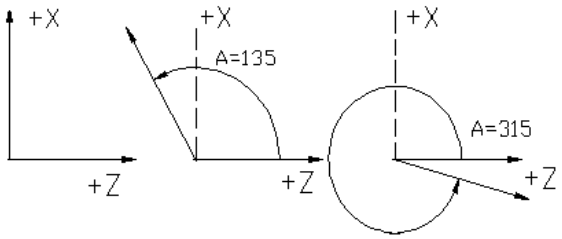
15. PROGRAMOWANIE Z WYKORZYSTANIEM CIĄGÓW KONTUROWYCH

Analizując sposób wymiarowania przedmiotów o symetrii osiowej można wyróżnić szereg typowych elementów geometrii zarysu. Wymiarowanie złożonych zarysów na rysunku konstrukcyjnym często nie dostarcza bezpośrednich danych do zapisu toru narzędzia. Obliczanie współrzędnych punktów charakterystycznych zarysu wydłuża czas opracowania programu oraz stanowi potencjalne źródło błędów. Wprowadzenie tzw. ciągów konturowych istotnie zmniejsza nakład pracy związany z obliczeniami geometrii zarysu przedmiotu i toru narzędzia. Ciąg konturowy stanowi połączenie kilku elementów geometrycznych typu odcinek lub łuk w konfiguracjach często występujących na zarysach przedmiotów. Do definiowania ciągu konturowego stosowane są słowa oznaczające współrzędne punktów (X, Z), promienie (B+), fazy (B-) oraz kąty (A). Sposób zapisu ciągu konturowego nawiązuje jak najściślej do typowych sposobów wymiarowania części maszyn. Ciągi konturowe można też wprowadzać do programu sterującego bezpośrednio z pulpitu układu CNC co w powiązaniu z analogiczną techniką stosowania cykli obróbkowych czyni zadość tzw. koncepcji programowania warsztatowego.

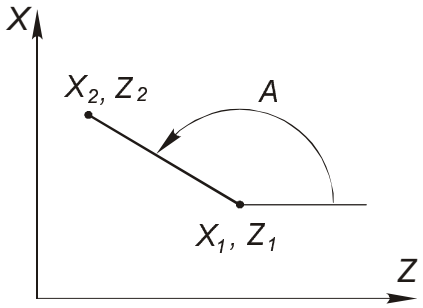
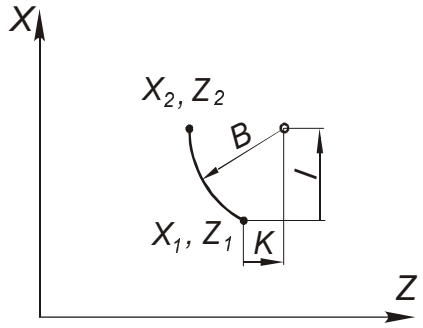
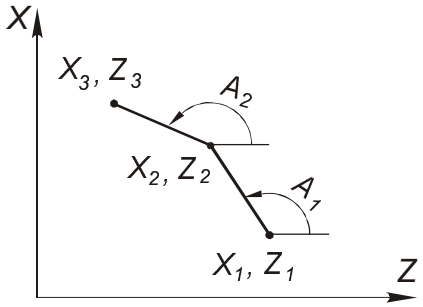
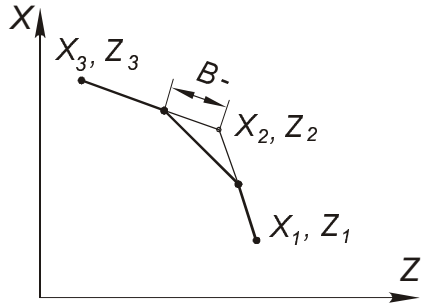
Kąty (adres A) w ciągach konturowych mogą zawierać się w zakresie 0.00001 - 359.99999 stopni. Dodatni kierunek kąta jest określany w prawoskrętnym układzie współrzędnych jako przeciwny do ruchu wskazówek zegara patrząc z wierzchołka trzeciej osi (rys. ...).

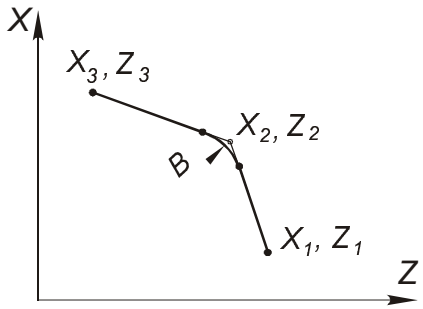
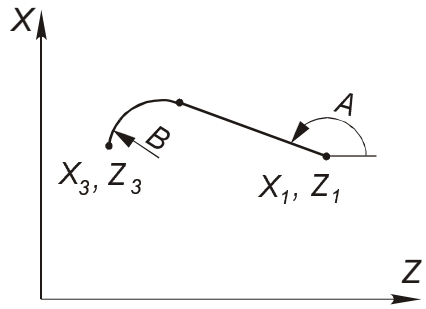
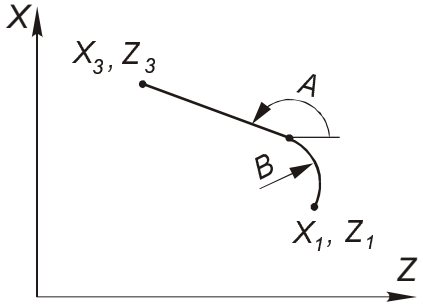
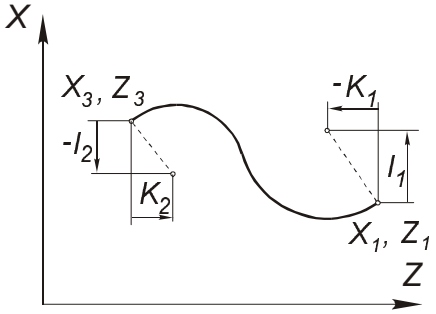


