymi jako IN i jako OUT.

```
DEF SCHWEISS (STROM:IN,SPANNUNG:IN,RESULT:OUT)
```

```
END
```

4.

W podprogramie zmienne są poddawane operacją obliczeniowym za pomocą zmiennej RECHNE (licz). Zmienne przybierają wartości A=1, B=2.

```
DEF PROG()
```

```
INT A,B
```

```
A=1
```

```
B=2
```

```
RECHNE(A,B)
```

```
END
```

```
DEF RECHNE(X1:OUT,X2:IN)
```

```
INT X1,X2
```

```
X1=X1+10
```

```
X2=X2+10
```

```
END
```

DEEDAT

Służy do definiowania list danych.

Argumenty:

Nazwa listy danych - w tym miejscu wstawia się nazwę zdefiniowanej listy danych. Jest to nazwa obiektu, która może mieć max 8 znaków. Długość jest ograniczona systemem katalogów. Jeśli nazwa listy danych jest identyczna z nazwą modułu to lista danych jest przypisana do tego modułu. Dzięki temu deklaracje wewnątrz listy danych są widoczne w całym module. Moduł i przypisane mu listy danych razem tworzą większy moduł. Public - użycie tego słowa pozwala na to, aby zdefiniowane dane w liście danych zmienne mogły być dostępne w innych modułach, aby to umożliwić w innych modułach należy użyć komendy IMPORT.

Lista definicji - zawiera ona zewnętrzne deklaracje podprogramu i funkcji, która będzie wykorzystywana w danym podmodule. Deklaracje zmiennych importowanych, deklaracje zmiennych, deklaracje nazw sygnałów i kanałów, deklaracje struktur i typów wyliczeniowych. Wszystkie powyższe deklaracje można również umieścić w centralnej liście danych w pliku \$CONFIG.DAT. Deklaracje zmiennych w części deklaracyjnej listy danych mogą zawierać przypisanie wartości początkowej.

Opis:

Oprócz predefiniowanych list danych można definiować własne listy danych. Służą one do zgrupowania i do udostępnienia specyficznych dla programu definicji. Listy danych mogą definiować archiwizowane wartości zmiennych, w szczególności listy danych mogą zawierać wyuczone pozycje. Listy danych mogą istnieć jako niezależne obiekty pod warunkiem że nie istnieje żaden podmoduł o identycznej nazwie. Słowo kluczowe ENDDAT kończy definicję listy danych.

UWAGA:

W liście danych nie może być żadnych komend.

Przykład:

1.

Definicja listy danych.

```
DEFDAT SCHWEISS
```

```
ENDDAT
```

2.

Definicja listy danych globalnie dostępnej.

```
DEFDAT ZENDAT PUBLIC
```

```
ENDDAT
```

3.

Moduł PROGI składa się z podmodułu i przypisanej mu listy danych PROGI. Jeśli zmiennej OTTO w programie głównym zostanie przypisana nowa wartość to zostanie ona wpisana do listy danych i zapisana na stałe. Sterowanie wejścia/wyjścia będzie się odbywało już z nową wartością zmiennej, jest to niezbędne do korekt online albo innych korekt programowych.. Jeśli chce się zawsze pracować z tą samą zmienną w programie głównym to trzeba odpowiednie zmienne ustawić w głównym programie.

```
DEFDAT PROGI
```

```
INT OTTO = 0
```

```
ENDDAT
```

```
DEFPROG10
```

```
HALT ; OTTO ma wartość 0
```

```
OTTO=25
```

```
HALT ; teraz w liście danych jest INT OTTO=25
```

```
END
```

4.

Globalne listy danych: zmienna OTTO powinna być widoczna w PROGI i PROG2.

Zdefiniowane w lisice danych zmienne można udostępnić obcemu programowi głównemu trzeba tylko zdefiniować listy danych jako PUBLIC a w obcym głównym programie należy użyć komendy IMPORT.

```
DEFDAT PROGI PUBLIC  
INT OTTO = 0 ENDDAT
```

```
DEFPROG10  
HALT OTTO = 25  
END
```

```
DEF PROG_2()  
IMPORT INT OTTO2 IS /R1/PROG1.. OTTO  
END
```

DEFFCT

Jest to deklaracja funkcji

Argumenty:

Nazwa funkcji - nie może być ona dłuższa niż 8 znaków, jest to ograniczenie narzucone przez system plików. Funkcje lokalne mogą mieć do 12 znaków.

Lista parametrów - są one danymi zwykłymi i podaje się je po przecinku lub mogą być również tablicami i wtedy podajemy je w nawiasach kwadratowych. Tablica może być maksymalnie 3 wymiarowa. Po nazwie parametru można podać :IN lub :OUT (parametr wejściowy lub parametr wyjściowy). Parametr wyjściowy jest przekazywany przez referencje.

Blok deklaracji - w tym bloku powinny znajdować się tylko deklaracje, funkcje są niedozwolone w tym bloku. Granica między blokiem deklaracji a blokiem funkcji jest zdefiniowana przez pierwszą występującą funkcję. W części deklaracyjnej można zadeklarować następujące rzeczy: deklaracje parametrów formalnych, deklaracje typów danych, deklaracje zmiennych lokalnych bez ustalenia wartości początkowej, deklaracje nazw sygnałów i kanałów. Część funkcyjna - powinny się w niej znajdować tylko funkcje i wywołania komend. Deklaracje nie są tu dopuszczone. Lokalne funkcje muszą być opuszczane za pomocą komendy RETTJRN.

Opis:

Funkcja zwraca wartość do wywołującego ją podmodułu. Wywołanie funkcji jest komendą i dlatego jest możliwe przypisanie wyniku funkcji do zmiennej, albo użycie funkcji w blokach arytmetycznych. Deklaracja DEFFCT służy zarówno jako definicja globalnej funkcji jak i lokalnej. Można ją wywoływać z parametrami wejściowymi i wyjściowymi. Parametry wejściowe przekazywane są przez wartość. Bezpośrednie przekazanie parametru działa tak jak zdefiniowanie zmiennej funkcji. Przekazany parametr może być stałą, zmienną, wywołaniem innej funkcji, prostym albo złożonym typem. Zwracanie wartości parametru wejściowego poleceniem RETURN nie jest możliwe. Parametr wejściowy służy wyłącznie do przekazywania wartości do funkcji. Parametrowi wyjściowemu należy przypisać nazwę zmiennej gdyż jest to wywołanie przez referencje. Zmienna ta może w momencie wywoływania funkcji posiadać jakąś wartość, ta wartość może być wykorzystywana w podprogramie. W funkcji parametrowi wejściowemu można przypisać nową wartość i będzie ona z powrotem przekazana podmodułowi i może być w dalszym bloku programu wykorzystana. Przekazywanie parametru jako parametr wyjściowy jest wykonywane domyślnie, tzn. że :OUT może zostać pominięty.

Komenda ENDFCT jest ostatnim wyrażeniem globalnej albo lokalnej funkcji. Ostatnie wykonywane polecenie to instrukcja RETURN. Jeśli interpreter podczas przetwarzania funkcji napotka ENDFCT, to zostanie wyświetlony błąd.

Przykład:

1.

Deklaracja funkcji zwracającej zmienną typu INT o nazwie FKT1 do której przekazuje się parametry P1 i P2.

```
DEFFCT INT FKT1(P1,P2)
```

```
ENDFCT
```

2.

Deklaracja funkcji zwracająca zmienną typu BOOL o nazwie SCHWEISS parametry wejściowe to STROM (napięcie) i SPANNUNG (natężenie).

```
DEFFCT BOOL SCHWEISS (STROM:IN, SPANNUNG:IN)
```

```
ENDFCT
```

3.

W podanej funkcji powinna zostać obliczona różnica dwóch zmiennych i przekazana programowi głównemu.

```
DEFPROG_1()  
EXTFCT INT DELTA(INT:OUT,INT:OUT)  
INT A,B,C  
A=1  
B=25  
C=DELTA(A,B)  
END
```

```
DEFFCT INT DELTA(X1:OUT,X2:OUT)  
INT X1,X2,X3  
X3=X2-X1  
RETURN(X3)  
ENDFCT
```


DIGIN

Służy do cyklicznego wczytywania wejścia cyfrowego.

Argumenty:

Stan - może przyjmować wartości ON/OFF, czyli rozpoczęcie wczytywania i zakończenie wczytywania;

Wartość sygnału - jest wynikiem operacji, może to być zmienna, element tablicy albo sygnał;

Faktor - może być stałą, zmienną, sygnałem albo elementem tablicy.

Nazwa sygnału - opisuje zdefiniowaną przy pomocy słowa kluczowego SIGNAL symboliczną nazwę. Signalname musi opisywać wejście cyfrowe.

Offset - jest typu rzeczywistego i może to być stała, zmienna, sygnał albo element tablicy.

Opis:

Sterowanie udostępnia 3 cyfrowe interfejsy, które mogą być odczytywane za pomocą zmiennych systemowych \$DIGIN1, \$DIGIN2 i \$DIGIN3. Każde z tych wejść cyfrowych ma długość 32 bitów i przynależne mu wyjście strobowe. Wejście cyfrowe może zostać wczytane w pewnym przedziale czasu za pomocą komendy DIGIN albo jednorazowo za pomocą operatora „=” i przypisania do zmiennej typu REAL. Jednocześnie dostępne jest wykorzystanie dwóch komend DIGIN ON. Obie komendy DIGIN ON mogą odczytywać to samo wejście cyfrowe. Za pomocą opcjonalnej arytmetyki możliwe jest połączenie wejścia cyfrowego z innymi operatorami i przypisanie jakiemuś sygnałowi. Komenda DIGIN ON może także odwoływać się do sygnału wejściowego analogowego (patrz przykład).

