

SINUMERIK WS 800 A
Software Version 1
Cycle language CL 800

Planning Guide

Edition 04.90

SINUMERIK WS 800 A

Software Version 1

CL 800 cycle language

Planning Guide

Manufacturer documentation

April 1990 Edition

Introduction	1
Overview of variables	2
Language notation	3
Structure of the CL 800 high-level language	4
Command description	5
Command overview	6
Examples	7
Using the interactive editor	8
Error messages of the cycle editor	9
Object code	10
Appendix	11

Printing history

Brief details of this edition and previous editions are listed below.

The status of each edition is shown by the code in the "Remarks" column.

Status code in "Remarks" column:

A . . . New documentation **B** . . . Unrevised reprint with new Order No.
C . . . Revised edition with new status

Edition	Order No.	Remarks
04.90	6ZB5 410-ODP02-0AA0	A

Other functions not described in this documentation might be executable in the control. This does not, however, represent an obligation to supply such functions with a new control or when servicing.

This publication was produced on the Siemens 5800 Office System.
Subject to change without prior notice.

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved.

Contents

	Page
1 Introduction	1-1
1.1 General	1-1
1.2 Using the planning workstation	1-1
1.2.1 File functions	1-3
1.2.2 Special functions	1-4
1.2.2.1 Interactive processing	1-4
1.2.2.2 Compilation	1-5
1.2.2.3 Cross-references	1-5
2 Overview of variables	2-1
2.1 General	2-1
2.2 R parameters	2-3
2.2.1 Central and channel-orientated R parameters	2-3
2.2.2 Channel-orientated R parameters	2-4
2.3 System memory	2-7
2.3.1 Machine data	2-7
2.3.2 Setting data	2-10
2.3.3 Tool offsets	2-12
2.3.4 Zero offsets	2-15
2.3.5 PLC signals	2-16
3 Language notation	3-1
3.1 General	3-1
3.2 CL 800 words	3-1
3.3 Operands	3-1
4 Structure of the CL 800 high-level language	4-1
4.1 Program structure	4-1
4.2 Data structure	4-3

	Page
5	Command description 5-1
5.0	Preliminary information 5-1
5.1	Program frame statements 5-1
5.1.1	General statements 5-1
5.1.2	Program header 5-3
5.2	Declarations 5-6
5.2.1	Declaration of variables 5-6
5.2.2	Declaration of variables in external lists 5-10
5.3	Statements 5-14
5.3.1	Repeat statements 5-14
5.3.1.1	REPEAT loop 5-14
5.3.1.2	WHILE loop 5-19
5.3.1.3	WHILE INT loop 5-23
5.3.1.4	FOR TO loop 5-24
5.3.1.5	FOR DOWNT0 loop 5-26
5.3.2	Decision statements 5-29
5.3.2.1	IF THEN ELSE branching 5-29
5.3.2.2	IF INT THEN ELSE branching 5-33
5.3.2.3	Case branching 5-34
5.3.3	Unconditional branching 5-37
5.3.3.1	Unconditional jump 5-37
5.3.4	General data transfer 5-38
5.3.4.1	Data transfer: R par./R par. 5-38
5.3.4.2	Data transfer: R par./input buffer memory for numerical variable 5-39
5.3.5	Data transfer: System memory into the R parameter 5-40
5.3.5.1	Transfer machine data into the R parameter 5-40
5.3.5.2	Transfer setting data into the R parameter 5-42
5.3.5.3	Transfer tool offset values into the R parameter 5-43
5.3.5.4	Transfer zero offsets into the R parameter 5-44
5.3.5.5	Read programmed setpoints into the R parameter 5-47
5.3.5.6	Read actual values into the R parameter 5-49
5.3.5.7	Read programmed data into the R parameter 5-54
5.3.5.8	Read PLC signal bits into the R parameter 5-55
5.3.5.9	Read PLC signal bytes into the R parameter 5-56
5.3.5.10	Read PLC signal words into the R parameter 5-57
5.3.5.11	Read PLC signal data words into R parameter 5-58
5.3.5.12	Read alarms into the R parameter 5-60
5.3.5.13	Read alarm pointer into the R parameter 5-61
5.3.5.14	Read system memory into the R parameter 5-61
5.3.6	Data transfer: R parameter into the system memory 5-62
5.3.6.1	Transfer R parameter into the machine data 5-62
5.3.6.2	Transfer R parameter into the setting data 5-65
5.3.6.3	Write R parameter into the tool offsets 5-66
5.3.6.4	Write R parameter into the zero offsets 5-67

	Page	
5.3.6.5	Write R parameter into the programmed setpoints	5-70
5.3.6.6	Write R parameter into the PLC signal bits	5-72
5.3.6.7	Write R parameter into the PLC signal bytes	5-73
5.3.6.8	Write R parameter into the PLC signal words	5-73
5.3.6.9	Write R parameter into the PLC signal data words	5-74
5.3.6.10	Write R parameter into the alarms	5-77
5.3.6.11	Write R parameter into the system memory	5-77
5.3.7	Mathematical and logical functions	5-78
5.3.7.1	Value assignment with arithmetic operations	5-78
5.3.7.2	Arithmetic functions	5-79
5.3.7.3	Arithmetic procedures	5-81
5.3.7.4	Trigonometric functions	5-82
5.3.7.5	Logarithmic functions	5-84
5.3.7.6	Logical functions	5-85
5.3.7.7	Logical procedures	5-87
5.3.8	NC-related functions	5-88
5.3.8.1	Changing the program and machine reference points	5-88
5.3.8.2	Single functions	5-89
5.3.8.3	Measuring functions	5-96
5.3.8.4	Program influence	5-97
5.3.9	I/O statements	5-98
5.3.9.1	NC I/O functions	5-98
5.3.9.2	General I/O functions	5-99
5.3.9.3	Operator control functions	5-102
6	Command overview	6-1
6.1	General statements for the program structure	6-1
6.2	Declarations	6-2
6.3	Repeat statements	6-4
6.4	Decision statements	6-5
6.5	Unconditional branching	6-5
6.6	Data transfer, general	6-6
6.7	Data transfer: System memory into the R parameter	6-7
6.8	Parameters in system memory	6-12
6.9	Mathematical and logical functions	6-16
6.10	NC-related functions	6-17
6.11	I/O statements	6-19

	Page
7 Examples	7-1
7.1 Program structure overview	7-1
7.2 Program nesting for IF THEN ELSE branching	7-2
7.3 Program example: Hole pattern	7-3
7.4 Program example: Deep-hole drilling cycle	7-5
8 Using the interactive editor	8-1
8.1 General	8-1
8.2 Interactive editor operating modes	8-2
8.2.1 Functions in the display mode	8-5
8.2.2 Functions in the command mode	8-10
9 Cycle editor error messages	9-1
9.1 General	9-1
9.2 Error message list	9-2
9.2.1 Warning messages	9-2
9.2.2 User error messages	9-2
9.2.3 System error messages	9-5
10 Object code	10-1
10.1 Structure of the @ code	10-1
10.1.1 Subdivision into main groups	10-1
10.1.2 Operands after the @ function	10-2
10.1.3 Notation	10-2
10.2 General statements for the program structure	10-3
10.3 Program branching	10-4
10.4 General data transfer	10-7
10.5 Data transfer: System memory into the R parameter	10-8
10.6 Data transfer: R parameter into the system memory	10-16
10.7 File handling, general: (in preparation)	10-22

	Page
10.8	Mathematical and logical functions 10-23
10.9	NC-specific functions 10-25
10.10	I/O statements 10-27
11	Appendix 11-1
11.1	Alphabetic keyword index/CL 800 names 11-1
11.2	Terminology 11-7
11.3	Overview of functions implemented according to the software version .. 11-10
11.4	Weighting of operators 11-17
11.5	G-group classification 11-18

Preliminary remarks

Instructions for reading

The SINUMERIK documentation is subdivided into three levels:

- User documentation,
- Manufacturer documentation and
- Service documentation.

This "Planning guide" is for machine tool manufacturers using SINUMERIK System 800 controls.

The document describes the program structure and the operation set of the CL 800 cycle language.

The following is available in addition to this document:

- User's Guide, WS 800 A software,
- Programming Guide, SINUMERIK System 800 cycles.

Software versions

The complete functional scope of this description of the cycle language is valid from the following software versions onwards:

Planning workstation

WS 800:	Software version V 2.1
WS 800 A:	Software version 1

SINUMERIK System 800

SINUMERIK 805:	Software version 2
SINUMERIK 810:	Software version 2 (05)
SINUMERIK 810 GA1:	Software version 3
SINUMERIK 810 GA2:	Software version 2
SINUMERIK 810 GA3:	Software version 1
SINUMERIK 810G/820G GA2:	Software version 1
SINUMERIK 810G/820G GA3:	Software version 1
SINUMERIK 820 GA2:	Software version 2
SINUMERIK 820 GA3:	Software version 1
SINUMERIK 840:	Software version 1
SINUMERIK 850:	Software version 3
SINUMERIK 880:	Software version 3

SINUMERIK 805 cannot be configured.

The software version for SINUMERIK 810 given in brackets is valid for non-configurable controls, which were supplied before 3/88.

Section 1

-Introduction-

Overview:

1.1 **General**

1.2 **Using the configuring station**

1.2.1 File functions

1.2.2 Special functions

1.2.2.1 Interactive processing

1.2.2.2 Compilation

1.2.2.3 Cross-references

1 Introduction

1.1 General

Structured programming of subroutines and processing cycles with the CL 800 high-level language is transparent, understandable, easy to change and especially reliable. It permits the program to be drafted in a structural fashion using modern methods.

Cycles are clearly arranged and easily readable using symbolic programming. This allows other software developers to continue software development and update. In the interactive CL 800 editor mode, the programmer is immediately notified of possible syntax errors which increases the reliability.

The CL 800 language scope extends far beyond the level expected from high-level languages. In addition to mathematical and logical functions, command structures (IF-THEN-ELSE) and transfer commands, the language offers comprehensive CNC-related functions, such as e.g. measuring functions, reference conditioning, and axis positioning information.

Machine diagnostics is simplified using direct addressing of PLC inputs and outputs using text programs.

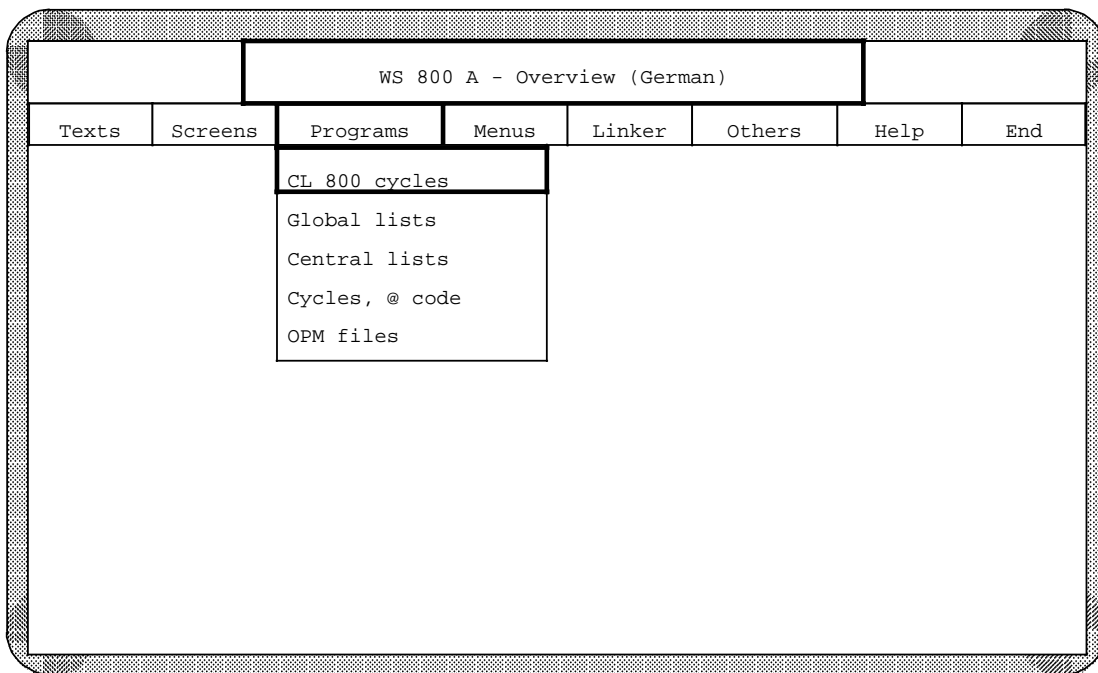
The cycles are converted into a memory space reducing CNC format using the compiler.

1.2 Using the planning workstation

The CL 800 cycles, external variable lists and the operator prompting macros **OPM** are combined in the object **program**. Statements for generating cycle program sections in the NC are designated as operator prompting macros **OPM**.

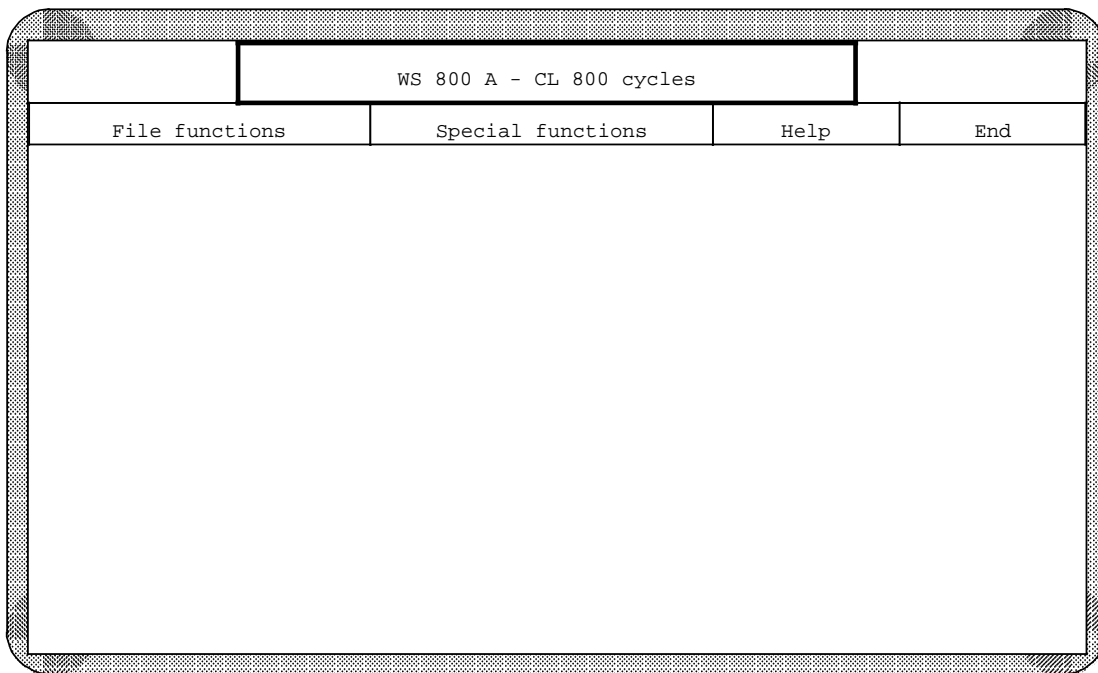
The following two possibilities of processing CL 800 files are available to the user in the object main menu:

- General text editor File functions
- Interactive editor Special functions



WS 800 A basic menu

If a cycle is to be processed, the "CL 800 cycles" file type is selected using the pull-down menu **program** in the basic menu. The object main menu with the file- and special functions then appears.

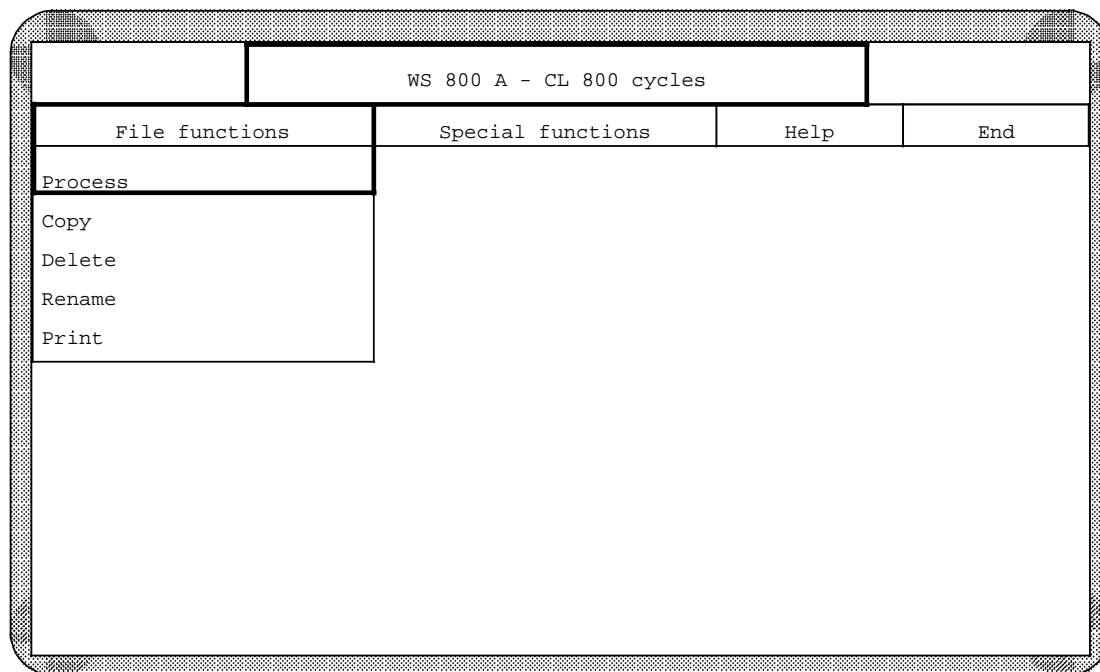


CL 800 cycles, object main menu

The user can call-up help texts on the screen, which support him during configuring, using the **help** menu. The user returns from the object main menu to the basic menu after actuating **end**.

1.2.1 File functions

Functions, which generally involve the whole file are involved here. The corresponding functions can be selected using a pull-down menu.



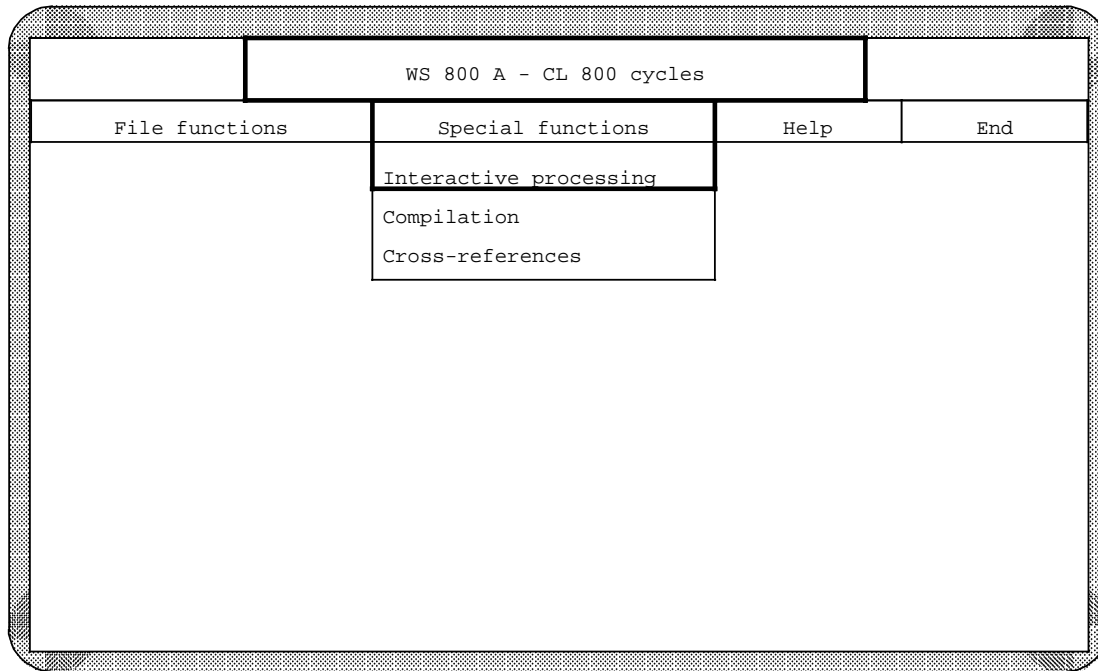
CL 800 cycles, file functions

The general text editor is selected after the **processing** function has been selected.

An existing file can be selected in the current catalog using a corresponding dialog box. After file selection, the general text editor is activated through windows. A source file for CL 800 cycles or for external data files can be newly generated or those already available, changed here. Input is realized without analysis.

1.2.2 Special functions

The following special functions can be selected using a pull-down menu for cycle processing, with the **program** data type.



CL 800 cycles, special functions

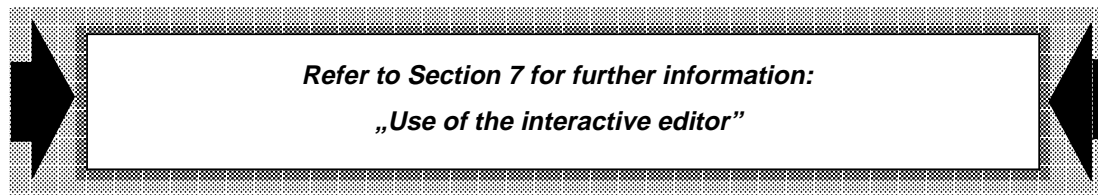
The **compilation** and **cross-references** functions are eliminated for the global and central lists.

1.2.2.1 Interactive processing

CL 800 file processing can be realized with, in addition to the general text editor, a CL 800 - specific interactive editor.

After the **interactive processing** function is selected, a file is first specified via a corresponding dialog box. After this, the interactive cycle editor is selected. A source file for CL 800 cycles or for external data files can now be newly generated, or an existing one changed. The essential difference to the general text editor is that each input line is directly checked for correct formatting. A corresponding error message appears in the command mode if the line was incorrectly input.

If a file was edited and stored up to the end criteria (END. or ENDEXTERN), then the already checked and analyzed file is converted and the object code file generated. The object main menu is automatically output after editing has been completed.



1.2.2.2 Compilation

Compilation is the pure conversion of CL 800 programs into the @ code.

The external variable lists are, if available, automatically compiled at the next compilation.

Similar to the file functions, a file is defined via a dialog box. The compilation is activated after file selection. The number of the line being processed is continually displayed during compilation. Compilation is faster, as the complete program contents are not output line for line.

Compilation either ends with an OK or error message. Compilation is terminated and the error is output with the corresponding line number as soon as the analysis discovers an error.

The user returns to the object main menu after the error message has been acknowledged. The current file must now be modified corresponding to the error messages.

1.2.2.3 Cross-references

It is difficult to find an @ code statement generated from a CL 800 statement, as the listings of source and @ codes are significantly different as far as the line numbering is concerned.

Thus, WS 800 A has the capability of generating a cross-reference list between CL 800 programs and @ code.

This cross-reference list is generated by a dedicated compilation. The difference to compilation is that no object code is created which can be loaded in the NC, but a listing file with the same file names as the source code.

Each **statement** which was converted from the source code into the NC code has the associated line number of the source code in the cross-reference list. The associated line number of the source code is located in the cross-reference listing at the line start, there, where a new **block** is generated in the NC code.

The **cross-reference** function can only be used for CL 800 programs.

Section 2

-Overview of variables-

Overview:

2.1 General

2.2 R parameters

2.2.1 Central and channel-orientated R parameters

2.2.2 Channel-orientated R parameters

2.3 System memory

2.3.1 Machine data

2.3.2 Setting data

2.3.3 Tool offsets

2.3.4 Zero offsets

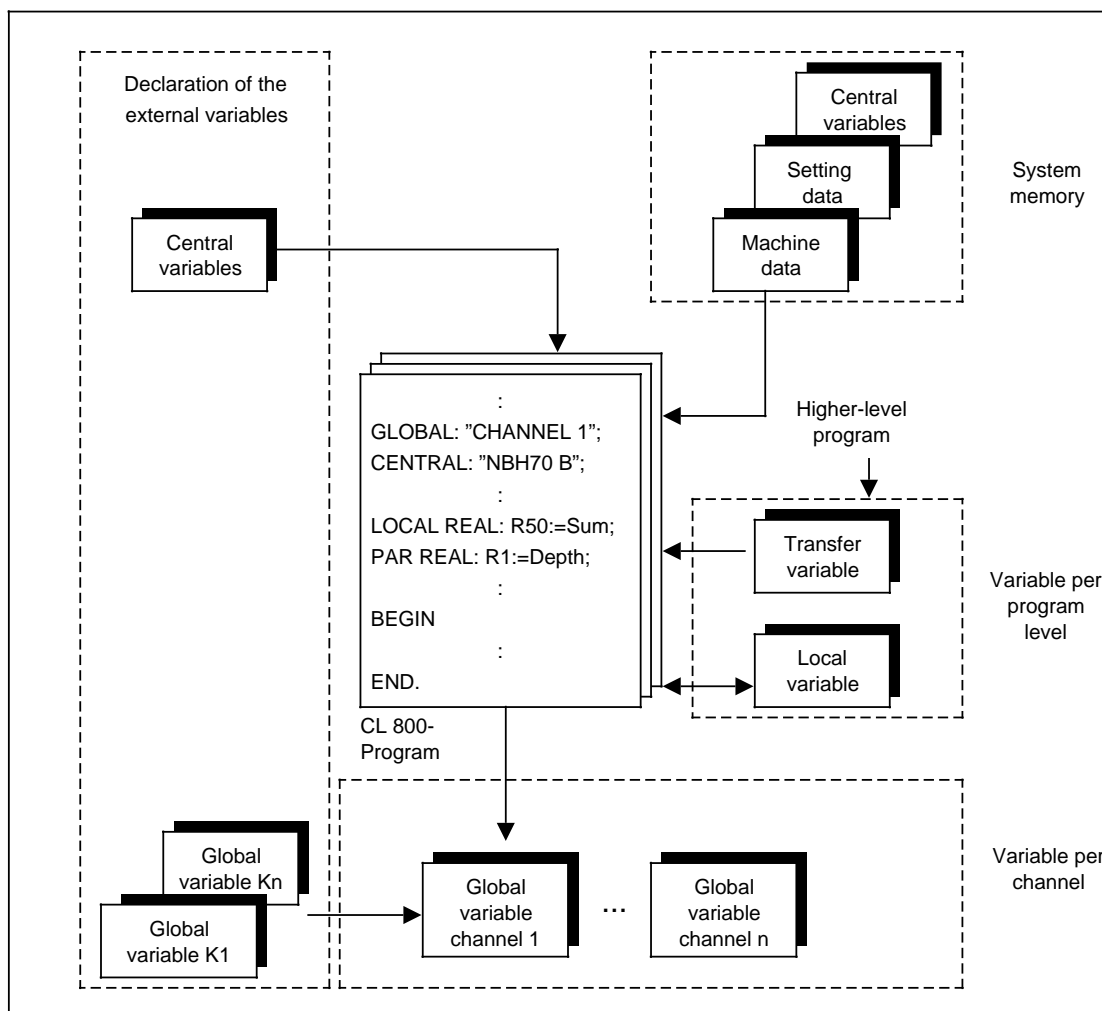
2.3.5 PLC signals

2 Overview of variables

2.1 General

The variables, with which the cycles operate, are subdivided into the following areas (Fig.):

- transfer variables
- local variables
- global variables
- central variables
- system memory



Variables and their declaration for cycles

Transfer variable

With the transfer variables, values from the higher-level program are transferred to the program called. The parameters are declared in the program called. Operator prompting during input of the higher-level program is implemented with these parameters.

Local variable

All local variables used in the program are declared in the declaration part. Their contents are zero at the beginning of the subroutine. Intermediate results **within a subroutine** are stored in the variables. The variables remain valid within the subroutine.

Global variable

Global variables are only present once per channel. They can be accessed by all programs which run **within a channel**.

Global variables (R100 - R149) are declared in a separate list for each channel. The declaration of global variables (R150 - R199) takes place jointly in one list for all channels. The corresponding list names are referred to in the declaration part. Using the separate lists for the declaration of global variables, the subroutines of a channel can be corrected and recompiled independently of the programs of another channel.

Central variable

Central variables are only present once in the communication RAM. They can be accessed **by all channels**.

They can therefore be used to transfer information between all channels. The central variables are declared in a separate list. The list is referred to in the declaration part.

System memory

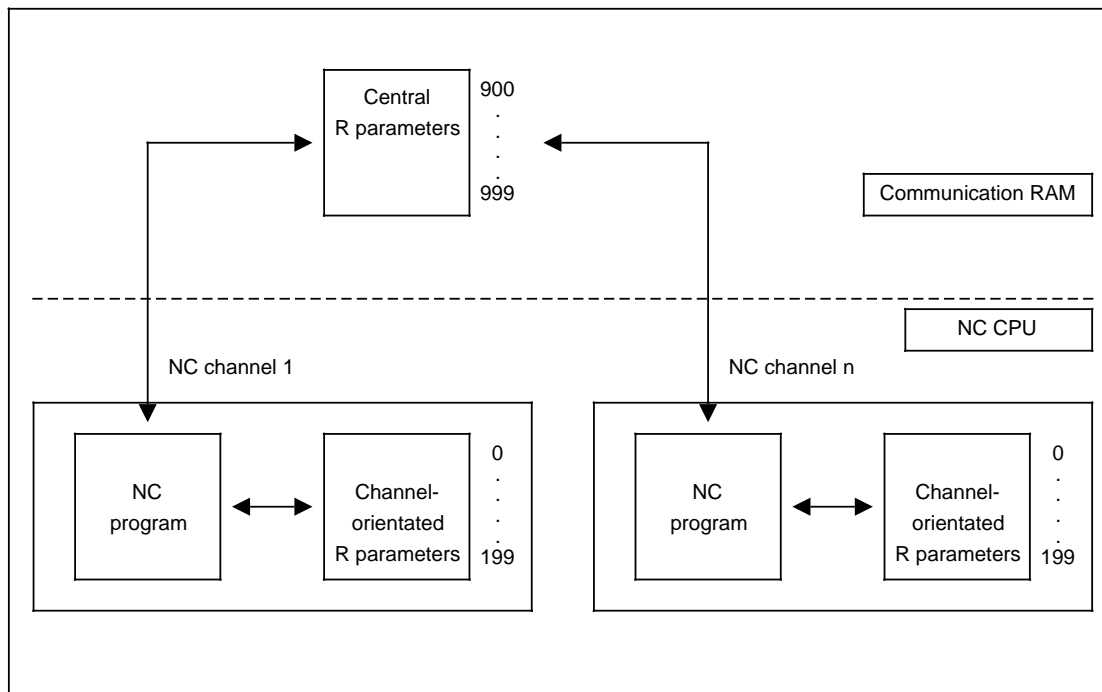
All variables, which have a permanent task within the NC (e.g. TO memory, ZO memory, etc.) come under system memory. Their designations are defined and are part of the CL 800 cycle language.

2.2 R parameters

2.2.1 Central and channel-orientated R parameters

Each NC CPU has channel-orientated R parameters per channel. The channel-orientated R parameters are broken down into transfer, local and global R parameters.

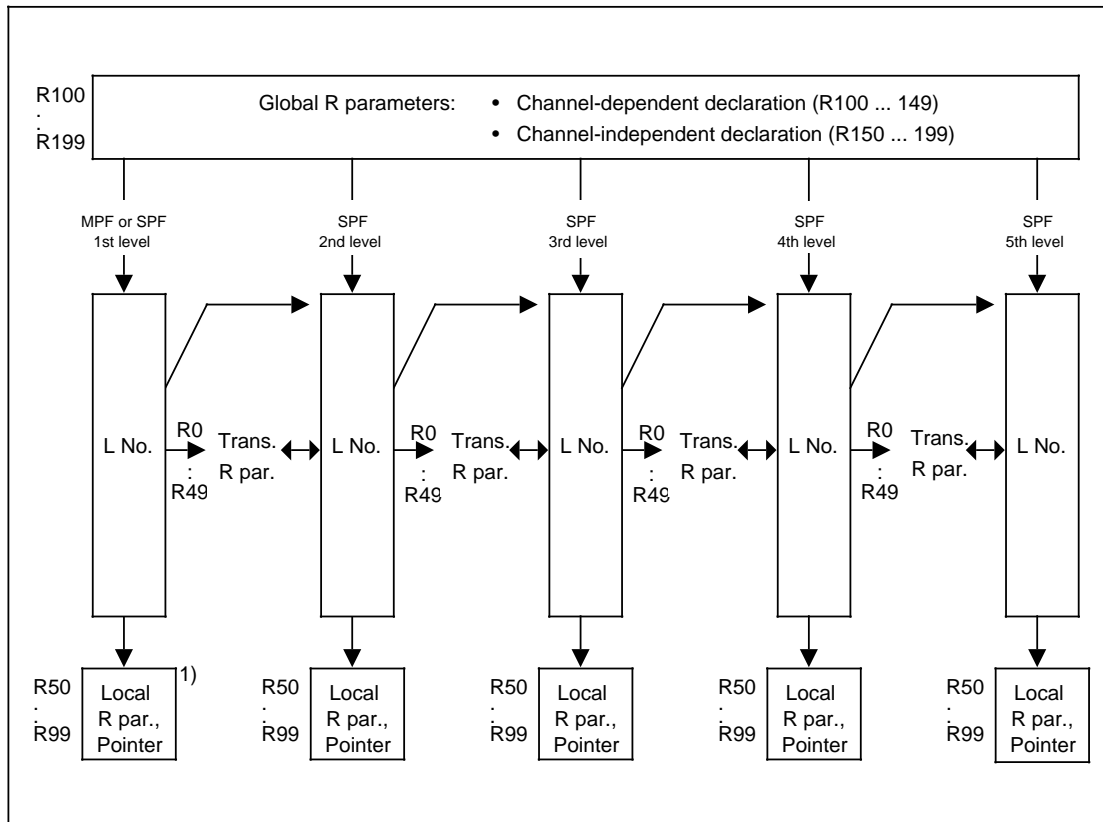
Central R parameters, which are common to all channels, are also present in the communication RAM. The distinction between channel-orientated and central parameters is made via the R parameter addresses.



Central/channel-orientated R parameters

2.2.2 Channel-orientated R parameters

Local, global and transfer R parameters are available per channel. The assignments of the R parameters for the individual tasks are defined as follows so that cycles can be created at different locations and to allow smooth joint operation:



Assignment of the R parameters

Abbreviations:

- MPF ... Part program
- SPF ... Subroutine
- L-No. ... Program number

1) omitted when MP is at the 1st level

Local R parameters (LOCAL)

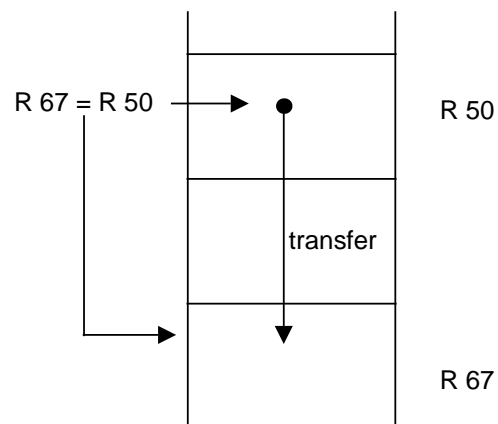
Local R parameters are only valid within a subroutine. When a subroutine is called, the contents of the declared local R parameters are saved at the beginning of the subroutine and re-loaded at the end. In this way, any computed intermediate result is retained in the "local" R parameters, although the same parameter is used at a different program level. The same parameters can thus be used more than once by saving and reloading the "local" R parameters.

Local R parameters as pointers (POINTER)

R parameters can be used as pointers for indirect addressing of R parameters. A pointer contains the address of the parameter into which something is to be entered or read out of.

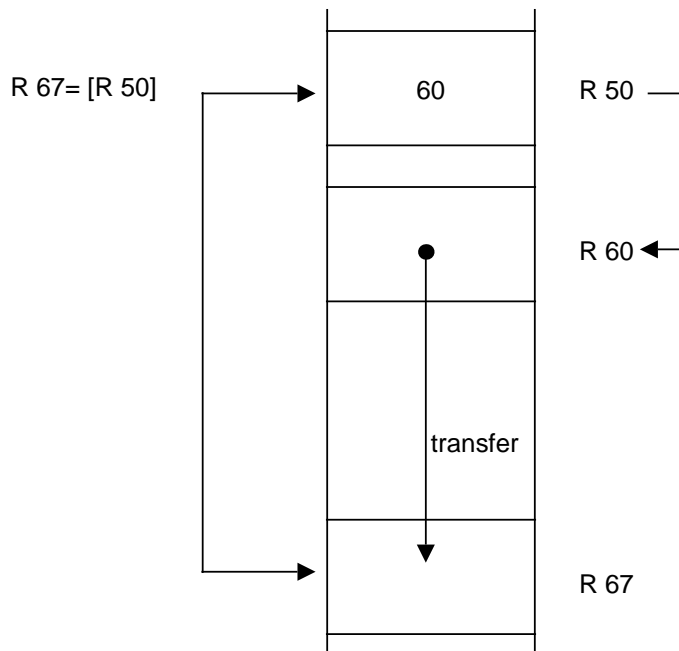
Example:

- **direct** addressing of an R parameter



The addresses of both R parameters are listed in the command.

- **indirect** addressing of an R parameter



The address of the R parameter, whose contents are to be read out, is located in the location to which the 2nd operand of the commands points. As the pointer belongs to the local variable group, they are also saved and reloaded, so that these can also be occupied a multiple number of times.

Transfer R parameters (PAR)

Transfer R parameters serve to initialize subroutines. They must be pre-assigned the desired values before a subroutine is called. The transfer parameters retain their values in the program at all levels, as long as they are not modified.

Transfer R parameters (GLOBAL)

Global R parameters remain valid at all program levels. They can thus be used to transfer information between all subroutines of **one channel**.

Global R parameters are subdivided into two areas for declaration: Channel-dependent declared R parameters R100 - R149, and channel-independent declared R parameters R150 - R199.

R parameters R100 - R149 can be declared differently for each channel, while R parameters R150 - R199 are declared common to all channels.

Subroutines which only run on a particular channel (e.g. machine-dependent cycles), thus use R parameters R100 - R149. All programs which are to be run on several channels (e.g. standard cycles) use R parameters R150 to R199.

2.3 System memory

2.3.1 Machine data

The values filed in the machine data are defined upon first startup of the machine, and are not usually changed again. If machine data is changed (e.g. for optimization purposes) it should be noted that the modification usually does not become effective immediately, but, for example only after a system reset, for example (e.g. after power-on).

The machine data is subdivided into NC-, cycle-, and PLC machine data. They are further subdivided into the following areas within these groups:

Machine data words:

Group	Area		CL 800 word FB parametrization
NC machine data	general	0-999	MDN <word>
	channel-specific	1000 - 1999	
	axis-specific	2000 - 3999	
	spindle-specific	4000 - 4999	
drive machine data	feed	0x - 256x	MDD <word> x axis address
	feed with spindle function	4000 - 4960	
cycles machine data	channel-specific standard cycles	0 - 49	MDZ <channel> <word>
	channel-specific customer cycles	400 - 449	
	central standard cycles	1000 - 1149	
	central customer cycles	4000 - 4149	

Group	Area		CL 800 word FB parametrization
PLC machine data for 810/820	operating system customer program	0- 999 1000 - 1999	MDP <word>
for 850/880	operating system standard FBs customer program	0 - 1999 2000 - 3999 4000 - 5999	
for 805	(in preparation)		(in preparation)
for 840	(in preparation)		(in preparation)

Machine data bits:

Group	Area		CL 800 word FB parametrization	
NC machine data	general	5000 - 5399	MDNBI <byte> <bit>	
	channel-specific axis-specific spindle-specific compensation FLAGS	5400 - 5599 5600 - 5799 5800 - 5999 6000 - 6999	MDNBY <byte>	
drive machine data	feed	244x - 252x	MDDBY <word> <byte>	
	feed with spindle function	4610 - 4630	MDDBI <word> <byte> <bit> x axis address	
cycles machine data	channel-specific standard cycles	800 - 849	MDZBI <channel> <byte> <bit>	
	channel-specific customer cycles	900 - 949	MDZBY <channel> <byte>	
	central standard cycles	7000 - 7049		
	central customer cycles	8000 - 8049		
PLC machine data	operating system	2000 - 2999	MDPBI <byte> <bit>	
	customer program	3000 - 3999	MDPBY <byte>	
	for 850/880	operating system standard FBs customer program	6000 - 6999 7000 - 7999 8000 - 8999	
	for 805	(in preparation)	(in preparation)	
	for 840	(in preparation)	(in preparation)	

2.3.2 Setting data

Values assigned to the setting data describe the current machine status or the workpiece machining. If setting data is modified, the modifications become immediately effective. Similar to the machine data, the setting data is subdivided into NC-, cycle- and PLC setting data. It is further subdivided into the following areas within these groups.

Setting data words:

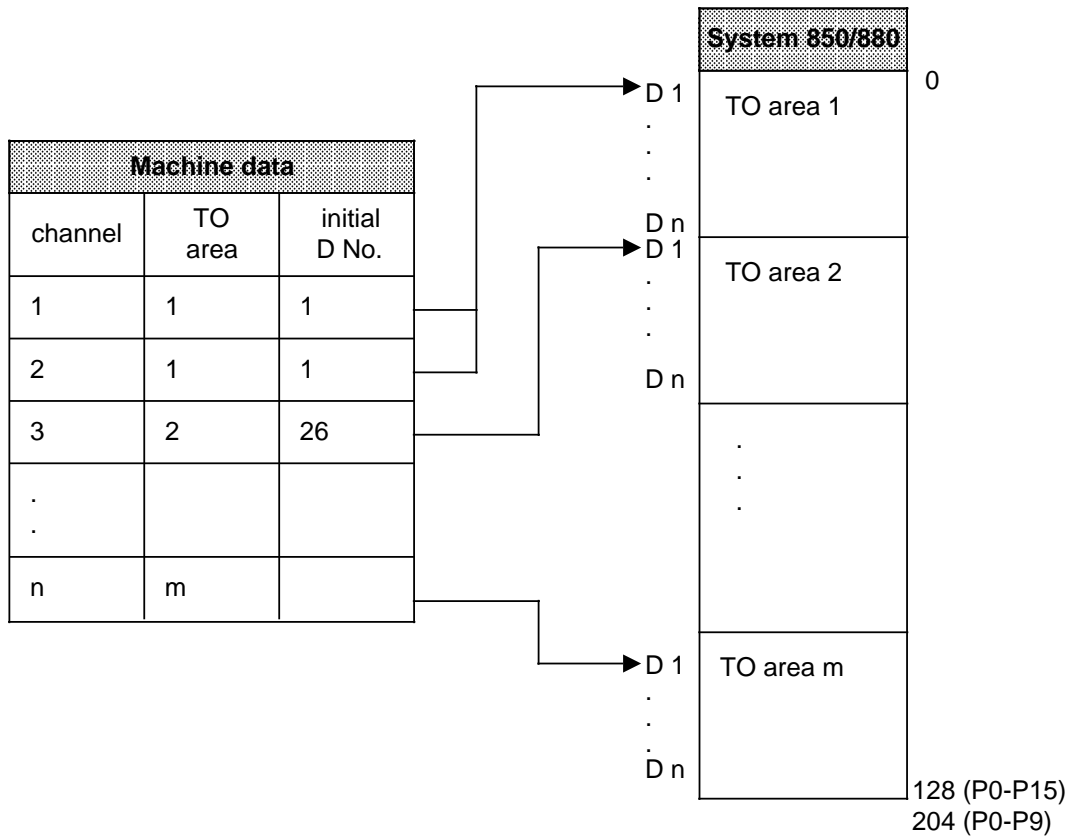
Group	Area		CL 800 word FB parametrization
NC setting data	general	0 - 1999	SEN <word>
	channel-specific	2000 - 2999	
	axis-specific	3000 - 3999	
	spindle-specific	4000 - 4999	
cycles setting data	channel-specific cycles	0 - 99	SEZ <channel> <word>
	standard cycles		
	channel-specific customer cycles	400 - 499	
PLC setting data for 850/880	operating system	0 - 1999	SEP <word>
	standard FBs	2000 - 3999	
	customer program	4000 - 5999	
for 805	(in preparation)		(in preparation)
for 840	(in preparation)		(in preparation)

Setting data bits:

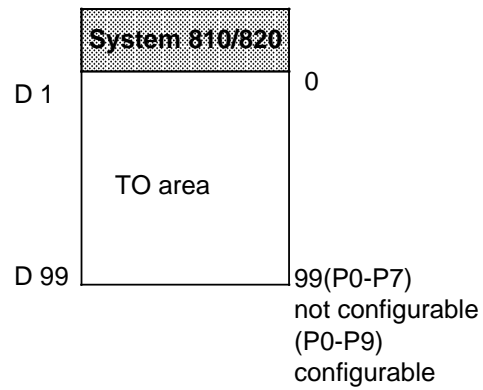
Group	Area		CL 800 word FB parametrization
NC setting data	general	5000 - 5199	SENBI <byte> <bit>
	channel-specific	5400 - 5595	SENBY <byte>
	axis-specific spindle-specific	5600 - 5799 5800 - 5999	
cycles setting data	channel-specific standard cycles	800 - 849	SEZBI <channel> <byte> <bit>
	channel-specific customer cycles	900 - 949	SEZBY <channel> <byte>
PLC setting data for 850/880 for 805 for 840	operating system standard FBs customer program	6000 - 6999 7000 - 7999 8000 - 8999	SEPBI <byte> <bit> SEPBY <byte>
	(in preparation)		(in preparation)
	(in preparation)		(in preparation)

2.3.3 Tool offsets

A max. of 99 (810) or 204 (850/880) tool offset blocks for active tools can be stored in the NC. With SINUMERIK system 850/880, machine data can be used to define the tool areas assigned to the tool magazines or turrets located at the machine.



With SINUMERIK system 810/820 there is only one TO area (0):



TO area for SINUMERIK system 805:

(in preparation)

TO area for SINUMERIK system 840:

(in preparation)

The individual TO areas are structured as follows:

	T No.	Type	Geometry			Wear			Add. TO		P10	..	P15
	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9			
D1													
D2													
Dn													

Columns P0 to P9 (P11) are permanently assigned, while columns P12 to P15 can be assigned according to the machine (e.g. stand time, quantity, permissible cutting force). The number of columns is defined via a machine datum.

Assignment of columns P0 to P9 of the TO areas :

- Tool No. P0
 T No. with up to 8 decades
- Coding of tool type P1
 Type with 2 decades. The tools are subdivided into the following groups according to type:
 - Type 1..9 : Tools with offsets X, Z and radius, e.g. lathe tools
 - Type 10 : Tools with length compensation, e.g. drills
 - Type 20 : Tools with length and radius compensation, e.g. shank cutters
 - Type 30 : Tools with 2 length and 1 radius compensation, e.g. angle milling cutters
 - Type 40 : Tools with 5D length compensation
 - Type 50 ..59 : Grinding wheel with cutting position as type 1 to 9 with grinding-related monitoring.

Decade 10° specifies the direction of the selected compensation plane in which the compensation values are to be effective.

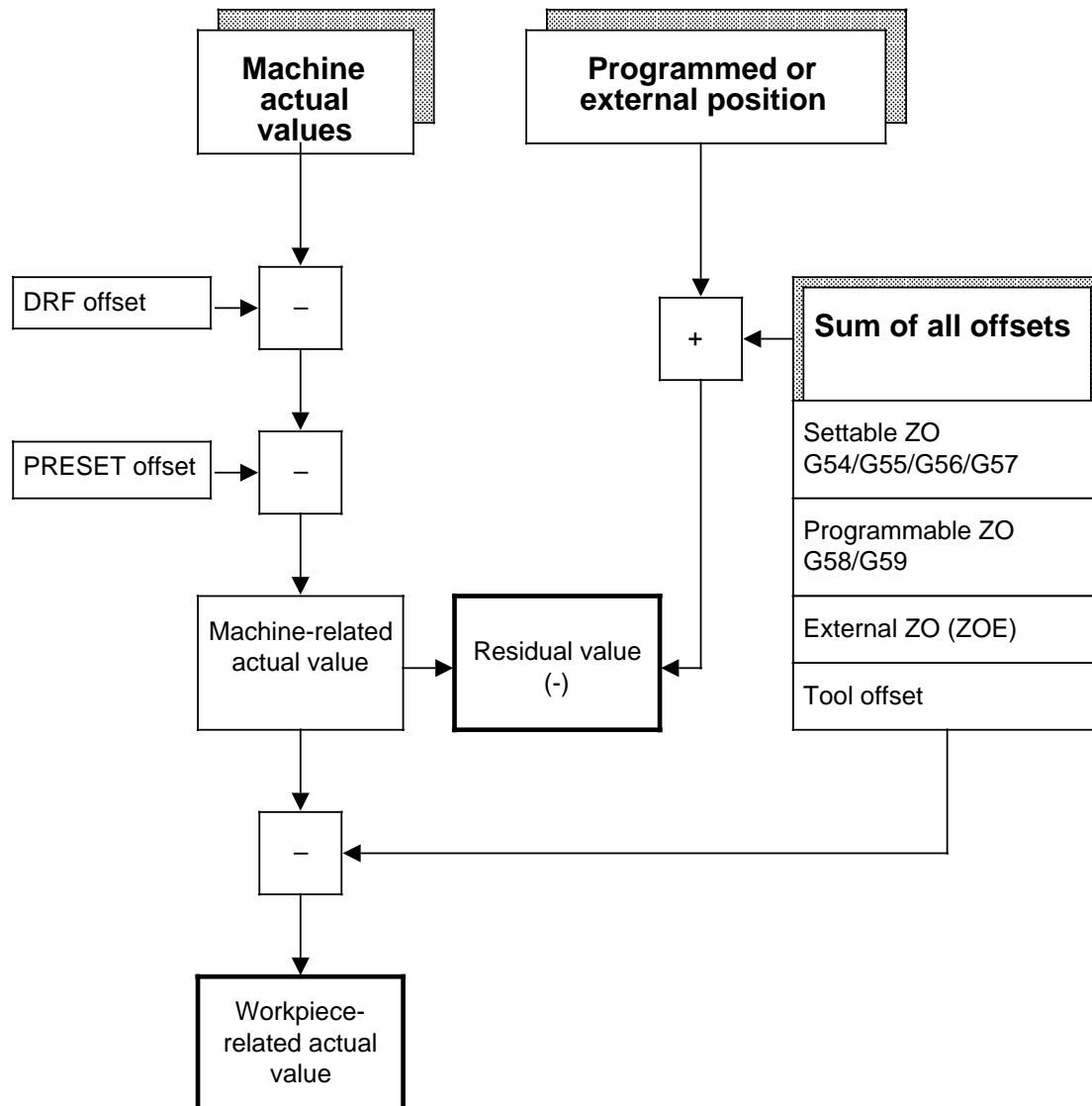
- Assignments P2 to P9 are dependent on the tool type as follows:

Type P1	Geometry			Wear			Add. TO	
	P2	P3	P4	P5	P6	P7	P8	P9
1 ... 9	length X	lgth Z	radius	length X	length Z	---	length X	length Z
10 ... 19	length	---	---	length	---	---	length	---
20 ... 29	length	---	radius	length	---	radius	length	---
30 ... 39	length 1	lgth 2	radius	length 1	length 2	radius	length 1	length 2
40 ... 49	length differ.	---	---	---	---	---	---	---
50 ... 59	length 1	lgth 2	radius	length 1	length 2	radius	length 1	length 2

Values P2/P5/P8, P3/P6/P9 as well as P4/P7 are added in accordance with one item of machine data.

