NCT2xx
NCT3xx
Machine Tool Controls

PLC Programming

From software version n.14.9

Manufacturer and developer: **NCT Ipari Elektronikai kft**.
H1148 Budapest Fogarasi út 7
⊠ Address: 1631 Bp. pf.26
☎ Phone: (+36 1) 467 63 00
☎ Fax:(+36 1) 467 63 09
E-mail: nct@nct.hu
Home page: www.nct.hu

Contents

1 The PLC Programming Language. 8
- 1.1 Sampling and Handling of Inputs and Outputs. . . . 9
- 1.2 The Order of Execution of a PLC Program. . . . 9
- 1.3 Editing PLC Programs. . . . 10

2 Memory Used by the PLC Program. 11
- 2.1 Addressing of Double Words (DWORD). . . . 13
- 2.2 Indexed Addressing of Double Words (DWORD) with Operator “,”. . . . 13
- 2.3 Direct Addressing of Bits with Operator “.”. . . . 15
- 2.4 Indirect Addressing of Bits with Operator “:”. . . . 17
- 2.5 Indexed Addressing of Bits with Operator “,”. . . . 18
- 2.6 Addressing Floating-point Numbers (double). . . . 19
- 2.7 Indexed Addressing of Floating-point Numbers (double) with Operator “,”. . . . 20

3 Modules of the PLC Program. 21
- 3.1 The Main Program. . . . 21
- 3.2 The Int0 Module. . . . 21
- 3.3 Updating the PLC Memory. . . . 21

4 Data Managed by the PLC Program. 23
- 4.1 Managing Bit-type Data. . . . 23
- 4.2 Query of Rising Edge of Memory Bits with Operator “@”. . . . 23
- 4.3 Query of Falling Edge of Memory Bits with Operator “%”. . . . 23
- 4.4 Immediate Query of Inputs, Immediate Issue of Outputs with Operator “!”. . . . 24
- 4.5 Definition of Decimal, Signed Number with Operator “#”. . . . 24
- 4.6 Definition of Hexadecimal Number With Operator “#$”. . . . 25
- 4.7 Definition of BCD Number without Sign with Operator “#$”. . . . 25
- 4.8 Definition of Floating-point Number with Operator “*”. . . . 25
- 4.9 Double Precision Representation of Floating-point Numbers in IEEE754 Standard. . . 26

5 Status Bits Set by PLC Instructions. 28
- 5.1 FL_ER Error Flag. . . . 28
- 5.2 FL_UF Underflow Flag. . . . 28
- 5.3 FL_OF Overflow Flag. . . . 28
- 5.4 FL_CY Carry Flag. . . . 28
- 5.5 FL_GT Greater than Flag. . . . 28
- 5.6 FL_EQ Equal Flag. . . . 29
- 5.7 FL_LT Lower than Flag. . . . 29

6 Instructions of the PLC Program. 30
- 6.1 Bit Operation Instructions. . . . 30
  - 6.1.1 Open Contact: Input of a Memory Bit. . . . 30
  - 6.1.2 Open Contact: Query of a Double Word. . . . 31
  - 6.1.3 Closed Contact: Input Not of a Memory Bit. . . . 32
  - 6.1.4 Closed Contact: Query Not of a Double Word. . . . 33

6.1.5 Relay Coil: Output to a Memory Bit. 34
6.1.6 Negated Relay Coil: Output Not to a Memory Bit. 35
6.1.7 Setting a Memory Bit: the SET Instruction. 36
6.1.8 Resetting a Memory Bit: the RST Instruction. 36
6.1.9 Pulse Generation for Rising Edge: the DIFU Instruction. 37
6.1.10 Pulse Generation for Falling Edge: the DIFD Instruction. 38
6.1.11 Bit Operation Instructions and the Operator "!" in the Two Modules. 39
6.2 Basic Rules of Connections of Ladder Network. 41
6.2.1 Connecting elements. 42
6.2.2 Commenting the Logic Sectors of a Ladder Diagram: the SEC Instruction. 43
6.3 Data Movement Instructions. 44
6.3.1 Movement of Double Words: the MOV and the MVN Instructions. 45
6.3.2 Movement of Floating-point data: the MOVF Instruction. 46
6.4 Timers. 47
6.4.1 On-delay Timer: TOND. 48
6.4.2 Off-delay Timer: TOFFD. 49
6.4.3 Programmable Pulse Width: TPULSE. 50
6.5 Counters. 51
6.5.1 Simple Counter CNT.. 52
6.5.2 Up-down Counter CNTR. 54
6.6 Rotation Control Instruction: ROT. 56
6.7 Data Shift Instructions.. 60
6.7.1 Shift Register: SHTR.. 60
6.7.2 Shift Instructions: ASHL, ASHR. 61
6.7.3 Rotate Instructions: ARTL, ARTR. 63
6.8 Logic Instructions. 66
6.8.1 Complement Instruction: NEG. 66
6.8.2 Two-operand Instructions: AND, OR, XOR.. 67
6.9 Integer Arithmetic Instructions. 70
6.9.1 Signed, Integer Addition, without Carry: ADD.. 71
6.9.2 Signed, Integer Subtraction, without Carry: SUB. 72
6.9.3 Signed, Integer Multiplication: MUL. 73
6.9.4 Signed, Integer Division: DIV.. 74
6.10 Floating-point Mathematical Instructions. 75
6.10.1 Floating-point Addition: +F. 76
6.10.2 Floating-point Subtraction: -F. 77
6.10.3 Floating-point Multiplication: *F. 78
6.10.4 Floating-point Division: /F. 79
6.10.5 Exponential Power: PWR. 80
6.10.6 Square Root: SQRT.. 81
6.10.7 Sine: SIN. 82
6.10.8 Cosine: COS. 83
6.10.9 Tangent: TAN. 84
6.10.10 Arc Sine: ASIN. 85
6.10.11 Arc Cosine: ACOS. 86
6.10.12 Arc Tangent: ATAN. 87
6.10.13 Natural Exponent: EXP.. 88
6.10.14 Natural Logarithm: LOG.. 89
6.11 Conversion Instructions.. 90

6.11.1 BCD to Binary Conversion: BIN. 91
6.11.2 Binary to BCD Conversion: BCD. 92
6.11.3 Signed Integer to Floating-point Conversion: FLT. 93
6.11.4 Floating-point to Signed Integer Conversion: FIX.. 94
6.11.5 Radian to Degree Conversion: DEG. 95
6.11.6 Degree to Radian Conversion: RAD. 96
6.12 Comparison Instructions. 97
6.12.1 The CMP and the FCMP Instructions. 97
6.12.2 Contact Type Comparison Instructions. 98
6.13 Messages Sent from the PLC Program. 100
6.13.1 Instructions Sending Messages: MSG, MSGF, ALR, ALRF, REM, REMF. . . 102
6.14 Program Control Instructions.. 106
6.14.1 End of Module Instruction: END.. 106
6.14.2 Conditional Branch: the JMP and JME Instructions. 107
6.14.3 Subroutine Call: the SBS, SBN and RET instructions.. 108
6.15 Axis Control Instruction: MOVCMD. 110
6.16 Read and Write of Common Macro Variables. 114
6.16.1 Read of Common Macro Variables: the MACR Instruction. 114
6.16.2 Write of Common Macro Variables: the MACW Instruction. 115
6.17 Query of the Internal Variables of the NC: the SCP Instruction. 116
6.18 Reading and Writing NC Memory Arrays.. 123
6.18.1 NC Memory Array Read: the MR Instruction. 123
6.18.2 NC Memory Array Write: the MW Instruction. 124
6.19 Transferring Data between Non-volatile Memory and PLC. 127
6.19.1 PLC Data Read from Non-volatile Memory. 127
6.19.2 PLC Data Write to Non-volatile Memory. 129
6.20 Reading and Writing Parameters from the PLC Program. 131
6.20.1 Reading the DWORD-type Parameters. 131
6.20.2 Reading the Double-type Parameters.. 133
6.20.3 Writing the DWORD-type Parameters. 135
6.20.4 Writing Double-type Parameters. 137
6.21 Assigning a Program for Execution.. 139
6.21.1 Assigning a Program Specified with its Program Number for Automatic Execution 139
6.22 Network Communication Instructions.. 140
6.22.1 Opening the Network Connection. 140
6.22.2 Closing the Network Connection.. 141
6.22.3 Receiving the Network Data Packet. 142
6.22.4 Sending the Network Data Packet. 143
6.23 Opening a Window in the Screen of the Control.. 145
6.24 Dumping the Drive Data. 149
6.25 Writing the Position Data. 151
6.26 Reading and Writing Data of Tool Management Table.. 153
6.26.1 The Tool Management Table.. 153
6.26.2 The Cartridge Management Table. 158
6.26.3 The Tool Pattern Table. 160
6.26.4 Exchange of Data Numbers in Cartridge Table. 164
6.26.5 Search of Empty Pot.. 165
6.26.6 Register New Tool Data in Tool Management Table. 167

6.26.7 Writing Each Tool Management Data of a Tool. 171
6.26.8 Reading Each Tool Management Data of a Tool. 176
6.26.9 Deletion of All Tool Management Data of a Tool. 179
6.26.10 Writing a Tool Management Data of a Tool. 180
6.26.11 Reading a Tool Management Data of a Tool.. 183
6.26.12 Searching a Tool by User Data. 185
6.27 Writing and Reading the Data of the Pallet Management Table.. 188
6.27.1 The Pallet Management Table.. 188
6.27.2 Data Interchange Between Two Different Places of Two Different Pallet Magazines 193
6.27.3 Rewriting the Pallet Data.. 195
6.27.4 Reading the Data of the Pallet.. 198
6.27.5 Rewriting a Pallet Management Data of the Pallet. 200
6.27.6 Reading a Pallet Management Data of the Pallet. 202
6.27.7 Searching a pallet by its data number value. 204
6.27.8 Reading out the pallet magazine's values of given data number.. 206
6.27.9 Assigning a program accessible by its pallet identifier for automatic execution 208
6.28 Mailbox Communication between the PLC program and an Arbitrary Ethercat Device 211
6.28.1 Reading the Data of the EtherCAT Mailbox.. 211
6.28.2 Writing the Data of the EtherCAT Mailbox. 218
6.29 Codes of Execution of MR, MW Instructions.. 224

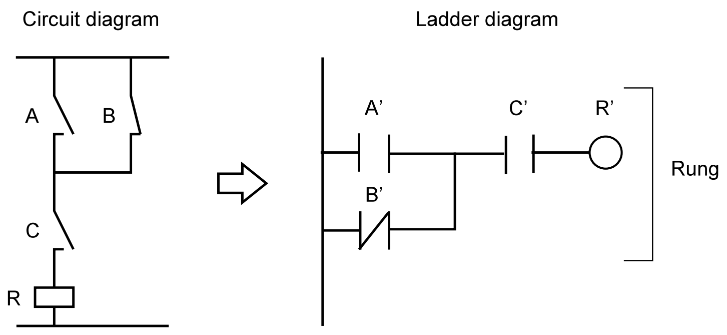
7 Communication between the PLC Program and the NC. 228
7.1 NCT Machine Control Panels.. 230
7.2 NCT Handwheels. 237
7.3 Two-state, 24V Interface In- and Outputs.. 240
7.4 In- and Outputs of NCT Probe Interface Cards.. 242
7.5 NCT Sensor Inputs. 244
7.6 NCT Analog Inputs.. 246
7.7 In- and Outputs of EtherCAT NCT Drives.. 247
7.8 Encoder Receiver and Analog/Stepping Motor/CAN Drive Interface Cards. 256
7.9 Function Buttons Accessible from the PLC. 262
7.10 Position Switches. 263
7.11 Access to Parameter Group PLC Constants. 265
7.12 Common Variables.. 269
7.12.1 Bit-type Common Variables. 269
7.12.2 DWORD-type Common Variables. 275
7.13 Axis Control Variables. 277
7.13.1 Bit-type Axis Control Variables. 277
7.14 Spindle Control Variables.. 294
7.14.1 Bit-type Spindle Control Variables.. 294
7.14.2 DWORD-type Spindle Control Variables.. 306
7.14.3 Double-type Spindle Control Variables. 309
7.15 Channel Control Variables. 310
7.15.1 Bit-Type Channel Control Variables.. 310
7.15.2 DWORD-type Channel Control Variables. 343
7.15.3 Double-type Channel Control Variables.. 355

Index in Alphabetical Order 358

19.11.28

1 The PLC Programming Language

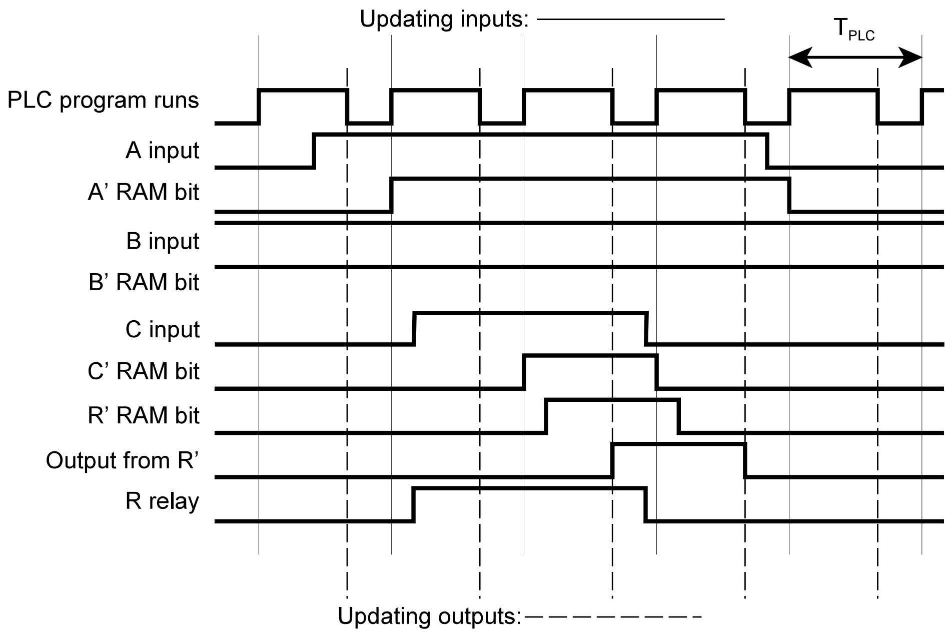
The PLC programming language has **ladder diagram** format.



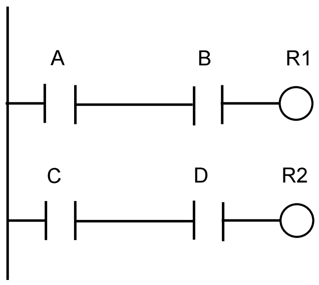
The ladder diagram is an approximate form of **relay circuit diagram** which is used in control technique.
The attached figure shows an example for representing logic circuit in ladder diagram: A and C are normally open and B is closed contacts, R is a relay.
The wires (logic lines) start from reference wire on the left side of ladder diagram followed by the contacts. They can be controlled by inputs/outputs, or they can be the open or closed contacts belonging to internal auxiliary relays, holding relays, timing relays. On the right end of logic line there are “coils” of outputs, timers, counters or instructions.
Rung is the name of contacts and wires which belong to an output.
Important rule, that each output, relay, timer, counter, etc. can appear only once in ladder diagram and the associated program. The contacts of these devices can be used as many times as required.
There are significant differences between the function of hardware connected logic circuits and PLC ladder programs executed by software..

1.1 Sampling and Handling of Inputs and Outputs

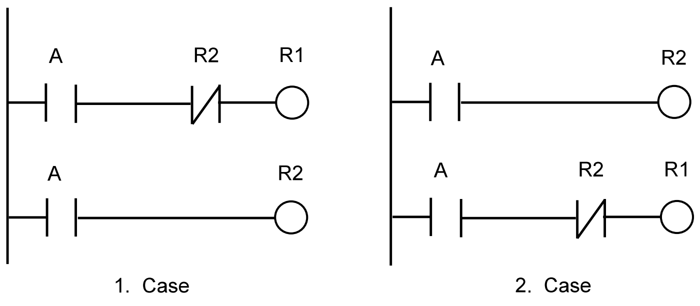
The PLC program runs by T_{PLC} intervals, where $\mathbf{T_{PLC}}$ is called as **sampling interval**.
Stay at the example of the previous figure. The PLC program **samples** from A, B and C signals (PLC inputs) and saves into the memory at addresses A', B' and C' before starting to execute the instructions.
After then PLC **program starts running**, calculates the R' value from A', B', C' values. The value of R' can be saved in memory at address R' after the execution of instruction.
After the PLC program ran to the end, the **outputs are updated** from memory, so the value of R' is written from the RAM to the output.
On the other hand, the circuit realized in hardware responds for the changes immediately. It means that the signals of A, B, or C contacts can activate R relay promptly.



1.2 The Order of Execution of a PLC Program



The ladder diagram PLC program, as any other sequence program written for a computer, is executed in the sequence of rungs.
In the attached figure, PLC sets the value of R1 relay depending on the state of contacts A and B (1. Rung) first, then sets the value of R2 relay depending on the state of C and D contacts. The execution always follows the order of rungs.
In the hardware built by relays, there is no operation precedence, every relay works about in the same time.
The ladder diagram does not correspond to the wired relays because of this feature. Let us take the following example.



– **In case of wired relays**: If the two cases in the figure above are realized by wired relays, then each case works the same way. If contact “A” closes, R1 and R2 relays will be switched on for a moment, then R1 will be switched off after R2 closes.

– **In case of ladder diagram**:

In the 1st case contact “A” closes, R1 is switched on, because R2 is not switched on. Then R2 is also switched on. R1 relay is switched off in the following PLC cycle, after T_{PLC} interval, because R2 has already operated. So R1 and R2 are also switched on for T_{PLC} interval.

In the 2nd case, if contact “A” closes, R2 is switched on, therefore R1 relay is not switched on, because contact R2 is already open.

The above example shows that hard wired circuits must be converted into ladder PLC program, after some consideration.

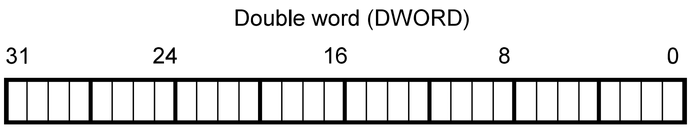
In the PLC program, it is possible to change the operation sequence with conditional subroutine call or jump.

1.3 Editing PLC Programs

Ladder diagram based PLC program can be edited graphically, therefore you have to use a special software developed for this purpose. The description of PLC program editor is not subject of this manual.

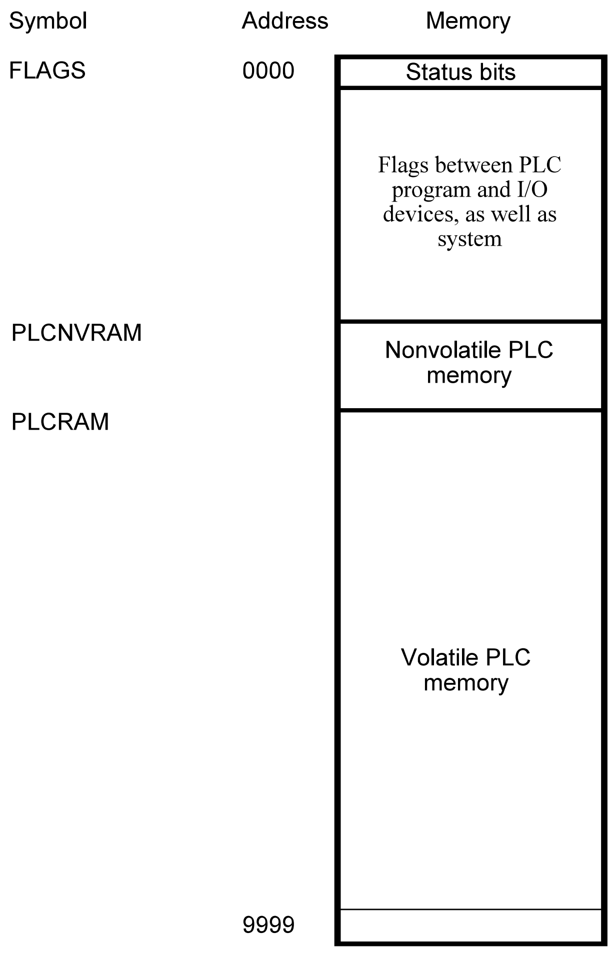
2 Memory Used by the PLC Program

Memory used by PLC program is a 10000 double word long coherent storage space. Double word (DWORD) is a 32 bit memory unit.



PLC storage space distributed to 4 main parts:
– Status bits: handled by the instructions of PLC program, mapped at address 0000, with the symbol FLAGS,
– Memory space for communication between PLC program and output, input hardware devices, as well as the system,
– Users' memory of the PLC program to be backed up after switch off, with the symbolic name PLCNVRAM as the starting address of nonvolatile variables,
– Users' memory of the PLC program, with the symbolic name PLCRAM as the starting address of volatile variables.
The limits of each parts title change by types.

Memory map of PLC program



PLC programmer has to map every variables of instructions in this storage space. For example, if a timer is necessary in PLC program, the variable of timer, which counts the passing of time, must be declared in this memory.
The meanings of status bits are described in this manual in another chapter.
The flags and registers of storage space which communicate between PLC program and I/O devices as well as between PLC program and system, are described later.
The basic unit of memory is double word (DWORD, 32 bit). Addressing smaller units, like word (WORD, 16 bit) or byte (8 bit), is not possible. Every part of memory is available by bits.
PLC program provides wide possibilities for symbolic access to memory. Every part of memory which can be accessed by addressing, also can be accessed symbolically.

2.1 Addressing of Double Words (DWORD)

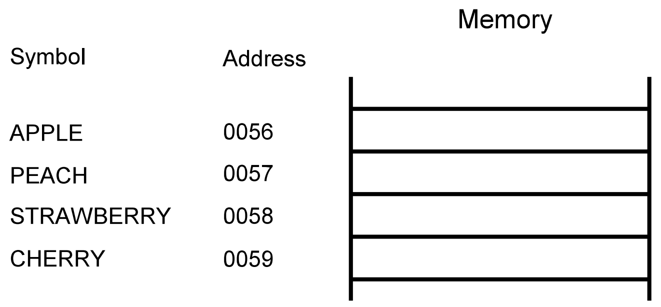
You can address the arbitrary double word of memory also by

numeral and

symbol.

You have to specify an address by 4 decimal numbers with **leading zeros** when referring to it by a numeral.

You have to declare the symbol in symbol storage before referring an address symbolically.



In the above example you can refer to memory space 56 by writing also

0056 or

APPLE.

2.2 Indexed Addressing of Double Words (DWORD) with Operator “,”

Indexed address contains two parts:

base address and

offset.

The two parts are separated by operator

, (comma) .

The address is calculated by adding to the base address the offset value:

address=base address+offset

The rules of addressing of double word are valid for **base address**:

defined by 4 decimal numbers, or

symbol.

Offset value can be defined by

directly: with numeral or with constant symbol, or

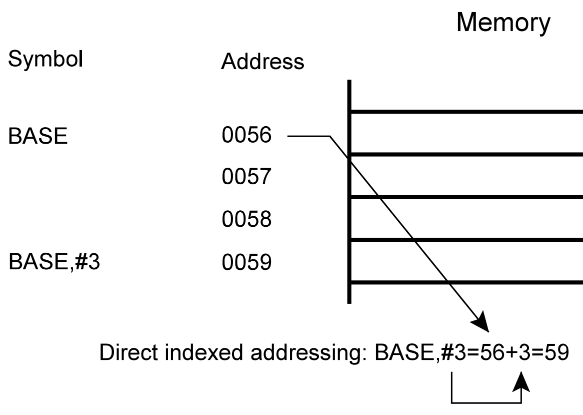
indirectly: by register reference.

– In case of defining **direct** offset you have to use operator **“#”** entering decimal number. If you want to offset address BASE by 3 than then write

0056,#3, or

BASE,#3, or

if BASE symbol is declared to address 0056.



You get the same result, if you declare BIAS symbol as constant symbol in symbol chart: BIAS #3. This time you have to write

BASE,BIAS

reference in program.

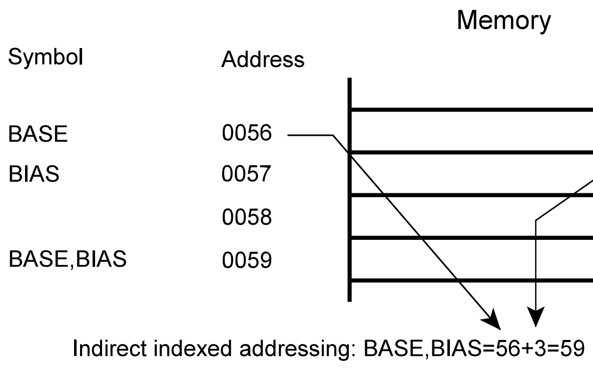
– In case of reference **indirect** offset (through register) you have to write the address of register which contains offset value after "," (comma) operator.

The addressing rules are valid for address which contains offset: you can define by number and also by symbol.

If you declare a BIAS register for segment address BASE, you can define an address by the reference:

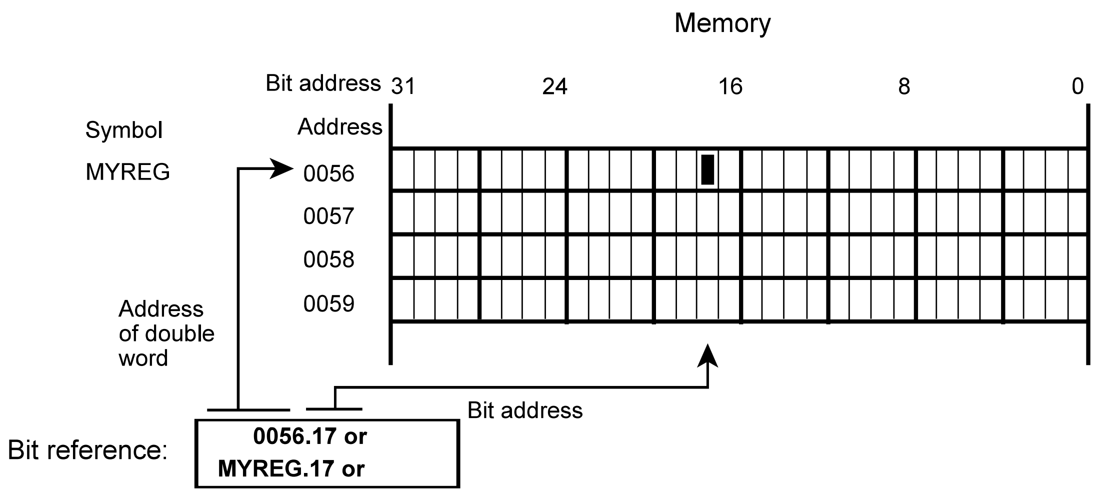
BASE,BIAS or
0056,0057.

Where symbol BASE is declared to address 0056, and symbol BIAS is declared to address 0057. Offset value is taken from register at address BIAS (0057).



2.3 Direct Addressing of Bits with Operator "."

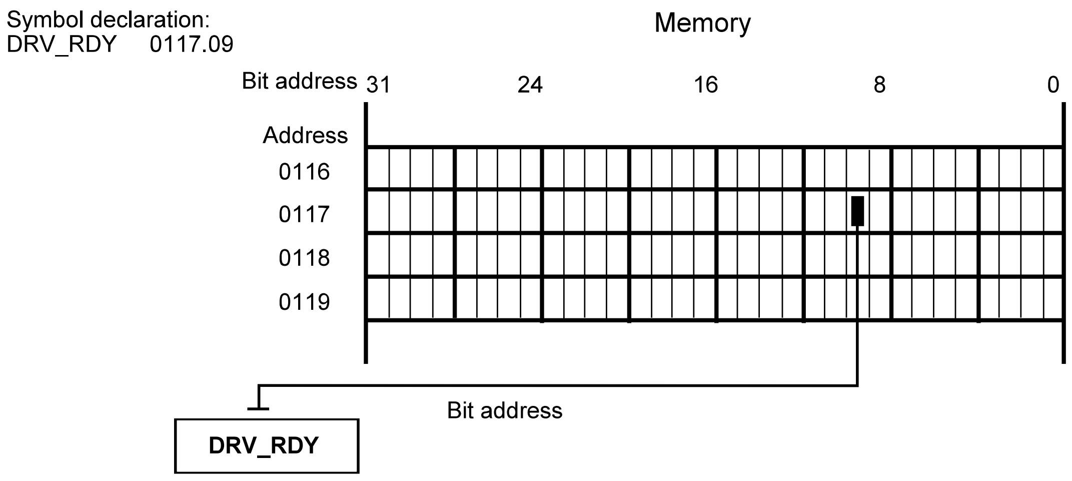
You can address any bit of PLC memory space. You can refer to bits by using "." (dot) operator or by using a symbol declared to that bit.
When using **"." operator**, bit address contains two parts:
a **double word address** and
a **bit address** inside the double word
The two parts are separated by
. (dot)
operator from each other.
You can refer to a double word address by
4 decimal numbers,
or also by symbol.
You can refer to a bit address by a value after the operator ".". You always have to define two numbers after "." , so you also have to write leading zeros. The range of a bit address is:
00 ... 31
For example, if you want to refer the 17th bit of the double word at address 0056, then you also can use the reference
0056.17 or
MYREG.17
Where MYREG is declared beforehand at address 0056.



If you have a declared symbol for a bit address, then you can refer directly to bit by **symbol**. For example, if you have declared DRV_RDY symbol to bit address 0117.09, then you can reach the bit 0117.09 directly by

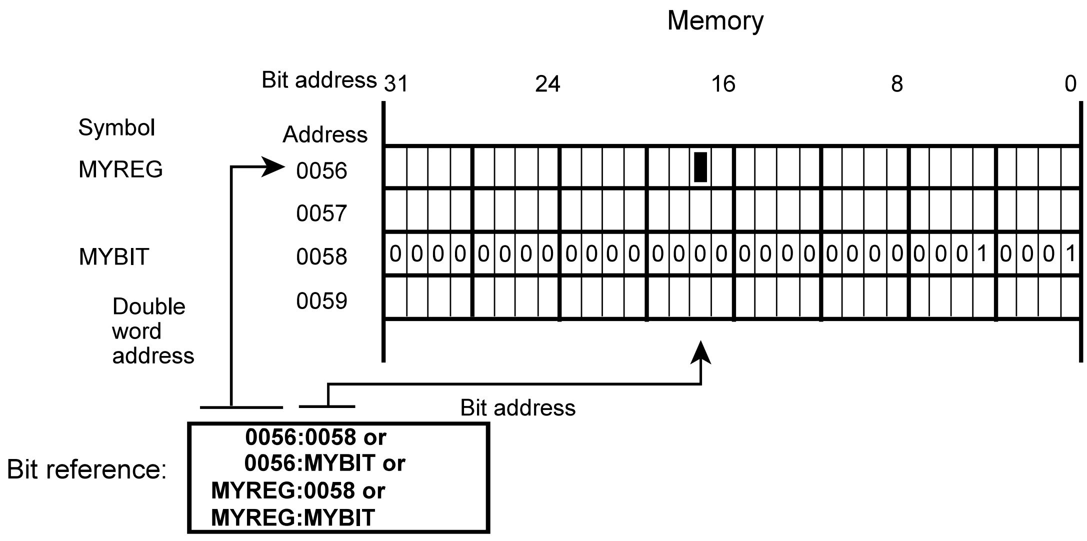
DRV_RDY

symbol.



2.4 Indirect Addressing of Bits with Operator “:”

When the **operator “:”** is used, bit address contains two parts:
double word address and
bit address in double word
The two parts separated by the operator
: (colon).
You can refer to double word address by
4 decimal numbers,
or also by a symbol.
You can refer to bit address after “:” operator
by numeral address defined by 4 decimal numbers or
by the symbol declared to the register.
For example the reference
0056:0058
refers to that bit of register 0056, which is specified by the content of register 0058.
If
symbol MYREG declared at address 0056, and
symbol MYBIT declared at address 0058,
then you can also specify
MYREG:MYBIT.



2.5 Indexed Addressing of Bits with Operator “,”

Indexed addressing of bit is similar to indexing double word. The address contains two parts:

base address of bit (**base.bit**) and

offset.

The base address of bit is referred to a bit of a double word (base). The offset always shifts the address of double words.

The two parts are separated by the operator

, (comma).

The address is calculated by adding the offset to the base address:

bit address=(base+offset).bit

The rules of bit addressing are valid for **base address of bit**:

symbol

defined by operator “.” dot, or

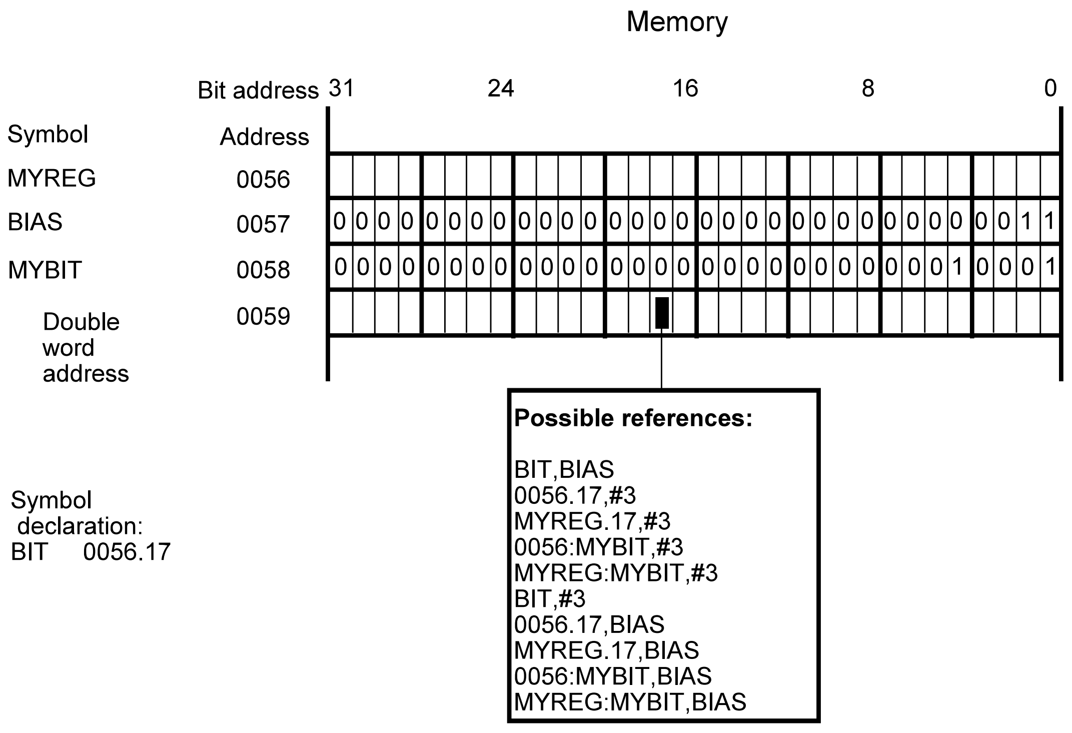
indirectly with operator “:”.

The **offset value** can be defined

directly with numeral, or

indirectly with register reference.

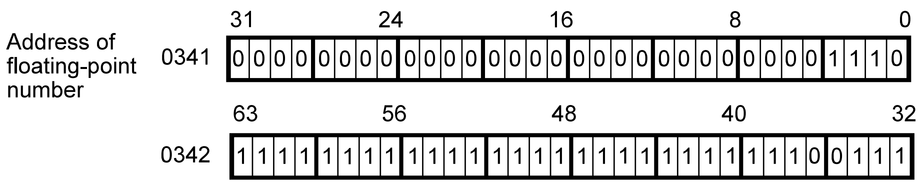
If you want to offset bit addressed to 0056.17 with 3, you can do it the way shown on the figure below:



2.6 Addressing Floating-point Numbers (double)

Instructions of PLC program handle 64 bit floating-point numbers (double). The floating-point numbers reserve two subsequent registers (DWORD) of memory space.

Symbol: FLOAT 0341



You can address a floating-point number by

numeral and also by

symbol.

In case of addressing it by numeral, you always have to specify 4 decimal numbers with **leading zeros**.

Beforehand referring to it by a symbol, you always have to declare the symbol in the symbol table.

In case of referring to a memory space containing floating-point number, you have to give the address of first word (it contains the lower 32 bits of floating-point number).

You can refer to floating-point number in the above example:

0341 or

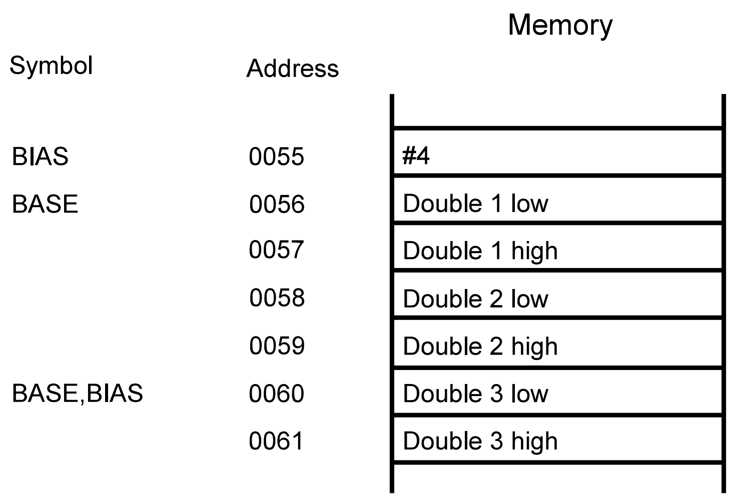
FLOAT.

2.7 Indexed Addressing of Floating-point Numbers (double) with Operator “,”

The rules of indexed addressing of DWORD registers are valid for indexed addressing of floating-point numbers.

☞ **Attention!**

Floating-point numbers always have to be indexed by even numbers, because they reserve two subsequent registers!



Symbol	Address	Memory
BIAS	0055	#4
BASE	0056	Double 1 low
	0057	Double 1 high
	0058	Double 2 low
	0059	Double 2 high
BASE,BIAS	0060	Double 3 low
	0061	Double 3 high

Direct indexed addressing: BASE,#4=56+4=60
Indirect indexed addressing: BASE,BIAS=56+4=60

3 Modules of the PLC Program

The PLC program contains **2 modules**:
the **main program** and
the **Int0** module.
Each module is written by ladder diagram and they are independent from each other. The connection between the two modules is possible via PLC memory.
The difference between the two modules is the frequency of runs.

3.1 The Main Program

The main program runs by T_{PLC} cycle time. The value of cycle time depends on control type. The value of cycle time can be read in the PlcPeriod line in msec unit of Diagnostics window of NCT201 control (for example: 10msec). All PLC activities are written in this module, except for which need faster reaction than T_{PLC} cycle time.

3.2 The Int0 Module

The Int0 module runs higher frequency than T_{PLC} cycle time. The value of this cycle time can be read in the TimeSlice line in µsec unit of Diagnostics window of NCT201 control (for example: 2000 µsec). Only PLC activities are allowed in this module, which need fast reaction, for example an interface output has to be switched on quickly after an interface input signal has been changed. If you overload this module, then it sends an error message “PLC timeout”.

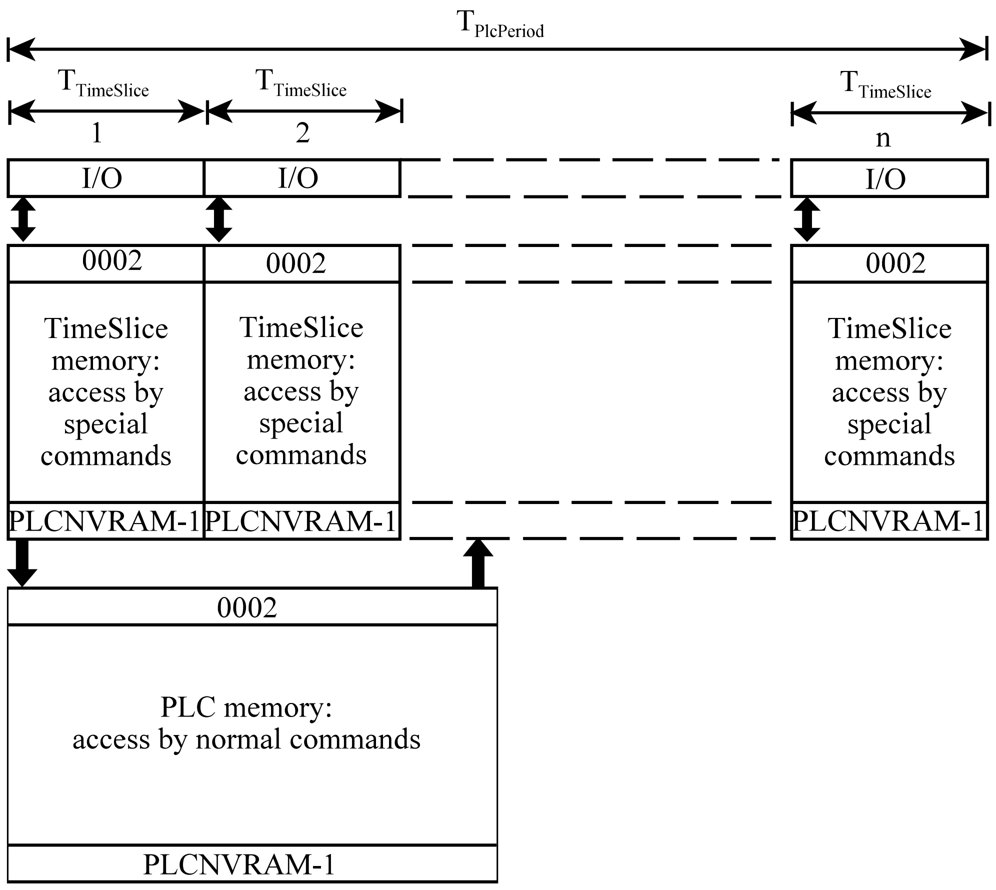
3.3 Updating the PLC Memory

The **memory space which communicates between the interface inputs and outputs, as well as PLC program and system** is read and written by the **frequency** of **Int0** module, namely the **TimeSlice** time. This memory space starts from the address 0002 and continues up to PLCNVRAM-1 address.

The physical inputs (for example interface input signals) are read in RAM, the physical outputs (for example interface output signals) are written out from RAM to hardware. PLC program can read or write this RAM storage updated by TimeSlice frequency, the **TimeSlice memory** only by **special instructions**. These instructions are only used in module Int0!

PLC input memory space available by normal PLC instructions updated from TimeSlice memory, that is **synchronized before running PLC main program**. PLC main program can run for several TimeSlice cycles, so it works from a RAM where inputs are not changed while the main program is running. **Outputs** controlled by normal instructions of PLC program **updated** in TimeSlice memory **after running PLC main program.**

Normal instructions of each module, the Main program and Int0 use the same synchronized memory space.


T_PlcPeriod
T_TimeSlice
1
T_TimeSlice
2
T_TimeSlice
n
I/O
I/O
I/O
0002
0002
0002
TimeSlice memory: access by special commands
TimeSlice memory: access by special commands
TimeSlice memory: access by special commands
PLCNVRAM-1
PLCNVRAM-1
PLCNVRAM-1
0002
PLC memory: access by normal commands
PLCNVRAM-1

4 Data Managed by the PLC Program

The PLC program can manage fixed-point, floating-point and bit-type data.
Fixed-point data stored in memory can be also in binary and in BCD (binary coded decimal) form.
Bit-type data can be also queried for variation.

4.1 Managing Bit-type Data

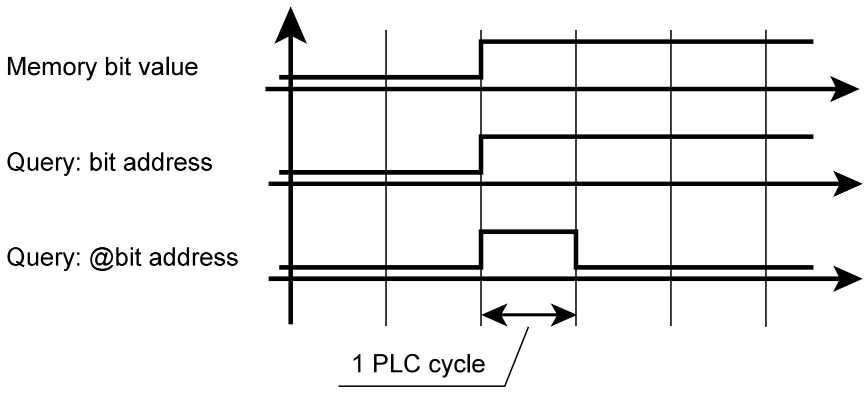
Bit-type data is the most frequently used data unit of a PLC program. You can query any bit of the whole memory space of PLC, so you can define a contact at bit addresses and write any output bit, so it can be used as a relay coil for example. In addition, you can start variation verification for any bit of the whole memory space.

4.2 Query of Rising Edge of Memory Bits with Operator “@”

Change from 0 to 1 of any bit of memory available for a PLC program can be queried, if you write “@” operator before address of memory bit:

@bit address: query of rising edge

Bit address can be defined by all possible methods: directly, or indirectly, symbolical or also indexed.
The queried data can be TRUE for 1 PLC cycle:

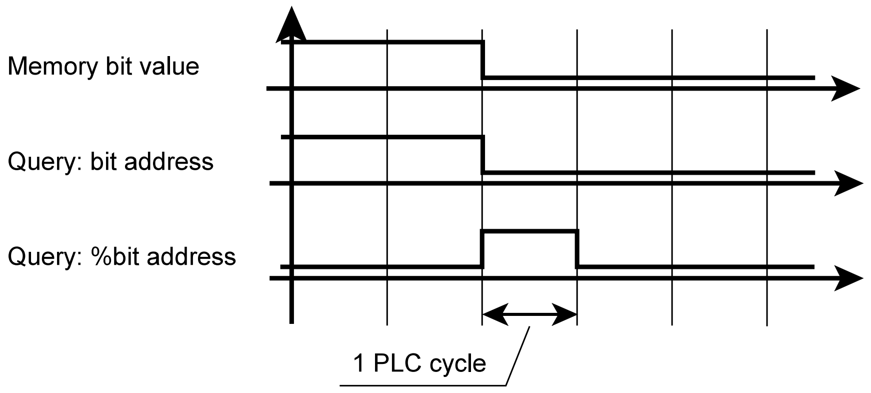


4.3 Query of Falling Edge of Memory Bits with Operator “%”

Change from 1 to 0 of any bit of memory available for a PLC program can be queried, if you write “%” operator before address of memory bit:

%bit address: query of falling edge

Bit address can be defined by all possible methods: directly or indirectly, symbolical or also indexed.
The queried data can be TRUE for 1 PLC cycle:



4.4 Immediate Query of Inputs, Immediate Issue of Outputs with Operator "!"

Any bit of memory available for PLC program can be queried immediately from TimeSlice memory, or issued immediately to TimeSlice memory if you write "!" operator before memory bit address:

!bit address: bit-type reading and writing of TimeSlice memory

Bit address can be defined by all possible methods: directly, or indirectly, symbolical or also indexed. It should be used in cases which need fast reaction in Int0 module.

☞ ***Attention!*** *The immediately refreshing operator "!" must not be used together with @, or % edge query operators!*

4.5 Definition of Decimal, Signed Number with Operator "#"

Decimal, signed constant can be defined with # operator in

$-2147483648 \leq \text{constant} \leq 2147483647$

range. PLC editor indicates error, if entered data is out of range.

The entered data can is stored in memory in **binary coded, two's complement** format.

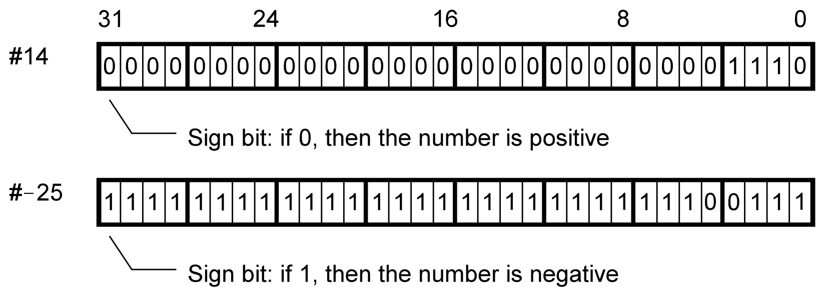
The bit no.31 is the sign bit. The + (positive) sign should not be specified.

So, if you want to write +14 in memory, you write

#14,

if you want to write −25, then you write

#−25.



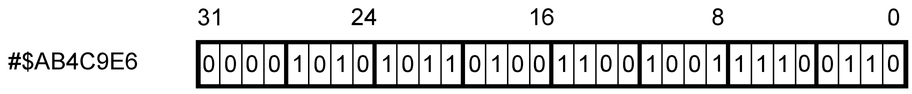
4.6 Definition of Hexadecimal Number With Operator “#$”

Max. 8 digit hexadecimal constant can be defined with “#$“ operator in

0≤constant≤FFFFFFFF

range. PLC editor indicates error, if entered data is out of range. The leading zeros can be neglected.
For example, if you would like to write 0AB4C9E6 hexadecimal value in memory, then write

#$AB4C9E6.



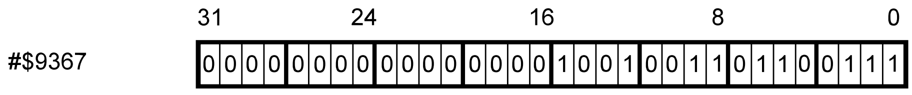
4.7 Definition of BCD Number without Sign with Operator “#$”

Max 8 digit BCD (binary coded decimal) constant without sign can be defined with #$ operator in

0≤constant≤99999999

range. So, if you would write BCD data in memory, then it is managed as a hexadecimal number, but only 0, 1, ..., 9 digits used. The leading zeros can be neglected.
For example, if you would like to write 9367 decimal number in memory in BCD coding, you write

#$9367.



4.8 Definition of Floating-point Number with Operator “*”

Representation of floating-numbers in a PLC program follows the double precision representation of floating-point number of IEEE754 standard. These numbers are represented in 64 bits. So, if you would write a floating-point data in memory, it always reserves

two double words (64 bits)

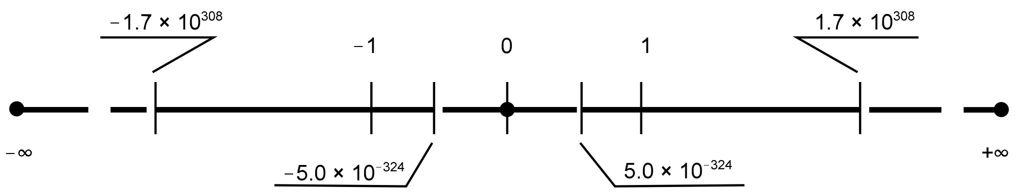
space. If the floating-point data is written to address n, the data is stored at the addresses

n and **n+1**.

A floating-point number must not be written to address 9999!
Floating-point numbers with * operator can be represented about in the range

from $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$

and 0 with a precision of 15-16 digit. You have to use a **decimal point** (.) when specifying them.



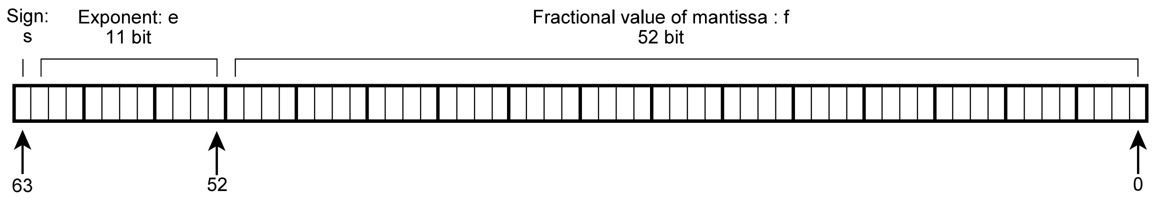
For example, if you would like to input −124.753 number, then you have to write

*−124.753

You need not specify neither leading nor trailing zeros. The + (positive) sign can be neglected.

4.9 Double Precision Representation of Floating-point Numbers in IEEE754 Standard

We describe briefly the structure of IEEE754 double precision floating point format for information, but *PLC programmer need not deal with it absolutely.*



The double precision floating-point number contains three parts:

– **Sign bit** (s): If its value is 0 then the number is positive, if its value is 1 then the number is negative.

– **Exponent** (e): 11 bits long, and its base is 2. Positive and negative exponents also have to be represented in this space. To manage it, you have to define **offset** for the effective exponent, its value is **1023**. So, if the value of a stored exponent is e=1201, the effective exponent received from the equation 1201−1023=178. The full 0 (000h) and full 1 (7FFh) exponent values are reserved for special application.

– **Mantissa**: The mantissa is 53 bits long. It consists of two parts: a **one-bit integer part** and a **fraction part** (f), which is 52 bits long. The value of integer part is 1 in case of normalized form numbers, and 0 in case of denormalized numbers. Therefore the mantissa is 53 bits long, but it reserves only 52 bits space representing numbers, because we represent only the fraction part.

The value of exponent (e) and fraction part of mantissa (f) affects the representation:

– **Normalized number**: if the exponent is not full of 0 (e≠000h) and not full of 1 (e≠7FFh) the number is normalized. Then we assume 1 for integer part of mantissa, and the number received from the following relation:

$(-1)^s \times 1.f \times 2^{e-1023}$, where “s” is the sign bit, “f” is the fraction part of mantissa, “e” is the exponent.

Substitute $1.f=2-2^{-52}$, the biggest absolute value can be represented:

$(-1)^s \times (2-2^{-52}) \times 2^{1023} \approx \pm 1.7 \times 10^{308}$

– **Overflow**: If the result of a floating-point operation could not be represent, because the result overflows the maximal value can be represented, the operation sets the **FL_OF** overflow status bit.

– **Subnormal or denormalized number**: if the exponent is full of 0 (e=000h) but the fraction part of mantissa is not 0, f≠0, the number is in subnormal form. We use this format representing very small numbers. This time we assume 0 for integer part of mantissa, and the number received from the following relation:

$(-1)^s\times 0.f\times 2^{-1022}$, where "s" is th sign bit, "f" is the integer part of mantissa.

Substitute $0.f=2^{-52}$, the smallest absolute value can be represented:

$(-1)^s\times 2^{-52}\times 2^{-1022}=(-1)^s\times 2^{-1074}\approx\pm 5.0\times 10^{-324}$

– **Underflow**: If the result of a floating-point operation could not be represent, because the result smaller than the minimal value can be represented, the operation sets the **FL_UF** underflow status bit.

– **Zero**: that number is zero, which value of exponent and fraction part of mantissa also equal 0:

e=0 and f=0

The standard distinguishes +0 and -0 in function of sign bit (s=0, or s=1). So the zero is a special, subnormal form number.

– **Infinity**: If the exponent is full of 1 and the fraction part of mantissa is 0

e=2047(=7FFh) and f=0

the number is ±∞, in function of s sign bit.

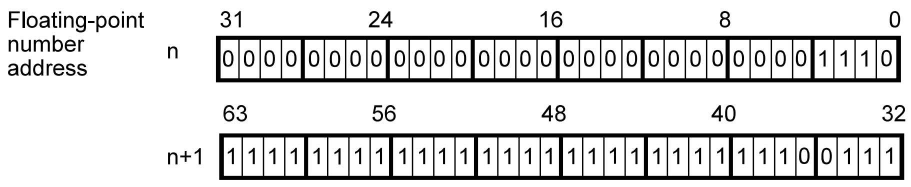
– **Not a Number**, **NaN**: the represented value is not considered as valid number, if the exponent is full of 1 and the fraction part of mantissa is not 0:

e=2047(=7FFh) and f≠0

– **Error signal for Not a Number value**: If any number in floating-point operation is Not a Number (NaN), for example because of a missed initialization, the control sets **FL_ER** status bit.

Location of floating point number in memory

The lower 32 bits of a floating-point number is located at the address defined, but the upper 32 bits reserves the double word after the address defined:



5 Status Bits Set by PLC Instructions

Flags switched by PLC instructions used for indicate the result or error or status of an operation. The status bits can be reached at the address 0000 or on symbol FLAGS. This word is **read only** for the PLC program.

5.1 FL_ER Error Flag

The system sets this flag, if found an error in the executed PLC instruction. These can be the following errors:

the defined address is out of range 0000....9999,
the defined data is out of possible range of the value,
the defined data is not in proper format,
one of the data of a floating-point operation is NaN (Not a Number)
the input parameters of an instruction are failed,
the instruction could not be executed.

We deal separately the managing of FL_ER flag when describing each instructions.

5.2 FL_UF Underflow Flag

The system sets this flag, if:

– the result of a fixed-point addition of two negative numbers falls in the range of positive numbers 00000000 ... 07FFFFFF, else 0,
– the result of a fixed-point subtraction of a positive number from a negative number falls in the range of positive numbers 00000000 ... 07FFFFFF, else 0,
– the result of a floating-point operation is so small that it cannot be represented by double precision floating-point number format.

5.3 FL_OF Overflow Flag

The system sets this flag, if:

– the result of a fixed-point addition of two positive numbers falls in the range of negative numbers, 80000000...FFFFFFFF else 0,
– the result of a fixed-point subtraction of a negative number from a positive number falls in the range of negative numbers 80000000 ... FFFFFFFF, else 0,
–the result of a floating-point operation is so big that it cannot be represented by double precision floated-point number format.

5.4 FL_CY Carry Flag

The system sets this flag, if:

– carry is generated in a fixed-point addition, so the result does not fit in 32 bits,
– borrow is generated in a fixed-point subtraction, else 0.

5.5 FL_GT Greater than Flag

The system sets this flag, when,

– comparing two numbers, and value on left side is greater than value on right side: A>B.

5.6 FL_EQ Equal Flag

The system sets this flag, when,
– comparing two numbers, and the two values equal to each other: A=B.

5.7 FL_LT Lower than Flag

The system sets this flag, when,
– comparing two numbers and value on left side is lower than value on right side: A<B.

6 Instructions of the PLC Program

6.1 Bit Operation Instructions

Bit operation instructions use two values, 0 and 1. Consider value 0 as FALSE state, and the value 1 as TRUE state. Bit operations realize different Boolean algebraic operations, like AND, OR and EXCLUSIVE OR. The inputs of logic operations always apply to corresponding bits of memory, the results of operations write the corresponding bits of memory.

6.1.1 Open Contact: Input of a Memory Bit

The symbol of normally open contact is: ┤ ├

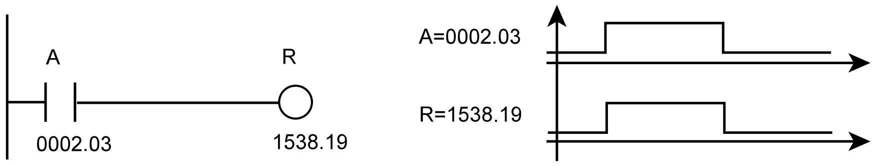
Input parameter of open contact:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31
The address of bit can be a numeral or a symbol. Indexed addressing is possible.
Possible modifiers:
Before address: managing @ rising edge or % falling edge,
After address: "," indexing operator.

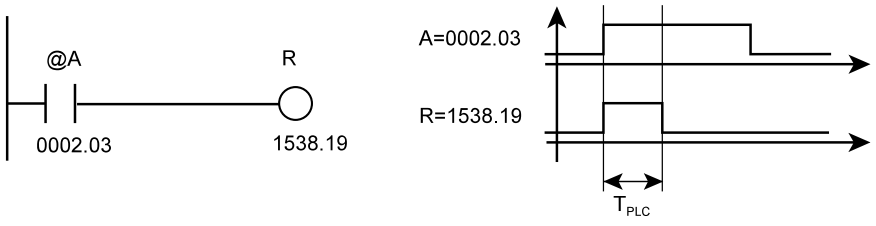
Operation of open contact:

The current flows through contact (the contact is closed), if value of memory bit is TRUE (1) and current does not flow (the contact is open), if value of memory bit is FALSE (0). The relay coil connected after contact is energized or released accordingly.



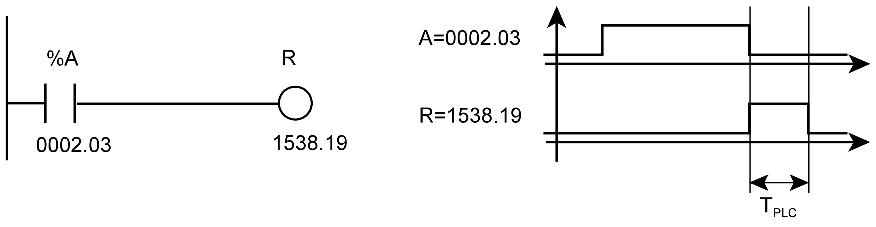
Operation of open contact with the operator of rising edge @

If you write operator @ before open contact address, current flows for 1 PLC cycle after the rising edge of signal (memory bit changes from 0 to 1), so relay R is energized for 1 T_{PLC} interval.



Operation of open contact with falling edge % operator

If you write % operator before open contact address, current flows for 1 PLC cycle after the falling edge of signal (memory bit changes from 1 to 0), so relay R is energized for 1 T_{PLC} interval.



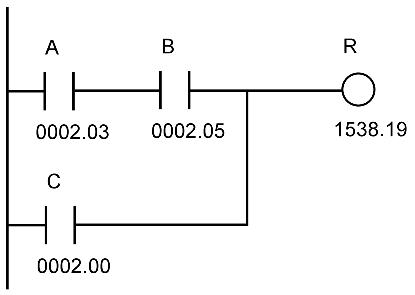
Operation of open contact with immediately refreshing ! operator

If you write operator ! before open contact address, it takes the state of contact from the TimeSlice memory.

Example for application of open contact

The series-connected open contacts realize logic AND operation, parallel-connected open contacts realize logic OR operation.
The attached figure shows the following logic operation:

(A AND B) OR C = R



6.1.2 Open Contact: Query of a Double Word

The symbol of normally open contact is: ┤ ├

Input parameter of open contact:

– **Address of Bit**:

Range of the value: n=0000...9999
Double word address can be a numeral or a symbol. Indexed definition is possible.
Possible modifiers:
Before address: managing @ rising edge or % falling edge,
After address: "," indexing operator.

Operation of open contact:

The current flows through contact (the contact is closed) if value of double word>0, and current does not flow (the contact is open) if value of double word=0. The relay coil connected after contact is energized or released accordingly.
The instruction can be used for query of timers, counters. For example, to check if the timer is still working (>0), so current flows through the contact, or it has been expired (=0), so current does not flow through the contact.

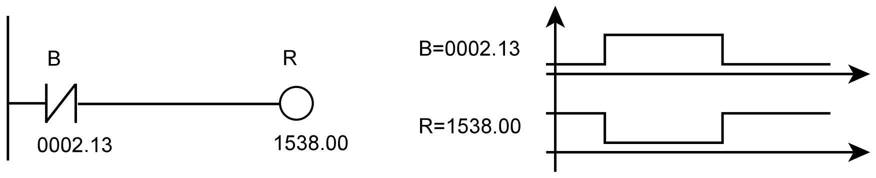
6.1.3 Closed Contact: Input Not of a Memory Bit

The symbol of normally closed contact is: ─|/|─

Input parameter of closed contact:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31
The address of bit can be a numeral or a symbol. Indexed addressing is possible.
Possible modifiers:
Before address: managing @ rising edge or % falling edge,
After address: "," indexing operator.

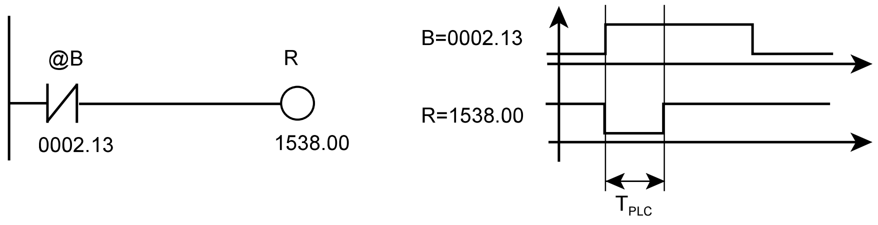


Operation of closed contact:

The current flows through contact (the contact closed), if value of memory bit is FALSE (0), and current does not flow (the contact open), if value of memory bit is TRUE (1). The relay coil connected after contact is energized or released accordingly.

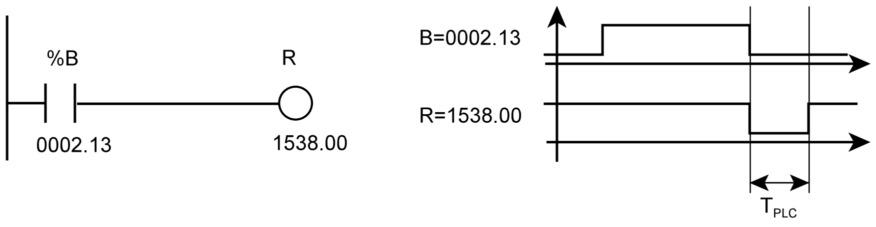
Operation of closed contact with rising edge @ operator

If you write operator @ before closed contact address, current does not flow until 1 PLC cycle time after rising edge of signal (memory bit changes from 0 to 1), so relay R is released for 1 T_{PLC} interval.



Operation of closed contact with falling edge % operator

If you write operator % before closed contact address, current does not flow until 1 PLC cycle time after falling edge of signal (memory bit changes from 1 to 0), so relay R is released for 1 T_{PLC} interval.

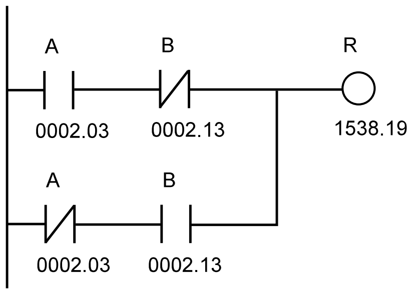


Operation of closed contact with immediately refreshing ! operator

If you write operator ! before closed contact address, it takes the state of closed contact from the TimeSlice memory.

Example for application of closed contact

You can realize EXCLUSIVE OR operation by using open and closed contacts.
EXCLUSIVE OR operation can be specified in the following form:
A XOR B = (A AND (NOT B)) OR ((NOT A) AND B)
The figure to the right shows this function realized with open and closed contacts.



6.1.4 Closed Contact: Query Not of a Double Word

The symbol of normally closed contact is: ┤/├

Input parameter of closed contact:

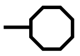
– **Address of Bit**:

Range of the value: n=0000...9999
Double word address can be a numeral or a symbol. Indexed definition is possible.
Possible modifiers:
Before address: managing @ rising edge or % falling edge,
After address: "," indexing operator.

Operation of closed contact:

The current flows through contact (the contact is closed), if the value of double word=0, and current does not flow (the contact is open), if the value of double word >0. The relay coil, connected after contact is energized or released accordingly.
The instruction can be used for query of timers, counters. For example, to check if the timer is still working (>0), so current flows through the contact, or it has been expired (=0), so current does not flow through the contact.

6.1.5 Relay Coil: Output to a Memory Bit

The symbol of relay coil is: 

Input parameter of relay coil:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31
The address of bit can be a numeral or a symbol. Indexed definition is possible.
Possible modifiers:

After address: "," indexing operator.

– **Remark:**

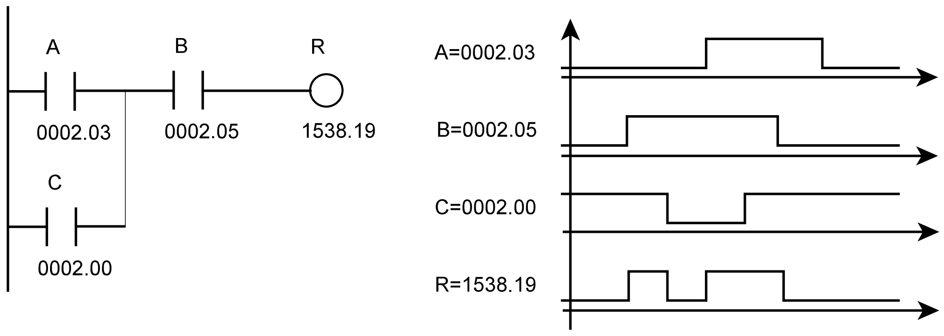
Remark: text written in Remark will be the comment of rung.

Operation of relay coil with immediately refreshing ! operator

If you write operator ! before relay coil address, it outputs the state of relay coil to TimeSlice memory.

Operation of relay coil

If current flows in the input of relay coil, so all conditions before coil are in TRUE state, the relay is energized, so the instruction sets memory bit to 1.
Inversely, if current does not flow in the input of relay coil, so the condition before coil is in FALSE state, the relay is released, so the instruction resets memory bit to 0.
The relay coil is a component that closes the rung, so it has not output, nothing can be connected after it.



6.1.6 Negated Relay Coil: Output Not to a Memory Bit

The symbol of negated relay coil is: —∅

Input parameter of negated relay coil:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31
The address of bit can be a numeral or a symbol. Indexed definition is possible.
Possible modifiers:
After address: "," indexing operator.

– **Remark:**

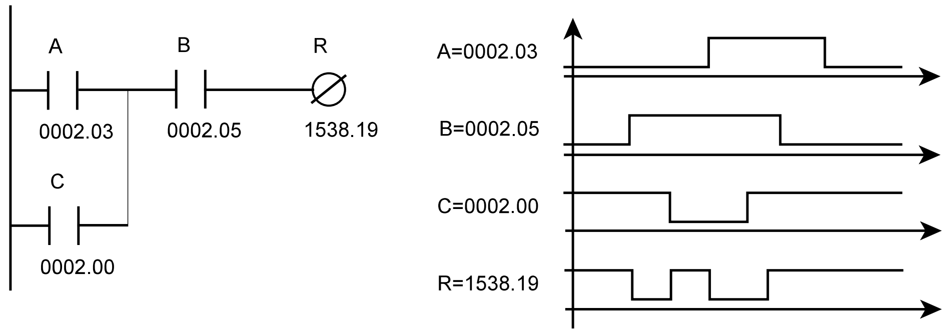
Remark: text written in Remark will be the comment of rung.

Operation of negated relay coil with immediately refreshing ! operator

If you write operator ! before negated relay coil address, it outputs the inverse state of relay coil to TimeSlice memory.

Operation of negated relay coil

If current flows in the input of negated relay coil, so all conditions before coil are in TRUE state, the relay is released, so the instruction resets memory bit to 0.
Inversely, if current does not flow in the input of negated relay coil, so the conditions before coil is in FALSE state, the relay is energized, so the instruction sets memory bit to 1.
The negated relay coil is a component that closes the rung, so it has not output, nothing can be connected after it.



6.1.7 Setting a Memory Bit: the SET Instruction

The symbol of SET instruction is:

Input parameter of SET instruction:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31
The address of bit can be a numeral or a symbol. Indexed definition is possible.
Possible modifiers:
After address: "," indexing operator.

– **Remark:**

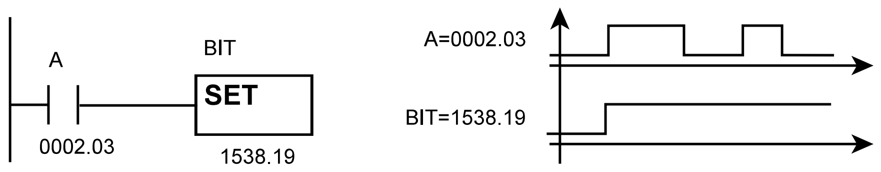
Remark: text written in Remark will be the comment of rung.

Operation of SET instruction with immediately refreshing ! operator

If you write operator ! before SET instruction address, bit is set in TimeSlice memory.

Operation of SET instruction

If all conditions before the instruction is in TRUE state, the instruction sets the corresponding memory bit to 1.
If all conditions before the instruction assume FALSE state after then, the memory bit remains the same value.
The SET instruction is a component that closes the rung, so it has not output, nothing can be connected after it.



6.1.8 Resetting a Memory Bit: the RST Instruction

The symbol of RST instruction is:

Input parameter of RST instruction:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31
The address of bit can be a numeral or a symbol. Indexed definition is possible.
Possible modifiers:
After address: "," indexing operator.

– **Remark:**

Remark: text written in Remark will be the comment of rung.

Operation of RST instruction with immediately refreshing ! operator

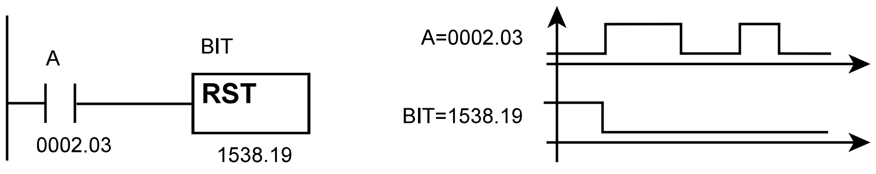
If you write operator ! before RST instruction address, bit is reset in TimeSlice memory.

Operation of RST instruction

If all conditions before the instruction is in TRUE state, the instruction resets the corresponding memory bit to 0.

If all conditions before the instruction assume FALSE state after then, the memory bit remains the same value.

The RST instruction is a component that closes the rung, so it has not output, nothing can be connected after it.



6.1.9 Pulse Generation for Rising Edge: the DIFU Instruction

The symbol of DIFU instruction is:

Input parameter of DIFU instruction:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31

The address of bit can be a numeral or a symbol. Indexed definition is possible.

Possible modifiers:

After address: "," indexing operator.

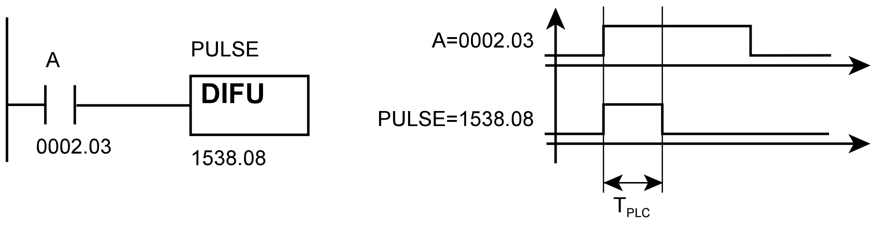
– **Remark:**

Remark: text written in Remark will be the comment of rung.

Operation of DIFU instruction

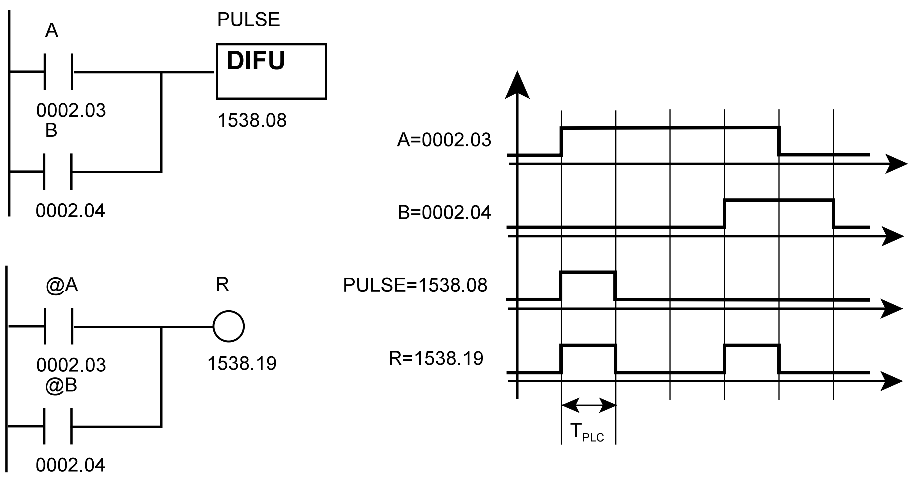
If all conditions before instruction turn from FALSE (0) state to TRUE (1) state, the instruction sets the corresponding memory bit to 1 for 1 PLC cycle (T_{PLC} time), so it differentiate for the rising edge of signal.

The DIFU instruction is a component that closes the rung, so it has not output, nothing can be connected after it.



Application of DIFU instruction

The following example shows the difference between the application of DIFU instruction and the @ operator. If you differentiate the result of logic operation A OR B by DIFU instruction, you get a different result than when the rising edge of A connected to the rising edge of B with logic operation OR and the result is stored in relay R.



6.1.10 Pulse Generation for Falling Edge: the DIFD Instruction

The symbol of DIFD instruction is: $\boxed{\text{DIFD}}$

Input parameter of DIFD instruction:

– **Address of Bit**:

Range of the value (n.i): for double word address: n=0000...9999, for bit address: i=00..31
The address of bit can be a numeral or a symbol. Indexed definition is possible.
Possible modifiers:

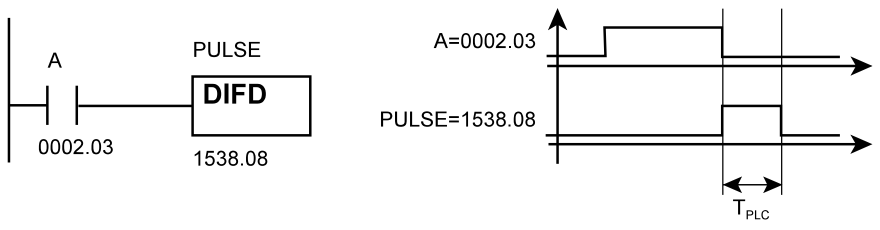
After address: “,” indexing operator.

– **Remark:**

Remark: text written in Remark will be the comment of rung.

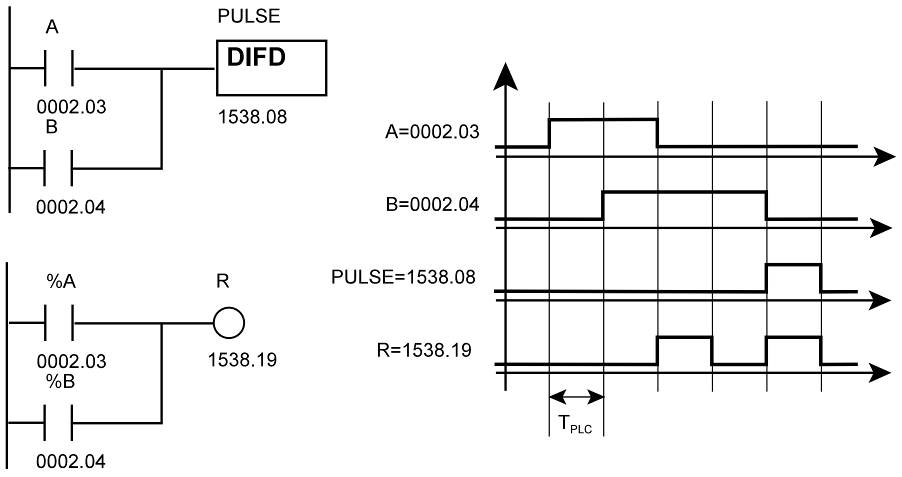
Operation of DIFD instruction

If all conditions before instruction turn from TRUE (1) state to FALSE (0) state, the instruction sets the correspond memory bit to 1 for 1 PLC cycle (T_{PLC} time), so it differentiate for the falling edge of signal.
The DIFD instruction is a component that closes the rung, so it has not output, nothing can be connected after it.