Cykliczne wczytywanie wejścia cyfrowego zostaje zakończone komendą DIGIN OFF.

UWAGA:

Wszystkie użyte zmienne i tablice muszą zostać zadeklarowane w liście danych.

Wskazówka:

Dostęp do wejścia cyfrowego powoduje zatrzymanie przebiegu programu. Indeksy tablic są obliczane jednorazowo w komendzie DIGIN ON natomiast wyrażenie, które powstaje po zastąpieniu indeksów tablicy wartościami

numerycznymi będzie obliczane cyklicznie. Wejścia cyfrowe nie mogą być używane w komendzie INTERRUPT.

Przykład:

1.

Binarne sygnały wejściowe od \$IN[1020] do \$IN[1026] zostają przypisane wejściu \$DIGIN1. Analogowemu wejściu \$ANIN[1] zostaje przypisana nazwa FAKTOR. Zmienna systemowa \$TECHIN[1] będzie zawierać wynik mnożenia wartości wejścia analogowego i cyfrowego podwyższoną o wartość zmiennej OFFSET. Komenda DIGIN OFF dezaktywuje cykliczne wczytywanie wejścia cyfrowego.

```
SIGNAL $DIGIN1 $IN[1020] TO $IN[1026]
SIGNAL FAKTOR $ANIN[1]
DIGIN ON $TECHIN[1] = FAKTOR * $DIGIN1 + OFFSET

DIGIN OFF $DIGIN1
```

ENUM

Pozwala zdefiniować typ wyliczeniowy.

Argumenty:

Pierwszym argumentem jest nazwa.

Pozostałe argumenty oddzielone przecinkami jest to lista wartości.

Opis:

Typ wyliczeniowy jest taki sam jak w każdym języku programowania. Możliwe jest używanie zmiennych wyliczeniowych w taki sposób, że przed wartością wyliczeniową stosujemy „#”. Jeśli trzeba użyć pełnej definicji to należy zastosować następującej adnotacji: nazwa typu numerycznego#nazwa konkretnej wartości, np.: WOCH_TYP#MONTAG. Pełna symboliczna pewnej stałej musi być podana w dwóch przypadkach:

1. Jeśli stała wyliczeniowa jest parametrem aktualnym podprogramu albo funkcji i jest używana w pojedynczej komendzie poza podmodulem;
2. Jeżeli stała wyliczeniowa jest po lewej stronie porównania;

Przykład:

1.

Deklaracja typu wyliczeniowego ZUSTANDTYP i zawierającego stałe ZSTART,, ZSTOP, ZWART.

```
ENUM ZUSTANDTYP ZSTOP, ZSTART, ZWART
```

2.

W tym programie typ wyliczeniowy o nazwie SCHALTERTYP zawiera stałe EIN (włączone) i AUS(Wyłączone). Te obie stałe zostają użyte wewnątrz programu przy pomocy krótkiej adnotacji.

```
DEF PROG()
  ENUM SCHALTERTYP EIN, AUS
  DECL SCHALTERTYP KLEBER IF
  A>10 THEN
    #KLEBER=EIN
  ELSE
    #KLEBER=AUS
  ENDIF END
```

EXIT

Służy do bezwarunkowego wyjścia z pętli. Może być używana w każdej pętli.

Przykład:

1.

Wyjście z nieskończonej pętli.

```
LOOP
    A=(A+1)*0.5/B IF
    A>=13.5 THEN
        EXIT
    ENDIF
ENDLOOP
```

EXT

Deklaracja zewnętrznego podprogramu.

Argumenty:

Parametry komendy - ścieżka i nazwa wykorzystywanego podprogramu

Lista parametru - jest to lista przekazywana do tego podprogramu.

Opis:

Komenda EXT daje możliwość wywoływania globalnych podprogramów z innych programów. Globalny podprogram zapisuje się jako osobny plik źródłowy. Nie można takiego podprogramu odróżnić od programu głównego. Każdy podprogram, który jest wykorzystywany musi być wywołany osobną komendą EXT. Nazwa i ścieżka wywoływanego podprogramu podobnie jak używane parametry muszą zostać przedstawione kompilatorowi. Podanie listy parametrów jednocześnie rezerwuje miejsce do ich zapisania.

UWAGA:

Komenda EXT stojąca w zewnętrznym podprogramie jest dostępna jedynie w części globalnej. W lokalnych podprogramach komenda EXT jest niedozwolona.

Przykład:

1.

Deklaracja zewnętrznego podprogramu o nazwie UP1 w katalogu R1.

EXT/R1/UP1()

2.

Deklaracja zewnętrznego podprogramu o nazwie UP1 i deklaracja niezbędnych parametrów.

EXT /UP1(INT, REAL:IN, CHAR[],

3.

Deklaracja zewnętrznego podprogramu o nazwie UP i deklaracja niezbędnych parametrów.

```
DEF PROG( )      ; Program główny
EXT UP(INT:OUT, REAL:OUT,BOOL:IN)
INTA
REALB

BOOLC

UP1 (A,B,C)

END

DEF UP (X1 :OUT,X2:OUT,X3:IN)  ; Zewnętrzna funkcja
INTX1
REALX2
BOOL X3

END
```

EXTFCT

Deklaracja zewnętrznej funkcji. Wszystkie parametry i dane są takie same jak w funkcji. Zewnętrzna funkcja jest zapisywana jako osobny plik. Każda zewnętrzna funkcja wykorzystywana w programie musi mieć swoją własną linię EXTFCT. Nazwa i ścieżka wywoływanej zewnętrznej funkcji tak samo jak parametry muszą być znane kompilatorowi tak samo jak w komendzie poprzedniej przy podawaniu listy parametrów odpowiednie miejsce zostanie zarezerwowane.

Przykład:

1.

Deklaracja zewnętrznej funkcji o nazwie FKT1 w katalogu R1.

EXTFCT/R1/FKT1()

2.

Deklaracja zewnętrznej funkcji o nazwie FKT1 i przekazywanych jej parametrów.

EXTFCT /FKT(INT,REAL:IN,CHAR[],INT[,],:IN)

3.

Deklaracja zewnętrznej funkcji o nazwie FKT1 i przekazywanych jej parametrów.

EXTFCTFKT1(INT:OUT,REAL:OUT,BOOL:IN)

4.

W funkcji jest obliczana różnica dwóch zmiennych i jest zwracana głównemu programowi, wartości zmiennych po wywołaniu funkcji są następujące: A=15, B=20, C=15.

DEF PROG() ; Program główny EXTFCTINT
DELTA(INT:OUT,INT:IN) INT A,B,C

A=25 B=20
C=DELTA(A,B)

END

DEFFCT INT DELTA(X1:OUT,X2:IN) ; Zewnętrzna funkcja
INT X1,X2,X3
X1=X2-X1
X3=X1
RETURN(X3)
ENDFCT

FOR

Argumenty:

Licznik - zmienne typu INT służą jako licznik pętli

Start - wyraz arytmetyczny podający wartość początkową licznika

Ende - wartość końcowa

Inkrementacja - może być ujemna, nie może być równa 0 i nie może być zmienną.

Opis:

Warunkiem wykonania pętli FOR jest:

- przy dodatniej inkrementacji: jeśli licznik jest większy niż wartość końcowa to kończy pętle.

- przy ujemnej inkrementacji: jeśli licznik jest mniejszy niż wartość końcowa to też kończy pętle.

Warunek wykonania pętli jest zawsze sprawdzany przed jej przejściem, tak samo jak wartości początkowe i końcowe. Inkrementacja nie może być 0, może być ujemna, jeżeli nie jest podana to przyjmuje wartość 1. Zmienne w pętli są zmiennymi globalnymi. Każda pętla FOR należy zakończyć komendą ENDFOR.

Przykład:

1.

```
FORA=1TO10  
    B=B+1  
ENDFOR
```

2.

```
FORA=1TO15STEP2  
    B=B+A  
    IF B==10 THEN  
        EXIT  
    ENDIF  
ENDFOR
```

GOTO

Jest to komenda skoku. Jako parametr przyjmuje etykietę, która opisuje miejsce docelowe skoku.

Przykład:

1.

Bezwarunkowy skok do miejsca oznaczonego MARKEI.

```
GOTO MARKEI
```

2.

Bezwarunkowy skok do miejsca w programie z etykietą ENDE.

```
IF X>100 THEN
    GOTO ENDE
ELSE
    X=X+1
ENDIF
A=A*X

ENDE:
END
```

HALT

Powoduje zatrzymanie wywołania programu i zatrzymuje prace robota.
Wznowienie programu może zostać wznowione po wciśnięciu klawisza start.

IMPORT

Służy do importu danych z list danych.

Argumenty:

Importname - można tu podać inną nazwę, jeżeli chce się zaimportować dane do zmiennej o innej nazwie.

Ścieżka i nazwa listy danych - podaje się skąd dane będą importowane.