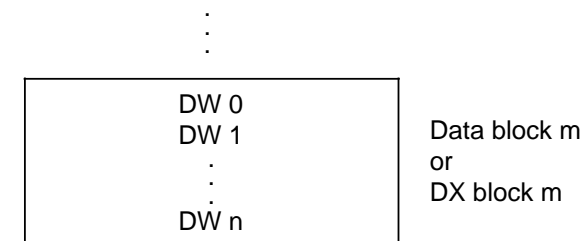
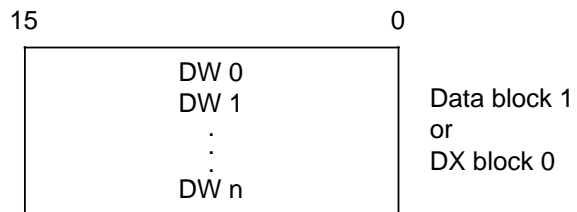
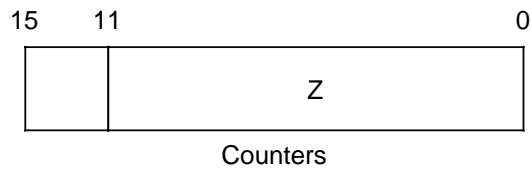
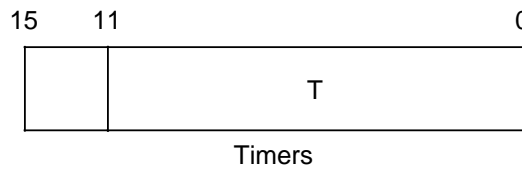
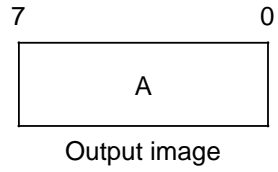
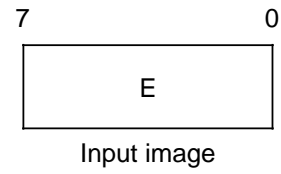
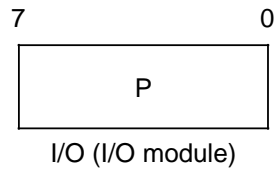
2.3.4 Zero offsets

Per NC axis, the zero offsets and tool offsets (TO) are allowed for as follows:

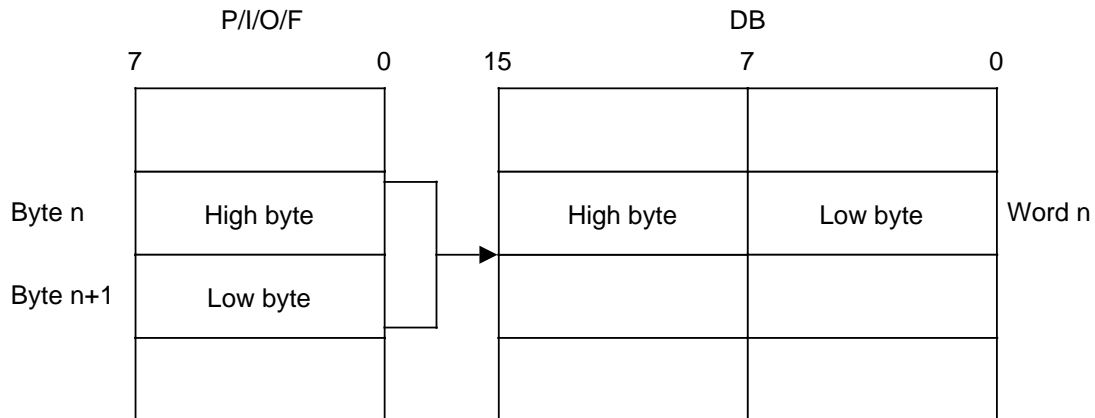


2.3.5 PLC signals

The PLC signals are subdivided into the following areas:



For byte or word addressing of PLC data words, the following assignments must be observed:



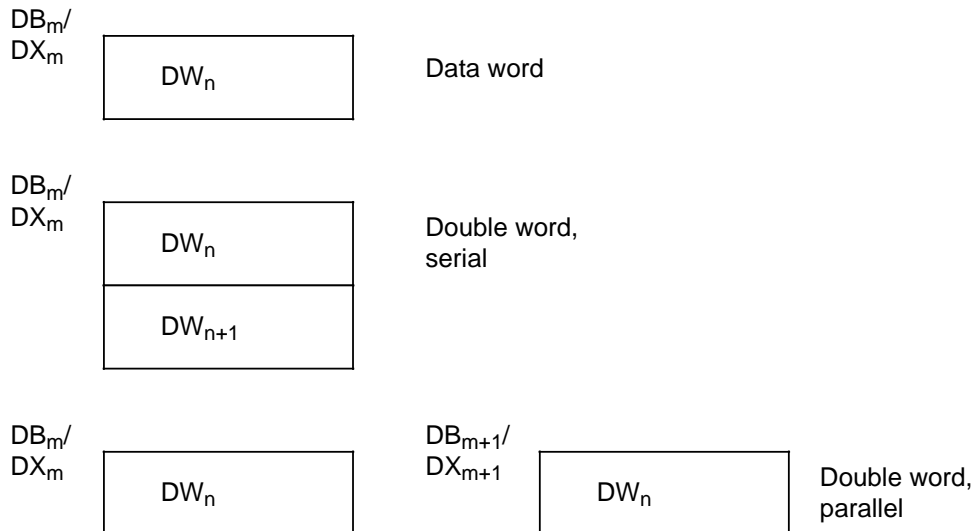
A byte consists of 8 associated bits with numbers 0...7.

A word is created by combining two bytes. Thus, the word contains 16 bits (0...15). The byte which has bit "0" in the word is designated as the "low byte", and the byte with bit "15" as the "high byte".

The following formats are possible for PLC data words:

- *Fixed point:*

A data word or double word (serial or parallel) is read into, or loaded into the PLC.



Dimension identifier:

The dimension identifier specifies the position of the decimal point for data transfer with fixed point.

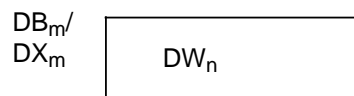
<Value>	Fixed point: Data word or double word, serial
0	value without decimal point
1	value with decimal point
2	1 digit after the decimal point
3	2 digits after the decimal point
4	3 digits after the decimal point
5	4 digits after the decimal point
6	5 digits after the decimal point
7	6 digits after the decimal point
8	7 digits after the decimal point
9	8 digits after the decimal point

<Value>	Fixed point: Double word, parallel
10	value without decimal point
11	value with decimal point
12	1 digit after the decimal point
13	2 digits after the decimal point
14	3 digits after the decimal point
15	4 digits after the decimal point
16	5 digits after the decimal point
17	6 digits after the decimal point
18	7 digits after the decimal point
19	8 digits after the decimal point

- *BCD*:

Data is read or loaded in serial or parallel, dependent on the defined number of data words.

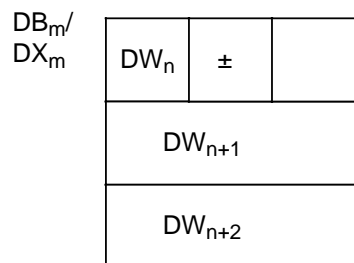
- Number of data words= 1 : ==> one data word is read/loaded



- Number of data words= 2 : ==> two data words are read/loaded in parallel



- Number of data words= 3 : ==> three data words are read/loaded serially



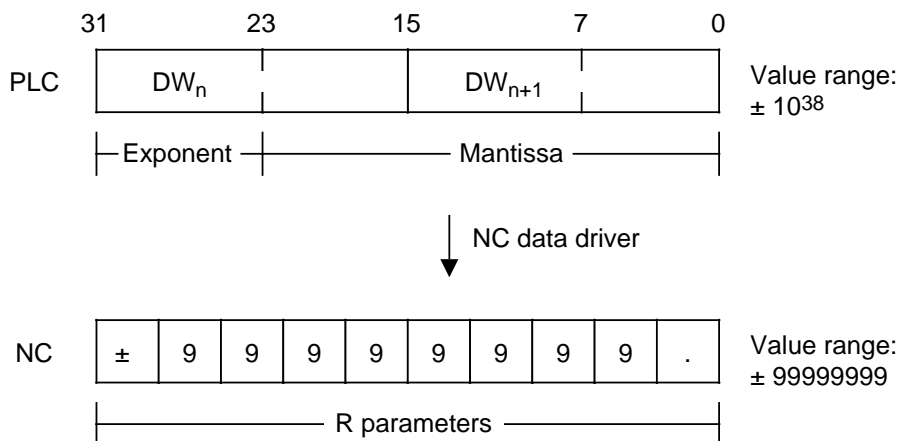
Dimension identifier:

Dimension identifier specifies the position of the decimal point for data transfer with BCD.

<Value>	BCD
100	value without decimal point
101	value with decimal point
102	1 digit after the decimal point
103	2 digits after the decimal point
104	3 digits after the decimal point
105	4 digits after the decimal point
106	5 digits after the decimal point
107	6 digits after the decimal point
108	7 digits after the decimal point
109	8 digits after the decimal point

- *Floating point:*

Two data words are always serially read/loaded into the PLC. In this case, a data driver realizes the conversion into the different NC and PLC formats.



General comments:

- NC alarm "CL 800 error" is initiated for overflow NC==> PLC at writing (e.g. 1 bit is written with R parameter contents 2).
- An overflow PLC ==> NC at reading causes the maximum value to be entered into the R parameter and an NC alarm issued, which does not interrupt the program. The alarm can then be evaluated in the program.
- A PLC machine data bit is used to determine whether PLC data can be written via CL 800 statements of the NC program or not.
- The NC alarm "CL 800 error" is issued for access to data which are not available in the PLC through CL 800 statements of the NC program.

Section 3

-Language notation-

Overview:

3.1 General

3.2 CL 800 words

3.3 Operands

3 Language notation

3.1 General

The cycle language for system 800 (abbreviated CL 800) is a higher-level programming language based on Pascal. Commands are specified by means of terms derived from plain English.

In order to rule out any ambiguities and to allow compilation into an instruction code by the compiler, the language is subject to strict rules which are summarized as a syntax. If these rules are violated, they are generally identified by the system (fault messages of the cycle editor, Section 9).

3.2 CL 800 words

CL 800 words are represented in uppercase letters;
e.g.: BEGIN, LOCAL, GOTO, SIN, PRAP.

They must be specified in the relevant commands as mandatory requirement. A CL 800 word list is provided in the Appendix under Section 11, alphabetic keyword index.

3.3 Operands

Variables and constants with which the commands operate are known as operands. A distinction is made between the following operands:

R parameter : <Var name>
 Pointer : [<Pointer name>]
 System memory : <System memory name>(<Value 1>, <Value 2>, . . .)

Constant (digit) : <Constant name>
 Number : Real or integral number

Significance of the different brackets and parentheses	
()	Parentheses The parentheses are part of the language. They must be entered wherever specified, and must be filled with the current parameters.
< >	Angle brackets Angle brackets are not part of the language. The enclosed expression must be replaced by the current parameter and specified as a mandatory requirement
[]	Square brackets Square brackets are only part of the language in the case of a pointer variable. Otherwise the enclosed expression need not be specified as a mandatory requirement.
{ }	Braces Braces are not part of the language. The enclosed expression may be specified any number of times or not at all.

Section 4

-Structure of the CL 800 high-level language-

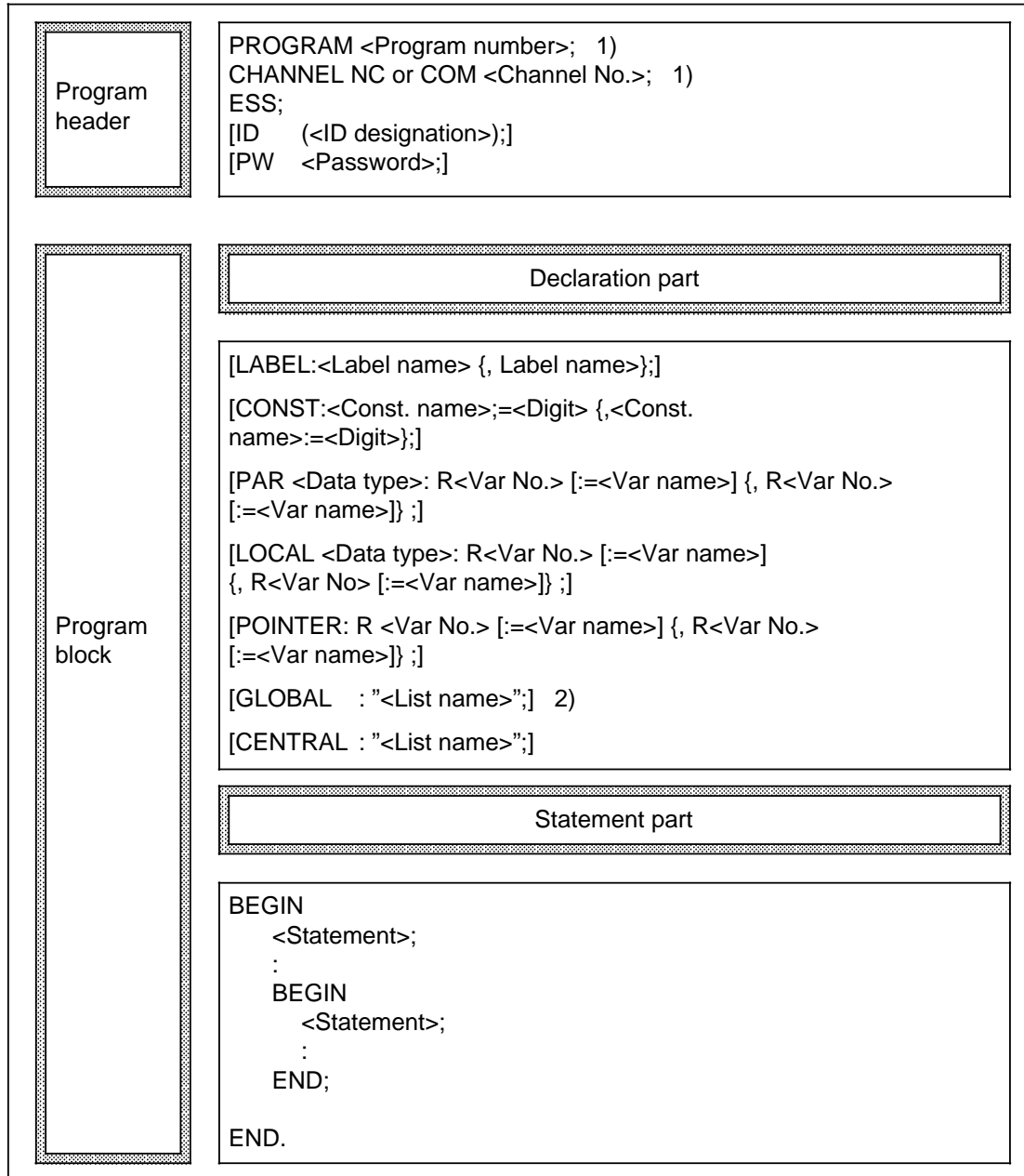
Overview:

4.1 Program structure

4.2 Data structure

4 Structure of the CL 800 high-level language overview

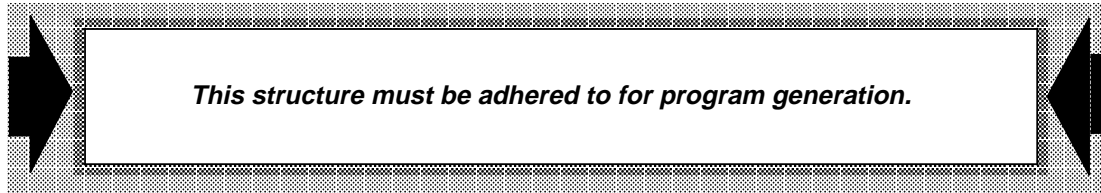
4.1 Program structure



Structure of a CL 800 program

- Legend:**
- 1) Program number and channel number must be at least specified with zero.
 - 2) Two lists can be defined for global variables; a channel-independent list (necessary when subroutines are to be run on several channels), and a channel-dependent list.

The structure of a CL 800 program consists of a program header and a program block. The program block is further subdivided into a declaration part, in which all objects available in the program are defined, and a statement part, in which the actions are specified which are to be executed with these objects.



Example of program structuring

The program variables R 50 should be assigned the instantaneous value of term 1, term 2 and term 3 from a higher-level program in the NC channel.

(* program header *)

```
PROGRAM 1;  
CHANNEL NC 1;  
ID (0-ASSIGN-01.03.85-KUB-SIEMENS:850:02);
```

(* declaration part *)

```
PAR INTEGER: R0: =term_1,      (* transfer parameter *)  
              R25: =term_2,  
              R49: =term_3;  
LOCAL INTEGER: R50:=value;    (* local parameter *)
```

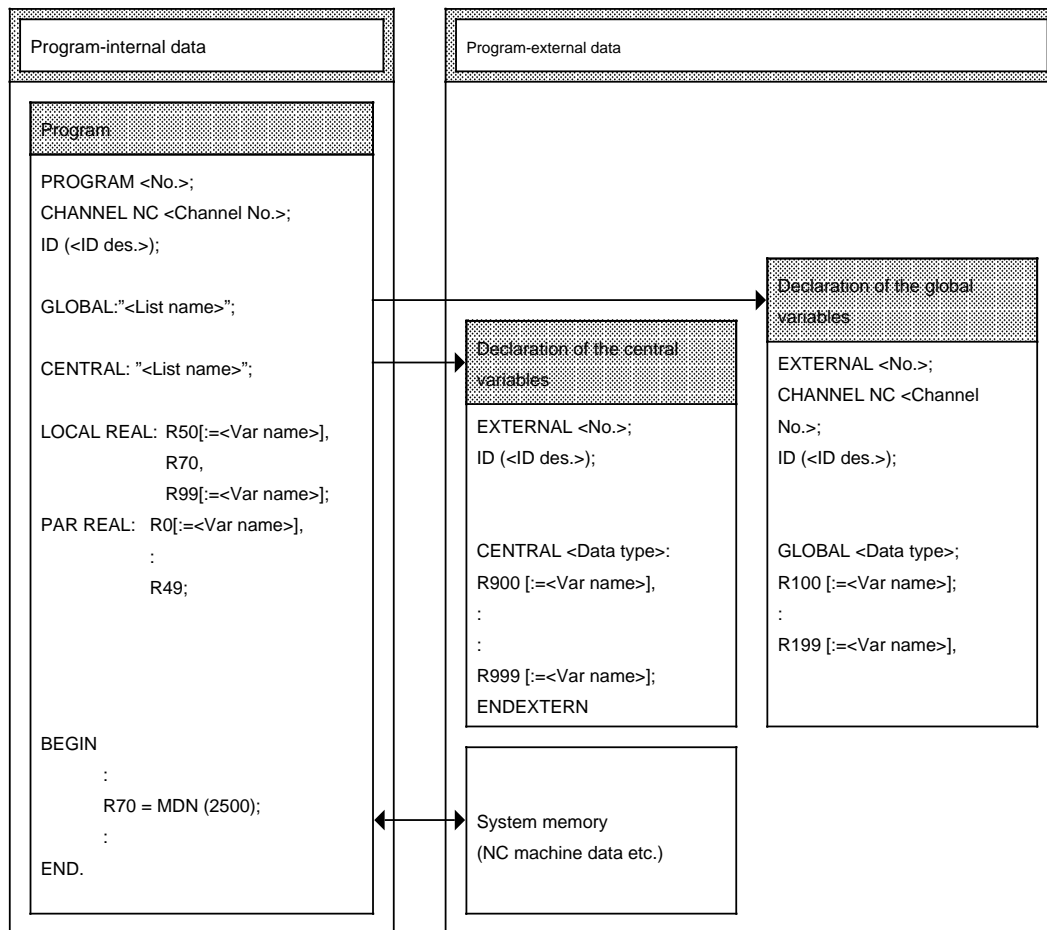
(* statement part *)

```
BEGIN  
  value = term_1 + term_2 +term_3;    (* first statement *)  
  :                                  (* additional statement *)  
END.
```

4.2 Data structure

The data, with which the cycles operate, is subdivided into two areas:

- **Program-internal data**
 - Transfer variable (R0-R49)
 - Local variable (R50-R99)
- **Program-external data**
 - Global variable (R100-R199)
 - Central variable (R900-R999)
 - System memory



Section 5

-Command description-

Overview:

5.0 Preliminary information

5.1 Program frame statements

5.1.1 General statements

5.1.2 Program header

5.2 Declarations

5.2.1 Declaration of variables

5.2.2 Declaration of variables in external lists

5.3 Statements

5.3.1 Repeat statements

5.3.1.1 REPEAT loop

5.3.1.2 WHILE loop

5.3.1.3 WHILE INT loop

5.3.1.4 FOR TO loop

5.3.1.5 FOR DOWNTO loop

5.3.2 Decision statements

5.3.2.1 IF THEN ELSE branching

5.3.2.2 IF INT THEN ELSE branching

5.3.2.3 Case branching

5.3.3 Unconditional branching

5.3.3.1 Unconditional jump

5.3.4 General data transfer

5.3.4.1 Data transfer: R par./R par.

5.3.4.2 Data transfer: R par./input buffer memory for numerical variable

5.3.5 Data transfer: System memory into the R parameter

5.3.5.1 Transfer machine data into the R parameter

5.3.5.2 Transfer setting data into the R parameter

5.3.5.3 Transfer tool offsets into the R parameter

5.3.5.4 Transfer zero offsets into the R parameter

5.3.5.5 Read programmed setpoints into the R parameter

5.3.5.6 Read actual values into the R parameter

5.3.5.7 Read program data into the R parameter

5.3.5.8 Read PLC signal bits into the R parameter

5.3.5.9 Read PLC signal bytes into the R parameter

5.3.5.10 Read PLC signal words into the R parameter

5.3.5.11 Read PLC signal data words into the R parameter

5.3.5.12 Read alarms into the R parameter

5.3.5.13 Read alarm pointer into the R parameter

5.3.5.14 Read system memory into the R parameter

5.3.6 Data transfer: R parameter into the system memory

5.3.6.1 Transfer R parameter into the machine data

5.3.6.2 Transfer R parameter into the setting data

5.3.6.3 Write R parameter into the tool offsets

5.3.6.4 Write R parameter into the zero offsets

5.3.6.5 Write R parameter into the program setpoints

Section 5

-Command description-

- 5.3.6.6 Write R parameter into the PLC signal bits
- 5.3.6.7 Write R parameter into the PLC signal bytes
- 5.3.6.8 Write R parameter into the PLC signal words
- 5.3.6.9 Write R parameter into the PLC signal data words
- 5.3.6.10 Write R parameter into the alarms
- 5.3.6.11 Write R parameter into the system memory
- 5.3.7 Mathematical and logical functions
 - 5.3.7.1 Value assignment with arithmetic operations
 - 5.3.7.2 Arithmetic functions
 - 5.3.7.3 Arithmetic procedures
 - 5.3.7.4 Trigonometric functions
 - 5.3.7.5 Logarithmic functions
 - 5.3.7.6 Logical functions
 - 5.3.7.7 Logical procedures
- 5.3.8 NC-specific functions
 - 5.3.8.1 Changing program and machine reference points
 - 5.3.8.2 Individual functions
 - 5.3.8.3 Measurement functions
 - 5.3.8.4 Program influence
- 5.3.9 I/O statements
 - 5.3.9.1 I/O functions, NC
 - 5.3.9.2 I/O functions, general
 - 5.3.9.3 Operator control functions

5 Command description

5.0 Preliminary information

The syntax of CL 800 is described with words in this section, which also includes examples for clarification.

5.1 Program frame statements

5.1.1 General statements

BEGIN . . . END.	statement part
-------------------------	-----------------------

These commands are used to start and terminate the complete **statement part**. The program is executed with its various statements and statement blocks within these boundaries.

Example:

```
BEGIN
:
  <Statement part>;
:
END.
```

BEGIN . . . END;	statement block
-------------------------	------------------------

Several statements are combined to form a **statement block** using these commands. For all commands with only one permissible statement, then this can be replaced by a statement block.

Example:

```
BEGIN
:
  <Statement(s)>;
:
END;
```

LF;	end of block
------------	---------------------

The code generator of the cycle compiler generates a compressed object code which is automatically limited to a maximum length of 120 characters per block. Any "CR LF" (<--!) entered in the CL 800 program as an end of line is not transferred to the object code as an end of block.

However, the programmer has a facility in the CL 800 program for issuing an end of block in the object code by inserting "LF;" at specific points. All block limits additionally required are deliberately inserted, e.g. before and after programmed axis movements and after subroutine calls.

Example:

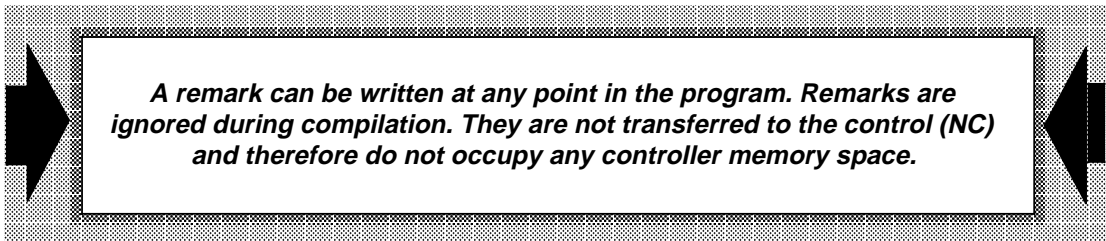
```
BEGIN
:
<Statement 1>;LF;
<Statement 2>;LF;
:
END.
```

(*<Remarks>*)	remarks
----------------------------	----------------

A **remark** of any length can be entered within the parenthesized expression and the two asterisks (* . . *). All principle characters and CR/LF are allowed. Readability of the program can be considerably improved by inserting remarks.

Example:

```
BEGIN
:
<Statement>;      (* plunge grinding in Z *)
:
END.
```



A remark can be written at any point in the program. Remarks are ignored during compilation. They are not transferred to the control (NC) and therefore do not occupy any controller memory space.

(:<DIN-CODE>:);

DIN command

With this statement, DIN commands which are not part of cycle language can be inserted in a CL 800 program.

The **DIN code** may only be used within the statement part. It is treated as a statement and must therefore be terminated with a semicolon.

Examples:

(: N100 G90 G01 F=R25 X=R91 Z500 :);

(: (STARTING POINT X):);

(: L50 P1 :);

Block numbers N1-999 are available to the programmer in the (<DIN CODE>:) The commands within the statement (<DIN CODE>:) are not checked by the compiler for correct syntax.

5.1.2 Program header

For CL 800, the program header is only optional and has absolutely no significance for the program. If available, it provides the program with a name and optionally lists the parameters through which the program communicates with the environment.

PROGRAM <Prog. number>;

definition of a program

A subroutine (cycle) is assigned a program number (1-999/9999) with this command. The program is entered in the control (NC) under this number (L 1 . . . L999/L9999). The number must be specified in order to call the program from other programs.

Example:

PROGRAM 100;

:

END.

PROGRAM 999;

:

END.

A subroutine is called up in a CL 800 program using a DIN command (refer to Section 5.1.1).

CHANNEL NC <Channel No.>;

channel number

Using the channel designation, the programmer defines the channel in which the program may be called. **Channel numbers 0 . . . 16** may be used for the NC area, depending on the control and degree of expansion. If a program is to be executable in several NC channels, 0 is preset as the channel number.

Example:

CHANNEL NC 2;



The program and channel numbers must be preset in each program.

ESS;

enable for softkey start

A subroutine start via softkey can be configured with the softkey function 69. The program enable for softkey start is realized with the statement **enable softkey start**.

Example:

```
PROGRAM 1;  
CHANNEL NC 1;  
ESS;  
ID (0-EXAMPLE-22.07.85-KUB-SIEM:MA:850:02);
```

ID (<Ident. des.>);	ID designation
----------------------------------	-----------------------

The **ID designation** allows the user to make a clear distinction between his program and other programs.

The following program identifiers are defined with the **ID designation**:

- 1) Consecutive number : 1 character
- 2) File designation : 8 characters
- 3) Date : 8 characters DD.MM.YY
- 4) Programmer : 3 characters
- 5) Object designation : e.g. 8 characters
- 6) Control type : e.g. 4 characters
- 7) Release : e.g. 2 characters

Explanations of individual terms of the ID designation:

- 1) The "consecutive number" is an internal revision identifier for the program. If the program is modified the consecutive number is incremented by 1 by the programmer (refer to example: 0).
- 2) The name of the file called is entered as the "file designation" (refer to example: EXAMPLE).
- 3) The creation date of the program is entered under "date" by the programmer (refer to example: 22.07.85).
- 4) The programmers initials are entered under programmer (refer to example: KUB).
- 5) With the "object designation" the user can describe the machine name, for example, manufacturers name etc. (refer to example: SIEM:MA).
- 6) The "control type" serves to enter the NC control in use, (refer to example: 850).
- 7) Under release an incrementing number can be entered for program revisions (refer to example: 02).

Example:

ID (0-EXAMPLE 22.07.85-KUB-SIEM:MA:850:02);

Terms 1-4 are fixed subdivisions of the ID designation. They must be entered by the user in the same sequence and with the same number of characters. Hyphens (-) must be entered as separators between the terms. Terms 5-7 are additional data with a maximum of 14 characters and can be handled flexibly. They are only a recommendation. The ID designation comprises a maximum of 40 characters.

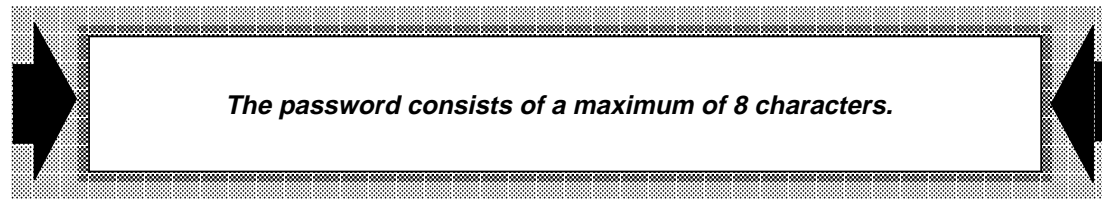
PW (<Password>);	password
-------------------------------	-----------------

The purpose of the **password** is to protect a program in the NC from access by unauthorized persons. When a password has been declared, the program can only be read out if the password is previously entered. This function is not implemented at present in the NC.

The password is a protection facility for the program, but need not be specified as a mandatory requirement when the program is written.

Example:

PW (KEY28);



5.2 Declarations

The declaration part in a program block declares all designators which are used in the statement part. The declarations are terminated with a semicolon.

5.2.1 Declaration of variables

The variables are subdivided into two main groups corresponding to the CL 800 data structure (Section 4.2):

- External variables which are declared in separate lists (external files):
 - global variables **GLOBAL** (R100-R199)
 - central variables **CENTRAL** (R900-R999)
- Internal variables and constants which are declared within the program itself:
 - transfer variables **PAR** (R00 - R49)
 - local variables **LOCAL** (R50 - R99)
 - constants **CONST**
 - pointer variables **POINTER** (R50 - R99)
 - branch labels **LABEL**

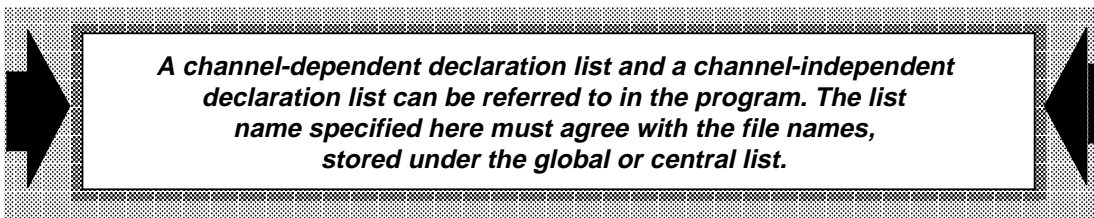
The reference to global and central variables, which are declared in separate lists, is programmed as follows:

CENTRAL: "<list name variant>";	central variable
GLOBAL: "<list name variant>";	channel-dependent global variable
GLOBAL: "<list name variant>";	channel-independent global variable

The reference to the separate lists is given by the **list names**, which may have a maximum length of 8 characters.

Example:

```
CENTRAL: "VARCEN";
GLOBAL:  "VARKUP";
GLOBAL:  "VARKAP";
```



PAR <Dat type>: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]};	transfer variable
--	--------------------------

The **transfer variables** (range R0-R49) are declared with this statement.

All parameters can be assigned a symbolic name, which must not be used within the DIN code.

Example:

```
PROGRAM 1; (* program header *)
:
PAR INTEGER: R00:= term_1,
             R25:= term_2; (* declaration part *)
PAR REAL:   R49:= term_3;
LOCAL REAL: R50:= VALUE;

BEGIN
  VALUE = term_1 + term_2 + term_3; (* statement part *)
END.
```

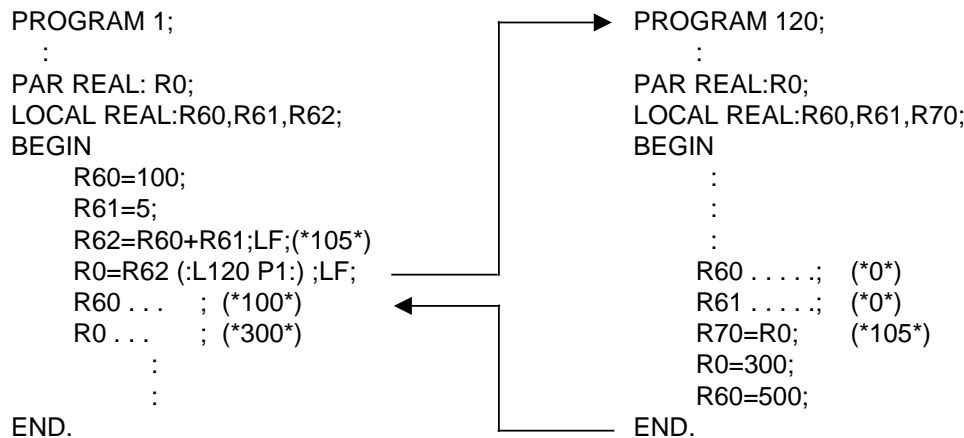
All transfer variables used in (:<DIN-CODE>:) need not be declared.

LOCAL <Dat type> : R<Var No.>[:=<Var name>] local variable
{,R<Var No.>[:=<Var name>]};

The **local variables** (range R50-R99) are declared in the program with this statement. The **local variables** can be assigned different values at each program level, and can be given new symbolic names. When the program level is exited because of a subroutine call, the **local variables** are saved and preassigned **zero** for the following program. Thus, they are available again as independent arithmetic parameters in the called subroutine. After return to the calling program, the saved values are available there again.

Example:

Mode of operation of the transfer and local variables



CONST <Const. name>:=<Number> constant
{,<Const. name>:=<Number>} ;

Constant numerical values are assigned symbolic names with this statement.

Constant values are often inconvenient in a program. They can thus be replaced by symbolic names (e.g. PI instead of 3.14159).

Example:

```

PROGRAM 3;
:
LOCAL REAL: R60, R61;
CONST : PI:=3.14159, MILL:=1000000;
(* PI=real number, MILL=integral number *)
:
  R60 = R 61 * PI;
:
END.

```

In the declaration of the constant, the data type is specified by the value of the constant, i.e. a number without a decimal point is treated as an integral number, and a number with a decimal point is treated as a real number.

```

POINTER R<Var No.>[:=<Var name>] pointer variable
      {,R<Var No.>[:=<Var name>]};

```

Statement for declaring parameters (R50 - R99) as **pointer variable (pointer)**.

The contents of the pointer determines the address of the R parameter to be used.

Example:

```

PROGRAM 3;
:
POINTER: R50, R51;
LOCAL INTEGER: R70, R71, R80;
:
BEGIN
:
  R50 = 70;      (* load address of an R parameter *)
  [R50] = 20;   (* contents of R70 = 20 *)
:
:
  R51 = 80      (* load address of an R parameter *)
  R71 = [R51];  (* contents of R80 are transferred to R71 *)
:
END.

```

Variables, which are declared as pointers are placed in square brackets for indirect addressing.

LABEL : <Name> {,<Name>};

label for jump destination

With the **label** declaration, names are declared which are used in the program as markers for jump destinations of the GOTO command.

The name of a label can have up to 8 characters; the first character must be a letter.

Example:

LABEL : Error;

Also refer to unconditional jump (GOTO) Section 5.3.3.1

5.2.2 Declaration of variables in external lists

EXTERNAL <List number>;

**definition of an
external file**

ENDEXTERN

A declaration list is assigned a list number (1 . . . 9999) with the **EXTERNAL** statement. If required, the list is read into the control (NC) as a "subroutine" with this number.

The creation of declaration lists takes place separately from the CL 800 program at the configuring station.

The **EXTERNAL** statement is terminated with **ENDEXTERN**.

Examples:

```
EXTERNAL 30;
ID (0-VARICEN-28.03.85-KUB- . . . );
PW (SECRET5);
:
(* declaration of central variables *)
:
ENDEXTERN
```

```

EXTERNAL 35;
CHANNEL NC 0;
ID (0-VARIKUP-03.04.85-KUB- . . . );
PW (PARAME1);
:
(* declaration of channel-independent global variables *)
:
ENDEXTERN

```

During creation of an EXTERNAL list, a list number must be specified. If the list is created for global parameters, the channel number must be additionally specified. Remarks may also be used within these declaration lists. The channel number must be specified as 0 for the channel-independent global variables, so that the list can be used for parameters in different channels. In the ID designation, the "file designation" must agree with the "file names" of the configuring station file. These file names are referred to in the program.

```

GLOBAL <Dat type>: R<Var No.>[:=<Var name>]                global variable
                  {,R<Var No.>[:=<Var name>]};

```

Global variables (R100-R199) are declared in external lists (EXTERNAL files) with this statement. The lists must already exist when they are called in the program.

Global variables are present once per channel. They can be accessed by all programs which are executed within a channel. **Global variables** R100-R149 are declared in a separate list for each channel. The declaration of **global variables** R150-R199 takes place jointly in one list for all channels.

Example:

Channel-dependent parameter list 1	Channel-dependent parameter list	Channel-dependent parameter list 2
<pre>EXTERNAL 50; CHANNEL NC 1; ID (0-KAPLI1-...) GLOBAL REAL: R100:=drill_Z; ENDEXTERN</pre>	<pre>EXTERNAL 60; CHANNEL NC 0; ID (0-KAPLIST-...) GLOBAL REAL: R150:=refpl; ENDEXTERN</pre>	<pre>EXTERNAL 70; CHANNEL NC 2; ID (0-KAPLI2-...) GLOBAL INTEGER: R100:=DWELL, R101:=drill_X, R109; ENDEXTERN</pre>
<pre>PROGRAM 81; CHANNEL NC 1; ID (0 BEISP2-...); GLOBAL: "KAPLI1"; GLOBAL: "KUPLIST"; PAR REAL: R0:=transfer; LOCAL REAL: R60:=start_Z, R61:= depth; BEGIN start_Z=refpl; transfer=start_Z; depth=refpl.bore_Z; (L200 P1); END</pre>	<pre>PROGRAM 200; CHANNEL NC 1; ID (0 SP2-...); PAR REAL: R0:=transfer; LOCAL REAL: R60; BEGIN R60:=transfer; : : END</pre>	<pre>PROGRAM 82; CHANNEL NC 2; ID (0 KANAL2-...); GLOBAL: "KAPLI2"; GLOBAL: "KUPLIST"; LOCAL REAL: R60:=start_X, R61:=depth_abs; BEGIN start_X=refpl; transfer=start_Z; depth_abs=refpl.bore_X; (L200 P1); END</pre>

The channel number must be specified as zero for the list of R parameters declared channel-independent (R150-R199).

CENTRAL <Dat type>: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]};	central variable
---	-------------------------

The **central variables** are declared in the external list with this statement. **Central variables** R900-R999 are only present once for all channels in the communication RAM.

All channels can access the **central variables**. They can be used by all channels for transferring values to other channels.

Example:

```
EXTERNAL 10;
ID (0-CENLIST- . . . )
:
CENTRAL REAL: R900:=Load_1,R950:=Read_3;
(* Declaration of central variables *)
:
:
ENDEXTERN

PROGRAM 81;
CHANNEL NC 1;
:
CENTRAL: "CENLIST";
PAR REAL: R0:=Transf.;
:
BEGIN
:
:
END.
```

5.3 Statements

The statement part defines the action which is to be executed by the program as a result of statements. Each statement specifies a part of the action. In this sense, the CL 800 cycle language is a sequential program language: Statements are sequentially executed, never simultaneously. The statement part is started and finished with **BEGIN** and **END**; the statements are separated by semicolons within these boundaries.

5.3.1 Repeat statements

Repeat statements are used, as indicated by the name, for repeated execution of specific program parts. So-called program loops are realized using the statements.

The following repeat statements are discussed:

- **REPEAT ... UNTIL**
- **WHILE; WHILE INT**
- **FOR; FOR DOWNT0**

5.3.1.1 REPEAT loop

<pre>REPEAT <Statement>; UNTIL <Var> "Vop" <Value>; ("Vop": >; >=; <; <=; =; <>;)</pre>	REPEAT loop
--	--------------------

The **REPEAT** loop is a repeat statement with scanning of the repeat condition at the end of the loop.

With the **REPEAT** loop, the statement is processed first. A check is then made as to whether the loop condition has been fulfilled.

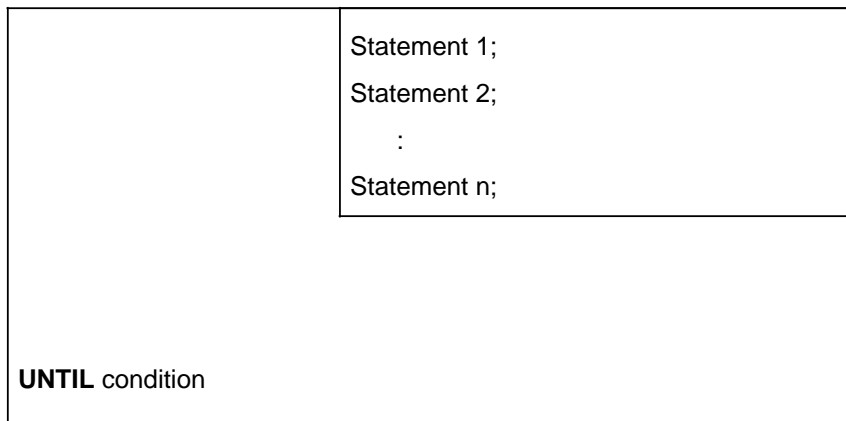
The loop statement is processed as long as the **condition is not fulfilled**.

If the **condition is fulfilled**, the following statements are processed.

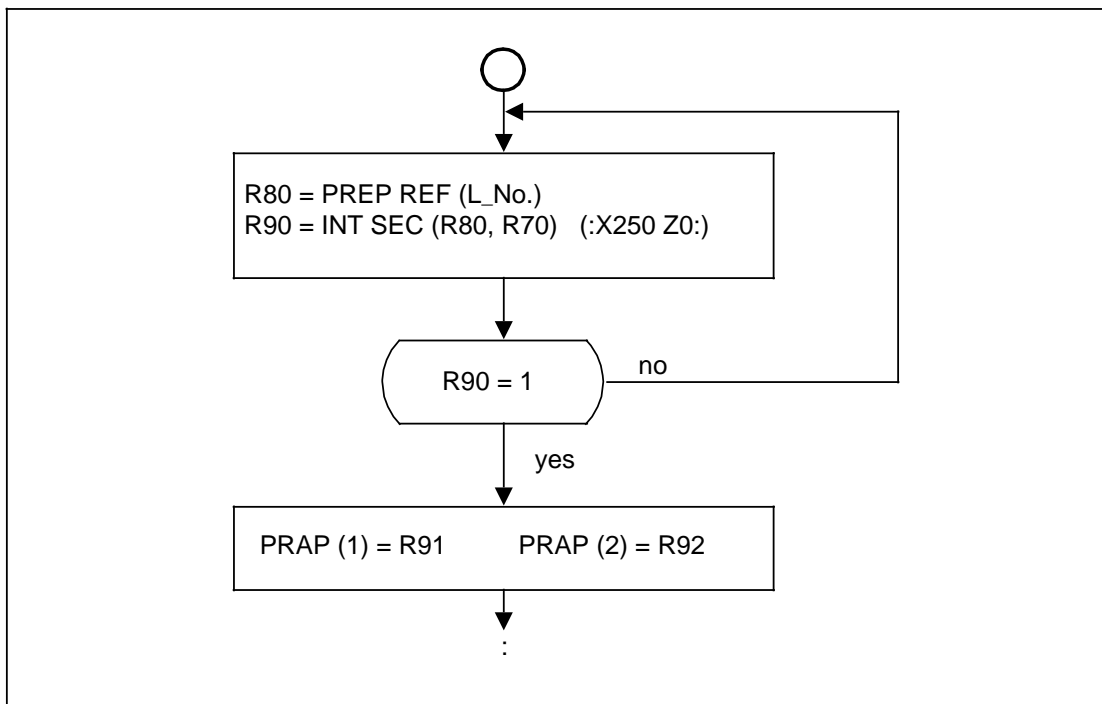
Although **REPEAT** can initiate the repetition of several individual statements, it is only interpreted as a single statement by the compiler. Thus, the identification of the statement block in the repeat loop using **BEGIN** and **END**; is not necessary. The condition can be any valid term, whose result is boolean

Generally, the **REPEAT** statement has the form:

```
REPEAT
  statement(s);
UNTIL condition
```

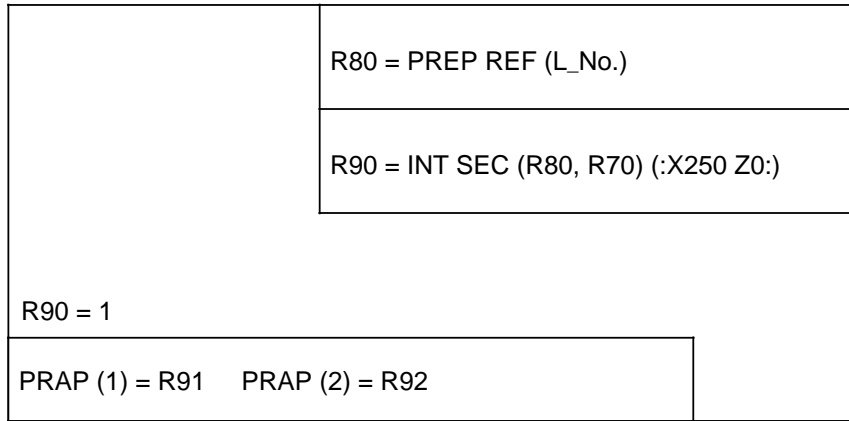
Structogram of the REPEAT statement:**Example:**

- flowchart



Flowchart: REPEAT loop

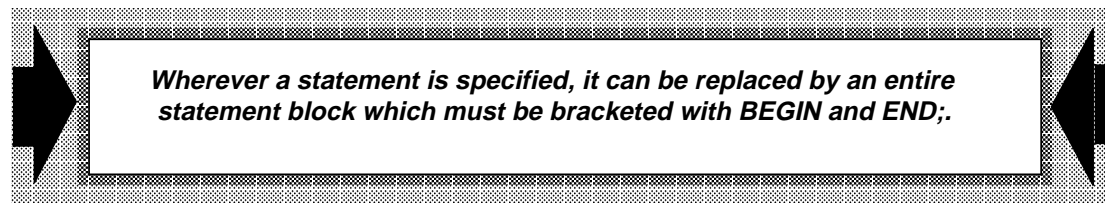
- **structogram**



Structogram: REPEAT loop

- **CL 800 programming**

```
PROGRAM 502;  
:  
LOCAL REAL: R50:=L_No, R70, R80, R90, R91, R92;  
:  
BEGIN  
:  
  REPEAT  
  BEGIN  
    R80 = PREP REF (L_No); LF;  
    R90 = INT SEC (R80,R70); (:X250 Z0);  
  END;  
  UNTIL R90=1;  
PRAP (1) = R91; PRAP (2) = R92; LF;  
:  
END.
```



REPEAT <Statement>; UNTIL <Var>.<Const>;	REPEAT loop
--	--------------------

REPEAT loop in which a program part is repeatedly processed dependent on the status of a bit in a PATTERN variable. The bit number between 0 . . . 7 is specified as <Const>.

Example:

```
PROGRAM 510;
:
LOCAL PATTERN: R70;
:
R70 = B00011001;
:
REPEAT
  <Statement>;
UNTIL R70.2;
:
END.
```

REPEAT <Statement>; UNTIL <Var>;	REPEAT loop
--	--------------------

REPEAT loop with which a statement block is processed, dependent on the status of a boolean variable. The boolean variable <Var> has the value 0 or 1.

Example:

```
PROGRAM 515;
:
LOCAL BOOLEAN: R70;
:
R70=1;
:
REPEAT
  <Statement>;
UNTIL R70;
:
END.
```

REPEAT loop with extended arithmetic

Extended arithmetic is allowed in the REPEAT loop. Extended arithmetic is understood to mean a facility for executing calculations with mathematical operations or logical functions within the statement, in accordance with Section 5.3.7.

Depending on the result of the logic operation, a program section is either processed or not processed.

Example:

```
PROGRAM 1;
:
PAR REAL:    R15, R25, R35;
PAR PATTERN: R10, R20, R30,
             R40:=INPUT;
PAR BOOLEAN: R21;
:
REPEAT
  <Statement>;
UNTIL R25 + 5 > R15;
:
REPEAT
  <Statement>;
UNTIL R10 AND R20 = R30;
:
REPEAT
  <Statement>;
UNTIL R10.3 ANDB R21;
:
REPEAT
  <Statement>;
UNTIL INPUT 3 ANDB INPUT.4;
:
REPEAT
  <Statement>;
UNTIL SIN (R25) + 0.5 > 0;
:
REPEAT
  <Statement>;
UNTIL ANGLE (15,10+R25) > R35;
:
REPEAT
  <Statement>;
UNTIL R25 > R15 ANDB R25 = R35;
:
END.
```

5.3.1.2 WHILE loop

<pre> WHILE <Var> "Vop" <Value> DO <Statement>; ("Vop": >; >=; <; <=; =; <>;) </pre>	WHILE loop
---	-------------------

The **WHILE** loop is a repeat statement with scanning of the repetition condition at the start of the loop.

With the **WHILE** loop a check is first made to establish whether the loop condition is fulfilled.

If the **condition is not fulfilled**, the statement for the loop is skipped and the following statements are processed.

The statement for the loop is processed as long as the loop **condition is fulfilled**.

If several statements are to be executed at each loop run, then these must be started and terminated with the word symbols **BEGIN** and **END**;

The **WHILE** loop is processed in the program as follows:

WHILE condition **DO**:

BEGIN

Statement 1;

Statement 2;

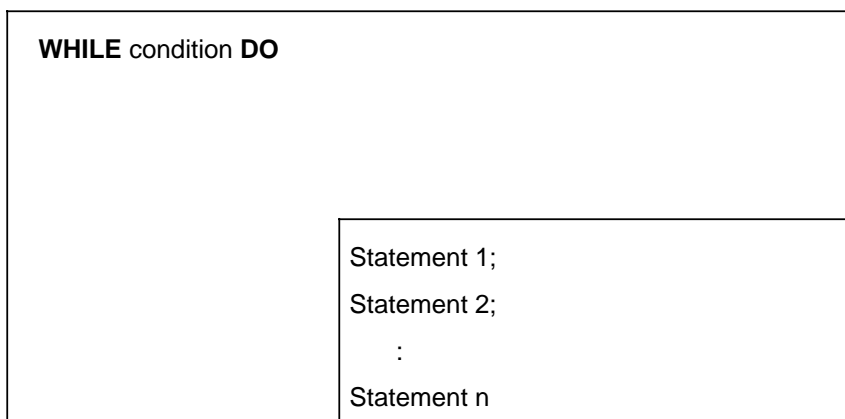
:

Statement n

END;

Each statement sequence included in **BEGIN** and **END**; counts just like a single statement as far as the syntax is concerned.

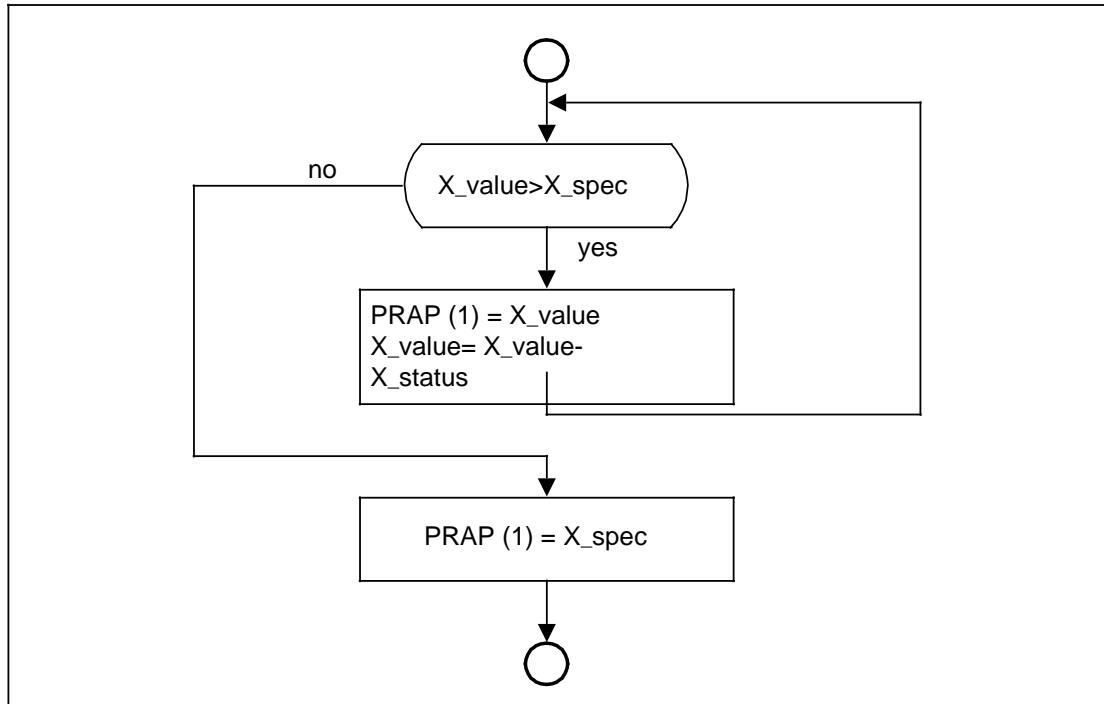
Structogram of the WHILE loop:



The key words **BEGIN** and **END** are not specified in the structogram. These are replaced by applicable graphic elements, in this case lines in the WHILE block.