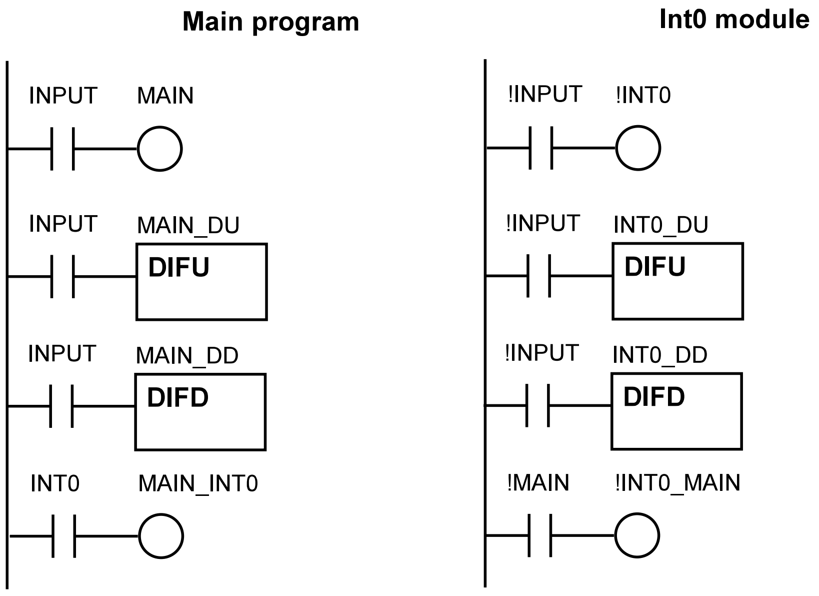
Application of DIFD instruction

The following example shows the difference between the application of DIFD instruction and the % operator. If you differentiate the result of logic operation A OR B by DIFD instruction, you get a different result than when the falling edge of A connected to the falling edge of B with OR logic operation and the result is stored in relay R.



6.1.11 Bit Operation Instructions and the Operator "!" in the Two Modules

Observe the difference of bit operation instructions in modules Main program and Int0 in the following two program details.

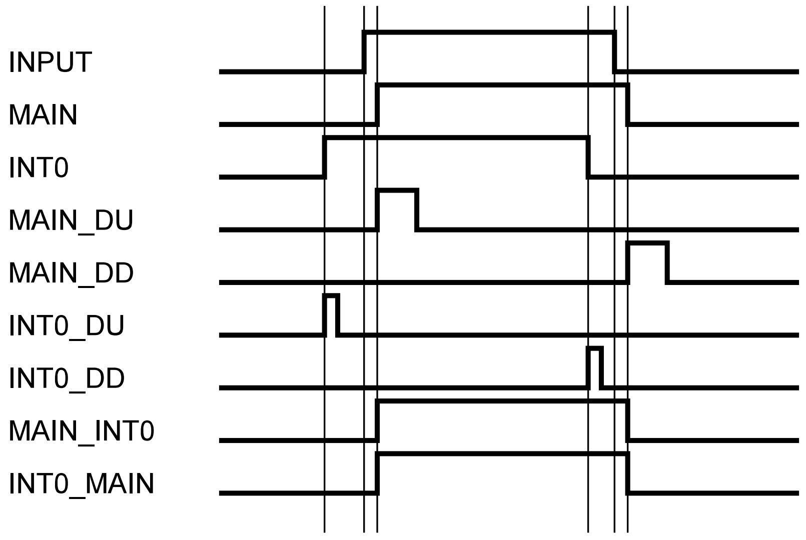


Input called INPUT

– Main program reads it from PLC memory,

– while Int0 module reads it from TimeSlice memory at address !INPUT.

The MAIN output
– is written by the Main program to PLC memory,
while the INT0 output
– is written by the Int0 module to TimeSlice memory to address !INT0.
The DIFU, DIFD instructions are used in both modules.
The outputs are stored in Main program at addresses MAIN_DU, MAIN_DD, and in module Int0 at addresses INT0_DU, INT0_DD.
The timing chart of individual signals can be seen in the diagram below.
The signal INPUT is updated from the PLC memory, while the signal INT0 receives its input and writes its output also from and to TimeSlice memory. Therefore, signal INT0 precedes in time signal INPUT.
On the outputs of DIFU, DIFD instructions can be seen, that module Int0 runs on higher frequency than the Main program, therefore the outputs INT0_DU, INT0_DD are set for a shorter interval than outputs MAIN_DU, MAIN_DD.
There is no difference between MAIN_INT0 and INT0_MAIN signals. The signal MAIN_INT0 is updated from PLC memory by the instruction INT0, which is updated for T_{PLC} interval. It is unnecessary to update signal INT0_MAIN to TimeSlice memory, and to receive input from TimeSlice memory, because the signal MAIN is updated by T_{PLC} interval, therefore MAIN_INT0 and INT0_MAIN signals have identical timing.



6.2 Basic Rules of Connections of Ladder Network

Contacts and instructions of a ladder diagram network must be connected. The connecting elements are not instructions, they have not got any input or output parameters, only "wires" for showing "circuit", graphic elements, their space is one cell. These elements are shown in the following picture, where one cell belongs to one row and one column.
The graphic elements arranged in rows and columns. One rung is a logic unit, which consist of one or more rows. In the PLC program the

number of rungs,

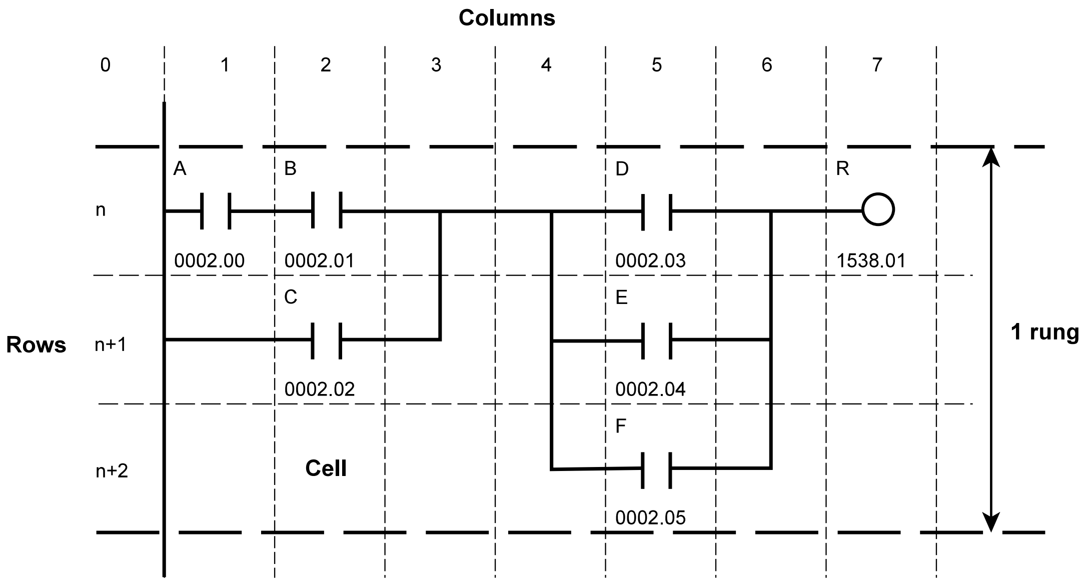
in a rung the

number of rows

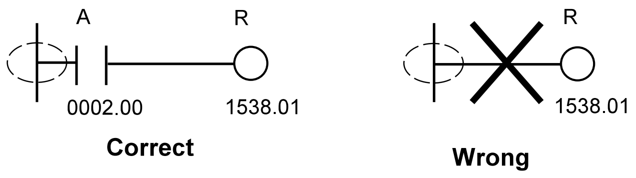
in a row the

number of columns

so the graphic elements (lines, contacts, instructions) written in a row is not limited.



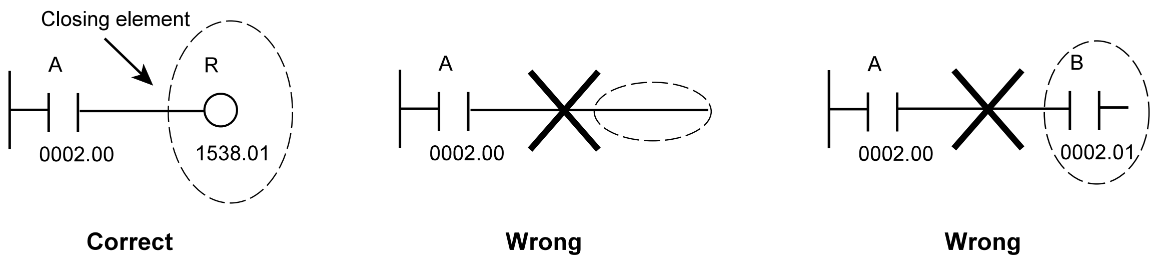
Most of the **instructions**, except some of them, **must not be written directly to the first column**, only after a contact. So, for example, relay coil or SET instruction could not be connected directly to the left side vertical wire.



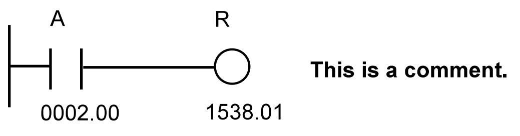
The last element of a rung, the far right instruction is not connected anywhere. The last element called as

closing element.

Closing element could not be graphic element, contact, or instruction which have output.



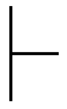
In case of a closing element, you can always define **comment** by filling Remark parameter. The comment is written in the row of closing element, from the column following the closing element.



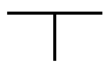
6.2.1 Connecting elements

The connecting elements are not instructions, they have not got any input or output parameters, only “wires” for showing “circuit”, graphic elements, the space occupied by them is one cell. A ladder diagram PLC uses the following connecting elements:

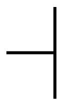
Vertical-Right:



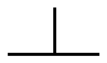
Horizontal-Downwards:



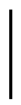
Left-Vertical:



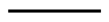
Horizontal-Upwards:



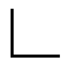
Vertical:



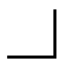
Horizontal:



Upwards-Right:

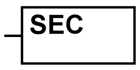


Left-Upwards:



6.2.2 Commenting the Logic Sectors of a Ladder Diagram: the SEC Instruction

Comment can be written after closing elements. There is an instruction especially for defining comments.

The symbol of SEC instruction is: 

☞ ***Attention!*** *The SEC instruction can be written in only the 1. column, even it is closing element!*

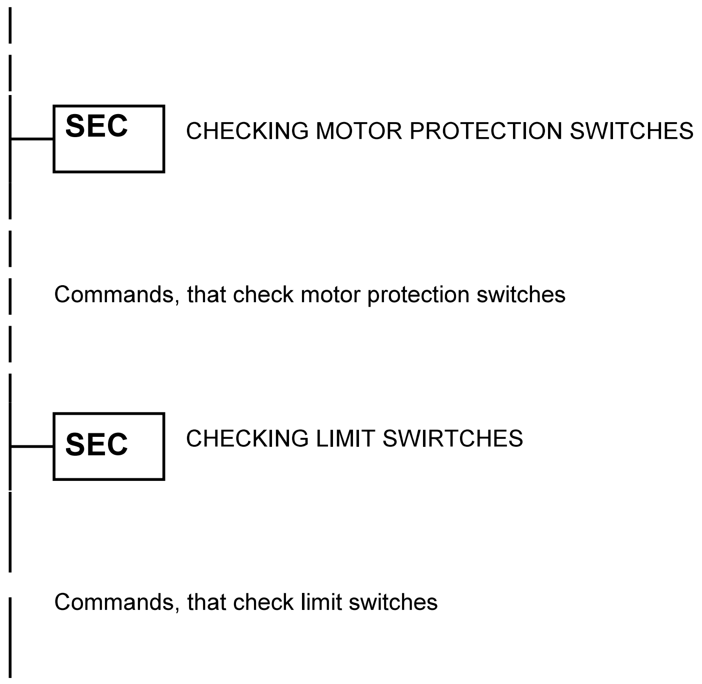
Input parameter of SEC instruction:

– Remark:

Remark: the text written in Remark will be the comment of logic sector.

Description of SEC instruction

The SEC instruction does not perform any operation, it is only used for to separate logic sectors of PLC program from each other. Only comment text can be the input parameter.


SEC
CHECKING MOTOR PROTECTION SWITCHES
Commands, that check motor protection switches
SEC
CHECKING LIMIT SWIRTCHES
Commands, that check limit switches

6.3 Data Movement Instructions

With data movement instructions, you can define a value for any register of PLC memory, or any register can be copied to another address.
You always have to specify the format of data to be defined:

#: signed decimal integer number,
#$: hexadecimal number, or
*: floating-point number.

Rules of addressing registers (by symbol, numeral, or indexed) are valid from the earlier.

The inputs and outputs of data movement instructions

Each data movement instruction has its own
enable input.
The movement is executed in the true state of the enable input.
You can *configure* data movement instructions to have an
output.
The output is in TRUE state, if the input of instruction is TRUE and movement has been executed. The output changes into FALSE state, if the input of instruction is FALSE, or instruction cannot be executed, so instruction sets the error flag FL_ER.
You can connect further data movement instructions to its output.
All data movement instructions have common input parameters.

Input parameters of data movement instructions

– **Address of Result**:
Range of the value: n...9999 (n: where starts user addresses)
The address of the *destination* register (result) can be a numeral or a symbol. Indexed addressing is possible.
The defined value, or content of register to be moved gets to this address.

– **Operand**:
The *value* of data or the *address* of the *source* register to be moved.
If you define data, you always have to specify data corresponding to the format of instruction.
The address of the source can be a numeral or a symbol. Indexed addressing is possible.

– **Output**:
You can connect further instructions to the output of the instruction box, in case of its enabled state.

– **Remark**:
If the output is disabled, then the remark written here is displayed as a comment. If you have not written here any remark, the comment of the symbol defined at address of result box is written here.

6.3.1 Movement of Double Words: the MOV and the MVN Instructions

The

MOV

instruction moves a DWORD long source to the destination register, if condition is satisfied in its input. The source can be:

a signed decimal number defined by # operator, or
a hexadecimal number defined by #$ operator, or
a register.

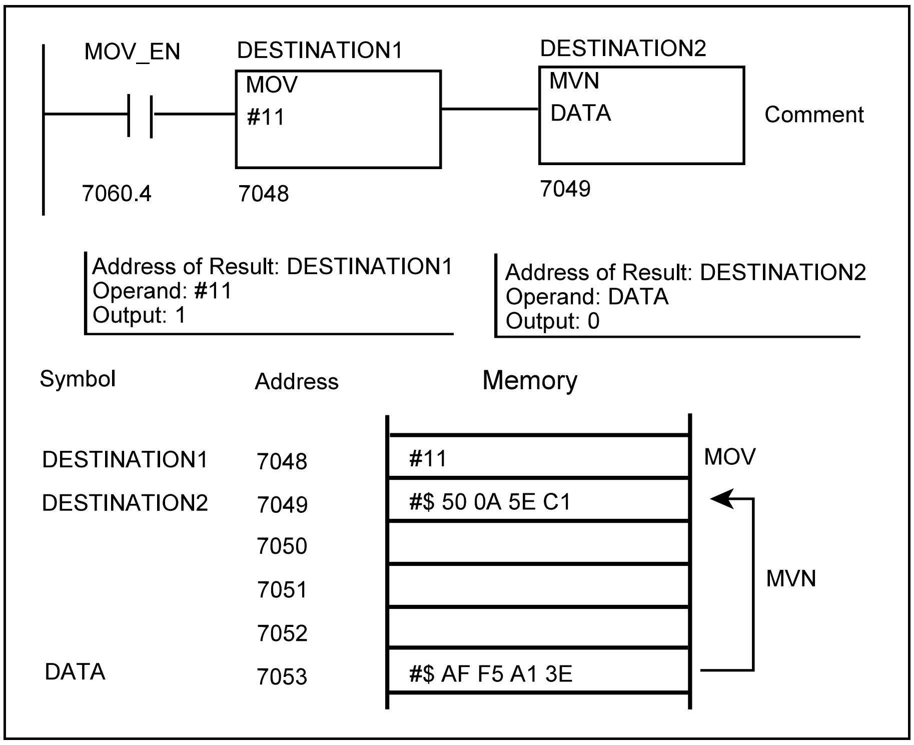
The

MVN

instruction moves the ***complement*** of a DWORD source to the destination register, if condition satisfied in its input. The source can be:

a hexadecimal number defined by #$ operator, or
a register.

You can write further instructions after this, if the parameter “Output” of instruction is enabled.



6.3.2 Movement of Floating-point data: the MOVF Instruction

The

MOVF

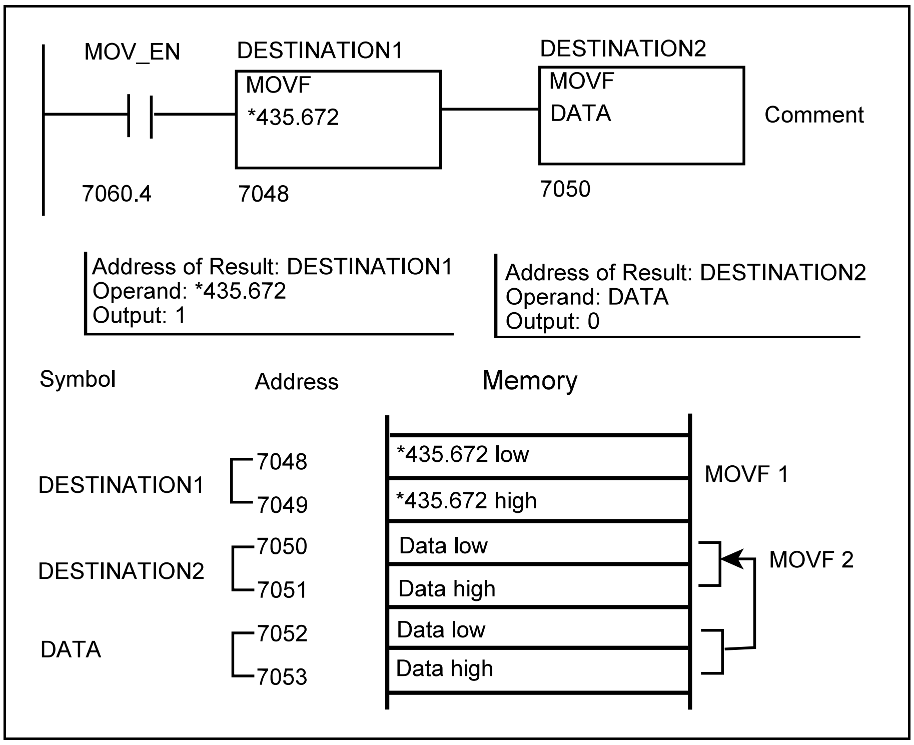
instruction moves two DWORD long double source to the destination register, if condition is satisfied in its input. The source can be:

double long floating point number defined by * operator, or

contents of 2 DWORD long registers.

☞ ***Attention!*** *You have to reserve 2 DWORD long registers also for Address of Result and Operand.*

You can write further instructions after this, if the parameter “Output” of instruction is enabled.



6.4 Timers

You can set different type of delays and different width of pulses in a PLC program with timers.

Input and output of timers

Each timer has its own

enable input.

The enable input triggers the operation of timer.

Each timer has its own

output.

The signal appears with delay in the output based on the operation of timer.

Each timers have common input parameters.

Input parameters of timers

– **Address of Timer**:

Range of the value: n...9999 (n: where starts user addresses)

The address of the timer can be a numeral or a symbol. Indexed addressing is possible.

The programmer has to define the timer in the users' memory space.

The address of timer points to the register which contains the actual value of the timer, where the timer is counting. You can read the actual value of a timer.

When the timer is triggered that is it receives a rising edge on its enable input, value defined by Set Value and Time Basis parameters is written into this register. This register can be overwritten anytime, even if the timer is working.

Always the value written in this register in milliseconds is counted down. If counting down is performed, then value of register will be 0 until it is triggered again.

☞ ***Attention:*** *The address, or symbolic name of a timer can be used as a contact. The value of condition is*

TRUE: if the value read at address ≠0,

FALSE: if the value read at address =0.

– **Set Value:**

Range of the value: 0...2^{31}

The value of timing.

It is an integer constant, or register reference. Indexed addressing is possible.

If the timer is triggered, this number is written in register defined in Address of Timer parameter.

– **Time Basis:**

Range of the value: 0, 1, 2, 3

It determines the interpretation of the Set Value parameter:

0: msec (millisecond)

1: sec (second)

2: min (minute)

3: hours (hour)

It is an integer constant, or a constant symbol (for example MSEC #0, SEC #1, MIN #2, HOURS #3), but it also can be a register reference. Indexed addressing is possible.

Range of the value of Set Value parameter refers to timings defined in msec unit. The timers always count with msec time base defined in Address of Timer register. If the value defined in parameter Time Basis deviate from msec, then you can define the following Set Value ranges:

=1 sec: 2147483

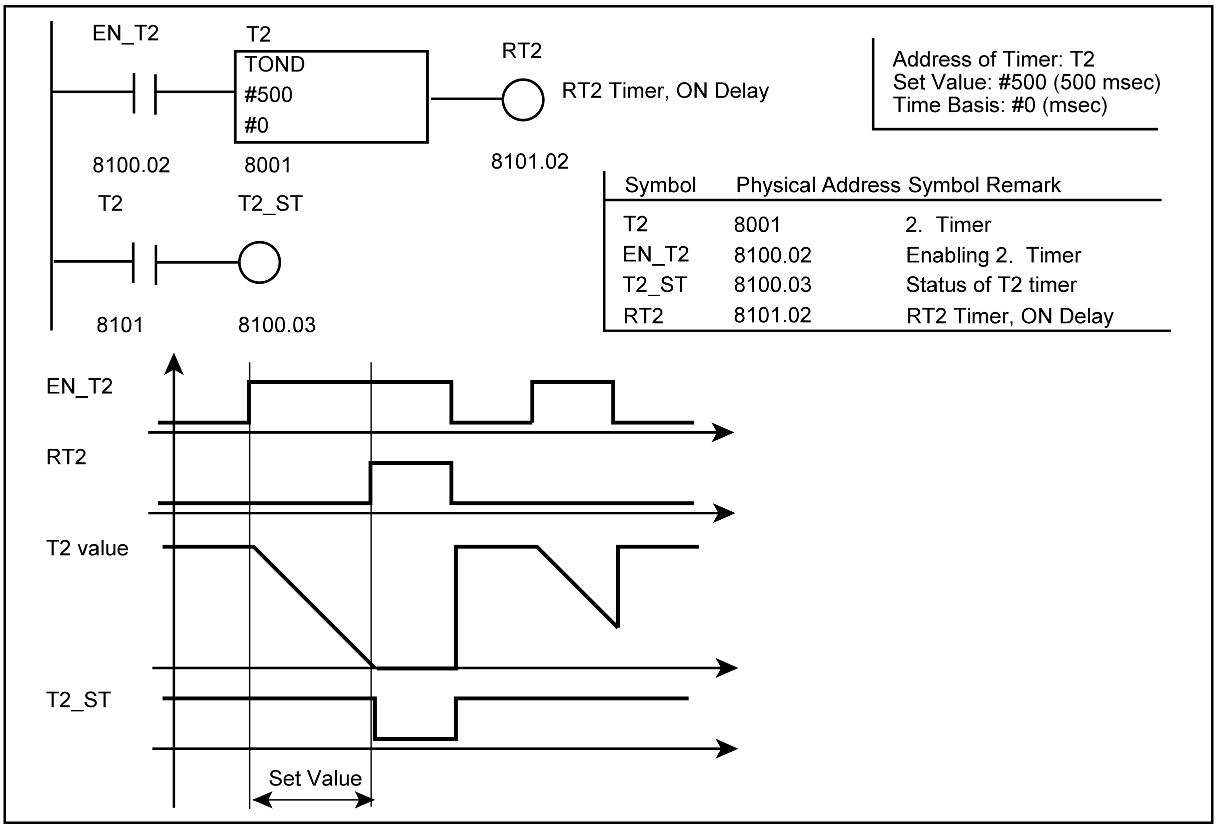
=2 min: 35791
=3 hours: 596

– **Remark:**
Not used.

6.4.1 On-delay Timer: TOND

If a TRUE value appears in its input, it appears delayed in the output.
When the input of the timer is triggered (its input is set to TRUE), then the timer starts to count down. After counting down, the output will be set to TRUE.
If the value of input is set to FALSE, the value specified in Set Value and Time Basis parameters is reloaded in the timer. The output of the timer is set to FALSE.
You can use the register Address of Timer as a contact. If the value of timer

≠0 the condition is TRUE (the contact is closed),
=0 the condition is FALSE (the contact is open).



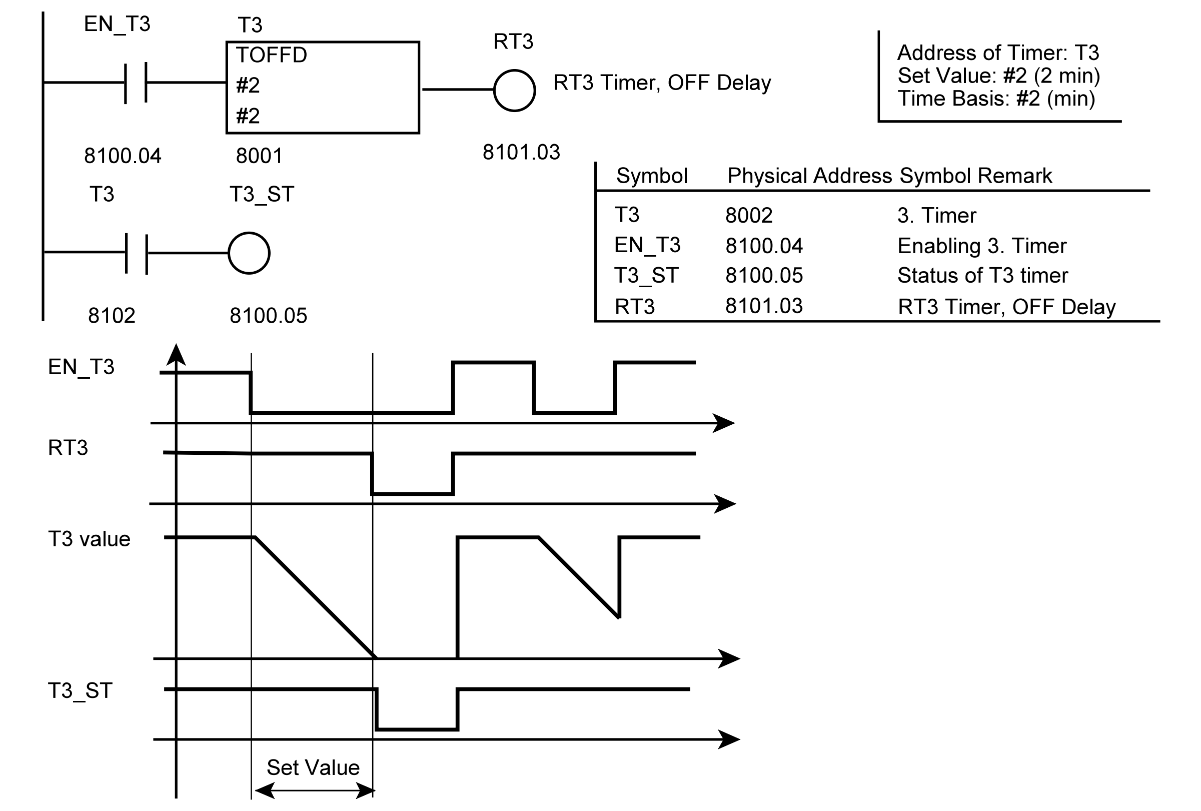
Symbol	Physical Address	Symbol Remark
T2	8001	2. Timer
EN_T2	8100.02	Enabling 2. Timer
T2_ST	8100.03	Status of T2 timer
RT2	8101.02	RT2 Timer, ON Delay

6.4.2 Off-delay Timer: TOFFD

If a FALSE value appears in its input, it appears delayed in the output.
When the input of the timer is triggered (its input is set to FALSE), then the timer starts to count down. After counting down, the output will be set to FALSE.
If the value of input is set to TRUE, the value specified in Set Value and Time Basis parameters is reloaded in the timer. The output of timer is set to TRUE.
You can use the register Address of Timer as a contact. If the value of timer

≠0 the condition is TRUE (the contact is closed),

=0 the condition is FALSE (the contact is open).



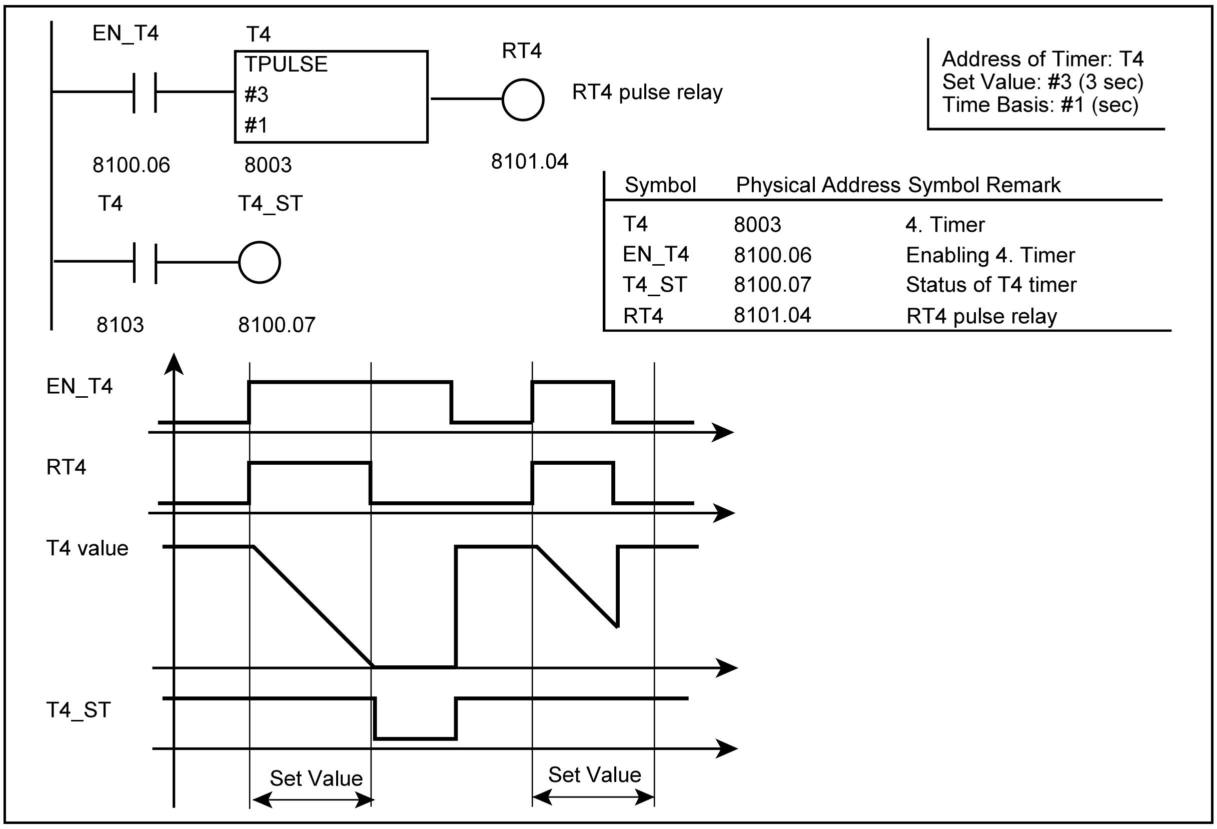
Symbol	Physical Address	Symbol Remark
T3	8002	3. Timer
EN_T3	8100.04	Enabling 3. Timer
T3_ST	8100.05	Status of T3 timer
RT3	8101.03	RT3 Timer, OFF Delay

6.4.3 Programmable Pulse Width: TPULSE

If a TRUE value appears in its input, its output goes TRUE immediately then after the delay set goes to FALSE.
When the input of the timer is triggered (its input is set to TRUE), then the timer starts to count down. After counting down, the output will be set to FALSE.
If the value of input is set to FALSE, the value specified in Set Value and Time Basis parameters is reloaded in the timer. The output of the timer is set to FALSE.
You can use the register Address of Timer as a contact. If the value of timer

$\neq$0 the condition is TRUE (the contact is closed),

=0 the condition is FALSE (the contact is open).



Symbol	Physical Address	Symbol Remark
T4	8003	4. Timer
EN_T4	8100.06	Enabling 4. Timer
T4_ST	8100.07	Status of T4 timer
RT4	8101.04	RT4 pulse relay

6.5 Counters

You can count pulses in a PLC program with counters.

Inputs and output of counters

Each counter has its own
clock signal (CK),
up/down (U/D) and a
reset (R)
input.

– **CK input**: The rising edge of pulse in CK input operates the counter.
– **U/D input**: The pulse in CK input modifies the value of counter:
increases, if the U/D input is FALSE, or
decreases, if the U/D input is TRUE.
– **R input**: If R input is TRUE, the counter is reset. The precedence of R input is higher than CK input: if its state is TRUE, the counter does not count, even if pulse received in CK input.
– **output** of counter is set TRUE, if the content of counter is equal to a pre-set number.

Each counters have common input parameters.

Input parameters of counters

– **Address of Counter**:
Range of the value: n...9999 (n: where starts user addresses)
The address of counter can be a numeral or a symbol. Indexed addressing is possible.
The programmer has to define the counter in the users' memory space.
The address of counter points to the register which contains the actual value of the counter, where the counter is counting. You can read the actual value of a counter.
When the input R of a counter is TRUE, the value defined in parameter Starting Value is written in this register. This register can be overwritten anytime, even if the counter is working.

☞ ***Attention:*** *The address, or symbolic name of a counter can be used as a contact. The value of condition is*
TRUE: if the value read at address ≠0,
FALSE: if the value read at address =0.

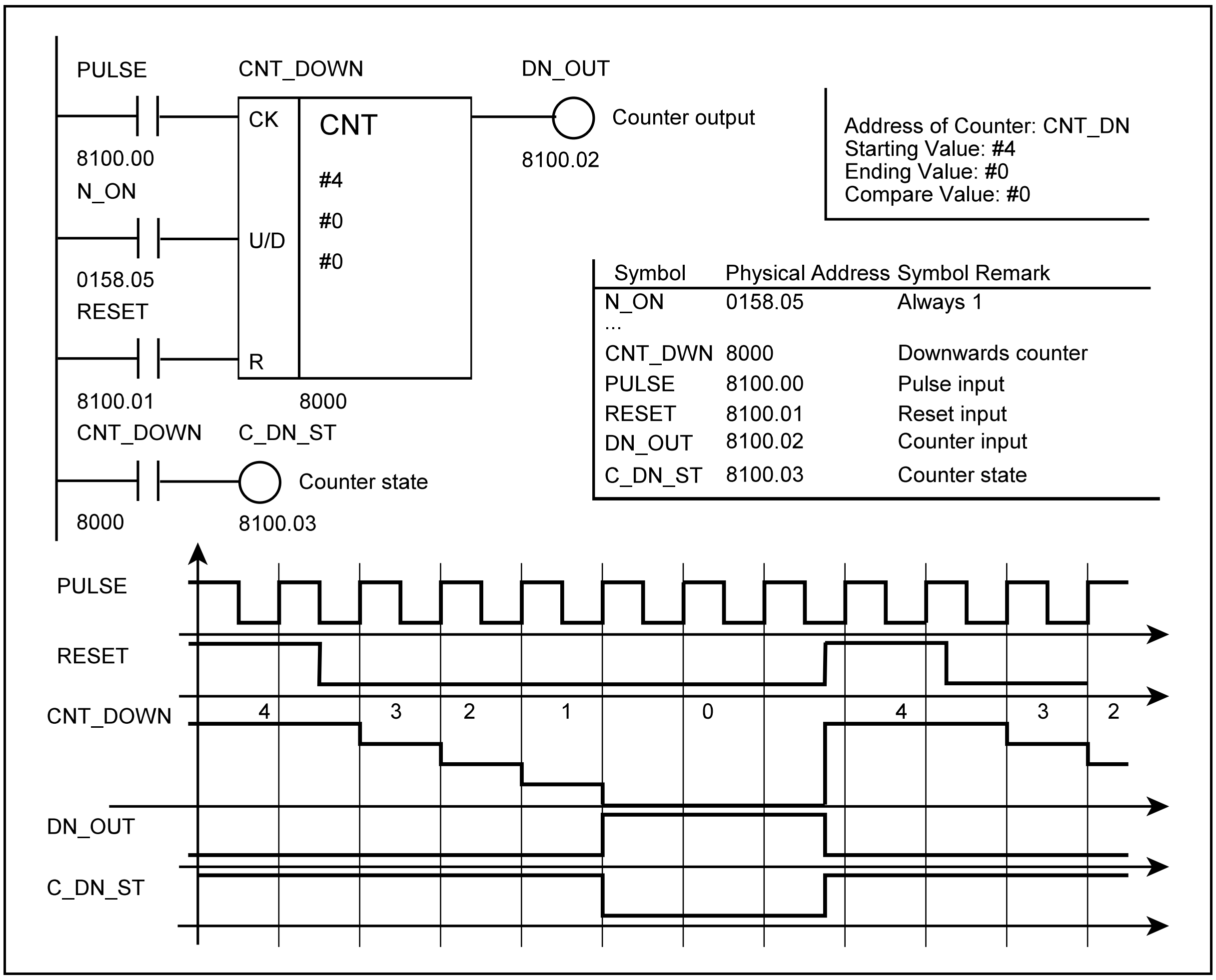
– **Starting Value:**
Range of the value: 0...2^{31}
It is an integer constant, or register reference. Indexed addressing is possible.
The initial value of a counter, that is set when input R is TRUE. This is the lower limit of rollover in ring counters.
– **Ending Value:**
Range of the value: 0...2^{31}
It is an integer constant, or register reference. Indexed addressing is possible.
Simple counters count until this value, this is the upper limit of rollover in ring counters.
– **Compare Value:**
Range of the value: 0...2^{31}
It is an integer constant, or register reference. Indexed addressing is possible.
The counter is in TRUE state, if the content of counter is equal to this value.
– **Remark:**
Note.

6.5.1 Simple Counter CNT

The simple counter counts until it reaches the value defined in parameter Ending Value. Then it stops, even pulses received in CK input. If reset input is set to TRUE, the counter restarts. There are two examples for using this counter.

Down counter

Set U/D input of the counter to TRUE by the flag N_ON (always TRUE). Then it counts down. The counter starts from 4 by RESET signal, because the Starting Value is 4 and it counts to 0, because Ending Value parameter is 0. It counts for the rising edge of CK input.
The output of the counter is set to TRUE, if its content becomes 0, because the Compare Value parameter is set to 0.
The counter restarts for the RESET signal.
You can use the register Address of Counter as a contact. If the value of counter
≠0 the condition is TRUE (the contact is closed),
=0 the condition is FALSE (the contact is open).



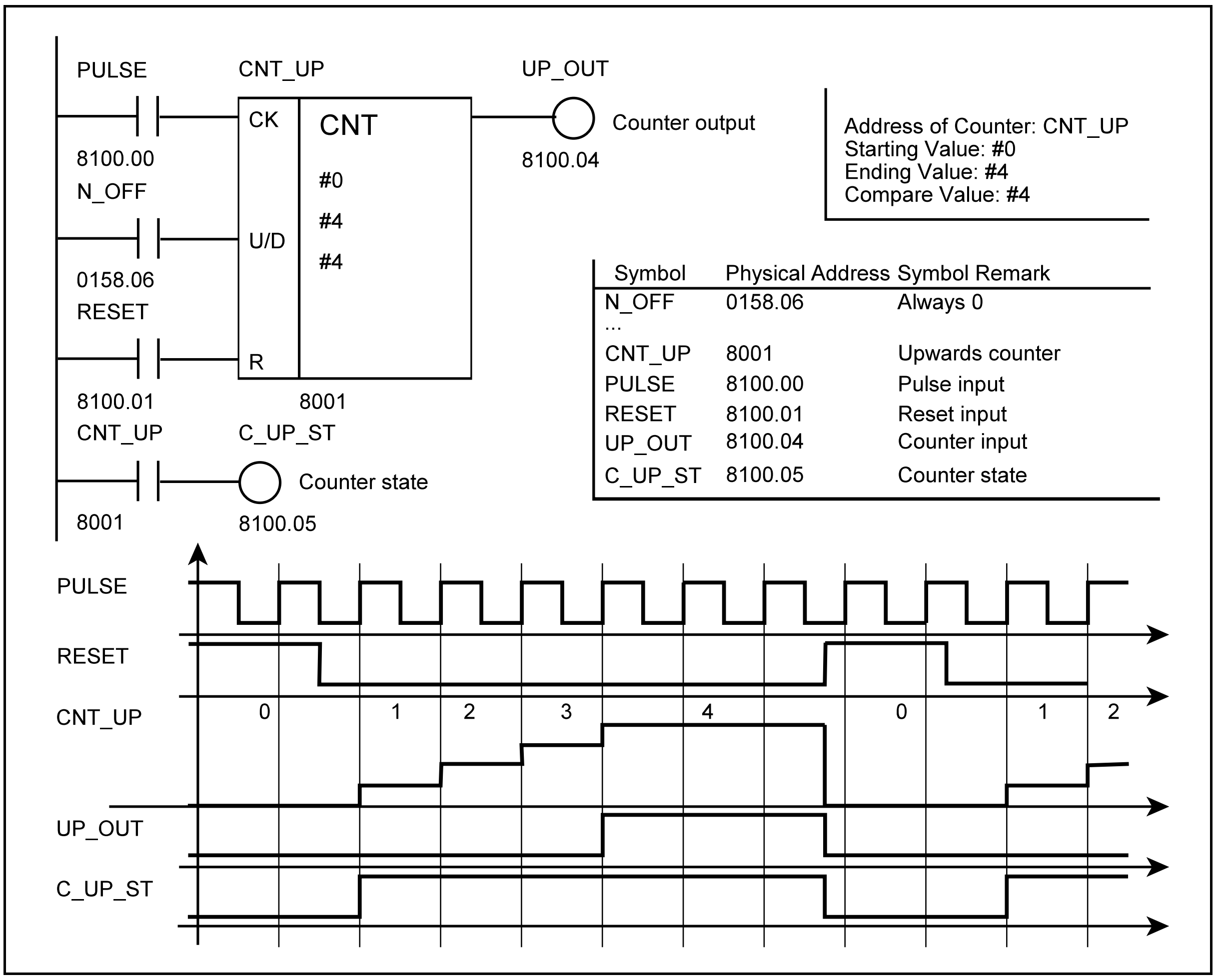
Address of Counter: CNT_DN
Starting Value: #4
Ending Value: #0
Compare Value: #0

Symbol	Physical Address	Symbol Remark
N_ON	0158.05	Always 1
...		
CNT_DWN	8000	Downwards counter
PULSE	8100.00	Pulse input
RESET	8100.01	Reset input
DN_OUT	8100.02	Counter input
C_DN_ST	8100.03	Counter state

Up counter

Set U/D input of the counter to FALSE by the flag N_OFF (always FALSE). Then it counts up. The counter starts from 0 by RESET signal, because the Starting Value is 0 and it counts up to 4, because Ending Value parameter is 4. It counts for the rising edge of CK input.
The output of the counter is set to TRUE, if its content becomes 4, because the Compare Value parameter is set to 4.
The counter restarts for the RESET signal.
You can use the register Address of Counter as a contact. If the value of counter

≠0 the condition is TRUE (the contact is closed),
=0 the condition is FALSE (the contact is open).

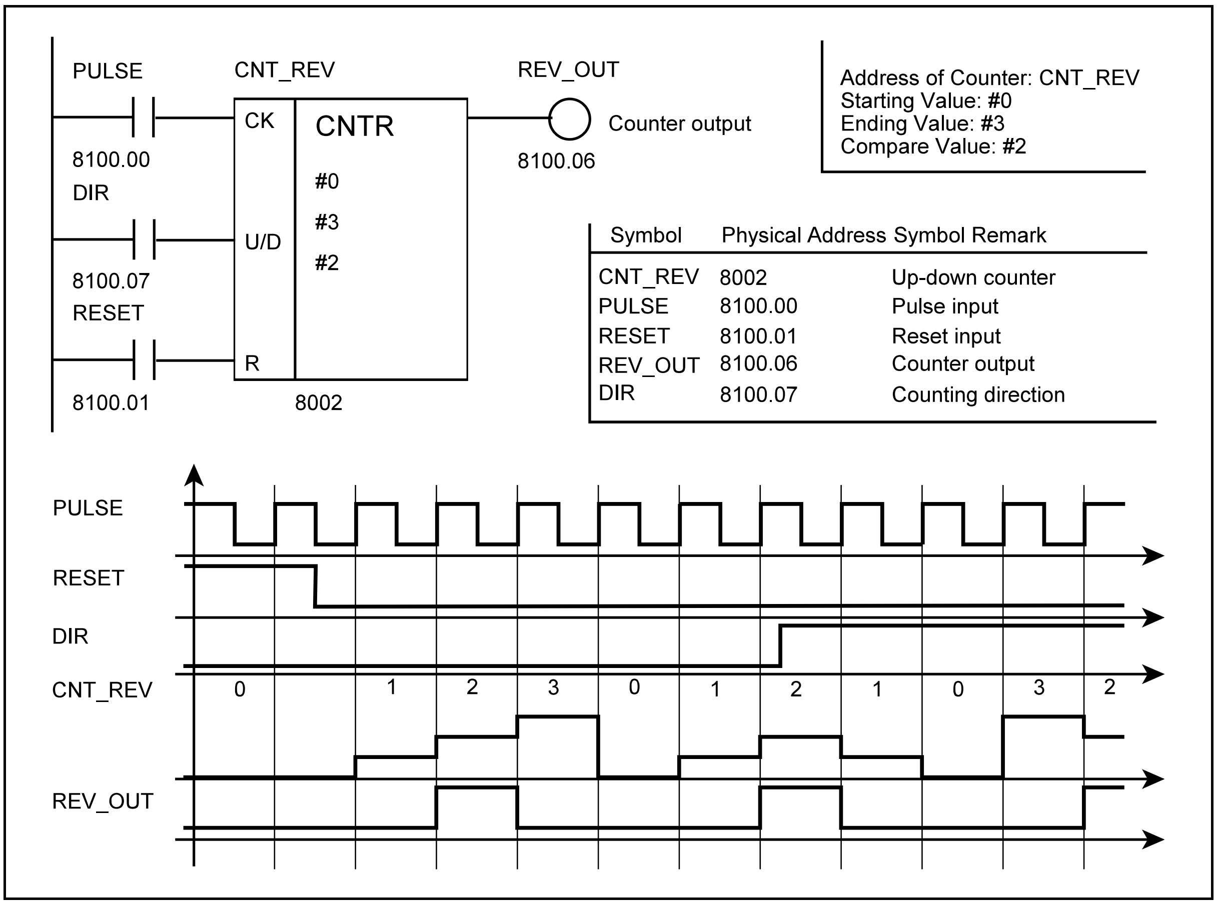


Address of Counter: CNT_UP
Starting Value: #0
Ending Value: #4
Compare Value: #4

Symbol	Physical Address	Symbol Remark
N_OFF	0158.06	Always 0
...		
CNT_UP	8001	Upwards counter
PULSE	8100.00	Pulse input
RESET	8100.01	Reset input
UP_OUT	8100.04	Counter input
C_UP_ST	8100.05	Counter state

6.5.2 Up-down Counter CNTR

If the up-down counter reaches the value defined in the Ending Value parameter while counting up, then it rolls over for the next pulse and continues counting from the value defined in the Starting Value parameter.
Contrary, if the counter reaches the value defined in the Starting Value parameter while counting down, then it rolls over for the next pulse and continues counting from the value defined in the Ending Value parameter.
If the reset (R) input is set to TRUE, the counter is set to the value defined in the Starting Value parameter.
The value defined in Compare Value parameter controls the output of counter. If the value of the counter is equal to this parameter, then the output is set to TRUE.
The following figure shows the operation of the up-down counter:



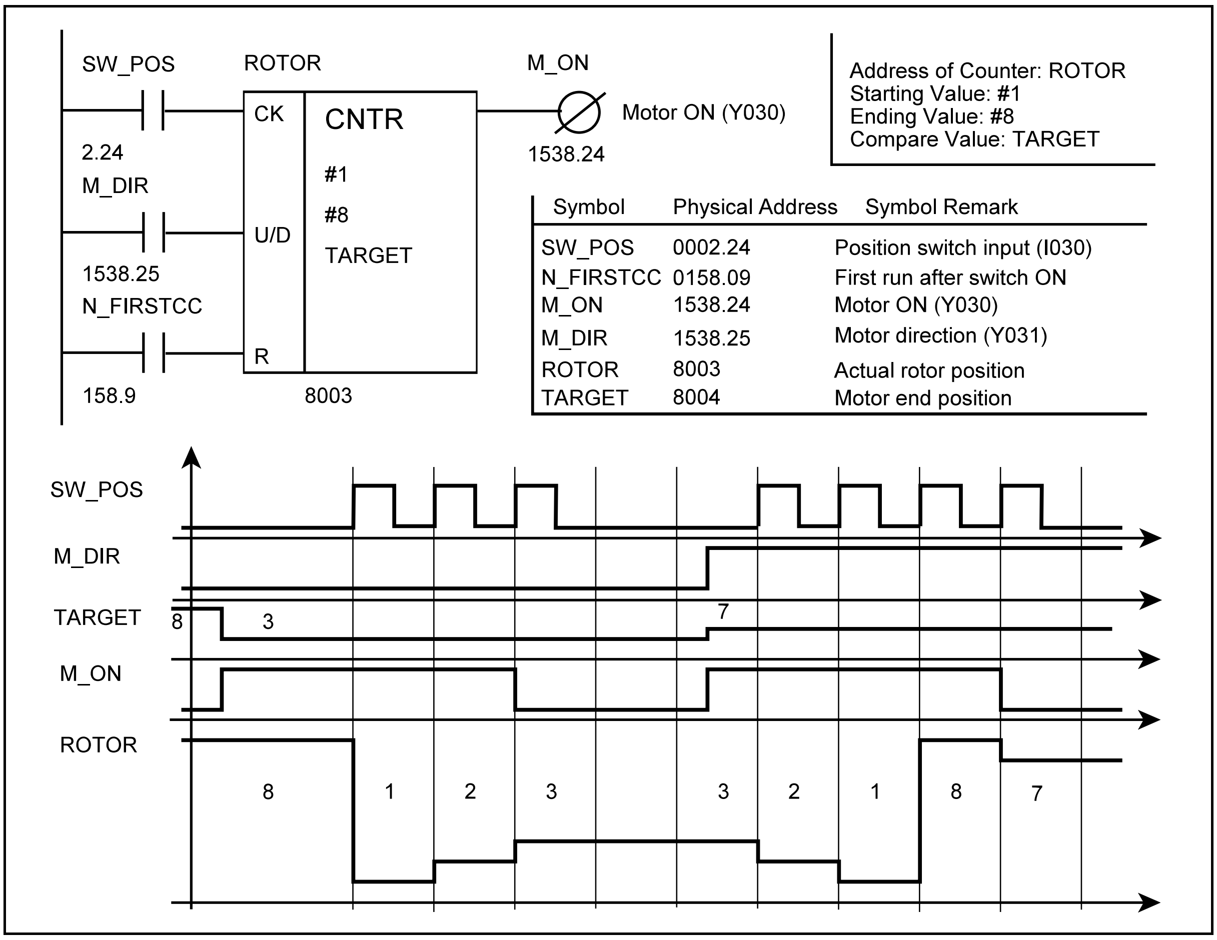
Address of Counter: CNT_REV
Starting Value: #0
Ending Value: #3
Compare Value: #2

Symbol	Physical Address	Symbol Remark
CNT_REV	8002	Up-down counter
PULSE	8100.00	Pulse input
RESET	8100.01	Reset input
REV_OUT	8100.06	Counter output
DIR	8100.07	Counting direction

Example for application of up-down counters

It is a frequently occurring task to move a rotating device into a determined position and record this position in PLC program. This device can be a turret, magazine or indexing table.
For example, you have a rotating device with 8 positions. Number these positions from 1 to 8. The Starting Value parameter of up-down counter is 1, the Ending Value parameter is 8. Let the Compare Value parameter of the counter be the symbol TARGET, where the position to go is written. The name of counter is ROTOR.
If PLC receives a new

, it overwrites the value of TARGET register. Since the value of counter is not equal to that of the Compare Value (TARGET position), then the output is set to FALSE state, it turns on the M_ON motor output. The output is turned on until the actual value of the counter becomes equal to value defined in TARGET register.
Connect motor direction signal of M_DIR to the U/D input of the counter and connect position switch of rotating device to the CK clock input.



Address of Counter: ROTOR
Starting Value: #1
Ending Value: #8
Compare Value: TARGET

Symbol	Physical Address	Symbol Remark
SW_POS	0002.24	Position switch input (I030)
N_FIRSTCC	0158.09	First run after switch ON
M_ON	1538.24	Motor ON (Y030)
M_DIR	1538.25	Motor direction (Y031)
ROTOR	8003	Actual rotor position
TARGET	8004	Motor end position

6.6 Rotation Control Instruction: ROT

The ROT instruction calculates that how many steps and in which direction to be moved a rotating device (for example turret, indexing table or magazine) to reach the destination defined by input parameters and properly set inputs.

Inputs and output of ROT instruction

The ROT instruction has got 5 inputs. These are:

– **EN input**: According to the input parameters and inputs set, on the rising edge of the enable (EN) input it calculates the value of position to go and sets the direction of rotation on its output.

– **BID input**: If the BID (bidirectional) input is FALSE the value of position to go is calculated according to the DIR input and the output is set in accordance with this input.
If the BID (bidirectional) input is TRUE the value of position to go is calculated in the shorter direction and the output is reset, if the shorter movement is positive, and set, if the shorter movement is negative.

– **DIR input:** If BID input is FALSE, the output is set in accordance with this input: If DIR=0 then the output is also 0, else it is 1. The value of position to go is calculated in accordance with the DIR input.

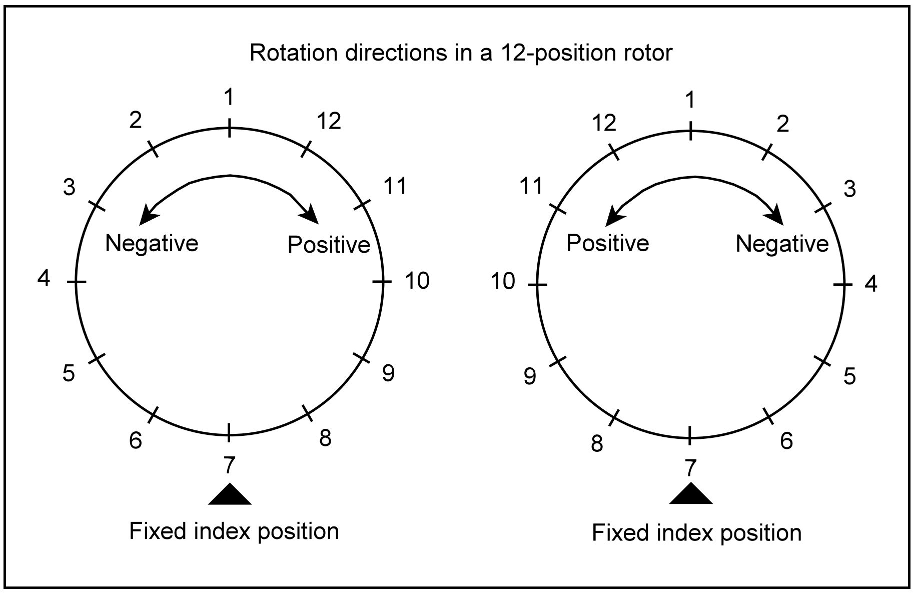
– **INC input**: If the input is FALSE, the value of position to go is calculated in absolute value, if the input is TRUE, the value of position to go is calculated incrementally, so it determines, how many steps is the device to be rotated.

– **PRE input**: If the input is FALSE, the value of position to go is equal to the goal position in INC=0 state, and in INC=1 state, the value of position to go is equal to the difference between goal position and current position.
If the input is TRUE, the value of position to go is equal to the first position before the goal position.

The **ROT** instruction has an output. The

– **output** shows in which direction is the rotor to be moved in function of state of BID and DIR inputs and in function of the content of registers Current Position and Goal Position.
If the output is in FALSE state in positive,
If the output is in TRUE state in negative direction.

If the instruction cannot be executed, then the instruction sets the error flag FL_ER and the output of instruction and the value of Position to Go register remain unchanged.



Input parameters of ROT instruction

– **Position to Go**:

Range of the value: 0...9999

The address of Position to Go register can be a numeral or a symbol. Indexed addressing is possible.

The programmer has to define it in the users' memory space.

This register is the output register of the instruction.

If the BID input is TRUE, the position to go value is calculated in the shorter way.

If the INC input is TRUE, the position to go value is calculated incrementally, so it determines, how many steps is the device to be rotated.

If the PRE input is TRUE, the value of position to go is equal to the first position before the goal position.

– **Current Position**:

Range of the value: 0...9999

The address of Current Position register can be a numeral or a symbol. Indexed addressing is possible.

The programmer has to define it in the users' memory space.

The register contains the ***current position*** of the rotor.

– **Goal Position**:

Range of the value: 0...9999

The address of Goal Position register can be a numeral or a symbol. Indexed addressing is possible.

The programmer has to define it in the users' memory space.

The register contains the ***goal position***. Usually it is a programmed value, for example a T code received from the NC.

If both INC and PRE inputs are FALSE, the Position to Go output register becomes equal to the value of Goal Position.

– **Starting Value**:

Range of the value: 0...2^{31}

It is an integer number, or a register reference. Indexed addressing is possible.

The starting or lowest position of the rotor, for example 1.

– **Ending Value**:

Range of the value: 0...2^{31}

It is an integer constant, or register reference. Indexed addressing is possible.

The end or highest position of the rotor, for example 12.

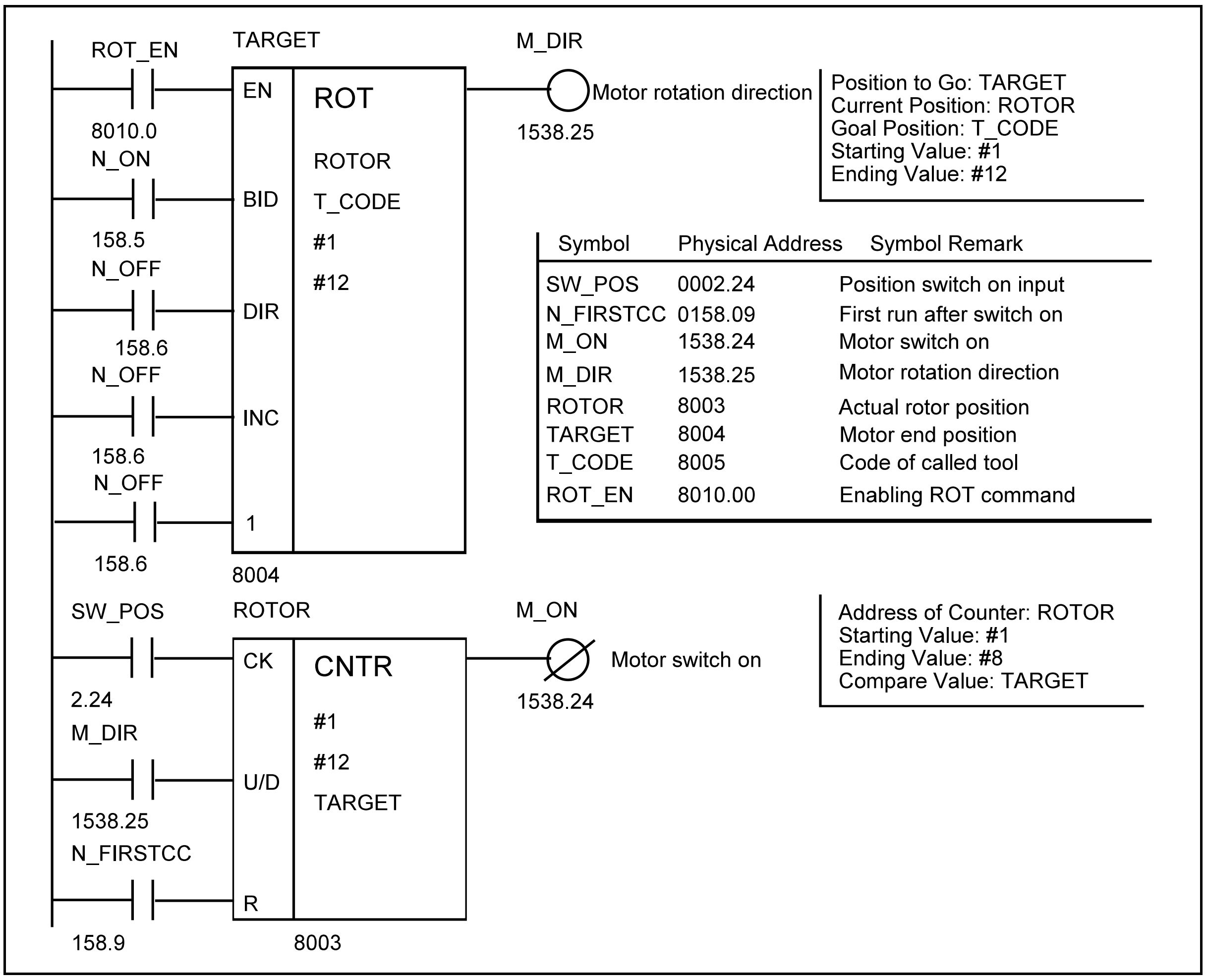
☞ ***Attention!***

If the defined addresses are out of its range of the value, then the instruction sets the error flag FL_ER and the output of instruction and the value of Position to Go register remain unchanged.

Example for application of ROT instruction

You have got a 12-position turret which can be rotated in both directions.
The position to go is calculated by ROT instruction and is saved into Position to Go register named TARGET.
The Current Position is the value of counter ROTOR.
The T code received from NC is saved in the Goal Position register named T_CODE.
The Starting and the Ending Values of the rotor are: #1 and #12.

The device rotates in both directions, so the BID input is TRUE.
You request the position to go value in absolute form, so INC input is FALSE.
You do not want to stop one position before the destination, so PRE input is FALSE.
The turret rotation starts after ROT_EN flag is written to 1. The value of position to go and the direction of movement is calculated by ROT instruction, written to TARGET register, and set the proper motor rotation direction M_DIR.
Since the value of Compare Value (TARGET) of ROTOR counter has changed, its output will be FALSE, which turns on M_ON motor switch output. The motor starts to rotate in the direction determined by M_DIR output, and it rotates until the content of ROTOR counter reaches the TARGET value. Then the output of counter is turned to TRUE state, it turns off M_ON output, which stops the motor.



Position to Go: TARGET
Current Position: ROTOR
Goal Position: T_CODE
Starting Value: #1
Ending Value: #12

Symbol	Physical Address	Symbol Remark
SW_POS	0002.24	Position switch on input
N_FIRSTCC	0158.09	First run after switch on
M_ON	1538.24	Motor switch on
M_DIR	1538.25	Motor rotation direction
ROTOR	8003	Actual rotor position
TARGET	8004	Motor end position
T_CODE	8005	Code of called tool
ROT_EN	8010.00	Enabling ROT command

Address of Counter: ROTOR
Starting Value: #1
Ending Value: #8
Compare Value: TARGET

6.7 Data Shift Instructions

6.7.1 Shift Register: SHTR

You can move a bit pattern to the right or left by using shift register.

Inputs and output of SHTR instruction

A shift register has got the following inputs:
clock signal (CK),
left/right (L/R) and
data (D).

– **CK input**: The ***rising edge*** of the pulse received in CK input of register shifts the bit pattern in register.

– **L/R input**: The pulse received in CK input of register shifts the bit pattern in register:
to the ***left*** by one bit, if ***L/R*** input is ***FALSE*** or
to the ***right*** by one bit, if ***L/R*** input is ***TRUE***.

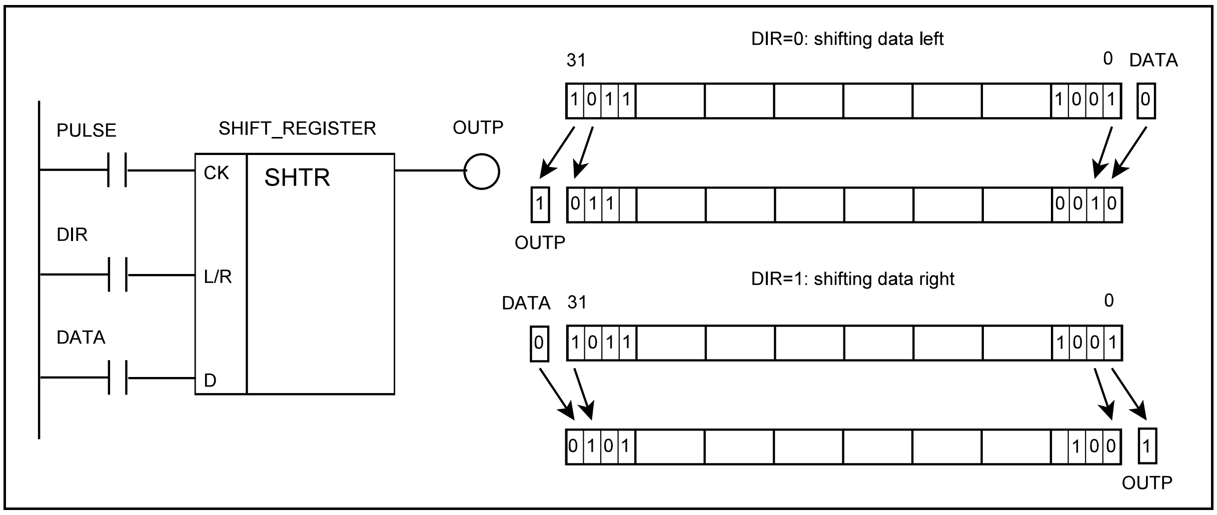
– **D input**: The state of data input (D) determines the value of the ***incoming bit***. If the input is
FALSE then the incoming bit is 0, if it is
TRUE, then the incoming bit is 1.
Shifting left the rightmost bit (0),
shifting right the leftmost bit (31)
is replaced by the incoming data bit..
The
– **ouput** of the shift register becomes the value of the ***outgoing bit***:
shifting left the leftmost bit (31),
shifting right the rightmost bit (0).
Its value is FALSE, if the value of leaving bit is 0, and TRUE, if the value of leaving bit is 1.

Input parameter of shift register

– **Address of Shift Register**:
Range of the value: 0...9999
The Address of Shift Register can be a numeral or a symbol. Indexed addressing is possible.
The programmer has to define the shift register in users' memory space.
The address of shift register shows the address containing the actual value of register. The register works here, and you can read the actual value of register anytime. The same correspond for any bit of the register. You can rewrite it anytime, also during the operation of the register.

☞ ***Attention!***
If the address defined in Address of Shift Register parameter is out of its range of the value, then the instruction sets error flag FL_ER.

– **Remark:**
Note.



6.7.2 Shift Instructions: ASHL, ASHR

There are two arithmetic shift instructions: the ***ASHL*** shifts the bit pattern ***left***, the ***ASHR*** shifts the bit pattern ***right***.
Each instruction has an

enable input, and an

output.

The selected operation is executed if the enable input is TRUE.
If the ***enable input*** is ***TRUE*** and ***the operation executed without error*** then the **output** of instruction is set to ***TRUE***, otherwise, the output will be FALSE. The output can be used for enabling a following, for example an arithmetic operation.
You can assign a continuous array consisting of n pieces double words, by defining the starting and the ending address of the array. In addition, you can determine that by how many bits the bit pattern will be shifted simultaneously. The outgoing bits will be lost, and zeros will come in into the empty spaces.
When shifting left, the outgoing bits are the upper bits of the double word specified on Ending Address and the zeros will come in into the lower bits of the double word specified on Starting Address.
When shifting right, the outgoing bits are the lower bits of the double word specified on Starting Address and the zeros will come in into the upper bits of the double word specified on Ending Address.

Input parameters of shift instructions

– **Starting Address**:

Range of the value: 0...9999

The Starting Address of the array can be a numeral or a symbol. Indexed addressing is possible.

– **Ending Address:**

Range of the value: 0...9999

The Ending Address of the array can be a numeral or a symbol. Indexed addressing is possible.

☞ ***Attention!***

The following condition must be true for the starting and ending address:

Starting Address≤Ending Address

If the upper condition is not satisfied or the defined addresses are out of its range of the value then the instruction sets error flag FL_ER and the output of instruction will be FALSE.

If the values of starting and ending addresses are equal then the instruction is executed only for a double word.

– **Shift Value:**

Range of the value: 0...2^{31}

It is an integer constant, or register reference. Indexed addressing is possible.

In this parameter you can specify by how many bits the bit pattern is to be shifted.

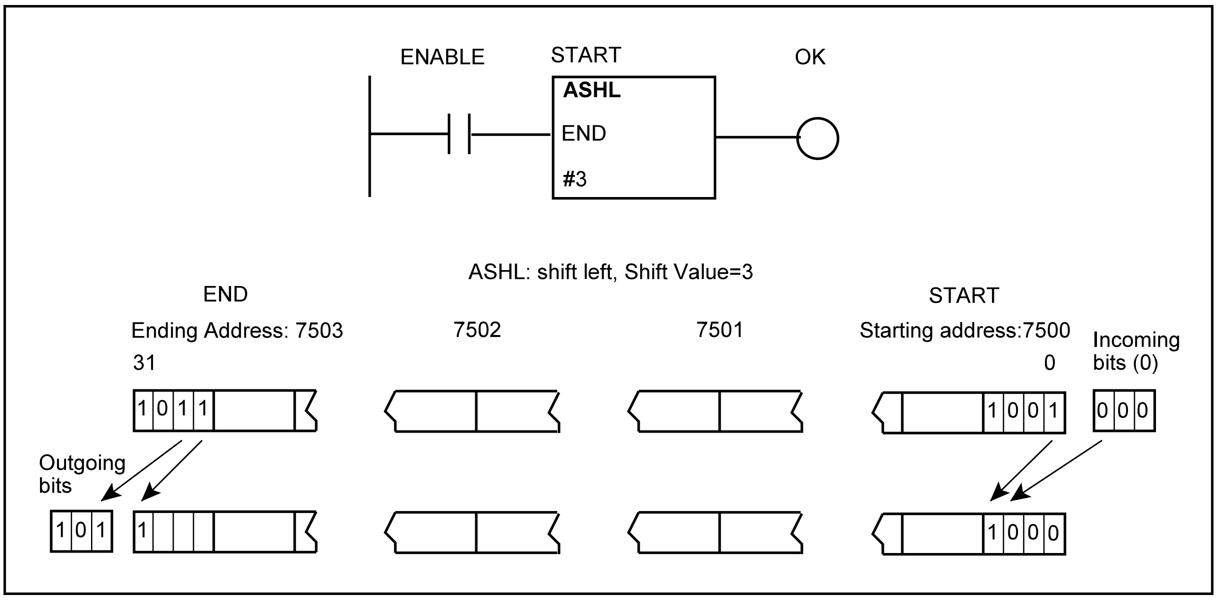
– **Output**:

You can connect further instructions to the output of the instruction box, in case of its enabled state.

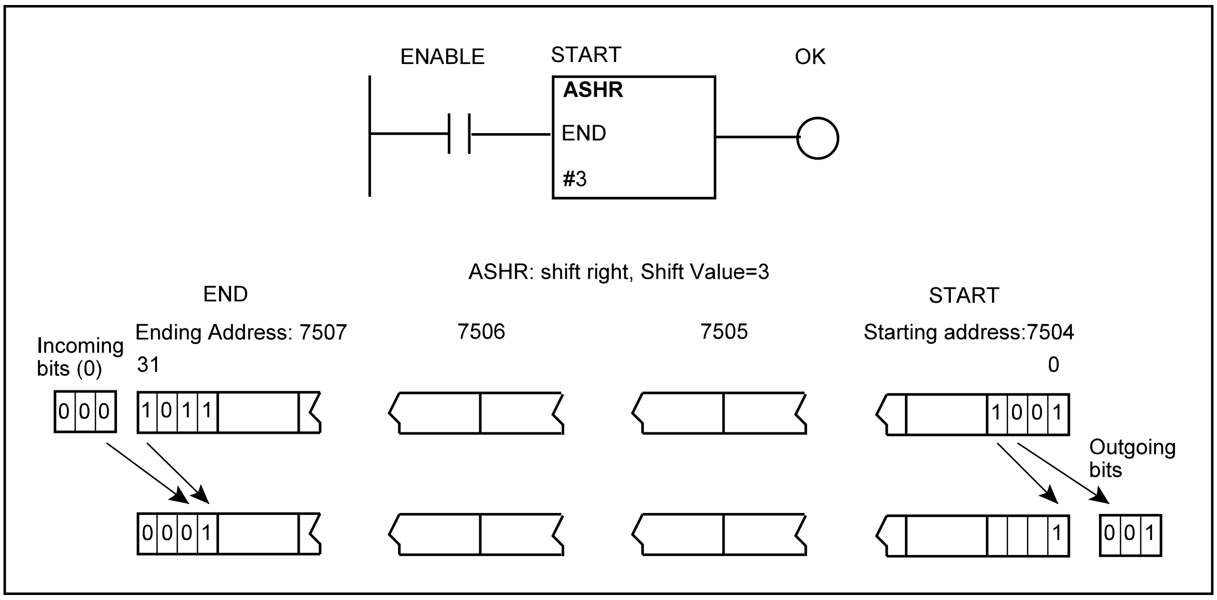
– **Remark:**

Note.

The operation of ASHL instruction (shift left by any bits):



The operation of ASHR instruction (shift right by any bits):



6.7.3 Rotate Instructions: ARTL, ARTR

The arithmetic rotate instructions rotate the assigned bit pattern, left ***(ARTL)*** or right ***(ARTR)***. Each instruction has an

enable input, and an

output.

If the ***enable input*** is ***TRUE*** and the ***instruction executed without error*** then the ***output*** of instruction will be ***TRUE***, otherwise the output will be FALSE. The output can be used for example enabling the next instruction (for example an arithmetic instruction).
You can assign a continuous array consisting of n pieces double words by defining the starting and the ending address of the array. You can also define by how many bits the bit pattern is to be rotated by the instruction. The outgoing bits in rotation will come in on the other side.
When rotating left, the outgoing bits are the upper bits of the double word specified on Ending Address and they will come in into the lower bits of the double word specified on Starting Address.
When rotating right, the outgoing bits are the lower bits of the double word specified on Starting Address and they will come in into the upper bits of the double word specified on Ending Address.

Input parameters of rotate instructions

– **Starting Address**:

Range of the value: 0...9999

The Starting Address of the array can be a numeral or a symbol. Indexed addressing is possible.

– **Ending Address:**

Range of the value: 0...9999

The Ending Address of the array can be a numeral or a symbol. Indexed addressing is possible.

☞ ***Attention!***

The following condition must be true for the starting and ending address:

Staring Address≤Ending Address

If the upper condition is not satisfied or the defined addresses are out of its range of the value then the instruction sets error flag FL_ER and the output of instruction will be FALSE.

If the values of starting and ending addresses are equal then the instruction is executed only for a double word.

– **Rotation Value:**

Range of the value: 0...2^{31}

It is an integer constant, or register reference. Indexed addressing is possible.

In this parameter you can specify by how many bits the bit pattern is to be rotated .

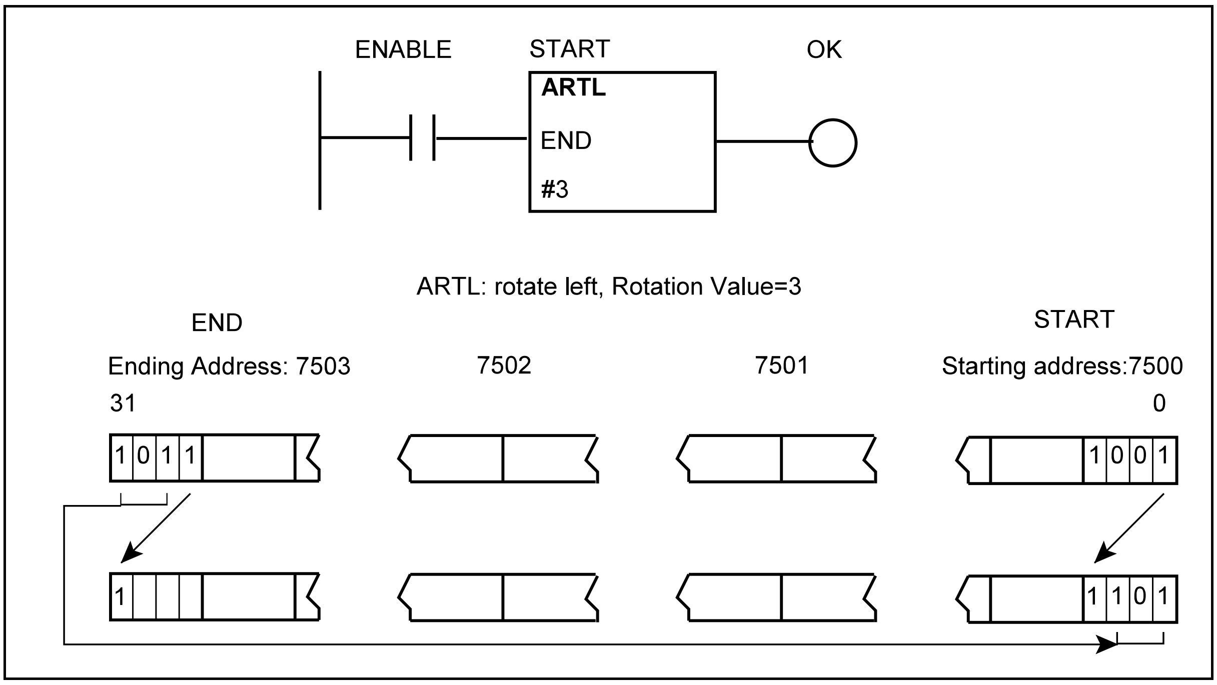
– **Output**:

You can connect further instructions to the output of the instruction box, in case of its enabled state.

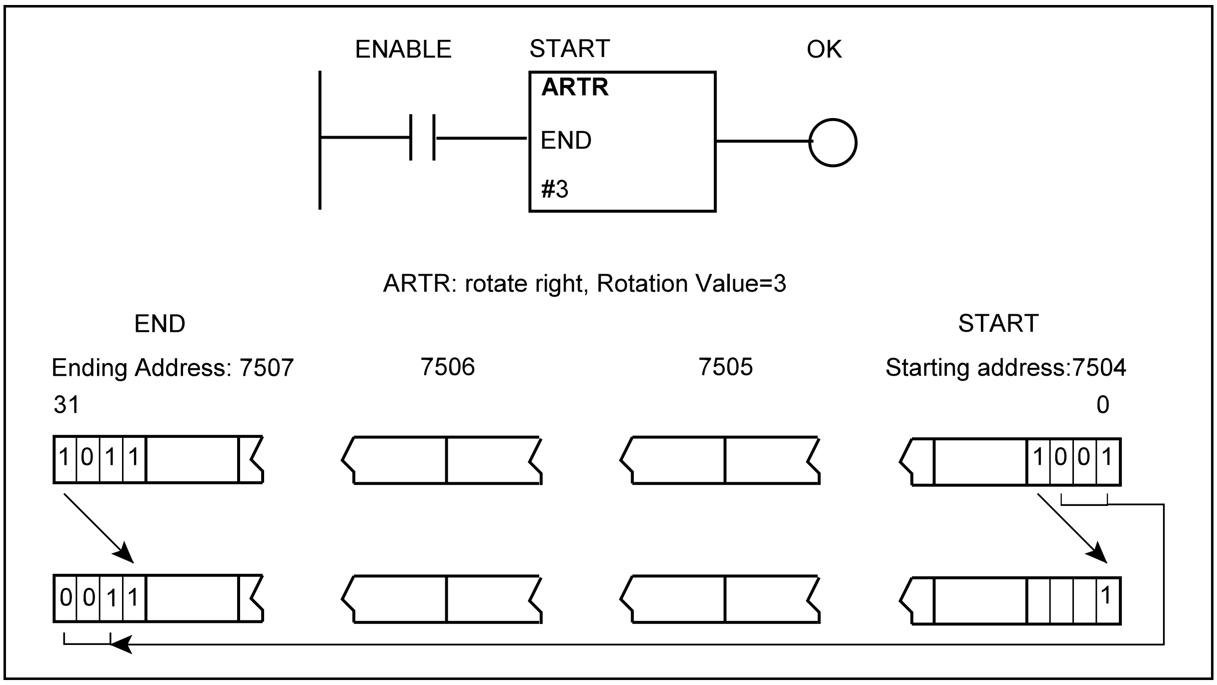
– **Remark:**

Note.