Datename - odpowiada nazwie zmiennej, która ma być wczytana z zewnętrznej listy danych.

Opis:

Funkcja IMPORT pozwala na dostęp do zewnętrznych list danych, które są opisane atrybutem PUBLIC. IMPORT musi być użyty dla każdej jednej importowanej zmiennej. Nie można użyć dwa razy komendy IMPORT do tej samej zmiennej. Zmienne można importować tylko z listy danych, w której ta zmienna została zadeklarowana. Dane należy importować z identycznym typem jak ten zdefiniowany w liście danych.. Dopuszczalne jest nadanie danej innej nazwy niż ta zdefiniowanej w zewnętrznej liście danych. Źródło i nazwa danych są oddzielone dwoma kropkami(..), między kropkami nie może być żadnego pustego znaku

Przykład:

1.

Import wartości zmiennej całkowitej o nazwie WERT z listy danych o nazwie DATEN. Nazwa zmiennej WERT jest przejmowana.

```
IMPORT INT WERT IS /DATEN. .WERT
```

2.

Import tablicy POS_EX z listy danych R1/POSITION. Nazwa zmiennej zmienia się na POS1 w programie importującym tą zmienną.

```
IMPORT POS POS1[] IS /R1/POSITION..POS_EX
```

INTERRUPT DECL

Jest to definicja przerwania.

Argumenty:

Priorytet - jest to wyrażenie arytmetyczne typu całkowitego, które definiuje priorytet o wartości od 1 do 128. Wartość 1 ma największy priorytet.

Warunek - jest to logiczne wyrażenie definiujące, kiedy ma nastąpić przerwanie. Dopuszczalne w tym miejscu są: stała boolowska, zmienna albo element tablicy, nazwa sygnału, proste porównanie, proste logiczna operacja (NOT, OR, AND i EXOR). Niedozwolone w tym miejscu są struktury.

Wywołanie podprogramu - nazwa i parametry podprogramu (nazywanego funkcją przerwania), które powinien zostać wywołany w momencie wystąpienia przerwania.

Opis:

Funkcja przerwania daje możliwość reakcji na jakieś wydarzenie wewnątrz programu, która to sytuacja nie jest zsynchronizowana czasowo z przebiegiem programu. Takie wydarzenie to np.: wystąpienie błędu, pewien sygnał wejściowy, etc. Deklaracja przerwania definiuje możliwe przyczyny przerwania i odpowiednia reakcje systemu. Każde przerwanie musi mieć priorytet, warunek i odpowiednia funkcje przerwania. Jednocześnie można zdefiniować 32 przerwania. Deklaracja przerwania w każdej chwili może być nadpisana przez inną deklarację.

UWAGA:

Deklaracją przerwania jest komenda, dlatego nie może się znaleźć w części deklaracyjnej.

Zdefiniowane przerwanie wyzwała reakcje dokładnie wtedy, kiedy wszystkie poniższe warunki są spełnione:

1. Przerwanie jest włączone (INTERRUPT ON)
2. Przerwanie jest aktywne (INTERRUPT ENABLE)
3. Przerwanie ma najwyższy priorytet ze wszystkich występujących przerw
4. Połączony z tym przerwanie warunek został spełniony. Jeśli jednocześnie wystąpi dwa lub więcej przerw to wykonane zostanie w pierwszej kolejności to o najwyższym prioryecie

UWAGA:

Przerwanie zostanie rozpoznane na tym poziomie, na którym zostanie zadeklarowane.

W wyższych warstwach programu przerwanie mimo aktywacji nie zostanie rozpoznane. Oznacza to, że przerwanie zadeklarowane w podprogramie nie zostanie obsłużone w programie głównym. Po rozpoznaniu warunku aktualna pozycja robota zostaje zapisana a funkcja przerwania wywołana, może to być lokalny podprogram albo zewnętrzny podprogram. Przerwanie kończy się komenda RETTJRN albo END. Następnie przerwany program jest wznawiany od miejsca, w którym został przerwany.

Wskazówka:

Zmienne \$TOOL i \$BASE nie są dostępne w podprogramie przerwania.

Wyrażenia, które w normalnym programie powodują przerwanie programu (np.: \$OUT[]), nie zatrzymują wywołania funkcji przerwania. Funkcja przerwania wykonywana jest z poziomu komend i jest wykonywana komenda po komendzie. Przerwania zmiennych systemowych \$EM_STOP i \$STOPMESS są obsługiwane także w przypadku błędu, czyli pomimo zatrzymania robota (ale żadne komendy ruchu nie są wykonywane).

UWAGA:

Jeśli program stoi na komendzie HALT to przerwania i tak będą rozpoznawane i wykonywane (także komendy ruchu). Po obsłużeniu przerwania program będzie w dalszym ciągu stał w komendzie HALT.

Przykład:

1.

Definicja przerwania z priorytetem 5, która wywołuje podprogram STOPUP, jeśli zmienna \$STOPMESS jest prawdą.

```
INTERRUPT DECL % WHEN $STOPMESS DO STOPUP()
```

2.

Definicja przerwania o priorytecie 23, która wywołuje podprogram UP1 z parametrami 20 i WERT, jeśli zmienna \$IN[12] jest prawdą.

```
INTERRUPT DECL23 WHEN $IN[12]==TRUE DO UP1(20,WERT)
```

3.

Na zaprogramowanym torze ruchu znajdują się dwa stanowiska, które dzięki dwóm sensorom podłączonym do wejść 6 i 7 mogą zostać rozpoznane. Następnie robot powinien ruszyć z rozpoznanych pozycji.

```
DEF PROG()
```

```
  INTERRUPT DECL 10 WHEN $IN[6]==TRUE DO UP1()
```

```
  INTERRUPT DECL 20 WHEN $IN[7]==TRUE DO UP2()
```

```
  LIN STARTPKT
```

```
  INTERRUPT ON
```

```
  INTERRUPT ENABLE
```

```
  LIN ENDPKT
```

```
  INTERRUPT OFF
```

```
  LIN PUNKT1
```

```
  LIN PUNKT2
```

```
END
```

```
DEF UP1()
```

```
  PUNKT1=$POS_INT
```

```
END
```

```
DEF UP2()
```

```
  PUNKT2=$POS_INT
```

```
END
```

INTERRUPT

Służy do aktywacji i dezaktywacji przerwań.

Argumenty:

Słowo kluczowe - może być ON (włączenie), OFF (wyłączenie), ENABLE (udostępnienie), DISABLE (dezaktywacja).

Priorytet - podawany jest po słowie kluczowym, może to być liczba albo wyrażenie arytmetyczne.

Opis:

Ta komenda umożliwia włączanie i wyłączanie przerwań o podanym priorytecie. Jeśli zadeklarowane przerwanie jest włączone to będzie cyklicznie sprawdzane czy wystąpiło. Włączone przerwanie może być aktywowane albo dezaktywowane. Dezaktywacja umożliwia ochronę części programu przed przerwaniem. Dezaktywowane przerwanie będzie rozpoznawane i zapisywane, ale nie wykonywane. Jak tylko nastąpi aktywacja przerwania INTERRUPT ON wszystkie przerwania które wystąpiły do tego czasu w odpowiedniej kolejności zostaną obsłużone. Przerwanie, które wystąpi więcej niż raz, zostanie obsłużone jednokrotnie. Przy wystąpieniu przerwania wszystkie przerwania niższego priorytetu mogą zostać zablokowane na czas obsługi tego przerwania. Po powrocie z procedury obsługi przerwania ta blokada zostaje usunięta. Tak samo działa to dla bieżącego przerwania, które zostanie ponownie wywołane jak tylko przerwanie, które miało wyższy priorytet zostanie skończone. Jeśli chce się to wywołanie ponownie zablokować trzeba to zrobić w procedurze wywołania przerwania. Procedura zawsze będzie w pełni wykonana niezależnie czy pojawi się błąd blokowania czy też wyłączenie przerwania. Przerwanie może zostać przerwane przerwaniem o wyższym priorytecie. W tym przypadku programista ma możliwość wyłączenia jednego lub więcej przerwań. Przerwane przerwanie zostaje wznowione w miejscu przerwania, czyli w miejscu, w którym została przerwana.

UWAGA:

Jednocześnie tylko 8 przerwań może być włączonych. Należy szczególnie uważać podczas globalnego aktywowania wszystkich zadeklarowanych przerwań.

Przykład:

1.

Przerwania zadeklarowane z priorytetem 2 zostają włączone.

INTERRUPT ON 2

2.

Przerwania zadeklarowane z priorytetem 5 zostają wyłączone.

INTERRUPT OFF 5

3.

Wszystkie zadeklarowane przerwani zostaną włączone.

INTERRUPT ON

4.

Wszystkie zadeklarowane przerwani zostaną wyłączone.

INTERRUPT OFF

5.

Włączone przerwania z priorytetem 3 zostają aktywowane.

INTERRUPT ENABLE 3

6.

Włączone przerwania z priorytetem 2 zostają dezaktywowane.

INTERRUPT DISABLE 2

7.

Wszystkie włączone przerwania zostają aktywowane.

INTERRUPT ENABLE

8.

Wszystkie włączone przerwania zostają dezaktywowane.

INTERRUPT DISABLE

9.