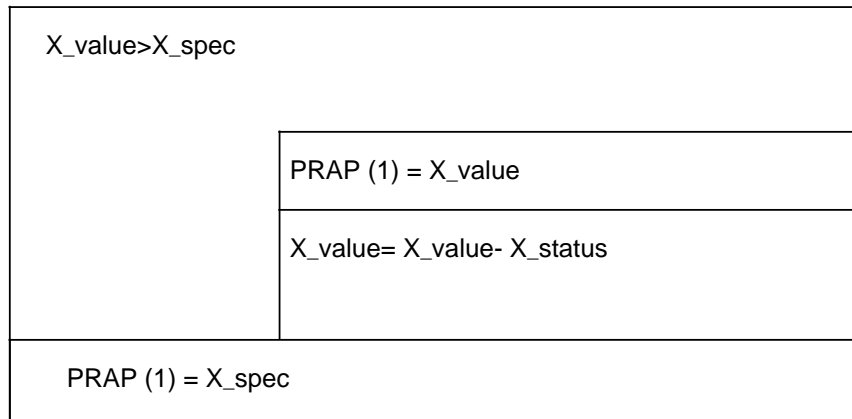
Example:

- **flowchart**



Flowchart: WHILE loop

- **structogram**



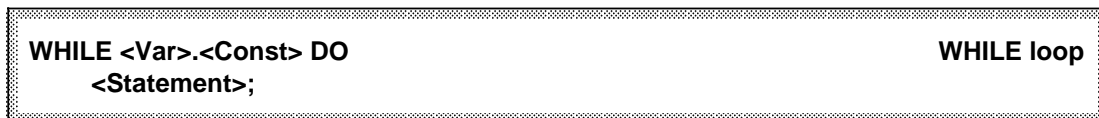
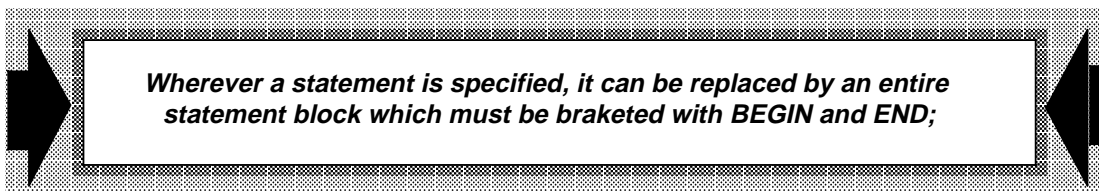
Structogram: WHILE loop

- **CL 800 programming**

```

PROGRAM 502;
:
LOCAL REAL: R80:=X_spec, R81:=X_value, R82:=X_status;
:
BEGIN
:
  WHILE X_value > X_spec DO
    BEGIN
      PRAP (1) = X_value; LF;
      X_value= X_value - X_status
    END;
    PRAP (1) = X_spec;LF;
  :
END.

```



A **WHILE** loop with which a program section is processed, dependent on the status of a bit from a PATTERN variable. Bit numbers between 0 . . . 7 are specified as <Const>.

Example:

```

PROGRAM 510;
:
LOCAL PATTERN: R70;
:
R70 = B00011101;
:
WHILE R70.2 DO
  <Statement>;
:
END.

```

**WHILE <Var>DO
<Statement>;**

WHILE loop

WHILE loop with which a statement block is processed, dependent on the status of a boolean variable. The boolean variable <Var> has the value 0 or 1.

Example:

```
PROGRAM 515;  
:  
LOCAL BOOLEAN: R70;  
:  
R70=1;  
:  
WHILE R70 DO  
  <Statement>;  
:  
END.
```

WHILE loop with extended arithmetic

Extended arithmetic is permitted within the **WHILE** loop. Extended arithmetic is understood to mean a facility for executing calculations with mathematical operations or logical functions within a statement, in accordance with Section 5.3.7. Depending on the result of the logic operation, a program section is either processed or not processed.

Example:

```
PROGRAM 1;  
:  
PAR REAL: R15, R25, R35;  
PAR PATTERN: R10, R20, R30,  
             R40:=INPUT;  
PAR BOOLEAN: R21;  
:  
WHILE R25 + 5 > R15 DO  
  <Statement>;  
:  
WHILE R10 AND R20 = R30 DO  
  <Statement>;  
:  
WHILE R10.3 ANDB R21 DO  
  <Statement>;  
:  
WHILE INPUT.3 ANDB INPUT.4 DO  
  <Statement>;  
:  
WHILE SIN (R25) + 0.5 > 0 DO  
  <Statement>;  
:  
WHILE ANGLE (15,10+R25) > R35 DO  
  <Statement>;  
:  
WHILE R25 > R15 ANDB R25 = R35 DO  
  <Statement>;  
:  
END.
```

5.3.1.3 WHILE INT loop

```
WHILE INT <Value 1>.<Value 2> DO
  <Statement>;
```

WHILE INT loop

WHILE INT loop is dependent on the status of a defined external input.

The next statement is processed as long as the external input has a "1" **signal**. The loop statement is skipped and the next statement processed if the external input has "0" **signal**.

The scan can be realized by the **WHILE INT NOT** for "0" **signal**.

The byte address (1 or 2) of the external input is defined by **<Value 1>** and the bit address (0 to 7) by **<Value 2>**.

Example:

```
PROGRAM 510;
  :
LOCAL INTEGER: R70;
  :
R70 = 1;
  :
WHILE INT R70.2 DO
  <Statement>;      (* is processed as long as the defined *)
                    (* external input has "1"signal *)
END;
```

*Wherever a statement is specified, it can be replaced by an entire statement block which must be bracketed with **BEGIN** and **END**;*

5.3.1.4 FOR TO loop

```
FOR <Var>:= <Value 1 > TO <Value 2> DO  
  <Statement>;
```

FOR TO loop

The **FOR TO** loop is a counting loop. The index of a variable is preset with an initial value. Taking this initial value, the statement is repeated until the index of a variable is **greater** than the specified final value. The index is scanned at the beginning of the loop. The index is **incremented** after statement processing before the scan.

The loop statement is repeated as long as the **index of the variable <> the final value**.

The following is also valid for the **FOR** statement: If several statements are to be executed at each loop run, then they must be bracketed with the word symbols **BEGIN** and **END**;

When the **index of the variable = the final value**, the counting loop statement is skipped and the following statement processed.

The **FOR TO** loop is processed in the program as follows:

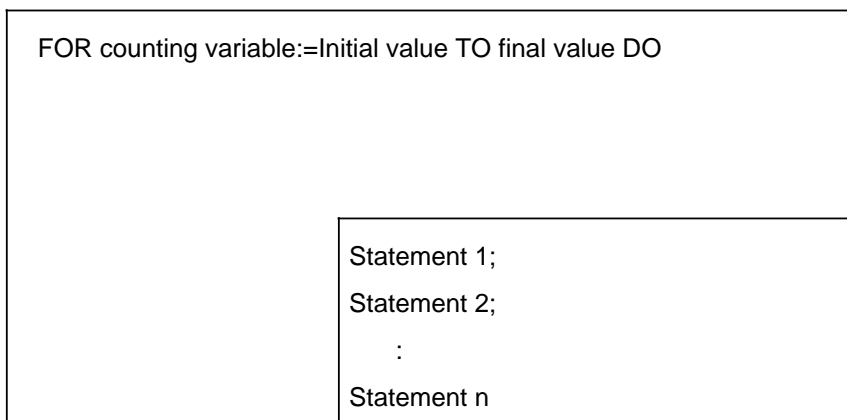
FOR count variable: =initial value **TO**, final value **DO**

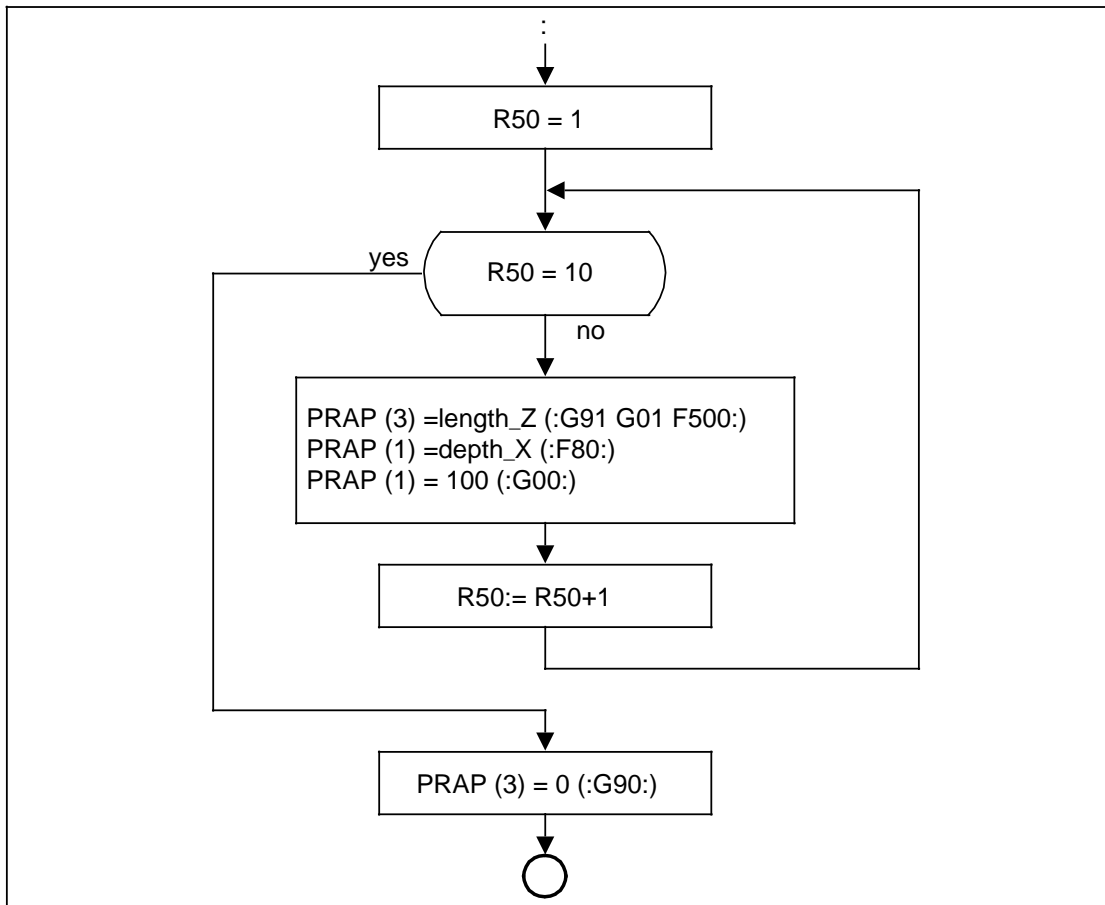
BEGIN

```
  statement 1;  
  statement 2;  
  :  
  statement n
```

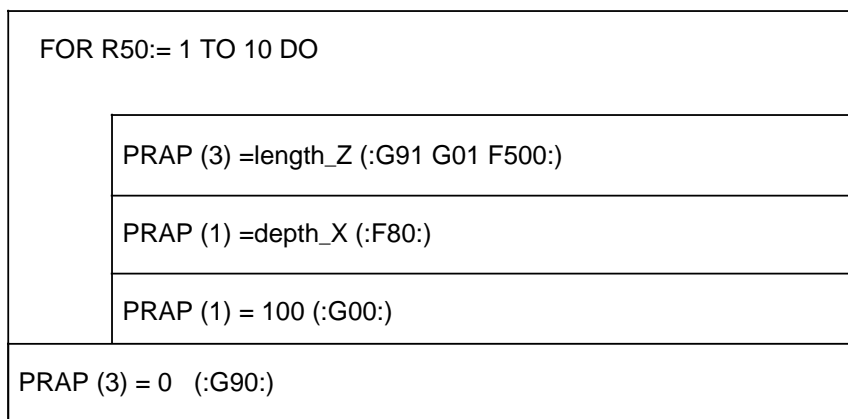
END;

Structogram of the FOR statement:



Example:• **flowchart**

Flowchart: FOR TO loop

• **structogram**

Structogram: FOR TO loop

- **CL 800 programming**

```
PROGRAM 502;  
:  
LOCAL REAL: R51:=length_Z, R52:=depth_X;  
LOCAL INTEGER: R50;  
:  
BEGIN  
:  
  FOR R50:= 1 TO 10 DO  
    BEGIN  
      PRAP (3) =length_Z; (:G91 G01 F500:); LF;  
      PRAP (1) =depth_X; (:F80:); LF;  
      PRAP (1) = 100; (:G00:); LF;  
    END;  
    PRAP (3) = 0; (:G90:); LF;  
  :  
END.
```

*Wherever a statement is specified, it can be replaced by an entire statement block which must be bracketed with **BEGIN** and **END**;*

5.3.1.5 FOR DOWNTO loop

<pre>FOR <Var>:= <Value 1> DOWNTO <Value 2> DO <Statement>;</pre>	FOR DOWNTO loop
---	------------------------

The **FOR DOWNTO** loop is a counting loop. The index of a variable is preset with an initial value. Taking this initial value, the statement is repeated until the index of a variable is **lower** than the specified final value. The index is scanned at the beginning of the loop. The index is **decremented** after each statement processing before the scan.

The loop statement is repeated as long as the **index of the variable <> the final value**.

As for the **FOR** statement, the same is also valid for the **FOR DOWNTO** statement:
If several statements are to be executed at each loop run, then they must be bracketed by the word symbols **BEGIN** and **END**;

When the **index of the variable = the final value**, the counting loop statement is skipped and the following statements processed.

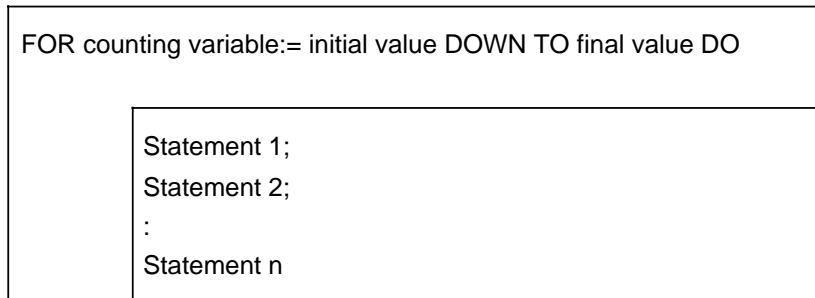
The **FOR DOWNTO** loop is processed as follows in the program:

```

FOR counting variable:=initial value DOWNTO final value DO
BEGIN
  Statement 1;
  Statement 2;
  :
  Statement n
END;

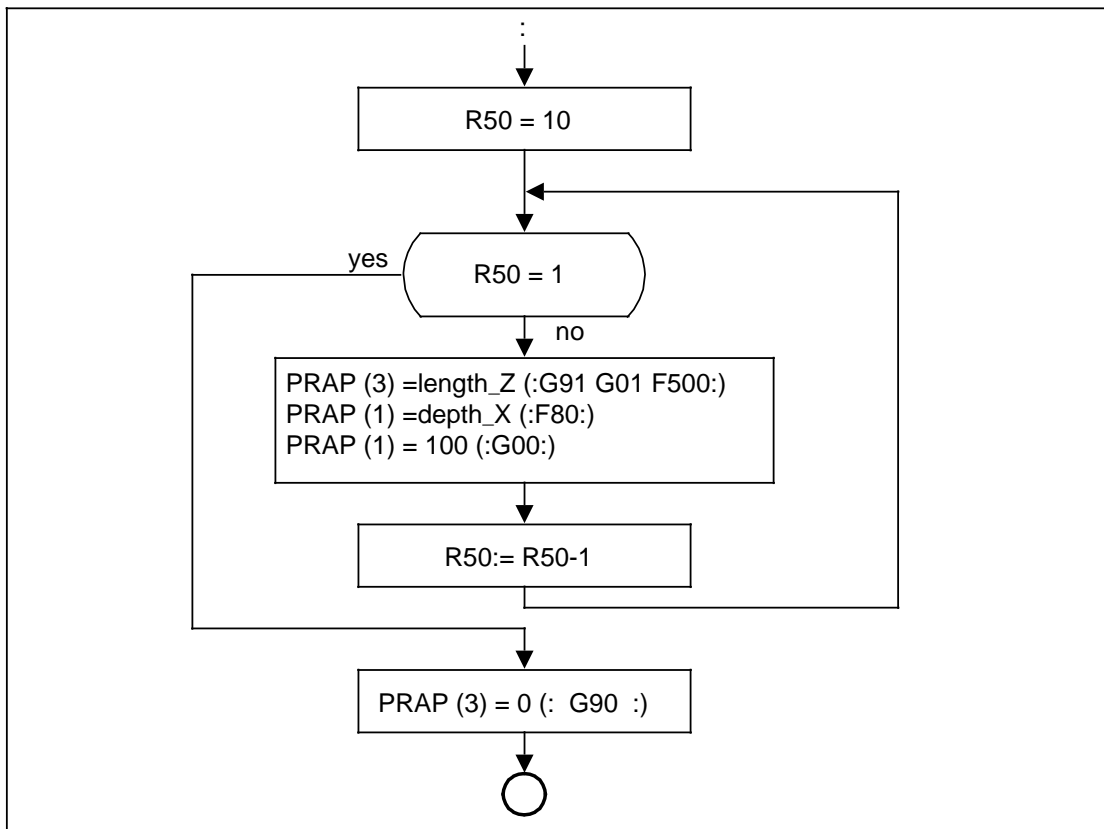
```

Structogram FOR DOWNTO statement



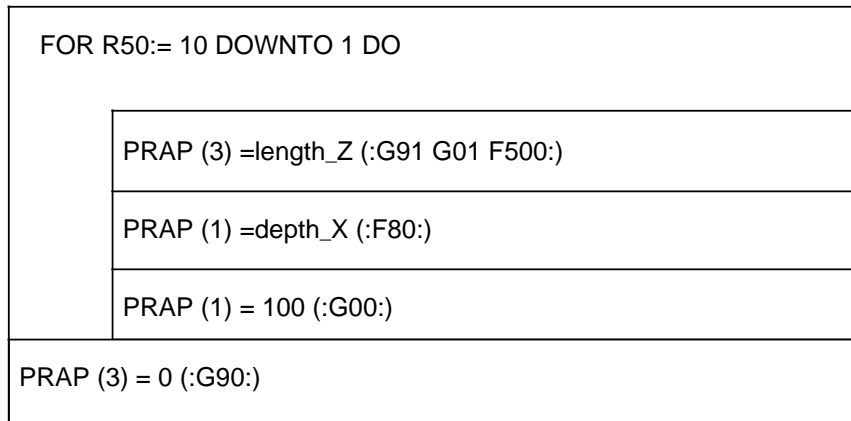
Example:

- flowchart



Flowchart: FOR DOWNTO loop

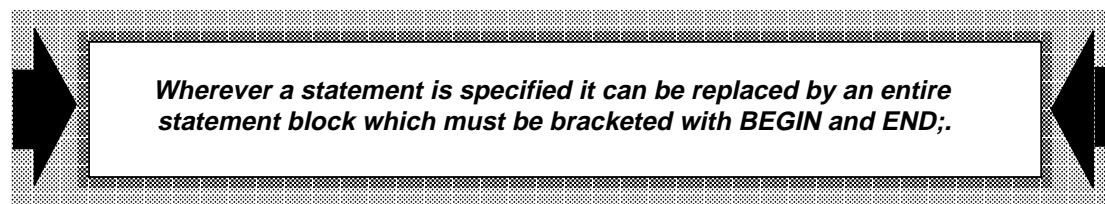
- **structogram**



Structogram: FOR DOWNTO loop

- **CL 800 programming**

```
PROGRAM 502;
:
LOCAL REAL: R51:=length_Z, R52:=depth_X;
LOCAL INTEGER: R50;
:
BEGIN
:
  FOR R50:= 10 DOWNTO 1 DO
  BEGIN
    PRAP (3) =length_Z; (:G91 G01 F500:); LF;
    PRAP (1) =depth_X; (:F80:); LF;
    PRAP (1) = 100; (:G00:); LF;
  END;
  PRAP (3) = 0; (:G90:); LF;
:
END.
```



5.3.2 Decision statements

The program run can be efficiently influenced with the indicated **REPEAT, WHILE, FOR** program control possibilities.

However, the problem often occurs that various statements are to be executed dependent on specific conditions. The fact that several alternatives can exist must be taken into account. These possibilities or combinations of these possibilities are taken into account using the decision statements **IF THEN ELSE** and **CASE**.

5.3.2.1 IF THEN ELSE branching

```

IF <Var> "Vop" <Value>                                IF THEN ELSE branching
  THEN <Statement 1>;
  [ELSE <Statement 2>;]
ENDIF;

("vop": >; >=; <; <=; =; <>;)

```

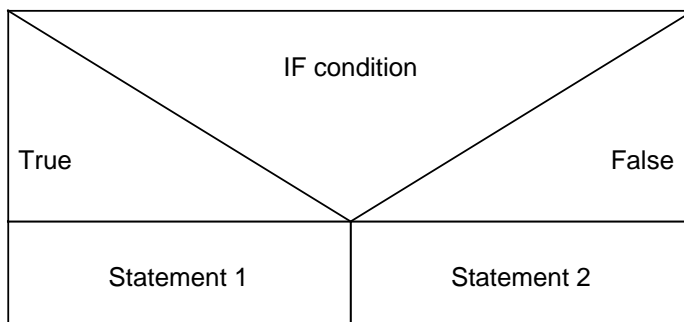
The **IF THEN ELSE** branching permits a choice to be made between the two alternatives **THEN** (if) and **ELSE** (otherwise).

The **IF THEN ELSE** statement is processed as follows in the program:
IF condition **THEN** statement 1 **ELSE** statement 2.

If the specified condition <Var> "Vop" <Value> is fulfilled, the instructions of the **THEN** branch will be executed, otherwise those of the **ELSE** branch. The **ELSE** branch can also be omitted. If the condition is not fulfilled, processing continues with the statement following **ENDIF**.

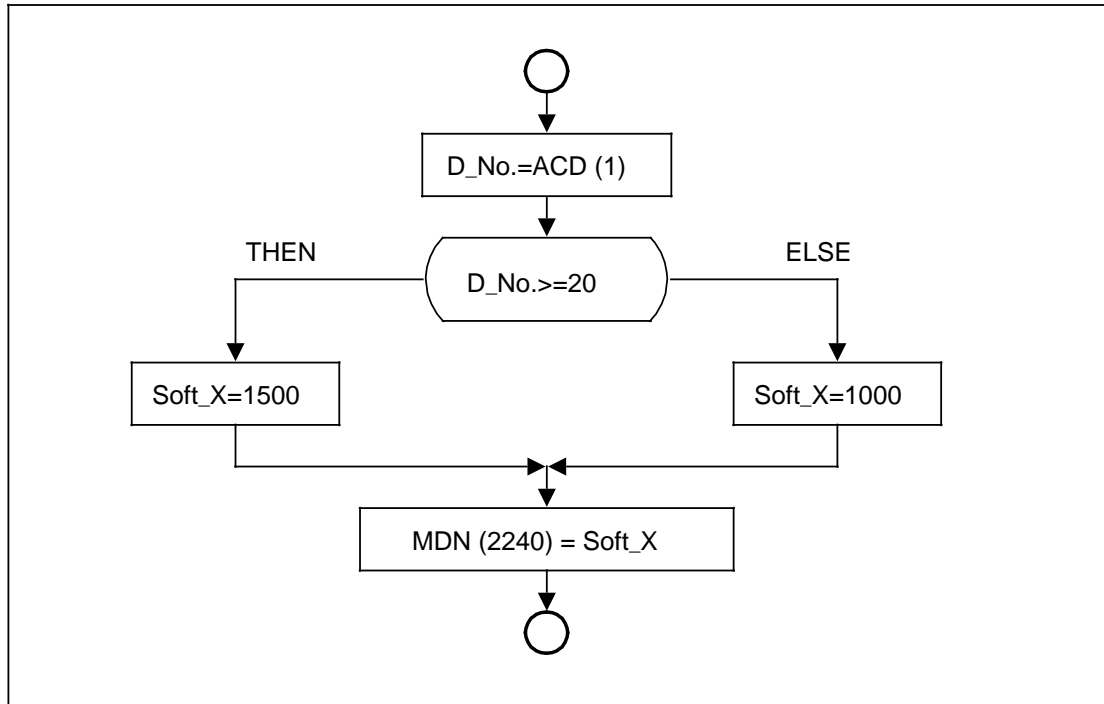
The term **IF THEN ELSE**; is a single statement and thus a ";" is not permitted in front of an **ELSE** as the semicolon is a separator for the statements.
Only one statement can be located after the word symbols **THEN** and **ELSE**. If several statements are to be checked using the condition, then they must be bracketed with **BEGIN** and **END** (as for **WHILE** and **FOR**).

Structogram of the IF statement



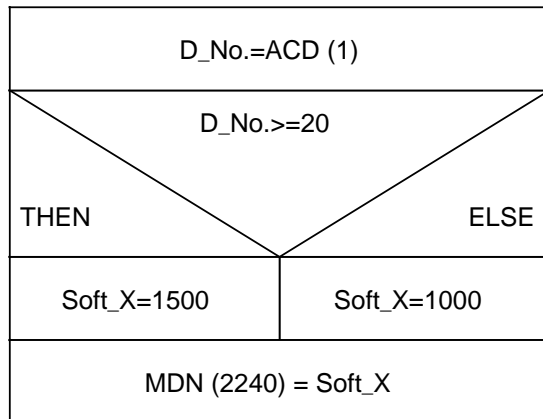
Example:

- **flowchart**



Flowchart: IF THEN ELSE branching

- **structogram**



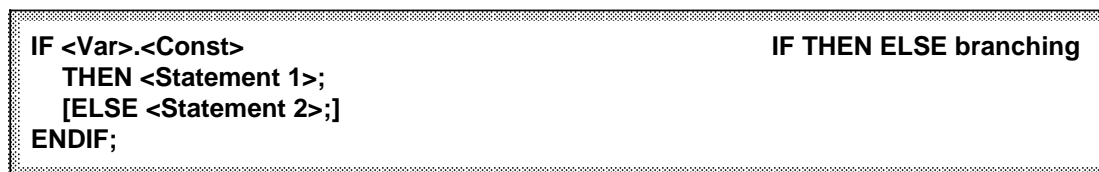
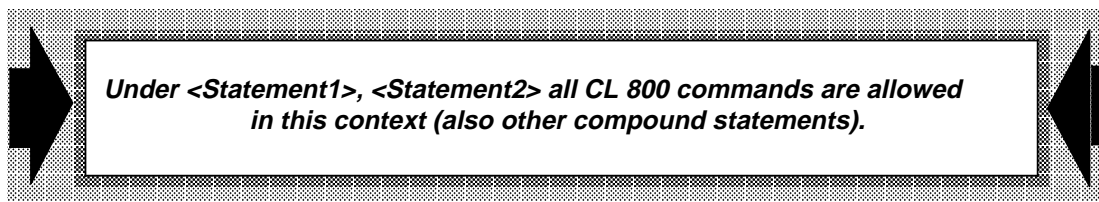
Structogram: IF THEN ELSE branching

- **CL 800 programming**

```

PROGRAM 505;
:
LOCAL REAL: R70:=D_No., R71:=Soft_X;
:
BEGIN
:
  D_No = ACD (1);
  (* read the current tool offset number*)
  IF D_No >= 20 THEN
    (* scan size of D number*)
    Soft_X = 1500;
  ELSE
    Soft_X = 1000;
    (* values for software limit switch*)
  ENDIF;
  MDN (2240) = Soft_X;
:
END.

```



IF THEN ELSE branching with which the program branches dependent on the status of a bit from a PATTERN variable. Bit numbers between 0 . . . 7 are specified as <Const>.

Example:

```

PROGRAM 60;
:
LOCAL PATTERN: R70;
:
R70=B00011111;
:
IF R70.3 THEN
  <Statement 1>; (* executed if bit 3 = 1 *)
ELSE
  <Statement 2>; (* executed if bit 3 is not 1 *)
ENDIF;
:

```

<pre>IF <Var> THEN <Statement 1>; [ELSE <Statement 2>;] ENDIF;</pre>	IF THEN ELSE branching
--	-------------------------------

IF THEN ELSE branching with which the program branches dependent on the status of a boolean variable. The boolean variable <Var> has the value 0 or 1.

Example:

```
PROGRAM 70;
:
LOCAL BOOLEAN: R70;
:
R70=1;
:
IF R70 THEN
  <Statement 1>; (* executed if R70=1 *)
ELSE
  <Statement 2>; (* executed if R70 is not 1 *)
ENDIF;
:
```

IF THEN ELSE branching with extended arithmetic
--

Extended arithmetic is permissible within the **IF THEN ELSE** branching. Extended arithmetic is understood to mean the execution of calculations with mathematical functions or logical operations with logical functions within the statement, in accordance with Section 5.3.7.

Program branching then takes place, depending on the result of the calculation or logic operation.

Example:

```
PROGRAM 1;
:
PAR REAL: R15, R25, R35;
PAR PATTERN: R10, R20, R30,
             R40:=OUTPUT;
PAR BOOLEAN: R21;
:
IF R25 + 5 > R15 THEN;
:
ENDIF;
:
IF R10.3 ANDB R21 THEN;
:
ENDIF;
:
```

```

:
IF OUTPUT.3 ANDB OUTPUT.4 THEN;
:
ENDIF;
:
IF SIN (R25) + 0.5 > 0 THEN;
:
ENDIF;
:
IF ANGLE (15,10+R25) > R35 THEN;
:
ENDIF;
:
IF MDN (2520) > 2000 THEN;
:
ENDIF;
:
IF NOTB R10 = R20 THEN;
:
ENDIF;
:
IF R25 > R15 ANDB R25 = R35 THEN;
:
ENDIF;
END.

```

5.3.2.2 IF INT THEN ELSE branching

<pre> IF INT <Value 1>.<Value 2> THEN <Statement 1>; [ELSE <Statement 2>;] ENDIF ; </pre>	IF INT THEN ELSE branching
---	-----------------------------------

IF INT THEN ELSE branching is dependent on the status of a defined external input.

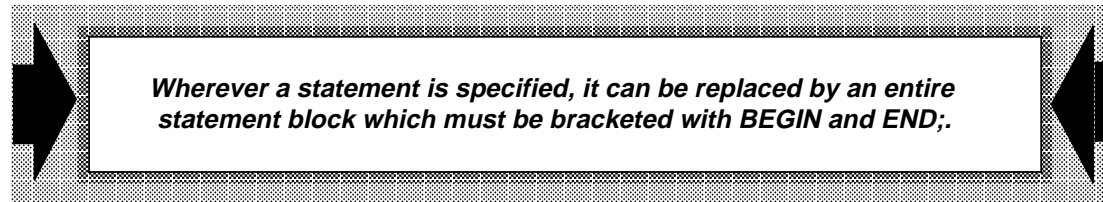
If the input, defined with the notation **<Value 1>** and **<Value 2>**, has a **"1" signal**, then the program is continued with the statement in the **THEN** branch, otherwise the statement of the **ELSE** branch is executed. The **ELSE** branch can be omitted. If the condition is not fulfilled, the statement which follows **ENDIF** is executed.

The scanning can also be realized using **IF INT NOT** for a **"0" signal**.

The byte address (1 or 2) of the external input can be defined with **<Value 1>** and the bit address (0 to 7) with **<Value 2>**.

Example:

```
IF INT 1.2 THEN
  <Statement 1>;      (* executed if the external input *)
                      (* has a "1" signal *)
ELSE
  <Statement 2>;      (* executed if the external input *)
                      (* has a "0" signal *)
ENDIF;
```



5.3.2.3 Case branching

```
CASE <Var>   =<Value 1> : <Statement 1>;
              :
              =<Value n> : <Statement n>;
[OTHERWISE : <Statement n+1>;]
ENDCASE;                                           CASE branching
```

The **CASE** branching can be used if a decision must be made between more than two solutions. Thus, the use of a multiple IF statement is superfluous.

With **CASE** branching, one of n+1 statements is executed, depending on the value of the variable <Var> .

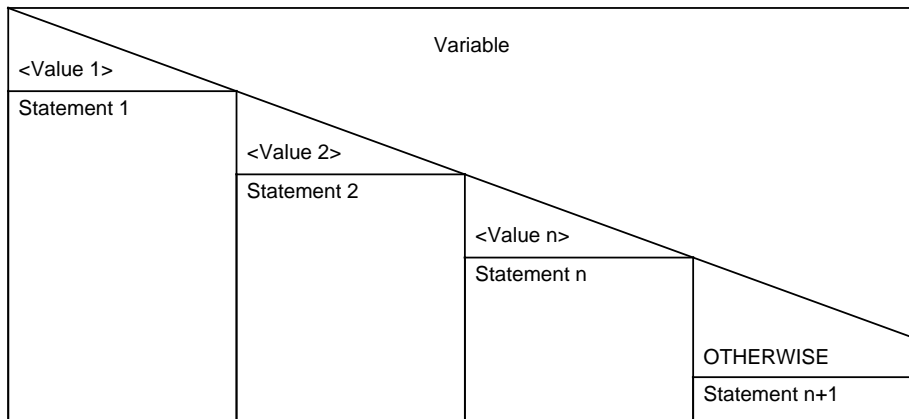
The **CASE** branching is processed in the program as follows:

```
CASE variable OF
  <Value 1> : BEGIN statement 1 END;
  <Value 2> : BEGIN statement 2 END;
  :
  :
  <Value n> : BEGIN statement n END;
  OTHERWISE: statement n+1;
ENDCASE
```

If several statements are to be executed per "CASE label" then these must be bracketed off with **BEGIN** and **END**.

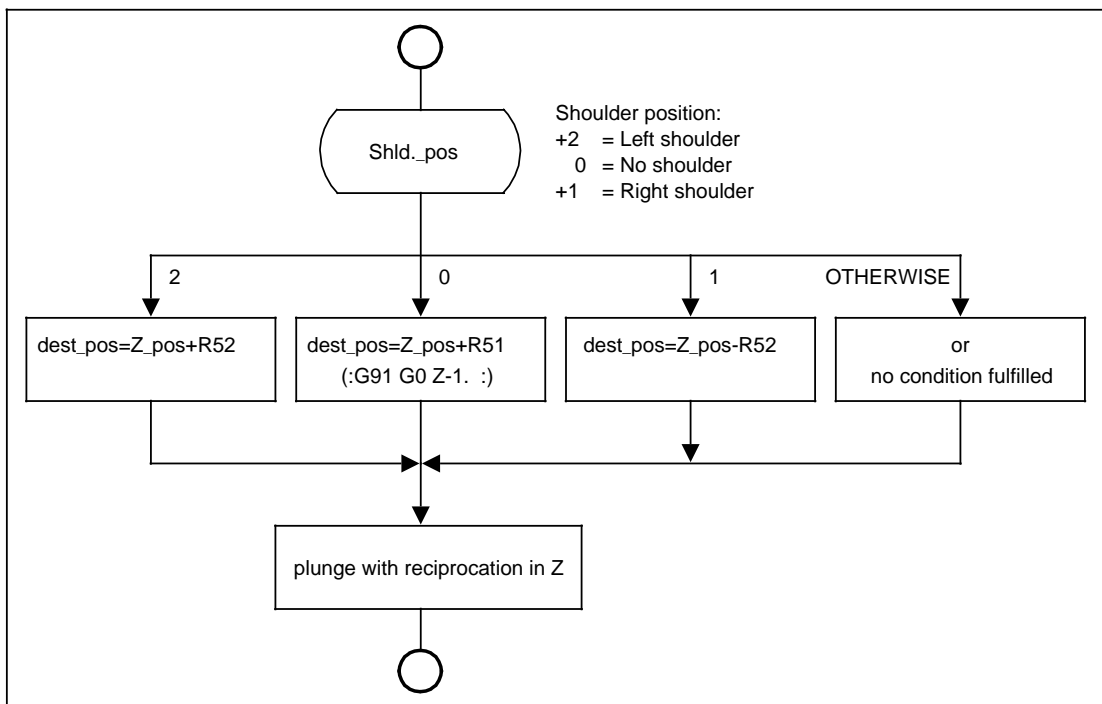
If an **OTHERWISE** statement is not programmed and none of the case statements are satisfied, the **CASE** branching is exited without being processed and the program is continued with the next statement.

Structogram of the CASE branching



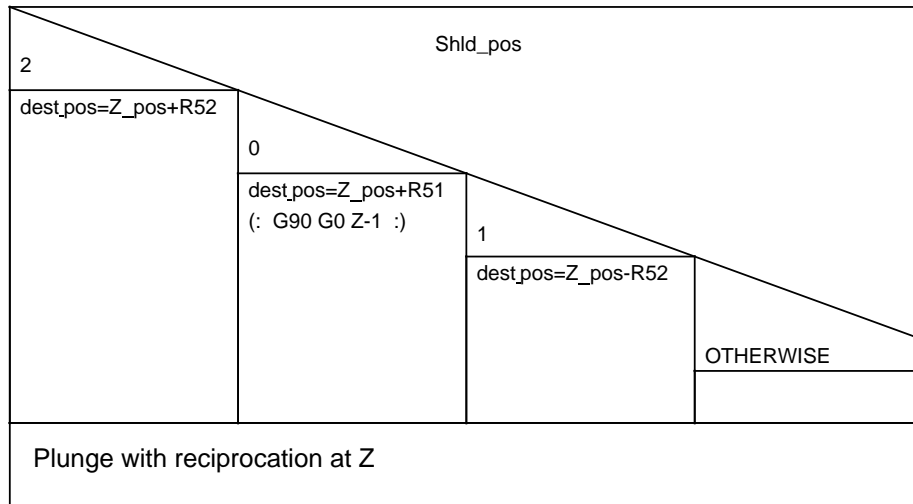
Example:

- flowchart



Flowchart: CASE branching

- **structogram**



Structogram: CASE branching

- **CL 800 programming**

```
PROGRAM 500 ;
CHANNEL NC 1;
```

```
PAR INTEGER: R0:=shld_pos;
LOCAL REAL: R50:=dest_pos, R53:=Z_pos;
```

```
BEGIN
```

```
      :
      CASE SHLD_pos = 2: dest_pos = Z_pos + 2;
                = 0: BEGIN
                    dest_pos = Z_pos + 1;
                    (:G91 G0 Z-1:);
                    END;
                = 1: dest_pos = Z_pos - 2;
```

```
    ENDCASE;
    BEGIN
```

```
      :
      (* plunge with reciprocation at Z *)
```

```
    END;
```

```
END.
```

5.3.3 Unconditional branching

5.3.3.1 Unconditional jump

GOTO <Label>;

Unconditional jump

A specific part of the program can be skipped with this command. Such a command is known as an unconditional jump. The jump destination is specified by a declared label.

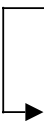
The result of the **GOTO** statement is that the statement marked with the declared label is the next one to be processed. The statement with the declared label must be located in the first program nesting level. It can be programmed before or after the **GOTO** statement in the program.

Example:

```

PROGRAM 2;
:
LABEL: Destination;
:
BEGIN
:
GOTO Destination;LF; (* jump to error indication *)
:
:
Dest: :(MSG:ERROR MESSAGE:);LF;
:
END.

```



The GOTO statement is essentially used for cases necessitating a program abort, e.g. as a result of an error.

5.3.4 General data transfer

5.3.4.1 Data transfer: R par./R par.

CLEAR (<Var>);

clear R parameter

The **CLEAR** statement is used to **clear a variable**. The specified R parameter **<Var>** is cleared and assigned zero with the **CLEAR** command.

Example:

```
PROGRAM 46;  
:  
LOCAL REAL: R60;  
BEGIN  
  R60 = 100;  
  :  
  CLEAR (R60);  
  (* R60 is assigned 0 *)  
  :  
END.
```

<Var> = <Value>;

value assignment

Statement for loading a value into a variable.
This command involves a simple **value assignment**.
The specified numerical quantity **<Value>** is assigned to the R parameter **<Var>**.

Example:

```
:  
R10 = 100;  
(* R10 has the contents 100 *)  
:
```

XCHG (<Var 1>,<Var 2>);

exchanging the variable contents

The **XCHG** statement allows the user to **exchange the variable contents** of **<Var 1>** and **<Var 2>**.

Example:

```
:  
R30 = 10;  
R20 = 30;  
:  
XCHG(R30,R20);  
:  
(* R30 has the contents 30 *)  
(* R20 has the contents 10 *)  
:
```

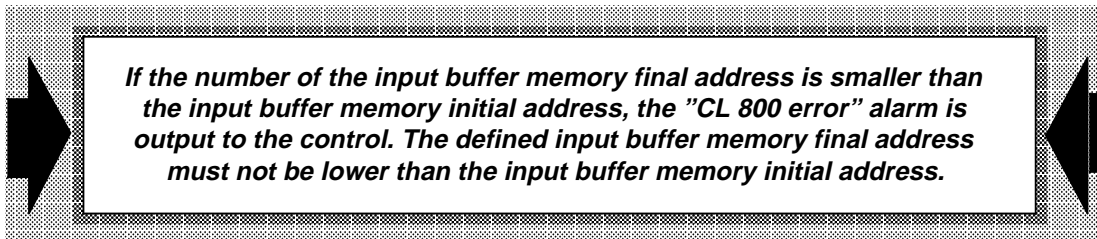
5.3.4.2 Data transfer: R par./input buffer memory for numerical variable

CLEAR MIB (<Value 3>,<Value 4>); **clear input buffer memory**

The **CLEAR MIB statement** is used to **clear input buffer memories**. The input buffer memory initial address is defined with **<Value 3>**, and the input buffer memory final address with **<Value 4>**.

Example:

CLEAR MIB (100,110); The input buffer memories from 100 to 110 are cleared.



<Var> = MIB (<Value 1>); **read input buffer memory**

The R parameter **<Var>** is loaded with the contents of the input buffer memory location **<Value 1>**.

Example:

R50 = MIB (101); The contents of the input buffer memory location 101 are read into R50.

MIB (<Value 1>) = <Value>; **write into input buffer memory**

The input buffer memory location **<Value 1>** is loaded with the numerical quantity **<Value>**.

Example:

MIB (102) = 5; 5 is written into the input buffer memory location 102.

5.3.5 Data transfer: System memory into the R parameter

5.3.5.1 Transfer machine data into the R parameter

<Var> = MDN (<Value 1>);

NC machine data

With the command read **NC machine data**, the contents of the machine data with the number **<Value 1>** is transferred into the parameter defined with **<Var>**.

Example:

R90 = MDN (2241); The value of the 1st software limit switch in the positive direction for the 2nd axis is located in parameter R90.

<Var> = MDNBY (<Value 1>);

NC machine data byte

An **NC machine data byte** is transferred into the parameter defined with **<Var>**. The byte address is specified by **<Value 1>**.

Example:

R91 = MDNBY (5001); The address of the angle which is required in the control is located in parameter R91 after the command has been executed.
R91 = 11 corresponds to address A.

<Var> = MDNBI (<Value 1>,<Value 2>);

NC machine data bit

An individual bit from the machine data area of the NC can be read into the parameter defined with **<Var>**. The byte address is specified with **<Value 1>** and the bit number with **<Value 2>**.

Example:

R92 = MDNBI (5002,4); Value 1 is located in parameter R92 if inch input is selected as basic setting in the NC system.
The relevant information "bit set" or "bit not set" is made available as parameter 1 or 0.

<Var> = MDZ	(<Value 1>,<Value 2>);	cycles machine data
<Var> = MDZBY	(<Value 1>,<Value 2>);	cycles machine data byte
<Var> = MDZBI	(<Value 1>,<Value 2>,<Value 3>);	cycles machine data bit

The **machine data** for **cycles** can be read, as for the machine data from the NC, as value, byte, or bit, in the parameter defined with **<Var>**. The channel number is selected using **<Value1>**. Its own channel is read when 0 is input for channel-orientated data. The channel No. is always specified with 0 for central data. The word- or byte address is specified with **<Value 2>** and the bit address with **<Value 3>**.

Example:

R50 = MDZ (0,400); The value of the channel-orientated machine data for cycles in its own channel is read into parameter R50.
R51 = MDZBY (1,801); The value of the machine data byte is read into parameter R51.
R52 = MDZBI (1,802,1); The value of the machine data bit in the 1st channel is read into parameter R52.

When system memories from other channels are read, which are not located on the same NC CPU, then the "CL 800 error" alarm signal is output to the control. Only those channels can be read, which are processed from the same NC CPU as the selected channel.

<Var> = MDP	(<Value 1>);	PLC machine data
<Var> = MDPBY	(<Value 1>);	PLC machine data byte
<Var> = MDPBI	(<Value 1>,<Value 2>);	PLC machine data bit

The **machine data** from the **PLC** can, as for the machine data, be read into the parameter defined with **<Var>** from the NC as value, byte, or bit .

Example:

R93 = MDP (20); The number of the 1st static M function which is used in the SINUMERIK system 810 is located in parameter R93 (e.g. 40).

R94 = MDPBY (3000); byte 3000

00010010

The 1st byte of the PLC user data bit is transferred into R94 (R94 has the contents 10010).

R95 = MDPBI (3001,2); byte 3001

00010110

The status of the 2nd bit from the PLC user byte 3001 is read into R95 (R95 has the contents 1).

<Var> = MDD	(<Value 1>);	drive machine data
<Var> = MDDBY	(<Value 1>,<Value 2>);	drive machine data byte
<Var> = MDDBI	(<Value 1>,<Value 2>,<Value 3>);	drive machine data bit

The **machine data** for the **drives** can, as for the machine data, be read into the parameter defined with **<Var>** from the NC as value, byte, or bit. The word address is selected by **<Value 1>**. **<Value 2>** is used to define as to whether the low byte (0) or high byte (1) is to be read. The bit address is specified by **<Value 3>**.

Example:

- R60 = MDD (2521); The value of the machine data for the 2nd axis is read into R60.
- R61 = MDDBY (2441,0); The low byte of the machine data for the 2nd axis is read into R61.
- R62 = MDDBI (2442,0,2); The status of the 2nd bit in the low byte of the machine data for the 3rd axis is read into R62.

5.3.5.2 Transfer setting data into the R parameter

<Var> = SEN	(<Value>);	NC setting data
<Var> = SENBY	(<Value>);	NC setting data byte
<Var> = SENBI	(<Value 1>,<Value 2>);	NC setting data bit

The **setting data** of the **NC** can, as for the machine data, be read into the parameter defined with **<Var>** as value, byte, or bit.

Example:

- R80 = SEN (3041); The maximum working field limiting of the 2nd axis is read into parameter R80.
- R81 = SENBY (5013); The transmission format for the 1st V24 interface for output is read into R81, e.g. R81=11000111 if 9600 baud is to be output.
- R82 = SENBI (5001,0); R82=1 if the display of the workpiece-related actual value system is active.

<Var> = SEZ	(<Value 1>,<Value 2>);	cycles setting data
<Var> = SEZBY	(<Value 1>,<Value 2>);	cycles setting data byte
<Var> = SEZBI	(<Value 1>,<Value 2>,<Value 3>);	cycles setting data bit

The **setting data** for **cycles** can, as for the machine data, be read into the parameter defined with **<Var>** as value, byte, or bit.

Example:

R50 = SEZ (1,400); The contents of the setting data for cycles in the 1st channel is read into parameter R50.

R51 = SEZBY (0,802); The contents of the setting data bytes for cycles in its own channel are read into parameter R51.

R52 = SEZBI (0,802,2); The status of the 2nd bit of the cycles setting data byte in its own channel is read into R52.

5.3.5.3 Transfer tool offsets into the R parameter

The individual tool offset areas are structured as follows:

TO area	Offset No.	T- No. P0	Type P1	Geometry			Wear		P7	P8	P15
				P2	P3	P4	P5	P6				
1	D1											
1	D2	30	3	(X) 33.5	(Z) 67	(R) 2.5	(X) 0.5	(Z) 40				Example
1	⋮											
1	Dn											
2	D1											
2	D2		3	36.4	67.94	1.5	0.22	0.01				Example
2	⋮											
2	Dn											

The individual offsets can be transferred from the table into a parameter **<Var>** with the TOS command.

<Var> = TOS (<Value 1>, <Value 2>, <Value 3>);

tool offsets

The required TO area is defined with **<Value 1>**. For SINUMERIK System 810/ 820, a 0 is always specified here, as only one tool offset area is available. For SINUMERIK System 850/880, the channel, assigned above the machine data is read when 0 is input into the TO area.

The required D No. is selected by **<Value 2>**.

The required P No. is specified by **<Value 3>**.

Example:

R80 = TOS (1,2,3);

The length of the tool in the 2nd axis from the tool offset memory D2 from TO area 1 is read into parameter R80.

The offset memories P2 and P5 are read as radius or diameter value, dependent on the NC machine data bit.

5.3.5.4 Transfer zero offsets into the R parameter

<Var> = ZOA (<Value 1>, <Value 2>, <Value 3>);

settable zero offset

The individual values of the **settable zero offsets** (G54 to G57) can be read into the parameter defined with **<Var>**.

The required zero offset is selected using **<Value 1>** (G54 = 1 to G57 = 4), and the required axis using **<Value 2>**.

<Value 3> is used to determine from which area a value is read (coarse value=0 and fine value= 1).

Example:

R81 = ZOA (1, 2, 0);

The coarse value of the 1st settable zero offset (G54) of the 2nd axis is read into parameter R81.

<Var> = ZOPR (<Value 1>, <Value 2>); **programmable zero offset**

The individual axis values of the **programmed zero offsets** can be read into the parameter defined with **<Var>**.

The required zero offset (G58=1 and G59=2) can be selected with **<Value 1>**, and the required axis with **<Value 2>**.

Example:

R80 = ZOPR (1,1); The 1st programmable zero offset of the 1st axis is read into parameter R80.

<Var> = ZOE (<Value 2>); **external zero offset**

External, zero offsets of the axes, selected through the PLC, can be read into the parameter defined with **<Var>**. The axis is specified with **<Value 2>**.

Example:

R60 = ZOE (2); The value of the external zero offset in the 2nd axis is read into R60.

The zero offsets are read as either radius or diameter value, dependent on the NC machine data bits.

<Var> = ZOD (<Value 2>); **DRF offset**

The **handwheel offset (DRF offset)** of each axis can be read into the parameter defined with **<Var>**. The axis is specified by **<Value 2>**.

Example:

R61 = ZOD (2); The handwheel offset of the 2nd axis is read into R61.

<Var> = ZOPS (< Value 2>);

PRESET offset

The offset (**PRESET offset**) of each axis, caused by an actual value setting, can be read into the parameter defined with **<Var>**. The axis is specified with **<Value 2>**.

Example:

R65 = ZOPS (1); The PRESET offset of the 1st axis is read into R65.

<Var> = ZOS (<Value 2>);

total of all offsets

The **sum of all current offsets** in each axis can be read into the parameter defined with **<Var>**. The axis is specified with **<Value 2>**.

The total of all offsets includes:

- the selected settable zero offset (G54 to G57)
- the programmable additive zero offset (G58 and G59)
- the external zero offset from the PLC
- the selected tool offset.

DRF- and PRESET offsets are not taken into account

Example:

R80 = ZOS (2); The total of all offsets of the 2nd axis are read into R80.

The DRF- or PRESET offset is read as radius or diameter value dependent on the NC machine data bit. The total of all offsets is always read as radius.

<Var> = ZOADW (<Value 1>,<Value 2>,<Value 3>);

**settable
coordinate rotation**

The angle of rotation of the **settable coordinate rotation** can be read into the parameter defined with the **<Var>**. The channel No. can be selected by **<Value 1>**. When 0 is input, the angle of rotation in its own channel is read.

The required zero offset (G54=1 to G57=4) is defined with **<Value 2>**, and the angle No. with **<Value 3>**. At the present time, **<Value 3>** is defined with 1.