The operation of ARTL instruction (rotate left by any bits):



The operation of ARTR instruction (rotating right by any bits):



6.8 Logic Instructions

You can execute logic and, or, exclusive or negation operations between 32 bit data by using logic instructions.

6.8.1 Complement Instruction: NEG

The **NEG** instruction inverts each bit of a 32 bit register bit by bit. Turns all OFF bits ON and turns all ON bits OFF.
The instruction has an

enable input, and an
output.

The output of instruction will be FALSE, if its input is FALSE, or the instruction cannot be executed. Then the instruction sets error flag FL_ER.

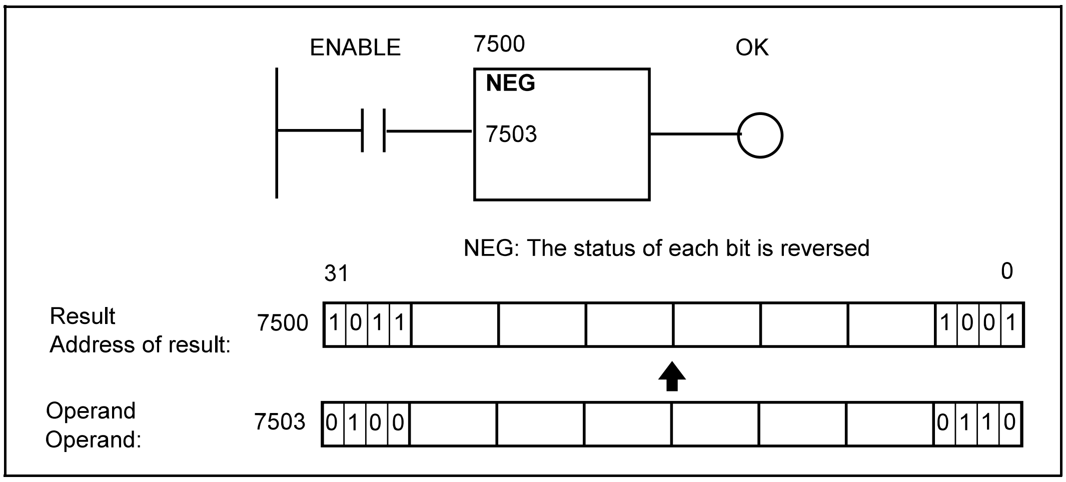
Input parameters of NEG instruction

– **Address of Result**:
Range of the value: 0...9999
The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

– **Operand**:
The data to be complemented can be
a hexadecimal constant defined by #$ operator, or
a DWORD long register content.
If the Operand is a register, its address can be a numeral or a symbol. Indexed addressing is possible.

☞ ***Attention!***
If the defined addresses are out of their range of the value then the instruction set FL_ER error flag and the output of instruction will be FALSE.

– **Output**:
You can connect further instructions to the output of the instruction box, in case of its enabled state.



6.8.2 Two-operand Instructions: AND, OR, XOR

The

AND instruction realizes AND function between two 32 bit data, the
OR instruction realizes OR function between two 32 bit data, the
XOR instruction realizes EXCLUSIVE OR function between two 32 bit data
bit by bit.

The instructions have an

enable input, and an
output.

The output of instruction will be FALSE, if its input is FALSE, or the instruction cannot be executed. Then the instruction sets error flag FL_ER.

Input parameters of instructions

– **Address of Result**:

Range of the value: 0...9999
The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

– **1st Operand**:

The first operand of instruction.
The 1st Operand can only be a register.
Its address can be a numeral or a symbol. Indexed addressing is possible.

– **2nd Operand**:

The second operand of instruction.
The 2nd Operand can be
a hexadecimal constant defined by #$ operator, or
a DWORD long register content.
If the 2nd Operand is a register, its address can be a numeral or a symbol. Indexed addressing is possible.

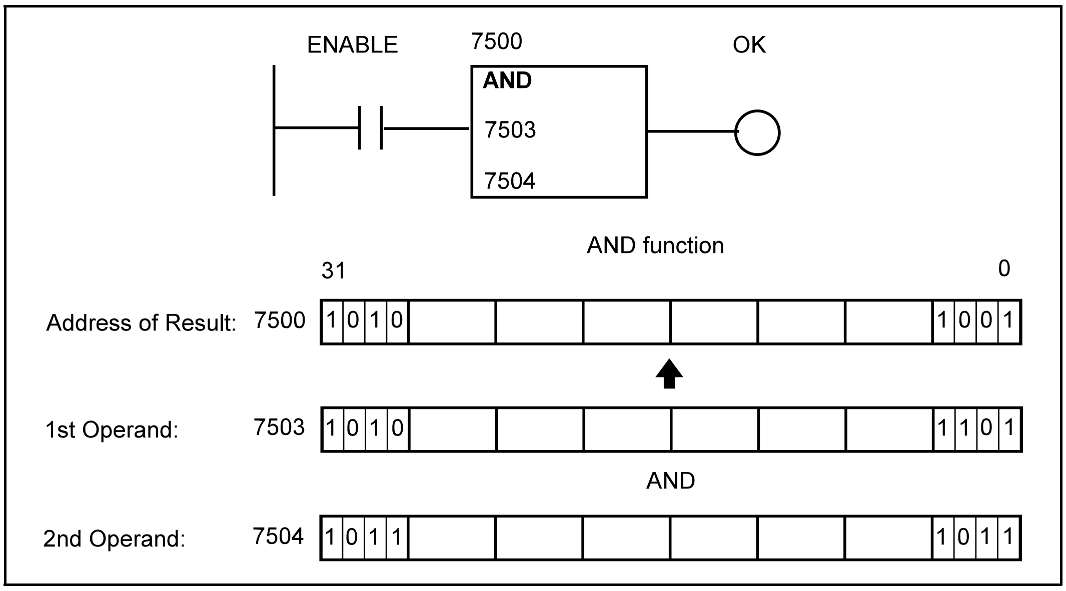
☞ ***Attention!***

If the defined addresses are out of their range of the value then the instruction sets error flag FL_ER and the output of instruction will be FALSE.

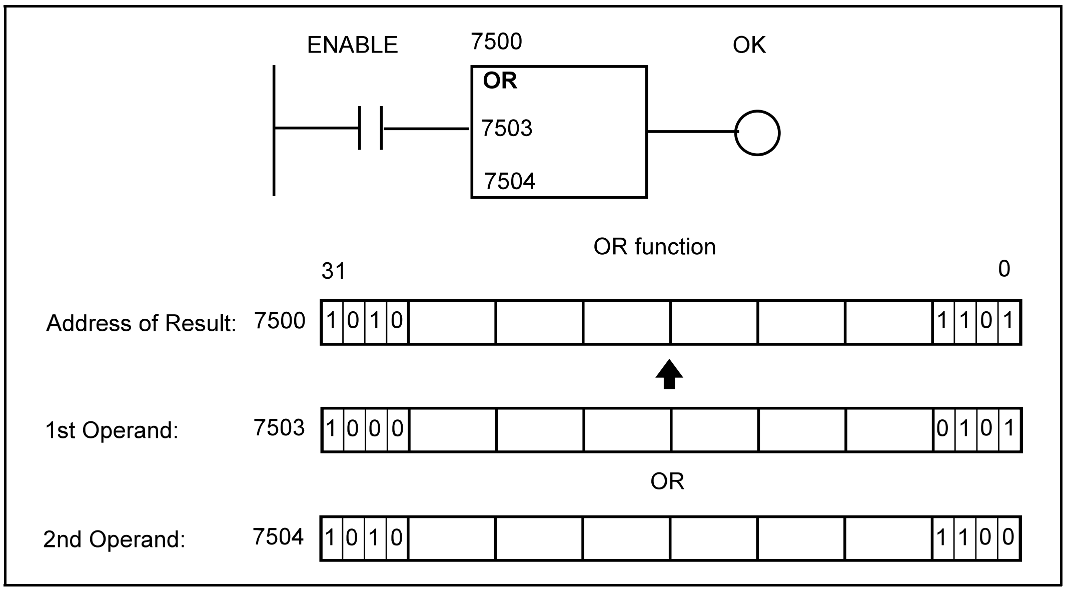
– **Output**:

You can connect further instructions to the output of the instruction box, in case of its enabled state.

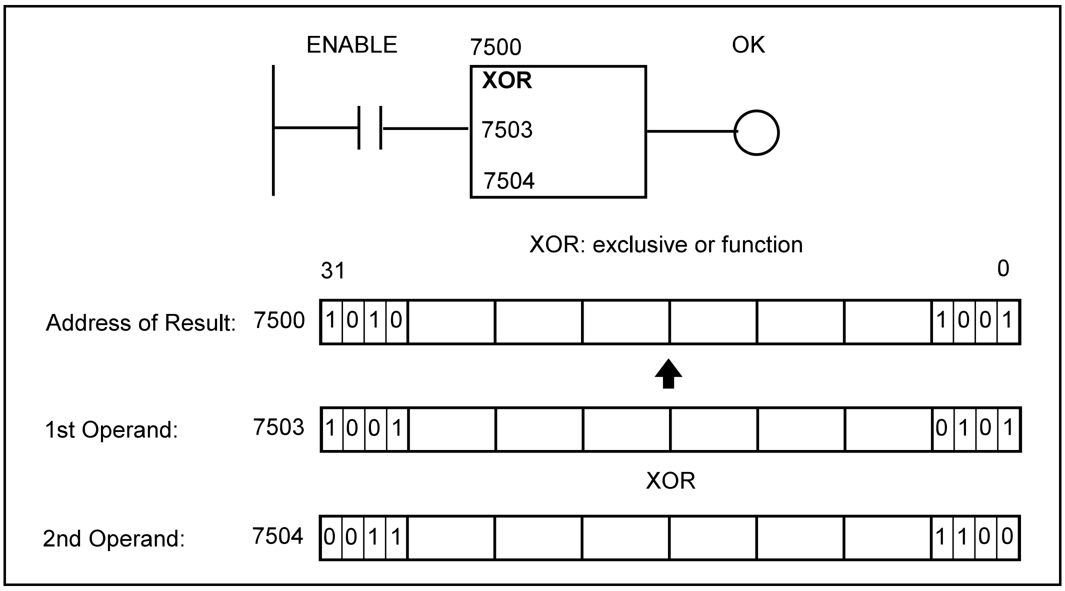
If the adequate bits of both operands are TRUE, then the resulting bit of **AND** instruction is TRUE:



If the adequate bits any of the operands is TRUE, then the resulting bit of **OR** instruction is TRUE:



If the adequate bits of both operands are different from each other, then the resulting bit of **XOR** instruction is TRUE.



6.9 Integer Arithmetic Instructions

The integer arithmetic instructions are adding, subtracting, multiplying and dividing double word numbers.
Each instruction has an

enable input, and an

output.

The output of instruction will be FALSE, if its input is FALSE, or the instruction cannot be executed. Then the instruction sets error flag FL_ER.
Integer arithmetic instructions also set the following flags:

FL_UF: underflow
FL_OF: overflow
FL_CY: carry.

The input parameters of instructions

– **Address of Result**:

Range of the value: 0...9999
The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

– **1st Operand**:

The first operand of instruction.
The 1st Operand can only be a register.
Its address can be a numeral or a symbol. Indexed addressing is possible.

– **2nd Operand**:

The second operand of instruction.
The 2nd Operand can be
a decimal or hexadecimal constant defined by # or #$ operator, or
a DWORD long register content.
If the 2nd Operand is a register, its address can be a numeral or a symbol. Indexed addressing is possible.

☞ ***Attention!***

If the defined addresses are out of their range of the value then the instruction sets error flag FL_ER and the output of instruction will be FALSE.

– **Output**:

You can connect further instructions to the output of the instruction box, in case of its enabled state.

6.9.1 Signed, Integer Addition, without Carry: ADD

*Addition without carry means, the carry (**FL_CY=1**), generated before the execution of the current ADD instruction, **is not added** to the addend of the current addition.*

The

result saved in register defined at address of result, the
augend is the 1st Operand, the
addend is the 2nd Operand.

FL_CY carry flag is set if a **carry** is created by the addition, so the result cannot be stored in 32 bits, else its value is 0.

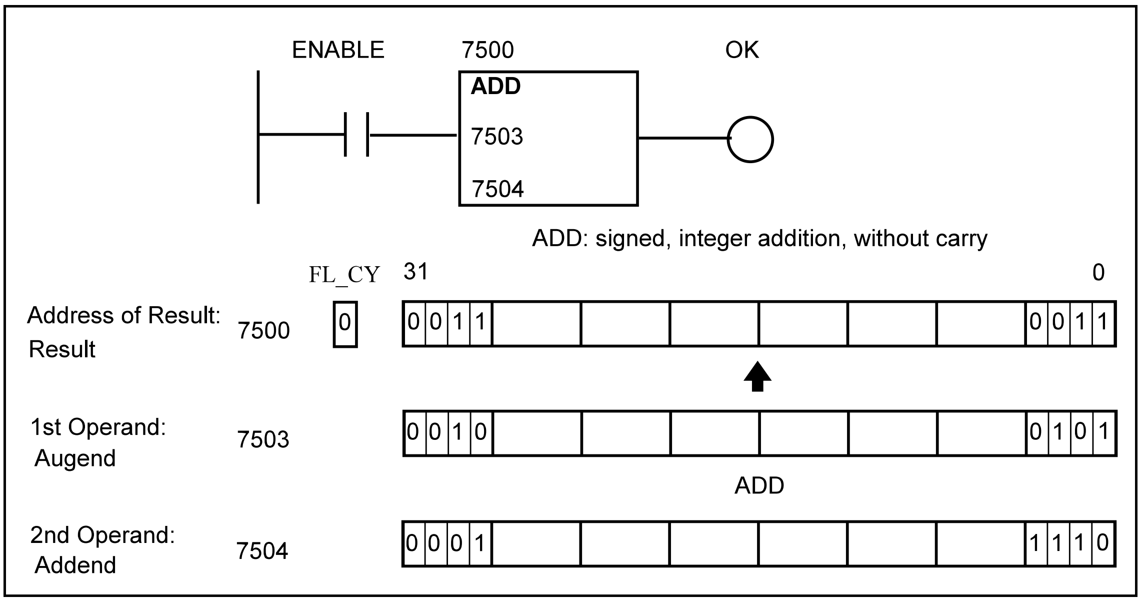
```
              7000000F
             +FC000012
FL_CY=1       6C000021
```

FL_OF overflow flag is set if the result of addition of two positive numbers (value of 31st bit is 0 in each number) is in the range of negative numbers 80000000...FFFFFFFF, else its value is 0.

```
                    7FFFFFF0
                   +000000FF
FL_CY=0, FL_OF=1    800000EF
```

FL_UF underflow flag is set if the result of addition of two negative numbers (value of 31st bit is 1 in each number) is in the range of positive numbers 00000000...07FFFFFF, else its value is 0.

```
                    8000FC22
                   +80F032A1
FL_CY=1, FL_UF=1    00F12EC3
```



6.9.2 Signed, Integer Subtraction, without Carry: SUB

*Subtraction without carry means, the carry (**FL_CY=1**), generated before the execution of the current SUB instruction, is not subtracted from the minuend of the current subtraction.*

The

result is saved in register defined at address of result, the
minuend is the 1st Operand, the
subtrahend is the 2nd Operand.

FL_CY carry flag is set if a **borrow** is created by the subtraction, else its value is 0:

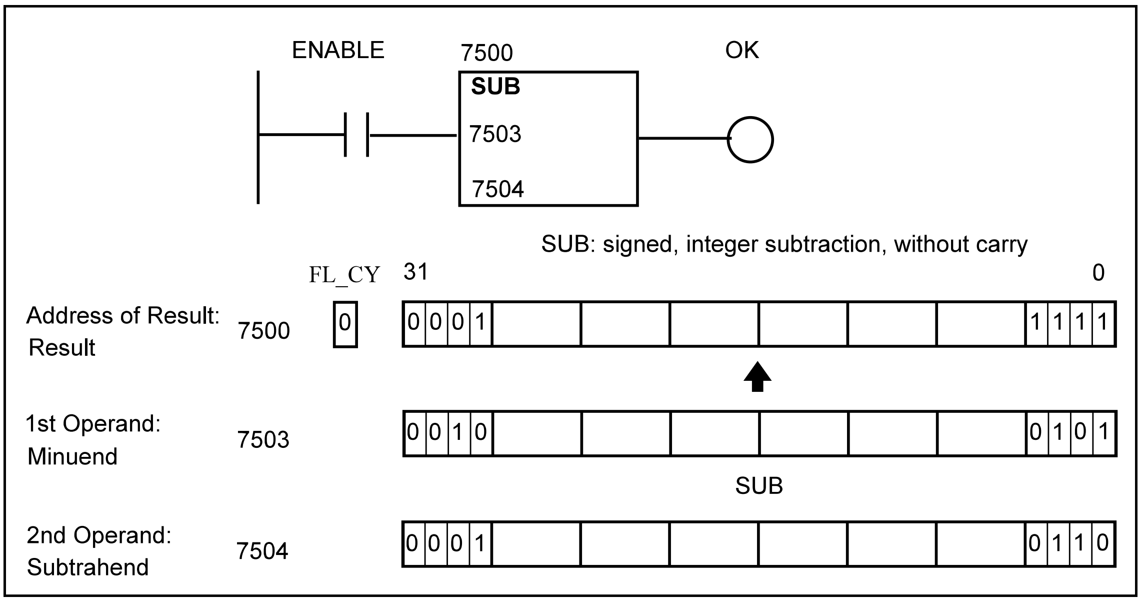
```
                 2A41FB53
                -B145E681
FL_CY=1          78FC14D2
```

FL_OF overflow flag is set if the result of subtracting a negative number from a positive number is in the range of negative numbers 80000000...FFFFFFFF, else its value is 0.

```
                        7FFFFF00
                       -FFFFF000
FL_CY=1, FL_OF=1        80000F00
```

FL_UF underflow flag is set if the result of subtracting a positive number from a negative number is in the range of positive numbers 00000000...07FFFFFF, else its value is 0.

```
                        800000FF
                       -0000A000
FL_CY=0, FL_UF=1        7FFF60FF
```



6.9.3 Signed, Integer Multiplication: MUL

The signed, integer multiplication (MUL) can be used when the result can be represented in 32 bits (DWORD). Else the instruction sets overflow flag FL_OF.

The

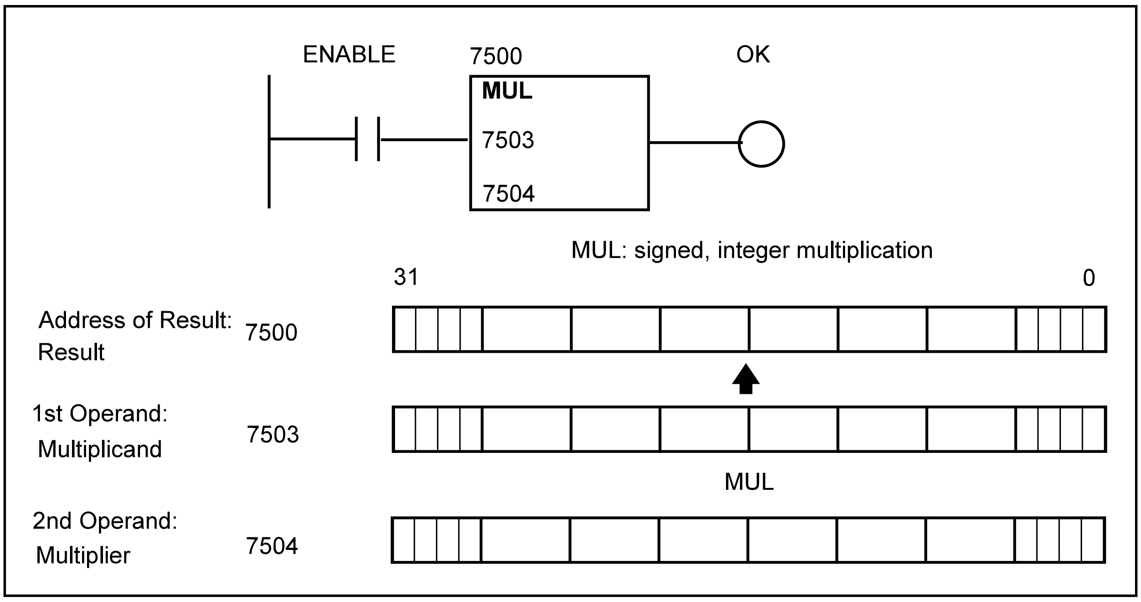
> **result** is saved in register defined at address of result, the
> **multiplicand** is the 1st Operand, the
> **multiplier** is the 2nd Operand register.

FL_OF overflow flag is set if the result cannot be represented in 32 bits:

```
           7000000F
          ×0C000012
FL_OF=1    9400010E
```

Because the MUL instruction is a signed operation, overflow is not generated in the following case:

```
           FFFFFFFF
          ×00000002
FL_OF=0    FFFFFFFE
```


ENABLE
7500
MUL
7503
7504
OK
MUL: signed, integer multiplication
31
0
Address of Result: Result
7500
1st Operand: Multiplicand
7503
MUL
2nd Operand: Multiplier
7504

6.9.4 Signed, Integer Division: DIV

The signed, integer division (DIV) can be used when the remainder of division is not needed. The

> **quotient** saved in register defined at address of result, the
> **dividend** is the 1st Operand, the
> **divisor** is the 2nd Operand.

The result of division is equal to 0, if the divisor is larger than the dividend:

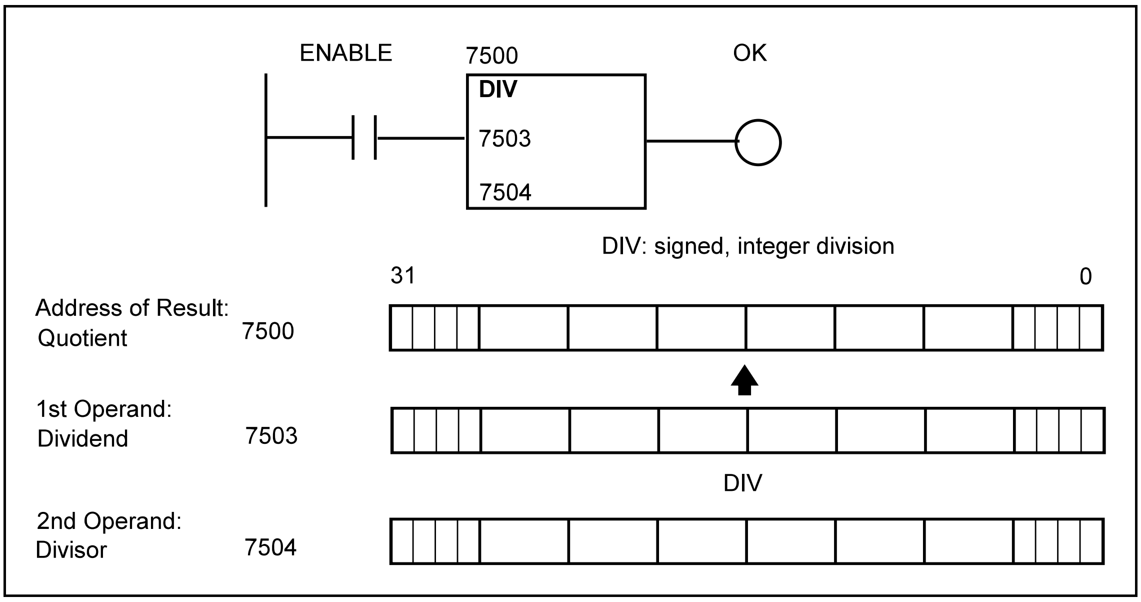
```
 0000002A
÷000000FF
---------
 00000000
```

The result of division will be a negative number, either the divisor or the dividend is negative:

```
 00000002
÷FFFFFFFF
---------
 FFFFFFFE
```

☞ ***Attention!***

*If the value of divisor is 0, the result of division will be 0 and the error flag **FL_ER** is set!*



6.10 Floating-point Mathematical Instructions

The instructions have an

enable input, and an

output.

The output of instruction will be FALSE, if its input is FALSE, or the instruction cannot be executed. Then the instruction sets error flag FL_ER.

The floating-point instructions also set the following flags:

FL_UF: underflow

FL_OF: overflow

Input parameters of instructions

– **Address of Result**:

Range of the value: 0...9998

The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

– **Operands**:

Floating-point instructions can be single-operand or two-operand instructions. For example single-operand instruction is the square-root, two-operand instruction is the addition.

The Operand can be

a floating-point constant defined by * operator, or

the content of two consecutive DWORD registers.

If the Operand is a register, its address can be a numeral or a symbol. Indexed addressing is possible. Indexing always refer to the lower DWORD addresses.

☞ ***Attention!***

If the defined addresses are out of their range of the value then the instruction sets error flag FL_ER and the output of instruction will be FALSE.

– **Output**:

You can connect further instructions to the output of the instruction box, in case of its enabled state.

Flags operated by the instructions

FL_OF overflow flag is set, if the absolute value of result is so high that it cannot be represented as a double precision floating-point number.

FL_UF underflow flag is set, if the absolute value of result is so low that it cannot be represented as a double precision floating-point number.

6.10.1 Floating-point Addition: +F

The result saved in register defined at address of result.

– **1st Operand**

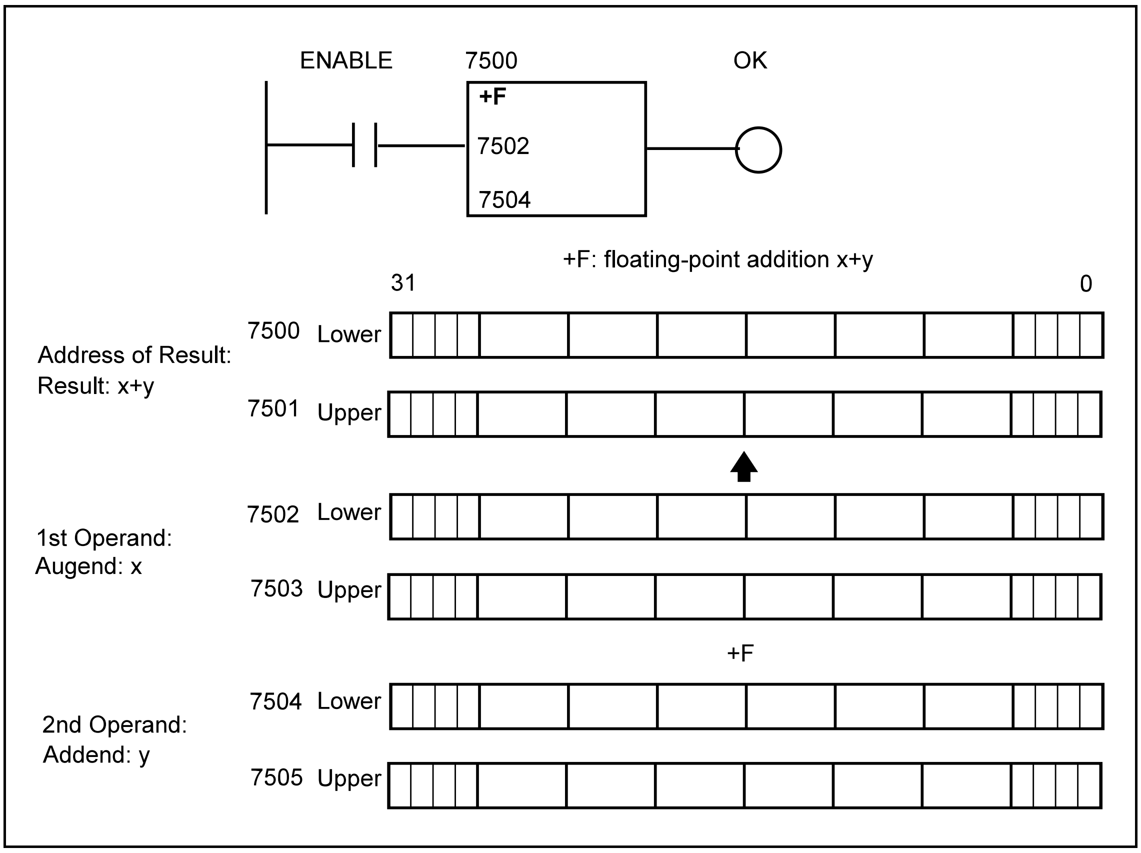
The value of the augend.

The address of a register or a constant. Indexed addressing is possible.

– **2nd Operand**

The value of the addend.

The address of a register or a constant. Indexed addressing is possible.



6.10.2 Floating-point Subtraction: -F

The difference saved in register defined at address of result.

– **1st Operand**

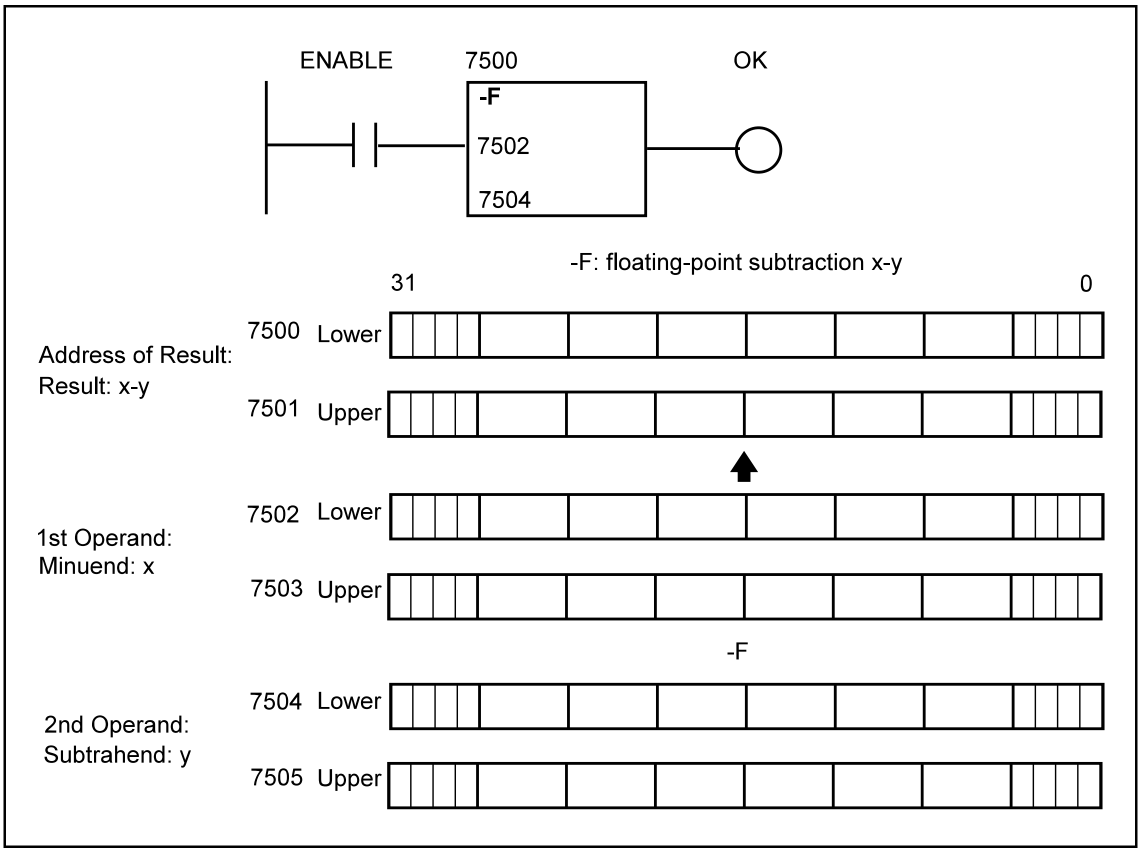
The value of the minuend.

The address of a register or a constant. Indexed addressing is possible.

– **2nd Operand**

The value of the subtrahend.

The address of a register or a constant. Indexed addressing is possible.



6.10.3 Floating-point Multiplication: *F

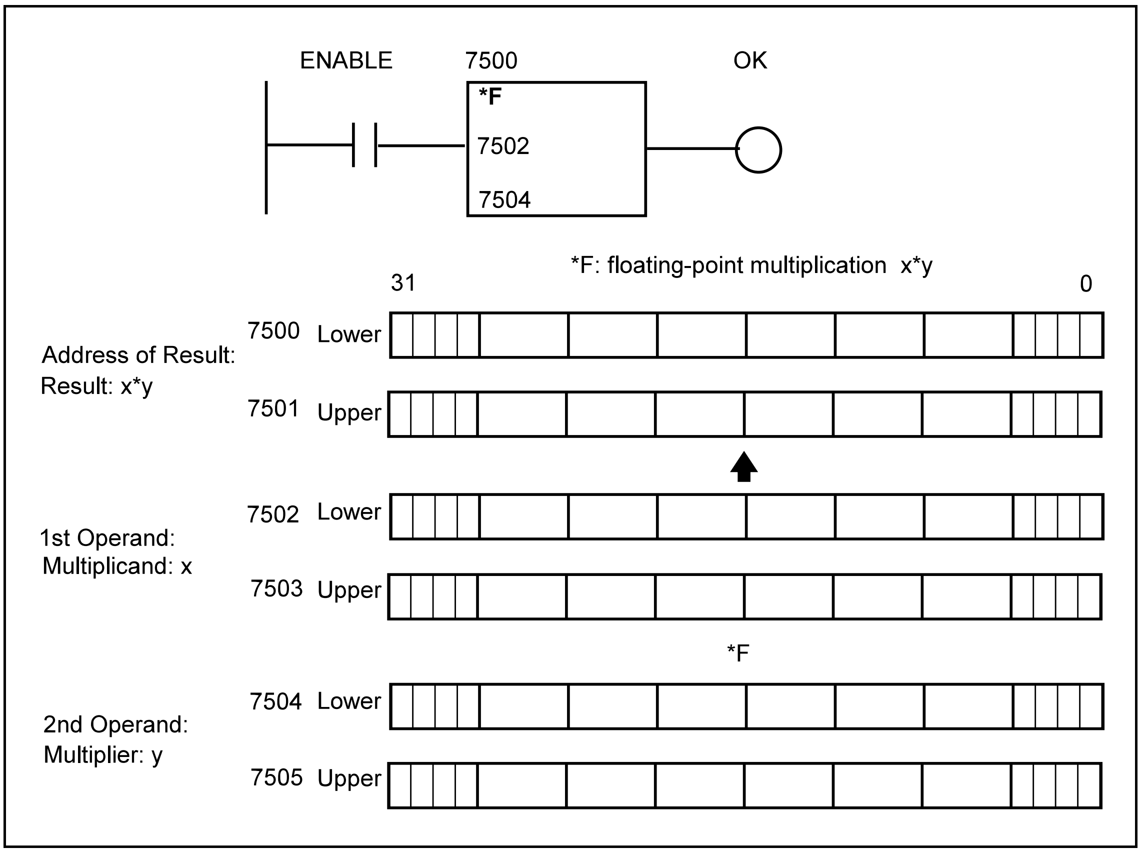
The multiplication saved in register defined at address of result.

– **1st Operand**

The value of the multiplicand.
The address of a register or a constant. Indexed addressing is possible.

– **2nd Operand**

The value of the multiplier.
The address of a register or a constant. Indexed addressing is possible.



6.10.4 Floating-point Division: /F

The quotient saved in register defined at address of result.

– **1st Operand**

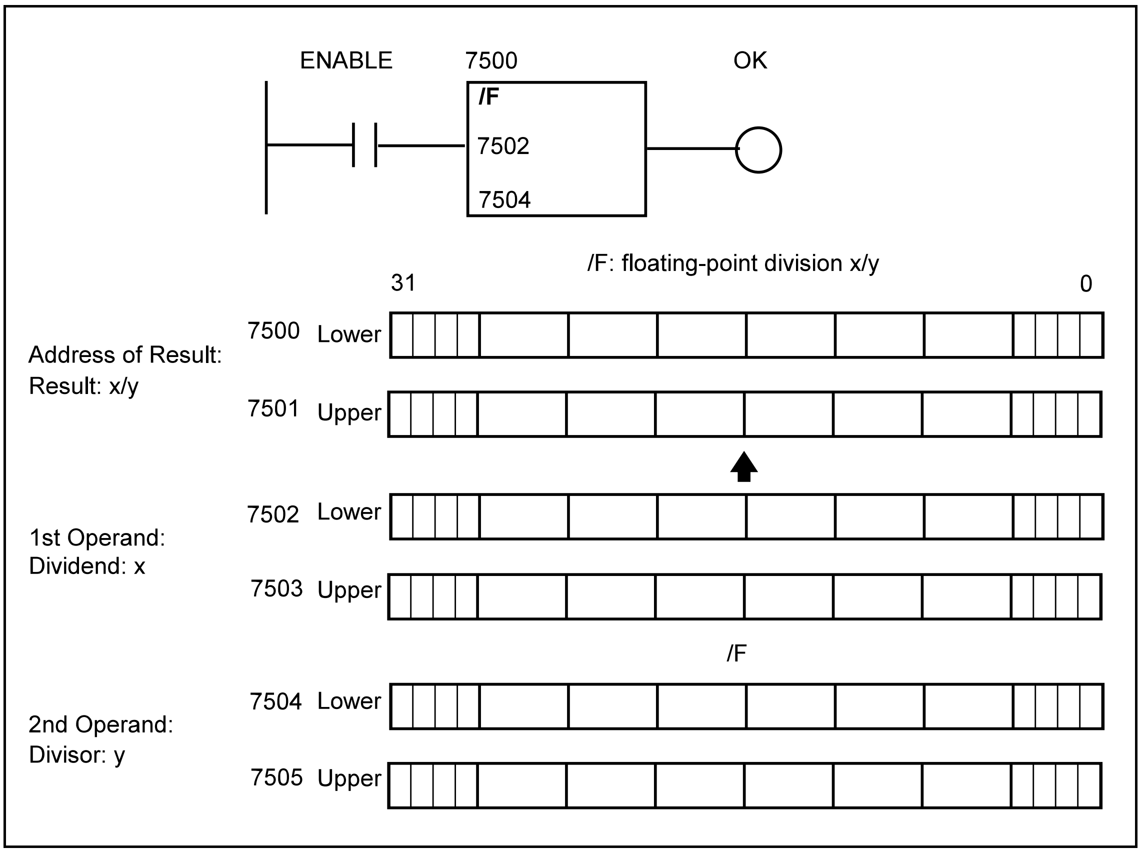
The value of the dividend.

The address of a register or a constant. Indexed addressing is possible.

– **2nd Operand**

The value of the divisor.

The address of a register or a constant. Indexed addressing is possible.



6.10.5 Exponential Power: PWR

The result is saved in register defined at address of result.

– **Base**

The value of the base.

The address of a register or a constant. Indexed addressing is possible.

– **Exponent**

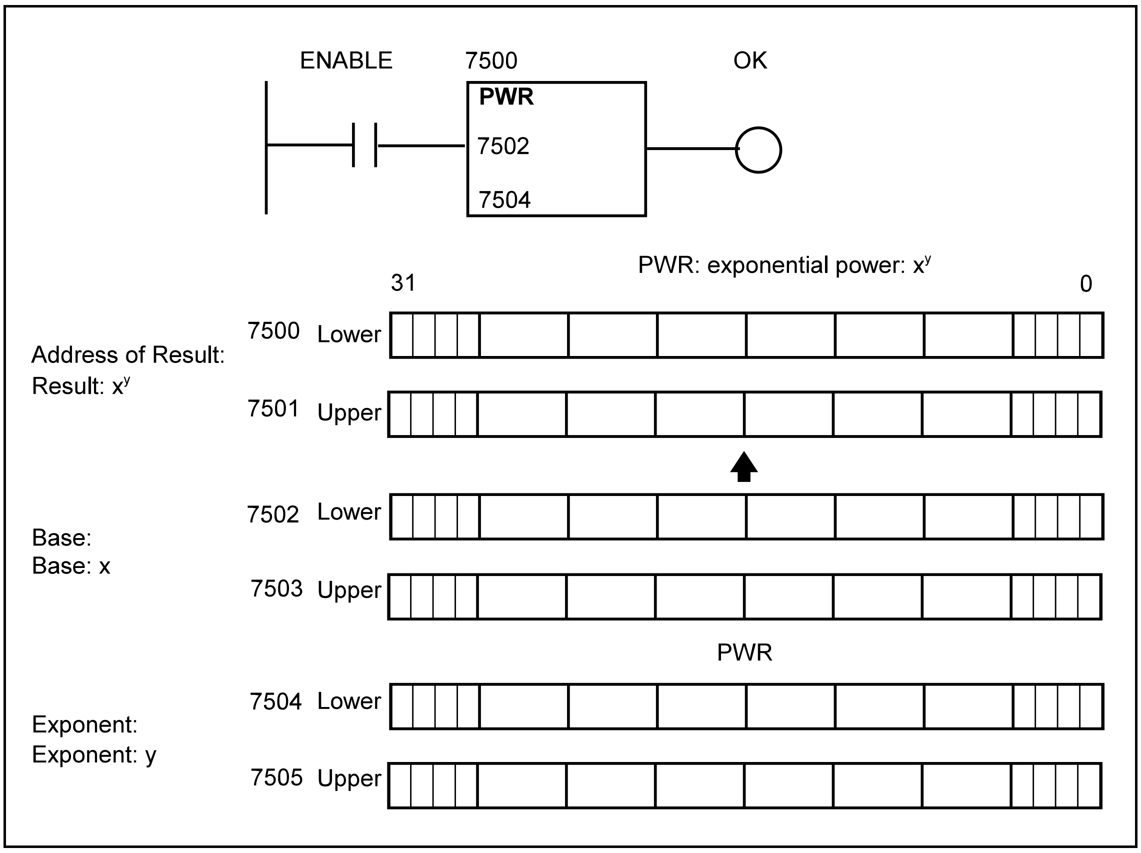
The value of the exponent.

The address of a register or a constant. Indexed addressing is possible.

☞ ***Attention:*** The following inequality must be met for the value of base:

Base ≥ 0

If the upper inequality is not met, the instruction cannot be executed and the error flag is set FL_ER=1.



6.10.6 Square Root: SQRT

The square root is saved in register defined at Address of Result.

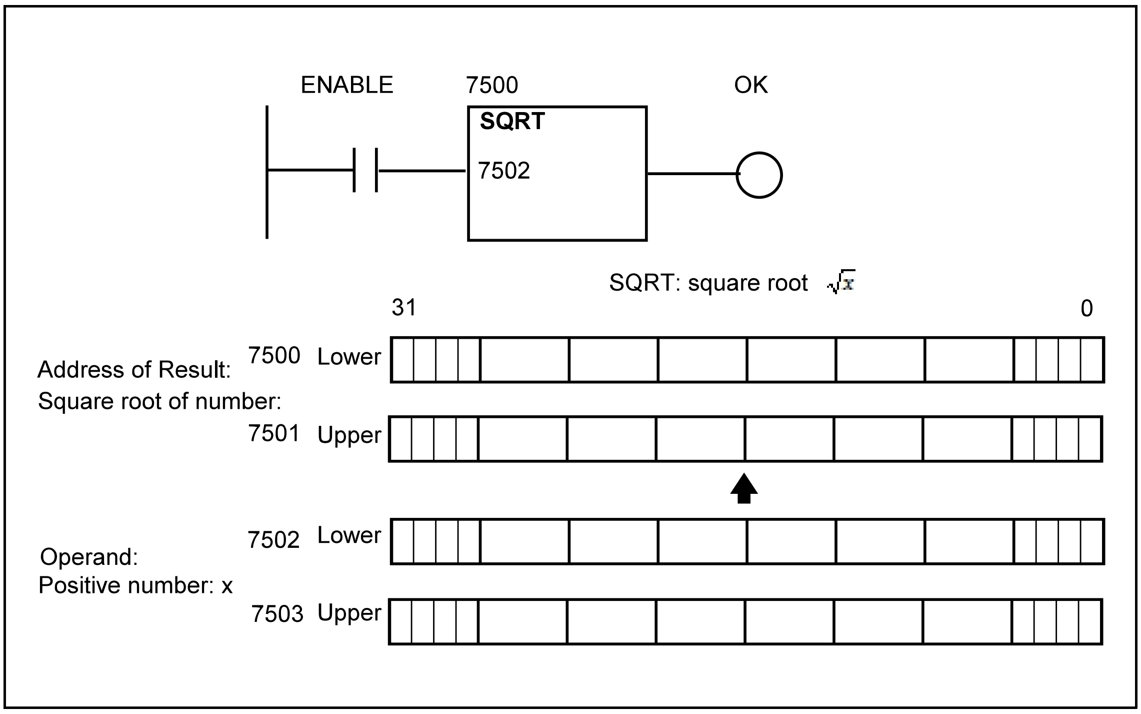
– **Operand**

The address of a register or the constant of which the square root is to be calculated. Indexed addressing is possible.

☞ ***Attention:*** The following inequality must be true for the value of Operand:

Operand ≥ 0

If the upper inequality is not met, the instruction cannot be executed and the error flag is set FL_ER=1.



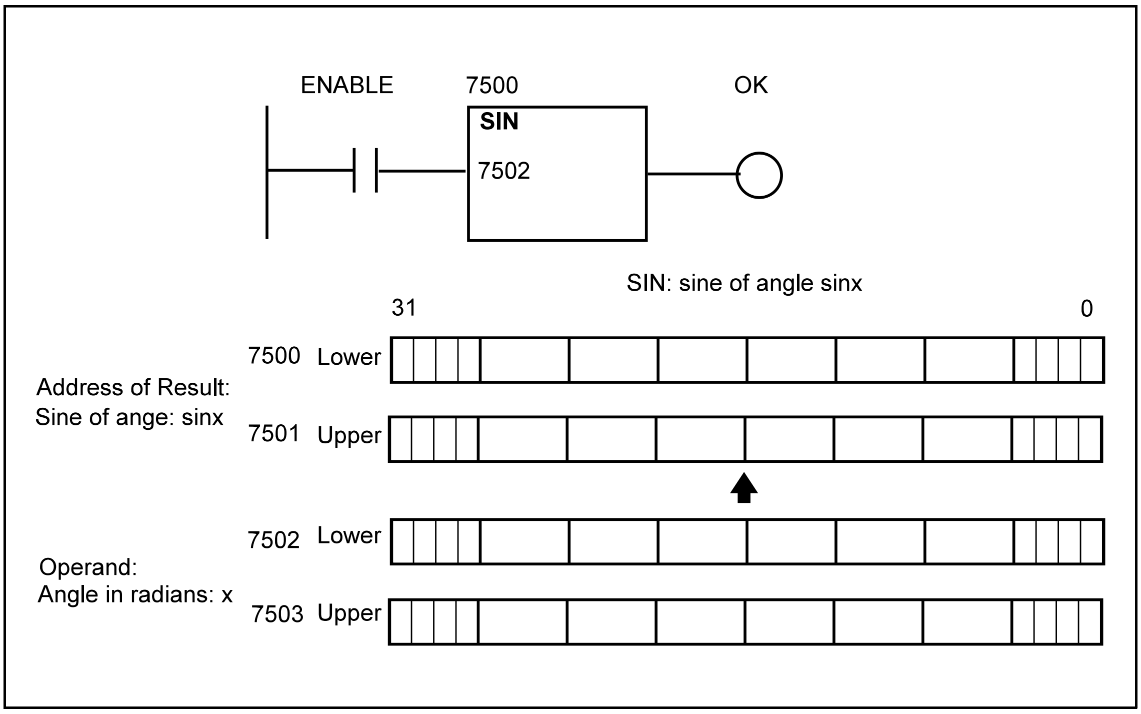
6.10.7 Sine: SIN

The sine of the angle is saved in register defined at Address of Result.

– **Operand**

The value of the angle is to be specified in ***radians***.

The address of a register or the constant of which the sine is to be calculated. Indexed addressing is possible.



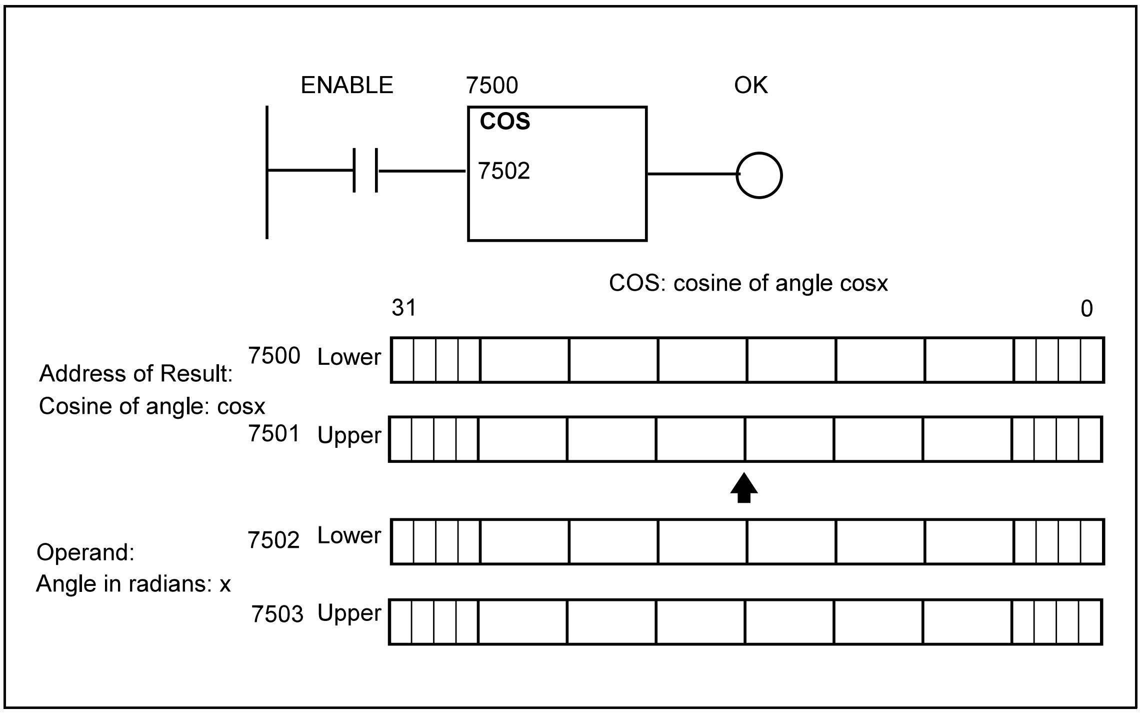
6.10.8 Cosine: COS

The cosine of the angle is saved in register defined at Address of Result.

– **Operand**

The value of the angle is to be specified in ***radians***.

The address of a register or the constant of which the cosine is to be calculated. Indexed addressing is possible.



6.10.9 Tangent: TAN

The tangent of the angle saved in register defined at Address of Result.

– **Operand**

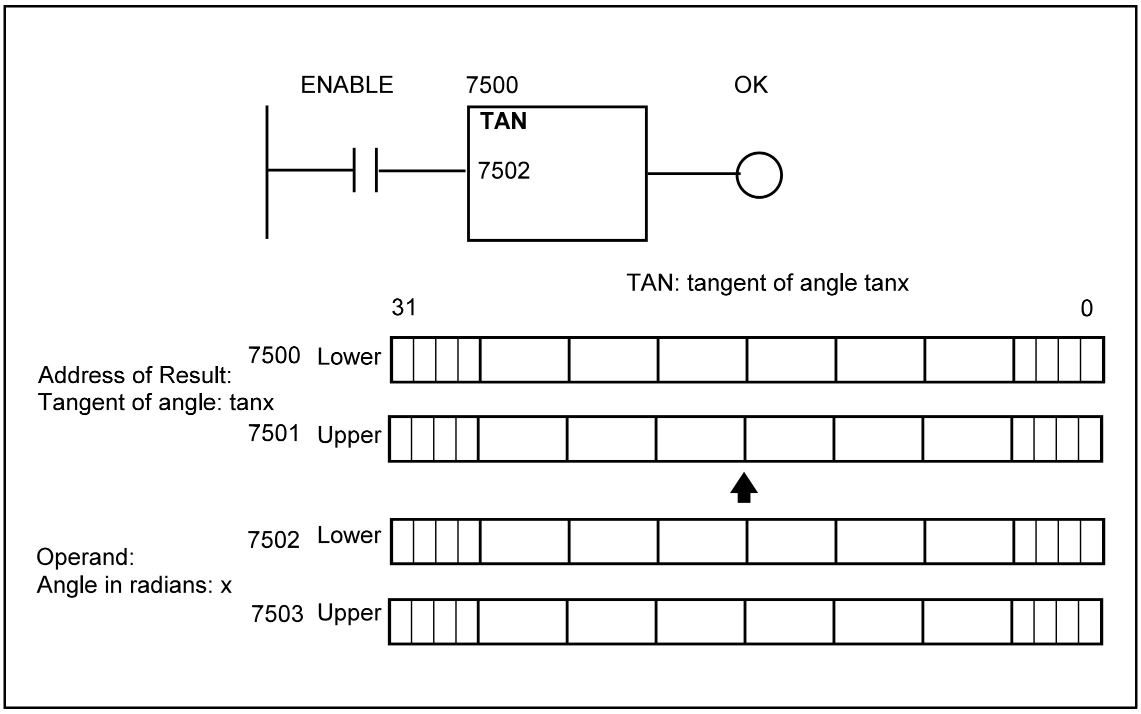
The value of the angle is to be specified in ***radians***.

The address of a register or the constant of which the tangent is to be calculated. Indexed addressing is possible.

☞ ***Attention:*** If the value of Operand:

Operand = π/2

then the instruction cannot be executed and the error flag is set FL_ER=1.



6.10.10 Arc Sine: ASIN

The arc sine of the number is saved in register defined at address of result in ***radians***.

– **Operand**

The address of a register or the constant of which the arc sine is to be calculated. Indexed addressing is possible.

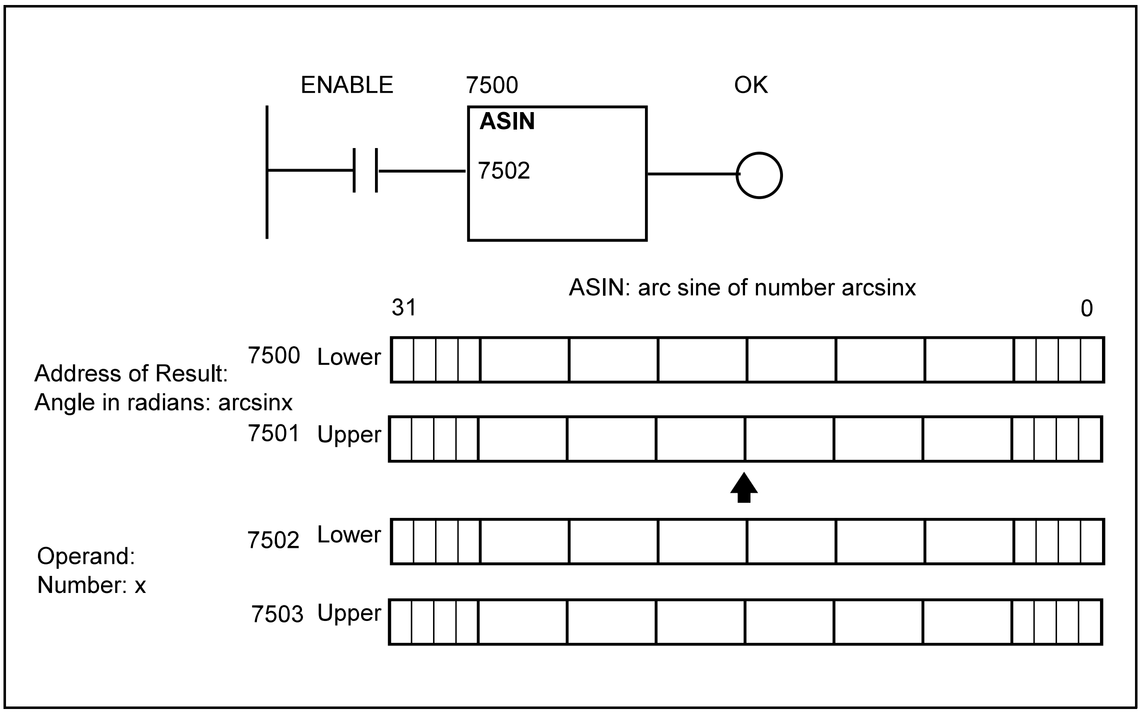
☞ ***Attention:*** The following inequalities must be true for the value of Operand:

$-1 \leq \text{Operand} \leq 1$

If the upper inequalities are not met, the instruction cannot be executed and the error flag is set FL_ER=1.

The result is obtained in the following angle range:

$-\pi/2 \leq \text{Result} \leq \pi/2$



6.10.11 Arc Cosine: ACOS

The arc cosine of the number saved in register defined at Address of Result in ***radians***.

– **Operand**

The address of a register or the constant of which the arc cosine is to be calculated. Indexed addressing is possible.

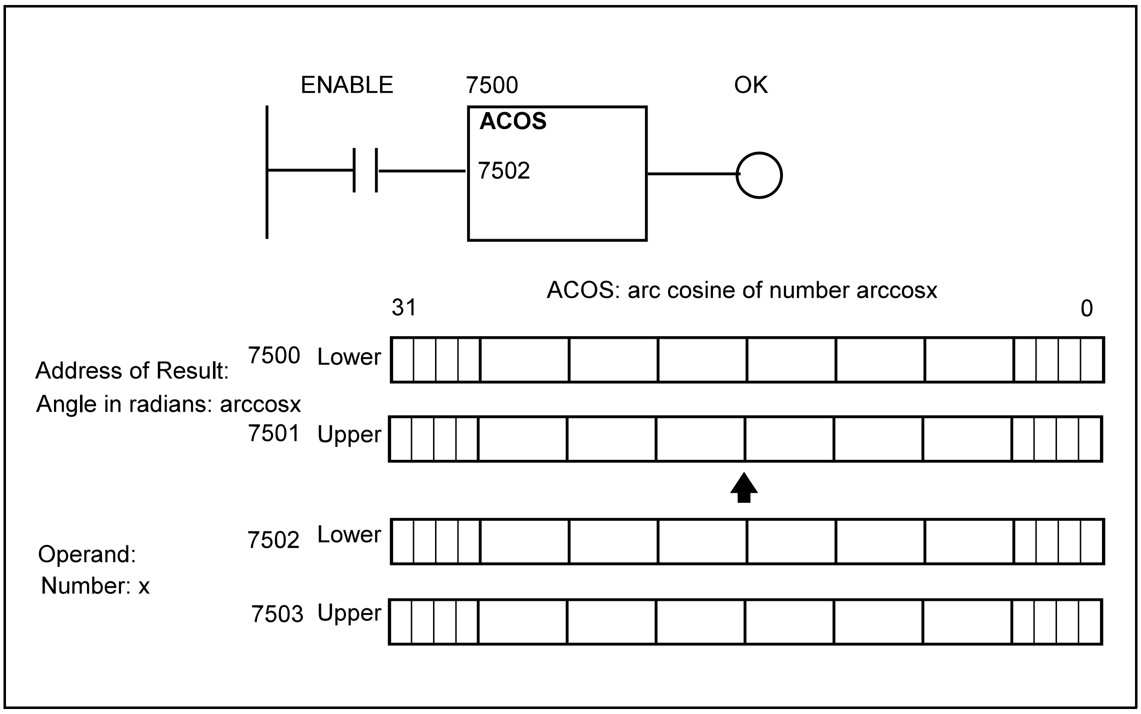
☞ ***Attention:*** The following inequalities must be true for the value of Operand:

$-1 \leq \text{Operand} \leq 1$

If the upper inequalities are not met, the instruction cannot be executed and the error flag is set FL_ER=1.

The result obtained in the following angle range:

$0 \leq \text{Result} \leq \pi$



6.10.12 Arc Tangent: ATAN

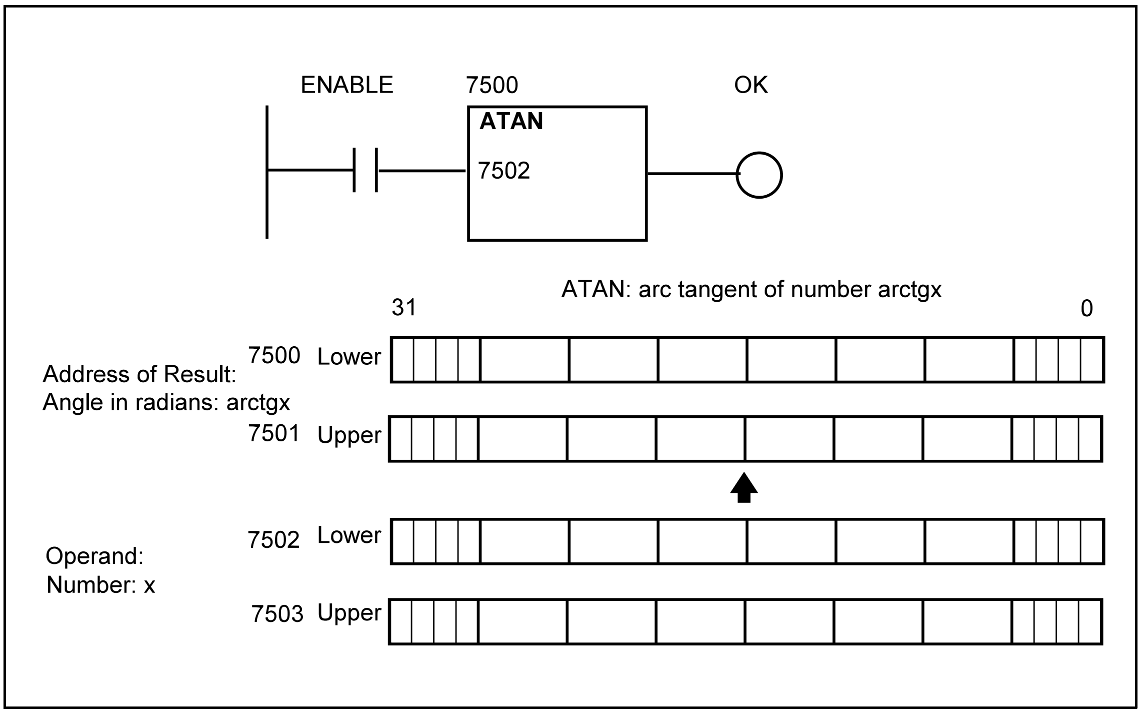
The arc tangent of the number saved in register defined at Address of Result in ***radians***.

– **Operand**

The address of a register or the constant of which the arc tangent is to be calculated. Indexed addressing is possible.

The result is obtained in the following angle range:

$-\pi/2 \leq \text{Result} \leq \pi/2$



6.10.13 Natural Exponent: EXP

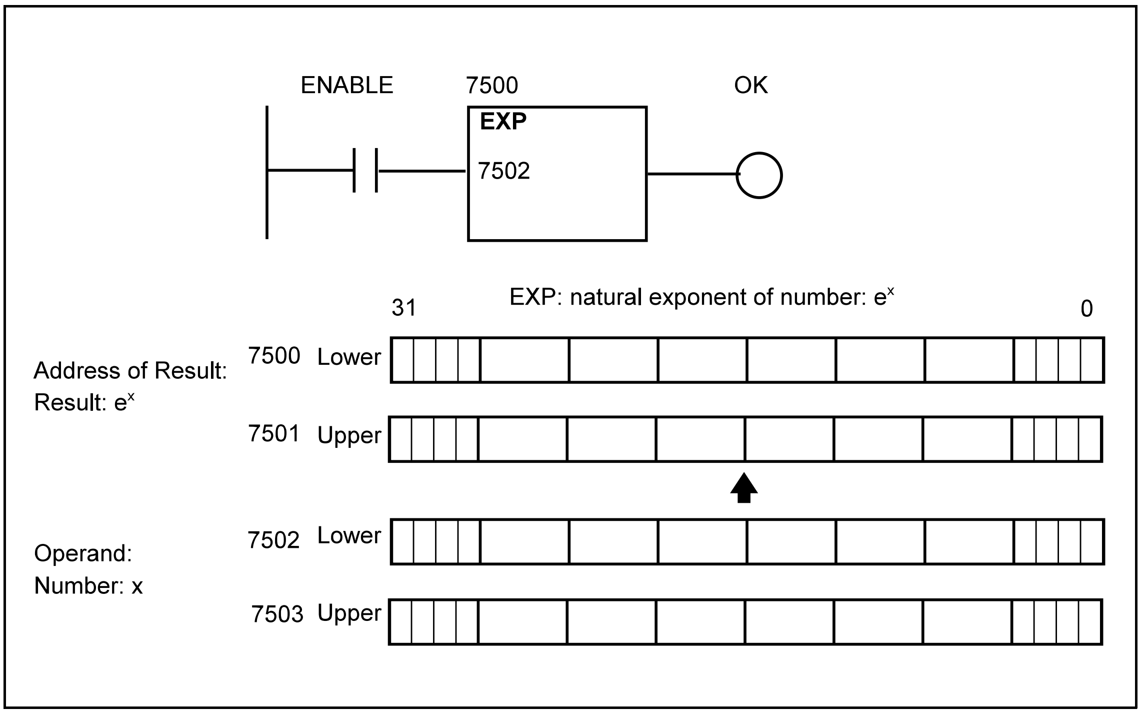
The natural base (e=2.718282...) exponential of the number is saved in register defined at Address of Result.

– **Operand**

The address of a register or the constant of which the natural base exponential is to be calculated. Indexed addressing is possible.

The result is obtained in the following range:

0 < Result



6.10.14 Natural Logarithm: LOG

The natural base (e=2.718282...) logarithm of the number is saved in register defined at Address of Result.

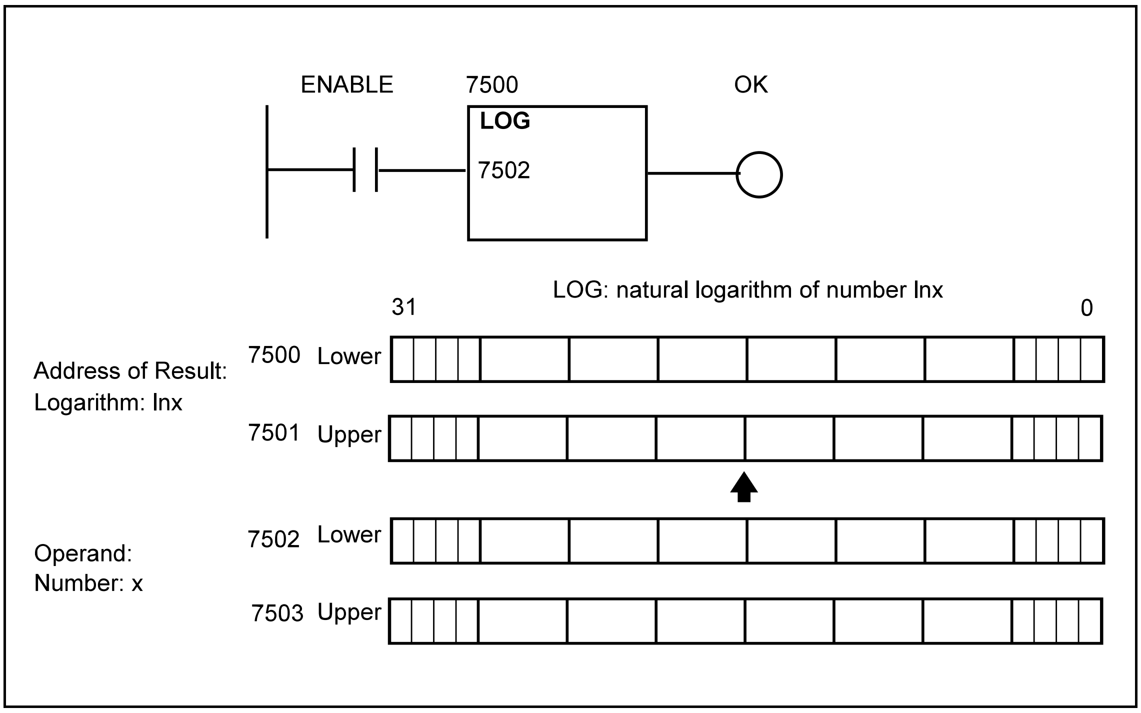
– **Operand**

The address of a register or the constant of which the natural base logarithm is to be calculated. Indexed addressing is possible.

☞ ***Attention:*** The following inequality must be true for the value of Operand:

0 < Operand

If the upper inequality is not met, the instruction cannot be executed and the error flag is set FL_ER=1.



6.11 Conversion Instructions

Data can be converted to another format by conversion instructions.

Input and output of conversion instructions

Each conversion instruction has an

enable input.

Data are converted when the enable input of the instruction is TRUE.

You can *configure* an

output for conversion instructions.

The output will be TRUE if the input of instruction is TRUE and the conversion can be executed. The output will be FALSE if the input is FALSE or the instruction cannot be executed, so the instruction sets the FL_ER error flag!

You can connect further instruction to its output.

All conversion instructions have common input parameters.

Input parameters of conversion instructions

– **Address of Result**:

Range of the value: 0...9999

The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

The converted data are saved in this register.

– **Operand**:

The address of a register or the constant to be converted.

The address of register can be a numeral or a symbol. Indexed addressing is possible.

– **Output**:

You can connect further instructions to the output in enabled state.

– **Remark**:

If output disabled, then the remark written here displayed as a comment, or if you did not write here any remark, the comment of the symbol defined at Address of Result box is written here.

6.11.1 BCD to Binary Conversion: BIN

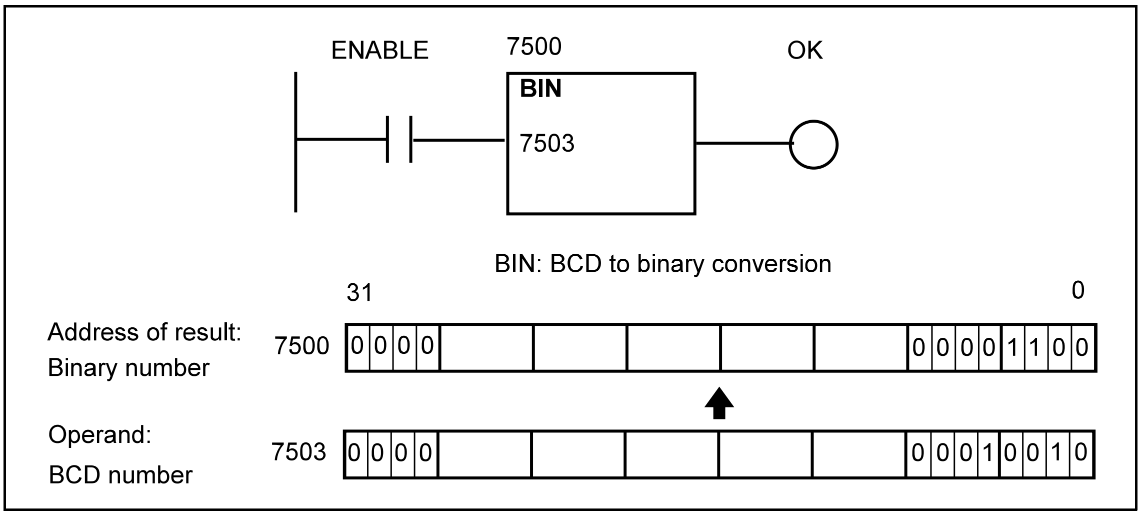
The operand of instruction has to be a BCD number.
The largest convertible BCD number is #$99999999, that is the largest number which can be represented in 8 decimal digits.

☞ ***Attention:***

If the number to be converted is not in BCD format, then the instruction cannot be executed and error flag is set FL_ER=1. For example the

#$000002AF

number is not BCD!



6.11.2 Binary to BCD Conversion: BCD

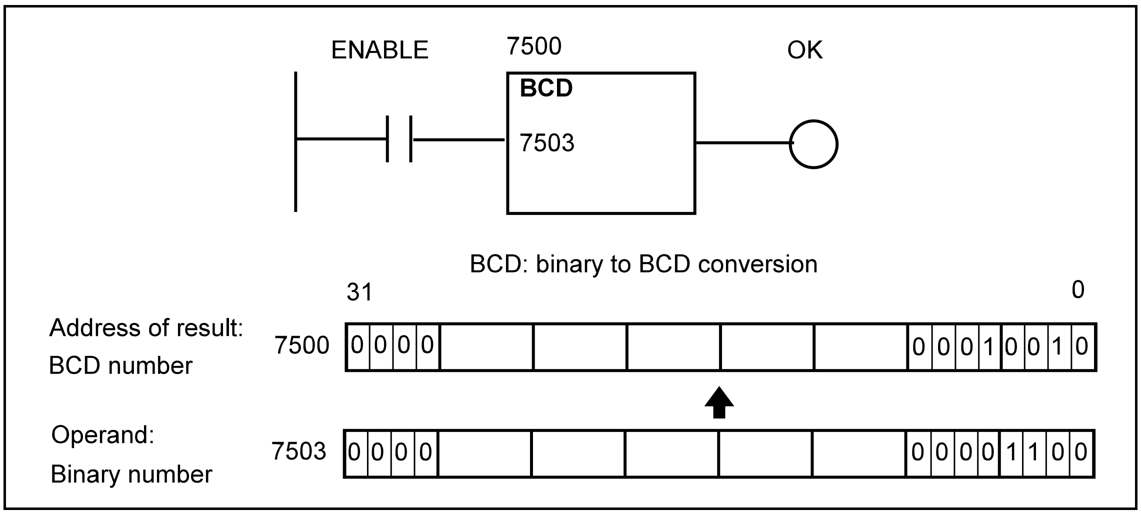
The largest binary number which can be converted to BCD: #$05F5E0FF.
This number corresponds to #$99999999 number of 8 decimal digits.

☞ ***Attention:***

If the binary number cannot be converted to 8 digit BCD number, then the instruction sets error flag FL_ER. For example the number

#$2FFFFFFF

cannot be converted!



6.11.3 Signed Integer to Floating-point Conversion: FLT

☞ ***Attention!***

The result is saved to

Address of Result and to

Address of Result+1

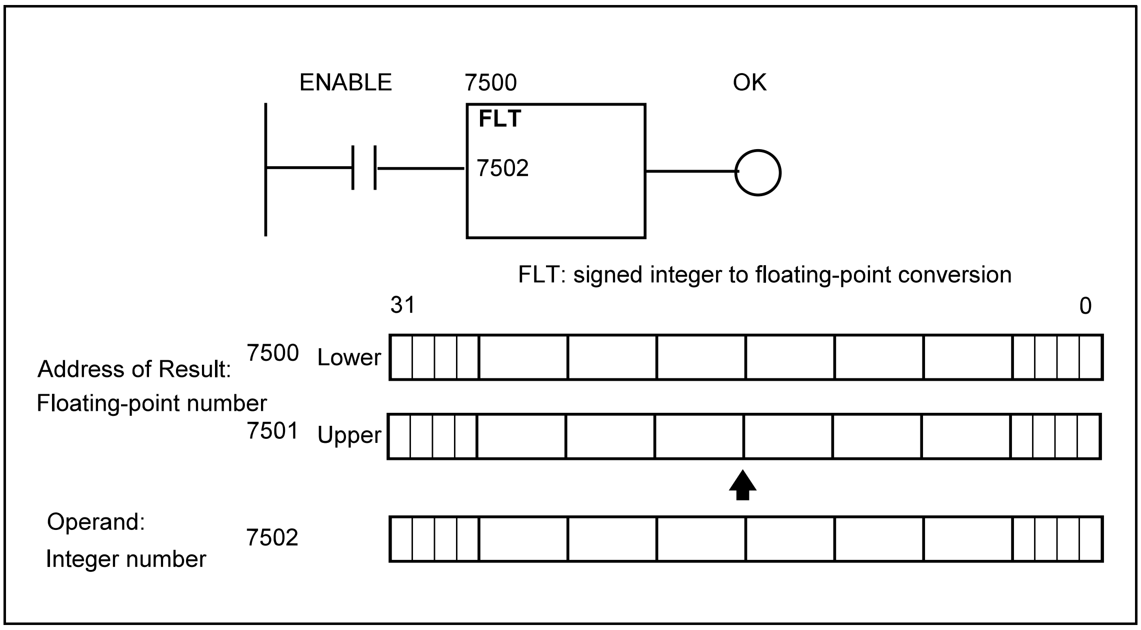
addresses!

Data defined in input parameter of Operand managed as signed, integer number, so the result has got the correct sign. For example, the result of conversion of the integer number

#$FFFFFFFF

to floating-point number will be:

-1.0



6.11.4 Floating-point to Signed Integer Conversion: FIX

☞ ***Attention!***

The number to be convert is loaded from

Operand

Operand+1

addresses!

The range of floating numbers can be converted:

-2147483648.000... ≤ Operand ≤ 2147483647.000...

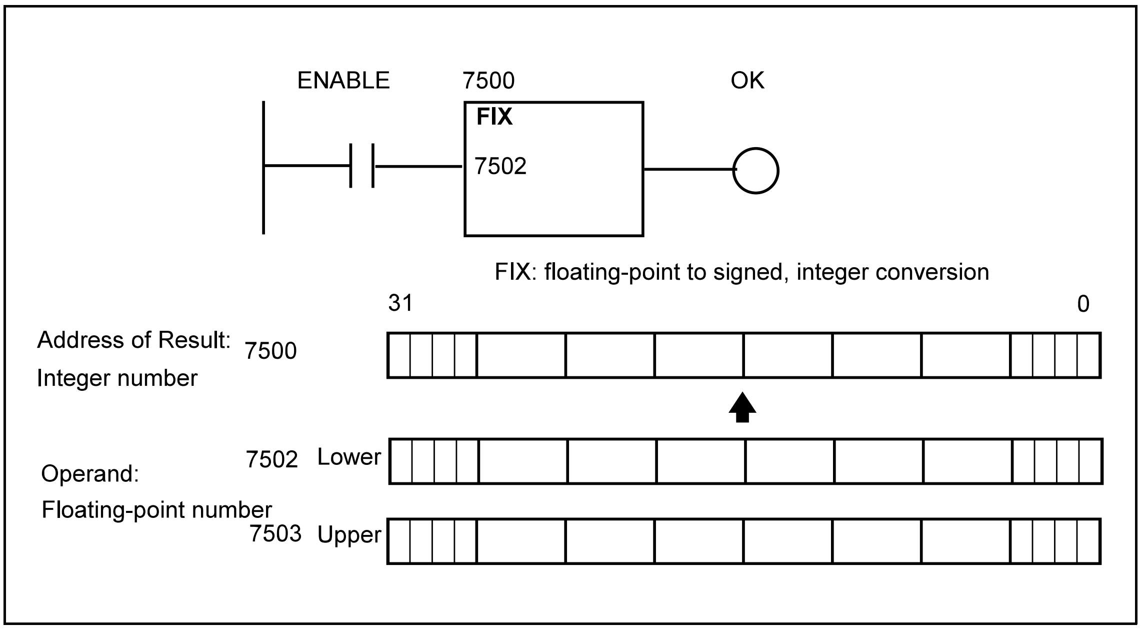
If the number to be converted is out of the range given, then the instruction cannot be executed and sets error flag FL_ER.