Podczas wykonywania programu klejenia z poziomu hardware'u zostaje wywołane awaryjne wyłączenie. Programowo chcemy zatrzymać tylko klejenie a po zwolnieniu (wejście 10) pistolet z klejem z powrotem ustawić na torze.

```
DEF PROG()
```

```
  INTERRUPT DECL 1 WHEN $STOPMESS DO STOPUP()
```

```
  LIN PUNKT1
```

```
  INTERRUPT ON
```

```
  INTERRUPT ENABLE
```

```
  LIN PUNKT2
```

```
  INTERRUPT OFF
```

```
END
```

```
DEF STOPUP0
```

```
  BRAKEF
```

```
  KLEBER+FALSE
```

```
  WAITFOR$IN[10]
```

```
  LIN $POS_RET
```

```
  KLEBER=TRUE
```

```
END
```

LIN

Ruch liniowy.

Argumenty:

Pozycja docelowa (Zielposition) - jest to wyrażenie geometryczne, które definiuje punkt końcowy ruchu liniowego. Dozwolone są tylko współrzędne kartezjańskie. Pozycja docelowa może zostać też wyuczona, aby tego dokonać należy użyć znaku „!”. System odniesienia używany do współrzędnych kartezjańskich jest zdefiniowany zmienną systemową \$BASE. Statusy kąta S i T wewnątrz POS i E6POS zostaną zignorowane. Jeśli pozycja docelowa zawiera nie zdefiniowane komponenty to te wartości zostaną wzięte z bieżącej pozycji.

Opis:

Komenda LIN oblicza, w jaki sposób przenieść się ruchem liniowym od aktualnej pozycji do wskazanego przez pozycję docelową miejsca. Ruch odbywa się po punktach, które zostają obliczone w każdym takcie interpolacji i robot przez te punkt przechodzi kolejno. Podobnie jak przy ruchu PTP muszą zostać zdefiniowane prędkości i przyspieszenia oraz zmienne systemowe \$TOOL i \$BASE. Jednak prędkości i przyspieszenia nie odnoszą się do ciągu motora wzdłuż każdej osi, a do TCP. Aby zdefiniować prędkości i przyspieszenie należy skorzystać ze zmiennych systemowych \$VEL (prędkość) i \$ACC (przyspieszenie).

Status kąta:

Przy ruchu LIN status kąta punktu końcowego jest zawsze identyczny ze statusem kąta punktu początkowego. Dane S i T wewnątrz pozycji docelowej typu POS albo E6POS zostaną zignorowane.

UWAGA:

Aby zagwarantować zawsze taki sam ruch należy jednoznacznie zdefiniować układ osi. Dlatego zawsze pierwszą komendą programu powinna być komenda PTP z podanymi S i T.

Orientacja:

Za pomocą zmiennych systemowych \$ORI_TYPE można wybrać pomiędzy stałym i zmiennym sposobem orientacji:

\$ORI_TYPE=#VAR - oznacza, że podczas ruchu liniowego orientacja zmienia się w jednakowy sposób od początku aż do końca.

\$ORI_TYPE=#CONST - oznacz, że podczas ruchu liniowego orientacja pozostaje stała. Orientacja zaprogramowana dla punktu końcowego zostaje zignorowana a użyta zostaje orientacja punktu początkowego.

UWAGA:

Jeśli podczas ruchu LIN kąt osi 3 albo kąt osi 5 musi zmienić znak to orientacja może zostać zachowana tylko, jeśli osie będą nieskończenie szybko się poruszać. Ponieważ nie jest to możliwe system sterowania po przekroczeniu wartości granicznych motoru dla ruchu zostanie przerwany z komunikatem błędu.

Wyglądanie:

Ponieważ obliczanie punktów pomocniczych jest niepotrzebne i zabiera czas, to można w określonym odstępie od pozycji docelowej rozpocząć wygładzanie toru ruchu.

Programowanie wygładzania następuje w dwóch krokach:

1. Definicja obszaru wygładzania za pomocą zmiennej systemowej \$APO:

\$APO.CDIS - jest to kryterium odległościowe przesunięcia (jest aktywowane przez CDIS): kontur wygładzania zostanie zapoczątkowany w określonym odstępie (mm) od punktu docelowego.

\$APO.CORI - jest to odległość orientacyjna (jest aktywowana przez CORI): pojedynczy kontur zostanie opuszczony, kiedy dominujący kąt dystansu do punku docelowego zostanie przekroczony.

\$APO.CVEL - jest to kryterium prędkości (jest aktywowana przez C_VEL): po osiągnięciu pewnego procentu zdefiniowanego \$APO.CVEL i prędkości zdefiniowanej w \$VEL.CP, kontur wygładzenia zostanie wprowadzony.

2. Programowanie funkcji ruchu z definiowaniem pozycji docelowej i rodzaju wygładzenia:

- LIN-LIN względnie LIN-CIRC

Przy wygładzaniu liniowo-liniowym system oblicza tor paraboliczny, przy wygładzaniu liniowo-kołowym nie można wyliczyć żadnego symetrycznego konturu wygładzania. Kontur wygładzania składa się z 2 parabolicznych segmentów, które wzajemnie się przenikają i jednocześnie dążą tangensoidalnie do pojedynczych zdań. W celu ustalenia początku wygładzania należy zaprogramować korzystając z jednego ze słów kluczowych: C_DIS, C_ORI lub C_VEL.

- LIN-PTP

Założeniem tego rodzaju wygładzania jest to, że żadna z osi robota w komendzie LIN nie obróciła się bardziej niż o 180° i że S się nie zmienia. Początek wygładzania jest ustawiony za pomocą zmiennych \$APO.CDIS, \$APO.CORI i \$APO.CVEL. Natomiast to, co jest wygładzane jest ustawiane za pomocą zmiennej \$APO.CPTP. W komendzie LIN należy użyć jednego ze słów kluczowych CDIS, CORI i C_VEL.

Przykład:

1.

Ruch liniowy z wyuczonymi współrzędnymi celu.

LIN!

2.

Ruch liniowy z zaprogramowanymi współrzędnymi celu, wygładzanie jest aktywowane.

LIN PUNKT1 CDIS

3.

Podanie pozycji docelowej we współrzędnych kartezjańskich.

LIN {X 12.3,Y 100.0,Z -505.3,A 9.2,B -50.5,C 20}

4.

Podanie tylko 2 wartości pozycji docelowej, pozostałe parametry zachowują bieżące wartości.

LIN {Z 500,X 123.6}

5.

Podanie pozycji docelowej za pomocą operatorów geometrycznych: uzyskane jest poprzez odjęcie 30.5 mm w osi X i dodania 20 mm w osi Z systemu współrzędnych TOOL dla punktu 1, który jest opisany we współrzędnych BASE.

LIN PUNKT1: {z -30.5,Z 20}

6.

Wygładzanie LIN-PTP z punktu 2 do punktu 3. Wygładzanie rozpoczyna się 30mm od punktu 2.

\$APO.CDIS=30
\$APO.CPTP=20 PTP
PUNKT1 LIN
PUNKT2 CDIS PTP
PUNKT3

LIN REL

Ruch liniowy ze współrzędnymi względnymi.

Argumenty:

Pozycja docelowa (Zielposition) - jest to wyrażenie geometryczne opisujące punkt końcowy. Dopuszczalne są tylko współrzędne kartezjańskie i są one podawane względem aktualnej pozycji. Pozycja docelowa nie może zostać wyuczona. Przesunięcia odbywają się zgodnie z osiami współrzędnych. Jeśli pozycja docelowa posiada niezdefiniowane części struktury to będą one miały wartość 0, czyli wartości absolutnie nie zmienia się. Działanie zaprogramowanych komponentów orientacji definiuje predefiniowana zmienna \$ROTSYS. Statusy kątów S i T zdefiniowane wewnątrz pozycji docelowej zostają zignorowane.

Opis:

Komenda LINREL działa tak samo jak komenda LIN. Jediną różnicą jest obliczanie współrzędnych, które są obliczane względem aktualnej pozycji a nie pewnego zdefiniowanego punktu w przestrzeni i układu współrzędnych. Wszystkie zasady opisane LIN zachowują tutaj ważność.

Przykład:

1.

Robot rusza się z aktualnej pozycji o 100mm wzdłuż osi X i o 200mm wzdłuż osi Z w kierunku ujemnym. Y,A,B,C,S i T pozostają stale.

```
LINREL {X 100,Z -200}
```

2.

Wygładzanie LIN-LIN od punktu 1 do punktu 2 i wygładzanie LIN-CIRC od punktu 2 do punktu 3. Wygładzanie od punktu 1 zacznie się, kiedy prędkość zwiększy się do 0.3m/s (30% z 0.9m/s). Wygładzanie od punktu 2 zacznie się 20mm przed tym punktem.

```
$VEL.CP=0.9 $APO.CVEL=30  
$APO.CDIS=20 LIN PUNKT1  
C_VEL LINREL PUNKT2REL  
CDIS CIRC  
HILFSPUNKT,PUNKT3
```

LOOP

Pętla nieskończona.

Opis:

Za pomocą pętli LOOP uzyskuje się cykliczne wykonywanie komend zdefiniowanych wewnątrz pętli. Za pomocą instrukcji EXIT wychodzi się z pętli.

Przykład:

1.

Pętla nieskończona.

```
LOOP
    A=A+1
    IF A==65 THEN
        EXIT
    ENDIF
ENDLOOP
```


PTP

Ruch od punktu do punktu.