Example:

R50 = ZOADW (0,1,1); The angle of rotation of the settable coordinate rotation in its own channel for the 1st zero offset is read into R50.

<Var> = ZOPRDW (<Value 1>,<Value 2>,<Value 3>);

**programmable
coordinate rotation**

The angle of rotation of the **programmable coordinate rotation** can be read into the parameter defined with **<Var>**. The channel No. can be selected with **<Value 1>**. The angle of rotation in its own channel is read when 0 is input.

The required zero offset (G58=1 and G59=2), is defined with **<Value 2>**, and the angle No. with **<Value 3>**. At the present time, **<Value 3>** is defined with 1.

Example:

R70 = ZOPRDW (1,2,1); The angle of rotation of the programmable coordinate rotation in the 1st channel for the 2nd zero offset is read into R70.

When reading from system memory locations from other channels which are not located on the same NC CPU, a "CL 800 error" alarm signal is output to the control. Only those channels can be read, which are processed by the same NC CPU as the selected channel.

5.3.5.5 Read programmed setpoints into the R parameter

<Var> = PRSS (<Value 1>, <Value 3>);

programmable spindle speed

The programmed spindle speed from a channel can be read into the parameter defined with **<Var>**.

The channel No. is specified with **<Value 1>**. If the programmed channel No. **<Value 1>** is 0, the channel No. of the current channel is used as channel No.

The spindle No. is specified with **<Value 3>**. If the programmed spindle No. **<Value 3>** is 0, the number of the leading spindle in the programmed channel **<Value 1>** is used as spindle No..

Example:

R55 = PRSS (1,0); The programmed spindle speed for the master spindle in channel 1 is read into R55.

<Var> = PCDA (<Value 1>,<Value 2>,<Value 3>);	programmed control words for digital axis drives
--	---

The status of a bit in the **programmed control word** for **digital axis drives** is read into the parameter defined with **<Var>**. The axis No. is specified by **<Value 1>**. The byte address (0 or 1) is defined in **<Value 2>**. The bit address is specified by **<Value 3>**.

Example:

R50 = PCDA (1,1,6); The status of the 6th bit in byte 1 for the 1st axis is read into R50.

<Var> = PCDS (<Value 1>,<Value 2>,<Value 3>);	programmed control words for digital spindle drives
--	--

The status of a bit in the **programmed control word** for **digital spindle drives** is read into the parameter defined with **<Var>**. The spindle No. is defined with **<Value 1>**. The byte address (0 to 5) is defined in **<Value 2>**. The bit address (0 to 7) is specified with **<Value 3>**.

Example:

R51 = PCDS (1,0,2); The status of the 2nd bit in byte 0 for the 1st spindle is read into R51.

5.3.5.6 Read actual values into the R parameter

<Var> = ACPW (<Value 2>);	workpiece-related axis position
--	--

The **workpiece-related actual Value** of each axis can be read into the parameter defined with **<Var>**. It is the dimension between the workpiece zero W and tool starting position P (refer to Fig., Page 4-51). The axis is defined by **<Value 2>**.

Example:

R80 = ACPW (2); The workpiece-related actual value of the 2nd axis is read into R80.

<Var> = ACPM (<Value 2>);

**programmed control word
for digital axis drives**

The machine-related actual value for each axis can be read into the parameter defined with <Var>. It is the dimension between the machine zero M and the slide reference point F (refer to Fig., page 4-51) The axis is specified with <Value>.

For rotary axes the result is stored from R parameter <Var> onwards (Rn) in a total of two R parameters dependent on an NC machine data bit.

The following parameters are loaded:

Rn = position within a revolution
Rn+1 = number of revolutions

Example:

R80 = ACPM (1); The machine-related actual value of the 1st axis is read.

If the axis position is read in its own channel, a "STOP DEC" (Section 5.3.8.2) should be programmed in the previous block. The axis position is read as radius or diameter value dependent on the NC machine data bits.

<Var> = ACP (<Value 2>);

current axis position

The current actual value of each axis can be read in the parameter defined with <Var>. It is the machine related axis position with calculated following error. The axis is specified with <Value 2>.

For rotary axes, the result is stored from R parameter <Var> onwards (Rn) in a total of two R parameters dependent on an NC machine data bit.

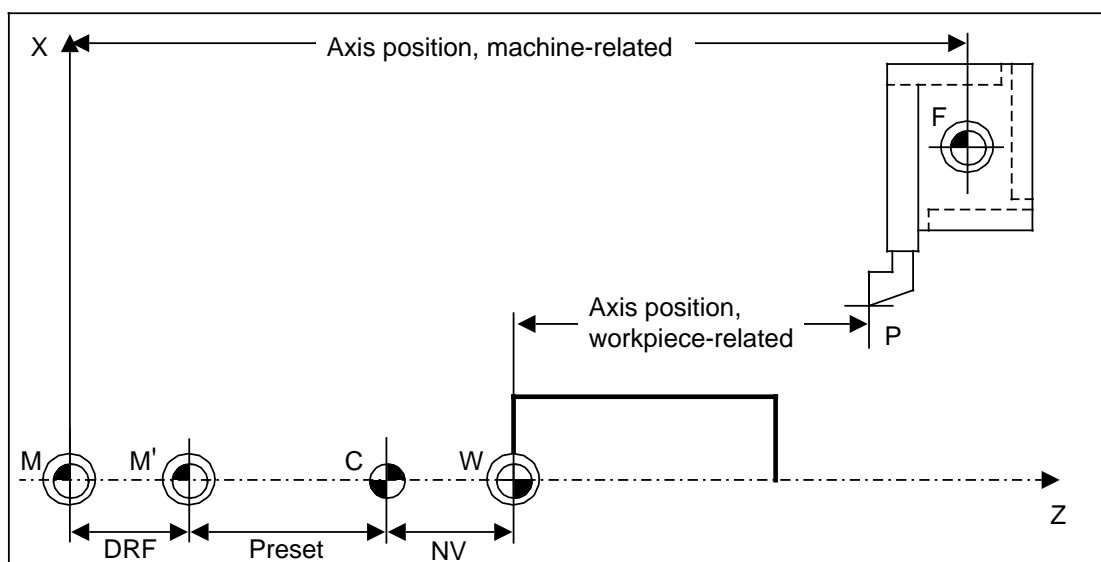
The following parameters are loaded:

Rn = position within a revolution
Rn+1 = number of revolutions

Example:

R80 = ACP (2); The current actual value of the 2nd axis is read.

If the axis position is read in its own channel, a "STOP DEC" (Section 5.3.8.2) should be programmed in the previous block. The axis position is read as radius or diameter value dependent on the NC machine data bits.



C	Control zero
F	Tool reference point or slide reference point
M, M'	Machine zero
P	Tool starting position
W	Workpiece zero
NV	Total of all zero offsets

<Var> = ACSP (<Value 2>);

actual spindle position

The **actual position of each spindle** can be read into the parameter defined with **<Var>**. The spindle is defined with **<Value 2>**. If 0 is specified, the position of the leading spindle is read from its own channel.

Example:

R80 = ACSP (0); The position of the master spindle in its own channel is read.

<Var> = ACSS (<Value 2>);

actual spindle speed

The **actual speed of each spindle** can be read into the parameter defined with **<Var>**. The spindle is defined with **<Value 2>**. If 0 is specified, the speed of the leading spindle is read from its own channel.

Example:

R81 = ACSS (1); The actual speed of the 1st master is read into R81.

If the spindle position or spindle speed is read in its own channel, a "STOP DEC" (Section 5.3.8.2) must be programmed in the previous block.

<Var> = ACAS (<Value 1>);

**axis number of the actual
planes/master spindle number**

With this command, the **axis numbers of the actual plane** and the **spindle number** can be read into the R parameter. The data is deposited for a total of five R parameters starting from R parameter **<Var>** (Rn).

The following R parameters are loaded:

Rn = Number of the horizontal axis
 Rn+1 = Number of the vertical axis
 Rn+2 = Number of the axis vertical to the plane
 Rn+3 = Number of the axis which acts in length 2 (tool type 30)
 Rn+4 = Number of the master spindle

The channel No. is defined by **<Value 1>**. If 0 is specified, data is read from its own channel.

Cycles can be programmed to be generally valid using this command.

Example:

R50 = ACAS (1); The data of the actual plane and spindle No. is read and deposited starting from R50.

When reading from system memories from other channels, which are not located on the same NC CPU, the "CL 800 error" is output to the control. Channels can only be read, which are processed by the same NC CPU as the selected channel. In order to be able to read from another channel, NC start must also be given in this channel. A "STOP DEC" (Section 5.3.8.2) must be programmed in front of the command ACAS, when its own channel is read.

If a negative length compensation is selected via G16, the minus sign= 128 is added to the axis number.

Example:

(: G16 X Y Z- :);
 R50 = ACAS (0); (* R50=1, R51=2, R52=131, R53=0, R54=1 *)

<Var> = ACD (<Value 1>);

current D-No.

The current **tool offset No.** from each channel can be read into the parameter defined with **<Var>**.

The channel No. is defined by **<Value 1>**. If 0 is specified, the own-channel tool offset No. is read. A 0 is always specified with the System 810/820, because the tool offset No. is only available for the first channel.

Example 1:

Program in Channel 1, the current tool offset No. is to be read from the programs own channel.

R81 = ACD (0); The current tool offset No. is read into R81.

Example 2:

Program in channel 2, the current tool offset No. is to be read from channel 3.

R81 = ACD (3); For System 850, the current tool offset No. in channel 3 is read into R81.

When reading from system memories from other channels, which are not located on the same NC CPU, the "CL 800 error" alarm signal is output to the control. Only those channels can be read which are processed by the same NC CPU as the selected channel. If the current D-No. is read in its own channel, then a "STOP DEC" (Section 5.3.8.2) should be programmed in the previous block.

<Var> = ACG (<Value 1>,<Value 3>);

G-function

The **G-function** of the parts program block presently being processed is read from the working memory into the parameter defined with **<Var>**. The channel No. is specified by **<Value 1>**. If 0 is specified then the own-channel is read. The G-group is defined by **<Value 3>**. A table with the G-group classification is located in the appendix.

Example:

R50 = ACG (0,0); The current G-function of the first G-group is read into parameter R50 in its own channel.

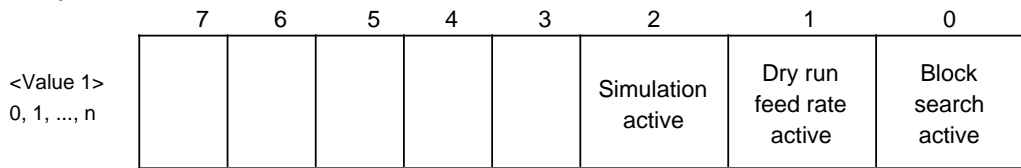
When reading from system memories from other channels, which are not located on the same NC CPU, the "CL 800 error" alarm signal is output to the control. Only those channels can be read which are processed by the same NC CPU as the selected channel. In order to be able to read values from other channels, NC start must also be given in these channels. If its own channel is to be read, a "STOP DEC" (Section 5.3.8.2) must be programmed in front of the command ACG.

5.3.5.7 Read program data into the R parameter

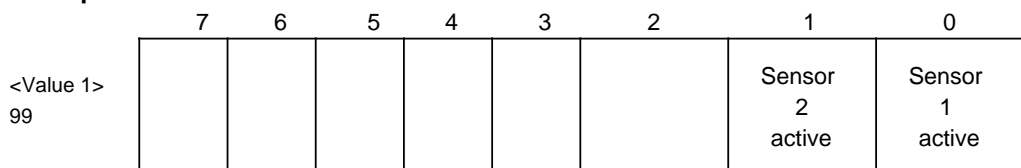
<Var> = SOB (<Value 1>,<Value 3>); special bits

The following **special bits** can be read into the parameter defined with **<Var>** with this command.

Channel-dependent bits:



Channel-independent bits:



The channel No. is defined by **<Value 1>**. When 0 is specified, the bit is read from its own channel. When reading channel-independent bits, **<Value 1>** is defined with 99. The bit No. is defined using **<Value 3>**.

Example 1:

Program in channel 2, the "dry run feed active" bit from the program's own channel is to be scanned.

R81 = SOB (0,1); The status of the special bit for dry run feed rate is read into R81.

Example 2:

Program in channel 2, the "dry run feed active" bit from channel 3 is to be scanned.

R81 = SOB (3,1); The status of the "dry run feed active" special bit in channel 3 is read into R81.

When reading from system memories from other channels, which are not located on the same NC CPU, the "CL 800 error" alarm signal is output to the control. Only those channels can be read, which are processed by the same NC CPU as the selected channel.

<Var> = PPCH ;	current channel number for the program
-----------------------------	---

The **channel number** of the NC channel in which the program is executed is read into the parameter defined with **<Var>**.

Example:

R51 = PPCH; The current channel number for the program is read into parameter R51.

5.3.5.8 Read PLC signal bits into the R parameter

<Var> = PLCI	(<Value 1>,<Value 2>,<Value 3>);	PLC input bit
<Var> = PLCQ	(<Value 1>,<Value 2>,<Value 3>);	PLC output bit
<Var> = PLCF	(<Value 1>,<Value 2>,<Value 3>);	PLC flag bit

The status of an **input-, output- or flag bit** in the PLC is read into the parameter defined with **<Var>**. The PLC No. is defined with **<Value 1>**, the byte address with **<Value 2>** and the bit address with **<Value 3>**.

Example:

R50 = PLCI (1,2,0); The status of the defined PLC input bit is read into R50.

<Var>=PLCW (<Value 1>,<Value 2>,<Value 3>,<Value 4>);	PLC data word bit
--	--------------------------

The status of a **PLC data word bit** is read into the parameter defined with **<Var>**. The PLC No. is defined by **<Value 1>**, the number of the DB or DX by **<Value 2>**, the data word number by **<Value 3>** and the bit address by **<Value 4>**.

Example:

R51 = PLCI (1,2,4,2); The status of the defined PLC data word bit is read into R51.

5.3.5.9 Read PLC signal bytes into the R parameter

<Var> = PLCIB	(<Value 1>,<Value 2>);	PLC input byte
<Var> = PLCQB	(<Value 1>,<Value 2>);	PLC output byte
<Var> = PLCPB	(<Value 1>,<Value 2>);	PLC peripheral byte
<Var> = PLCFB	(<Value 1>,<Value 2>);	PLC flag byte

The status of an **input-, output-, peripheral- or flag byte** in the PLC is read into the parameter defined with <Var>. The PLC No. is defined by <Value 1> and the byte address by <Value 2>.

Example:

R52 = PLCIB (1,1); The status of the defined PLC input byte is read into R52.

If the command, read PLC peripheral byte (PLC PB) addresses a byte which is not available in the PLC, the PLC goes into the STOP condition! A "cold restart" is then necessary.

<Var> = PLCDBL (<Value 1>,<Value 2>,<Value 3>);	PLC data word, left
<Var> = PLCDBR (<Value 1>,<Value 2>,<Value 3>);	PLC data word, right

The status of a **PLC data word, left or right**, is read into the parameter defined with <Var>. The PLC number is defined by <Value 1>, the number of the DB or DX by <Value 2> and the data word number by <Value 3>.

Example:

R53 = PLCDBL (1,2,4); The status of the defined PLC data word, left, is read into R53.

5.3.5.10 Read PLC signal words into the R parameter

<Var> = PLCIW	(<Value 1>,<Value 2>,<Value 3>);	PLC input word
<Var> = PLCQW	(<Value 1>,<Value 2>,<Value 3>);	PLC output word
<Var> = PLCPW	(<Value 1>,<Value 2>,<Value 3>);	PLC peripheral word
<Var> = PLCFW	(<Value 1>,<Value 2>,<Value 3>);	PLC flag word

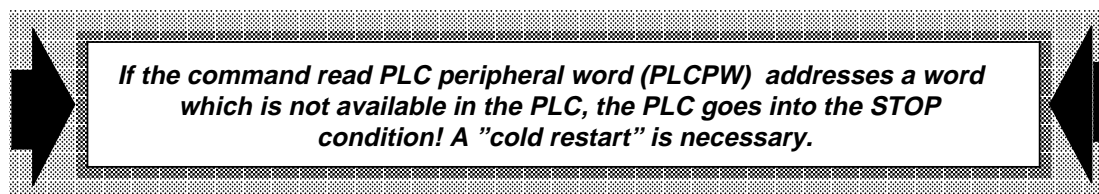
The status of an **input-, output-, peripheral or flag word** in the PLC is read into the parameter defined with <Var>. The PLC number is defined by <Value 1> and the word address by <Value 2>. The dimension identifier for <Value 3> specifies how the status of the PLC signal word is to be read.

Definition of the dimension identifier <Value 3>:

<Value 3>	Fixed point	<Value 3>	BCD
0	Value without decimal point	100	Value without decimal point
1	Value with decimal point	101	Value with decimal point
2	1 digit after the decimal point	102	1 digit after the decimal point
3	2 digits after the decimal point	103	2 digits after the decimal point
4	3 digits after the decimal point	104	3 digits after the decimal point
5	4 digits after the decimal point	105	4 digits after the decimal point
6	5 digits after the decimal point	106	5 digits after the decimal point
7	6 digits after the decimal point	107	6 digits after the decimal point
8	7 digits after the decimal point	108	7 digits after the decimal point
9	8 digits after the decimal point	109	8 digits after the decimal point

Example:

R50 = PLCIW (1,3,100); The status of the defined PLC input word is read into R50 in BCD.



<Var> = PLCT (<Value 1>,<Value 2>);	PLC timer
-------------------------------------	-----------

The status of a **timer** in the PLC is read into a parameter defined with <Var>. It is read as a value in seconds with 2 digits after the decimal point. The PLC number is defined by <Value 1>, and the timer address by <Value 2>.

Example:

R80 = PLCT (1,2); The status of the defined PLC timer is read into R80.

<Var> = PLCC (<Value 1>,<Value 2>);

PLC counter

The status of a **counter** in the **PLC** is read into the parameter defined with **<Var>**. The PLC number is defined by **<Value 1>** and the counter address by **<Value 2>**.

Example:

R81 = PLCC (1,2); The status of the defined PLC counter is read into R81.

5.3.5.11 Read PLC signal data words into the R parameter

**<Var> = PLCDF (<Value 1>,<Value 2>,<Value 3>,
<Value 4>,<Value 5>);**

**PLC data word,
fixed point**

The **fixed point value** of a **data word** or **double word** (serial or parallel) in the PLC is read into the parameter defined with **<Var>**. The PLC number is specified by **<Value 1>**, the number of DB or DX by **<Value 2>**, the data word number by **<Value 3>**, the number of data words by **<Value 4>**. The dimension identifier **<Value 5>** specifies how the fixed point value is to be read..

Definition of the dimension identifier <Value 5>:

<Value 5>	Fixed point: Data word or double word, serial	<Value 5>	Fixed point: Double word, parallel
0	Value without decimal point	10	Value without decimal point
1	Value with decimal point	11	Value with decimal point
2	1 digit after the decimal point	12	1 digit after the decimal point
3	2 digits after the decimal point	13	2 digits after the decimal point
4	3 digits after the decimal point	14	3 digits after the decimal point
5	4 digits after the decimal point	15	4 digits after the decimal point
6	5 digits after the decimal point	16	5 digits after the decimal point
7	6 digits after the decimal point	17	6 digits after the decimal point
8	7 digits after the decimal point	18	7 digits after the decimal point
9	8 digits after the decimal point	19	8 digits after the decimal point

Example:

R60 = PLCDF (1,3,2,2,10); The fixed point value of the defined double word is read into R60.

<Var> = PLCDB (<Value1>, <Value2>, <Value3>, <Value4>, <Value5>);	PLC data word BCD
--	------------------------------

The **BCD value** of the defined data words in the PLC is read into the parameter defined with **<Var>**. The PLC number is defined by **<Value1>**, the number of DB or DX by **<Value2>**, the data word number by **<Value3>**, the number of data words by **<Value 4>** and the dimension identifier by **<Value5>**.

The data words are either read in series or in parallel depending on the defined number of data words <Value 4>:

<Value 4>	BCD
1	A data word is read, dimension identifier <Value 5> has no significance.
2	Two data words are read in parallel, dimension identifier <Value 5> has no significance.
3	Three data words are read serially. The <Value 5> dimension identifier specifies how the BCD value is to be read.

Definition of the dimension identifier <Value 5>:

<Value 5>	BCD
100	Value without decimal point
101	Value with decimal point
102	1 digit after the decimal point
103	2 digits after the decimal point
104	3 digits after the decimal point
105	4 digits after the decimal point
106	5 digits after the decimal point
107	6 digits after the decimal point
108	7 digits after the decimal point
109	8 digits after the decimal point

Example:

R51 = PLCDB (1,2,10,1,100); The BCD value of the defined double word is read into R51.

<Var> = PLCDG (<Value1>,<Value2>,<Value3>;

**PLC data word,
floating point**

The **floating-point value** of the defined data words in the PLC is read into the parameter defined with **<Var>**. Two data words are always read into the PLC serially. The PLC number is defined by **<Value1>**, the number of DB or DX by **<Value2>** and the data word number by **<Value3>**.

Example:

R71=1; R72=100; R73=1; The floating point of the defined double word is read into R70.
R70 = PLCDG (R71,R72,R73);

5.3.5.12 Read alarms into the R parameter

<Var> = ALNP ();

NC alarms

The **NC alarms** signaled in the control are read, and the numbers are sequentially deposited starting from the **<Var>** variables, using this command.

Example:

R50 = ALNP (); The existing NC alarms are loaded starting at R50.

The associated channel number is added to the alarm number in the two decimal places after the alarm number, so that the alarm number as well as the channel number in which the alarm occurred, is available in one R parameter.

The contents of the R parameter are interpreted as follows using an example:

R50 = 3004 . 01

Channel number

Alarm number

5.3.5.13 Read alarm pointer into the R parameter

```
<Var> = ALNPZ ( );
```

NC alarm pointer

The number of the entered NC alarms is read in the **<Var>** variable using this command.

Example:

R51 = ALNPZ (); The number of available NC alarms is read into R51.

5.3.5.14 Read system memory into the R parameter

```
<Var> = RSDA (<Value1>,<Value2>,<Value3>);
```

**status of the axes
for digital drives**

A bit from the **status of axes for digital drives** is read into the parameter defined with **<Var>**.

The axis No. is specified by **<Value1>**. The byte address (0 or 1) is defined in **<Value2>**.
The bit address is defined by **<Value3>**.

Example:

R50 = RSDA (1,0,2); The 2nd bit in byte 0 for the 1st axis is read into R50.

```
<Var> = RSDS (<Value1>,<Value2>,<Value3>);
```

**spindle status
for digital drives**

A bit from the **spindle status for digital drives** is read into the parameter defined with **<Var>**.

The spindle No. is specified by **<Value1>**. The byte address (0 to 3) is defined by **<Value2>**.
The bit address is specified by **<Value3>**.

Example:

R51 = RSDS (1,0,1); The 1st bit in the byte 0 for the 1st spindle is read into R51.

<Var> = RSDD (<Value1>,<Value2>,<Value3>);

**unit status
for digital drives**

A bit from the **equipment status for digital drives** is read into the parameter defined with **<Var>**.

The unit No. (1 or 2) is specified by **<Value1>**. The byte address (0 or 1) is defined in **<Value2>**. The bit address is specified by **<Value3>**.

Example:

R52 = RSDD (1,1,3); The 3rd bit in byte 1 for the 1st unit is read into R52.

<Var> = AGS (<Value1>);

active gear stage

Depending on the spindle No., the **active gear stage** is read into the parameter defined with **<Var>**.

The spindle number is specified by **<Value1>**. The master spindle number is used as spindle number if the programmed spindle number **<Value1>** is equal to 0.

Example:

R65 = 1; The active gear stage of the spindle with the number 1 is entered into R65.
R60 = AGS (R65);

5.3.6 Data transfer: R parameter into the system memory

5.3.6.1 Transfer R parameter into the machine data

MDN (<Value1>) = <Value>;

NC machine data

The **NC machine data <Value1>** is loaded via parameter, pointer or constant.

The machine datum number is defined by **<Value1>**.

Example:

MDN (2241) = R90; The parameter R90 is loaded into the machine datum of the 1st software limit switch for the 2nd axis in the positive direction.

MDNBY (<Value1>) = <Value>;

NC machine data, byte

The **NC machine data byte <Value1>** is loaded via a parameter, pointer or constant.

The machine data byte number is defined by **<Value1>**.

Example:

MDNBY (5001) = 11; The name of the angle, which is used in the control, is loaded into the machine data byte (11=Address A).

MDNBI (<Value1>,<Value2>) = <Value>;

NC machine data, bit

The **machine data bit** is loaded via a parameter, pointer or constant, with 1 or 0.

The byte address is defined in **<Value1>** and the bit address in **<Value2>**.

Example:

MDNBI (5400,0) = R92; The machine data bit is loaded via R92. This simultaneously defines whether auxiliary functions are to be output to the PLC or not

MDZ (<Value1>,<Value2>) = <Value>;

MDZBY (<Value1>,<Value2>) = <Value>;

MDZBI <Value1>,<Value2>,<Value3>) = <Value>;

cycles machine data

cycles machine data, byte

cycles machine data, bit

The **machine data for cycles** can be loaded, as for the NC machine data, via parameter, pointer, or constant.

The channel No. is selected using **<Value1>**. When 0 is specified for channel-orientated data, then its own channel is written into. The channel number is always specified with 0 for the central data. The word or byte address is specified by **<Value2>** and the bit address by **<Value3>**.

Examples:

MDZ (0,400) = R93; The channel-orientated machine data for cycles is loaded into its own channel via R93.

MDZBY (1,801) = R94; The machine data byte for cycles in the 1st channel is loaded with the contents of the PATTERN variables R94.

MDZBI (1,802,1)= R95; The 1st bit of the machine data byte for cycles is loaded with the contents of the BOOLEAN variable R95.

MDP (<Value1>) = <Value>; MDPBY (<Value1>) = <Value>; MDPBI (<Value1>,<Value2>) = <Value>;	PLC machine data PLC machine data, byte PLC machine data, bit
---	--

The commands for **LOADING the PLC machine data** via parameter, pointer or constant essentially operate the same as for **LOADING NC machine data**.

Examples:

MDP (20) = R93; The number of the 1st static M function for SINUMERIK System 810 is loaded via R93.

MDPBY (3000) = R94; The 1st PLC user data byte is loaded with the contents of the PATTERN variables R94.

MDPBI (3001,2) = R95; The 2nd bit of the PLC user datum 3001 is loaded with the contents of the BOOLEAN variable R95.

MDD (<Value1>) = <Value> ; MDDBY (<Value1>,<Value2>) = <Value> ; MDDBI (<Value1>,<Value2>,<Value3>) = <Value>;	drives machine data drives machine data, byte drives machine data, bit
---	---

The **machine data for drives** can be loaded, as for NC machine data, via parameter, pointer or constant.

The word address is selected with **<Value1>**. **<Value2>** is used to define whether a low byte (0) or high byte (1) is to be read. The bit address is specified by **<Value3>**.

Examples:

MDD (1200) = R60; The machine datum for the 1st axis is loaded via R60.

MDDBY (2441,0) = R61; The low byte of the machine datum for the 2nd axis is loaded via the PATTERN variable R61.

MDDBI (2442,0,2) = R62; The 2nd bit in the low byte of the machine datum for the 3rd axis is loaded via the BOOLEAN variable R62.

5.3.6.2 Transfer R parameter into the setting data

SEN (<Value1>) = <Value>; SENBY (<Value1>) = <Value>; SENBI (<Value1>, <Value2>) = <Value>;	NC setting data NC setting data, byte NC setting data, bit
--	---

The **NC setting data** can be loaded, as for machine data, as value, byte, or bit.

Examples:

SEN (3041) = R80; The maximum operating field limiting of the 2nd axis is loaded with the contents of parameter R80.

SENBY (5013) = R81; The transmission format for the first V24 interface for data output is loaded with 9600 baud via R81 (R81=11000111).

SENBI (5001,0) = R82; The display of the workpiece-related actual value system is activated via R82. (R82=1).

SEZ (<Value1>,<Value2>) = <Value>; SEZBY (<Value1>,<Value2>) = <Value>; SEZBI (<Value1>, <Value2>,<Value3>) = <Value>;	cycles setting data cycles setting data, byte cycles setting data, bit
---	---

The **cycles setting data** can be loaded, as for machine data, as value, byte, or bit.

Examples:

SEZ (1,400) = R50; The cycles setting data is loaded via variable R50.

SEZBY (0,802) = R51; The cycles setting data byte is loaded via the PATTERN variable R51.

SEZBI (0,802,2) = R52; The 2nd bit of the cycles setting data byte is loaded via the BOOLEAN variable R52.

5.3.6.3 Write R parameter into the tool offsets

The individual tool offset ranges are structured as represented in Section 5.3.5.3.

TOS (<Value1>, <Value2>, <Value3>) = <Value>; **tool offset**

The individual offset values can be loaded with the command LOAD tool offset value via parameter, pointer or constant.

The desired TO range is defined by <Value1>. With the SINUMERIK System 850/880, 0 is always specified here as only one tool offset range is present. For SINUMERIK System 850/880, 0 is written into the TO area which is assigned to the channel via the machine data, at input.

The desired D No. is selected with <Value2>. The desired P No. is selected with <Value3>.

Example:

TOS (1,2,3) = R80; In TO range 1, the tool length in the 2nd axis is loaded with the contents of R80 in offset memory D2.

TOAD (<Value1>, <Value2>, <Value3>) = <Value>; **additive tool offset**

The individual P Nos. are additively changed via parameter, pointer or constant with this command.

The desired TO range is defined with <Value1>. With the SINUMERIK System 810/820, 0 is always specified here because only one tool offset range is present. For SINUMERIK System 850/880, the value in the TO range, which is assigned to the channel via the machine datum, is additively changed when a 0 is specified.

The desired D No. is selected with <Value2>. The desired P No. is selected with <Value3>.

Example:

TOAD (1,2,3) = 50; In TO range 1, the constant 50 is added to the tool length in offset memory D2 for the 2nd axis.

Depending on the NC machine data bits, the offset memories P2 and P5 are loaded as radius or diameter value.

5.3.6.4 Write R parameter into the zero offsets

ZOA (<Value1>, <Value2>, <Value3>) = <Value>; **settable ZO**

The **settable zero offsets** are loaded via R parameter, pointer or constant. The desired zero offset (G54=1 to G57=4) is selected with **<Value1>**, and the desired axis with **<Value2>**. **<Value3>** defines the range into which the value is loaded (coarse value=0 and fine value=1).

Example:

ZOA (1,2,0) = 500; The coarse value of the 1st settable 0 offset (G54) in the 2nd axis is loaded with the constant 500.

ZOFA (<Value1>, <Value2>, <Value3>) = <Value>; **additive
settable ZO**

The individual axis values of the **settable zero offsets** are additively changed via parameter, pointer or constant.

The desired zero offset (G54=1 to G57=4), is selected with **<Value1>**, and the desired axis with **<Value2>**. **<Value3>** defines whether the coarse or fine value of the **settable zero offset is to be changed additively**.

Example:

ZOFA (1,2,0) = -50; The coarse value of the 1st settable zero offset (G54) in the Z axis is reduced by the amount 50 when loading the constant.

Depending on the NC machine data bits, the zero offsets are loaded as radius or diameter value.

ZOPR (<Value1>, <Value2>) = <Value>; **programmable ZO**

The individual axis values of the **programmed zero offsets** are loaded via parameter, pointer or constant. The desired zero offset G58=1 and G59=2 is selected with **<Value1>**, and the desired axis with **<Value2>**.

Example:

ZOPR (1,1) = 50;
ZOPR (1,2) = 20; The 1st programmed zero offset is loaded with constants in the 1st and 2nd axes.

ZOD (<Value2>) = <Value>;

DRF offset

The individual axis values of the **handwheel offset (DRF offset)** are loaded via parameter, pointer or constant.

The desired axis is selected with **<Value2>**.

Example:

ZOD (2) = R61; The handwheel offset of the 2nd axis is loaded with the contents of R61.

ZOPS (<Value2>) = <Value>;

PRESET offset

The **PRESET offsets** of the individual axes are loaded via parameter, pointer, or constant. The actual values of the individual axes can thus be changed.

The desired axis is selected by **<Value2>**.

Example:

ZOPS (1) = 100; The actual value of the 1st axis is set to 100 with the constant.

Depending on the NC machine data bits, the programmed zero offsets, the DRF and PRESET offsets are loaded as radius or diameter value. The PRESET offset is always effective after program end (M2, M30) or "reset". The "set actual value in the program" function cannot be realized with the PRESET offset.

ZOADW (<Value1>,<Value2>,<Value3>) = <Value>;

**settable
coordinate rotation**

The angle of rotation of the **settable coordinate rotation** is loaded via R parameter, pointer or constant.

The channel No. is selected with **<Value1>**. The angle of rotation is loaded in its own channel when 0 is specified.

The desired zero offset (G54=1 to G57=4), is defined by **<Value2>**, and the angle No. by **<Value3>**. At the present time, **<Value3>** is defined with 1.

Example:

ZOADW (0,2,1) = R50; The angle of rotation of the settable coordinate rotation in its own channel for the 2nd zero offset is loaded via R50.

ZOFADW (<Value1>,<Value2>,<Value3>) = <Value>;

**additive,
settable coordinate
rotation**

The angle of the **settable coordinate rotation** is additively changed via R parameter, pointer or constant.

The channel No. is selected by **<Value1>**. The angle of rotation is loaded into its own channel if a 0 is specified. The desired zero offset (G54=1 to G57=4), is defined by **<Value2>**, and the angle No. by **<Value3>**. At the present time, **<Value3>** is defined with 1.

Example:

ZOFADW (2,1,1) = 10; The angle of rotation of the settable coordinate rotation in the 2nd channel for the 1st zero offset is increased by the amount 10 when loading the constants.

When writing from system memories from other channels, which are not located on the same NC CPU, the "CL 800 error" alarm signal is output to the control. Only those channels can be written into which are processed by the same NC CPU, as the selected channel.

ZOPRDW (<Value1>,<Value2>,<Value3>) = <Value>;

**programmable
coordinate rotation**

The angle of rotation of the **programmable coordinate rotation** is loaded via R parameter, pointer or constant.

The channel No. is selected with **<Value1>**. The angle of rotation is loaded into its own channel if 0 is specified. The desired zero offset (G58=1 and G59=2), is determined by **<Value2>**, and the angle No. by **<Value3>**. At the present time **<Value3>** is defined with 1.

Example:

ZOPRDW (1,2,1) = R55; The angle of rotation of the programmable coordinate rotation in the 1st channel for the 2nd zero offset is loaded via R55.

5.3.6 Data transfer: R parameter into the system memory

ZOFPRDW (<Value1>,<Value2>,<Value3>) = <Value>;

**additive
programmable
coordinate rotation**

The angle of rotation of the **programmable coordinate rotation** is additively changed via R parameter, pointer or constant.

The channel No. is selected with **<Value1>**. The angle of rotation is loaded into its own channel when 0 is specified. The desired zero offset (G58=1 and G59=2), is defined by **<Value2>**, and the angle No. by **<Value3>**. At the present time **<Value3>** is defined with 1.

Example:

ZOFPRDW (2,1,1)=10; The angle of rotation of the programmable coordinate rotation in the 2nd channel for the 1st zero offset is increased by 10 when the constants are loaded.

When writing from system memories from other channels, which are not located on the same NC CPU, the "CL 800 error" alarm signal is output to the control. Only those channels can be written into which are processed by the same NC CPU as the selected channel.

5.3.6.5 Write R parameter into the programmed setpoints

PRAP (<Value3>) = <Value>;

programmed axis position

The **travel of each axis** is programmed via parameter, pointer or constant.

The axis is defined with **<Value3>**. Depending on the NC machine data bits, the travel is specified as either radius or diameter.

Example:

PRAP (2) = 100; The 2nd axis travel is programmed via a constant.

PRSS (<Value3>) = <Value>;

programmed spindle speed

The **speed of each spindle** is programmed via parameter, pointer or constant. The spindle No. is defined with **<Value3>**. If the programmed spindle No. is 0, then the number of the master spindle in the current channel is used as spindle number.

Example:

R50 = 1;

PRSS (R50) = 200; The 1st spindle speed is programmed via a constant.

PRAD () = <Value>;

programmed radius

The address defined for the **programmed radius** is entered via parameter, pointer or constant.

Example:

PRAD () = 2; The value 2 is assigned to the radius via a constant.

PANG () = <Value>;

programmed angle

The address defined for the **programmed angle** is entered via parameter, pointer or constant.

Example:

PANG () = R60; The angle is entered via R60.

PRIP (<Value3>) = <Value>;

programmable interpolation parameter

The **interpolation parameter** of each axis is programmed via parameter, pointer or constant. The axis is defined with **<Value3>**. The programmed sequence of the interpolation parameters must coincide with the programmed sequence of the axes.

Example:

(:G02:); PRAP (1)=R50; PRAP (2)=R51; PRIP(1)=R52; PRIP(2)=R53; LF;

Programming a circular block with the programmed interpolation parameter.

PCDA (<Value1>,<Value2>,<Value3>) = <Value>;

programmable control words for digital axis drives

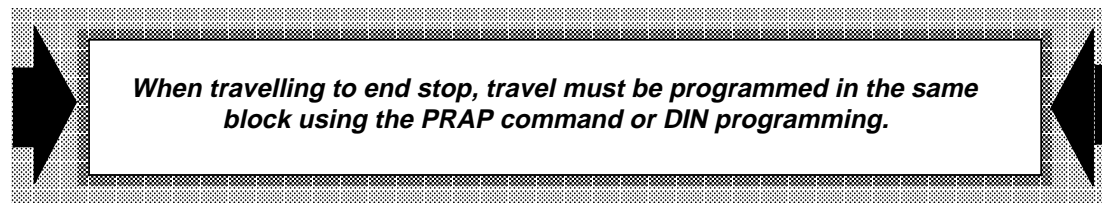
A bit of the **programmed control words for digital axis drives** can be loaded via parameter, pointer or constant.

The axis No. is selected with **<Value1>**. The byte address (0 or 1) is defined in **<Value2>**. The bit address (4 to 6) is specified with **<Value3>**.

Example:

LF; PCDA (1,1,6) = 1; PRAP (1) = 100; LF;

The 6th bit in byte 1 for the 1st axis (travel to end stop) is loaded.



PCDS(<Value1>,<Value2>,<Value3>) = <Value>;

Programmable control words for digital spindle drives

A bit of the **programmed control words for digital spindle drives** can be loaded via parameter, pointer or constant.

The spindle No. is selected with **<Value1>**. The byte address (0 to 5) is defined in **<Value2>**. At the present time the bit address **<Value3>** is not defined.

5.3.6.6 Write R parameter into the PLC signal bits

PLCF (<Value1>,<Value2>,<Value3>) = <Value>;

PLC flag bit

The status of a **flag bit** in the PLC is loaded via parameter, pointer or constant.

The PLC No. is specified with **<Value1>**, the byte address with **<Value2>** and the bit address with **<Value3>**.

Example:

PLCF (1,2,0) = 0;

The status of the defined PLC flag bit is loaded with 0.

PLCW (<Value1>,<Value2>,<Value3>,<Value4>)=<Val.>;

PLC data word bit

The status of a **data word bit** in the PLC is loaded via parameter, pointer or constant.

The PLC number is defined with **<Value1>**, the number of the DB or DX with **<Value2>**, the data word number with **<Value3>** and the bit address with **<Value4>**.

Example:

R60=1; R62=2; R63=4; The status of the defined PLC data word bit is loaded with 1.
 PLCW (R60,R62,R63,2) = 1;

5.3.6.7 Write R parameter into the PLC signal bytes

PLCFB (<Value1>,<Value2>) = <Value>;

PLC flag byte

The status of a **flag byte** in the PLC is loaded via parameter, pointer or constant.

The PLC number is defined with **<Value1>**, and the byte address with **<Value2>**.

Example:

PLCFB (1,1) = 01100111; The status of the defined PLC flag byte is loaded with the constants.

PLCDBL (<Value1>,<Value2>,<Value3>) = <Value>;
PLCDBR (<Value1>,<Value2>,<Value3>) = <Value>;

PLC data word, left
PLC data word, right

The status of a **data word, left** or **right**, in the PLC is loaded via parameter, pointer or constant.

The PLC number is defined with **<Value1>**, the number of the DB or DX with **<Value2>** and the data word number with **<Value3>**.

Example:

R53=10010110;
 PLCDBL (1,2,4) = R53; The status of the defined PLC data word, left is loaded via R53.

5.3.6.8 Write R parameter into the PLC signal words

PLCFW (<Value1>,<Value2>,<Value3>) = <Value>;

PLC flag word

The status of a **flag word** in the PLC is loaded via parameter, pointer or constant.

The PLC number is defined with **<Value1>** and the word address by **<Value2>**. The dimension identifier **<Value3>** specifies how the PLC flag word status is to be loaded.

5.3.6 Data transfer: R parameter into the system memory

Definition of the dimension identifier <Value3>:

<Value3>	Fixed point	<Value3>	BCD
0	Value without decimal point	100	Value without decimal point
1	Value with decimal point	101	Value with decimal point
2	1 digit after the decimal point	102	1 digit after the decimal point
3	2 digits after the decimal point	103	2 digits after the decimal point
4	3 digits after the decimal point	104	3 digits after the decimal point
5	4 digits after the decimal point	105	4 digits after the decimal point
6	5 digits after the decimal point	106	5 digits after the decimal point
7	6 digits after the decimal point	107	6 digits after the decimal point
8	7 digits after the decimal point	108	7 digits after the decimal point
9	8 digits after the decimal point	109	8 digits after the decimal point

Example:

PLCFW (1,3,100) = 2219; The status of the defined PLC flag word is loaded in BCD via a constant.

5.3.6.9 Write R parameter into the PLC signal data words

PLCDF (<Value1>,<Value2>,<Value3>, <Value4>,<Value5>) = <Value>;	PLC data word, fixed point
---	---------------------------------------

The **fixed-point value** of a **data word** or **double word** (serial or parallel) is loaded into the PLC via parameter, pointer or constant.

The PLC number is defined by **<Value1>**, the number of the DB or DX by **<Value2>**, the data word number by **<Value3>**, and the number of data words by **<Value4>**.

The dimension identifier **<Value5>** specifies how the fixed-point value of the data word or double word is to be loaded.

Definition of the dimension identifier <Value5>:

<Value 5>	Fixed point: Data word or double word, serial	<Value 5>	Fixed point: Double word, parallel
0	Value without decimal point	10	Value without decimal point
1	Value with decimal point	11	Value with decimal point
2	1 digit after the decimal point	12	1 digit after the decimal point
3	2 digits after the decimal point	13	2 digits after the decimal point
4	3 digits after the decimal point	14	3 digits after the decimal point
5	4 digits after the decimal point	15	4 digits after the decimal point
6	5 digits after the decimal point	16	5 digits after the decimal point
7	6 digits after the decimal point	17	6 digits after the decimal point
8	7 digits after the decimal point	18	7 digits after the decimal point
9	8 digits after the decimal point	19	8 digits after the decimal point

Example:

PLCDF (1,3,2,2,10) = 24500; The fixed-point value of the defined PLC double word is loaded via a constant.

PLCDB (<Value1>,<Value2>,<Value3>, <Value4>,<Value5>) = <Value>;	PLC data word, BCD
---	-------------------------------

The **BCD value** is loaded into the defined PLC data words via parameter pointer or constant.

The PLC number is defined with **<Value1>**, the number of the DB or DX by **<Value2>**, the data word number with **<Value3>**, the number of data words with **<Value4>** and the dimension identifier with **<Value5>**.

Data is loaded serially or in parallel depending on the defined number of data words <Value4>.

<Value4>	BCD
1	A data word is read, dimension identifier <Value 5> has no meaning.
2	Two data words are read in parallel, dimension identifier <Value 5> has no meaning.
3	Three data words are serially read. The dimension identifier <Value 5> specifies how the BCD value is to be read.

Definition of the dimension identifier <Value5>:

<Value 5>	BCD
100	Value without decimal point
101	Value with decimal point
102	1 digit after the decimal point
103	2 digits after the decimal point
104	3 digits after the decimal point
105	4 digits after the decimal point
106	5 digits after the decimal point
107	6 digits after the decimal point
108	7 digits after the decimal point
109	8 digits after the decimal point

Example:

PLCDB (1,2,10,1,100) = 2413; The BCD value of the defined PLC data word is loaded via a constant.

PLCDG (<Value1>,<Value2>,<Value3> = <Value>;

**PLC data word,
floating point**

The **floating-point value** is loaded in the defined PLC data words via parameter, pointer or constant. Two data words are always serially loaded in the PLC.

The PLC number is defined with **<Value1>**, the number of the DB or DX with **<Value2>** and the data word number with **<Value3>**.

Example:

PLCDG (1,100,1) = 21; The constant is loaded as floating-point value in the defined data words.

5.3.6.10 Write R parameter into the alarms

ALNZ () = <Value>;

cycle alarms

A **cycle alarm** at the control can be displayed with this command. The alarm number is specified with parameter, constant or pointer with **<Value>**.

Example:

ALNZ () = 4001; The cycle alarm is displayed at the control.

5.3.6.11 Write R parameter into the system memory

SATC (<Value1>,<Value2>) = <Value>;

**spindle acceleration
time constant**

The **spindle acceleration constant** is changed via parameter, pointer or constant. The **spindle acceleration time constant** specified under **<Value>** does not overwrite the values specified in the machine data, but an internal data location. The machine data is again active after thread machining has been cancelled (e.g. G00...), after machine data change, after RESET and after POWER ON. If the programmed **spindle acceleration time constant <Value>** is less than 4, this value is automatically increased to 4.

The spindle number is defined with **<Value1>** and the gear stage with **<Value2>**. If the programmed spindle number **<Value1>** is zero, the master spindle number in the current channel is used as spindle number.

Example:

R50=5; R51=12; The system memory for the spindle acceleration constant of the 1st spindle of gear stage 5 is loaded with the corresponding value K for 12.

SATC (1,R50) = R51;

Determining the spindle acceleration constant K:

$$K = \frac{8191 \cdot \text{IPO sampling time}}{\langle \text{Value} \rangle \cdot 2 \cdot 2}$$

5.3.7 Mathematical and logical functions

5.3.7.1 Value assignment with arithmetic operations

<code><Var> = (<Value1> + <Value2>);</code>	addition
<code><Var> = (<Value1> - <Value2>);</code>	subtraction
<code><Var> = (<Value1> * <Value2>);</code>	multiplication
<code><Var> = (<Value1> / <Value2>);</code>	division

The result of an arithmetic operation is assigned to the parameter <Var>.

Example:

```
PROGRAM 50;  
:  
PAR INTEGER: R10, R11, R12,R13:=Name;  
LOCAL INTEGER: R50, R51, R52;  
:  
BEGIN  
:  
R50 = 17;  
R51 = 4;  
R52 = 3;  
:  
R10 = R50 + R51;  
(* contents R10 equals 21 *)  
:  
R11 = R10 - R51;  
(* contents R11 equals 17 *)  
:  
R12 = R10 * R11;  
(* contents R12 equals 357 *)  
:  
Name = R12 / R52;  
(* contents R13 (name) equals 119 *)  
:  
END.
```

Extended arithmetic for arithmetic operations

A variable can be defined by arithmetic operations via the extended arithmetic.

<Var>=<Value1>+,-,*or/<Value2>{+,-,*or/<Value3>. .<Valuen>;

The variable is defined by a chain calculation of arithmetic operations. Multiplication and division have priority over addition and subtraction. This can be modified by the applicable use of parentheses. A variable, constant or function (SIN, COS etc.) can be substituted for <Value (1-n)>.

Example:

```
PROGRAM 81;
:
PAR REAL: R10:= Z_VALUE, R20, R30;
LOCAL REAL: R51;

BEGIN
  Z_VALUE = Z_VALUE+ R20 · R30;
  :
  Z_VALUE = SIN (Z_VALUE) + 5;
  :
  Z_VALUE = (Z_VALUE + R20) · R30;
  :
  Z_VALUE = (Z_VALUE + 5/3) · 10 /80.009;
  :
  R51 = Z_VALUE;
  :
END.
```

5.3.7.2 Arithmetic functions

<Var> = ABS (<Value>;

generate absolute amount

The **ABS** statement loads the absolute amount of a specified quantity **<Value>** into parameter **<Var>**.

Example:

```
:
LOCAL INTEGER: R50;
LOCAL REAL: R51,R52;
:
R50 = ABS(-253);
(* contents R50 =+253 *)
R51 = ABS(R52) (*e.g. R52= -3.8435);
(* contents R51 =+3.8435 *)
:
```

The ABS statement does not influence the data type, i.e. an integral or real value remains an integral or real value, even after the command has been processed.

<Var> = SQRT (<Value>);

square root

The **SQRT** statement extracts the square root of a specified quantity **<Value>**.

When this statement is applied, the **real** and **integral** data types can be used.

Example:

```
:  
R60 = SQRT(25);  
(* contents R60 = 5*)  
R61 = SQRT(R60);  
(* contents R61 = the square root of 5 *)  
:
```



It is not possible to extract the square root of a negative quantity!

<Var> = SQRTS (<Value1>,<Value2>);

root of the sum of the square

The **SQRTS** statement takes the root of the sum of (**<Value 1>² + <Value 2>²**).

Example:

```
:  
R61 = 10;  
R62 = SQRTS(5,R61);  
(* contents R62 = 11.1803 *)  
:
```

5.3.7.3 Arithmetic procedures

INC (<Var>);**increment**

The **INC** statement increments the variable index by one.

Example:

```

:
R70 = 1; (* R70 has index value 1 *)
INC (R70); (* R70 has index value 2 *)

```

DEC (<Var>);**decrement**

The **DEC** statement decrements the variable index by one.

Example:

```

:
R70 = 1; (* R70 has index value 1 *)
DEC (R70); (* R70 has index value 0 *)
:

```

TRUNC (<Var>);**integral part**

The **TRUNC** statement forms the integral part of a variable of the real data type (without rounding off).

Example:

```

PROGRAM 55;
:
LOCAL REAL: R60, R70;
:
BEGIN
:
R60 = 1.9;
TRUNC(R60);
(* contents R60 = 1*)
:
R70 = -2.3;
TRUNC(R70)
(* contents R70 = 2 *)
:
END.

```

5.3.7.4 Trigonometric functions

<Var> = SIN (<Value>);

sine

The **SIN** statement generates the sine of an angle whose magnitude is specified by **<Value>**.

The result is stored in variable **<Var>**. The **SIN** statement operates with integral and real arguments over the range -360° to +360°.

Example:

```
:  
R10 = SIN (45) ;  
(* contents R10 = 0.70710678 *)  
:  
R60 = 15.55 ;  
R10 = SIN (R60) ;  
(* contents R10 = 0.26807920 *)  
:
```

<Var> = COS (<Value>);

cosine

The **COS** statement generates the cosine of an angle whose magnitude is specified by **<Value>**.

The result is stored in the relevant variable **<Var>**. The **COS statement** operates with integral and real arguments over the range -360° to +360°.

Example:

```
:  
R11 = COS (45) ;  
(* contents R11 = 0.70710678 *)  
:  
R72 = 30 ;  
R12 = COS (R72) ;  
(* contents R12 = 0.86602540 *)  
:
```

<Var> = TAN (<Value>);

tangent

The **TAN** statement generates the tangent of an angle whose magnitude is specified by **<Value>**.

The result is stored in the relevant variable **<Var>**. The **TAN** statement operates with integral and real arguments over the range 0° to +/-89.999999°, +/- 90.00001° to 269,9999°, +/- 270.0001° to +/- 360°.

Example:

```
:  
R12 = TAN (45) ;  
(* contents R12 = 1 *)  
:
```

<Var> = ARC SIN (<Value>);

arc sine

The **ARC SIN** statement (arc sine function) is the inverse trigonometric function of the sine function.

The result of the statement is the computed angle in degrees, and is stored in the variable **<Var>**.

Example:

```

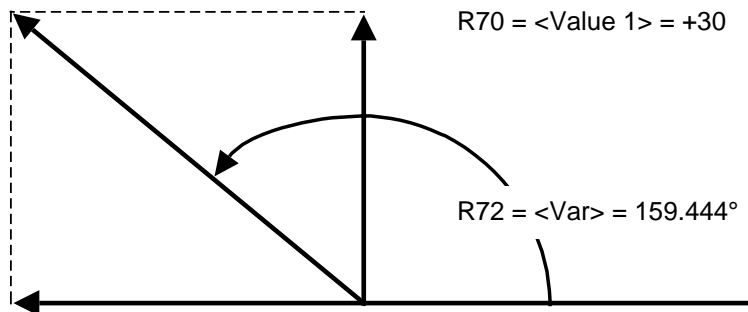
:
R10 = ARC SIN (0.70710678) ;
(* contents R10 = 45 *)
:

```

<Var> = ANGLE (< Value1>,<Value2>);
--

angle formed by two vector components
--

The **ANGLE** statement computes, from the two vectors **<Value1>** and **<Value2>** an angle, which is stored in variable **<Var>**.