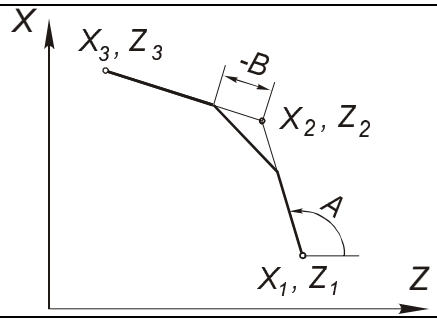
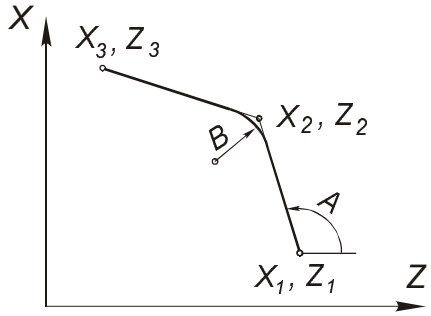
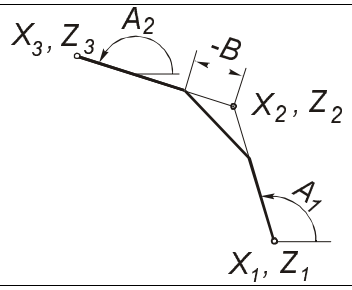
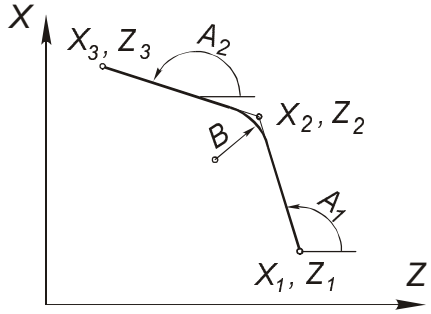
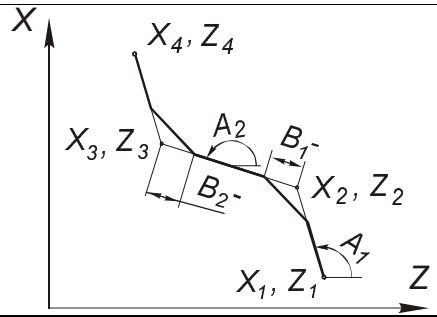
Rys. 16. Dodatni kierunek kątów przy programowaniu ciągów konturowych

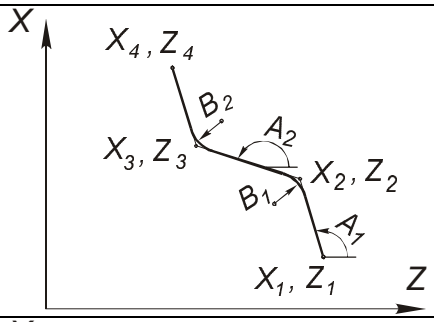
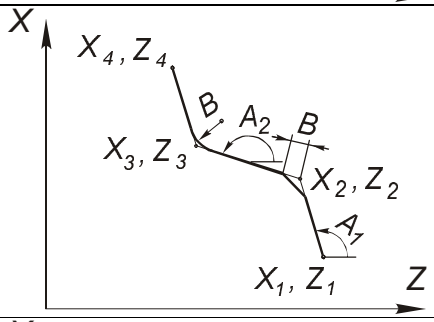
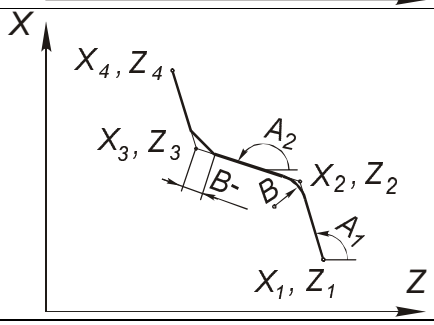
Współrzędne punktów w ciągach konturowych mogą być programowane w układzie bezwzględnym (G90) lub przyrostowym (G91). Ogólnie wybór kolejności dla adresów (A, X, Z, B i in.) w zapisie ciągów konturowych jest dowolny. W niektórych ciągach kolejność adresów jest jednak ważna co będzie wyraźnie podkreślone w ich opisach.