Argumenty:

Pozycja docelowa (Zielposition) - jest to wyrażenie geometryczne opisujące punkt końcowy, są tu dopuszczalne współrzędne kartezjańskie albo współrzędne specyficzne dla osi. Pozycja docelowa może być wyuczona wtedy nie podajemy wartości tylko „!”. Układ odniesienia dla pozycji docelowej opisanej zmiennymi kartezjańskimi jest definiowany zmienna systemowa \$BASE. Jeśli pozycja docelowa zawiera nie zdefiniowane komponenty to ich wartości zostaną wzięte z aktualnej pozycji. CPTP - oznacza wygładzenie danego punktu docelowego.

Opis:

Ruch od punktu do punktu jest najszybszą możliwością przesunięcia ramienia robota z aktualnej pozycji do innej zaprogramowanej pozycji. Obrót wzdłuż osi przebiega synchronicznie tzn., że wszystkie osie zaczynają i kończą ruch jednocześnie. System sterowania oblicza prędkość każdej osi tak, że przynajmniej jedna os przekroczy predefiniowana wartość graniczną prędkości i przyspieszenia. Maksymalna prędkość i maksymalne przyspieszenie musi zostać zaprogramowane osobno dla każdej osi. Do tego należy użyć zmiennych systemowych:

- \$VEL_AXIS[Nr] dla prędkości
- \$ACC_AXIS[Nr] dla przyspieszenia

Wszystkie te parametry podaje się w procentach. Procenty te odnoszą się do zdefiniowanych w danych maszyny maksimum.

UWAGA:

W przypadku, jeśli obie te zmienne systemowe nie zostaną zaprogramowane na początku ruchu to pojawi się komunikat błędu podczas wykonywania programu. To samo dotyczy zmiennych systemowych \$TOOL i \$BASE w momencie, jeśli pozycja docelowa jest podana we współrzędnych kartezjańskich.

Status kąta:

Z powodu kinematyczno-pojedynczego robot może jedną i tą samą pozycję w przestrzeni przyjąć przy różnych ustawieniach kątów osi. Za pomocą ustawień S (Status) i T (Obrót) opisanej geometrycznie jesteśmy w stanie zdefiniować jednoznacznie ustawienie kąta osi. Oba te parametry wymagają danych całkowitych w formie binarnej. Bity oznaczają następujące rzeczy:

Status: Bit 1: Pozycja punktu nadgarstka (0 obszar podstawowy, 1 obszar nad głową)

Bit 2: Ustawienie kąta osi 3 (0 ujemny, 1 dodatni) Bit

3: Ustawienie kąta osi 5 (0 dodatni, 1 ujemny)

Obrót: Bit x: Ustawienie Kąta osi x (0 dodatni, 1 ujemny)

Ustawienie kąta jest rozważane każdorazowo dla każdej osi od pewnego zdefiniowanego położenia zerowego.

UWAGA:

Jeśli brakuje parametrów S i T przy komendzie PTP robot używa najkrótszej drogi. Aby zagwarantować identyczny ruch za każdym razem pierwsza komenda ruchu programu powinna być komendą PTP z podaniem S i T.

Wyglądanie:

Ponieważ obliczanie punktów pomocniczych jest niepotrzebne i zabiera czas, to można w określonym odstępie od pozycji docelowej rozpocząć wyglądanie toru ruchu.

Programowanie wyglądania następuje w dwóch krokach:

1. Definicja obszaru wyglądania za pomocą zmiennej systemowej \$APO:

\$APO.CPTP - jest to osiowe kryterium wyglądania (aktywowane przez CPTP): oznacza, że wyglądanie zacznie się, kiedy prowadząca os przekroczy zdefiniowanej zmiennej \$APO.CPTP procent zdefiniowanego w \$APO_DIS_PTP[Nr] pewnego ustalonego maksymalnego kąta.

\$APO.CDIS - jest to kryterium odległościowe przesunięcia (jest aktywowane przez CDIS): kontur wyglądania zostanie zapoczątkowany w określonym odstępie (mm) od punktu pośredniego.

\$APO.CORI - jest to odległość orientacyjna (jest aktywowana przez CORI): pojedynczy kontur zostanie opuszczony, kiedy dominujący kąt dystansu do punktu pośredniego zostanie przekroczony.

\$APO.CVEL - jest to kryterium prędkości (jest aktywowana przez C_VEL): po osiągnięciu pewnego procentu zdefiniowanego \$APO.CVEL i prędkości zdefiniowanej w \$VEL.CP, kontur wygładzenia zostanie wprowadzony.

2. Programowanie komend ruchu z pozycją docelową i rodzajem wygładzania.

- PTP-PTP

Należy zaprogramować komendy PTP ze słowem kluczowym CPTP, które zawiera pozycję docelową wygładzanego ruchu. Obszar wygładzania ustala się poprzez zmienną \$APO.CPTP. Wygładzanie zaczyna się, kiedy ostatnia osi przekroczy pewien kąt do położenia docelowego. Kontur wygładzania opisuje parabolę w przestrzeni. Będzie ona obliczona przez system sterowania a programista nie ma żadnego wpływu na jej formę. Programowanie pozwala ustalić jedynie początek i koniec wygładzania.

- PTP-LIN względnie PTP-CIRC

Założeniem tego wygładzania jest, że żadna z osi robota nie przekroczyła w komendach LIN i CIRC więcej niż 180° i parametr S się nie zmienia. Początek wygładzania jest ustalany przez zmienną \$APO.CPTP. Koniec wygładzania jest ustalany poprzez zmienne \$APO.CDIS, \$APO.CORI i \$APO.CVEL. W komendzie PTP programuje się tylko słowo kluczowe CPTP i aby ustalić wygładzenie CDIS (wartość domyślna), CORI lub CVEL.

UWAGA:

Aby umożliwić wygładzanie musi być pozostawione swobodne działanie komputera. Jeśli to nie jest możliwe pojawi się komunikat, że wygładzenie jest niemożliwe.

Przykład:

1.

Ruch PTP z wyuczonymi współrzędnymi celu.

PTP! 2. Ruch PTP z zaprogramowanymi współrzędnymi celu, włączono wygładzanie.

PTP PUNKT1 CPTP

3.

Podanie pozycji docelowej w kartezjańskich współrzędnych bazowych (BASE).

PTP {X 12.3, Y 100.0, Z 50, A 9.2, B 50, C 0, S 'B010', T 'B1010'}

4.

Podanie pozycji docelowej we współrzędnych specyficznych dla osi.

PTP {A1 10,A2 -80.6,A3 -50,A4 0,A5 14.2, A6 0}

5.

Podanie tylko 2 wartości pozycji docelowej. Pozostałe wartości pozostaną takie, jakie są.

PTP {Z 500, X 123.6}

6.

Podanie pozycji docelowej z pomocą operatorów geometrycznych: obliczona zostanie poprzez dodanie 100mm do osi X we współrzędnych systemu TOOL do punktu 1, który jest opisany zmiennymi BASE.

PTPPUNKT1:{X100}

7.

Wygładzanie punktowo liniowe z punktu 2 do punktu 3. Wygładzanie zaczyna się, jeśli osi kierującej pozostanie kąt mniejszy niż 20% jej maksimum zdefiniowanego w \$APO_DIS_PTP[Nr] do punktu 2.

\$APOP.CORI=10

\$APO.CPTP=20

PTP PUNKT1

PTP PUNKT2 CPTP CORI

LIN PUNKT3

PTP REL

Ruch z punktu do punktu we współrzędnych względnych.

Argumenty:

Pozycja docelowy (Zielposition) - jest wyrażeniem geometrycznym opisującym punkt końcowy. Można tu używać współrzędnych kartezjańskich jak i współrzędnych specyficznych dla osi. Są one względne, czyli w stosunku do aktualnej pozycji. Pozycja ta nie może zostać wyuczona. Odległości są liczone w kierunku zgodnym z osiami w systemie współrzędnych wpisanych w zmienną \$BASE. Jeśli pozycja docelowa zawiera niezdefiniowane komponenty to dostaną one wartość 0, czyli absolutne wartości nie zmienia się. Działanie zaprogramowanych komponentów orientacji ustala predefiniowana zmienna \$ROTSYS.

CPTP - jest to słowo kluczowe i oznacza, że wygładzanie będzie aktywne. Rodzaj wygładzania podaje się po tym słowie kluczowym: może to być CDIS, CORI, C_VEL.

Opis:

Komenda PTPREL działa podobnie jak komenda PTP z tą różnicą, że współrzędne celu są liczone względem pozycji bieżącej.

Przykład:

1.

Oś druga zostanie obrócona o 30° w kierunku ujemnym. Wszystkie inne osie zostają nieruchomo.

PTPREL {A2 -30}

2.

Robot porusza się z aktualnej pozycji o 100mm w kierunku osi X i o 200mm w kierunku osi Z z ujemną wartością. Y,A,B,C,S i T pozostają bez zmian.

PTPREL {X 100, Z -200}

3.

Wygładzanie PTP-PTP z punktu 1 do punktu 2 i wygładzanie PTP-CIRC z punktu 2 do punktu 3. Wygładzanie zaczyna się, gdy osi prowadzącej pozostanie mniej niż 40% kąta zdefiniowanego w \$APO_DIS_PTP[Nr], do punktu 1 względnie punktu 2.

```
$APO.CDIS=30 $APO.CPTP=40  
PTP PUNKT1 CPTP PTPREL  
PUNKT2REL CPTP CIRC  
HILFSPUNKT, PUNKT3
```

PULSE

Aktywowanie impulsu wyjściowego.