The fixed-point number at Address of Result has got the correct sign. For example the result of the conversion of the floating-point number

-1.000

to integer will be:

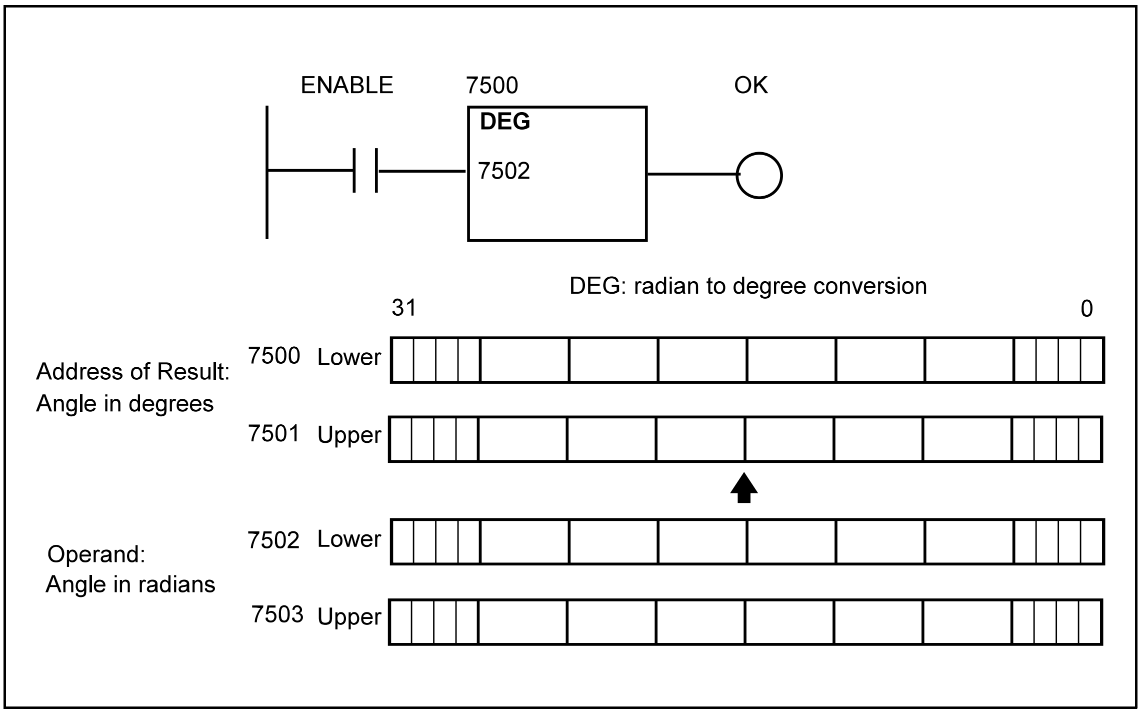
#$FFFFFFFF.



6.11.5 Radian to Degree Conversion: DEG

☞ ***Attention!***

The angle in radians is loaded from the addresses
Operand
Operand+1
as a floating-point number!
The result is saved in degrees at addresses
Address of Result
Address of Result+1
as a floating-point number!



6.11.6 Degree to Radian Conversion: RAD

☞ ***Attention!***

The angle in degrees is loaded from the addresses

Operand

Operand+1

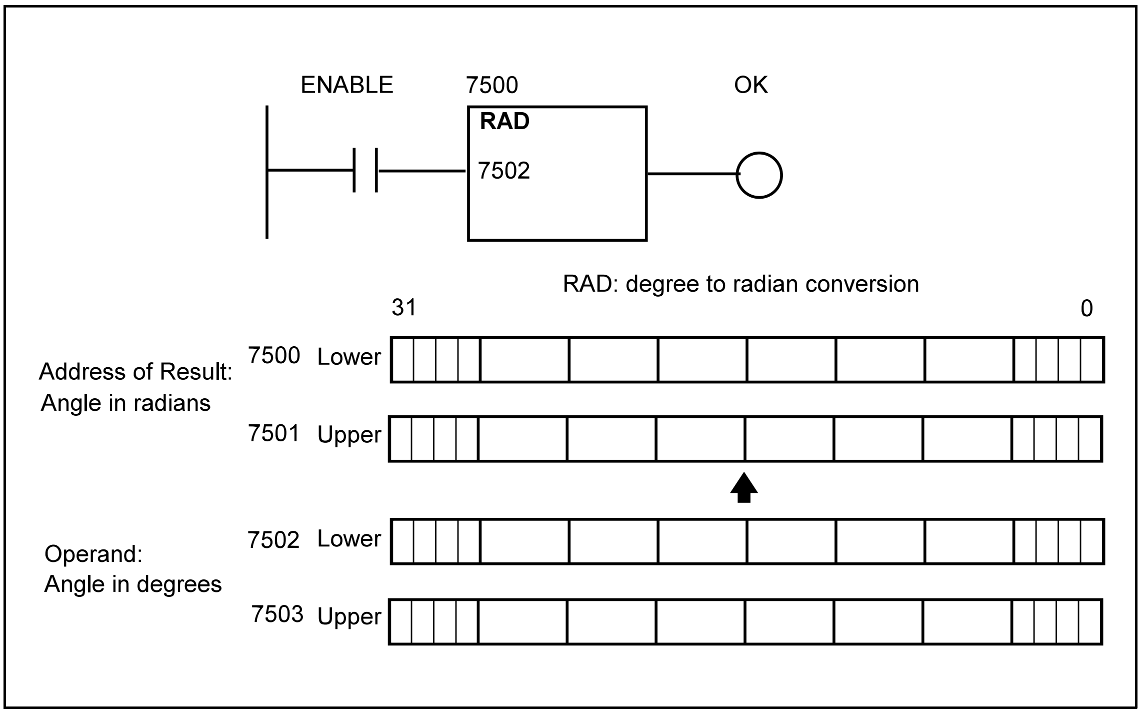
as a floating-point number!

The result in radians is saved at addresses

Address of Result

Address of Result+1

as a floating-point number!



6.12 Comparison Instructions

Two signed, 32 bit integer values or two, 64 bit floating-point values can be compared by using comparison instructions.

6.12.1 The CMP and the FCMP Instructions

The

CMP instruction compares two signed, 32 bit **integer** values, the
FCMP instruction compares two, 64 bit **floating-point** values.

Each instruction has an

enable input.

None of them has **output**. The instructions must be specified at the end of the rung.
The result of comparison can be determined on the flags

FL_GT greater, than (>), the
FL_EQ equal (=), and the
FL_LT less, than (<).

Input parameters of CMP and FCMP instructions

– **Address of Value**:

Range of the value: 0...9999
The address of the left side value to be compared.
The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

– **Data**:

The right side value to be compared.
Data can be a register reference or a constant. Indexed addressing is possible.
In case of CMP instruction, it is a signed, integer number,
in case of FCMP instruction, it is a floating-point number.

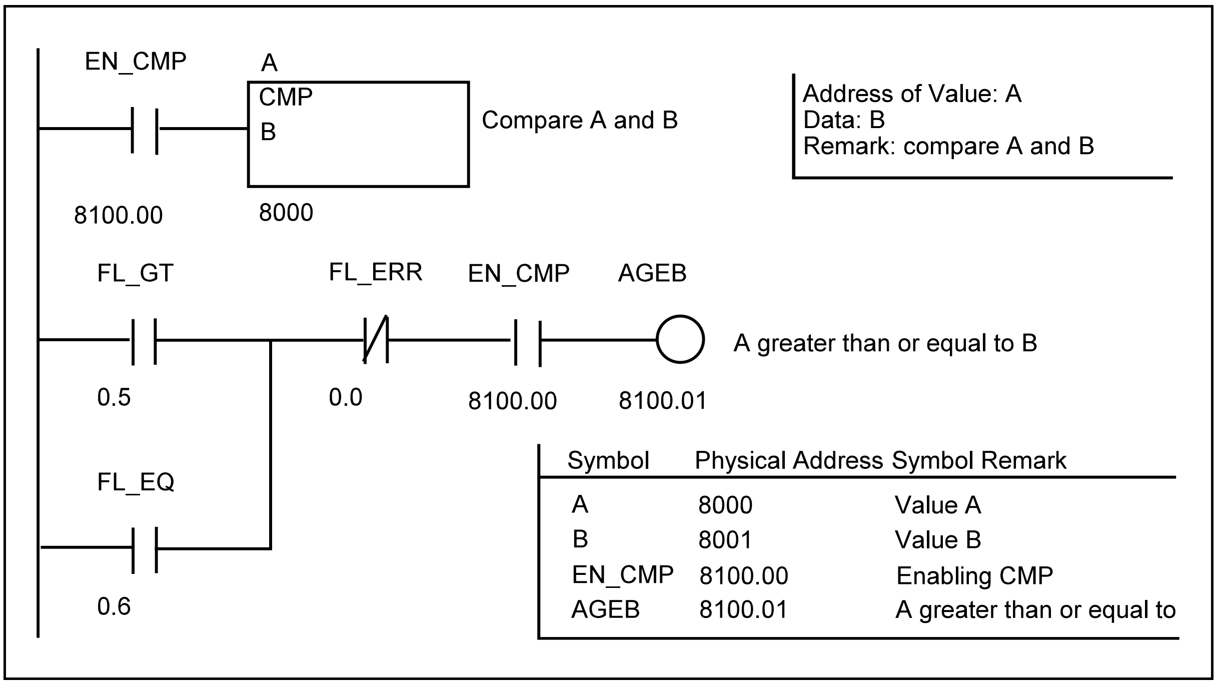
☞ ***Attention:***

Floating-point data are loaded from the Address of Value and from the Address of Value+1 in case of FCMP instruction. It is also valid for the Data parameter, if you have specified a register reference there.

☞ ***Attention!***

If the defined addresses are out of their range of value, then the instruction sets error flag FL_ER .

The following figure shows how to evaluate the condition A≥B using CMP instruction.



Symbol	Physical Address	Symbol Remark
A	8000	Value A
B	8001	Value B
EN_CMP	8100.00	Enabling CMP
AGEB	8100.01	A greater than or equal to

6.12.2 Contact Type Comparison Instructions

The instructions can be used similar to relay contacts. *If the condition is **TRUE** for the data specified, the **contact is closed, else** it is **open**.* Any of the instructions can be the first element of the rung.

Instructions used for comparing two, 32 bit, **signed**, **integer** numbers are:

- CLT (<) less than
- CLE (<=) less than, or equal to
- CEQ (=) equal to
- CNE (<>) unequal
- CGE (>=) greater than, or equal to
- CGT (>) greater than

Instructions used for comparing two, 64 bit **floating-point** numbers are:

- FLLT (<) less than
- FLLE (<=) less than, or equal to
- FLEQ (=) equal to
- FLNE (<>) unequal
- FLGE (>=) greater than, or equal to
- FLGT (>) greater than

Each instruction has an

- **enable input**, and an
- **output**.

The output is TRUE, if the input is TRUE and the condition is met.

The output will be FALSE if the input is FALSE or the instruction cannot be executed, so the instruction sets the error flag FL_ER!

Input parameters of instructions

– **Address of Value**:

Range of the value: 0...9999
The address of the left side value to be compared.
The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

– **Data**:

The right side value to be compared.
Data can be a register reference or a constant. Indexed addressing is possible.
In case of CLT, CLE, CEQ, CNE, CGE, CGT instructions, it is a signed, integer value.
In case of FLLT, FLLE, FLEQ, FLNE, FLGE, FLGT instructions, it is a floating-point number.

☞ ***Attention:***

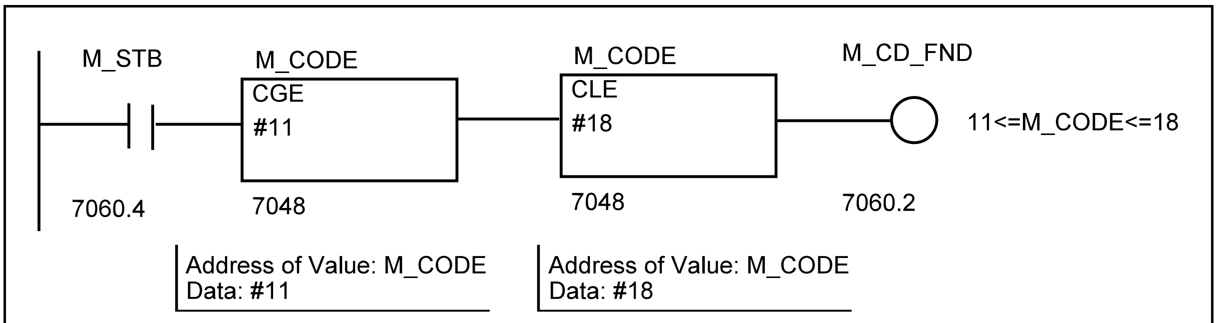
Floating-point data is loaded from the Address of Value and the Address of Value+1 in case of FLLT, FLLE, FLEQ, FLNE, FLGE, FLGT instructions. It is also valid for the Data parameter, if you have specified a register reference there.

☞ ***Attention!***

If the defined addresses are out of their range of the value, then the instruction sets error flag FL_ER.

The following example shows that the coil M_CD_FND is activated if the following conditions are met for the variable M_CODE:

11≤M_CODE≤18



Symbol	Address	Remark
M_CODE	7048	Value of M code
M_STB	7060.04	M_STB received
M_CD_FND	7060.02	Executable M code found

6.13 Messages Sent from the PLC Program

Messages can be sent from the PLC program to the **screen** and to the **log file**. There are instructions, which send messages only to the screen, there are instructions, which send messages only to the log file, and there are instructions, which send messages to the screen and also to the log file.
The messages sent to the screen are stored in a message buffer and they remain there until they have been deleted by an intervention. This intervention can be using the cancel or start buttons or ceasing the cause of the message.
The NC stores the most important events and errors in a log file. The messages and events sent to the log file are stored for about one month. Messages can also be sent here by PLC program.
The PLC **message**s have **code**s and **texts**. In all 999 messages can be sent from the PLC program, which are identified by a code and inside the code the **number of the message**. PLC messages are part of common messages sent by the NC.

Order of appearing messages on the screen
Always the message sent last appears on the screen first: Last in, first out. If all messages are displayed, the last message will be on the top, the first one on the bottom in the queue.

Message codes
All common messages are identified by an 8-digit decimal number. This number is called as message code.
The structure of **message codes** is:

AABCCCDD

where:

AA: **Channel index**. In case of messages independent of a channel its value is 0, while in case of messages dependent upon a channel its value is equal to the number of the channel.
In a **PLC message** the PLC programmer has to decide wether the message is channel-independent or have to assign it to a channel. For example “Emergency state” is a channel-independent message, so its index is AA=0, but “Please start the spindle” message arrives from a dedicated channel, so its index is AA>0. This is important, because the same message number can be sent from several channels, but the message code has to uniquely identify the channel from which the message was sent.

B: **Module index**: index of the system module (measuring system, interpolator, block interpreter, etc.) sends the message.
In case of **PLC messages** its value is B=0.

CCC: **Message number**. The message numbers are ranging from 1 to 999.

DD: **Optional index**: There can be axis-, spindle- etc. indexes in NC messages.
In case of **PLC messages** the PLC programmer decides it. For example there can be spindle-, or turret indexes, so it codes that the message which spindle or turret refers to. This is important, because the same message number can refer to several spindles or turrets, but the message code has to uniquely identify which one it refers to.

Classifying of instructions sending messages

Instructions sending messages can be classified as follows:

– Displayed on the screen or not:

Instructions **displayed** on the screen are: **MSG, MSGF, ALR, ALRF**

Instructions **not displayed** on the screen are: **REM, REMF**

– Registered to the log file or not:

Instructions **registered** to the log file are: **REM, REMF, ALR, ALRF**

Instructions **not registered** to the log file are: **MSG, MSGF**

Writing PLC programs, take care which instruction to be used.
For example a "Door open" message has to be displayed to inform the operator why the machining stopped. Because this message can appear several times in a day, but it only informs the operator and it does not indicate error or problem, it is unnecessary to register it in log file. So in this case the MSG instruction can be used.
If a message is sent to inform the operator on an error, for example a motor starter is off, the ALR instruction can be used, because this message is displayed and registered to the log file.
REM instruction can be used, if the message is not to be displayed because the operator's intervention unnecessary, but is should be registered.
A **numerical value** also can be sent with a **PLC message**. The number can be an integer or a floating-point one. Messages can be classified by if they are containing integers or floating-point data:

– Messages containing **integers**: **MSG**, **ALR**, **REM**,

– Messages containing **floating-point** data: **MSGF**, **ALRF**, **REMF**.

Text of PLC messages

The PLC editor program eases editing message texts. They are saved in the file *.plc together with the PLC program.
Message texts can be saved separately as *.mes files, using Export button of the PLC editor. The *.mes message file also can be imported into an existing PLC program.
The encoding of *.mes text file is UTF8. It means that every currently known character can be represented here.

Printing the content of a PLC variable in the message

The content of a register can be printed anywhere in the message text.
The PLC instruction specifies the number to be printed is an integer or floating-point one.
The number of digits after the decimal point to be printed can be specified in the editor in case of floating-point numbers.
For example the following text is to print: " Mount the tool No. n to the spindle ", where "n" is the value of a PLC register.
The following format will be in the exported message file:

Mount the tool No. {#0:F0} to spindle.

The string {#0:F0} means to convert the integer defined in instruction MSG, ALR, or REM to decimal number and print it in the text. The symbol # means it is an integer number and F0 means no decimal point is used when printing.
The following format will be in the exported message file if text contains a floating-point number:

The temperature of the spindle {*0:F1} °C

The {*0:F1}string means to convert floating-point number defined in MSGF, ALRF, or REMF instruction to decimal number and print it in the text with a precision of one decimal digit. Symbol * means it is a the floating-point number, F1 means that the precision is one decimal

digit.
Rules of formatting:

{	(start of formatting)		
#	(fixed-point data), or	*	(floating-point data)
0	(required)		
:	(separator)		
F			
i	(required number of decimal digits in case of floating-point representation, in case of integers it is 0)		
}	(end of formatting)		

6.13.1 Instructions Sending Messages: MSG, MSGF, ALR, ALRF, REM, REMF

PLC messages sent by ***MSG, MSGF, ALR, ALRF*** instructions are **displayed on the screen**, in the message queue. The PLC programmer is to decide what intervention of the operator deletes the message from the queue.
The messages of ***MSG, MSGF*** instructions **are not registered in the log file**.
The messages of ***ALR, ALRF*** instructions **are always registered the in log file**.
PLC messages sent by ***REM, REMF*** instructions **are not displayed**, but they **are always registered in the log file**.
The optional input parameters of ***MSG, ALR, REM*** instructions are **integers**, while the optional input parameters of ***MSGF, ALRF, REMF*** instructions are **floating-point** numbers.

Inputs and outputs of instructions

Each instruction has an
enable input, and an
output.
Each message instruction does the following operations if its **input** is enabled:
– For the **rising edge** at the input it puts the message into the message buffer and registers it in into the log file,
– For the **falling edge** at the input it deletes the message from the message buffer of the screen.
These instructions **have output**: if the message appears on the screen or the message is stored in the log file when using REM,REMF instruction.

Input parameters of MSG, MSGF, ALR, ALRF, REM, REMF instructions:

– **Message Number:**
Range of the value: 1,...999
The serial number of the message
It is an integer constant or register address. Indexed addressing is possible.

– **Channel Index:**
Range of the value: 0...99
The channel index of the message.
It is an integer constant or register address. Indexed addressing is possible.

– **Arbitrary Index:**
Range of the value: 0...99
It is the arbitrary index of the message.
It is an integer constant or register address. Indexed addressing is possible.
It is possible to define for example axis-, spindle- or turret number.

– **Optional Variable:**
In case of MSG, ALR, REM instructions it is an integer constant or register address.
In case of MSGF, ALRF, REMF instructions it is a floating-point constant or register address.
Indexed addressing is possible.

– **Remark:**
Remark

Output and function of MSG, MSGF, ALR, ALRF instructions:

For the rising edge at the input the instruction puts the message into the message buffer to be displayed.
For the falling edge at the input the instruction deletes the message from the message buffer to be displayed.
If the ALR or ALRF instruction detects a rising edge at its input, it registers the text and code of message also in the log file.
If the code and text of the message appear on the screen, the output of instructions will be TRUE.
The code of a message can be:

AA0CCCDD

where:

AA:	Value of the Channel Index parameter,
0:	Indicates that the message is sent by PLC,
CCC:	Value of the Message Number parameter,
DD:	Value of the Arbitrary Index parameter.

Output and function of REM, REMF instructions:

If the REM, REMF instruction detects a rising edge at its input, it registers the text and code of message in the log file.
As soon as the message of the instruction has been registered in the log file, the output of the instructions will be TRUE.

Examples:

Stopping error flag for spindle with index S1:

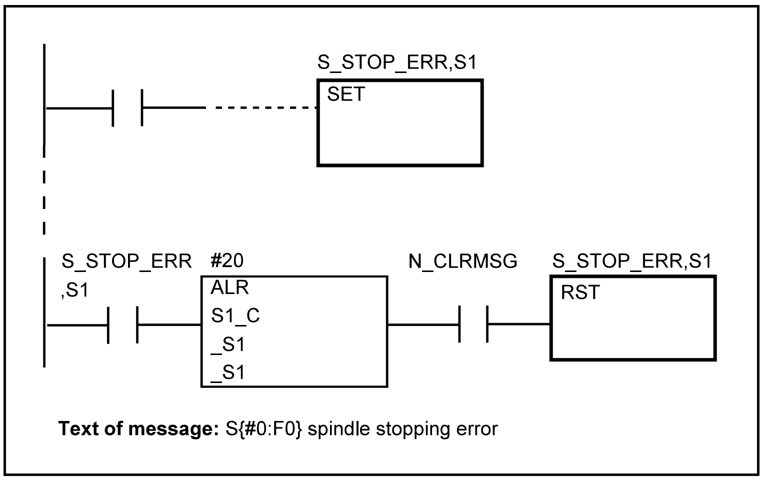
S_STOP_ERR

Number of message is #20

The message is received from the channel S1_C, where the register S1_C stores the actual channel number of spindle S1.

The message applies for spindle number _S1, where _S1 is the number of the spindle S1.

(In this case the index of the spindle S1=0, the number of spindle _S1=1)



Write the number of spindle also in the text of message: the value of optional variable: _S1.
The message is fully encoded by the number of channel and spindle. The number of spindle has also taken to text of message to ensure transparency.
The identifier of text of message is 20. This text of message can used for 2nd, 3rd, etc. spindles, it has not necessary to write the message again, you only have to change the number of spindle.
If TRUE state appears at the input of the message box, then the message is stored in the message buffer and also in the log file.
If the message appears on the screen, the output of the ALR instruction will be TRUE. Then the message can be deleted by N_CLRMSG flag which is controlled by CANCEL button. The N_CLRMSG flag deletes S_STOP_ERR,S1 flag.

Send request for starting in CH1 channel with

ST_MB_REQ

flag.

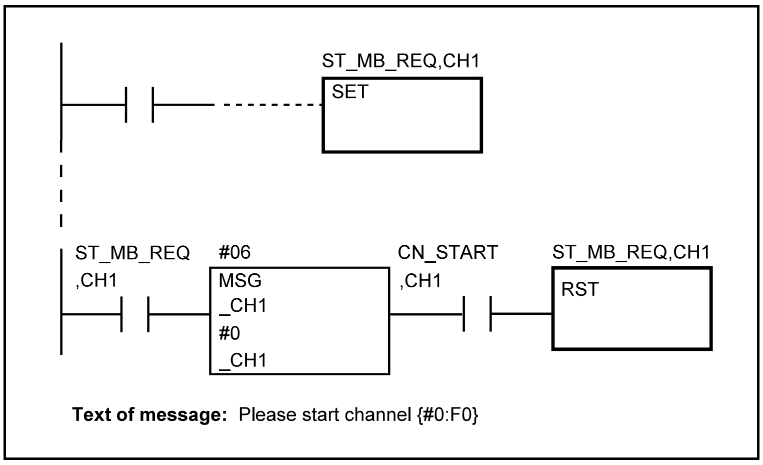
The number of message is #06

The message requested from channel number _CH1.

(In this case, the index of channel CH1=0, the number of channel is _CH1=1.)

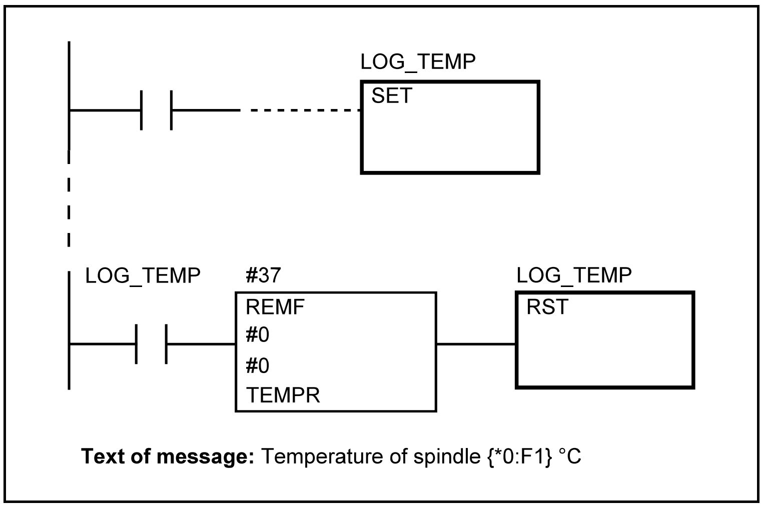
The value of arbitrary index is #0.

The number of channel has also written to text of message: _CH1.



The message is fully encoded by the number of channel. The number of channel has also taken to text of message to ensure transparency.
The identifier of text of the message is 06. The text of the message can be used for further channels, it is not necessary to write the message again, you only have to change the number of the channel.
If TRUE state appears at the input of the message box, then the message is stored in the message buffer, but it is not stored in the log file.
If the message appears on the screen, the output of MSG instruction will be TRUE. Then the

message can be deleted by the START button because the flag ST_MB_REQ,CH1 is reset.



Text of message: Temperature of spindle {*0:F1} °C

The PLC stores in the log file the temperature of spindle after a pre determined interval: it sets the LOG_TEMP flag.
The number of message is #37.
The machine has only one channel and spindle, so both channel- and spindle indexes are #0.
The temperature of spindle stored in the register declared as TEMPR in floating-point format.
Floating-point data is displayed with the accuracy of one decimal digit {*0:F1}. It is specified in the text of the message.
If TRUE state appears at the input of the message box, then the message is stored in the log file and the output of the message box will be TRUE simultaneously. It deletes the log request flag.

6.14 Program Control Instructions

6.14.1 End of Module Instruction: END

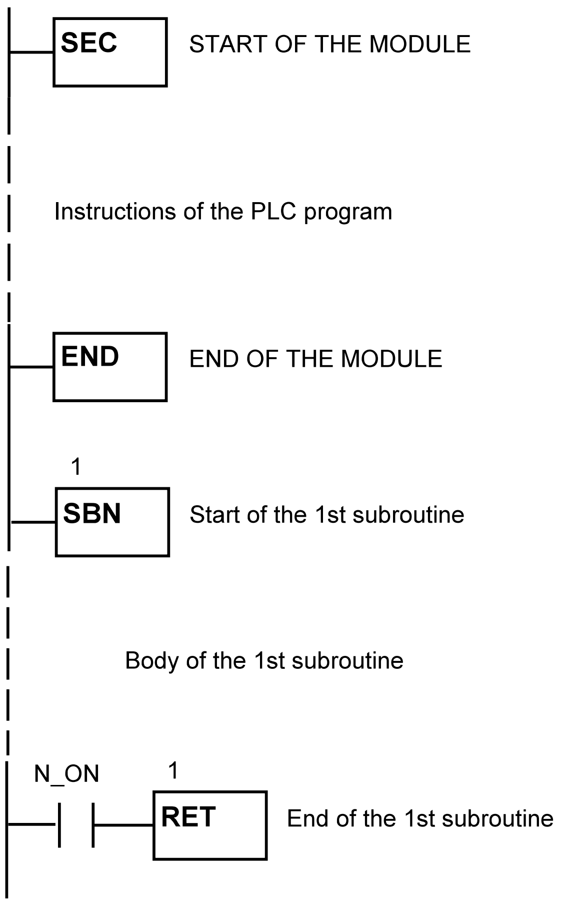
This instruction indicates the end of a cyclical PLC module. It can also be used in the Main program and in the Int0 module.
A PLC program module is executed from the start of the ladder diagram till the END instruction and it is restarted after the cycle time elapsed (in case of Main program T_{PLC}, in case of Int0 TimeSlice).
The END instruction has a separation function. Bodies of subroutines used in a PLC program module have to be specified after the END instruction. If subroutines are not used in the module, the instruction does not have to be used.

☞ ***Attention!*** *The END instruction has to be specified in the 1st column, although it is a closing element.*

Input parameter of END instruction:

– **Remark:**

Remark: text written in Remark will be the comment of logic unit.


SEC
START OF THE MODULE
Instructions of the PLC program
END
END OF THE MODULE
1
SBN
Start of the 1st subroutine
Body of the 1st subroutine
N_ON
1
RET
End of the 1st subroutine

6.14.2 Conditional Branch: the JMP and JME Instructions

A sector of the PLC program module could be jumped over by using JMP and JME instructions. The use of JMP instruction has a condition: if the condition is satisfied at the input, then the jump is executed. The instruction jumps to the number (label) specified in the instruction.
JME instruction indicates the number (label), where JMP instruction transfers program control.

☞***Attention!*** *Numbers used in JMP and JME instructions are ranging from 0 to 99. Each label has to be individual, they must not match the numbers of SBS, SBN, RET instructions of subroutines!*

☞***Attention!*** *By using JMP instruction you can jump only inside a module, you cannot jump into another module!*

Input of JMP instruction

It has got an
enable input.
JMP is executed if the enable input is in TRUE state.

Input parameters of JMP instruction

– **Jump Number**:
Range of the value: 0...99
Number of the target label, where the instruction transfers program control.

– **Remark**:
Remark: text written in Remark will be the comment of the logic unit.

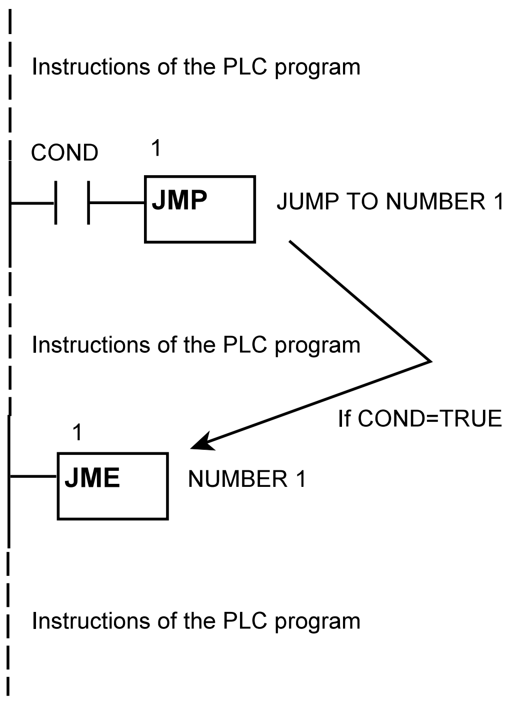
Input of JME instruction

☞ ***Attention!*** *The JME instruction has to be specified in the 1st column, although it is a closing element.*

Input parameters of JME instruction

– **Jump Number**:
Range of the value: 0...99
Number of the target label, where the JMP instruction transfers program control.

– **Remark**:
Remark: Text written in Remark will be the comment of the logic unit.


Instructions of the PLC program
COND
1
JMP
JUMP TO NUMBER 1
Instructions of the PLC program
If COND=TRUE
1
JME
NUMBER 1
Instructions of the PLC program

6.14.3 Subroutine Call: the SBS, SBN and RET instructions

Subroutines can be called from a PLC program module by the SBS instruction. A condition has to be defined for which the subroutine is called. A number (label) has to be specified where the control is passed.
The body of a subroutine starts with SBN instruction with the proper number (label).
RET instruction defines the end of a subroutine. A return condition has to be defined at the input of the instruction.
The bodies of subroutines that are the ladder part between the SBN and RET instructions always have to be written after the END instruction of the module!
Multiple, nested subroutine call is possible. The depth of call only depends on the quantity of available labels.

☞***Attention!*** *Numbers (labels) used in SBS, SBN, RET instructions are ranging from 0 to 99. Each number has to be individual, they must not match the numbers of JMP, JME instructions of conditional branch!*

☞***Attention!*** *Subroutines are belonging to the module where they were specified. They could not be called from another module!*

Input of **SBS** instruction

It has got an

enable input.

Subroutine call is executed if the enable input is in TRUE state.

Input parameters of **SBS** instruction

– **Subroutine Number**:

Range of the value: 0...99

Number (label) of the subroutine to be called.

– **Remark**:

Remark: text written in Remark will be the comment of logic unit.

Input of **SBN** instruction

☞ ***Attention!*** *The SBN instruction has to be specified in the 1st column, although it is a closing element.*

Input parameters of **SBN** instruction

– **Subroutine Number**:

Range of the value: 0...99

Number (label) of the subroutine called by the SBS instruction.

– **Remark**:

Remark: text written in Remark will be the comment of logic unit.

Input of **RET** instruction

It has got an

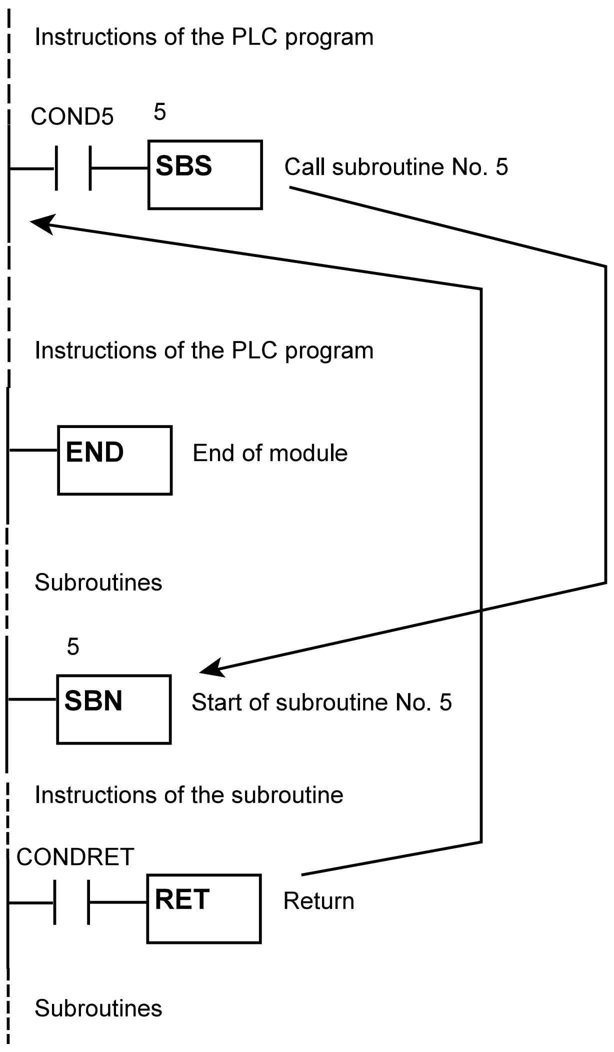
enable input.

If the enable input is in TRUE state, it returns from the subroutine.

Input parameters of **RET** instruction

– **Remark**:

Text written in Remark, will be the comment of the logic unit.


Instructions of the PLC program
COND5
5
SBS
Call subroutine No. 5
Instructions of the PLC program
END
End of module
Subroutines
5
SBN
Start of subroutine No. 5
Instructions of the subroutine
CONDRET
RET
Return
Subroutines

6.15 Axis Control Instruction: MOVCMD

Control of axes is basically the task of the NC.
Sometimes the **PLC** needs to **request** one or more **axis from the NC**, the PLC program moves them with its own instructions then gives them back for the NC. For example, some axes have to be moved to a certain position during tool- or palette-change. The request, giving back, starting, etc. are controlled by axis control flags especially for this purpose.

☞ ***Attention!*** *Such moves have to be done strictly by* ***NC buffer emptying*** *functions in the PLC. These functions can be assigned by parameters. After the end of function NC has to refresh the positions of axes moved by PLC and it continues machining from this new position. This is the reason to stop buffering.*

The other way of use is when an axis is controlled only by the PLC program. For example tool turret or magazine is rotated by a servomotor. In this case the PLC requests the axis from NC in the first cycle and never gives it back.

The axis control instructions are not written directly into the interpolator, but into the **single-block buffer of the axis control instruction**. The interpolator takes the instruction from the buffer after it executed the previous instruction. Interpolator is operated continuously by continuously writing the buffer of the axis control instruction.

Input and output of the axis control instruction

The axis control instruction has an

enable input.

If the enable input is in TRUE state, then it waits until the single block buffer of the axis control instruction is emptied then it writes the instruction into the buffer.
You can *configure* an

output for axis control instruction.

If the instruction has been written into the buffer, then the output will be TRUE. The output will be FALSE if the input is FALSE or the instruction cannot be executed, so the instruction sets the error flag FL_ER!
You can connect further instructions to its output.

Input parameters of axis control instruction

– **Axis Address**:

It is the index of the axis to be controlled. Range of the value: 0...31.
The address of a register or an integer constant. Indexed addressing is possible.

– **Instruction Code**:

The instruction code.
The address of a register or an integer constant. Indexed addressing is possible.
There is a chart in the following page for the interpretation and possible values of the instruction code.

– **Speed**:

The feedrate of the movement.
The address of a register or a floating-point constant. Indexed addressing is possible.
Data is interpreted according to N0104 Unit of Measure parameter bit IND. Its unit: mm/min, degree/min or inch/min. A floating-point number.

– **Distance**:
The distance of travel.
The address of a register or a floating-point constant. Indexed addressing is possible.
Data is interpreted according to N0104 Unit of Measure parameter bit IND. Its unit: mm, degree or inch. A floating-point number.

– **Output**:
You can connect further instructions to its output in enabled state.

– **Remark**:
If output is disabled, then the remark written here displayed as a comment, or if you did not write here any remark, comment of symbol defined in Axis Address box written here.

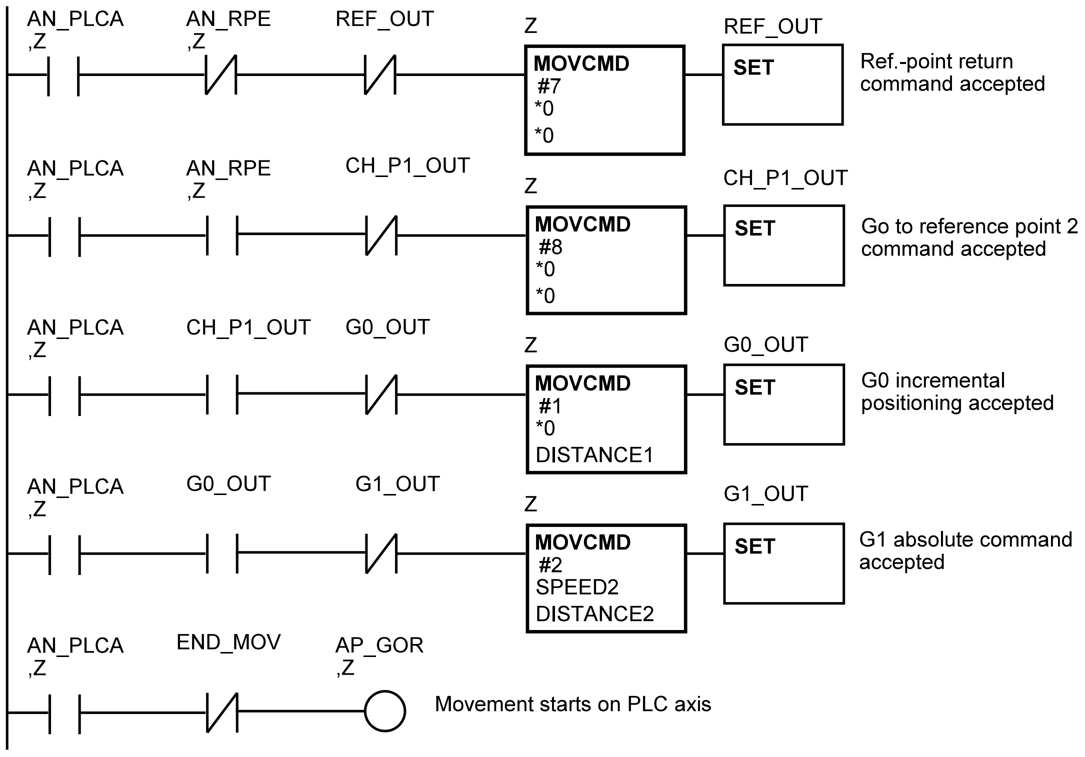
It sets error flag FL_ER in the following cases:
The axis to be moved has not been requested by PLC,
The code of instruction is wrong (correct codes are in the following chart),
In case of movements, where the parameter Speed is necessary and the value of the parameter is 0.

Interpretation of **Instruction Code**:

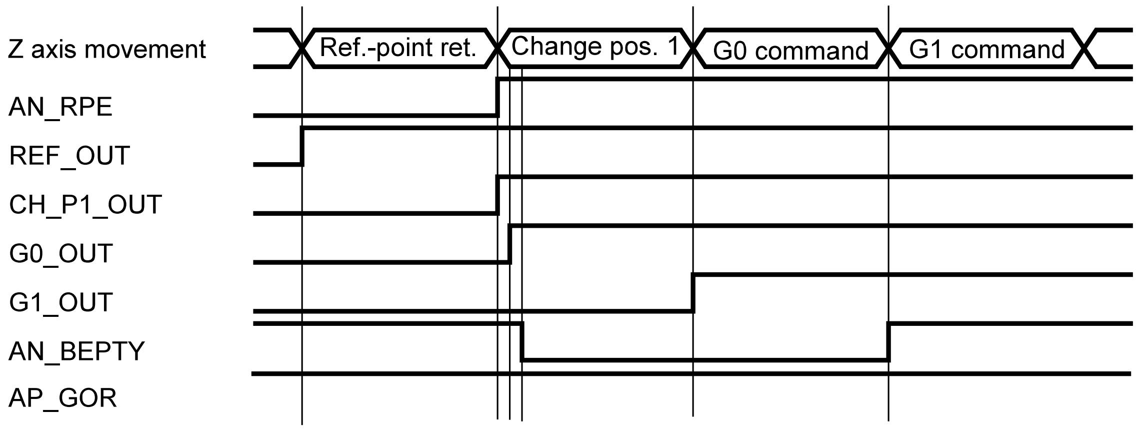
Com. Code	**Equivalent NC code**	**Description**	**Speed**	**Distance**
0	G00 G90	Rapid positioning	Not used	Absolute position defined in the actual coordinate system
1	G00 G91	Rapid positioning	Not used	Incremental movement
2	G01 G90 G94	Linear interpo-lation	Feed per minute	Absolute position defined in the actual coordinate system
3	G01 G91 G94	Linear interpo-lation	Feed per minute	Incremental movement
4	G01 G90 G95	Linear interpo-lation	Feed per revolution	Absolute position defined in the actual coordinate system
5	G01 G91 G95	Linear interpo-lation	Feed per revolution	Incremental movement
6	G53	Positioning in the machine coordinate system	Not used	Absolute position defined in the actual coordinate system
7	G28	Automatic reference point return	Not used	Not used

Com. Code	Equivalent NC code	Description	Speed	Distance
8	G30 P2	Automatic return to reference point 2	Not used	Not used
9	G30 P3	Automatic return to reference point 3	Not used	Not used
10	G30 P4	Automatic return to reference point 4	Not used	Not used
11	G4 G94	Dwell	Not used	Dwell time in seconds
12	G4 G95	Dwell	Not used	Dwell time in spindle revolutions
13	JOG movement. This instruction is deleted by a new instruction or by the flag AP_RES	AP_JOGP=1: moves to positive direction, AP_JOGN=1: moves to negative direction	Feed per minute: if its value=0: then moves according to parameter 0316 Jog F Contr , if its value >0 then moves by the defined feed	Not used
14	Handwheel movement. This instruction deleted by a new instruction or by the flag AP_RES	If one of P_HnAS registers is assigned to the PLC axis, it is controlled by the appropriate handwheel	Not used	Not used

There is below an example of application of axis control by PLC. The MOVCMD instructions follow each other. For the first, the example checks if reference-point has been returned on Z-axis. If not, it issues the instruction for reference-point return, if returned, then it sends the axis to the reference point 2 (change place 1) immediately. Then it starts a rapid, incremental positioning, with DISTANCE1, then it starts linear interpolation to the absolute position DISTANCE2 by a feedrate of SPEED2.



The interpolator immediately takes off the command reference point return from the buffer, therefore flag AN_BEPTY (buffer is empty) is not changed, so it indicates that the buffer is empty.
The next instruction (move to change position 1.) issued after flag “reference point found” set (AN_RPE=1). The interpolator takes off it immediately from the buffer (AN_BEPTY is still 1), then G0 instruction is written into the buffer in the following PLC cycle. After one PLC cycle delay, the state of AN_BEPTY=0 indicates that buffer is not empty: the interpolator moves into change position 1 and G0 command is in the buffer. If the axis has arrived at change position 1, then the interpolator takes off the instruction G0 from the buffer, G1 instruction is written immediately, so the AN_BEPTY flag keeps the value 0. After finishing G0 command, the interpolator takes off the G1 instruction from the buffer and AN_BEPTY=1 because there is no more instruction.
If continuous movement is necessary, then PLC programmer does not have to inspect AN_BEPTY flag, because MOVCMD instruction does it.



6.16 Read and Write of Common Macro Variables

The values of common macro variables #100...#499 and #500...#599 can be written and read from the PLC program.
Since these macro variables exist in each channels, always define the index of the channel when using this instruction.

6.16.1 Read of Common Macro Variables: the MACR Instruction

Common macro variables of a channel (#100...#499 and #500...#599) can be read by using MACR instruction. These variables are always floating-point types, so always have to reserve 2 double word space for them.

Input and output of MACR instruction

The MACR instruction has got an

enable input.

The read is done in the TRUE state of the input.
You can *configure* an

output for MACR instruction.

If the input is in TRUE state and reading has executed, then the output will be in TRUE state. The output will be FALSE if the instruction cannot be executed, so the instruction sets the error flag FL_ER .
You can connect further instructions to its output.

Input parameters of MACR instruction

– **Address of Result**:
Range of the value: n...9998 (n: where starts user addresses)
The Address of Result can be a numeral or a symbol. Indexed addressing is possible.
The value read is stored at this address. *You always have to reserve two double words because the result is a floating-point value!*

– **Number of macro variable**:
The Number of macro variable can be a register reference or a constant. Indexed addressing is possible.
Character # is to be used in case of numerical specification.
Range of the value in numerical format: #100...#999.

– **Channel**:
Index of a channel where macro variable is to be read from. It can be also in numerical or symbolic format.

– **Output**:
You can connect further instructions in its output in enabled state.

– **Remark**:
If the output is disabled, then the remark written here displayed as a comment. If you did not write here any remark, comment of the symbol defined in the Address of Result box is written here.

If the macro variable defined is out of range #100...#999, or the index of the specified channel is wrong, the output will be FALSE and the flag FL_ER will be 1.

6.16.2 Write of Common Macro Variables: the MACW Instruction

Common macro variables of a channel (#100...#499 and #500...#599) can be read by using MACW instruction. These variables are always floating-point types, so always have to reserve 2 double word space for them in PLC memory.

Input and output of MACW instruction

The MACW instruction has got an

enable input.

The writing is done in the TRUE state of the input.

You can *configure*

output for MACW instruction.

If the input is in TRUE state and write is executed, then the output will be in TRUE state. The output will be FALSE if the instruction cannot be executed, so the instruction sets the error flag FL_ER.

You can connect further instructions to its output.

Input parameters of MACW instruction

– **Number of macro variable**:

The Number of macro variable can be a register reference or a constant. Indexed addressing is possible.

Character # is to be used in case of numerical specification.

Range of the value in numerical format: #100...#999.

– **Operand**:

Range of the value: n...9998 (n: where starts user addresses)

The address of source PLC register can be a numeral or a symbol. Indexed addressing is possible.

You always have to reserve two double words because the result is a floating-point value!

– **Channel**:

Index of a channel where macro variable is to be written to. It can be also in numerical or symbolic format.

– **Output**:

You can connect further instructions to its output in enabled state.

– **Remark**:

If output is disabled, then the remark written here displayed as a comment. If you have not written here any remark, comment of the symbol defined in the Number of macro variable box written here.

If the macro variable defined is out of range #100...#999, or the index of the specified channel is wrong, the output will be FALSE and the flag FL_ER will be 1.

6.17 Query of the Internal Variables of the NC: the SCP Instruction

The internal variables, which can be visualized by the oscilloscope function built in the NC, can be queried by using the SCP instruction. These variables can be of several types: bit, double word or floating-point type. Although the result is always floating-point type!

Input and output of SCP instruction

The SCP instruction has got an

enable input.

The query is executed if the enable input is in TRUE state.

You can *configure* an

output for MACW instruction.

If the input is TRUE and the query has been executed, the output will be in TRUE state.

You can connect further instructions to its output.

Input parameters of SCP instruction

– **Address of Result**:

Range of the value: n...9998 (n: where starts user addresses)

The Address of Result can be a numeral or a symbol. Indexed addressing is possible.

The value queried is stored at this address. You always have to reserve two double words because the result is a floating-point value!

– **Scope Symbol**:

Select the symbol of data to be queried from the drop-down menu. The description of the symbols is in the following chart.

– **Id**:

ID number of symbol selected from menu. It need not be specified.

– **Param1**:

Value of the 1st parameter belonging to the selected symbol: #<number>. The rules of definition are in the following chart.

– **Param2**:

Value of the 2nd parameter belonging to the selected symbol: #<number>. The rules of definition are in the following chart.

– **Output**:

You can connect further instructions to its output in enabled state.

– **Remark**:

If the output is disabled, then the remark written here displayed as a comment. If you have not written here any remark, comment of the symbol defined in the Address or Result box is written here.

NC variables defined by the following symbols can be queried by using SCP instruction:

Symbol	Description	Param1	Param2
PlcBit	Query of a PLC bit variable from the PLC memory. Floating-point type data.	Double word address: a numeral, behind # character Value: #0...#9999	Bit address: a numeral, behind # character Value: #0...#31
PlcInt	Query of a PLC integer variable from the PLC memory. Floating-point type data.	Double word address: a numeral, behind # character Value: #0...#9999	#0
PlcDouble	Query of a PLC floating-point variable from the PLC memory. Floating-point type data.	A floating-point variable address: a numeral, behind # character Value: #0...#9998	#0
DirectPlcBit	Query of PLC bit variable from TimeSlice memory. Floating-point type data.	Double word address: a numeral, behind # character Value: #0... #<PLCNVRAM-1>	Bit address: a numeral, behind # character Value: #0...#31
DirectPlcInt	Query of PLC integer variable from TimeSlice memory. Floating-point type data.	Double word address: a numeral, behind # character Value: #0... #<PLCNVRAM-1>	#0
DirectPlcDouble	Query of PLC floating-point variable from TimeSlice memory. Floating-point type data.	A floating-point variable address: a numeral, behind # character Value: #0... #<PLCNVRAM-2>	#0
ComPosAx	Position command sent to the position control loop of an axis. Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0

Symbol	Description	Param1	Param2
ComVelAx	Velocity command sent to the position control loop of an axis. Floating-point type data in output units (mm/sec, degree/sec, inch/sec).	Axis number. Value: #1...#32	#0
FolErrAx	Following error of an axis (lag). Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0
CommandAx	Command signal of an axis sent to servo drive. Floating-point type data in output units (mm/sec, degree/sec, inch/sec).	Axis number. Value: #1...#32	#0
ActPosAx	Actual position of an axis measured by encoder. Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0
PosErrAx	Difference between the predicted position calculated from the position command and the position measured by encoder. Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0
TachAx	Axis speed measured by encoder. Floating-point type data in output units (mm/sec, degree/sec, inch/sec).	Axis number. Value: #1...#32	#0
TachRealAx	Axis speed measured by encoder. Quantity of pulses from encoder in Time Slice, in floating-point data.	Axis number. Value: #1...#32	#0
EGBvTarAx	Speed of EGB master axis. Floating-point type data in output units (mm/sec, degree/sec, inch/sec).	Define the slave spindle number! Value: #1...#32	#0

Symbol	Description	Param1	Param2
EGBcCurrAx	Speed of EGB slave axis. Floating-point type data in output units (mm/sec, degree/sec, inch/sec).	Axis number. Value: #1...#32	#0
SyncErrAx	Position difference between master and slave axes, in case of EGB or gantry synchronization. Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0
PitchAx	Current value of the pitch error compensation. Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0
StraightnessAx	Current value of the straightness compensation. Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0
CompenValAx	Sum of the current compensation values. Floating-point type data in output units (mm, degree, inch).	Axis number. Value: #1...#32	#0
ComPosSp	Position command sent to the position control loop of a spindle in case of a closed loop. Floating-point type data in spindle revolutions.	Spindle number. Value: #1...#16	#0
ComVelSp	Velocity command sent to the position control loop of a spindle in case of a closed loop. Floating-point type data in spindle revolution/sec.	Spindle number. Value: #1...#16	#0
FolErrSp	Following error of a spindle (lag) in case of a closed loop. Floating-point type data in spindle revolutions.	Spindle number. Value: #1...#16	#0

Symbol	Description	Param1	Param2
CommandSp	Command signal of a spindle sent to drive in case of a closed loop. Floating-point type data in spindle revolution/sec.	Spindle number. Value: #1...#16	#0
ActPosSp	Actual position of a spindle. Floating-point type data in spindle revolution.	Spindle number. Value: #1...#16	#0
PosErrSp	Difference between the predicted position calculated from the position command and the position measured by encoder, in case of a closed loop. Floating-point type data in spindle revolution.	Spindle number. Value: #1...#16	#0
TachSp	Spindle speed measured by encoder. Floating-point type data in spindle revolution/sec.	Spindle number. Value: #1...#16	#0
NActSp	Spindle speed measured by encoder. Floating-point type data in spindle revolution/min.	Spindle number. Value: #1...#16	#0
NSetSp	Speed command modified by override at the input of spindle handler. Floating-point type data in spindle revolution/min.	Spindle number. Value: #1...#16	#0
NCommandSp	Command signal for spindle drive modified by spindle handler. Floating-point type data in spindle revolution/min.	Spindle number. Value: #1...#16	#0
SyncVTargetSp	Master spindle speed in synchronized state. Floating-point type data in spindle revolution/sec.	Define the slave spindle number! Value: #1...#16	#0

Symbol	Description	Param1	Param2
SyncVSlaveSp	Slave spindle speed in synchronized state. Floating-point type data in spindle revolution/sec.	Spindle number. Value: #1...#16	#0
SyncErrSp	Difference between slave spindle and master spindle position in synchronized state. Floating-point type data in spindle revolution.	Spindle number. Value: #1...#16	#0
Measured	For internal use.		
RealTime	Time usage of the real time thread in µsec units. Floating-point type data.	#0	#0
HardwareTime	Refresh time of I/Os in µsec unit. Floating-point type data.	#0	#0
ChannelsTime	Time usage of channel control in µsec units. Floating-point type data.	#0	#0
AxesTime	Time usage of axis control in µsec units. Floating-point type data.	#0	#0
PlcTime	Time usage of Int0 PLC module in µsec units. Floating-point type data.	#0	#0
PlcCycle	Time usage of PLC Main in µsec unit. Floating-point type data. It contains real time interrupts.	#0	#0
CANErr	For internal use.		
TSliceErr	For internal use.		
CNCBufferCount	Quantity of records in buffer. Floating-point type data.	Channel number. Value: #1...#8	#0
RotaryAx	Value of cyclical position in case of a rotary axis. Floating-point type position data in degree unit.	Axis number. Value: #1...#32	#0

Symbol	Description	Param1	Param2
TachRealSp	Spindle speed measured by encoder. Quantity of pulses from encoder in Time Slice, floating-point data.	Spindle number. Value: #1...#16	#0
NCTDriveMess	Data sent by NCT servo drive, in floating-point data.	Axis- or spindle servo drive number: #1...#48	#0: motor speed [rev/min]
			#1: motor current [A],
			#2: motor relative current I/I_n [%]
			#3: bus voltage [V]
			#4: motor temperature [°C]
			#5: motor power [kW]

6.18 Reading and Writing NC Memory Arrays

An array of NC memory can be read into the PLC memory by using array read instructions.
An array of PLC memory can be written into the NC memory by using array write instructions.

6.18.1 NC Memory Array Read: the MR Instruction

A data arrays, stored in NC memory can be read by using this instruction.

Input and output of MR instruction

The MR instruction has got an
enable input.
Reading is executed in TRUE state of the enable input.
You can *configure* an
output for MR instruction.
If the input is TRUE and read has been executed, the output will be TRUE.
You can connect further instructions to its output.

☞ ***Attention!*** *The execution of instruction may last for several PLC cycles, so the enable input has to be kept in TRUE state until the output of instruction changes into TRUE state. Then the enable input has to be reset.*

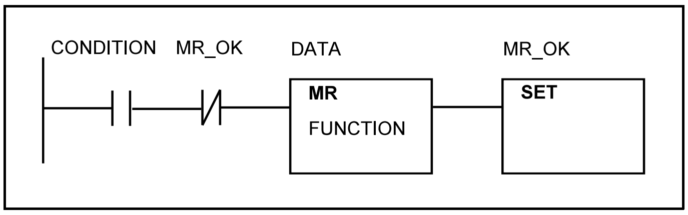
Input parameters of MR instruction

– **Data Address**:
Range of the value: 0...9999
Data Address is the starting address of the PLC memory array. It can be a numeral or a symbol. Indexed addressing is possible.
Messages of the instruction execution and data read from the NC are written here.

– **Function Code**:
It specifies the part of NC memory where the data array is to be read from. The different function codes are described in the following chapters.

– **Output**:
You can connect further instructions to its output in enabled state.

– **Remark:**
Remark.



The space in PLC memory to be reserved is the function of the amount of data to be read. The instruction writes execution information at the address and at the following one specified in parameter Data Address.

- **Address**: Data array starts at the address defined in parameter Data Address. The execution code is written here.
- **Execution code**: Output data, it is created during the execution of the instruction.
- **Amount of data**: Input data. It determines the amount of double word memory space reserved for data to be read.
- **Function specific memory space**: Special data, defined by the parameter Function Code are written here. These data can be:
  - Further input data configuring the MR instruction and
  - Data of the memory read.

Address	**Data**
0	Execution code
1	Amount of data (input data)
2	Function specific memory
3	
...	...
n	...

☞*Attention! The amount of data is always the sum of data configuring the instruction and the amount of data to be read in DWORD units!*

Execution code:

Code	**Explanation**
0	Normal execution
1	Invalid function code
2	Invalid amount of data
..	Function specified errors
n	

6.18.2 NC Memory Array Write: the MW Instruction

Data arrays, stored in NC memory can be written by using this instruction.

Input and output of MW instruction

The MW instruction has got an
enable input.
Writing is executed in TRUE state of the enable input.
You can *configure* an
output for MW instruction.
If the input is TRUE and writing has been executed, the output will be TRUE.
You can connect further instructions to its output.

☞ **Attention!** *The execution of instruction may last for several PLC cycles, so the enable input has to be kept in TRUE state until the output of instruction changes into TRUE state. Then the enable input has to be reset.*

Input parameters of MW instruction

– **Data Address**:

Range of the value: 0...9999

Data Address is the starting address of the PLC memory array. It can be a numeral or a symbol. Indexed addressing is possible.

Messages of the instruction execution are written here and data to be written to the NC are read from here.

– **Function Code**:

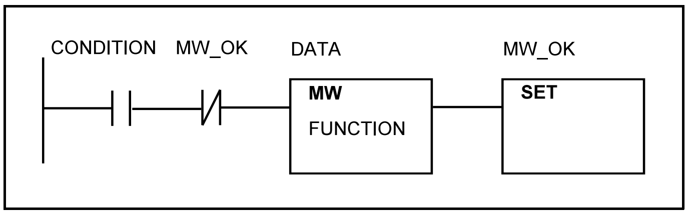
It specifies the part of NC memory where the data array is to be written to. The different function codes are described in the following chapters.

– **Output**:

You can connect further instructions to its output in enabled state.

– **Remark:**

Remark.



The space in PLC memory to be reserved is the function of the amount of data to be written. The instruction writes execution information at the address and at the following one specified in parameter Data Address.

Address	Data
0	Code of execution
1	Amount of data (input data)
2	Function specific memory
3	
...	...

– **Address**: Data array starts at the address defined in parameter Data Address. The execution code is written here.

– **Code of execution**: Output data, it is created during the execution of the instruction.

– **Amount of data**: Input data. It determines the amount of double word memory space reserved for data to be written.

– **Function specific memory space**: Special data, defined by the parameter Function Code, are written here. These data can be:

– Further input data configuring the MW instruction and

– Data to be written.

☞***Attention!*** *The amount of data is always the sum of data configuring the instruction and the amount of data to be written in DWORD units!*

Code of execution:

Code	Explanation
0	Normal execution
1	Invalid function code
2	Invalid amount of data
3	Overwrite protected
...	Function specified errors
n	

6.19 Transferring Data between Non-volatile Memory and PLC

Array of PLC variables can be saved to or loaded from non-volatile memory.
Some controls save automatically the 1024 double word long data array started from PLCVRAM address, PLC programmer does not have to save it by own.
Other types of control *do not support the automatic storage of* 1024 double word long data array started from PLCVRAM address *after switching off*.
Data changed are to be saved by array write instruction from the PLC program. Similarly, the saved data array is to be loaded by array read instruction from the PLC program, for example after switching on. Control also reserves 1024 double word long memory space for saving PLC variables.

The identifier of first variable in non-volatile memory: 0,
and the last one is 1023.

Executing MW, MR instructions for saving and loading PLC variables take for several PLC cycles, so you always have to wait for their output turn into TRUE state.

6.19.1 PLC Data Read from Non-volatile Memory

Instruction: MR
Function code: 10

Reading the variables of an array defined by its starting address and the number of variables from the non-volatile memory to the PLC memory.

Input parameters:

Address	Data
0	Code of execution *
1	Amount of data: 3
2	Starting address of the first variable in non-volatile memory to be read: 0...1023 (source)
3	Starting address of the first variable is in PLC memory to be read (destination)
4	Number of variables to be read: 1...1024

*: need not be set

The amount of data has to be equal to 3. PLC memory is loaded continuously from the destination address in case of reading several variables.

Output parameters:

Address	Data
0	Code of execution: refer to table
1	Amount of data: 3
2	Starting address of the first variable in non-volatile memory to be read: 0...1023 (source)
3	Starting address of the first variable is in PLC memory to be read (destination)
4	Number of variables to be read: 1...1024

Code of execution can be:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid amount of data: is not equal to 3
4	Memory checksum error
10	Array specified in non-volatile memory not in the range between 0...1023
11	Starting address specified in the PLC memory is not higher than or equal to PLCNVRAM
12	Data number error: (address of first variable to be read)+ (number of variables to be read)>1024

6.19.2 PLC Data Write to Non-volatile Memory

Instruction: MW
Function code: 11

Writing the variables of an array defined by its starting address and the number of variables from the PLC memory to the non-volatile memory. The output of the instruction returns 0 code of execution only after the calculation of the checksum of memory.

Input parameters:

Address	Data
0	Code of execution *
1	Amount of data: 3
2	Starting address of the first variable in non-volatile memory to be written: 0...1023 (destination)
3	Starting address of the first variable is in PLC memory to be written (source)
4	Number of variables to be written: 1...1024

*: need not be set

The amount of data has to be equal to 3. The non-volatile memory is loaded continuously from starting address in case of writing several variables.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 3
2	Starting address of the first variable in non-volatile memory to be written: 0...1023 (destination)
3	Starting address of the first variable is in PLC memory to be written (source)
4	Number of variables to be written: 1...1024

Code of execution can be:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid amount of data: is not equal to 3
10	Array specified in non-volatile memory not in the range between 0...1023
11	Starting address specified in the PLC memory is not higher than or equal to PLCNVRAM
12	Data number error: (address of first variable to be read)+ (number of variables to be read)>1024

6.20 Reading and Writing Parameters from the PLC Program

By the use of MR and MW instruction, the parameters of the NC are accessible for the PLC program, for reading or writing.
In the course of reading and writing, the following point have to be taken into account:
data representation of the parameter,
whether the parameter is global one or it is indexed one.
In terms of data representation, the following kinds of parameters can be distinguished:
– ***bit-type ones***,
– ***DWORD-type ones: they are fixed-point, double-word*** parameters,
– ***double-type ones: with floating point*** representation**.**
Space reservation from PLC of the bit-type parameters is DWORD. The different bit values are represented on the bits 0 ... 7 of DWORD.
A parameter can be indexed according to:
– **machine group**,
– **channel,**
– **axis** or
– **spindle**.
The indexed parameters do not have separate identification number.
A parameter can be referred to by its identification number and its index. When a global parameter is referred to, the value of the index is 0. Array-type parameter writing or reading (reading or writing several parameters simultaneously) is possible only for non-global parameters. For example, reading a parameter being different by spindles can be executed for all the spindles at the same time.
Identification number (Nnnnn) and format (bit, DWORD, double) of a parameter and the basis of its indexing can be found at the description of that given parameter.

6.20.1 Reading the DWORD-type Parameters

Instruction: MR
Function code: 31

Reading the data of the DWORD-type parameters specified by the use of their identification number and their initial index , among the PLC variables. The amount of data to be read is determined by the Amount of data parameter of the instruction.

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: min. 3, max. n+2
2	Identifier of the parameter to be read: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be read
4	Value of the 1st parameter*

Address	Data
5	Value of the 2nd parameter*
...	...
n+3	Value of the nth parameter*

*: need not be set

The minimum number to be written for the Amount of data is 3, in such a case only one double word will be read. In the case of reading several data, the PLC memory will be filled continuously.
If, for example, a parameter indexed by axis is to be read and the parameter of the 2nd and the 3rd axes is to be known, the following values will be valid: Amount of data=4 and Initial index=2.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: min. 3, max. n+2
2	Identifier of the parameter to be read: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be read
4	Value of the 1st parameter
5	Value of the 2nd parameter
...	...
n+3	Value of the nth parameter

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data: < 3 or > (index of the last variable of the parameter array − number of the specified initial index) + 2

Code	Explanation
4	Memory checksum error
30	Referring to a non-existing parameter
31	The parameter is not global (parameter index = 0)
32	The parameter is global (parameter index indexe > 0)
33	Wrong reading code: the parameter format is not DWORD-type

6.20.2 Reading the Double-type Parameters

Instruction: MR
Function code: 32

Reading the data of the double-type parameters specified by the use of their identification number and their initial index , among the PLC variables. The amount of data to be read is determined by the Amount of data parameter of the instruction.

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: min. 4, max. 2n+2
2	Identifier of the parameter to be read: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be read
4	Value of the 1st parameter*
5	
6	Value of the 2nd parameter*
7	
...	...
2n+2	Value of the nth parameter*
2n+3	

*: need not be set

The minimum number to be written for the Amount of data is 4, in such a case only one floating-point data will be read. In the case of reading several data, the PLC memory will be filled

continuously.
If, for example, a parameter indexed by axis is to be read and the parameter of the 2nd and the 3rd axes is to be known, the following values will be valid: Amount of data=6 and Initial index=2.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: min. 4, max. 2n+2
2	Identifier of the parameter to be read: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be read
4	Value of the 1st parameter
5	
6	Value of the 2nd parameter
7	
...	...
2n+2	Value of the nth parameter
2n+3	

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data: < 3 or > (index of the last variable of the parameter array − number of the specified initial index) ×2 + 2
4	Memory checksum error
30	Referring to a non-existing parameter
31	The parameter is not global (parameter index = 0)
32	The parameter is global (parameter index > 0)
33	Wrong reading code: the parameter format is not double-type

6.20.3 Writing the DWORD-type Parameters

Instruction: MW
Function code: 34

Writing the data of the DWORD-type parameters specified by the use of their identification number and their initial index, from the PLC. The amount of data to be written is determined by the Amount of data parameter of the instruction

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: min. 3, max. n+2
2	Identifier of the parameter to be written: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be written
4	Value of the 1st parameter
5	Value of the 2nd parameter
...	...
n+3	Value of the nth parameter

*: need not be set

The minimum number to be written for the Amount of data is 3, in such a case only one double word will be written. In the case of writing several data, writing from the PLC memory will be executed continuously.
If, for example, a parameter indexed by axis and the parameter of the 2nd and the 3rd axes is to be written, the following values will be valid: Amount of data=4 and Initial index=2.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: min. 3, max. n+2
2	Identifier of the parameter to be written: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be written
4	Value of the 1st parameter

Address	Data
5	Value of the 2nd parameter
...	...
n+3	Value of the nth parameter

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data: < 3 or > (index of the last variable of the parameter array − number of the specified initial index) + 2
30	Referring to a non-existing parameter
31	The parameter is not global (parameter index = 0)
32	The parameter is global (parameter index > 0)
33	Wrong writing code: the parameter format is not DWORD -type

6.20.4 Writing Double-type Parameters

Utasítás: MR
Funkció kód: 35

Writing the data of the double-type parameters specified by the use of their identification number and their initial index, from the PLC. The amount of data to be written is determined by the Amount of data parameter of the instruction.

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: min. 4, max. 2n+2
2	Identifier of the parameter to be read: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be read
4	Value of the 1st parameter
5	
6	Value of the 2nd parameter
7	
...	...
2n+2	Value of the nth parameter
2n+3	

*: need not be set

The minimum number to be written for the Amount of data is 4, in such a case only one floating-point data will be written. In the case of reading several data, writing from the PLC memory will be executed continuously.
If, for example, a parameter indexed by axis is to be read and the parameter of the 2nd and the 3rd axes is to be known, the following values will be valid: Amount of data=6 and Initial index=2.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: min. 4, max. 2n+2
2	Identifier of the parameter to be read: <number>
3	Initial index of the parameter: =0: global parameter >0 index of the 1st parameter to be read
4	Value of the 1st parameter
5	
6	Value of the 2nd parameter
7	
...	...
2n+2	Value of the nth parameter
2n+3	

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data: < 3 or > (index of the last variable of the parameter array − number of the specified initial index) ×2 + 2
4	Memory checksum error
30	Referring to a non-existing parameter
31	The parameter is not global (parameter index = 0)
32	The parameter is global (parameter index > 0)
33	Wrong writing code: the parameter format is not double-type

6.21 Assigning a Program for Execution

Arbitrary part program stored in the memory of the control may be assigned for execution.

6.21.1 Assigning a Program Specified with its Program Number for Automatic Execution

Instruction: MW
Function code: 40

Assigning a part program specified with its 4-digit or 8-digit program number (Onnnn or Onnnnnnnn) for execution in automatic mode, in the specified channel. The write operation will be executed only in the case, i.e. the output of the instruction will be true only in the case when no program is running in the automatic mode.

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: 2
2	Program number: <number>
3	Channel number: 1...8

*: need not be set

For Amount of data, always 2 has to be written.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 2
2	Program number: <number>
3	0

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data: not 1
40	Non-existing program: the givenprogram is not in the memory

6.22 Network Communication Instructions

The PLC program can receive and send data through the Ethernet network. Communication is realized via UDP protocol. Structure of an Ethernet UDP/IP frame is as follows:

Ethernet header UDP/IP	**Sent or received data**			**CRC**
	...	...	...	

The PLC works by the use of data sent or received.

6.22.1 Opening the Network Connection

Instruction: MW
Function code: 60

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: 5
2	Connection identifier: always 1
3	Windows error code *
4	Protocol: 1: UDP 2: TCP server 3: TCP client
5	IP address of the target device. For example: 192.168.1.12 = #$C0A8010C Its value must not be: 0.0.0.0 és 255.255.255.255
6	The number of the port through which communication is intended. Proposed values: 49202...49205

*: need not be set
For Amount of data, always 5 has to be written.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 5
2	Connection identifier: always 1
3	Windows error code: if the code of execution is 61, the re-decoded error sent by Windows will be placed here (see Windows Sockets Error Codes) *

Address	Data
4	Protokoll: 1: UDP 2: TCP server 3: TCP client
5	IP address of the target device. For example: 192.168.1.12 = #$C0A8010C Its value must not be: 0.0.0.0 és 255.255.255.255
6	The number of the port through which communication is intended. Proposed values: 49202...49205

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data
60	Invalid connection identifier
61	Establishing connection is failed
66	False input parameters

6.22.2 Closing the Network Connection

Instruction: MW
Function code: 61

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: 1
2	Connection identifier: always 1

*: need not be set
For Amount of data, always 2 has to be written.

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 1

Address	Data
2	Connection identifier: always 1

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data
60	Invalid connection identifier

6.22.3 Receiving the Network Data Packet

Instruction: MR
Function code: 62

Output parameters:

Address	Data
0	Code of execution*
1	Amount of data: 3-5
2	Connection identifier: always 1
3	Initial address of the data array in the PLC memory
4	Data array length: so many data of PLC register will be received 0 < value < = 350
5	Windows error code **
6	IP address of the source device. For example:: 192.168.1.12 = #$C0A8010C **
7	Number of the source device port**

*: need not be set
**: optional data, need not be set

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 3-5

Address	Data
2	Connection identifier: always 1
3	Initial address of the data array in the PLC memory
4	Data array length: so many data of PLC register will be received 0 < value < = 350
5	Windows error code: if the code of execution is 62, the re-decoded error sent by Windows will be placed here (see Windows Sockets Error Codes)
6	IP address of the source device. For example: 192.168.1.12 = #$C0A8010C

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data
60	Invalid connection identifier
62	Error while receiving the data
63	The specified connection is not open
64	The data array length is false: =0 or >350
65	Data is not received

6.22.4 Sending the Network Data Packet

Instruction: MW
Function code: 63

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: 3-5
2	Connection identifier: always 1
3	Initial address of the data array in the PLC memory
4	Data array length: so many data of PLC register will be received 0 < value < = 350

Address	Data
5	Windows error code**
6	IP address of the target device. For example: 192.168.1.12 = #$C0A8010C **
7	Number of the target device port. For example: 49202 **

*: need not be set
**: optional data, it has to be set only when UDP is used

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 3-5
2	Connection identifier: always 1
3	Initial address of the data array in the PLC memory
4	Data array length: so many data of PLC register will be received 0 < value < = 350
5	Windows error code: if the code of execution is 62, the re-decoded error sent by Windows will be placed here (see Windows Sockets Error Codes)
6	IP address of the source device. For example: 192.168.1.12 = #$C0A8010C

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data
60	Invalid connection identifier
62	Error while sending the data
63	The specified connection is not open
64	The data array length is false: =0 or >350

6.23 Opening a Window in the Screen of the Control

Windows, which can be displayed from the menu system of the control by operator's intervention, can be selected from the PLC program by the use of MW instruction, too.

Instruction: MW
Function code: 70

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: 2-4
2	Window identifier: see the table
3	Channel number: =0: window independent of channel =1...8: window has to be displayed together with data valid in the channel
4	Parameter 1: see the table
5	Parameter 2: see the table

*: need not be set

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 2-4
2	Window identifier: see the table
3	Channel number: =0: window independent of channel =1...8: window has to be displayed together with data valid in the channel
4	Parameter 1: see the table
5	Parameter 2: see the table

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data

Code	Explanation
6	Invalid channel number
70	Invalid window number
71	There is no connection to the display (the display is not receive ready)

Identification the windows that can be displayed from the PLC program

Window name	Identifier	Parameter 1	Parameter 2
Alfanumerical keyboard	100	-	-
PLC buttons	110	-	-
Window selection buttons	120	-	-
Machine operation panel	130	-	-
Active G codes in the view set last		-1	
Active G codes	1000	0	-
Active G codes together with text	1000	1	-
All the G codes	1000	2	-
All the G codes together with text	1000	3	-
Active M codes in the view set last		-1	
Active M codes	1010	0	-
Active M codes together with text	1010	1	-
All the M codes	1010	2	-
All the M codes together with text	1010	3	-
FST	1020	-	-
All the spindles	1030	-	-
Single block input	1040	-	-
Program list	1050	-	-
Left directory window	1090	-	-
Right directory window	1091	-	-
List of programs running (main programs and subprograms)	1110	-	-

Window name	Identifier	Parameter 1	Parameter 2
All the messages	1130	-	-
Time/workpiece counter	1150	-	-
Position	3000	-	-
Workpiece probing	3011	-	-
Tool probing	3012	-	-
Tool compensation table for milling machine	3020	-	-
Tool compensation table for lathe	3031	-	-
Tool compensation table 2 for lathe	3032	-	-
Zero point offset window	3040	-	-
Tool management table	4010	-	-
Tool pocket table	4020	-	-
Particular window for tool construction	4030	Magazine number	Pocket number
Tool shape table	4040	-	-
Aggregate tool life table	4050	-	-
#1...#33 local macro variables	5010	-	-
#100...#499 global macro variables	5020	-	-
#500...#999 global macro variables	5030	-	-
Calculator	6000	-	-
Cutting speed calculation	6010	-	-
Division counter	6020	-	-
DXF converter	6030	-	-
Setting the spindle orientation position	6040	-	-
Setting the spindle phase shift	6041	-	-
Workpiece probe calibration	6060	-	-
Workpiece surface measurement	6061	-	-
Workpiece corner measurement	6062	-	-

Window name	Identifier	Parameter 1	Parameter 2
Workpiece pocket/web, bore/boss measurement	6063	-	-

6.24 Dumping the Drive Data

The values of current required by the motors can be visualized on position display and in the FST window, the speed of the spindle motors can be displayed in the FST window. Data of drives manufactured by the NCT are received automatically by the control from the drives.
In the case of drives other than manufactured by the NCT, the PLC program can read the different data from the drive or, for example, it can read in the current data through an A-D converter, and it can transmit them to the control for displaying, using the instruction 81.

Instruction: MW
Function code: 81

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: 3
2	Drive address: value set in the N0501 Axis Input Address or in the N0601 Spindle Input Address parameter
3	Type of the data transmitted: 0 – Motor speed [1/min] 1 – Instantaneous value of the motor current [10 mA] (125 = 1,25 A) 2 – Motor load (relative current) [%] 3 – DC bus voltage [V] 4 – Motor temperature [°C] 5 – Instantaneous motor power [kW]
4	Value of the data transmitted (DWORD)

*: need not be set

Output parameters:

Address	Data
0	Code of execution*
1	Amount of data: 3
2	Drive address: value set in the N0501 Axis Input Address or in the N0601 Spindle Input Address parameter

Address	Data
3	Type of the data transmitted: 0 – Motor speed [1/min] 1 – Instantaneous value of the motor current [10 mA] (125 = 1,25 A) 2 – Motor load (relative current) [%] 3 – DC bus voltage [V] 4 – Motor temperature [°C] 5 – Instantaneous motor power [kW]
4	Value of the data transmitted (DWORD)

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data
80	Invalid drive address
81	Invalid data type
82	Invalid transmitted data value

6.25 Writing the Position Data

The position recorded in the axis controlling module can be overwritten by the PLC program if
– the position control loop is open for the given axis (AN_OPNA=1), and
– receiving the position from the encoder is disabled (AP_EFD=1 or N0514 Servo Control parameter #4 EFD=1).
Due to the instruction, the state ***reference point is taken will be recorded*** by the axis controlling module ***for the given axis*** (AN_RPE=1).
This instruction can be used, for example, when a rotating axis is placed on a Hirth disc, and it is the PLC program that executes position recording and saving. In such a way, positioning to the reference point after turn-on can be activated. Its application is useful particularly in the case when positions of several axes are measured by one encoder.