Example:

```

:
R70 = 30; R71 = -80;
R72 = ANGLE (R70,R71);
(* R72 has the value 159.444 *)
:

```


5.3.7.5 Logarithmic functions

<Var> = LN (<Value>);

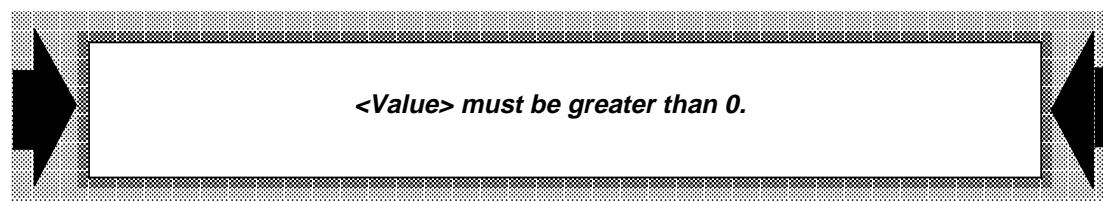
natural logarithm

The **LN** statement (natural logarithm) forms the natural logarithm of the specified variable **<Value>**.

The result of the statement is stored in variable **<Var>**.

Example:

```
:  
R80 = LN (10);  
(*contents of R80 = 2.3025846 *)  
:
```



<Var> = INV LN (<Value>);

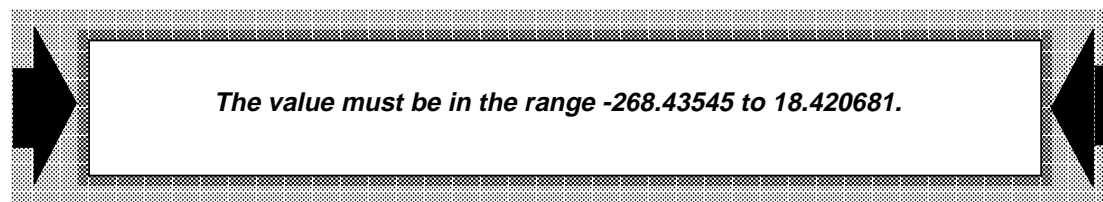
exponential function

The **INV LN** statement (exponential function) raises the specified variable **<Value>** to the power of e.

The result of the logarithmic operation is stored in variable **<Var>**.

Example:

```
:  
R80 = INV LN (2.5);  
(*contents of R80 = 12.182496 *)  
:
```



5.3.7.6 Logical functions

The logical **OR-XOR-AND-NAND statements** combine all bit positions of pattern variables **<Var1>** and **<Value>** .

The result of the logic operation is stored in variable **<Var>**.

The truth table for the logic operations has the following form:

For the PATTERN data type	
<Var> = <Var 1> OR	<Value>;
<Var> = <Var 1> XOR	<Value>;
<Var> = <Var 1> AND	<Value>;
<Var> = <Var 1> NAND	<Value>;
<Var> = NOT <Value>;	

OR
EXCLUSIVE-OR
AND
NOT-AND
NEGATION

Bit position 0-7		Result of logic op. of <Var1> with <Value>			
Var1	Value	OR	XOR	AND	NAND
0	0	0	0	0	1
1	0	1	1	0	1
0	1	1	1	0	1
1	1	1	0	1	0

The logical **NOT statement** inverts the status of all bit positions of the PATTERN variable **<Value>** and stores the result in a variable **<Var>**.

Example:

```

PROGRAM 59;
LOCAL PATTERN: R50, R51, R52, R53,
               R54, R55,R56
BEGIN
  R50 = B00101100; R51 = B10110011;

  R52 = R50 OR R51;
  (* contents of R52 = B10111111 *)
  R53 = R50 XOR R51;
  (* contents of R53 = B10011111 *)
  R54 = R50 AND R51;
  (* contents of R54 = B00100000 *)
  R55 = R50 NAND R51;
  (* contents of R55 = B11011111 *)
  R56 = NOT R50;
  (* contents of R56 = B11010011 *)
END.

```

For the BOOLEAN data type

<Var> = <Var 1> ORB	<Value>;	OR BIT
<Var> = <Var 1> XORB	<Value>;	EXCLUSIVE-OR BIT
<Var> = <Var 1> ANDB	<Value>;	AND BIT
<Var> = <Var 1> NANDB	<Value>;	NOT-AND BIT
<Var> = NOTB <Value>;		NEGATION BIT

The logical **ORB-XORB-ANDB-NANDB** statements combine the bits of the BOOLEAN variables **<Var1>** and **<Value>** with each other.

The logical **NOTB** statement inverts the statuses of BOOLEAN variable **<Value>**. The result of the logic operation is stored in variables **<Var>**.

Example:

```
PROGRAM 60;
:
LOCAL BOOLEAN: R50, R51, R52, R53, R54, R55, R56;
:
BEGIN
:
R50 = 1;
R51 = 0;
R52 = R50 ORB R51; (* contents of R52 = 1 *)
R53 = R50 XORB R51; (* contents of R53 = 0 *)
R54 = R50 ANDB R51; (* contents of R54 = 0 *)
R55 = R50 NANDB R51; (* contents of R55 = 1 *)
R56 = NOTB R50; (* contents of R56 = 0 *)
:
END.
```

Extended arithmetic for arithmetic functions

A variable can also be defined by chaining several logical functions. The desired priority is achieved by using parentheses.

Example:

```
PROGRAM 82;
:
PAR PATTERN: R10, R15, R20;
PAR BOOLEAN: R25, R30;
:
BEGIN
R30 = R 10 OR R20 AND R15;
:
R10 = NOTB R30;
:
R30 = NOT (R10 OR B11101101) AND B00011111;
:
R30 = NOTB (R10 NANDB R25),
:
END.
```

5.3.7.7 Logical procedures

CLEAR BIT (<Var>.<Const.>);**clear bit**

A particular bit in the bittern pattern of an R parameter is cleared (i.e. assigned 0) with the **CLEAR BIT** statement. The variable **<Var>**, containing the bit pattern, and the bit number **<Const>** of the bit to be cleared (a digit between 0 . . . 7) must be specified in the command.

Example:

```

:
LOCAL PATTERN: R60;
:
BEGIN
:
  R60 = B01100111;
:
  CLEAR BIT (R60.6); (*the bit pattern of R60 is B00100111 *);
:
END.

```

For this statement, the R parameter must be of the PATTERN data type.

SET BIT (<Var>.<Const.>);**set bit**

A particular bit in the bit pattern of an R parameter is set (loaded with "1") with the **SET BIT** statement. The relevant variable **<Var>**, and the bit number **<Const>** (a digit between 0 . . . 7) must be specified in the command.

Example:

```

:
LOCAL PATTERN: R70;
:
BEGIN
:
  R70 = B00000000;
:
  SET BIT (R70.2); (*the bit pattern of R70 is B00000100 *);
:
END.

```

For this statement, the R parameter must be of the PATTERN data type.

5.3.8 NC-related functions

5.3.8.1 Changing the program and machine reference points

POS MSYS ;	specifying a position referred to the machine actual value system
-------------------	--

A position, referred to the machine actual value system, is specified with the **POS MSYS** statement (position machine actual value system) and is approached by the specified axes. The command is only effective blockwise. As many axes can be specified as the NC can simultaneously traverse.

The axis numbers 1...24 are available. The assignment of the axis number to the axis address is specified in the NC machine data 2000-2039. The axis positions to be approached are programmed using the DIN code or PRAP statement.

All zero point-, PRESET-, and the DRF offsets are suppressed with the **POS-MSYS** statement, i.e. the tool offsets must be cancelled for approaching a fixed machine point.

Example:

Approaching the fixed tool change point X=1000 and Z=500

```
PROGRAM 508;  
:  
LOCAL REAL: R50:= POS_X, R51:= POS_Z,  
            R53:= X, R54:= Z;  
:  
BEGIN  
:  
  POS_X = 1000; POS_Z =500;LF;  
  POS MSYS; PRAP (X) = POS_X; PRAP (Z) = POS_Z; LF;  
  (*the programmed tool change point X=1000 and Z=500 referred to the machine zero  
  point are approached*)  
:  
END.
```

Additional block limits are generated in the target code with "LF;"

5.3.8.2 Single functions

<Var1> = REP REF (<Var2>);

prepare reference

With the **PREP REF** statement (prepare reference), a contour programmed in a subroutine (also Sprint contouring) is broken down into individual blocks: Data is stored in R parameters.

The contour element is stored in a total of eight R parameters, from R parameter **<Var 1>** (Rn).

The prepare reference requires a total of four R parameters as input data. The first of these R parameters is specified with **<Var 2>** (Rm).

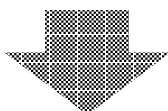
The input control parameter (Rm+3) must be set to "1" before the 1st **PREP REF** call. The first contour element of the subroutine is stored in R parameters beginning with the starting point (Rm+1, Rm+2) . This point is not programmed in the subroutine. The subroutine should only contain the geometrical values of the contour.

The command **PREP REF** sets the control parameter to "0", so that the values from the next contour element are loaded with each subsequent **PREP REF** call.

If **PREP REF** detects the "subroutine end", the output control parameter (Rn+7) is automatically set to "1" or "2".

Prerequisite

1. Contour in the subroutine
2. Starting points
3. Control parameter (Rm+3) = 1

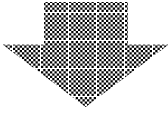


Parameter

<Var2>:	Rm	Subroutine number is deposited in the contour
	Rm+1	Starting point of the contour, vertical to the axis
	Rm+2	Starting point of the contour, horizontal to the axis
	Rm+3	Input control parameter

Permissible parameter range: "**Rm** = R0-R197 and R900-R997"

PREP REF loads in:



- <Var 1>:**
- Rn Block starting point, vertical axis (2nd axis of the G16 plane)
 - Rn+1 Block starting point, horizontal axis (1st axis of the G16 plane)
 - Rn+2 Block end point, vertical axis
 - Rn+3 Block end point, horizontal axis
 - Rn+4 Interpolation parameter for vertical axis
 - Rn+5 Interpolation parameter for horizontal axis
 - Rn+6 G-function
 - Rn+7 Output control parameter
- ↓
- Rn+7 = 0: block without M17
 - Rn+7 = 1: block with M17
 - Rn+7 = 2: only M17 in block

Permissible parameter range "**Rn** = R0-R192 and R900-R992".

The **interpolation parameters** determine the distance from the starting point of the circle to the center of the circle in the axis coordinates.

The **G-function** determines the interpolation type (e.g. G01, G02, . . .).

Example: refer to intersection computation "INT SEC"

<Var1> = INT SEC (<Var 2>, <Var 3>); **intersection computation**

The intersection with a contour element is determined with the **INT SEC** statement (intersection computation).

The intersection computation requires a total of eight parameters as input data for the first contour element. The first R parameter number is specified with <Var 2> (Rn). The second contour, starting at <Var 3> (Rr) is not realized at the present time, however an R parameter must be specified under <Var 3> at programming. The output data of the intersection computation is deposited in a total of three R parameters, starting at R parameter <Var 1> (Rm). Both axis values is required to determine the intersection. The output data of the reference preparation is used as input data for the intersection computation. The intersection computation can be used in conjunction with the prepare reference PREP REF.

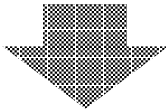
- <Var 2>:**
- Rn Block starting point, vertical axis (2nd axis of the G16 plane)
 - Rn+1 Block starting point, horizontal axis (1st axis of the G16 plane)
 - Rn+2 Block end point, vertical axis
 - Rn+3 Block end point, horizontal axis
 - Rn+4 Interpolation parameter for vertical axis
 - Rn+5 Interpolation parameter for horizontal axis
 - Rn+6 G-function
 - Rn+7 Control parameter

Permissible parameter range "Rn=R0-R192 and R900-R992".

The **interpolation parameters** determine the distance from the starting point of the circle to the center of the circle in the axis coordinates.

The **G-function** determines the interpolation type (e.g. G01, G02,...).

INTEL SEC loads in:



<Var 1>: Rm=1 Intersection found
 Rm=0 Intersection not found
 Rm+1 Intersection, vertical axis (2nd axis of the G16 plane)
 Rm+2 Intersection, horizontal axis (1st axis of the G16 plane)

Permissible parameter range "Rm=R0-R197 and R900-R997"

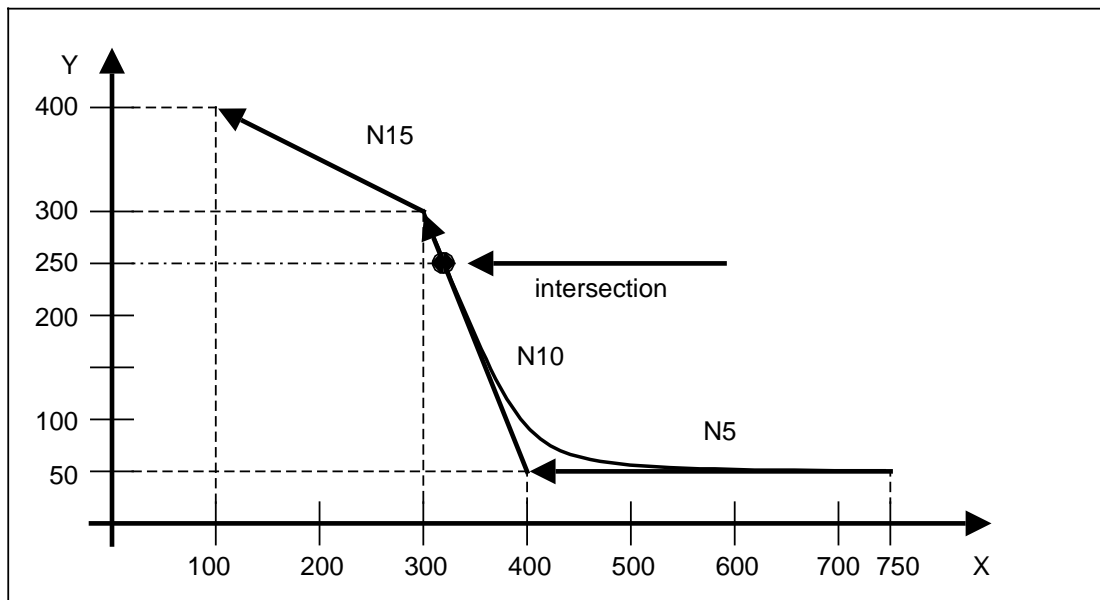
The second contour, starting at **<Var3>** is not realized at the present time, however a R parameter must be specified under **<Var3>**.

The intersection computation can be used in conjunction with the prepare reference (PREP REF).

The output data of the reference preparation is used as input data for the intersection computation (refer to example). The intersection computation requires both axis values of the destination point to determine the intersection.

Example for intersection computation with INT SEC statement:

The object is to find the intersection, for example with a programmed contour at Y = 250 mm.



Contour subroutine:

```
L20  
N5 X400 Y50 B100 LF  
N10 X300 Y300 LF  
N15 X100 Y400 M17 LF
```

Main program:

```
PROGRAM 30;  
:  
LOCAL REAL: R50:=subroutine_No, R51:=A_con_Y, R52:=A_con_X,  
R53:=start_ref, R54:=contour_1,  
R62:=contour_2, R70:=S_travelled,  
R71:=intersection_Y, R72:=intersection_X;  
(* R50 to R53 input data PREP REF *)  
(* R54 to R61 output data PREP REF, which are simultaneously the input data from INT  
SEC*)  
(* R62 to R69 input data from INT SEC *)  
(* R70 to R72 output data from INT SEC *)  
:  
BEGIN  
:  
LF; (: GO Y250 X750 :); LF; (* with rapid traverse  
to starting point *)  
subroutine_No=20; A_con_X=750; A_con_Y=50;  
start_ref=1; S_travelled=0;  
WHILE S_travelled=0 DO  
BEGIN  
LF; contour_1=PREP REF (subroutine_No); LF;  
S_travelled=INT SEC (contour_1, contour_2);  
(: Y250 X0 :); (* destination point *)  
END;  
(* result of intersection computation is stored in intersection_Y(R71)  
and intersection_X (R72) from R parameter S_travelled (R70)  
For this example: intersection_X=320  
intersection_Y=250 *)  
:  
END.
```

<Var> = PREP CYC;**preparation for cycles**

The PREP CYC statement (preparation for cycles) is a function which stores two assertions in a total of two R parameters, from R parameter **<Var>** onwards:

- Rn This is loaded with a numerical value which corresponds to the safety clearance of 1 mm in the current input format
 Rn = 1 for metric (G71)
 Rn=0.03937 for inch (G70)
- Rn+1 Rn+1=1 radius programming, 1st axis
 Rn+1=2 diameter program, 1st axis
 (dependent on the NC machine datum)

Permissible parameter range **Rn=R0-R98** and **R900-R998**

Example:

```
PROGRAM 500;
:
LOCAL REAL; R52, R53;
:
BEGIN
:
(* G71 is active *)
(* diameter programming is current *)
R52 = PREP CYC;
(* contents of R52 = 1 *)
(* contents of R53 = 2 *)
:
END.
```

STOP DEC;**stop decoding until
buffer is empty**

During program processing, several program blocks are decoded in advance in the controller and loaded into the NC buffers. This results in faster program processing but can lead to a faulty program run in conjunction with NC commands (read actual value, measure, data transfer NC-PLC). The **STOP DEC** statement (stop decoding) stops pre-decoding of the NC blocks, present after the statement, until the block is processed with the **STOP DEC** statement. This ensures that the buffers are empty and the information required in the next NC blocks is available.

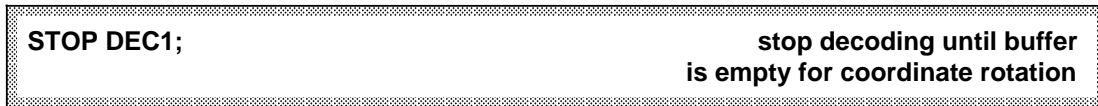
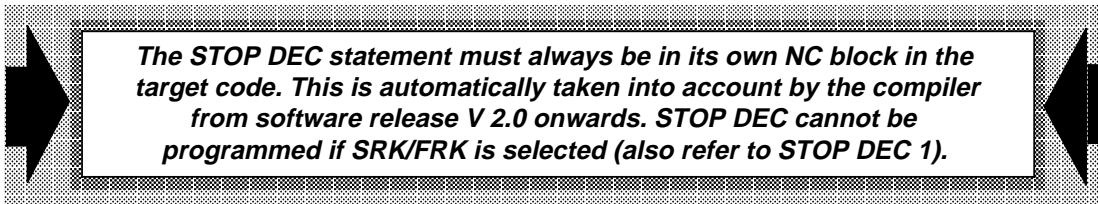
For the following information from the interface control, the **STOP DEC** statement must be programmed if required in the NC blocks which follow:

- Machine data
- Setting data
- Tool offsets
- Zero offsets
- R parameters
- "Mirror image" signal

The **STOP-DEC** command must be programmed prior to each "read actual value" in its own channel and after each "measure".

Example:

```
      :  
(: M94 :);  
      (* part number is transferred from the PLC to R60 *)  
STOP DEC;  
      (* current part number can be evaluated in the next NC block *)  
IF R60 > 100 THEN  
      (* branching according to part number *)  
  BEGIN  
    (: G54 X... Y... :);          (* NV 1 *)  
  END;  
  ELSE  
    BEGIN  
      (: G55 X... Y... :);          (* NV 2 *)  
    END;  
  ENDIF;  
      :
```

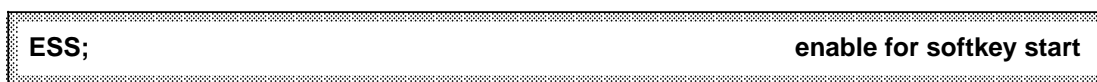
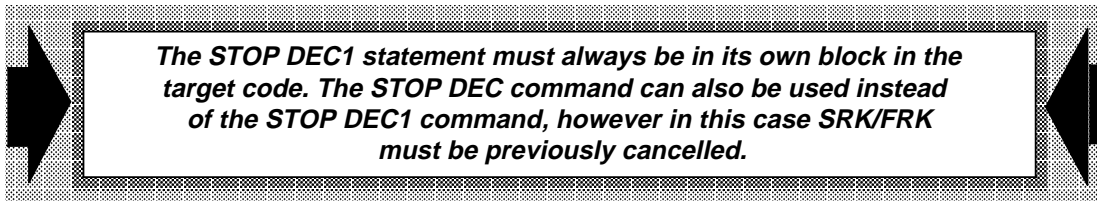


If the rotation angle of the selectable or programmable coordinate rotation is loaded using the CL 800 language, this angle value is immediately used for the calculation. This can mean, that this angle was already calculated in previous traverse blocks. If the angle is only to be valid at the next block, then all previous blocks must be executed (empty buffer).

This can also be attained with SRK/FRK selected by programming **STOP DEC1**. The decoding is then only re-activated when all buffers are empty up to coordinate rotation.

Example:

```
STOP DEC1;
ZOADW (0, 1, 1) = 30;
```



A subroutine start via softkey can be configured with the softkey function 69. The program enable for softkey start is realized with the **ESS** command which must be located at the start of the statement part in the program.

Example:

```
PROGRAM 1;
CHANNEL NC 1;

PAR INTEGER: R0, R25, R49;
LOCAL INTEGER: R50;

BEGIN

ESS;      (* enable subroutine start for softkey *)
:
```

5.3.8.3 Measuring functions

<Var>=MEAS M (<Value>);

**flying measurement relative
to machine zero**

The actual values of the traversing axis are determined at the instant of an input signal from the measuring sensor with the **MEAS M** statement. The actual values are detected directly from the measuring circuit of the control on identification of the switching edge of the measuring sensor. The actual values are deposited starting from parameter <Var> with increasing axis number, and are referred to the machine zero.

For rotary axes, the result is deposited, starting from parameter <Var> (Rn), in a total of two R parameters, depending on an NC machine data bit.

The following R parameters are loaded:

Rn = Position within a rotation
Rn+1 = Number of rotations

Finally, the control generates a "delete remaining travel" i.e. the remaining travel (setpoint-actual value differences) of all axes are deleted. Setpoint 0 is specified as step function from the control.

The braking travel of the axis is still traversed, i.e. the following errors are reduced. Thus, the next traversing blocks must be programmed in the absolute dimension (G90).

The sensed actual values of the traversing axes at the instant of measurement are deposited, with increasing axis number, starting at the parameter defined with <Var>. The measuring input number (1 or 2) is input with <Value>.

The axes travel (command positions) are programmed in the same NC block using DIN code or PRAP statement. The command positions of the axes are referred to the tool zero.

Example:

```
PROGRAM 605;  
:  
LOCAL REAL: R70:=X_COMMAND, R71:=Z_COMMAND,  
             R72:=DIFF_X, R73:=DIFF_Z,  
             R93:=X_ACT, R94:=Z_ACT;  
BEGIN  
X_COMMAND=100; Z_COMMAND=200; LF;  
X_ACT=MEAS M (1); PRAP (1)=X_COMMAND; PRAP (2)=Z_COMMAND; LF;  
STOP DEC;  
DIFF_X=X_COMMAND-X_ACT;  
DIFF_Z=Z_COMMAND-Z_ACT;  
END.
```

Also refer to STOP DEC statement (Section 5.3.8.2). G00 or G01 must be active at the flying measurement. A maximum of 2 axes can be simultaneously measured.

5.3.8.4 Program influence

REL (<Value1>, <Value2>);

**read-in inhibit via
external input**

This command stops program processing until the defined external input is available.

The external input is defined with the byte address (1 or 2) <Value1> and the bit address (0...7) <Value2>.

Example:

LF; REL (1, 2); (: G00 G90 X5 Z20:); LF;

If the external input has "0" signal, the program is stopped after the position has been reached. Program processing is continued if the level at the external input **changes** from "0" to "1".

The processing is not interrupted if a "1" signal was already present.

INTA (<Value1>, <Value2>, <Value3>);

**axis-specific remaining
travel delete via external input**

This command permits an **axis-specific remaining travel delete**, dependent on the signal level of the defined external input.

The axis in which the remaining travel is to be deleted is defined with <Value1>. Remaining travel delete is realized for all axes programmed in the block when a 0 is input.

The external input is defined with the byte address (1 or 2) <Value2> and the bit address (0...7) <Value3>. If the bit address <Value2> is positive, scanning is for a "1" signal, and when it is negative, scanning is for a "0" signal.

Example:

LF, INTA (1, 1, 0); (:X20 Z10 F100:); LF;

The block end criterium is fulfilled when the travel motion is interrupted by the defined external signal or the programmed position is reached.

*If axis-specific remaining travel delete is identified for an axis
which is in an interpolation relationship with another axis,
then both axes are stopped.*

5.3.9 I/O statements

5.3.9.1 NC I/O functions

WRT PIC (<Value4>, <Value5>);

menu selection from NC program

Menus (displays with softkey texts) can be called up from the user or standard sector from the NC program using this command. **<Value4>** is used to define as to whether the menu is to be called from the user sector or standard sector:

<Value4> = 0 : User sector
 = 1 : Standard sector

The menu number is specified by **<Value5>**.

Example:

WRT PIC (1, 20); The standard display for the data input/output is called.

RECALL PIC;

return jump to the output menu

This statement can be used to jump back to the menu which was selected before WRT PIC was called, after any menu was called with the WRT PIC statement.

Example:

For instance, at the present time the PLC status display is selected.

```
WRT PIC (1, 20); (* call the data input/output display *)  
:  
RECALL PIC; (* return to the PLC status display *)  
:
```

5.3.9.2 General I/O functions,

PORT (<Value>);

select V24 interface

For data input/output it is necessary to preselect the applicable interface. This can be realized with the command select V24 interface. The number of the V24 interface can be defined with <Value>

Example:

PORT (1); The first V24 interface is selected for the input/output.

The interface selected via the operator panel is not influenced by this command. Before data is output with statements OOTP ZOA, OOTP DATA, OOTP ETX or input with the statement INP, the interface must be selected in the program with the statement PORT.

OOTP ZOA (<Value1>);

zero offset output via V24

Zero point offsets or coordinate rotation angle can be output **via V24** using this command.

The channel No. is defined with <Value1>. The channel-independent settable zero point offsets G54 to G57 are output when 0 is input, and when the channel No. is input, the channel-specific rotation angle of the settable coordinate rotation of the defined channel is output.

Example:

PORT (2); OOTP ZOA (0); The settable zero offsets are output through the second V24 interface.

The V24 interface must be selected with the PORT statement before data output. A machine data bit is used to select as to whether the block change is to be inhibited or not during the actual transfer. If the data to be output is to be changed in the subsequent program section, then a STOP DEC must be programmed after the command.

When 0 is specified, the channel-independent settable zero offsets (G54 to G57) are output in the defined channel.

When a channel number is specified, the channel-specific angle of rotation of the settable coordinate rotation is output in the defined channel.

OUTP DATA (<Value1>, <Value1>, <Value1>); **output data via V24**

Specific **data** can be output **via V24** using this statement. **<Value2>** is used to specify the data type:

<Value 2>	Data type
1	Main programs
2	Subroutines
5	NC machine data
6	Tool offsets
8	Setting data
9	PLC machine data

The start address is defined with **<Value3>** and the end address of the data block with **<Value4>**.

Example:

PORT (1); OUTP DATA (1, 1, 10); The existing parts program starting with program number 1 to 10 are output through the first V24 interface.

*The V24 interface must be selected with the **PORT** statement before data output. A machine data bit is used to select as to whether the block change is to be inhibited or not during the actual transfer. If the data to be output is to be changed in the subsequent program section, then a **STOP DEC** must be programmed after the command.*

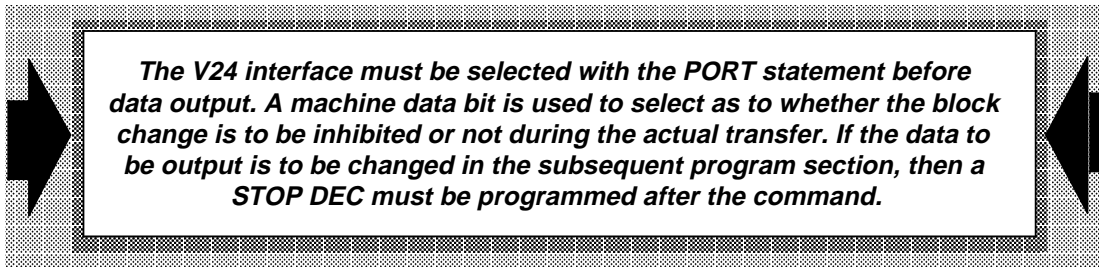
OUTP PARA (<Value1>, <Value3>, <Value4>); **parameter output via V24**

Individual **R parameter blocks** can be output via V24 using this command. The channel No. is defined with **<Value1>**.

R parameters are output from their own channel when 0 is input. The channel No. is always specified with 0 for central variables. The start address of the R parameter block is defined with **<Value3>** and the end address with **<Value4>**.

Example:

PORT (1); OUTP PARA (1, 100, 120); The parameter block (R100 to R120) from the 1st channel is output via the first V24 interface.



INP (<Value 2>);

**read-in data
via V24**

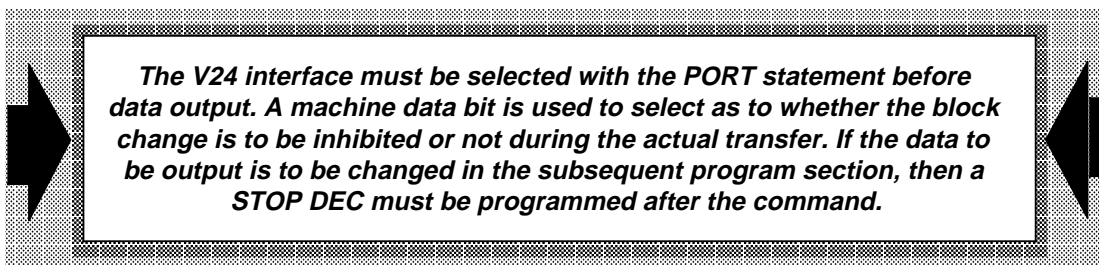
Data input can be started from the program via V24 with this command. Using <Value2>, a check can be made using the command as to which data type is involved:

<Value 2>	Data type
0	No data type check
1	Main programs
2	Subroutines
3	Clear programs
5	NC machine data
6	Tool offsets
7	Zero offsets
8	Setting data
9	PLC machine data
10	R parameters

Example:

PORT (1); INP (0);

Data input is started via the first V24 interface



OUTP ETX;

output ETX via V24

The end-of-transmission character ETX (e.g. 03H) can be output via the defined V24 interface with this statement. Using setting data it can be determined, for each E24 interface, whether the output is with or without trailer.

Example:

Setting data 5017.7=1

PORT (1); (* select first V24 interface*)
OUTP PARA (1, 100, 120); (* output R parameter *)
OUTP ETX; (* output ETX without trailer*)

5.3.9.3 Operator control functions

CHAN = (<Value>);

**select channel No. for
screen display**

The screen display is channel-specific and can be selected at the operator panel. This selection can be activated from the program with the command **select channel number for screen display**. The channel No. is selected with **<Value>**.

Example:

CHAN = 2; The screen display is updated for channel 2.

Section 6

-Command summary-

Summary:

- 6.1 General statements for program configuration
- 6.2 Declarations
- 6.3 Repeat statements
- 6.4 Decision statements
- 6.5 Unconditional branching
- 6.6 Data transfer, general
- 6.7 Data transfer: System memory into the R parameter
- 6.8 Data transfer: R parameter into the system memory
- 6.9 Mathematical and logical functions
- 6.10 NC-specific functions
- 6.11 I/O statements

6 Command summary

The following SINUMERIK basic versions are valid for the SINUMERIK 810 control:

SINUMERIK 810 GA1/GA2/GA3

SINUMERIK 810G GA2/GA3

The SINUMERIK basic versions are valid for the SINUMERIK 820 control:

SINUMERIK 820 GA2/GA3

The functions for the SINUMERIK 805 and 840 controls are in preparation.

6.1 General statements for program configuration

Program frame statements							
CL 800 statement	Function	805	810	820	840	850	880
PROGRAM <Progr. number.>;	Definition of a program <Progr. number>: 810/820 = 1...999 850/880 = 1...9999 805 = ... 840 = ...		*	*		*	*
CHANNEL NC <Channel No.>;	NC Channel No.		#	#		#	#
ESS;	Enable for softkey start		*	*			
ID (<ID des.>);	ID designation of subroutine		*	*		*	*
PW (<Password>);	Password		*	*		*	*
BEGIN	Start of statement part or block		*	*		*	*
END;	End of statement block		*	*		*	*
END.	End of statement part (program end)		*	*		*	*
LF;	End of block		*	*		*	*
(*Remarks*)	Remarks exclusively for CL 800 source language (not passed on by compiler)		*	*		*	*
(<:DIN code>);	DIN code (passed on by the compiler)		*	*		*	*

Explanation:

* Function implemented

Function implemented, but skipped in the NC

6.2 Declarations

Reference to higher-level variables which are defined separately							
CL 800 statement	Function	805	810	820	840	850	880
CENTRAL: "<List name>";	Reference to list in which the central variables are declared		#	#		#	#
GLOBAL: "<List name>";	Reference to lists in which the global variables are declared channel-dependent and channel-independent		#	#		#	#

Declaration of constants and variables in the subroutine							
CL 800 statement	Function	805	810	820	840	850	880
PAR INTEGER: R<Var No.> [:=<Var name>] {,R<Var No.>:=<Var name>};	Declaration of INTEGER transfer variables		#	#		#	#
PAR REAL: R<Var No.> [:=<Var name>] {,R<Var No.>:=<Var name>};	Declaration of REAL transfer variables		#	#		#	#
PAR PATTERN: R<Var No.>:=<Var name> {,R<Var No.>:=<Var name>};	Declaration of PATTERN transfer variables		#	#		#	#
PAR BOOLEAN: R<Var No.>:=<Var name> {,R<Var No.>:=<Var name>};	Declaration of BOOLEAN transfer variables		#	#		#	#
LOCAL INTEGER: R<Var No.>:=<Var name> {,R<Var No.>:=<Var name>};	Declaration of local INTEGER transfer variables		#	#		#	#
LOCAL REAL: R<Var No.> [:=<Var name>] {,R<Var No.>:=<Var name>};	Declaration of local REAL transfer variables		#	#		#	#
LOCAL PATTERN: R<Var No.>:=<Var name> {,R<Var No.>:=<Var name>};	Declaration of local PATTERN variables		#	#		#	
LOCAL BOOLEAN: R<Var No.>:=<Var name> {,R<Var No.>:=<Var name>};	Declaration of local BOOLEAN variables		#	#		#	#
CONST: <Const. name>:=<Number> {,<Const. name>:=<Number>};	Declaration of constants		#	#		#	#
POINTER: R<Var No.>:=<Var name> {,R<Var No.>:=<Var name>};	Declaration of R parameters as POINTERS		#	#		#	#
LABEL: <Name> {,<Name>};	Label declaration <Name>: Label for jump designation. In the program the label is set before the corresponding statement.		#	#		#	#

Explanation:

- * Function implemented
- # Function implemented, but skipped in the NC

Frame statement for declaration lists, central/global							
CL 800 statement	Function	805	810	820	840	850	880
EXTERNAL<List number>;	Definition of an EXTERNAL file (contains higher/level variables) <List number>; 1...9999		#	#		#	#
CHANNEL NC or COM<Channel No.>;	NC- or COM Channel No.		#	#		#	#
ID (<ID des.>);	ID designation of subroutine		#	#		#	#
PW (<Password>);	Password		#	#		#	#
ENDEXTERN	End of declaration		#	#		#	#

Declaration of variables in EXTERNAL files							
CL 800 statement	Function	805	810	820	840	850	880
GLOBAL INTEGER: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of global INTEGER variables (channel-specific)		#	#		#	#
GLOBAL REAL: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of global REAL variables (channel-specific)		#	#		#	#
GLOBAL PATTERN: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of global PATTERN variables (channel-specific)		#	#		#	#
GLOBAL BOOLEAN: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of global BOOLEAN variables (channel-specific)		#	#		#	
CENTRAL INTEGER: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of central INTEGER variables		#	#		#	#
CENTRAL REAL: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of central REAL variables		#	#		#	#
CENTRAL PATTERN: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of central PATTERN variables		#	#		#	#
CENTRAL BOOLEAN: R<Var No.>[:=<Var name>] {,R<Var No.>[:=<Var name>]]};	Declaration of central BOOLEAN variables		#	#		#	#

Explanation:

* Function implemented

Function implemented, but skipped over in the NC

6.3 Repeat statements

REPEAT loop							
CL 800 statement	Function	805	810	820	840	850	880
REPEAT <Statement>; UNTIL "Condition"; 1)	Repeat statement with scanning of repeat condition at end		*	*		*	*

WHILE loop							
CL 800 statement	Function	805	810	820	840	850	880
WHILE "Condition" 1) DO <Statement>;	Repeat statement with scanning of repeat condition at the start		*	*		*	*

WHILE INT loop							
CL 800 statement	Function	805	810	820	840	850	880
WHILE INT <Value 1>, <Value 2> DO <Statement>;	Repeat statement with scanning of status of a specific external input at the start		*	*			

FOR TO loop							
CL 800 statement	Function	805	810	820	840	850	880
FOR<Var>:= <Value1>TO<Val.2>DO <Statement>;	FOR TO loop (count loop, index is incremented)		*	*		*	*

FOR DOWNTO loop							
CL 800 statement	Function	805	810	820	840	850	880
FOR<Var>:= <Value 1>DOWNTO <Value 2>DO <Statement>;	FOR DOWNTO loop (count loop, index is decremented)		*	*		*	*

Explanation:

* Function implemented

1) "Condition":

a) <Var>= Boolean variable

b) <Var>.<Const>=Bit from pattern (
<Var>=pattern variable)
(
<Const>= bit number)

c) <Var> "Vop" <Value> = Variable, pointer or constant

d) Chain of conditions

Comparison operator "Vop"

= equal to

<> not equal to

> greater than

>= greater than or equal to

< less than

<= less than or equal to

6.4 Decision statements

IF THEN ELSE branch							
CL 800 statement	Function	805	810	820	840	850	880
IF "condition" 1) THEN <Statement 1>; [ELSE <Statement 2>]; ENDIF;	IF THEN ELSE branch, with which the program branches depending on comparison between two variables based on =, <>, >, >=; <, <=		*	*		*	*

IF INT THEN ELSE branch							
CL 800 statement	Function	805	810	820	840	850	880
IF INT <Value 1>, <Value 2> THEN <Statement 1>; [ELSE <Statement 2>]; ENDIF;	IF THEN ELSE branch, with which the program branches depending on the status of an external input.		*	*			

CASE branch							
CL 800 statement	Function	805	810	820	840	850	880
CASE <Var> = <Value 1>= <Statement 1>; . . . = <Value n>: <Statement n>; [OTHERWISE : <Statement n+1>]; ENDCASE;	CASE branch		*	*		*	*

6.5 Unconditional jump

Unconditional jump							
CL 800 statement	Function	805	810	820	840	850	880
GOTO <Label>;	Unconditional jump to NC block		*	*		*	*

Explanation:

* Function implemented

1) "Condition":

a) <Var>= Boolean variable

b) <Var>.<Const>=Bit from pattern (<Var>=Pattern variable)
(<Const>= Bit number)

c) <Var> "Vop" <Value> = Variable, pointer or constant

d) Chain of conditions

Comparison operator "Vop"

= equal to

<> not equal to

> greater than

>= greater than or equal to

< less than

<= less than or equal to

6.6 Data transfer, general

Data transfer: R parameter/R parameter								
CL 800 statement	Function	Buf. mem. No.	805	810	820	840	850	880
CLEAR (<Var>);	Clear R parameter	-		*	*		*	*
<Var>=<Value>;	Load variable with value	-		*	*		*	*
XCHG (<Var1>,<Var2>);	Exchange contents of variables	-		*	*		*	*

Data transfer: R parameter / input buffer memory for numerical variables								
CL 800 statement	Function	Buf. mem. No.	805	810	820	840	850	880
CLEAR MIB (<Start input buffer mem. No.>, <End input buffer mem. No.>);	Clear input buffer memory	0...149(249) 0...499		*	*		*	*
<Var>=MIB (<Input buffer mem. No.>);	Read input buffer memory	0...149(249) 0...499		*	*		*	*
MIB (<Input buffer memory No.>)=<Value>;	Write into input buffer memory	0...149(249) 0...499		*	*		*	*

Explanation:

* Function implemented

6.7 Data transfer: System memory into the R parameter

Machine data											
CL 800 statement	Function	Ch. No.	Word or byte address	Byte	Bit addr.	805	810	820	840	850	880
<Var>= MDN (<Word addr.>);	NC machine data	-	0 ... 4999	-	-		*	*		*	*
<Var>= MDNBY (<Byte addr.>);	NC machine data, bytes	-	5000 ... 6999	-	-		*	*		*	*
<Var>= MDNBI (<Byte addr.>,<Bit addr.>);	NC machine data, bits	-	5000 ... 6999	-	0..7		*	*		*	*
<Var>= MDZ (<Ch. No.>,<Word addr.>);	Cycle machine data	0..16	0...449, 1000...4149	-	-					*	*
<Var>= MDZBY (Ch. No.>,<Byte addr.>);	Cycle machine data, bytes	0..16	800..949, 7000...8049	-	-					*	*
<Var> = MDZBI (<Ch. No.> <Byte addr. >, <Bit addr.>).	Cycle machine data, bits	0..16	800..949, 7000...8049	-	0..7					*	*
<Var>= MDP (<Word addr.>);	PLC machine data	-	0 ... 5999	-	-		*	*		*	*
<Var>= MDPBY (<Byte addr.>);	PLC machine data, bytes	-	6000 ... 8999	-	-		*	*		*	*
<Var>= MDPBI (<Byte addr.>,<Bit addr.>);	PLC machine data, bits	-	6000 ... 8999	-	0..7		*	*		*	*
<Var>= MDD (<Word addr.>);	Drive machine data	-	0x ... 256x ¹⁾ 4000 ... 4960	-	-		*	*			
<Var>= MDDBY (<Word addr.>,<Byte>);	Drive machine data, bytes	-	244x , 252x ¹⁾ 4610 , 4630	0/1	-		*	*			
<Var>=MDDBI (<Word addr.>, <Byte>,<Bit addr.>);	Drive machine data, bits	-	244x , 252x ¹⁾ 4610 , 4630	0/1	0..7		*	*			

Setting data											
CL 800 statement	Function	Ch. No.	Word or byte address	Byte	Bit addr.	805	810	820	840	850	880
<Var>= SEN (<Word addr.>);	NC setting data	-	0 ... 4999	-	-		*	*		*	*
<Var>= SENBY (<Byte addr.>);	NC setting data, bytes	-	5000 ... 9999	-	-		*	*		*	*
<Var>= SENBI (<Byte addr.>,<Bit addr.>);	NC setting data, bits	-	5000 ... 9999	-	0..7		*	*		*	*
<Var>= SEZ (<Ch. No.>,<Word addr.>);	Cycles setting data	0..16	0 ... 499	-	-					*	*
<Var>= SEZBY (Ch. No.>,<Byte addr.>);	Cycles setting data, bytes	0..16	800 ... 949	-	-					*	*
<Var>= SEZBI (<Ch. No.>, <Byte addr.>,<Bit addr.>);	Cycles setting data, bits	0..16	800 ... 949	-	0..7					*	*

Explanation:

* Function implemented

1) x=axis address

6.7 Data transfer: System memory into the R parameter

Tool offsets											
CL 800 statement	Function	TO range	D No.	T No.	P No.	805	810	820	840	850	880
<Var>= TOS (<TO range.>,<D No.>,<P No.>);	Tool offset	0...16	1...818	-	0...15						*
		0...8	1...409	-	0...15					*	*
		0	1...99	-	0...7(9)		*	*			

Zero offsets											
CL 800 statement	Function	Ch. No.	Group	Axis No.	Coarse/fine	805	810	820	840	850	880
<Var>= ZOA (<Group>,<Axis No.>,<Coarse/fine>);	Settable zero offset (G54-G57)	-	1...4	1...24	0/1						*
		-	1...4	1...12	0/1					*	*
		-	1...4	1...4	0/(1)		*	*			
<Var>= ZOPR (<Group>,<Axis No.>);	Programmable zero offset (G58, G59)	-	1 or 2	1...24	-						*
		-	1 or 2	1...12	-					*	*
		-	1 or 2	1...4	-		*	*			
<Var>= ZOE (<Axis No.>);	External zero offset from PLC	-	-	1...24	-						*
		-	-	1...12	-					*	*
		-	-	1...4	-		*	*			
<Var>= ZOD (<Axis No.>);	DRF offset	-	-	1...24	-						*
		-	-	1...12	-					*	*
		-	-	1...4	-		*	*			
<Var>= ZOPS (<Axis No.>);	PRESET offset	-	-	1...24	-						*
		-	-	1...12	-					*	*
		-	-	1...4	-		*	*			
<Var>= ZOS (<Axis No.>);	Total offset	-	-	1...24	-						*
		-	-	1...12	-					*	*
		-	-	1...4	-		*	*			
<Var>= ZOADW (<Ch. No.>,<Group>,<Angle No.>);	Settable coordinate rotation	0...16	1...4	-	-						*
		0...8	1...4	-	-					*	*
		0...3	1...4	-	-		*	*			
<Var>= ZOPRDW (<Ch. No.>,<Group>,<Angle No.>);	Programmable coordinate rotation	0...16	1 or 2	-	-						*
		0...8	1 or 2	-	-					*	*
		0...3	1 or 2	-	-		*	*			

Programmed setpoints												
CL 800 statement	Function	Ch. No.	linear/rev.	Axis No. Spindle No.	Byte addr.	Bit addr.	805	810	820	840	850	880
<Var>= PRSS (<Ch. No.>,<Spindle No.>);	Programmed spindle speed	0...3	-	0...1	-	-		*	*			
<Var>= PRVC (<Ch. No.>,<Value 2>);	Programmed cutting speed (Value 2=0)	0...16	-	-	-	-						*
		0...8	-	-	-	-					*	*
		1	-	-	-	-		*	*			
<Var>= PCDA (Axis No.>,<Byte addr.>,<Bit addr.>);	Programmed control word for digital axis drives	-	-	1...4	0/1	0...7		*	*			
<Var>= PCDS (Spindle No.>,<Byte addr.>,<Bit addr.>);	Programmed control word for digital spindle drives	-	-	1	0...5	0...7		*	*			

Explanation:

* Function implemented

Actual values											
CL 800 statement	Function	Ch. No.	Axis No. Spindle No.	Group		805	810	820	840	850	880
<Var>= ACPW (<Axis No.>);	Axis position workpiece-related	-	1...24	-							*
		-	1...12	-			*	*		*	
		-	1...4	-							
<Var>= ACPM (<Axis No.>);	Axis position machine-related	-	1...24	-						*	*
		-	1...12	-			*	*		*	
		-	1...4	-							
<Var>= ACP (<Axis No.>);	Actual axis position	-	1...24	-						*	*
		-	1...12	-			*	*		*	
<Var>= ACSP (<Spindle No.>);	Actual spindle position	-	0..6	-						*	*
		-	0..4	-			*	*		*	
		-	0...2	-							
<Var>= ACSS (<Spindle No.>);	Actual spindle speed	-	0..6	-						*	*
		-	0..4	-			*	*		*	
		-	0...2	-							
<Var>= ACAS (<Channel No.>);	Axis number of the current plane/master spindle number	0...16	-	-						*	*
		0...8	-	-			*	*		*	
		0...2	-	-							
<Var>= ACD (<Channel No.>);	D function, actual	0...16	-	-						*	*
		0...8	-	-			*	*		*	
		0	-	-							
<Var>= ACG (<Channel No.>, <Group>);	G function, actual	0...16	-	0...23						*	*
		0...8	-	0...23			*	*		*	
		0	-	0...23							

Program data											
CL 800 statement	Function	Ch. No.		Bit addr.		805	810	820	840	850	880
<Var>= SOB (<Channel No.>, <Bit addr.>);	Special bits	0...16,99		0...7						*	*
		0...8,99		0...7			*	*		*	
		0...2,99		0...7							
<Var>= PPCH;	Actual channel No.	-		-						*	*

PLC signal bits											
CL 800 statement	Function	PLC No.	DB/DX No.	Byte addr. Word addr.	Bit addr.	805	810	820	840	850	880
<Var>= PLCI (<PLC No.>, <Byte addr.>, <Bit addr.>);	PLC input bit	1...4	-	0...127	0...7					*	*
		1...2	-	0...127	0...7					*	
<Var>= PLCQ (<PLC No.>, <Byte addr.>, <Bit addr.>);	PLC output bit	1...2	-	0...127	0...7					*	*
			-	0...127	0...7					*	
<Var>= PLCF (<PLC No.>, <Byte addr.>, <Spindle No.>);	PLC flag bit	1...2	-	0...255	0...7					*	*
			-	0...255	0...7					*	
<Var>= PLCW (<PLC No.>, <DB/ DX No.>, <DW No>,<Bit addr.>);	PLC data word bit	1...2	1...255/ 1000...1255	0...2043	0...15					*	*
			1...255	0...255	0...15					*	

Explanation:

* Function implemented

6.7 Data transfer: System memory into the R parameter

PLC signal bytes											
CL 800 statement	Function	PLC No.	DB/DX No.	Byte addr. Word addr.	Bit addr.	805	810	820	840	850	880
<Var>= PLCIB (<PLC No.>, <Byte addr.>);	PLC input byte	1...4	-	0...127							*
		1...2	-	0...127						*	*
<Var>= PLCQB (<PLC No.>, <Byte addr.>);	PLC output byte	1...4	-	0...127							*
		1...2	-	0...127						*	*
<Var>= PLCPB (<PLC No.>, <Byte addr.>);	PLC peripheral byte	1...4	-	0...127							*
		1...2	-	0...127						*	*
<Var>= PLCFB (<PLC No.>, <Byte addr.>);	PLC flag byte	1...4	-	0...255							*
		1...2	-	0...255						*	*
<Var>= PLCDL (<PLC No.>, <DB/DX No.>, <Word addr.>);	PLC data word, left	1...4	1...255/ 1000...1255	0...2043							*
		1...2	1...255	0...255						*	*
<Var>= PLCDR (<PLC No.>, <DB/DX-Nr.>, <Word addr.>);	PLC data word, right	1...4	1...255/ 1000...1255	0...2043							*
		1...2	1...255	0...255						*	*

PLC signal words											
CL 800 statement	Function	PLC No.	Word addr. Timer addr. Count. addr.	DIM ident.	805	810	820	840	850	880	
<Var>= PLCIW (<PLC No.>, <Word addr.>, <DIM ident.>);	PLC input word	1...4	0...126	0...9, 100...109						*	
		1...2	0...126	0...9, 100...109					*	*	
<Var>= PLCQW (<PLC No.>, <Word addr.>, <DIM ident.>);	PLC output word	1...4	0...126	0...9, 100...109						*	
		1...2	0...126	0...9, 100...109					*	*	
<Var>= PLCPW (<PLC No.>, <Word addr.>, <DIM ident.>);	PLC peripheral word	1...4	0...126	0...9, 100...109						*	
		1...2	0...126	0...9, 100...109					*	*	
<Var>= PLCFW (<PLC No.>, <Word addr.>, <DIM ident.>);	PLC flag word	1...4	0...254	0...9, 100...109						*	
		1...2	0...254	0...9, 100...109					*	*	
<Var>= PLCT (<PLC No.>, <Timer addr.>);	PLC timer	1...4	0...127	-						*	
		1...2	0...254	-					*	*	
<Var>= PLCC (PLC No.>, <Counter addr.>);	PLC counter	1...4	0...127	-						*	
		1...2	0...254	-					*	*	

Explanation:

* Function implemented

PLC signal data words												
CL 800 statement	Function	PLC No.	DB/DX No.	Word No.	No. DW	DIM ident.	805	810	820	840	850	880
<Var>= PLCDF (<PLC- No.>, <DB/DX No.>, <Word No.>, <No.DW>, <DIM ident.>);	PLC data word, fixed point	1...4 1...2	1...255/ 1000...1255 1...255	0...2043 0...255	1,2 1,2	0...9, 10...19, 0...9, 10...19					*	*
<Var>= PLCDB (<PLC No.>, <DB/DX No.>, <Word No.>, <No.DW>, <DIM ident.>);	PLC data word, BCD	1...4 1...2	1...255/ 1000...1255 1...255	0...2043 0...255	1...3 1...3	100...109 100...109					*	*
<Var>= PLCDG (<PLC No.>, <DB/DX No.>, <Word No.>);	PLC data word, floating point	1...4 1...2	1...255/ 1000...1255 1...255	0...2043 0...255	- -	- -					*	*