Argumenty:

Sygnal - jest typu BOOL. Dopuszczalne wartości to:

- OUT[Nr]
- Zmienne określające wygnal

Poziom (Pegel) - jest równy:

- TRUE, kiedy impuls ma wartość 1 (wysoki (high))
- FALSE, kiedy impuls ma wartość 0 (niski (low))

Czas trwania impulsu (Impulsdauer) - jest typu REAL. Wyrażenie to arytmetyczne, które opisuje długość trwania impulsu. Zakres wartości rozciąga się od 0.1 do 3 sekund.

Opis:

Komenda PULSE służy do aktywacji impulsu wyjściowego. Podczas wykonywania tej komendy wyjście binarne przez pewien ustalony czas będzie miało zdefiniowaną wartość. Po upływie jednostki czasu sygnał wyjściowy zostanie automatycznie zmieniony przez system. Ustawianie i kasowanie sygnału wyjściowego następuje niezależnie od poprzedniego stanu wyjścia. Jeśli podczas jednego impulsu pojawi się sygnał przeciwny to impuls zostanie skrócony. Jeżeli wyjście jest opadające i zostanie ponownie aktywowane to czas trwania impulsu rozpoczyna się od nowa. Podczas gdy impuls wyjściowy ma dodatnią wartość to wyjście binarne jest ustawiane na TRUE, jeżeli impuls wyjściowy ma opadające zbocze to wyjście binarne jest ustawione na FALSE. Opóźnienie czasowe wynosi około 1% - 2%. Dla bardzo krótkich impulsów opóźnienie wynosi około 13%.

Wskazówka:

- czasy impulsów poza dopuszczalnym interwałem będą rozpoznawane tylko w czasie programu i powodują zatrzymanie programu;
- można jednocześnie zaprogramować max. 16 impulsów wyjściowych;
- przy zatrzymaniu procesu czas impulsu dalej biegnie;
- komendy RESET i CANCEL kasują, przerywają trwanie impuls;
- trwający impuls może być zmieniany przez komendy przerwań;
- komenda PULSE powoduje zatrzymanie działania programu, nie działa to w komendzie TRIGGER;
- jeśli program podczas trwania aktywnego impulsu osiągnie komendę END to impuls nie zostanie przerwany;

UWAGA:

Awaryjne wyłączenie, stop z błędem i stop obsługi nie zatrzymują impulsu.

Przykład:

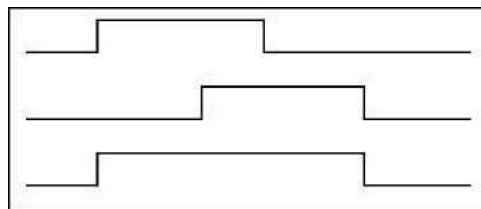
1.

Jeśli wyjście impulsu przy zboczu opadającym jest ponownie aktywowane to czas trwania impulsu jest mierzony od nowa.

```
PULSE ($OUT[50], TRUE, 0.5)
```

```
PUSLE ($OUT[50], TRUE, 0.5)
```

Ausgang 50



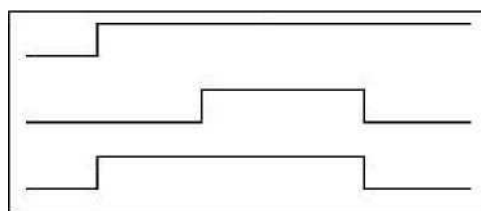
2.

Jeśli wyjście impulsu jest ustawione to przez zbocze opadające zostanie usunięte.

```
$OUT[50] = TRUE
```

```
PULSE ($OUT[50], TRUE, 0.5)
```

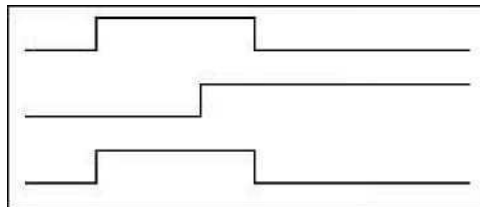
Ausgang 50



3.

W czasie trwania impulsu identyczne wyjście zostanie ustawione to zbocze opadające usunie impuls.

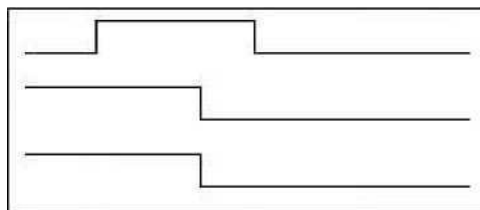
```
PULSE ($OUT[50], TRUE, 0.5)  
$OUT[50] = TRUE Ausgang 50
```



4.

Jeśli podczas trwania impulsu zostanie ustawione wyjście 50(Ausgang 50) to skróci się czas trwania impulsu.

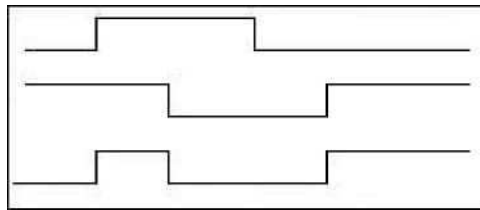
```
PULSE ($OUT[50], TRUE, 0.5)  
$OUT[50]=FALSE Ausgang 50
```



5.

Jeśli podczas trwania impulsu o dodatniej wartości pojawi się impuls ujemny to wyjście zostanie ustawione na ujemną wartość, a na dodatnią jak tylko ta ujemna zniknie. Wszystkie impulsy muszą być dodatnie żeby wyjście też było dodatnie.

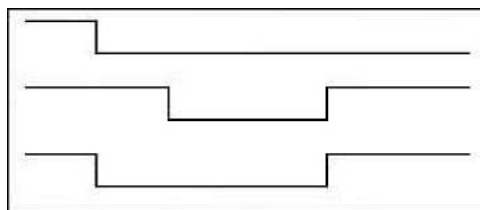
```
PULSE($OUT[50],TRUE,0.5)  
PULSE($OUT[50],FALSE,0.5)  
Ausgang 50
```



6.

Jeśli wyjście jest ustawione na Low i zostanie potraktowane ujemnym impulsem, to wyjście zostaje do końca taktu na Low a potem ustawia się na High.

```
$OUT[50] = FALSE  
PULSE($OUT[50],FALSE,0.5)  
Ausgang 50
```



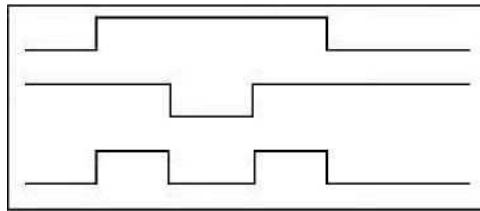
7.

Jeśli wyjście podczas trwania impulsu zostało ustawione na Low i z powrotem ustawione na to, co było poprzednio to po skończeniu impulsu znowu zostanie ustawione na Low.

```
PULSE($OUT[50],TRUE,0.8)
```

Ausgangsmanipulation

Ausgang 50



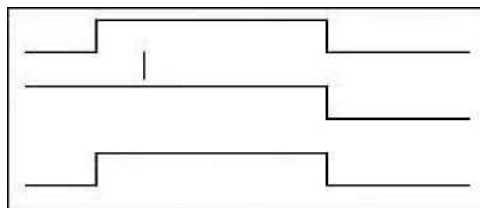
8.

Jeśli komenda END napotka na moment, kiedy impuls jest zaprogramowany to czas działania programu wydłuża się aż do czasu trwania impulsu.

```
PULSE($OUT[50],TRUE,0.8)
```

END Anweisung Programm

aktiv Ausgang 50



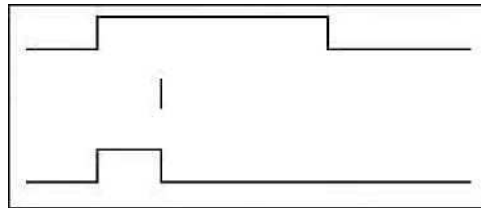
9.

Jeśli program, podczas gdy wyjście impulsu jest aktywne zostanie przerwany przez komendy RESET albo CANCEL to impuls zostanie niezwłocznie usunięty.

PULSE(\$OUT[50],TRUE,0.8)

RESET oder CANCEL

Ausgang 50



REPEAT...UNTTL

Jest to pętla, która jest powtarzana dopóki warunek nie jest spełniony.

Przykład:

1.

Pętla zostanie wykonana 100 razy. Po ostatnim wykonaniu R ma wartość 101.

```
R=1
REPEAT
    R=R+1
UNTIL R>100
```

2.

Pętla będzie wykonywana tak długo dopóki \$IN[1] nie będzie prawdą.

```
REPEAT
    Anweisungen UNTIL
$IN[1]==TRUE
```

3.

Pętla wykona się jeden raz mimo, iż warunek jest już spełniony, ponieważ najpierw wykonywane są komendy w pętli a później dopiero jest obliczany warunek. R po zakończeniu pętli będzie miało wartość 102.

```
R=101
REPEAT
    R=R+1
UNTIL R>100
```

RESUME

Jest to zakończenie wszystkich podprogramów i procedur obsługi przerwań.