Instruction: MW
Function code: 90

Input parameters:

Address	Data
0	Code of execution*
1	Amount of data: 3
2	Axis index (0, ..., 31)
3	Position to be written in the machine coordinate system (double)
4	

*: need not be set

Output parameters:

Address	Data
0	Code of execution
1	Amount of data: 3
2	Axis index (0, ..., 31)
3	Position to be written in the machine coordinate system (double)
4	

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data
30	Invalid axis index: <0 or >31
90	Internal error, the instruction cannot be used
91	The axis index indicates a non-existing axis, or the position control loop is not open (AN_OPNA=0), or receiving the position from the encoder is not disabled (AP_EFD=0)

6.26 Reading and Writing Data of Tool Management Table

Tool management function is to be used in the following cases:
– tool life counting is applied,
– referring to tools by code, not by pot numbers,
– tool change applied in machine requires random tool pot management.
Data of tool management are stored in tables in the NC. PLC program can read and write them by using MR and MW instructions.

6.26.1 The Tool Management Table

You can write the T codes of tools, used in the machine, into the tool management table. T codes, by which a tool is referred to in a part program, are called type numbers.
You can define the properties of tools in the table. They can be normal sized, or oversize ones, can take part in life management or not, an offset number can be assigned to them, etc. You can assign further machining data to a tool, e.g.: spindle speed, feed-rate, etc.
The tool management is common for each channel.

Data number	Type number (T)	Tool info	Figure number	Life status	Life counter	Life
1	10002001	UENCV		Expired	150	150
2	10002001	SENCV		Used	131	200
3	10002001	SENCV		Not used	0	170
4	150	SDLCV	2	Not performed		
5	3210	UENTV		Broken	1h42m26s	2h30m00s
6	3210	SENTV		Used	1h34m14s	1h50m30s
...						

Elements of tool management table:

1 Data number (DWORD)

It is the serial number of rows in the table. It cannot be edited, the number of rows in the table determined by parameter. Tools are registered by data number in the cartridge table.

2 Type number (T) (DWORD)

Tools are referred by their type numbers at the address T in a part program. If a group of the tools takes part in tool life management, then all tools in the same group are referred to by the same type number. Type number is specified by maximum 8 digits:

T: from 1 to 99 999 999.

That tool is selected by the NC among the tools with the same type number, life counter of which is maximal but it has not expired yet. According to the table above, the tool with data number 2 will be selected if you specify the T code

T10002001.

If each tool with the same type number has identical value of life counters, the tool with lower data number will be selected.

3 Tool info (DWORD)

The tool info contains the following bit information:

#0 **I** (=0, Invalid): the whole row of tool management table is invalid, it can be deleted

V (=1, Valid): the whole row of tool management table is valid.

#1 **C** (=0, Count): life management is done for counts of usage

T (=1, Time): life management is done for machining time.

☞ *Attention:*

You have to set the same C or T tool info for tools with the same type number.

#2 **N** (=0, Normal): The tool has normal size (it occupies one tool pot).

L (=1, Large): The tool is oversize, it occupies more tool pots.

#3 **E** (=0, Enabled): The whole row of tool management table can be edited.

D (=1, Disabled): The whole row of tool management table cannot be edited.

#4 If life status states that this tool does not take part in tool life management and this bit

U (=0, Unsearchable): NC does not search for this tool

4 Figure number (DWORD)

If an oversize tool is defined by “L” in the tool info column, then you have to specify the figure number of the tool. The tool pattern table contains the definitions of figure numbers of the oversize tools (pot occupied).

The figure number points to the row number of the tool pattern table.

If the value of figure number is 0 then the tool occupies 1 pot space in the cartridge. If a normal tool is defined in tool info by using “N”, then the value of figure number is invalid.

5 Life status (DWORD)

The life status contains the following codes:

=0: Not performed

If the tool does not take part in life management, but it is indicated by “S” in tool info column, then its status is “not performed”. The tool can be searched.

=1: Not used

If the tool takes part in life management but it has not been used yet, that is the value of its life counter is 0, then its status is “not used”. The tool can be searched.

=2: Used

If the tool takes part in life management and the value of its life counter is lower than the defined life then its status is “used”. The tool can be searched.

=3: Expired

If the tool takes part in life management and the value of its life counter has reached the defined life value then its status is “expired”. The tool cannot be searched.

=4: Broken

If PLC indicates, the tool is broken, then its status becomes “broken”. This status is the same as the “expired” status: The tool cannot be searched. This status is valid also for those tools which do not take part in life management.

6 Life counter (DWORD)

If bit #1 is ***set to C*** in the tool info column, tool life management is enabled and life management is done for counts of usage. It counts the execution of tool changes. When the FIN signal arrives after the execution of an M06 or a T code, then the value of life counter is incremented.

If the value of life counter reaches the value set in the life column then the status of tool will be “expired”. To count for M06 or T decided by parameter.

If bit #1 is ***set to T*** in the tool info column, tool life management is enabled and life management is done for the time spent with machining. It measures the time

spent in Start state and

when the value of override is not 0 and

the spindle is turning and

a movement with a feed-rate is done.

If the time measured by life counter has reached the value set in the life column, then the status of tool will be “expired”.

7 Life (DWORD)

You can set the life value of a tool for counts or time here. If the life counter of a tool reaches the value set here, then the status of tool will be “expired”.

Further elements of the table:

Data number	Type number (T)	Tool info	Notice life	H	D	S
1	10002001	UENCV	20	1	1	12500
2	10002001	SENCV	12	2	3	11400
3	10002001	SENCV	15	31	31	13000
4	150	SDLCV		12	13	5400
5	3210	UENTV	5m30s	326	326	2500

6	3210	SENTV	4m10s	63	63	2700
...						

8 Notice life (DWORD)

If the difference between the life and life counter reaches the value set on notice life then the control sends a warning signal to the PLC.

9 H: tool length compensation number (DWORD)

In a milling channel it is the tool length compensation number to be used (H code) if the tool takes part in life management.

10 D: cutter compensation number (DWORD)

In a milling channel it is the cutter compensation number to be used (D code) if the tool takes part in life management.

11 G: tool geometry compensation number (DWORD)

In a turning channel it is the tool geometry compensation number to be used if the tool takes part in life management.

12 W: tool wear compensation number (DWORD)

In a turning channel it is the tool wear compensation number to be used if the tool takes part in life management.

13 Spindle speed (DWORD)

Spindle speed for the tool in [1/min] unit.

Further elements of table

Data number	**Type number (T)**	**Tool info**	**F**	**User data 1**	**User data 2**	**...**
1	10002001	UENCV	3000			
2	10002001	SENCV	2800			
3	10002001	SENCV	3300			
4	150	SDLCV	650			
5	3210	UENTV	1800			
6	3210	SENTV	1700			
...						

14 Feed (double)

Feed-rate for the tool in mm/min, inch/min, or mm/revolution, inch/revolution unit.

15 User data 1 (DWORD)

You can define 8 ***bit type data*** per tool type, for arbitrary application.

16 User2, 3, 4, ...20 (double)

Columns for storing ***double type*** floating-point ***data***. Number of columns set in a parameter.

6.26.2 The Cartridge Management Table

Data numbers of tools in the pots of cartridges are written in the cartridge management table, that is the serial numbers of rows of the tool management table.
The cartridge management table is common for each channel.

Serial number	Cartridge number	Pot number	Data number
1	1	1	0
2	1	2	4
3	1	3	3
...	...	...	...
24	1	24	1
25	2	1	12
26	2	2	28
27	2	3	0
...	...	...	...
40	2	16	62
...	...	...	...
41	10	1	2
42	11	1	0
43	20	1	5
44	21	1	6
...	...	...	...

The cartridge management table is divided as the follows:

1 Serial number

It is the serial number of rows in the table. This is equal to the available tool pots in the machine.

2 Cartridge number

You can define up to 4 cartridges in parameter. Their serial numbers are ranging from 1 up to 4. It depends on the mechanical construction of machine tool that the tool changing arms how could reach the cartridges and which spindles could they supply.
You can define in parameters that cartridges are chain-type or matrix-type ones and the amount of tool pots in each cartridge.

Special tool cartridges

All spindles can be assigned by parameter to the cartridge table and a standby cartridge for spindles. The spindle and the standby cartridges are single-pot cartridges.

The cartridge number of the 1st spindle is 10, the cartridge number of the 2nd spindle is 20, and so on.

The numbers of standby cartridges assigned to the spindles (for example tool change arms) are 11, 21 and so on.

☞ *Remark.*

The tool machining may not be in a spindle but in a tool holder. E.g.: A vertical turning machine with a single table has 2x2 axes. Each channel is machining with a separate tool, which are fixed in the left and right tool holders. Spindles also have to be defined to both channels (E.g.: S1, S2), that is tool holders, because the PLC-NC communication flags belonging to tool management are handled per spindle cartridges.

3 Pot number

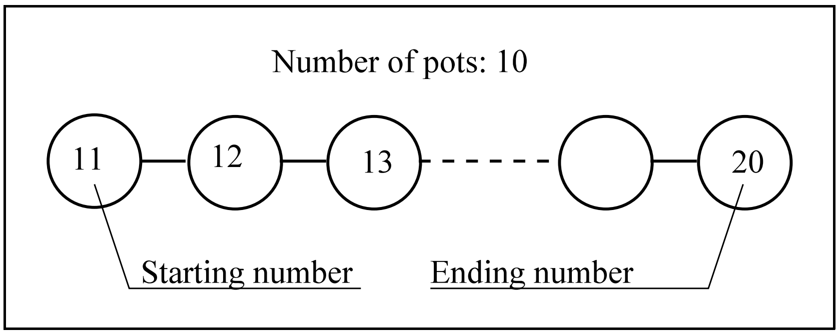
The pot numbers are increased by 1 in a cartridge. The starting pot number determined by parameter.

4 Data number

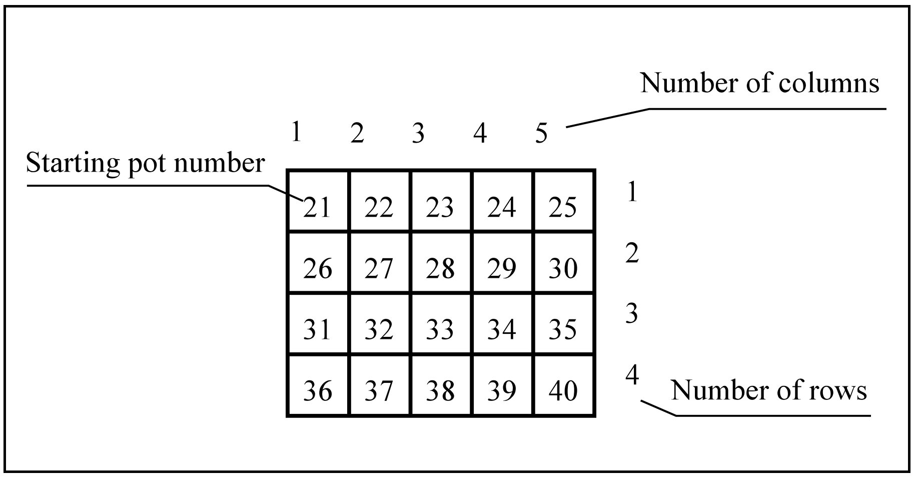
Data number of the tool management table is written here that specifies the tool in the pot. If the value of data number is 0, there is no tool in the pot.

6.26.3 The Tool Pattern Table

The figure of oversize tools, that may occupy several pots in the cartridge, can be determined in the tool pattern table.
In case of an oversize tool
– you have to set #2 bit of **Tool info** column of the tool management table into “L”, and
– you have to specify a code in the **Figure number** column of the tool management table. Figure number points to the appropriate row of the tool pattern table that describes the figure of the tool.
Types of each cartridges have to be determined by parameter before using tool pattern table (chain-type or matrix-type).
You also have to determine by parameter
– the starting pot number in case of **chain-type cartridges** (it is not necessary to start from 1), and the number of pots in the cartridge,



– the starting pot number in case of **matrix-type cartridges** (it is not necessary to start from 1), and the number of columns and rows of the matrix.



The control interprets the numbering of pots of matrix-type cartridges shown in the above figure, so it starts at the upper left corner and lasts till the right bottom corner.

The tool pattern table is common for each channel. The maximum number of rows of the tool pattern table is 20, so it can manage up to 20 different figures.

Figure number	**Left**	**Right**	**Up**	**Down**	**Geometry**
1					
...					

1 Figure number (DWORD)

It is the serial number of rows in the table. The number, determined in the figure number column of tool management table, points to the appropriate row of the tool pattern table.

2 Left, Right, Up, Down

The Left, Right columns of the table are also used in case of chain- and matrix-type cartridges. The Up, Down columns of the table are used only in case of matrix-type cartridges. The number written in columns shows the space requirement of the tool in the corresponding direction in

½ pot

units. It is not possible to put a tool into a pot, which is occupied by half. However an oversize tool can be placed in its adjacent pot that also occupies ½, 1 ½, 2 ½ pots in the opposite direction.

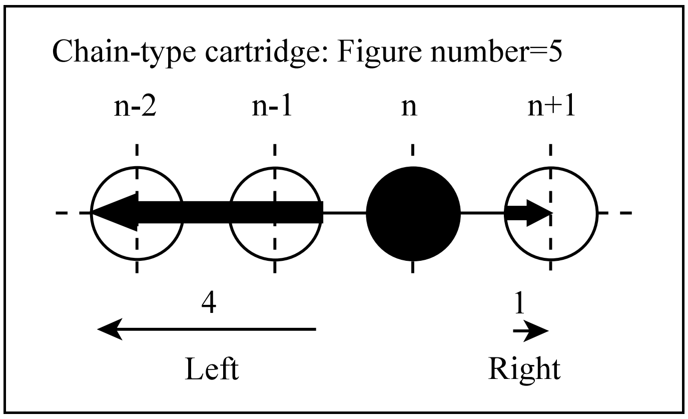
The explanation of the different directions:

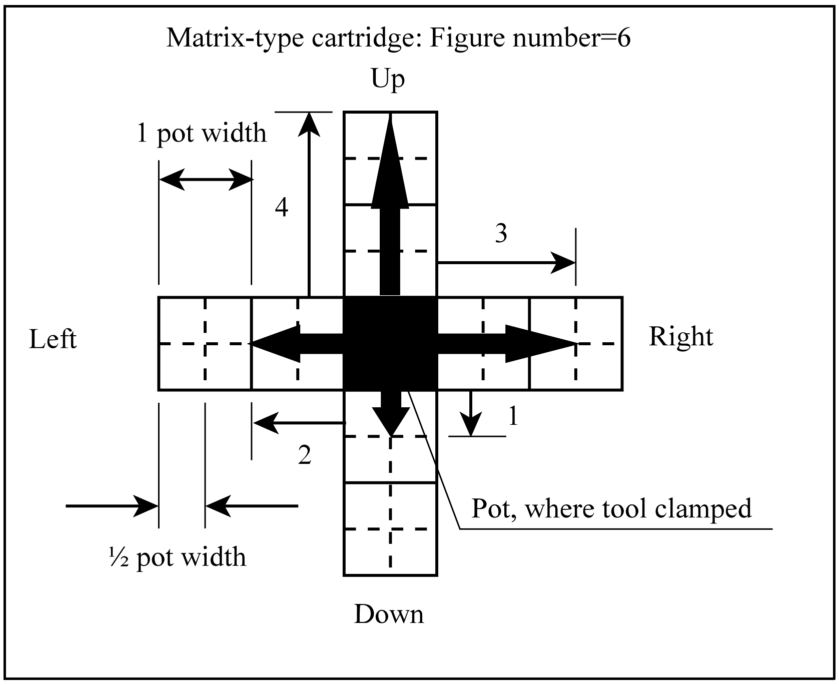
Left: decreasing pot numbers (also in chain- and matrix-type cartridges)

Right: increasing pot numbers (also in chain- and matrix-type cartridges)

Up: decreasing number of rows (only matrix-type)

Down: increasing number of rows (only matrix-type)





The spaces, occupied by tools shown in the upper figures, have to be specified in the following format:

Figure number	Left	Right	Up	Down	Geometry
...					
5	4	1	0	0	
6	2	3	4	1	

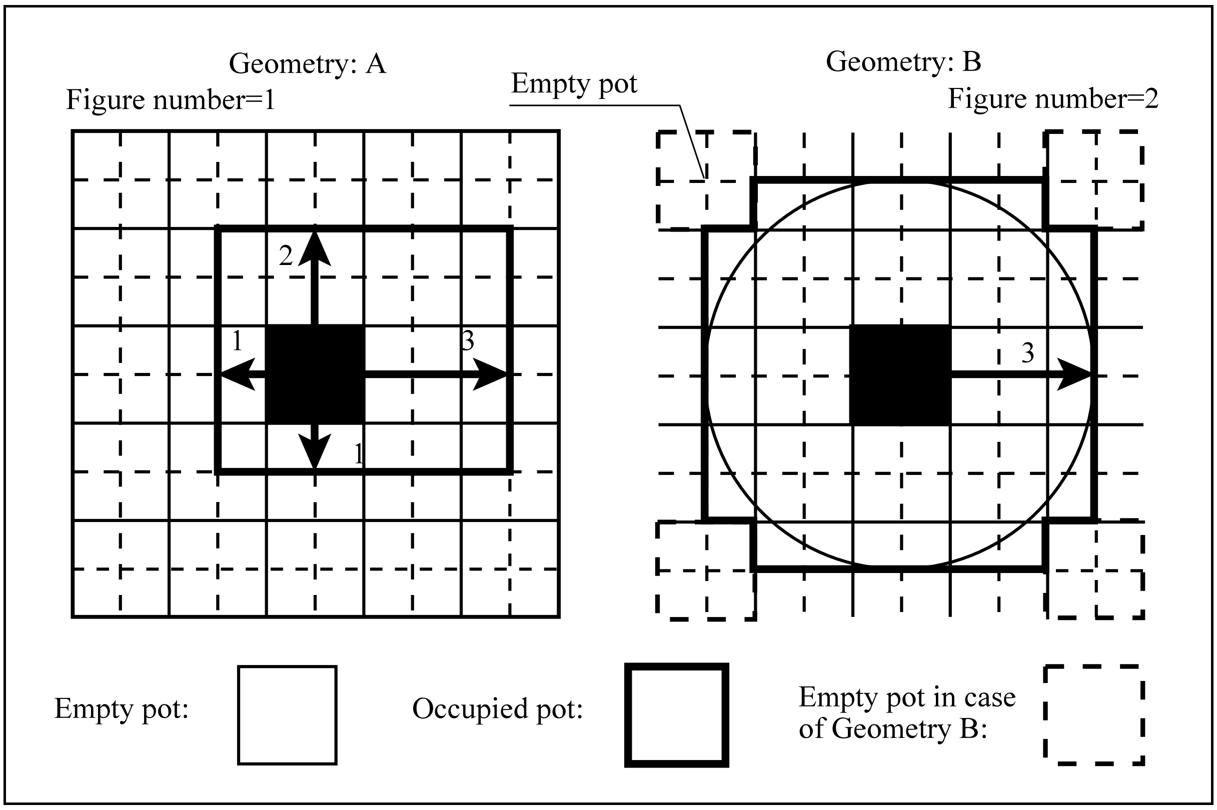
3 Geometry

The space requirement of a tool is defined in the Geometry column. Two types are available:

A: rectangular,

B: circular

shapes. When using circular shape tools and geometry is defined as B, in the corners there are empty pots, opposed to the space requirement defined as rectangle.



As the above figure shows, the Geometry column is used only in case of matrix-type cartridges. The completion of the table is based on the above figure:

Figure number	Left	Right	Up	Down	Geometry
1	1	3	2	1	A
2	3	3	3	3	B

6.26.4 Exchange of Data Numbers in Cartridge Table

Instruction: MW
Function code: 100

***Data numbers**, defined by cartridge and pot numbers, are **exchanged** in the cartridge management table.*
Example to use:

Tool is removed from the magazine and is placed into the changer arm, then
tool is removed from the changer arm and is placed into the spindle, while tool in spindle
is placed into the changer arm, then
tool is removed from the changer arm and is placed into the magazine.

Each of the above mentioned tool change operations has to be followed by a data number exchange in the cartridge management table.

Input data:

Address	Data
0	Code of execution *
1	Amount of data: 4
2	Cartridge number 1
3	Pot number 1
4	Cartridge number 2
5	Pot number 2

*: need not be set

Output data:

Address	Data
0	Code of execution: shown in table below
1	Amount of data: 4
2	Cartridge number 1
3	Pot number 1
4	Cartridge number 2
5	Pot number 2

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code
2	Invalid amount of data (not 4)
3	Overwrite protected
100	Specified Cartridge number 1 does not exist
101	Specified Pot number 1 does not exist
102	Specified Cartridge number 2 does not exist
103	Specified Pot number 2 does not exist

6.26.5 Search of Empty Pot

Instruction: MR
Function code: 101

*It searches an **empty pot** for the tool returning to the magazine according to the figure number of the tool.*
You have to specify which pot of the target cartridge has to search from. The tool is identified by the number of the source cartridge and pot. Empty pots are not searched in standby and spindle cartridges.

☞ ***Attention!** Execution of the instruction is influenced by the #0 MRW bit of the N1274 PLC PrgConfig parameter!*

Input data:

Address	Data
0	Code of execution *
1	Amount of data *
2	Number of the target cartridge
3	Number of pot in the target cartridge, where the search starts from

Address	Data
4	Search direction in table: =0: two-way =1: positive direction =2: negative direction
5	Number of the source cartridge
6	Number of the pot in the source cartridge
7	Pot number found *
8	Rotation direction of the magazine *

*: need not be set

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data: 7
2	Number of the target cartridge
3	Number of pot in the target cartridge, where the search starts from
4	Search direction in table: =0: two-way =1: positive direction =2: negative direction
5	Number of the source cartridge
6	Number of the pot in the source cartridge
7	Pot number found
8	Magazine rotation direction: =1: positive direction =2: negative direction

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code
2	Invalid amount of data (not 7)
3	Overwrite protected
100	Specified target cartridge number does not exist
101	Specified target pot number does not exist
102	Specified source cartridge number does not exist
103	Specified source pot number does not exist
104	No empty pot: Searched pot number=0

6.26.6 Register New Tool Data in Tool Management Table

Instruction: MW
Function code: 102

You can register a new tool through the PLC program by using this instruction. The instruction searches the first empty or invalid (I) data number (row) in the tool management table and the data of the new tool are registered at this data number. Each column of the table can be input.
For example: Tools are loaded into the magazine through the spindle manually.

Switch tool registration mode in the PLC.

Type number of tools, specified at the address T, is entered in a manual mode as a single block. The tool is registered with the type number specified by the T code and with the number of the spindle cartridge. Then the other data of the tool management table (figure number, life, etc.) are to be specified manually.

Input data:

Address	Data
0	Code of execution *
1	Amount of data: from 3, to 55
2	Number of cartridge
3	Number of pot in the cartridge: where the tool, to be registered, is

Address	Data
4	Type number
5	Tool info
6	Figure number
7	Life status
8	Life counter
9	Life
10	Notice life
11	H: tool length compensation number
12	D: cutter compensation number
13	G: tool geometry compensation number
14	W: tool wear compensation number
15	S: spindle speed
16	F: feed-rate (double)
17	
18	User data 1 (DWORD, 8 bit type)
19	User data 2 (double)
20	
21	User data 3 (double)
22	
...	
...	
55	User data 20 (double)
56	

*: need not be set

The minimum amount of data will be at least 3, because defining a cartridge number, a pot number and a type number are required. The instruction fills as many columns in the tool management table, as many are determined in the amount of data parameter minus 2 (cartridge number, pot number).

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data: from 3 to 55
2	Number of cartridge
3	Number of pot in the cartridge: where the tool, to be registered, is
4	Type number
5	Tool info
6	Figure number
7	Life status
8	Life counter
9	Life
10	Notice life
11	H: tool length compensation number
12	D: cutter compensation number
13	G: tool geometry compensation number
14	W: tool wear compensation number
15	S: spindle speed
16	F: feed-rate (double)
17	
18	User data 1 (DWORD, 8 bit type)
19	User data 2 (double)
21	
22	User data 3 (double)
23	
...	
...	
55	User data 20 (double)

Address	Data
56	

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code
2	Invalid amount of data: is not greater than 2, is not lower than 57 the amount of data specified cannot be input into integer number of columns
3	Overwrite protected
100	The cartridge number specified does not exist
101	The pot number specified does not exist
105	Type number error: cannot be represented in 8 decimal digits
106	Tool info error:
107	Figure number error:
108	Life status error:
109	Life counter error:
110	Life error:
111	Notice life error:
112	H definition error: The value is greater than the length of compensation table
113	D definition error: The value is greater than the length of compensation table
114	G definition error: The value is greater than the length of compensation table
115	W definition error: The value is greater than the length of compensation table
116	S: spindle speed error

Code	Explanation
117	F: feed-rate (double) error
118	User error 1
119	User error 2
120	User error 3
...	
137	User error 20
138	Life table is full

6.26.7 Writing Each Tool Management Data of a Tool

Instruction: MW
Function code: 103

Data of a tool, identified by the cartridge number and the pot number, can be written into the tool management table.

Input data:

Address	Data
0	Code of execution: shown in table *
1	Amount of data: from 3 to 55
2	Number of cartridge
3	Number of pot in the cartridge
4	Type number
5	Tool info
6	Figure number
7	Life status
8	Life counter
9	Life
10	Notice life
11	H: tool length compensation number

Address	Data
12	D: cutter compensation number
13	G: tool geometry compensation number
14	W: tool wear compensation number
15	S: spindle speed
16	F: feed-rate (double)
17	
18	User data 1 (DWORD, 8 bit type)
19	User data 2 (double)
20	
21	User data 3 (double)
22	
...	
...	
55	User data 20 (double)
56	

*: need not be set

Value of the amount of data will be at least 3, because defining a cartridge number and a pot number are required and at least type number is to be written. The instruction fills as many columns in the tool management table, as many are determined in the amount of data parameter minus 2 (cartridge number, pot number).

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data: from 3 to 55
2	Number of cartridge
3	Number of pot in the cartridge
4	Type number

Address	Data
5	Tool info
6	Figure number
7	Life status
8	Life counter
9	Life
10	Notice life
11	H: tool length compensation number
12	D: cutter compensation number
13	G: tool geometry compensation number
14	W: number of wear compensation pocket in milling channel
15	S: spindle speed
16	F: feed-rate (double)
17	
18	User data 1 (DWORD, 8 bit type)
19	User data 2 (double)
21	
22	User data 3 (double)
23	
...	
...	
55	User data 20 (double)
56	

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code

Code	Explanation
2	Invalid amount of data: does not greater than 2, is not lower than 57 the amount of data specified cannot be input into integer number of columns
3	Overwrite protected
100	The cartridge number specified does not exist
101	The pot number specified does not exist
105	Type number error: cannot be represented in 8 decimal digits
106	Tool info error:
107	Figure number error:
108	Life status error:
109	Life counter error:
110	Life error:
111	Notice life error:
112	H definition error: The value is greater than the length of compensation table
113	D definition error: The value is greater than the length of compensation table
114	G definition error: The value is greater than the length of compensation table
115	W definition error: The value is greater than the length of compensation table
116	S: spindle speed
117	F: feed-rate (double)
118	User error 1
119	User error 2
120	User error 3
...	
137	User error 20

Code	**Explanation**
139	There is no tool in the pot of the cartridge specified

6.26.8 Reading Each Tool Management Data of a Tool

Instruction: MR
Function code: 104

Data of a tool, identified by the cartridge number and the pot number, can be read from the tool management table.

☞ ***Attention!*** *Execution of the instruction is influenced by the #0 MRW bit of the N1274 PLC PrgConfig parameter!*

Input data:

Address	Data
0	Code of execution *
1	Amount of data: from 3 to 55
2	Number of cartridge
3	Number of pot in the cartridge
4	Type number *
5	Tool info *
6	Figure number *
7	Life status *
8	Life counter *
9	Life *
10	Notice life *
11	H: tool length compensation number *
12	D: cutter compensation number *
13	G: tool geometry compensation number *
14	W: tool wear compensation number *
15	S: spindle speed *
16	F: feed-rate (double) *
17	
18	User data 1 (DWORD, 8 bit type) *
19	User data 2 (double) *

Address	Data
20	
21	User data 3 (double) *
22	
...	
...	
55	User data 20 (double) *
56	

*: need not be set

Value of the amount of data will be at least 3, because defining a cartridge number and a pot number are required and at least type number is to be read. The instruction reads as many columns from the tool management table, as many are determined in the amount of data parameter minus 2 (cartridge number, pot number).

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data: from 3 to 55
2	Number of cartridge
3	Number of pot in the cartridge: where tool has stored
4	Type number
5	Tool info
6	Figure number
7	Life status
8	Life counter
9	Life
10	Notice life
11	H: tool length compensation number

Address	Data
12	D: cutter compensation number
13	G: tool geometry compensation number
14	W: tool wear compensation number
15	S: spindle speed
16	F: feed-rate (double)
17	
18	User data 1 (DWORD, 8 bit type)
19	User data 2 (double)
20	
21	User data 3 (double)
22	
...	
...	
55	User data 20 (double)
56	

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data: is not greater than 2, is not lower than 55 to the amount of data specified cannot be read from integer number of columns
100	The cartridge number specified does not exist
101	The pot number specified does not exist
139	There is no tool in the pot of determined cartridge

6.26.9 Deletion of All Tool Management Data of a Tool

Instruction: MW
Function code: 105

This function deletes all data of the tool, identified by cartridge number and pot number, in the tool management table.

Input data:

Address	Data
0	Code of execution *
1	Amount of data: 2
2	Number of cartridge
3	Number of pot in the cartridge

*: need not be set

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data: 2
2	Number of cartridge
3	Number of pot in the cartridge

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code
2	Invalid amount of data (is not 2)
3	Overwrite protected
100	The cartridge number specified does not exist
101	The pot number specified does not exist

Code	Explanation
139	There is no tool in the pot of cartridge specified

6.26.10 Writing a Tool Management Data of a Tool

Instruction: MW
Function code: 106

This function writes a single data of a tool identified by cartridge number and pot number in the tool management table.

Input data:

Address	Data
0	Code of execution *
1	Amount of data: from 4 or 5
2	Number of cartridge
3	Number of pot in the cartridge
4	Data number: shown in data number table below
4	Data word 1
5	Data word 2: in case of floating-point data is to be written only

*: need not be set

Amount of data has to be 4 in case of an integer (DWORD) data and 5 in case of floating-point data.

Explanation of data numbers in the tool management table:

Data number	Explanation
1	Type number
2	Tool info
3	Figure number
4	Life status
5	Life counter

Data number	Explanation
6	Life
7	Notice life
8	H: tool length compensation number
9	D: cutter compensation number
10	G: tool geometry compensation number
11	W: tool wear compensation number
12	S: spindle speed
13	F: feed-rate (double)
14	User data 1 (DWORD, 8 bit type)
15	User data 2 (double)
16	User data 3 (double)
...	
33	User data 20 (double)

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data
2	Number of cartridge
3	Number of pot in the cartridge
4	Data number: shown in data number table
5	Data word 1
6	Data word 2: in case of floating-point data is to be written only

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code
2	Invalid Amount of data: is not 4 or 5
3	Overwrite protected
100	The cartridge number specified does not exist
101	The pot number specified does not exist
105	Type number error: cannot be represented in 8 decimal digits
106	Tool info error:
107	Figure number error:
108	Life status error:
109	Life counter error:
110	Life error:
111	Notice life error:
112	H definition error: The value is greater than the length of compensation table
113	D definition error: The value is greater than the length of compensation table
114	G definition error: The value is greater than the length of compensation table
115	W definition error: The value is greater than the length of compensation table
116	S: spindle speed
117	F: feed-rate
118	User 1 error
119	User 2 error
120	User 3 error
...	
137	User 20 error

Code	Explanation
139	There is no tool in the pot of the cartridge specified
140	Invalid data number

6.26.11 Reading a Tool Management Data of a Tool

Instruction: MR
Function code: 107

This function reads a single data of a tool identified by cartridge number and pot number from the tool management table.

☞ ***Attention!*** *Execution of the instruction is influenced by the #0 MRW bit of the N1274 PLC PrgConfig parameter!*

Input data:

Address	Data
0	Code of execution *
1	Amount of data: from 4 to 5
2	Number of cartridge
3	Number of pot in the cartridge
4	Data number: shown in data number table
5	Data word 1 *
6	Data word 2: in case of floating-point data is to be read only *

*: need not be set

Amount of data has to be 4 in case of an integer (DWORD) data and 5 in case of floating-point data.

Explanation of data numbers in the tool management table:

Data number	Explanation
1	Type number
2	Tool info

Data number	Explanation
3	Figure number
4	Life status
5	Life counter
6	Life
7	Notice life
8	H: tool length compensation number
9	D: cutter compensation number
10	G: tool geometry compensation number
11	W: tool wear compensation number
12	S: spindle speed
13	F: feed-rate
14	User data 1 (DWORD, 8 bit type)
15	User data 2 (double)
16	User data 3 (double)
...	
33	User data 20 (double)

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data
2	Number of cartridge
3	Number of pot in the cartridge
4	Data number: shown in data number table
5	Data word 1
6	Data word 2: in case of floating-point data is to be read only

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code
2	Invalid amount of data: is not 4 or 5 does not meet the amount of data of variable referred in data number
100	The cartridge number specified does not exist
101	The pot number specified does not exist
139	There is no tool in the pot of the cartridge specified
140	Invalid data number

6.26.12 Searching a Tool by User Data

Instruction: MR
Function code: 108

This function searches a tool by user data in the tool management table, and returns the cartridge and the pot numbers. It will find the first tool whose user data corresponds the data specified in the input data.

Input data:

Address	Data
0	Code of execution *
1	Amount of data: 4 or 5
2	Number of cartridge *
3	Number of pot in the cartridge *
4	Data number: number of user data, shown in data number table to be compared
5	Data word 1: to be compared
6	Data word 2: in case of floating-point data is to be compared only

*: need not be set

If User data 1 is searched, 1 word, for further user data (floating-point) 2 words of data is to be specified.

The comparison is executed according to the increment system, set by parameter. If ISB=1, so 0.001 mm increment system is selected. Then it considers two data to be equal to if the absolute value of their difference is lower than 0.001.

Explanation of data numbers in the tool management table:

Data number	Explanation
14	User data 1 (DWORD, 8 bit type)
15	User data 2 (double)
16	User data 3 (double)
...	
33	User data 20 (double)

Output data:

Address	Data
0	Code of execution: shown in table
1	Amount of data: 4 or 5
2	Number of cartridge
3	Number of pot in the cartridge
4	Data number
5	Data word 1
6	Data word 2

Codes of execution:

Code	Explanation
0	Normal execution
1	Invalid function code

Code	Explanation
2	Invalid amount of data: is not 4 or 5
140	Invalid data number
141	No match found

6.27 Writing and Reading the Data of the Pallet Management Table

In the control, the data of the pallet management are stored in the pallet management table. Using the MR and MW instructions, the PLC program has access to the data of the table, it can also read and write them. Application of the pallet management table can be enabled at the bit #0 PAL of the parameter N3300 Pallet Contr. by writing 1 in the bit.
The pallet management table is global, it is common for each channel.
The last two rows of the pallet management table is as follows provided that the M60 is selected by the PLC for pallet change:

```
Part program
...
...
M60 (pallet change)
M30
```

In the M60 code, the PLC executes the M30 instruction, looks up the next pallet from the table and selects the program assigned to the pallet for execution.

6.27.1 The Pallet Management Table

The pallet manager table stores the data of the pallets in the pallet magazine, at the working space and at the loading-unloading point, as well as of those pallets deleted from the above locations.
Pallet magazine number 1 is the name of the storage place where the pallets having workpieces ready to be machined are located. The number of seats in the magazine can be specified in the parameter N3301 Pallet Pool Length.
Pallet magazine number 10 is the name of the working space of the machine tool where machining occurs. In the case of pallet change due to M60 function for example, the machined workpiece comes back to the magazine 1, the new workpiece comes into the magazone 10, i,e, into the working space.
In front of the pallet magazine 1 there can be a loading-unloading point named **magazine 11**. If there is loading-unloading point on the machine, the #1 MNT bit of the parameter N3300 Pallet Contr. will have to be set in 1. From the loading-unloading point, i.e. from the magazine 11, the pallet having the loaded unmachined workpiece can be sent into the magazine 1, and the pallet having finished machined workpiece can be carried out into the loading-unloading point.
Pallet magazine number 0 or **virtual magazine** is the name of those rows of the table behind which there is no physical storage place; it stores only the the data of the pallets on the basis of their identification numbers. If a pallet is removed from the loading-unloading point, the data of the pallet can be transferred to the magazine 0 by the use of PLC instruction. If a new pallet comes to the loading-unloading point, the data of the pallet can be called from the virtual magazine 0 by the use of PLC instruction (for example, due to RFID reading) provided that the pallet with this identification number already participated in machining earlier and its data have been stored in the virtual magazine 0.

Columns 1-5 of the pallet management table:

Magazine numbers [] and Place numbers	**Data numbers**					
	1	**2**	**3**	**4**	**5**	**..**
	Pallet identifier	**Status**	**Priority**	**Number of executed programs**	**Number of assigned programs**	**..**
Working space [10] 1						
Loading-unloading point [11] 1						
[1] 1						
[1] 2						
...						
...						
[1] n						
[0] 1						
[0] 2						
...						

The column of Magazine numbers and Place numbers:

The row of Working space[10] 1: It contains the data of the pallet being in the working space. It cannaot be edited in the automatic mode. As for the PLC, it is the magazine 10.

The row of loading-unloading point [11] 1: It contains the data of the pallet being in the loading-unloading point. It can be seen only in the case if the value of the #2 MNT bit of the parameter N3300 Pallet Contr. is 1. It can only be edited when it is not forbidden by the PLC because, for example, a pallet change between the loading-unloading point and the magazine has already started. As for the PLC, it is the magazine 11.

Pallet magazine [1] place number rows 1, 2, ... n: It contains the data of the pallets being in the pallet magazine. It contains as many rows as many have been specified in the parameter N3301 Pallet Pool Length. It can only be edited if it is not forbidden by the PLC because, for example, a pallet change between the working space and the magazine has already started.

Virtual magazine [0] place number rows 1, 2, ...: The pallet magazine 0, i.e. the virtual magazine contains the data of the pallets deleted from the physical magazine. The number of the rows available in the magazine 0 is 128 minus the rows of the magazine 1, 10 and 11. It can only be edited if it is not forbidden by the PLC because, for example, a pallet change between the working space and the magazine has already started.

The column 1 of the pallet identifier:
Data number=1, DWORD data. It contains arbitrary identification number of the pallets. It can be written by the operator manually, or by the PLC too, for example, after read-in of the RFID code. If, in the magazine 0, the pallet manager finds a pallet identifier of which is written/read in, it will automatically transmit the data of the appropriate pallet to the appropriate (physical) row of the table.
The PLC instruction MW41 assigns the program(s) to execute based on pallet identifier of the working space.

The column 2 of the status:
Data number=2, DWORD value. It contains the status of the pallet place. It can be written by the operator manually, or by the PLC program too.
Its value=0: ***It cannot be used.*** It is not possible to use the pallet place due to mechanical trouble for example. It is not allowed to place a pallet there.
Its value=1: ***It is empty.*** At the given station, there is neither pallet nor pallet support plate. This is the case when the pallets in the magazine 1 are stored on a support plate and the support plate can also be exchanged together with the pallet, for example, between the magazine and the loading-unloading point.
Its value=2: ***Only support plate***. At the given station there is no pallet but there is pallet support plate. This is the case when a pallet is exchanged to the working space but there is no need for the support plate.
Its value=3: ***It is under loading-unloading***. The pallet is not enabled for exchange yet. When, for example, the pallet arrives to the magazine 1 from the loading-unloading point, it can take the status 3. It will physically be one of the exchangeable pallets, however it cannot be exchanged into the working space until it is enabled for exchange.
Its value=4: ***It is ready for exchange***. The pallett is prepared and enabled. Előkészített és érvényesített paletta. The pallet is ready to be exchanged for machining.
Its value=5: ***It is under machining***. Machining is in progress, the part program runs.
Its value=6: ***It is finished***. There is properly machined workpiece on the pallet.
Its value=7: ***It is faulty***. There is machined workpiece on the pallet but is is faulty.
Its value=8: ***Remachining***. The pallet has been sent back to the working space for correction.

The column 3 of priority:
Data number=3, DWORD-type data. The smaller the priority value is, the sooner the workpiece(s) on the pallet will be machined. It can be written by the operator manually, or by the PLC program too.
If several pallets have the same priority, the closest pallet to the working space will be exchanged, i.e. the position of this pallet will be provided by the MR308 search instruction.

The column 4 of the number of executed programs:
Data number=4, DWORD-type data. During execution of the M60 instruction, the PLC program increments the number of executed programs in the working space, i.e. in the magazine 10.
As long as the MW41 program selection instruction returns with 0, not all the programs are executed on the palette. In this case the PLC does not search for a new pallet in the magazine, but it starts the next program.
If the MW41 program selection instruction returns with 47, a new pallet should be searched.

The column 5 of the number of assigned programs:
Data number=5, DWORD-type data. Several main programs can be assigned to a pallet for execution. The number of assigned programs can be found in this column.
As long as the MW41 program selection instruction returns with 0, not all the programs are executed on the palette. If the number of the programs executed is equal to the number of the programs assigned, the instruction will return with the code 47.

The columns from 6 to 10:

Magazine numbers [] and Place numbers	**Data numbers**					
	..	**6**	**7**	**8**	**9**	**10**
		Program identifier	**Custom 1**	**Custom 2**	**Custom 3**	**Custom 4**
Working space [10] 1						
Loading-unloading point [11] 1						
[1] 1						
[1] 2						
...						
...						

[1] n						
[0] 1						
[0] 2						
...						

The column 6 of Program identifier:
Data number=6
If the N3300 Pallet Contr. parameter’s #3 PRI=0:

The assigned programs will be identified by 4-digit or 8-digit program numbers. The programs should be in the root of the Programs directory with the file name Ooooo.nct or Ooooooooo.nct. In this case, the variable belonging to the data number will be DWORD-type, and the PLC program can write and read this column.

If the N3300 Pallet Contr. parameter’s #3 PRI=1:

The program (s) to be executed are selected manually based on their filename and path by the use of the palette manager table. In this case, the PLC program cannot write and read this column.

In the parameter N3302 No. of Custom Columns, maximum 4 custom columns can be assigned.

The column 7 of Custom 1:
Data number=7, it is bit-data, its length is 8 bits.
It can be used if the N3302 No. of Custom Columns parameter>0.

The column 8-10 of Custom 2-4:
Data number=8-10, double-type data.
It can be used if N3302 No. of Custom Columns parameter>1.

6.27.2 Data Interchange Between Two Different Places of Two Different Pallet Magazines

Instruction: MW
Function code: 300

In the pallet management table, the data of two rows specified by the magazine number and the place number will be interchanged.

Input parameters:

Address	Data
0	Code of execution*
1	Data length: 3
2	Number of the pallet magazine 1 (1, 10, 11)
3	Number of the place 1
4	Number of the pallet magazine 2 (0, 10 or 11)

*: This data should not be entered.

If the data interchange should be done between the pallet magazine and the working space,

2	the number of the pallet magazine 1=1
3	the place number of the magazine 1= the number of the adequate pocket
4	the number of the pallet magazine 2=10

If the data interchange should be done between the pallet magazine and the loading-unloading point,

2	the number of the pallet magazine 1=1
3	the place number of the magazine 1= the number of the adequate pocket
4	the number of the pallet magazine 2=11

If the data interchange should be done between the loading-unloading point and the virtual magazine,

2	the number of the pallet magazine 1=11
3	the place number of the magazine 1= 1
4	the number of the pallet magazine 2=0

If the data interchange should be done between the pallet magazine and the virtual magazine,

2	the number of the pallet magazine 1=1
3	the place number of the magazine 1= the number of the adequate pocket
4	the number of the pallet magazine 2=0

Output parameters:

Address	Data
0	Code of execution: See the table
1	Data length: 3
2	Number of the pallet magazine 1
3	Number of the place 1
4	Number of the pallet magazine 2

Possible codes of execution:

Code	Description
0	Normal execution
1	Invalid Function code
2	Invalid data length (it is not 3)
3	Write-protected
300	The given number of the pallet magazine 1 does not exist
301	The given pallet place does not exist
302	The given number of the pallet magazine 2 does not exist

6.27.3 Rewriting the Pallet Data

Instruction: MW
Function code: 303

In the pallet management table, rewriting the data of the pallet identified in accordance with the magazine number and the place number.

Input parameters:

Address	Data
0	Code of execution*
1	Data length: minimum 3, maximum 14
2	Number of the pallet magazine
3	Place number of the pallet magazine: Where the pallet is
4	Pallet identifier
5	Status
6	Priority
7	Number of the programs executed
8	Number of the programs assigned
9	Custom 1
10	Custom 2 (double)
11	
12	Custom 3 (double)
13	
14	Custom 4 (double)
15	

*: This data should not be entered.
For data length, minimum 3 should be written because entering the data of magazin number and place number is mandatory, and at least the Pallet identifier should be written. The instruction will rewrite as many columns in the pallet management table as many columns have been specified in the Data length parameter, minus 2 (magazine number, place number). When the data of the pallet in the working space (the magazine number is 10) or at the loading-unloading point the magazine number is 11) are rewritten, the place number will be 1.

The instruction **does not manage the Program identifier**, it will be accessible only by the use of specific pallet data writing, i.e. MW306 instruction in the case if the value of the #3 PRI bit of the parameter N3300 Pallet Contr. is 0.

For rewiting the pallet identifier, the system transposes all the data from the virtual pallet table and rewrites the other possible data provided that it finds row of such identifier.

Output parameters:

Address	Data
0	Code of execution
1	Data length: minimum 3, maximum 14
2	Number of the pallet magazine
3	Place number of the pallet magazine: Where the pallet is
4	Pallet identifier
5	Status
6	Priority
7	Number of the programs executed
8	Number of the programs assigned
9	Custom 1
10	Custom 2 (double)
11	
12	Custom 3 (double)
13	
14	Custom 4 (double)
15	

Possible codes of execution:

Code	Description
0	Normal execution
1	Invalid Function code

Code	Description
2	Invalid data length: not greater than 2, or not smaller than 14
3	Write-protected
300	The given pallet magazine nimber does not exist
301	The given place number does not exist
305	Invalid pallet identifier (<1, or >99999999)
306	Invalid pallet status (>9)
307	Priority error (>128)
308	The number of the programs executed is out of bounds (it is greater than the number of the programs assigned)
309	The number of the programs is out of bounds (> 64)
339	The pallet place is empty

6.27.4 Reading the Data of the Pallet

Instruction: MR
Function code: 304

Reading the data of the pallet identified in accordance with the magazine number and the place number from the pallet management table.

Input parameters:

Address	Data
0	Code of execution*
1	Datalength: minimum 3, maximum 14
2	Number of the pallet magazine
3	Place number of the pallet magazine: Where the pallet is
4	Pallet identifier*
5	Status*
6	Priority*
7	Number of the programs executed*
8	Number of the programs assigned*
9	Custom 1*
10	Custom 2* (double)
11	
12	Custom 3* (double)
13	
14	Custom 4* (double)
15	

*: This data should not be entered.

For data length, minimum 3 should be written because entering the data of magazin number and place number is mandatory, and at least the pallet identifier should be read. The instruction will only read as many columns from the pallet management table as many columns have been specified in the Data length parameter, minus 2 (magazine number, place number). When the data of the pallet in the working space (the magazine number is 10) or at the loading-unloading point the magazine number is 11) are read, the place number will be 1.

The instruction **does not manage the Program identifier**, it will be accessible only by the use of specific pallet data reading, i.e. MW307 instruction in the case if the value of the #3 PRI bit of the parameter N3300 Pallet Contr. is 0.

Output parameters:

Address	Data
0	Code of executionTeljesítés kódja
1	Data length: minimum 3, maximum 14
2	Number of the pallet magazine
3	Place number of the pallet magazine: Where the pallet is
4	Pallet identifier
5	Status
6	Priority
7	Number of the programs executed
8	Number of the programs assigned
9	Custom 1
10	Custom 2 (double)
11	
12	Custom 3 (double)
13	
14	Custom 4 (double)
15	

Possible codes of execution:

Code	Description
0	Normal execution
1	Invalid Function code
2	Invalid data length: not greater than, or not smaller than 14
3	Write-protected

Code	Description
300	The given pallet magazine number does not exist
301	The given place number does not exist
339	The pallet place is empty

6.27.5 Rewriting a Pallet Management Data of the Pallet

Instruction: MW
Function code: 306

Rewriting a given data of the pallet identified in accordance with the pallet magazine number and the place number in the pallet management table.

Input parameters:

Address	Data
0	Code of executionTeljesítés kódja *
1	Datalength: 4 or 5
2	Number of the pallet magazine
3	Place number of the pallet magazine
4	Data number: See the Data number table
5	Data (if the data is double, 2 places should be reserved)
6	

*: This data should not be entered.

When the data of the pallet in the working space (the magazine number is 10) or at the loading-unloading point the magazine number is 11) are rewritten, the place number will be 1.

For rewriting the pallet identifier, the system transposes all the data from the table of the selectable pallets and overwrites other possible data.

It will only be allowed to write the Program identifier (data number=6) if the value of the #3 PRI of the parameter N3300 Pallet Contr. is 0.
With writing, the previous assignments will be overwritten.
Several programs can be assigned at once. In this case, the data of Number of assigned programs should be given first. Then, the data length parameter of the MW306 instruction should be increased depending on the program number to be entered. The program numbers can be uploaded by the use of the MW306 instruction.

Interpretation of the data numbers in the pallet management table is as follows:

Data number	Description
1	Pallet identifier
2	Status
3	Priority
4	Number of the programs executed
5	Number of the programs assigned
6	Program identifier
7	Custom 1
8	Custom 2
9	Custom 3
10	Custom 4

Output parameters:

Address	Data
0	Code of execution: See the table
1	Data length: 4 or 5
2	Number of the pallet magazine
3	Péace number of the pallet magazine
4	Data number: See the Data number table
5	Data (if the data is double, 2 places should be reserved)
6	

Possible codes of execution:

Code	Description
0	Normal execution
1	Invalid Function code
2	Invalid data length: not 4 or 5
3	Write-protected
300	The given pallet magazine number does not exist

Code	Description
301	The given place number does not exist
305	Invalid pallet identifier (<1 or >99999999)
306	Invalid pallet status (>10)
307	Priority error (>128)
308	The number of the programs executed is out of bounds (it is greater than the number of the programs assigned)
309	Program number error (>64)
339	The pallet place is empty

6.27.6 Reading a Pallet Management Data of the Pallet

Instruction: MR
Function code: 307

Reading a given data of the pallet identified in accordance with the pallet magazine number and the place number from the pallet management table.

Input parameters:

Address	Data
0	Code of execution*
1	Data length: 4 or 5
2	Number of the pallet magazine
3	Place number of the pallet magazine
4	Data number: See the Data number table
5	Data (if the data is double, 2 places should be reserved)*
6	

*: This data should not be entered.

When the data of the pallet in the working space (the magazine number is 10) or at the loading-unloading point the magazine number is 11) are read, the place number will be 1.

It will only be allowed to read the Program identifier (data number=6) if the value of the #3 PRI of the parameter N3300 Pallet Contr. is 0.

The data of Number of assigned programs should be read first. Then, the data length parameter of the MW307 instruction should be increased depending on the program number to be entered. The program numbers can be uploaded by the use of the MW307 instruction.

Interpretation of the data numbers in the pallet management table is as follows:

Data number	Description
1	Pallet identifier
2	Status
3	Priority
4	Number of the programs executed
5	Number of the programs assigned
6	Program identifier
7	Custom 1
8	Custom 2
9	Custom 3
10	Custom 4

Output parameters:

Address	Data
0	Code of execution: See the table
1	Data length: 4 or 5
2	Number of the pallet magazine
3	Place number of the pallet magazine
4	Data number: See the Data number table
5	Data (if the data is double, 2 places should be reserved)
6	

Possible codes of execution:

Code	Description
0	Normal execution
1	Invalid Function code

Code	Description
2	Invalid data length: not 4 or 5
3	Write-protected
300	The given pallet magazine number does not exist
301	The given place number does not exist
339	The pallet place is empty

6.27.7 Searching a pallet by its data number value

Instruction: MR
Function code: 308

In the pallet magazine 1, the instruction looks up the place of that pallet the data number and the value of which is specified in input parameter, and the found place number will be given back.
If a ready-to exchange pallet place of status 4 is searched upon the data number 2 and there are several pallets with the same status in the magazine, the place of that pallet will be given back which has the highest priority (the lowest priority number). If there are several pallets with the same priority in the magazine, the place of that pallet will be given back which is the nearmost one to the given initial position.
If a non-status 4 pallet place is searched upon the data number 2 or a pallet of a given value is searched upon another data number, the place of that pallet will be given back which is the nearmost one to the given initial position.
Depending on the mechanical design of the pallet magazine, i.e. it can only be rotated in one or both directions, the direction of the search can also be specified as an input parameter. Accordingly, the instruction gives back the direction of the magaine rotation.

Inpur parameters:

Address	Data
0	Code of execution *
1	Data length: 6
2	Data number
3	Value to be searched
4	Place number of the pallet magazine: the position from which search should be done
5	The direction of the search in the table: =0: bidirectional =1: in positive direction =2: in negative direction
6	The found place number *
7	The direction of the magazine rotation*

*: This data should not be entered.

Output parameters:

Address	Data
0	Code of execution: See the table
1	Data length: 6
2	Data number
3	Value to be searched
4	Place number of the pallet magazine
5	The direction of the search in the table
6	The found place number
7	Magazine rotation =1: in positiev direction =2: in negative direction

Possible codes of execution:

Code	Description
0	Normal execution
1	Invalid Function code
2	Invalid data lengtgh: it is not 2
12	Data number error
301	The given place number does not exist
339	The pallet place is empty
341	In the magazine there is no the pallet with the status searched

6.27.8 Reading out the pallet magazine's values of given data number

Instruction: MR
Function code: 309

The values of any column of a magazine can be read out from the pallet management table by the use of this instruction. It can be specified which magazine position (which row of the table) to start reding from and how many rows to read. If the reading reaches the end of the magazine and the specified number of items has not been exhausted, reading will continued from the beginning of the table.

Input parameters:

Address	Data
0	Code of execution*
1	Data length: n ≥4
2	Data number: See the Data number table
3	Number of the pallet magazine
4	Place number of the pallet magazine
5	Value at the given place number*
...	...*
4+n	...*

*: This data should not be entered.

Output parameters:

Address	Data
0	Code of execution
1	Data length: n ≥4
2	Data number: See the Data number table
3	Number of the pallet magazine
4	Place number of the pallet magazine
5	Value at the given place number
...	...
4+n	...

Example:
Let the pallet magazine 1 be a 5-seat one and let's read the data beginning from the place number 4. Let's read the first column of the table, that is, the pallet identification codes. The result after reading will be as follows:

0: 0 (perfect operation)
1: Data length: 10
2: Data number 1 (pallet identifier)
3: Number of the pallet magazine:1
4: Place number of the pallet magazine: 4
5: Read out identifier: 12345679 (place 4)
6: Read out identifier: 98764 (place 5)
7: Read out identifier: 123 (place 1)
8: Read out identifier: 456 (place 2)
9: Read out identifier: 789 (place 3)
10: Read out identifier: 12345679 (place 4)
11: Read out identifier: 98764 (place 5)

The code of execution can be the following:

Code	Description
0	Normal execution
1	Invalid Function code
2	Invalid data length: it is not n ≥4
12	Data number error
300	The given pallet magazine number does not exist

Code	Description
301	The given place number does not exist
339	The pallet place is empty (internal error)

6.27.9 Assigning a program accessible by its pallet identifier for automatic execution

Instruction: MW
Function code: 41

The write operation will be executed only in the case, when there is no program running in automatic mode and there is no interrupt status in the given channel, i.e. the following conditions are true:

CN_START=CN_STOP=CN_INTD=0.

It is always necessary to give the identifier of that pallet which is in the working space (in the magazine 10).
If several programs are assigned in the pallet, the next program will be assigned based on the Number of the programs executed in the table.

Input parameters:

Address	Data
0	Code of execution *
1	Data length: 2
2	The identifier of the pallet in the magazine 10
3	Channel number: 1...8

*: This data should not be entered.

It is always 2 that should be written for data length.

Output parameters:

Address	Data
0	Code of execution
1	Data length: 2
2	Identifier of the pallet
3	Channel number: 1...8

Possible codes of execution:

Code	Description
0	Normal execution
1	Invalid Function code
2	Invalid data length: it is not 2
41	Invalid channel number
45	The file does not exist
46	A program is being run in the given channel
47	There is no executable program on the pallet
342	The working space is empty (there is no pallet there)

In the case when several programs are assigned on the pallets for execution and the Code of execution of the MW41 instruction is 0, not all the programs are executed yet and the next program can be started in the M60 pallet change function.

In the case when the Code of execution of the MW41 instruction is 47, all the programs are executed on the pallet and the M60 function should search a new and ready-to-exchange pallet. Then, the M41 program assignment instruction should be applied to the new pallet.

6.28 Mailbox Communication between the PLC program and an Arbitrary Ethercat Device

EtherCAT mailboxes of a device, manufacturer of which is not the NCT and which uses known protocol, can be accessed from the PLC program by the use of instructions MR201 and MW202. The protocols used can be the following:

SoE: Sercos over EtherCAT,
CoE: CANopen over EtherCAT,
VoE: Vendor over EtherCAT (protocol defined by the manufacturer).

The use of mailbox communication instructions may be necessary in the following cases:

SoE, CoE drives: error deletion, reading and displaying the current data,
SoE, CoE drives: reading and displaying current and speed data.

6.28.1 Reading the Data of the EtherCAT Mailbox

Instruction: MR
Function code: 201

Input parameters:

The case of the VoE protocol:

Address	Data
0	Code of execution *
1	Amount of data: 4 (VoE)
2	Protocol code: 15
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to

*: need not be set

The case of the CoE protocol, when CompleteAccess is missing and the error code is indifferent:

Address	Data
0	Code of execution*
1	Amount of data: 5 (CoE)
2	Protocol code: 3

Address	Data
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)

*: need not be set

The case of the CoE protocol, when CompleteAccess is missing:

Address	Data
0	Code of execution*
1	Amount of data: 6 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device*

*: need not be set

The case of the SoE protocol:

Address	Data
0	Code of execution*
1	Amount of data: 7 (SoE)
2	Protocol code: 5
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte

Address	Data
5	The PLC memory address where the data is waited to
6	IDN: 2 bytes (0-15th bit): element identifier
	Drive Index: 1 byte (16-23th bit)
7	Error code given back by the device*
8	ElementFlags **

*: need not be set

**** ElementFlags**

Data of the element selected on the IDN, which are set in the ElementFlags bits can be inquired. All the data might as well inquired.
Interpretation of the ElementFlags data bit by bit is as follows:

bit	Name
0	DataState (parameter state)
1	Name (ASCII-coded name)
2	Attribute (attribute)
3	Unit (unit of measurement)
4	Min (minimum value)
5	Max (maximum value)
6	Value (actual value)
7	DefaultValue (reference value)

The case of the CoE protocol:

Address	Data
0	Code of execution*
1	Amount of data: 7 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to

Address	Data
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device*
8	CompleteAccess (it can be 0/1)**

*: need not be set

**** CompleteAccess:**
– If CompleteAccess=0, the object specified in the Index and SubIndex fields will be downloaded.
– If CompleteAccess=1 and SubIndex=0 or 1, all the objects specified in the Index field will be downloaded.

Output parameters:

The case of the VoE protocol:

Address	Data
0	Code of execution
1	Amount of data: 4 (VoE)
2	Protocol code: 15
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to

The case of the CoE protocol, when CompleteAccess is missing and the error code is indifferent:

Address	Data
0	Code of execution
1	Amount of data: 5 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window

Address	Data
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)

The case of the CoE protocol, when CompleteAccess is missing:

Address	Data
0	Code of execution
1	Amount of data: 6 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device

The case of the SoE protocol:

Address	Data
0	Code of execution
1	Amount of data: 7 (SoE)
2	Protocol code: 5
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	IDN: 2 bytes (0-15th bit): element identifier
	Drive Index: 1 byte (16-23th bit)
7	Error code given back by the device

Address	Data
8	ElementFlags

The case of the CoE protocol:

Address	Data
0	Code of execution
1	Amount of data: 7 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device
8	CompleteAccess (it can be 0/1)

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid amount of data: nem 3
200	There is no EtherCAT device with specified index
201	SOE or COE communication is not supported by the device with specified index
202	The PLC memory address or the amount of data of the mailbox is incorrect
203	Timeout reading
204	
205	An error was sent by the device: the error code sent back by the device can be read from the field 7, which is the field of the error code given back by the device

6.28.2 Writing the Data of the EtherCAT Mailbox

Instruction: MW
Function code: 202

Input parameters:

The case of the VoE protocol:

Address	Data
0	Code of execution*
1	Amount of data: 4 (VoE)
2	Protocol code: 15
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to

*: need not be set

The case of the CoE protocol, when CompleteAccess is missing and the error code is indifferent:

Address	Data
0	Code of execution*
1	Amount of data: 5 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)

*: need not be set

The case of the CoE protocol, when CompleteAccess is missing:

Address	Data
0	Code of execution*
1	Amount of data: 6 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device*

*: need not be set

The case of the SoE protocol:

Address	Data
0	Code of execution*
1	Amount of data: 7 (SoE)
2	Protocol code: 5
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	IDN: 2 byte (0-15th bit): element identifier
	Drive Index: 1 byte (16-23th bit)
7	Error code given back by the device*
8	ElementFlags **

*: need not be set

**** ElementFlags**

Data of the element selected on the IDN, which are set in the ElementFlags bits can be written. All the data might as well written.
Interpretation of the ElementFlags data bit by bit is as follows:

bit	Name
0	DataState (parameter state)
1	Name (ASCII-coded name)
2	Attribute (attribute)
3	Unit (unit of measurement)
4	Min (minimum value)
5	Max (maximum value)
6	Value (actual value)
7	DefaultValue (reference value)

The case of the CoE protocol:

Address	Data
0	Code of execution*
1	Amount of data: 7 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device*
8	CompleteAccess (it can be 0/1) **

*: need not be set

**** CompleteAccess:**

– If CompleteAccess=0, the object specified in the Index and SubIndex fields will be written.

– If CompleteAccess=1 and SubIndex=0 or 1, all the objects specified in the Index field will be written.

Output parameters:

The case of the VoE protocol:

Address	Data
0	Code of execution
1	Amount of data: 4 (VoE)
2	Protocol code: 15
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to

The case of the CoE protocol, when CompleteAccess is missing and the error code is indifferent:

Address	Data
0	Code of execution
1	Amount of data: 5 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)

The case of the CoE protocol, when CompleteAccess is missing:

Address	Data
0	Code of execution
1	Amount of data: 6 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window

Address	Data
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device

The case of the SoE protocol:

Address	Data
0	Code of execution
1	Amount of data: 7 (SoE)
2	Protocol code: 5
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	IDN: 2 bytes (0-15th bit): element identifier
	Drive Index: 1 byte (16-23th bit)
7	Error code given back by the device
8	ElementFlags

The case of the CoE protocol:

Address	Data
0	Code of execution
1	Amount of data: 7 (CoE)
2	Protocol code: 3
3	ECAT device index: Serial number of the device that can be seen in the EtherCAT window
4	Amount of data for the device, in byte
5	The PLC memory address where the data is waited to
6	Index: 2 bytes (0-15th bit)

Address	Data
	SubIndex: 1 byte (16-23th bit)
7	Error code given back by the device
8	CompleteAccess (it can be 0/1)

The code of execution can be the following:

Code	Explanation
0	Normal execution
1	Invalid Function code
2	Invalid Amount of data: not 3
200	There is no EtherCAT device with specified index
201	SOE or COE communication is not supported by the device with specified index
202	The PLC memory address or the amount of data of the mailbox is incorrect
203	
204	Timeout writing
205	An error was sent by the device: the error code sent back by the device can be read from the field 7, which is the field of the error code given back by the device

6.29 Codes of Execution of MR, MW Instructions

Address	Explanation
0	Normal execution
1	Invalid function code
2	Invalid amount of data
3	Overwrite protected
4	Memory checksum error
6	Invalid channel number
10	Array specified in non-volatile memory not in the range between 0...1023
11	Starting address specified in the PLC memory is not higher than or equal to PLCNVRAM
12	Data number error: (address of first variable to be read)+ (number of variables to be read)>1024
20	Referring to a non-existing macro variable
21	The macro variable is not global (macro variable index = 0)
22	The macro variable is global (macro variable index > 0)
23	Incorrect writing/reading code: the format of the macro variable (DWORD, double) is not in harmony with the writing/reading code
24	The macro variable cannot be read
25	The macro variable cannot be written
30	Referring to a non-existing macro variable/Invalid axis index
31	The parameter is not global (parameter index = 0)
32	The parameter is global (parameter index > 0)

Address	Explanation
33	Incorrect writing/reading code: the format of the parameter (bit, DWORD) is not in harmony with the writing/reading code
40	Non-existing program: the specified program is not in the memory
41	Invalid channel number
45	The file does not exist
46	A program is being run in the given channel
47	There is no executable program on the pallet
60	Invalid connection identifier
61	Establishing connection is failed
62	Error in receiving data
63	The specified connection is not open
64	False data array Amount of data: =0 or >350
65	No data received
66	False incoming parameters
70	Invalid window number
71	There is no connection to the display (the display is not receive ready)
80	Invalid drive address
81	Invalid data type
82	Invalid value of the data transmitted
90	Internal error, the instruction cannot be used
91	The axis index indicates a non-existing axis, or the position control loop is not open (AN_OPNA=0), or receiving the position from the encoder is not disabled (AP_EFD=0)

Address	Explanation
100	The cartridge number specified does not exist
101	The pot number specified does not exist
102	Specified source cartridge number does not exist
103	Specified source pot number does not exist
104	No empty pot: Searched pot number=0
105	Type number error: can not represent in 8 decimal digits
106	Tool info error:
107	Figure number error:
108	Life status error:
109	Life counter error:
110	Life error:
111	Notice life error:
112	H definition error: The value is greater than the length of compensation table
113	D definition error: The value is greater than the length of compensation table
114	G definition error: The value is greater than the length of compensation table
115	W definition error: The value is greater than the length of compensation table
116	S: spindle speed
117	F: feed-rate
118	User error 1
119	User error 2
120	User error 3
...	
137	User error 20
138	Life table is full

Address	Explanation
139	There is no tool in the pot of the cartridge specified
140	Invalid data number
141	No match found
200	There is no EtherCAT device with specified index
201	SOE or COE communication is not supported by the device with specified index
202	The PLC memory address or the amount of data of the mailbox is incorrect
203	Timeout reading
204	Timeout writing
205	An error was sent by the device: the error code sent back by the device can be read from the field 7, which is the field of the error code given back by the deviceaner
300	The first given pallet magazine number does not exist
301	The given pallet place does not exist
302	The given second magazine number does not exist
305	Invalid pallet identifier (<1 or >99999999)
306	Invalid pallet status (>9)
307	Priority error (>128)
308	The number of the programs executed is out of bounds (it is greater than the number of the programs assigned)
309	The number of the programs is out of bounds (> 64)
339	The pallet place is empty
341	In the magazine there is no the pallet the status of which is searched
342	The working space is empty (there is no pallet there)

7 Communication between the PLC Program and the NC

The communication between the PLC program and the NC, respectively, with the outer world is made through the memory used by the PLC program. The memory area starts after the word containing the Status bits (FLAGS) and lasts till the address PLCNVRAM. This memory area contains the so called ***NC symbols***.
On this memory area, through the NC symbols the communication takes place between the PLC program and the
– NC peripheries, i.e. the input and output hardware units, such as the:
    machine control panel(s),
    handwheel(s),
    binary and analog interface out- and inputs,
    probes,
    EtherCAT drives,
    encoder receiver, respectively, the analog or CAN-bus servo control units,
– certain software modules of the NC, such as:
    modules of several services, e.g. function buttons, PLC parameters, etc,
    common module,
    axis controlling module,
    spindle controlling module and
    channel controlling module.

We can set the address of output and input units on the EtherCAT setting panel of the given element, on the control, and these addresses correspond to the appropriate area of the PLC memory.

From among the flags of communication with the modules of the NC software, the common flags do not become indexed, in contrast with the axis-, spindle- or channel controlling variables, which become indexed per axis, spindle or channel. We mean by this that the single reference to a symbol, refers always to the first axis, spindle or channel. The appropriate variable of the other axes, spindles or channels can be accessed by an indexed reference.

The NC symbol may be a
    bit-,
    double-word- (DWORD), or
    floating point (double)
value.
We refer to every single communication memory area in a symbolic way. In our guide we do not provide the physical address of the communication memory, i.e. the NC symbols, as it may vary during the several software versions.

Due to the above reason, in the PLC program we cannot define symbols with fix addresses from the FLAGS address to the PLCNVRAM address, but only such addresses which contain a relative reference to an NC symbol.

Example 1:
Symbol MB_JOG1 refers to the upper left button in the matrix of jog buttons. On the factual machine this button moves axis X in a negative direction.
In case we would like to refer to the button by a B_JOG_XN symbol, we have to declare this symbol in the PLC Editor by a reference to the MB_JOG1 symbol as the basis, with a zero offset – and no way shall we enter the numeric address of the MB_JOG1 symbol on the B_JOG_XN symbol.

Example 2:
We would like to refer to bit nr. 17 of the DWORD INP000 by a MGZ_RPT symbol (magazine on the refpoint switch).
We shall declare the MGZ_RPT symbol by a zero-offset to the INP000 symbol as the basis and we shall enter 17 on the bit number.

7.1 NCT Machine Control Panels

The system receives the signals of the machine control panel installable under

MK19: 19-inch

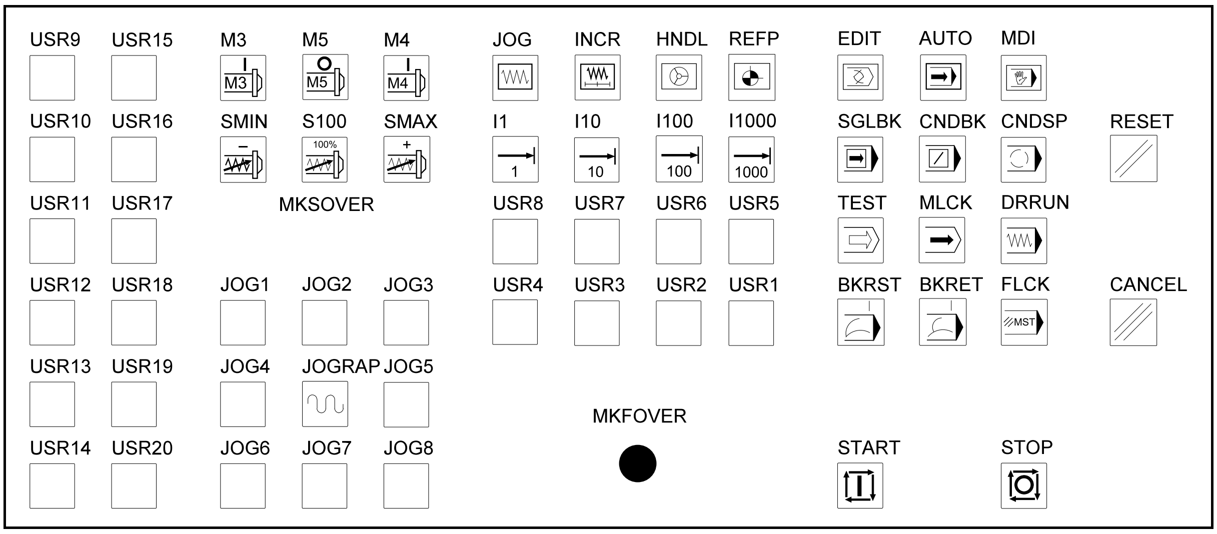
MK15: 15-inch

monitor through the EtherCAT network.

The button layout of the machine control panel can be seen in the below figure. A lamp belongs to each button. Above every button we have indicated the common text part of symbol of the button and the lamp. Freely usable button inputs and lamp outputs belong to the control panel, too. These may be connected to any place, the figure does not contain them.

The MKSOVER under the spindle override buttons, respectively the MKFOVER above the feedrate override switch is the symbol of the handover register of the corresponding override value.

☞ ***Attention!*** *Control panel type MK15 does not contain buttons between the range from USR9 to USR20!*



The following symbols of the machine control panel buttons and lamps are all symbols with ***bit*** reference.

Bit variables of the control panel:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
MB_START	Start button	**ML_START**	Start lamp
MB_STOP	Stop button	**ML_STOP**	Stop lamp
MB_FLCK	Function lock button	**ML_FLCK**	Function lock lamp
MB_INP1	Machine control panel general input 1	**ML_OUT1**	Machine control panel general output 1

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
MB_M3	M3 button	**ML_M3**	M3 lamp
MB_M4	M4 button	**ML_M4**	M4 lamp
MB_M5	M5 button	**ML_M5**	M5 lamp
MB_INP2	Machine control panel general input 2	**ML_OUT2**	Machine control panel general output 2
MB_JOG1	Jog1 button	**ML_JOG1**	Jog1 lamp
MB_JOG2	Jog2 button	**ML_JOG2**	Jog2 lamp
MB_JOG3	Jog3 button	**ML_JOG3**	Jog3 lamp
MB_JOG4	Jog4 button	**ML_JOG4**	Jog4 lamp
MB_JOG5	Jog5 button	**ML_JOG5**	Jog5 lamp
MB_JOG6	Jog6 button	**ML_JOG6**	Jog6 lamp
MB_JOG7	Jog7 button	**ML_JOG7**	Jog7 lamp
MB_JOG8	Jog8 button	**ML_JOG8**	Jog8 lamp
MB_REFP	Refpoint mode button	**ML_REFP**	Refpoint mode lamp
MB_HNDL	Handwheel mode button	**ML_HNDL**	Handwheel mode lamp
MB_INCR	Incremental jog mode button	**ML_INCR**	Incremental jog mode lamp
MB_JOG	Jog mode button	**ML_JOG**	Jog mode lamp
MB_B20	Not used.	**ML_B20**	Not used.
MB_MDI	Manual data input mode button	**ML_MDI**	Manual data input mode lamp
MB_AUTO	Automatic mode button	**ML_AUTO**	Automatic mode lamp
MB_EDIT	Edit mode button	**ML_EDIT**	Edit mode lamp
MB_TEST	Test button	**ML_TEST**	Test lamp
MB_MLCK	Machine lock button	**ML_MLCK**	Machine lock lamp
MB_DRRUN	Dry run button	**ML_DRRUN**	Dry run lamp
MB_BKRST	Block restart button	**ML_BKRST**	Block restart lamp
MB_BKRET	Block return button	**ML_BKRET**	Block return lamp
MB_CNDSP	Conditional stop button	**ML_CNDSP**	Conditional stop lamp
MB_CNDBK	Conditional block button	**ML_CNDBK**	Conditional block lamp
MB_SGLBK	Single block button	**ML_SGLBK**	Single block mode lamp
MB_I1	1 increment button	**ML_I1**	1 increment lamp
MB_I10	10 increment button	**ML_I10**	10 increment lamp
MB_I100	100 increment button	**ML_I100**	100 increment lamp
MB_I1000	1000 increment button	**ML_I1000**	1000 increment lamp
MB_SMAX	S+% button	**ML_SMAX**	S+% button lamp
MB_S100	S100% button	**ML_S100**	S100% button lamp
MB_SMIN	S-% button	**ML_SMIN**	S-% button lamp

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
MB_JOGRAP	Jog rapid travel button	**ML_JOGRAP**	Jog rapid travel lamp
MB_USR1	PLC-defined button No. 1	**ML_USR1**	PLC-defined lamp No. 1
MB_USR2	PLC-defined button No. 2	**ML_USR2**	PLC-defined lamp No. 2
MB_USR3	PLC-defined button No. 3	**ML_USR3**	PLC-defined lamp No. 3
MB_USR4	PLC-defined button No. 4	**ML_USR4**	PLC-defined lamp No. 4
MB_USR5	PLC-defined button No. 5	**ML_USR5**	PLC-defined lamp No. 5
MB_USR6	PLC-defined button No. 6	**ML_USR6**	PLC-defined lamp No. 6
MB_USR7	PLC-defined button No. 7	**ML_USR7**	PLC-defined lamp No. 7
MB_USR8	PLC-defined button No. 8	**ML_USR8**	PLC-defined lamp No. 8
MB_USR9	PLC-defined button No. 9	**ML_USR9**	PLC-defined lamp No. 9
MB_USR10	PLC-defined button No. 10	**ML_USR10**	PLC-defined lamp No. 10
MB_USR11	PLC-defined button No. 11	**ML_USR11**	PLC-defined lamp No. 11
MB_USR12	PLC-defined button No. 12	**ML_USR12**	PLC-defined lamp No. 12
MB_USR13	PLC-defined button No. 13	**ML_USR13**	PLC-defined lamp No. 13
MB_USR14	PLC-defined button No. 14	**ML_USR14**	PLC-defined lamp No. 14
MB_B23	Not used	**ML_RESET**	Reset button lamp
MB_B24	Not used	**ML_CANCEL**	Cancel button lamp
MB_USR15	PLC-defined button No. 15	**ML_USR15**	PLC-defined lamp No. 15
MB_USR16	PLC-defined button No. 16	**ML_USR16**	PLC-defined lamp No. 16
MB_USR17	PLC-defined button No. 17	**ML_USR17**	PLC-defined lamp No. 17
MB_USR18	PLC-defined button No. 18	**ML_USR18**	PLC-defined lamp No. 18
MB_USR19	PLC-defined button No. 19	**ML_USR19**	PLC-defined lamp No. 19
MB_USR20	PLC-defined button No. 20	**ML_USR20**	PLC-defined lamp No. 20
MB_USR21	PLC-defined button No. 21	**ML_USR21**	PLC-defined lamp No. 21
MB_USR22	PLC-defined button No. 22	**ML_USR22**	PLC-defined lamp No. 22
NB_RESET	Reset button		
NB_CANCEL	Cancel button		
NB_NCPC	Not used		

The control panel hands over the state of feed-rate and spindle override buttons, too, through a 2 DWORD register.

DWORD variables of the control panel:

Inputs		**Outputs**	
Symbol	**Description**	**Symbol**	**Description**
MKBTNS0	Lower 32 buttons of the machine control panel (DWORD)	**MKLEDS0**	Lamps of the lower 32 buttons of the machine control panel (DWORD)
MKBTNS1	Upper 32 buttons of the machine control panel (DWORD)	**MKLEDS1**	Lamps of the upper 32 buttons of the machine control panel (DWORD)
NCTBTNS	Buttons (DWORD)		
MKFOVER	Value of the feed-rate override switch: 0, 1, 2...		
MKSOVER	Value of the spindle override: 0...10 (DWORD)		

Addressing of control panels

A maximum of 4 machine control panels can be connected to the control.
We can carry out the setting of address of the machine control panel in the ***Service menu*** of the control, after exchanging the window ***ECAT settings***. Select the appropriate unit on the left-side panel and by clicking on the ***Setting*** tab we can set the ***address*** of the unit.
Values of the addresses can be:

    1, 2, 3, 4.

The addresses of the control panel No. 1 can be accessed by referring directly to the symbol. For example:

    MB_START

refers to the START button of control panel No. 1.
Symbols of the other control panels (No. 2, 3, 4) can be accessed by an indexed reference. For example:

    MB_START,#2

refers to the START button of control panel No. 3 (with index No. 2).