Tabela. ... Zestawienie ciągów konturowych

1. Ciąg dwupunktowy (odcinek) N...A...X_2...(lub Z_2...) W bloku zapisuje się kąt i jedną ze współrzędnych końca odcinka. Nieznana współrzędna końca odcinka jest obliczana przez układ sterowania.	
2. Łuk N...G02 (lub G03) I...K...B...X_2...(lub Z_2...) Programowany w tym ciągu łuk powinien leżeć w jednej ćwiartce układu współrzędnych. Nieznana współrzędna końca łuku jest obliczana przez układ sterowania. Obowiązkowy jest zapis obydwóch parametrów interpolacji (I..., K...) także wtedy, gdy któryś z nich jest równy zero.	
3. Ciąg trzypunktowy N...A_1...A_2...X_3...Z_3... Ciąg konturowy generuje dwa bloki programu sterującego, współrzędne punktu przecięcia się odcinków są obliczane przez układ sterowania. Kolejność zapisu kątów (A_1, A_2) jest interpretowana jak na rysunku. Kąt A_2 określa położenie drugiego odcinka.	
4. Faza na przejściu elementów N...X_2...Z_2...B-... N...X_3...Z_3... Zapis obejmuje dwa bloki. Zakończenie pierwszego bloku adresem B(-) oznacza wprowadzenie symetrycznej fazy przejściowej pomiędzy elementami zapisanymi w obydwu blokach. Znak minus po adresie B ma znaczenie umowne, przyjęte dla oznaczenia fazy. W drugim bloku można również zapisać ciąg konturowy.	

5. Promień na przejściu elementów N...X_2...Z_2...B... N...X_3...Z_3... Ciąg konturowy podobny do ciągu nr 4. Wprowadzony promień B (o umownym znaku +) nie może być większy od krótszego z dwóch odcinków. W drugim bloku można wprowadzić także element konturowy.	
6. Prosta - łuk styczny N...G02 (lub G03) A...B...X_3...Z_3... Kąt środkowy na którym jest oparty łuk nie może przekroczyć 180^0. Kolejność adresów A (kąt) i B (promień) musi być zachowana.	
7. Łuk kołowy - odcinek styczny N...G02 (lub G03) B.. A... X_3...Z_3... Kąt środkowy na którym jest oparty łuk nie może przekraczać 180^o. Kolejność B (promień) i A (kąt) musi być zachowana.	
8. Łuk - łuk styczny N...G02 (lub G03) I_1...K_1...I_2...K_2...X_3...Z_3... Funkcję przygotowawczą G02 lub G03 programuje się tylko dla pierwszego łuku. Dla drugiego łuku przyjmowany jest domyślnie przeciwny kierunek interpolacji kołowej. Parametry interpolacji drugiego łuku odnoszą się do punktu końcowego tego łuku. Należy programować obydwa parametry interpolacji nawet wtedy, gdy mają wartość zero.	

9. Ciąg dwupunktowy z fazą, złożenie wariantów (1)+(4) N...A...X_2...(lub Z_2...) B-... N...X_3...Z_3	
10. Ciąg dwupunktowy z promieniem, złożenie wariantów (1)+(5) N...A...X_2...(lub Z_2...) B... N...X_3...Z_3... Wprowadzony promień nie może być większy od krótszego z dwóch odcinków łączących się w punkcie (X_2, Z_2). W drugim bloku można wprowadzić także ciąg konturowy.	
11. Ciąg trzy punktowy z fazą wariant (3)+(4) N...A_1...A_2...X_3...Z_3...B-...	
12. Ciąg trzypunktowy z promieniem, wariant (3)+(5) N...A_1...A_2...X_3...Z_3...B... Jeżeli przejście do następnego ciągu konturowego odbywa się przez fazę lub promień, w bieżącym ciągu musi być zaprogramowany adres B nawet wtedy gdy promień (faza) jest równy 0 (patrz również wariant 3). W bloku wystąpią wówczas dwa adresy B, których kolejność musi być zgodna z kierunkiem obróbki.	
13. Ciąg trzypunktowy + faza + faza wariant (3)+(4)+(4) N...A_1...A_2...X_3...Z_3...B_1-...B_2-... N...X_4...Z_4... Wstawienie drugiej fazy w punkcie końcowym X_3, Z_3. Drugi blok może być także ciągiem konturowym.	

14. Ciąg trzypunktowy + promień + promień, złożenie wariantów (3)+(5)+(5) N...A_1...A_2...X_3...Z_3...B_1...B_2... N...X_4...Z_4... Wstawienie drugiego promienia w punkcie końcowym X_3, Z_3. Drugi blok może być także ciągiem konturowym.	
15. Ciąg trzypunktowy + faza + promień, złożenie wariantów (3)+(4)+(5) N...A_1...A_2...X_3...Z_3...B-...B... N...X_4...Z_4... Wstawienie promienia w punkcie końcowym X_3, Z_3.	
16. Ciąg trzypunktowy + promień + faza, złożenie wariantów (3)+(5)+(4) N...A_1...A_2...X_3...Z_3...B...B-... N...X_4...Z_4... Wstawienie fazy w punkcie końcowym X_3, Z_3. Drugi blok może być także ciągiem konturowym.	