Opis:

Komenda ta jest wywoływana tylko podczas obsługi przerwań. Dlatego dopuszczalna jest tylko w podmodule, który obsługuje dane przerwanie. Komenda RESUME kończy wywołanie, obsługę wszystkich obecnie wykonujących komend obsługi przerwań i wszystkich podprogramów, aż do poziomu, w którym zostało zadeklarowane przerwanie, z którego ta komenda jest wywoływana.

UWAGA:

W momencie wykonania komendy RESUME zmienna \$ADVANCE=0 (nie może być żadnego przebiegu programu). Wskaźnik programu nie może być na tym samym poziomie, na którym zostało zadeklarowane przerwanie, musi on być przynajmniej o jeden poziom niżej. Jeżeli to nie będzie spełnione to nastąpi przerwa w programie i będzie konieczny zresetowanie programu.

Wskazówka:

- ruch po komendzie RESUME nie powinien być żadnym ruchem kołowym, gdyż punkt początkowy jest inny i wyjdą inne koła;
- zmiany w zmiennej \$BASE są ważne tylko w danej procedurze obsługi przerwania, nie będą widoczne na zewnątrz;
- używanie zmiennej \$ADVANCE w procedurze obsługi przerwania jest niedozwolone;

Przykład:

1.

Robot na zaprogramowanym z góry torze powinien szukać części. Część powinna zostać rozpoznana sensorem na wyjściu 15. Po znalezieniu części robot nie powinien iść do punktu końcowego toru, a powinien wrócić do pozycji przerwania i podnieść część. Po czym zanieść ją do punktu rozładunkowego.

DEF PROG() ;Program główny

INTERRUPT DECL 1 WHEN \$IN[15] DO GEFUNDEN()

SUCHEN() ;Trasa poszukiwań musi być
zaprogramowana w podprogramie

LIN ABLAGEPUNKT

END

DEF SUCHEN() ;Podprogram szukania części
LIN ANFANGSPUNKT CDIS
LIN ENDPUNKT
\$ADVANCE=O ;Żaden przebieg nie jest dopuszczany
END

DEF GEFUNDENO ;Procedura przerwania
INTERRUPT OFF ;Aby procedura przerwania nie została
wywołana dwukrotnie

BRAKE
LIN \$POS_INT ;Droga powrotna do punktu, w którym
nastąpiło przerwanie

RESUME ;Opuszczenie toru poszukiwań
END

RETURN

Jest to wyjście z funkcji podprogramów.

Opis:

RETURN jest używany w funkcjach i podprogramach. Używany w funkcji musi zwracać jakąś wartość, natomiast w podprogramie nie.

Przykład:

1.

```
RETURN 0 2.
```

```
RETURN (X*3.1415)/360
```

3.

```
DEFFCTINT X
    X=10
    RETURN X
ENDFCT
```

4.

```
DEF PR0G_2()
    Vereinbarungen
    Anweisungen
RETURN
END
```


SIGNAL

Jest to skojarzenie nazwy sygnału z sygnałem wejściowym lub wyjściowym.

Argumenty:

Nazwa sygnału (Signalname) - podajemy dowolną nazwę;

Nazwa interfejsu (Schnittstellename) - możliwe typy, jakie mogą wystąpić:

- \$IN[Nr] binarne wejście
- \$OUT[Nr] binarne wyjście
- \$DIGIN[Nr] cyfrowe wejście
- \$ANIN[Nr] analogowe wejście
- \$ANOUT[Nr] analogowe wyjście

Opis:

System sterowania robotem ma dwie klasy interfejsu:

1. Proste interfejsy procesowe (sygnały)
2. Interfejsy logiczne (kanały)

Wszystkie interfejsy są wywoływane za pomocą symbolicznych nazw. Komenda SIGNAL łączy nazwy symboliczne z predefiniowanymi zmiennymi sygnałowymi. Komendy SIGNAL muszą się znajdować w części deklaracyjnej. Każde wyjście może występować w więcej niż jednej komendzie SIGNAL. Liczba sygnałów odpowiada liczbie wejść/wyjść systemu sterowania. Należy rozróżnić między wejściem/wyjściem binarnym, a cyfrowym. Mówiąc binarnym mamy na myśli pojedyncze wejście/wyjście, natomiast mówiąc cyfrowe rozumiemy to jako zbiór pojedynczych wejść/wyjść.

Komenda SIGNAL i opcja TO można zgrupować razem więcej niż jedno binarne wejście/wyjście do jednego cyfrowego wejścia/wyjścia. Takie złożone sygnały mogą być dziesiętne, hexadecymalne (prefix H) albo bitowe (prefix B). Na sygnałach można operować operatorami boolowskimi. Max. można 32 sygnały binarne zgrupować w jeden cyfrowy sygnał.

Aby to było możliwe wszystkie predefiniowane nazwy sygnałów muszą być binarne a ich indeksy muszą tworzyć rosnący zakres, czyli nie może być żadnych luk pomiędzy indeksami poszczególnych sygnałów składanych w jedną grupę. Max. można złożyć razem 32 wejścia/wyjścia bitowe. W systemie sterowania robotem są 32 wejścia i 32 wyjścia. Wyjścia od 1 do 28 mają natężenie 100mA, wyjścia od 29 do 32 mają natężenie 2A. Niewykorzystane wyjścia mogą służyć jako markery.

Przykład:

1.

Symbolicznej nazwie schalter zostało przyporządkowane wyjście binarne \$OUT[7]

SIGNAL SCHALTER \$OUT[7]

2.

Binarne wejścia \$IN[1] do \$IN[8] zostają przypisane symbolicznej nazwie INWORT i zgrupowane w wejście cyfrowe.

SIGNAL INWORT \$IN[1] TO \$IN[8]

3.

Binarne wyjścia \$OUT[1] do \$OUT[8] zostają zgrupowane w jedno cyfrowe wyjście o nazwie OUTWORT. Następnie sygnałom \$OUT[3], \$OUT[4], \$OUT[5] i \$OUT[7] zostaje przypisana wartość jeden.

SIGNAL OUTWORT \$ OUT[1] TO \$ OUT [8]

OUTWORT = 'B01011100'

STRUĆ

Jest to komenda deklarowania typów struktur.

Opis:

Typ strukturalny jest kompleksowym typem danych, który łączy takie same lub różne typy danych. Ważne predefiniowane typy strukturalne to: AXIS, FRAME, POS, E6POS i E6AXIS. Nie wolno tych nazw wykorzystywać jako nazw własnych typów struktur. Ważne jest, aby najpierw zdefiniować strukturę a następnie definiować zmienne typu danej struktury, a nie na odwrót.

WAŻNE:

- predefiniowana lista danych \$CONFIG jest umieszczona przed lokalnie definiowanymi listami danych.
- lokalna lista danych jest umieszczona przed należącym do niej podmodułem.

DOSTĘP DO KOMPONENTÓW STRUKTUR:

Do komponentów struktur dostęp jest swobodny. Aby dostać się do komponentu należy podać:

nazwa struktury, nazwa zmiennej

Nazwa zmiennej, jaką podajemy jest to zmienna, do której chcemy się odnosić. Gdy struktura zawiera podstruktury to, aby się do nich odwołać należy podać:

nazwa struktury, nazwa podstruktury, nazwa komponentu

DOSTĘP DO WARTOŚCI DANYCH W STRUKTURACH:

Do wartości danych w strukturach dostęp także jest swobodny. Aby uzyskać dostęp do więcej niż jednego albo wszystkich elementów struktury należy użyć agregatów.

Przykład:

1.

Deklaracja typu strukturalnego S1TYP z komponentami STROM(napięcie), SPANNUNG(natężenie), VORSCHUB, które są typu REAL.

STRUĆ S1TYP REAL STROM, SPANNUNG, VORSCHUB

2.

Deklaracja typu strukturalnego S2TYP z komponentami STROM(napięcie) typu REAL i tablicą TEXT[80] typu CHAR.

```
STRUĆ S2TYP REAL STROM, CHAR TXY[80]
```

3.

W podprogramie do spawania powinny zostać przekazane następujące informacje w zmiennych:

- prędkość spawania
- linia, po której się spawa
- opcja z/bez światła (symulacja)

```
DEF PROG()  
STRUĆ STYP REAL DRAHT, INT KENNL, BOOL LIBO  
DECL STYP SPARAMETER  
S_PARAMETER.DRAHT=10.2; zainicjalizowanie pierwszego komponentu  
S_PARAMETER.KENNL=60  
SPARAMETER.LIBO TRUE  
SUP(SPARAMETER) ; wywołanie podprogramu
```

...; inicjalizacja z pomocą agregatu

```
S_PARAMETER= {DRAHT 7.3, KENNL 50, LIBO=TRUE}  
S_UP(S_PARAMETER) ; ponowne uruchomienie podprogramu
```

```
END
```

SWITCH...CASE

Jest to komenda wyboru.

Argumenty:

Kryterium wyboru (Auswahlkriterium) - kryterium musi być typem zwracającym, liczbą całkowitą albo typem wyliczeniowym. Blok identyfikacji (Blockkennung) - bloki CASE są podobnie jak kryterium wyboru typu INT, CHAR lub typu wyliczeniowego. Można definiować dowolnie wiele bloków, jeżeli zostanie użyty kilka razy ten sam blok to w takim przypadku będzie wywoływane jego pierwsze wystąpienie. Gdy używamy kilku wartości to należy pamiętać, aby oddzielać je przecinkami.