PLC-defined buttons

The programmer of PLC can give functions to PLC-defined buttons and lamps. The same applies to JOG1...JOG8 buttons and lamps, too. For these ones the programmer of the PLC may define unique symbols depending on which axes are moved by the given buttons.
These bit symbols are always to be entered to the appropriate NC symbol as the basis (e.g. MB_JOG1) with a zero offset.

Example:
In the matrix of jog buttons, MB_JOG1 Symbol refers to the upper left-side button. On the factual machine this button moves axis X in a negative direction.
In case we would like to refer to the button by the symbol B_JOG_XN, then we have to declare in the PLC Editor the symbol by referencing to the symbol MB_JOG1 as the basis, with a zero offset – and no way shall we enter the numeric address of MB_JOG1 Symbol at the symbol B_JOG_XN.

Buttons which have both inputs and outputs toward the channels
The below mode-changing, operating condition-setting buttons and the start, stop buttons have outputs - indexed per channel - towards NC with a CP_ prefix and from the NC they have inputs indexed per channel with a CN_ prefix for the lamps:

MB_JOG	– CP_JOG	CN_JOG	– ML_JOG
MB_INCR	– CP_INCR	CN_INCR	– ML_INCR
MB_HNDL	– CP_HNDL	CN_HNDL	– ML_HNDL
MB_REFP	– CP_REFP	CN_REFP	– ML_REFP
MB_EDIT	– CP_EDIT	CN_EDIT	– ML_EDIT
MB_AUTO	– CP_AUTO	CN_AUTO	– ML_AUTO
MB_MDI	– CP_MDI	CN_MDI	– ML_MDI
MB_TEST	– CP_TEST	CN_TEST	– ML_TEST
MB_MLCK	– CP_MLCK	CN_MLCK	– ML_MLCK
MB_DRRUN	– CP_DRRUN	CN_DRRUN	– ML_DRRUN
MB_BKRST	– CP_BKRST	CN_BKRST	– ML_BKRST
MB_BKRET	– CP_BKRET	CN_BKRET	– ML_BKRET
MB_FLCK	– CP_FLCK	CN_FLCK	– ML_FLCK
MB_START	– CP_START	CN_START	– ML_START
MB_STOP	– CP_STOP	CN_STOP	– ML_STOP

The PLC program has to decide whether the pushing of the button is enabled on the machine side or not. For example, by pushing the MB_START button, whether the machine is switched on or not and whether the machining area is closed or not.
In case there aren't any obstacles from the machine's side, the PLC program requests through the CP channel handling flags the desired effect from the NC. The NC will examine whether the pushing of the button is enabled or not. For example, in case of CP_START=1 whether the program can be run in the given mode or not, etc.
In case the NC has accepted the pushing of the button, it will signal it through the a appropriate CN channel handling flag to the PLC. For example, by the CN_START=1 status. Thereafter the PLC may unconditionally switch on the lamp belonging to the button. By taking our example again: ML_START=1.

Buttons which have only outputs toward the channel

The below condition switching buttons have outputs indexed per channel towards NC with a CP_ prefix, but no feedback belongs to them from the NC side (CN flag):

MB_JOGRAP	– CP_JOGRAP	– ML_JOGRAP
MB_SGLBK	– CP_SGLBK	– ML_SGLBK
MB_CNDBK	– CP_CNDBK	– ML_CNDBK
MB_CNDSP	– CP_CNDSP	– ML_CNDSP

At these condition switching buttons the lamp of the button is to be handled based on the state stored in the PLC program, or simply the state of the button has to be copied onto the lamp.

Buttons which have outputs toward the axis control

The jog buttons have to be linked to the input - indexed per axis, with an AP_ prefix - of the axis to be moved:

MB_JOGn – AP_JOGP (+ direction), or AP_JOGN (- direction) – ML_JOGn

The lamps of jog buttons have to be linked directly to the button.

Control of the increment selection buttons

The size of the step calculated from the commands of increment selection buttons shall be entered into the floating-point registers - indexed per channel - of the control, in a floating-point form.
For example: *0.001, *0.01, *0.1, *1.
Based on the CN_INCH flag, it shall be taken into consideration whether the control is used by a metric or inch data input, and if it is necessary, to convert the data for the output unit of measurement system. Output unit of measurement system: N0104 Unit of Measure parameter #0 IND bit.

MB_I1, MB_I10, MB_I100, MB_I1000 – CP_INC

Control of the feed-rate and rapid override

The override value calculated from the feedrate override switch MKFOVER (DWORD) state shall be entered into the floating-point registers (double) - indexed per channel - of NC, in a floating-point form.
*1.0 corresponds to 100% in case of both overrides.
The value received in the MKFOVER register may change depending on the hardware structure of the override switch.
We can connect the rapid override input to the feed-rate override switch, too.

MKFOVER – CP_FOVER, CP_ROVER

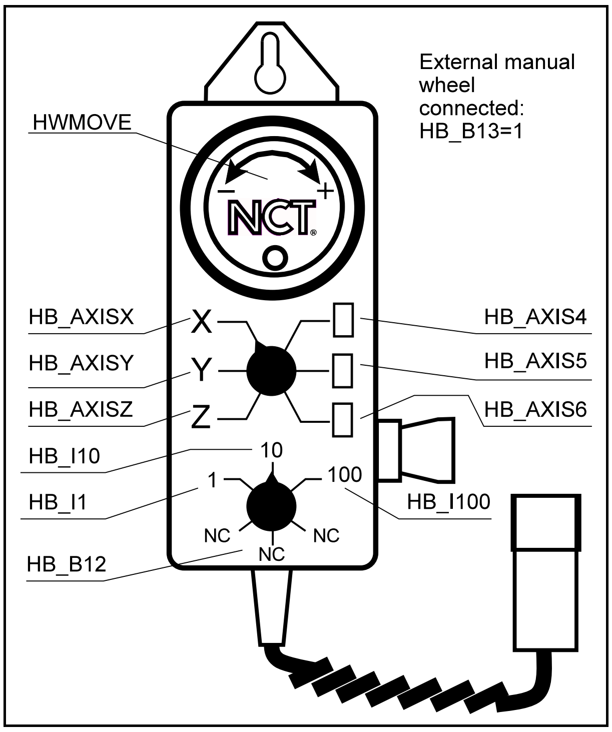
Control of the spindle override

We control the spindle override from buttons MB_SMAX, MB_S100 and MB_SMIN.
By pushing these buttons, or by using data of the MKSOVER register, the PLC program calculates a floating-point override value.
MKSOVER: DWORD, it hands over a value ranging from 0 to 10 to the PLC.
For the spindle override values, floating-point registers - indexed per spindle, with an SP prefix - are available. *1.0 corresponds to 100%:

MB_SMAX, MB_S100, MB_SMIN or MKSOVER – SP_SOVER

7.2 NCT Handwheels

In case of a handwheel mounted on the front panel, the selection of axes and increments is made by the buttons of the control panel. We call as an external handwheel the unit which is in a separate box and on the box there are axis- and increment selection rotary switches. Built-in bit symbols belong to the positions of the rotary switches of the external handwheel.



Bit variables of the external handwheel are the following:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
HB_AXISX	X axis input (external handwheel)	**HL_AXISX**	Not used
HB_AXISY	Y axis input (external handwheel)	**HL_AXISY**	Not used
HB_AXISZ	Z axis input (external handwheel)	**HL_AXISZ**	Not used
HB_AXIS4	4. axis input (external handwheel)	**HL_AXIS4**	Not used
HB_AXIS5	5. axis input (external handwheel)	**HL_AXIS5**	Not used
HB_AXIS6	6. axis input (external handwheel)	**HL_AXIS6**	Not used
HB_AXIS7	7. axis input (external handwheel)	**HL_AXIS7**	Not used
HB_AXIS8	8. axis input (external handwheel)	**HL_AXIS8**	Not used
HB_I1	1 increment input (external handwheel)	**HL_I1**	Not used
HB_I10	10 increment input (external handwheel)	**HL_I10**	Not used
HB_I100	100 increment input (external handwheel)	**HL_I100**	Not used

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
HB_I1000	1000 increment input (external handwheel)	**HL_I1000**	Not used
HB_B12	Not used	**HL_B12**	Not used
HB_B13	Not used	**HL_B13**	Not used
HB_B14	Not used	**HL_B14**	Not used
HB_B15	Not used	**HL_B15**	Not used

In case of the external handwheel indicated in the above picture, the below symbols are connected:

HB_B12: The **NC** position of the increment selector switch

HB_B13: The flag's true state means that an **external handwheel** is **attached** to the control.

A handwheel double-word variables are the following:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
HWMOVE	Handwheel: displacement per cycle (exclusively in Int0 module!) (DWORD)		
HWBITS	External handwheel inputs (DWORD)	**HWLEDS**	Not used
		P_H1AS	The number of axis assigned to the handwheel No. 1 (=0 inactive) (DWORD)
		P_H2AS	The number of axis assigned to the handwheel No. 2 (=0 inactive) (DWORD)
		P_H3AS	The number of axis assigned to the handwheel No.3 (=0 inactive) (DWORD)
		P_H4AS	The number of axis assigned to the handwheel No.4 (=0 inactive) (DWORD)

HWMOVE: the displacement of the handwheel compared to the previous cycle can be read out from the register. As the updating of the register is carried out by the frequency of the TimeSlice cycle time, the register shall be read out in the Int0 module of the PLC program. The register can be read both in case of a built-in and external handwheel.

P_HnAS: the number of that axis shall be entered into the register assigned to handwheel 1-4, which we would like to move by the given handwheel. In case the value of the register is 0, the handwheel is inactive.

Addressing of the handwheels

A maximum of 4 handwheels can be connected to the control.
We can carry out the setting of the address of the handwheel in the ***Service menu*** of the control, after exchanging the window ***ECAT settings*** at the settings of the ***machine control panel***. Select the appropriate unit on the left-side panel and by clicking on the ***Setting*** tab we can set the ***address*** of the unit.
Values of the addresses can be:

1, 2, 3, 4.

The address of the built-in handwheel is always 4.
The address of the external handwheel is usually (default setting): 4.
☞ ***Attention!*** *As in the standard way we use the handwheel No. 4, do not forget to index the symbols (symbol,#3)!*

7.3 Two-state, 24V Interface In- and Outputs

The below NCT-brand, EtherCAT, two-state, 24V interface in- and output units can be connected to the control:

I16: 16-bit 24V interface input,
I16S: 3x16-bit 24V interface input, and to each input point a 24V and 0V power supply voltage access point belongs (for sensors),
I32: 32-bit 24V interface input,
O16: 16-bit 24V interface output with transistors,
O8RM: 8-bit 24V interface output with Morse type contact relays
O8R: 8-bit 24V interface output with open contact relays

The *NC symbols* of the two-state, 24V interface in- and outputs are *32-bit* (DWORD) references.

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
INP000	Interface Inputs 0. DWORD	**OUT000**	Interface outputs 0. DWORD
INP001	Interface Inputs 1. DWORD	**OUT010**	Interface outputs 1. DWORD
INP010	Interface Inputs 2. DWORD	**OUT020**	Interface outputs 2. DWORD
INP011	Interface Inputs 3. DWORD	**OUT030**	Interface outputs 3. DWORD
INP020	Interface Inputs 4. DWORD	**OUT040**	Interface outputs 4. DWORD
INP021	Interface Inputs 5. DWORD	**OUT050**	Interface outputs 5. DWORD
INP030	Interface Inputs 6. DWORD	**OUT060**	Interface outputs 6. DWORD
INP031	Interface Inputs 7. DWORD	**OUT070**	Interface outputs 7. DWORD
INP040	Interface Inputs 8. DWORD		
INP041	Interface Inputs 9. DWORD		
INP050	Interface Inputs 10. DWORD		
INP051	Interface Inputs 11. DWORD		
INP060	Interface Inputs 12. DWORD		
INP061	Interface Inputs 13. DWORD		
INP070	Interface Inputs 14. DWORD		
INP071	Interface Inputs 15. DWORD		

Addressing of I/O cards

The number of in- and outputs of the *in- and output hardware units* connected to the control is the integer multiple of the byte (8 bits).
We can carry out the setting of the starting addresses of in- and output units in the ***Service menu*** of the control, after exchanging the window ***ECAT settings.*** Select the appropriate unit on the left-side panel and by clicking on the ***Setting*** tab we can set the ***starting address*** of the unit.

The addressing is carried out per bytes. The addresses of units, separately for the in- and output units may range from 1 to 64. The address occupation of 8-bit units is 1 address, address occupation of 16-bit ones is 2 addresses, etc.
Address 1

in case of an input unit is the 0th byte of the word INP000,
in case of an output unit is the 0th byte of the word OUT000.

i.e. bits of 00...07.
Address 2

is the byte 1 of the 0th DWORD, i.e.the bits of 08...15 and so on.

The order of the addresses does not have to follow the physical order of connection to the EtherCAT chain.

The below table shows a simple sample of filling in the starting addresses and the place of signs in the memory.

Hardware unit	Starting address	Reference to the signal
8-bit input unit	1	INP000: 00...07 bit
16-bit input unit	2	INP000: 08...23 bit
16-bit input unit	4	INP000: 24...31, INP001: 00...07 bit
8-bit input unit	6	INP001: 08...15 bit
16-bit output unit	1	OUT000: 00...15 bit
8-bit output unit	3	OUT000: 16...23 bit
16-bit output unit	4	OUT000: 24...31, OUT010: 00...07 bit
8-bit output unit	6	OUT010: 08...15 bit

The symbols of the unique, in- and output bits can be determined by the PLC programmer itself. These bit symbols have always to be declared for the appropriate NC symbol as the basis (e.g. INP000). Within the double word the selection of the bit is carried out numerically (00, ..., 31) according to the I/O allocation.

Example:
We would like to refer to the 17th bit of the DWORD INP000 by the symbol MGZ_RPT (magazine in the reference point switch).
We have to declare the MGZ_RPT symbol with a 0-offset reference to the INP000 symbol as the basis and we have to enter 17 to the bit number.

7.4 In- and Outputs of NCT Probe Interface Cards

Through the EtherCAT bus, cards suitable for receiving the signals of probes can be connected to the control. Their types are:

ETPC: 2-channel probe control card. On the card, besides the signal Probe pushed, 3 pieces of 24V inputs and 2 pieces of 24V outputs are available per probe.

The control is able to manage signals of 8 probes.

In- and outputs of probe interface cards:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
TN_TS1	Probe 1 is pushed	**TP_OUT11**	Probe 1 output 1
TN_INP11	Probe 1 input signal 1	**TP_OUT12**	Probe 1 output 2
TN_INP12	Probe 1 input signal 2	**TP_OUT21**	Probe 2 output 1
TN_INP13	Probe 1 input signal 3	**TP_OUT22**	Probe 2 output 2
TN_TS2	Probe 2 is pushed	**TP_OUT31**	Probe 3 output 1
TN_INP21	Probe 2 input signal 1	**TP_OUT32**	Probe 3 output 2
TN_INP22	Probe 2 input signal 2	**TP_OUT41**	Probe 4 output 1
TN_INP23	Probe 2 input signal 3	**TP_OUT42**	Probe 4 output 2
TN_TS3	Probe 3 is pushed	**TP_OUT51**	Probe 5 output 1
TN_INP31	Probe 3 input signal 1	**TP_OUT52**	Probe 5 output 2
TN_INP32	Probe 3 input signal 2	**TP_OUT61**	Probe 6 output 1
TN_INP33	Probe 3 input signal 3	**TP_OUT62**	Probe 6 output 2
TN_TS4	Probe 4 is pushed	**TP_OUT71**	Probe 7 output 1
TN_INP41	Probe 4 input signal 1	**TP_OUT72**	Probe 7 output 2
TN_INP42	Probe 4 input signal 2	**TP_OUT81**	Probe 8 output 1
TN_INP43	Probe 4 input signal 3	**TP_OUT82**	Probe 8 output 2
TN_TS5	Probe 5 is pushed		
TN_INP51	Probe 5 input signal 1		
TN_INP52	Probe 5 input signal 2		
TN_INP53	Probe 5 input signal 3		
TN_TS6	Probe 6 is pushed		
TN_INP61	Probe 6 input signal 1		
TN_INP62	Probe 6 input signal 2		
TN_INP63	Probe 6 input signal 3		
TN_TS7	Probe 7 is pushed		
TN_INP71	Probe 7 input signal 1		
TN_INP72	Probe 7 input signal 2		

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
TN_INP73	Probe 7 input signal 3		
TN_TS8	Probe 8 is pushed		
TN_INP81	Probe 8 input signal 1		
TN_INP82	Probe 8 input signal 2		
TN_INP83	Probe 8 input signal 3		

Addressing of the probe fitter

The control handles the signals of a maximum of 8 probes.
A probe interface card fits 2 probe in- and outputs.
We can carry out the setting of the addresses of the interface card in the ***Service menu*** of the control, after exchanging the window ***ECAT settings*** at the settings of the ***probe fitter***. Select the appropriate unit on the left-side panel and by clicking on the ***Setting*** tab we can set the in- and output ***addresses*** of the two probes used on the card.
Values of the addresses can be:

Not used, 1, 2, ..., 8

The values of addresses correspond to the symbol indexes of the in- and outputs of the probe.

Interface inputs of the probe

TN_TSn: Probe n is pushed

This input indicates that the stylus of the probe with address n is deflected or the button of the probe is pushed.
The Probe is pushed sign is mandatorily an active 0! The interface card shall be set in a way to fulfil this condition.

TN_INPn1: Probe n input signal 1
TN_INPn2: Probe n input signal 2
TN_INPn3: Probe n input signal 3

3 pieces of optional 24V inputs of the probe with address n.
The inputs can be used for e.g. signaling the ready state of the probe or the discharged state of the battery.

Interface outputs of the probe

TP_OUTn1: Probe n output 1
TP_OUTn2: Probe n output 2

2 pieces of optional 24V outputs of the probe with address n.
The outputs can be used for example for switching on and off the probe.

7.5 NCT Sensor Inputs

The type of card - which can be connected to the EtherCAT network - containing a sensor inputs:

SENS: The card contains 8 pieces of KTY84/130 temperature sensor analog inputs and 1 piece of 12-bit 4-20 mA A-D converter. A comparison value can be set for the temperature sensor inputs.

Bit inputs of the sensor:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
IN_1EN0	0. analog input > comparison value		
IN_1EN1	1. analog input > comparison value		
IN_1EN2	2. analog input > comparison value		
IN_1EN3	3. analog input > comparison value		
IN_1EN4	4. analog input > comparison value		
IN_1EN5	5. analog input > comparison value		
IN_1EN6	6. analog input > comparison value		
IN_1EN7	7. analog input > comparison value		
IN_1EN8	Not used		

DWORD inputs of the sensor

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
ANINPUTS	Analog Inputs: 32db DWORD		
IN_1	Status of analog comparator inputs (DWORD)		

Addressing of the SENS card and setting of the comparison value

The control handles the signals of a maximum of 32 SENS cards.

We can carry out the setting of addresses of the SENS card in the ***Service menu*** of the control after exchanging the window ***ECAT settings*** at the settings of ***card***. Select the appropriate unit on the left-side panel and by clicking on the ***Setting*** tab we can set the address of the card.

Values of the addresses can be:

Not used, 1, 2, ..., 32.

Symbols IN_1ENn, IN_1 and ANINPUTS refer to card No. 1. (indexed as 0), and the signals of other cards can be accessed by an indexed reference.

We can carry out the comparison values in degrees (centigrade), per channel, by exchanging the window ***ECAT settings***, at the settings of the card.

Bit inputs of the sensor:

IN_1ENn: analog input No. n > comparison value

In case on the nth input of the card the value of the analog signal exceeds the comparison value set, the flag will turn to 1.

Double-word inputs of the sensor

ANINPUTS: Analog inputs: 32 pieces DWORD

On the ANINPUTS variable, with an appropriate indexation, the value of the analog input of the card can be read.

IN_1: Status of analog comparator inputs (DWORD)

The comparator inputs can be accessed from the variable in a DWORD form.

7.6 NCT Analog Inputs

Type of card - which can be connected to the EtherCAT network - containing analog inputs:

DANI: The card contains 6 pieces of 12-bit analog to digital converters. The analog inputs can be configured to either +/-10V or 0-20 mA.

DWORD inputs of analog signals

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
ANINPUTS	Analog inputs: 32 pieces DWORD		

Addressing of DANI card

The control manages the signals of a maximum of 32 analog inputs.
We can carry out the setting of addresses of the DANI card in the ***Service menu*** of the control, after exchanging the window ***ECAT settings*** at the settings of the ***card***. Select the appropriate unit on the left-side panel and by clicking on the ***Setting*** tab you can set the addresses of the analog inputs one-by-one.
Values of the addresses can be:

Not used, 1, 2, ..., 32.

The symbol ANINPUTS refers to the analog input No. 1 (indexed as 0), and the signals of the other inputs can be accessed by an indexed reference.

DWORD inputs of analog signals

ANINPUTS: Analog inputs: 32 pieces, DWORD

On the ANINPUTS variable, with an appropriate indexation, it is possible to read the appropriate analog input of the card.

7.7 In- and Outputs of EtherCAT NCT Drives

NCT drives of following types can be connected to the control through the EtherCAT bus:
DS-i/I EE: to synchronous servo motors,
DA-i/I EE: asynchronous motors,
where: i: is the nominal current,
I: maximum current
EE: EtherCAT, EnDat 2.2 interface.
The system is able to handle the signals of a maximum of 48 pieces of EtherCAT drives (32 axes + 16 spindles).
The EtherCAT drives hand over to NC through the EtherCAT network the position, number of revolution, etc. measured from the encoder mounted on the motor; their states, respectively, the codes of their error states. They receive in the same way the speed set-point signal from NC; respectively, other commands arriving from NC.
Signals of the above type NCT EtherCAT drives are the following:

Bit status and control signals of drives

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
DN_ENA	Enable acknowledge	**DP_ENA1**	Drive enable bit 1
DN_RDY	Drive ready for operation	**DP_ENA2**	Drive enable bit 2
DN_INC	Incremental encoder on the drive	**DP_EMG**	No emergency braking
DN_PRM1	Active parameter table bit 1	**DP_POSLCK**	Drive position keeping on
DN_PRM2	Active parameter table bit 2	**DP_PRM1**	Drive parameter table selection bit 1
		DP_PRM2	Drive parameter table selection bit 2
		DP_MOD1	Drive regulator mode selection bit 1
		DP_MOD2	Drive regulator mode selection bit 2
		DP_SILCK	Speed integrator lock
		DP_ERRCLR	Drive error clear

Bit error flags of drives

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
DN_ERROR	There is a drive error (DN_ERR>0)		
DN_EDERR1	Encoder error		
DN_EDERR2	Temperature of the encoder is too high		
DN_IEERR1	Not used		

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
DN_IEERR2	Not used		
DN_BVERR	Excess voltage error on the bus		
DN_CURERR	Motor current too high		
DN_CMEERR	Current measurement error		
DN_HALERR	Hall commutation signal error		
DN_HASERR	Hall sequence signal error		
DN_SRTERR	Timeout error		
DN_CWDER1	CAN Watchdog error		
DN_CWDER2	Not used		
DN_CHERR1	CAN other error		
DN_CHERR2	Not used		
DN_ECTERR	EtherCAT timeout error		
DN_PDPINT	IGBT error		
DN_PRMERR	Parameter table error		
DN_PRGERR	Drive firmware error		
DN_FOLERR	Speed following error		
DN_OVHERR	Overheat protection		

DWORD variables of drives

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
DN_STAT	Drive status register (DWORD)	**DP_CTRL**	Drive control register (DWORD)
DN_ERR	Register of drive error flags (DWORD)		

Addressing of EtherCAT drives

The control handles the signals of a maximum of 48 pieces of EtherCAT drives.
We can carry out the setting of addresses of drives in the ***Service menu*** of the control after exchanging the window ***ECAT settings*** by selecting the ***appropriate drive***. Select the appropriate unit on the left-side panel and by clicking on the ***Setting*** tab and set the ***address*** of the drive.
Values of the addresses can be:

Not used, 1, 2, ..., 48.

Binding of addresses of ***drives*** and ***axes***

The control handles a maximum of 32 axes. The addresses of axes in the control can be:

1, 2, ..., 32.

Issuance of the speed set-point (command) signal:

Let's select the option EtherCAT-32 on the parameter N0502 Axis Output Type.

On the parameter N0503 Axis Output Address we have to enter the address of the drive by which we would like to move the given axis.

Example:

The parameter

N0100 Axis Name1 A4=C

determines the address of axis 4 as C.

The address of drive of axis C between the EtherCAT settings is 6.

Then the N0503 Axis Output Address A4=6,

which means that control's axis 4 issues the command signal to drive 6.

Reception of the encoder's signals:

Let's select the option EtherCAT on the parameter N0500 Axis Input Type.

Let's write on the parameter N0501 Axis Input Address the address of the drive from which we would like to receive the signals of the encoder.

☞ ***Attention!*** *The N0503 Axis Output Address parameter and the N0501 Axis Input Address parameter may be different!*

Example:

The parameter

N0100 Axis Name1 A4=C

determines the address of axis 4 as C.

The address of the drive of axis C between the EtherCAT settings is 6.

We would like to receive the signals of the encoder from the encoder handled by the drive.

Then the N0501 Axis Input Address A4=6,

i.e. axis 4 of the control receives the signals of the encoder from drive 6.

Binding of addresses of ***drives*** and ***spindles***

The control handles a maximum of 16 spindles. The addresses of spindles in the control can be:

1, 2, ..., 16.

Issuance of the speed set-point (command) signal:

Let's select the option EtherCAT on the parameter N0602 Spindle Output Type.

Let's write on the parameter N0603 Spindle Output Address the address of the drive by which we would like to move the given spindle.

Example:
Parameter

N0605 Spindle Name2 S3=3

determines the address of spindle 3 as S3.
The address of the drive of spindle S3 between the EtherCAT settings is 8.
Then the N0603 Spindle Output Address S3=8,
i.e. the control's spindle 3 issues the command signal to drive 8.

Reception of the encoder's signals:
Let's select the option EtherCAT on the parameter N0600 Spindle Input Type.
Let's write on the parameter N0601 Spindle Input Address the address of the drive from which we would like to receive the signals of the encoder.
☞ ***Attention!*** *The N0603 Spindle Output Address parameter and the N0601 Spindle Input Address parameter may be different!*
Example:
Parameter

N0605 Spindle Name2 S3=3

determines the address of spindle 3 as S3.
The address of the drive of spindle S3 between the EtherCAT settings is 8.
We would like to receive the signals of the encoder from the encoder handled by the drive.
Then the N0601 Spindle Input Address S3=8,
i.e. the control's spindle 3 receives the signals of the encoder from drive 8.

Reference in the PLC program to axes, spindles and drives
The

AN_ xx and AP_xx

NC symbols always refer to axes. We have to refer to these symbols always based on axis index:

#0: axis 1, ..., #31: axis 32.

The

SN_ xx and SP_xx

NC symbols always refer to spindles. We have to refer to these symbols always based on spindle index:

#0: spindle 1, ..., #15: spindle 16.

The

DN_ xx and DP_xx

NC symbols always refer to drives. We have to refer to these symbols always based on drive index:

#0: drive 1, ..., #47: drive 48.

The axis-, respectively, spindle index of a given axis or spindle may be different from the drive index of the given axis or spindle!

Status signals of drives (inputs)

DN_ENA: Enable acknowledge

It is the acknowledgment flag of signals' DP_ENA1 and DP_ENA2. If the value of the flag is 1, the motor is operated by the drive, and it is under voltage.

DN_RDY: Drive ready for operation

If the value of the signal is 1 the drive is ready for operation, and can be enabled.

DN_INC: Incremental encoder on the drive

If the flag DN_INC=0, there is an absolute, EnDAT encoder on the motor,
if the flag DN_INC=1, there is an incremental encoder on the motor.

DN_PRM1: Active parameter table bit 1
DN_PRM2: Active parameter table bit 2

The above two bits tell from which parameter table does the drive take the data. See also: description of DP_PRM1 and DP_PRM2 bits.

DN_PRM2	DN_PRM1	Active parameter table
0	0	1. parameter table active
0	1	2. parameter table active
1	0	3. parameter table active
1	1	4. parameter table active

Control signals of drives (outputs)

DP_ENA1: Drive enable bit 1
DP_ENA2: Drive enable bit 2

The enabling of the drive happens upon the below bit change

DP_ENA1	1 –> 0, and
DP_ENA2	0 –> 1.

DP_ENA2	DP_ENA1	Enable status
0	0	Not enabled
0	1	Not enabled
1	0	Enabled
1	1	Not enabled

DP_EMG: No emergency braking

If DP_EMG=0, the drive, independently of the command signal received on its input, will stop the motor. The motor will be stopped till the drive is enabled.

DP_EMG=1 normal operation, according to the received command signal.

DP_POSLCK: Drive position keeping on

DP_POSLOCK=0, normal mode,

DP_POSLOCK=1, at the 0 –> 1 transition of the signal the drive will keep the motor in the position measured from the encoder, independently of the received command signal.

DP_PRM1: Drive parameter table selection bit 1
DP_PRM2: Drive parameter table selection bit 2

The above two bits tell from which parameter table shall the drive take data. After the parameter change, we have to wait till the drive returns the set bit sample at bits DN_PRM1 and DN_PRM2.

DP_PRM2	DP_PRM1	Parameter table selection
0	0	1. parameter table selection
0	1	2. parameter table selection
1	0	3. parameter table selection
1	1	4. parameter table selection

DP_MOD1: Drive regulator mode selection bit 1
DP_MOD2: Drive regulator mode selection bit 2

By the setting of flags we can select from the below regulation modes:

DP_MOD2	DP_MOD1	Regulation mode selection
0	0	Speed regulating mode
0	1	Current regulating (torque) mode
1	0	Position regulating mode
1	1	Speed regulating mode with an increased precision

DP_SILCK: Speed integrator lock

DP_SILCK=0, normal mode,

DP_SILCK=1: switch-off of the integrator of the speed regulator.

Example:

On a lathe, with a counter spindle, working from a bar, we have to take over the piece into the counter spindle. We synchronize the counter spindle to the main spindle, then we close the chuck on the counter spindle, so that after the recession the counter spindle can hold the piece. Till the piece is not recessed, and it is held by both chucks, the integrator of the speed regulator has to be switched off on the drive of the counter spindle, otherwise the over-determinedness of the system an oscillation may occur.

DP_ERRCLR: Drive error clear

If there is an error on the drive, i.e. DN_ERR>0 (the error register is not 0) the error shall be cleared.

Status DP_ERRCLR=1 clears the error. The signal shall be kept in 1 till the time the DN_ERR error register turns 0, or till the status DN_ERROR=0 occurs.

Bit error flags of the drive (inputs)

The PLC program does not have to send messages upon the various drive errors, as this is done by the NC. Only the clearance of the error shall be issued by the DP_ERRCLR=1 flag setting.

DN_ERROR: There is a drive error (DN_ERR>0)

DN_ERROR=1, if any of the further bits of the error register signals an error.

DN_EDERR1: Encoder error

DN_EDERR1=1, if the drive recognizes an error from the encoder mounted on the motor.

DN_EDERR2: Temperature of the encoder is too high

DN_EDERR2=1, in case an EnDat encoder is mounted on the motor, and the temperature sensor built into the encoder measures a temperature higher than the set value.

DN_BVERR: Excess voltage error on the bus

DN_BVERR=1, in case the bus voltage has exceeded the set value.

DN_CURERR: Motor current too high

DN_CURERR=1, in case the motor current has exceeded the I_{max} value set on the drive.

DN_CMEERR: Current measurement error
DN_CMEERR=1, in case a current measurement error has occurred.

DN_HALERR: Hall commutation signal error
DN_HALERR=1, in case a commutation signal error has occurred.

DN_HASERR: Hall sequence signal error
DN_HASERR=1, in case the sequence of the commutation signal is not according to the Gray code.

DN_SRTERR: Timeout error
DN_SRTERR=1, in case in the drive the central watchdog timer has elapsed. The drive is broken down.

DN_CWDER1: CAN watchdog error
DN_CWDER1=1, in case in the drive the watchdog timer of the CAN communication has elapsed. The CAN communication is broken down.

DN_CHERR1: CAN other error
DN_CHERR1=1, in case an error has occurred in the CAN communication.

DN_ECTERR: EtherCAT timeout error
DN_ECTERR=1, in case in the drive the watchdog timer of the EtherCAT communication has elapsed. The EtherCAT communication is broken down.

DN_PDPINT: IGBT error
DN_PDPINT=1, in case an error has occurred during the feeding of the motor. It may be a circuit or a cable error.

DN_PRMERR: Parameter table error
DN_PRMERR=1, In case the active parameter table is damaged.

DN_PRGERR: Drive firmware error
DN_PRGERR=1, if the operating program of the drive is damaged, that is there is a checksum error.

DN_FOLERR: Speed following error
DN_FOLERR=1, if the drive was not able to follow the speed set-point within a set period of time.

DN_OVHERR: Overheat protection

DN_OVHERR=1, if the heat protection of the motor has issued an alarm signal. It may be a PTC or a temperature model.

DWORD input registers of drives

DN_STAT: Drive status register (DWORD)

By this symbol it is possible to access the status bits of the drive in a double-word format.

DN_ERR: Register of drive error flags (DWORD)

By this symbol it is possible to access the error bits of the drive in a double-word format.

DWORD output registers of drives

DP_CTRL: Drive control register (DWORD)

By this symbol it is possible to access the control signals of the drive in a double-word format.

7.8 Encoder Receiver and Analog/Stepping Motor/CAN Drive Interface Cards

The encoder receiver and analog/stepping motor/CAN drive interface cards can be fitted onto the EtherCAT bus.

We use these units in the below cases:

– The EtherCAT drives hand over the position measured from the encoder mounted onto the motor to the NC. In cases where instead of the encoder mounted on the motor we need to use another encoder, the signal shall be taken from an appropriate receiver card. Such cases are for example the thread cutting encoder mounted onto the spindles of lathes, or if we use a linear scale on an axis for position measurement.

– The second case is where instead of an EtherCAT drive, it is necessary to fit an analog, stepping motor or CAN bus drive to the control.

The below interface cards are available:

– **ENDAT**:

Inputs:

It is suitable for receiving the signals of 2 pieces of encoders with EnDat 2.2 communication protocol (rotary encoder, linear scale, angle encoder).

– **TTLAI**:

Inputs:

The unit is able to receive on its input the signals of 2 incremental, TTL encoders.

Outputs:

In case of an

ECAT-TTLASM software version,

analog speed set-point signals are issuable on its 2 outputs, or

Pulse-Dir pulse series and direction bit for stepping motors, or

CW-CCW two-direction pulse series for stepping motors.

All three outputs are issued per channel, and we can select by connection, which one to apply.

In case of an

ECAT-TACHO software version, it gives out an analog signal created from the difference between the speed set-point signal and tach signal, issuable on its 2 outputs, for analog-type drives.

– **TTLCAN**:

Inputs:

The unit is able to receive on its input the signals of 2 incremental, TTL encoders.

Outputs:

It is suitable for the handling of 2 CAN-bus-input NCT drives.

The communication registers and flags of interface cards keep connection with the PLC on the DN_, DP_drive flags. The system is able to handle the signals of a maximum of 48 pieces of EtherCAT drives and interface units (32 axes + 16 spindles).

Below we have indicated only those signals which are handled by the interface units.

Bit status and control signals of interface cards

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
DN_INC	Incremental encoder on the drive/unit	**DP_ERRCLR**	Drive/unit error clear

Bit error flags of interface cards

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
DN_ERROR	There is a drive/unit error (DN_ERR>0)		
DN_EDERR1	Encoder error		
DN_ECTERR	EtherCAT timeout error		

DWORD variables of interface cards

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
DN_STAT	Drive/unit status register (DWORD)	**DP_CTRL**	Drive/unit control register (DWORD)
DN_ERR	Register of drive/unit error flags (DWORD)		

Addressing of EtherCAT interface cards
The control is able to handle signals of a maximum of 48 pieces of EtherCAT drives and interface cards. We can carry out the setting of addresses of interface cards in the ***Service menu*** of the control after exchanging the window ***ECAT settings*** by selecting the ***appropriate interface card***. Let's select the appropriate unit in the left-side panel and by selecting the -***Setting*** tab we can set the ***address*** of the interface card.
The values of addresses can be:

Not used, 1, 2, ..., 48.

Binding of addresses of ***interface cards*** and ***axes***
The control handles a maximum of 32 axes. Addresses of axes in the control may be:

1, 2, ..., 32.

Issuance of the set-point (command) signal:

Let’s select the option EtherCAT-32 on the parameter N0502 Axis Output Type.

Let’s write onto the parameter N0503 Axis Output Address the address of the interface cards by which we intend to move the given axis.

Example:

The parameter

N0100 Axis Name1 A4=C

determines the address of axis 4 as C.

Axis C is moved by a drive with an analog input. The address of the TTLAI’s interface card between the EtherCAT settings is: 6.

Then the N0503 Axis Output Address A4=6,

i.e. axis 4 of the control issues the command signal to the interface card No. 6.

Reception of signals of the encoder:

Let’s select the option EtherCAT on the parameter N0500 Axis Input Type.

Let’s write onto the parameter N0501 Axis Input Address the address of the interface card from which we intend to receive the signals of the encoder.

☞ ***Attention!*** *The parameters N0503 Axis Output Address and N0501 Axis Input Address may be different!*

Example:

The parameter

N0100 Axis Name1 A2=Y

determines the address of axis 2 as Y.

The address of the EtherCAT drive of axis Y between the EtherCAT settings can be 2 (N0503 Axis Output Address A2=2).

An EnDat linear scale is mounted onto axis Y. The signals of linear scale Y are handled by the ENDAT interface card under address 12.

Then the N0501 Axis Input Address A2=12,

i.e. axis 2 of the control takes the signals of the encoder from interface card No. 12.

Binding of the addresses of ***interface cards*** and ***spindles***

The control handles a maximum of 16 spindles. Addresses of spindles in the control can be:

1, 2, ..., 16.

Issuance of the set-point (command) signal:

Let’s select the option EtherCAT on the parameter N0602 Spindle Output Type.

Let’s write onto the parameter N0603 Spindle Output Address the address of the interface card by which we intend to move the given spindle.

Example:
The parameter
N0605 Spindle Name2 S1=1
determines the address of spindle 1 as S1.
The S1 spindle drive has a CAN bus input, the set-point signal is given to the drive by the TTLCAN interface card, the address of which between the EtherCAT settings is 8.
Then the N0603 Spindle Output Address S1=8,
i.e. spindle 1 of the control issues the set-point signal to interface card No. 8.

Reception of signals of the encoder:
Let's select the option EtherCAT on the parameter N0600 Spindle Input Type.
Let's write onto the parameter N0601 Axis Input Address the address of the interface card from which we intend to receive the signals of the encoder.
☞ ***Attention!*** *The parameters N0603 Spindle Output Address and N0601 Spindle Input Address may be different!*
Example:
The parameter
N0605 Spindle Name2 S1=1
determines the address of spindle 1 as S1.
The address of the interface card of spindle drive S1 between the EtherCAT settings is 8.
We would like to receive the signals of the encoder from an encoder mounted onto the spindle. (And not from the motor)
The signals of the encoder mounted onto the spindle are received by a TTLAI interface card, the address of which is 21. Then the N0601 Spindle Input Address S1=21,
i.e. spindle 1 of the control receives the encoder signals from interface card No. 21.

Reference in the PLC program to axes, spindles and drives
NC symbols
AN_ xx and AP_xx
always refer to axes. We have to refer to these symbols always based on axis index:
#0: axis 1, ..., #31: axis 32.
NC symbols
SN_ xx and SP_xx
always refer to spindles. We have to refer to these symbols always based on spindle index:
#0: spindle 1, ..., #15: spindle 16.
NC symbols
DN_ xx and DP_xx
NC always refer to drives or interface cards. We have to refer to these symbols always based on drive index:
#0: drive 1, ..., #47: drive 48.

The axis-, respectively, spindle index of a given axis or spindle may be different from the drive index of the given axis or spindle!

Status signals of interface cards (inputs)

DN_INC: Incremental encoder on the drive/unit
If the flag DN_INC=0, there is an absolute, EnDAT encoder on the motor,
if the flag DN_INC=1, there is an incremental encoder on the motor.

Control signals of the interface cards (outputs)

DP_ERRCLR: Drive/unit error clear
In case there is an error on the interface card, i.e. DN_ERR>0 (the error register is not 0) the error shall be cleared.
Status DP_ERRCLR=1 clears the error. The signal shall be kept in 1 till the time the DN_ERR error register turns 0, or till the status DN_ERROR=0 occurs.

Bit error flags of interface cards (inputs)

The PLC program does not have to send messages to the various interface card errors, as this is done by the NC. It only has to clear the error by the flag status DP_ERRCLR=1.

DN_ERROR: There is a drive/unit error (DN_ERR>0)
DN_ERROR=1, if any of the further bits of the error register indicates an error.

DN_EDERR1: Encoder error
DN_EDERR1=1, in case the interface card recognizes an error from the encoder.

DN_ECTERR: EtherCAT timeout error
DN_ECTERR=1, in case in the interface card the EtherCAT communication's watchdog timer has elapsed. The EtherCAT communication is broken down.

DWORD input registers of interface cards

DN_STAT: Drive/unit status register (DWORD)

By this symbol it is possible to access the status bits of interface card in a double-word way.

DN_ERR: Register of drive/unit error flags (DWORD)

By this symbol it is possible to access the error bits of interface card in a double-word way.

DWORD output registers of interface cards

DP_CTRL: Drive/unit control register (DWORD)

By this symbol it is possible to access the control signals of the interface card in a double-word way.

In case of TTLAI interface card ECAT-TACHO software, the handling of the control word:

Through the PLC we have to enter an integer into the upper word of the DP_CTRL (from bit 16 to 31) based on the following formula:

(pulse number of the encoder*4)*(maximum number of rev. of the motor/60)/5000

In case we enter a zero here, the unit will not carry out tach-compensation, and it will give out an analog signal corresponding to the speed set-point (command) signal.

7.9 Function Buttons Accessible from the PLC

32 pieces of function buttons accessible for the PLC program on the screen of the control. We have to enter the title of function buttons in the PLC program in the bottom right-side panel of the PLC Editor by clicking on the PLC buttons tab.
2 DWORD registers belong to the 32 function buttons: the input register belongs to the button, whilst the output register belongs to the lamp of the button.

DWORD variables of the function buttons:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_MSG	Buttons of 32 function buttons on the screen, which may be used by the PLC (DWORD)	**P_MSG**	Lamps of 32 function buttons on the screen, which may be used by the PLC (DWORD)

Referencing to registers N_MSG and P_MSG we can declare bit symbols, which symbols indicate the function buttons and their lamps.

Example:
Let's say the B_LUB Symbol is the symbol of manual start of the lubricating pump, which we would like to start from the function button F23.
On the PLC buttons panel, we entered the title LUBRICATION into the text box of button F23, which text appears on the screen on the appropriate function button.
We have to fill in the panel called 'AddSymbolForm' in the following way:

Let's select radio button Reference,
Symbolic Text be: B_LUB, lubrication button
Base: N_MSG,
Offset: 0,
Bit: 22 (23. button = 22-bit).
Symbolic Text be: L_LUB, lubrication button lamp
Base: P_MSG,
Offset: 0,
Bit: 22 (23. button = 22-bit).

7.10 Position Switches

32 position switches can be specified on the parameters.
Parameter

N1100+n SWn Axis Number

tells, for which axis shall we set the nth switch.
Parameter

N1132+n SWn Min Pos n

determines the machine position of the end of the nth switch falling in a negative direction, on the given axis.
Parameter

N1164+n SWn Max Pos n

determines the machine position of the end of the nth switch falling in a positive direction, on the given axis.
Before all **PLC cycles** the control checks whether the nth switch does exist or not. If yes, based on the absolute position of the axis, appointed to the switch, calculated from its ***encoder*** it will handle the below signals for the nth switch (n-1 indexed PLC flag).
The signals of the position switches are input signals for the PLC.

Bit variables of the position switches:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_SW	The axis is on the 1st switch		
N_SWN	The axis is in a negative direction from the 1st switch		
N_SWP	The axis is in a positive direction from the 1st switch		

Inputs of the position switches

N_SW: The axis is on the 1st switch

If the highlighted axis is on switch 1, i.e.

N1133 SW1 Min Pos 1 < Position < N1165 SW1 Max Pos 1

the flag becomes true.

N_SWN: The axis is in a negative direction from the 1st switch

If the highlighted axis is in a negative direction from switch 1, i.e.

Position < N1133 SW1 Min Pos 1

the flag becomes true.

N_SWP: The axis is in a positive direction from the 1st switch

If the highlighted axis is in a positive direction from switch 1, i.e.

Position > N1133 SW1 Max Pos 1

the flag becomes true.

The other switches are accessible by indexing referenced to the address of switch 1. We can add symbolic names to switches, too; but we shall pay attention to taking up the names with a relative reference to symbols N_SW, ... etc. as bases.

☞ ***Attention**: During the application of switch signals you have to take into consideration the maximum movement speed of the axis, the length of the switch and PLC cycle time.*

DWORD variables of the position switches:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_SW0	The register of switch 1 determined on the Position Switches parameter (DWORD)		

N_SW0: The register of switch 1 determined on the Position Switches parameter (DWORD)

The signals of position switch 1 are accessible in the N_SW0 register also in a dword-format. The registers of other switches can be read by an indexed access.

7.11 Access to Parameter Group PLC Constants

In the parameter group PLC Constants, for the PLC program

64 pieces of bit-,

32 pieces of dword integer and

32 pieces of floating-point-type,

freely usable parameters are available. These parameters can be read by the PLC program directly from the PLC memory, and a contact can be defined for the bit parameters.
PLC parameters cannot be directly written out from the PLC program!

Bit-type PLC parameters:

Inputs		**Outputs**	
Symbol	**Description**	**Symbol**	**Description**
N_P00	PLCBits0 parameter's P00 bit		
N_P01	PLCBits0 parameter's P01 bit		
N_P02	PLCBits0 parameter's P02 bit		
N_P03	PLCBits0 parameter's P03 bit		
N_P04	PLCBits0 parameter's P04 bit		
N_P05	PLCBits0 parameter's P05 bit		
N_P06	PLCBits0 parameter's P06 bit		
N_P07	PLCBits0 parameter's P07 bit		
N_P10	PLCBits1 parameter's P10 bit		
N_P11	PLCBits1 parameter's P11 bit		
N_P12	PLCBits1 parameter's P12 bit		
N_P13	PLCBits1 parameter's P13 bit		
N_P14	PLCBits1 parameter's P14 bit		
N_P15	PLCBits1 parameter's P15 bit		
N_P16	PLCBits1 parameter's P16 bit		
N_P17	PLCBits1 parameter's P17 bit		
N_P20	PLCBits2 parameter's P20 bit		
N_P21	PLCBits2 parameter's P21 bit		
N_P22	PLCBits2 parameter's P22 bit		
N_P23	PLCBits2 parameter's P23 bit		
N_P24	PLCBits2 parameter's P24 bit		
N_P25	PLCBits2 parameter's P25 bit		
N_P26	PLCBits2 parameter's P26 bit		
N_P27	PLCBits2 parameter's P27 bit		
N_P30	PLCBits3 parameter's P30 bit		

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_P31	PLCBits3 parameter’s P31 bit		
N_P32	PLCBits3 parameter’s P32 bit		
N_P33	PLCBits3 parameter’s P33 bit		
N_P34	PLCBits3 parameter’s P34 bit		
N_P35	PLCBits3 parameter’s P35 bit		
N_P36	PLCBits3 parameter’s P36 bit		
N_P37	PLCBits3 parameter’s P37 bit		
N_P40	PLCBits4 parameter’s P40 bit		
N_P41	PLCBits4 parameter’s P41 bit		
N_P42	PLCBits4 parameter’s P42 bit		
N_P43	PLCBits4 parameter’s P43 bit		
N_P44	PLCBits4 parameter’s P44 bit		
N_P45	PLCBits4 parameter’s P45 bit		
N_P46	PLCBits4 parameter’s P46 bit		
N_P47	PLCBits4 parameter’s P47 bit		
N_P50	PLCBits5 parameter’s P50 bit		
N_P51	PLCBits5 parameter’s P51 bit		
N_P52	PLCBits5 parameter’s P52 bit		
N_P53	PLCBits5 parameter’s P53 bit		
N_P54	PLCBits5 parameter’s P54 bit		
N_P55	PLCBits5 parameter’s P55 bit		
N_P56	PLCBits5 parameter’s P56 bit		
N_P57	PLCBits5 parameter’s P57 bit		
N_P60	PLCBits6 parameter’s P60 bit		
N_P61	PLCBits6 parameter’s P61 bit		
N_P62	PLCBits6 parameter’s P62 bit		
N_P63	PLCBits6 parameter’s P63 bit		
N_P64	PLCBits6 parameter’s P64 bit		
N_P65	PLCBits6 parameter’s P65 bit		
N_P66	PLCBits6 parameter’s P66 bit		
N_P67	PLCBits6 parameter’s P67 bit		
N_P70	PLCBits7 parameter’s P70 bit		
N_P71	PLCBits7 parameter’s P71 bit		
N_P72	PLCBits7 parameter’s P72 bit		
N_P73	PLCBits7 parameter’s P73 bit		
N_P74	PLCBits7 parameter’s P74 bit		

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_P75	PLCBits7 parameter's P75 bit		
N_P76	PLCBits7 parameter's P76 bit		
N_P77	PLCBits7 parameter's P77 bit		

We can add also symbolic names to certain N_Pij parameters, but we have to ensure that we enter the names with a relative reference to N_Pij symbols as the basis.

DWORD-type PLC parameters are:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_PDW1	PLC DWord1 parameter value (DWORD)		

N_PDW1: PLC DWord1 parameter value (DWORD)

From the N_PDW1 register the value of N1209 PLC DWord1 parameter can be read out. The values of other parameters can be accessed by an indexed command.

We can add also symbolic names to certain parameters, but we have to ensure that we enter the names with a relative reference to N_PDW1 symbols, as the basis.

The floating-point type PLC parameters are:

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_PDB1	PLC Double1 parameter value (double)		

N_PDB1: PLC Double1 parameter value (double)

From the N_PDB1 register the value of N1241 PLC Double1 parameter can be read out. The values of other parameters can be accessed by an indexed reference.

☞ ***Attention!*** *In case of an indexed reference we have to take into consideration that the double value is represented on 2 DWORDs, therefore the index has to be stepped two-by-two.*

We can add also symbolic names to certain parameters, but we have to ensure that we enter the names with a relative reference to N_PDB1 symbols, as the basis.

7.12 Common Variables

The common variables are such variables either going from NC to PLC, or from the PLC to NC, which are ***common for all channels***.
Variables starting with a prefix

N: go from NC to PLC, while variables starting with

P: go from PLC to NC.

7.12.1 Bit-type Common Variables

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_P2MS	Clock signal with 2 Time Slice period (only in Int0 module, with a direct query)	**P_MONREQ**	Machine switch-on request from the PLC
N_P2T	Clock signal with 2 PLC cycles' period	**P_HOLD0**	Feed-hold in all channels
N_P100MS	Clock signal with 100 msec period	**P_SHTDNREQ**	NC shutdown request
N_P1S	Clock signal with 1 sec period		
N_P1M	Clock signal with 1 min period		
N_ON	always 1 (true)		
N_OFF	always 0 (false)		
N_B7	maintained		
N_NVRAMOK	The non-volatile PLC variables have been successfully reloaded		
N_FIRSTCC	First PLC cycle after power on		
N_NCREADY	NC is ready for operation		
N_MONST	Machine ON signal status		
N_MONDIS	Machine ON signal's switch-on is disabled		
N_CLRMSG	Clear message		
N_MSGA	Message on the screen		
N_MSG0	PLC message on the screen		
N_MSG1	Axis or spindle message on the screen		
N_MSG2	Channel block preprocessor message on the screen		
N_MSG3	Channel execution message on the screen		

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_MSG4	Macro error (#3000) on the screen		
N_MSG5	Macro message (#3006) on the screen		
N_MSG6	maintained		
N_MSG7	maintained		
N_MSG8	Real-time system message on the screen		
N_MSG9	Human-machine interface message on the screen		
N_TLSRCH	Tool search in progress		
N_TLMD	Tool tables under modification		
N_TLSV	Saving of tool tables is in progress		
N_TLEDT	Editing of tool tables is in progress		
N_PTEDT	The pallet management table is under editionA palettakezelő táblázat szerkesztés alatt		
N_SIMU	The NC software runs on simulator (PC)		
N_NVECAT	The non-volatile variables are accessible through EtherCAT		
		P_DIR	Access forbidden: Library operations
		P_PRGE	Access forbidden: Program edition
		P_WOFFS	Access forbidden: Workpiece offsets
		P_COMPG	Access forbidden: Geometry compensations
		P_COMPW	Access forbidden: Wear compensations
		P_TLTAB	Access forbidden: Tool tables
		P_MAC	Access forbidden: Macro variables
		P_TRCTR	Access forbidden: Timers and workpiece counters
		P_RUNAUT	Access forbidden: Run in Auto
		P_RUNMDI	Access forbidden: Run in MDI

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
		P_PAR	Access forbidden: Parameters
		P_PLC	Access forbidden: PLC program
		P_SVRC	Access forbidden: Service operations
		P_PTTAB	Access forbidden: Pallet table

Bit-type common variables going from NC to PLC

N_P2MS: Clock signal with 2 Time Slice period (only in Int0 module, with a direct query)
N_P2T: Clock signal with 2 PLC cycles' period
N_P100MS: Clock signal with 100 msec period
N_P1S: Clock signal with 1 sec period
N_P1M: Clock signal with 1 min period

From among the clock signals handed over by NC it is worth to use the signal N_P2MS due to the signal's frequency only in the Int0 module. The other clock signals can be used in the Main program, too.

N_ON: always 1 (true)
N_OFF: always 0 (false)

We can use the two symbols if, for example, to the input of an instruction box a fix value has to be connected, or if we would like to comment out a rung.

N_NVRAMOK: The non-volatile PLC variables have been successfully reloaded

In case we store data in the non-volatile PLC RAM, in the first cycle after power on (N_FIRSTCC=1) we can examine the flag. In case the value of the flag is 0, the PLC variables have been damaged.

N_FIRSTCC: First PLC cycle after power on

The flag after power on or after the restart of the control is 1 for the time of 1 PLC cycle. During this time the PLC program has to carry out the necessary initializations.

N_NCREADY: NC is ready for operation

It indicates the ready state of the NC.

N_MONST: Machine ON signal status

In case the flag turns to 0 in the switched-on state of the machine, an emergency stop has to be initiated from the PLC.

N_MONDIS: Machine ON signal’s switch-on is disabled

We can initiate the switch-on of the machine, i.e. to switch on the P_MONREQ flag only if the switch-on of the machine is not forbidden, i.e. N_MONDIS flag 0.

N_CLRMSG: Clear message

The message appears in the top upper row of the display.
If there are several messages waiting at the same time, always the latest received message will be displayed in the message field. (Last in First Out)
The button

CANCEL

will delete the last received message which can be seen on the display, and it will set the N_CLRMSG flag to 1.
In case we click on the message field, in a drop-down list all messages can be seen.
We can select any of the messages and by clicking on the

CLEAR (selected)

function button we can clear it. Then (also without pushing the CANCEL button) the N_CLRMSG flag will operate.
We can select also the function button

Clear all

which clears all messages being in the buffer. Flag N_CLRMSG will operate this time, too.
In the PLC program we always have to use the N_CLRMSG flag for clearing the message.

N_MSGA: Message on the screen

The value of the flag is 1, if there is any kind of message in the upper top message row of the screen. The flag may be used for, e.g. switching on an alarm lamp on the machine.

N_MSG0: PLC message on the screen
N_MSG1: Axis or spindle message on the screen
N_MSG2: Channel block preprocessor message on the screen
N_MSG3: Channel execution message on the screen
N_MSG4: Macro error (#3000) on the screen
N_MSG5: Macro message (#3006) on the screen
N_MSG6: Maintained
N_MSG7: Maintained
N_MSG8: Real-time system message on the screen
N_MSG9: Human-machine interface message on the screen

The N_MSG0, ..., N_MSG9 flags indicate the type of the last received message which can be seen in the top upper message row of the screen.

N_TLSRCH: Tool search in progress
N_TLMD: Tool tables under modification
N_TLSV: Saving of tool tables is in progress
N_TLEDT: Editing of tool tables is in progress

The above flags provide information on the reasons why the tool tables are occupied.

N_PTEDT: The pallet management table is under edition

A row of the pallet management table is being edited.

N_SIMU: The NC software runs on simulator (PC)

If the NC software runs on simulator, i.e. on PC, the PLC will be informed about this fact based on the status 1 of the N_SIMU flag.

N_NVECAT: The non-volatile variables are accessible through EtherCAT

If there is no battery built in the control, but there is a FRAM unit, operating through EtherCAT connected to it, then N_NVECAT=1. In such a case the PLC memory ranging from the PLCNVRAM to the PLCRAM will not be saved during the switch-off.

The PLC programmer shall ensure to save PLC variables, respectively, to read them, by using commands MR10 and MW11.

Bit-type common variables going from PLC to NC

P_MONREQ: Machine switch-on request from the PLC

The MON (Machine ON) output carrying on the switch-on of the machine, can be found on every EtherCAT head units.

We can carry out the setting of the address of MON output in the ***Service menu*** of the control, after exchanging the window ***ECAT settings*** at the settings of the ***machine control panel***. Select the EtherCAT-Head unit on the left-side panel and by clicking on the ***Setting*** tab, we can set the ***address*** of the MON output. In case we select the option

Not used

on a head unit, on that unit MON output will not be handled. If we set the address

1, 2, etc.

it will handle the MON output on it.

The switch-on of the P_MONREQ flag switches on the MON output on all head units which have not been set to Not used. It is enabled to switch on the flag only if the switch-on of the machine is not forbidden, i.e. N_MONDIS flag 0.

P_HOLD0: Feed-hold in all channels

The switching of the flag into 1 stops the movement of all axes in all channels even if - the override and the stop is forbidden. In case the override and stop are forbidden, during tapping or thread cutting the PLC has to stop the spindle.

P_SHTDNREQ: NC shutdown request

The NC can be mounted also with a battery unit, which is able to provide the power supply of the control and the necessary EtherCAT electronics for 1-2 minutes. In case of a network outage or the switch-off of the main switch of the machine the battery unit issues a signal to the PLC through an interface input. The PLC program can request the shutdown of the system by switching on the P_SHTDNREQ flag.

P_DIR: Access forbidden: Library operations

In P_DIR=1 status, files and folders

cannot be deleted,

cannot be overwritten.

Creating new files, respectively, copying files are allowed.

P_PRGE: Access forbidden: Program edition

In P_PRGE=1 state the edition of all part programs is forbidden.

P_WOFFS: Access forbidden: Workpiece offsets

In P_WOFFS=1 state the workpiece zero point offsets cannot be overwritten, neither in the offset table nor by the measurement of offset.

P_COMPG: Access forbidden: Geometry compensations

In P_COMPG=1 state geometry compensation values cannot be overwritten, neither in the tool offset table nor by the measurement of tool-offset.

P_COMPW: Access forbidden: Wear compensations

In P_COMPW=1 state wear compensation values cannot be overwritten, neither in the tool offset table nor by the measurement of tool-offset.

P_TLTAB: Access forbidden: Tool tables

In P_TLTAB=1 state any element of the tool management table cannot be overwritten.

P_MAC: Access forbidden: Macro variables

In P_MAC=1 state any local, or common macro variable cannot be overwritten.

P_TRCTR: Access forbidden: Timers, workpiece counters

In P_TRCTR=1 state the timers and workpiece counters cannot be overwritten.

P_RUNAUT: Access forbidden: Run in Auto

In P_RUNAUT=1 state

programs to run in automatic mode cannot be appointed,
cannot be deleted any program to run appointed for an automatic run.

P_RUNMDI: Access forbidden: Run in MDI

In P_RUNMDI=1 state

programs to run in MDI mode cannot be appointed,
cannot be deleted any program to run appointed for an MDI run.

P_PAR: Access forbidden: Parameters

In P_PAR=1 state

parameters cannot be edited,
parameters cannot be imported,

but parameters can be saved or exported.

P_PLC: Access forbidden: PLC program

In P_PLC=1 state

PLC program cannot be edited,
PLC programs cannot be imported,

but PLC program can be exported or you can watch the green flow.

P_SVRC: Access forbidden: Service operations

The state P_SVRC=1 influences the service operations in the following way:

I/O test: BitSet, BitReset, HexaSet operations are forbidden,
Symb. I/O, Logic analyser: overwriting is forbidden during saving
ECAT settings: the editing of panels is forbidden.

P_PTTAB: Acces forbidden: Pallet table

When its status is P_PTTAB=1, the pallet table cannot be edited.

7.12.2 DWORD-type Common Variables

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
N_ACTMSG	The identification number of the active message on the display (DWORD)	**P_CHSEL**	Which channel is the keyboard assigned to (0,1,2...) (DWORD)

DWORD-type common variables going from NC to PLC

N_ACTMSG: The identification number of the active message on the display (DWORD)
In the small window situated left in the upper message row we can read the 8 decimal digit code of the last message appearing on the display. In the N_ACTMSG register this code will appear.

DWORD-type common variables going from PLC to NC

P_CHSEL: Which channel is the keyboard assigned to (0,1,2...) (DWORD)
It has a meaning on a multi-channel control. During opening a screen, for example, in the Offsets menu selecting the option Measurement by clicking the item, the control will first load the Measurement screen of the channel the number of which has been set in the P_CHSEL register. After that we can move between the various channels by pushing the PgUp, PgDn buttons.
It is not necessary to use it on a single-channel control.

7.13 Axis Control Variables

The axis control variables are such variables going from NC to PLC or from PLC to NC which become indexed per axis. All symbols published here refer to the first axis (indexed as 0). The appropriate variables of other axes can be accessed by an indexed addressing. The control handles a maximum of 32 axes.
Variables starting with

AN go from NC to PLC (Inputs),

while variables starting with a prefix

AP go from PLC to NC (Outputs).

7.13.1 Bit-type Axis Control Variables

Inputs		**Outputs**	
Symbol	**Description**	**Symbol**	**Description**
AN_DETCHA	Axis detach acknowledge	**AP_DETCHR**	Axis detach request
AN_OPNA	Position control loop open acknowledge	**AP_OPNR**	Position control loop open request
AN_INPOS	Axis in position	**AP_END**	Encoder error monitoring disable
AN_AXALM	Alarm state on the axis	**AP_FLWU**	Follow up on
AN_RAPR	Rapid motion request	**AP_JOGP**	Jog in +direction request
AN_MTNRP	Motion request in positive direction	**AP_JOGN**	Jog in -direction request
AN_MTNRN	Motion request in negative direction	**AP_DECSW**	Axis on deceleration switch
AN_LUBR	Lubrication request on the axis	**AP_RAPD**	Rapid motion disable
AN_RPE	Reference point established	**AP_MTNDP**	Motion disable in positive direction
AN_REFEND	End of travel to reference point	**AP_MTNDN**	Motion disable in negative direction
AN_OTP	Axis on over-travel position in positive direction	**AP_LIMP**	Axis on limit switch in positive direction
AN_OTN	Axis on over-travel position in negative direction	**AP_LIMN**	Axis on limit switch in negative direction
AN_REFP1	Axis on reference point 1	**AP_LIMSELP**	Selection of 1B Positive limit range
AN_REFP2	Axis on reference point 2	**AP_LIMSELN**	Selection of 1B negative limit range
AN_REFP3	Axis on reference point 3-on	**AP_LCK**	Axis lock
AN_REFP4	Axis on reference point 4-en	**AP_DISPD**	Display disable
AN_PARKA	Parking acknowledge	**AP_PARKR**	Parking request

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
AN_SYNCA	Synchronous control acknowledge	**AP_SYNCR**	Synchronous control request
AN_MIXA	Axis exchange acknowledge	**AP_MIXR**	Axis exchange request
AN_SPRPNA	Superimposed control acknowledge	**AP_SPRPNR**	Superimposed control request
AN_EGBS	EGB slave axis	**AP_DIARAD**	Axis programmed in diameter
AN_INDP	Axis in index position	**AP_SSLOP**	Slave axis loop opening and takeover of only the command signal
AN_MIXM	Master axis of axis exchange	**AP_FEEDD**	Motion with feed-rate disable
AN_SYNCM	Master axis of synchronous control	**AP_PLCR**	PLC axis control request
AN_SPRPNM	Master axis of superimposed control	**AP_GOR**	PLC axis go request
AN_PLCA	PLC axis control acknowledge	**AP_RES**	PLC axis reset
AN_BEPTY	PLC axis block buffer empty	**AP_EFD**	Position measurement disable
AN_GOA	PLC axis go acknowledge	**AP_RPE**	There is a valid reference point
AN_IEPTY	PLC axis movement done	**AP_MIRR**	Request for axis direction exchange (mirroring)
AN_MIRA	Acknowledge of the axis direction exchange (mirroring) requested by the PLC		

Bit-type axis variables going from NC to PLC

AN_DETCHA: Axis detach acknowledge

AN_DETCHA signal is the acknowledge flag of the AP_DETCHR axis detach request signal.
Upon the axis detach request AP_DETCHR=1 the axis control gives back AN_DETCHA=1. Upon the attach request AP_DETCHR=0 the axis control gives back AN_DETCHA=0. See also: AP_DETCHR flag description.

AN_OPNA: Position control loop open acknowledge

AN_OPNA is the acknowledge flag of AP_OPNR position control loop open request signal.
Upon position control loop open request AP_OPNR=1 the axis control gives back AN_OPNA=1. Upon the AP_OPNR=0 position control loop close request the axis control gives back AN_OPNA=0. See also: AP_OPNR flag description.

AN_INPOS: Axis in position

If the difference of the command position of the axis and its position measured from the encoder stays within the window determined by the N0516 Inpos parameter, the value of the signal is 1.

AN_AXALM: Alarm state on the axis

If the axis control recognizes a servo error on any element of the servo loop (encoder, drive operating through EtherCAT, position control loop), it will set the AN_AXALM flag to 1.

☞ ***Attention!*** *The task of the PLC program is to react to the error signal, to stop the drive, and to create an emergency status! The NC does not switch off automatically the machine, only upon the command of the PLC!*

AN_RAPR: Rapid motion request

The axis control handles the flag together with motion request flags AN_MTNRP, AN_MTNRN.
If AN_RAPR=0, the interpolator would like to move with feed-rate and it waits for the status AP_FEEDD=0 (motion with feed-rate enable) and AP_RAPD=1 (rapid motion disable).
If AN_RAPR=1, the interpolator would like to move with rapid traverse and it waits for the status AP_FEEDD=1 (motion with feed-rate disable) and AP_RAPD=0 (rapid motion enable).
See also AN_MTNRP, AN_MTNRN, AP_RAPD, AP_FEEDD flags.
This flag can be used for example for gear range change between feed-rate and rapid traverse movements.

AN_MTNRP: Motion request in positive direction
AN_MTNRN: Motion request in negative direction

Before the start of any movement the interpolator sets the motion request flag with the appropriate direction. In case of circular interpolation, on the axis falling into the plain of the circle, it requests motion in both directions.
The PLC enables the movement in the appropriate direction, by resetting flags AP_MTNDP=0 or AP_MTNDN=0.
See also: AN_RAPR, AP_RAPD, AP_FEEDD flags.
The flags can be used e.g. for clamping and unclamping axes.

AN_LUBR: Lubrication request on the axis

In case the value of the N1273 Lubrication Distance parameter is not 0, and on the given axis the amount of displacement has exceeded the distance written on the parameter, AN_LUBR flag will be set for 1 PLC cycle.
Upon the signal e.g. a lubrication pump can be started.

AN_RPE: Reference point established

In the status of flag AN_RPE=1 on the axis the travel to the reference point has already been made. In case of absolute encoders the flag is always 1.

AN_REFEND: End of travel to reference point

The flag is operating in reference point mode. It will be set on the given axis, if the travel to the reference point has been carried out and the axis does not move any more.

AN_OTP: Axis on over-travel position in positive direction
AN_OTN: Axis on over-travel position in negative direction

If the axis has run to a stroke limit set in the Axis Limits parameter group, or it has run onto a limit switch, the flag of the appropriate direction will be set.

AN_REFP1: Axis on reference point 1

| Reference point – machine position measured from the encoder| < N0516 Inpos
If the difference of the machine position of the axis measured from the encoder and the value of N0200 Reference Position1 parameter is within the range determined by the N0516 Inpos parameter, the value of the signal will be 1.

AN_REFP2: Axis on reference point 2
|N0201 Reference position 2 – machine position measured from the encoder| < N0516 Inpos

If the difference of the machine position of the axis measured from the encoder and the value of N0201 Reference Position2 parameter is within the range determined by the N0516 Inpos parameter, the value of the signal will be 1.

AN_REFP3: Axis on reference point 3
|N0202 Reference Position3 – machine position measured from the encoder| < N0516 Inpos

If the difference of the machine position of the axis measured from the encoder and the value of N0202 Reference Position3 parameter is within the range determined by the N0516 Inpos parameter, the value of the signal will be 1.

AN_REFP4: Axis on reference point 4
|N0203 Reference Position4 – machine position measured from the encoder| < N0516 Inpos

If the difference of the machine position of the axis measured from the encoder and the value of N0203 Reference Position4 parameter is within the range determined by the N0516 Inpos parameter, the value of the signal will be 1.

Positions belonging to signals AN_ REFP2, AN_ REFP3, AN_ REFP4 are measured positions of exchanges. Before the PLC carries out an activity for which an exchange position of an axis is necessary, by examining the signal, the PLC can check whether the axis is in a right position or not.

AN_PARKA: Parking acknowledge

AN_PARKA is the acknowledge flag of the AP_PARKR parking request signal. Upon parking request AP_PARKR=1 the axis control will give back AN_PARKA=1. Upon parking cancel AP_PARKR=0 the axis control will give back the signal AN_PARKA=0. See also: AP_PARKR flag description.

AN_SYNCA: Synchronous control acknowledge

AN_SYNCA is the acknowledge flag of the AP_SYNCR synchronous control request signal. It gives back the signal on the flag belonging to the axis (***slave***) requesting synchronous control.

Upon the synchronous control request AP_SYNCR=1, the axis control will give back AN_SYNCA=1. Upon synchronous control cancel AP_SYNCR=0 the axis control will give back the signal AN_SYNCA=0. See also: AP_SYNCR flag description.

AN_MIXA: Axis exchange acknowledge

AN_MIXA is the acknowledge flag of the AP_MIXR axis exchange request signal. It gives back the signal on the flag belonging to the axis (***slave***) requesting the exchange. Upon the axis exchange request AP_MIXR=1 the axis control gives back AN_MIXA=1. Upon axis exchange cancel AP_MIXR=0 the axis control gives back the signal AN_MIXA=0. See also: AP_MIXR flag description.

AN_SPRPNA: Superimposed control acknowledge

AN_SPRPNA is the acknowledge flag of the AP_SPRPNR superimposed control request signal. It gives back the signal on the flag belonging to the axis (***slave***) requesting superimposed control.

Upon superimposed control request AP_SPRPNR=1 the axis control gives back AN_SPRPNA=1. Upon superimposed control cancel AP_SPRPNR=0 the axis control gives back the signal AN_SPRPNA=0. See also: AP_SPRPNR flag description.

AN_EGBS: Axis EGB slave

The flag switches to 1,

In case the given axis is appointed on the N1802 EGB Slave parameter as an electronic gear box slave axis, and

G81.8 EGB (hobbing) command is valid.

AN_INDP: Axis in index position

The value of the flag is 1, if the difference of the axis position and its index position is within the range determined by the N0516 Inpos parameter.

Under an index position we mean positions being on an axis in discrete, equally spaced distances from each other, and which are special from a particular aspect. E.g.: a Hirth-crowned rotary table, which can be clamped and indexed every 5 degrees.

At the N0106 Axis Properties parameter, by a setting #3 IDX=1 we can appoint an axis an indexed axis. On the N0110 Indexing Amount parameter we can set the indexation distance. E.g.: in case of the above, Hirth-crowned rotary table, it can be 5.

AN_MIXM: Master axis of axis exchange

If the axis is the ***master*** axis of axis exchange, the flag is set.

The flag AN_MIXA is set to the slave axis. The two axis indices are different!

AN_SYNCM: Master axis of synchronous control

If the axis is the ***master*** axis of synchronous control, the flag is set.

The flag AN_SYNCA is set to the slave axis. The two axis indices are different!

AN_SPRPNM: Master axis of superimposed control

If the axis is the ***master*** axis of superimposed control, the flag is set.

The flag AN_SPRPNMA is set to the slave axis. The two axis indices are different!

AN_PLCA: PLC axis control acknowledge

AN_PLCA is the acknowledge flag of the AP_PLCR PLC axis control request signal. Upon PLC axis control request AP_PLCR=1 the axis control will give back AN_PLCA=1. Upon PLC axis control cancel AP_PLCR=0, the axis control will give back the signal AN_PLCA=0. See also: AP_PLCR flag description.

AN_BEPTY: PLC axis block buffer empty

The MOVCMD PLC axis control instructions have a single-block buffer per axis. This means that if the interpolator has taken out a block from the buffer for execution, the PLC may load the next block. See the chapter 6.15 named Axis Control Instruction: MOVCMD. In fact the command MOVCMD will not be executed either ways until it can load the block to the buffer.

If AN_BEPTY=1 the block buffer is empty.

AN_GOA: PLC axis go acknowledge

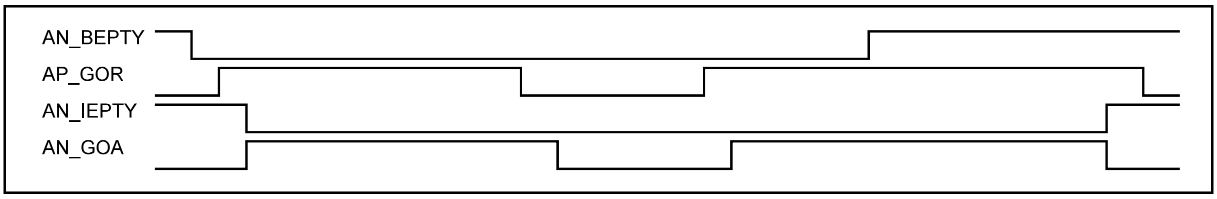
If the interpolator of PLC axis is not empty, i.e. AN_IEPTY=0, **AND** upon the status PLC axis go request AP_GOR=1, the axis will start the movement and the flag will take up the status AN_GOA=1.

If the interpolator of PLC axis is empty, i.e. AN_IEPTY=1, **OR** upon the status PLC axis stop AP_GOR=0, the axis will stop and the flag will take up the status AN_GOA=0.

See the AN_GOR flag.

AN_IEPTY: PLC axis movement done

If the PLC axis movement has done, i.e. the interpolator is empty, the flag will take up state 1.



AN_MIRA: Acknowledge of the axis direction exchange (mirroring) requested by the PLC

The PLC can request the axis direction exchange on an axis through the AP_MIRR flag. The AP_MIRA flag is the acknowledge signal of the request.

In this case, when the axis movement is controlled from the part program, the motion direction of the given axis becomes opposite.

When the axis movement is controlled manually (jog buttons, handwheel), the motion direction does not change.

If the value of the flag is:

=0: the axis will move in accordance with the part program in the original direction set in parameter;

=1: the axis will move in accordance with the part program in the opposite direction to that set in parameter.

☞ ***Warning!*** *The axis direction exchange should always be carried out in the function preventing the block buffering (see the Program parameter group).*

Bit-type axis variables going from PLC to NC

AP_DETCHR: Axis detach request

In case the operation of an axis needs to be detached, PLC can request this from the system by setting the AP_DETCHR flag of the appropriate axis. After the axis control has stopped the operation of the axis, it will give back the acknowledge signal AN_DETCHA=1. During the cancellation of detach, the AP_DETCHR flag has to be reset and PLC has to wait till the axis control gives back the status AN_DETCHA=0.
In case the AN_DETCHA signal is in status 1, the axis control is out of operation:

it will clear on the axis the record “reference point established”: AN_RPE=0,
it will not measure the position from the encoder,
it will not close the position control loop,
it will not issue any signals towards the drive of the axis,
in the part program and PLC no reference can be made to the axis,

the axis does not exist.
In an opposite case, if the AP_DETCHR=0 and the NC switches off the AN_DETCHA=0 flag, the axis will exist again.
An axis exists if the

value on the N0002 Axis Assign parameter is > 0, and
AN_DETCHA=0.

☞ ***Attention!*** *The detach and cancellation of detach of the axis shall always be carried out in functions* ***suppressing the buffering of blocks*** *(see Program parameter group)*

Example:

The spindle of a lathe need to be operated as axis C (rotary table), then it shall be used as a spindle. Let’s say the

number of axis C is 3 (N0002 Axis Assign A3=1, N0100 Axis Name1 A3=C)
and the number of spindle S1 is 1. (N0605 Spindle Name2 S1=1)

After the power-on, it will work as a spindle:

SP_SDETCHR,#0=0, SN_SDETCHA,#0=0
AP_DETCHR,#2=1, AN_DETCHA,#2=1

If we would like to transform spindle S1 to axis C, we have to program an appropriate buffer-suppressive function M and we have to set and reset the flags in the following way:

SP_SDETCHR,#0=1, SN_SDETCHA,#0=1
AP_DETCHR,#2=0, AN_DETCHA,#2=0

From this on, the encoder signals arriving from the same spindle drive will not be received by the S1 spindle control but the C axis control. The command signal will not be issued by the S1 spindle control but by the C axis control to the spindle drive. Axis C can be moved, and it is possible to refer to address C in a part program.
The transformation of axis C back to spindle S1 is carried out in the same way.

AP_OPNR: Position control loop open request

If the position control loop of an axis needs to be opened, we have to request this from the system by setting the AP_OPNR flag of the appropriate axis. After the position control has stopped the closure of the loop, it will give back the acknowledge signal AN_OPNA=1. At the switch-back the AP_OPNR flag has to be reset, and we have to wait till the position control gives back the status AN_OPNA=0. Towards the drive the command signal 0 is issued.

The fact whether the closure of the position control loop is carried out with or without follow-up, is determined by the #5 FUP bit of the N0514 Servo Control parameter together with the AP_FLWU flags. (See also: AP_FLWU signal description).

☞ ***Attention!*** *The opening and closing of the position control loop has always to be carried out in functions* ***suppressing the buffering of the blocks*** *(see: Program parameter group).*

AP_END: Encoder error monitoring disable

If we set the flag, the servo control will not monitor the errors of the encoder (it will not issue any Encoder error signals).

AP_FLWU: Follow up on

It regulates the way of closing the position control loop:

In case the

N0514 Servo Control parameter #5 FUP=0 **AND** the PLC flag AP_FLWU=0

before closing the position control loop the ***command position will take up*** the value of the absolute position (measured from the encoder) and on the axis there ***will not be any displacement***. The machine will remain in the displaced position until we program an absolute movement.

In case the

N0514 Servo Control parameter #5 FUP=1 **OR** the PLC flag AP_FLWU=1 before closing the position control loop the ***command position will not take up*** the value of the absolute position (measured from the encoder) and on the axis there ***will be a displacement***, and it will step the displacement accumulated in the open status of the servo.

AP_JOGP: Jog in +direction request
AP_JOGN: Jog in -direction request

In case either in Jog, or Incremental jog mode we set the flag, the axis will move in the appropriate (+/-) direction, and if we reset the flag, the axis will stop.
MB_JOGn buttons can be linked to these signals.
In case of Reference point travel mode, we can set any of AP_JOGP, or AP_JOGN flags, the axis will not move in the direction selected, however in the direction determined by the parameters N0901 Reference Distance and N0900 Reference Type #5 DIR till any of the flags is set; and it will stop if neither of the flags is set, or if the travel to the reference point has been carried out.