Sposób działania funkcji G09, F, S, T, H, M13

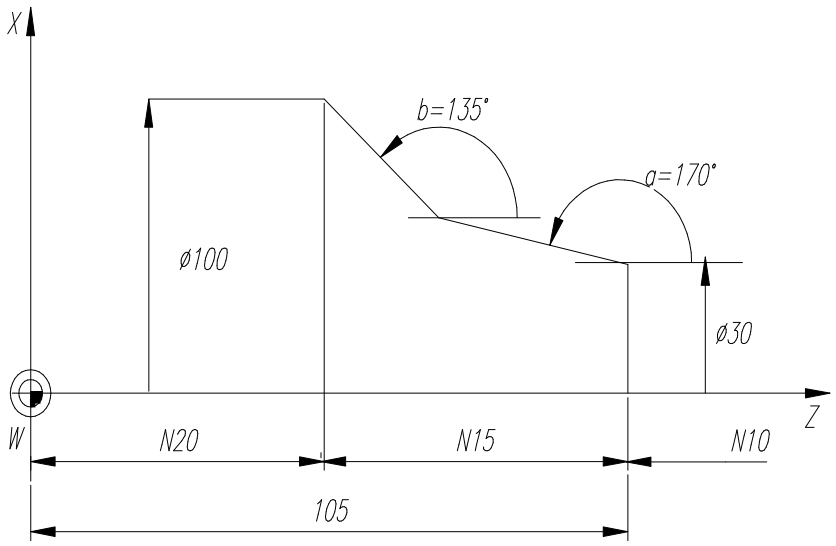
Jeżeli w ciągu opisującym kontur jest zaprogramowana funkcja G09, to działa ona dopiero na końcu bloku wraz z osiągnięciem końcowej pozycji
W obrębie konturu, w miejscach nieciągłości (naroża, krawędzie) układ sterowania automatycznie generuje G09.

- Jeżeli w ciągu opisującym kontur zostały zaprogramowane F, S, T, H to działają one na początku bloku.
- Jeżeli w ciągu opisującym kontur zostały zaprogramowane M00, M02, M17, M30 to działają one na końcu bloku.

Łańcuchy bloków, zawierające dane o kącie i w których dołączano promienie i fazy, można łączyć w dowolnej kolejności

Przykład 1

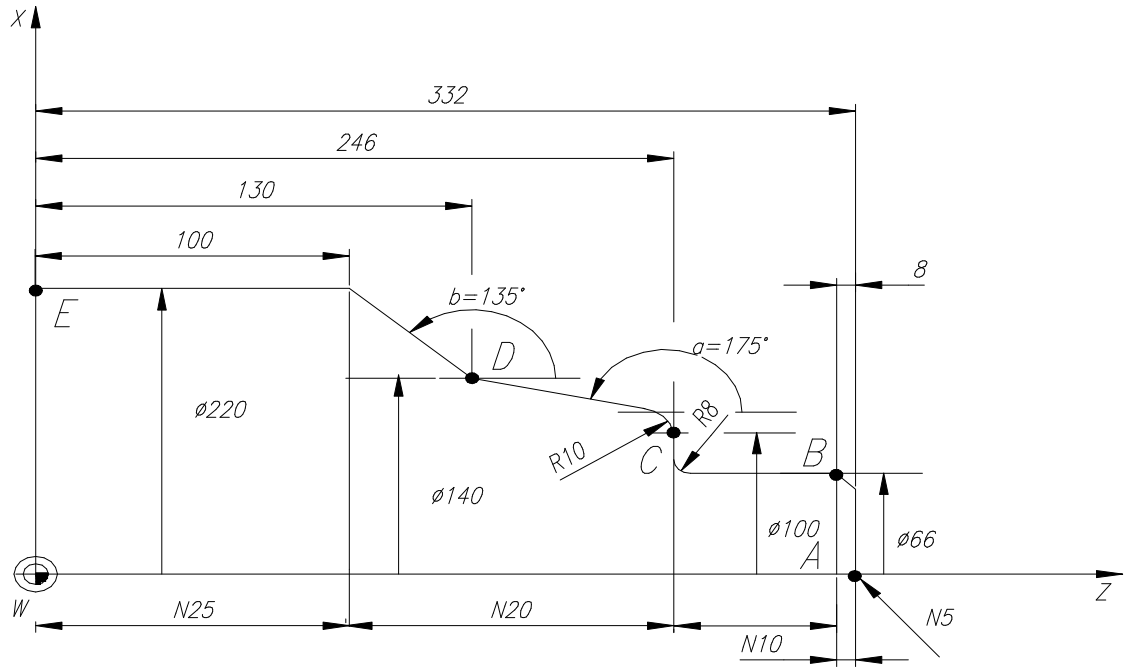
Kąt (a) odnosi się do punktu początkowego, kąt (b) do brakującego punktu oparcia. Należy podać obie współrzędne punktu końcowego. Układ sterowania określa na podstawie znanego punktu początkowego i końcowego oba kąty.