Alarms												
CL 800 statement	Function	Ch. No.					805	810	820	840	850	880
<Var>= ALNP ();	NC alarms	-									*	*

Alarm pointers												
CL 800 statement	Function	Ch. No.					805	810	820	840	850	880
<Var>= ALNPZ ();	NC alarm pointer	-									*	*

System memory												
CL 800 statement	Function	Axis No. Spindle No. Unit No.	Byte addr.	Bit addr.			805	810	820	840	850	880
<Var>= RSDA (<Axis No.>, <Byte addr.>, <Bit addr.>);	Axis status for digital drives	1...4	0/1	0...7				*	*			
<Var>= RSDS (<Spindle No.>, <Byte addr.>, <Bit addr.>);	Spindle status for digital drives	1	0...3	0...7				*	*			
<Var>= RSDD (<Unit No.>, <Byte addr.>, <Bit addr.>);	Unit status for digital drives	1/2	0/1	0...7				*	*			
<Var>= AGS (<Spindle No.>);	Active gearstage	0/1						*	*			

Explanation:

* Function realized

6.8 Data transfer: R parameter into system memory

Machine data											
CL 800 statement	Function	Ch. No.	Word or byte addr.	Byte	Bit addr.	805	810	820	840	850	880
MDN (<Word addr.>)=<Val.>;	NC machine data	-	0 ... 4999		-		*	*		*	*
MDNBY (<Byte addr.>)=<Val.>;	NC machine data, bytes	-	5000 ... 6999		-		*	*		*	*
MDNBI (<Byte addr.>, <Bit addr.>)=<Value>;	NC machine data, bits	-	5000 ... 6999		0..7		*	*		*	*
MDZ (<Channel No.>, <Word addr.>)=<Val.>;	Cycle machine data	0..16 0..8	0...449, 1000...4149 0...449, 1000...4149	- -	- -					*	*
MDZBY (<Channel No.>, <Byte addr.>)=<Val.>;	Cycle machine data, bytes	0..16 0..8	800..949, 7000...8049 800..949, 7000...8049	- -	- -					*	*
MDZBI (<Channel No.>, <Byte addr.>,<Bit addr.>)=<Value>;	Cycle machine data, bits	0..16 0..8	800..949, 7000...8049 800..949, 7000...8049	- -	0..7 0..7					*	*
MDP (<Word addr.>)=<Val.>;	PLC machine data	-	0 ... 5999		-		*	*		*	*
MDPBY (<Byte addr.>)=<Val.>;	PLC machine data, bytes	-	0 ... 1999 6000 ... 8999		- -		*	*		*	*
MDPBI (<Byte addr.>, <Bit addr.>)=<Value>;	PLC machine data, bits	-	2000 ... 3999 6000 ... 8999 2000 ... 3999		- 0..7 0..7		*	*		*	*
MDD (<Word addr.>)=<Val.>;	Drive machine data	-	88x ... 256x 1) 4220 ... 4960	-	-		*	*			
MDDBY (<Word addr.>, <Byte>)=<Value>;	Drive machine data, bytes	-	244x , 4610 1)	0/1	-		*	*			
MDDBI (<Word addr.>,<Byte>, <Bit addr.>)=<Value>;	Drive machine data, bits	-	244x , 4610 1)	0/1	0..7		*	*			

Setting data											
CL 800 statement	Function	Ch. No.	Word or byte addr.	Byte	Bit addr.	805	810	820	840	850	880
SEN (<Word addr.>)=<Val.>;	NC setting data	-	0 ... 4999		-		*	*		*	*
SENBY (<Byte addr.>)=<Value>;	NC setting data, bytes	-	5000 ... 9999		-		*	*		*	*
SENBI (<Byte addr.>, <Bit addr.>)=<Value>;	NC setting data, bits	-	5000 ... 9999		0..7		*	*		*	*
SEZ (<Channel No.>, <Word addr.>)=<Val.>;	Cycle setting data	0..16 0..8	0 ... 499 0 ... 499	- -	- -					*	*
SEZBY (<Channel No.>, <Byte addr.>)=<Val.>;	Cycles setting data,bytes	0..16 0..8	800 ... 949 800 ... 949	- -	- -					*	*
SEZBI (<Channel No.>, <Byte addr.>,<Bit addr.>)=<Value>;	Cycles setting data,bits	0..16 0..8	800 ... 949 800 ... 949	- -	0..7 0..7					*	*

Explanation:

* Function implemented

1) x = Axis address

Tool offset											
CL 800 statement	Function	TO area	D No.	T No.	P No.	805	810	820	840	850	880
TOS (<TO area>,<D No.>,<P No.>) =<Value>;	Tool offset	0...16	1...818	-	0...15					*	*
		0...8	1...409	-	0...15		*	*			
		0	1...99	-	0...7(9)						
TOAD (<TO area>,<D No.>,<P No.>) =<Value>;	Additive tool offset	0...16	1...818	-	0...15					*	*
		0...8	1...409	-	0...15		*	*			
		0	1...99	-	0...7(9)						

Zero offset											
CL 800 statement	Function	Ch. No.	Group	Axis No.	Coarse/fine	805	810	820	840	850	880
ZOA (<Group>,<Axis No.>,<coarse/fine>) =<Value>;	Adjustable zero offset (G54-G57)	-	1...4	1...24	0/1					*	*
		-	1...4	1...12	0/1		*	*			
		-	1...4	1...4	0/(1)						
ZOFA (<Group>,<Axis No.>,<coarse/fine>) =<Value>;	Adjustable zero offset (G54-G57) additive	-	1...4	1...24	0/1					*	*
		-	1...4	1...12	0/1		*	*			
		-	1...4	1...4	0/(1)						
ZOPR (<Group>,<Axis No.>) =<Value>;	Programmable zero offset (G58, G59)	-	1 od.2	1...24	-					*	*
		-	1 od.2	1...12	-		*	*			
		-	1 od.2	1...4	-						
ZOD (<Axis No.>) =<Value>;	DRF offset	-	-	1...24	-					*	*
		-	-	1...12	-		*	*			
		-	-	1...4	-						
ZOPS (<Axis No.>) =<Value>;	PRESET offset	-	-	1...24	-					*	*
		-	-	1...12	-		*	*			
		-	-	1...4	-						
ZOADW (<Ch. No.>,<Group>,<Ch. No.>,<Angle No.>) =<Value>;	Adjustable coordinates rotation	0...16	1...4	-	-					*	*
		0...8	1...4	-	-		*	*			
		0...3	1...4	-	-						
ZOFADW (<Ch. No.>,<Group>,<Angle No.>) =<Value>;	Adjustable coordinates rotation additive	0...16	1...4	-	-					*	*
		0...8	1...4	-	-		*	*			
		0...3	1...4	-	-						
ZOPRDW (<Ch. No.>,<Group>,<Angle No.>) =<Value>;	Programmable coordinates rotation absolute	0...16	1 od.2	-	-					*	*
		0...8	1 od.2	-	-		*	*			
		0...3	1 od.2	-	-						
ZOFPRDW (<Ch. No.>,<Group>,<Angle No.>) =<Value>;	Programmable coordinates rotation additive	0...16	1 od.2	-	-					*	*
		0...8	1 od.2	-	-		*	*			
		0...3	1 od.2	-	-						

Explanation:

* Function implemented

6.8 Data transfer: R parameter into system memory

Programmed setpoints												
CL 800 statement	Function	Ch. No.	Linear/ rev.	Axis No. Spindle No.	Byte addr.	Bit addr.	805	810	820	840	850	880
PRAP (<Axis No.>)=<Value>;	Programmed axis position	- - -	- - -	1...24 1...12 1...4	- - -	- - -			*	*	*	*
PRSS (<Spindle No.>)=<Value>;	Programmed spindle speed			0, 1	-	-		*	*			
PRAD ()=<Value>;	Programmed radius	-	-	-	-	-		*	*		*	*
PANG ()=<Value>;	Programmed angle	-	-	-	-	-		*	*		*	*
PRIP (<Axis No.>)=<Value>;	Programmed interpolation parameter	- - -	- - -	1...24 1...12 1...4	- - -	- - -			*	*	*	*
PCDA (<Axis No.>, <Byte addr.>,<Bit addr.>)	Progr. control word for digital axis drive	-	-	1...4	0/1	4...6		*	*			
PCDS (<Spindle No.>,<Byte addr.>, <Bit addr.>)=<Value>;	Progr. control word for digital spindle drive	-	-	1	0...5	undef.		*	*			

PLC signal bits												
CL 800 statement	Function	PLC No.	DB/DX No.	Byte addr. Word addr.	Bit addr.	805	810	820	840	850	880	
PLCF (<PLC No.>,<Byte addr.>, <Bit addr.>)=<Value>;	PLC flag bit	1...4 1...2	- -	0...255 0...255	0...7 0...7					*	*	
PLCW (<PLC No.>, <DB/DX No.>,<DW No.>, <Bit addr.>)=<Value>;	PLC data word bit	1...4 1...2	1...255/ 1000...1255 1...255	0...2043 0...255	0...15 0...15					*	*	

PLC signal bytes												
CL 800 statement	Function	PLC No.	DB/DX No.	Byte addr. Word addr.	Bit addr.	805	810	820	840	850	880	
PLCFB (<PLC No.>, <Byte addr.>)=<Value>;	PLC flag byte	1...4 1...2	- -	0...255 0...255						*	*	
PLCDBL (<PLC No.>,<DB/DX No.>, <Word addr.>)=<Value>;	PLC data word, left	1...4 1...2	1...255/ 1000...1255 1...255	0...2043 0...255						*	*	
PLCDBR (<PLC No.>,<DB/DX No.>, <Word addr.>)=<Value>;	PLC data word, right	1...4 1...2	1...255/ 1000...1255 1...255	0...2043 0...255						*	*	

PLC signal words												
CL 800 statement	Function	PLC No.	Word addr.	Dim ident.	805	810	820	840	850	880		
PLCFW (<PLC No.>,<Word addr.>, <DIM ident.>)=<Value>;	PLC flag word	1...4 1...2	0...254 0...254	0...9, 100...109 0...9, 100...109					*	*		

Explanation:

* Function implemented

PLC signals, data words												
CL 800 statement	Function	PLC No.	DB/DX No.	Word No	No. DW	DIM ident.	805	810	820	840	850	880
PLCDF(<PLC No.>,<DB/DX No.>,<Word No.>,<Qty. DW>,<DIM ident.>)=<Value>;	PLC data word, fixed point	1...4	1...255/ 1000...1255	0...2043	1,2	0...9, 10...19						*
		1...2	1...255	0...255	1,2	0...9, 10...19					*	
PLCDB(<PLC No.>,<DB/DX No.>,<Word No.>,<Qty. DW>,<DIM ident.>)=<Value>;	PLC data word, BCD	1...4	1...255/ 1000...1255	0...2043	1...3	100...109						*
		1...2	1...255	0...255	1...3	100...109					*	
PLCDG(<PLC No.>,<DB/DX No.>,<Word No.>)=<Value>;	PLC data word, floating point	1...4	1...255/ 1000...1255	0...2043	-	-						*
		1...2	1...255	0...255	-	-					*	

Alarms												
CL 800 statement	Function	Alarm No.					805	810	820	840	850	880
ALNZ () = <Alarm No.>;	Cycle alarms	4000...4299, 5000...5299 4000...4299, 5000...5209						*	*		*	*

System memory											
CL 800 statement	Function	Spindle No.	Gear stage		805	810	820	840	850	880	
SATC (<Spindle No.>,<Gear stage>)=<Value>;	Spindle acceleration time constant	0,1	1...8			*	*				

Explanation:

* Function implemented

6.9 Mathematical and logical functions

Value assignment, arithmetic operations							
CL 800 statement	Function	805	810	820	840	850	880
<Var>=<Value 1> + <Value 2>; <Var>=<Value 1> - <Value 2>; <Var>=<Value 1> * <Value 2>; <Var>=<Value 1> / <Value 2>; (chain calculation and parentheses also permitted)	Addition Subtraction Multiplication Division		*	*		*	*

Arithmetic functions							
CL 800 statement	Function	805	810	820	840	850	880
<Var>=ABS (<Value >); <Var>=SQRT(<Value >); <Var>=SQRTS (<Val. 1>,<Val. 2>);	Generate absolute value Square root Root of sum of squares		*	*		*	*

Arithmetic procedures							
CL 800 statement	Function	805	810	820	840	850	880
INC (<Var>); DEC (<Var>); TRUNC (<Var>);	Increment Decrement Integral part		*	*		*	*

Trigonometric functions							
CL 800 statement	Function	805	810	820	840	850	880
<Var>=SIN (<Value >); <Var>=COS (<Value >); <Var>=TAN (<Value >); <Var>=ARC SIN (<Value >); <Var>=ANGLE (<Val. 1>,<Val. 2>);	Sine Cosine Tangent Arc sine Angle between two vector components		*	*		*	*

Logarithmic functions							
CL 800 statement	Function	805	810	820	840	850	880
<Var>=LN (<Value >); <Var>=INV LN (<Value >);	Natural logarithm Exponential function		*	*		*	*

Explanation:

* Function implemented

Logical functions							
CL 800 statement	Function	805	810	820	840	850	880
PATTERN data type							
<Var>=<Var1> OR <Var>=<Var1> XOR <Var>=<Var1> AND <Var>=<Var1> NAND <Var>=NOT <Wert>;	<Value>; <Value>; <Value>; <Value>; <Value>;		*	*		*	*
	OR EXCLUSIVE-OR AND NAND NOT		*	*		*	*
BOOLEAN data type							
<Var>=<Var1> ORB <Var>=<Var1> XORB <Var>=<Var1> ANDB <Var>=<Var1> NANDB <Var>=NOTB <Value>;	<Value>; <Value>; <Value>; <Value>; <Value>;		*	*		*	*
	OR bit EXCLUSIVE-OR bit AND bit NAND bit NOT bit		*	*		*	*

Logical procedures							
CL 800 statement	Function	805	810	820	840	850	880
CLEAR BIT (<Var>.<Bit No.>); SET BIT (<Var>.<Bit No.>);	Clear bit Set bit		*	*		*	*
			*	*		*	*

6.10 NC-specific functions

Changing the program and machine reference points							
CL 800 statement	Function	805	810	820	840	850	880
POS MSYS;	Input of a position referred to the machine actual value system (effective block by block)		*	*		*	*

Explanation:

* Function implemented

Single functions							
CL 800 statement	Function	805	810	820	840	850	880
<Var1>=PREP REF (<Var2>);	Reference preparation Var1:output data from Var1 onwards Var2:input data from Var2 onwards		*	*		*	*
<Var1>=INT SEC (<Var2>,<Var3>);	Intersection computation Var1:output data from Var1 onwards Var2:first contour from Var2 onwards Var3:2nd contour from Var3 onwards ²⁾		*	*		*	*
<Var> = PREP CYC;	Prepare start for cycles Var:output data from Var onwards		*	*		*	*
STOP DEC;	Stop decoding until buf. mem.empty		*	*		*	*
STOP DEC1;	Stop decoding until buffer memory is empty at coordinate rotation		*	*		*	*

Measurement functions							
CL 800 statement	Function	805	810	820	840	850	880
<Var>=MEAS M (<Value>);	Flying measurement actual value relative to machine data Var: data stored from R parameter onwards Value: No. of measuring input (effective block-by-block for all programmed axes)		*	*		*	*

Program influence							
CL 800 statement	Function	805	810	820	840	850	880
REL (<Value1>,<Value2>);	Read-in inhibit via external input Value1 : byte address 1 or 2 Value2 : bit address 0..7		*	*			
INTA (<Value1>,<Value2>,<Value3>);	Axis-specific clear remaining path via external input Value1 : axis number 0..4 Value2 : byte address 1 or 2 Value1 : bit address +/- 0..7		*	*			

Explanation:

- * Function implemented
- 2) at the present time not possible

6.11 I/O statements

NC I/O functions											
CL 800 statement	Function	Range	Menu No.			805	810	820	840	850	880
WRT PIC (<Range>,<Menu No.>);	Menu selection from NC program	0/1	1...254				*	*			
RECALL PIC	Return jump to initial menu	-	-				*	*			

I/O functions, general											
CL 800 statement	Function	No. V24	Ch. No.	Data type		805	810	820	840	850	880
PORT (<No. V24>);	Select V24 interface	1/2	-	-			*	*			
OUTP ZOA (<Channel No.>);	Output zero offsets via V24	-	0...3	-			*	*			
OUTP DATA <Data type>,<Start addr.>,<End addr.>;	Output data via V24	-	-	1...9			*	*			
OUTP PARA (<Channel No.>,<Start addr.>,<End addr.>;	Parameter output via V24	-	0...3	-			*	*			
INP (<Data type>);	Read-in data via V24	-	-	0...10			*	*			
OUTP ETX;	Output ETX via V24	-	-	-			*	*			

Operator control functions											
CL 800 statement	Function	No. V24	Ch. No.	Data type		805	810	820	840	850	880
CHAN = <Channel No.>;	Select channel No. for screen display	-	1...3				*	*			

Explanation:

* Function implemented

Section 7

-Examples-

Overview:

- 7.1 Program structure overview
- 7.2 Program nesting for IF THEN ELSE branching
- 7.3 Program example: Hole pattern
- 7.4 Program example: Deep-hole drilling cycle

7 Examples

7.1 Program structure overview

- CL 800 programming

(** Program header ****)**

```
PROGRAM 150;  
CHANNEL NC 0;  
ID (0-EXAMPLE-25.02.86-KUB-SIEMENS-850-3);
```

(** Declaration part ****)**

```
CENTRAL: "VARCEN"; (* Reference to variables which are *)  
GLOBAL: "LISTE1"; (* declared in separate lists *)
```

```
PAR REAL:R0:=Name, R10:=X_value; (* Variables which are declared in  
the program *)
```

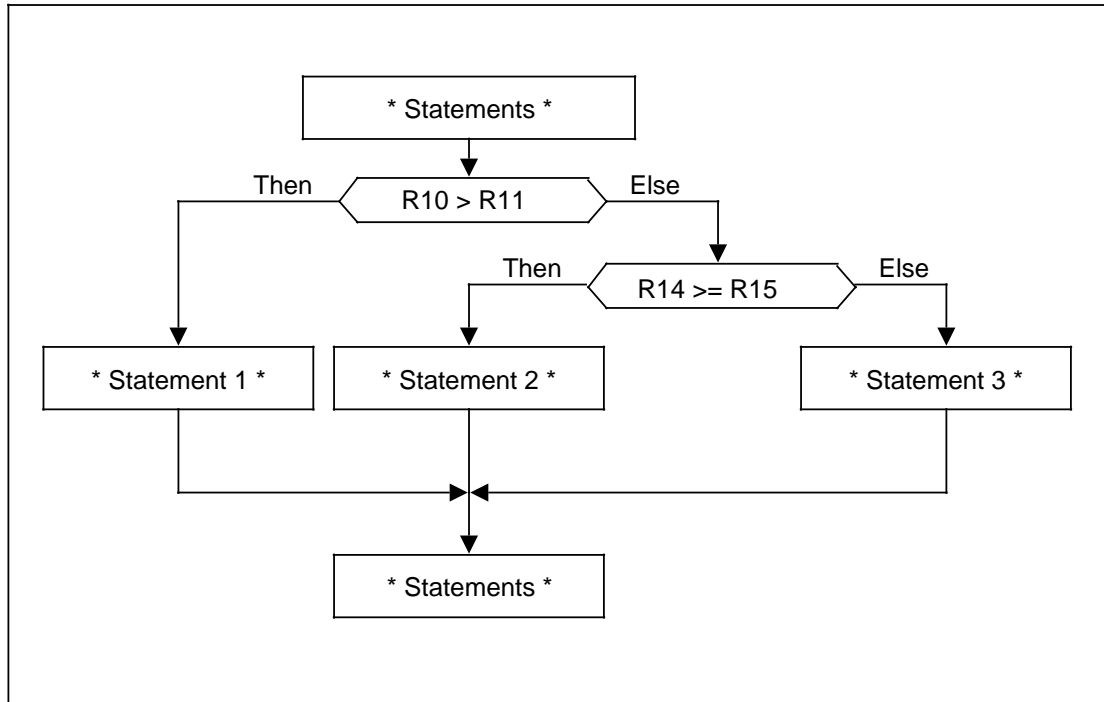
```
LOCAL REAL:R60:=z_value, R61:=y_value;
```

(** Statement part ****)**

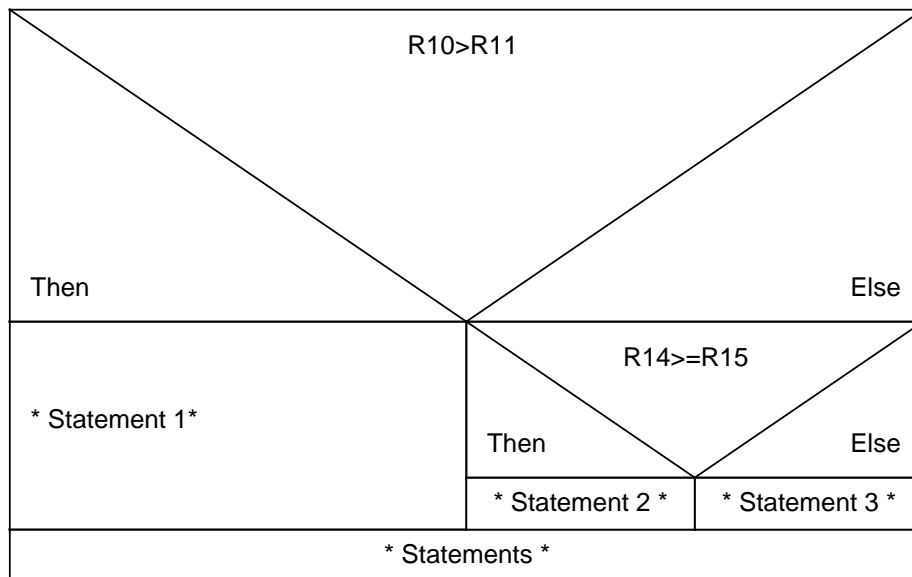
```
BEGIN  
  y_value= X_value+ Name * 2  
  z_value= SIN (y_value) + Name;  
  :  
  (:NOTE:); (* DIN programming *)  
  :  
  (* Further statements in the statement part *)  
  :  
END.
```

7.2 Program nesting for IF THEN ELSE branching

- Flow diagram



- Structogram



- **CL 800-Programming**

(** Program header ****)**

```
PROGRAM 30;
CHANNEL NC 1;
ID (0-EXAMPLE25-...);
PW (NEST);
```

(** Declaration part ****)**

```
PAR REAL: R10, R11, R14, R15;
```

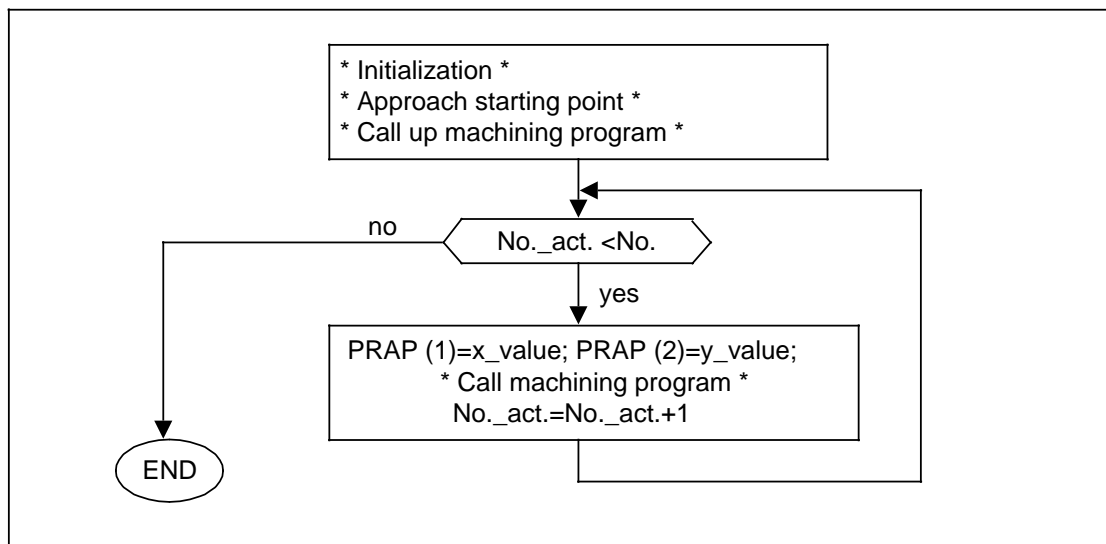
```
:
```

(** Statement part ****)**

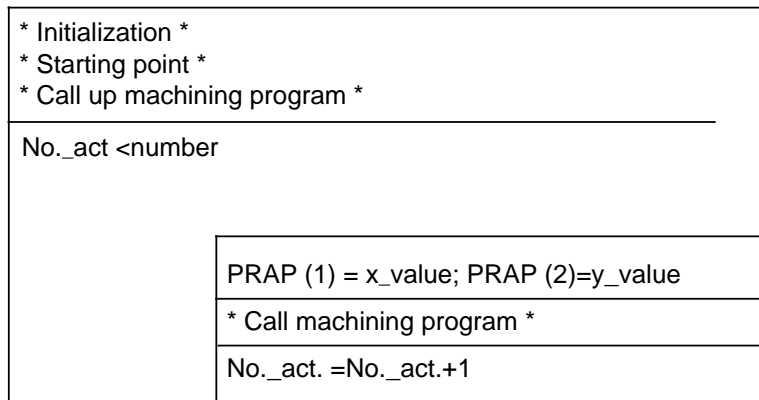
```
BEGIN
:
IF R10 > R11 THEN
  BEGIN
    < Statements >;
  END;
ELSE
  IF R14 >= 15 THEN
    BEGIN
      < Statements >;
    END;
  ELSE
    :
  ENDIF;
ENDIF;
:
END.
```

7.3 Program example: Hole pattern

- **Flow diagram**



- **Structogram**



- **CL 800-Programming**

(** Program header ****)**

```
PROGRAM 101;
CHANNEL NC 4;
ID (0-HOLE PATTERN-25.02.86-HFH-PJP: PC16/11);
```

(** Declaration part ****)**

```
PAR REAL:      R20:=subroutine_No.,R21:=start_X,R22:=start_Y,
               R30:=distance,R31:=alpha;
PAR INTEGER:   R32:=number;

LOCAL REAL:    R60:=x_value,R61:=y_value,R62:=beta;
LOCAL INTEGER: R70:=No._act.;
```

(** Statement part ****)**

```
BEGIN
```

```
No._act.=1                                     (* Initialization *)
y_value=SIN (Alpha) * distance;
x_value=SIN (90 - Alpha) * distance; LF;
(:G90 G00:); PRAP (1)=start_X; PRAP (2)=start_Y; LF;      (* Approach starting point *)
(:G01 F200 L=R20 P1:); LF;                               (* Call machining program *)
```

```
WHILE No._act. < number DO
```

```
  BEGIN
```

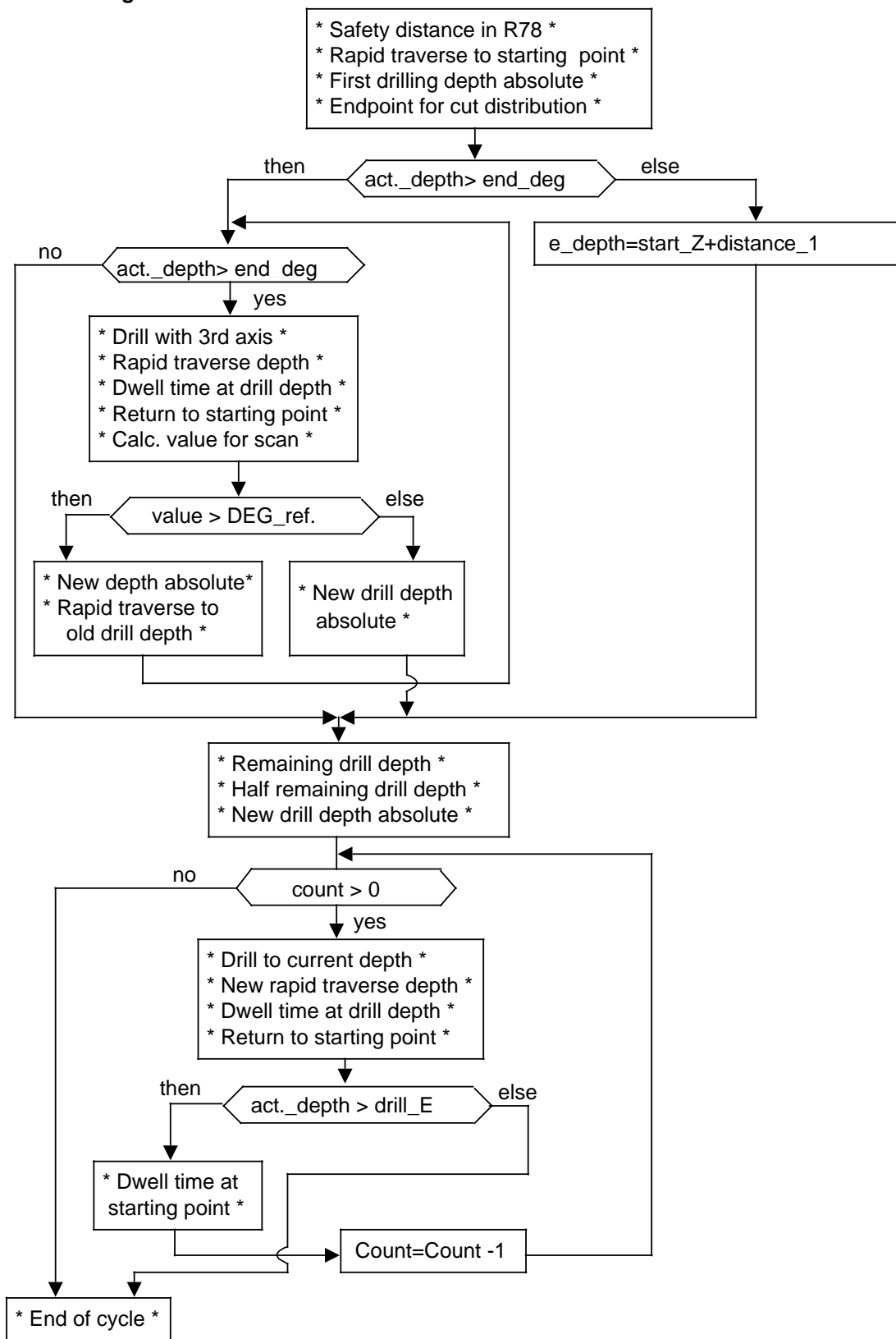
```
    LF; (:G91 G00:); PRAP (1)=x_value; PRAP (2)=y_value; LF;
    (:G01 F200 L=R20 P1:); LF;                               (* Call machining program *)
    No._act. =No._act.+1
```

```
  END;
```

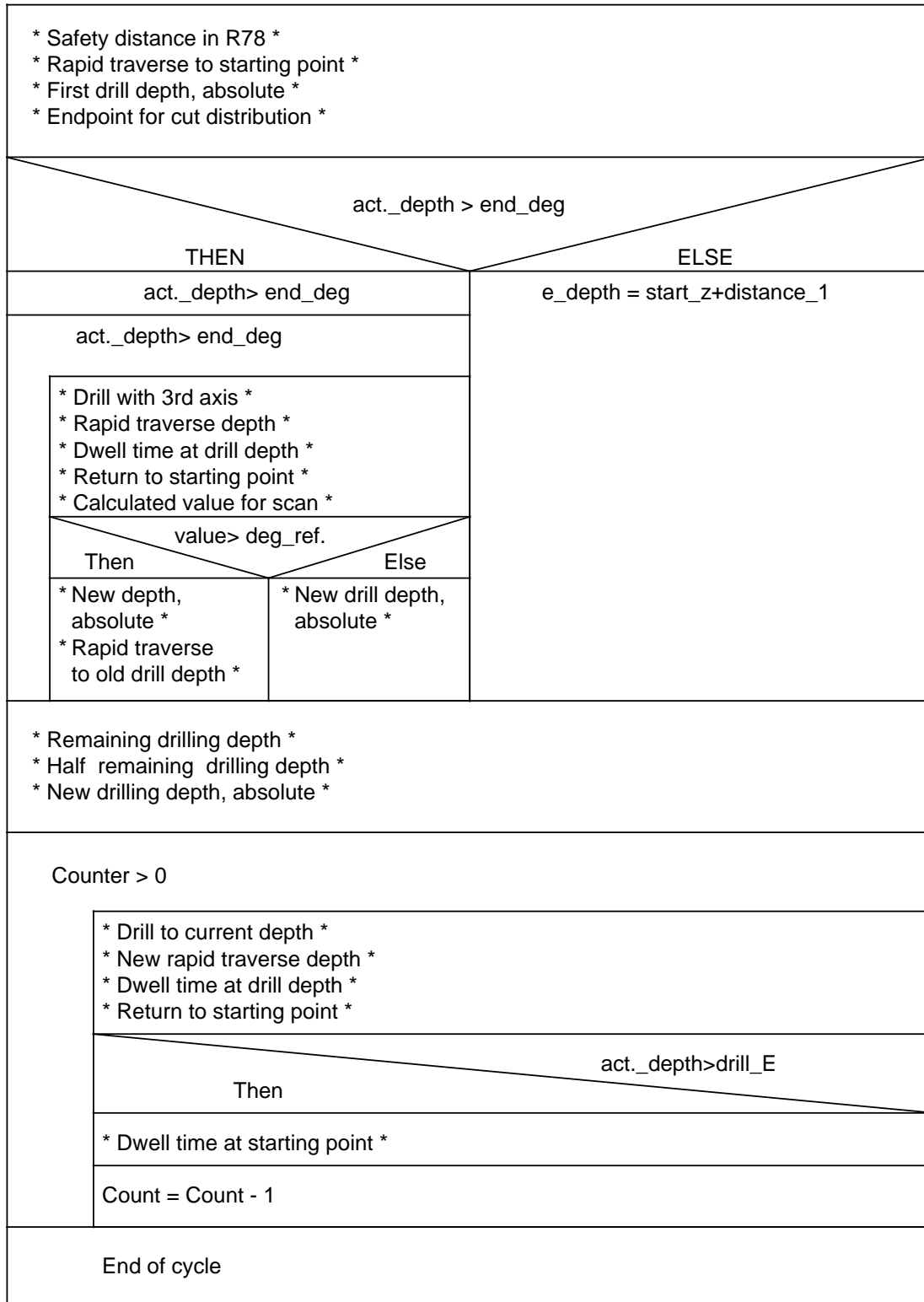
```
END.
```

7.4 Program example: Deep-hole drilling cycle

• Flow diagram



• **Structogram**



- **CL 800-Programming**

(** Program header ****)**

```
PROGRAM 81;
CHANNEL NC 2;
ID (0-TDRILL-25.02.86-HFH-CYCLE/B/850/);
PW (GENERAL);
```

(** Declaration part ****)**

```
PAR REAL:          R22:=start_Z,R24:=deg_ref, R25:=drill_1,R26:=drill_E,
                  R27:=dwell_A, R28:=dwell_B;
LOCAL REAL:       R62:=act._depth, R63:=end_deg, R64:=value, R77,
                  R78:=dist._1, R66:=remain., R68:=e_depth;
LOCAL INTEGER:    R67, R93:=counter;
```

(** Statement part ****)**

```
BEGIN
```

```
counter= 2; LF;
R77=PREP CYC; LF; (*Safety distance in R78*)
(:G0 G64 G90:); PRAP (3)=start_Z; LF; (*Rapid traverse to start point*)
act._depth=start_Z - drill_1; (*First drilling depth. absolute*)
end_deg= drill_E + Deg_ref; (*Endpoint for cut distribution*)
value= drill_1; LF;
```

```
IF act._depth> end_deg THEN (*Scan whether cut distribution*)
```

```
  WHILE act._depth> end_deg DO
```

```
    BEGIN
```

```
      LF; (:G1:); PRAP (3) = act._depth; LF; (*Drill with 3rd axis*)
      e_depth=act._depth+dist._1; LF; (*Rapid traverse depth*)
      (:G4 F=R27:); LF; (*Dwell time at drilling depth*)
      (:G0:); PRAP (3)=start_Z; LF; (Return to starting point*)
      (:G4 F=R28:); LF (*Dwell time at starting point*)
      value=value- deg_ref.; (*Computed value for scan*)
```

```
      IF value> deg_ref.THEN
```

```
        BEGIN
```

```
          LF; act._depth=act._depth-value; LF; (*New depth, absolute*)
```

```
          (:G0:); PRAP (3)=e_depth; LF; (*Rapid traverse to old drilling depth*)
```

```
        END;
```

```
      ELSE act._depth=act._depth- deg_ref; (*New drilling depth, absolute*)
```

```
      ENDIF;
```

```
    END;
```

```
  ELSE e_depth=start_Z+dist._1; (*Intermediate calculation*)
```

```
ENDIF;
```

```
remain.=e_depth - dist._1;remain.=remain.- drill_E; (*Remaining drilling depth*)
```

```
remain.=remain./ 2; (*Half-remaining drilling depth*)
```

```
act._depth= drill_E +remain.; (*New drilling depth, absolute*)
```

7.4 Program example: Deep-hole drilling cycle

```
WHILE counter> 0 DO                                     (*Drilling loop*)
  BEGIN
    LF; (:G1:); PRAP (3) = act._depth; LF;               (*Drill to current depth*)
    e_depth=act._depth+ 1; LF;                          (*New rapid traverse depth*)
    (:G4 F=R27:); LF;                                   (*Dwell time at drilling depth*)
    (:G0:); PRAP (3)=start_Z; LF;                       (*Return to start point*)
    IF act._depth>drill_E THEN                          (*Last drilling depth reached?*)
      BEGIN
        LF; (:G4 F=R28:); LF;                          (*Dwell time at the start point*)
        act._depth=drill_E;
      END;
    ENDIF
    count=count - 1;
  END;
END.                                                     (*End of cycle*)
```


Section 8

-Using the interactive editor-

Overview:

8.1 General

8.2 Interactive editor operating modes

8.2.1 Functions in the display mode

8.2.2 Functions in the command mode

8 Using the interactive editor

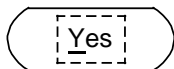
8.1 General

After the editor has been called using the "interactive operation" menu, the editor and the source file are read into the main memory.

The following window is displayed if the source file still does not exist.

Note:	
CL 800.ZPQ: File not available. Setup again? <input type="radio"/> Yes <input type="radio"/> No	

The new source file (e.g. CL 800.ZPQ) is stored, if the pad



is selected.

Finally, the first screen page and the status line of the editor are output, after which the editor waits for an input.

A terminated line is immediately analyzed. The cursor is positioned at the next line if it was error-free, otherwise it remains at the line analyzed as erroneous, at that position where an error was found. An error message is issued. The error can be corrected immediately and processing continued.

For very large files (more than 700 lines) it may be necessary to store data in temporary files in order to create space in the main memory. With this type of data management, the word WAIT always appears in inverse video near the actual cursor position and the cursor itself is switched-off. No further input should be made until the data management is completed (whereby the word WAIT disappears and the cursor reappears), in order to be able to respond to possible errors.

Data management normally lasts 1-3 seconds; however it can be significantly longer for lengthy operations, such as long jumps in the file. The editor does not accept any inputs during this time.

8.2 Interactive editor operating modes

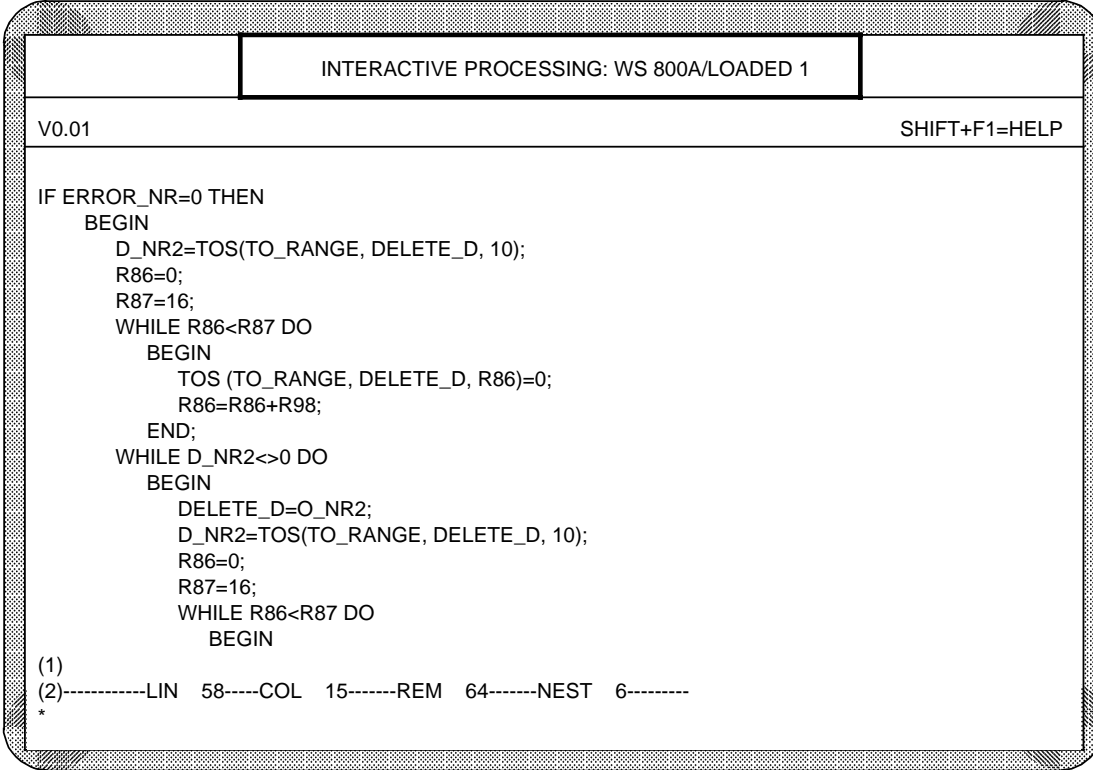
The editor has two possible modes:

- **Display mode**
- **Command mode**

In the **display mode**, the selected file is edited. Data can be created, deleted and modified in this mode. The cursor can be positioned and text inserted. The line is analyzed when terminated with the RETURN key. In the event of an error, the cursor is positioned where the error was found.

A blank line (1) and status line (2) are located under the data where the current file position is displayed .

Example:



```
INTERACTIVE PROCESSING: WS 800A/LOADED 1
V0.01                                     SHIFT+F1=HELP
IF ERROR_NR=0 THEN
  BEGIN
    D_NR2=TOS(TO_RANGE, DELETE_D, 10);
    R86=0;
    R87=16;
    WHILE R86<R87 DO
      BEGIN
        TOS(TO_RANGE, DELETE_D, R86)=0;
        R86=R86+R98;
      END;
    WHILE D_NR2<>0 DO
      BEGIN
        DELETE_D=O_NR2;
        D_NR2=TOS(TO_RANGE, DELETE_D, 10);
        R86=0;
        R87=16;
        WHILE R86<R87 DO
          BEGIN
(1)
(2)-----LIN 58-----COL 15-----REM 64-----NEST 6-----
*
```

In the **command mode** the file can be viewed (scrolled) without analysis. Editor commands can be input. The so-called command prompt "*" identifies when the command mode is ready for input.

```

INTERACTIVE PROCESSING: WS 800A/LOADED 1
V0.01                                SHIFT+F1=HELP
BEGIN
  D_NR2=TOS(TO_RANGE, DELETE_D, 10);
  R86=0;
  R87=16;
  WHILE R86<R87 DO
    BEGIN
      TOS (TO_RANGE, DELETE_D, R86)=0;
      R86=R86+R98;
    END;
  WHILE D_NR2<>0 DO
    BEGIN
      DELETE_D=O_NR2;
      D_NR2=TOS(TO_RANGE, DELETE_D, 10);
      R86=0;
      R87=16;
      WHILE R86<R87 DO
        BEGIN
          *
-----LIN 58-----COL 15-----REM 64-----NEST 6-----

```

Mode change:



The display mode and command mode are selected with the **F1** function key. This key is located in the function keypad. However, as no analysis is realized in the command mode, the current file position is again selected which was available when the display mode was exited, when the display mode is again selected with the function key **F1**. If scrolling had taken place, then this position will be generally different than the pseudo cursor position. The pseudo cursor always marks the current position in the file, and consists of the inverse-video representation of the current character, whose position in the file is to be displayed in the status line. The **JP** command must be input (jump to pseudo cursor), if a jump is to be made explicitly to this position (which may require program re-analysis).

Help mask:

Contrary to the WS 800 configuring station, there is no second second screen available for a help mask. Thus, in the editor mode, a help screen can be called at any time.

The editor mode is briefly interrupted using the key sequence



and a help mask is inserted with the following commands.

F1	Mode change	SHIFT+F1	Help information
F2	Insertion on/off	SHIFT+F2	Cut line
F3	Save data	SHIFT+F3	-
F4	Last command active	SHIFT+F4	Copy line
F5	Undo	SHIFT+F5	Delete line from cursor onwards
F6	-	SHIFT+F6	Paste line
F7	Jump left by one word	SHIFT+F7	Jump to line start
F8	Jump right by one word	SHIFT+F8	Jump to end of line
F9	-	SHIFT+F9	Delete line
F10	-	SHIFT+F10	-

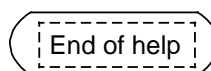
1. Help mask

When the pad is selected, then a second help mask is provided with commands which can only be used in the command mode.

Cycle editor commands	
EQ	End without storage
EX	End with storage
US	Save without end
LD	Load file
JP	Jump to pseudo cursor position
LOC n	Jump to line n
I (+/-/n)	Automatic indentation (On/off indentation n)

2. Help mask

The display mode is again selected after terminating with



8.2.1 Functions in the display mode

In addition to entering alphanumeric characters, a whole series of functions can be executed in the display mode. If not otherwise specified, the functions are permitted in the display mode and in the command mode, however their effect can be different.

In the following, <CR> always means the carriage return + line feed, generated by actuating the **RETURN** key.

Cursor control functions:



Shift cursor one line up. The column position in the new line is governed by the line length and the current column position (the cursor can never be positioned to the right of the end of line). Only a limited number of lines can be rapidly be gone back (at the present time: 9 lines), as the analysis must be reset for this function. If this number is exceeded, then the program must be re-analyzed from the start of file. In order to prevent this happening erroneously, the operator is requested to confirm the return jump.


Return further with restart of analysis? (Y/N)

When "Y" is input, the program is re-analyzed from the start up to the jump destination; the processing is indicated in the last line with the line number of the just processed line. This key is only permissible in the display mode.




Shift cursor one line down. The column position in the new line is governed by the line length and the current column position (the cursor can never be positioned to the right of the end of line). It is only positioned, if the current line has been terminated with <CR>. The current line is analyzed, and in the event of an error, the cursor is re-positioned on the current line. This key is only permissible in the display mode.



Shift cursor one column to the right. At the end of a line, terminated with <CR>, this key acts like the  key, the cursor is however positioned at the beginning of the new line.



Shift cursor one column to the left. At the beginning of line, this key acts like the  key, however the cursor is positioned at the end of the previous line.



RETURN key

With the insertion mode cancelled, the cursor is positioned at the first character of the next line. If the current line was not terminated with <CR>, the termination is executed. The entire line is analyzed.

With the insertion mode selected, a carriage return +line feed is inserted at the cursor position, the remainder of the is line written into a newly inserted line and the cursor positioned at the first character of the new line. The terminated line is analyzed.

In the event of an error, the cursor is re-positioned to the column where the error was identified.

When <CR> is inserted at the end or start of a line, a blank line is inserted before or after the current line, whereby the cursor is either positioned at the beginning of the inserted blank line (when inserting at the end of line) or at the first character of the current line (when inserting at the start of the line).



Jump to line start

Position cursor at the first character in the current line.



Jump to end of line

Position cursor at the end of the current line.



Jump left by one word

Position cursor at the beginning of word to the left.

A word is defined as a character sequence, which is limited by the character: space , <>, ; . () [] ^ ' * + - / \$. This command is valid extending beyond the end of line.



Jump right by one word

Position the cursor at the beginning of the word to the right.
This command also positions the cursor to the next line.

Delete functions:**BACKSPACE key**

Delete character to the left of the cursor.
At the start of a line this key acts like



Thus, <CR> at the end of the previous line is **not** deleted.

**SHIFT + BACKSPACE key**

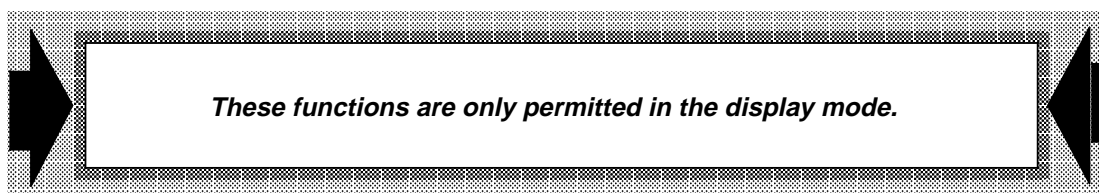
Delete current character
The <CR> at the end of the line can also be deleted with this function. The next line, if present, is then attached to the current line. If the line is too long so that it cannot be completely attached (the line length is limited to one display line), then it is split. The line section which cannot be attached remains on the next line.

**Delete line from cursor position**

Delete current line from and including the cursor position to the end of line.

**Delete line**

Delete the complete current line.
Any existing lines which follow lines are scrolled up. The cursor is then positioned at the first character of the next scrolled line.

Other functions:**Insertion on/off**

Insertion mode switched-on/off
With the insertion mode switched-on, the **INSERT** word appears in the status line in inverse video. The default setting is with the insertion mode **switched-off**.

F3

Save file

Save data

The file is saved, and the file start is reloaded. The command mode is active, and the file start appears on the screen after this function is selected. The pseudo cursor is located at the first character. With function key **F1** the file position is reselected at which data save took place. A jump to the the pseudo cursor position is only possible with the **JP** command. This function allows a particularly fast return jump to the beginning of file from a position located near the end of file.

↑ SHIFT

F2

Cut line

Transfer current line to an internal memory.

This function is required in order to shift the current line to another file position.

The visible effect is the same as

↑ SHIFT

F9

(delete complete current line).

↑ SHIFT

F4

Copy line

Store current line. Contrary to

↑ SHIFT

F2

this line remains. This function is required to copy the current line to another file position. There is no visible effect.

↑ SHIFT

F6

Paste line

Insert stored line at the cursor position. The line, stored with

↑ SHIFT

F2

or

↑ SHIFT

F4

is inser-

ted before the current line (in this version, still independent of the cursor position in the current line). If a line was not stored

with **↑ SHIFT**

F2

or

↑ SHIFT

F1

then a blank line is inserted. The cursor is then located at the first character of the inserted line.

F5**Undo**

Cancel line change.

As long as the current line has only been edited, all changes since the line has been selected can be cancelled with this function. (If the line was exited, then this of course is no longer possible). The cursor is then positioned at the first character of the current line.

Error messages appear in the display mode for :

- illegal entries,
- entries, which would exceed the maximum line length,
- errors occurring during memory update
- error messages from the analysis function

Example:

```

INTERACTIVE PROCESSING: WS 800A/LOADED 1
V0.01                                SHIFT+F1=HELP
IF FEHL_NR=0 THEN
  BEGIN
    D_NR2=TOS(TO_RANGE, DELETE_D, 10);
    R86=0;
    R87=16;
    WHILE R86<R87 DO
      BEGIN
        TOS(TO_RANGE, DELETE_D, R86)=0;
        R86=R86+R98;
      END;
    WHILE D_NR2<>0 DO
      BEGIN
        DELETE_D=D_NR2;
        D_NR2=TOS(TO_RANGE, DELETE_D, 10);
        R86=0;
        R87=16;
        WHILE R86<R87 DO
          BEGIN
            -----LIN 58-----COL 15-----REM 64-----NEST 6-----
            >> E206 Function not implemented <<

```

8.2.2 Functions in the command mode

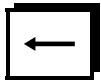
Essentially, editing is allowed in the command mode the same as in the display mode. However, this is with the restriction that the cursor can only be moved along the current command line. In this case, the maximum permissible command line length should be noted. Otherwise, deletion, insertion etc. are permitted.

If not otherwise stipulated, the following functions are permitted in both the command and display mode.