AP_DECSW: Axis on deceleration switch

If on the axis the parameter state N0900 Reference Type #0 SWT=1 is set, the travel to the reference point is carried out by running on a switch.
In case the value of the AP_DECSW flag is 1, the axis has run on the reference point switch.

AP_RAPD: Rapid motion disable

It is the response signal issued for the AN_RAPR rapid motion request.
If AN_RAPR=1, the rapid motion will start if the PLC has reset the signal AP_RAPD=0.
If AP_RAPD=1 the rapid motion will not take place.
The flag shall be handled together with motion disable flags AP_MTNDP and AP_MTNDN.
This flag can be used for example for gear range change between feed-rate and rapid traverse movements.

AP_MTNDP: Motion disable in positive direction
AP_MTNDN: Motion disable in negative direction

These are the response signals issued for the AN_MTNRP and AN_MTNRN motion request flags.
Before starting any of the movements, the interpolator will set the motion request flag of the appropriate direction (AN_MTNRP=1, AN_MTNDN=1). In case of circular interpolation, it will request motion on both axis falling in the plane of the circle, in both directions.
The axis will not move till the PLC enables the movement in the appropriate direction, by the flag status AP_MTNDP=0, or AP_MTNDN=0.
The axis will stop as soon as the PLC sets the motion disable flag in the appropriate direction.
See also the flags AN_RAPR, AP_RAPD, AP_FEEDD.
The flags may be used for example for the clamping and unclamping of axes.

AP_LIMP: Axis on limit switch in positive direction
AP_LIMN: Axis on limit switch in negative direction

In case in any of the directions, the axis has run onto the limit switch, the appropriate flag shall be set by the PLC: AP_LIMP=1, or AP_LIMN=1. Afer setting the flag, the interpolator will stop by decelerating and it will be possible to start it only in the other direction. The deceleration distance necessary for stopping shall be taken into consideration during the setting of the switch.

AP_LIMSELP: Selection of 1B positive limit range
AP_LIMSELN: Selection of 1B negative limit range

It is the selection of the stroke limit specified on parameters per axis and direction from PLC.
In case the

N1001 StrkCont parameter #4 ABA=0

in the given channel, the PLC flags doing the parameter selection per axis and direction are effective: AP_LIMSELP in positive direction AP_LIMSELN in negative direction, and it will select from ranges 1A and 1B.
In positive direction it will select the

AP_LIMSELP=0: N1002 Range1A Positive parameter,
AP_LIMSELP=1: N1004 Range1B Positive parameter.

In negative direction it will select the

AP_LIMSELN=0: N1003 Range1A Negative parameter,
AP_LIMSELN=1: N1005 Range1B Negative parameter.

Changes made to the stoke limit selection shall be made in the standstill position of axes.

☞ ***See also:*** *CP_LIMSEL flag.*

AP_LCK: Axis lock

In AP_LCK=1 state the interpolator will not issue any movement command toward the selected axis. The movement however can be seen on the position display.
By resetting the flag AP_LCK=0, the interpolator will update the positions from the measuring system and it will be possible to move the axis again.
It is allowed to change the flag only if no part programs are running and the axis is in a standstill!

AP_DISPD: Display disable

If an N0002 Axis Assign parameter of an axis is not 0 and we have given a name to it on N0100 Axis Name1, ... parameters, the axis will be automatically displayed on the position display.

If the flag is set AP_DISPD=1, the given axis will disappear from the position display. For example, if we detach an axis by setting the AP_DETCHR flag, its position display can be cancelled, too.

AP_PARKR: Parking request

Parking may be requested only for an axis participating in a synchronous control, i.e. if the axis is either

AN_SYNCA=1: slave axis of synchronous control or

AN_SYNCM=1: master axis of synchronous control.

The parking is initiated by the axis to be parked by setting the appropriate AP_PARKR parking request flag. The parking is acknowledged if the axis control has set the AN_PARKA acknowledge signal. During the cancellation of parking the AP_PARKR signal shall be reset, and one shall wait till the axis control returns the AN_PARKA=0 state.

Under parking we mean the situation where a part program is written with a synchronous operation, but we do not want to move either the synchronous slave or the synchronous master axis, as, for instance, on that side there is not workpiece. In such a case either by a switch on the operator panel or by an M function we can switch on the parking on the appropriate axis. See also the AP_SYNCR signal.

☞ ***Attention!*** *The switch-on and switch-off of parking needs always to be carried out in functions suppressing buffering of blocks (see the program parameter group). The acknowledge signal is not going to be issued till the buffer is not empty. It is permitted to operate the parking by a switch only if no program is running and the axis is in rest.*

AP_SYNCR: Synchronous control request

The synchronous control is initiated by the synchronous slave axis, by setting its AP_SYNCR synchronization request flag. The synchronous connection is created when the axis control sets the AN_SYNCA acknowledge. At the cancellation of the synchronous control the AP_SYNCR signal shall be reset and wait till the axis control gives back the AN_SYNCA=0 state.

Synchronous machining is called when an axis, the synchronous slave is moved based on the movement commands of another axis, the synchronous master. In this state it is not possible to issue a movement command on the synchronous slave. From a part program, the synchronous control can be switched on, respectively off, e.g., by M functions.

We enter the number of the master axis of the synchronous slave in the N2101 Synchronous Master parameter to the axis number belonging to the synchronous slave.

If the parameter value is 0, the axis is not a synchronous slave. For example, if axis 4 is a synchronous slave and the number of its master axis is 2, the parameter shall be filled in the following way:

N2101 Synchronous Master A4=2

The displacement direction of the synchronous slave compared to the master is determined by the #0 MSY bit of the N2102 Synchron Config parameter.

☞ ***Attention!*** *The synchronous control is not to be confused with the position synchrony when e.g. on a gantry-type machine two motors are moving the same axis. Then the N2102 Synchron Config parameter #4 PSN=1, while during synchronous control PSN=0 is to be set.*

☞ ***Attention!*** *The switch-on and switch-off of the synchronous control needs always to be carried out in functions suppressing the buffering of blocks (see the program parameter group). The acknowledge signal is not going to be issued till the buffer is not empty.*

AP_MIXR: Axis exchange request

The axis exchange is initiated by the slave axis (to be exchanged) by setting the AP_MIXR exchange request flag. The exchange takes place if the axis control has set the AN_MIXA acknowledge signal. During the cancellation of the axis exchange the AP_MIXR signal has to be reset and wait till the axis control gives back the AN_MIXA=0 state.

In case of an axis exchange on the address of the slave axis (determined on parameters N0100 Axis Name1, ...) an axis with a different number, the master axis can be moved. Vice versa on the address of the master axis the slave axis is moving. From a part program, the axis exchange can be switched on, respectively off, e.g., by M functions.

The axis exchange refers only to the reference to the address of the axis from a part - program, and not for the jog and handwheel movements, because these refer to axis numbers, therefore these cases shall be handled from the PLC program!

The N2104 Composit Axis parameter, belonging to the slave axis will tell the number of the master axis with which the slave axis shall be exchanged. If the parameter value is 0, the axis is not to be exchanged. For example, if axis 4 is an axis to be exchanged and the number of its exchange axis is 1, the parameter shall be filled in in the following way:

N2104 Composit Axis A4=1

During an axis exchange both axes can work also in another channel (not in the one which has been assigned for it by the axis assign parameter), from there it receives the movement commands and from the other channel it receives the zero-point offset of the original axis. After the exchange, it is possible to move the axes in both channels, - except for the case if one of the axes is a hypothetical one. If an axis is appointed a hypothetical one in a given channel (N0106 Axis Properties #7 HYP=1), after the exchange, in the other channel or in the same channel it is not possible to refer to it. The fact whether the slave, respectively, the master axis shall move in the same, or in an opposite direction compared to the original one, is determined by the N2105 Composit Config #0 MMI parameter.

☞ ***Attention!*** *The switch-on and switch-off of the axis exchange shall always be carried out in functions suppressing buffering of blocks (see the Program parameter group). The acknowledge signal will not arrive till the time the buffer becomes empty.*

AP_SPRPNR: Superimposed control request

Superimposed control is initiated by the slave axis by setting AP_SPRPNR superimposed control request flag. The superimposed control will be established if the axis control has set the AN_SPRPNA acknowledge signal. During the cancellation of the superimposed control, the AP_SPRPNR signal shall be reset, and wait till the axis control gives back the state AN_SPRPNA=0.

We call superimposed machining if an axis, the superimposition slave moves upon two movement commands: The displacements received on its own address and the displacements received from the superimposition master become added and the movement of the slave will be the sum of the two displacements. The slave and the master axis may be in the same, but also in two different channels. From a part program, the superimposed control may be switched on, respectively, off, e.g., upon M functions.

We write the number of the master axis of the superimposition slave in the N2107 Superimposed Master parameter on the axis number belonging to the superimposition slave. If the parameter value is 0, the axis is not a super position slave. For example, if axis 4 is a superimposition slave and the number of its master axis is 2, the parameter shall be filled in the following way:

N2107 Superimposed Master A4=2

The direction of displacement of the superimposition slave, compared to the master, is determined by the N2108 Superimposed Config parameter's #0 MSU bit.

☞ ***Attention!*** *The switch-on and switch-off of the superimposed control shall always be carried out in functions suppressing the buffering of the blocks (see the Program parameter group). The acknowledge signal will not arrive till the time the buffer becomes empty.*

AP_DIARAD: Axis programmed in diameter

The flag state will be taken into account by the system if the value of the #5 MGD bit of the N0106 Axis Properties parameter is 0.

On a given axis, it regulates the way of data input and position display:

In case the

N0106 Axis Properties parameter #0 DIA=0 **AND** the PLC flag AP_DIARAD=0

on the given axis the data input and position display is carried out in **radius**.

In case the

N0106 Axis Properties parameter #0 DIA=1 **OR** the PLC flag AP_DIARAD=1

on the given axis the data input and position display is carried out in **diameter**.

The data input covers the part programs, compensations and zero-point offsets.

☞ ***Attention!*** *The switch-over from radius to diameter, or from diameter to radius shall always be carried out in functions suppressing the buffering of the blocks (see the Program parameter group).*

AP_SSLOP: Slave axis loop opening and takeover of only the command signal

If the axis **is not a synchronous slave**, its effect will be the same as that of the AP_OPNR signal.

If the axis is a **synchronous slave**, i.e. on the axis AN_SYNCA=1, the position control loop of the slave axis will be opened, the speed command signal of the master axis will be taken over and it will be issued to the slave's drive.

Example:

Both, the main-spindle and the sub-spindle are at the same time axes Z, i.e., both can be moved. If we would like to machine a long axis, we can support the workpiece by the sub-spindle. Let's appoint axis Z of the sub-spindle as the synchronous slave of axis Z of the main-spindle and let's connect the two movements (AP_SYNCR=1). After this, every time the chuck is closed on both the main and sub- spindle, in order to prevent the over-determinedness of the chuck, by the setting AP_SSLOP=1 we can open the position control loop on the slave axis. Axis Z of the main-spindle can be moved, the slave takes over the speed command signals from the master and they will move together. See also the DP_SILCK drive signal.

AP_FEEDD: Motion with feed-rate disable

It is the response signal issued for the AN_RAPR rapid motion request flag.

If AN_RAPR=0, the feed-rate movement will start if the PLC has reset the AP_FEEDD=0 signal.

If AP_FEEDD=1 the feed-rate movement will not start.

The flag shall be used together with AP_MTNDP, AP_MTNDN motion disable flags.

This flag can be used for example for gear range changes between feed-rate and rapid movements.

AP_PLCR: PLC axis control request

Any axis, determined on the N0002 Axis Assign parameter will be handled after power-on by the NC. PLC may request any axis from the NC and it may move it according the content of chapter 6.15 Axis Control Instruction: MOVCMD.
PLC may ask for the axis control by setting the AP_PLCR flag. One shall always wait till the axis control gives back the AN_PLCA=1 status. Vice versa, when the PLC cancels axis control, it will reset the AP_PLCR flag and it has to wait for the AN_PLCA=0 status.

☞ ***Attention!*** *The axis request and cancellation by the PLC shall always be carried out in functions suppressing the buffering of blocks (see the Program parameter group). The acknowledge signal will not arrive till the time the buffer becomes empty.*

AP_GOR: PLC axis go request

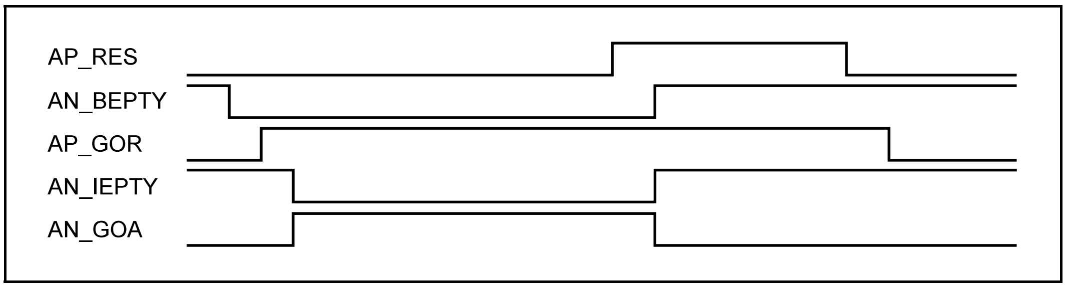
If the interpolator of the PLC axis is not empty, i.e. AN_IEPTY=0, the movement will start upon the AP_GOR=1 flag status, while it will stop upon AP_GOR=0. About the fact whether the movement goes or not, AN_GOA flag will inform. See also the AN_GOA flag.

AP_RES: PLC axis reset

In case the **AP_RES** signal goes to 1:

the PLC axis will stop with deceleration,
it will delete the command currently under execution,
it will delete the command in the buffer and it will set the following flags:

AN_BEPTY=1
AN_GOA=0
AN_IEPTY=1



AP_EFD: Distance measurement disable

The flag is effective only in the open state of the position control loop (AN_OPNA=1). If the value of the flag is 1, the axis control module will not update the axis position by the encoder pulses and will freeze the axis position value until the flag changes over to 0 again. Beginning from this, the pulses coming from the encoder will be counted from

the frozen position. This function can be used if the same encoder is used by several axes to measure distance.
The PLC flag is in connection with the #4 EFD bit of the N0514 Servo Control parameter.

AP_RPE: There is a valid reference point
If position control is not performed by the NC but by the drive, then the reference point return is also performed by the drive. After the read-out of the control bits of the drive, the PLC program informs the NC about existence of the reference point by switching the AP_RPE flag.
This case can be specified by settings #3 ABS=1 és #4 FLO=1in the parameter N0900 Reference Type.

AP_MIRR: Request for axis direction exchange (mirroring)
The PLC program can request the axis direction exchange on an axis through the AP_MIRR flag. The AP_MIRA flag is the acknowledge signal of the request which should always be waited.
In this case, when the axis movement is controlled from the part program, the motion direction of the given axis becomes opposite.
When the axis movement is controlled manually (jog buttons, handwheel), the motion direction does not change.
If the value of the flag is:
=0: the axis will move in accordance with the part program in the original direction set in parameter;
=1: the axis will move in accordance with the part program in the opposite direction to that set in parameter.

☞ ***Warning!*** *The axis direction exchange should always be carried out in the function preventing the block buffering (see the Program parameter group).*

7.14 Spindle Control Variables

The spindle control variables are such variables going from the NC to the PLC, or from the PLC to the NC, which are indexed per spindle. All symbols published here refer to the first spindle (indexed with 0). The appropriate variables of other spindles can be accessed by an indexed addressing. The control handles a maximum of 16 spindles.
Variables starting with

SN go from NC to PLC (Inputs), while variables starting with
SP go from PLC to NC (Outputs).

7.14.1 Bit-type Spindle Control Variables

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
SN_RMPD	Speed command signal ramped	**SP_SEN**	Spindle output signal enable
SN_NS	Spindle speed reached the commanded value (N=Ns)	**SP_SSTRT**	Spindle start
SN_N0	Spindle speed is 0 (N=0)	**SP_PAR**	Spindle speed from parameter
SN_FLU	Spindle speed fluctuation	**SP_NEG**	Spindle rotation in CCW (M4, negative command signal)
SN_FLOFF	Monitoring of fluctuation is off	**SP_OREQ**	Orientation request
SN_LPCLSD	Position control loop closed	**SP_OSHRT**	Orientation in the shorter direction
SN_ORIP	Position control loop closed and on the orientation position	**SP_SSYNCR**	Synchronization request
SN_SINPOS	Spindle in position	**SP_PHSHFTR**	Phase shift request
SN_SSYNA	Synchronization acknowledge	**SP_POLYR**	Not used
SN_PHSHFTA	Phase shift acknowledge	**SP_SEND**	Encoder error monitoring disable
SN_SYNCPOS	Not used	**SP_SMTNDP**	Motion disable in positive direction
SN_POLYA	Polygonal turning acknowledge	**SP_SMTNDN**	Motion disable in negative direction
SN_SMTNRP	Motion request in positive direction	**SP_SDETCHR**	Spindle detach request
SN_SMTNRN	Motion request in negative direction	**SP_SLCLR**	Position control loop closing request
SN_SRAPR	Rapid motion request	**SP_SSROFF**	Slave spindle loop opening and takeover of only the command signal
SN_SDETCHA	Spindle detach acknowledge	**SP_SDISPD**	Spindle display disable
SN_SALM	Alarm state on the spindle	**SP_FEEDD**	Spindle motion with feed-rate disable

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
SN_RPE	Reference point established	**SP_RAPD**	Spindle rapid motion disable
SN_SINDP	Spindle in index position	**SP_TLCHIA**	Individual tool change acknowledge
SN_TLCHI	Individual tool change request	**SP_TLCHA**	Tool group change acknowledge
SN_TLCH	Tool group change request	**SP_TLSKP**	Tool skip signal
SN_TLSKPA	Tool skip acknowledge	**SP_TLCD**	Life counter disable
SN_TLNL	Tool notice life signal	**SP_OSW**	Spindle on orientation switch

Bit spindle variables going from NC to PLC

SN_RMPD: Speed command signal ramped

The spindle handler will ramp up the command signal issued to the drive, if the revolution number has to be increased in an absolute value, based on the parameter N0665+n Rn S Ramp Up; in case the revolution number has to be decreased in an absolute value, it will ramp it down based on the parameter N0673+n Rn S Ramp Down, where “n” is the number of the gear range.

As soon as the spindle handler has finished the ramping-up or ramping-down of the command signal, it will set the SN_RMPD flag to 1.

SN_NS: Spindle speed reached the commanded value (N=Ns)

In case the spindle speed measured from the encoder has reached the commanded revolution number within the tolerance determined by parameters N0627 S N% and N0628 S NW, the spindle handler will set the flag SN_NS.

If the revolution number is good in absolute value but the SN_NS signal is not received, the direction of the encoder has to be changed in the N0609 Spindle Encoder Config parameter (#1 ID, or #2 AD parameter), provided that the spindle is not unipolar: #5 UNI=0.

SN_N0: Spindle speed is 0 (N=0)

In case the spindle speed measured from the encoder has reached the 0 value within the tolerance determined on the parameter N0629 S N0, the spindle handler will set the flag SN_N0 (together with the flag SN_NS).

SN_FLU: Spindle speed fluctuation

In case the spindle speed measured from the encoder falls outside of the tolerance range determined by parameters N0632 S Fluct% and N0633 S FluctW, the spindle handler will set the flag SN_FLU.

SN_FLOFF: Monitoring of fluctuation is off

In case the monitoring of spindle fluctuation is cancelled from the part program by the command G25, the SN_FLOFF flag will be set.

SN_LPCLSD: Position control loop closed

If the spindle the position control loop is closed, the flag will be set. These cases are the following:

– after the execution of SP_OREQ orientation, or SP_SLCLR loop closing command,
– after the synchronization of two spindles and the spindle is a synchronous master or synchronous slave,
– if the spindle is the master or slave spindle of polygonal turning (G51.2),
– if the spindle is the master of the electronic gear box control after programming of G81.8 function.

SN_ORIP: Position control loop closed and on the orientation position

The spindle handler sets the flag, if on the spindle the position control loop is closed and the difference of spindle position and parameter value of N0684 Spindle Grid Shift is in the range determined by the parameter N0743+n Rn S Inpos.
If N0684 Spindle Grid Shift=0, it is in the range of the zero-pulse.

SN_SINPOS: Spindle in position

The spindle handler sets the flag,

– after the execution of an SP_OREQ orientation or SP_SLCLR loop closure command, if the absolute value of the difference between the command position issued on the spindle and the position measured from the encoder is lower than the value determined on the N0743+n Rn S Inpos parameter.
– In case the *spindle* is a *synchronous slave* when two spindles are synchronized, or
– In case the *spindle* is the *slave* spindle of G51.2 *polygon turning*,
the difference of absolute value of cyclical positions of the master and slave spindles (the position difference of zero-pulses by taking into consideration the offsets, too) is lower than the value determined on the N0743+n Rn S Inpos parameter.

SN_SSYNA: Synchronization acknowledge

It is the acknowledge flag of the SP_SSYNCR synchronization request.
If PLC requests a synchronization on a spindle, to another one, by setting the SP_SSYNCR flag and the synchronization has been carried out, the spindle handler will set the flag.

SN_PHSHFTA: Phase shift acknowledge

It is the acknowledge flag of the SP_PHSHFTR phase shift request.

If the PLC requests the synchronization with a phase shift with the SP_PHSHFTR=1 condition, and both the synchronization and phase shift has been carried out, the spindle handler will set the flag SN_PHSHFTA.

SN_SYNCPOS: Not used

SN_POLYA: Polygonal turning acknowledge

In case the spindle is the slave spindle of G51.2 polygonal turning, and the synchronization to the master spindle has been carried out, the spindle handler will set the flag SN_POLYA.

SN_SMTNRP: Motion request in positive direction
SN_SMTNRN: Motion request in negative direction

In case of indexing a spindle, before the start of any movements the interpolator will set the motion request flag in the appropriate direction.
If the PLC enables the movement in the appropriate direction, it has to reset one of the appropriate flags SP_SMTNDP=0, or the SP_SMTNDN=0.
See also the SN_SRAPR, SP_RAPD, SP_FEEDD flags.
The flags can be used for example for clamping and unclamping of spindles.

SN_SRAPR: Rapid motion request

The flag is handled by the interpolator together with SN_SMTNRP, SN_SMTNRN motion request flags.
If SN_SRAPR=0, the interpolator wants to move with feed-rate, and waits for the SP_FEEDD=0 (motion with feed-rate enabled) and SP_RAPD=1 (rapid motion disabled) statuses.
If SN_SRAPR=1, the interpolator wants to move with a rapid traverse, and waits for the SP_FEEDD=1 (motion with feed-rate disabled) and SP_RAPD=0 (rapid motion enabled) statuses.
See also SN_SMTNRP, SN_SMTNRN, SP_RAPD, SP_FEEDD flags.
The handling of this flag can be used for example for gear range changes between feed-rate and rapid traverse movements.

SN_SDETCHA: Spindle detach acknowledge

SN_SDETCHA is the acknowledge flag of the SP_SDETCHR spindle detach request signal.
Upon the SP_SDETCHR=1 spindle detach request the spindle handler gives back SN_SDETCHA=1. Upon the SP_SDETCHR=0 attach request the spindle handler gives back SN_SDETCHA=0. See also: SP_SDETCHR flag’s description.

SN_SALM: Alarm state on the spindle

In case the spindle handler on any element of the spindle control (encoder, drive operating through EtherCAT, position control loop) recognized a servo error, it will set the flag SN_SALM.

☞ ***Attention!*** *The task of the PLC program is to react to the alarm signal, to stop the drive, and to activate an emergency status! The NC will not automatically switch the machine off, only upon the command of the PLC!*

SN_RPE: Reference point established

In SN_RPE=1 status on the spindle the position of the zero-pulse is known. In case of absolute encoders, the flag is always 1.

SN_SINDP: Spindle in index position

The value of the flag is 1, if the difference of the spindle position and its index position is within the range determined by the N0743+n Rn S Inpos parameter.
Under an index position we mean positions, being on a spindle in discrete, equally spaced distances from each other. E.g.: Clamping a spindle happens by pushing a bar into a hole on a disc on which the holes are equally spaced in each 5 degrees, then the spindle can be indexed every 5 degrees.
On the parameter N0822 Basic Angle of Spnd. Pos. we can set the indexing angle.
E.g.: In case of the above spindle clamping it will be 5.

SN_TLCHI: Individual tool change request

The flag will be set by the NC, if tool management is enabled on the parameter N2900 Tool M. Config #0 TMU=1 and
– the life of the tool being in the given spindle has expired, or
– the PLC sets the tool skip flag SP_TLSKP because the tool is broken.
The SN_TLCHI flag remains set till the time the PLC sets the SP_TLCHIA individual tool change acknowledge signal. After the NC has reset the SN_TLCHI signal, the PLC has to reset the acknowledge signal, too.
The SN_TLCHI flag *will not be cleared* upon a reset.

SN_TLCH: Tool group change request

The flag will be set by the NC, if tool management is enabled on the parameter N2900 Tool M. Config #0 TMU=1 and it is true for the whole tool group (tools with the same type code) that
– its tool life “has expired” or
– it is “broken”.
The SN_TLCH flag will remain set till the time the PLC sets the SP_TLCHA tool group change acknowledge signal. After the NC has reset the SN_TLCH signal, the PLC has to reset the acknowledge signal, too.

The SN_TLCH flag *will not be cleared* upon a reset.

SN_TLSKPA: Tool skip acknowledge

If the PLC detects that a tool is broken, it sets the tool skip signal SP_TLSKP. The NC will record in the tool management table the “broken” status then it will set the tool skip acknowledge signal SN_TLSKPA, in case tool management is enabled on the parameter the N2900 Tool M. Config #0 TMU=1.
After that, NC will set the individual tool change request signal SN_TLCHI.

SN_TLNL: Tool notice life signal

In case the tool notice life of the tool being in the spindle has expired, the NC will set flag SN_TLNL for the period of 1 PLC cycle, if the N2900 Tool M. Config parameter #0 TMU=1.

<u>Bit spindle variables going from the PLC to NC</u>

SP_SEN: Spindle output signal enable

Every time the PLC issues a command to the spindle (rotation, orientation synchronization, polygonal turning), it has to enable the issuance of signals by setting the flag SP_SEN.
After stopping the spindle, the flag has to be reset.

SP_SSTRT: Spindle start

Upon ***SP_SSTRT=1*** the spindle handler will rotate the spindle with the revolution number set.
Upon ***SP_SSTRT=0*** the spindle handler will stop the rotation of the spindle.
See also the SP_PAR, SP_NEG flags and the SP_PRG register.

SP_PAR: Spindle speed from parameter

In ***SP_PAR=0*** status, upon SP_SSTRT=1 the spindle handler will rotate the spindle with speed determined in the SP_PRG register.
In ***SP_PAR=1*** status, upon SP_SSTRT=1 the spindle handler will rotate the spindle with the speed determined on the az N0657+n Rn S Jog Speed parameter.

SP_NEG: Spindle rotation in CCW (M4, negative command signal)

In ***SP_NEG=0*** state, upon SP_SSTRT=1 the spindle handler will rotate the spindle in a positive, CW or M3 direction (positive command signal for the drive).
In ***SP_NEG=1*** state, upon SP_SSTRT=1 the spindle handler will rotate the spindle in a negative, CCW or M4 direction (negative command signal for the drive).

In case the rotation direction of the spindle is not right, in the parameter N0609 Spindle Encoder Config we have to alter parameter bit #0 MD for changing the rotation direction of the motor. (We should not change the rotation direction by inverting the flag SP_NEG.)
In ***SP_NEG=0*** status, upon SP_SSYNCR=1 synchronization request the spindle handler will rotate the spindle in the same direction as the master is turning which is determined in the SP_MAST register.
In ***SP_NEG=1*** status, upon SP_SSYNCR=1 synchronization request the spindle handler will rotate the spindle in the opposite direction as the master is turning which is determined in the SP_MAST register. (Synchronization of sub-spindles.)

SP_OREQ: Orientation request
Upon ***SP_OREQ=1,*** the spindle handler
– In case the spindle is rotating,
Will slow down the rotation of the spindle to the speed determined on the N0800+n Rn S OrientSpeed parameter,
closes the position control loop (SN_LPCLSD=1),
then it will take up the position calculated from the zero-pulse, determined on the N0684 Spindle Grid Shift parameter (to the zero-pulse if the parameter is 0)
sets the SN_ORIP flag.
– In case the spindle is in standstill and SP_OSHRT=0,
in the direction determined on the SP_NEG flag, with the speed determined on the N0800+n Rn S OrientSpeed parameter it will occupy the position counted from the zero-pulse and determined on the N0684 Spindle Grid Shift parameter (the zero-pulse if the parameter is 0),
sets the SN_ORIP flag.
– In case the spindle is in standstill and SP_OSHRT=1, and the position of the zero-pulse is known, furthermore, the N0607 Spindle Config parameter #4 ZOR=1
in the nearer direction, with the speed determined on the N0800+n Rn S OrientSpeed parameter it will occupy the position counted from the zero-pulse and determined on the N0684 Spindle Grid Shift parameter (the zero-pulse if the parameter is 0),
sets the SN_ORIP flag.
Upon the ***SP_OREQ=0*** the spindle handler will open the position control loop and it will reset the SN_LPCLSD flag.

SP_OSHRT: Orientation in the shorter direction
In case the spindle is in standstill and SP_OSHRT=0, it will orientate in the direction determined on the SP_NEG flag.

In case the spindle is in standstill and SP_OSHRT=1, and the position of the zero-pulse is already known, furthermore, the N0607 Spindle Config parameter is #4 ZOR=1 it will orientate on the shorter way.

SP_SSYNCR: Synchronization request

The flag serves for the synchronization of two spindles. Synchronization means the position controlled co-running of the zero-pulses of two spindles when they are turning. The two zero-pulses may run together or in a distance from each other, set on the N0685 Spindle Phase Shift parameter. The gain of the synchronous control can be adjusted by the N0784+n Rn S Synchr K parameter. Also after turning off the master spindle, the slave will follow the master. The synchronization to a master spindle is always requested by the slave spindle.

We use a synchronization for example in cases when on a machine with a sub-spindle, the workpiece has to be taken over by the sub-spindle as a synchronous slave, in a rotating status. The position controlled synchronization means that if in the main spindle we machine something on the workpiece compared to the zero-pulse of the main spindle, in the sub-spindle on the other side we can carry out machining compared to this. Other application can be when the sub-spindle must grasp a non-standard (non-round) workpiece.

Before a synchronization request, we have to write the index of the master spindle into the slave spindle's SP_MAST register.

In case we would like to shift the two zero-pulses with the value set on the N0685 Spindle Phase Shift parameter from each other, before the synchronization request we have to set the SP_PHSHFTR flag.

Upon the ***SP_SSYNCR=1*** the spindle handler

– if the revolution number of the master spindle is higher than the value determined on the N0319+n Rn S Rapid n=1...8 parameter, it will slow it down to the value determined on the parameter,

– it will close the position control loop on the master spindle,

– it will close the position control loop on the slave spindle,

– it will increase the speed of the slave to the speed of the master, in the direction determined on the SP_NEG flag compared to the master,

– according to the status of the SP_PHSHFTR flag, it will bring the zero-pulses of the two spindles to the appropriate distance,

– it will switch on the co-running control which can be set by the N0784+n Rn S Synchr K parameter,

– it will set the SN_SSYNA acknowledge flag on the slave spindle and the SN_SINPOS flag, if the absolute value of the synchronization error of the two spindles is lower than the value determined on the N0743+n Rn S Inpos parameter.

Upon the ***SP_SSYNCR=0*** the spindle handler

– will open the position control loop on the master spindle,
– on the master spindle, if it is rotating, it will set the speed determined in the SP_PRG register,
– it will open the position control loop on the slave spindle,
– it will reset the SN_SSYNA and SN_SINPOS flags.

SP_PHSHFTR: Phase shift request

In the ***SP_PHSHFTR=1*** status, upon ***SP_SSYNCR=1*** synchronization request it will set the zero-pulses of the two spindles to the distance determined on the N0685 Spindle Phase Shift parameter and on the slave spindle it will set the SN_SSYNA acknowledge flag and the SN_SINPOS flag, if the absolute value of the synchronization error of the two spindles is lower than the value determined on the N0743+n Rn S Inpos parameter

In ***SP_PHSHFTR=0*** status, upon ***SP_SSYNCR=1*** synchronization request it will synchronize the zero-pulses of the two spindles to each other and it will set the SN_SSYNA acknowledge flag on the slave spindle and the SN_SINPOS flag, if the absolute value of the synchronization error of the two spindles is lower than the value determined on the N0743+n Rn S Inpos parameter.

SP_POLYR: Not used

SP_SEND: Encoder error monitoring disable

If we set the flag, the spindle handler will not monitor the errors of the encoder mounted on the spindle (it will not issue encoder error signals).

SP_SMTNDP: Motion disable in positive direction
SP_SMTNDN: Motion disable in negative direction

They are the response signals issued upon the SN_SMTNRP and SN_SMTNRN motion request flags.

Before every spindle movement (in SP_OREQ=1, or SP_SLCLR=1 status) the interpolator sets the motion request flag with the appropriate direction (SN_SMTNRP=1, SN_SMTNDN=1).

The spindle will not move till the PLC enables the movement in the appropriate direction, by resetting the appropriate flag SP_SMTNDP=0, or the SP_SMTNDN=0.

The spindle will stop if the motion disable flag with the appropriate direction is set by the PLC.

See also the SN_SRAPR, SP_RAPD, SP_FEEDD flags.

The flags may be used for example for clamping and unclamping of spindles.

SP_SDETCHR: Spindle detach request

In case the operation of a spindle has to be detached, we can request it from the system by setting the SP_SDETCHR flag of the appropriate spindle. After the spindle handler has stopped the operation of the spindle, it will give back the SN_SDETCHA=1 acknowledge signal. During the attaching a spindle the SP_SDETCHR flag has to be reset, and wait till the spindle handler gives back the SN_SDETCHA=0 status.

In case the SN_SDETCHA signal is in a 1 status, the spindle handler will not operate:

it will clear on the spindle the record reference point established: SN_RPE=0,
it will not measure and will not register the position from the encoder,
it will not close the position control loop,
it will not issue any signals towards the spindle drive,
in a part program and PLC it is not possible to refer to the spindle,

the spindle does not exist.

Vice versa, in case the SP_SDETCHR=0 and the NC resets the SN_SDETCHA=0 flag, the spindle will start to exist again.

A spindle exists if on the

N0607 Spindle Config parameter #0 SEX value 1, and
AN_DETCHA=0.

☞ ***Attention!*** *The detach and attach of a spindle shall always be carried out in a function suppressing the buffering of the blocks (see the Program parameter group)*

Example:

The spindle of a lathe need to be operated as axis C (rotary table), then it shall be used as a spindle. Let's say the

number of axis C is 3 (N0002 Axis Assign A3=1, N0100 Axis Name1 A3=C)
and the number of spindle S1 is 1. (N0605 Spindle Name2 S1=1)

After the power-on, it will work as a spindle:

SP_SDETCHR,#0=0, SN_SDETCHA,#0=0
AP_DETCHR,#2=1, AN_DETCHA,#2=1

If we would like to transform spindle S1 to axis C, we have to program an appropriate buffer-suppressive function M and we have to set and reset the flags in the following way:

SP_SDETCHR,#0=1, SN_SDETCHA,#0=1
AP_DETCHR,#2=0, AN_DETCHA,#2=0

From this on, the encoder signals arriving from the same spindle drive will not be received by the S1 spindle control but the C axis control. The command signal will not be issued by the S1 spindle control but by the C axis control to the spindle drive. Axis C can be moved, and it is possible to refer to address C in a part program.

The transformation of axis C back to spindle S1 is carried out in the same way.

SP_SLCLR: Position control loop closing request

Upon ***SP_SLCLR=1*** the spindle handler

– In case the spindle is rotating,

will stop the rotation of the spindle,

will close the position control loop (SN_LPCLSD=1),

– In case the spindle is in standstill, it will close the position control loop (SN_LPCLSD=1).

Upon ***SP_SLCLR=0*** the spindle handler will open the position control loop (SN_LPCLSD=0).

The function can be used, e.g., for the rapid execution of the rigid tapping cycle, if it is not necessary to hit the same bore hole again. See also: N0823 M Code for Closing S Loop parameter and N1503 Drilling Cycles Config. parameter #1 TSC bit.

SP_SSROFF: Slave spindle loop opening and takeover of only the command signal

In case the spindle is a **synchronous slave**, i.e. on the spindle SN_SSYNA=1, it will open the position control loop, the slave spindle will take over the speed command signal of the master spindle and it will send it to the slave's drive.

Example:

We synchronize the sub-spindle to the main spindle due to a takeover of a workpiece, then the sub-spindle chuck grabs the workpiece, too. Afterwards, every time the chuck is closed on both the main and sub-spindle side, in order to prevent over-determinedness, by setting the SP_SSROFF=1 we can open on the slave spindle the position control loop. Then the slave spindle will take over the speed command signals from the master and they will move together. See also the DP_SILCK drive signal.

SP_SDISPD: Spindle display disable

If on a spindle, on the N0607 Spindle Config parameter #0 SEX value 1, the spindle will automatically appear on the FST display.

If the flag SP_SDISPD=1, the given spindle will disappear from the FST display.

For example, if we detach a spindle by the SP_SDETCHR flag, its displaying can be switched off, too.

SP_FEEDD: Spindle motion with feed-rate disable

It is the response signal issued on the SN_SRAPR rapid motion request flag.

If SN_SRAPR=0, a movement with feed-rate will start only after the PLC has reset the SP_FEEDD=0 signal.

If SP_FEEDD=1 a movement with feed-rate will not go.

The flag shall be handled together with the SP_SMTNDP, SP_SMTNDN motion disable flags.

SP_RAPD: Spindle rapid motion disable

It is the response signal issued on the SN_RAPR rapid motion request flag.
If SN_RAPR=1,a movement with rapid traverse will start only after the PLC has reset the SP_RAPD=0 signal.
If SP_RAPD=1 a movement with rapid traverse will not go.
The flag shall be handled together with the SP_SMTNDP, SP_SMTNDN motion disable flags.

SP_TLCHIA: Individual tool change acknowledge

It is the acknowledge signal of the flag SN_TLCHI.
If the PLC sets the signal SP_TLCHIA, the NC will reset the SN_TLCHI flag. After the NC has reset the SN_TLCHI signal, the PLC has to reset the acknowledge signal, too.

SP_TLCHA: Tool group change acknowledge

It is the acknowledge signal of the flag SN_TLCH.
If the PLC sets the SP_TLCHA signal, the NC will reset the SN_TLCH flag.After the NC has reset the SN_TLCH signal, the PLC has to reset the acknowledge signal, too.

SP_TLSKP: Tool skip signal

If the PLC detects, that the tool in the spindle is broken, it sets the SP_TLSKP signal. Then the NC in the tool management table will record on the tool the “broken” status and after that it will set the SN_TLSKPA acknowledge signal.
After this the NC will set the SN_TLCHI individual tool change request signal.
After the NC has given back the SN_TLSKPA acknowledge signal, the PLC has to reset the SP_TLSKP flag.

SP_TLCD: Life counter disable

If the PLC sets the flag, the tool life counter of the tool being in the given spindle will stop counting.

☞ ***Attention!*** *The flags*

SP_SSTART,
SP_OREQ,
SP_SSYNCR,
SP_SLCLR

mean commands excluding each other. From these always only one flag can be set (=1)!

SP_OSW: Spindle on orientation switch.

In the case of N0609 Spindle Encoder Config parameter #4 OSW=1, spindle orientation will occur to interface input. Receiving the signal of the switch has to be executed in the Int0 fast module of the PLC program in any case, because of the sampling frequency. The state of the switch has to be copied on the SP_OSW flag in the PLC program. The spindle control module begins to start orientation when the state of the SP_OSW flag is 1

Accuracy of this solution is casual, measurement is requested.

7.14.2 DWORD-type Spindle Control Variables

Inputs		**Outputs**	
Symbol	**Description**	**Symbol**	**Description**
SN_1	Spindle control bits handed over from NC to PLC (DWORD)	**SP_1**	Spindle control bits handed over from PLC to NC (DWORD)
SN_NCOM	Spindle speed command issued to the drive, in rev/min (DWORD)	**SP_PRG**	Programmed spindle speed (DWORD)
SN_NACT	Current spindle speed measured from the encoder (DWORD)	**SP_ROT**	Spindle rotation status code (3, 4, ...) (DWORD)
		SP_RNG	Spindle range code (11, 12, …, 18) (DWORD)
		SP_MAST	Index of the master spindle of the spindle (0,1,2...) (DWORD)
		SP_ASSIGN	Assignment of the spindle to a channel (1,2,...) (DWORD)
		SP_ACTT	Number of the tool being in the spindle (DWORD)

DWORD-type spindle variables going from the NC to PLC

SN_NCOM: Spindle speed command issued to the drive, in rev/min (DWORD)

In status ***1*** of the ***SP_SSTART flag*** it is the value of the spindle speed command signal issued to the spindle drive in rev/min unit.

The SN_NCOM register value:

– If ***SP_PAR=0***, the spindle speed determined in the SP_PRG register
by taking into consideration of the ramping up and down,
multiplied by the spindle override (SP_SOVER),
limited by N0641+n Rn S Min and N0649+n Rn S Max parameters,
in G96 state limited by the maximum spindle speed determined by the G92 S command and the N0688 Min Spindle Speed G96 parameter.

– If ***SP_PAR=1***, it is the spindle speed determined on the N0657+n Rn S Jog Speed parameter. It is influenced only by the ramping up and down, but neither the spindle override, nor the limiting parameters are validated.

SN_NACT: Current spindle speed measured from the encoder (DWORD)

In case an encoder is mounted on the spindle, in any status of the spindle, it shows the current speed of the spindle in rev/min units.

DWORD-type spindle variables going from the PLC to NC

SP_PRG: Programmed spindle speed (DWORD)

The PLC writes the value of the S code handed over by the channel in CN_SC register into the appropriate SP_PRG register. Unit: rev/min.

In case of constant surface speed, in G96 status (CN_CSURFS=1), the spindle speed written into the CN_CSPN register, calculated by the NC for the surface speed, has to be copied into the appropriate SP_PRG register.

SP_ROT: Spindle rotation status code (3, 4, ...) (DWORD)

Into the SP_ROT register the regular rotation codes have to be written in: 3: M3, 4: M4, 5: M5, 19: M19.

The loop closure code without orientation, determined on the N0821 No. of M Code for Spnd. Pos parameter shall be written here.

Into this register also the M codes determined on the N0689 Spindle M Low and N0690 Spindle M High parameters shall be written.

The rotation status code of the spindle becomes displayed out on the FST screen.

SP_RNG: Spindle range code (11, 12, …, 18) (DWORD)

In case on the spindle the range change happens upon M codes, for the 8 ranges the M11, M12, ..., M18 codes shall be used.

The current range code of the spindle shall be written into the register, even if we change the range to an S code. Only 11, 12, ..., 18 can be written into the register. The spindle handler takes into consideration the range-dependant parameters of spindles based on the SP_RNG code.
It shall be mandatorily filled in!

SP_MAST: Index of the master spindle of the spindle (0,1,2...) (DWORD)
In case of synchronization of two spindles the slave spindle will request the synchronization by setting the SP_SSYNCR flag. The index of the master spindle shall be determined in the SP_MAST register of the slave spindle requesting the synchronization. (Spindle number - 1.)

SP_ASSIGN: Assignment of the spindle to a channel (1,2,...) (DWORD)
In this register we can determine, in which channel will the given spindle operate. Into the register always the number of the channel shall be written: in case of channel 1 it shall be 1, etc. We can refer to the spindle from a part program in only that channel to which the spindle has been appointed by the SP_ASSIGN register.
During program running, it may be necessary to get a spindle to another channel, from which we can be able to machine with it. The redirection may be carried out by an M function by the PLC program.
☞ ***Attention!*** *The overwriting of the SP_ASSIGN register shall be carried out exclusively in a function suppressing the block buffer!*
It shall be mandatorily filled in, and it shall be initialized after the power-on!

SP_ACTT: Number of the tool being in the spindle (DWORD)
In case the N2901 Search Config parameter's bit #7 TSP is 1, the displaying of the current tool number in the FST window is carried out from the register SP_ACTT indexed per spindle.
The PLC program writes here the number of tool being in the given spindle. Usually it can be used on a multi-spindle milling machines.

7.14.3 Double-type Spindle Control Variables

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
		SP_SOVER	Spindle override: if =1: 100% (double)

Floating-point spindle variables going from the PLC to NC

SP_SOVER: Spindle override: if =1: 100% (double)

An override register is available per spindle. The override value shall be written into the register in the floating-point format. If e.g.

SP_SOVER=0.32 it means 32%

SP_SOVER=1.0 it means 100%

In case of using an NCT machine control panel the status of the spindle override switch can be taken from the MKSOVER register, or to originate it from the MB_SMAX, MB_S100, MB_SMAX buttons.

The upper and lower limit of the spindle override shall be set in the PLC program!

☞ ***Attention!*** *MKSOVER is an integer, DWORD-type, SP_SOVER is, on the other hand, a floating-point double, i.e. the setting of the override requires a conversion from integer to floating-point (FLT command). The indexation of SP_SOVER is carried out two-by-two!*

7.15 Channel Control Variables

The channel control variables are such variables going from the NC to PLC, or from the PLC to the NC which become indexed per channel. All symbols published here refer to the first channel (with 0 index). The appropriate variables of the other channels are accessible by indexed addressing. The control can handle a maximum of 8 channels.
Variables starting with a

CN go from NC to PLC (Inputs), whilst variables starting with
CP go from the PLC to NC (Outputs).

7.15.1 Bit-Type Channel Control Variables

Inputs		**Outputs**	
Symbol	**Description**	**Symbol**	**Description**
CN_M1STB	M function 1 strobe signal	**CP_START**	Start request
CN_M2STB	M function 2 strobe signal	**CP_STOP**	Stop request
CN_M3STB	M function 3 strobe signal	**CP_JOG**	Jog mode request
CN_M4STB	M function 4 strobe signal	**CP_INCR**	Incremental jog mode request
CN_M5STB	M function 5 strobe signal	**CP_HNDL**	Handwheel mode request
CN_M6STB	M function 6 strobe signal	**CP_REFP**	Reference point travel mode request
CN_M7STB	M function 7 strobe signal	**CP_EDIT**	Edit mode request
CN_M8STB	M function 8 strobe signal	**CP_AUTO**	Automatic mode request
CN_SSTB	S function strobe signal	**CP_MDI**	Manual data input mode request
CN_TSTB	T function strobe signal	**CP_JOGRAP**	Jog with rapid traverse
CN_AUX1STB	Auxiliary function 1 strobe signal	**CP_TAXF**	Request for manual mode of moving in the tool direction
CN_AUX2STB	Auxiliary function 2 strobe signal	**CP_TRGAF**	Request for manual mode of moving in the direction perpendicular to the tool
CN_AUX3STB	Auxiliary function 3 strobe signal	**CP_TTCRF**	Request for manual mode of moving around the tool center point
CN_GSTB	Not used	**CP_TBLB**	Request for manual mode of moving according to the table
CN_BKBUF	Executable block in the buffer	**CP_INTDREQ**	Not used
CN_STPREQ	Not used	**CP_TLCM**	Tool length offset measurement mode on
CN_ALRM	Not used	**CP_S2TS**	Selection of tool offset setter belonging to S2
CN_OPMES	Not used	**CP_WPCM**	Workpiece zero offset measurement mode on

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
CN_INTD	Interrupted status	**CP_S2WS**	Selection of work offset setter belonging to S2
CN_WTNG	Waiting for the synchronization M code(s) of another channel(s)	**CP_AHND**	Request for zero point offset using handwheel
CN_ITFCHK	Not used	**CP_WMCAXF**	Request for feed parallel with the misalignment compensation
CN_ITFALM	Not used		
CN_CSURFS	Constant surface speed (G96)		
CN_POLYT	Polygonal turning (G51.2)		
CN_INCH	Data input in inches (G20)		
CN_HSHP	High-speed high-precision mode (G5.1)		
CN_CSACK	Not used		
CN_WPCNT	Machined workpiece = workpiece to be machined		
CN_EGBMD	EGB mode (G81.8)		
CN_RTRFIN	Retraction finished		
CN_IPSTP	Interpolator in standstill	**CP_SGLBK**	Single block mode on
CN_IPEPTY	Interpolator empty	**CP_CNDSP**	Conditional stop on
CN_CBFR	Cutting block feed request	**CP_TEST**	Test mode request
CN_OVDIS	Override disable (G63)	**CP_MLCK**	Machine lock mode request
CN_THRD	Thread cutting (G33, G34)	**CP_DRRUN**	Dry run request
CN_THRDC	Thread cutting cycle (G76, G78)	**CP_BKRST**	Block restart request
CN_TAP	Tapping (G74, G84)	**CP_BKRET**	Block return request
CN_RTAP	Rigid tapping (G84.2...)	**CP_FLCK**	Function lock request
CN_REFPG	Programmed reference point travel (G28)	**CP_ABSOFF**	Not used
CN_DWELL	Dwell (G04)	**CP_CNDBK_1**	Conditional block 1 on
CN_SKIP	Skip function (G31)	**CP_CNDBK_2**	Conditional block 2 on
CN_FREV	Feed-rate per revolution (G95)	**CP_CNDBK_3**	Conditional block 3 on
CN_POSCHK	Waiting for the in position signal	**CP_CNDBK_4**	Conditional block 4 on
CN_CHOP	Not used	**CP_CNDBK_5**	Conditional block 5 on
CN_CHARP	Not used	**CP_CNDBK_6**	Conditional block 6 on
CN_TLLE	Life of the tool referenced by T code expired	**CP_CNDBK_7**	Conditional block 7 on
		CP_CNDBK_8	Conditional block 8 on

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
CN_START	Start status	**CP_FIN**	Finish signal, all functions executed
CN_STOP	Stop status	**CP_RST**	Not used
CN_JOG	Jog mode	**CP_RSTREW**	Not used
CN_INCR	Incremental jog mode	**CP_CSREQ**	Not used
CN_HNDL	Handwheel Mode	**CP_NOWT**	Waiting for synchronization M code cancelled
CN_REFP	Reference point travel mode	**CP_MINT**	Interrupt macro calling
CN_EDIT	Edit mode	**CP_TMREN**	Free-purpose timer enable
CN_AUTO	Automatic mode	**CP_TSBD**	Not used
CN_MDI	Manual data input mode	**CP_EGBRRQ**	Retraction request in EGB mode
CN_FLCK	Function lock mode	**CP_M1ACK**	M function 1 acknowledge
CN_TAXF	Manual mode of moving in the tool direction	**CP_M2ACK**	M function 2 acknowledge
CN_TRGAF	Manual mode of moving in the direction perpendicular to the tool	**CP_M3ACK**	M function 3 acknowledge
CN_TTCRF	Manual mode of moving around the tool center point	**CP_M4ACK**	M function 4 acknowledge
CN_TBLB	Manual mode of moving according to the table	**CP_M5ACK**	M function 5 acknowledge
CN_ABSOFF	Not used	**CP_M6ACK**	M function 6 acknowledge
CN_TEST	Test mode	**CP_M7ACK**	M function 7 acknowledge
CN_MLCK	Machine lock mode	**CP_M8ACK**	M function 8 acknowledge
CN_DRRUN	Dry run mode	**CP_SACK**	S function acknowledge
CN_BKRST	Block restart state	**CP_TACK**	T function acknowledge
CN_BKRET	Block return state	**CP_AUX1ACK**	Auxiliary function 1 acknowledge
CN_AHND	Zero point offset using handwheel is changed	**CP_AUX2ACK**	Auxiliary function 2 acknowledge
CN_WMCAXF	The feed parallel with the misalignment compensation is turned on.	**CP_AUX3ACK**	Auxiliary function 3 acknowledge
		CP_GACK	Not used
CN_1100	#1100 macro variable value (bit)	**CP_HOLD**	Feed-hold for all axes in the channel
CN_1101	#1101 macro variable value (bit)	**CP_CBFEN**	Cutting block feed enable

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
CN_1102	#1102 macro variable value (bit)	**CP_FHNDL**	Feed-rate from handwheel mode on
CN_1103	#1103 macro variable value (bit)	**CP_OVC**	Not used
CN_1104	#1104 macro variable value (bit)	**CP_HNDLS1**	Feed-rate from the 1st handwheel
CN_1105	#1105 macro variable value (bit)	**CP_HNDLS2**	Feed-rate from the 2nd handwheel
CN_1106	#1106 macro variable value (bit)	**CP_HNDLS3**	Feed-rate from the 3rd handwheel
CN_1107	#1107 macro variable value (bit)	**CP_HNDLS4**	Feed-rate from the 4th handwheel
CN_1108	#1108 macro variable value (bit)	**CP_LIM1DIS**	Stroke range 1 disable
CN_1109	#1109 macro variable value (bit)	**CP_LIM2DIS**	Stroke range 2 disable
CN_1110	#1110 macro variable value (bit)	**CP_LIM3DIS**	Stroke range 3 disable
CN_1111	#1111 macro variable value (bit)	**CP_LIMSEL**	1B stroke range selection for all axes
CN_1112	#1112 macro variable value (bit)	**CP_CHOPON**	Not used
CN_1113	#1113 macro variable value (bit)	**CP_SGOEN**	Use of second geometry offset table enabled
CN_1114	#1114 macro variable value (bit)	**CP_SGOX**	Second geometry tool offset selection on axis X
CN_1115	#1115 macro variable value (bit)	**CP_SGOY**	Second geometry tool offset selection on axis Y
CN_1116	#1116 macro variable value (bit)	**CP_SGOZ**	Second geometry tool offset selection on axis Z
CN_1117	#1117 macro variable value (bit)	**CP_OSGNX**	Tool offset direction selection signal on axis X
CN_1118	#1118 macro variable value (bit)	**CP_OSGNY**	Tool offset direction selection signal on axis Y
CN_1119	#1119 macro variable value (bit)	**CP_OSGNZ**	Tool offset direction selection signal on axis Z
CN_1120	#1120 macro variable value (bit)	**CP_ROVLD**	Overlapping of rapid traverse blocks disable
CN_1121	#1121 macro variable value (bit)	**CP_RLSOT3**	Not used
CN_1122	#1122 macro variable value (bit)		
CN_1123	#1123 macro variable value (bit)		
CN_1124	#1124 macro variable value (bit)		
CN_1125	#1125 macro variable value (bit)		
CN_1126	#1126 macro variable value (bit)		
CN_1127	#1127 macro variable value (bit)		

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
CN_1128	#1128 macro variable value (bit)		
CN_1129	#1129 macro variable value (bit)		
CN_1130	#1130 macro variable value (bit)		
CN_1131	#1131 macro variable value (bit)		
		CP_1000	#1000 macro variable (bit)
		CP_1001	#1001 macro variable (bit)
		CP_1002	#1002 macro variable (bit)
		CP_1003	#1003 macro variable (bit)
		CP_1004	#1004 macro variable (bit)
		CP_1005	#1005 macro variable (bit)
		CP_1006	#1006 macro variable (bit)
		CP_1007	#1007 macro variable (bit)
		CP_1008	#1008 macro variable (bit)
		CP_1009	#1009 macro variable (bit)
		CP_1010	#1010 macro variable (bit)
		CP_1011	#1011 macro variable (bit)
		CP_1012	#1012 macro variable (bit)
		CP_1013	#1013 macro variable (bit)
		CP_1014	#1014 macro variable (bit)
		CP_1015	#1015 macro variable (bit)
		CP_1016	#1016 macro variable (bit)
		CP_1017	#1017 macro variable (bit)
		CP_1018	#1018 macro variable (bit)
		CP_1019	#1019 macro variable (bit)
		CP_1020	#1020 macro variable (bit)
		CP_1021	#1021 macro variable (bit)
		CP_1022	#1022 macro variable (bit)
		CP_1023	#1023 macro variable (bit)
		CP_1024	#1024 macro variable (bit)
		CP_1025	#1025 macro variable (bit)
		CP_1026	#1026 macro variable (bit)
		CP_1027	#1027 macro variable (bit)
		CP_1028	#1028 macro variable (bit)
		CP_1029	#1029 macro variable (bit)
		CP_1030	#1030 macro variable (bit)

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
		CP_1031	#1031 macro variable (bit)

Bit-type channel control variables going from NC to PLC

CN_M1STB: M function 1 strobe signal
CN_M2STB: M function 2 strobe signal
CN_M3STB: M function 3 strobe signal
CN_M4STB: M function 4 strobe signal
CN_M5STB: M function 5 strobe signal
CN_M6STB: M function 6 strobe signal
CN_M7STB: M function 7 strobe signal
CN_M8STB: M function 8 strobe signal

In a part program, we can write a maximum of 8 different M codes into a block, i.e. in a block the NC is able to hand over 8 different M codes to the PLC.

In case into the part program an M function has been programmed, the channel handler
– will write the value of the M code into the CN_MnC (n=1, 2, ..., 8) register
– thereafter, it sets the ***CN_MnSTB*** strobe signal for the ***period of 1 PLC cycle***.

The values of the 8 different M codes will be written by the control into 8 different CN_MnC (n=1, 2, ..., 8) registers. 8 strobe signals CN_MnSTB (n=1, 2, ...8) belong to the 8 handover registers. Upon the strobe signal, the PLC decodes the function based on the received code.

In case the received code covers a function existing on the machine, the PLC resets the CP_MnACK acknowledge flag corresponding to the strobe signal. It is enabled to set the CP_MnACK acknowledge flag exclusively after the full execution of the function.

CN_SSTB: S function strobe signal

In case an S function has been programmed in the part program, the channel handler
– will write the value of the spindle speed programmed on the S function to the CN_SC register,
– will write the number of the referred spindle to the CN_SSEL register (1...16)
– will write the code of the range (11...18) belonging to the programmed revolution number into the CN_RNGREQ register,
– thereafter, it will set the ***CN_SSTB*** flag for the ***period of 1 PLC cycle***.

Upon the strobe signal, the PLC decodes the codes written into the above registers and resets the CP_SACK acknowledge signal. It is enabled to set the CP_SACK acknowledge flag exclusively after the full execution of the function.

CN_TSTB: T function strobe signal

In case a T function has been programmed in the part program, the channel handler

– will write the number of the tool programmed on the T function to the CN_TC register, in a lathe channel by ***cutting the tool offset number*** called on the T address,

– it will write into the CN_MGZNO register that in which magazine can the referred tool be found,

– it will write into the CN_POTNO register that in which pot of the given magazine can the referred tool be found,

– it will examine whether the tool life of the referred tool, or tool group has expired or not, and it will set the CN_TLLE flag accordingly,

– then it will set the ***CN_TSTB*** flag for the ***period of 1 PLC cycle***.

Upon the strobe signal, the PLC decodes the codes written into the above registers and resets the CP_TACK acknowledge signal. It is enabled to set the CP_TACK acknowledge flag exclusively after the full execution of the function.

CN_AUX1STB: Auxiliary function 1 strobe signal
CN_AUX2STB: Auxiliary function 2 strobe signal
CN_AUX3STB: Auxiliary function 3 strobe signal

Three different addresses can be selected from among the A, B, C, U, V, W addresses on parameters N1332+n Aux Fu Addrn (n=1..3). If we refer to the address of the three auxiliary functions determined here in a part program, the channel handler

– will write into the CN_AUXnC (n=1...3) register the value programed on the address of the auxiliary function (which is always an integer DWORD),

– then it will set the ***CN_AUXnSTB*** (n=1...3) flag for a ***period of 1 PLC cycle***.

Upon the strobe signal, the PLC decodes the codes written into the above registers and resets the CP_AUXnACK (n=1...3) acknowledge signal. It is enabled to set the CP_AUXnACK acknowledge flag exclusively after the full execution of the function.

CN_GSTB: Not used

CN_BKBUF: Executable block in the buffer

The NC sets the flag if

– in automatic mode (CN_AUTO=1) a program is appointed to be run in automatic mode,

– it is in MDI mode (CN_MDI=1),

– in jog, incremental jog, or handwheel mode (CN_JOG=1, CN_INCR=1, or CN_HNDL=1) we enter a block to be executed until the block is executed.

CN_STPREQ: Not used
CN_ALRM: Not used

CN_OPMES: Not used

CN_INTD: Interrupted status

The flag will be set by the channel handler, if in automatic mode the running of the program has been interrupted. An interruption will occur in the following cases:

– During running the program we have exited from the automatic mode,
– Emergency-stop occurs in the automatic mode.

During the interruption process, the channel handler

– will store the status of the execution of the program,
– the interruption position
– the status of function acknowledge signals (CP_MnACK, CP_SACK, CP_TACK, CP_AUXnACK),

The interrupted program, by stepping back to automatic mode, can be continued depending on the CN_BKRST, CN_BKRET conditions.

Those functions, the acknowledge signal of which has been 0 during the interruption, will be issued again.

CN_WTNG: Waiting for the synchronization M code(s) of another channel(s)

On multi-channel controls it is necessary to suspend the execution of the part program running in one of the channels till one or more other channels reach a certain point in the execution of their program. We call this as the synchronization of channels. The synchronization of channels can be carried out by M codes. On the N2201 Waiting M Codes Min and N2202 Waiting M Codes Max parameters we can appoint an M code range.

The synchronization is fully carried out by the NC, the PLC has nothing to do with it. When a channel is waiting a synchronization signal, the NC will set the CN_WTNG flag.

CN_ITFCHK: Not used

CN_ITFALM: Not used

CN_CSURFS: Constant surface speed (G96)

In case the channel is in the G96 (constant surface speed calculation) status,

– the NC will set the CN_CSURF flag,
– it will calculate the spindle speed belonging to the instantaneous coordinate per PLC cycle and write it to the CN_CTSPN register,
– it will write the maximum spindle speed programmed by the G92 S command to the CN_NMAX register.

The task of the PLC program is to copy the value received in the CN_CTSPN register in the CN_CSURF=1 status to the SP_PRG register of the appropriate spindle (determined by the CP_SINP register).

CN_POLYT: Polygonal turning (G51.2)

If the channel handler is executing a G51.2 polygonal turning command, it will set the flag. The flag will be reset by the G50.2 command.
In case the flag is in a 1 status, the PLC has to prevent the spindle appointed in the CP_POLYSL register from executing any function.

CN_INCH: Data input in inches (G20)

The flag =0 in case of a metric data input G21.
The flag =1 in case of an inch data input G20.
The PLC has to write the increment size into the CP_INC increment register depending on the flag and the #0 IND bit of the N0104 Unit of Measure parameter,, in incremental jog and handle mode. Example:
– If IND=0, CN_INCH=0 and MB_I100 has been pushed: CP_INC=0.1
– If IND=0, CN_INCH=1 and MB_I100 has been pushed: CP_INC=0.254

CN_HSHP: High-speed high-precision mode (G5.1)

CN_CSACK: Not used

CN_WPCNT: Machined workpiece = workpiece to be machined

The workpiece counter on the control can be set by the operator on the screen Timers/Counters. In case the number of machined workpieces has reached the number of the workpieces to be machined, the control will set the CN_WPCNT flag for a period of 1 PLC cycle.
The control will increase the counter of the machined workpieces to the M function determined on the N2305 Part Count M parameter. E.g.: If the parameter =30, it will increase the counter to every M30 function.

CN_EGBMD: EGB mode (G81.8)

If the channel handler executes a G81.8 EGB function (electronic gear box), it will set the CN_EGBMD flag. The flag will be reset by the G80.8 function (electronic gear box off). (See also: N1800 EGB Contr parameter group.)

CN_RTRFIN: Retraction finished

If the channel handler executes a G81.8 EGB function (electronic gear box) (CN_EGBMD=1) and the PLC requests the retraction of the hob by setting the flag CP_EGBRRQ=1, the NC will retract the tool (hob) in the direction and to the distance determined on the N1804 Retr. Dist. parameter. At the end of the retraction the channel handler will set the CN_RTRFIN flag. After that the PLC has to reset the CP_EGBRRQ=0 flag.

CN_IPSTP: Interpolator in standstill

The flag takes up the CN_IPSTP=1 status in the below cases:

– in the channel there aren't any programs running,

– in the channel there is a program running but the override value is 0 CP_FOVER=0. This is true also in G0 positioning blocks, too.

– the channel gets to a stop status CN_STOP=1 (e.g.: upon the effect of the stop button, in a single block mode, etc.).

CN_IPEPTY: Interpolator empty

The flag takes up the CN_IPEPTY=1 status in the below cases:

– in the channel there aren't any programs running,

– in the channel there is a program running but the interpolator is empty, for example, due to the reason that the channel is executing a function block.

CN_CBFR: Cutting block feed request

The channel handler requests from the PLC to enable movement with feed-rate in the below cases, by the setting the flag CN_CBFR=1:

– G01 straight interpolation,

– G02, G03 circular interpolation,

– G33 thread cutting,

– in every cycle which executes any from among the above blocks.

The movement with feed-rate will not start until it is enabled by the PLC on the CP_CBFEN=1 flag, e.g. until the spindle is rotating.

CN_OVDIS: Override disable (G63)

The channel handler signals the override and stop disabled status to the PLC by the setting the flag CN_OVDIS=1 in the below cases:

– In G63 override disabled status (enabled by: G61, G62, G64),

– In G33 in thread cutting blocks (enable by other interpolation code, e.g.: G0, G1),

– In G74, G84 tapping cycles, while it is tapping,

– In G78 simple thread cutting cycles, when the thread is being cut,

– In G76 multiple repetitive thread cutting cycles, when the thread is being cut.

CN_THRD: Thread cutting (G33, G34)

The channel handler sets the flag CN_THRD=1, when

– It is executing a G33 thread cutting block (resetted by other interpolation code, e.g.: G0, G1),

– In G78 simple thread cutting cycles, when the thread is being cut,
– In G76 multiple repetitive thread cutting cycles, when the thread is being cut.

CN_THRDC: Thread cutting cycle (G76, G78)
The channel handler sets the flag CN_THRDC=1, when
– In G78 simple thread cutting cycles, when the thread is being cut,
– In G76 multiple repetitive thread cutting cycles, when the thread is being cut.

CN_TAP: Tapping (G74, G84)
The channel handler sets the flag CN_TAP=1, when
– In G74, G84 tapping cycles while it is tapping.

CN_RTAP: Rigid tapping (G84.2...)
The channel handler sets the flag CN_RTAP=1, when
– In G84.2, G84.3 rigid tapping cycles while it is tapping.

CN_REFPG: Programmed reference point travel (G28)
The channel handler sets the flag CN_REFPG=1, when
– In G28 command it travels to the reference point.

CN_DWELL: Dwell (G04)
The channel handler sets the flag CN_DWELL=1, when
– In G4 block it is executing a programmed dwell,
– In drilling cycles a dwell is being executed.

CN_SKIP: Skip function (G31)
The channel handler sets the flag CN_SKIP=1, when
– It is carrying out a G31 skip function.

CN_FREV: Feed-rate per revolution (G95)
The channel handler sets the flag CN_FREV=1, when
– It is moving by a feed-rate per revolution (G95).
In case of G94 (feed-rate per minute) the flag CN_FREV=0. In a positioning block (G0, G53) the flag will be reset even if there is a G95 status.

CN_POSCHK: Waiting for the in position signal
The channel handler sets the flag CN_POSCHK=1,
– at the end of all rapid traverse movement (G0, G53), if the N1337 Execution Config parameter #0 PCH=1,
– at the end of all movement blocks, into which a G9 has been programmed,
– at the end of all movement blocks, in a G61 exact stop status,

and it will wait till the lag value on all axes participating in the movement gets within the window determined on the N0516 Inpos parameter. After this it resets the CN_POSCHK flag.

CN_CHOP: Not used

CN_CHARP: Not used

CN_TLLE: Life of the tool referenced by T code expired

During all tool call (T code) the channel handler checks whether the life of the tool, or tool group has expired or not, provided that the N2900 Tool M. Config parameter #0 TMU=1.

In case the tool life has expired, the NC will set the CN_TLLE signal, simultaneously with the CN_TSTB signal.

CN_START: Start status

It is the acknowledge signal of the CP_START start request.

When the PLC sets the CP_START signal, the channel handler checks whether

– a program or a single block can be executed in the given mode (which are: auto, MDI, jog, incremental jog, handwheel) or not,

– there is an appointed program to run, or a single block entered for execution,

– there is any other obstacle for the start, e.g. error signal.

In case the above conditions are met, the channel handler gives back the CN_START=1 status, upon which the lamp of the start button (e.g. ML_START) can be turned on.

The channel handler resets the CN_START status, in case the NC

– gets into a stop status (CN_STOP=1)

– has executed the program or the single block.

CN_STOP: Stop status

The acknowledge signal of the CP_STOP stop request.

When the PLC sets the CP_STOP signal, the channel handler checks whether

– a program or a single block is under execution in the given mode (which are: auto, MDI, jog, incremental jog, handwheel) or not,

– there is any other obstacle for the stop, e.g. override and stop disabled status, or not.

In case the above conditions are met, the channel handler stops the interpolation and gives back the CN_STOP=1 status, upon which the lamp of the start button (e.g. ML_STOP) can be switched on.

The channel handler resets the CN_STOP status, in case the NC

– gets into a start status (CN_START=1)

– has executed the program or the single block.

CN_JOG: Jog mode
CN_INCR: Incremental jog mode
CN_HNDL: Handwheel mode
CN_REFP: Reference point travel mode
CN_EDIT: Edit mode
CN_AUTO: Automatic mode
CN_MDI: Manual data input mode

They are the acknowledge signals of the CP_JOG, CP_INCR, CP_HNDL, CP_REFP, CP_EDIT, CP_AUTO and CP_MDI mode request flags.
When the PLC requests a mode change on any of the CP_xxx flags, the channel handler checks whether

– in the mode in which the channel is, there is a program or a single block under execution or not,
– if there is a program execution, it will immediately stop the interpolator, or in an override and stop disabled status it will wait till the status terminates,
– in case it executes a single block in a jog, incremental jog or handwheel mode, or if it executes a program in MDI mode, it will clear the whole execution,
– in case it executes a program in an automatic mode, it will set the interrupted status CN_INTD=1,
– it will switch to the mode requested on the CP_xxx flag and sets the CN_xxx flag.

In status 1 of the CN_xxx flag the PLC may switch on the lamp of the appropriate mode button (e.g. ML_xxx).

CN_FLCK: Function lock mode

It is the acknowledge signal of the CP_FLCK function lock request.
When the PLC sets the function lock request signal CP_FLCK=1, the channel handler checks whether

– the status of the CN_FLCK signal may be modified or not, i.e. whether the control is in any of the automatic (CN_AUTO=1), or MDI (CN_MDI=1) modes or not,
– if not, it switches the status of the CN_FLCK flag to its opposite.

In status 1 of the CN_FLCK flag the PLC may switch on the lamp of the machine lock button (e.g.: ML_FLCK).
In status 1 of the CN_FLCK flag the channel handler will not hand over any function to the PLC: it will not issue the CN_MnSTB, CN_SSTB, CN_TSTB, CN_AUXnSTB strobe signals towards the PLC.

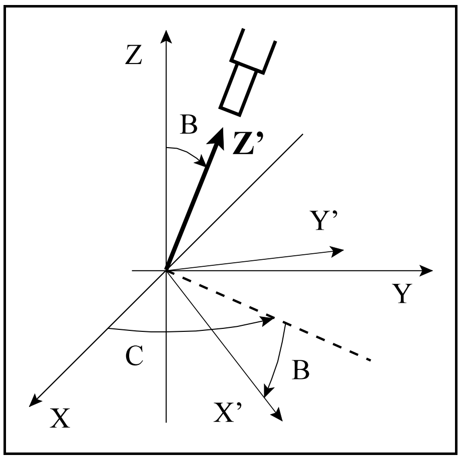
CN_TAXF: Manual mode of moving in the tool direction

It is the acknowledge signal of the CP_TAXF request for manual mode of moving in the tool direction. The control switches on the flag on the 5-axis machine tools of ***head-head or head-table configuration in jog, incremental jog and handwheel modes***. It does not work on the machine tools of table-table configuration.

In the state CN_TAXF=1, and ***having selected the tool direction axis specified in the N3201 Tool Axis Direction parameter,*** the control ***moves the tool in the tool direction according to the angular position of the rotary axes*** specified in the N3204 No. of the First Rot. Ax. and N3208 No. of the Second Rot. Ax. parameters.

For example:

Let our machine tool be of head-head configuration, i.e.N3200 Mechanical Type=Head-Head; N3201 Tool Axis Direction=Z, i.e. the tool is parallel with the Z direction in the initial position of the rotary axes; N3204 No. of the First Rot. Ax.=C and N3208 No. of the Second Rot. Ax.=B. In such a case, ***when Z jog button is pushed or Z axis is selected for handwheel moving***, the ***motion will be executed in the tool direction*** along all the three axes (X, Y, Z) might as well, taking positions of the B and C axes into account.

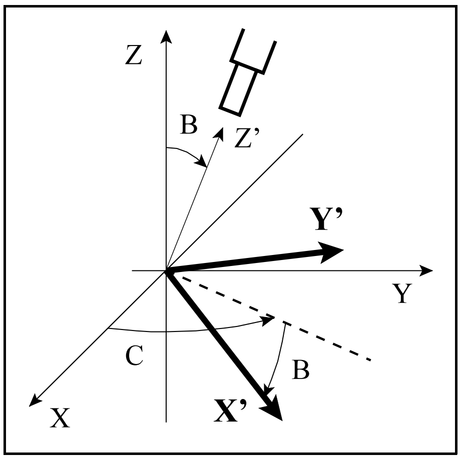


CN_TRGAF: Manual mode of moving in the direction perpendicular to the tool

It is the acknowledge signal of the CP_TRGAF request for manual mode of moving in the direction perpendicular to the tool. The control switches on the flag on the 5-axis machine tools of ***head-head or head-table configuration in jog, incremental jog and handwheel modes***. It does not work on the machine tools of table-table configuration. In the state CN_TRGAF=1, and ***having selected one of the directions perpendicular to the axis specified in the N3201 Tool Axis Direction parameter,*** the control ***moves the tool in the plane perpendicular to the axis of the tool according to the angular position of the rotary axes*** specified in the N3204 No. of the First Rot. Ax. and N3208 No. of the Second Rot. Ax. parameters.

For example:

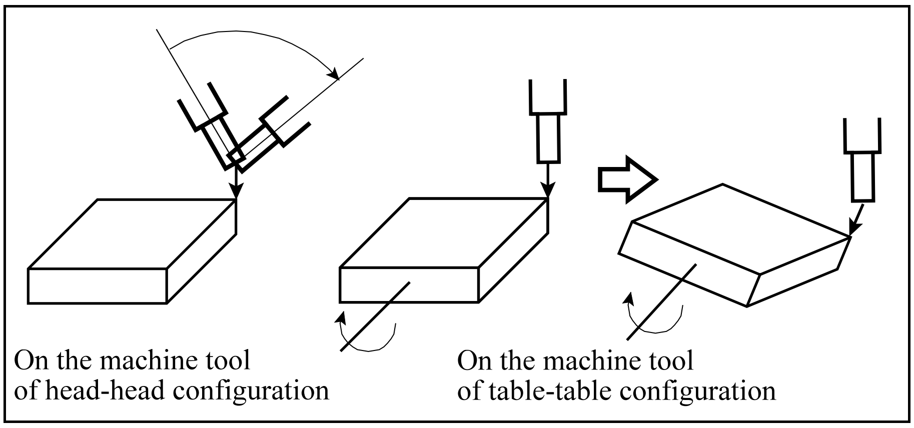
Let our machine tool be of head-head configuration, i.e.N3200 Mechanical Type=Head-Head; N3201 Tool Axis Direction=Z, i.e. the tool is parallel with the Z direction in the initial position of the rotary axes; N3204 No. of the First Rot. Ax.=C and N3208 No. of the Second Rot. Ax.=B. In such a case, ***when X or Y jog button is pushed or X or Y axis is selected for handwheel moving***, the ***motion will be executed in the plane perpendicular to the tool*** along all the three axes (X, Y, Z) might as well, taking positions of the B and C axes into account.



CN_TTCRF: Manual mode of moving around the tool center point

It is the acknowledge signal of the CP_TTCRF request for manual mode of moving around the tool center point. The control gives back the flag on the 5-axis machine tools of ***all three configurations in jog, incremental jog and handwheel modes***.

In the state CN_TTCRF=1, ***moving the rotary axes*** specified in the N3204 No. of the First Rot. Ax. or N3208 No. of the Second Rot. Ax. parameter, ***motion will be executed in such a way so that the position of the tool center point relative to the appropriate point of the workpiece will remain unchanged.***



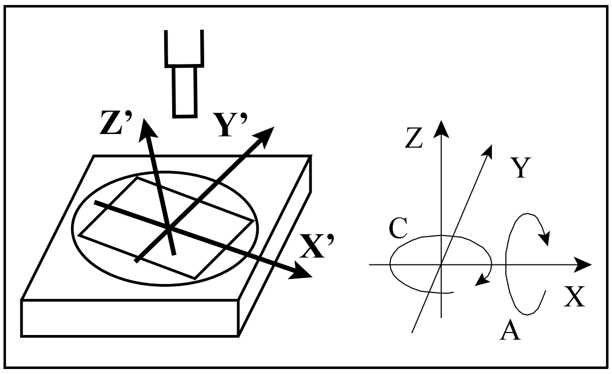
CN_TBLB: Manual mode of moving according to the table

It is the acknowledge signal of the CP_TBLB request for manual mode of moving according to the table. The control switches on the flag on the 5-axis machine tools of ***table-table or head-table configuration in jog, incremental jog and handwheel modes***. It does not work on the machine tools of head-head configuration.

In the state CN_TBLB=1, ***the control moves the linear axes*** (X, Y, Z) ***according to the angular position of the rotary axes specified in*** the N3204 No. of the First Rot. Ax. and N3208 No. of the Second Rot. Ax. ***parameters*** in such a way so that ***it considers the axis specified*** in the N3201 Tool Axis Direction ***parameter as direction perpendicular to the table, while the other two axes as directions being in the plane of the table***.

For example:

Let our machine tool be of table-table configuration, i.e.N3200 Mechanical Type=Table-Table; N3201 Tool Axis Direction=Z, i.e. the tool is parallel with the Z direction in the initial position of the rotary axes; N3204 No. of the First



Rot. Ax.=A and N3208 No. of the Second Rot. Ax.=C. In such a case, ***when X or Y jog button is pushed or Z axis is selected for handwheel moving***, the ***motion will be executed in the plane of the table***, taking positions of the A and C axes into account. ***When Z jog button is pushed or X or Y axis is selected for handwheel moving***, the ***motion will be executed in the direction perpendicular to the plane of the table***, taking positions of the A and C axes into account.

CN_ABSOFF: Not used

CN_TEST: Test mode

It is the acknowledge signal of the CP_TEST test mode request flag.

When the PLC sets the test mode request signal CP_TEST=1, the channel handler checks whether

– the status of the CN_TEST signal can be modified or not, i.e. whether the control is in any of the automatic (CN_AUTO=1), or MDI (CN_MDI=1) modes or not,

– if not, it changes the status of the CN_TEST flag to its opposite.

In status 1 of the CN_TEST flag the PLC may switch on the lamp of the test mode button (e.g: ML_TEST).