Rys. ... Przykład zastosowania ciągów konturowych

N10 G00 G90 X30 Z105
N15 G01 A170 A135 X100 Z20 F300

Przykład 2



Rys. ... Przykład zastosowania ciągów konturowych

Zapis ciągu konturowego w programie:

```
%200
N5 G00 G90 X0 Z332                (narzędzie wykonuje ruch do punkt A)
N10 G01 G09 A90 X33 B-8 F0.2      (fragment A-B)
N15 A180 A90 X50 Z246 B8          (fragment B-C)
N20 G03 B10 A175 X140 Z130        (fragment C-D)
N25 G01 A135 A180 X220 Z0         (fragment D-E)
N30 M30
```

16. PARAMETRYZACJA PROGRAMÓW

W programach sterujących CNC SINUMERIK 810T istnieje możliwość stosowania zmiennych, podobnie jak w językach programowania komputerów. Zmienne w programach sterujących nazywane są parametrami. Parametryzacja programów i podprogramów ułatwia projektowanie operacji dla części podobnych technologicznie oraz pozwala pisać programy w sposób zwięzły i uniwersalny. Parametr składa się z adresu R i maksymalnie trzycyfrowego numeru. Na parametrach można wykonywać działania matematyczne, można wykorzystywać je do tworzenia pętli, skoków i rozgałęzień. Parametrem można zastąpić dowolny kod w słowach bloku informacji programu sterującego (oprócz numeru bloku N i numeru programu %). Przy pisaniu programów, podprogramów i w cyklach obróbkowych wykorzystywane są parametry o numerach R00 do R99. Należy zwracać szczególną uwagę ażeby numery parametrów użyte we własnych fragmentach programu nie pokrywały się z numerami parametrów wykorzystywanych w standardowych cyklach obróbkowych (jeżeli się takie cykle stosuje we programie). Z tych względów zaleca się unikać we własnym programie zakresu R00-R50. W przypadkach wątpliwych należy odczytać wykorzystywany cykl obróbkowy i sprawdzić czy nie występuje konflikt numerów parametrów.

Niżej przedstawiono sposób zapisu i przykłady podstawowych działań matematycznych wykonywanych na parametrach.

1. Definiowanie parametru	R1=100.33		
2. Podstawienie	R1=R2		
2. Negacja	R1=-R2,	R1=-15.1	
3. Dodawanie	R1=R2+R3,	R1=15.1+R3,	R1=R2+15.1
4. Odejmowanie	R1=R2-R3,	R1=10.22-R3,	R1=R2-10.22
5. Mnożenie	R1=R2*R3,	R1=100*R3,	R1=R2*100
6. Dzielenie	R1=R2/R3,	R1=99.9/R3,	R1=R2/99.9
7. Pierwiastek kwadratowy	@613 R1 R2	*(wynik w R1)*	

Możliwy jest zapis złożonych wyrażeń algebraicznych. Dużym utrudnieniem jest jednak brak możliwości użycia nawiasów do budowy tych wyrażeń. Kolejność wykonania działań arytmetycznych w złożonych wyrażeniach jest określona kolejnością zapisu tych działań licząc od strony lewej do prawej. Nie jest więc zachowana przyjęta powszechnie konwencja priorytetów działań.

Przykład:

Wyrażenie R1=R2+R3-R4*R5/R6 jest obliczane następująco:

1 krok R1=R2 *(czytaj pod R1 podstaw R2),*

2 krok R1=R1+R3,

3 krok R1=R1-R4,

4 krok R1=R1*R5,

5 krok R1=R1/R6,

W efekcie parametr R1 zostanie obliczony według wzoru zapisanego matematycznie:

*R1=[(R2+R3-R4)*R5]/R6*

Gdyby wyrażenie zapisać w zmienionej kolejności: R1=-R4*R5/R6+R2+R3 wynik byłby inny, zgodny z zapisem matematycznym (bez nawiasów). Dla uniknięcia pomyłek zaleca się rozdzielanie złożonych działań na działania prostsze z użyciem pomocniczych parametrów, np:

R10=R4*R5/R6

R1=R2+R3-R10

Po wykonaniu obliczeń parametry mogą zostać wykorzystane w bloku programu sterującego w słowach wymiarowych (programowania toru narzędzia) oraz w innych słowach (funkcjach) z wyjątkiem numeru bloku "N" i numeru programu "%". Poniżej podano sposób zapisu i przykłady przyporządkowania parametrów do różnych słów w programie sterującym.

N5 Z=R5 LF

N10 G=R16 X=R20 Z=R25 F=R28 LF

Istnieje możliwość przyporządkowania parametrów do adresów współrzędnych z jednoczesnym wykorzystaniem działania dodawania lub odejmowania wartości stałej. Nie należy natomiast przypisywać złożonych wyrażeń do adresów współrzędnych.

Przykład :

N38 R1=9.7 R2=-2.1

N40 X=20.3+R1

N42 Z=19.7-R1

Powyższy fragment jest równoważny zapisowi:

N40 X30

N42 Z10

17. WYBRANE KODY @

Kody @ umożliwiają programowanie układu sterowania SINUMERIK 810T na poziomie zbliżonym do assemblera (języka wewnętrznego). Za pomocą tych kodów programować można skoki, rozgałęzienia, pętle, funkcje matematyczne można wykonywać operacje na rejestrach wewnętrznych itp. Zapis składa się ze znaku @ i trzypozycyjnego kodu, np.: @123. Znaczenie kolejnych cyfr w kodzie jest następujące:

1 - pierwsza cyfra określa grupę główną,

2 - środkowa cyfra określa podgrupę,

3 - ostatnia cyfra określa funkcje specjalne.