Opis:

Zasada działania komend SWITCH...CASE jest taka sama jak w innych językach programowania.

Bardzo ważna informacja, o której należy pamiętać to to, aby typ danych, kryterium wyboru i warunki znajdujące się w CASE były zgodne.

Pomiędzy komendą SWITCH, a pierwszym CASE nie może być żadnej wolnej linii ani komentarza.

Komenda DEFAULT może w ogóle nie wystąpić. Natomiast gdy się już pojawia to może się pojawić tylko jeden raz. Nie wolno wychodzi z komendy SWITCH poleceniem EXIT.

Przykład:

1.

Kryterium wyboru i blok identyfikacji są typu integer. Wykorzystywana jest komenda DEFAULT jako zwrócenie komunikatu błędu.

```
SWITCH VERSION
CASE1

CASE 2,3
    UP_2()
    UP_3()
    UP_3A()
DEFAULT
    ERROR_UP()
ENDSWITCH
```

2.

Kryterium wyboru i bloki identyfikacji są typu charakter. Funkcja UP_5() nie zostanie nigdy wywołana gdyż blok „HANS” istnieje dwa razy.

```
SWITCH NAME
  CASE „ALFRED”

  CASE „BERT”, "HANS
    UP_2()
    UP_3()
  CASE „HANS”
    UP_5()
ENDSWITCH
```

TRIGGER

Służy do wywołania pewnej akcji równoległej w czasie do pracy robota.

Argumenty:

Odległość (Strecke) - jest to wartość typu rzeczywistego. Może być zmienna lub stałą, przy czym:

- DISTANCE = 0 początek
- DISTANCE = 1 koniec

Wolno używać tylko tych dwóch wartości.

Czas (Zeit) - jest to zmienna typu całkowitego. Może być zmienną lub stałą, w przypadku, gdy będzie:

- dodatnia to komenda zostanie wywołana o tyle czasu później
- ujemna to komenda zostanie wywołana o tyle czasu wcześniej

Jednostką czasu wykorzystywana jest milisekunda.

Komenda (Anweisung) - może to być:

- funkcja podprogramu
- przypisanie do zmiennej
- komenda PULSE

Podprogram będzie obsługiwany tak jak procedura obsługi przerwania, dlatego przy podawaniu podprogramu jako komendy musi zostać użyte słowo kluczowe PRIO znak równości i priorytet operacji.

Priorytet - priorytet operacji podaje się od 1 do 128 przy czym priorytet 1 jest najwyższy.

Opis:

TRIGGER pozwala równolegle do ruchu robota wywoływać podprogram, albo przypisanie do zmiennej. Jest możliwe odwołanie w czasie wykonanie tej komendy przez opcje DELAY, albo też wykonać tą operację o pewien okres czasu wcześniej. Jednakże punkt przełączenia może być przesunięty tylko tak daleko, aby wciąż pozostawał wewnątrz tego samego zdania. Granica zdania jest automatycznie ograniczeniem tego przesunięcia. Dlatego można np. ustawić DISTANCE = 1 i DELAY ustawić na wartość negatywną (ujemną). Podanie pozytywnej (dodatniej) wartości DELAY spowodowałoby przekroczenie granicy.

Zdanie jest określeniem na pojedynczy ruch robota.

Przy wygładzaniu DISTANCE = 1 oznacza środek łuku wygładzenia, jeśli poprzednie zdanie jest już zdaniem wygładzenia to DISTANCE = 0 oznacza punkt końcowy poprzedniego łuku wygładzającego.

Przykłady:

1.

Włączenie 130 ms po starcie następnego ruchu włączenia ustawienia sygnału.

```
TRIGGER WHEN DISTANCE=0 DELAY=130 DO $OUT[8]=TRUE
```

2.

Na końcu następnego ruchu włączyć wywołanie podprogramu z Priorytetem 5.

```
TRIGGER WHEN DISTANCE=1 QUOTIENT(DIVIDEND,DIVISOR) PRIO=5
```

3.

Zakresy włączania przy różnych ruchach i operacjach opóźnienia.

```
DEF PROG()
```

```
PTP PUNKTO
```

```
TRIGGER WHEN DISTANCE=0 DELAY=40 DO A=12
```

```
;przejście 0 - 1
```

```
TRIGGER WHEN DISTANCE=1 DELAY=-20 DO UP1() PRIO=10
```

```
;przejście 0 - 1
```

```
LIN PUNKT1 TRIGGER WHEN DISTANCE=0 DELAY=10 DO  
UP2(A) PRIO=5
```

```
;przejście 1 - 2'B
```

```
TRIGGER WHEN DISTANCE=1 DELAY=15 DO B=1
```

```
;przejście 2'B - 2'E
```

```
LIN PUNKT2 CDIS TRIGGER WHEN DISTANCE=0 DELAY=10 DO  
UP2(B) PRIO=12
```

```
;przejście 2'E-3'B
```

```
TRIGGER WHEN DISTANCE=1 DO UP(A,B,C) PRIO=6
```

```
;przejście 3'B-3'E
```

```
LIN PUNKT3 CDIS TRIGGER WHEN DISTANCE=0 DELAY=50 DO  
UP2(A) PRIO=4
```

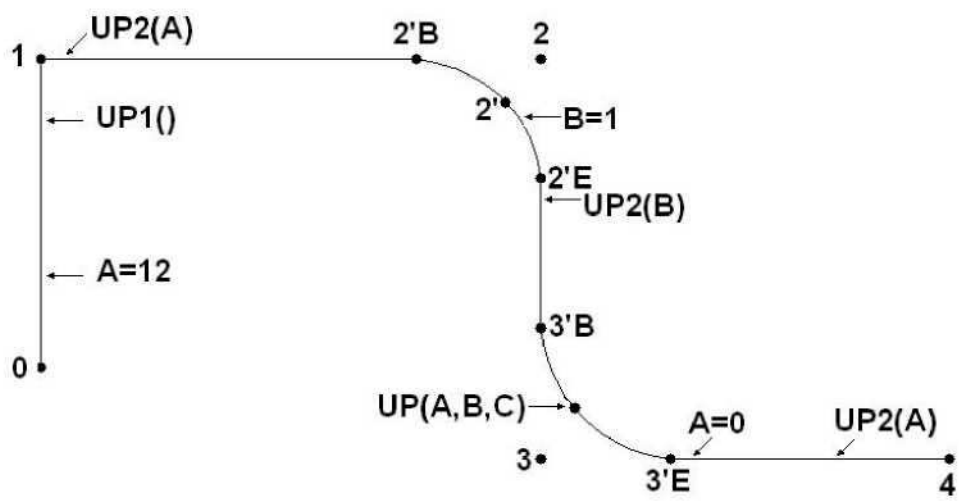
```
;przejście 3'E-4
```

```
TRIGGER WHEN DISTANCE=1 DELAY=-80 DO A=0
```

```
;przejście 3'E-4
```

```
LIN PUNKT4
```

```
END
```

WAIT FOR

Argumenty:

Wyrażenie logiczne fFortsetzungsbedingung) - jest typu BOOL. Robot oczekuje na wykonanie pewnego wyrażenia i dopiero wtedy może przejść dalej.

UWAGA:

Kompilator nie potrafi rozpoznać sytuacji, kiedy wyrażenie z powodu złego sformułowania nigdy nie przyjmie wartości TRUE. W takim przypadku program zostanie zatrzymany na zawsze, gdyż oczekuje na niemożliwy do spełnienia warunek.

Przykład:

1.

Przerwanie programu do momentu aż \$IN[17] będzie TRUE.

WAITFOR\$IN[17]

2.

Przerwanie programu do momentu aż BIT1 będzie FALSE.

WAIT FOR BIT1 == FALSE

WAIT SEC

Czas oczekiwania.

Opis:

Komenda ta zatrzymuje wykonywanie programu na określona ilość sekund. Maksymalna dopuszczalna wartość to $(2^{31}) \cdot \text{długość taktu}$.

Przykład:

1.
Zatrzymanie programu na 17.156 sekundy.

WAIT SEC 17.156

2.
Zatrzymanie programu na ilość sekund zapisaną w zmiennej V_ZEIT.

WAIT SEC V ZEIT

WHILE

Jest to pętla wykonująca się do póki jest spełniony warunek.

Opis:

Warunek jest sprawdzany przed wywołaniem pętli, dlatego jeśli jest spełniony od razu to pętla się nie wykona ani razu. Pętla wykona się, kiedy warunek ma wartość TRUE (jest spełniony). Przy wartości FALSE program kończy pętlę WHILE i kontynuuje działanie. Każda komenda WHILE jest zakończona komendą END WHILE.

Przykład:

1.

Pętla wykona się 99 razy, po wyjściu z pętli W ma wartość 100.

```
W=1
WHILE W<100
    W=W+1
ENDWHILE
```

2.

Pętla będzie się wykonywać tak długo jak długo \$IN[1] jest prawdą.

```
WHILE $IN[1]==TRUE
    Anweisungen
ENDWHILE
```

3.

Pętla nie wykona się ani razu. Poza pętlą W będzie miało wartość 100.

```
W=100 WHILE
W<100
    W=W+1
ENDWHILE
```