In status 1 of the CN_TEST flag the channel handler

– executes all interpolation blocks, the feed-rate blocks (G1, G2, G3, G33) too, with an increased feed-rate, by exponentially increasing the speed depending on the status of the override switch,

– it does not hand over any movement commands to the position control loop, thus the axes are not moving,

– it does not hand over any functions to the PLC: it does not issue any of the strobe signals CN_MnSTB, CN_SSTB, CN_TSTB, CN_AUXnSTB towards the PLC.

CN_MLCK: machine lock mode

It is the acknowledge signal of the CP_MLCK machine lock mode request.

When the PLC sets the machine lock request signal CP_MLCK=1, the channel handler checks whether

– the status of the CN_MLCK signal can be modified or not, i.e. whether the control is in any of the automatic (CN_AUTO=1), or MDI (CN_MDI=1) modes or not,

– if not, it changes the status of the CN_MLCK flag to its opposite.

In status 1 of the CN_MLCK flag the PLC may switch on the lamp of the machine lock mode button (e.g.: ML_MLCK).

In status 1 of the CN_MLCK flag, the channel handler

– will execute all interpolation blocks with the programmed feed-rate, by taking into consideration the override switches (feed-rate, rapid override),

– in case the dry run is switched on CN_DRRUN=1, it will execute the feed-rate blocks (G1, G2, G3, G33) by the increased feed-rate determined on the N0305 Max Feed parameter,
– it does not hand over any movement commands to the position control loop, thus the axes are not moving,
– it does not hand over any functions to the PLC: it does not issue any of the strobe signals CN_MnSTB, CN_SSTB, CN_TSTB, CN_AUXnSTB towards the PLC.

CN_DRRUN: Dry run mode

It is the acknowledge signal of the CP_DRRUN dry run request.
When the PLC sets the dry run request signal CP_DRRUN=1, the channel handler checks whether
– the status of the CN_DRRUN signal can be modified or not, i.e. whether the control is in any of the automatic (CN_AUTO=1), or MDI (CN_MDI=1) modes or not,
– if not, it changes the status of the CN_DRRUN flag to its opposite.
In status 1 of the CN_DRRUN flag the PLC may switch on the lamp of the dry run mode button (e.g.: ML_DRRUN).
In status 1 of the CN_DRRUN flag the channel handler
– executes the feed-rate blocks (G1, G2, G3, G33) by the increased feed-rate determined on N0305 Max Feed parameter,
– issues all movement commands to the position control loop, thus the axes will move, provided that the machine is not locked (CN_MLCK=0),
– hands over all functions to the PLC, provided that neither the machine (CN_MLCK=0), nor the function is locked (CN_MLCK=0).

CN_BKRST: Block restart state

It is the acknowledge signal of the CP_BKRST block restart request.
When the PLC switches on the signal for block restart request CP_BKRST=1, the channel handler checks whether
– there is a program interrupted in automatic execution or not, i.e. CN_INTD=1,
– if yes, it sets the CN_BKRST flag,
In status 1 of the CN_BKRST flag the PLC may switch on the lamp of the block restart button (e.g.: ML_BKRST).
In an automatic mode, upon the start, the channel handler
– issues to PLC the functions which has not been executed yet by the strobe signals through the handover-registers till the moment of interruption,
– it will return to the starting point of the interrupted block,
– it continues with machining from here.

CN_BKRET: Block return state

It is the acknowledge signal of the CP_BKRET block return request.

When the PLC sets the signal for requesting block return CP_BKRET=1, the channel handler checks whether

– there is a program interrupted in automatic execution or not, i.e. CN_INTD=1,

– if yes, it sets the CN_BKRET flag.

In status 1 of the CN_BKRET flag the PLC may switch on the lamp of the block return button (e.g.: ML_BKRET).

In an automatic mode, upon the start, the channel handler

– issues to PLC the functions which has not been executed yet by the strobe signals through the handover-registers till the moment of interruption,

– it returns to the interruption position of the interrupted block,

– it continues with machining from here.

CN_AHND: Zero point offset using handwheel is changed

It is the acknowledge signal of the CP_AHND zero point offset using handwheel request. It can be changed in automatic and manual data input (MDI) mode, even in start state too.

In this case, as it is usual in handwheel mode, selection of axis and step magnitude have to be enabled in the PLC. For safety reason, it is recommended to limit the step magnitude to the value of 0.001 mm, possibly 0.01 mm.

Due to the handwheel, the selected axis moves, and the displacement will be taken into account in the row Handwheel of the Zero point table, and not in the absolute position. Zero point offset, which is input by the handwheel, will be deleted by M30 and reset.

CN_WMCAXF: The feed parallel with the misalignment compensation is turned on

In the Jog, Incremental jog or Handwheel mode, it can be requested by the PLC on the CP_WMCAXF flag whether the manual moving the axes should occur in accordance with the original directions or the misalignment compensation. The CP_WMCAXF flag is the acknowledge signal of the request. If the value of the flag is

=0: the manual move occurs in accordance with the original axis directions;

=1: the manual move occurs in accordance with the rotated axis directions.

CN_1100: #1100 macro variable value (bit)
CN_1101: #1101 macro variable value (bit)
CN_1102: #1102 macro variable value (bit)
CN_1103: #1103 macro variable value (bit)
CN_1104: #1104 macro variable value (bit)
CN_1105: #1105 macro variable value (bit)
CN_1106: #1106 macro variable value (bit)
CN_1107: #1107 macro variable value (bit)
CN_1108: #1108 macro variable value (bit)
CN_1109: #1109 macro variable value (bit)
CN_1110: #1110 macro variable value (bit)
CN_1111: #1111 macro variable value (bit)
CN_1112: #1112 macro variable value (bit)
CN_1113: #1113 macro variable value (bit)
CN_1114: #1114 macro variable value (bit)
CN_1115: #1115 macro variable value (bit)
CN_1116: #1116 macro variable value (bit)
CN_1117: #1117 macro variable value (bit)
CN_1118: #1118 macro variable value (bit)
CN_1119: #1119 macro variable value (bit)
CN_1120: #1120 macro variable value (bit)
CN_1121: #1121 macro variable value (bit)
CN_1122: #1122 macro variable value (bit)
CN_1123: #1123 macro variable value (bit)
CN_1124: #1124 macro variable value (bit)
CN_1125: #1125 macro variable value (bit)
CN_1126: #1126 macro variable value (bit)
CN_1127: #1127 macro variable value (bit)
CN_1128: #1128 macro variable value (bit)
CN_1129: #1129 macro variable value (bit)
CN_1130: #1130 macro variable value (bit)
CN_1131: #1131 macro variable value (bit)

The user may set or reset macro variables #1100, #1101, ..., #1131 from a part program. These variables are available for the PLC program through the above flags. For example, the instruction

```
#1109=1
```

written into the part program will set the CN_1109=1 PLC flag. Instruction

#1109=0 will reset the CN_1109=0 PLC flag.

Bit-type channel control variables going from the PLC to NC

CP_START: Start request

When the operator pushes the start button, the PLC program has to check whether it is allowed to start from the machine’s side or not. If it is allowed, it will set the CP_START start request flag, by which it requests the start status from the channel handler.

In CP_START=1 status the channel handler will check whether a machining can be started in the channel or not. If yes, it will acknowledge the request with the CN_START=1 status.

In case of using an NCT machine control panel the MB_START flag will give the status of the start button.

CP_STOP: Stop request

When the operator pushes the stop button, the PLC program has to check whether it is allowed to stop from the machine’s side or not. If it is allowed, it will set the CP_STOP stop request flag, by which it requests the stop status from the channel handler.

In CP_STOP=1 status the channel handler will check whether the machining can be stopped in the channel or not. If yes, it will acknowledge the request with a CN_STOP=1 status.

In case of using an NCT machine control panel the MB_STOP flag will give the status of the stop button.

CP_JOG: Jog mode request
CP_INCR: Incremental jog mode request
CP_HNDL: Handwheel mode request
CP_REFP: Reference point travel mode request
CP_EDIT: Edit mode request
CP_AUTO: Automatic mode request
CP_MDI: Manual data input mode request

When the operator pushes any of the mode changing buttons, the PLC program has to check whether it is allowed to change the mode from the machine’s side or not. If it is allowed, it will set the CP_xxx mode change request flag belonging to the mode button pushed.

In CP_xxx=1 status, the channel handler will check whether the requested mode can be exchanged or not. If yes, it will acknowledge the request by the CN_xxx=1 status of the appropriate mode flag.

In case of using an NCT machine control panel the MB_JOG, MB_INCR, MB_HNDL, MB_REFP, MB_EDIT, MB_AUTO és az MB_MDI flags will give the statuses of the mode changing buttons.

CP_JOGRAP: Jog with rapid traverse

When the operator pushes the jog rapid traverse button, the PLC program has to check whether it is allowed to move the axes with rapid traverse, from the machine's side or not. If it is allowed, it will set the CP_JOGRAP jog with rapid traverse flag.

In the CP_JOGRAP=1 state the axis control moves the axes belonging to the channel

– in jog mode with a rapid traverse, if it is pushed together with the appropriate direction selecting jog button,

– during the execution of the program, it increases the programmed F feed-rate by the multiplier determined on the N0313 Feed Mult parameter.

In case of using an NCT machine control panel the MB_JOGRAP flag will give the status of the jog rapid button. The button's lamp (ML_JOGRAP) will be handled by the CP_JOGRAP flag.

CP_TAXF: Request for manual mode of moving in the tool direction

When the operator pushes the button, the PLC requests the mode by switching on the CP_TAXF flag. The control acknowledges the request by switching on the CN_TAXF flag. The CN_TAXF flag will be switched on by the control in one of the manual modes only. For detailed description of the manual mode of moving in the tool direction, see the description of the CN_TAXF flag.

CP_TRGAF: Request for manual mode of moving in the direction perpendicular to the tool

When the operator pushes the button, the PLC requests the mode by switching on the CP_TRGAF flag. The control acknowledges the request by switching on the CN_TRGAF flag. The CN_TRGAF flag will be switched on by the control in one of the manual modes only. For detailed description of the manual mode of moving in the direction perpendicular to the tool, see the description of the CN_TRGAF flag.

CP_TTCRF: Request for manual mode of moving around the tool center point

When the operator pushes the button, the PLC requests the mode by switching on the CP_TTCRF flag. The control acknowledges the request by switching on the CN_TTCRF flag. The CN_TTCRF flag will be switched on by the control in one of the manual modes only. For detailed description of the manual mode of moving around the tool center point, see the description of the CN_TTCRF flag.

CP_TBLB: Request for manual mode of moving according to the table

When the operator pushes the button, the PLC requests the mode by switching on the CP_TBLB flag. The control acknowledges the request by switching on the CN_TBLB flag. The CN_TBLB flag will be switched on by the control in one of the manual modes only. For detailed description of the manual mode of moving according to the table, see the description of the CN_TBLB flag.

CP_TLCM: Tool length offset measurement mode on

Tool length offset measurement mode can be used in lathe channels in which a tool setter has been mounted. The length offset measurement mode becomes switched on by the PLC by setting the CP_TLCM flag.

The setting of the flag may be carried out for the **unfolding of the tool setter**, or upon pushing any **button** specified by the PLC program.

When the PLC sets the CP_TLCM bit

– the channel automatically switches into Jog mode: CN_JOG=1. The mode cannot be left till the CP_TLCM flag state is TRUE.

– In such cases the feed-rate of the movement made by the jog buttons is taken by the interpolator from the N0319 T Meas Feed parameter. It is possible to move only one axis at one time, i.e. it excludes the moving of several axes at the same time.

– the display side exchanges the offset setting screen belonging to the channel of the offset measurement flag.

CP_S2TS: Selection of tool offset setter belonging to S2

A channel is able to handle the signals of a maximum of 2 tool setters. Two tool setters can be used, e.g., on lathes with sub-spindle.

The system selects the probe to be used based on the CP_S2TS PLC bit status. If the CP_S2TS value

=0: it will use the probe selected on the N3012 Sensor Input of Tool Setter S1 parameter,

=1: it will use the probe selected on the N3013 Sensor Input of Tool Setter S2 parameter.

CP_WPCM: Workpiece zero offset measurement mode on

The workpiece zero offset measurement mode can be used in lathe channels and usually it measures the Z-direction offset of the of the workpiece. The workpiece zero offset measurement mode becomes switched on by the PLC by setting the CP_WPCM flag.

The switch-on of the flag may be carried out during the switch-on of the probe, or upon any button determined by the PLC program.

When the PLC switches on the CP_WPCM bit

– the NC side will automatically switch into Jog mode: CN_JOG=1. The mode cannot be quit till the time the status of the CP_WPCM flag is TRUE.

– in such cases the feed-rate of the movement by the jog buttons is taken by the interpolator from the N0319 T Meas Feed parameter. It is possible to move only one axis at one time, i.e. it excludes the moving of several axes at the same time.

– the display side exchanges the offset screen belonging to the channel of the flag of the offset measurement.

CP_S2WS: Selection of work offset setter belonging to S2

A channel is able to handle the signals of a maximum of 2 work zero setters. Two work zero setters can be used, e.g., on lathes with sub-spindle.

The system selects the probe to be used based on the CP_S2WS PLC bit status. If the CP_S2WS value

=0: it will use the probe selected on the a N3014 Sensor Input of Workpiece Setter S1 parameter,

=1: it will use the probe selected on the a N3015 Sensor Input of Workpiece Setter S2 parameter.

It uses the latter one if on a one-turret machine with sub-spindle a work setter probe is mounted to both spindles.

CP_AHND: Request for zero point offset using handwheel

When the operator pushes the button, the PLC requests the mode by switching on the CP_AHND flag. The control acknowledges the request by switching on the CN_AHND flag. The CN_AHND flag will be switched on by the control in automatic or manual data input (MDI) mode only. For detailed description of the zero point offset using handwheel, see the description of the CN_AHND flag.

CP_WMCAXF: Request for feed parallel with the misalignment compensation

In the Jog, Incremental jog or Handwheel mode, it can be requested by the PLC that the manual moving the axes should occur in accordance with the misalignment compensation.:

=0: request for manual move in accordance with the original axis directions,

=1: request for manual move in accordance with the rotated axis directions.

CP_SGLBK: Single block mode on

In case the single block mode button is pushed, the PLC program has to switch the value of the CP_SGLBK flag to its opposite.

In CP_SGLBK=1 status the channel handler, after executing all blocks takes up a stop status CN_STOP=1, i.e. it stops machining.

In case of using an NCT machine control panel the MB_SGLBK flag gives the single block mode button status. The lamp of the button (ML_SGLBK) is handled by the CP_SGLBK flag.

CP_CNDSP: Conditional stop on

In case the conditional stop button is pushed, the PLC program has to switch the value of the CP_CNDSP flag to its opposite.

If the execution of the program runs on M01 code, the channel handler will check the status of the CP_CNDSP flag. In CP_CNDSP=1 status it will stop machining and take up a stop status CN_STOP=1. In an opposite case CP_CNDSP=0 there is no stopping.

In case of using an NCT machine control panel the MB_CNDSP flag will give the status of the conditional stop button. The lamp of the button (ML_CNDSP) will be handled by the CP_CNDSP flag.

CP_TEST: Test mode request

When the operator pushes the test mode button, the PLC program will set the CP_TEST test mode request flag, by which it will request the test mode from the channel handler.

In CP_TEST=1 status the channel handler will check whether it is possible to switch on the test mode in the channel. If yes, it will acknowlwdge the request with a CN_TEST=1 status.

In case of using an NCT machine control panel the MB_TEST flag will give the status of the test button.

CP_MLCK: Machine lock mode request

When the operator pushes the machine lock button, the PLC program will set the CP_MLCK machine lock mode request flag, by which it will request the machine lock mode from the channel handler.

In CP_MLCK=1 status the channel handler will check whether it is possible to switch on the machine lock mode in the channel. If yes, it will acknowledge the request by the CN_MLCK=1 status.

In case of using an NCT machine control panel az MB_MLCK flag will give the machine lock button's status.

CP_DRRUN: Dry run request

When the operator pushes the dry run button, the PLC program will set the CP_DRRUN dry run request flag, by which it will request the dry run mode from the channel handler.

In CP_DRRUN=1 status the channel handler will check whether it is possible to switch on the dry run mode in the channel. If yes, it will acknowledge the request by the CN_DRRUN=1 status.

In case of using an NCT machine control panel, MB_DRRUN flag will give the dry run button's status.

CP_BKRST: Block again request

When the operator pushes the block again button, the PLC program will write the CP_BKRST block again request flag to 1, by which it requests from the channel handler to restart the block.

In CP_BKRST=1 status the channel handler will check whether it is possible to switch on the restart of the block in the channel. If yes, it will confirm the request by the CN_BKRST=1 state.

In case of using an NCT machine control panel, MB_BKRST flag will give the block again button's status.

CP_BKRET: Block return request

When the operator pushes the block return button, the PLC program is going to set the CP_BKRET block return request flag, by which it will request from the channel handler to return to the interruption point.

In CP_BKRET=1 status the channel handler will check whether it is possible to switch on the block return in the channel. If yes, it will acknowledge the request by the CN_BKRET=1 status.

In case of using an NCT machine control panel the MB_BKRET flag will give the block return button's status.

CP_FLCK: Function lock request

When the operator pushes the Function lock button, the PLC program will set the CP_FLCK function lock request flag, by which it requests from the channel handler the closing of the function issuance.

In CP_FLCK=1 status the channel handler will check whether the function lock mode can be switched on or not in the channel. If yes, it will acknowledge the request by the CN_FLCK=1 state.

In case of using an NCT machine control panel, MB_FLCK flag will give the status of the function lock button.

CP_ABSOFF: Not used

CP_CNDBK_1: Conditional block 1 on
CP_CNDBK_2: Conditional block 2 on
CP_CNDBK_3: Conditional block 3 on
CP_CNDBK_4: Conditional block 4 on
CP_CNDBK_5: Conditional block 5 on
CP_CNDBK_6: Conditional block 6 on
CP_CNDBK_7: Conditional block 7 on
CP_CNDBK_8: Conditional block 8 on

It is possible to set 8 different conditions per channel. In case in the part program a block starts with the a /n (n=1, ..., 8) instruction, the channel handler will check before the execution of the block whether the nth condition is fulfilled or not, i.e. whether the status of the CP_CNDBK_n flag is 1 or not. If
– CP_CNDBK_n=0 it will execute the block,
– CP_CNDBK_n=1 it will not execute the block and it will step on the next one.
In case of using an NCT machine control panel, the MB_CNDBK flag will give the status of a conditional block button. The lamp of the button (ML_CNDBK) will be handled by the CP_CNDBK_n flag selected by the PLC programmer for the button.

CP_FIN: Finish signal, all functions executed

PLC program indicates by a CP_FIN=1 state to the channel handler that it has carried out all functions. In CP_FIN=1 state the channel handler
– in a start status, will take the next block from the buffer and execute it,
– in case of execution in single block mode, it will take up a stop status and wait for the start,
– at the end of the program, i.e. in case of an empty buffer, it will reset the start status.
In the PLC program it is allowed to set the CP_FIN signal if the statuses of all the function acknowledge signals (CP_MnACK, CP_SACK, CP_TACK, CP_AUXnACK) are 1 and from the PLC program’s side there aren’t other reasons for suspending the execution of the part program.

CP_RST: Not used

CP_RSTREW: Not used

CP_CSREQ: Not used

CP_NOWT: Waiting for synchronization M code cancelled

On multi-channel controls it is necessary to suspend the execution of the part program running in one of the channels till the time one or more other channels reach a certain point in the execution of the program. We call this the synchronization of channels. It is possible to carry out the synchronization of channels by M codes. We can determine

an M code range on the N2201 Waiting M Codes Min and N2202 Waiting M Codes Max parameters.
The synchronization will be carried out fully by the NC, and PLC does not have to deal with it.
In case we have written a program in a way that it will wait for the synchronization M code of the other channel, or channels but we would like to run the program only alone, without the synchronization to programs running in the other channels, the channel handler
– in CP_NOWT=1 status will skip the synchronization M codes, i.e. it will not wait for the channels in which the flag CP_NOWT is true.

CP_MINT: Interrupt macro calling

CP_TMREN: Free-purpose timer enable
In the Timer row of Timers/counters window of the control, a time can be read out which shows the value of the free-purpose timer, in the form of day/hour/minute/second/millisecond units.
The free-purpose timer
– will be started by the PLC by the CP_TMREN=1 status,
– will be stopped by the PLC by the CP_TMREN=0 status.
The timer value can be overwritten/read out
– from the control panel,
– from the part program, through #3001 macro variable.

CP_TSBD: Not used

CP_EGBRRQ: Retraction request in EGB mode
In case the channel handler executes a G81.8 EGB function (electronic gear box) it will set the CN_EGBMD flag.
In status 1 of the CN_EGBMD flag the PLC may request the retraction of the tool (hob) by setting the CP_EGBRRQ flag. Then the NC
– will pull out the tool in the direction and to the distance determined on the N1804 Retr. Dist. parameter,
– at the end of the retraction the channel handler will set the CN_RTRFIN flag.
After that, the PLC has to reset the CP_EGBRRQ=0 flag.

CP_M1ACK: M function 1 acknowledge
CP_M2ACK: M function 2 acknowledge
CP_M3ACK: M function 3 acknowledge
CP_M4ACK: M function 4 acknowledge
CP_M5ACK: M function 5 acknowledge
CP_M6ACK: M function 6 acknowledge
CP_M7ACK: M function 7 acknowledge
CP_M8ACK: M function 8 acknowledge

In a part program, it is possible to write into a block a maximum of 8 different M codes, i.e. in a block 8 different M codes can be handed over by the NC to the PLC.
In case into the part program an M function has been programmed, the channel handler will set the CN_MnSTB strobe signal for the period of 1 PLC cycle. Upon the signal, PLC will decode the function based on the received code.
If the received code contains a function existing on the machine, the PLC will
– CP_MnACK=0 reset the acknowledge flag corresponding to the strobe signal.
After the execution of the function the PLC
– CP_MnACK=1 will set the acknowledge flag.
Based on the status of the CP_MnACK flags, the channel handler registers which M functions have been executed from the block under execution. By interrupting, then restarting the program, the not-yet-executed functions will be given out again by the channel handler to the PLC.
After power-on, the state of acknowledge signals shall be initialized by the PLC program (CP_MnACK=1).

CP_SACK: S function acknowledge

In case in the part program an S function has been programmed, the channel handler will set the CN_SSTB flag for the period of 1 PLC cycle. Upon the strobe signal the PLC
– CP_SACK=0 will reset the S function acknowledge flag.
After the execution of the function the PLC
– CP_SACK=1 will set the acknowledge flag.
Based on the status of the CP_SACK flags, the channel handler registers whether the S function has been executed from the block under execution or not. By interrupting, then restarting the program, the not-yet-executed S function will be given out again by the channel handler to the PLC.
After power-on, the state of acknowledge signals shall be initialized by the PLC program (CP_SACK=1).

CP_TACK: T function acknowledge

In case in the part program a T function has been programmed, the channel handler will set the CN_TSTB flag for the period of 1 PLC cycle. Upon the strobe signal the PLC

– CP_TACK=0 will reset the T function acknowledge flag.

After the execution of the function the PLC

– CP_TACK=1 will set the acknowledge flag.

Based on the status of the CP_TACK flags, the channel handler registers whether the T function has been executed from the block under execution or not. By interrupting, then restarting the program, the not-yet-executed T function will be given out again by the channel handler to the PLC.

After power-on, the state of acknowledge signals shall be initialized by the PLC program (CP_TACK=1).

CP_AUX1ACK: Auxiliary function 1 acknowledge
CP_AUX2ACK: Auxiliary function 2 acknowledge
CP_AUX3ACK: Auxiliary function 3 acknowledge

In case in the part program an auxiliary function has been programmed, the channel handler will set the CN_AUXnSTB flag, belonging to the auxiliary function, for a period of 1 PLC cycle. Upon the strobe signal the PLC

– CP_AUXnACK=0 resets the acknowledge flag of the appropriate auxiliary function.

After the execution of the function, the PLC

– CP_AUXnACK=1 sets the appropriate acknowledge flag.

The channel handler, based on the status of CP_AUXnACK flags registers whether from the block under execution it has executed the given auxiliary function or not. By interrupting the program and then restarting it again the channel handler will issue the not yet executed auxiliary functions to the PLC.

After power-on the state of acknowledge signals shall be initialized by the PLC program (CP_AUXnACK=1).

CP_GACK: Not used

CP_HOLD: Feed-hold for all axes in the channel

If the PLC sets the flag, the channel handler unconditionally stops the movement of all axes belonging to the channel.

It differs from the CP_STOP flag by the fact that while in the override and stop disabled status (G63) the stop is ineffective, the CP_HOLD will be effective also in such cases.

Due to the above, during thread cutting or tapping (G74, G84) at the shutdown of the spindle CP_HOLD flag shall be used for stopping the feed-rate, respectively, the feed-rate can be stopped through the CP_HOLD flag by shutting down the spindle.

CP_CBFEN: Cutting block feed enable

The channel handler requests to enable of the feed-rate movement from the PLC by setting the CN_CBFR=1 flag.
The feed-rate (movement) will not start until it is enabled by the PLC on the CP_CBFEN=1 flag. The enable of the feed-rate can be linked to various conditions from the PLC program's side, for example, for spindle rotation on cutting machines, for the switch-on of laser or flame on cutting machines, etc.

CP_FHNDL: Feed-rate from handwheel mode on

If CP_FHNDL flag is set during the execution of the program, the interpolator in the given channel will not move according to the programmed feed-rate (G94, or G95 F), but according to pulses arriving from the handwheel. The speed of the movement depends on:
– the size of increment selection
– the speed of rotation of the handwheel.
By using the CP_HNDLSn flags it has to be selected which handwheel shall be used by the channel.
The function can be used, e.g., on multi-channel machines for the collision test of programs.

CP_OVC: Not used

CP_HNDLS1: Feed-rate from the 1. handwheel
CP_HNDLS2: Feed-rate from the 2. handwheel
CP_HNDLS3: Feed-rate from the 3. handwheel
CP_HNDLS4: Feed-rate from the 4. handwheel

In case the feed-rate from handwheel mode is switched on (CP_FHNDL=1) the channel handler can determine on the above flags which one to use from among the 4 possible handwheels for the function, by setting the appropriate CP_HNDLSn flag.

CP_LIM1DIS: Stroke range 1 disable
CP_LIM2DIS: Stroke range 2 disable
CP_LIM3DIS: Stroke range 3 disable

In the control it is possible to specify 3 different stroke ranges per axis which can be enabled on parameter bits the #0 RE1, #1 RE2, #2 RE3 of the N1000 Range Enable parameter.
The PLC program, on all axes belonging to a given channel, may disable any of the 3 stroke ranges, by setting the appropriate CP_LIMnDIS flag.

☞ ***Attention!** In such a case the control will not handle the disabled stroke range!*

CP_LIMSEL: 1B stroke range selection for all axes

It is a stroke range selection per channel from the PLC for every axis of the channel. In case the

N1001 StrkCont parameter #4 ABA=1

in the given channel the CP_LIMSEL PLC flag will be effective. CP_LIMSEL flag will tell that in the range No. 1 which stroke range parameter group

CP_LIMSEL=0: 1A,

CP_LIMSEL=1: 1B

shall be valid for all axes of the channel, in both directions.

The modification of the stroke range selection shall be carried out in a standstill status of the axes.

☞ ***See also:*** *AP_LIMSELP and AP_LIMSELN flags.*

CP_CHOPON: Not used

CP_SGOEN: Use of second geometry offset table enabled

The flag is effective exclusively in a lathe channel.

On lathes using a linear turret, it is advisable to introduce a second geometry offset table. The second geometry offset table has the same length as the first one. When referencing to the geometry offset, values stored in the second geometry offset table, in case of fulfilment of certain conditions, shall be added to the values stored in the first table.

In the second geometry offset table the position of the tool holders in the machine coordinate system can be determined for axes X, Y, Z. By this it becomes possible to determine in the first geometry offset table the real X, Y, Z length of the tool, i.e., in the first geometry table the values measured in an external tool measuring device can be determined directly. Thus the offset value taken into consideration will be:

offset = 1. geometry offset + 2. geometry offset + wear offset

If the #4 SGC bit of the N1414 Comp. Config on Lathes parameter is:

=0: the second geometry offset table does not exist and will not be displayed,

=1: the second geometry offset table does exist, will be displayed and the taking into consideration of offset values are controlled by PLC flags.

In case the CP_SGOEN second geometry offset table enable flag is

=0 the channel handler will not take into consideration the second geometry offset,

=1 the channel handler will take into consideration the second geometry offset during the calling of the offset by code T.

See also the CP_SGOX, CP_SGOY, CP_SGOZ flags.

The CP_SGOEN can be used if, for example, in a lathe channel where there is also a rotary and linear turret, and it is not necessary to call the 2. geometry offset for tools being in the rotary turret, but it is necessary for the tools in the linear one.

CP_SGOX: Second geometry tool offset selection on axis X
CP_SGOY: Second geometry tool offset selection on axis Y
CP_SGOZ: Second geometry tool offset selection on axis Z

If the use of the 2. geometry offset is enabled CP_SGOEN=1 in the channel, on axes X, Y, Z of the channel their use can be separately enabled:
CP_SGOX=1 on axis X,
CP_SGOY=1 on axis Y,
CP_SGOZ=1 on axis Z
enables the use of the 2. geometry offset.
See also the CP_OSGNX, CP_OSGNY, CP_OSGNZ flags.

CP_OSGNX: Tool offset direction selection signal on axis X
CP_OSGNY: Tool offset direction selection signal on axis Y
CP_OSGNZ: Tool offset direction selection signal on axis Z

If the use of the 2. geometry offset is enabled in the channel CP_SGOEN=1 and also on the given axis by flags CP_SGOX, CP_SGOY, CP_SGOZ, then we can control the sign (direction) of compensation amount, per channel:
the offset will be taken into consideration
CP_OSGNX=1 on axis X with an opposite sign,
CP_OSGNY=1 on axis Y with an opposite sign,
CP_OSGNZ=1 on axis Z with an opposite sign
offset = −(1. geometry offset + 2. geometry offset + wear offset)
It can be used if the direction of any of the tools mounted on the axis is the same as the positive direction of the axis.

CP_ROVLD: Overlapping of rapid traverse blocks disable

The flag can be used if the overlapping of rapid traverse blocks is enabled on the bit #0 ROL of parameter N0407 Acc Contr.
In case the overlapping has to be disabled in certain cases, the PLC program can do that by setting CP_ROVLD=1 flag.

CP_RLSOT3: Not used

CP_1000: #1000 macro variable (bit)
CP_1001: #1001 macro variable (bit)
CP_1002: #1002 macro variable (bit)
CP_1003: #1003 macro variable (bit)
CP_1004: #1004 macro variable (bit)
CP_1005: #1005 macro variable (bit)

CP_1006: #1006 macro variable (bit)
CP_1007: #1007 macro variable (bit)
CP_1008: #1008 macro variable (bit)
CP_1009: #1009 macro variable (bit)
CP_1010: #1010 macro variable (bit)
CP_1011: #1011 macro variable (bit)
CP_1012: #1012 macro variable (bit)
CP_1013: #1013 macro variable (bit)
CP_1014: #1014 macro variable (bit)
CP_1015: #1015 macro variable (bit)
CP_1016: #1016 macro variable (bit)
CP_1017: #1017 macro variable (bit)
CP_1018: #1018 macro variable (bit)
CP_1019: #1019 macro variable (bit)
CP_1020: #1020 macro variable (bit)
CP_1021: #1021 macro variable (bit)
CP_1022: #1022 macro variable (bit)
CP_1023: #1023 macro variable (bit)
CP_1024: #1024 macro variable (bit)
CP_1025: #1025 macro variable (bit)
CP_1026: #1026 macro variable (bit)
CP_1027: #1027 macro variable (bit)
CP_1028: #1028 macro variable (bit)
CP_1029: #1029 macro variable (bit)
CP_1030: #1030 macro variable (bit)
CP_1031: #1031 macro variable (bit)

The user may query 32 PLC flags from the part program through macro variables. The macro variables, available for the user per each channel for the communication with the PLC are: #1000, #1001, ..., #1031.

For example, if the instruction

```
#1025EQ1 GOTO30
```

written into the part program will jump on the block N30, if CP_1025 flag is 1.

7.15.2 DWORD-type Channel Control Variables

Inputs		**Outputs**	
Symbol	**Description**	**Symbol**	**Description**
CN_1	Channel control bits handed over from NC to PLC DWORD1	**CP_1**	Channel control bits handed over from PLC to NC DWORD1
CN_2	Channel control bits handed over from NC to PLC DWORD2	**CP_2**	Channel control bits handed over from PLC to NC DWORD2
CN_3	Channel control bits handed over from NC to PLC DWORD3	**CP_3**	Channel control bits handed over from PLC to NC DWORD3
CN_1132	#1132 macro variable value (DWORD)	**CP_4**	Channel control bits handed over from PLC to NC DWORD4
CN_M1C	Code of M function 1 (DWORD)	**CP_JOGFD**	Jog feed-rate selection (DWORD)
CN_M2C	Code of M function 2 (DWORD)	**CP_SINP**	Number of spindle machining (1,2...) (DWORD)
CN_M3C	Code of M function 3 (DWORD)	**CP_CSAX**	Not used
CN_M4C	Code of M function 4 (DWORD)	**CP_POLYSL**	Number of slave spindle of poly-gonal turning (1,2...) (DWORD)
CN_M5C	Code of M function 5 (DWORD)	**CP_ACTT**	Number of the current tool (DWORD)
CN_M6C	Code of M function 6 (DWORD)	**CP_MGR1**	M code in group 1 to be displayed (DWORD)
CN_M7C	Code of M function 7 (DWORD)	**CP_MGR2**	M code in group 2 to be displayed (DWORD)
CN_M8C	Code of M function 8 (DWORD)	**CP_MGR3**	M code in group 3 to be displayed (DWORD)
CN_SC	Code of S function (DWORD)	**CP_MGR4**	M code in group 4 to be displayed (DWORD)
CN_SSEL	Number of spindle to which S code refers (DWORD)	**CP_MGR5**	M code in group 5 to be displayed (DWORD)
CN_RNGREQ	Number of range belonging to code S (DWORD)	**CP_MGR6**	M code in group 6 to be displayed (DWORD)
CN_TC	Code of T function (DWORD)	**CP_MGR7**	M code in group 7 to be displayed (DWORD)
CN_AUX1C	Code of auxiliary function 1 (DWORD)	**CP_MGR8**	M code in group 8 to be displayed (DWORD)
CN_AUX2C	Code of auxiliary function 2 (DWORD)	**CP_MGR9**	M code group 9 to be displayed (DWORD)
CN_AUX3C	Code of auxiliary function 3 (DWORD)	**CP_MGR10**	M code group 10 to be displayed (DWORD)
CN_CTSPN	Spindle speed in case of G96 (DWORD)	**CP_MGR11**	M code in group 11 to be displayed (DWORD)

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
CN_NMAX	Value of the maximum spindle speed in case of G96 (G92 S_) (DWORD)	**CP_MGR12**	M code in group 12 to be displayed (DWORD)
CN_MGZNO	Number of magazine belonging to the T code (DWORD)	**CP_MGR13**	M code in group 13 to be displayed (DWORD)
CN_POTNO	Pot number belonging to the T code (DWORD)	**CP_MGR14**	M code in group 14 to be displayed (DWORD)
		CP_MGR15	M code in group 15 to be displayed (DWORD)
		CP_MGR16	M code in group 16 to be displayed (DWORD)
		CP_1032	#1032 macro variable (DWORD)
		CP_OFFSNO	Number of offset to be used at tool measurement (DWORD)

DWORD-type channel control variables going from the NC to PLC

CN_1132: #1132 macro variable value (DWORD)

The user may set or reset macro variables #1100, #1101, ..., #1131 from a part program. These PLC flags may be written also in a DWORD format through the #1132 macro variable.

Instruction

#1132=130=$2^7 + 2^1$

sets the flags CN_1101=1 and CN_1107=1.

In the same way, the PLC may make a query in a DWORD format on the CN_1100, CN_1101, ... CN_1131 bits through the CN_1132 variable.

CN_M1C: Code of M function 1 (DWORD)
CN_M2C: Code of M function 2 (DWORD)
CN_M3C: Code of M function 3 (DWORD)
CN_M4C: Code of M function 4 (DWORD)
CN_M5C: Code of M function 5 (DWORD)
CN_M6C: Code of M function 6 (DWORD)
CN_M7C: Code of M function 7 (DWORD)
CN_M8C: Code of M function 8 (DWORD)

In a part program we can write a maximum of 8 different M codes into a block, i.e. in a block 8 different M codes can be handed over by the NC to the PLC.

In case into the part program an M function has been programmed, the channel handler

– will write the value of the M code into the CN_MnC (n=1, 2, ..., 8) register

– the CN_MnSTB strobe signal will be set by the NC for the period of 1 PLC cycle.
The values of the 8 different M codes the control writes into 8 different ***CN_MnC*** (n=1, 2, ..., 8) registers. There are 8 strobe signals belonging to the 8 code registers CN_MnSTB (n=1, 2, ...8). Upon the strobe signal, based on the received code, the function is decoded by the PLC.
If the received code covers a function existing on the machine, the PLC will reset the CP_MnACK acknowledge flag corresponding to the strobe signal. It is enabled to set the CP_MnACK acknowledge flag exclusively after the full execution of the function.

CN_SC: Code of S function (DWORD)

In case in the part program an S function has been programmed, the channel handler
– will write the value of the spindle speed programmed on the S function to the CN_SC register,
– will write into the CN_SSEL register the number of the referred spindle (1...16)
– will write into the CN_RNGREQ register the code of the range belonging to the programmed spindle speed (11...18),
– then it will set the CN_SSTB flag for the period of 1 PLC cycle.
Upon the strobe signal, the PLC decodes the codes written into the above registers and it will reset the CP_SACK acknowledge signal. It is enabled to set the CP_SACK acknowledge flag only after the full execution of the function.
The code received in the CN_SC register shall be written into the SP_PRG register of the spindle appointed by the CN_SSEL register during the execution of the function.

CN_SSEL: Number of spindle to which S code refers (DWORD)

In case in the part program an S function has been programmed, the channel handler
– will write the value of the spindle speed programmed on the S function to the CN_SC register,
– will write into the CN_SSEL register the number of the referred spindle (1...16)
– will write into the CN_RNGREQ register the code of the range belonging to the programmed spindle speed (11...18),
– then it will set the CN_SSTB flag for the period of 1 PLC cycle.
Upon the strobe signal, the PLC decodes the codes written into the above registers and it will reset the CP_SACK acknowledge signal. It is enabled to set the CP_SACK acknowledge flag only after the full execution of the function.
The channel handler sets the number of spindle set in the CN_SSEL register according to the reference made in the part program.
We can refer to the spindles from a part program under address S, or in case of an extended name, under an S+2-character address. In case of a reference under a extended address we can determine the 2nd and 3rd character of the address on the N0605 Spindle Name2 and N0606 Spindle Name3 parameters. The first character is

always and mandatorily S. In case the last character is a number, during data input we have to use an = mark.
If there are several spindles on the machine and we do not want to use extended addresses, we can refer to a given spindle under addresses' S and P, where on S we determine the spindle speed, and on P the number of the spindle to which the S code refers.
The channel handler, based on the programmed Sxx, or Ss Pp code determines to which spindle the reference has been made in the part program, and it will write the number of the referred spindle to the CN_SSEL register.
In case of using extended addresses, let's say the address of the 3rd spindle is S3.

```
In case of programming S3=1000,
```

the handover registers of the appropriate channel will receive the following data:

```
CN_SC=1000
CN_SSEL=3
```

In case of referring to addresses S and P the above reference is:

```
S1000 P3
```

The same values will get into the handover registers.
On the N0604 Default Spindle parameter we can appoint a spindle per channel, to which we can refer under address S. If e.g. the parameter value is 2, we can refer to the 2nd spindle both under address S2 and S. In both cases CN_SSEL=2 will be handed over by the channel handler.
The PLC program issues all commands to the spindle determined in the ***CN_SSEL*** register
– M3, M4, M5 and M19,
– the loop closing command set on the N0823 M Code for Closing S Loop parameter,
– the synchronization command referring to the slave spindle (M code),
– and the M code serving for the selection of slave spindle of polygonal turning.
At the same time several spindles can be rotated in a channel from the program. In such cases let's write:

```
N10 S1=1000 M3
N20 S2=1500 M4
```

If a spindle has to be stopped from a program, let's write:

```
N100 S1=0 M5
N110 S2=0 M5
```

CN_RNGREQ: Number of range belonging to code S (DWORD)
In case in the part program an S function has been programmed, the channel handler
– will write the value of the spindle speed programmed on the S function to the CN_SC register,
– will write into the CN_SSEL register the number of the referred spindle (1...16)

– will write into the CN_RNGREQ register the code of the range belonging to the programmed spindle speed (11...18),
– then it will set the CN_SSTB flag for the period of 1 PLC cycle.

Upon the strobe signal, the PLC decodes the codes written into the above registers and it will reset the CP_SACK acknowledge signal. It is enabled to set the CP_SACK acknowledge flag only after the full execution of the function.

The channel handler decodes the range of the given spindle into which the programmed spindle speed falls in:

– In case the speeds between the ranges overlap each other, as usually on lathes, the CN_RNGREQ register cannot be used (a decoding is carried out in such a case, too). In such a case the range change shall be selected by an M code (mandatorily to M11, M12, ..., M18), in order to enable the user to decide which will be the optimum range of cutting.
– In case the speeds between the ranges do not overlap each other, as usually on milling machines, the CN_RNGREQ register can be used.

In such a case the range change can be carried out in a PLC program without programming separate M functions, it is not necessary to introduce M functions.
The channel handler will decode the ***CN_RNGREQ*** register in the following way:
If ***S≤N0650 R1 S Max*** parameter value: then ***CN_RNGREQ=11***,
if ***S≤N0651 R2 S Max*** parameter value: then ***CN_RNGREQ=12***,
and so on.

CN_TC: Code of T function (DWORD)

In case in the part program a T function has been programmed, the channel handler
– will write into the CN_TC register the number of the tool programmed on the T function, in a lathe channel ***cutting down the offset number*** called under address T,
– will write into the CN_MGZNO register, in which magazine can the referred tool be found,
– will write into the CN_POTNO register, in which pot of the given magazine can the referred tool be found,
– will check whether the life of the referred tool or tool group has expired or not and it will set the CN_TLLE flag accordingly,
– then it will set the CN_TSTB flag for the period of 1 PLC cycle.

Upon the strobe signal, the PLC decodes the codes written into the above registers and it will reset the CP_TACK acknowledge signal. It is enabled to set the CP_TACK acknowledge signal only after the full execution of the function.
Into the CN_TC register always the number (type number) of tool referred to in the part program shall be written, even if in case of using the tool management table, when there can be more tools with the same type number in the magazine. In such a case the

position of the tool to be changed can be read out from CN_MGZNO and CN_POTNO registers.

CN_AUX1C: Code of auxiliary function 1 (DWORD)
CN_AUX2C: Code of auxiliary function 2 (DWORD)
CN_AUX3C: Code of auxiliary function 3 (DWORD)
On the N1332+n Aux Fu Addrn (n=1..3) parameter it is possible to appoint three different ones from among the A, B, C, U, V, W addresses. In case in a part program, we refer to the address of the three auxiliary functions determined here, the channel handler
– will write the value programmed under the address of the auxiliary function into the ***CN_AUXnC*** (n=1...3) register (always an integer DWORD),
– then it will set the CN_AUXnSTB (n=1...3) flag for the period of 1 PLC cycle.
Upon the strobe signal, the PLC decodes the codes written into the above registers and it resets the CP_AUXnACK (n=1...3) acknowledge signal. It is enabled to set the CP_AUXnACK acknowledge flag only after the full execution of the function.

CN_CTSPN: Spindle speed in case of G96 (DWORD)
In case the channel is in a G96 (constant surface speed calculation) status,
– the NC will set the CN_CSURF flag,
– the NC will calculate the spindle speed belonging to the instantaneous coordinate and write it into the CN_CTSPN register,
– it will write the maximum spindle speed programmed by the G92 S instruction into the CN_NMAX register.
The task of the PLC program is to copy the value received in the ***CN_CTSPN*** register into the ***SP_PRG*** register of the appropriate (determined by the CP_SINP register) spindle in the CN_CSURF=1 status.

CN_NMAX: Value of the maximum spindle speed in case of G96 (G92 S_) (DWORD)
If the channel is in a G96 (constant surface speed calculation) status,
– the NC will set the CN_CSURF flag,
– the NC will calculate the spindle speed belonging to the instantaneous coordinate and write it into the CN_CTSPN register,
– it will write the maximum spindle speed programmed by the G92 S instruction into the CN_NMAX register.
The PLC programmer needn't cut the value written into SP_PRG register with the value in CN_NMAX register, it will be done by the spindle control before outputting the command signal.

CN_MGZNO: Number of magazine belonging to T code (DWORD)
CN_POTNO: Pot number belonging to T code (DWORD)

In case in the part program a T function has been programmed, the channel handler
– will write the number of tool programmed on the T function into the CN_TC register, in a lathe channel ***by cutting down the offset number*** called under address T,
– it will write into the CN_MGZNO register, in which magazine can the referred tool be found,
– it will write into the CN_POTNO register, in which pot of the given magazine can the referred tool be found,
– it will check whether the life of the referred tool or tool group has expired or not and it will set the CN_TLLE flag accordingly,
– then it will set the CN_TSTB flag for the period of 1 PLC cycle.

Upon the strobe signal, the PLC will decode the codes written into the above registers and it will reset the CP_TACK acknowledge signal. It is enabled to set the CP_TACK acknowledge signal only after the full execution of the function.

In case the N2900 Tool M. Config parameter’s #0 TMU=1 bit is set on the machine, the tool management table is operating. In such a case the CN_MGZNO and CN_POTNO registers shall be handled!

The tool management function shall be set up in the below cases on the control:
– if we would like to apply life management on the tools,
– if we would like to refer to the tools stored in the magazine from the part program not based on pot codes but tool codes,
– if the tool change applied on the machine requests a random tool pot management.

Into the tool management table we can write in the T codes of tools applied, i.e. their type numbers, to which we refer in the part program.

In case there are more tools in the cartridge which are doing the same machining operations and we want to use life management for these tools, we have to register these tools with the same type number in the table. In a part program, under address T we have to refer to the type number and the tool management will decide which tool with the same type number will it apply. Usually it applies the tool with the lowest but not-yet-expired tool life. The tool management table is common, i.e., it is common for all channels.

The tool management handles a maximum of 4 magazines. Besides this, any spindle on the machine tool, into which a tool may be clamped, can be defined as a spindle magazine on the N2922 Spindle Magazines parameter. To the spindle magazines, on the previous parameter, we can define a standby magazine for the storage of tools being in the changer arm.

Into the CN_TC register always the number of the tool referred to in the part program (type number) will be entered, even if using the tool management table, when there can be several tools with the same type number in the magazine.

In case of using a tool management table, the tool management will write into the
CN_MGZNO register, that in which magazine can the referred tool be found, respectively, into the
CN_POTNO register, in which pot of the given magazine can the tool be found.
Interpretation of the ***CN_MGZN*** register:
Meaning of CN_MGZN=0: the referred tool cannot be found in any of the magazines,
Meaning of CN_MGZN=1, 2, 3, 4: the tool is in one of the 1., 2., 3., 4. magazines,
Meaning of CN_MGZN=10, 20, 30, ... : the tool is in one of the 1., 2., 3., ... spindles,
Meaning of CN_MGZN=11, 21, 31, ... : the tool can be found in the standby magazine (changer arm) belonging to the 1., 2., 3., ... spindle.
Interpretation of the ***CN_POTNO*** register:
If CN_MGZN=0, it does not have any meaning,
if CN_MGZN=1, 2, 3, 4, then CN_POTNO=1, 2, 3, ... in the Tool Management parameter group the value set on the length of the given magazine,
if CN_MGZN=10, 20, 30, ..., then CN_POTNO=1, (it is in 1., 2., 3., ... spindle)
if CN_MGZN=11, 21, 31, ..., then CN_POTNO=1. (It is in the arm belonging to 1, 2, 3, ... spindle).

DWORD-type channel control variables going from the PLC to NC

CP_JOGFD: Jog feed-rate selection (DWORD)

In case on the N0316 Jog F Contr parameter the JFT option is selected (the parameter value is 1), in jog mode the feed-rate value is set by the channel handler based on the CP_JOGFD PLC register value, according to an exponential function. The feed-rate is issued always in the 1/min units.

The value of the CP_JOGFD register shall be linked to the status of the override switch. In case of an NCT machine control panel, to the value of the MKFOVER register.

The below table, depending on the value of the CP_JOGFD register shows the feed-rate values from 0 to 15, if the JFT parameter is set. Upon higher CP_JOGFD values the row will continue.

CP_JOGFD	**G21 mm/min**	**G20 in/min**	**rotary axis °/min**
0	0	0	0
1	2	0.08	0.4
2	3.2	0.12	0.64
3	5	0.2	1
4	7.9	0.3	1.58
5	12.6	0.5	2.52
6	20	0.8	4
7	32	1.2	6.4
8	50	2	10
9	79	3	15.8
10	126	5	25.2
11	200	8	40
12	320	12	64
13	500	20	100
14	790	30	158
15	1260	50	252

CP_SINP: Number of spindle machining (1,2...) (DWORD)

The PLC program has to assign the number of active the spindle carrying out the machining in the CP_SINP register per channel: ***CP_SINP***=1, 2, 3, ...

The ***channel handler*** takes the encoder pulses from the encoder of the spindle machining

– in case of G95 feed-rate per revolution,

– during G33 thread cutting,

– in case of G84.2, G84.3 rigid tapping it waits from the active spindle for the SN_LPCLSD loop closed signal and it will tap with the active spindle,

– in case of G51.2 polygon turning the master spindle will always be the active spindle, respectively

The ***PLC program***

– will always enable the feed-rate in a cutting block based on the rotation status of the spindle machining determined in the CP_SINP register, by the CP_CBFEN flag,

– in case of G96 constant surface speed control, in a CN_CSURF=1 status it will copy the value received in the CN_CTSPN register to the ***SP_PRG*** register of the spindle machining determined by the appropriate ***CP_SINP*** register.

The same spindle may be assigned to several channels. For instance, on a 2 path lathe, by assigning the left-side spindle to both channels, on the same workpiece it is possible for both paths to work with a feed-rate per revolution. The programer of the part may decide to which channel's axis X shall the constant surface speed calculated by the control (it has to program the G96 code in the necessary channel).

The machine tool builder can decide how to select the spindle machining. The appointment of the active spindle may be determined by using M codes. For example:

M31 code S1 is active,

M32 code S2 is active, etc.

☞ ***Attention!*** *The M function appointing the active spindle shall be a buffer suppressing function, as the execution of the code has an effect on the preparation of the blocks.*

CP_CSAX: Not used

CP_POLYSL: Number of slave spindle of polygonal turning (1,2...) (DWORD)

In the CP_POLYSL register one shall write per channel the number of the spindle which will be the slave spindle of polygonal turning, i.e. in which the tool will rotate. The master spindle of the polygonal turning will be appointed by the CP_SINP register.

Before issuing the G51.2 polygonal turning command, by an M function the PLC program

– has to select the slave spindle by determining the CP_POLYSL register

– the spindle control SP_SEN=1 shall be enabled

– the spindle drive has to be switched on.

CP_ACTT: Number of the current tool (DWORD)

In case the N2901 Search Config parameter's #7 TSP bit is 0, the displaying of the current tool number in the FST window will be carried out from the CP_ACTT register, indexed per channel.

The PLC program will write here the number of the exchanged tool.

CP_MGR1: M code in group 1 to be displayed (DWORD)

CP_MGR2: M code in group 2 to be displayed (DWORD)
CP_MGR3: M code in group 3 to be displayed (DWORD)
CP_MGR4: M code in group 4 to be displayed (DWORD)
CP_MGR5: M code in group 5 to be displayed (DWORD)
CP_MGR6: M code in group 6 to be displayed (DWORD)
CP_MGR7: M code in group 7 to be displayed (DWORD)
CP_MGR8: M code in group 8 to be displayed (DWORD)
CP_MGR9: M code in group 9 to be displayed (DWORD)
CP_MGR10: M code in group 10 to be displayed (DWORD)
CP_MGR11: M code in group 11 to be displayed (DWORD)
CP_MGR12: M code in group 12 to be displayed (DWORD)
CP_MGR13: M code in group 13 to be displayed (DWORD)
CP_MGR14: M code in group 14 to be displayed (DWORD)
CP_MGR15: M code in group 15 to be displayed (DWORD)
CP_MGR16: M code in group 16 to be displayed (DWORD)

On the N1341 M GR Low 1, ..., N1356 M GR Low 16 and the N1357 M GR High 1, ..., N1372 M GR High 16 parameters 16 different M code groups can be specified. On the parameter marked as Low the lowest-number code of the group shall be written, -and on the High the highest-number code of the group shall be written.

The parameters shall be set in a way that the codes in an M code group should represent each other excluding functions.

The block reader filters the M codes in a way that from among M codes belonging to the same group only one can be in the given block, otherwise it will issue a *Contradictory M-codes* error message.

The values set on the parameter are taken into consideration by the control also during the search for a block, by listing M codes. From among M codes belonging to the same group it will list only the code determined for the last time. Let's take a look on the following M codes:

M51: chuck clamp on spindle S1
M52: chuck unclamp on spindle S1
M53: chuck unclamp on spindle S1 when it is rotating

Let's set the parameter in the following way:

N1341 M GR Low 1=51
N1357 M GR High 1=53

Clamping status:

M54: clamping S1 inward
M55: clamping S1 outward

The parameters:

N1342 M GR Low 2=54
N1358 M GR High 2=55

During block search, from among the M51, M52, M53 codes it will sort out only the one programmed as the last one and it will have it executed by the operator. The same applies to the M54, M55 group, too. From among the appropriate machine statuses, only one will be written out in the CP_MGR1, 2 registers, e.g. after the clamping of the chuck the content of registers:

CP_MGR1=51 – chuck clamped on spindle S1 and

CP_MGR2=54 – clamped inward on spindle S1.

The M GR Low n, M GR High n pair corresponds to the CP_MGRn register. The beginning and ending number of those M code groups shall be written on the parameters which are written by the PLC program

– ***to the CP_MGR1, ..., CP_MGR16*** registers, and which become displayed in the M codes window.

CP_1032: #1032 macro variable (DWORD)

The user may make queries on PLC flags from the part program through macro variables. Per every channel, altogether 32 bits are available for the user for the communication with the PLC: #1000, #1001, ..., #1031. These variables may be queried also in a double-word format through the #1032 macro variable.
For instance, the command

```
IF #1032EQ16 GOTO30
```

written into the part program, will jump on the N30 block, if the #1004=1 (CP_1004 flag is true), the others are false.

CP_OFFSNO: Number of offset to be used at tool measurement (DWORD)

During manual measurement of tool length/work zero offset with a setter, the offset number where the tool length compensation value is written/from where the length compensation is taken to calculate the work offset, in case the value of the N3016 Tool/WP Setter Config parameter's #0 ONS bit is:

=0: the operator selects it manually on the control panel,

=1: the PLC selects it in the CP_OFFSNO register.

For example, if the operator wishes to measure the tools in a way that the offset number is the same as the number of the tool (e.g. T1212), the PLC program will write into the CP_OFFSNO register the number of the current tool. Thus, after the tool exchange, the measuring window will jump to the offset with the number equals to the current tool, it is not necessary to select manually the offset number to be measured.

7.15.3 Double-type Channel Control Variables

Inputs		Outputs	
Symbol	**Description**	**Symbol**	**Description**
CN_1133	#1133 macro variable value (double)	**CP_INC**	Size of step in incremental jog and handwheel mode (double)
CN_1134	#1134 macro variable value (double)	**CP_FOVER**	Feed-rate override: if =1: 100% (double)
CN_1135	#1135 macro variable value (double)	**CP_ROVER**	Rapid traverse override: if =1: 100% (double)
CN_GC	Not used	**CP_COV**	Not used
CN_G1DAT	Not used	**CP_1033**	#1033 macro variable (double)
CN_G2DAT	Not used	**CP_1034**	#1034 macro variable (double)
CN_G3DAT	Not used	**CP_1035**	#1035 macro variable (double)

Double-type channel control variables going from NC to PLC

CN_1133: #1133 macro variable value (double)
CN_1134: #1134 macro variable value (double)
CN_1135: #1135 macro variable value (double)

From a part program, the user, by giving values to #1133, #1134, #1135 macro variables may hand over floating-point data to the PLC program per channel. These data can be read out by the PLC program directly from the CN_1133, CN_1134, CN_1135 registers.

The command

```
#1134=167.832
```

written into the part program writes the number 167.832 to the CN_1134 PLC register.

CN_GC: Not used

CN_G1DAT: Not used

CN_G2DAT: Not used

CN_G3DAT: Not used

Double-type channel control variables going from PLC to NC

CP_INC: Size of step in incremental jog and handwheel mode (double)

In incremental jog (CN_INCR=1), handwheel (CN_HNDL=1) and feed-rate from handwheel modes (CP_FHNDL=1) the size of step shall be written into the CP_INC register per channel in floating-point format.

The selection of the step size, in case of an NCT machine control panel is carried out from the

MB_I1, MB_I10, MB_I100, MB_I1000 buttons.

The selection of the step size, in case of an NCT external handwheel is carried out from the

HB_I1, HB_I10, HB_I100, HB_I1000

flags.

Into the CP_INC register always a floating-point number shall be written. The unit of the number to be written in is determined by the #0 IND bit of the N0104 Unit of Measure parameter, which shows the output unit. (The unit in which the position measurement is carried out.)

If CP_INC=0.01 and

IND=0: CP_INC=0.01 means 0.01 mm

IND=1: CP_INC=0.01 means 0.01 inch

Depending on the input unit applied in the control the data shall be converted. For the conversion the CN_INCH flag can be used. If the flag

CN_INCH= 0: a G21 metric data input

CN_INCH=1: a G20 inch data input

is valid.

For example when the button MB_I10 (0.01) has been pushed:

– in case of ***IND=0*** (metric measurement) ***CN_INCH=0*** (G21 metric data input) a ***CP_INC=0.01*** (step size ***0.01 mm***) shall be written

– in case of ***IND=0*** (metric measurement) ***CN_INCH=1*** (G20 inch data input) a ***CP_INC=0.0254*** (step size 0.0254 mm = 0.0254/25.4 = ***0.01 inch***) shall be written.

CP_FOVER: Feed-rate override: if =1: 100% (double)

It is the feedrate override value per channel. A floating-point number. If, for example

CP_FOVER=1.0 means 100%

CP_FOVER=1.427 means 142.7%

In case of using an NCT machine control panel, the status of the override switch shall be taken from the MKFOVER register.

The upper limit of the feed-rate override shall be set in the PLC program!

☞ ***Attention!*** *MKFOVER is of an integer DWORD type, CP_FOVER is of a floating-point double, thus the setting of the override requires a conversion from integer to floating-point (FLT instruction). The indexation of CP_FOVER happens two-by-two!*

CP_ROVER: Rapid traverse override: if =1: 100% (double)

It is the value of the rapid override per channel. It is a floating-point number. If e.g.

CP_ROVER=0.272 means 27.2%

CP_ROVER=1.0 means 100%

In case of using an NCT machine control panel, the value of the rapid override can be taken also from the MKFOVER register.

The upper limit of the rapid override is 100%, and the channel handler will not let it further than this!

☞ ***Attention!*** *The MKFOVER is of an integer DWORD type, CP_ROVER is a floating-point double, thus the setting of the override requires a conversion from integer to floating-point (FLT instruction). The indexation of CP_ROVER happens two-by-two!*

CP_COV: Not used

CP_1033: #1033 macro variable (double)
CP_1034: #1034 macro variable (double)
CP_1035: #1035 macro variable (double)

The PLC program, by writing into the CP_1033, CP_1034, CP_1035 registers may hand over floating-point data directly to the part program. The part program may use the data through #1033, #1034, #1035 macro variables.

The instruction

```
#100=#1134
```

written into the part program writes the floating-point number written into the CP_1134 PLC register to the #100 macro variable.

Index in Alphabetical Order:

. (dot). 15
, (comma). 13, 18
: (colon). 17
!. 24
+F. 76
-F. 77
*. 25
*F. 78
/F. 79
%. 23
#. 24, 25
#$. 25
@. 23
ACOS. 86
ActPosAx. 118
ActPosSp. 120
ADD. 71
ALR. 102
ALRF. 102
AN_AXALM. 279
AN_BEPTY. 282
AN_DETCHA. 278
AN_EGBS. 281
AN_GOA. 283
AN_IEPTY. 283
AN_INDP. 282
AN_INPOS. 279
AN_LUBR. 279
AN_MIRA. 283
AN_MIXA. 281
AN_MIXM. 282
AN_MTNRN. 279
AN_MTNRP. 279
AN_OPNA. 278
AN_OTN. 280
AN_OTP. 280
AN_PARKA. 281
AN_PLCA. 282
AN_RAPR. 279
AN_REFEND. 280
AN_REFP1. 280
AN_REFP2. 280
AN_REFP3. 280
AN_REFP4. 280
AN_RPE. 280
AN_SPRPNA. 281
AN_SPRPNM. 282
AN_SYNCA. 281
AN_SYNCM. 282
AND. 67
ANINPUTS. 245, 246
AP_DECSW. 286
AP_DETCHR. 284
AP_DIARAD. 291
AP_DISPD. 288
AP_EFD. 292
AP_END. 285
AP_FEEDD. 291
AP_FLWU. 285
AP_GOR. 292
AP_JOGN. 286
AP_JOGP. 286
AP_LCK. 287
AP_LIMN. 287
AP_LIMP. 287
AP_LIMSELN. 287
AP_LIMSELP. 287
AP_MIRR. 293
AP_MIXR. 289
AP_MTNDN. 286
AP_MTNDP. 286
AP_OPNR. 285
AP_PARKR. 288
AP_PLCR. 292
AP_RAPD. 286
AP_RES. 292

AP_RPE. 293
AP_SPRPNR. 290
AP_SSLOP. 291
AP_SYNCR. 288
ARTL. 63
ARTR. 63
ASHL. 61
ASHR. 61
ASIN.. 85
ATAN.. 87
AxesTime. 121
base address. 13
BCD. 92
BIN. 91
byte. 12
CANErr.. 121
CEQ. 98
CGE. 98
CGT. 98
ChannelsTime.. 121
CLE.. 98
closed contact. 32, 33
closing element.. 41
CLT.. 98
CMP. 97
CN_1100.. 328
CN_1101.. 328
CN_1102.. 328
CN_1103.. 328
CN_1104.. 328
CN_1105.. 328
CN_1106.. 328
CN_1107.. 328
CN_1108.. 328
CN_1109.. 328
CN_1110.. 328
CN_1111.. 328
CN_1112.. 328
CN_1113.. 328
CN_1114.. 328
CN_1115.. 328
CN_1116.. 328
CN_1117.. 328
CN_1118.. 328
CN_1119.. 328
CN_1120.. 328
CN_1121.. 328
CN_1122.. 328
CN_1123.. 328
CN_1124.. 328
CN_1125.. 328
CN_1126.. 328
CN_1127.. 328
CN_1128.. 328
CN_1129.. 328
CN_1130.. 328
CN_1131.. 328
CN_1132.. 344
CN_1133.. 355
CN_1134.. 355
CN_1135.. 355
CN_ABSOFF. 325
CN_AHND.. 327
CN_ALRM.. 316
CN_AUTO. 322
CN_AUX1C.. 348
CN_AUX1STB.. 316
CN_AUX2C.. 348
CN_AUX2STB.. 316
CN_AUX3C.. 348
CN_AUX3STB.. 316
CN_BKBUF.. 316
CN_BKRET.. 326
CN_BKRST. 326
CN_CBFR. 319
CN_CHARP.. 321
CN_CHOP. 321
CN_CSACK.. 318
CN_CSURFS. 317
CN_CTSPN. 348
CN_DRRUN.. 326
CN_DWELL.. 320

CN_EDIT. 322
CN_EGBMD. 318
CN_FLCK. 322
CN_FREV. 320
CN_G1DAT. 355
CN_G2DAT. 355
CN_G3DAT. 355
CN_GC. 355
CN_GSTB. 316
CN_HNDL. 322
CN_HSHP. 318
CN_INCH. 318
CN_INCR. 322
CN_INTD.. 317
CN_IPEPTY. 319
CN_IPSTP. 319
CN_ITFALM. 317
CN_ITFCHK. 317
CN_JOG.. 322
CN_M1C. 344
CN_M1STB. 315
CN_M2C. 344
CN_M2STB. 315
CN_M3C. 344
CN_M3STB. 315
CN_M4C. 344
CN_M4STB. 315
CN_M5C. 344
CN_M5STB. 315
CN_M6C. 344
CN_M6STB. 315
CN_M7C. 344
CN_M7STB. 315
CN_M8C. 344
CN_M8STB. 315
CN_MDI. 322
CN_MGZNO. 349
CN_MLCK. 325
CN_NMAX. 348
CN_OPMES.. 317
CN_OVDIS. 319
CN_POLYT. 318
CN_POSCHK. 320
CN_POTNO.. 349
CN_REFP. 322
CN_REFPG. 320
CN_RNGREQ. 346
CN_RTAP. 320
CN_RTRFIN. 318
CN_SC.. 345
CN_SKIP. 320
CN_SSEL.. 345
CN_SSTB. 315
CN_START. 321
CN_STOP. 321
CN_STPREQ.. 316
CN_TAP. 320
CN_TAXF. 322
CN_TBLB. 324
CN_TC.. 347
CN_TEST. 325
CN_THRD.. 319
CN_THRDC. 320
CN_TLLE. 321
CN_TRGAF.. 323
CN_TSTB. 316
CN_TTCRF. 324
CN_WMCAXF. 327
CN_WPCNT. 318
CN_WTNG. 317
CNCBufferCount. 121
CNE. 98
CNT. 52, 54
column. 41
CommandAx. 118
CommandSp.. 120
comment.. 42
CompenValAx. 119
ComPosAx.. 117
ComPosSp. 119
ComVelAx.. 118
ComVelSp. 119

COS. 83
CP_1000. 341
CP_1001. 341
CP_1002. 341
CP_1003. 341
CP_1004. 341
CP_1005. 341
CP_1006. 342
CP_1007. 342
CP_1008. 342
CP_1009. 342
CP_1010. 342
CP_1011. 342
CP_1012. 342
CP_1013. 342
CP_1014. 342
CP_1015. 342
CP_1016. 342
CP_1017. 342
CP_1018. 342
CP_1019. 342
CP_1020. 342
CP_1021. 342
CP_1022. 342
CP_1023. 342
CP_1024. 342
CP_1025. 342
CP_1026. 342
CP_1027. 342
CP_1028. 342
CP_1029. 342
CP_1030. 342
CP_1031. 342
CP_1032. 354
CP_1033. 357
CP_1034. 357
CP_1035. 357
CP_ABSOFF. 334
CP_ACTT.. 352
CP_AHND. 332
CP_AUTO. 329
CP_AUX1ACK. 338
CP_AUX2ACK. 338
CP_AUX3ACK. 338
CP_BKRET. 334
CP_BKRST. 334
CP_CBFEN. 339
CP_CHOPON.. 340
CP_CNDBK_1. 335
CP_CNDBK_2. 335
CP_CNDBK_3. 335
CP_CNDBK_4. 335
CP_CNDBK_5. 335
CP_CNDBK_6. 335
CP_CNDBK_7. 335
CP_CNDBK_8. 335
CP_CNDSP. 333
CP_COV.. 357
CP_CSAX. 352
CP_CSREQ. 335
CP_DRRUN.. 334
CP_EDIT. 329
CP_EGBRRQ.. 336
CP_FHNDL. 339
CP_FIN.. 335
CP_FLCK.. 334
CP_FOVER. 356
CP_GACK. 338
CP_HNDL. 329
CP_HNDLS1. 339
CP_HNDLS2. 339
CP_HNDLS3. 339
CP_HNDLS4. 339
CP_HOLD. 338
CP_INC. 356
CP_INCR. 329
CP_JOG. 329
CP_JOGFD.. 351
CP_JOGRAP. 330
CP_LIM1DIS. 339
CP_LIM2DIS. 339
CP_LIM3DIS. 339

CP_LIMSEL. 340
CP_M1ACK.. 337
CP_M2ACK.. 337
CP_M3ACK.. 337
CP_M4ACK.. 337
CP_M5ACK.. 337
CP_M6ACK.. 337
CP_M7ACK.. 337
CP_M8ACK.. 337
CP_MDI.. 329
CP_MGR1. 352
CP_MGR10. 353
CP_MGR11. 353
CP_MGR12. 353
CP_MGR13. 353
CP_MGR14. 353
CP_MGR15. 353
CP_MGR16. 353
CP_MGR2. 353
CP_MGR3. 353
CP_MGR4. 353
CP_MGR5. 353
CP_MGR6. 353
CP_MGR7. 353
CP_MGR8. 353
CP_MGR9. 353
CP_MINT. 336
CP_MLCK. 333
CP_NOWT.. 335
CP_OFFSNO. 354
CP_OSGNX.. 341
CP_OSGNY.. 341
CP_OSGNZ. 341
CP_OVC. 339
CP_POLYSL. 352
CP_REFP.. 329
CP_RLSOT3. 341
CP_ROVER.. 357
CP_ROVLD.. 341
CP_RST.. 335
CP_RSTREW. 335
CP_S2TS. 331
CP_S2WS. 332
CP_SACK. 337
CP_SGLBK. 333
CP_SGOEN.. 340
CP_SGOX. 341
CP_SGOY. 341
CP_SGOZ. 341
CP_SINP. 351
CP_START. 329
CP_STOP.. 329
CP_TACK. 338
CP_TAXF. 330
CP_TBLB.. 330
CP_TEST.. 333
CP_TLCM. 331
CP_TMREN.. 336
CP_TRGAF. 330
CP_TSBD. 336
CP_TTCRF. 330
CP_WMCAXF.. 332
CP_WPCM. 332
DA-i/I EE.. 247
DANI. 246
DEG.. 95
DIFD. 38
DIFU. 37
DirectPlcBit. 117
DirectPlcDouble.. 117
DirectPlcInt. 117
DIV.. 74
DN_BVERR. 253
DN_CHERR1. 254
DN_CMEERR.. 254
DN_CURERR. 253
DN_CWDER1.. 254
DN_ECTERR. 254, 260
DN_EDERR1.. . . . . 253, 260
DN_EDERR2.. 253
DN_ENA. 251
DN_ERR. 255, 261

DN_ERROR. 253, 260
DN_FOLERR. 254
DN_HALERR. 254
DN_HASERR. 254
DN_INC. 251, 260
DN_OVHERR. 255
DN_PDPINT. 254
DN_PRGERR. 254
DN_PRM1. 251
DN_PRM2. 251
DN_PRMERR. 254
DN_RDY. 251
DN_SRTERR. 254
DN_STAT. 255, 261
Double word. 11
DP_CTRL. 255, 261
DP_EMG. 252
DP_ENA1. 251
DP_ENA2. 251
DP_ERRCLR. 253, 260
DP_MOD1. 252
DP_MOD2. 252
DP_POSLCK. 252
DP_PRM1. 252
DP_PRM2. 252
DP_SILCK. 253
DS-i/I EE. 247
DWORD. 11
EGBcCurrAx. 119
EGBvTarAx. 118
END. 106
ENDAT. 256
ETPC. 242
EXP. 88
Exponent. 26
FCMP. 97
FIX. 94
FL_CY. 28
FL_EQ. 29
FL_ER. 28
FL_GT. 28
FL_LT. 29
FL_OF. 28
FL_UF. 28
FLEQ. 98
FLGE. 98
FLGT. 98
FLLE. 98
FLLT. 98
FLNE. 98
FLT. 93
FolErrAx. 118
FolErrSp. 119
FUP. 285
HardwareTime. 121
HB_AXIS4. 237
HB_AXIS5. 237
HB_AXIS6. 237
HB_AXIS7. 237
HB_AXIS8. 237
HB_AXISX. 237
HB_AXISY. 237
HB_AXISZ. 237
HB_B12. 238
HB_B13. 238
HB_I1. 237
HB_I10. 237
HB_I100. 237
HB_I1000. 238
HWMOVE. 238
I16. 240
I16S. 240
I32. 240
IEEE754. 26
IN_1. 245
IN_1ENn. 245
Infinity. 27
Input Not of a Memory Bit. 32
Input of a Memory Bit. 30
Int0. 21
JME. 107
JMP. 107

ladder diagram. 8
LOG. 89
M GR High n. 353
MACR. 114
MACW.. 115
Main Program. 21
Mantissa.. 26
MB_AUTO. 234
MB_BKRET. 234
MB_BKRST. 234
MB_CNDBK. 235
MB_CNDSP. 235
MB_DRRUN. 234
MB_EDIT. 234
MB_FLCK.. 234
MB_HNDL. 234
MB_I1. 235
MB_I10. 235
MB_I100. 235
MB_I1000. 235
MB_INCR. 234
MB_JOG. 234
MB_JOGn. 235
MB_JOGRAP. 235
MB_MDI. 234
MB_MLCK. 234
MB_REFP. 234
MB_S100.. . . . . 236, 309
MB_SGLBK. 235
MB_SMAX. 236, 309
MB_SMIN. 236
MB_START.. 234
MB_STOP. 234
MB_TEST. 234
Measured. 121
Mechanical Type. 323, 324
MK15.. 230
MK19.. 230
MKFOVER. 236
MKSOVER. 236
ML_AUTO.. 234
ML_BKRET.. 234
ML_BKRST.. 234
ML_CNDBK. 235
ML_CNDSP.. 235
ML_DRRUN. 234
ML_EDIT.. 234
ML_FLCK. 234
ML_HNDL.. 234
ML_INCR. 234
ML_JOG. 234
ML_JOGn. 235
ML_JOGRAP. 235
ML_MDI. 234
ML_MLCK. 234
ML_REFP. 234
ML_SGLBK.. 235
ML_START.. 234
ML_STOP. 234
ML_TEST. 234
MOV. 45
MOVCMD.. 110
MOVF. 46
MR.. 123
MR, MW function codes
  10. 127
  100. 164
  101. 165
  102. 167
  103. 171
  104. 176
  105. 179
  106. 180
  107. 183
  108. 185
  11. 129
  201. 211
  202. 218
  31. 131
  32. 133
  34. 135
  35. 137

40. 139
60. 140
61. 141
62. 142
63. 143
70. 145
81. 149
90. 151
MR, MW funkció kódok
300. 193
303. 195
304. 198
306. 200
307. 202
308. 204
309. 206
41. 208
MSG. 102
MSGF. 102
MUL. 73
MVN. 45
MW. 124
N_ACTMSG. 276
N_CLRMSG. 272
N_FIRSTCC. 271
N_MONDIS. 272
N_MONST. 271
N_MSG. 262
N_MSG0. 272
N_MSG1. 272
N_MSG2. 272
N_MSG3. 272
N_MSG4. 272
N_MSG5. 272
N_MSG6. 272
N_MSG7. 272
N_MSG8. 272
N_MSG9. 272
N_MSGA. 272
N_NCREADY. 271
N_NVECAT. 273
N_NVRAMOK. 271
N_OFF. 271
N_ON. 271
N_P100MS. 271
N_P1M. 271
N_P1S. 271
N_P2MS. 271
N_P2T. 271
N_PDB1. 267
N_PDW1. 267
N_Pij. 267
N_PTEDT. 273
N_SIMU. 273
N_SW. 263
N_SW0. 264
N_SWN. 263
N_SWP. 264
N_TLEDT. 273
N_TLMD. 273
N_TLSRCH. 273
N_TLSV. 273
NActSp. 120
NaN. 27
NCommandSp. 120
NCTDriveMess. 122
NEG. 66
Negated Query of a Double Word. 33
negated relay coil. 35
Normalized number. 26
Not a Number. 27
NSetSp. 120
O16. 240
O8R. 240
O8RM. 240
offset. 13
open contact. 30, 31
OR. 67
Output Not to a Memory Bit. 35
Output to a Memory Bit. 34
P_CHSEL. 276
P_COMPG. 274

P_COMPW. 274
P_DIR. 274
P_HnAS. 239
P_HOLD0. 274
P_MAC. 274
P_MONREQ. 273
P_MSG.. 262
P_PAR. 275
P_PLC. 275
P_PRGE. 274
P_PTTAB.. 275
P_RUNAUT.. 275
P_RUNMDI. 275
P_SHTDNREQ. 274
P_SVRC.. 275
P_TLTAB.. 274
P_TRCTR. 274
P_WOFFS. 274
PitchAx. 119
PLC Doublen. 267
PlcBit. 117
PlcCycle. 121
PlcDouble.. 117
PlcInt. 117
PlcTime. 121
PosErrAx. 118
PosErrSp. 120
Pulse Generation for Falling Edge. 38
Pulse Generation for Rising Edge. 37
PWR.. 80
Query of a Double Word. 31
RAD.. 96
RealTime. 121
relay circuit. 8
relay coil.. 34
REM.. 102
Remark.. 42
REMF.. 102
Resetting a Memory Bit. 36
RET. 108
ROT. 56
RotaryAx. 121
row.. 41
RST. 36
sampling interval. 9
SBN. 108
SBS. 108
SCP. 116
SEC. 43
SENS. 244
SET. 36
Setting a Memory Bit.. 36
SHTR. 60
Sign bit.. 26
SIN.. 82
SN_FLOFF.. 296
SN_FLU. 295
SN_LPCLSD. 296
SN_N0. 295
SN_NACT. 307
SN_NCOM. 307
SN_NS. 295
SN_ORIP. 296
SN_PHSHFTA.. 296
SN_POLYA. 297
SN_RMPD. 295
SN_RPE.. 298
SN_SALM. 298
SN_SDETCHA. 297
SN_SINDP. 298
SN_SINPOS.. 296
SN_SMTNRN. 297
SN_SMTNRP. 297
SN_SRAPR. 297
SN_SSYNA. 296
SN_SYNCPOS. 297
SN_TLCH. 298
SN_TLCHI. 298
SN_TLNL. 299
SN_TLSKPA. 299
SP_ACTT.. 308
SP_ASSIGN.. 308

SP_FEEDD.. 304
SP_MAST. 308
SP_NEG. 299
SP_OREQ.. 300
SP_OSHRT. 300
SP_OSW.. 306
SP_PAR. 299
SP_PHSHFTR. 302
SP_POLYR.. 302
SP_PRG. 307
SP_RAPD. 305
SP_RNG. 307
SP_ROT. 307
SP_SDETCHR. 303
SP_SDISPD. 304
SP_SEN. 299
SP_SEND. 302
SP_SLCLR. 304
SP_SMTNDN.. 302
SP_SMTNDP. 302
SP_SOVER. 309
SP_SSROFF. 304
SP_SSTRT. 299
SP_SSYNCR. 301
SP_TLCD. 305
SP_TLCHA. 305
SP_TLCHIA. 305
SP_TLSKP. 305
SQRT. 81
StraightnessAx. 119
SUB. 72
Subnormal or denormalized number. 27
SyncErrAx. 119
SyncErrSp.. 121
SyncVSlaveSp. 121
SyncVTargetSp.. 120
TachAx. 118
TachRealAx. 118
TachRealSp.. 122
TachSp. 120
TAN. 84
TN_INPn1. 243
TN_INPn2. 243
TN_INPn3. 243
TN_TSn. 243
TOFFD. 49
TOND.. 48
Tool Axis Direction. 323, 324
TP_OUTn1. 243
TP_OUTn2. 243
TPLC. 9
TPULSE. 50
TSliceErr.. 121
TTLAI.. 256
  ECAT-TACHO. 256
  ECAT-TTLASM. 256
TTLCAN. 256
WORD. 12
XOR. 67
Zero.. 27