Wyróżnia się następujące grupy główne:

@0 - ogólne instrukcje dotyczące budowy programu,

@1 - rozgałęzienia w programach,

@2 - ogólne instrukcje przesyłania danych,

@3 - przesyłanie danych z pamięci do parametrów,

@4 - przesyłanie danych z parametrów do pamięci,

@6 - funkcje matematyczne i logiczne,

@7 - funkcje specjalne dla NC,

@a - funkcje E/A.

W instrukcjach z kodami @ występują argumenty należące do ściśle zdefiniowanych typów. Typ argumentu oznacza się następującymi literami:

K - stała,

R – parametr,

P - wskaźnik (Pointer).

Przykład instrukcji z kodem @

@201 R13 P37.

Znaczenie poszczególnych słów jest następujące:

@201 - kod funkcji z grupy „ogólne instrukcje przesyłania danych”,

R13 - wskaźnik, podający adres rejestru docelowego

P37 - wskaźnik, podający adres rejestru źródłowego

Rozgałęzienia w programach, kod: @ 1 X Y

Znaczenie cyfry stojącej na pozycji Y

0: nie ma operacji porównania

1: = równy

2: ≠ nierówny

3: > większy

4: >= większy lub równy

5: < mniejszy

6: <= mniejszy lub równy

Znaczenie cyfry stojącej na pozycji X

0: skok bezwarunkowy

1: rozgałęzienie CASE

2: rozgałęzienie warunkowe IF-THEN-ELSE

3: pętla WHILE

4: pętla REPEAT
5: pętla FOR TO
6: pętla FOR DOWNTO

Grupa główna

1. Skok bezwarunkowy

@100 <Const> lub @100 <R-Par>

Numer bloku do którego ma być wykonany skok jest określony przez stałą (Const) lub przez parametr R. Podanie dodatniego numeru bloku oznacza, że szukanie bloku będzie wykonywane w kierunku końca programu. Ujemny numer bloku wymusza poszukiwanie w kierunku początku programu.

Przykład :

@100 K375 - skok bezwarunkowy do bloku 375 (w kierunku końca programu),

@100 K-150 - skok bezwarunkowy do bloku 150 (w kierunku początku programu).

2. Instrukcja wyboru (CASE)

@111 <Var> <Wartość_1> <Const_1>
<Wartość_2> <Const _2>
...
<Wartość_ n> <Const_n>

Zmienna <Var> porównywana jest z kolejnymi wartościami zapisanymi pod <Wartość...>. Jeżeli wynik porównania jest prawdą, to zostanie wykonany skok do bloku, którego numer zapisano pod <Const...>. Jeżeli żadna z par nie spełnia warunku równości wykonywany jest blok programu stojący po instrukcji @111.

3. Instrukcje typu IF-THEN-ELSE

@121 <Var> <Wartość> <Const>

Jeżeli warunek równości pomiędzy <Var> i <Wartość> jest spełniony, to zostanie wykonany blok (lub bloki) stojące bezpośrednio za instrukcją @121. W przeciwnym razie nastąpi skok do bloku zapisanego pod <Const>.

Przykład :

@121 R13 R27 K375
N100 ...
...
N375

Jeżeli R13=R27 program będzie wykonywany od bloku N100 w przeciwnym razie nastąpi skok do bloku N375. Blok będzie poszukiwany w kierunku końca programu. Kolejne pięć instrukcji należy do tej samej grupy IF-THEN-ELSE. Instrukcje różnią się postacią warunku logicznego porównującego <Var> i <Wartość>.

@122 <Var> <Wartość> <Const> - jeżeli nierówny to blok następny,
w przeciwnym razie skok do <Const>

@123 <Var> <Wartość> <Const> - jeżeli większy to blok następny,

w przeciwnym razie skok do <Const>

@124 <Var> <Wartość> <Const> - jeżeli większy lub równy to blok następny w przeciwnym razie skok do <Const>

@125 <Var> <Wartość> <Const> - jeżeli mniejszy to blok następny, w przeciwnym razie skok do <Const>.

@126 <Var> <Wartość> <Const> - jeżeli mniejszy lub równy to blok następny w przeciwnym razie skok do <Const>

4. Pętla WHILE (dopóki)

@13i <Var> <Wartość> <Const>

Pętla WHILE jest instrukcją do tworzenia pętli programowych z odczytem warunku logicznego na początku pętli. Jest więc pierwszą instrukcją w pętli. Na końcu pętli należy zaprogramować skok bezwarunkowy do początku pętli (@100 ...). Działanie instrukcji WHILE polega na powtarzaniu fragmentu programu i sprawdzaniu warunku logicznego (=,≠, >, ≥, <, ≤) pomiędzy <Var> i <Wartość>. Warunek jest sprawdzany przed rozpoczęciem realizacji serii instrukcji wewnątrz pętli. Dopóki warunek logiczny (równości lub nierówności - odpowiednio do zaprogramowanego kodu) jest spełniony następuje powtarzanie fragmentu programu. Jeżeli warunek nie jest spełniony następuje skok do bloku zapisanego pod <Const>. Blok do którego odbywa się skok powinien leżeć poza pętlą. Niżej zestawiono kody @13i odpowiadające różnym warunkom porównania <Var> i <Wartość>.