Cursor control functions:



Shift cursor one column to the right, but not further than the column following the last character of the currently displayed command.

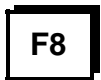


Shift cursor one column to the left, but not further than the start of the command.



Jump one word to the left

Shift cursor to the start of the command.



Jump one word to the right

Shift cursor after the last character of the currently displayed command.



RETURN key

Execute displayed command. The function is **independent** of the current cursor position in the command line.

The screen display is scrolled up by one line, and the cursor is positioned at the start of the last display line. When the command was correctly executed, only the command prompt "*" appears (if the command does not result in a changeover to the display mode). If an error occurs, the appropriate error message appears and the cursor is re-positioned to the beginning of the executed command. A blank command input is permitted.



Page back by one screen page. The paging function permits the file to be viewed without analysis. Paging is such that the uppermost data line becomes the lowest data line after paging. Thus, there is no loss of cohesion between two screen pages. The pseudocursor marks the current file position specified in the status line. The position of the real cursor remains unchanged. This function is only permitted in the command mode as it is executed without analysis.



Page forwards by one page. Paging is such that the bottom data line at the start becomes the uppermost data line after paging. Thus, there is no loss of cohesion between the two pages. The pseudocursor marks the current file position specified in the status line. The position of the real cursor remains unchanged. This command is only permitted in the command mode as it is executed without analysis.



HOME key

Position the pseudocursor at the beginning of the file.



END key

Position the pseudocursor at the end of file.

Delete functions:



BACKSPACE key

Delete the character to the left of the cursor, but no further than the beginning of command.



SHIFT + BACKSPACE key

Delete current character.



Delete line from cursor position

Delete command from and including the cursor position and up to the end of the command.



Delete line

Delete entire command.

Other functions:

F4

Activate last command

The last executed command is activated.
The last executed command is displayed in the command line from the cursor position onwards. If the insertion mode is selected, an insertion is made after the cursor position. Display or insertion is **not** realized if this means that the maximum length of the command line is exceeded (an error message appears). The cursor is then positioned after the command displayed or inserted. It should be noted that the command is first executed after depressing the RETURN key. "Activation" in this case, is only the display! This function is only permitted in the command mode.

F2

Insertion on/off

Insertion mode on/off.
When the insertion mode is selected, the word **INSERT** appears in the status line, in inverse video. The default setting is with the insertion mode **switched-out**.

Cycle editor commands:

EQ

Exit with Quit

Editor terminated without storage.
The user should again confirm this entry. The screen is cleared and the editor terminated (the header lines are **not** deleted).

EX

EXit

Edited file stored and editor terminated.
The screen is cleared after storage (the header lines are **not** deleted).

US

Update and Save

Save edited file.
In this case, the editor is **not** terminated. After successful saving, the beginning of the file is displayed on the screen. The pseudocursor is positioned at the first character.
This command provides the fastest possibility of jumping from end of file to the start of the file in a large file.

LD Load file

The **LD** function permits a file to be reloaded, without the editor having to be exited.

"Load file" sequence:

- Input **LD** in the command line.
The question appears as to whether the old file should be first saved or not.
- Input **Y** or **N** in order to acknowledge save.
The request to input the complete file name appears in the command line (path designation and extension).
- The required file is loaded after the complete file name is entered. The editor automatically returns to the screen mode.

An error message is displayed if the file does not exist. The editor however remains in the command mode.

JP Jump to pseudo cursor

Jump to the pseudo cursor position.

This commands causes the display mode to be selected. If the pseudo cursor position is different from the current position before the transition into the command mode, the program must be re-analyzed up to the jump destination. The completion of this re-analysis is signaled with the line number of the currently analyzed line.

Loc n Position pseudo cursor to specified line.

I (+/-/n) Automatic indent

The **automatic indent** function only functions in the **interactive mode**. This allows the program structure, i.e. the current nesting level in the program, to be made visible. Indentation by one step is made for each new level. This allows nested check structures to be clearly represented. The automatic indent only functions correctly when new blocks are also written into a new line, as the analysis is activated linewise (e.g. a BEGIN or END must always be written in a new line).

Activation:

Input: In (with n=0.....8)

The function is activated using the command I. The number of blanks, which are automatically generated at each indent level at the start of line is specified with the digit n. Standard indentation is with three blanks when **I+** is input. This can be disabled by **I-**. The current set number is displayed in the editor status line.

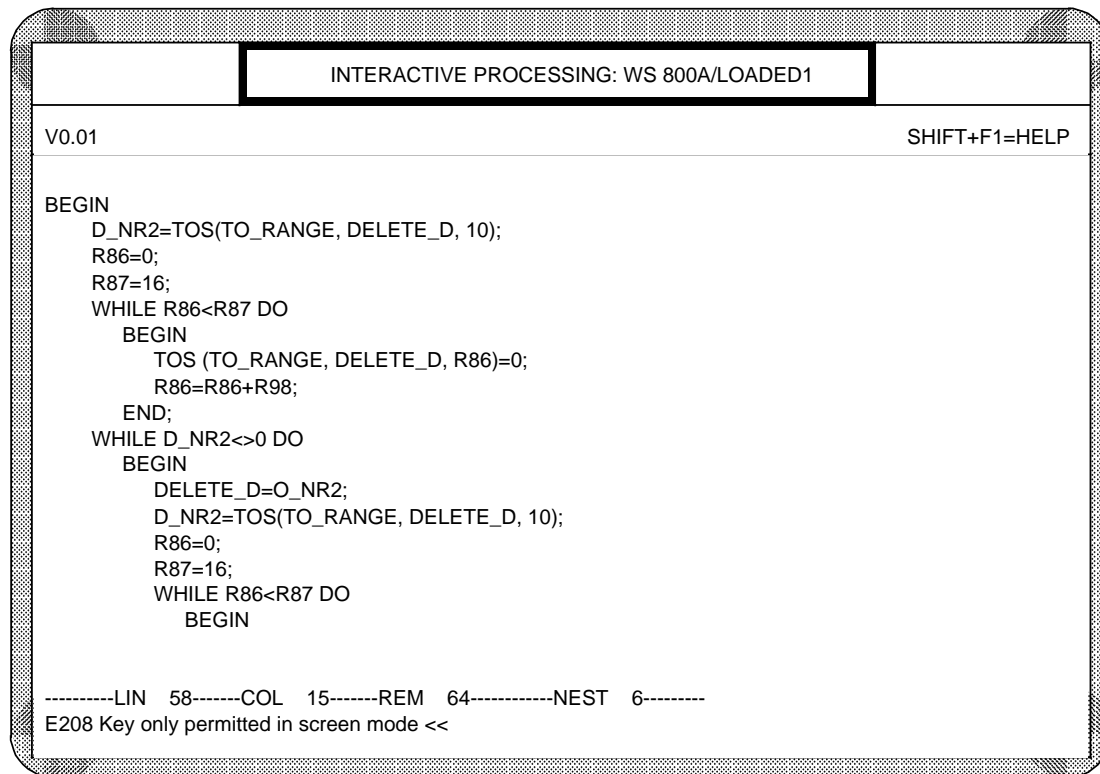
Note:

The function is switched-out when the editor is called. Automatic indent is executed from that instant onwards where the function is activated, i.e. indentation is not realized backwards from the start of the file.

Error messages appear in the command mode for:

- illegal inputs,
- inputs, where the maximum length of a command would be exceeded,
- when errors occur during memory update,
- when erroneous or non-interpretable commands are input.

Example:



```
INTERACTIVE PROCESSING: WS 800A/LOADED1
V0.01                                SHIFT+F1=HELP
BEGIN
  D_NR2=TOS(TO_RANGE, DELETE_D, 10);
  R86=0;
  R87=16;
  WHILE R86<R87 DO
    BEGIN
      TOS (TO_RANGE, DELETE_D, R86)=0;
      R86=R86+R98;
    END;
  WHILE D_NR2<>0 DO
    BEGIN
      DELETE_D=O_NR2;
      D_NR2=TOS(TO_RANGE, DELETE_D, 10);
      R86=0;
      R87=16;
      WHILE R86<R87 DO
        BEGIN
          -----LIN 58-----COL 15-----REM 64-----NEST 6-----
          E208 Key only permitted in screen mode <<
```

Section 9

-Cycle editor error messages-

Overview:

9.1 **General**

9.2 **Error message list**

9.2.1 Warning messages

9.2.2 User error messages

9.2.3 System error messages

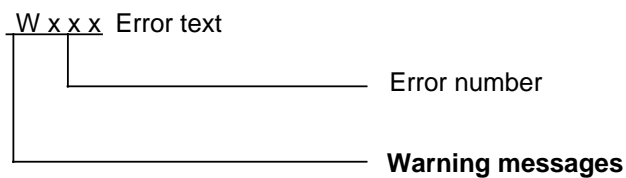
9 Cycle editor error messages

9.1 General

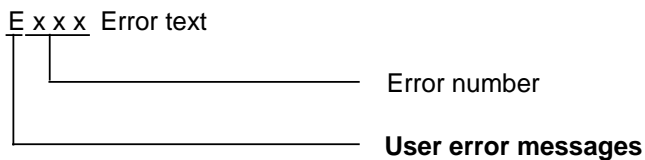
The cycle editor differentiates between three message groups:

- **Warning messages**
Cycle editor messages, which are displayed as warnings, should not be evaluated as faults/errors, but as cautionary notes to the user.
- **User error messages**
If the user makes an illegal input of a cycle program or an external data file then this is displayed as user error by the cycle editor.
- **System error messages**
If a system error occurs during cycle processing, then this is displayed by the cycle editor. With only a few exceptions, system errors cannot be rectified by the user.

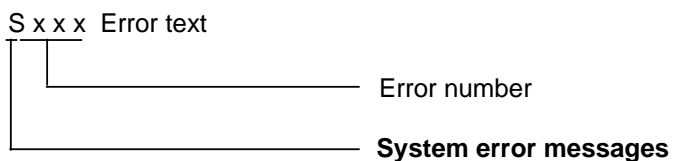
In the interactive editor, the messages in the message line are displayed as follows on the screen.



e.g. W001 warning: Comment text open!



e.g. E037, variable name not declared



e.g. S062, incorrect constant in the symbol list

All the possible cycle editor messages are listed on the subsequent pages. Most of them are self-explanatory, however several require more detailed explanation.

9.2 Error message list

9.2.1 Warning messages

Message No.	Error text	Notes
W001	Warning: Comment text open!	In the previous lines a comment text was opened with ' (* ', which was not closed up to the current line.
W002	Warning: DIN code open!	In the previous lines a DIN code was opened with ' (: ' which was not closed up to the current line.
W010	File was not compiled!	An object code is not generated if the edited file has an error, or it is not terminated with "END".

9.2.2 User error messages

Message No.	Error text
E001	"," expected
E002	." expected
E003	":=" expected
E004	"=" expected
E005	"(" expected
E006	")" expected
E007	"[" expected
E008	"]" expected
E009	"," expected
E010	"." expected
E011	"PROGRAM" expected
E012	"CHANNEL" expected
E013	"NC" or "COM" expected
E014	"EXTERNAL" expected
E015	"CENTRAL" expected
E016	"GLOBAL" expected
E017	"OTHERWISE" or "ENDCASE" expected
E019	"ELSE" or "ENDIF" expected
E020	"M" expected
E021	"SEC" expected
E022	"MSYS" expected

Message No.	Error text
E023	"DO" expected
E024	"UNTIL" expected
E025	"WITH" expected
E026	"FOR" expected
E027	Constant expected
E028	String expected
E029	File name expected
E030	Comparison operator expected
E031	Blank or other separator expected
E032	Ident. designation incorrect
E033	Password incorrect
E034	File name not correct
E035	Only one file name possible per line
E036	Variable name double
E037	Variable name not declared
E038	Variable name exceeds 25 characters
E039	Variable names in program header are illegal
E040	Number at the start of a variable name illegal
E041	Parameter not declared
E042	Parameter doubled
E043	Parameter in program header illegal
E044	R parameter with bit in declaration illegal
E045	% in string illegal
E046	String exceeds 120 characters
E047	Bit information not permissible
E048	Real numbers in program header illegal
E049	Real number outside limits
E050	Integer number outside limits
E051	Bit pattern outside limits
E052	Maximum of 5 digits before the decimal point permitted
E053	Character illegal
E054	Symbol cannot be used
E055	End of comment text without start
E056	DIN code end without start

Message No.	Error text
E057	DIN code end:) must be located in one line
E058	Maximum block level exceeded
E059	Maximum indent level exceeded
E060	Program was not correctly terminated
E061	Syntax error
E100	Label used twice
E200	File not available

The following Exxx error messages are caused by incorrect editor operation.

Message No.	Error text	Notes
E201	Command unknown	
E202	Command too long	A command must have a maximum of 75 characters
E203	Incorrect command syntax	
E204	Function not implemented	
E205	Character illegal	
E206	Key only permitted in the display mode	
E207	Key only permitted in the command mode	
E208	Line length maximum one screen line	A screen line is maximum of 79 characters long (not including the line terminating character)
E209	Search term not found	
E300	File cannot be compiled	This message appears on the screen if a cycle, which is error-free, is compiled through the pull-down menu "special functions", but is not terminated with 'END. <CR>.

9.2.3 System error messages

System errors which cannot be rectified

Message No.	Error text	Notes
S001	System error, function identification unknown	
S002	System error editor mode unknown	
S003	System error at initialization	
S004	System error queue cannot be read	
S005	System error queue cannot be written	
S006	System error at set wait flag	
S007	System error at system text access	
S008	System error: too little space in main memory	
S009	Main memory full	The editor cannot be started if the main memory is full.
S010	Floppy disk full	The cycle editor has no possibility of configuring the memory. The cycle editor must be exited in order to create space on the floppy disk.
S011	Temporary memory defective	This message indicates a serious non-removable error in the editor. Under certain circumstances, the CMD file is destroyed. A remedy is to reload the complete cycle editor program.
S012	File access error	File access errors always indicate a significant error in the file maintenance system of the operating system and cannot be rectified by the user.
S013	Bdos error	Bdos errors are operating system errors The error type is indicated by an error identifier (this is important for error diagnostics !).
S020	Limiter list not available	
S021	Internal file not terminated with EOF	
S022	Analysis exceeds > 64 Kbyte	

Message No.	Error text	Notes
S025	System error when writing in string	
S026	Compiler flag faulty	
S028	Compiler error: Stack empty	
S029	Compiler error: Stack full	
S030	CL 800 source file cannot be opened	
S031	CL 800 destination file cannot be opened	
S032	CL 800 destination file cannot be closed	
S033	Read error in CL 800 source file	
S034	Search key missing from CL800 source file	
S035	Write error in CL 800 target file	
S040	Internal file cannot be opened	
S041	Internal file cannot be closed	
S042	Unexpected end of internal file (TAKE)	
S043	Unexpected end of internal file (SEARCH)	
S044	Incorrect numeric representation in internal file	
S045	Temp. target file cannot be opened	
S046	Temp. target file cannot be closed	
S050	Illegal label for GOTO statement	
S051	"ENDEXTERN" not found	
S052	Expected system location assignment missing	
S060	Too many constants in the constants file	
S061	Incorrect constant in the constants file	

Message No.	Error text	Notes
S062	Incorrect constant in the symbol list	
S063	Unknown group assignment (procedure No. 5)	
S064	Excessive block length of destination code	
S065	Memory space cannot be released	
S070	Overflow of arithmetic buffer	
S071	Stack overflow with arithmetic resolution	
S072	General error with arithmetic resolution	
S073	Beginning of arithmetic statement not found	
S074	End of arithmetic statement not found	
S075	Label to programmed GOTO missing	
	SYSTEM TEXT: FATAL ERROR	This message appears in the event of a fatal error in the central text access of the WS800 system. The error type is indicated by an error identifier (this is important for error diagnostics).

System errors which can be removed

Message No.	Error text	Notes
S100	Overflow, E symbol list	The maximum number of declarations of nine-digit constants is exceeded
S101	Overflow, symbol list	The maximum number of declarations is exceeded. No further R parameters, variable names, constants etc. can be declared. Remedy: Several variable names should be eliminated.
S102	Overflow, constant buffer	Remedy: Insert blank line
S103	Overflow, internal buffer	Remedy: Insert blank line
S104	Overflow, constant buffer for a line	The maximum number of digits and strings within a line has been exceeded. Remedy: Separate the line
S105	Maximum number of words per line reached	Only a limited number of words can be processed per line. Remedy: Separate the line
S120	Block number too high as a result of program branching	The maximum number of block numbers within a nesting level of a program branching type, which is generated by the code generator, is exceeded. Remedy: Insert an additional block or eliminate a block level.
S125	Too few intermediate locations for bracketing	
S126	Too few intermediate locations for multiplication and division before addition and subtraction	
S127	Too few intermediate locations for functions	There are no longer sufficient intermediate locations for arithmetic and logical assignments or conditions. Remedy: Separate assignments or conditions
S200	Printer cannot be addressed	

Section 10

-Object code-

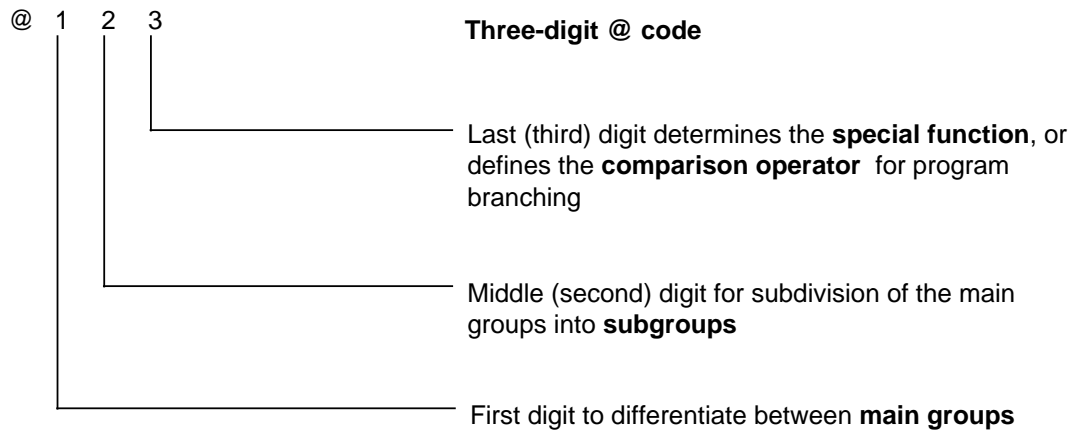
Overview:

- 10.1 Structure of the @ code**
 - 10.1.1 Subdivision into main groups
 - 10.1.2 Operands after the @ function
 - 10.1.3 Notation
- 10.2 General statements for the program structure**
- 10.3 Program branching**
- 10.4 General data transfer**
- 10.5 Data transfer: System memory into the R parameter**
- 10.6 Data transfer: R parameter into the system memory**
- 10.7 File handling, general: (in preparation)**
- 10.8 Mathematical and logical functions**
- 10.9 NC-specific functions**
- 10.10 I/O statements**

10 Object code

10.1 Structure of the @ code

The @ code consists of three digits which are interpreted as follows:



10.1.1 Subdivision into main groups

First digit of the @ code	Function
0	General statement for program structure
1	Program branching
2	Data transfer, general
3	Data transfer, system locations into the R parameter
4	Data transfer, R parameter into the system memory
5	File handling, general (in preparation)
6	Mathematical and logical functions
7	NC specific functions
a	I/O statements

10.1.2 Operands after the @ function

The @ code requires supplementary information (operands) for its function. These operands are defined using the following letters:

K... Constants

R... R parameter (register)

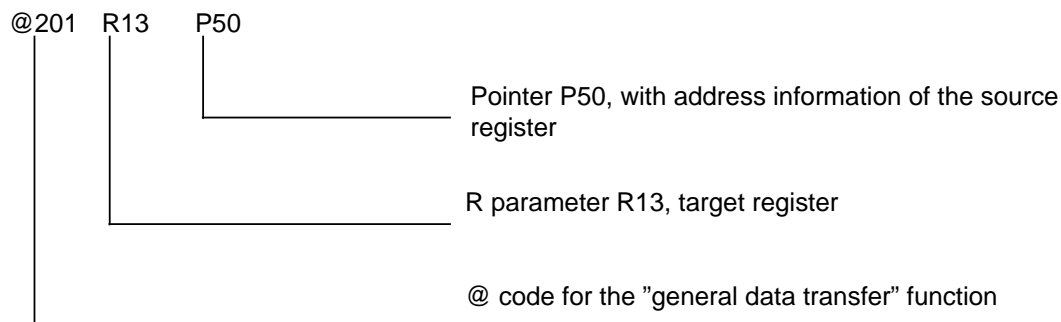
P... Pointer variable (pointer)

The value defined by constant K is fixed in the program and cannot be changed (direct value specification).

The value located in an R parameter can be changed by the program (indirect value specification).

The pointer points to a parameter in which the address of the parameter is located and to the contents to which the function should be applied (indirect value specification).

Example for @ code with operands:



Load the contents of the source register, whose address is located in register R37, into target register R13.

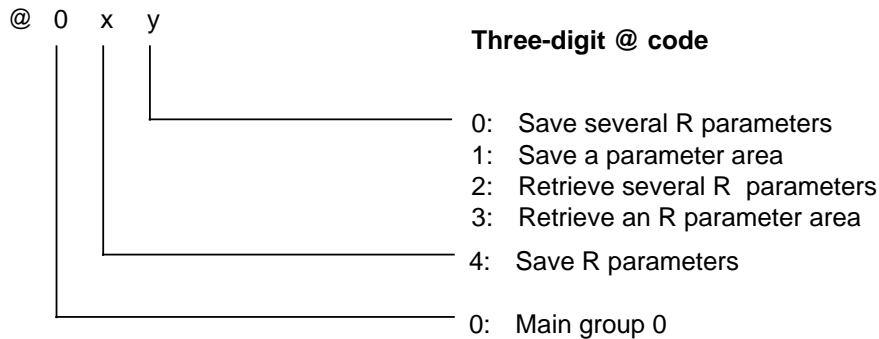
10.1.3 Notation

The @ code requires a rigid notation. The three-digit @ code in the command list is followed by a series of notation specifications, which are located in angled brackets (see Section 10.2). The individual notations have the following meaning:

- <Const> direct value specification (constant K)
- <R-Par> indirect value specification (R parameter)
- <Var> indirect value specification (R parameter or pointer)
- <Value > composite value specification (constant, R parameter or pointer)

10.2 General statements for the program structure

The main group 0 is subdivided as follows:



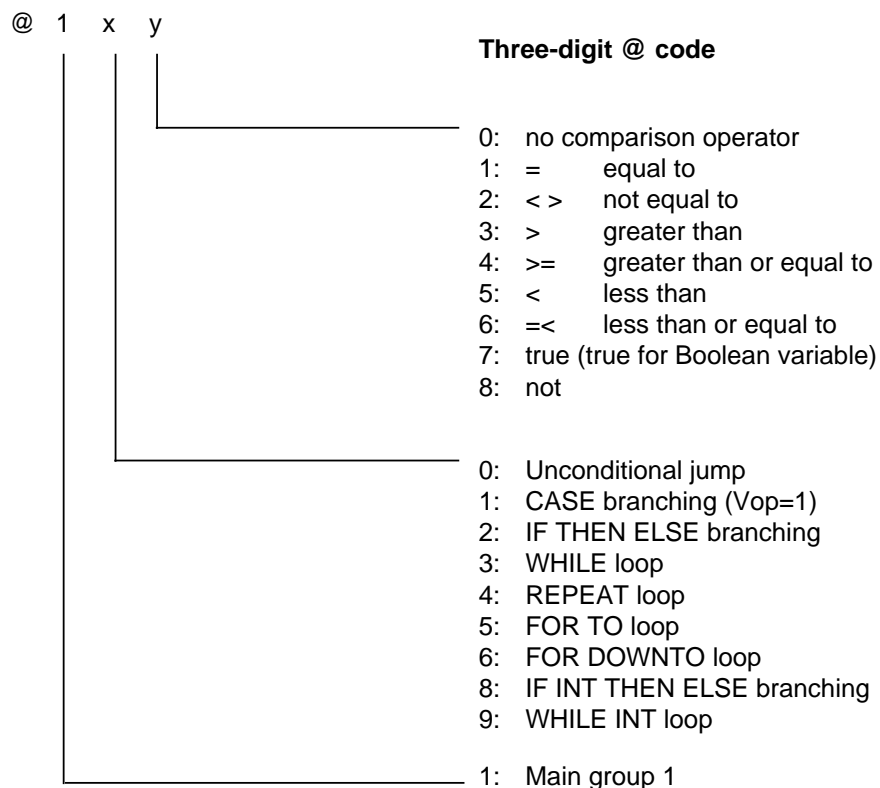
These commands are used if R parameters are processed in a subroutine, which could already possibly be used in a higher level. If a push command (@ 040 or @ 041) is written at the start of subroutine, the values are saved, and the specified R parameter is set with the value "0".

The original status is re-established with a pop command (@ 042 or @ 043) at the end of the subroutine.

Program header statement		
@ code	CL 800 statement	Function
%SPFnr	PROGRAM <Progr. number>;	Program definition
@00f	ESS;	Enable for softkey start
M17	END;	End of program
LF	LF;	End of block
	(*Comments*)	Comments exclusively for CL 800 source language.
<DIN code>	(:<DIN code>);	DIN code (is passed through by the compiler).
(Text)	(:(Text):);	Note in the NC program

Save R parameters: Main group 0/subgroup 4		
@ code	CL 800 statement	Function
@040 <Const><R Par 1>...<R Par n>	Push	not at the CL 800 level
@041 <R Par 1><R Par 2>	Push block	
@042 <Const><R Par 1>...<R Par n>	Pop	
@043 <R Par 1><R Par 2>	Pop block	
		Save the specified local R parameters in the stack
		Save a group of local R parameters in the stack
		Fetch saved R parameters from the stack
		Fetch group of saved R parameters from the stack

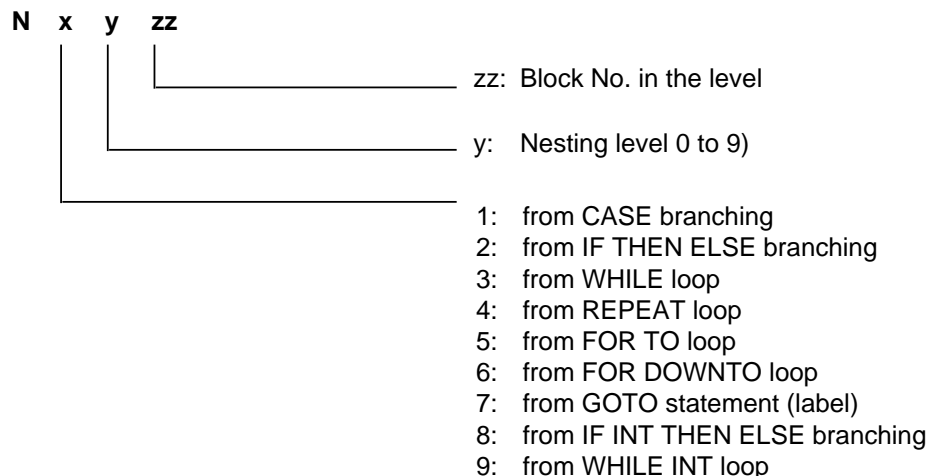
10.3 Program branching



Notes to main group 1:

The block numbers generated by the compiler always have four digits. Thus, a maximum of three-digit block numbers are available in the CL800 program.

The block number generated from the compiler has the following structure:



Unconditional jump: Main group1/ subgroup 0		
@ code	CL 800 statement	Function
@100 <Const, +/- block No.>	GOTO <Label>;	Unconditional jump to NC block

CASE branching: Main group 1/ subgroup 1		
@ code	CL 800 statement	Function
@111 <Var> <Value 1><Const, block No. for Statement 1> . . <Value n><Const,block No. for state. n > @100 <Const,block No., cont. without statement n+1> N1... <Statement 1> @100 <Const, block No. continuation> . N1... <Statement n> @100 <Const, block No. continuation> [N1... <Statement n+1>] N1... <Continuation>	CASE <Var> = <Value 1>:<Statement 1>; . . =<Value n>:<Statement n>; [OTHERWISE:<Statement n+1>;] ENDCASE;	CASE branching

IF THEN ELSE branching: Main group 1 / subgroup 2		
@ code	CL 800 statement	Function
@12x <Var><Value ><Const. block No. statement 2 w/out cont.><Statement 1> [@100 <Const,block No., continuation> N2... <Statement 2>] N2... <Continuation>	IF "condition" ¹⁾ THEN <Statement 1>; [ELSE <Statement 2>;] ENDIF;	IF THEN ELSE branching x --> comparison operator (Vop)

WHILE loop: Main group 1 / subgroup 3		
@ code	CL 800 statement	Function
N3... <Block No. start> @13x <Var> <Value ><Const. block No., continuation><Statement > @100 <Const. block No. start> N3... <Continuation>	WHILE "condition" ¹⁾ DO <Statement >;	Repeat statement with scanning of repeat condition at the start. x --> comparison operator (Vop)

- 1) "Condition":
- <Var> = Boolean variable
 - <Var>.<Const> = Bit from pattern
 - <Var> "Vop" <Value>
 - Extended condition

REPEAT loop: Main group 1 / subgroup 4		
@ code	CL 800 statement	Function
N4... <Block No. start> <Statement>; @14x <Var><Value ><Const, block No. start> N4... <Continuation>	REPEAT <Statement>; UNTIL "Condition"; 1)	Repeat statement with scanning of the repeat condition at the end x -> comparison operator (Vop)

FOR TO loop: Main group 1 / subgroup 5		
@ code	CL 800 statement	Function
<Var> = <Value 1> N5... <Block No. start> @151 <Var><Value 2><Const. block No. cont.> <Statement> @620 <Var> @100 <Const, block No. start> N5... <Continuation>	FOR <Var>= <Value 1> TO <Value 2> DO <Statement>;	Counting loop with statement repetition from index <Value 1> up to <Value 2>. The variable index is incremented at each pass.

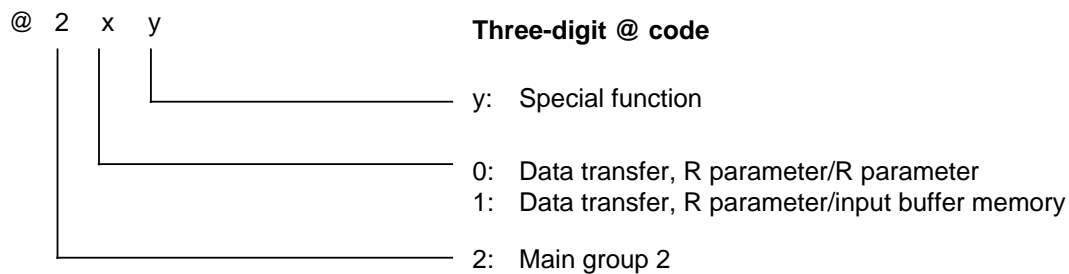
FOR DOWNTO loop: Main group 1 / subgroup 6		
@ code	CL 800 statement	Function
<Var> = <Value 1> N6... <Block No. start> @161 <Var><Value 2><Const. block No. cont.> <Statement> @621 <Var> @100 <Const. block No. start> N6... <Continuation>	FOR <Var>= <Value 1>DOWNTO <Value 2>: DO <Statement>;	Counting loop with statement repetition from index <Value 1> to <Value 2>. The variable index is decremented at each pass.

IF THEN ELSE branch: Main group 1 / subgroup 8		
@ code	CL 800 statement	Function
@187 <Value 1><Value 2><Const. block No. statement 2 without cont.> <Statement1> [@100 <Const. block No. continuation> N8... <Statement2>] N8... <Continuation>	IF INT <Value 1>,<Value 2> THEN <Statement1>; [ELSE <Statement2>;] ENDIF;	IF THEN ELSE branch with scanning of a specific external input for "1" signal.
@188 <Value 1><Value 2><Const. block No. statement 2 or continuation> <Statement1> [@100 <Const. block No., continuation> N8... <Statement2>] N8... <Continuation>	IF INT NOT <Value 1>,<Value 2> THEN <Statement1>; [ELSE <Statement2>;] ENDIF;	IF THEN ELSE branch with scanning of a specific external input for "0" signal.

WHILE INT loop: Main group 1 / subgroup 9		
@ code	CL 800 statement	Function
N9... <Block No. start> @197<Val. 1><Val. 2><Const. block No. cont.> <Statement> @100 <Const. block No. start> N9... <Continuation>	WHILE INT <Value 1>,<Value 2> DO <Statement> ;	Repeat statement with scanning of a specific external input at the start for "1" signal
N9... <Block No. start> @197<Val. 1><Val. 2><Const. block No. cont.> <Statement> @100 <Const. block No. start> N9... <Continuation>	WHILE INT <Value 1>,<Value 2> DO <Statement> ;	Repeat statement with scanning of a specific external input at the start for "0" signal.

10.4 General data transfer

Main group 2 (general data transfer) is subdivided as follows:



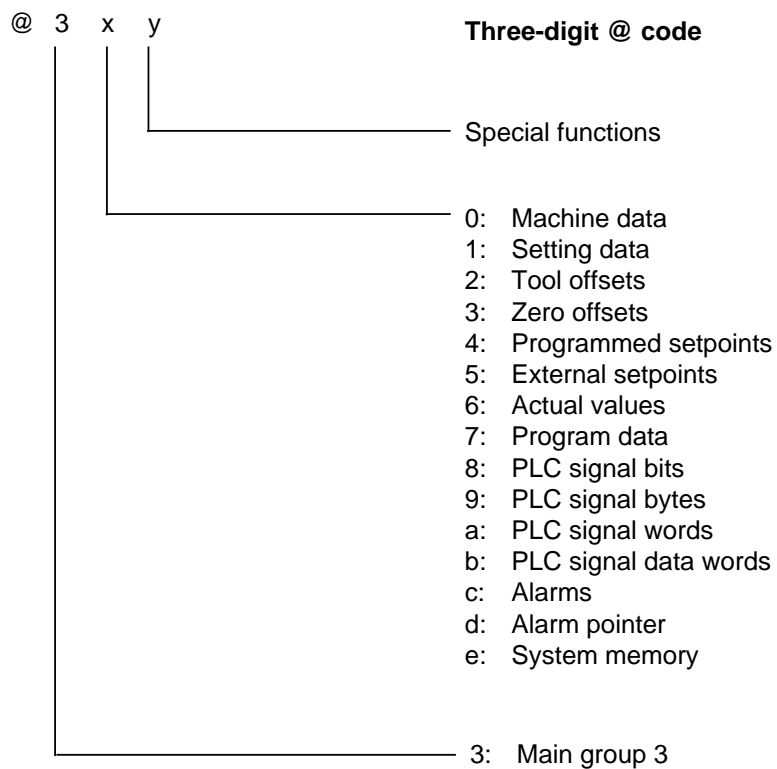
Data transfer/R parameter: Main group 2 / subgroup 0		
@ code	CL 800 statement	Function
@200 <Var>	CLEAR (<Var>);	Clear variable
@201 <Var><Value >	<Var>=<Value >;	Load variable with value
@202 <Var1><Var2>	XCHG (<Var1>,<Var2>);	Exchange contents of the variables
@203 <Var1><Var2><Const>		Read bit from bit pattern

10.5 Data transfer: System memory into the R parameter

Data transfer R parameter / input buffer memory for numerical variable main group 2/ subgroup 1		
@ code	CL 800 statement	Function
@210 <Value3><Value4>	CLEAR MIB (<Value3>,<Value4>);	Clear input buf. mem.: Value3:Input buf. start address 0..499 Value4:Input buf. mem. final address 0..499
@211 <Var><Value1>	<Var>=MIB (<Value1>);	Read input buf.mem. Value1:Input buffer memory No. 0...499
@212 <Value1><Value>	MIB (<Value>)=<Value>;	Write input buf. mem. Value1:Input buf. mem. No. 0...499

10.5 Data transfer: System memory into the R parameter

The **main group 3** (data transfer: System memory into the R parameter) is structured as follows:



Transfer machine data into R parameter: Main group 3 / subgroup 0		
@ code	CL 800 statement	Function
@300 <Var><Value1>	<Var>= MDN (<Value1>);	NC machine data Value1: Word addr. 0...4999
@301 <Var><Value1>	<Var>= MDNBY (<Value1>);	NC machine data, bytes Value1: Byte addr. 5000...6999
@302 <Var><Value1><Value2>	<Var>= MDNBI (<Value1>, <Value2>);	NC machine data, bits Value1: Byte addr. 5000...6999 Value2: Bit addr. 0...7
@303 <Var><Value1><Value2>	<Var>= MDZ (<Value1>, <Value2>);	Cycle machine data Value1: Chan. No. 0...16 Value2: Word addr. - Ch.-orient. 0...449 - Central 1000...4149
@304 <Var><Value1><Value2>	<Var>= MDZBY (<Value1>, <Value2>);	Cycle machine data, bytes Value1: Chan. No. 0...16 Value2: Byte addr. -Chan.-orient. 800..949 -Central 7000...8049
@305 <Var><Value1><Value2><Value3>	<Var>= MDZBI (<Value1>, <Value2>, <Value3>);	Cycle machine data, bits Value1: Chan. No. 0...16 Value2: Byte addr. -Ch.-orient. 800..949 -Central 7000...8049 Value3: Bit addr. 0...7
@306 <Var><Value1>	<Var>= MDP (<Value1>);	PLC machine data Value1: 850/880=Word addr. 0-5999 810=Word addr. 0-1999
@307 <Var><Value1>	<Var>= MDPBY (<Value1>);	PLC machine data, bytes Value1: 850/880=Word addr. 6000-8999 810=Word addr. 2000-3999
@308 <Var><Value1><Value2>	<Var>= MDPBI (<Value1>, <Value2>);	PLC machine data, bits Value1: 850/880=Word addr. 6000-8999 810=Word addr. 2000-3999 Value2: Bit addr. 0...7
@309 <Var><Value1>	<Var>= MDD (<Value1>);	Drives machine data Value1: Word addr. 0...4960
@30a <Var><Value1><Value2>	<Var>= MDDBY (<Value1>, <Value2>);	Drives machine data, bytes Value1: Word addr. 2440...4630 Value2: 0 = low byte 1 = high byte
@30b <Var><Value1><Value2><Value3>	<Var>= MDDBI (<Value1>, <Value2>, <Value3>);	Drives machine data, bits Value1: Word addr. 2440...4630 Value2: 0 = low byte 1 = high byte Wert3: Bit addr. 0...7

10.5 Data transfer: System memory into the R parameter

Transfer setting data into the R parameter: Main group 3 / subgroup 1		
@ code	CL 800 statement	Function
@310 <Var><Value1>	<Var>= SEN (<Value1>);	NC setting data Value1 :Word addr. 0...4999
@311 <Var><Value1>	<Var>= SENBY (<Value1>);	NC setting data, bytes Value1 :Byte addr. 5000...9999
@312 <Var><Value1><Value2>	<Var>= SENBI (<Value1>, <Value2>);	NC setting data, bits Value1 :Byte addr. 5000...9999 Value2 :Bit addr. 0...7
@313 <Var><Value1><Value2>	<Var>= SEZ (<Value1>, <Value2>);	Cycles setting data Value1 :Chan. No. 0...16 Value2 :Word addr. - Chan.-orientated 0...449
@314 <Var><Value1><Value2>	<Var>= SEZBY (<Value1>, <Value2>);	Cycles setting data, bytes Value1 :Chan. No. 0...16 Value2 :Byte addr. - Chan.-orient. 800...949
@315 <Var><Value1><Value2><Value3>	<Var>= SEZBI (<Value1>, <Value2>,<Value3>);	Cycles setting data, bits Value1 : Chan. No. 0...16 Value2 : Byte addr. - Chan.-orient. 800...949 Value3 : Bit addr. 0...7

Transfer tool offsets into the R parameter: Main group 3 / subgroup 2		
@ code	CL 800 statement	Function
@320 <Var><Value1><Value2><Value3>	<Var>= TOS (<Value1>, <Value2>,<Value3>);	Tool offset Value1: TO range 0...16 Value2: D No. 1...818 Value3: P No. 0...15

Transfer zero offsets into the R parameter: Main group 3 / subgroup 3		
@ code	CL 800 statement	Function
@330 <Var><Value1><Value2><Value3>	<Var>= ZOA (<Value1>, <Value2>,<Value3>);	Settable zero offset (G54-G57) Value1 :Gr. 1..4 (G54-G57) Value2 :Axis No. 1...24 Value3 :Coarse/fine (0/1)
@331 <Var><Value1><Value2>	<Var>= ZOPR (<Value1>, <Value2>);	Programmable zero offset (G58, G59) Value1:Gr. 1 o. 2 (G58 o.G59) Value2 :Axis No. 1...24
@332 <Var><Value2>	<Var>= ZOE (<Value2>);	External zero offset from PLC Value2 :Axis No. 1...24
@333 <Var><Value2>	<Var>= ZOD (<Value2>);	DRF offset Value2 :Axis No. 1...24
@334 <Var><Value2>	<Var>= ZOPS (<Value2>);	PRESET offset Value2 :Axis No. 1...24
@336 <Var><Value2>	<Var>= ZOS (<Value2>);	Sum of all offsets Value2 :Axis No. 1...24
@337 <Var><Value1><Value2><Value3>	<Var>= ZOADW (<Value1>, <Value2>,<Value3>);	Settable coordiante rotation (G54-G57) Value1 :Chan. No. 0...16 Value2 :Gr. 1...4 (G54-G57) Value3 :Angle No.
@338 <Var><Value1><Value2><Value3>	<Var>= ZOPRDW (<Value1>, <Value2>,<Value3>);	Program. coordinate rotation (G58, G59) Value1 :Chan. No. 0...16 Value2:Gr. 1 o. 2 (G58, G59) Value3 :Angle No.

Read programmed setpoints into the R parameter: Main group 3 / subgroup 4		
@ code	CL 800 statement	Function
@342 <Var><Value1><Value3>	<Var>= PRSS (<Value1>, <Value3>);	Programmed spindle speed Value1 :Chan. No. 0...16 Value3 :Spindle No. 0...6
@345 <Var><Value1><Value2>	<Var>= PRVC (<Value1>, <Value2>);	Programmed cutting speed Value1 :Chan. No. 0...16 Value2 :0 = G96
@34b <Var><Value1><Value2><Value3>	<Var>= PCDA (<Value1>, <Value2>,<Value3>);	Programmed control words for digital axis drive Value1 :Axis No. 1...24 Value2 :Byte addr. 0 or 1 Value3 :Bit addr. 0...7
@34c <Var><Value1><Value2><Value3>	<Var>= PCDS (<Value1>, <Value2>,<Value3>);	Programmed control words for digital spindle drives Value1 :Spindle No. 1...10 Value2 :Byte addr. 0...5 Value3 :Bit addr. 0...7

Read actual values into the R parameter: Main group 3 / subgroup 6

@ code	CL 800 statement	Function
@360 <Var><Value2>	<Var>= ACPW (<Value2>);	Actual axis position, workpiece-related Value2 :Axis No. 1...24
@361 <Var><Value2>	<Var>= ACPM (<Value2>);	Actual axis position, machine-related Value2 :Axis No. 1...24
@362 <Var><Value2>	<Var>= ACP (<Value2>);	Actual axis position Value2 :Axis No. 1...24
@363 <Var><Value2>	<Var>= ACSP (<Value2>);	Actual spindle position Value2 :Spindle No. 0...6
@364 <Var><Value2>	<Var>= ACSS (<Value2>);	Actual spindle speed Value2 :Spindle No. 0...6
@367 <Var><Value1>	<Var>= ACAS (<Value1>);	Axis No. of the actual plane/leading spindle No. Value1 :Channel No. 0...16
@36a <Var><Value1>	<Var>= ACD (<Value1>);	Actual D-function Value1 :Channel No. 0...16
@36b <Var><Value1><Value3>	<Var>= ACG (<Value1>, <Value3>);	Actual G-function Value1 :Channel No. 0...16 Value3 :Group 0...23

Read program data into the R parameter: Main group 3 / subgroup 7

@ code	CL 800 statement	Function
@371 <Var><Value1><Value3>	<Var>= SOB (<Value1>, <Value3>);	Special bits Value1:channel-dependent bits =Channel No. 0-n channel-independent bits = 99 Value3: Bit No. 0...7
@372 <Var>	<Var>= PPCH;	Actual channel No. for program

Read PLC signal bits into the R parameter: Main group 3 / subgroup 8		
@ code	CL 800 statement	Function
@380 <Var><Value1><Value2><Value3>	<Var>= PLCI (<Value1>, <Value2>,<Value3>);	PLC input bit Value1 :PLC No. 1...4 Value2 :Byte addr. 0...127 Value3 :Bit No. 0...7
@381 <Var><Value1><Value2><Value3>	<Var>= PLCQ (<Value1>, <Value2>,<Value3>);	PLC output bit Value1 :PLC No. 1...4 Value2 :Byte addr. 0...127 Value3 :Bit No. 0...7
@382 <Var><Value1><Value2><Value3>	<Var>= PLCF (<Value1>, <Value2>,<Value3>);	PLC flag bit Value1 :PLC No. 1...4 Value2 :Byte addr. 0...255 Value3 :Bit No. 0...7
@383 <Var><Value1><Value2><Value3> <Value4>	<Var>= PLCW (<Value1>, <Value2>,<Value3>, <Value4>);	PLC data word bit Value1 :PLC No. 1...4 Value2 :DB No. 1...255 DX No. 1000...1255 Value3 :DW No. 0...2043 Value4 :Bit No. 0...15

Read PLC signal bytes into the R parameter: Main group 3 / subgroup 9		
@ code	CL 800 statement	Function
@390 <Var><Value1><Value2>	<Var>= PLCIB (<Value1>, <Value2>);	PLC input byte Value1 :PLC No. 1...4 Value2 :Byte addr. 0...127
@391 <Var><Value1><Value2>	<Var>= PLCQB (<Value1>, <Value2>);	PLC output byte Value1 :PLC No. 1...4 Value2 :Byte addr. 0...127
@392 <Var><Value1><Value2>	<Var>= PLCPB (<Value1>, <Value2>);	PLC peripheral byte Value1 :PLC No. 1...4 Value2 :Byte addr. 0...127
@393 <Var><Value1><Value2>	<Var>= PLCFB (<Value1>, <Value2>);	PLC flag byte Value1 :PLC No. 1...4 Value2 :Byte addr. 0...255
@394 <Var><Value1><Value2><Value3>	<Var>= PLCDBL (<Value1>, <Value2>,<Value3>);	PLC data word, left Value1 :PLC No. 1...4 Value2 :DB No. 1...255 DX No. 1000...1255 Value3 :DW No. 0...2043
@395 <Var><Value1><Value2><Value3>	<Var>= PLCDBR (<Value1>, <Value2>,<Value3>);	PLC data word, right Value1 :PLC No. 1...4 Value2 :DB No. 1...255 DX No. 1000...1255 Value3 :DW No. 0...2043

10.5 Data transfer: System memory into the R parameter

Read PLC signal words into the R parameter: Main group 3 / subgroup a		
@ code	CL 800 statement	Function
@3a0 <Var><Value1><Value2><Value3>	<Var>= PLCIW (<Value1>, <Value2>,<Value3>);	PLC input word Value1: PLC No. 1...4 Value2: Word addr. 0...126 Value3: DIM ident. 0 ... 9 fixed point 100 ... 109 BCD
@3a1 <Var><Value1><Value2><Value3>	<Var>= PLCQW (<Value1>, <Value2>,<Value3>);	PLC output word Value1: PLC No. 1...4 Value2: Word addr. 0...126 Value3: DIM ident. 0 ... 9 fixed point 100 ... 109 BCD
@3a2 <Var><Value1><Value2><Value3>	<Var>= PLCPW (<Value1>, <Value2>,<Value3>);	PLC peripheral word Value1: PLC No. 1...4 Value2: Word addr. 0...126 Value3: DIM ident. 0 ... 9 fixed point 100 ... 109 BCD
@3a3 <Var><Value1><Value2><Value3>	<Var>= PLCFW (<Value1>, <Value2>,<Value3>);	PLC flag word Value1: PLC No. 1...4 Value2: Word addr. 0...254 Value3: DIM ident. 0 ... 999 fixed point 100 ... 109 BCD
@3a4 <Var><Value1><Value2>	<Var>= PLCT (<Value1>, <Value2>);	PLC timer Value1: PLC No. 1...4 Value2: Timer addr. 0...255
@3a5 <Var><Value1><Value2>	<Var>= PLCC (<Value1>, <Value2>);	PLC timer Value1: PLC No. 1...4 Value2: Counter addr. 0...255

Read PLC signal data words into the R parameter: Main group 3/ subgroup b		
@ code	CL 800 statement	Function
@3b0 <Var><Value1><Value2><Value3> <Value4><Value5>	<Var>= PLCDF (<Value1>, <Value2>,<Value3>, <Value4>,<Value5>);	PLC data word, fixed point Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000...1255 Value3: DW No. 0...2043 Value4: No. of DW 1 or 2 Value5: DIM ident. 0...9 serial 10...19 parallel
@3b1 <Var><Value1><Value2><Value3> <Value4><Value5>	<Var>= PLCDB (<Value1>, <Value2>,<Value3>, <Value4>,<Value5>);	PLC data word BCD Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000...1255 Value3: DW No. 0...2043 Value4: No. of DW 1...3 Value5: DIM ident. 100...109 parallel
@3b2 <Var><Value1><Value2><Value3>	<Var>= PLCDG (<Value1>, <Value2>,<Value3>);	PLC data word, floating point Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000...1255 Value3: DW No. 0...2043

Read alarms into the R parameter: Main group 3/ subgroup c		
@ code	CL 800 statement	Function
@3c0 <Var>	<Var>= ALNP ();	NC alarms

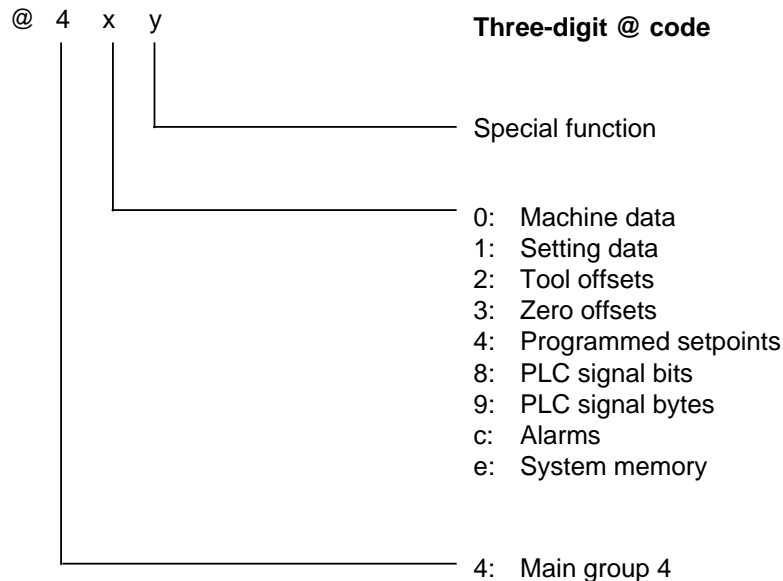
Read alarm pointer into the R parameter: Main group 3/ subgroup d		
@ code	CL 800 statement	Function
@3d0 <Var>	<Var>= ALNPZ ();	NC alarm pointer

10.6 Data transfer: R parameter into the system memory

Read system memory into R parameter: Main group 3 / subgroup e		
@ code	CL 800 statement	Function
@3e1 <Var><Value1><Value2><Value3>	<Var>= RSDA (<Value1>, <Value2>,<Value3>);	Axis status for digital drives Value1: Axis No. 1 ... to Value2: Byte addr.0 or 1 Value3: Bit addr. 0 ... 7
@3e2 <Var><Value1><Value2><Value3>	<Var>= RSDS (<Value1>, <Value2>,<Value3>);	Spindle status for digital drives Value1: Spindle No.1... 10 Value2: Byte addr.0 ... 3 Value3: Bit addr. 0 ... 7
@3e3 <Var><Value1><Value2><Value3>	<Var>= RSDD (<Value1>, <Value2>,<Value3>);	Equipment status for dig. drives Value1: Unit No. 1 or 2 Value2: Byte addr. 0 or 1 Value3: Bit addr. 0 ... 7
@3e4 <Var><Value1>	<Var>= AGS (<Value1>);	Active gear stage Value1: Spindle No. 0 ... 6

10.6 Data transfer: R parameter into the system memory

Main group 4 (data transfer: R parameter into the system memory) is structured as follows:



All @ commands of this main group have <Value> as last notation. Thus, the numerical value to be transferred is either defined directly with a constant, or indirectly through an R parameter or pointer.