@131 <Var> <Wartość> <Const> - równy "="

@132 <Var> <Wartość> <Const> - nierówny "≠"

@133 <Var> <Wartość> <Const> - większy ">"

@134 <Var> <Wartość> <Const> - większy lub równy "≥"

@135 <Var> <Wartość> <Const> - mniejszy "<"

@136 <Var> <Wartość> <Const> - mniejszy lub równy "≤"

Przykłady:

N300 @131 R13 R27 K375 *(początek pętli WHILE z warunkiem „=”)*
... *(instrukcje wewnątrz pętli)*
@100 K-300 *(powrót na początek pętli)*
N375... *(dalszy ciąg programu)*

Pętla jest wykonywana dopóki spełniony jest warunek R13=R27.

N300 @133 R13 R27 K375 *(początek pętli WHILE z warunkiem „=”)*
... *(instrukcje wewnątrz pętli)*
@100 K-300 *(powrót na początek pętli)*
N375 ... *(dalszy ciąg programu)*

Pętla jest wykonywana dopóki spełniony jest warunek R13>R27.

5. Pętla REPEAT (powtarzaj)

@14i <Var> <Wartość> <Const>

Instrukcja REPEAT (powtarzaj) organizuje pętlę ze sprawdzeniem warunku logicznego po wykonaniu wszystkich instrukcji wewnątrz pętli. Jest więc ostatnią instrukcją w pętli. Po-

wtarzanie jest wykonywane do chwili spełnienia warunku logicznego. Inaczej mówiąc jeżeli warunek porównania <Var> i <Wartość> nie jest spełniony następuje skok do bloku o numerze zapisanym pod <Const> (na początek pętli). Spełnienie warunku powoduje przejście do wykonywania następnego bloku programu sterującego czyli wyjście poza pętlę. Niżej zestawiono kody @14i odpowiadające różnym warunkom porównywania <Var> i <Wartość>.

@141 <Var> <Wartość> <Const> - równy "="
@142 <Var> <Wartość> <Const> - nierówny "≠"
@143 <Var> <Wartość> <Const> - większy ">"
@144 <Var> <Wartość> <Const> - większy lub równy "≥"
@145 <Var> <Wartość> <Const> - mniejszy "<"
@146 <Var> <Wartość> <Const> - mniejszy lub równy "≤"

Przykład:

N400 ... *(początek pętli)*
... *(instrukcje wewnątrz pętli)*
@141 R13 R27 K-400 *(powtarzaj do chwili spełnienia warunku R13=R27)*

Fragment programu od bloku N400 do bloku z funkcją @141 będzie powtarzany tak długo, aż zostanie spełniony warunek R13=R27.

N400 ... *(początek pętli)*
... *(instrukcje wewnątrz pętli)*
@143 R13 R27 K-400 *(powtarzaj do chwili spełnienia warunku R13>R27)*

Fragment programu od bloku N400 do zapisu instrukcji @143 będzie powtarzany tak długo aż zostanie spełniony warunek R13>R27.

6. Pętla FOR-TO

@151 <Var> <Wartość> <Const>

Działanie pętli FOR-TO polega na zwiększaniu w każdym przebiegu o 1 (inkrementacja) parametru określonego zapisanego pod <Var> i porównywaniu zmiennej <Var> z <Wartość>. Instrukcja zapisywana jest na początku pętli. Dopóki występuje nierówność (czyli warunek mniejszości Var<Wartość) pętla jest powtarzana. Wystąpienie równości <Var> i <Wartość> powoduje wykonanie skoku do bloku zapisanego pod <Const>. Na końcu pętli należy powiększyć zmienną <Var> (stosując np. instrukcję @620) oraz użyć instrukcji skoku bezwarunkowego @100 w celu powrotu na początek pętli.

Przykład :

R5=1 R51=5 R52=10
@201 R50 P51 *(przepisanie wartości R5 do R50)*
N500 @151 R50 R52 K505 *(początek pętli)*
... *(instrukcje wewnątrz pętli)*
@620 R50 *(inkrementacja R50)*
@100 K-500 *(skok do bloku N500)*
N505 ...

Parametr R50 podczas wykonywania pętli będzie przyjmował wartości od 1 do 10. Po osiągnięciu wartości 10 nastąpi skok do bloku N505. Instrukcje wewnątrz pętli zostaną więc powtórzone 9 razy.

7. Pętla FOR-DOWNTO

@161 <Var> <Wartość> <Const>

Pętla FOR-DOWNTO jest pętlą zliczającą „w dół". Wartość parametru określonego w <Var> jest zmniejszana o 1 (dekrementacja) przy każdym przebiegu pętli i porównywana na początku pętli z wartością zmiennej <Wartość>. Dopóki występuje nierówność pętla jest powtarzana. Wystąpienie równości <Var> i <Wartość> powoduje wykonanie skoku do bloku zapisanego pod <Const>. Prawidłowe działanie pętli wymaga użycia na końcu pętli instrukcji dekrementującej zmienną <Var> (np. @621) oraz instrukcji skoku bezwarunkowego @100.

Przykład:

```
R5=10 R51=5 R52=1          (podstawienie wartości pod parametry)
@201 R50 P51               (przepisanie wartości R5 do R50)
N600 @161 R50 R52 K605     (początek pętli)
...                        (instrukcje wewnątrz pętli)
@621 R50                   (dekrementacja R50)
@100 K-600                 (skok do bloku N600)
N605 ...                   (dalszy ciąg programu)
```

Parametr R50 podczas wykonywania pętli będzie przyjmował wartości od 10 do 1. Przyjęcie wartości 1 spowoduje przeskok do bloku N605. Fragment programu zostanie więc powtórzony 9 razy.

18. Funkcje matematyczne

Funkcje matematyczne należą do grupy głównej nr 6 i cyfra ta występuje na pierwszej pozycji po znaku @. Na drugiej pozycji kodowana jest podgrupa funkcji, której kod może przyjmować następujące wartości:

0 - wpisywanie wartości z operacjami arytmetycznymi
1 - funkcje arytmetyczne
2 - procedury arytmetyczne
3 - funkcje trygonometryczne
4 - funkcje logarytmiczne
5 - funkcje logiczne
6 - procedury logiczne
7 - boolowskie funkcje porównawcze

Niżej wyjaśnione zostaną najważniejsze funkcje, które będą wykorzystane w późniejszych przykładach.

1. Wartość bezwzględna.

@610 <Var> <Wartość>

Pod zmienną <Var> zostanie podstawiona wartość bezwzględna z parametru (lub stałej) zapisanego pod <Wartość>.

Przykład:

```
R12=-34
```

@610 R76 R12

Pod parametr R76 zostanie podstawiona liczba |-34| =34.

2. Pierwiastek kwadratowy.

@613 <Var> <Wartość>

Pod zmienną <Var> zostanie podstawiony pierwiastek kwadratowy z parametru lub stałej zapisanych pod <Wartość>.

Przykład:

@613 R13 K64

Pod parametr R13 zostanie podstawiona liczba 8. Zapis K64 oznacza stałą równą 64.

3. Pierwiastek z sumy kwadratów.

@614 <Var> <Wartość1> <Wartość2>

Wynik obliczeń pierwiastka z sumy kwadratów *Wartość1* i *Wartość2* zostanie podstawiony pod zmienną <Var>.

Przykład:

R25=15 R26=20

@614 R77 R25 R26

Wynik obliczeń równy 25 zostanie podstawiony pod parametr R77.

4. Inkrementacja czyli powiększenie o 1.

@620 <Var>

Efektem działania funkcji jest powiększenie o 1 wartości zmiennej <Var>

Przykład:

R70=1

@620 R70

Nowa wartość R70 wynosi 2. Taki sam wynik można uzyskać zapisując działanie: R70=R70+1

5. Dekrementacja czyli zmniejszenie o 1.

@621 <Var>

Efektem działania funkcji jest zmniejszenie o 1 wartości zmiennej <Var>

Przykład:

R70=1

@621 R70

Nowa wartość parametru R70 wynosi 0. Taki sam wynik można uzyskać zapisując działanie: R70=R70-1.

6. Część całkowita z liczby.

@622 <Var>

Funkcja oblicza część całkowitą z liczby zapisanej pod <Var>. Wynik zostaje podstawiony pod tę samą zmienną <Var>.

Przykład:

R60=2.9
@622 R60

Nowa wartość parametru R60 wynosi 2.

7. Funkcje trygonometryczne sin, cos, tan, arctg, kąt pomiędzy wektorem i jego składową.

@630 <Var> <Wartość> - sinus
@631 <Var> <Wartość> - cosinus
@632 <Var> <Wartość> - tangens
@634 <Var> <Wartość> - arcsinus
@637 <Var> <Wartość1> <Wartość2> - kąt pomiędzy wektorem a jego składową

Argument każdej z funkcji jest zapisywany pod <Wartość>, natomiast wynik zostaje podstawiany pod zmienną <Var>. Kąty powinny być podawane w stopniach.

Przykład:
R27=30
@630 R15 R27

Pod parametr R15 zostanie podstawiona wartość funkcji sinus z kąta podstawionego wcześniej pod parametr R27. Parametr R15 przyjmie wartość sin(30°)=0.5.

Funkcja @637 wymaga zapisania pod <Wartość1> i <Wartość2> współrzędnych dowolnego wektora. Wynikiem jest kąt pomiędzy składową tego wektora zapisaną pod <Wartość2> i kierunkiem wektora wypadkowego. Kąt liczony jest w kierunku dodatnim czyli przeciwnie do ruchu wskazówek zegara.

Przykład:
R35=20 R36=30
@637 R17 R35 R36

Pod parametrami R35 i R36 zapisane zostały składowe wektora. Wynik zostanie umieszczony pod parametrem R17=146.30993.

8. Funkcja logarytmiczna

@640 <Var> <Wartość> - logarytm naturalny ln(x)

Argument każdej z funkcji jest zapisywany pod <Wartość>, natomiast wynik jest podstawiany pod <Var>.

Przykład:
@640 R80 K10 *(wynik R80=2.3025846)*

9. Funkcja expotencjalna.

@641 <Var> <Wartość> - funkcja expotencjalna e^x

Przykład:
@641 R80 K2.5 *(wynik R80=12.182496)*