Transfer R parameter into the machine data: Main group 4 / subgroup 0		
@ code	CL 800 statement	Function
@400 <Value1><Value>	MDN (<Value1>) = <Value>;	NC machine data Value1:Addr. 0...4999
@401 <Value1><Value>	MDNBY (<Value1>) = <Value>;	NC machine data, bytes Value1:Addr. 5000...6999
@402 <Value1><Value2><Value>	MDNBI (<Value1>,<Value2>) =<Value>;	NC machine data, bits Value1:Byte addr. 5000...6999 Value2: Bit addr. 0...7
@403 <Value1><Value2><Value>	MDZ (<Value1>,<Value2>) =<Value>;	Cycle machine data Value1:Channel No. 0...16 Value2:Word addr. - Chan. orient. 0...449 - Central 1000...4149
@404 <Value1><Value2><Value>	MDZBY (<Value1>,<Value2>) =<Value>;	Cycle machine data, bytes Value1:Channel No. 0...16 Value2:Byte addr. - >Chan. orient. 800...949 - >Central 7000...8049
@405 <Value1><Value2><Value3> <Value>	MDZBI (<Value1>,<Value2>, <Value3>) =<Value>;	Cycle machine data, bits Value1:Channel No. 0...16 Value2:Byte addr. - >Chan. orient. 800...949 - >Central 7000...8049 Value3:Bit addr. 0...7
@406 <Value1><Value>	MDP (<Value1>) =<Value>;	PLC machine data Value1: 850/880=Addr.0...5999 810=Addr. 0...1999
@407 <Value1><Value>	MDPBY (<Value1>) =<Value>;	PLC machine data, bytes Value1: 850/880=byte addr. 6000...8999 810=Byte addr. 2000...3999
@408 <Value1><Value2><Value>	MDPBI (<Value1>,<Value2>) =<Value>;	PLC machine data, bits Value1: 850/880=byte addr. 6000...8999 810=byte addr. 2000...3999 Value2:Bit addr. 0...7
@409 <Value1><Value>	MDD (<Value1>) =<Value>;	Drives machine data Value1:Word addr. 880...4960
@40a <Value1><Value2><Value>	MDDBY (<Value1>,<Value2>) =<Value>;	Drives machine data, bytes Value1:Word addr. 2440...4610 Value2:0=Low byte 1=High byte
@40b <Value1><Value2><Value3> <Value>	MDDBI (<Value1>,<Value2>, <Value3>) =<Value>;	Drives machine data, bits Value1:Word addr. 2440...4610 Value2:0=Low byte 1=High byte Value3: Bit addr. 0...7

10.6 Data transfer: R parameter into the system memory

Transfer R parameter into the setting data: Main group 4 / subgroup 1		
@ code	CL 800 statement	Function
@410 <Value1><Value>	SEN (<Value1>) =<Value>;	NC setting data Value1: addr.. 0...4999
@411 <Value1><Value>	SENB (<Value1>) =<Value>;	NC setting data, bytes Value1: Byte addr. 5000...9999
@412 <Value1><Value2><Value>	SENB (<Value1>,<Value2>) =<Value>;	NC setting data, bits Value1: Byte addr. 5000...9999 Value2: Bit-Adr. 0...7
@413 <Value1><Value2><Value>	SEZ (<Value1>,<Value2>) =<Value>;	Cycle setting data Value1: Chan. No. 0...16 Value2: Word addr. - Chan. orient. 0...499
@414 <Value1><Value2><Value>	SEZB (<Value1>,<Value2>) =<Value>;	Cycle setting data, bytes Value1: Chan. No. 0...16 Value2: Byte addr. - Chan. orient. 800...949
@415 <Value1><Value2><Value3><Value>	SEZB (<Value1>,<Value2>, <Value3>) =<Value>;	Cycle setting data, bits Value1: Chan. No. 0...165 Value2: Byte addr. - Chan. orient. 800...949 Value3: Bit addr. 0...7

Write R parameter into the tool offset: Main group 4 / subgroup 2		
@ code	CL 800 statement	Function
@420 <Value1><Value2><Value3> <Value>	TOS (<Value1>,<Value2>, <Value3>) =<Value>;	Tool offset Value1: TO range 0...16 Value2: D No. 1...818 Value3: P No.
@423 <Value1><Value2><Value3> <Value>	TOAD (<Value1>,<Value2>, <Value3>) =<Value>;	Tool offset, additive Value1: TO range 0...16 Value2: D No. 1...818 Value3: P No. 0...15

Write R parameter into the zero offsets: Main group 4 / subgroup 3		
@ code	CL800 statement	Function
@430 <Value1><Value2><Value3> <Value>	ZOA (<Value1>,<Value2>, <Value3>) =<Value>;	Settable zero offset, various Value1: Group 1...4 (G54...G57) Value2: Axis No. 1...24 Value3: Coarse/fine(0/1)
@431 <Value1><Value2><Value3> <Value>	ZOFA (<Value1>,<Value2>, <Value3>) =<Value>;	Settable zero offset, additive Value1: Group 1...4 (G54...G57) Value2: Axis No. 1...24 Value3: Coarse/fine(0/1)
@432 <Value1><Value2><Value>	ZOPR(<Value1>,<Value2>) =<Value>;	Programmable zero offset, various Value1: Group 1 or 2 (G58 o. G59) Value2: Axis No. 1...24
@434 <Value2><Value>	ZOD (<Value2>) =<Value>;	DRF offset Value2: Axis No. 1...24
@435 <Value2><Value>	ZOPS(<Value2>) =<Value>;	PRESET offset Value2: Axis No. 1...24
@437 <Value1><Value2><Value3><Value>	ZOADW (<Value1>,<Value2>, <Value3>)=<Value>;	Settable coordinate rotation (G54-G57) Value1: Chan. No. 0...16 Value2: Group 1...4 (G54-G57) Value3: Angle No.
@438 <Value1><Value2><Value3><Value>	ZOFADW (<Value1>,<Value2>, <Value3>)= <Value>;	Settable coordinate rotation (G54-G57) additive Value1: Chan. No. 0...16 Value2: Group 1...4 (G54-G57) Value3: Angle No.
@439 <Value1><Value2><Value3><Value>	ZOPRDW (<Value1>,<Value2>, <Value3>)= <Value>;	Programmable coordinate rotation (G58, G59) Value1: Chan. No. 0...16 Value2: Group 1 or 2 (G58, G59) Value3: Angle No.
@43a <Value1><Value2><Value3><Value>	ZOFPROW(<Value1>, <Value2>, <Value3>)= <Value>;	Programmable coordinate rotation (G58, G59) additive Value1: Chan. No. 0...16 Value2: Group 1 or 2 (G58, G59) Value3: Angle No.

10.6 Data transfer: R parameter into the system memory

Write R parameter into the programmed setpoints: Main group 4 / subgroup 4		
@ code	CL800 statement	Function
@440 <Value3><Value>	PRAP (<Value3>) =<Value>;	Programmed axis position Value3 :Axis No. 1...24
@442 <Value3><Value>	PRSS (<Value3>) =<Value>;	Programmed spindle speed Value3 :Axis No. 0...6
@446 <Value>	PRAD () =<Value>;	Programmed radius
@447 <Value>	PANG () =<Value>;	Programmed angle
@448 <Value3><Value>	PRIP (<Value3>) =<Value>;	Progr. interpolation parameter Value3 :Axis No. 1...24
@44b <Value1><Value2><Value3><Value>	PCDA (<Value1>,<Value2>, <Value3>)=<Value>;	Programmed control words for digital axis drives Value1 :Axis No. 1...24 Value2 :Byte addr. 0 od. 1 Value3 :Bit addr. 4...6
@44c <Value1><Value2><Value3><Value>	PCDS (<Value1>,<Value2>, <Value3>)=<Value>;	Programmed control words for digital spindle drives Value1 :Spindle No. 1...10 Value2 :Byte addr. 0...5 Value3 :Bit addr. undefined

Write R parameter into the PLC signal bits: Main group 4/subgroup 8		
@ code	CL800 statement	Function
@482 <Value1><Value2><Value3><Value>	PLCF (<Value1>,<Value2>, <Value3>)=<Value>;	PLC flag bit Value1 :PLC No. 1...4 Value2 :Byte addr. 0...255 Value3 :Bit No. 0...7
@482 <Value1><Value2><Value3> <Value4><Value>	PLCW (<Value1>,<Value2>, <Value3>,<Value4>) =<Value>;	PLC data word bit Value1 :PLC No. 1...4 Value2 :DB No. 1...255 DX No. 1000..1255 Value3 :DW No. 0...2043 Value4 :Bit No. 0...15

Write R parameter into the PLC signals bytes: Main group 4/subgroup 9		
@ code	CL 800 statement	Function
@493 <Value1><Value2><Value>	PLCFB (<Value1>,<Value2>) =<Value>;	PLC flag byte Value1: PLC No. 1...4 Value2: Byte addr. 0...255
@494 <Value1><Value2><Value3><Value>	PLCDBL (<Value1>,<Value2>, <Value3>)=<Value>;	PLC data word, left Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000...1255 Value3: DW No. 0...2043
@495 <Value1><Value2><Value3><Value>	PLCDBR (<Value1>,<Value2>, <Value3>) =<Value>;	PLC data word, right Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000...1255 Value3: DW No. 0...2043

Write R parameter into the PLC signal word: Main group 4 / subgroup a		
@ code	CL800 statement	Function
@4a3 <Value1><Value2><Value3> <Value>	PLCFW(<Value1>,<Value2>, <Value3>)=<Value>;	PLC flag word Value1: PLC No. 1...4 Value2: Word addr. 0...254 Value3: DIM ident. 0...9 fixed point 100...109 BCD

Write R parameter into the PLC signal data word: Main group 4 / subgroup b		
@ code	CL 800 statement	Function
@4b0 <Value1><Value2><Value3> <Value4><Value5><Value>	PLCDF (<Value1>,<Value2>, <Value3>,<Value4>, <Value5>)=<Value>;	PLC data word, fixed point Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000...1255 Value3: DW No. 0...2043 Value4: No. of DW 1 or 2 Value5: DIM ident. 0 ... 9 serial 10 ... 19 parallel
@4b1 <Value1><Value2><Value3> <Value4><Value5><Value>	PLCDB (<Value1>,<Value2>, <Value3>,<Value4>, <Value5>)=<Value>;	PLC data word, BCD Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000...1255 Value3: DW No. 0...2043 Value4: No. of DW 1...2 Value5: DIM ident. 100...109 BCD
@4b2 <Value1><Value2><Value3> <Value>	PLCDG (<Value1>,<Value2>, <Value3>)=<Value>;	PLC data word, floating point Value1: PLC No. 1...4 Value2: DB No. 1...255 DX No. 1000..1255 Value3: DW No. 0...2043

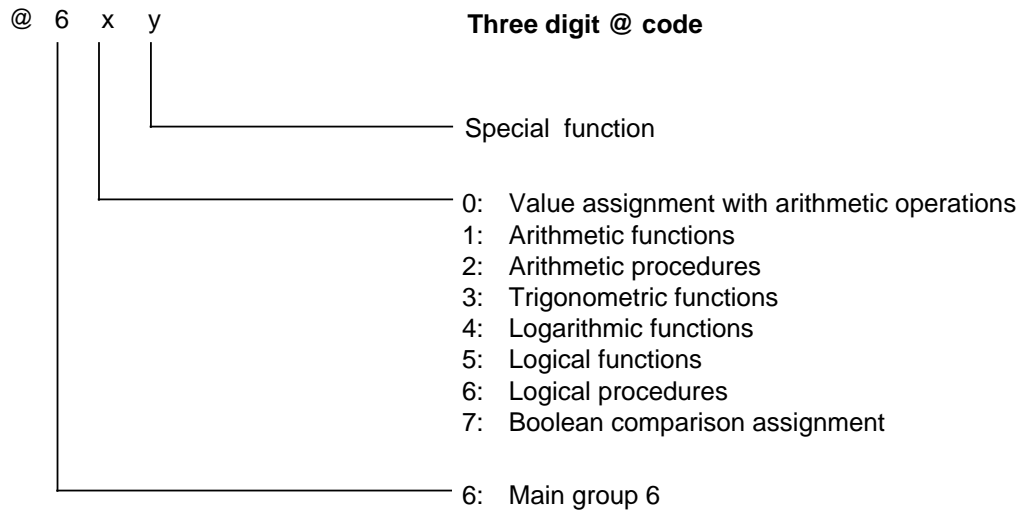
Write R parameter into the alarms: Main group 4 / subgroup c		
@ code	CL 800 statement	Function
@4c0 <Value>	ALNZ () =<Value>;	Cycle alarms Value: Alarm No. 880: 4000...4299 5000...5299 810/820/850: 4000...4299 5000...5099

Write R parameter into the system memory: Maingroup 4 / subgroup e		
@ code	CL 800 statement	Function
@4e1 <Value1><Value2><Value>	SATC (<Value1>,<Value2>) =<Value>;	Spindle acceleration time constant Value1: Spindle No. 0...6 Value2: Gear stage 1...8 Value3: Spindle acceleration time constant: 0...16000

10.7 File handling, general: (in preparation)

10.8 Mathematical and logical functions

Main group 6 (mathematical and logical functions) is structured as follows:



Value assignment with arithmetic operations: Main group 6 / subgroup 0		
<p>@ is not required in this subgroup. A chain calculation with several notations is permissible on the righthand side of the equation.</p>		
@ code	CL 800 statement	Function
<Var> = <Value1> +<Value2>	<Var>=<Value1>+<Val.2>	Addition
<Var> = <Value1> -<Value2>	<Var>=<Value1> -<Val.2>	Subtraction
<Var> = <Value1> * <Value2>	<Var>=<Value1> * <Val.2>	Multiplication
<Var> = <Value1> / <Value2>	<Var>=<Value1> / <Val.2>	Division

Arithmetic functions: Main group 6 / subgroup 1		
@ code	CL 800 statement	Function
@610 <Var><Value>	<Var>= ABS (<Value>);	Generate abs. value
@613 <Var><Value>	<Var>= SQRT (<Value>);	Square root
@614 <Var><Value1><Value2>	<Var>= SQRTS (<Value1>, <Value2>);	Root of sum of squares

Arithmetic procedures: Main group 6 / subgroup 2		
@ code	CL 800 statement	Function
@620 <Var>	INC (<Var>);	Increment
@621 <Var>	DEC (<Var>);	Decrement
@622 <Var>	TRUNC (<Var>);	Integral part

Trigonometric functions: Main group 6 / subgroup 3

@ code	CL 800 statement	Function
@630 <Var><Value>	<Var>= SIN (<Value>);	Sine
@631 <Var><Value>	<Var>= COS (<Value>);	Cosine
@632 <Var><Value>	<Var>= TAN (<Value>);	Tangent
@634 <Var><Value>	<Var>= ARCSIN (<Value>);	Arc sine
@637 <Var><Value1><Value2>	<Var>= ANGLE (<Value1>, <Value1>);	Angle between 2 vector components

Logarithmic functions: Main group 6 / subgroup 4

@ code	CL 800 statement	Function
@640 <Var><Value>	<Var>= LN (<Value>);	Natural logarithm
@641 <Var><Value>	<Var>= INV LN (<Value>);	Exponential function

Logical functions: Main group 6 / subgroup 5

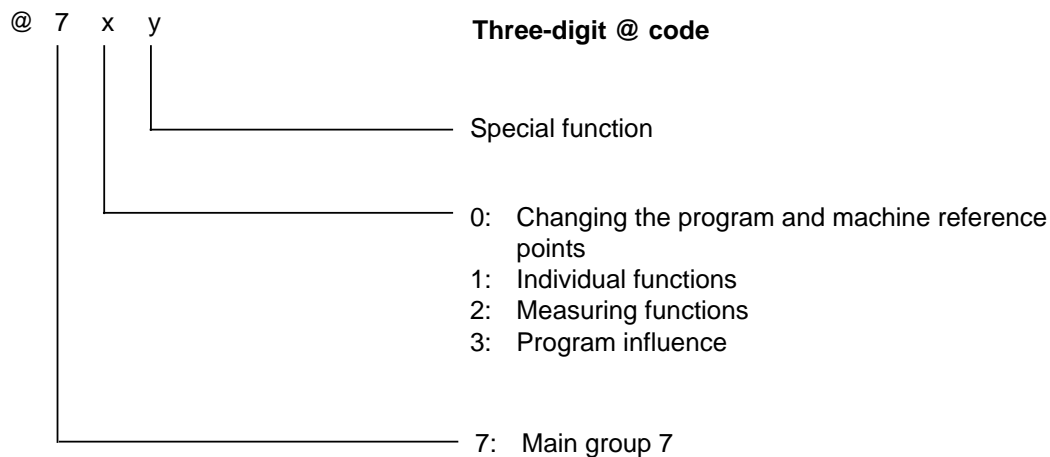
@ code	CL 800 statement	Function
For pattern:		
@650 <Var><Var1><Value>	<Var>=<Var1>OR<Value>;	OR
@651 <Var><Var1><Value>	<Var>=<Var1>XOR<Value>;	EXCLUSIVE OR
@652 <Var><Var1><Value>	<Var>=<Var1>AND<Value>;	AND
@653 <Var><Var1><Value>	<Var>=<Var1>NAND <Value>;	NAND
@654 <Var><Value>	<Var>=NOT <Value>;	NOT
For Boolean or bits:		
@655 <Var><Var1><Value>	<Var>=<Var1>ORB <Value>;	OR bit
@656 <Var><Var1><Value>	<Var>=<Var1>XORB <Value>;	EXKLUSIVE OR bit
@657 <Var><Var1><Value>	<Var>=<Var1>ANDB <Value>;	AND bit
@658 <Var><Var1><Value>	<Var>=<Var1>NANDB <Value>;	NAND bit
@659 <Var><Value>	<Var>=NOTB <Value>;	NOT bit

Logical procedures: Main group 6 / subgroup 6		
@ code	CL 800 statement	Function
@660 <Var><Const>	CLEAR BIT (<Var>.<Const>);	Clear bit in PATTERN Const = Bit No. 0...7
@661 <Var><Const>	SET BIT (<Var>.<Const>);	Set bit Const=Bit No. 0...7

Boolean comparison assignment: Main group 6 / subgroup 7		
@ code	CL 800 statement	Function
@67x <Var1><Var2><Value>		The Boolean variable VAR1 is set to "1" if the comparison of VAR2 and value is fulfilled. Comparison operator "Vop". 0: No comparison operator 1: = equal 2: <> not equal to 3: > greater than 4: >= greater than, equal to 5: < less than 6: <= less than, equal to

10.9 NC-specific functions

The **main group 7** (NC-specific functions) is structured as follows:



Changing program and machine reference points: Main group 7 / subgroup 0		
@ code	CL800 statement	Function
@706	POS MSYS;	Positions in the block ref. to the machine actual value system.

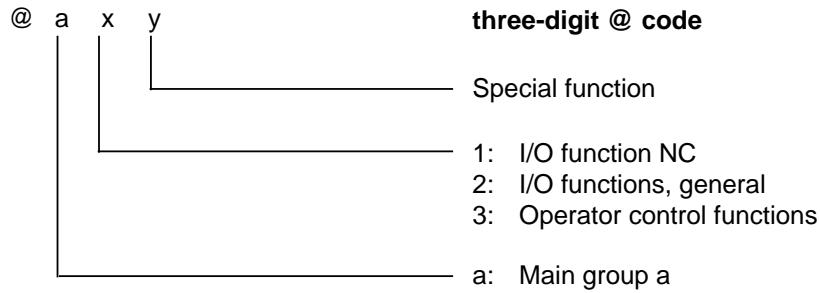
Individual functions: Main group 7 / subgroup 1		
@ code	CL 800 statement	Function
@710 <Var1><Var2>	<Var1>= PREP REF (<Var2>);	Reference preparation Var1:Output data from Var1 Var2:Input data from Var2
@711 <Var1><Var2><Var3>	<Var1>=INT SEC (<Var2>, <Var3>)	Intersection computation. Var1:Output data from Var1 Var2:1st contour from Var2 Var3:2nd contour from Var3 (not possible at present)
@713 <Var>	<Var>=PREP CYC;	Start preparation for cycles Var:Output data from Var
@714	STOP DEC;	Stop decoding until buffer memory empty
@715	STOP DEC1;	Stop decoding until buffer memory is empty at coordinate rotation

Measuring functions: Main group 7 / subgroup 2		
@ code	CL 800 statement	Function
@720 <Var><Value>	<Var>=MEAS M <Value>;	Actual value referred to machine zero. Command position of the axes referred to the workpiece 0. Flying measurement Var : Data stored from Value: No. of measuring input 1 or 2.

Program influence: Main group 7 / subgroup 3		
@ code	CL 800 statement	Function
@730 <Value1><Value2>	REL (<Value1>,<Value2>);	Read inhibit via ext. input Value1:Byte addr. 1 or 2 Value2:Bit addr. 0...7
@736 <Value1><Value2><Value3>	INTA (<Value1>,<Value2>, <Value3>);	Axis-specific remaining travel delete via external input Value1:Axis No. 0...24 Value2:Byte addr. 1 or 2 Value3:Bit addr. +/- 0...7

10.10 I/O statements

Main group a (I/O statements) is structured as follows:



I/O functions NC: Main group a / subgroup 1		
@ code	CL 800 statement	Function
@a15 <Value4><Value5>	WRT PIC (<Value4>,<Value5>);	Menu selection from NC progr. Value4:0=User area 1=Standard area Value5:Menu No. 1...254
@a1b	RECALL PIC;	Return jump to output menu

I/O functions, general: Main group a / subgroup 2		
@ code	CL 800 statement	Function
@a20 <Value>	PORT (<Value>);	Select V24 interface Value: No. V24 interface 1...4
@a25 <Value1>	OUTP ZOA (<Value1>);	Output zero offset via V24 Value1: Chan. No. 0...16
@a26 <Value2><Value3><Value4>	OUTP DATA (<Value2>, <Value3>,<Value4>);	Output data via V24 Value2: Data type 1...9 Value3: Start address Value4: Final address
@a27 <Value1><Value3><Value4>	OUTP PARA (<Value1>, <Value3>,<Value4>);	Parameter output via V24 Value1: Chan. No. 0...16 Value3: Start address Value4: Final address
@a28 <Value2>	INP (<Value2>);	Read-in data via V24 Value2: Data type 0...10
@a29	OUTP ETX;	Output ETX via V24

Operator control functions: Main group a / subgroup 4		
@ code	CL 800 statement	Function
@a40 <Value>	CHAN (<Value>);	Select the channel No. for the screen display Value: Chan. No. 1...16

Section 11

-Appendix-

Overview:

- 11.1 **Alphabetic keyword index/CL 800 names**
- 11.2 **Terminology**
- 11.3 **Overview of functions implemented according to the software releases**
- 11.4 **Weighting of operators**
- 11.5 **G-group classification**

11 Appendix

11.1 Alphabetic keyword index/CL 800 names

Command	Function	Command overview	Command description	Object code
ABS	Absolute amount (abs. value generation)	6-16	5-79	10-23
ACAS	Axis No. of the current plane/master spindle No.	6-9	5-52	10-12
ACD	D-function, actual	6-9	5-53	10-12
ACG	G-function, actual	6-9	5-53	10-12
ACP	Current axis position	6-9	5-50	10-12
ACPM	Axis position, machine-related,	6-9	5-50	10-12
ACPW	Axis position, workpiece-related	6-9	5-49	10-12
ACSP	Actual spindle position	6-9	5-51	10-12
ACSS	Actual spindle speed	6-9	5-51	10-12
AGS	Active gear stage	6-11	5-62	10-16
ALNP	NC alarms	6-11	5-60	10-15
ALNPZ	NC alarm pointer	6-11	5-61	10-15
ALNZ	Cycle alarms	6-15	5-77	10-22
AND	And function	6-17	5-85	10-24
ANDB	And function bit	6-17	5-86	10-24
ANGLE	Angle from two vector components	6-16	5-83	10-24
ARC SIN	Arc sine	6-16	5-83	10-24
Addition	Tool assignment using addition	6-16	5-78	10-23
BEGIN	Start of the statement part	6-1	5-1	
BOOLEAN	Boolean data type	6-2	5-86	
CASE	Case branching	6-5	5-34	10-5
CENTRAL	Reference to a list, which is declared in the central variables	6-2	5-13	
CHAN	Selection of the channel No. for screen display	6-19	5-102	10-27
CHANNEL	NC or COM channel No.	6-1	5-4	
CLEAR	Clear R parameter	6-6	5-38	10-7
CLEAR BIT	Clear bit	6-17	5-87	10-25

11.1 Alphabetic keyword index/CL 800 names

Command	Function	Command overview	Command description	Object code
CLEAR MIB	Clear input buffer memory	6-6	5-39	10-8
CONST	Constants	6-2	5-8	
COS	Cosine	6-16	5-82	10-24
DEC	Decrement	6-16	5-81	10-23
DIN-Code	DIN commands	6-1	5-3	10-3
Division	Value assignment through division	6-16	5-78	10-23
END.	End of the statement part	6-1	5-1	
END;	End of a group statement or a statement block	6-1	5-1	10-3
ENDEXTERN	End of declaration	6-3	5-10	
ESS	Enable for softkey start	6-1	5-4	10-3
EXTERNAL	Definition of an external file	6-3	5-10	
FOR DOWNT0	Repeat statement for decrementing counter loop	6-4	5-26	10-6
FOR TO	Repeat statement for incrementing counter loop	6-4	5-27	10-6
GLOBAL	Reference to lists, in which global variables are declared	6-2	5-11	
GOTO	Unconditional jump	6-5	5-37	10-5
ID	Ident. designation	6-1	5-5	
IF INT THEN ELSE	IF INT THEN ELSE branching	6-5	5-33	10-6
IF THEN ELSE	IF THEN ELSE branching	6-5	5-29	10-6
INC	Increment	6-16	5-81	10-23
INP	Read-in data through V24	6-19	5-101	10-27
INTA	Axis-specific remaining travel delete via external input	6-18	5-97	10-26
INT SEC	Intersection computation	6-18	5-90	10-26
INTEGER	Integral data type	6-2		
INV LN	Exponential function	6-16	5-84	10-24
Kommentar	Comments in the program	6-1	5-2	10-3
LABEL	Label for jump destination	6-2	5-10	
LF	End of block	6-1	5-2	10-3
LN	Natural logarithm	6-16	5-84	10-24

Command	Function	Command overview	Command description	Object code
LOCAL	Local variable	6-2	5-8	
MDD	Drives machine data	6-7, 6-12	5-42, 5-64	10-9, 10-17
MDDBI	Drives machines data, bits	6-7, 6-12	5-42, 5-64	10-9, 10-17
MDDBY	Drives machines data, bytes	6-7, 6-12	5-42, 5-64	10-9, 10-17
MDN	NC machine data	6-7, 6-12	5-40, 5-62	10-9, 10-17
MDNBI	NC machine data, bits	6-7, 6-12	5-40, 5-63	10-9, 10-17
MDNBY	NC machine data, bytes	6-7, 6-12	5-40, 5-63	10-9, 10-17
MDP	PLC machine data	6-7, 6-12	5-41, 5-64	10-9, 10-17
MDPBI	PLC machine data, bits	6-7, 6-12	5-41, 5-64	10-9, 10-17
MDPBY	PLC machine data, bytes	6-7, 6-12	5-41, 5-64	10-9, 10-17
MDZ	Cycles machine data	6-7, 6-12	5-41, 5-63	10-9, 10-17
MDZBI	Cycles machine data, bits	6-7, 6-12	5-41, 5-63	10-9, 10-17
MDZBY	Cycles machine data, bytes	6-7, 6-12	5-41, 5-63	10-9, 10-17
MEAS M	Flying measurement (referred to machine zero)	6-18	5-96	10-26
MIB	Input buffer memory	6-6	5-39	10-8
Multiplik.	Value assignment through multiplication	6-16	5-78	10-23
NAND	Not and function	6-17	5-85	10-24
NANDB	Not and function bit	6-17	5-86	10-24
NOT	Negation	6-17	5-85	10-24
NOTB	Negation bit	6-17	5-86	10-24
OR	Or function	6-17	5-85	10-24
ORB	Or function bit	6-17	5-86	10-24
OUTP DATA	Output data via V24	6-19	5-100	10-27
OUTP ETX	Output EXT via V24	6-19	5-102	10-27
OUTP PARA	Parameter output via V24	6-19	5-100	10-27
OUTP ZOA	Output zero offsets via V24	6-19	5-99	10-27
PANG	Programmed angle	6-14	5-71	10-20
PAR	Transfer variable	6-2	5-7	
PATTERN	Pattern data type	6-2	11-8	10-25
PCDA	Programmed control word for digital axis drives	6-8, 6-14	5-49, 5-71	10-11, 10-20
PCDS	Programmed control word for digital spindle drives	6-8, 6-14	5-49, 5-72	10-11, 10-20

11.1 Alphabetic keyword index/CL 800 names

Command	Function	Command overview	Command description	Object code
PLCC	PLC counter	6-10	5-58	10-14
PLCDB	PLC data word, BCD	6-11, 6-15	5-59, 5-75	10-15, 10-21
PLCDBL	PLC data word, left	6-10, 6-14	5-56, 5-73	10-13, 10-21
PLCDBR	PLC data word, right	6-10, 6-14	5-56, 5-73	10-13, 10-21
PLCDF	PLC data word, fixed point	6-11, 6-15	5-58, 5-74	10-15, 10-21
PLCDG	PLC data word, floating point	6-11, 6-15	5-60, 5-76	10-15, 10-21
PLCF	PLC flag bit	6-9, 6-14	5-55, 5-72	10-13, 10-20
PLCFB	PLC flag byte	6-10, 6-14	5-56, 5-73	10-13, 10-21
PLCFW	PLC flag word	6-10, 6-14	5-57, 5-73	10-14, 10-21
PLCI	PLC input bit	6-9	5-55	10-13
PLCIB	PLC input byte	6-10	5-56	10-13
PLCIW	PLC input word	6-10	5-57	10-14
PLCPB	PLC peripheral byte	6-10	5-56	10-13
PLCPW	PLC peripheral word	6-10	5-57	10-14
PLCQ	PLC output bit	6-9	5-55	10-13
PLCQB	PLC output byte	6-10	5-56	10-13
PLCQW	PLC output word	6-10	5-57	10-14
PLCT	PLC timer	6-10	5-57	10-14
PLCW	PLC data word bit	6-9, 6-14	5-55, 5-72	10-13, 10-20
POINTER	Pointer variable	6-2	5-9	
PORT	Select V24 interface	6-19	5-99	10-27
POS MSYS	Input a position referred to the machine actual value system	6-17	5-88	10-26
PPCH	Current channel No. for program	6-9	5-55	10-12
PRAD	Programmed radius	6-14	5-70	10-20
PRAP	Programmed axis position	6-14	5-70	10-20
PREP CYC	Start preparation for cycles	6-18	5-93	10-26
PREP REF	Reference position	6-18	5-89	10-26
PRIP	Programmed interpolation parameter	6-14	5-71	10-20
PROGRAM	Definition of a program	6-1	5-3	10-3
PRSS	Programmed spindle speed	6-8, 6-14	5-47, 5-70	10-11, 10-20

Command	Function	Command overview	Command description	Object code
PRVC	Programmed cutting speed	6-8	5-47	10-11
PW	Password	6-1	5-66	
R<Var-Nr.>	R parameter number	6-2	11-8	10-3
REAL	Real data type	6-2	11-8	
RECALL PIC	Return jump to output menu	6-19	5-98	10-27
REL	Read-in inhibit via external input	6-18	5-97	10-26
REPEAT	Repeat statement with scan of the repeat condition at the end	6-4	5-14	10-6
RSDA	Axis status for digital drives	6-11	5-61	10-16
RSDD	Unit status for digital drives	6-11	5-62	10-16
RSDS	Spindle status for digital drives	6-11	5-61	10-16
SATC	Spindle acceleration time constant	6-15	5-77	10-22
SEN	NC setting data	6-7, 6-12	5-42, 5-65	10-10, 10-18
SENBI	NC setting data, bit	6-7, 6-12	5-42, 5-65	10-10, 10-18
SENB	NC setting data, byte	6-7, 6-12	5-42, 5-65	10-10, 10-18
SET BIT	Set bit	6-17	5-87	10-25
SEZ	Cycles setting data	6-7, 6-12	5-43, 5-65	10-10, 10-18
SEZBI	Cycles setting data, bits	6-7, 6-12	5-43, 5-65	10-10, 10-18
SEZBY	Cycles setting data, bytes	6-7, 6-12	5-43, 5-65	10-10, 10-18
SIN	Sine	6-16	5-82	10-24
SOB	Special bit	6-9	5-54	10-12
SQRT	Square root	6-16	5-80	10-23
SQRTS	Root of sum of squares	6-16	5-80	10-23
STOP DEC	Stop decoding until input buffer memory empty	6-18	5-93	10-26
STOP DECI	Stop decoding until input buffer memory empty at coordinate rotation	6-18	5-94	10-26
Subtraktion	Value assignment through subtraction	6-16	5-78	10-23
TAN	Tangent	6-16	5-82	10-24
TOAD	Tool offset, additive	6-13	5-66	10-18
TOS	Tool offset	6-8, 6-13	5-43, 5-66	10-10, 10-18
TRUNC	Integral part	6-16	5-81	10-23
VOP	Comparison operands		11-9	

Command	Function	Command overview	Command description	Object code
Var	Variable	6-2	11-8	
Var-Name	Definition of a variable, name	6-2	11-9	
WHILE	Repeat statement with scan of the repeat condition at the start	6-4	5-19	10-4
WHILE INT	Repeat statement with scan of an external input at the start	6-4	5-23	10-7
Wert	Value of a variable	6-4		
WRT PIC	Menu selection from the NC program	6-19	5-98	10-27
XCHG	Exchanging variable contents	6-6	5-38	10-7
XOR	Exclusive-or function	6-17	5-85	10-24
XORB	Exclusive-or function bit	6-17	5-86	10-24
ZOA	Settable zero offset (G54-G57)	6-8, 6-13	5-44, 5-67	10-11, 10-19
ZOADW	Settable coordinate rotation (G54-57)	6-8, 6-13	5-46, 5-68	10-11, 10-19
ZOD	DRF offset	6-8, 6-13	5-45, 5-68	10-11, 10-19
ZOE	External zero offset	6-8	5-45	10-11
ZOFA	Settable zero offset (G54-G57), additive	6-13	5-67	10-19
ZOFADW	Settable coordinate rotation (G54-G57), additive	6-13	5-69	10-19
ZOFPRDW	Progr. coordinate rotation (G58,G59), additive	6-13	5-70	10-11, 10-19
ZOPR	Programmable zero point offset (G58, G59)	6-8, 6-13	5-45, 5-67	10-11, 10-19
ZOPRDW	Programmable coordinate rotation (G58, G59)	6-8, 6-13	5-47, 5-69	10-11, 10-19
ZOPS	Preset offset	6-8, 6-13	5-46, 5-68	10-11, 10-19
ZOS	Sum of all offsets	6-8	5-46	10-11

11.2 Terminology

BOOLEAN

Boolean type variables are bits, which can only have the value 1 or 0.

<Const>

- Direct value (constant)
- Bit No. for PATTERN variable

<Const-Name>

refer to <Var name>

<Data name>

refer to <Var name>, however max. of 8 characters

<Data type>

Before a variable is first used, its data type must always be specified in the declaration part. The following basic types are in CL 800:

- INTEGER (integral numbers)
- REAL (real numbers)
- PATTERN (bit pattern)
- BOOLEAN (bit)

File type

Character sequence (IBF, MBF, SPF, RPF, ZOF)

File designation

- For IBF: Character sequence or pointer (text variable)
- For the other files: Number, constant or pointer (R parameter)

INTEGER

Integer variables can include positive or negative integral numbers. The maximum number of decades is eight. The value range is from 0 to +/- 99999999.

<List name>

refer to <Var name>, however a max. of 8 characters

Password

refer to Var name, however a max. of 8 characters

PATTERN

PATTERN type variables are binary numbers, which consist of eight individual bits. Each bit can only have the value 1 or 0.

Pattern type variables can thus be assigned a bit pattern in the range B00000000 to B11111111. A bit pattern is introduced with the character "B", e.g. bit pattern: B00010001.

REAL

Real number correspond, in CL 800, to normal decimal numbers in floating point, representation. The value range is from +/- 0.00000001 to +/- 99999999.
e.g. real number: -123.27

R<Var No.>

Designation for a numerical variable (R parameter). The R parameter number is specified in the "Var No." term.

System memory name

Max. of 5 upper case letters for designating a system memory.

<Var>

- Designation for a numerical variable (R parameter).
- Designation for a pointer variable in which the address of a variable is located. Pointer variables (pointers) are located in angled brackets '[<Var>]'.

<Var name>

Definition of a variable name. This provides the user with the possibility of assigning a symbolic name to an R parameter. The variable name consists of a maximum of 25 characters, and the first character must be a letter. From the second character onwards, either letters, digits or underline(_) can follow. Lower case letters can also be used.

If a variable name is longer than 8 characters, only the first 8 characters are evaluated to make a distinction.

"Vop"

The program run can be specifically influenced by scan conditions. Depending on the value of the programmed variables and the comparison operator ("Vop"), a decision is made as to whether the following program section is processed or skipped.

The following comparison operators are available:

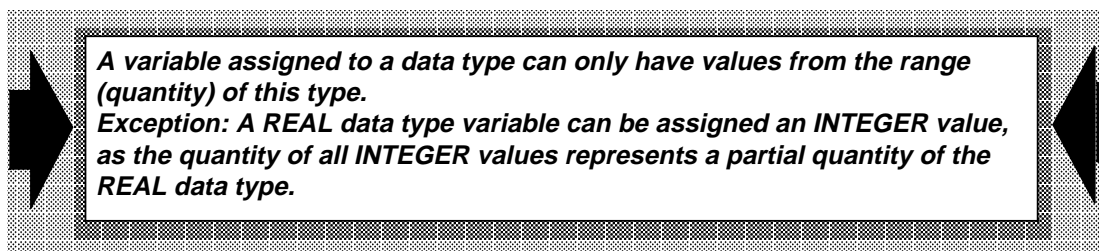
> greater than
 >= greater than or equal to
 < less than
 <= less than or equal to
 = equal
 <> unequal

<Value>

This is valid for both 'Const' and 'Var'

- Direct value (constant)
- Designation for a numerical variable (R parameter)
- Designation for a pointer variable, in which the address of a variable is located. Pointer variables (pointer) are located in angled brackets '[<Var>]'.

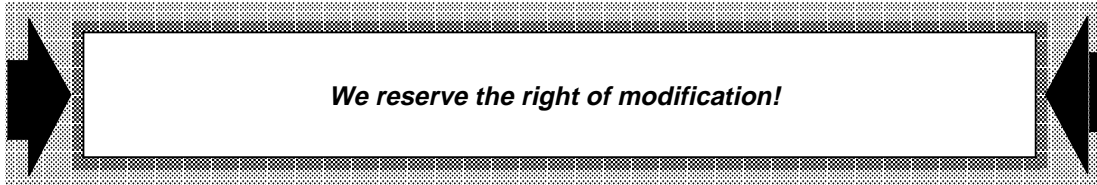
Note for using the data types:



11.3 Overview of functions implemented according to the software releases

Note:

The software release for SINUMERIK 810 specified in brackets is valid for controls which cannot be configured which were supplied before 3/88.



@ code	WS800		Control types												
	800A	800	805 *)	810	810 GA1	810 GA2	810 GA3	810G/ 820G GA2	810G/ 820G GA3	820 GA2	820 GA3	840 *)	850	880	
@00f								1	1						
@040 <Const> <R par 1> ... <R par n>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@041 <R par 1> <R par 2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@042 <Const> <R par 1> ... <R par n>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@043 <R par 1> <R par 2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@100 <Const> @100 <R par > 1)	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@111 <Var> <Value1><Const1> <Value2><Const2> : <Value n><Const n>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@12x <Var><Value><Const>	1	V 1.3		2	1	1	1	1	1	1	1		1	1	
@13x <Var><Value><Const>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@14x <Var><Value><Const>	1	V 2.0			2	1	1	1	1	2	1		2	1	
@151 <Var><Value><Const>	1	V 2.0		2 (05)	2	1	1	1	1	1	1		1	1	
@161 <Var><Value><Const>	1	V 2.0		2 (05)	2	1	1	1	1	1	1		2	1	
@18x <Value1><Value2><Const>								1	1						
@19x <Value2><Const>								1	1						
@200 <Var>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@201 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@202 <Var1><Var2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	

1) No pointer possible, only <Const> can be specified at the CL 800 level

*) Function in preparation

11.3 Overview of functions implemented according to the software releases

@ code	WS800		Control types												
	800A	800	805 *)	810	810 GA1	810 GA2	810 GA3	810G/ 820G GA2	810G/ 820G GA3	820 GA2	820 GA3	840 *)	850	880	
@203 <Var1><Var2><Const>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@210 <Value3><Value4>					3	2	1	1	1	2	1		3	3	
@211 <Var><Value1>	1	V 2.0		2	2	1	1	1	1	1	1		3	3	
@212 <Value1><Value>	1	V 2.0		2	3	1	1	1	1	1	1		3	3	
@300 <Var><Value1>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@301 <Var><Value1>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@302 <Var><Value1><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@303 <Var><Value1><Value2>	1	V 2.0											3	3	
@304 <Var><Value1><Value2>	1	V 2.0											3	3	
@305 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	
@306 <Var><Value1>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@307 <Var><Value1>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@308 <Var><Value1><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@309 <Var><Value1>	1	V 2.0			3	2	1			2	1				
@30a <Var><Value1><Value2>	1	V 2.0			3	2	1			2	1				
@30b <Var><Value1><Value2><Value3>	1	V 2.0			3	2	1			2	1				
@310 <Var><Value1>	1	V 1.3		2	1	1	1	1	1	1	1		1	1	
@311 <Var><Value1>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@312 <Var><Value1><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@313 <Var><Value1><Value2>	1	V 2.0											3	3	
@314 <Var><Value1><Value2>	1	V 2.0											3	3	
@315 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	
@320 <Var><Value1><Value2> <Value3>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@330 <Var><Value1><Value2><Value3>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@331 <Var><Value1><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@332 <Var><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@333 <Var><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@334 <Var><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	

*) Function in preparation

11.3 Overview of functions implemented according to the software releases

@ code	WS800			Control types											
	800A	800	805 *)	810	810 GA1	810 GA2	810 GA3	810G/ 820G GA2	810G/ 820G GA3	820 GA2	820 GA3	840 *)	850	880	
@336 <Var><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@337 <Var><Value1><Value2><Value3>	1	V 2.0		2	2	1	1	1	1	1	1		3	3	
@338 <Var><Value1><Value2><Value3>	1	V 2.0		2	2	1	1	1	1	1	1		3	3	
@342<Var><Value1> <Value3>					3	2	1			2	1				
@345<Var><Value1> <Value2>	1	V 1.3		2 (05)	2	1	1	1	1	1	1		1	1	
@34b <Var><Value1><Value2><Value3>	1	V 2.0			3	2	1			2	1				
@34c <Var><Value1><Value2><Value3>	1	V 2.0			3	2	1			2	1				
@360 <Var><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@361 <Var><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@362 <Var><Value2>	1	V 2.1											3	3	
@363 <Var><Value2>	1	V 2.0		2 (05)	2	1	1	1	1	1	1		2	1	
@364 <Var><Value2>	1	V 2.0		2 (05)	2	1	1	1	1	1	1		2	2	
@367 <Var><Value1>	1	V 2.0		2	2	1	1	1	1	1	1		2	1	
@36a <Var><Value1>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@36b <Var><Value1><Value3>	1	V 2.0		2	2	1	1	1	1	1	1		1	1	
@371 <Var><Value1><Value3>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@372 <Var>													3	3	
@380 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	
@381 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	
@382 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	
@383 <Var><Value1><Value2><Value3><Val.4>	1	V 2.0											3	3	
@390 <Var><Value1><Value2>	1	V 2.0											3	3	
@391 <Var><Value1><Value2>	1	V 2.0											3	3	
@392 <Var><Value1><Value2>													3	3	
@393 <Var><Value1><Value2>	1	V 2.0											3	3	
@394 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	
@395 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	
@3a0 <Var><Value1><Value2><Value3>	1	V 2.0											3	3	

*) Function in preparation

11.3 Overview of functions implemented according to the software releases

@ code	WS800		Control types													
	800A	800	805 *)	810	810 GA1	810 GA2	810 GA3	810G/ 820G GA2	810G/ 820G GA3	820 GA2	820 GA3	840 *)	850	880		
@3a1 <Var><Value1><Value2><Value3>	1	V 2.0												3	3	
@3a2 <Var><Value1><Value2><Value3>	1	V 2.1												3	3	
@3a3 <Var><Value1><Value2><Value3>	1	V 2.0												3	3	
@3a4 <Var><Value1><Value2>	1	V 2.1												3	3	
@3a5 <Var><Value1><Value2>	1	V 2.1												3	3	
@3b0 <Var><Value1><Value2><Value3> <Value4><Value5>														3	3	
@3b1 <Var><Value1><Value2><Value3> <Value4><Value5>														3	3	
@3b2 <Var><Value1><Value2><Value3>														3	3	
@3c0 <Var>	1	V 2.0												3	3	
@3d0 <Var>	1	V 2.0												3	3	
@3e1 <Var><Value1><Value2><Value3>	1	V 2.0		3	2	1				2	1					
@3e2 <Var><Value1><Value2><Value3>	1	V 2.0		3	2	1				2	1					
@3e3 <Var><Value1><Value2><Value3>	1	V 2.0		3	2	1				2	1					
@3e4 <Var><Value1>				3	2	1				2	1					
@400 <Value1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@401 <Value1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@402 <Value1><Value2><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@403 <Value1><Value2><Value>	1	V 2.0												3	3	
@404 <Value1><Value2><Value>	1	V 2.0												3	3	
@405 <Value1><Value2><Value3><Value>	1	V 2.0												3	3	
@406 <Value1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@407 <Value1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@408 <Value1><Value2><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@409 <Value1><Value>	1	V 2.0			3	2	1			2	1					
@40a <Value1><Value2><Value>	1	V 2.0			3	2	1			2	1					
@40b <Value1><Value2><Value3><Value>	1	V 2.0			3	2	1			2	1					
@410 <Value1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	

*) Function in preparation

11.3 Overview of functions implemented according to the software releases

@ code	WS800			Control types												
	800A	800	805 *)	810	810 GA1	810 GA2	810 GA3	810G/ 820G GA2	810G/ 820G GA3	820 GA2	820 GA3	840 *)	850	880		
@411 <Value1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@412 <Value1><Value2><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@413 <Value1><Value2><Value>	1	V 2.0											3	3		
@414 <Value1><Value2><Value>	1	V 2.0											3	3		
@415 <Value1><Value2><Value3><Value>	1	V 2.0											3	3		
@420 <Value1><Value2><Value3><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@423 <Value1><Value2><Value3><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@430 <Value1><Value2><Value3><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@431 <Value1><Value2><Value3><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@432 <Value1><Value2><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@434 <Value2><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@435 <Value2><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@437 <Value1><Value2><Value3><Value>	1	V 2.0		2	2	1	1	1	1	1	1		3	3		
@438 <Value1><Value2><Value3><Value>	1	V 2.0		2	2	1	1	1	1	1	1		3	3		
@439 <Value1><Value2><Value3><Value>	1	V 2.0		2	2	1	1	1	1	1	1		3	3		
@43a <Value1><Value2><Value3><Value>	1	V 2.0		2	2	1	1	1	1	1	1		3	3		
@440 <Wert3><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1		
@442 <Wert3><Value>					3	2	1			2	1					
@446 <Value>	1	V 2.0		1 (04)	2	1	1	1	1	1	1		1	1		
@447 <Value>	1	V 2.0		1 (04)	2	1	1	1	1	1	1		1	1		
@448 <Wert3><Value>	1	V 2.0			3	2	1			2	1		3	3		
@44b <Value1><Value2><Value3><Value>	1	V 2.0			3	2	1			2	1					
@44c <Value1><Value2><Value3><Value>	1	V 2.0			3	2	1			2	1					
@482 <Value1><Value2><Value3><Value>	1	V 2.0											3	3		
@483 <Wert1><Wert2><Wert3><Wert4><Wert>	1	V 2.0											3	3		
@493 <Value1><Value2><Value>	1	V 2.0											3	3		
@494 <Value1><Value2><Value3><Value>	1	V 2.0											3	3		
@495 <Value1><Value2><Value3><Value>	1	V 2.0											3	3		

*) Function in preparation

11.3 Overview of functions implemented according to the software releases

@ code	WS800		Control types													
	800A	800	805 *)	810	810 GA1	810 GA2	810 GA3	810G/ 820G GA2	810G/ 820G GA3	820 GA2	820 GA3	840 *)	850	880		
@4a3 <Value1><Value2><Value3><Value4>	1	V 2.0												3	3	
@4b0 <Value1><Value2><Value3><Value4> <Value5><Value6>														3	3	
@4b1 <Value1><Value2><Value3><Value4> <Value5><Value6>														3	3	
@4b2 <Value1><Value2><Value3><Value4>														3	3	
@4c0 <Value>	1	V 2.0		3	2	1	1	1	1	2	1			3	3	
@4e1 <Value1><Value2><Value3>				3	2	1				2	1					
@610 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@613 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@614 <Var><Value1><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@620 <Var>	1	V 2.0		2 (05)	2	1	1	1	1	1	1			2	1	
@621 <Var>	1	V 2.0		2 (05)	2	1	1	1	1	1	1			2	1	
@622 <Var>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@630 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@631 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@632 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@634 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@637 <Var><Value1><Value2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@640 <Var><Value>	1	V 1.3		1	2	1	1	1	1	1	1			1	1	
@641 <Var><Value>	1	V 2.0												2	1	
@650 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@651 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@652 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@653 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@654 <Var><Value>	1	V 1.3		1	2	1	1	1	1	1	1			1	1	
@655 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@656 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	
@657 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1			1	1	

*) Function in preparation

11.3 Overview of functions implemented according to the software releases

@ code	WS800			Control types											
	800A	800	805 *)	810	810 GA1	810 GA2	810 GA3	810G/ 820G GA2	810G/ 820G GA3	820 GA2	820 GA3	840 *)	850	880	
@658 <Var><Var1><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@659 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@660 <Var><Const>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@661 <Var><Const>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@67x <Var1><Var2><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@706	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@710 <Var1><Var2>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@711 <Var1><Var2><Var3>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@713 <Var>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@714	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@715				2	2	1	1	1	1	1	1		3	3	
@720 <Var><Value>	1	V 1.3		1 (04)	2	1	1	1	1	1	1		1	1	
@730 <Value1><Value2>										1	1				
@736 <Value1><Value2><Value3>										1	1				
@a15 <Value4><Value5>										1	1				
@a1b										1	1				
@a20 <Value>	1	V 2.0								1	1				
@a25 <Value1>										1	1				
@a26 <Value2><Value3><Value4>										1	1				
@a27 <Value1><Value3><Value4>	1	V 2.0								1	1				
@a28 <Value2>	1	V 2.0								1	1				
@a29										1	1				
@a40 <Value>	1	V 2.0								1	1				

*) Function in preparation

11.4 Weighting of operators

The following list indicates the weighting of logical and arithmetic operations:

Weighting	Source code	Function
1.	()	Brackets
2.	/	Division
	*	Multiplication
3.	-	Subtraction
	+	Addition
4.	AND	And
	NAND	Not and
5.	OR	Or
	XOR	Exclusive-or
6.	=	Equal to
	>	Greater than
	<	Less than
	>=	Greater than or equal to
	<=	Less than or equal to
7.	< >	Not equal to
	ANDB	And bit
	NANDB	Not-and bit
	ORB	Or bit
	XORB	Exclusive-or bit

The weighting decreases going down the table. Using suitable bracketing, a lower-priority command can be processed before a higher-priority command.

11.5 G-group classification

The following table indicates the classification of the G-functions according to G-groups for the ACG statement (or object code: @36b):

Internal G-group classification for @36b																	
Internal G-group	G-functions																
0	00	01	10	11	02	03	33	34	35	06	12	13	05	07			
1	09																
2	17	18	19	16													
3	40	41	42														
4	53																
5	54	55	56	57													
6	04	25	26	58	59	92	74										
7	60	63	64	62													
8	70	71															
9	80	81	82	83	84	85	86	87	88	89							
10	90	91	68														
11	94	95	96	97													
12	147	247	347	148	248	348	48	110	111								
13	50	51															

850/880											
14	150	151	152	153	154	155	156	157	158	159	
15	130	131									
16	230	231									
17	330	331									
18	930	931	932	933	934						

810G/820G					
27					
180	181	182	183	184	185
65	66	67			
196	197				
163					

G-groups 14-18 only preliminary for 810G/820G!

840											
14											
15											
16											
17											
18											

805					